ЛЕКЦІЯ 7. МОДЕЛЮВАННЯ ФРАКТАЛЬНИХ СТРУКТУР

Фрактали (фрактальні структури) знаходять все більше застосування в різних областях. Часто вони описують реальний світ навіть краще, ніж традиційна фізика або математика. Загалом, можна стверджувати, що все, що існує в реальному світі, є фракталами, будь то хмара, молекула, сніжинки, крони дерев, кровоносна система тощо. Фрактальну геометрію вважають одним з найголовніших відкриттів ХХ століття, що перевернуло математику, а сьогодні – таким, що перевертає й фізику.

Навколишній світ демонструє виключну математичну впорядкованість, що легко перевірити. Подивіться на дерево – кожна маленька гілочка нагадує більшу гілку, а та, в свою чергу, – іншу, ще більшу гілку, а та – сам стовбур. Це принцип рекурсивної самоподібності у природі. Та ось парадокс – для повноцінного математичного опису того ж дерева, або хмари, традиційної математики не вистачить. І тут з'являється математик Бенуа Мандельброт. Саме він у 70-х роках минулого століття ввів поняття фракталу (від латинського *fractus* – подрібнений, розбитий). Б. Мандельброт практично все життя присвятив вивченню фракталів, створивши підґрунтя фрактальної геометрії, тієї самої, за допомогою якої можна описати хмару, дерево, сніжинку, структуру кристалів та протеїнів, хвилі, кратери вулканів та багато іншого.

В чому ж відмінність фрактальної геометрії від традиційної? Традиційна евклідова геометрія працює з ідеальними абстрактними об'єктами – точкою, лінією, кругом, квадратом тощо та використовує їх для опису оточуючого світу. Більш складні об'єкти в геометрії Евкліда просто складаються з цих основних «блоків», простих фігур. Та ось в чому проблема – природа, як відомо, не використовує прямих ліній. Ми зможемо побудувати модель, наприклад, квітки або листка з кубиків, довго будуватимемо, але зможемо. Кубики будуть такими маленькими, що ніхто й не помітить (ніхто й не помічає) жодної різниці. Дивлячись на монітор комп'ютеру або екран телевізору, ми бачитимемо листок, а не пікселі та світлодіоди, що його формують. То в чому ж проблема? А в тому, що складаючи складні об'єкти з кубиків (простих фігур), ми не зрозуміємо закономірності їх побудови, «універсальної формули». Фрактали ж та фрактальна геометрія виходять за межі традиційного математичного опису. Вони, ніби самостійна ДНК, за допомогою формул породжують те, що ми потім знаходимо всередині не лише об'єктів, а й процесів живої та неживої

природи. «Мені довелось створити геометрію того, що не має своєї геометрії», – писав Б. Мандельброт.

Фрактальні структури утворюють унікальні фізичні властивості. Так антена, зроблена за фрактальним малюнком, має більший ККД та покриває ширший частотний діапазон, порівняно з класичними рішеннями. Тобто маємо нові фізичні властивості фрактальної структури. Саме такі антени, маленькі та чутливі, стоять у сучасних мобільних телефонах. В самої людини той же мозок має фрактальну структуру, і теж для максимальної ефективності.

Сьогодні, коли деякі відомі теорії руйнуються, в науці може виникнути новий об’єднуючий напрям. Він є і математичним, і біологічним, і фізичним і пов'язаний з фракталами.

7.1 Деякі поняття фрактальної геометрії та хаотичної динаміки

Розглянемо окремі поняття фрактальної геометрії та хаотичної динаміки.

Самоподібність. Розділимо відрізок прямої на N рівних частин. Тоді кожну отриману частину відрізку можна вважати копією всього відрізку, зменшеною в $\frac{1}{r}$ разів (рис.7.1,а).

При цьому N та r для одновимірного випадку пов’язані співвідношенням

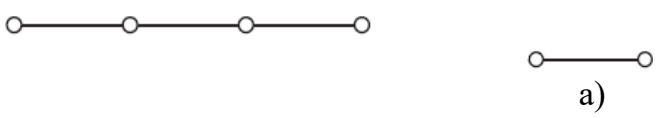
$$Nr = 1. \tag{7.1}$$

Одновимірний випадок легко узагальнюється для об’єктів більших розмірностей. Якщо квадрат розбити на N рівних квадратів, площа яких у $\frac{1}{r^2}$ разів менша площі вихідного квадрату (рис.7.1,б), то співвідношення (7.1) набуде вигляду

$$Nr^2 = 1. \tag{7.2}$$

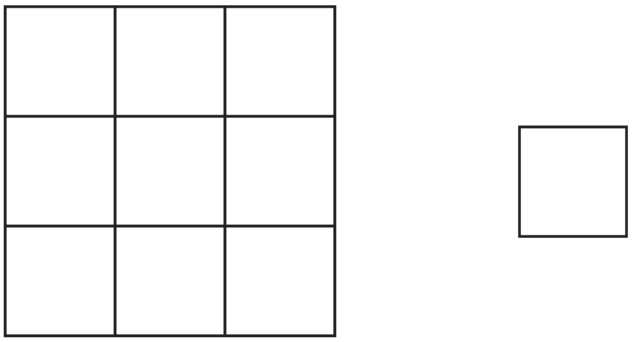
Якщо ж розбити куб на N рівних кубів з об’ємом у $\frac{1}{r^3}$ разів меншим за вихідний (рис.7.1,в), то співвідношення, аналогічне (7.1) та (7.2), запишеться у вигляді

