Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

ФРАКТАЛЬНІ МНОЖИНИ: ЕКСПЕРИМЕНТИ З КОМПЛЕКСНИМИ ЧИСЛАМИ, КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ

Текстова частина до курсової роботи
за спеціальністю 113 „Прикладна математика"

Керівник курсової роботи
к.ф.-м.н., доц. Щестюк Н.Ю.
(прізвище та ініціали)
_____
(підпис)
"____" _____ 2020 р.

Виконав студент
Миколайчик Я.А.
(прізвище та ініціали)
"17" квітня 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри математики,
проф., д.ф.-м.н.
Олійник Б. В.
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Миколайчику Я.А. факультету інформатики 4 курсу
ТЕМА: Фрактальні множини: експерименти з комплексними числами, комп’ютерене моделювання
Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1 Основні означення
2 Фрактали у комплексній динаміці
3 Побудова множин Мандельброта і Жюліа. Експерименти
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2020 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Фрактальні множини: експерименти з комплексними числами, комп’ютерене моделювання

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	10.10.2019	
2.	Ознайомлення з поняттями фрактальних множин.	15.12.2019	
3.	Ознайомлення з поняттям множини Мандельброта.	05.01.2020	
4.	Ознайомлення з поняттям множини Жюліа.	05.01.2020	
5.	Аналіз алгорииммів побудови множин Мандельброта та Жюліа.	07.01.2020	
6.	Програмування проаналізованих алгоритмів.	03.04.2020	
7.	Застосування запрограмованого застосунку для проведення експериментів.	04.04.2020	
8.	Написання пояснювальної роботи.	05.04.2020	
9.	Створення слайдів для доповіді та написання доповіді.	10.04.2020	
10.	Аналіз отриманих результатів з керівником.	11.04.2020	
11.	Корегування роботи за результатами попереднього аналізу.	15.04.2020	
12.	Остаточне оформлення пояснювальної роботи та слайдів.	17.04.2020	
13.	Здача курсової роботи.	19.04.2020	

Студент: Миколайчик Я.А.

Керівник: Щестюк Н.Ю.

“_____” _____

Зміст

ВСТУП

Майже щодня люди стикаються з фрактальними множинами, хоч самі про це не задумуються. Теорія фрактальних множин – дуже цікавий і, мабуть, один з найгарніших розділів математики. Фрактальні множини є досить поширеними як у прикладних науках, таких, як інформатика, радіотехніка, фізика та біологія, так і у мистецтві, і навіть природі, чим є дуже особливими та захоплюючими.

Завданням цього дослідження є розглянути і дослідити суть фрактальних множин, їхні особливості, глибше дізнатись про їхню користь і області застосування, а також навчитись їх програмно моделювати з метою проведення цікавих експериментів.

Робота складається з трьох розділів.

В першому розділі буде розкрито основні базові поняття, необхідні для розкриття та повного розуміння всіх сутностей, пов’язаних з темою дослідження.

У другому розділі розкриваються поняття фракталів, їхні особливості, властивості та методи побудови на декількох прикладах.

Третій розділ присвячено дослідженню способу програмного проектування фрактальних множин, а саме множин Мандельброта і Джулія, опису практичної розробки програмного застосунку для побудови фракталів, а також проведенню експериментів над ними за допомогою цього застосунку.

РОЗДІЛ 1. ОСНОВНІ ОЗНАЧЕННЯ

1.1 Комплексне число

Комплексне число — число виду $a + bi$, де a, b — дійсні числа, i — уявна одиниця, тобто число, для якого виконується рівність: $i^2 = -1$. Множина комплексних чисел зазвичай позначається символом $\mathbb{C}$.[1]

1.2 Комплексна площина

Комплексна площина – це геометричне представлення множини комплексних чисел через модифіковану двовимірну площину дійсних чисел, де зміщення по осі x відповідає за дійсну частину комплексного числа, а зміщення по перпендикулярній до неї осі y за уявну частину.[2]

1.3 Евклідовий простір

Евклідовий простір – фундаментальний простір класичної геометрії, властивості якого описуються аксіомами евклідової геометрії.[3]

1.4 Многовид

Многовид – об’єкт, що локально має характер евклідового простору.[4]

1.5 Голоморфна функція

Голоморфна функція – функція комплексної змінної, визначена на відкритій підмножині комплексного простору $\mathbb{C}$ і комплексно диференційована в кожній точці.[5]

1.6 Голоморфна динаміка

Голоморфна динаміка – розділ математики, що вивчає властивості багатократної ітерації голоморфних функцій на одномірних комплексних многовидах (наприклад, на комплексній площині $\mathbb{C}$), а також рішення функціональних та диференціально-функціональних рівнянь з такими ітераціями.[6]

1.7 Самоподібний об’єкт

Самоподібний об’єкт – об’єкт, що в точності або наближено співпадає з частиною самого себе.[7]

1.8 Фазовий простір

Фазовий простір – простір, кожна точка якого відповідає одному і тільки одному стану з множини всіх можливих станів системи.[8]

1.9 Динамічна система

Динамічна система – множина елементів, для якої задана функціональна залежність між часом і положенням у фазовому просторі кожного елемента системи.[9]

1.10 Границя множини

Границя множини – множина всіх точок, що знаходяться як завгодно близько як до точок самої множини, так і до точок поза нею.[10]

РОЗДІЛ 2. ФРАКТАЛИ У КОМПЛЕКСНІЙ ДИНАМІЦІ

2.1 Фрактали та фрактальні об’єкти

2.1.1 Означення

[11][12] Фрактал – нерегулярна самоподібна структура. В широкому розумінні фрактал означає фігуру, малі частини якої в довільному збільшенні є подібними до неї самої. В математиці під фракталами мають на увазі множини точок в евклідовому просторі, що мають дробову метричну розмірність (Мінковського або Хаусдорфа), або метричну розмірність, що відрізняється від топологічної.

Фракталом може називатись об’єкт, що володіє принаймні однією з наступних характеристик:

- має нетривіальну структуру при будь-якому масштабуванні, на відміну від регулярних геометричних фігур;
- є самоподібним або наближено самоподібним;
- має дробову метричну розмірність або метричну розмірність, що переважає топологічну.

2.1.2 Самоподібні множини зі специфічними властивостями

2.1.2.1 Множина Кантора

[13] Множина Кантора – ніде не щільна незліченна досконала множина, тобто така, що збігається з множиною своїх граничних точок.

Рисунок 2.1 Ітерації перетворення прямої на множину Кантора

Побудова:

Множина Кантора створюється шляхом ітеративного видалення з лінійного сегменту середнього проміжку розміром у його третину. Спочатку видаляється проміжок $(\frac{1}{3}, \frac{2}{3})$ з інтервалу $[0, 1]$, що залишить два лінійних сегменти: $[0, \frac{1}{3}] \cup [\frac{2}{3}, 1]$. Потім видаляються середні третинні проміжки утворених сегментів, що утворить нам наступні чотири сегменти: $\left[0, \frac{1}{9}\right] \cup \left[\frac{2}{9}, \frac{1}{3}\right] \cup \left[\frac{2}{3}, \frac{7}{9}\right] \cup \left[\frac{8}{9}, 1\right]$. Цей процес продовжується до нескінченності, де n-на множина буде такою:

$$C_n = \frac{C_{n-1}}{3} \cup \left(\frac{2}{3} + \frac{C_{n-1}}{3}\right) \text{ для } n \geq 1, C_0 = [0,1].$$

Множина Кантора містить всі точки інтервалу $[0,1]$, які не були видалені на жодному кроці цього нескінченного процесу:

$$\mathrm{C} \coloneqq \bigcap_{n=1}^{\infty} C_n.$$

Перші шість кроків ітерації зображені на рисунку 2.1.

2.1.2.2 Трикутник Сєрпінського

[14] Трикутник Сєрпінського – аналог множини Кантора на площині:

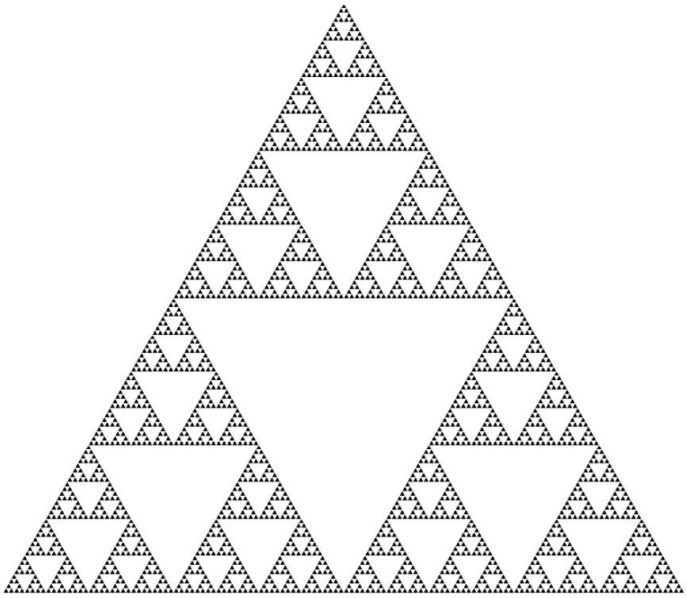

Рисунок 2.2. Трикутник Сєрпінського

Побудова ітеративним методом:

Середини сторін рівностороннього трикутника T_0 з'єднуються відрізками. Вийде чотири нових трикутника. З вихідного трикутника видаляється внутрішність серединного трикутника. Вийде множина T_1, що складається з трьох трикутників, що залишились. Зробивши те саме для кожного з цих трикутників, отримаємо множину T_2, що складається з дев'яти рівносторонніх трикутників. Продовжуючи цей процес нескінченно, отримаємо послідовність

$$T_0 \supset T_1 \supset \cdots \supset T_n \supset \cdots,$$

перетин членів якої є трикутником Сєрпінського.

Ітерації побудови зображені на рисунку 2.3.

Рисунок 2.3. Ітерації побудови трикутника Сєрпінського

2.1.2.3 Губка Менгера

[15] Губка Менгера – аналог множини Кантора в тривимірному просторі.

