Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програми візуалізації фракталів для GameDev»

Студента 4 курсу, 6 групи,
спеціальності 121 «Інженерія програмного забезпечення» _____ Чудіка Михайла Івановича
підпис студента

Науковий керівник
кандидат педагогічних наук,
доцент _____ Жирова Тетяна Олександрівна
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович
підпис керівника

КИЇВ – 2020

Київський національний торговельно-економічний університет

Факультет *інформаційних технологій*
Кафедра *інженерії програмного забезпечення та кібербезпеки*
Освітній ступінь *бакалавр*
Спеціальність *121 «Інженерія програмного забезпечення»*
Спеціалізація / освітня програма *121 «Інженерія програмного забезпечення»*

Затверджую
Зав. кафедри інженерії
програмного забезпечення
та кібербезпеки
Криворучко О. В.
"7" листопада 2019 р.

Завдання
на випускну кваліфікаційна робота студентові

Чудіку Михайлові Івановичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи *«Розробка програми візуалізації фракталів для GameDev»*

Затверджена наказом ректора від *«02» грудня 2019 р. № 4112*
2. Строк здачі студентом закінченої роботи

3. Цільова установка та вихідні дані до роботи
Мета роботи *застосування фракталів та їх вплив на сферу GameDev*

Об'єкт дослідження *фрактали*

Предмет дослідження *застосування фракталів у сфері GameDev*

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ФРАКТАЛИ ТА ТЕХНІЧНЕ ЗАВДАННЯ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

1.1. Загальні відомості та характеристики фракталів

1.2. Різновиди фракталів

1.3. Вплив фракталів на різні галузі

1.4. Технічне завдання

1.5. Висновки до розділу 1

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТУ ДОСЛІДЖЕННЯ ТА ХАРАКТЕРИСТИКА СФЕРИ GAMEDEV

2.1. Застосування фракталів у GameDev

2.2. Постановка задачі

2.3. Основна характеристика сфери GameDev

2.4. GameDev в Україні

2.5. Висновки до розділу 2

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІЗУАЛІЗАЦІЇ ФРАКТАЛІВ

3.1. Вибір мови програмування та середовища для розробки

3.2. Вибір графічної бібліотеки

3.3. Створення програмного продукту

3.4. Опис функціональних характеристик

3.5. Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*		
2.	*Вступ та перелік літературних джерел*		
3.	*Розділ 1. «Основні теоретичні відомості про фрактали та технічне завдання для розробки програмного продукту»*		
4.	*Розділ 2.* «Аналіз предмету дослідження та характеристика сфери GameDev»		
5.	*Розділ 3. «Розробка програмного забезпечення візуалізації фракталів»*		
6.	*Висновки*		
7.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*		
8.	*Підготовка автореферату та презентації доповіді*		
9.	*Попередній захист випускної кваліфікаційної роботи*		
10.	*Зовнішнє рецензування випускної кваліфікаційної роботи*		
11.	*Здача прошитої випускної кваліфікаційної роботи на кафедру*		
12.	*Публічний захист випускної кваліфікаційної роботи*		

7. Дата видачі завдання «___»_____20__ р.

8. Науковий керівник випускної кваліфікаційної роботи Жирова Т. О.

9. Гарант освітньої програми Цензура М. О.

10. Завдання прийняв до виконання студент Чудік М. І.

11. Відгук керівника випускної кваліфікаційної роботи

Науковий керівник випускної кваліфікаційної роботи _____
(підпис, дата)

Відмітка про попередній захист *Жирова Т. О.* _____
(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну робота

Випускна кваліфікаційна робота студента *Чудіка М. І.* може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Цензура М. О.

Завідувач кафедри Криворучко О. В.

«____»_____20 ___ р.

АНОТАЦІЯ

випускної кваліфікаційної роботи Чудіка Михайла Івановича

на тему: «*Розробка програми візуалізації фракталів для GameDev*»

Дана випускна кваліфікаційна робота присвячена розробці програми з візуалізації фракталів для сфери GameDev. Метою роботи є застосування фракталів та їх вплив на сферу GameDev. У роботі було досліджено основні теоретичні відомості про фрактали. Проведено аналіз та дослідження використання фракталів у сфері GameDev, а також проаналізовано її загальні характеристики та стан в Україні. Створено – технічне завдання та постановку задачі. Також, було наведено доцільність використання для даної роботи таких компонентів, як: мова С, графічна бібліотека SDL та середовища розробки Visual Studio 2019. Детально описано процес створення програми, та її головних функціональних моментів.

Загальний обсяг роботи: 50 сторінок, 3 розділи, 39 рисунків, 38 джерел.

Ключові слова: фрактал, GameDev, самоподібність, SDL, Мандельброт, рекурсія, комплексні числа.

ANNOTATION

final qualifying work Chudik Mykhailo Ivanovych

on the topic: "Development of fractal visualization program for GameDev"

This final qualifying work is devoted to the development of a program for visualization of fractals for the field of GameDev. The aim of the work is the use of fractals and their impact on the scope of GameDev. The basic theoretical information about fractals was investigated in the work. The analysis and research of the use of fractals in the field of GameDev, as well as its general characteristics and state in Ukraine are analyzed. Created - technical task and problem statement. Also, it was advisable to use for this work components such as: C language, SDL graphics library and Visual Studio 2019 development environment The process of creating the program and its main functional moments are described in detail.

Total volume of work: 50 pages, 3 sections, 39 form, 38 sources.

Keywords: fractal, GameDev, self-similarity, SDL, Mandelbrot, recursion, complex numbers.

ЗМІСТ

					КНТЕУ 121 06-25.БР		
Зм.	*Аркуш*	*№ докум*	*Підпис*	*Дата*			
Зав. кафедри		*Криворучко О.В.*			*«Розробка програми візуалізації фракталів для GameDev»*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		*Жирова Т.О.*				*Зміст*	*1*	*51*
Гарант		*Цензура М.О.*						
Розроб.		*Чудік М.І.*			*Зміст*	Факультет інформаційних технологій, 4 курс, 6 група		

ВСТУП

Нові технології, які зазвичай являються геніальними відкриттями, зробленими вченими у різних галузях науки, приносять у наші життя кардинальні зміни, які захоплюють усі можливі галузі. Ці відкриття є відомими для усіх. Наприклад, створення деякою людиною вакцини від вірусу рятувало, свого часу, тисячі, сотні тисяч, мільйони, а то і більшу кількість життів. А таке відкриття як електрика, котре відбулось в межах історії зовсім нещодавно, так і взагалі кардинально змінює життя усього людства. Без цих відкриттів минулого людина уже не може уявити свого життя. Проте є відкриття котрі залишаються у тіні інших, хоч вони і також вносять значну частину в життя людей. Так, одним з таких відкриттів став фрактал. Швидше за все більшість людей навіть ніколи і не чули про таке поняття, а навіть якщо і чули, то не зможуть пояснити його значення. Зазвичай для цих людей здається, що геометрія в природі обмежується лише простими фігурами. Наприклад, це може бути лінія, коло, багатокутник, сфера, площина, чи різними їхніми комбінаціями.

Проте, якщо придивитись до того, що нас оточує, то можна виявити, що чимало природніх систем є настільки складними і нерегулярними, що як би сильно не хотілось використовувати тільки знайомі об'єкти з геометрії, до якої більшість звикла, ми не зможемо ними змоделювати ці речі. Так, в прагненні людини до пізнання, вона намагається користуватись логікою в різних міркуваннях. Під час аналізування процесу, які є навколо, людина намагається вирахувати якийсь взаємозв’язок. Навіть найкращі розуми планети займаються цими питаннями. Грубо кажучи, вони намагаються знайти закономірність там, де її немає і в принципі взагалі не повинно бути. Проте, навіть в хаосі все-таки є певний зв’язок між подіями, які, на перший погляд, є непов’язаними між собою.

					КНТЕУ 121 06-25.БР		
Зм.	Аркуш	№ докум	Підпис	Дата			
Зав. кафедри		Криворучко О.В.			«Розробка програми візуалізації фракталів для GameDev»	Стадія	Аркуш	Аркушів
Керівник		Жирова Т.О.				В	3	51
Гарант		Цензура М.О.						
Розроб.		Чудік М.І.			Вступ	Факультет інформаційних технологій, 4 курс, 6 група		

Саме цим зв’язком для поєднання того, що здавалось неможливо поєднати і виступає фрактал.

Якщо озирнутись навколо, то фрактали можна знайти скрізь і всюди, адже як би це дивно не звучало, багато органічних і неорганічних форм створюються аналогічно. Це можуть бути і хмари, і мушлі, і дерева, і навіть кровоносна система. Якщо захотіти, то цей список можна продовжувати ще дуже довго. Як приклад можна розглянути папороть. Якщо придивитись до неї, то можна побачити, що кожне відгалуження повторює собою саму папороть. І ця схожість здавалось би окремого відгалуження з початковою формою і виступає в якості доказу про так званий принцип рекурсивної самоподібності. Теоретично, елемент «розгалуження» повторюється нескінченно багато разів, стаючи все менше і менше. Так проявляється характерна для фракталів властивість самоподібності. І відповідно всі ці випадкові форми з легкістю описує фрактальний алгоритм.

Відкриття фракталів було відкриттям нової естетики мистецтва, науки і математики, а також революцією в людському сприйнятті світу. Вивчення фракталів і хаосу відкриває чудові можливості, як в дослідженні нескінченного числа додатків, так і в області чистої математики. Саме тому фрактали і набули свого законного поширення у багатьох галузях. Один із найбільших вплив від фракталів зазнає саме комп’ютерна галузь. Навіть з’явився такий напрямок комп’ютерної графіки як фрактальна. Програмісти і фахівці в області комп'ютерної техніки захоплюються фракталами, так як фрактали нескінченної складності і краси можуть бути згенеровані простими формулами на простих домашніх комп'ютерах.

Фрактали являються важливим та нескінченним колодязем можливостей для розважальних сфер, а саме для сфери GameDev. Адже зазвичай величезна кількість часу витрачається на створення унікальних візуальних ефектів, котрі були би максимально наближенні до реальних. Фрактали же дозволяють це зробити максимально швидко. Тому у наш час багато провідних гігантів, або

невеликих інді-розробників використовують фрактали у своїх дітищах. Тому, мета випускної кваліфікаційної роботи, а саме «Розробка програми візуалізації фракталів для GameDev» є актуальною.

Метою дослідження випускної кваліфікаційної роботи є застосування фракталів та їх вплив на сферу GameDev.

Об'єктом дослідження виступають фрактали.

Предметом дослідження є застосування фракталів у сфері GameDev.

Завданням дослідження виступає створення програми для візуалізації фракталів.

Методи дослідження: аксіоматичний, емпіричний, розрахунок, формалізація, аналіз.

РОЗДІЛ 1.
ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ФРАКТАЛИ ТА ТЕХНІЧНЕ ЗАВДАННЯ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

1.1. Загальні відомості та характеристики фракталів

Термін фрактал був введений Бенуа Мандельбротом у 1975 році і був похідним від латинського *fractus*, що означає «зламаний». У своїй книзі «Фрактальна геометрія природи» Мандельброт описує фрактал як «грубу або роздроблену геометричну форму, яку можна розділити на частини, кожна з яких є (принаймні приблизно) копією цілого зменшеного розміру», як правило, це корисно, але обмежено. Багато науковців не погоджуються з точним визначенням фрактала, але найчастіше детальніше досліджують основні ідеї самоподібності та незвичайні взаємозв'язки фракталів з простором, в який вони вкладені. [2, с.13]

Фрактальні візерунки характеризуються фрактальними розмірами , але хоча ці числа кількісно оцінюють складність (тобто, змінюють деталі зі зміною масштабу), вони ні однозначно не описують, ні конкретизують деталі того, як побудувати конкретні фрактальні візерунки. У 1975 році, коли Мандельброт придумав слово "фрактал", він зробив це для позначення об'єкта, розмір якого більший за його топологічний вимір . Однак цій вимозі не відповідають криві, що заповнюють простір, такі як крива Гільберта котра вказана на рис.1.1.

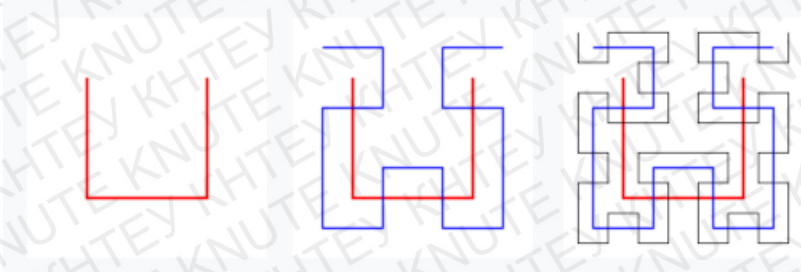

Рис. 1.1. Крива Гільберта, 1-3 ітерація

Через проблеми, пов'язані з пошуком одного визначення фракталів, деякі дослідники стверджують, що фрактали взагалі не повинні бути чітко визначеними. На думку Фальконера, фрактали, мало того, що не можуть диференціюватись і мають бути здатними мати фрактальний розмір, а й також мають загалом характеризуватися лише списком наступних особливостей [21]:

- Самоподібність, яка може включати:
  - Точну самоподібність – однакова у всіх масштабах. Наприклад, сніжинка Коха.
  - Квазі самоподібність – наближає один і той же візерунок до різних масштабів; може містити невеликі копії всього фрактала у викривленому та виродженому вигляді. Наприклад, супутники набору Мандельброта – це наближення всього набору, але не точні копії.
  - Статистична подібність – закономірність повторюється стохастично, тому числові або статистичні заходи зберігаються в масштабах. Наприклад, випадково породжені фрактали, як добре відомий приклад узбережжя Британії, для якого не можна було б сподіватися знайти відрізок масштабування та повторення так само акуратно, як повторний блок, який визначає фрактали, як сніжинка Коха.
  - Якісна самоподібність – як у часовій серії.
  - Багатофрактальне масштабування – характеризується більш ніж одним фрактальним розміром або правилом масштабування.
- Тонка або деталізована структура в довільно малих масштабах. Наслідком цієї структури є фрактали, які можуть мати виникаючі властивості.
- Локальна та глобальна нерегулярність, котру важко описати традиційною евклідовою геометричною мовою. Для зображень фрактальних візерунків це виражається такими фразами, як «плавно накопичувальні поверхні» та «авитки на завихреннях».

- Прості та «можливо рекурсивні» визначення.

Як група, ці критерії формують вказівки щодо виключення певних випадків, наприклад тих, які можуть бути подібними до себе, не маючи інших типових фрактальних ознак. Наприклад, пряма лінія є самоподібною, але не фрактальною, оскільки їй не вистачає деталей, вона легко описується евклідовою мовою, має такий же гаусдорфський вимір, як і топологічний вимір , і повністю визначається без необхідності рекурсії.