$$Nr^3 = 1. \tag{7.3}$$



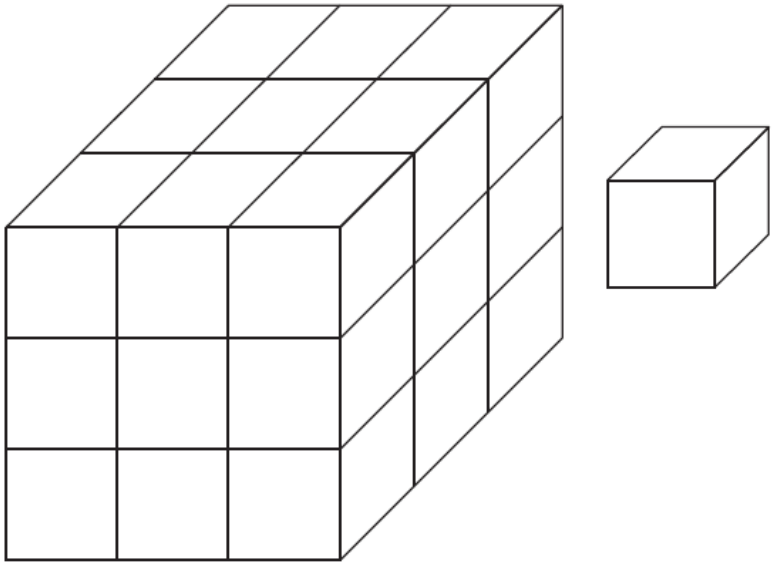
$N = 3,\ r = \frac{1}{3},\ d = 1$

а)



$N = 9,\ r = \frac{1}{3},\ d = 2$

б)



$N = 27,\ r = \frac{1}{3},\ d = 3$

в)

Рисунок 7.1 – Зв’язок розмірності та самоподібності

З (7.1)–(7.3) видно, що для кожного з розглянутих об’єктів (відрізок, двовимірний квадрат, тривимірний куб) степінь *d*, число рівних підоб’єктів *N* та коефіцієнт подібності *r* пов’язані співвідношенням

$$Nr^d = 1. \tag{7.4}$$

Об’єкти, наведені на рис.7.1, мають цілу розмірність (показник *d* цілий).

Фрактальна розмірність. Бенуа Мандельброт показав, що *існує можливість побудови таких геометричних об’єктів, у яких показник степені у рівності (7.4) не є цілим*, тобто при розбитті вихідного об’єкту на *N* підоб’єктів, що не перетинаються і

отримані масштабуванням оригіналу з коефіцієнтом r, значення d не вражатиметься цілим числом. Дані об'єкти називаються ***самоподібними фрактальними об'єктами (структурами)***. Величину d називають ***фрактальною розмірністю*** або ***розмірністю подібності***.

Явний вираз для d через N знаходиться логарифмуванням обох частин (7.4):

$$d = \frac{\log(N)}{\log\left(\frac{1}{r}\right)}. \tag{7.5}$$

Отже, застосовуючи фрактальну геометрію, можна отримати дробову розмірність. В спрощеному розумінні, дробова розмірність – це степінь кривизни нашого світу.

7.2 Фрактальні структури

Фрактальні об'єкти (структури) – це складні самоподібні структури, які часто виникають в результаті самоорганізації. ***Фрактал*** – термін, що означає геометричну фігуру, що має властивість самоподібності, тобто складену з декількох частин, кожна з яких подібна до всієї фігурі в цілому. Класичними прикладами фрактальних об'єктів є сніжинка Коха (d≈1.2618), килим Серпінського (d≈1.5850) тощо.

Фрактальні структури поділяються на групи. Найбільші з них це: геометричні, алгебраїчні, стохастичні структури та системи ітерованих функцій.

7.2.1 Геометричні фрактальні структури

Саме з них починалася історія фракталів. Цей тип фрактальних структур можна отримати шляхом простих геометричних побудов за допомогою так званих «L-систем».

7.2.1.1 Рекурсивний алгоритм побудови фрактальних об'єктів

Розглянемо алгоритм побудови фрактальних об'єктів, ґрунтований на використанні рекурсивної функції, на прикладі простого самоподібного фрактального

об’єкту – килима Серпінського. В даному алгоритмі використовується спосіб побудови, ґрунтований на послідовному видаленні з початкової (в нашому випадку, трикутної) області внутрішніх підобластей згідно із заданими правилами. Оберемо за початкову множину S_0 – рівнобічний трикутник разом з областю, яку він замикає. Видалимо внутрішню центральну трикутну область і позначимо множину, що залишилась, як S_1 (рис.7.2). Потім повторимо описаний процес для кожного з трьох залишених трикутників і отримаємо наступне наближення S_2. Продовжуючи таким чином, отримаємо послідовність вкладених множин S_n, перетин яких і створює килим Серпінського (рис.7.2). З побудови видно, що килим є об’єднанням N=3 зменшених вдвічі копій, що суттєво не перетинаються (коефіцієнти подібності по горизонталі й вертикалі, у даному випадку, виявляються однаковими $r = \frac{1}{2}$).

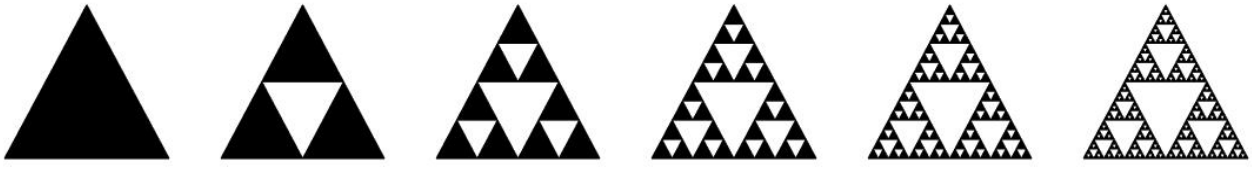


Рисунок 7.2 – Послідовність побудови килима Серпінського

Фрактальна розмірність килима Серпінського дорівнює: $d = \frac{\log(3)}{\log(2)} \approx 1.5850$.

Для побудови килима Серпінського можна використовувати такий алгоритм:

1. Задати порядок N килима.

