

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

ГЕНЕРУВАННЯ МНОЖИН МАНДЕЛЬБРОТА І ЖУЛІА

**Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення”**

Керівник курсової роботи
к.ф-м.н., доц. Бублик В. В.

_____ (підпис)
“ _____ ” _____ 2020 р.

Виконав студент Осадчук В. І.
“ _____ ” _____ 2020 р.

Київ 2020

Зміст

Вступ.....	5
Анотація	6
Біографія.....	7
Бенуа Мандельброт	7
Гастон Жуліа.....	7
Математичні поняття	8
Множина Мандельброта.....	8
Множина Жуліа.....	9
Зв'язок між множинами Мандельброта та Жуліа.....	10
Практична частина	11
Проблеми обчислення множин.....	11
Вирішення проблем	12
Поєднання множин з зображеннями	13
Розрахунок множин	14
Підрахунок функції.....	16
Результати програми.....	18
Багатопоточні обчислення.....	20
Забарвлення зображень.....	22
Порівняння продуктивності та результати обчислень	26
Висновки	27
Список джерел.....	28
Додатки.....	29
Головний метод для генерації Множини Мандельброта.....	29
Підрахунок кількості ітерацій в Множині Мандельброта.....	30
Отримання кольору для пікселя в Множині Мандельброта.....	30
Головний метод для генерації Множини Жуліа	31
Обчислення R для Множини Жуліа	32
Підрахунок кількості ітерацій в Множині Жуліа	32
Отримання кольору для пікселя в Множині Жуліа.....	32

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

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.ф.-м.н.

_____ О.П.Жежерун

(підпис)

“ ____ ” _____ 20__ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту _____ Осадчуку Володимиру _____

____ 3-го _____ курсу факультету інформатики

ТЕМА: _____ Генерування множин Мандельброта і Жуліа _____

Вихідні дані:

- Програма, що генерує множини Мандельброта та Жуліа у графічному вигляді.

Зміст ТЧ до курсової роботи:

1. Вступ
2. Анотація
3. Біографія
4. Математичні поняття
5. Практична частина
6. Порівняння продуктивності та результати обчислень
7. Висновки
8. Додатки (за необхідністю)

Дата видачі “ ____ ” _____ 202__ р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Генерування множин Мандельброта і Жуліа

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2019 р.	
2.	Огляд літератури за темою роботи	листопад - грудень 2019 р.	
3.	Оволодіння навичками розробки вільних від блокувань структур даних	грудень - лютий 2019 р.	
4.	Створення практичної частини роботи	лютий - березень 2020 р.	
5.	Написання пояснювальної роботи	березень 2020 р.	
6.	Створення слайдів для доповіді та написання доповіді	березень 2020 р.	
7.	Надання роботи керівнику для перевірки	березень - квітень 2020 р.	
8.	Корегування роботи за результатами перевірки керівником	квітень 2020 р.	
9.	Остаточне оформлення пояснювальної роботи та слайдів	квітень- травень 2020 р.	
10.	Захист курсової роботи	15 травня 2020 р.	

Студент Осадчук В.І. _____

Керівник Бублик В.В. _____

“ _____ ” _____ р.

Вступ

Багатьом невідомо, яка саме множина Жуліа чи Мандельброта. Дехто знає, що це фрактали і що вони мають деякі особливі властивості (наприклад, пов'язані зі збільшенням), але мало хто знає, що це саме, і як їх знайти.

Моя мета пояснити, що це таке. Спочатку більш математичним способом. Це може бути трохи важко зрозуміти. Однак друга частина підходить до множин більш практичним чином, як їх можна обчислити мовою програмування.

Анотація

Роботу присвячено фрактальним зображенням, роботі з графікою та з комплексними числами в C#.NET. Порівняння з мовою C++, різні підходи до завдання, асинхронні функції та багатопотокове програмування. На основі результатів дослідження було зроблено програму генерування множин Мандельброта та Жуліа, і подання результату в графічному вигляді.

Біографія

Бенуа Мандельброт

(20 листопада 1924 – 14 жовтня 2010)

Французько-американський математик єврейського походження, засновник фрактальної геометрії. Його ім'я відоме багатьом в зв'язку з фракталом, названим на його честь, — множиною Мандельброта.

Математик також займався економікою, теорією інформації, космологією та іншими науками. Мандельброт є лауреатом премії Вольфа з фізики у 1993 році, Японської премії за інноваційні ідеї в науці у 2003 році та інших численних нагород.

Гастон Жуліа

(34 лютого 1893 – 1 березня 1978)

Французький математик, що відкрив множину Жуліа. В 1970 році його роботи популяризував Бенуа Мандельброт.

Досліджував і отримав результати в теорії конформних відображень та їхнє застосування до функціональних рівнянь. Займався теорією Гільбертових просторів, в якій вирішив деякі питання, наприклад проблема моментів.

Математичні поняття

Множина Мандельброта

Множина Мандельброта – це набір комплексних чисел, що визначені наступним чином:

$$M = \{c \in \mathbb{C} \mid \lim_{n \rightarrow \infty} Z_n \neq \infty\}$$

Де:

$$Z_0 = c$$

$$Z_{n+1} = Z_n^2 + c$$

Це означає, що Множина Мандельброта – це множина всіх комплексних чисел, що відповідають умові, описаній вище. Тобто якщо значення рекурсивної функції Z_n для значення c не є нескінченним при $n \rightarrow \infty$, то комплексне число c належить до набору.

Іншими словами кажуть, що ця функція має атрактор, який знаходиться в нескінченності.

Атрактори пов'язані з “орбітою” функції. Вона утворена значеннями Z на кожному кроці n . На кожному кроці орбіта або тяжіє до атрактору, або ні. Атрактор не завжди може знаходитись в нескінченності, більше того, він може бути не один (фрактал magnet1).