Побудова ітеративним методом:

Куб C_0 з ребром одиничного розміру ділиться прощинами, паралельними його граням, на двадцять сім рівних кубів. З куба C_0 видаляється центральний куб і все прилягаючі до нього двомірними гранями куби. Вийде множина C_1, що складається з двадцяти кубів. Повторивши процедуру для кожного з цих кубів, отримаємо множину C_2, що складається з чотирьохсот кубів. Продовжуючи цей процес нескінченно, отримаємо послідовність

$$C_0 \supset C_1 \supset \cdots \supset C_n \supset \cdots,$$

перетин членів якої є губкою Менгера.

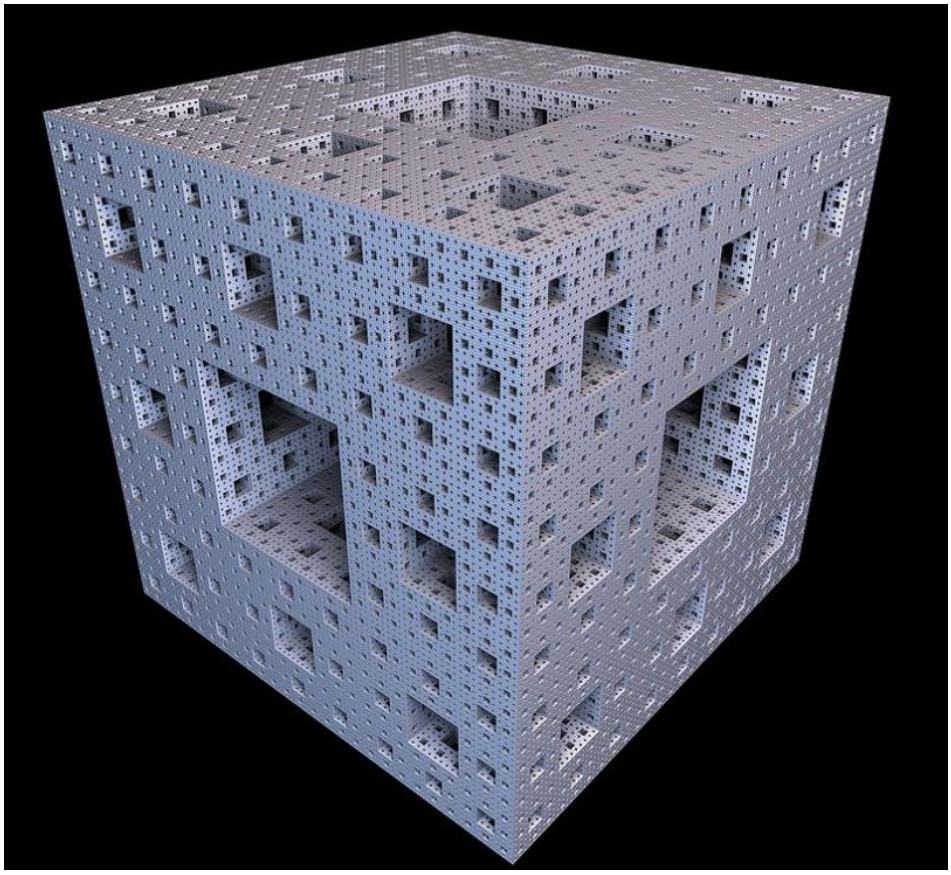

Рисунок 2.4. Губка Менгера (після 6 ітерацій)

2.1.2.4 Функція Веєрштраса

[16] Функція Веєрштраса – ніде не диференційовна неперервна функція.

Визначається на всій дійсній прямій єдиним аналітичним виразом:

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

де a – довільне непарне число, а b – додатне число, менше одиниці.

Функція w неперервна і визначена на всіх дійсних x, але не має похідної принаймні при $ab > \frac{3}{2}\pi + 1$.

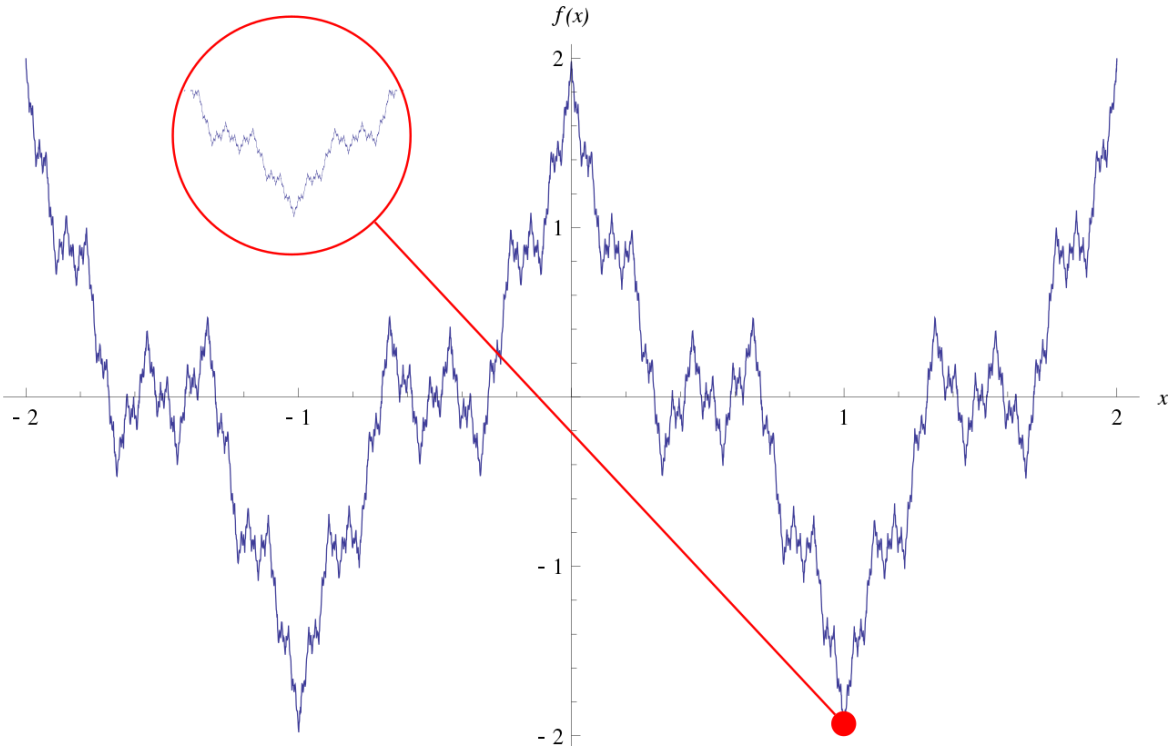

Рисунок 2.5. Функція Веєрштраса

2.1.2.5 Крива Коха

[17] Крива Коха – фрактальна крива, що є всюди неперервною, але ніде не диференційовна.

Побудова:

Потрібно взяти одиничний відрізок, поділити його на три рівні частни і замінити середній інтервал на рівносторонній трикутник без цього сегмента. У результаті утворюється ламана з чотирьох ланок довжиною 1/3 довжини початкового відрізка. На наступних кроках повторюємо операцію для кожного з утворених ланок і так до нескінченності. Гранична крива буде кривою Коха.

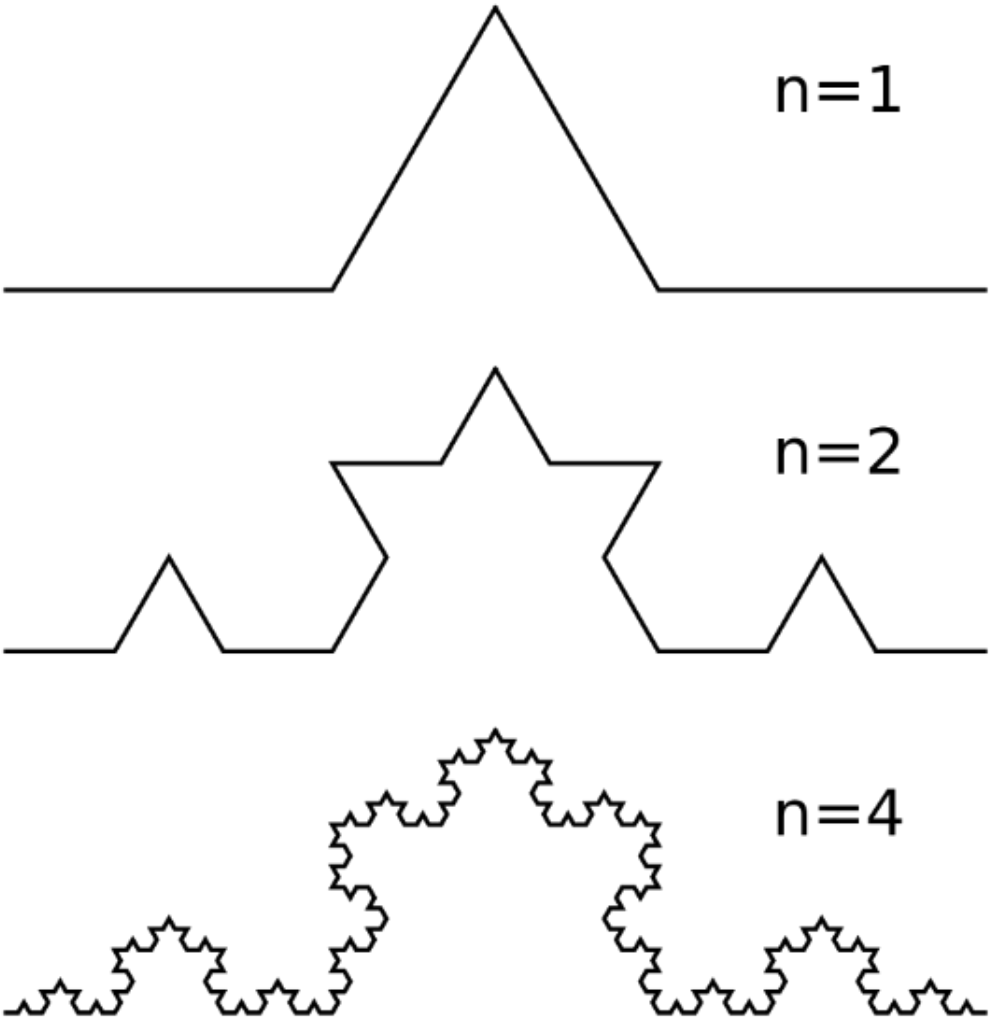

Рисунок 2.6. Ітерації утворення кривої Коха

2.1.3 Види самоподібності

2.1.3.1 Точна самоподібність

Точна самоподібність – найсильніший тип самоподібності. Фрактал виглядає однаково при різних збільшеннях. Точна самоподібність часто виявляється у фракталів, згенерованих з використанням ітеративних функцій. [12]

2.1.3.2 Квазі-самоподібність

Квазі-самоподібність – самоподібність, при якій фрактал має наближений, але не однаковий, візерунок на різних масштабах. Такі фрактали можуть містити маленькі деформовані копії цілого фракталу. Квазі-самоподібними зазвичай є фрактали, згенеровані з використанням рекурентних співвідношень. [12]

2.1.3.3 Статистична самоподібність

Статистична самоподібність – вид самоподібності фракталів, які стохастично повторюють свій шаблон, тому статистичні або чисельні міри зберігаються крізь масштабування. Прикладом є випадково згенеровані фрактали, як відомий фрактал берегової лінії Британії. [12]

2.1.4 Методи генерування фракталів

2.1.4.1 Ітераційні функції

Метод побудови фракталів ітераційними функціями полягає у використанні фіксованих правил геометричних заміщень, які можуть бути стохастичними або детерміністичними.