2. Задати координати вершин вихідного трикутника ABC: (X_A, Y_A), (X_B, Y_B), (X_C, Y_C).

3. Побудувати рівнобічний трикутник ABC і зафарбувати його чорним кольором.

4. Обчислити координати середин сторін трикутника ABC:

$$dx = \frac{X_B - X_A}{2}, \quad X_{A'} = X_A + dx, \quad X_{B'} = X_B + dx + \frac{dx}{2}, \quad X_{C'} = X_C + \frac{dx}{2},$$

$$dy = \frac{Y_B - Y_A}{2}, \quad Y_{A'} = Y_A + dy, \quad Y_{B'} = Y_B + dy, \quad Y_{C'} = Y_C + dy.$$

5. Побудувати трикутник $A'B'C'$ і зафарбувати його білим кольором.

6. Повторити *N* разів дії, описані у пп.4–5 для трикутників $AA'C'$, $A'BB'$, $C'B'C$, відповідно.

Для реалізації алгоритму слід створити наведену нижче функцію Serpinsky.m, що повертає зображення килима Серпінського.

```
function z=Serpinsky(Lmax)
% Функція, що повертає зображення трикутного килима Серпінського
% Lmax - порядок килима
% задання координат вершин рівнобічного трикутника
x1=0;  y1=0;  x2=1;  y2=0;  x3=0.5;  y3=sin(pi/3);
h=figure; % ініціалізація графічного вікна
hold on;
fill([x1 x2 x3],[y1 y2 y3],'k'); % прорисовування рівнобічного трикутника
set(gca,'xtick',[],'ytick',[]); % відключення режиму оцифрування осей
set(gca,'XColor','w','YColor','w'); % встановлення кольору осей
% виклик функції, що прорисовує рівнобічні трикутники білого кольору
Simplex(x1,y1,x2,y2,x3,y3,0,Lmax);
hold off % відключення режиму прорисовування фігур в одному графічному вікні
end

function z=Simplex(x1,y1,x2,y2,x3,y3,n,Lmax)
% Рекурсивна функція, що прорисовує рівнобічні трикутники білого кольору
if n<Lmax   % задання координат вершин рівнобічного трикутника
  dx=(x2-x1)/2;        dy=(y3-y1)/2;
  x1n=x1+dx;           y1n=y1;
  x2n=x1+dx+dx/2;   y2n=y1+dy;
  x3n=x1+dx/2;         y3n=y1+dy;
  % прорисовування поточного рівнобічного трикутника
  n=n+1;      fill([x1n x2n x3n],[y1n y2n y3n],'w');
  % застосування функції Simplex.m до трикутника
  % з вершинами (Xa',Ya); (Xa',Ya); (Xc',Yc')
  Simplex(x1,y1,x1n,y1n,x3n,y3n,n,Lmax);
  % застосування функції Simplex.m до трикутника
  % з вершинами (Xa',Ya); (Xb',Yb'); (Xb',Yb')
  Simplex(x1n,y1n,x2,y2,x2n,y2n,n,Lmax);
  % застосування функції Simplex.m до трикутника
 %  з вершинами (Xc',Yc'); (Xb',Yb'); (Xc,Yc)
  Simplex(x3n,y3n,x2n,y2n,x3,y3,n,Lmax);
end
end
```

Для виводу зображення килима Серпінського, наприклад, шостого порядку, слід ввести у командному рядку MatLab ім’я функції з відповідним значенням:

```
>> Serpinsky(6);
```

Результат, що повертає функція Serpinsky.m (множину S_6), наведений на рис.7.3.

Описаний вище алгоритм легко узагальнюється для фрактальних об'єктів, правила побудови яких аналогічні правилам побудови килима Серпінського. Так, алгоритм візуалізації фрактального об'єкту, основні етапи побудови якого наведені на рис.7.4 (квадрат Серпінського), реалізуються наступною послідовністю дій.

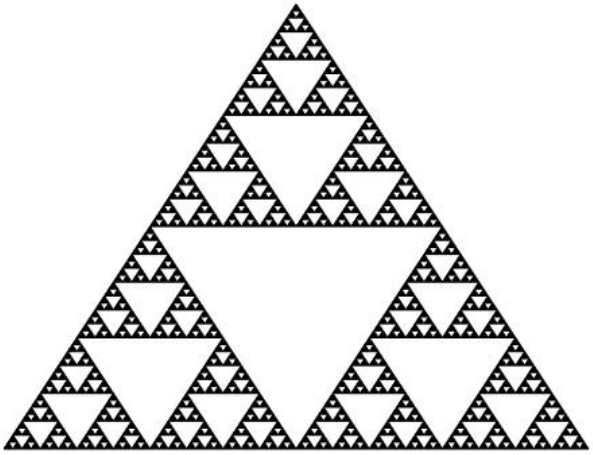


Рисунок 7.3 – Килим Серпінського 6-го порядку

Рисунок 7.4 – Послідовність побудови квадрата Серпінського

1. Задати порядок *N* квадрата.

2. Задати координати вершин вихідного квадрата *ABCD*: (X_A, Y_A), (X_B, Y_B), (X_C, Y_C), (X_D, Y_D).

3. Побудувати квадрат *ABCD* і зафарбувати його чорним кольором.

4. Обчислити координати точок, що ділять сторони квадрата на 3 рівні частини:

$$dx = \frac{X_B - X_A}{3}, \qquad dy = \frac{Y_B - Y_A}{3},$$

$$X_{A'} = X_A + dx, \qquad Y_{A'} = Y_A + dy,$$

$$X_{B'} = X_A + dx + dx, \quad Y_{B'} = Y_A + dy,$$

$$X_{C'} = X_A + dx + dx, \quad Y_{C'} = Y_A + dy + dy,$$
$$X_{D'} = X_A + dx, \qquad Y_{D'} = Y_A + dy + dy$$

5. Побудувати квадрат $A'B'C'D'$ і зафарбувати його білим кольором.