Множина Жуліа

Математичне визначення множини Жуліа ідентичне до множини Мандельброта:

$$J = \{c \in \mathbb{C} \mid \lim_{n \rightarrow \infty} Z_n \neq \infty\}$$

Де:

$$Z_0 = c$$

$$Z_{n+1} = Z_n^2 + K$$

Єдина відмінність, що при обрахунку Z_{n+1} ми замість c додаємо K – це наперед визначене комплексне число.

Це означає, що існує безліч наборів Жуліа (один для кожного числа K).

Зв'язок між множинами Мандельброта та Жуліа

Тож який зв'язок між цими множинами окрім того, що їхні формули майже однакові?

Насправді, якщо взяти будь-яке комплексне число K , що знаходиться всередині множини Мандельброта, то множина Жуліа для нього буде зв'язною, тобто з будь-якої точки множини можна потрапити в будь-яку іншу точку цієї ж множини, що легко побачити на зображенні. І навпаки, якщо K буде знаходитись за межами множини Мандельброта, то множина Жуліа для нього буде незв'язною.

Також, якщо обрати K , розташоване дуже близько до межі множини Мандельброта, то можна побачити тісний зв'язок між формою множини Жуліа для числа K та формою межі біля цього числа в множині Мандельброта.

Практична частина

Проблеми обчислення множин

Після ознайомлення з математичною частиною, можуть виникнути деякі проблеми, наприклад ми говорили про “нескінченність”: n наближається до нескінченності, тобто Z рекурсивно обчислюється нескінченну кількість разів, і нам потрібно перевіряти чи значення Z наближається до нескінченності.

Але на сучасних комп'ютерах ми не можемо мати справу з нескінченністю. Навіть якщо б ми взяли якесь велике число замість нескінченності, ми б все одно не змогли обчислити Z_n нескінченну кількість разів. Обчисливши його дуже багато раз, результат можна було б вважати достатньо хорошим, але це займе дуже багато часу для комп'ютера.

Отже, маємо дві проблеми:

- Значення Z прямує до нескінченності.
- Це значення оцінюється нескінченно багато разів.

Вирішення проблем

Для множини Мандельброта за допомогою математичної індукції можна довести, що якщо абсолютне значення Z (відстань від $0 + 0i$) стане більшим за 2, то воно вже ніколи не стане меншим, а навпаки буде стрімко збільшуватись до нескінченності.

Це означає, що нам досить перевірити, чи “вийшов” Z за межі двійки, і якщо вийшов, то ми знаємо що він піде у нескінченність.

Для множини Жуліа існує теорема, що якщо для квадратичного поліному виду $f(z) = z^2 + K$ обчислити $R = \frac{1 + \sqrt{1 + 4|K|}}{2}$, і абсолютне значення Z стане більшим за R , то ця точка не належатиме множині.

Скільки ж разів нам треба проітерувати Z_n щоб побачити, чи виходить він за межі чи ні?

На щастя, достатньо всього декілька десятків (50-100). Це викликано обмеженнями роздільної здатності зображення, яке ми маємо отримати в результаті. Зображення не є нескінченно точними, адже вони складаються зі скінченної кількості пікселів. І після певної кількості ітерацій воно суттєво не зміниться.

Якщо ж ми збільшуємо масштаб, то ми робимо обчислення зображення із значно більшою роздільною здатністю (хоча ми обчислюємо лише частину зображення), і таким чином кількість ітерацій потрібно збільшити відповідно.

Поєднання множин з зображеннями

Інше питання полягає в тому, що якщо ми просто обчислюємо набір комплексних чисел, то як ми можемо отримати з нього двовимірне зображення?

Комплексні числа складаються з 2-х частин: дійсної(real) та уявної(imaginary). В математиці вони малюються в декартовій системі координат так, що вісь X представляє дійсну частину, а вісь Y – уявну.

Саме так ми і будемо малювати наше зображення. Якщо комплексне число c належить множині, ми малюватимемо крапку (наприклад чорного кольору) у відповідному місці, в іншому випадку будемо використовувати інший колір. В результаті отримаємо чорно-біле зображення.

Розрахунок множин

По-перше, нам потрібно встановити еквівалентність між координатами пікселів та комплексними числами.

Кожен піксель нашого зображення представляє певне число. І ми замалюємо цей піксель відповідно до того, чи належить він множині, чи ні.

Існує два підходи до цього завдання:

- Нехай перший піксель (зверху зліва) це певне число, а відстань між пікселями це певна сума. Наприклад якщо перший піксель це число $(-2 - 2i)$, а відстань $= 0,01$, тоді піксель правіше буде $(-1,99 - 2i)$, а піксель нижче $(-2 - 1,99i)$.
- Ми можемо визначити для кутових пікселів, яке комплексне число вони представляють, а потім просто інтерполювати проміжні пікселі під час обчислення.

Обидва способи схожі, адже ми повинні ввести початкове значення, а інші можна обчислити.

Я буду використовувати другий підхід, адже на мою думку він легший в розумінні та реалізації:

```
double r = 2.0; // |k| <= 2
double xMin = -r;
double yMin = -r;
double xMax = r;
double yMax = r;
```

(Для множини Мандельброта $r = 2$, а для Жуліа r обчислюється по вищеописаній формулі)

В даному прикладі піксель зверху зліва матиме значення $(xMin + yMin * i)$, а піксель знизу справа $-(xMax + yMax * i)$.

Які ж значення матимуть пікселі всередині?

Для цього потрібно порахувати різницю між сусідніми пікселями:

```
double xStep = Math.Abs(xMax - xMin) / width;  
double yStep = Math.Abs(yMax - yMin) / height;
```

І тоді число в пікселі $(i; j)$ матиме значення:

```
new Complex(real: xMin + i * xStep, imaginary: yMin + j * yStep);
```

Підрахунок функції

Нам залишилось порахувати Z_n для певного числа c .

