Зад. 20 (Изобразяване на фрактал, v6)

Увод

Фракталите (фракталната геометрия) са (е) нов клон в Математиката, който често бива определян като граничещ с изкуството. Причината за това са красивите изображения свързани с фракталите, които биват използвани като background за компютърният екран или поздравителна картичка например. Фракталите са припознавани от повечето хора, точно като такива изображения.

Фракталната геометрия се старае да излезе от традиционното схващане за математиката, определяща я като купчина скучни и сложни формули. Граничейки с изкуството, фракталната геометрия някак си показва, че уравненията са повече от купища числа отговарящи на определени условия. Това, което прави фракталите още по интересни е че те са най-добрите (точните) съществуващи математически описания на различни природни форми, като: брегови ивици, планински върхове или дори части от различни живи организми дори.

Целта на тази задача е да Ви запознае със изображенията генерирани от фрактали (и по-специално фракталите определени в комплексни числа). Макар и доста красиви, те изискват значителен изчислителен ресурс, което е и причината да разглеждаме подобна задача. Съвременните компютри разполагат с повече от един централен процесор, което пък ни дава възможност да изобразяваме фрактал с доста висока разделителна способност в почти реално време. Въоръжени с „оръдията“ :) на паралелното (конкурентното) програмиране, можем да си позволим да използваме този ресурс сравнително бързо и лесно.

Преди точните изисквания на задачата, ще се постарая да дам кратко /и надявам се не много скучно :)/ описание на конкретен тип фрактали, а именно тези определени в комплексни числа (определени в комплексната равнина).

Фрактали определени в комплексни числа (complex number fraktals).

Кратко представяне на комплексните числа.

Комплексното число се състои от реална и имагинерна част. Комплексните числа могат да се представят като точки в равнина, снабдена с декартова координатна система, която още се нарича комплексна равнина. Ако е дадено комплексното число:

$$(1)\ Z=a+ib,\ i=\sqrt{-1}$$

то, координатите на точката са (a, b), като

a – координатата на точката по абсцисата (хоризонталната ос, още наричана реална),
b – координатата на точката по ординатата (вертикалната ос, още наричана имагинерна).

Двамата най-известни изследователи в областта на този тип фрактали (всъщност техните откриватели) са Gaston Maurice Julia и Benoit Mandelbrot.

Роден в края на 19ти век Gaston Maurice Julia става известен с изследване на поведението на рационални функции в комплексни числа. По-късно, работата му бива забравена, за да бъде преоткрита през 1970 година от служителят на IBM Corp., Benoit Mandelbrot, който вдъхновен от работата на Julia и използвайки възможностите на тогавашната компютърна графика е показал първите изображения на фрактали, които познаваме днес.

Множество на Манделброт (Mandelbrot set)

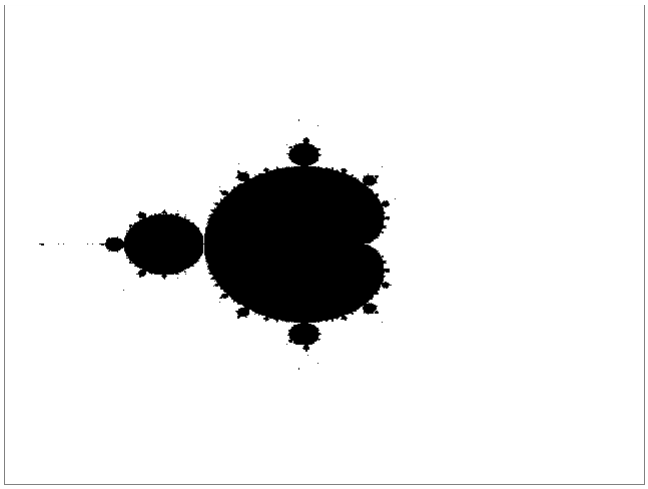
Множеството на Манделброт се определя от точки в комплексната равнина. За да построим (определим) това множество трябва да използваме алгоритъм базиран на следната „рекурсивна“ формула:

$$(2) Z_n = Z_{n-1} + C$$

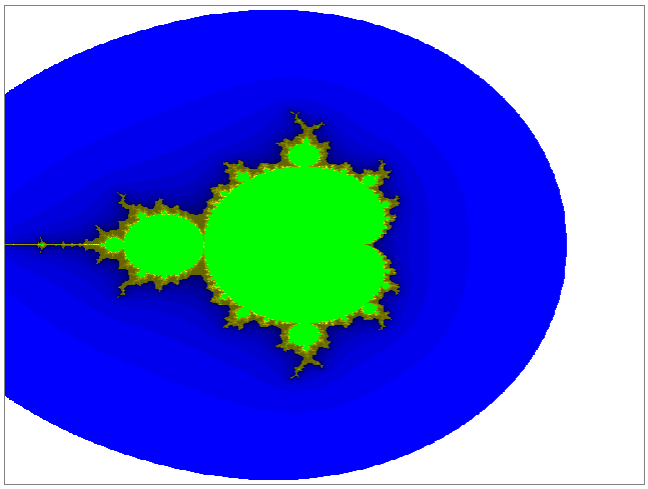
Формулата разделя точките в комплексната равнина на две категории:

(о) точки, принадлежащи на множеството на Манделброт,
(о) точки, не-принадлежащи (извън) множеството на Манделброт.

Следващото изображение показва област от комплексната равнина. Точките принадлежащи на множеството на Манделброт са оцветени в черно:



Да се генерират изображения само с помощта на черен и бял цвят е малко скучно. Възможно е да асоцииране цвят (различен от горните два) с точките извън множеството на Манделброт. Например техният цвят може да зависи от това колко близо са до множеството. Друг критерии е броят итерации, които са необходими, за да се определи дали дадена точка принадлежи или не на множеството:



Частта от комплексната равнина, показана на изображенията е определена от: $a \in [-2.0, 2.0], b \in [-2.0, 2.0]$.

Как все пак създаваме (генерираме) множеството на Манделброт?

За да създадем множеството на Манделброт, избираме точка от комплексната равнина – С. Комплексното число, отговарящо на тази точка има вида:

$$(3) C = a + ib$$

Използвайки (2) имаме:

$(4)\, Z_1 = Z_0^2 + C$ *, като за* $Z_0 = (0,0)$ *получаваме* $Z_1 = C$

$(5)\, Z_2 = Z_1^2 + C = C^2 + C$

$(6)\, Z_3 = Z_2^2 + C = \ldots$

Този изчислителен процес, можем да си представим, като движение („миграция“) на първоначалната точка C в комплексната равнина. Какво се случва с тази точка в процеса на изчисление ($Z_4, Z_5, \ldots, Z_n, Z_{n+1}, \ldots$)? Дали точката ще остане близо до началото или ще се отдалечи от него (увеличавайки разстоянието безгранично)?

В първият случай, казваме че точката принадлежи на множеството на Манделброт (съответно е оцветена в черен цвят в първото изображение).

Във вторият случай казваме че точката клони към безкрайност и съответно не принадлежи на множеството на Манделброт. Можем да си присвоим цвят, в зависимост от скоростта с която точката „бяга“ от началото (т.е. колко „бързо“ клони към безкрайност).

Нашият алгоритъм за генериране на множеството на Манделброт, като че ли се оформи :). Нека си представим, че всяка една точка от равнината, е „привлечена“ както от безкрайността, така и от множеството на Манделброт. Приемайки това, можем да класифицираме точките в равнината според следните три условия:

(о) точките, които са далеч от (извън) множеството на Манделброт се придвижват сравнително „бързо“ (клонят „бързо“) към безкрайността;

(о) точките, които са близко до (но, все още извън) множеството на Манделброт се придвижват сравнително „бавно“ (клонят „бавно“) към безкрайността;

(о) точките, които са от множеството на Манделброт, никога не клонят към безкрайността;

Условие на задачата.

Нека разгледаме формулата:

$(7)\, F(Z) = Z^2 * e^{Z^2} + C$

Вашата задача е да напишете програма за визуализиране на множеството на Манделброт, определено от формула (7). Програмата трябва да използва паралелни процеси (нишки) за да разпредели работата по търсенето на точките от множеството на Манделброт на повече от един процесор. Програмата трябва да осигурява и генерирането на изображение (например **.png**), показващо така намереното множество. Изискванията към програмата са следните:

(о) Програмата да позволява (разбира от) команден параметър, който задава големината на генерираното изображение, като широчина и височина в брой пиксели. Той има вида: **„-s 640x480“** (или **„-size“**); При не-въведен от потребителя команден параметър, за големина на изображението, програмата подразбира - широчина (width) **640px** и височина (height) **480px**;

(о) Команден параметър, който да задава частта от комплексната равнина, в която ще търсим визуализация на множеството на Манделброт: **„-r -2.0:2.0:-1.0:1.0“** (или **„-rect“**). Стойността на параметъра се интерпретира както следва: $a \in [-2.0, 2.0], b \in [-1.0, 1.0]$. При не въведен от потребителя параметър програмата приема че е зададена стойност по подразбиране: **„-2.0:2.0:-2.0:2.0“**.