6. Повторити N разів дії, описані у пп.4–5 для квадратів з вершинами з такими координатами, відповідно:

$$\begin{array}{llll}
(X_A,Y_A), & (X_{A'},Y_A), & (X_{A'},Y_{A'}), & (X_A,Y_{A'});\\
(X_{A'},Y_A), & (X_{B'},Y_A), & (X_{B'},Y_{B'}), & (X_{A'},Y_{B'});\\
(X_{B'},Y_A), & (X_B,Y_B), & (X_B,Y_{B'}), & (X_{B'},Y_{B'});\\
(X_{B'},Y_{B'}), & (X_B,Y_{B'}), & (X_B,Y_{C'}), & (X_{C'},Y_{C'});\\
(X_{C'},Y_{C'}), & (X_B,Y_{C'}), & (X_C,Y_C), & (X_{C'},Y_{C'});\\
(X_{D'},Y_{D'}), & (X_{C'},Y_{C'}), & (X_C,Y_C), & (X_{D'},Y_{C'});\\
(X_A,Y_{D'}), & (X_{D'},Y_{D'}), & (X_{D'},Y_C), & (X_D,Y_D);\\
(X_A,Y_{A'}), & (X_{A'},Y_{D'}), & (X_{D'},Y_{D'}), & (X_A,Y_{D'}).
\end{array}$$

Нижче наведена функція, що повертає зображення квадрату Серпінського, наведеного на рис.7.4.

```
function z=SerpinskyQ(Lmax)
% Функція, що повертає зображення квадратного килима Серпінського
% Lmax - порядок килима
% задання координат вершин вихідного квадрата
x1=0; y1=0; x2=1; y2=0; x3=1; y3=1; x4=0; y4=1;
% ініціалізація графічного вікна
% встановлення параметрів прорисовування квадрату Серпінського
figure;   hold on;
set(gca,'xtick',[],'ytick',[]);   set(gca,'XColor','k','YColor','k');
fill([x1 x2 x3 x4],[y1 y2 y3 y4],'k');
% рекурсивна функція, що прорисовує квадрати білого кольору на чорному фоні
Quadrate(x1,y1,x2,y2,x3,y3,x4,y4,0,Lmax);   hold off
end

function z=Quadrate(x1,y1,x2,y2,x3,y3,x4,y4,n,Lmax)
% Рекурсивна функція, що прорисовує квадрати білого кольору
if n<Lmax
  dx=(x2-x1)/3;   dy=(y3-y1)/3;  % обчислення нових довжин сторін квадратів
  % обчислення координат вершин нових квадратів
  x1n=x1+dx;          y1n=y1+dy;
  x2n=x1+dx+dx;       y2n=y1+dy;
  x3n=x1+dx+dx;       y3n=y1+dy+dy;
  x4n=x1+dx;          y4n=y1+dy+dy;
  % прорисовування нового квадрату
  fill([x1n x2n x3n x4n],[y1n y2n y3n y4n],'w');
```

```
  n=n+1;
  % виклик функції для квадрату з вершинами  (Xa,Ya), (Xa',Ya), (Xa',Ya'), (Xa,Ya')
  Quadrate(x1,y1,x1+dx,y1,x1+dx,y1+dy,x1,y1+dy,n,Lmax);
  % виклик функції для квадрату з вершинами (Xa',Ya), (Xb',Ya),(Xb',Yb'),(Xa',Yb')
  Quadrate(x1+dx,y1,x1+2*dx,y1,x1+2*dx,y1+dy,x1+dx,y1+dy,n,Lmax);
  % виклик функції для квадрату з вершинами  (Xb',Ya),(Xb,Yb),(Xb,Yb'),(Xb',Yb')
  Quadrate(x1+2*dx,y1,x2,y1,x2,y1+dy,x1+2*dx,y1+dy,n,Lmax);
  % виклик функції для квадрату з вершинами (Xb',Yb'), (Xb,Yb'),(Xb,Yc'),(Xc',Yc')
  Quadrate(x1+2*dx,y1+dy,x2,y1+dy,x2,y1+2*dy,x1+2*dx,y1+2*dy,n,Lmax);
  % виклик функції для квадрату з вершинами (Xc',Yc'), (Xb,Yc'), (Xc,Yc),(Xc',Yc')
  Quadrate(x1+2*dx,y1+2*dy,x2,y1+2*dy,x2,y3,x1+2*dx,y3,n,Lmax);
  % виклик функції для квадрату з вершинами  (Xd',Yd'); (Xc',Yc'),(Xc,Yc),(Xd',Yc')
  Quadrate(x1+dx,y1+2*dy,x1+2*dx,y1+2*dy,x1+2*dx,y4,x1+dx,y4,n,Lmax);
  % виклик функції для квадрату з вершинами (Xa,Yd'),(Xd',Yd'),(Xd',Yc),(Xd,Yd)
  Quadrate(x1,y1+2*dy,x1+dx,y1+2*dy,x1+dx,y4,x1,y4,n,Lmax);
  % виклик функції для квадрату з вершинами  (Xa,Ya'),(Xa',Yd'),(Xd',Yd'),(Xa,Yd')
  Quadrate(x1,y1+dy,x1+dx,y1+dy,x1+dx,y1+2*dy,x1,y1+2*dy,n,Lmax);
end
end
```

Розглянемо типову фрактальну структуру – криву Коха. Для її побудови зручно використовувати рекурсивний алгоритм. Побудова цієї кривої починається з відрізка K_0 одиничної довжини. Видалимо з відрізка K_0 відрізок довжини $\frac{1}{3}$ і додамо два нових відрізка такої ж довжини, як наведено на рис.7.5.