Ще однією важливою складовою фракталів є так звана розмірність Хаусдорфа, або більш конкретно фрактальна розмірність. У фрактальній геометрії, фрактальний розмір – це відношення, що забезпечує статистичний показник складності, порівнюючи те, як деталізація у шаблоні (строго кажучи, фрактальна картина) змінюється зі шкалою, за якою вона вимірюється. Він також був охарактеризований як міра місткості заповнення простору візерунка, який показує те, як фрактал масштабується від простору, в який він вбудований. Фрактальний вимір не повинен бути цілим числом [11].

Суттєва ідея «розбитих» вимірів має довгу історію в математиці, але сам термін був висунутий на перший план Бенуа Мандельбротом на основі його статті 1967 року про самоподібність, в якій він обговорював дробові розміри . У цій роботі Мандельброт цитував попередню роботу Льюїса Фрі Річардсона, описуючи антиінтуїтивне уявлення про те, що вимірювана довжина берегової лінії змінюється з довжиною використовуваної вимірювальної палиці [1]. Це можна помітити на рис. 1.2.

Рис. 1.2. Зміна довжини берегової лінії

З огляду на це поняття, фрактальний розмір берегової лінії кількісно визначає, як кількість масштабованих вимірювальних паличок, необхідних для вимірювання зміни берегової лінії, змінюється зі шкалою, нанесеною на палицю. Існує кілька формальних математичних визначень фрактального виміру, які ґрунтуються на цій базовій концепції зміни детально зі зміною масштабу.

Зрештою, термін фрактальний вимір став фразою, з яку сам Мандельброт визначив найзручнішою щодо інкапсуляції значення слова фрактал, терміна, який він створив. Після кількох ітерацій протягом багатьох років Мандельброт зупинився на такому використанні мови: "... використовувати фрактал без педантичного визначення, використовувати фрактальний вимір як загальний термін, застосовний до всіх варіантів"[24, с.46].

Один нетривіальний приклад - фрактальний вимір сніжинки Коха рис.1.3. Він має топологічний вимір 1, але це аж ніяк не звичайна крива.

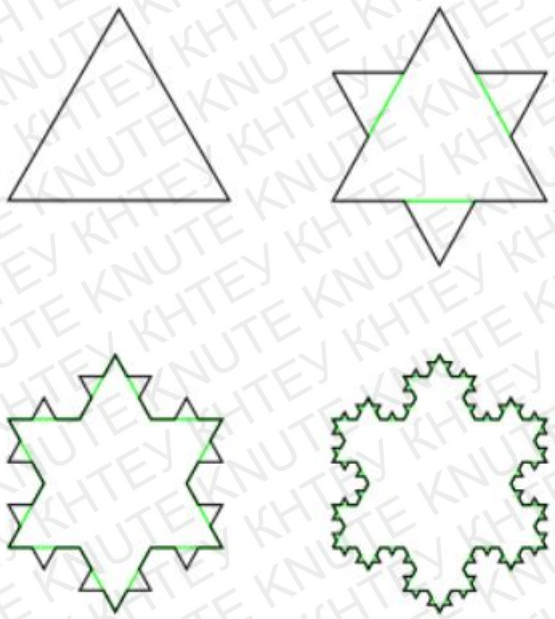

Рис. 1.3. Сніжинка Коха, 1-4 ітерація

Довжина кривої між будь-якими двома точками на сніжинці Коха нескінченна. Жоден невеликий шматок цієї кривої не схожий на лінію, а складається з нескінченної кількості сегментів, з'єднаних під різними кутами. Фрактальний вимір кривої можна пояснити інтуїтивно, розглядаючи фрактальну лінію як об'єкт, занадто детальний, щоб бути одновимірним, але занадто простим, щоб бути двовимірним. Тому його розмір найкраще можна описати не його звичайним топологічним виміром 1, а фрактальним виміром, який часто є числом між одним і двома; у випадку сніжинки Коха - це приблизно 1,226.

Також хотілося б відзначити ще одну особливість фракталів, а саме застосування комплексних чисел.

Комплексне число – це число, яке може бути виражено у вигляді $a + bi$, де a і b є дійсними числами, а i є рішенням рівняння $x^2 = -1$. Оскільки жодне реальне число не задовольняє це рівняння, i називається уявним числом . Для комплексного числа $a + bi$, a називається реальною частиною, а b називається уявною частиною [22]. Незважаючи на досить таку незвичну номенклатуру "уявне", комплексні числа розглядаються в математичних науках настільки ж

"реальними", як і звичайні числа до яких усі звикли, і є основоположними в багатьох аспектах наукового опису природного світу.

1.2. Різновиди фракталів

Існує багато різних типів фрактальних формул, проте із них однозначно можна виділити три основні типи, а саме:

- алгебраїчні;
- геометричні;
- стохастичні;

Алгебраїчні фрактали – найбільша група фракталів. Одержують їх за допомогою нелінійних процесів у *n*-мірних просторах. Як можна зрозуміти із раніше написаного ці фрактали можна створити, обчисливши рівняння знову і знову. Оскільки рівняння має бути повторене тисячі разів, комп'ютерам необхідний час на обрахування, щоб потім можна було дослідити та побачити мальовниче зображення фрактала. Одразу після того, як були винайдені персональні комп’ютери, можна було створити комп'ютерні фрактали. Напевно найбільш відомими представниками цього типу є *Mandelbrot set, Julia set* та *Burning Ship*.

Два провідних дослідники у галузі складних числових фракталів - Гастон Моріс Джулія та Бенуа Мандельброт. Не важко здогадатись, що саме в їхню честь і було названо перших два фрактали наведені вище.

Гастон Моріс Джулія народився наприкінці 19 століття в Алжирі. Він провів своє життя, вивчаючи ітерацію поліномів та раціональних функцій [16]. Близько 1920-х років, опублікувавши статтю про ітерацію раціональної функції, Юлія прославилася. Однак після смерті його забули.

У 1970-х роках творчість Гастона Моріса Джулії була відроджена та популяризована поляком Бенуа Мандельбротом. Надихнувшись роботою Джулії та за допомогою комп’ютерної графіки, працівник IBM Мандельброт зміг

показати перші зображення фракталів, котрі зображені на рис.1.4. і є найвідомішими на сьогодні.

Рис. 1.4 Перше зображення фракталу Мандельброт

Множина Мандельброта – це сукупність точок на комплексній площині. Щоб побудувати набір Мандельброта, необхідно використовувати алгоритм, котрий засновано на рекурсивній формулі:

$$Z_n = Z_{n-1}^2 + C \quad (1.1)$$

На комплексній площині відбувається розділення точок на дві категорії:

- точки всередині набору Мандельброта;
- точки поза набором Мандельброта.

На рис. 1.5. показана частина складної площини. Точки набору Мандельброта пофарбовані в чорний колір.

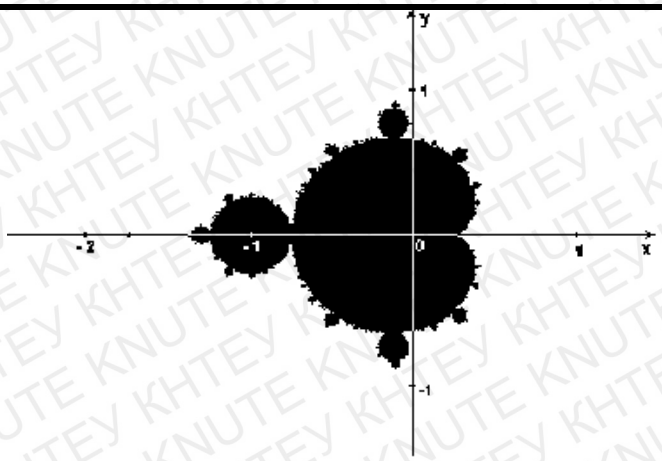

Рис. 1.5. Область значень Мандельброта

Можна також призначити колір точкам поза набором Мандельброта. Їх кольори залежать від того, скільки ітерацій потрібно, щоб визначити, що вони знаходяться поза межами набору. Приклад того як це може виглядати наведено на рис. 1.6.

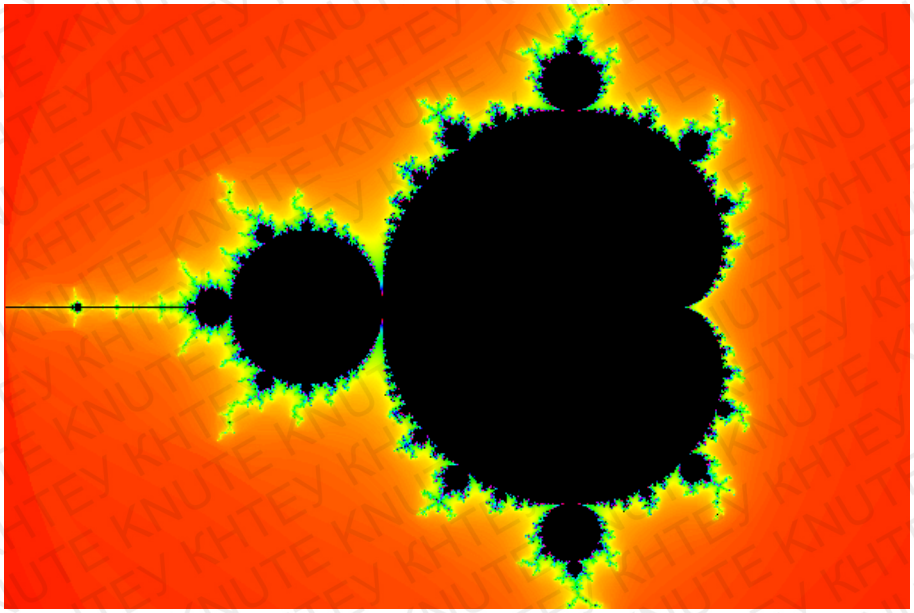

Рис. 1.6. Колір в залежності від ітерацій

Набори Джулії суворо пов’язані з набором Мандельброта. Ітераційна функція, яка використовується для їх отримання, така ж, як і для набору Мандельброта. Єдина відмінність – спосіб використання цієї формули. Для того, щоб намалювати картину множини Мандельброта, ми повторюємо формулу для кожної точки C складної площини, завжди починаючи *з* $Z_0 = 0$. Якщо ми хочемо

зробити зображення набору Джулії, *C* повинне бути постійним протягом усього процесу, тоді як значення Z_0 змінюється. Значення C визначає форму набору Джулії. Інше кажучи, кожна точка комплексної площини пов'язана з певним набором Джулії. Виконавши цю умову можна отримати зображення наведене на рис. 1.7.

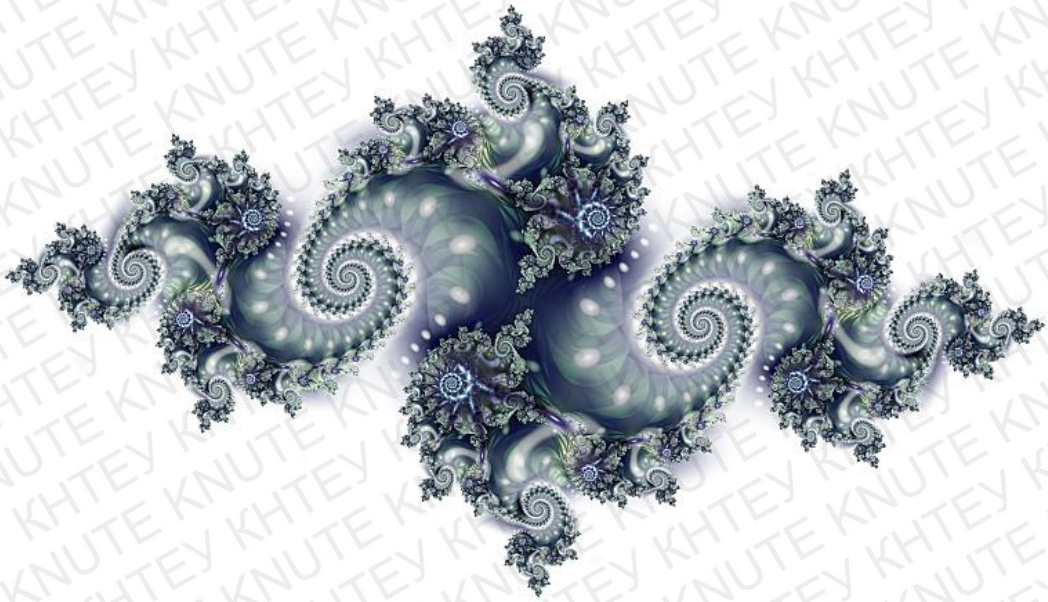

Рис. 1.7. Фрактал «Джулія»

Фрактал палаючий корабель зображений на рис 1.8., створений і вперше описаний Майклом Мішелічем та Отто Е. Росслером у 1992 році, генерується шляхом ітерації функції:

$$Z_{n+1} = (|\mathrm{Re}(Z_n)| + i|\mathrm{Im}(Z_n)|)^2 + c, \quad Z_0 = 0 \tag{1.2}$$

Відмінність цієї функції від її відповідника для множини Мандельброта зводиться до застосування модуля до дійсної і уявної частини перед піднесенням до квадрату в кожній ітерації. Відобразивши його візуально можна отримати наступне зображення:

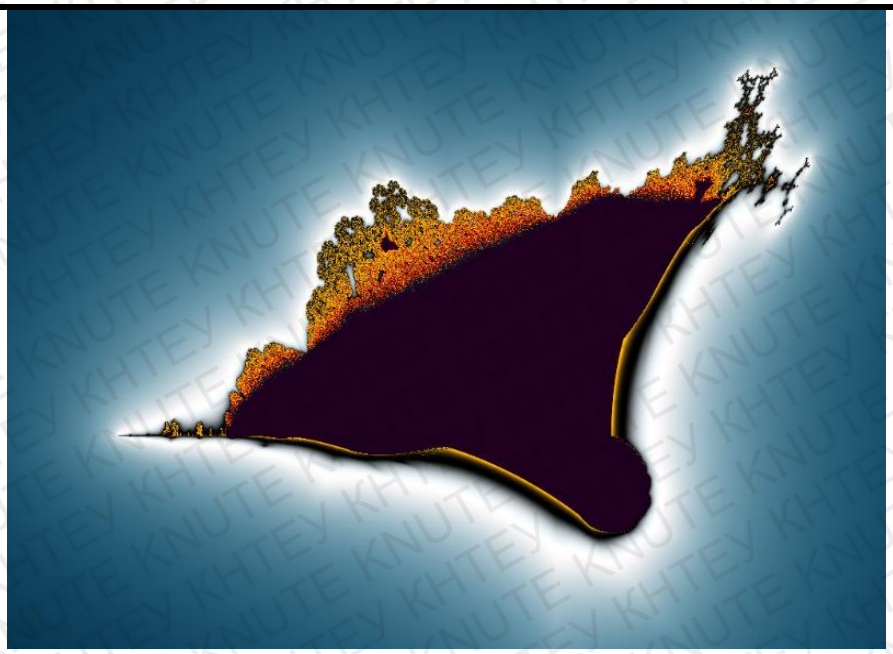

Рис 1.8. Фрактал «Палаючий корабель»

Здавалося б, чому ж цей фрактал назвали «Палаючим кораблем»? А чому це так – можна побачити на рис. 1.9. Дане зображення, отримується при наближенні.