Приклади: сніжинка Коха, множина кантора, килим Сєрпінського, крива Пеано, губка Менгера. [12]

2.1.4.2 Рекурентні співвідношення

Фрактали, що визначаються рукурентним співвідношенням в кожній точці простору (такому як комплексна площина). Зазвичай є квазі-самоподібними. Такі фрактали також називають фракталами часу втечі або орбітальними.

Приклади: множина Мандельброта, множина Жюліа, фрактал Ньютона, фрактал Ляпунова. [12]

2.1.4.3 Випадкові процеси

Фрактали, що генеруються з використанням стохастичних процесів.

Приклади: фрактальні ландшафти, броунівське дерево, траекторія Леві. [12]

2.1.5 Області застосування

2.1.5.1 Комп’ютерна графіка

Фрактали широко застосовуються у комп’ютерній графіці для побудови зображень природних об’єктів, таких як гірські ландшафти, кущі, дерева, поверхня морів та інших. Також існує багато програм для генерації фрактальних зображень. [12]

2.1.5.2 Стиснення зображень

Існують алгоритми стиснення зображень за допомогою фракталів. Їх ідея заключається в тому, що можна зберігати замість самого зображення

його стискаюче відображення, для якого це зображення, або близьке до нього, є нерухомою точкою. [12]

2.1.5.3 Фрактальні антени

У радіотехніці фрактальна геометрія використовується при проектуванні антенних пристроїв. Перевагою таких антен є відносна широкополосність та багатодіапазонність. [12]

2.1.5.4 Природничі науки

У фізиці фрактали природним чином виникають при моделюванні нелінійних процесів, таких як полум’я, хмари, турбулентий потік рідини і їм подібні. В біології вони використовуються для опису систем внутрішніх органів та моделювання популяцій. Криву Коха було запропоновано застосовувати при обрахуванні довжини берегової лінії. [12]

2.2 Комплексна динаміка

[11] Фрактали природнім чином виникають при вивченні нелінійних динамічних систем. Зазвичай, коли динамічна система задана ітераціями многочлена або голоморфної функції комплексної змінної на площині.

Нехай $F(z)$ – многочлен, z_0 – комплексне число. Розглянемо наступну послідовність:

$$z_0, z_1 = F(z_0), z_2 = F\big(F(z_0)\big) = F(z_1), z_3 = F\left(F\big(F(z_0)\big)\right) = F(z_2), \ldots. \quad (1)$$

Нас цікавить поведінка цієї послідовності при прямуванні n до нескінченності. Тоді послідовність може повести себе таким чином:

- прямувати до скінченної границі;

- прямувати до нескінченності;
- показувати циклічну поведінку;
- показувати хаотичну поведінку, тобто не демонструвати себе жодним чином, з вказаних вище.

Множини значень z_0, для яких послідовність показуватиме конкретний тип поведінки, а також множини точок біфуркації між різними типами, часто володіють властивостями фракталів.

Точка біфуркації – критичний стан системи, при якому вона стає нестійкою до збурень і виникає невизначеність: чи стане система хаотичною, чи перейде на вищий рівень впорядкованості. [18]

2.3 Множина Мандельброта

[19][20] Множина Мандельброта – це множина комплексних точок c, для яких функція $f_c(z) = z^2 + c$ не розбігається при ітерації від $z_0 = 0$. Тобто це множина таких c, для яких існує таке дійсне R, що нерівність $|z_n| < R$ виконується для всіх натуральних n.

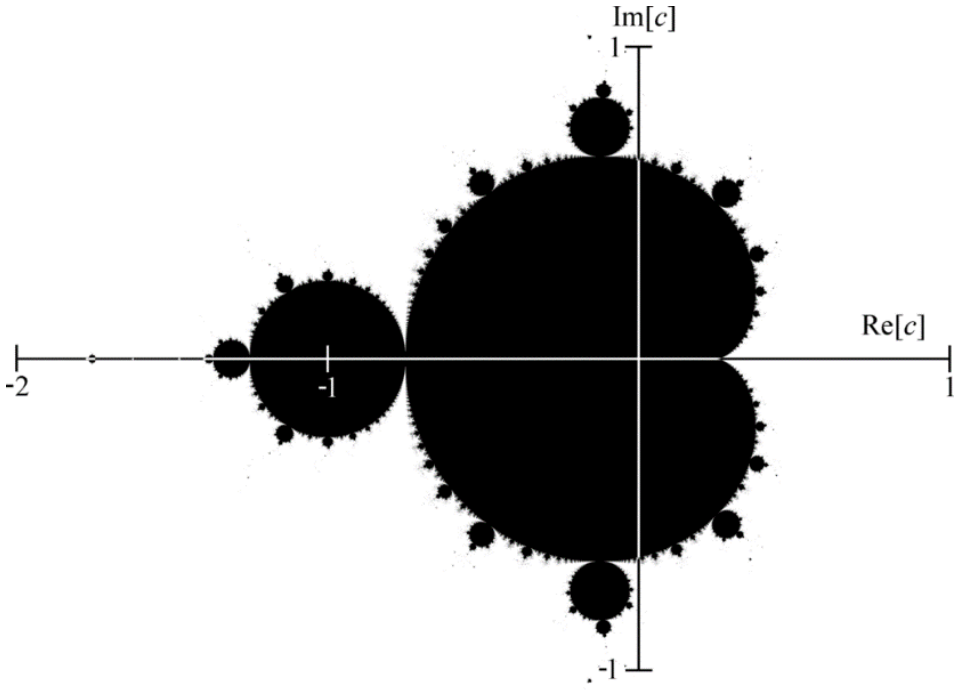

Рисунок 2.6. Множина Мандельброта

Іншими словами, це набір значень комплексного числа c, для якого орбіта числа нуль при ітерації квадратичного полінома $z_n = z_n^2 + c$ залишається обмеженою. Звідси, комплексне число c належить множині Мандельброта якщо, починаючи від $z_0 = 0$ і повторно застосовуючи ітерацію, абсолютне значення z_n залишається обмеженим для будь-якого натурального $n > 0$.

Орбіта точки – значення, що отримуються при послідовному застосуванні функції до заданого елемента. Наприклад, орбіта точки z при дії відображення $f: \mathbb{C} \to \mathbb{C}$ - це послідовність $z, f(z), f\big(f(z)\big), \ldots$. [21]

Ми можемо розкрити вищевказану послідовність для кожної точки c наступним чином:

$$c = x + iy$$

$$Z_0 = 0,$$

$$Z_1 = Z_0^2 + c = x + iy$$

$$Z_2 = Z_1^2 + c = (x + iy)^2 + x + iy = x^2 + 2xiy - y^2 + x + iy =$$

$$= x^2 - y^2 + x + (2xy + y)i,$$

$$Z_3 = Z_2^2 + c = \cdots$$

Якщо звести ці вирази до виду ітеративної послідовності значень координат комплексної площини (x, y), тобто поклавши

$$z_n = x_n + iy_n, \quad (2)$$

$$c = x_0 + iy_0, \quad (3)$$

отримаємо формули, за якими можемо порахувати значення координат точки в наступній ітерації за значенням в попередній.

$$x_{n+1} = x_n^2 - y_n^2 + x_0, \quad (4)$$

$$y_{n+1} = 2x_n y_n + y_0 \quad (5)$$

Хоча зазвичай під множиною Мандельброта мається на увазі множина, описана вище, однак будь-яка функція комплексної змінної має відповідну множину Мандельброта. Наприклад, замість квадратичні функції в основу для побудови множини можна покласти кубічну.

2.4 Множина Жюліа

2.4.1 Означення та застосування в комплексній динаміці

[22][23] Заповнена множина Жюліа $K(p_c)$ многочлена $p_c(z) = z^2 + c$ – множина точок z, орбіти яких при дії p_c не виходять в нескінченність.

Множина Жюліа $J(p_c)$ многочлена p_c – границя заповненої множини Жюліа. В голоморфній динаміці – множина точок, динаміка в околі яких є нестійкою по відношенню до малих збурень початкового положення.

Для послідовності (1), множина Жюліа – множина точок біфуркації для многочлена $F(z) = z^2 + c$ (або іншої схожої функції), тобто тих значень z_0, для яких поведінка послідовності z_n може різко змінюватись при як завгодно малих змінах z_0.

2.4.2 Зв’язок з множиною Мандельброта

[24]Множина Мандельброта по суті є каталогом множин Жюліа: кожній точці на комплексній площині відповідає своя множина Жюліа. Точки, що належать множині Мандельброта, відповідають зв’язним множинам Жюліа, а точки, що не належать, – незв’язним. Найбільш цікаві варіанти множини Жюліа відповідають точкам, що належать границі множини Мандельброта.