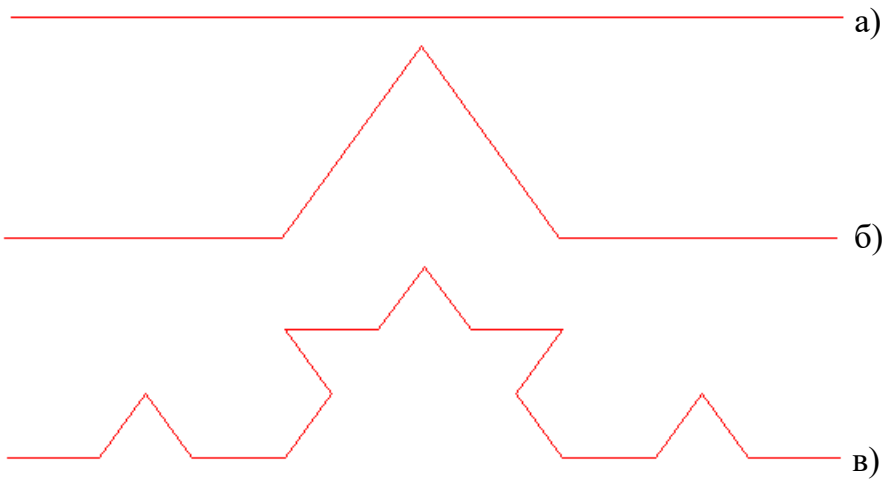


Рисунок 7.5 – Побудова кривої Коха: а) K_0; б) K_1; в) K_2

Назвемо отриману множину K_1. На наступному кроці розділимо кожний відрізок довжини $\frac{1}{3}$ на три частини довжиною $\frac{1}{9}$ і повторимо цю процедуру, замінюючи на кожному кроці середню гілку двома новими відрізками. Позначимо через K_n фігуру, отриману після n-го кроку. Можна довести, що послідовність кривих $\{K_n\}_1^\infty$ збігається до граничної кривої K нескінченної довжини, фрактальна розмірність якої дорівнює $d = \frac{log(4)}{log(3)} \approx 1.2618$.

Нижче наведена рекурсивна функція, що повертає зображення (рис.7.5) кривих Коха.

```
function Koch(N)
% Функція, що повертає зображення кривої Коха
x1=0; y1=0; % ліва точка початкового відрізку
x2=1; y2=0; % права точка початкового відрізку
figure;     axis([0 1 0 1]);
Coord(x1,y1,x2,y2,N);  % функція, що виводить криву Коха
set(gca,'XColor','w','YColor','w');  set(gca,'xtick',[],'ytick',[]);   hold on;
end

function z=Coord(x1,y1,x2,y2,n)
% Рекурсивна функція, що виводить криву Коха
if n>0
  % Обчислення координат кінців відрізків на черговому кроці рекурсії
  dx=(x2-x1)/3; dy=(y2-y1)/3;
  x1n=x1+dx; y1n=y1+dy;              x2n=x1+2*dx; y2n=y1+2*dy;
  xmid=dx/2-dy*sin(pi/3)+x1n;       ymid=dy/2+dx*sin(pi/3)+y1n;
  % рекурсія
  Coord(x1,y1,x1n,y1n,n-1);          Coord(x1n,y1n,xmid,ymid,n-1);
  Coord(xmid,ymid,x2n,y2n,n-1);    Coord(x2n,y2n,x2,y2,n-1);
else
  r1=[x1 y1];    r2=[x2 y2];     R=cat(1,r1,r2);
  plot(R(:,1),R(:,2),'Color','r');  % виведення  кривої  Коха
end
end
```

7.2.1.2 L-системи та терл-графіка

Поняття «L-система» було введено А.Лідермайером ще у 1968 р. при вивченні формальних мов. За допомогою L-системи можна не лише будувати багато відомих

самоподобніх фрактальних структур, наприклад, сніжинку Коха, килим Серпінського, кривих Пеано, Гільберта, Серпінського та ін., а й створювати нескінченну різноманітність нових об’єктів, що укладаються в цю схему.

Терл-графіка (від *turtle* – черепашка) є підсистемою виведення графічного подання фрактального об'єкту. Основним виконавцем даної системи є так звана «черепашка» (точка), яка переміщується по екрану дискретними кроками, прокреслюючи або не прокреслюючи свій слід. «Миттєве» положення «черепашки» задається трьома параметрами (x,y,α), де (x,y) – координати «черепашки», α – напрям наступного кроку (кут, який відліковується від позитивного напрямку осі x). Послідовність команд, що визначає напрям переміщення та дії «черепашки», задається кодовим словом, літери якого читаються зліва направо. Кодове слово, що є результатом роботи L-системи, може включати в себе наступні літери:

F – пересунутися на один крок вперед, прокреслюючи слід;

b – пересунутися на один крок вперед, не прокреслюючи слід;

[– відкрити гілку;

] – закрити гілку;

+ – збільшити кут α на величину Θ;

– – зменшити кут α на величину Θ;

X, *Y* – допоміжні змінні.

Розмір кроку і величина збільшення за кутом Θ задаються з самого початку і залишаються незмінними для всіх переміщень «черепашки». Формально, детермінована L-система складається з алфавіту, слова ініціалізації, яке називається *аксіомою* або *ініціатором*, та набору *породжуючих правил*, які вказують, яким чином слід перетворювати слово при кожній наступній ітерації. Наприклад, L-система, що відповідає сніжинці Коха, наведеній на рис.7.6, задається так:

Аксіома: F++F++F.

Породжуюче правило: Newf=F-F++F-F.

$\alpha = 0$, $\Theta = \pi/3$.