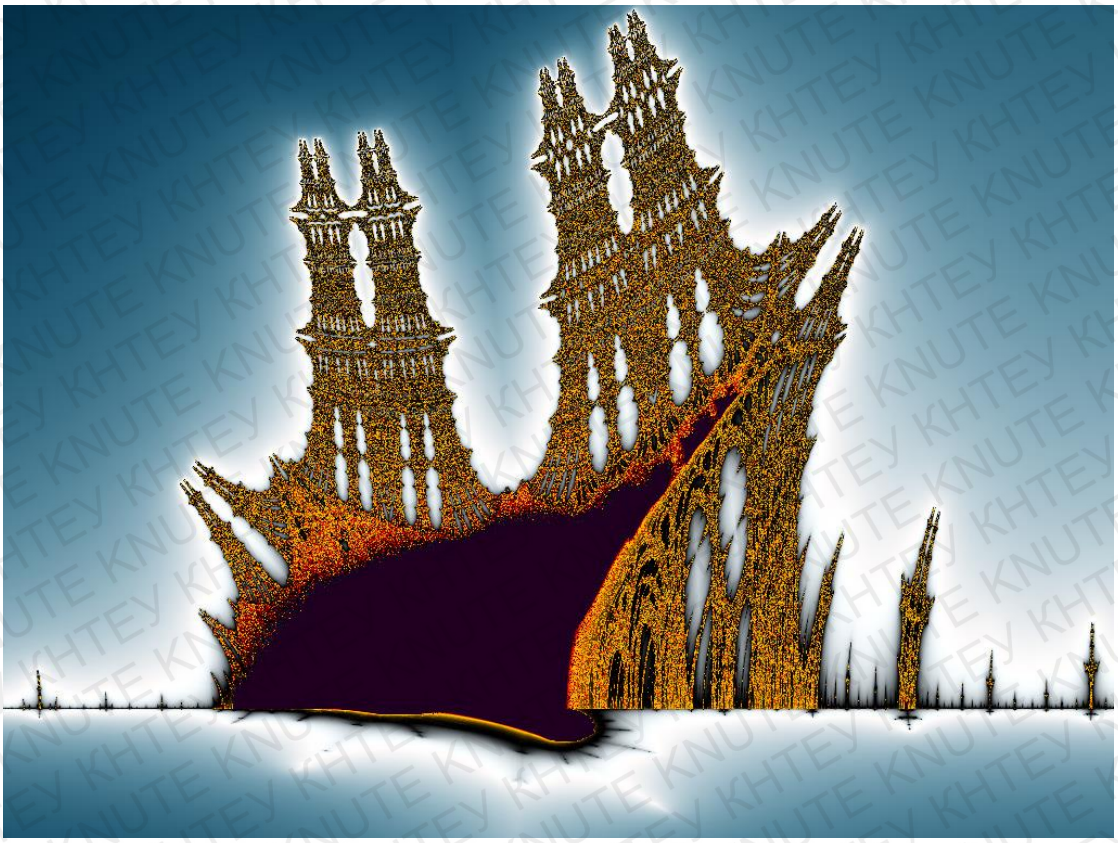

Рис. 1.9. «Справжній корабель»

Наступним типом є геометричні фрактали, вони створюються за допомогою повторення процесу або малюнка знову і знову. Фрактали цього типу найнаочніші. Два найпростіші фрактали, які можна створити, - це трикутник Серпінського, сніжинка фон Коха та Крива Гільберта-Пеано.

Трикутник Серпінського, що зображений на рис. 1.10, також називається прокладкою Серпинського або Серпінським ситом, є фракталом із загальною формою рівностороннього трикутника, розділеного рекурсивно на більш дрібні рівносторонні трикутники. По факту він являється математично сформованим візерунком, який відтворюється при будь-якому збільшенні чи зменшенні. Він названий на честь польського математика Вацлава Сьєрпіньського, але з'явився, як декоративний візерунок за багато століть до творчості Сєрпіньського [8].

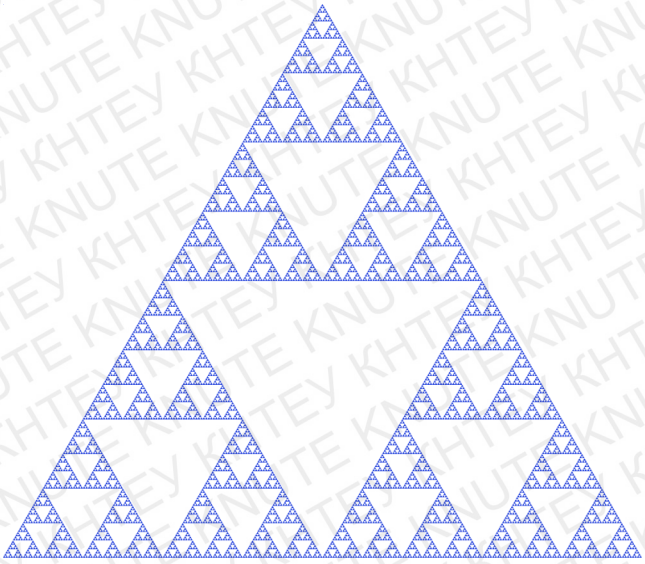

Рис 1.10. Фрактал «Трикутник серпінського»

Ще один відомий тип фракталів – це стохастичні фрактали, які отримують, якщо в процесі ітерації випадковим чином змінюють будь-який з його параметрів. Це створює об’єкти, дуже схожі на природні - асиметричні дерева, вирубані берегові лінії тощо. Найвідомішим фракталом з цього типу є папороть Барнслі [3].

Папороть Барнслі — фрактал названий в честь британського математика Майкла Барнслі, який вперше описав його у своїй книзі Fractals Everywhere.

Рис. 1.11. Фрактал «Папороть Барнслі»

1.3. Вплив фракталів на різні галузі

Сфера застосування фракталів людиною досить велика. Вони набули великого поширення в абсолютно різних галузях, таких як природнича, медична, телекомунікаційна, комп'ютерна і тд.

Прикладом застосування фракталів в природничих науках є їх застосування в астрофізиці. Адже ніхто насправді не знає, скільки зірок насправді блищать на нашому небі. Астрофізики вважають, що ключем до цієї проблеми є фрактальний характер міжзоряного газу. Фрактальні розподіли є ієрархічними, як димові стежки або хвилясті хмари на небі. Турбулентність формує як хмари на небі, так і хмари в просторі, надаючи їм неправильний, але повторюваний малюнок, який неможливо було б описати без допомоги фрактальної геометрії. Проте більш близьким прикладом для розуміння буде застосування фракталів для моделювання турбулентності, котра візуально показана на рис.1.12 [26].

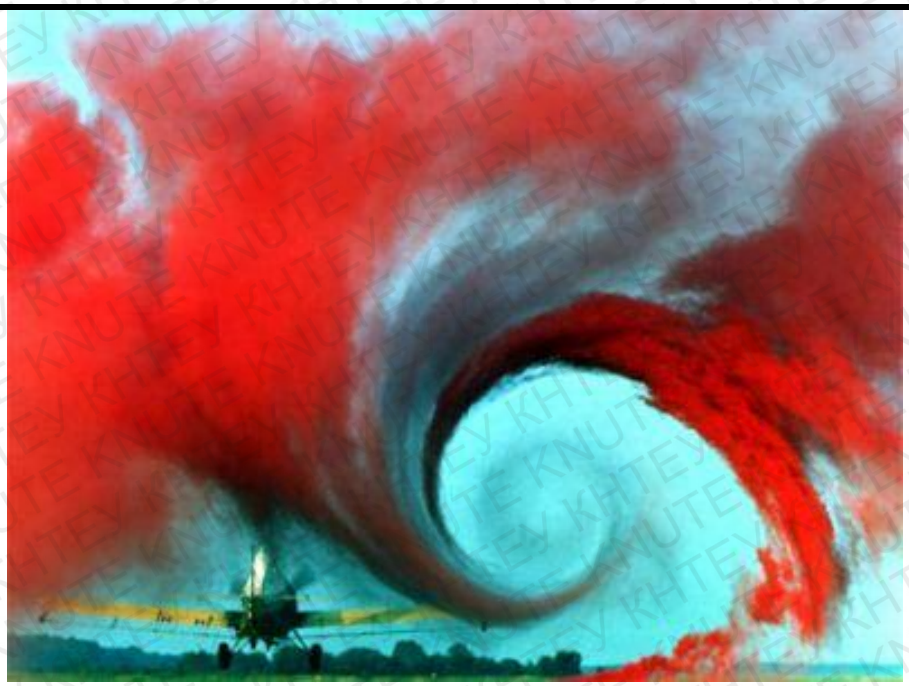

Рис. 1.12. Турбулентність

Фрактали набирають все більше поширення і в медицині. У зв’язку з тим, що людький організм має багато структур подібних на фрактали наприклад кровоносна система, м’язи, легкі і т.п. На прикладі бронхів рис. 1.13. можна помітити їхню фрактальність. А тому вчені задумались над тим чи не можна їх застосовувати і для вирішення медичних проблем, таких як діагностики або лікування. Дослідження показали, що це цілком можливо. Так, досліджуючи кардіограму за допомогою фракталів можна більш точно та на ранній стадії спрогнозувати порушення гомеостазису та розвиток певних захворювань [15].

Рис. 1.13. Бронхи

У телекомунікаціях фрактальну геометрію використовують при проектуванні антенних пристроїв. Головною перевагою яких є багатодіапазонна і порівняльна широкополосність. Приклад такої антени наведено на рис. 1.14 [4].

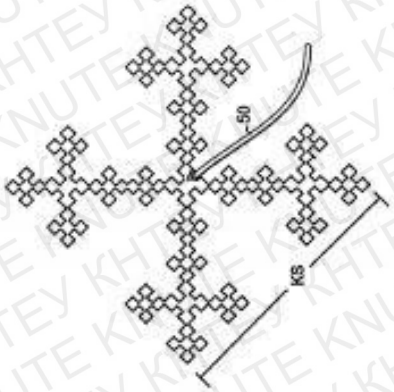

Рис. 1.14. Фрактальна антена

Як не дивно найбільше поширення фрактали зазнали саме в комп'ютерній галузі. Вони тут застосовуються для децентралізованих мереж, а саме для призначення IP-адрес в мережі Netsukuku, де успішно використовується принцип фрактального стиснення. Також, фрактальне стиснення використовується і для стиснення зображень у інформатиці, коли замість початкового зображення можна зберігати стискуюче відображення, для якого зображення є нерухомою точкою [5]. Так на рис. 1.15. наведено приклад фрактального стиснення.

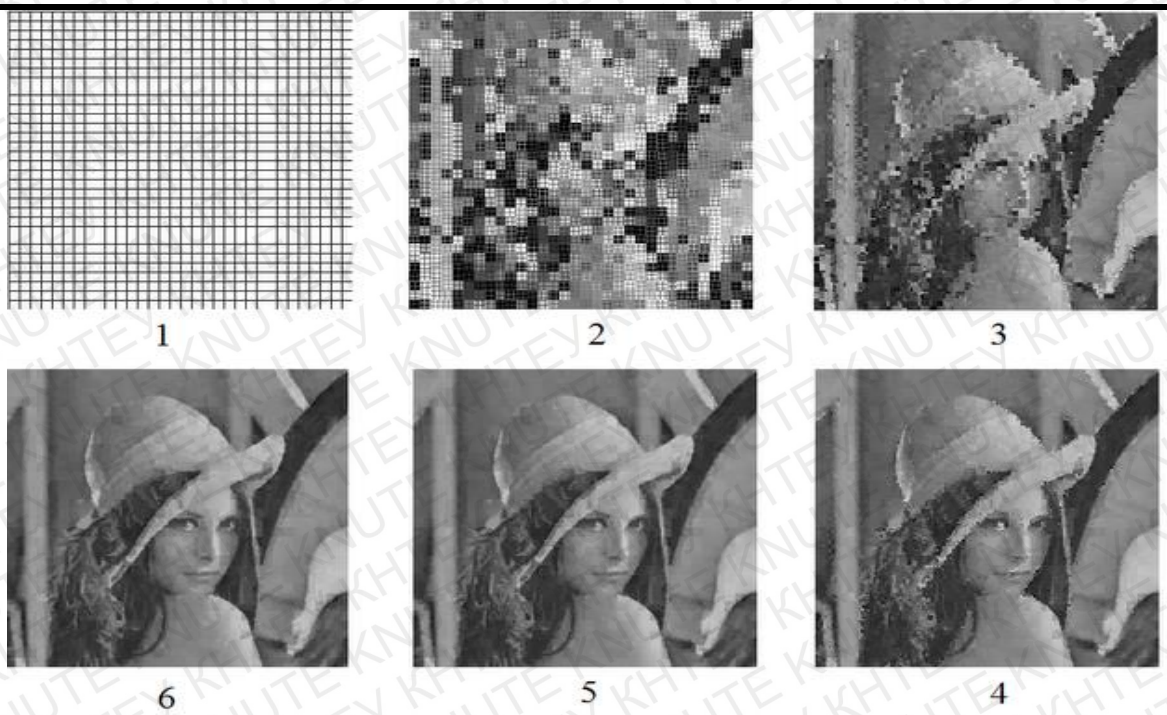

Рис. 1.15. Фрактальне стиснення

І важко не згадати про комп'ютерну графіку, де фрактали широко застосовуються для побудови зображень природніх об'єктів, таких як дерева, кущі, гірські ландшафти, поверхні морів і так далі. Безперечною перевагою фрактальної графіки є те, що у файлі фрактального малюнка зберігаються тільки алгоритми та формули. Тому розміри цих файлів значно менші, ніж файли де використовується векторна або растова графіка.

1.4. Технічне завдання

1. Загальні відомості:

- Повна назва: «Програма візуалізації фракталів «miracle»;
- Коротка назва: «miracle»;
- Початок робіт: 01.03.2020;
- Кінець робіт: 01.05.2020;
- Фінансування: відсутнє;

- Потенційні користувачі: потенційними користувачами програми є компанії, які працюють у сфері gamedev та пересічні зацікавленні люди.

2. Мета та призначення створення програми:

- Призначення програми: візуалізація фракталів для їх подальшого використання у сфері gamedev в якості текстур;
- Мета створення: можливість отримання прибутку від програми в майбутньому, а також поглиблення навичок в області програмування графіки.

3. Перелік необхідних функцій:

- Робота програми без багів;
- Відсутність витоків пам’яті;
- Простота взаємодії користувача з програмою;
- Можливість зміни ітерацій;
- Можливість використовувати колесо миші для збільшення/зменшення;
- Можливість зміни кольору фракталу;
- Можливість рухати фрактал.

4. Вимоги до програми:

- Програма повинна якісно візуалізувати вибраний фрактал, для того щоб його можна було потім використовувати у цілях користувача.

5. Вимоги до програмного забезпечення:

- Наявність встановленої графічної бібліотеки SDL 2.0;
- Наявність Microsoft Visual C++ 2015-2019 Redistributable (x64);
- 64-розрядна операційна система Windows 7/8(8.1)/10.