Також за допомогою множини Мандельброта можна перевірити зв’язність множини Жюліа для заданої функції. Наприклад, можна покласти $f_c(z) = z^3 + c$. Тоді якщо c належить множині Мандельброта, то множина Жюліа буде зв’язною для заданої функції і значення c. В цьому випадку зв’язність перевіряється тим самим способом, що і для $f_c(z) = z^2 + c$.

РОЗДІЛ 3. ПОБУДОВА МНОЖИН МАНДЕЛЬБРОТА ТА ЖЮЛІА. ЕКСПЕРИМЕНТИ

3.1 Алгоритм побудови множини Мандельброта та Жюліа

[20] Для побудови множини Мандельброта буде застосовано найбільш широко використовуваний і найпростіший “алгоритм часу втечі” (“escape time algorithm”). У основі цього алгоритму лежить проведення ітеративних обрахунків для кожної точки області побудови зображення і встановлення кольору пікселя для цієї точки, в залежності від результату обрахунків.

Значення координат x та y кожної точки використовуються як стартові значення ітеративних обрахунків за формулами (4) та (5), описаних у розділі 2. Результат кожної ітерації використовується як стартові значення для наступної. На кожній ітерації ми перевіряємо, чи досягли значення критичного значення “втечі”, після якого послідовність піде в нескінченність. Якщо досягли критичного значення, зупиняємо цикл обрахунків, малюємо піпсель і переходимо до розгляду наступного.

Колір кожної точки показує наскільки швидко значення послідовності досягають критичного значення, тобто скільки ітерацій обрахунків було витрачено на дослідження даної точки. Чим більше ітерацій, тим яскравішим буде піксель. Білим будуть позначатись точки, які належать множині Мандельброта.

Для побудови множини Жюліа буде використано цей самий алгоритм, але з додатковим заданням дійсного та комплексного значень константи c.

3.2 Програма «Mandelbrot & Julia»

Практичним завданням моєї роботи було комп’ютерне моделювання множин Мандельброта та Жюліа і проведення експериментів над ними.

Для виконання цього завдання мною був розроблений програмний застосунок «Mandelbrot & Julia» з використанням для ілюстрації засобів графічної бібліотеки opencv на мові програмування C++.

В рамках цього проекту під множиною Жюліа матимемо на увазі заповнену множину Жюліа, оскільки саме її зображення будує моя програма.

3.2.1 Принцип роботи програми та її можливості

Програма виконується як exe-файл. При запуску програми з’являється консоль з інструкцією з використання, а саме списком активних клавіш і за що вони відповідають, а також вікно, у якому графічно будуть зображатись множини, і у якому при старті виконання виводиться множина Мандельброта з заданими за замовчуванням параметрами.

Застосунок дає можливість обирати чотири множини (режими роботи), з якими можна працювати: звичайні множини Мандельброта та Жюліа, побудовані на основі квадратичної функції, а також ці ж множини, побудовані на основі кубічної функції.

Для кожної з множин передбачені наступні дії:

- рух вгору, вниз, ліворуч та праворуч (зміна зміщення по осям);
- збільшення та зменшення масштабу;
- зміна максимальної кількості ітерацій (при дуже великих значеннях негативно впливає на продуктивність. За замовчуванням – сто) з кроком в п’ять одиниць.

Крім цього, для квадратичної та кубічної множин Жюліа передбачена можливість зміни дійсного та комплексного значення параметра c з кроком у п’ять сотих.

Також можна змінювати яскравість зображення як коефіцієнта, який буде додаватись до множника (alpha) кольорів, що задаватимуть пікселі.

Усі вище перечислені дані також завжди виводяться на зображення згори.

3.2.2 Кнопки керування

Клавіша 1 – режим множини Мандельброта.

Клавіша 2 – режим множини Жюліа.

Клавіша 3 – режим кубічної множини Мандельброта.

Клавіша 4 – режим кубічної множини Жюліа.

Клавіші - , = – збільшення та зменшення масштабування.

Клавіші w, a, s, d – рух по осям X та Y.

Клавіші , , . – зменшити/збільшити кількість ітерацій на 5.

Клавіші [,] – зменшити/збільшити дійсну частину константи c на 0.1.

Клавіші ; , ‘ – зменшити/збільшити уявну частину константи c на 0.1.

Клавіші q , e – зменшити/збільшити яскравість на 0.01.

Клавіша Esc – закрити вікно зображення.

Для того, щоб натискання клавіш спрацьовували, потрібно щоб вікно графічного представлення було виділеним.

3.2.3 Колірні позначення

На різних інтервалах досягнутої кількості перевірочних ітерацій точки зафарбовуються різними кольорами.

За замовчуванням коефіцієнт яскравості становить 0, і ітераціям відповідатимуть такі кольори:

- 0 ітерацій – чорний колір;
- від 1 до 25 ітерацій – відтінки синього кольору;
- від 26 до 50 ітерацій – відтінки зеленого кольору;
- від 51 до 75 ітерацій – відтінки червоного кольору;
- від 76 до максимальної кількості ітерацій – відтінки сірого кольору.

За замовчуванням, точки, що належать множинам, будуть зафарбовані білим кольором.

3.3 Експерименти над множинами Мандельброта та Жюліа. Демонстрація роботи програми «Mandelbrot & Julia»

Запускаємо програму.

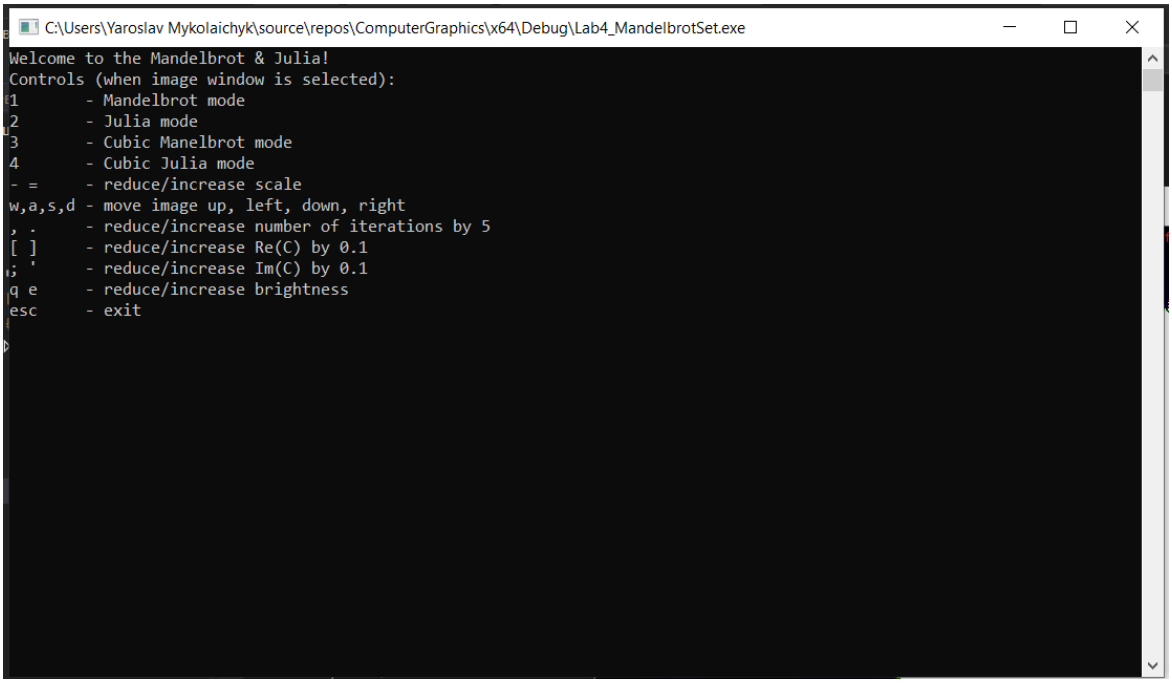

Рисунок 3.1 Вікно консолі з інструкцією

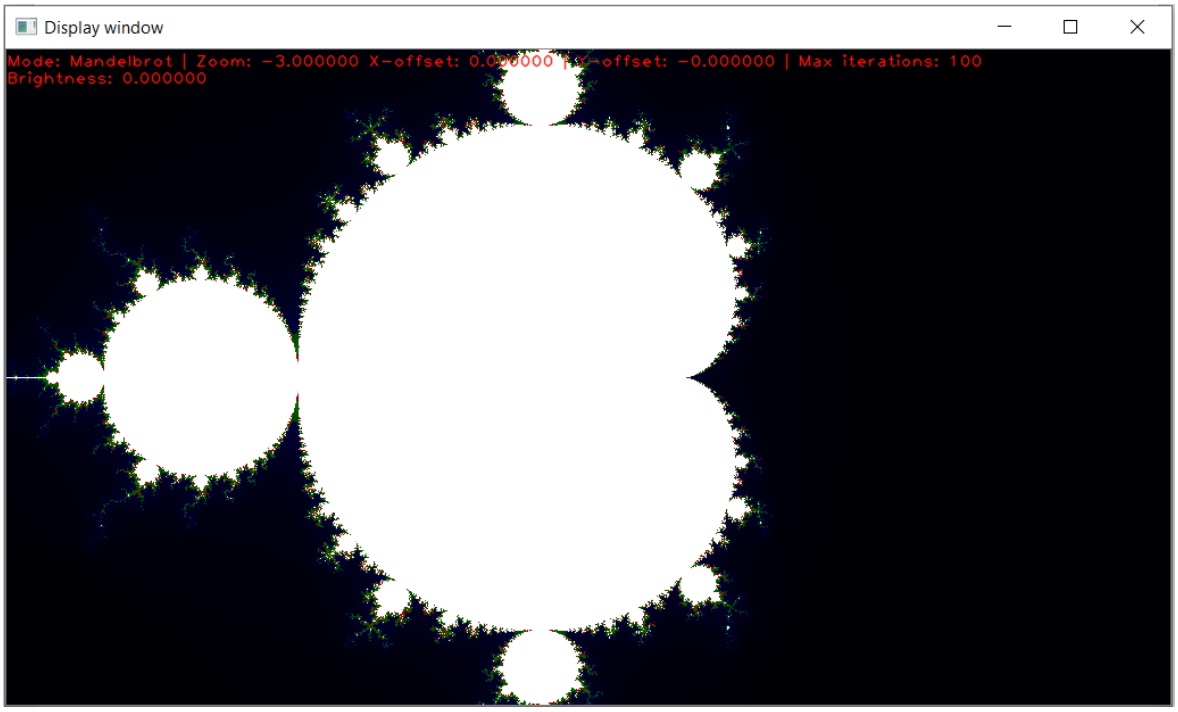

Рисунок 3.2 Вікно графічного представлення з множиною Мандельброта

Можемо бачити, як відкрилось два вікна: консоль з підказками до користування і вікно графічного представлення, на якому ілюстровано множину Мандельброта. У верхній частині зображення також видно допоміжні дані, а саме назву множини, що представляється(“mode”), масштабування(“zoom”), відступ від початкової позиції по координатам(“X-offset”, “Y-offset”), максимальну кількість ітерацій(“Max iterations”), а також яскравість(“brightness”).

