**Софийски универитет „Св. Климент Охридски“**

*Факултет по математика и информатика*

*Специалност: “Информационни системи”*

*Курс: 3, Група: 1*

*Дисциплина: “Фрактали”*

**Курсов Проект**

Тема: Снежинка на Кох

***Изготвил:***

Лилия Иванова Божанина (ФН: 71873)

***Преподавател:***

Доц. Д-р Милко Такев

София

Летен семестър 2020/2021

**1. Niels Fabian Helge von Koch**

Niels Fabian Helge von Koch (25 January 1870 – 11 March 1924) е шведски математик. Той е автор на един от най – известните фрактали на съвременния сват, а именно едноименния фрактал Снежинка на Кох. Това е една от първите фрактални криви, която е била подробно описана.

Роден в семейство на благородници. Неговият дядо, Nils Samuel von Koch (1801 – 1881) е бил министър на правосъдието на Швеция. Баща му, Richer Vogt von Koch (1838 - 1913) е бил подполковник в Кралската конна стража. Бъдещият гений в математиката получава познания в Стокхолмския университет (приемат го през 1887, където учи под ръководството на [Gösta Mittag-Leffler](https://en.wikipedia.org/wiki/Gösta_Mittag-Leffler)), освен в новооткрития университет, Кох учи и в университета Упсала (Uppsala University), през 1888, където е получил и своята бакалавърска степен (поради факта, че Стокхолмския университет все още няма легитимни дипломи). През 1892 год. Кох получава своята докторска степен в Упсала. Назначен за преподавател (професор по математика) през 1905 в Кралския Институт по Технологии в Стокхолм. След това неговата кариера се развива и през 1911 става професор по чиста математика в Стокхолмския Университет. Автор е на няколко труда в сверата на Теория на числата. Едно от известните му постижения е през 1901, когато доказва, че хипотезата на Рейнман е по – стабилна форма на теоремата за простите числа.

В следващия доклад тема на нашите разсъждения ще бъде един от най – известните трудове на Кох, а именно едноименната крива – кривата на Кох, описана през 1904 година в "На непрекъсната крива без допирателни елементи, изработени от елементарна геометрия" на английски  “on a continuous curve without tangents constructible from elementary geometry” (*original French title: "Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire"*)

Като изтъкнат математик, Кох е бил канен като оратор на различни форуми във Франция, Англия и Швеция.

**2. Снежинка на Кох (Koch’s Snowflake)**

Снежинката на Кох (още позната като крива на Кох, звезда на Кох или остров на Кох) е математическа крива и една от най – ранните фрактални криви, която е описана. Базирана и описана през 1904 във вече споменатия труд "Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire", от Хелдж вон Кох.

## Изграждане

Един от начините за изграждане на снежинката е следният: започваме с равностранен триъгълник, след това рекурсивно променяме всяка страна (всеки линеен сегмент) по следния начин:

* 1) Разделяме страната на три части с равна дължина
* 2) Чертаем равностранен триъгълник , който има средната част от разделението от първа стъпка за своя основа (т.е. е насочен навън)
* 3) Премахваме страната която е основа на триъгълника от стъпка 2

След една итерация от този процес получената фигура е Хексаграм. Снежинката на Кох е границата, която може да се стигне като следваме горните стъпки. Оригинално Кох конструира кривата само по едната страна от оригиналния триъгълник. Можем да заключим следното: Снежинката на Кох представлява три криви на Кох, свързани заедно.

## Свойства

**Периметър**

Интересно свойство на снежинката на Кох е периметърът. При всяка итерация ние делим всяка страна в две точки като добавяме нови страни. Тъй като това го правим безкраен брой пъти (добавянето на нови страни (т.е. добавянето на равностранни триъгълници със съответната дължина)), периметърът ще клони към безкрайност.