Спочатку нам треба визначити, скільки разів ми будемо ітерувати нашу функцію, і далі в циклі знаходимо наше значення:

```
int iterCounter = 0;
Complex complex = z0;
for (int i = 0; i < iter; i++)
{
    if (complex.Magnitude > r)
        break;
    ++iterCounter;
    complex = complex * complex + z0;
}

return iterCounter;
```

(Для Жуліа замість $z0$, додаватимемо наперед визначене K)

В даному прикладі:

$z0$ – це значення числа в певному пікселі,

$r = 2$, бо як було сказано вище, нам достатньо перевірити, чи “вийшло” абсолютне значення Z за межі 2-ки.

Найцікавішою тут є умова:

```
if (complex.Magnitude > r)
```

Адже властивість *Magnitude* вираховується за формулою:

$$|c| = \sqrt{c.re^2 + c.im^2}$$

І це дуже затратно щоразу вираховувати корінь числа, тому ми піднесемо обидві сторони до квадрату і отримаємо:

```
if (complex.Real*complex.Real + complex.Imaginary*complex.Imaginary > r2)
```


Залишилось розібратись з виразом:

```
complex = complex * complex + z0;
```

Додавання 2х комплексних чисел є простим, просто окремо додати дійсну та уявну частини. Однак з множенням все трохи складніше, математично можна довести:

$$(a + bi)^2 = (a + bi)(a + bi) = a^2 + abi + abi + (bi)^2 = a^2 - b^2 + 2abi$$

Отже, можна побачити, що в результаті піднесення комплексного числа до квадрату, в результаті отримаємо комплексне число, дійсна частина якого дорівнюватиме $(a^2 - b^2)$, а уявна - $2abi$.

Так і запишемо:

```
for (int i = 0; i < iter; i++)
{
    double complexRe2 = complex.Real * complex.Real;
    double complexIm2 = complex.Imaginary * complex.Imaginary;

    if (complexRe2 + complexIm2 > r2)
        break;
    complex = new Complex(
        complexRe2 - complexIm2,
        2 * complex.Real * complex.Imaginary
    ) + z0;
    ++iterCounter;
}
```

Таким чином ми покращили продуктивність програми:

До: **Ready! Executing time = 00:01:49.1613557**

Після: **Ready! Executing time = 00:01:33.4184002**

Результати програми

Нехай якщо точка належить до множини, замалюємо її чорною, а колір фону зробимо білим.

В результат отримаємо:

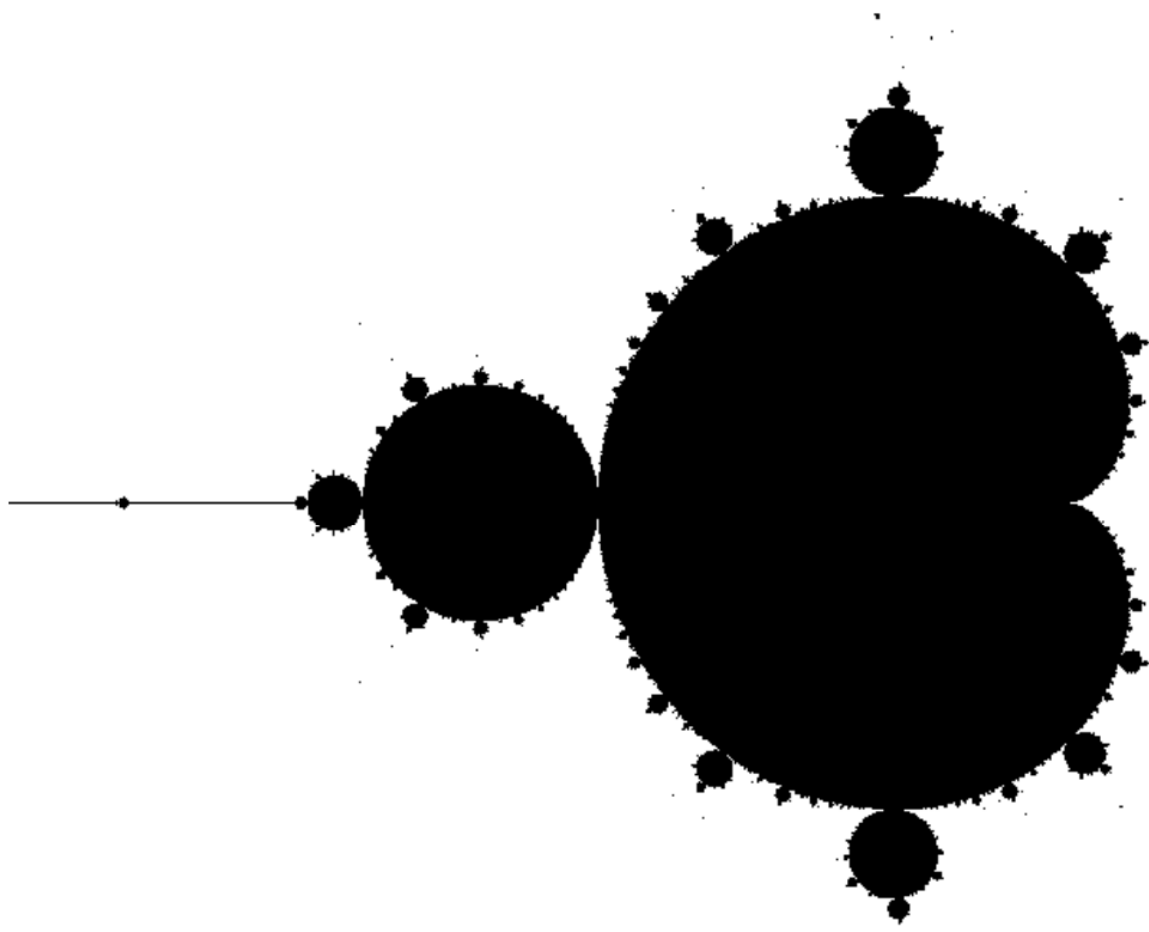


Рисунок 1. Множина Мандельброта, ч-б.