Подивимось на множину Жюліа:

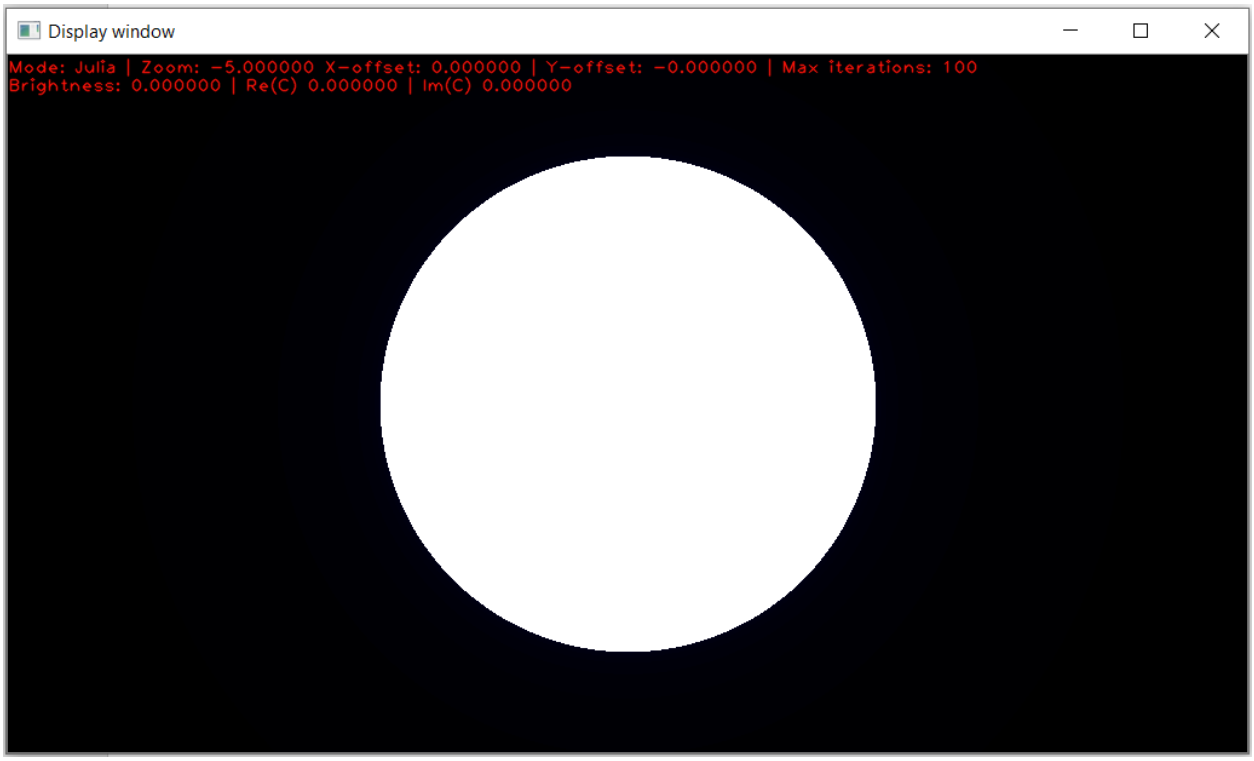

Рисунок 3.3 Графічне представлення множини Жюліа з вхідними параметрами (0,0)

Перемкнувшись в режим відображення множини Жюліа і трохи зменшивши масштаб бачимо, що з вхідними значеннями за замовчуванням Re(c) = 0 і Im(c) = 0, множина має форму кола і не має ніяких особливостей.

Спробуємо поекспериментувати з множинами.

3.3.1 Зміна координати початкової точки множини Жюліа

Спробуємо змінити вхідні координати для побудови множини Жюліа і подивимось, як вона виглядатиме.