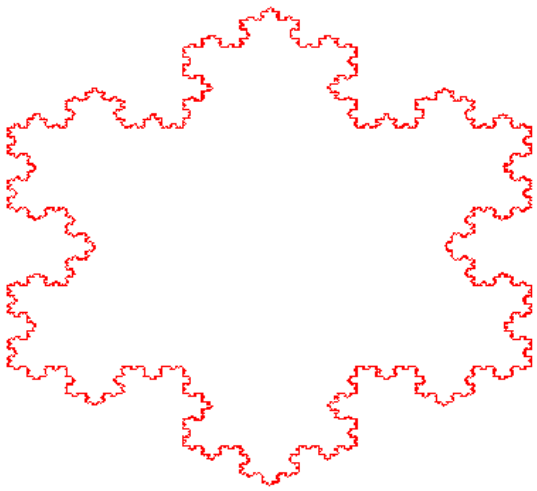


Рисунок 7.6 – Сніжинка Коха 4–го порядку

Очевидно, що графічним представленням аксіоми *F++F++F* є рівнобічний трикутник. «Черепашка» робить один крок вперед, потім кут збільшується на 2π/3 – і «черепашка» робить ще один крок вперед, далі кут знову збільшується на 2π/3 – і «черепашка» робить ще один крок.

На першому кроці кожна буква *F* у слові-ініціаторі (аксіомі) замінюється на слово *Newf* : *F-F++F-F++F-F++F-F++F-F++F-F*.

Повторюючи цей процес, на другому кроці отримаємо: *F-F++F-F-F-F++F-F++ F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F- F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F- F-F++F-F* тощо.

В пакеті MatLab найпростіше реалізувати L-систему, використовуючи рекурсивні функції. Нижче наведений приклад рекурсивної функції, яка повертає сніжинку Коха.

```
function [X,Y]=Snowflake(Lmax)
% Функція, що повертає  зображення сніжинки Коха
% Lmax — порядок сніжинки
% породжуючі правила
Axiom='F++F++F';   Newf='F-F++F-F';   teta=pi/3;
alpha=0;               p=[0;0]; % стартова точка
p=CoordSnow(p,Lmax,Axiom,Newf,alpha,teta);
% звертання до функції, що повертає координати вершин
M=size(p,2);  % число вершин сніжинки Коха
X=p(1:1,1:M); % створення вектора,  що містить  X-ті координати вершин сніжинки
Y=p(2:2,1:M); % створення вектора,  що містить  Y-ті координати вершин сніжинки
figure;     plot(X,Y,'Color','r'); % побудова сніжинки Коха
end

function z=CoordSnow(p,Lmax,Axiom,Newf,alpha,teta)
% Функція, що повертає  координати вершин сніжинки Коха
```

```
Rule=RuleKoch(Lmax,Axiom,Newf,1,' '); % задання L-системи
M=length(Rule);
for i=1:M
    Tmp=p(1:2,size(p,2):size(p,2));
    if Rule(i)=='F' % крок вперед
      R=[cos(alpha);sin(alpha)]; R=R/(4^Lmax);         Tmp=Tmp+R; p=cat(2,p,Tmp);
    end
    if Rule(i)=='+' % збільшення кута, що задає напрям руху
      alpha=alpha+teta;
    end
    if Rule(i)=='-' % зменшення кута, що задає напрям руху
      alpha=alpha-teta;
   end
end
z=p;
end

function z=RuleKoch(Lmax,Axiom,Newf,n,tmp)
% Функція, що повертає  L-систему для сніжинки Коха
% Вхідні параметри:   Lmax - порядок сніжинки,      Axiom - рядок, що містить аксіому,
% Newf - рядок, що містить породжуюче правило,  tmp - вхідна L-система
while n<=Lmax
     if n==1
       tmp=Axiom;          n=n+1;
     else
       tmp=strrep(tmp,'F',Newf); % заміна всіх букв F на породжуюче правило
       n=n+1;                  tmp=RuleKoch (Lmax,Axiom,Newf,n,tmp) ; % рекурсія
     end
end
z=tmp;
end
```

Однак при побудові інших фрактальних структур, наприклад, дракона Хартера-Хайтуея (рис.7.7), на деяких кроках виникає необхідність у зміні напряму читання правила (не зліва направо, а справа наліво).

Для вирішення даної проблеми вводять дві додаткові команди, що позначаються *X* і *Y*, які використовуються для створення відповідної L-системи, але ігноруються «черепашкою» при переміщенні.

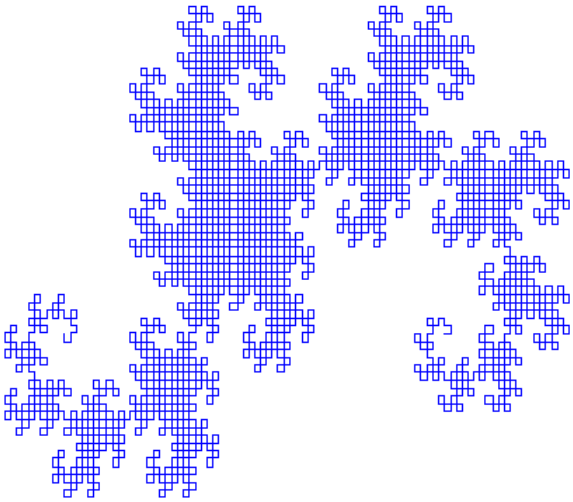


Рисунок 7.7 – Дракон Хаттера-Хайтуея 12-го порядку

При використанні цих команд породжуюче правило для дракона має вигляд:

Аксіома: FX

Породжуючі правила: *Newf=FX*

Newx=X+YF+

Newy=−FX−Y

$\alpha = 0, \Theta = \pi/2$.

Відповідно до даних правил L-система має такий вигляд:

1 крок: FX+Y+F

2 крок: FX+YF++-FX-YF+

3 крок: FX+YF++-FX-YF++-FX+YF+--FX-YF+

4 крок: FX+YF++-FX-YF++-FX+YF+--FX-YF++-FX+YF++-FX-YF+--FX+YF+--FX-YF+ .