След всяка итерация броят на страните се увеличава по 4, т.е. след n итерации ще имаме следното :

Ако на началния тригълник дължината на страната е *s,* тогава на итерация *n* дължината ще е :

Периметърът на снежинката след *n* итерации е:

Кривата на Кох има безкрайна дължина, защото общата дължина на кривата се увеличава с една трета с всяка една итерация. Освен това всяка итерация създава четири пъти повече страни (линейни сегменти) от колкото в предишната, като са с дължина всяка от тях една трета от дължината на предишния сегмент. По този начин дължината на кривата след *n* итерации ще бъде (4/3)n умножено по дължината на страната на началния триъгълник. Като пуснем *n* да клони към безкрайност цялата сума клони към безкрайност.

#### **Граница на периметъра**

Казаното по рано, че при брой на итерациите клонящи към безкрайност границата на периметъра е:

### 

### **Лице на Снежинката**

На всяка итерация се добавя нов триъгълник към всяка страна от предишната итерация, като по този начин броя на новите триъгълници добавени на итерация *n* са:

Лицето на всеки от новите триъгълници добавено в итерация е една девета от лицето на всеки триъгълник добавен в предишна итерация, от където следва, че лицето на всеки триъгълник който добавяме в итерация *n* е:

където а0 е лицето на началния триъгълник. Общото лице добавено на итерация *n* е:

Крайното лице на снежинката след *n* итерации:

Преработването на геометричната сума дава:

#### **Граница на лице**

Лицето представено гранично е:

като |4/9| < 1.

/

Виждаме, че лицето на снежинката е 8/5 от лицето на оригиналния триъгълник.

Ако изразим лицето чрез дължината на страната на началния триъгълник (означим го със *s*), получаваме:

Нека имаме равностранен триъгълник:

Ако приложим рекурсивно няколко пъти правилата за построяване, то на различните итерации ще имаме:

Нека вземем началната страна с дължина *s.* Делим я на три равни части, като добавяме равностранен триъгълник със страни s/3.

Дължината на новата част е 4\*s/3 = 4/3 \* s.

Ако оригиналният периметър е P0, след едно добавяне ще стане

P1 = 4/3 \* P0 , P2  = 4/3 \* P1  … , ако го направим безкраен брой пъти ще получим безкрайна дължина .

Тук идва и интересната част, а именно наличието на безкраен периметър, но крайно лице. Крайността на лицето се доказва най – лесно, като начертаем фигура около „снежинката“ , то тази „снежинка“ никога няма да се разшири достатъчно, че да премине отвъд тази фигура. Без формално доказателство можем да скицираме така:

Построяваме прави и забелязваме, че колкото

и да добавяме нови триъгълници никога няма да

преминем получилия хексагон.

Тази фигура има крайно лице => крайно лице за снежинката. Ако допуснем противното ще получим противоречие с крайното лице на хексагона (няма как крайно лице да покрие безкрайно лице)

Вече разбрахме, че фигурата има крайно лице, сега ще го изчислим:

Първо лицето на равностранен триъгълник със страни *s* е

За намирането на лицето ще построим отново фигурата стъпка по стъпка, за да забележим няколко модела.

На всяка стъпка ще следим две неща: брой страни и текущото лице. Започваме с 3 страни и лице

На следваща стъпка имаме:

Сега делим на по 3 , като забелязваме, че едната страна се преобразува до 4, това го правим за всички страни => сега имаме 4 \* 3 страни, а лицето е лицето на началния триъгълник + лицата на новите, т.е:

Страни : 3 -> 12 (новите с дължина s/3)

Лице:

След още една итерация имаме: големина на новите страни s/9 брой триъгълници 12

Страни: 3 -> 12 -> 48

Лице:

Вече забелязваме модела и ако направим още една итерация лицето ще е, ако добавяме безкраен брой членове ще получим лицето на истинската снежинка. Трикът за намирането на точното лице е разписването на сумата, за да може да получим крайно число. Сега преобразуваме сумата от лицата.

Сега ще умножа и разделя на 4:

Сега ще използвам , за да получа

Оцветеното е безкрайна геометрична прогресия която ще преобразуваме до -> = 3 \* 4/5

=> лицето е:

**3. Варианти на Снежинката на Кох.**

Има множество вариации, по отношение на изграждането и вида в който ще се получи конкретната снежинка. В следващата таблица са представени част от разновидностите на кривата.