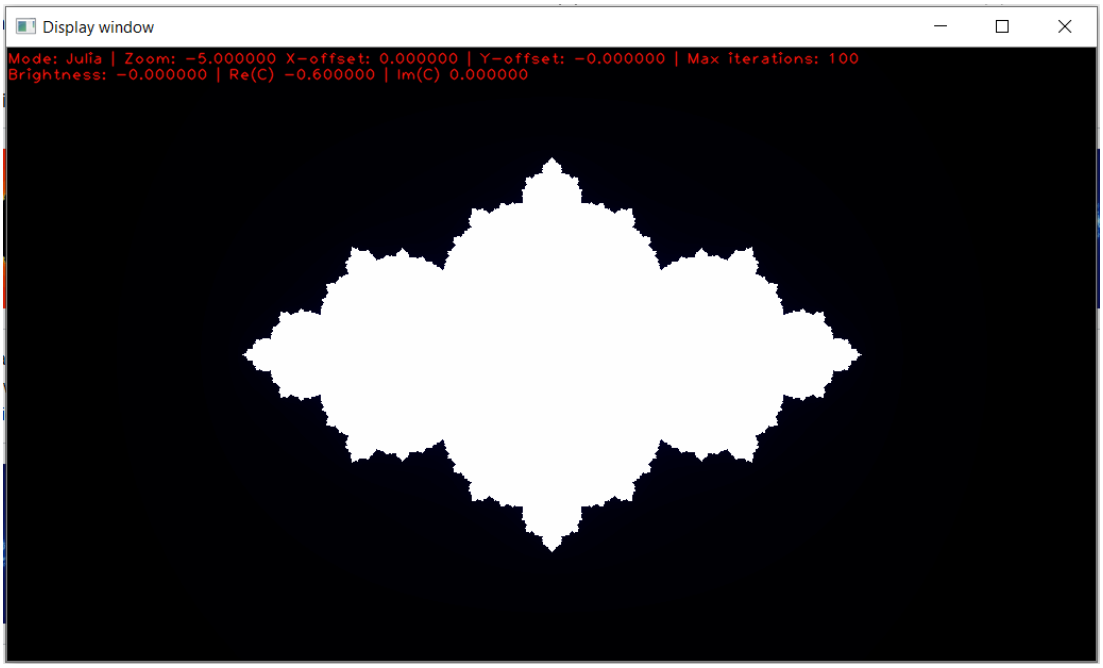

Рисунок 3.4 Множина Жюліа. Re(c) = -0.6

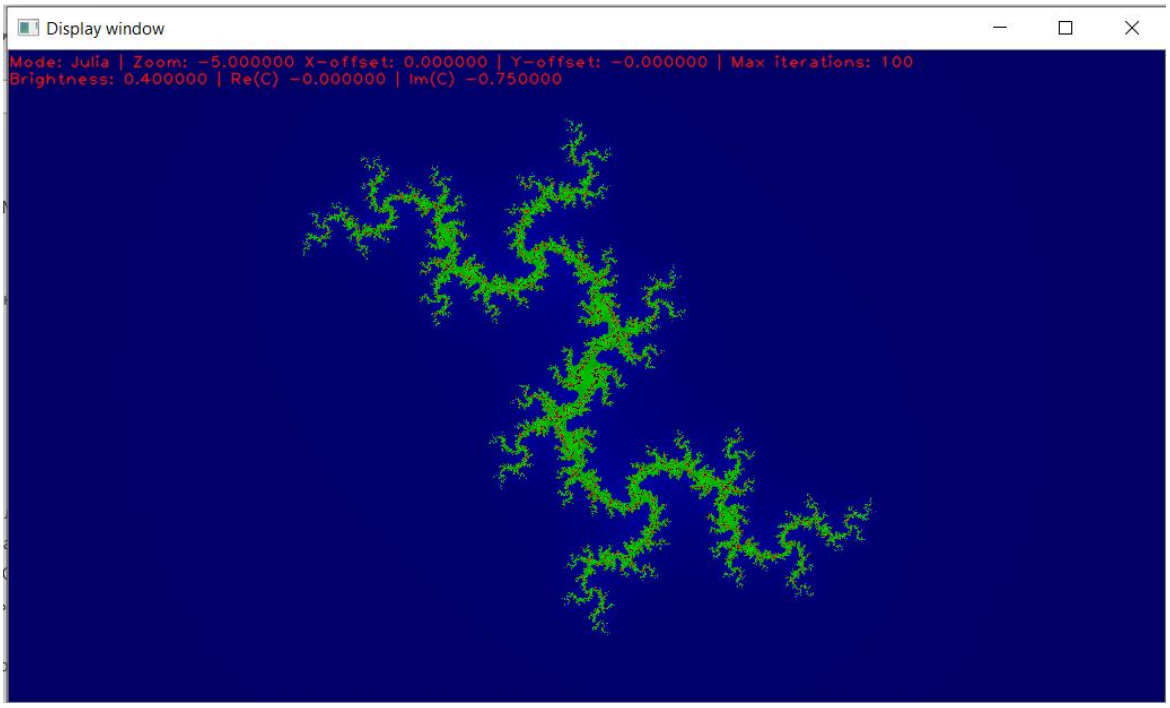

Рисунок 3.5 Множина Жюліа. Im(c) = -0.75. Brightness = 0.4

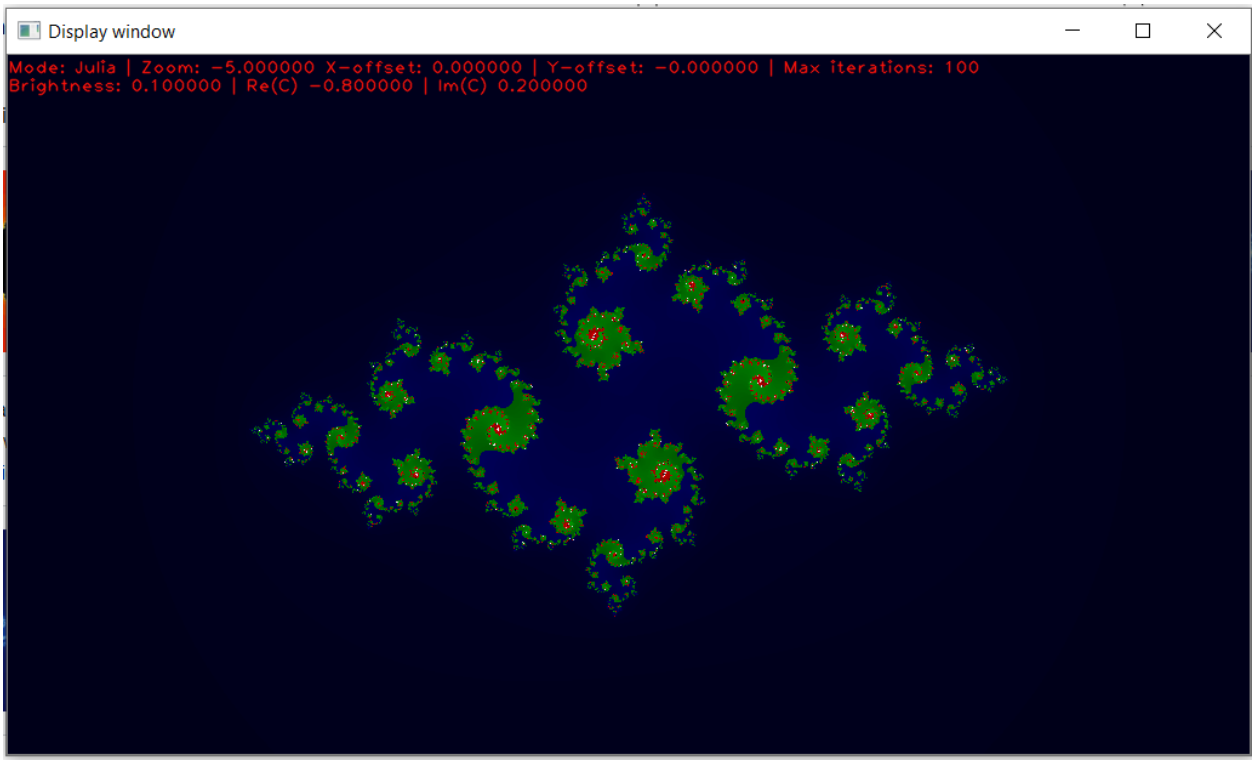

Рисунок 3.6 Множина Жюліа. Re(c) = -0.8, Im(c) = 0.2, Brightness = 0.1

3.3.2 Зміна максимальної кількості ітерацій

Тепер спробуємо змінити максимальну кількість ітерацій.

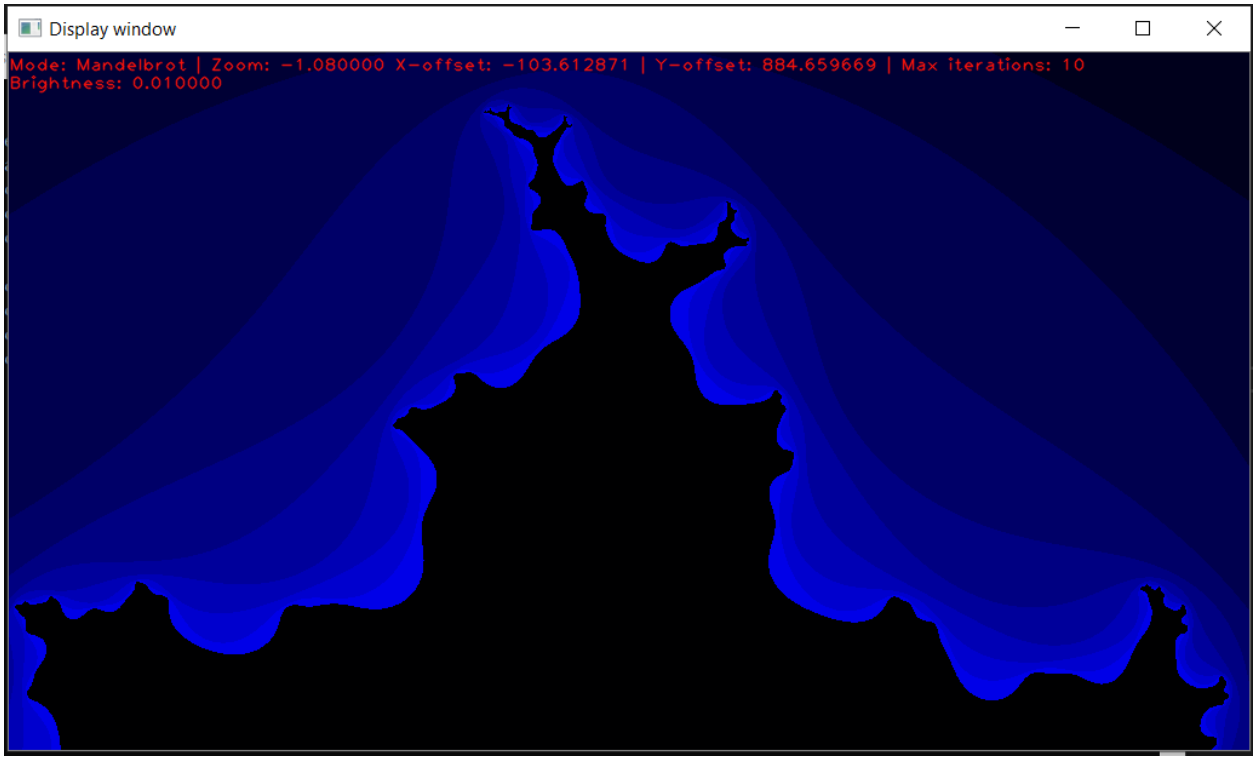

Рисунок 3.7 Множина Мандельброта. 10 ітерацій

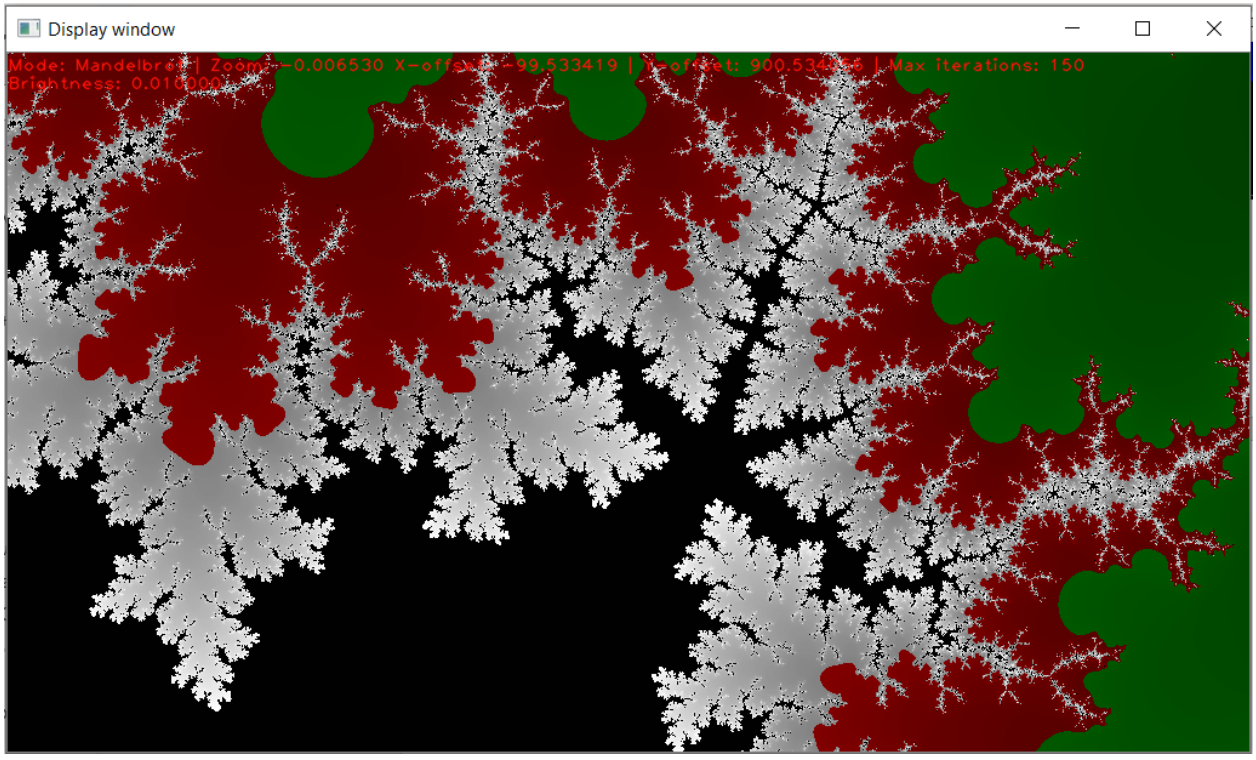

Рисунок 3.8 Множина Мандельброта. 150 ітерацій, віддалення – 0.0065

Як бачимо, при збільшенні кількості ітерацій чіткість множини багатократно збільшується і в рази переважає чіткість при малій кількості ітерацій навіть у значно приближених масштабах.