Нижче наведена функція, що повертає зображення дракона відповідно до описаної вище L-системи.

```
function [X,Y]=Dracon(Lmax)
% Функція, що повертає  зображення дракона,  Lmax — порядок дракона
% породжуючі правила
Axiom='FX';      Newf='F';      Newx='X+YF+';      Newy='-FX-Y';
teta=pi/2;    alpha=0;    p=[0;0];
% звертання до функції, що повертає  координати кутових  точок  дракона
p=CoordDragon(p,Lmax,Axiom,Newf,Newx,Newy,alpha,teta);
M=size(p,2);
X=p(1:1,1:M);% створення  вектора, що містить  X-ві координати кутових точок дракона
Y=p(2:2,1:M);% створення вектора, що містить  Y-ві координати кутових точок дракона
figure;  plot(X,Y,'Color','b');  set(gca,'xtick',[],'ytick',[]);    set(gca,'XColor','w','YColor','w');
```

```
end

function z=CoordDragon(p,Lmax,Axiom,Newf,Newx,Newy,alpha,teta)
% Функція, що повертає  координати  кутових  точок  дракона
Rule=DraconString(Lmax,Axiom,Newf,Newx,Newy,1,' '); % задання L-системи
M=length(Rule);
for i=1:M
   Tmp=p(1:2,size(p,2):size(p,2));
   if Rule(i)=='F' % крок вперед
     R=[cos(alpha);sin(alpha)];      R=R/(2^Lmax);       Tmp=Tmp+R;          p=cat(2,p,Tmp);
   end
   if Rule(i)=='+' % збільшення кута, що задає напрям руху
     alpha=alpha+teta;
   end
   if Rule(i)=='-' % зменшення кута, що задає напрям руху
     alpha=alpha-teta;
   end
end
z=p;
end
function z=DraconString(Lmax,Axiom,Newf,Newx,Newy,n,tmp)
% Функція, що повертає  L-систему
if n<=Lmax
  if n==1       tmp=Axiom;
  end
  M=length(tmp);              tmp1=' ';
  for i=1:M
      if tmp(i)=='F'         tmp1=strcat(tmp1,Newf);
      end
      if tmp(i)=='X'         tmp1=strcat(tmp1,Newx);
      end
      if tmp(i)=='Y'         tmp1=strcat(tmp1,Newy);
      end
      if not(tmp(i)=='F') &&not(tmp(i)=='X') &&not(tmp(i)=='Y')
         tmp1=strcat(tmp1,tmp(i));
      end
  end
  tmp=tmp1;    n=n+1;    tmp=DraconString(Lmax,Axiom,Newf,Newx,Newy,n,tmp);
end
z=tmp;
end
```

Замінюючи в описаній програмі породжуючі правила, можна отримати й інші фрактальні криві.

Далі розглянемо операцію розгалуження. Коли в L-системі зустрічається символ [(відкрити гілку), необхідно запам'ятати координати точки розміщення «черепашки» і напрямок її руху, тобто змінні (x,y,α). До збережених змінних слід повернутися після виявлення символу] (закрити гілку). Для зберігання триплетів (x,y,α) пропонується використовувати стек: $\begin{bmatrix} x_1 & y_1 & \alpha_1 \\ x_2 & y_2 & \alpha_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & \alpha_n \end{bmatrix}$, в кінець якого записуються нові дані. При закритті гілки змінним (x,y,α) присвоюються значення, зчитані з кінця стека, потім ці значення зі стеку видаляються. В пакеті MatLab виявляється зручнішим використовувати матрицю зі змінним числом стовпців: $M = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix}$, причому координати кожної нової точки розгалуження додаються в новий стовпець матриці M.

Після закриття гілки змінним (x,y,α) присвоюються значення, зчитані з останнього стовпця матриці M, потім цей стовпець видаляється. Отже, розгалуження задається двома символами: [– відкрити гілку: додати вектор $\begin{bmatrix} x \\ y \\ \alpha \end{bmatrix}$ як новий стовпець матриці M та] – закрити гілку: присвоїти змінним (x,y,α) значення координат вектора, який є останнім стовпцем матриці M.

Приклад фрактальної структури, побудованої за допомогою операції розгалуження, наведений на рис.7.8. Нижче наведена функція, що повертає зображення квітки відповідно до описаної вище L–системи.

Замінюючи в описаній програмі породжуючі правила, можна отримати й інші розгалужені фрактальні об'єкти, наприклад, кущ або сніжинку.