6. Вимоги до технічного забезпечення:

- Процесор на рівні Intel Core i7 3770 3,4 Ггц або AMD FX-8350 4 Ггц;

- Відеокарта рівня Nvidia GeForce GTX 770 або AMD Radeon R9 290;
- 2 гігабайта оперативної пам’яті;
- 100 мегабайтів вільного простору на жорсткому диску.

1.5. Висновки до розділу 1

У першому розділі були проаналізовані загальні відомості та характеристики фракталів, їх види та вплив, що вони створюють на різні галузі. Таким чином, було виявлено, що хоч фрактали і оточують людей у повсякденному житті, їх почали досліджувати відносно нещодавно. Проте не дивлячись на це, науковою спільнотою відкрито велике різноманітя фракталів. Також було виявлено, що фрактали мають чималий вплив на різні галузі, які розвиваються за допомогою них. Також автором було складено технічне задання для розробки програми з візуалізації фракталів.

РОЗДІЛ 2.
АНАЛІЗ ПРЕДМЕТУ ДОСЛІДЖЕННЯ ТА ХАРАКТЕРИСТИКА СФЕРИ GAMEDEV

2.1. Застосування фракталів у GameDev

Під час розробки будь-якої гри у якій є елемент дослідження світу, що оточує гравця, для багатьох розробників одним із найважливіших моментів є збереження у гравця бажання досліджувати світ гри. Важливими пунктами для збереження цього бажання є цікаві квести та схожість із реальним світом. Коли розробники вирішують, що аби зацікавити гравця поринути у світ гри необхідно, щоб природні види у віртуальному просторі були максимально наближенні до образів реального світу. Саме в один із таких моментів і стають в нагоді фрактали.

Сьогодні у багатьох іграх, де є різні види природних ландшафтів, використовуються фрактальні алгоритми. Цей метод виявився досить ефективним. Справа в тому, що реальні природні об’єкти в основному мають фрактальну структуру. Взявши це за основу, програмісти намагалися створити комп'ютерні пейзажі на основі фрактальних алгоритмів. Спостерігаючи сьогоднішнє різноманіття ігор, де можна спостерігати прекрасні природні ландшафти, можна однозначно сказати, що вони досягли успіху. Напевно, найкращим прикладом може виступити відео гра Minecraft скріншот із якої наведено на рис. 2.1.

Рис. 2.1. Minecraft

Один із способів створення такого ландшафту – використання алгоритму переміщення випадкових середніх точок, в якому квадрат поділяється на чотири менших квадрати, а центральна точка вертикально зміщується на деяку випадкову величину. Даний процес повторюється на чотирьох нових квадратах до тієї пори, поки не буде досягнуто бажаного рівня деталізації. Існує багато фрактальних процедур. Наприклад, поєднання декількох октав шуму Simplex, здатних створювати дані про місцевість, проте термін "фрактальний пейзаж" став більш загальним [12]. На рис. 2.2. можна побачити результат цього алгоритму при створенні гірського пейзажу.

Рис. 2.2. "Фрактальний пейзаж"

Також варто відзначити, що фрактали у сфері GameDev застосовуються і для створення об’ємних текстур. Також, ще одним із способів використання є встановлення фрактального зображення у вигляді заднього фон, або як додатковий ефект в меню. Проте, це застосовується досить не часто.

Одне із нововведень, котре принесли фрактали в сферу GameDev і котре зараз просувається ентузіастами програмістами є створення ігрового двигуна фрактальної фізики котрий дозволяє швидко обраховувати колізії з фракталами та іншими процедурно створеними об’єктами. Так, на рис. 2.3. наведено скриншот із відеогри побудованої на прототипі такого двигуна.

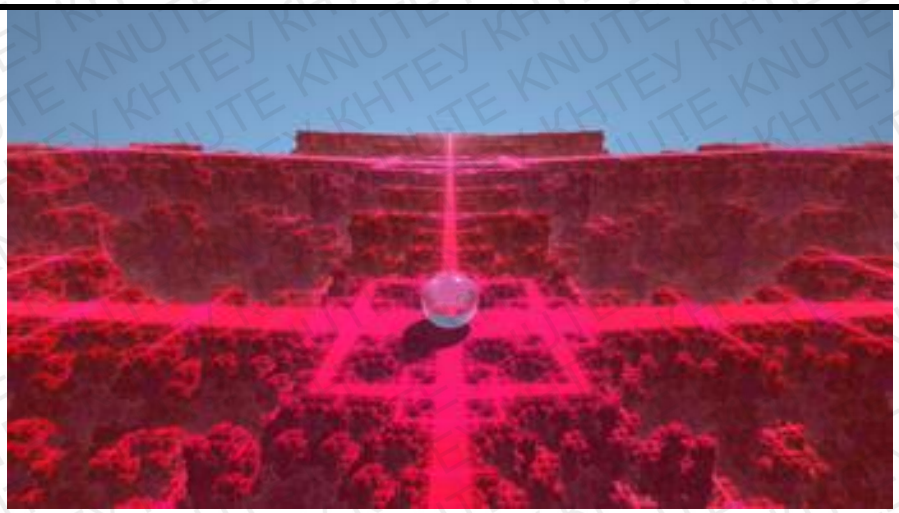

Рис. 2.3. Гра побудована на фрактальному двигуні

2.2. Постановка задачі

Необхідно написати програму для візуалізації фракталів. Згідно до вподобань автора, дана програма буде використовувати командну строку для зчитування даних. Ці данні міститимуть інформацію про те, який фрактал із списку (список буде реалізовано підчас розробки програмного продукту) необхідно візуалізувати. У випадку, якщо користувач не вибиратиме фрактал із списку, або вводитиме невірні дані. Програма повинна показати користувачеві помилку та запропонувати знову вибрати один із фракталів.

Для полегшення користування програмою новими користувачами, необхідно створити меню, у котрому буде інформація про всі діючі кнопки для того чи іншого фракталу.

Для більшості фракталів повинна бути реалізована можливість наближення або віддалення до фракталу.

Для усіх має фракталів має бути реалізова можливість зміни таких пунктів як:

- кількість ітерацій;
- зміна кольору;
- пересування за допомогою «стрілок» на клавіатурі.

Також для фракталу Джулія необхідно реалізувати можливість керування певною клавішою ввімкнення режиму зміни зображення в залежності від позиції комп’ютерної миші на екрані.

2.3. Основна характеристика сфери GameDev

GameDev (game development) – процес розробки відеоігор. Під час цього процесу зусилля можуть прикладатись як однієї людиною, так і міжнародною компанією у якій працює величезна кількість людей з різних куточків світу. Світовий ринок відеоігор 2019 року, за даними спеціалізованої аналітичної компанії Newzoo, оцінюється в 148,1 млрд доларів.

Перші відеоігри, розроблені в 1960-х роках, були некомерційними і не були доступні широкій публіці. Комерційний розвиток відеоігор почався в 70-х роках минулого століття з появою першого покоління ігрових консолей і ранніх домашніх комп'ютерів, таких як Apple I. У той час, завдяки низьким витратам і малим можливостям комп’ютерів, самотній програміст міг розробити повноцінну і повну гру. Однак в кінці 80-х і 90-х, постійно зростаюча потужність комп'ютерної обробки та посилення очікувань від геймерів ускладнювали створення відеоігор Середня вартість виробництва відеоігри рівня ААА повільно зростала - з 1–4 млн. Дол. США у 2000 р. до понад 5 млн. Дол. США у 2006 р, а потім до понад 20 млн. Дол. США до 2010 р [34, с.7].

Розробка повноцінної гри зазвичай передбачає команду складом від 20 до 100 осіб з різними обов'язками, включаючи дизайнерів, художників, програмістів та тестувальників.

Ігри виробляються в процесі розробки програмного забезпечення. Ігри розробляються для отримання прибутку, зазвичай розробка фінансується видавцем. Видавець зацікавлений в тому щоб гра охопила як найбільшу кількість гравців, та буда зроблена якісно і вчасно, адже добре зроблені ігри приносять прибуток легше. Однак важливо оцінити фінансові потреби гри, такі як витрати на розробку окремих особливостей. Адже, якщо не правильно встановити

очікування від гри і витратити на створення надто велику суму, то таким чином можна втратити гроші. Насправді більшість комерційних ігор не приносять прибутку. Більшість розробників не можуть дозволити собі змінити графік розвитку посередині, і вимагають оцінити свої можливості за наявними ресурсами перед виробництвом

Ігрова індустрія вимагає інновацій, оскільки видавці не можуть отримати прибуток від постійного випуску повторюваних сіквелів та наслідувань. Щороку з'являються нові незалежні компанії і деякі з них примудряються створювати хіти.

Проте, багато розробників закриваються, оскільки вони не можуть знайти видавничий договір або їх виробництво не є прибутковим. Важко створити нову компанію через великі розміри початкових інвестицій. Тим не менш, зростання ринку казуальних і мобільних ігор дозволило розробникам з меншими командами вийти на ринок. Як тільки компанії стають фінансово стабільними, вони можуть розширюватися, щоб розвивати проекти побільше. Більшість розробників починають з малого і поступово розширюють свій бізнес. Розробник, який отримує прибуток від успішного проекту, може накопичувати капітал для розширення та переобладнання своєї компанії, а також стійкіше переносити можливість провалювання термінів.

Середній бюджет необхідний для створення крос платформної гри становить від 18 до 28 млн. Дол. США , а деякі надто гучні ігри часто перевищують 40 млн. Дол [34, с.34].

Розробка, виробництво чи дизайн ігор – це процес, що починається від ідеї чи концепції. Часто ідея заснована на модифікації уже існуючої концепції гри. Ідея гри може входити в один або кілька жанрів. Дизайнери часто експериментують з різними поєднаннями жанрів. Дизайнер ігор, як правило, пише початковий сценарій із ігровими пропозиціями, в яких описується основна концепція, геймплей, список функцій, параметри та історія, цільова аудиторія,

вимоги та графік роботи, і нарешті кошторис персоналу та бюджету. У різних компаніях є різні формальні процедури та філософії щодо дизайну та розробки ігор. Не існує стандартизованого методу розробки; однак спільні існують.

Існують, як незалежні, так і видавничі студії розробки. Незалежні розробники покладаються на фінансову підтримку видавця ігор. Зазвичай їм доводиться розробляти гру від концепції до прототипу без зовнішнього фінансування. Потім формальна пропозиція щодо гри подається видавцям, які можуть фінансувати розвиток гри від декількох місяців до декількох років. Видавець зберігає за собою ексклюзивні права на розповсюдження та продаж гри, а також часто володіє правами інтелектуальної власності на ігрову франшизу. Компанія видавця може також володіти компанією розробника, або вона може мати студії з внутрішньої розробки. Зазвичай видавець — той, хто володіє правами інтелектуальної власності на гру.

Усі компанії, окрім найменших компаній розробників, працюють над декількома заголовками одночасно. Це необхідно через час, який проходить між доставкою гри та отриманням платежів за роялті, який може становити від 6 до 18 місяців. Невеликі компанії можуть складати договори, просити аванси на роялті, використовувати розподіл програмного забезпечення, використовувати працівників, які працюють за сумісництвом, та використовувати інші методи для задоволення потреб у оплаті праці.

Виробники консолей, такі як Microsoft, Nintendo або Sony, мають стандартний набір технічних вимог, яким повинна відповідати гра для затвердження. Крім того, концепція гри повинна бути затверджена виробником, який може відмовитись у затвердженні певних назв.

Розробка більшості сучасних ігор для ПК або консолей займає від трьох до п'яти років. В той самий час, як мобільна гра може бути розроблена за кілька місяців. На тривалість розвитку впливає ряд факторів, таких як жанр , масштаб, платформа розвитку та кількість активів.

Деякі ігри можуть зайняти набагато більше часу, ніж середній термін виконання. Наприклад сумнозвісний приклад 3D Realms «Duke Nukem Forever», оголошений у виробництво в квітні 1997 року і випущений чотирнадцять років по тому, в червні 2011 року. Також як гра Prey була коротко проаналізована у 1997 році випуску PC Gamer, але випущена тільки у 2006 році, і навіть тоді в сильно зміненому вигляді. Нарешті, Team Fortress 2 наведеної на рис. 2.4, котрий розроблявся Гейбом Ньюеллом з 1998 р. до випуску аж у 2007 р [34, с.62].

Рис. 2.4. Team Fortress 2

Дохід від гри у роздрібній торгівлі розподіляється між сторонами по ланцюжку дистрибуції, такими як: розробник, видавець, роздріб, виробник та консольні роялті. Багато розробників не отримують прибутку від цього і банкрутіють. Багато розробників шукають альтернативні економічні моделі через інтернет-маркетингу та каналів розповсюдження для покращення прибутку, оскільки через канал мобільного розповсюдження частка розробника може становити до 70% від загального доходу. а через онлайн-канал розповсюдження майже усі 100% [34, 98].

2.4. GameDev в Україні

Стан української галузі з розробки відеоігор завжди був якщо не на найвищому то на вищому рівні. Напевно не знайти геймера із України котрий не чув би про таку вітчизняну компанію, як GSC Game World. Котра випустила дві

культові ігрові серії, так першим її дітищем стала гра «Козаки» скриншот із якої наведено на рис. 2.5.

Рис. 2.5. «Козаки»

Проте дана гра однозначно не набула такої популярності у гравців, як наступна культова гра випущена компанією у 2007 році, а саме «S.T.A.L.K.E.R. - Тінь Чорнобиля» котра наведена на рис. 2.6. Дану гру офіційно можна назвати вітчизняним довгобудом, так як вона перенесла величезну кількість переносів та кардинальних змін. Проте це не помішало їй поселитись у серцях фанатів [35, с.41].

Рис 2.6. «S.T.A.L.K.E.R. – Тінь Чорнобиля»

Ще одна компанія, котра дуже відома на світовому ринку, і яка була заснована вихідцями із компанії GSC Game World носить назву 4A Games. Дана українська компанія подарила світу культову ігрову серію «METRO» рис. 2.7 [35 с.77].

Рис 2.7. «METRO 2033»

2.5. Висновки до розділу 2

У другому розділі було проаналізовано застосування фракталів у сфері GameDev та виявлено, що в основному у цій сфері фрактали використовуються для моделювання карт, накладання текстур, а також проводяться спроби створити фрактальний ігровий двигун. Також створено постановку задачі для програми котра буде розроблятись автором для візуалізації фракталів. Досліджено основні характеристики сфери GameDev, та зокрема стан цієї галузі на території Україні.

РОЗДІЛ 3.
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІЗУАЛІЗАЦІЇ ФРАКТАЛІВ

3.1. Вибір мови програмування та середовища для розробки

При написанні програми вибір пав на середовище Visual Studio – це сучасне середовище для програмування багатофункціональних додатків. Існує декілька версій Visual Studio, кожна з яких має певні плюси та орієнтованість. Зокрема використовувалась остання версія середовища Visual Studio 2019 у якій були реалізовані нові можливості, такі як:

- Розробка, котра дозволяє зосереджуватись на головному і підвищувати продуктивність, завдяки оптимізованої продуктивності, можливості миттєвої очищення коду і отримання більш точних результатів пошуку.