3.3.3 Зміна квадратичної функції на кубічну

Для побудови множин від кубічних функцій потрібно знайти формули для знаходження значень координат точки z_n на наступному кроці за попереднім, аналогічно до формул (4),(5) (див. Розділ 2.3). Але тепер ми це зробимо для кубічної функції

$$z_{n+1} = z_n^3. \tag{6}$$

Покладемо рівняння (2), (3) в (5). Отримаємо

$$x_{n+1} + iy_{n+1} = (x_n + iy_n)^3 + c. \tag{7}$$

Розкриємо дужки за формулою бінома Ньютона степеня 3:

$$x_{n+1} + iy_{n+1} = x_n^3 + 3x_n^2 y_n i - 3x_n y_n^2 - y_n^3 i + x_0 + iy_0 \qquad (8)$$

Виділимо цілу і комплексну частину з (8) і отримаємо шукані ітеративні співвідношення:

$$x_{n+1} = x_n^3 - 3x_n y_n^2 + x_0, \qquad (9)$$

$$y_{n+1} = 3x_n^2 y_n - y_n^3 + y_0. \qquad (10)$$

Отже, за формулами (9), (10) ми можемо побудувати нашу послідовність для пошуку кубічних множин Мандельброта та Жюліа, що і було мною зроблено під час розробки «Mandelbrot & Julia».

Кубічні множини Мандельброта та Жюліа матимуть наступний вигляд:

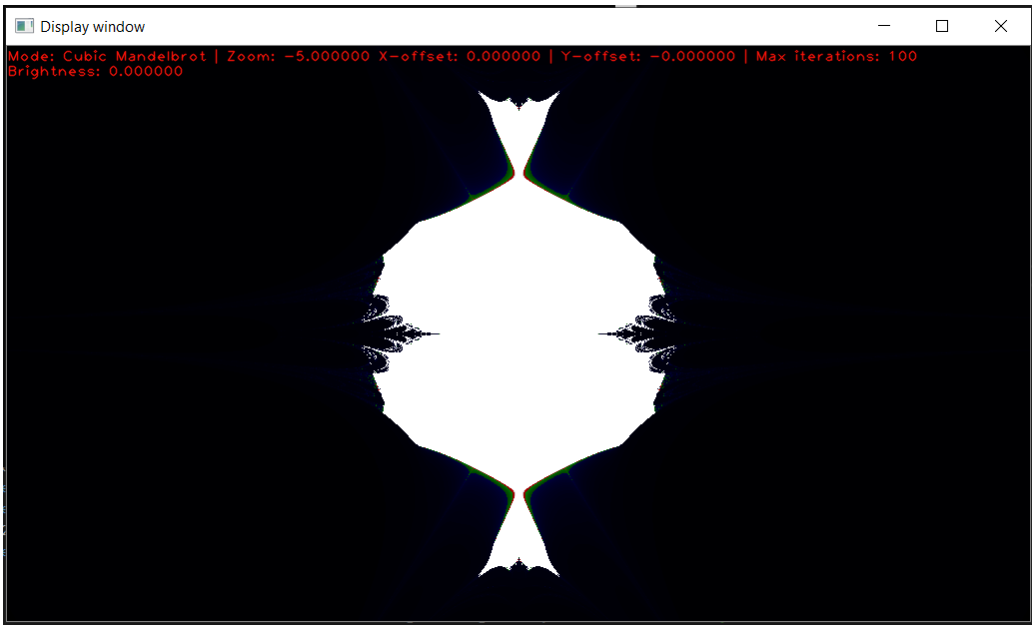

Рисунок 3.9 Кубічна множина Мандельброта

Рисунок 3.10 Кубічна множина Жюліа

ВИСНОВКИ

Отже, в ході даної роботи мною було досліджено суть та особливості фрактальних множин. Мною було розглянуто приклади практичного застосування цих множин для вирішення специфічних проблем. Також було розглянуто особливості та шляхи побудови конкретних фракталів. Особливу увагу я зосередив увагу на таких фрактальних множинах, як множина Мандельброта та множина Жюліа, а також навчився і практично виконав завдання з їх проектування за допомогою програмного засобу і проведення над ними дослідницьких експериментів.

ЛІТЕРАТУРА ТА ДЖЕРЕЛА

1. Комплексное число [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Комплексное_число
2. Complex plane [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Complex_plane
3. Евклидово пространство [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Евклидово_пространство
4. Многовид [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Многовид
5. Голоморфная функция [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Голоморфная_функция
6. Голоморфная динамика [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Голоморфная_динамика
7. Самоподобие [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Самоподобие
8. Фазовое пространство [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Фазовое_пространство
9. Динамическая система [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Динамическая_система
10. Граница (топология) [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Граница_(топология)
11. Фрактал [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Фрактал

12. Фрактал [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Фрактал

13. Cantor set [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cantor_set

14. Треугольник Серпинского [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Треугольник_Серпинского

15. Губка Менгера [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Губка_Менгера

16. Функція Веєрштраса [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Функція_Веєрштраса

17. Крива Коха [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Крива_коха

18. Точка біфуркації [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Точка_біфуркації

19. Mandelbrot set [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Mandelbrot_set

20. Множина Мандельброта [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Множина_Мандельброта

21. Обріта [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Орбита_(значения)

22. Множество мандельброта. Лекция 1. Владлен Тиморин [Електронний ресурс] // youtube.com – Режим доступу до ресурсу: https://www.youtube.com/watch?v=nVDrdGkLP9c

23. Julia set [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Julia_set

24. Множество Жюлиа [Електронний ресурс] // wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Множество_Жюлиа

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ “MANDELBROT & JULIA”

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <vector>
#include <thread>
#include <opencv2/imgproc.hpp>

const int WIDTH = 1024;
const int HEIGHT = 576;
const float ASPECT_RATIO = (float)WIDTH / HEIGHT; // for proper representation

int mandelbrot(double, double, int);
int mandelbrotJulia(double, double, double, double, int);
int mandelbrotCubic(double, double, int);
int mandelbrotCubicJulia(double, double, double, double, int);
void updateImage(double, double, double, cv::Mat&);
void updateImageJulia(double, double, double, double, double, cv::Mat&);
void updateImageSlice(double, double, double, cv::Mat&, int, int);
void updateImageSliceJulia(double, double, double, double, double, cv::Mat&, int,
int);
void updateImageCubic(double, double, double, const cv::Mat&);
void updateImageCubicJulia(double, double, double, float, float, const cv::Mat&);
void updateImageSliceCubic(double, double, double, cv::Mat&, int, int);
void updateImageSliceCubicJulia(double, double, double, double, double, cv::Mat&,
int, int);


cv::Vec3b getColor(int);

// default values
double zoom = 3;
double offsetX = 0.0;
double offsetY = 0.0;
int MAX = 100;
float realC = 0;
float imagC = 0;
float brightness = 0;

enum Mode { Mandelbrot, Julia, CubicMandelbrot, CubicJulia };
Mode mode = Mandelbrot;

// converts enum to string
inline char* ToString(Mode m)
{
	switch (m)
	{
	case Mandelbrot:   return "Mandelbrot";
	case Julia:   return "Julia";
	case CubicMandelbrot: return "Cubic Mandelbrot";
	case CubicJulia: return "Cubic Julia";
	}
}
```

```
int main()
{
	// creating canvas
	cv::Mat Image(HEIGHT, WIDTH, CV_8UC3, cv::Scalar(0));

	// creating window and displaying canvas in the window
	cv::namedWindow("Display window", cv::WINDOW_AUTOSIZE);
	cv::imshow("Display window", Image);

	// instructions
	std::cout << "Welcome to the Mandelbrot & Julia! \n" <<
		"Controls (when image window is selected): \n" <<
		"1	- Mandelbrot mode \n" <<
		"2	- Julia mode \n" <<
		"3	- Cubic Manelbrot mode \n" <<
		"4	- Cubic Julia mode \n" <<
		"- =	- reduce/increase scale \n" <<
		"w,a,s,d - move image up, left, down, right \n" <<
		", .	- reduce/increase number of iterations by 5 \n" <<
		"[ ]	- reduce/increase Re(C) by 0.1 \n" <<
		"; '	- reduce/increase Im(C) by 0.1 \n" <<
		"q e	- reduce/increase brightness \n" <<
		"esc	- exit" << std::endl;



	char key;
	bool stateChanged = true; // for initialization at startup

	while(true)
	{
		if (stateChanged)
		{
			stateChanged = false;

			if (mode == Mode::Mandelbrot)
				updateImage(zoom, offsetX, offsetY, Image);
			else if (mode == Mode::Julia)
				updateImageJulia(zoom, offsetX, offsetY, realC, imagC,
Image);
			else if (mode == Mode::CubicMandelbrot)
				updateImageCubic(zoom, offsetX, offsetY, Image);
			else
				updateImageCubicJulia(zoom, offsetX, offsetY, realC, imagC,
Image);

			std::string state_1 = "Mode: " + std::string(ToString(mode)) +
(std::string)" | Zoom: " + std::to_string(-zoom)
				+ " X-offset: " + std::to_string(offsetX) + " | Y-offset: "
+ std::to_string(-offsetY) + " | Max iterations: " + std::to_string(MAX);
			std::string state_2 = "Brightness: " + std::to_string(brightness)
				+ std::string(!(mode == Mode::Julia || mode ==
Mode::CubicJulia) ? "" : " | Re(C) " + std::to_string((realC)))
				+ std::string(!(mode == Mode::Julia || mode ==
Mode::CubicJulia) ? "" : " | Im(C) " + std::to_string((imagC)));


			// display stats on the image
			cv::putText(Image, state_1, cv::Point(0,15),
cv::HersheyFonts::FONT_HERSHEY_PLAIN, 1.0, cv::Vec3b(10, 20, 255), 1, cv::LINE_AA);
```

```
                cv::putText(Image, state_2, cv::Point(0, 30),
cv::HersheyFonts::FONT_HERSHEY_PLAIN, 1.0, cv::Vec3b(10, 20, 255), 1, cv::LINE_AA);

                cv::imshow("Display window", Image);