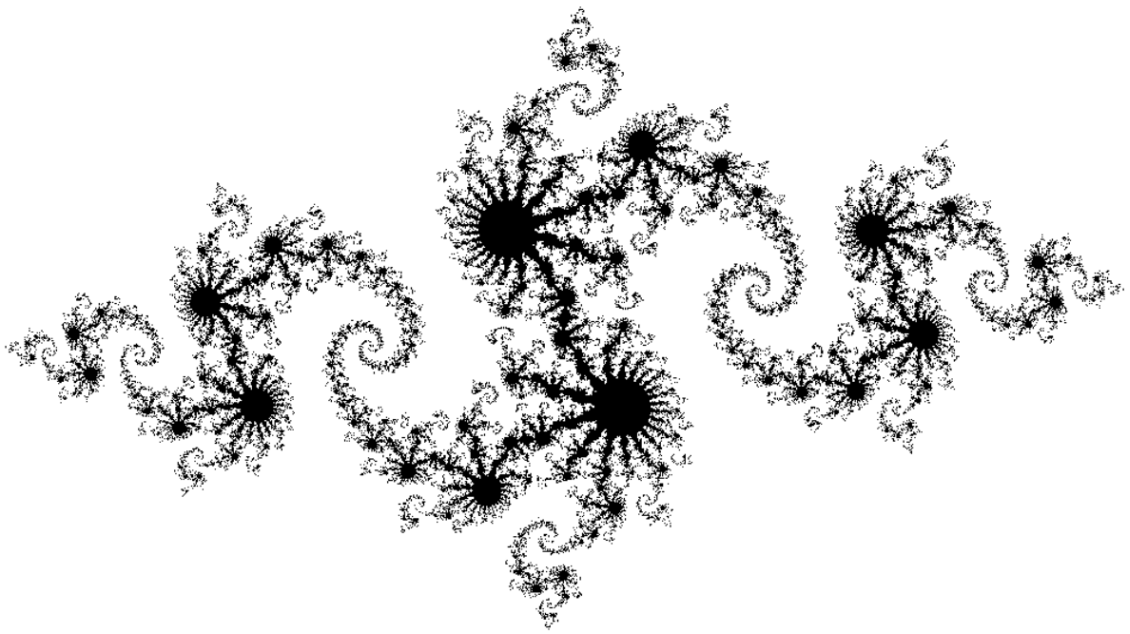


Рисунок 2. Множина Жуліа ч-б для $(-0,8+0,156i)$

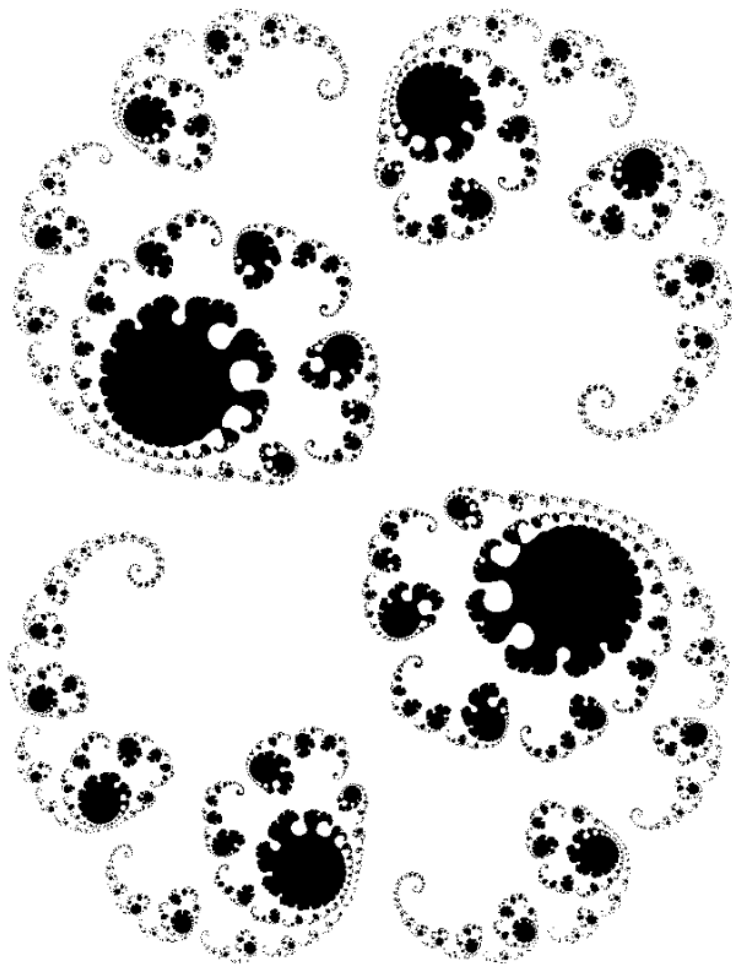


Рисунок 3. Множина Жуліа ч-б для $(0,285+0,01i)$

Багатопоточні обчислення

Єдиний недолік програми заключається в тому, що всі обчислення проходять в 1-му потоці. І це варто виправити.

Я це вирішив зробити за допомогою асинхронних функцій. Нехай кожен потік обраховує певну частину зображення, поділену горизонтально:

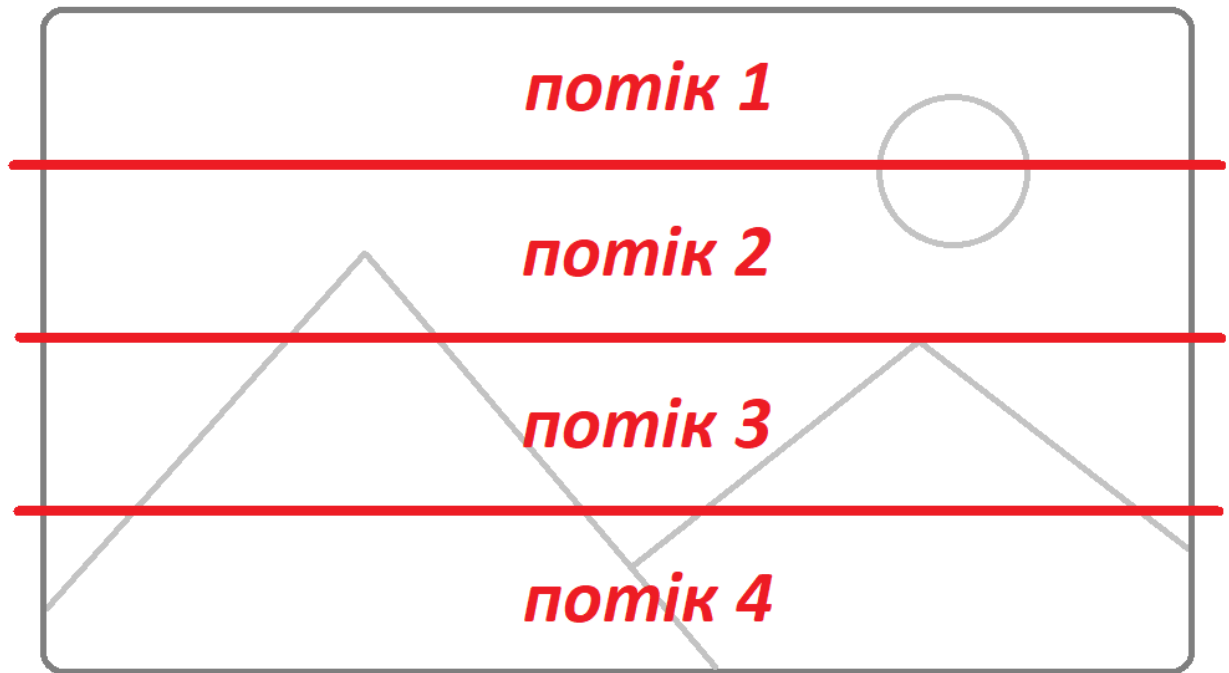


Рисунок 4. Схема розподілення потоків.

Єдина проблема заключається в тому, що результат усі потоки записують в 1-не зображення і в нас може виникнути *Exception*, або ж більшість пікселів залишаться чорними.

Для синхронізованого доступу я використаю “синтаксичний цукор” мови C#, а саме оператор *lock*.

Для цього створимо деякий приватний об’єкт:

```
object lockBmp = new object();
```


І при записі в зображення будемо блокувати його:

```
lock (lockBmp)
{
    bmp.SetPixel(x:i, y:j, GetColor(iter, maxIter));
}
```


Таким чином один потік блокує об'єкт **lockBmp** та фарбує певний піксель, в цей час інші очікують розблокування об'єкту.

Тепер при роботі програми процесор не відпочиватиме

Один потік:

Имя	Состояние	ЦП
>  OsadchukCourseWork (32 бита) (2)		15,5%

Вісім потоків:

Имя	Состояние	ЦП
>  OsadchukCourseWork (32 бита)		98,8%

Але й на продуктивність це непогано вплинуло:

1 потік:

```
Ready! Executing time = 00:04:44.9028569
Image saved to juliaSet_-0,74543_0,11301_1000.png
```

8 потоків:

```
Ready! Executing time = 00:03:52.0194912
Image saved to juliaSet_-0,74543_0,11301_1000.png
```

(Результати генерування множини Жуліа 20000x20000 при максимальній кількості ітерацій – 1000)

Забарвлення зображень

Чорно-біла картинка виглядає трохи нудно, і хочеться її трохи прифарбувати.

Є безліч способів як це зробити. Множини Мандельброта та Жуліа я вирішив розфарбовувати різними способами.

Множина Мандельброта

Будемо використовувати кількість ітерацій, необхідних для Z , щоб “вийти” за межі двійки.

В моєму випадку, я задав певну палітру на 16 кольорів у вигляді масиву:

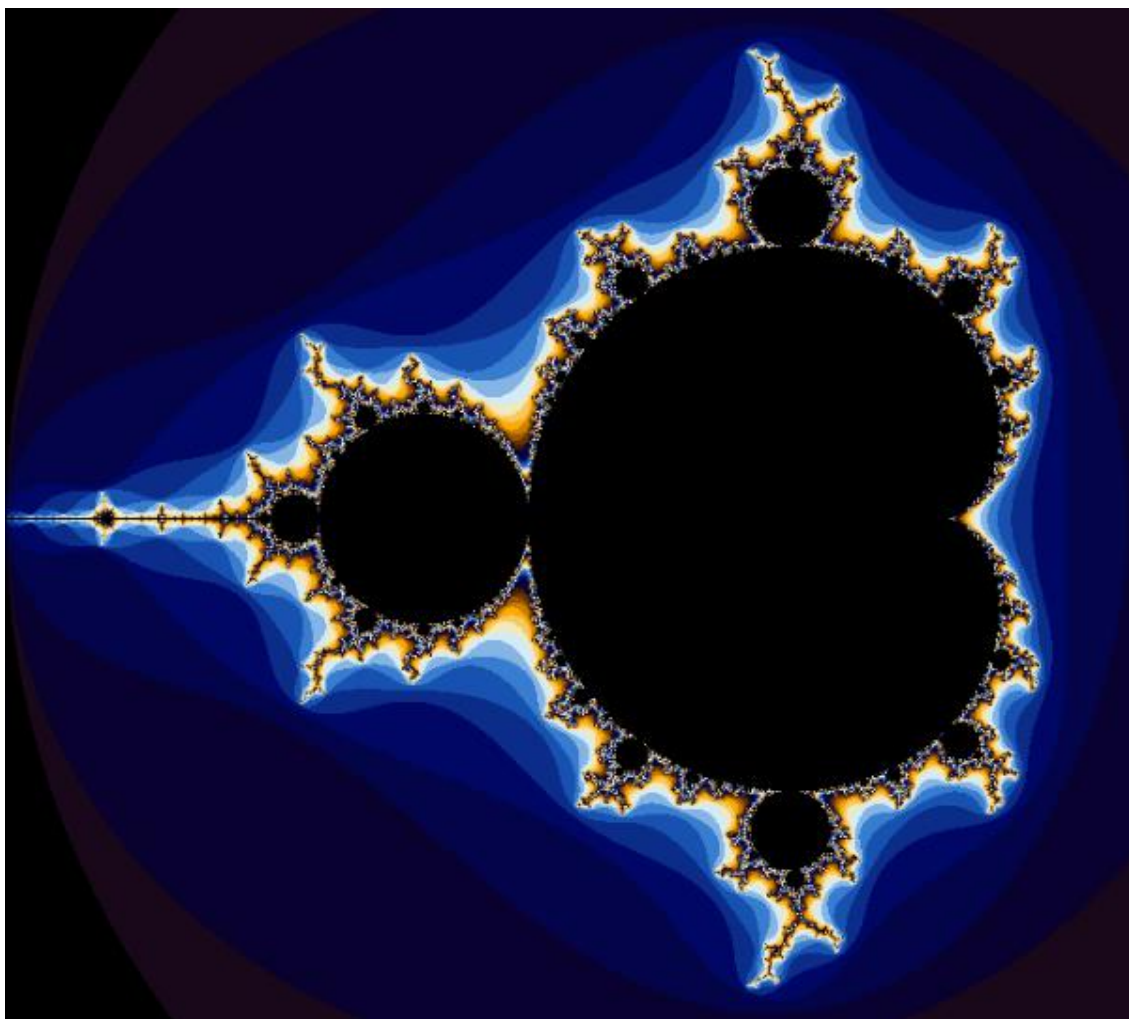
```
private readonly Color[] _colors = {  
    Color.FromArgb(red: 66, green: 30, blue: 15),  
    Color.FromArgb(red: 25, green: 7, blue: 26),  
    Color.FromArgb(red: 9, green: 1, blue: 47),  
    Color.FromArgb(red: 4, green: 4, blue: 73),  
    Color.FromArgb(red: 0, green: 7, blue: 100),  
    Color.FromArgb(red: 12, green: 44, blue: 138),  
    Color.FromArgb(red: 24, green: 82, blue: 177),  
    Color.FromArgb(red: 57, green: 125, blue: 209),  
    Color.FromArgb(red: 134, green: 181, blue: 229),  
    Color.FromArgb(red: 211, green: 236, blue: 248),  
    Color.FromArgb(red: 241, green: 233, blue: 191),  
    Color.FromArgb(red: 248, green: 201, blue: 95),  
    Color.FromArgb(red: 255, green: 170, blue: 0),  
    Color.FromArgb(red: 204, green: 128, blue: 0),  
    Color.FromArgb(red: 153, green: 87, blue: 0),  
    Color.FromArgb(red: 106, green: 52, blue: 3),  
};
```

І фарбую кожен піксель наступним чином:

```
private Color GetColor(int iter, int maxIter)  
{  
    return iter == 0 || iter == maxIter ? Color.Black : _colors[iter % 16];  
}
```

Де *iter* – це кількість ітерацій, для відповідного пікселя,
а *maxIter* – задана максимальна кількість ітерацій.

В результаті отримаємо:



Множина Жуліа

Тут колір пікселя також залежить від кількості ітерацій. Як ми знаємо, будь-який колір можна задати трьома значеннями (червоний, зелений та синій).

Тому я фарбую зображення наступним чином:

```
private Color GetColor(double iter, double maxIter, Complex k, double r)
{
    double val = iter / maxIter;

    return Color.FromArgb(
        red: Convert.ToByte(255 * val),
        green: Convert.ToByte(255 * (1 - val)),
        blue: Convert.ToByte(255 * (k.Magnitude/r > 1 ? 1 : k.Magnitude/r))
    );
}
```

Чим більше значення *iter*, тим колір червоніший, але менш зелений і навпаки. Значення синього кольору залежить від того, наскільки близько піксель розташований до центру зображення.

В результаті отримаємо такі зображення:

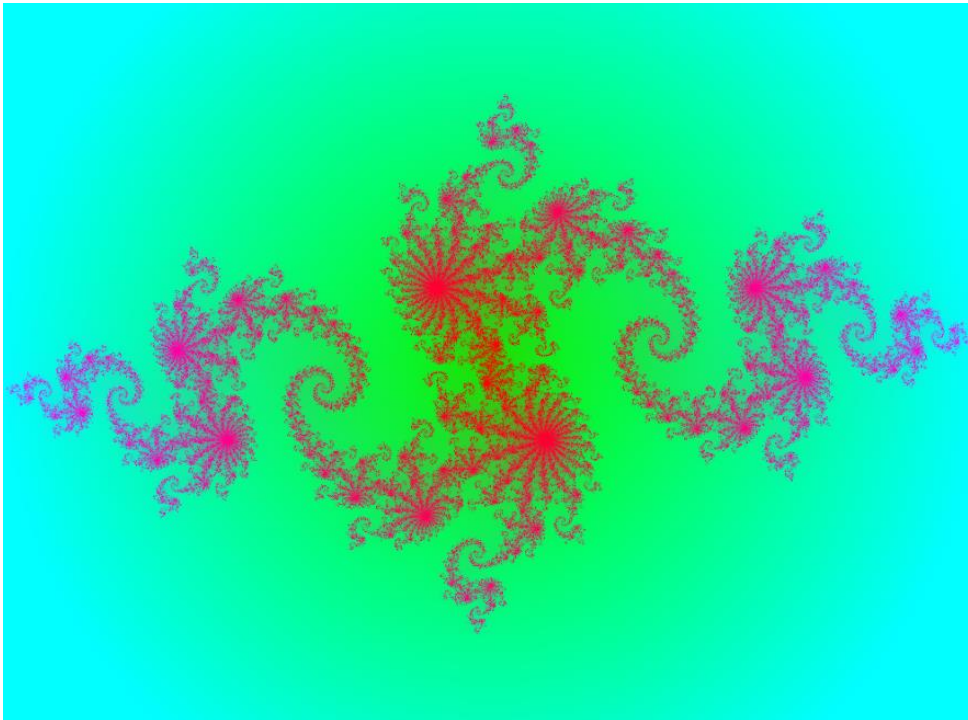


Рисунок 5. Множина Жуліа кольорова для $(-0,8+0,156i)$

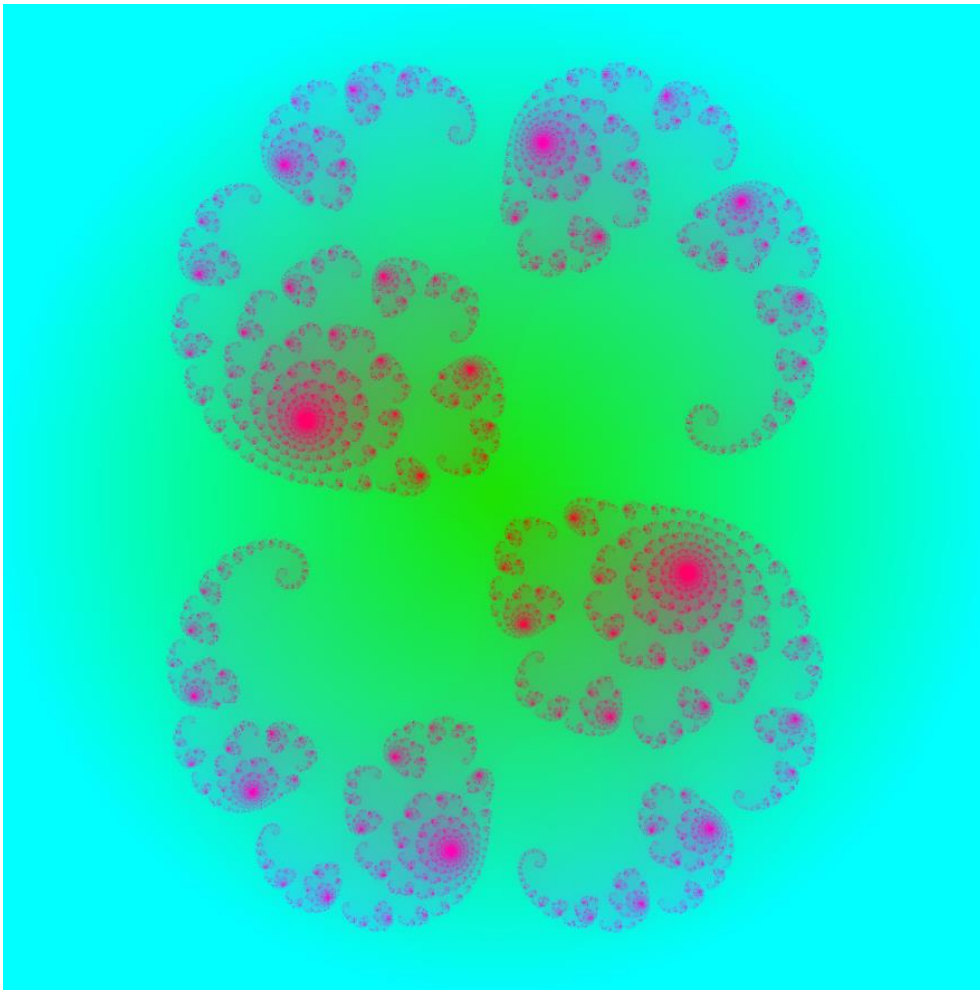


Рисунок 6. Множина Жуліа кольорова для $(0,285+0,01i)$

Кольори не лише візуально привабливі, але й надають нам багато додаткової інформації, яку не дізнатись від чорно-білих зображень. Кольори покажуть нам, де є межа. І чим яскравіший колір, тим ближче вона знаходиться.

Порівняння продуктивності та результати обчислень

Пітер Готтшлінг, автор книги *Discovering Modern C++: An Intensive Course for Scientists, Engineers, and Programmers*, запропонував свою реалізацію обчислення множини Мандельброта на C++, з якою я і буду порівнювати результати.

На малих даних з C++ мені не змагатись:

C++

```
File saved to mandelbrot.bmp  
Time = 5.661sec
```

C# (1 потік)

```
Ready! Executing time = 00:00:10.1147559  
Image saved to mandelbrotSet_100.png
```

(Результати генерування множини Мандельброта 5000x5000 при максимальній кількості ітерацій – 100)

В програмі, запропонованій Готтшлінгом, є один недолік – вона працює в 1-му потоці. І вже на великих даних це відчувається:

C++

```
File saved to mandelbrot.bmp  
Time = 444.838sec
```

C#

```
Ready! Executing time = 00:03:48.3889421  
Image saved to mandelbrotSet_1000.png
```

(Результати генерування множини Мандельброта 20000x20000 при максимальній кількості ітерацій – 1000)

Висновки

Мету роботи було досягнуто: ми розглянули математичне визначення множин Мандельброта та Жуліа, а також зв'язок між ними. Розглянули проблеми, які виникають при обчисленні множин та їх вирішення. Написали застосунок, що генерує множини та виводить їх у графічному вигляді. Також додали фарбів до зображень, розглянули способи покращення програми та порівняли її з програмою, запропонованою Пітером Готтшлінгом.

Для множини Жуліа ми розглянули множини для чисел $(-0,8 + 0,156i)$ та $(0,285 + 0,01i)$. Хоча існує безліч інших множин, які генерують не менш цікаві картинки.