- Спільна робота, котра дозволяє користуватись можливостями роботи в рамках робочого процесу Git-first, функціями редагування та налагодження, а також рецензування коду прямо в Visual Studio.

- Налагодження, котре дозволяє виділяти певні значення і переходити до них, оптимізує використання пам'яті і створюйте автоматичні моментальні знімки при виконанні програми.

В якості мови програмування для написання програми буде використовуватись мова С.

С – мова програмування середнього рівня, розроблена Деннісом Річі на початку 1970-х, працюючи в AT&T Bell Labs в США

Мова С стала основою для багатьох мов, включаючи C ++, Java, JavaScript, Go, Rust, Limbo, LPC, C #, PHP, Python, Perl, Verilog та C-shell.

Переваги С

Як мова середнього рівня, С поєднує в собі риси як мов високого, так і низького рівня. Він може використовуватися для програмування низького рівня, наприклад, сценаріїв для драйверів та ядер, а також підтримує функції мов програмування високого рівня, такі як сценарії для програмних програм тощо.

С – це структурована мова програмування, яка дозволяє розбити складну програму на більш прості програми, що називаються функціями. Це також дозволяє вільний рух даних по цих функціях.

Мова С залежить від регістру.

С дуже портативний і використовується для створення сценаріїв системних додатків, які складають основну частину операційної системи Windows, UNIX та Linux.

С – мова програмування загального призначення і може ефективно працювати на корпоративних програмах, іграх, графіці та додатках, що вимагають розрахунків.

Мова С має багату бібліотеку, яка забезпечує ряд вбудованих функцій. Він також пропонує динамічне розподіл пам'яті.

3.2. Вибір графічної бібліотеки

Для написання програми автор вибрав кросплатформену графічну бібліотеку Simple DirectMedia Layer (SDL).

Simple DirectMedia Layer – це графічна бібліотека для розробки, створена для забезпечення низького рівня доступу до аудіо, клавіатури, миші, джойстика та графічного обладнання через OpenGL та Direct3D.

SDL офіційно підтримує Windows, Mac OS X, Linux, iOS та Android. Підтримка інших платформ може бути знайдена у вихідному коді.

SDL написаний на С, працює в основному з С ++, і є прив'язки, доступні для декількох інших мов, включаючи С # і Python.

3.3. Створення програмного продукту

Щоб приступити до написання програми з реалізації фракталів, перш за все необхідно створити консольний проект у Visual Studio 2019. Наступним кроком, після того як шаблон проекту було створено, необхідно підключити графічну бібліотеку SDL2.0. Після того, як ці кроки виконано, можна приступати до написання коду.

Для створення вікна у якому надалі буде відображатись фрактал необхідно використати команди вказані на рис.3.1. Кожна із яких відповідає на певну дію:

- SDL_Init(SDL_INIT_EVERYTING) – для ініціалізації всіх підсистем бібліотеки SDL;
- SDL_CreateWindow() – створює вікно із заданою назвою, розмірами, та розташуванням екрані;
- SDL_GetWindowSurface() – створює масив пікселів розміром x*y, який потім буде використовуватись для передачі зображення.

```
void    init_img(t_f* f)
{
    SDL_Init(SDL_INIT_EVERYTHING);
    f->surface = NULL;
    f->window = SDL_CreateWindow("Miracle", SDL_WINDOWPOS_UNDEFINED,
        SDL_WINDOWPOS_UNDEFINED, W, H, SDL_WINDOW_SHOWN);
    SDL_Renderer* render = SDL_CreateRenderer(f->window, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
    f->surface = SDL_GetWindowSurface(f->window);
    f->open = SDL_TRUE;
}
```

Рис. 3.1. Команди для створення вікна програми

Для створення множини Мандельброта потрібно вибрати складну площину (C). Комплексне число, відповідне їй, має вигляд:

$$C = a + b * i \tag{3.1}$$

Після обчислення значення попереднього виразу:

$$Z_1 = Z_0^2 + C \tag{3.2}$$

Використовуючи нуль як значення , отримуємо C як результат. Наступний крок складається з присвоєння результату та повторення обчислення: тепер

результат - це комплексне число . Тоді нам слід присвоїти значення і повторити процес знову і знову.

Цей процес можна представити як "міграцію" початкової точки C по площині. Що відбувається з моментом, коли ми повторно повторюємо функцію? Чи залишиться він поблизу від походження чи відійде від нього, збільшуючи віддалення від походження без обмежень? У першому випадку ми говоримо, що C належить до множини Мандельброта, в іншому випадку ми говоримо, що це йде до нескінченності, і ми присвоюємо колір C залежно від швидкості, з якою точка "втече" від початку.

Можна поглянути на алгоритм з іншої точки зору. Уявімо, що всі точки на площині притягуються обома: нескінченністю та множиною Мандельброта. Це дозволяє зрозуміти, чому:

- точки, далекі від набору Мандельброта, швидко рухаються до нескінченності,
- точки, близькі до набору Мандельброта, повільно виходять у нескінченність,
- точки всередині набору Мандельброта ніколи не виходять у нескінченність.

Програмним чином за допомогою коду даний фрактал реалізується наступним чином. Для кожного пікселя вікна виконується код вказаний на рис. 3.2:

```
void    mandelbrot(t_f* f)
{
    f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex - 0.5;
    f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
    f->draw.newim = 0;
    f->draw.newre = 0;
    f->draw.oldre = 0;
    f->draw.oldim = 0;
    f->i = -1;
    while (++f->i < f->draw.maxiterations)
    {
        f->draw.oldre = f->draw.newre;
        f->draw.oldim = f->draw.newim;
        f->draw.newre = f->draw.oldre * f->draw.oldre - f->draw.oldim *
            f->draw.oldim + f->draw.cre;
        f->draw.newim = 2 * f->draw.oldre * f->draw.oldim + f->draw.cim;
        if ((f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim) > 4)
            break;
    }
    (f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f->color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
}
```

Рис. 3.2. Код для множини Мандельброта

Функція put_pixel_in_img() використовується для розміщення пекселя певного кольору за координатами x та y. Її реалізація у вигляді коду наведена на рис. 3.3.

```
void    put_pixel_in_img(t_f *f, int color)
{
    f->mass = (Uint8*)f->surface->pixels + f->y * f->surface->pitch + f->x * 4;
    *(Uint32*)f->mass = color;
}
```

Рис. 3.3. Функція put_pixel_in_img()

Щоб побачити результат виконання програми необхідно запустити цикл котрий буде залишати вікно програми видимим. У цьому циклі, для оновлення інформації у вікні необхідно використовувати SDL_UpdateWindowSurfase(), опісля для відстеження різних подій, таких як рух мишки або натискання клавіші використовується SDL_PollEvent(). Якщо користувач робить якусь дію у програмі необхідно очистити інформацію на вікні за допомогою SDL_FillRect() та розпочати обробку дії користувача. У вигляді коду це наведено на рис. 3.4.

```
void    main_sdl_work(t_f* f)
{
    while (f->open)
    {
        SDL_UpdateWindowSurface(f->window);
        if (SDL_PollEvent(&f->event))
        {
            SDL_FillRect(f->surface, NULL, 0x000000);
            wtrite_key(f);
            color(f);
            mouse_move(f);
            draw_fr(f);
        }
    }
    SDL_DestroyWindow(f->window);
    SDL_Quit();
}
```

Рис. 3.4. Головний цикл

У зв’язку з тим, що папороть Барнслі належить до іншого типу фракталів ніж Мандельброт її побудова відрізняється. У теорії папороть Барнслі може бути побудована вручну за допомогою ручки та паперу з графіком, але так як кількість необхідних ітерацій переходить у десятки тисяч, використання комп'ютера є практично обов'язковим. Багато сучасних комп’ютерних моделей папороті Барнслі популярні у сучасних математиків. Поки математика буде запрограмована правильно, використовуючи матрицю констант Барнслі, котра вказана на рис. 3.5, вийде така ж форма як і у папороті.

ш	a	б	c	г	e	f	p
f_1	0	0	0	0,16	0	0	0,01
f_2	0,85	0,04	−0.04	0,85	0	1,60	0,85
f_3	0,20	–0,26	0,23	0,22	0	1,60	0,07
f_4	−0,15	0,28	0,26	0,24	0	0,44	0,07

Рис. 3.5. Матриця констант Барнслі

Перша намальована точка знаходиться у початку (x 0 = 0, y 0 = 0), а потім нові точки ітераційно обчислюються випадковим чином, застосовуючи одне з

чотирьох перетворень координат. Перше перетворення координат малює стебло. Друге генерує послідовні копії стебла і нижньої плойки, щоб зробити повну папороть. Третє малює нижній фронт зліва. Четверте малює нижній фронт праворуч. Рекурсивний характер даного типу фракталів гарантує, що ціле - це більша копія кожного фронту. Варто зауважити, що повна папороть знаходиться в межах –2.1820 < x <2.6558 та 0 ≤ y <9.9983.

У вигляді коду описані вище дії показані на рис.3.6.

```
for (i = 0; i < POINTS; i++)
{
    random = drand1();

    if (random < threshold[0])
    {
        dx = (0.85 * p[0]) + (0.04 * p[1]) + 0.0;
        dy = (-0.04 * p[0]) + (0.85 * p[1]) + 1.6;
    }
    else if (random < threshold[1])
    {
        dx = (0.20 * p[0]) - (0.26 * p[1]) + 0.0;
        dy = (0.23 * p[0]) + (0.22 * p[1]) + 1.6;
    }
    else if (random < threshold[2])
    {
        dx = (-0.15 * p[0]) + (0.28 * p[1]) + 0.0;
        dy = (0.26 * p[0]) + (0.24 * p[1]) + 0.44;
    }
    else
    {
        dx = (0.00 * p[0]) + (0.00 * p[1]) + 0.0;
        dy = (0.00 * p[0]) + (0.16 * p[1]) + 0.0;
    }
    p[0] = dx;
    p[1] = dy;
    x = dx * f->barn.multw + win_w / 2;
    y = dy * f->barn.multh;
    setPixel(f, f->barn.surface, x, 800 - y, f->barn.r, f->barn.g, f->barn.b);
}
f->barn.lastMouse_x = mx_c;
f->barn.lastMouse_y = my_c;
SDL_UpdateTexture(f->barn.texture, NULL, f->barn.surface->pixels, f->barn.surface->pitch);
SDL_RenderCopy(f->barn.renderer, f->barn.texture, NULL, NULL);
```

Рис.3.6. Перетворення координат згідно матриці Барнслі

3.4. Опис функціональних характеристик

У зв’язку із тим що дана програма являється консольним додатком щоб її запустити необхідно спершу відкрити консоль. Це можна зробити натиснувши комбінацію клавіш «Win» + R та ввівши у рядок команду cmd, що і наведено на рис. 3.7.

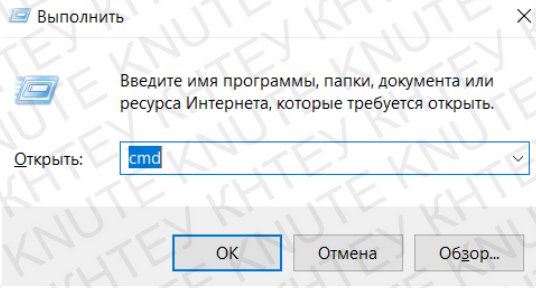

Рис.3.7. Запуск консолі

Після запуску консолі необхідно перейти у папку де розташовується програма і запустити її. Якщо користувач хоче побачити список доступних фракталів йому необхідно запусти програму без параметру. Приклад того як це виглядає наведено на рис. 3.8.

```
C:\Users\User\Desktop\діч(1)\fractal_win\test_sld\Debug>Miracle.exe
Enter number fractal from the list
1 Julia
2 Mandelbrot
3 Mandelbrot2
4 MandelbrotSin
5 Ship
6 Ship2
7 Dich
8 Barnsley
```

Рис. 3.8. Список фракталів

При запуску із параметром буде створено вибраний фрактал. Таким чином, якщо ввести параметром «1» відкриється фрактал Джулія рис 3.9.

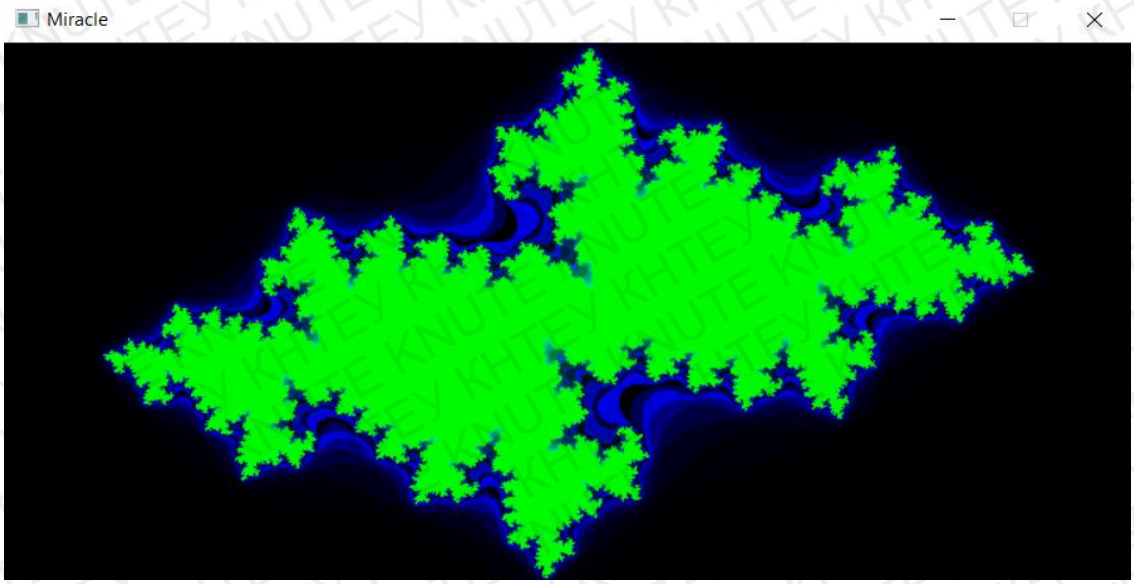

Рис. 3.9. «1 Julia»

Одночасно із вікном фракталу у консолі буде представлено функціональне меню. Його вигляд наведено на рис. 3.10.

```
C:\Users\User\Desktop\діч(1)\fractal_win\test_sld\Debug>Miracle.exe 1

For everyone
Use mouse wheel for zooming
Top 1-9 for change color
S for start position
Right + for add iterations
Right - for minus iterations
Arrows for moving

Only for Julia
R for start changing
P for stop changing
```

Рис. 3.10. Меню

Якщо вибрати фрактал Julia і натиснути на клавіатурі клавішу R, можна запустити зміну вигляду фракталу в залежності від координат курсору миші і отримати результат, як на рис. 3.11.