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Илюстрация** | **Методика на конструкция** |
| 1D, 85° ъгъл | [https://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/Koch_Curve_85degrees.png/150px-Koch_Curve_85degrees.png](https://en.wikipedia.org/wiki/File:Koch_Curve_85degrees.png)  Cesàro fractal | Още известно като Фрактал на Цезаро, това е вариация на кривата при която се използва ъгъл между 60° и 90° (на предоставената графика е 85°) |
| 1D, 90° ъгъл | [https://upload.wikimedia.org/wikipedia/commons/thumb/3/3b/Quadratic_Koch_2.svg/150px-Quadratic_Koch_2.svg.png](https://en.wikipedia.org/wiki/File:Quadratic_Koch_2.svg)  Quadratic type 1 curve | [https://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/Quadratic_Koch_curve_type1_iterations.png/450px-Quadratic_Koch_curve_type1_iterations.png](https://en.wikipedia.org/wiki/File:Quadratic_Koch_curve_type1_iterations.png)  Първите две преобразувания |
| 1D, 90° ъгъл | [https://upload.wikimedia.org/wikipedia/commons/thumb/5/55/Quadratic_Koch.svg/150px-Quadratic_Koch.svg.png](https://en.wikipedia.org/wiki/File:Quadratic_Koch.svg)  Quadratic type 2 curve | [https://upload.wikimedia.org/wikipedia/commons/thumb/1/1e/Quadratic_Koch_curve_type2_iterations.png/450px-Quadratic_Koch_curve_type2_iterations.png](https://en.wikipedia.org/wiki/File:Quadratic_Koch_curve_type2_iterations.png)  Първите две преобразувания. Фракталното измерение е 3/2 и е точно на половината от 1 до 2 измерение. |
| 1D, ln 3/ln √5 | [https://upload.wikimedia.org/wikipedia/commons/thumb/2/28/Karperienflake.gif/150px-Karperienflake.gif](https://en.wikipedia.org/wiki/File:Karperienflake.gif)  Quadratic flake | [https://upload.wikimedia.org/wikipedia/commons/thumb/0/04/Karperienflakeani2.gif/450px-Karperienflakeani2.gif](https://en.wikipedia.org/wiki/File:Karperienflakeani2.gif)  Първите две преобразувания. Фрактално измерение ln 3/ln √5 = 1.37. |
| 1D, ln 3.33/ln √5 | [https://upload.wikimedia.org/wikipedia/commons/thumb/c/c5/Quadriccross.gif/150px-Quadriccross.gif](https://en.wikipedia.org/wiki/File:Quadriccross.gif)  Quadratic Cross | Фрактално измерение ln ln 3.33/ln √5 = 1.49. |
| 2D, триъгълници | [https://upload.wikimedia.org/wikipedia/commons/thumb/6/63/Koch_surface_3.png/150px-Koch_surface_3.png](https://en.wikipedia.org/wiki/File:Koch_surface_3.png)  von Koch surface | [https://upload.wikimedia.org/wikipedia/commons/thumb/a/a7/Koch_surface_0_through_3.png/450px-Koch_surface_0_through_3.png](https://en.wikipedia.org/wiki/File:Koch_surface_0_through_3.png)  Първите 3 итерации на нормалното развитие на Кривата на Кох в 2 измерения. |
| 2D, 90° ъгъл | [https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Quadratic_Koch_3D_%28type1_stage2%29.png/150px-Quadratic_Koch_3D_%28type1_stage2%29.png](https://en.wikipedia.org/wiki/File:Quadratic_Koch_3D_(type1_stage2).png)  Quadratic type 1 surface | Разширение на квадратична тип 1 крива. |
| 3D | [https://upload.wikimedia.org/wikipedia/commons/thumb/3/39/Koch_Curve_in_Three_Dimensions_%28%22Delta%22_fractal%29.jpg/150px-Koch_Curve_in_Three_Dimensions_%28%22Delta%22_fractal%29.jpg](https://en.wikipedia.org/wiki/File:Koch_Curve_in_Three_Dimensions_(%22Delta%22_fractal).jpg)  Koch curve in 3D | Триизмерен фрактал конструиран от Кривата на Кох. Тази форма може да се приеме като триизмерно разширение на кривата по същия начин, в който се отнасят Пирамидата на Серпински към Триъгълника на Серпински. Тази версия на Кривата използва 85° ъгли. |