        }

        key = cv::waitKey(0);

        if (key != -1)
        {
                if (key == 27)
                        break;
                switch (key)
                {
                case '=':
                        zoom *= 0.6;
                        stateChanged = true;
                        break;
                case '-':
                        zoom /= 0.6;
                        stateChanged = true;
                        break;
                case 'w':
                        offsetY -= 100 * zoom;
                        stateChanged = true;
                        break;
                case 's':
                        offsetY += 100 * zoom;
                        stateChanged = true;
                        break;
                case 'a':
                        offsetX -= 100 * zoom;
                        stateChanged = true;
                        break;
                case 'd':
                        offsetX += 100 * zoom;
                        stateChanged = true;
                        break;
                case '[':
                        if(mode == Mode::Julia || mode == Mode::CubicJulia)
                        {
                                stateChanged = true;
                                realC -= 0.05;
                        }
                        break;
                case ']':
                        if (mode == Mode::Julia || mode == Mode::CubicJulia)
                        {
                                stateChanged = true;
                                realC += 0.05;
                        }
                        break;
                case ';':
                        if (mode == Mode::Julia || mode == Mode::CubicJulia)
                        {
                                stateChanged = true;
                                imagC -= 0.05;
                        }
                        break;
                case '\'':
```

```
                        if (mode == Mode::Julia || mode == Mode::CubicJulia)
                        {
                              stateChanged = true;
                              imagC += 0.05;
                        }
                        break;
                  case '1':
                        mode = Mode::Mandelbrot;
                        stateChanged = true;
                        break;
                  case '2':
                        mode = Mode::Julia;
                        stateChanged = true;
                        break;
                  case '3':
                        mode = Mode::CubicMandelbrot;
                        stateChanged = true;
                        break;
                  case '4':
                        mode = Mode::CubicJulia;
                        stateChanged = true;
                        break;
                  case ',':
                        MAX -= 5;
                        stateChanged = true;
                        break;
                  case '.':
                        MAX += 5;
                        stateChanged = true;
                        break;
                  case 'q':
                        brightness -= 0.01;
                        stateChanged = true;
                        break;
                  case 'e':
                        brightness += 0.01;
                        stateChanged = true;
                        break;
                  default:
                        break;
                  }
            }
      }

      return(0);
}

// update horisontal segment of the image (Cubic Julia)
void updateImageSliceCubicJulia(double zoom, double offsetX, double offsetY, double
reC, double imC, cv::Mat& image, int minY, int maxY)
{
      for (int x = 0; x < WIDTH; x++) {
            for (int y = minY; y < maxY; y++) {
                  // convert x and y to the appropriate complex number
                  double real = (x - WIDTH / 2.0) * zoom + offsetX;
                  double imag = (y - HEIGHT / 2.0) * zoom + offsetY;
                  real = real / WIDTH;
                  imag = imag / (HEIGHT * ASPECT_RATIO);
                  int value = mandelbrotCubicJulia(real, imag, reC, imC, MAX);
                  image.at<cv::Vec3b>(y, x) = getColor(value);
            }
```

```
	}
}

// update horisontal segment of the image (Cubic Mandelbrot)
void updateImageSliceCubic(double zoom, double offsetX, double offsetY, cv::Mat&
image, int minY, int maxY)
{
	for (int x = 0; x < WIDTH; x++) {
		for (int y = minY; y < maxY; y++) {
			// converting coordinates of the window to coordinates of
			double real = (x - WIDTH / 2.0) * zoom + offsetX;
			double imag = (y - HEIGHT / 2.0) * zoom + offsetY;
			real = real / WIDTH;
			imag = imag / (HEIGHT * ASPECT_RATIO);
			int value = mandelbrotCubic(real, imag, MAX);
			image.at<cv::Vec3b>(y, x) = getColor(value);
		}
	}
}


// update horisontal segment of the image (Julia)
void updateImageSliceJulia(double zoom, double offsetX, double offsetY, double reC,
double imC, cv::Mat& image, int minY, int maxY)
{
	for (int x = 0; x < WIDTH; x++) {
		for (int y = minY; y < maxY; y++) {
			// convert x and y to the appropriate complex number
			double real = (x - WIDTH / 2.0) * zoom + offsetX;
			double imag = (y - HEIGHT / 2.0) * zoom + offsetY;
			real = real / WIDTH;
			imag = imag / (HEIGHT * ASPECT_RATIO);
			int value = mandelbrotJulia(real, imag, reC, imC, MAX);
			image.at<cv::Vec3b>(y, x) = getColor(value);
		}
	}
}

// update horisontal segment of the image (Mandelbrot)
void updateImageSlice(double zoom, double offsetX, double offsetY, cv::Mat& image,
int minY, int maxY)
{
	for (int x = 0; x < WIDTH; x++) {
		for (int y = minY; y < maxY; y++) {
			// converting coordinates of the window to coordinates of
			double real = (x - WIDTH / 2.0) * zoom + offsetX;
			double imag = (y - HEIGHT / 2.0) * zoom + offsetY;
			real = real/WIDTH;
			imag = imag / (HEIGHT * ASPECT_RATIO);
			int value = mandelbrot(real, imag, MAX);
			image.at<cv::Vec3b>(y, x) = getColor(value);
		}
	}
}

// multithreaded update of the image (Cubic Julia)
void updateImageCubicJulia(double zoom, double offsetX, double offsetY, float reC,
float imC, const cv::Mat &image)
{
	// multithreading
	const int STEP = HEIGHT / std::thread::hardware_concurrency();
```

```
	std::vector<std::thread> threads;
	for (int i = 0; i < HEIGHT; i += STEP) {
		threads.push_back(std::thread(updateImageSliceCubicJulia, zoom, offsetX,
offsetY,
			reC, imC, image, i, std::min(i + STEP, HEIGHT)));
	}
	for (auto &t : threads) {
		t.join();
	}
}

// multithreaded update of the image (Cubic Mandelbrot)
void updateImageCubic(double zoom, double offsetX, double offsetY, const cv::Mat
&image)
{
	//multithreading
	const int STEP = HEIGHT / std::thread::hardware_concurrency();
	std::vector<std::thread> threads;
	for (int i = 0; i < HEIGHT; i += STEP) {
		threads.push_back(std::thread(updateImageSliceCubic, zoom, offsetX,
offsetY,
			image, i, std::min(i + STEP, HEIGHT)));
	}
	for (auto &t : threads) {
		t.join();
	}
}


// multithreaded update of the image (Julia)
void updateImageJulia(double zoom, double offsetX, double offsetY, double reC, double
imC, cv::Mat &image)
{
	// multithreading
	const int STEP = HEIGHT / std::thread::hardware_concurrency();
	std::vector<std::thread> threads;
	for (int i = 0; i < HEIGHT; i += STEP) {
		threads.push_back(std::thread(updateImageSliceJulia, zoom, offsetX,
offsetY,
			reC, imC, std::ref(image), i, std::min(i + STEP, HEIGHT)));
	}
	for (auto &t : threads) {
		t.join();
	}
}

// multithreaded update of the image (Mandelbrot)
void updateImage(double zoom, double offsetX, double offsetY, cv::Mat &image)
{
	//multithreading
	const int STEP = HEIGHT / std::thread::hardware_concurrency();
	std::vector<std::thread> threads;
	for (int i = 0; i < HEIGHT; i += STEP) {
		threads.push_back(std::thread(updateImageSlice, zoom, offsetX, offsetY,
			std::ref(image), i, std::min(i + STEP, HEIGHT)));
	}
	for (auto &t : threads) {
		t.join();
	}
}
```

```
// find depth of iteration for cubic mandelbrot set
int mandelbrotCubic(double startReal, double startImag, int max)
{
	double zReal = startReal;
	double zImag = startImag;

	for (int counter = 0; counter < MAX; ++counter) {
		double r2 = zReal * zReal;
		double i2 = zImag * zImag;
		if (r2 + i2 > 8.0) {
			return counter;
		}
		zImag = 3.0 * zReal * zReal * zImag - zImag * zImag * zImag + startImag;
// yn+1 = 3x^2y - y^3 + y0
		zReal = zReal * zReal * zReal - 3.0 * zReal * zImag * zImag + startReal;
// xn+1 = x^3 - 3xy^2 + x0
	}
	return MAX;
}

// find depth of iteration for cubic julia set
int mandelbrotCubicJulia(double startReal, double startImag, double reC, double imC,
int max)
{
	double zReal = startReal;
	double zImag = startImag;

	for (int counter = 0; counter < MAX; ++counter) {
		double r2 = zReal * zReal;
		double i2 = zImag * zImag;
		if (r2 + i2 > 8.0) {
			return counter;
		}
		zImag = 3.0 * zReal * zReal * zImag - zImag * zImag * zImag + imC; //
yn+1 = 3x^2y - y^3 + y0
		zReal = zReal * zReal * zReal - 3.0 * zReal * zImag * zImag + reC; //
xn+1 = x^3 - 3xy^2 + x0
	}
	return MAX;
}

// find depth of iteration for mandelbrot set
int mandelbrot(double startReal, double startImag, int max) {
	double zReal = startReal;
	double zImag = startImag;

	for (int counter = 0; counter < MAX; ++counter) {
		double r2 = zReal * zReal;
		double i2 = zImag * zImag;
		if (r2 + i2 > 4.0) {
			return counter;
		}
		zImag = 2.0 * zReal * zImag + startImag;
		zReal = r2 - i2 + startReal;
	}
	return MAX;
}

// find depth of iteration for julia set
int mandelbrotJulia(double startReal, double startImag, double reC, double imC, int
max) {
```

```
	double zReal = startReal;
	double zImag = startImag;

	for (int counter = 0; counter < MAX; ++counter) {
		double r2 = zReal * zReal;
		double i2 = zImag * zImag;
		if (r2 + i2 > 4.0) {
			return counter;
		}
		zImag = 2.0 * zReal * zImag + imC;
		zReal = r2 - i2 + reC;
	}
	return MAX;
}

// get the color depending on amount of iterations
cv::Vec3b getColor(int iterations)
{
	int r, g, b;
	double alpha = (float)iterations / MAX + brightness;

	if (iterations <= 25) {
		r = 0;
		g = 0;
		b = 255 * alpha;
	}
	else if (iterations <= 50) {
		r = 0;
		g = 255 * alpha;
		b = 0;
	}
	else if (iterations <= 75) {
		r = 255 * alpha;
		g = 0;
		b = 0;
	}
	else { // <= max
		r = 255 * alpha;
		g = 255 * alpha;
		b = 255 * alpha;
	}
	return cv::Vec3b(b, g, r);
}
```