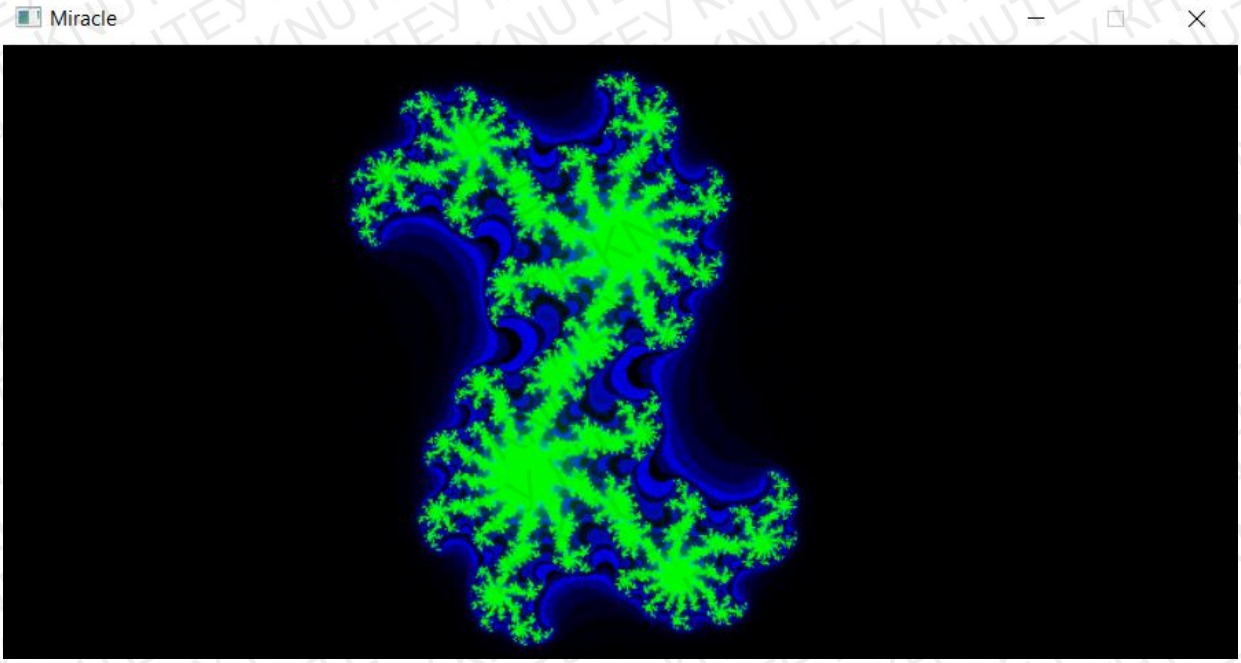

Рис. 3.11. Ввімкнена залежність від курсору миші

Обрахунок цих змін відбувається за допомогою формул наведених на рис. 3.12.

```
f->draw.cim = 0.008 * (double)(H / 2 - f->event.button.y);
f->draw.cre = 0.004 * (double)(W / 2 - f->event.button.x);
```

Рис. 3.12. Формули залежності

Верхній список клавіш із нумерацією 1-9 відповідає за зміну кольору фракталу. На рис. 3.13. наведений приклад такої зміни.

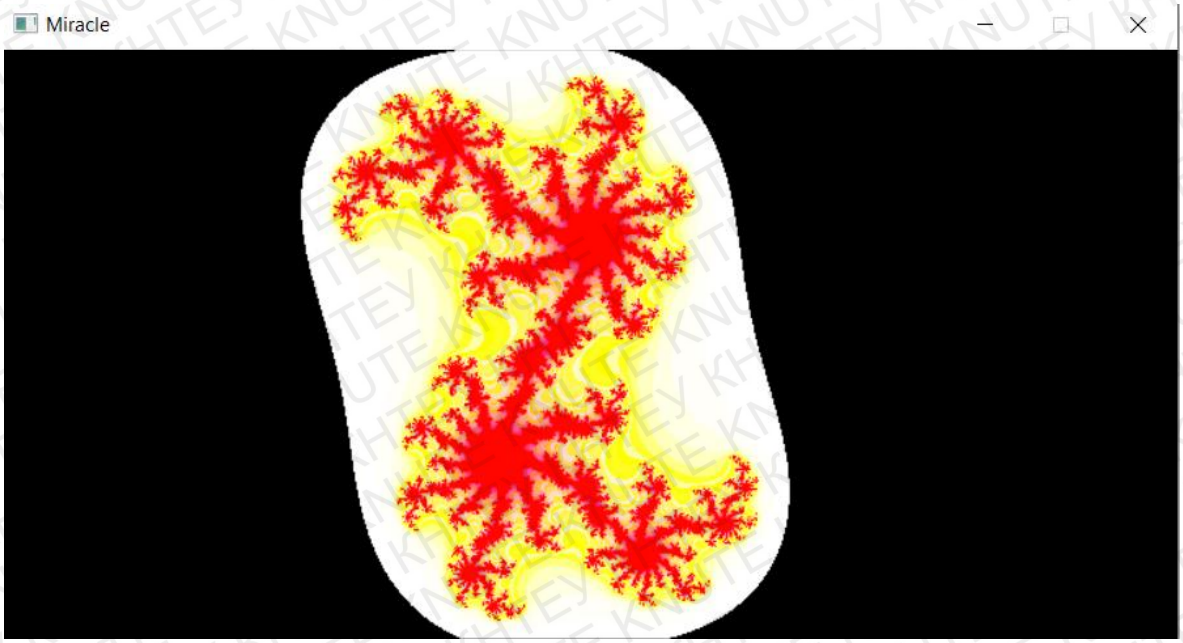

Рис. 3.13. Зміна кольору при натисненні клавіші «7»

Можливість наближення та віддалення фракталу відбувається за рахунок обчислення формул, де при руху колесом миші верх або них відбувається зміна центральної точки розташування фракталу на розмір значення змінних movex та movey, а також відбувається зміна параметру zoom. У вигляді коду це наведено на рис. 3.14, а можливості на рис. 3.15.

```
if (f->event.wheel.y > 0)
{
    f->draw.movex = (f->z.x / f->draw.zoom + f->draw.movex) - (f->z.x / (f->draw.zoom * 0.9));
    f->draw.movey = (f->z.y / f->draw.zoom + f->draw.movey) - (f->z.y / (f->draw.zoom * 0.9));
    f->draw.zoom *= 0.9;
}
if (f->event.wheel.y < 0)
{
    f->draw.movex = (f->z.x / f->draw.zoom + f->draw.movex) - (f->z.x / (f->draw.zoom * 1.1));
    f->draw.movey = (f->z.y / f->draw.zoom + f->draw.movey) - (f->z.y / (f->draw.zoom * 1.1));
    f->draw.zoom *= 1.1;
}
```

Рис. 3.14. Формули для наближення та віддалення

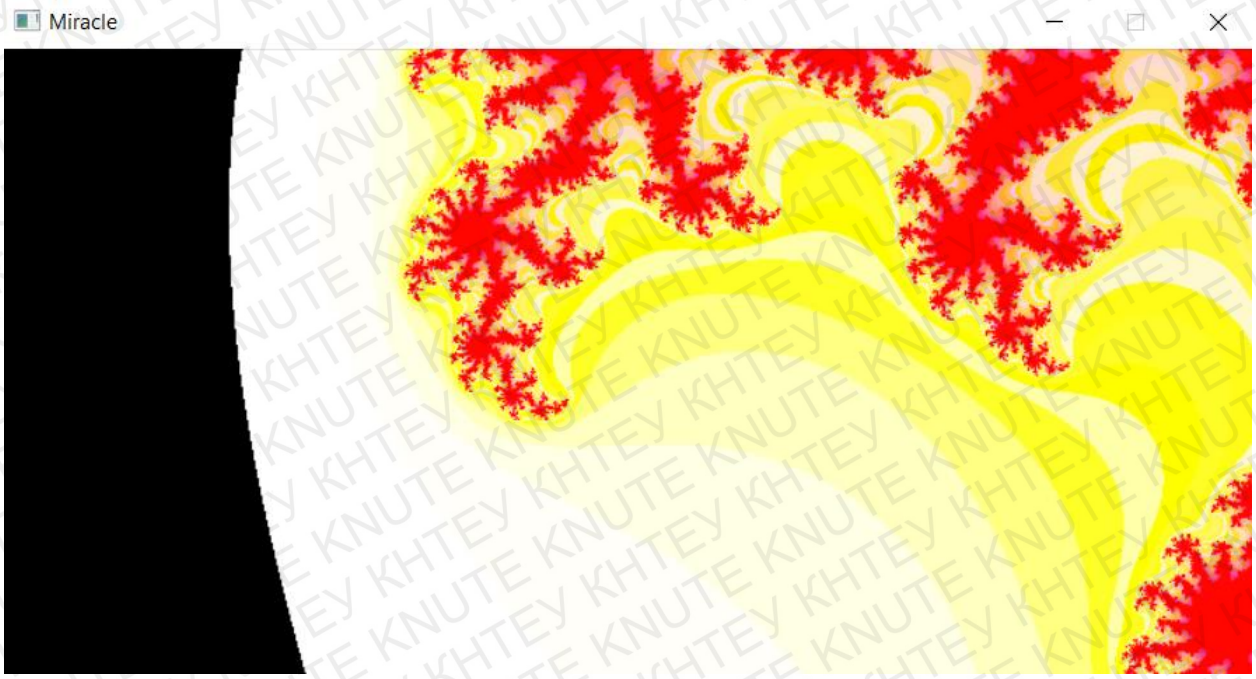

Рис. 3.15. Можливості наближення

При натисненні клавіші + або – відбувається відповідна зміна ітерацій, що у свою чергу змінює вигляд фракталу, так як глибина обрахунків стає більшою, завдяки чому при однаковому параметрі zoom можна отримати абсолютно різні зображення. Також у зв’язку із тим, що автор зробив часткову прив’язку кольору до кількості ітерацій зображення набуває неймовірних відтівків. На рис. 3.16. наведено зміну фракталу від кількості ітерацій.

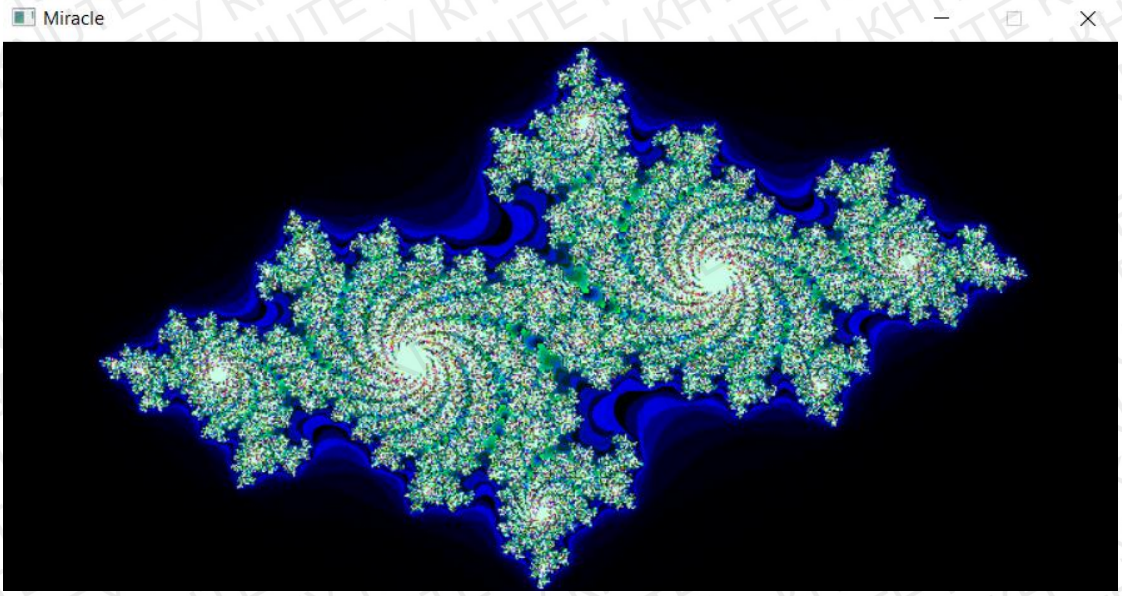

Рис. 3.16. Julia із кількістю ітерацій 300+

Проте на колір фракталу Barnsley не впливає кількість ітерацій, також у зв’язку із його особливостями він має в сотні раз більше ітерацій ніж інші, і його не можна наближувати або віддаляти. На рис. 3.17. наведено вигляд даного фракталу при кількості ітерацій більше ніж 100000.

Рис. 3.17.Barnsley 100000 ітерацій

Повний код програми знаходиться у додатку А.

3.5. Висновки до розділу 3

У третьому розділі було розроблено програму візуалізації фракталів. Вибраною мовою програмування та графічною бібліотекою для розробки програми були мова С та бібліотека SDL2.0. Середовищем у якому проходила розробка виступила Visual Studio 2019. При написанні програми дані засоби допомогли створити програму без зайвих труднощів. Також, щоб візуалізувати

фрактали була проведена чимала кількість математичних розрахунків, що дозволило глибше зануритись у тематиму фракталів. Кінцевою дією став опис функціональних можливостей програми котрий можна використовувати як гайд для корисувача.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У ході написання випускної кваліфікаційної роботи, було досліджено основні теоретичні відомості про фрактали. Також було виявлено, що фрактальна геометрія є потужним інструментом для розкриття таємниць із найрізноманітніших систем та вирішення важливих проблем у прикладній науці. Список відомих фрактальних систем є великим і швидко продовжує зростати.

Фрактали покращили нашу точність в описі та класифікації "випадкових" або органічних об'єктів, але, можливо, вони не є ідеальними. Можливо, вони просто є ближчими до нашого природного світу, ніж інші речі. Напевно саме це допомогло їм зайняти свою нішу у багатьох сучасних сферах.

Також було виявлено, що фрактали вносять чи не найбільший вклад, саме у розвиток комп'ютерної графіки, завдяки чому вони змогли набули широкого застосування у сфері GameDev. Де їх, зазвичай, використовують для створення текстур, фонових зображень, анімації, моделей природнього оточення (дерева, природні ландшафти, хмари) і т.д. Також відбуваються спроби створити ігровий двигун на основі фракталів, що може в майбутньому теоретично зменшити розмір ігрових програм та створювати відкритий світ із неодноманітним оточенням.

Було досліджено характеристику сфери GameDev в світі та Україні. З чого було зроблено висновки, що дана сфера з'явилась зовсім нещодавно, проте з кожним роком поступово збільшує свій, як культурний так і економічний вплив на життя пересічної людини. Адже навряд чи знайдеться хоч одна людина котра жодного разу не грала у відеоігри, або не чула про них. Саме тому автором було вирішено створити програму з візуалізаціїї фракталів для сфери GameDev.

В роботі було створено технічне завдання з вимогами, виконана постановка задачі та створена перша версія програми. Дана програма створена для

операційної системи Windows. Мовою програмування була вибрана C, також у роботі додатково використовувались Visual Studio 2019, графічна бібліотека SDL2.0. Вибір даної мови та бібліотеки дозволяє запросто створити кросплатформенну версію програми.