**4. Подход за програмна обработка**

4.1 Използвана среда за програмиране

За изпълнение на заданието e използван: IntelliJ IDEA

4.2 Използвана език за програмиране

За изпълнение на заданието e използван: Java

Класът Fractal във файлът Fractal.java съдържа цялата функционалност на програмата.

Първо задаваме панелът и ширината на рамката, където ще чертаем:

MyPanel **drawingArea** = **null**;  
**public final int size** = 730;

Създаваме вътрешен клас Point, който ще ни е нужен за задаване на точка използвайки нейните х и у координати:

**private static class** Point {  
 **double x**, **y**;  
  
 Point(**double** \_x, **double** \_y) {  
 **x** = \_x;  
 **y** = \_y;  
 }  
  
 Point(**int** \_x, **int** \_y) {  
 **x** = \_x;  
 **y** = \_y;  
 }  
  
 Point(Point p) {  
 **x** = p.**x**;  
 **y** = p.**y**;  
 }  
}

Създаваме друг вътрешен клас - MyPanel, в който ще е функционалността за рисуване на снежинка на Кох:

**private class** MyPanel **extends** JPanel {  
  
 **int recursion** = 0; *// How many levels of recursion to use*  
 **final int SNOWFLAKE** = 1;  
 **int whichFractal** = **SNOWFLAKE**;

В този клас са и функциите drawSnawFlake и paintComponent, които служат за, съответно, създаване на снежинка на Кох и рисуване на желания фрактал.

**1. drawSnawFlake**

Дъно на рекурсията – Ако сме стигнали края на рекурсията си, чертаем линия:

**public void** drawSnowFlake(Graphics g, Point p1, Point p2, **int** n) {  
  
 **if** (n == 0) {  
 g.setColor(Color.***red***);  
 g.drawLine((**int**) p1.**x**, (**int**) p1.**y**, (**int**) p2.**x**, (**int**) p2.**y**);  
 **return**;  
 }

Следва рекурсивната стъпка, разбиване на реда в: \_ / \ \_(разбиваме реда на 4 линии, съответно ни трябват 5 точки):

Point[] pts = **new** Point[5];  
*// Same as p1*pts[0] = **new** Point(p1);  
*// 1/3 the way from p1 to p2*pts[1] = **new** Point(p1.**x** + (p2.**x** - p1.**x**) / 3.0, p1.**y** + (p2.**y** - p1.**y**) / 3.0);

Сега трябва да изтеглим точка на 90 градуса от линията, да намерим вектора от pt1 до pt2, както и средната точка, която ще бъде нашата референтна точка.

Пресичаме vec (0,0,1), за да получим ортогонален вектор в 2-D равнина. т.е.: \_ | \_

a X b = (1,2,3) x (4,5,6) = ((2 x 6 - 3 x 5), - (1 x 6 - 3 x 4), + (1 x 5 - 2 x 4)) = (-3,6, -3).

x и y определят новия вектор. Нормализираме го до дължина 1 и правим новата дължина да е 1/3 от първоначалната:

Point vec = **new** Point(p2.**x** - p1.**x**, p2.**y** - p1.**y**);  
Point midPoint = **new** Point((p2.**x** + p1.**x**) / 2.0, (p2.**y** + p1.**y**) / 2.0);  
  
**double** x = vec.**y** \* 1;  
**double** y = -(vec.**x** \* 1);  
  
**double** rs = Math.*sqrt*((x \* x) + (y \* y));  
x /= rs;  
y /= rs;  
  
**double** newLength = Math.*sqrt*((Math.*pow*(p2.**x** - p1.**x**, 2)) + Math.*pow*(p2.**y** - p1.**y**, 2)) / 3.0;

Намираме тази средна точка. Тя е на 90 градуса от оригиналната линия или на newLength разстояние.