Список джерел

1. *Discovering Modern C++: An Intensive Course for Scientists, Engineers, and Programmers* – Пітер Готтшлінг, 2015 рік.
2. *Programming in C#. Exam Ref 70-483* – Боутер де Корт, 2013 рік.
3. *Fractals and Chaos: The Mandelbrot Set and Beyond* – Бенуа Мандельброт, 2004 рік.
4. *Dynamics in one complex variable* – Джон Мілнор, 1990 рік.
5. <https://docs.microsoft.com/en-us/dotnet/api/system.drawing> - базові функціональні можливості графічного інтерфейсу GDI+.
6. <https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task> - асинхронні операції в C#.
7. <https://docs.microsoft.com/en-us/dotnet/api/system.numerics.complex> - робота з комплексними числами в C#.
8. <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/lock-statement> - оператор lock.

Додатки

Головний метод для генерації Множини Мандельброта

```
private Bitmap PlotMandelbrotSet(
    int width, int height, int maxIter, int threadCounter)
{
    double r = 2.0; // |k| <= 2
    double xMin = -r;
    double yMin = -r;
    double xMax = r;
    double yMax = r;

    double xStep = Math.Abs(xMax - xMin) / width;
    double yStep = Math.Abs(yMax - yMin) / height;

    Bitmap bmp = new Bitmap(width, height);
    object lockBmp = new object();

    var allTasks = new Task[threadCounter];
    for (int thread = 0; thread < threadCounter; thread++)
    {
        int wFrom = thread * width / threadCounter;
        int wTo = (thread + 1) * width / threadCounter;

        allTasks[thread] = Task.Run(() =>
        {
            for (int i = wFrom; i < wTo; i++)
            {
                double x = xMin + i * xStep;
                for (int j = 0; j < height; j++)
                {
                    double y = yMin + j * yStep;
                    Complex k = new Complex(x, y);
                    int iter = CountIterations(k, maxIter, r);

                    lock (lockBmp)
                    {
                        bmp.SetPixel(i, j, GetColor(iter, maxIter));
                    }
                }
            }
        });
    }

    Task.WaitAll(allTasks);

    return bmp;
}
```

Підрахунок кількості ітерацій в Множині Мандельброта

```
private int CountIterations(Complex z0, int iter, double r)
{
    int iterCounter = 0;
    Complex complex = z0;
    double r2 = r * r;
    for (int i = 0; i < iter; i++)
    {
        double complexRe2 = complex.Real * complex.Real;
        double complexIm2 = complex.Imaginary * complex.Imaginary;

        if (complexRe2 + complexIm2 > r2)
            break;
        complex = new Complex(
            complexRe2 - complexIm2,
            2 * complex.Real * complex.Imaginary
        ) + z0;
        ++iterCounter;
    }

    return iterCounter;
}
```

Отримання кольору для пікселя в Множині Мандельброта

```
private readonly Color[] _colors =
{
    Color.FromArgb(66, 30, 15),
    Color.FromArgb(25, 7, 26),
    Color.FromArgb(9, 1, 47),
    Color.FromArgb(4, 4, 73),
    Color.FromArgb(0, 7, 100),
    Color.FromArgb(12, 44, 138),
    Color.FromArgb(24, 82, 177),
    Color.FromArgb(57, 125, 209),
    Color.FromArgb(134, 181, 229),
    Color.FromArgb(211, 236, 248),
    Color.FromArgb(241, 233, 191),
    Color.FromArgb(248, 201, 95),
    Color.FromArgb(255, 170, 0),
    Color.FromArgb(204, 128, 0),
    Color.FromArgb(153, 87, 0),
    Color.FromArgb(106, 52, 3),
};

private Color GetColor(int iter, int maxIter)
{
    return iter==0 || iter==maxIter ? Color.Black : _colors[iter % 16];
}
```

Головний метод для генерації Множини Жуліа

```
private Bitmap PlotJuliaSet(
    Complex c, int width, int height, int maxIter, int threadCounter)
{
    double r = CalculateR(c);
    double xMin = -r;
    double yMin = -r;
    double xMax = r;
    double yMax = r;

    double xStep = Math.Abs(xMax - xMin) / width;
    double yStep = Math.Abs(yMax - yMin) / height;

    Bitmap bmp = new Bitmap(width, height);
    object lockBmp = new object();

    var allTasks = new Task[threadCounter];
    for (int thread = 0; thread < threadCounter; thread++)
    {
        int wFrom = thread * width / threadCounter;
        int wTo = (thread + 1) * width / threadCounter;

        allTasks[thread] = Task.Run(() =>
        {
            for (int i = wFrom; i < wTo; i++)
            {
                double x = xMin + i * xStep;
                for (int j = 0; j < height; j++)
                {
                    double y = yMin + j * yStep;
                    Complex k = new Complex(x, y);
                    int iter = CountIterations(k, c, maxIter, r);

                    lock (lockBmp)
                    {
                        bmp.SetPixel(i, j, GetColor(iter, maxIter, k, r));
                    }
                }
            }
        });
    }

    Task.WaitAll(allTasks);

    return bmp;
}
```

Обчислення R для Множини Жуліа

```
private static double CalculateR(Complex c)
{
    return (1 + Math.Sqrt(1 + 4 * c.Magnitude)) / 2;
}
```

Підрахунок кількості ітерацій в Множині Жуліа

```
private int CountIterations(Complex z0, Complex c, int iter, double r)
{
    int iterCounter = 0;
    Complex complex = z0;
    double r2 = r * r;
    for (int i = 0; i < iter; i++)
    {
        double complexRe2 = complex.Real * complex.Real;
        double complexIm2 = complex.Imaginary * complex.Imaginary;

        if (complexRe2 + complexIm2 > r2)
            break;
        complex = new Complex(
            complexRe2 - complexIm2,
            2 * complex.Real * complex.Imaginary
        ) + c;
        ++iterCounter;
    }

    return iterCounter;
}
```

Отримання кольору для пікселя в Множині Жуліа

```
private Color GetColor(double iter, double maxIter, Complex k, double r)
{
    double val = iter / maxIter;

    return Color.FromArgb(
        Convert.ToByte(255 * val),
        Convert.ToByte(255 * (1 - val)),
        Convert.ToByte(255 * (k.Magnitude / r > 1 ? 1 : k.Magnitude / r))
    );
}
```