У даній версії програми реалізовано візуалізацію декількох найвідоміших фракталів таких як: Mandelbrot, Julia, Barnsley Ferm, Burning Ship. У програмі реалізова можливість наближати або віддаляти фрактал, змінювати його колір, рухати його, змінювати кількість ітерацій, повернення до початкової форми, а також для спеціально фракталу Julia було реалізовано можливість зміни в залежності від координат курсору, що дозволяє отримувати величезну кількість можливих форм цього фракталу.

Пропозицією для розвитку програми на майбутнє є збільшення кількості фракталів, можливість збереження зображення без додаткових засобів, можливість запису відео наближення фракталу до певної глибини, візуалізація 3D фракталів, а також можливість генерації тривимірної карти з використанням фрактальних алгоритмів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основний

1. Льюїс Р. Фрактали у вашому майбутньому. Глава 1. Онтаріо 2000 р.
2. Мандельброт, Б. Б. Фрактальна геометрія природи. Сан-Франциско 1982р.
3. Тернер, MJ Моделювання природи з фракталами. Лестер 2000 р.
4. Потапов А. А. Фрактали в радіофізики та радіолокації: Топологія вибірки. Вид. 2-е, 2005.– 848 с.
5. Бондаренко В.А., Дольник В. Л. Фрактальное стиснення зображень по Барнслі-Слоану // Автоматика і телемеханіка. 1994. №5. – С. 12-20.
6. Федер Е. Фрактали. Пер. з англ. – М .: Мир. 1991. – 254 с.
7. Ferraro P., Godin C., Prusinkiewicz P. Toward a quantification of self-similarity in plants // Fractals. Vol. 13. No.2. 2005. – P. 91-109.
8. Peitgen H.-O., Richter P.H. The Beauty of Fractals. – Berlin: Springer. 1986.
9. Prusinkiewicz P., Lindenmayer A. The Algorithmic Beauty of Plants. – New York: Springer-Verlag. 1996. – 230 p.
10. Prusinkiewicz P. Selfsimilarity in plants: integrating mathematical and biological perspectives. In Novak M.M., editor, Thinking in Patterns: Fractals and Related Phenomena in Nature. – Singapore: World Scientific. 2004. – P. 103-118.
11. Russ J.C. Fractal Surface. – New York and London: Plenum Press. 1994.
12. Wegner T., Peterson M., Tyler B., Branderhorst P. Fractals for Windows. – Weit Group Press. 1992.
13. Proceedings of the 5th International Workshop on Functional-Structural Plant Models. Abstracts of Papers and Posters. / Przemyslaw Prusinkiewicz, Jim Hanan, and Brendan Lane. – Napier, New Zealand. 2007. – 333 p.

14. Єгорова Є. В., Бузилев Ф. Н., Нефедов В. І. Алгоритм обробки зображень. INTERMATIC-2009, ч. 4. Матеріали Міжнародної науково-технічної конференції «Фундаментальні проблеми радіоелектронного приладобудування», 7-11 грудня 2009 р. с. 138-139.Pentland A. P. Fractal-Based Description of Natural Scenes // IEEE Trans. 1984. V. PAMI-6, 6. P. 661-674.

15. Потапов А. А. Фрактали в дистанційному зондування /Закордонна радіоелектроніка. Успіхи сучасної радіоелектроніки. 2000. № 6. С. 3-65.

16. Arduini F., Dambra C., Dellepiane S. et al. Fractal Dimension by Adaptive Mask Selection//Proc. IEEE Intern. Conf. on Acoustics, Speech and Signal Processing. – N. Y.: 1988. P. 1116 – 1119.

17. Xion M., Zhuang Z., Xiao S., Guo G. The Analyse and Recognition of Radar Targets Scattering Signal with Chaos Multifractal Theory//J. National University of Defense Technology (China). 1998. V. 20, 2. P. 60–64.

18. Реконструкція зображень / Под ред. Г. Старка: Пер. з англ .; Під ред. Б. С. Кругликова. – М .: Світ, 1992. С.636.

19. Фор А. Сприйняття і розпізнавання образів: Пер. з франц .; Під ред. Г. П. Катиса. – М .: Машинобудування, 1989. с.272.

20. James Gleick, Chaos: Making a New Science, Viking, New York, 1987.

21. Kenneth Falconer, 2003, Fractal Geometry: Mathematical Foundations and Applications. Chichester, UK: John Wiley & Sons.

22. R. Hohlfeld, and N. Cohen. 1999. Self-similarity and the geometric requirements for frequency independence in antennae.

23. Michael Frame, and Benoît B. Mandelbrot. A Panorama of Fractals and Their Uses. Yale. Retrieved January 14, 2009.

24. Peng, Gongwen, Decheng Tian. 1990. The fractal nature of a fracture surface. Journal of Physics A. Retrieved January 14, 2009.

25. Мандельброт Б. Б. Фракталы и хаос. Множество Мандельброта и другие чудеса. – М., НИЦ "Регулярная и хаотическая динамика", 2009. – 392 с.

26. Річард М. Кроновер Фрактали і хаос в динамічних системах. – М., Постмаркет, 2000. – 352 с.

27. Шредер М. Фрактали, хаос, статечні закони. Мініатюри з нескінченного раю. 2001. – 528 с.

28. М.Газале. Гномон. Від фараонів до фракталів. 2002. - 272 с.

29. Х.-О. Пайтген, П. Х. Ритхер Красота фракталов. - М. Мир, 1993. – 176 с.

30. Божокин СВ., Паршин В.А. Фракталы и мультифракталы. «Регулярная и хаотическая динамика», 2001. – 128 с.

31. Е.Федер Фракталы. – М., Мир, 1991. – 261 с.

32. Е. Петерс Фрактальний аналіз фінансових ринків: Застосування теорії Хаосу в інвестиціях і економіці. 2004 - 304 с.

33. В.В. Ісаєва, Ю.А. Каретін, А.В. Чернишов, Д.Ю. Шкуратов Фрактали і хаос в біологічному морфогенезі. 2004. - 128 с.

34. Джейсон Шреєр. «Кров, піт та пікселі. Зворотня сторона індустрії відеоігор», 2019.

35. Андрій Подшибякін. «Час ігор! Вітчизняна ігрова індустрія в обличях та мріях», 2020.

Інтернет ресурси

36. Веб-сторінка керівництво для програмування мовою С режим доступу: https://metanit.com/cpp/c/

37. Веб-сторінка керівництво по SDL2.0 режим доступу: https://www.libsdl.org/index.php

38. Веб-сторінка керівництво Visual Studio 2019 режим доступу: https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2019

ДОДАТКИ

Додаток А

Код файлу fractal.h

```
#ifndef FRACTAL_H
# define FRACTAL_H
# define H 360
# define W 720

# define WIDTH 600
# define HEIGHT 800
# define PI 3.1415926535
# define POINTS 2000

# include <iostream>
# include <SDL.h>
# include "stdlib.h"
# include "math.h"
# include "stdio.h"
# include "time.h"
# include "string.h"

using namespace std;

typedef	struct s_draw_map
{
	int		maxiterations;
	double		cre;
	double		cim;
	double		zoom;
	double		movex;
	double		movey;
	int		y;
	int		x;
	int		stop;
	double		newre;
	double		newim;
	double		oldre;
	double		oldim;
}			t_draw_map;
```

```
typedef struct		s_zoom
{
	int			x;
	int			y;
}				t_zoom;

typedef struct		s_barnsley
{
	int			r;
	int			g;
	int			b;
	int			multw;
	int			multh;
	SDL_Window*		screen;
	SDL_Renderer*	renderer;
	SDL_Surface*	surface;
	SDL_Surface*	text;
	SDL_Texture*	texture;
	int			lastMouse_x;
	int			lastMouse_y;
}				t_barnsley;

typedef struct		s_f
{
	t_barnsley		barn;
	t_draw_map		draw;
	SDL_Surface		*surface;
	SDL_Window		*window;
	Uint8			*mass;
	SDL_Event		event;
	SDL_bool		open;
	t_zoom			z;
	SDL_Renderer	*renderer;
	int			thread_id;
	int			y;
	int			x;
	int			fractal;
	double			real;
	double			imag;
	int			up_down;
	int			lef_rig;
	double			mirror;
	int			color;
	int			i;
```

```
        int                     barns;
}                               t_f;

void                    start(t_f* f);
void                    init_img(t_f* f);
void                    main_sdl_work(t_f* f);
void                    draw(t_f* f);
void                    draw_fr(t_f* f);
void                    mouse_move(t_f* f);
void                    wtrite_key(t_f* f);
void                    put_pixel_in_img(t_f* f, int color);
void                    color(t_f* f);
void                    barnsley_start(t_f* f);

void                    setPixel(t_f* f, SDL_Surface* surface, int x, int y, unsigned int
r, unsigned int g, unsigned int b);
void                    renderFractal(t_f* f);
void                    drawLeaf(t_f* f);
void                    inputs(t_f* f);
void                    render(t_f* f);
void                    init(t_f* f);
void                    step(t_f* f);
void                    barns_color(t_f* f);
void                    menu1(void);

void                    mandelbrot(t_f* f);
void                    julia(t_f* f);
void                    ship(t_f* f);
void                    barnsley(t_f* f);
void                    dich(t_f* f);
void                    ship2(t_f* f);
void                    mandelbrot2(t_f* f);
void                    mandelbrotsin(t_f* f);

#endif
```

Код файлу main.c

```
#include "fractal.h"

void   start(t_f* f)
{
        f->draw.maxiterations = 41;
        f->draw.zoom = 200;
        f->draw.movex = -1.8;
```

```
	f->draw.movey = -0.9;
	f->draw.cim = 0.27015;
	f->draw.cre = -0.7;
	f->color = 1;
	f->draw.stop = 0;
	f->barns = 0;
}

void	draw_errror(void)
{
	cout<<"Enter number fractal from the list\n";
	cout<<"1 Julia\n";
	cout<<"2 Mandelbrot\n";
	cout<<"3 Mandelbrot2\n";
	cout<<"4 MandelbrotSin\n";
	cout<<"5 Ship\n";
	cout<<"6 Ship2\n";
	cout<<"7 Dich\n";
	cout<<"8 Barnsley\n";
}

void	wht_draw(t_f* f, char* arg)
{
	if (strcmp(arg, "1") == 0)
		f->fractal = 1;
	if (strcmp(arg, "2") == 0)
		f->fractal = 2;
	if (strcmp(arg, "3") == 0)
		f->fractal = 3;
	if (strcmp(arg, "4") == 0)
		f->fractal = 4;
	if (strcmp(arg, "5") == 0)
		f->fractal = 5;
	if (strcmp(arg, "6") == 0)
		f->fractal = 6;
	if (strcmp(arg, "7") == 0)
		f->fractal = 7;
	if (strcmp(arg, "8") == 0)
		f->fractal = 8;
}

void test(t_f* f)
{
	cout << f->fractal;
```

```
}

int		main(int argc, char* argv[])
{
	t_f* f;

	f = (t_f*)malloc(sizeof(t_f));
	if (argc == 2)
	{
		wht_draw(f, argv[1]);
		f->fractal == 0 ? draw_errror() : draw(f);
	}
	else
		draw_errror();
	return (0);
}
```

Код файлу Draw.c

```
#include "fractal.h"

void	draw(t_f* f)
{
	menu1();
	if (f->fractal == 8)
	{
		f->barns = 1;
		barnsley(f);
	}
	else
	{
		init_img(f);
		start(f);
		draw_fr(f);
		main_sdl_work(f);
	}
}

void	main_sdl_work(t_f* f)
{
	while (f->open)
	{
		SDL_UpdateWindowSurface(f->window);
		if (SDL_PollEvent(&f->event))
		{
```

```
                SDL_FillRect(f->surface, NULL, 0x000000);
                wtrite_key(f);
                color(f);
                mouse_move(f);
                draw_fr(f);
            }
        }
        SDL_DestroyWindow(f->window);
        SDL_Quit();
}

void    check_frac(t_f* f)
{
        if (f->fractal == 1)
            julia(f);
        if (f->fractal == 2)
            mandelbrot(f);
        if (f->fractal == 3)
            mandelbrot2(f);
        if (f->fractal == 4)
            mandelbrotsin(f);
        if (f->fractal == 5)
            ship(f);
        if (f->fractal == 6)
            ship2(f);
        if (f->fractal == 7)
            dich(f);
}

void    draw_fr(t_f* f)
{
        f->y = 0;
        while (f->y < H)
        {
            f->x = 0;
            while (f->x < W)
            {
                check_frac(f);
                f->x++;
            }
            f->y++;
        }
}
```

```
void	init_img(t_f* f)
{
	SDL_Init(SDL_INIT_EVERYTHING);
	f->surface = NULL;
	f->window = SDL_CreateWindow("Miracle",
SDL_WINDOWPOS_UNDEFINED,
		SDL_WINDOWPOS_UNDEFINED, W, H, SDL_WINDOW_SHOWN);
	SDL_Renderer* render = SDL_CreateRenderer(f->window, -1,
SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
	f->surface = SDL_GetWindowSurface(f->window);
	f->open = SDL_TRUE;
}

void	put_pixel_in_img(t_f *f, int color)
{
	f->mass = (Uint8*)f->surface->pixels + f->y * f->surface->pitch + f->x * 4;
	*(Uint32*)f->mass = color;
}
```

Код файлу julia.c

```
#include "fractal.h"

void	julia(t_f* f)
{
	f->draw.newre = f->x * 1.0 / f->draw.zoom + f->draw.movex;
	f->draw.newim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
	f->i = -1;
	while (++f->i < f->draw.maxiterations)
	{
		f->draw.oldre = f->draw.newre;
		f->draw.oldim = f->draw.newim;
		f->draw.newre = f->draw.oldre * f->draw.oldre - f->draw.oldim * f-
>draw.oldim + f->draw.cre;
		f->draw.newim = 2 * f->draw.oldre * f->draw.oldim + f->draw.cim;
		if ((f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim) >
4)
			break;
		(f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
	}
}
```

Код файлу keys.c

```
#include "fractal.h"
```

```
void	wtrite_key(t_f* f)
{
	if (SDL_QUIT == f->event.type)
		f->open = SDL_FALSE;
	if (f->event.key.keysym.sym == SDLK_ESCAPE)
		f->open = SDL_FALSE;
	if (f->event.key.keysym.sym == SDLK_r && f->draw.stop == 0)
		f->draw.stop = 1;
	if (f->event.key.keysym.sym == SDLK_p && f->draw.stop == 1)
		f->draw.stop = 0;
	switch (f->event.key.keysym.sym)
	{
	case SDLK_UP:
		f->draw.movey += ((double)H / (100 * f->draw.zoom));
		break;
	case SDLK_DOWN:
		f->draw.movey -= ((double)H / (100 * f->draw.zoom));
		break;
	case SDLK_LEFT:
		f->draw.movex += ((double)H / (100 * f->draw.zoom));
		break;
	case SDLK_RIGHT:
		f->draw.movex -= ((double)H / (100 * f->draw.zoom));
		break;
	case SDLK_s:
		start(f);
		break;
	case SDLK_KP_PLUS:
		f->draw.maxiterations += 1;
		break;
	case SDLK_KP_MINUS:
		f->draw.maxiterations -= 1;
		break;
	}
}

void	color(t_f * f)
{
	switch (f->event.key.keysym.sym)
	{
		case SDLK_1:
			f->color = 0xff0000;
			break;
```

```
                case SDLK_2:
                        f->color = 0x00ff00;
                        break;
                case SDLK_3:
                        f->color = 0x0000ff;
                        break;
                case SDLK_4:
                        f->color = 0xC0C0C0;
                        break;
                case SDLK_5:
                        f->color = 0xFF8C00;
                        break;
                case SDLK_6:
                        f->color = 0xD2691E;
                        break;
                case SDLK_7:
                        f->color = 0xFFFFFF;
                        break;
                case SDLK_8:
                        f->color = 0x800080;
                        break;
                case SDLK_9:
                        f->color = 0x00FFFF;
                        break;
        }
}
```