(о) Друг команден параметър, който задава максималния брой нишки (паралелни процеси) на които разделяме работата по генерирането на изображението: **„-t 3“** (или **„-tasks“**); При не-въведен от потребителя команден параметър за брой нишки – програмата подразбира **1** нишка;

(о) Команден параметър указващ името на генерираното изображение: **„-o zad20.png“** (или **„-output“**). Съответно програмата записва генерираното изображение в този файл. Ако този параметър

е изпуснат (не е зададен от потребителя), се избира име по подразбиране: **„zad20.png“**;

(о) Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за завършване на всички изчисления по визуализирането на точките от множеството на Манделброт (пресмятане на множеството на Манделброт);

Примери за подходящи съобщения:

„**Thread-<num> started.**“,

„**Thread-<num> stopped.**“,

„**Thread-<num> execution time was (millis): <num>**“,

„**Threads used in current run: <num>**“,

„**Total execution time for current run (millis): <num>**“ и т.н.;

(о) Да се осигури възможност за **„quiet“** режим на работа на програмата, при който се извежда само времето през което програмата е работила (без „подходящите“ съобщения от предходната точка). Параметърът за тази цел нека да е **„-q“** (или **„-quiet“)**; Тихият режим не отменя записването на изображението във изходният файл;

ЗАБЕЛЕЖКА:

(о) При желание за направата на подходящ графичен потребителски интерфейс (**GUI**) с помощта на класовете от пакета **javax.swing** задачата може да се изпълни от **двама души**; Разработването на графичен интерфейс не отменя изискването Вашата програма да поддържа изредените командни параметри. В този случай към функцията на параметъра параметъра **„-q“** се добавя изискването **да не пуска** графичният интерфейс. Причината за това е, че Вашата програма трябва да позволява отдалечено тестване, а то ще се извършва в **terminal**.

Уточнения (hints) към задачата:

(о) В условието на задачата се говори за разделянето на работата на две или повече нишки. Работата върху съответната задача, в случаят в който е зададен **„-t 1“** (т.е. цялата задача се решава от една нишка) ще служи за еталон, по който да измерваме евентуално ускорение (т.е. това е **T1**). В кода реализиращ решението на задачата трябва да се предвиди и тази възможност – задачата да бъде решавана от единствена нишка (процес); Пускайки програмата да работи върху задачата с помощта на единствена нишка, ще считаме че използваме серийното решение на задачата; Измервайки времето за работа на програмата при използването на **„p“** нишки – намираме **Tp** и съответно можем да изчислим **Sp**. Представените на защитата данни за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се изчисли и покаже **Ep**.

Като обобщение - данните събрани при тестването на програмата Ви, трябва да отразяват **Tp**, **Sp** и **Ep**. Желателно е освен табличен вид, да добавите и графичен вид на **Tp**, **Sp**, **Ep**, в три отделни графики.

(о) Не се очаква от Вас да реализирате библиотека, осигуряваща математически операции със комплексни числа. Подходяща за тази цел е например **Apache Commons Math3 (http://commons.apache.org/proper/commons-math/userguide/complex.html).** При изчисленията, свързани с генерирането на множеството на Манделброт (задачите за фрактали), определено ще имате нужда от нея.

(о) Не се очаква от вас да реализирате библиотека, осигуряваща математически операции със голяма точност. Подходяща за тази цел библиотека е например **Apfloat (http://www.apfloat.org)**. Ако програмата Ви има нужда от работа с големи числа, можете да използвате нея.

Разбира се **BigInteger** и **BigDecimal** класовете в **java.math** са също възможно решение – въпрос на избор и вкус.

Преди да направите избора, проверете дали избраната библиотека не използва също нишки – това може да доведе до неочаквани и доста интересни резултати; ;)

(о) Не се очаква от Вас да търсите (пишете) библиотека за генериране на **.png** изображения. Java има прекрасна за нашите цели вградена библиотека, която може да се ползва. Примерен проект, показващ генерирането на чернобялата и цветната версия на фрактала на Манделброт /множество на Манделброт за формула (2)/, цитирани в задачите за фрактали, е качена на **http://rmi.yaht.net/docs/example.projects/ - pfg.zip**.

(о) Командните аргументи (параметри) на терминална Java програма, получаваме във масива **String args[]** на **main()** метода, намиращ се в стартовият клас. За „разбирането“ им (анализирането им) може да ползвате и външни библиотеки писани специално за тази цел . Един добър пример за това е: **Apache Commons CLI (http://commons.apache.org/cli/)**.