Намираме и другите 2 точки, по сходен на първите 2 начин:

pts[2] = **new** Point(x \* newLength + midPoint.**x**, y \* newLength + midPoint.**y**);  
*// 1/3 the way from p2 to p1*pts[3] = **new** Point(p1.**x** + (p2.**x** - p1.**x**) \* 2.0 / 3.0, p1.**y** + (p2.**y** - p1.**y**) \* 2.0 / 3.0);  
*// Same as p2*pts[4] = **new** Point(p2);

Накрая рекурсивно начертаваме получените четири линии, като спадаме нивото на рекурсията с 1:

drawSnowFlake(g, pts[0], pts[1], n - 1);  
drawSnowFlake(g, pts[1], pts[2], n - 1);  
drawSnowFlake(g, pts[2], pts[3], n - 1);  
drawSnowFlake(g, pts[3], pts[4], n - 1);

**2. paintComponent**

Създаваме крайните точки на снежинката:

**public void** paintComponent(Graphics g) {  
 *//snowflake endpoints* Point p1 = **new** Point(10, **size** / 2);  
 Point p2 = **new** Point(**size** - 10, **size** / 2);

Начертаваме фракталът, който се иска и извеждаме текущото ниво на рекурсия:

**if** (**whichFractal** == **SNOWFLAKE**) {  
 drawSnowFlake(g, p1, p2, **recursion**);  
}

g.setFont(**new** Font(**"Serif"**, Font.***BOLD***, 20));  
g.drawString(**"Levels of recursion: "** + **recursion**, **size** / 2 - 100, 60);

Вече излизаме от вътрешния клас и в главния започваме да настройваме зоната за рисуване, да създаваме бутони и тн.

Във конструктора по подразбиране Fractal() на главния клас:

- Създаваме нов панел;

- Задаваме панела drawingArea като негово съдържание;

- Казваме на този кадър да се прекрати, когато дисплеят на JFrame-а се затвори;

- Задаваме заглавието на прозореца, който се показва;

- Добавяме два бутона и action - listeners за тях;

**public** Fractal() {  
  
 **drawingArea** = **new** MyPanel();  
 **drawingArea**.setVisible(**true**);  
 setPreferredSize(**new** Dimension(**size**, (**size** / 2) + 95));  
  
 setContentPane(**drawingArea**);  
  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
  
 setTitle(**"Koch Snowflake"**);  
  
 JButton button1 = **new** JButton(**"Less"**);  
 button1.setActionCommand(**"less"**);  
 button1.addActionListener(**this**);  
 JButton button2 = **new** JButton(**"More"**);  
 button2.setActionCommand(**"more"**);  
 button2.addActionListener(**this**);  
  
 JButton button3 = **new** JButton(**"Koch Snowflake"**);  
 button3.setActionCommand(**"snowflake"**);  
 button3.addActionListener(**this**);

След това добавяме бутоните към панела, правим го видим и опаковаме тази рамка:

**drawingArea**.add(button1);  
**drawingArea**.add(button2);  
**drawingArea**.add(button3);  
  
setVisible(**true**);  
  
pack();

Функцията actionPerformed понижава нивото на рекурсията или го повишава:

**public void** actionPerformed(ActionEvent e) {  
  
 **if** (**"less"**.equals(e.getActionCommand())) {  
 **drawingArea**.**recursion**--;  
 **if** (**drawingArea**.**recursion** < 0) **drawingArea**.**recursion** = 0;  
 }  
  
 **else if** (**"more"**.equals(e.getActionCommand())) {  
 **drawingArea**.**recursion**++;  
 }

**else if** (**"snowflake"**.equals((e.getActionCommand()))) {  
 **drawingArea**.**whichFractal** = **drawingArea**.**SNOWFLAKE**;  
 }  
  
 repaint();  
}

Файлът CreateFractal.java служи само за задействане на приложението:

**public class** CreateFractal {  
 **public static void** main(String[] args) {  
 Fractal snowFlake = **new** Fractal();  
 }  
}

**5. Използвана литература:**

* Wikipedia.org
* KhanAcademy