Код файлу mandelbrot.c

```
#include "fractal.h"

void   mandelbrot(t_f* f)
{
        f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex - 0.5;
        f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
        f->draw.newim = 0;
        f->draw.newre = 0;
        f->draw.oldre = 0;
        f->draw.oldim = 0;
        f->i = -1;
        while (++f->i < f->draw.maxiterations)
        {
                f->draw.oldre = f->draw.newre;
                f->draw.oldim = f->draw.newim;
                f->draw.newre = f->draw.oldre * f->draw.oldre - f->draw.oldim *
```

```
                f->draw.oldim + f->draw.cre;
            f->draw.newim = 2 * f->draw.oldre * f->draw.oldim + f->draw.cim;
            if ((f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim) >
4)
                break;
        }
        (f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
```

Код файлу menu.c

```
#include "fractal.h"

void menu1(void)
{
        cout << "\n";
        cout << "For everyone\n";
        cout << "Use mouse wheel for zooming\n";
        cout << "Top 1-9 for change color\n";
        cout << "S for start position\n";
        cout << "Right + for add iterations\n";
        cout << "Right - for minus iterations\n";
        cout << "Arrows for moving\n";
        cout << "\n";
        cout << "Only for Julia\n";
        cout << "R for start changing\n";
        cout << "P for stop changing\n";
}
```

Код файлу mouse.c

```
#include "fractal.h"

void   zoom(t_f* f)
{
        if (f->event.type == SDL_MOUSEWHEEL)
        {
            if (f->event.wheel.y > 0)
            {
                f->draw.movex = (f->z.x / f->draw.zoom + f->draw.movex) -
                    (f->z.x / (f->draw.zoom * 0.9));
                f->draw.movey = (f->z.y / f->draw.zoom + f->draw.movey) -
                    (f->z.y / (f->draw.zoom * 0.9));
                f->draw.zoom *= 0.9;
            }
```

```
		if (f->event.wheel.y < 0)
		{
			f->draw.movex = (f->z.x / f->draw.zoom + f->draw.movex) -
				(f->z.x / (f->draw.zoom * 1.1));
			f->draw.movey = (f->z.y / f->draw.zoom + f->draw.movey) -
				(f->z.y / (f->draw.zoom * 1.1));
			f->draw.zoom *= 1.1;
		}
	}
}

void	mouse_move(t_f* f)
{
	if (f->event.type == SDL_MOUSEMOTION)
	{
		if (f->event.button.x >= 0 && f->event.button.x <= W && f-
>event.button.y >= 0 && f->event.button.y <= H)
		{
			f->z.x = f->event.button.x;
			f->z.y = f->event.button.y;
			if (f->draw.stop == 1)
			{
				f->draw.cim = 0.008 * (double)(H / 2 - f->event.button.y);
				f->draw.cre = 0.004 * (double)(W / 2 - f->event.button.x);
			}
		}
	}
	zoom(f);
}
```

Код файлу other_fractals.c

```
#include "fractal.h"

void	ship(t_f* f)
{
	f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex - 0.5;
	f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey - 0.5;
	f->draw.newre = 0;
	f->draw.newim = 0;
	f->i = -1;
	while (++f->i < f->draw.maxiterations)
	{
		f->draw.oldre = f->draw.newre;
		f->draw.oldim = f->draw.newim;
```

```
            f->draw.newre = f->draw.oldre * f->draw.oldre -
                    f->draw.oldim * f->draw.oldim + f->draw.cre;
            f->draw.newim = 2 * fabs(f->draw.oldre * f->draw.oldim) + f->draw.cim;
            if (f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim >
4)
                    break;
        }
        (f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
}

void   dich(t_f* f)
{
        f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex - 0.5;
        f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
        f->draw.newim = 0;
        f->draw.newre = 0;
        f->draw.oldre = 0;
        f->draw.oldim = 0;
        f->i = -1;
        while (++f->i < f->draw.maxiterations)
        {
            f->draw.oldre = f->draw.newre;
            f->draw.oldim = f->draw.newim;
            f->draw.newre = f->draw.oldre * f->draw.oldre * f->draw.oldre *
                    f->draw.oldre + f->draw.oldim * f->draw.oldim * f->draw.oldim *
                    f->draw.oldim - 2 * f->draw.oldim * f->draw.oldim * f->draw.oldre
*
                    f->draw.oldre + f->draw.cre;
            f->draw.newim = 4 * f->draw.oldre * f->draw.oldim * (f->draw.oldim *
                    f->draw.oldim + f->draw.oldre * f->draw.oldre) + f->draw.cim;
            if ((f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim) >
8)
                    break;
        }
        (f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
}

void   ship2(t_f* f)
{
        f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex;
        f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
        f->draw.newre = 0;
```

```
	f->draw.newim = 0;
	f->i = -1;
	while (++f->i < f->draw.maxiterations)
	{
		f->draw.oldre = f->draw.newre;
		f->draw.oldim = f->draw.newim;
		f->draw.newre = f->draw.oldre * f->draw.oldre - f->draw.oldim *
			f->draw.oldim + f->draw.cre;
		f->draw.newim = -2 * f->draw.oldre * f->draw.oldim + f->draw.cim;
		if (f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim >
4)
			break;
	}
	(f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
}

void	mandelbrot2(t_f* f)
{
	f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex;
	f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
	f->draw.newim = 0;
	f->draw.newre = 0;
	f->draw.oldre = 0;
	f->draw.oldim = 0;
	f->i = -1;
	while (++f->i < f->draw.maxiterations)
	{
		f->draw.oldre = f->draw.newre;
		f->draw.oldim = f->draw.newim;
		f->draw.newre = f->draw.oldre * f->draw.oldre * f->draw.oldre - 3 *
			f->draw.oldre * f->draw.oldim * f->draw.oldim + f->draw.cre;
		f->draw.newim = 3 * f->draw.oldre * f->draw.oldre * f->draw.oldim -
			f->draw.oldim * f->draw.oldim * f->draw.oldim + f->draw.cim;
		if ((f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim) >
4)
			break;
	}
	(f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
}

void	mandelbrotsin(t_f* f)
{
```

```
	f->draw.cre = f->x * 1.0 / f->draw.zoom + f->draw.movex;
	f->draw.cim = f->y * 1.0 / f->draw.zoom + f->draw.movey;
	f->draw.newim = 0;
	f->draw.newre = 0;
	f->draw.oldre = 0;
	f->draw.oldim = 0;
	f->i = -1;
	while (++f->i < f->draw.maxiterations)
	{
		f->draw.oldre = f->draw.newre;
		f->draw.oldim = f->draw.newim;
		f->draw.newre = sin(f->draw.oldre * f->draw.oldre -
			f->draw.oldim * f->draw.oldim + f->draw.cre);
		f->draw.newim = 2 * f->draw.oldre * f->draw.oldim + f->draw.cim;
		if ((f->draw.newre * f->draw.newre + f->draw.newim * f->draw.newim) >
4)
			break;
	}
	(f->i == f->draw.maxiterations) ? put_pixel_in_img(f, f->i * f->i * f->i * f-
>color) : put_pixel_in_img(f, f->i * f->i * f->i * f->color);
}
```

Код файлу barnsl.c

```
#include "fractal.h"

void setPixel(t_f* f, SDL_Surface* surface, int x, int y, unsigned int r, unsigned int g,
unsigned int b)
{
	int win_w, win_h;
	Uint32 color;
	Uint8* p;

	SDL_GetWindowSize(f->barn.screen, &win_w, &win_h);

	if (x > win_w || y > win_h)
		return;
	color = SDL_MapRGB(surface->format, r, g, b);

	p = (Uint8*)surface->pixels + y * surface->pitch + x * surface->format-
>BytesPerPixel;
	*(Uint32*)p = color;
}

int mid_x(int x1, int x2)
```

```
{
	return (x1 + x2) / 2;
}

int mid_y(int y1, int y2)
{
	return (y1 + y2) / 2;
}

void drawLeaf(t_f *f)
{
	int win_w, win_h;
	int mx, my;
	SDL_GetMouseState(&mx, &my);
	SDL_GetWindowSize(f->barn.screen, &win_w, &win_h);
	renderFractal(f);
}

void render(t_f *f)
{
	SDL_RenderClear(f->barn.renderer);
	drawLeaf(f);
	SDL_RenderPresent(f->barn.renderer);
}

void init(t_f* f)
{
	f->barn.surface = SDL_CreateRGBSurface(0, WIDTH, HEIGHT, 32,
		0x00FF0000,
		0x0000FF00,
		0x000000FF,
		0xFF000000);
	if (!f->barn.surface)
	{
		fprintf(stderr, "Could not setup surface %s", SDL_GetError());
		exit(1);
	}
	f->barn.texture = SDL_CreateTextureFromSurface(f->barn.renderer, f-
>barn.surface);
	if (!f->barn.texture)
	{
		fprintf(stderr, "Could not setup texture %s", SDL_GetError());
		exit(1);
	}
```

```
}

void step(t_f* f)
{
	render(f);
	inputs(f);
}

void barnsley(t_f *f)
{
	SDL_Init(SDL_INIT_VIDEO);
	SDL_CreateWindowAndRenderer(WIDTH, HEIGHT,
SDL_WINDOW_RESIZABLE, &f->barn.screen, &f->barn.renderer);
	init(f);
	SDL_SetRenderDrawColor(f->barn.renderer, 0x00, 0x00, 0x00, 1);
	SDL_RenderClear(f->barn.renderer);
	barnsley_start(f);
	while (1)
	{
		step(f);
		SDL_Delay(16);
	}
	SDL_Quit();
}
```

Код файлу barnsl1.c

```
#include "fractal.h"

void	barnsley_start(t_f* f)
{
	f->barn.r = 0xff;
	f->barn.g = 0xff;
	f->barn.b = 0xff;
	f->barn.multh = 80;
	f->barn.multw = 100;
}

double drand1(void)
{
	return (double)rand() / (double)RAND_MAX;
}

void renderFractal(t_f* f)
{
```

```
int mx, my;
int win_w, win_h;
mx = 0;
my = 0;
win_h = 0;
win_w = 0;
int mx_c, my_c;

double threshold[4];
double p[2];
double dx, dy;
double random;
int x, y;

int i;

p[0] = drand1();
p[1] = drand1();

threshold[0] = 0.83;
threshold[1] = 0.91;
threshold[2] = 0.98;
threshold[3] = 1.00;

mx_c = mx - win_w / 2;
my_c = my - win_h / 2;

SDL_GetWindowSize(f->barn.screen, &win_w, &win_h);
SDL_GetMouseState(&mx, &my);

mx_c /= 8;
my_c /= 8;

x = win_w / 2;
y = win_h / 2;

for (i = 0; i < POINTS; i++)
{
        random = drand1();

        if (random < threshold[0])
        {
                dx = (0.85 * p[0]) + (0.04 * p[1]) + 0.0;
                dy = (-0.04 * p[0]) + (0.85 * p[1]) + 1.6;
```

```
                }
                else if (random < threshold[1])
                {
                        dx = (0.20 * p[0]) - (0.26 * p[1]) + 0.0;
                        dy = (0.23 * p[0]) + (0.22 * p[1]) + 1.6;
                }
                else if (random < threshold[2])
                {
                        dx = (-0.15 * p[0]) + (0.28 * p[1]) + 0.0;
                        dy = (0.26 * p[0]) + (0.24 * p[1]) + 0.44;
                }
                else
                {
                        dx = (0.00 * p[0]) + (0.00 * p[1]) + 0.0;
                        dy = (0.00 * p[0]) + (0.16 * p[1]) + 0.0;
                }
                p[0] = dx;
                p[1] = dy;
                x = dx * f->barn.multw + win_w / 2;
                y = dy * f->barn.multh;
                setPixel(f, f->barn.surface, x, 800 - y, f->barn.r, f->barn.g, f->barn.b);
        }
        f->barn.lastMouse_x = mx_c;
        f->barn.lastMouse_y = my_c;
        SDL_UpdateTexture(f->barn.texture, NULL, f->barn.surface->pixels, f-
>barn.surface->pitch);
        SDL_RenderCopy(f->barn.renderer, f->barn.texture, NULL, NULL);
}

void keys_b(t_f* f)
{
        switch (f->event.key.keysym.sym)
        {
        case SDLK_ESCAPE:
                exit(0);
                break;
        case SDLK_UP:
                f->barn.multh += 2;
                break;
        case SDLK_DOWN:
                f->barn.multh -= 2;
                break;
        case SDLK_LEFT:
                f->barn.multw += 2;
```

```
                break;
        case SDLK_RIGHT:
                f->barn.multw -= 2;
                break;
        case SDLK_s:
                barnsley_start(f);
                break;
        }
}

void  barns_color(t_f* f)
{
        switch (f->event.key.keysym.sym)
        {
        case SDLK_1:
                f->barn.r = 0xff;
                f->barn.g = 0x00;
                f->barn.b = 0x00;
                break;
        case SDLK_2:
                f->barn.r = 0x00;
                f->barn.g = 0xff;
                f->barn.b = 0x00;
                break;
        case SDLK_3:
                f->barn.r = 0x00;
                f->barn.g = 0x00;
                f->barn.b = 0xff;
                break;
        case SDLK_4:
                f->barn.r = 0xc0;
                f->barn.g = 0xc0;
                f->barn.b = 0xc0;
                break;
        case SDLK_5:
                f->barn.r = 0xff;
                f->barn.g = 0x8c;
                f->barn.b = 0x00;
                break;
        case SDLK_6:
                f->barn.r = 0xd2;
                f->barn.g = 0x69;
                f->barn.b = 0x1e;
                break;
```

```
		case SDLK_7:
			f->barn.r = 0xff;
			f->barn.g = 0xff;
			f->barn.b = 0xff;
			break;
		case SDLK_8:
			f->barn.r = 0x80;
			f->barn.g = 0x00;
			f->barn.b = 0x80;
			break;
		case SDLK_9:
			f->barn.r = 0x00;
			f->barn.g = 0xff;
			f->barn.b = 0xff;
			break;
		}
}

void inputs(t_f* f)
{
		while (SDL_PollEvent(&f->event))
		{
			switch (f->event.type)
			{
			case SDL_QUIT:
				exit(0);
				break;
			case SDL_KEYDOWN:
				keys_b(f);
				barns_color(f);
				break;
			default:
				break;
			}
		}
}
```