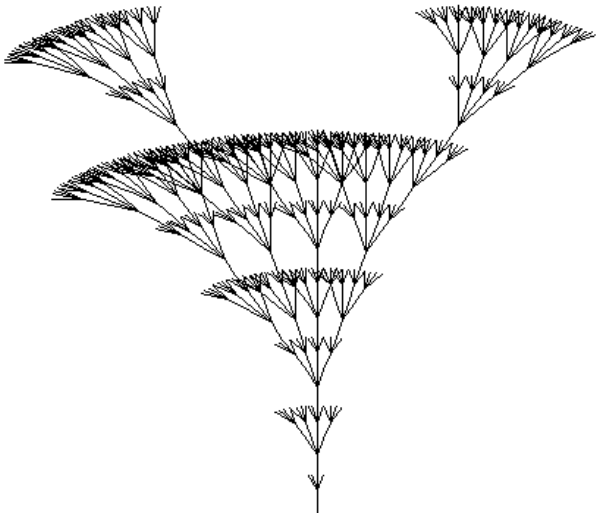


Рисунок 7.8 – Квітка 3-го порядку

```
function Flower(Lmax)
% Функція, що повертає зображення квітки,  Lmax – порядок квітки
% породжуючі правила
Axiom='F[+F+F][-F-F][++F][-F]F';   Newf='FF[++F][+F][F][-F][-F]';
teta=pi/16;  alpha=pi/2;
p=[0;0]; % початкова точка
CoordFlower(p,Lmax,Axiom,Newf,alpha,teta);  % звертання до функції координат квітки
end
function CoordFlower(p,Lmax,Axiom,Newf,alpha,teta)
% Функція координат квітки
Rule=FlowerString(Lmax,Axiom,Newf,1,''); % задання L-системи
figure;    hold on;    M=length(Rule);   L=0;    x0=p(1);    y0=p(2);
for i=1:M
    if Rule(i)=='F' % крок вперед
       x1=x0+cos(alpha); y1=y0+sin(alpha);  X=[x0,x1]; Y=[y0,y1]; x0=x1; y0=y1;
       plot(X,Y,'Color','k');       end
    if Rule(i)=='+' % збільшення кута, що задає напрям руху
      alpha=alpha+teta;         end
    if Rule(i)=='-' % зменшення кута, що задає напрям руху
      alpha=alpha-teta;         end
   if Rule(i)=='['  % відкрити гілку
      if L==0    St=[x0;y0;alpha]; L=1;
      else       St=cat(2,St,[x0;y0;alpha]);
      end
   end
   if Rule(i)==']' % закрити гілку
     M=size(St,2);   R=St(1:3,M:M);   x0=R(1);   y0=R(2);   alpha=R(3);   St=St(1:3,1:M-1);
   end
 end
set(gca,'xtick',[],'ytick',[]);   set(gca,'XColor','w','YColor','w');
```

```
hold off
end

function z=FlowerString(Lmax,Axiom,Newf,n,tmp)
% Функція, що повертає L-систему
while n<=Lmax
    if n==1        tmp=Axiom; n=n+1;
    else           tmp=strrep(tmp,'F',Newf);    n=n+1;
                   tmp=FlowerString(Lmax,Axiom,Newf,n,tmp); %рекурсія
    end
end
z=tmp;
end
```

7.2.2 Системи ітерованих функцій (IFS - Iterated Function Systems)

Ця група фрактальних об’єктів набула широкого поширення завдяки роботам Майкла Барнслі з технологічного інституту штату Джорджія. Він намагався кодувати зображення за допомогою фрактальних структур (шляхом переводу зображення з растрової форми у фрактальну). Це дозволяло досягти високих ступенів стиснення. При низьких ступенях стиснення якість рисунків поступалася якістю формату JPEG, але при високих ступенях картинки виходили якіснішими. У будь-якому випадку, цей формат не прижився, але роботи з його удосконалення ведуться й досі. Адже цей формат не залежить від розділення зображення. Оскільки зображення закодовано за допомогою формул, то його можна збільшити до будь-яких розмірів і при цьому будуть з'являтися нові деталі, а не просто збільшиться розмір пікселів.

В IFS в ході кожної ітерації замінюється якийсь полігон (квадрат, трикутник, коло) на набір полігонів, кожен з яких піддається афінним перетворенням. При афінних перетвореннях вихідне зображення змінює масштаб, паралельно переноситься уздовж кожної з осей і обертається на деякий кут. В результаті можна отримати приголомшливі коефіцієнти стиснення. Наприклад, рисунок папороті кодується за допомогою 28 !!! цифр, і один і той же малюнок виходить незалежно від того, що взяли за основу – прямокутник, коло, трикутник або що-небудь ще. *Але, на жаль, процес створення набору коефіцієнтів для довільного зображення є дуже трудомістким і займає дуже багато часу.*

У загальному випадку для побудови IFS вводиться сукупність стискаючих відображень, що діють у R^n: $T_1,\ s_1,$

$T_2,\ s_2,$

$\vdots$

$T_m,\ s_m.$

Ці m відображень використовуються для побудови одного стискаючого відображення T у просторі Ω всіх непустих компактів з R^n.

Тут перетворення Хатчінсона T: $\Omega \to \Omega$ визначається таким чином:

$$T(E) = T_1(E) \cup T_2(E) \cup \ldots \cup T_m(E), \quad E \in \Omega.$$

Дане перетворення ставить у відповідність «точкам» з Ω, під якими розуміємо компактні множини, також «точки» з Ω.

Системою ітерованих функцій називається сукупність визначених вище відображень разом з ітераційною схемою:

$$E_1 = T(E_0), \quad E_2 = T(E_1), \ldots, E_n = T(E_{n-1}), \ldots,$$

де E_0 – довільна компактна множина.

Існування граничної множини $E = \lim\limits_{n\to\infty} E_n$ системи ітерованих функцій, у сенсі збіжності розмірності Хаусдорфа, $H(E,F) = \min\{\varepsilon > 0 : E \subset F + \varepsilon \,\&\, F \subset E + \varepsilon\}$, де E та F – непусті компактні підмножини в R^n, доводить відповідна теорема.

Наприклад, IFS при побудові килима Серпінського (див. рис.7.2) задається трьома афінними перетвореннями, які у матричній формі мають такий вигляд:

$$T_1\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad T_2\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix},$$

$$T_3\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{4} \\ \frac{\sqrt{3}}{4} \end{bmatrix}.$$

Розрізняють два підходи до реалізації IFS:

1) *детермінований (DIFS)*, в якому афінні перетворення застосовуються послідовно до кожної точки початкової конфігурації;

2) *рандомізований (RIFS)*, в якому випадково обрані афінні перетворення застосовують до єдиної початкової точки.

Відомий детермінований алгоритм обчислення IFS має такі недоліки:

1. Залежність якості зображення від розміру графічного вікна (задовільна якість зображення досягається для $m \geq 256$).

2. Прив'язка алгоритму до розміру графічного вікна, і, як наслідок, великий обсяг обчислень (число операцій прямо пропорційне числу точок m^2 і числу ітерацій).

3. Виникнення аварійних зупинок програми з повідомленням про помилку «індекс вийшов за межі» при попаданні точки за межі вікна $m \times m$.

Проведемо аналіз модифікації алгоритму DIFS, що дозволяє реалізувати його в MatLab, на прикладі вже розглянутого вище килима Серпінського. По-перше, слід зауважити, що, як видно з рис. 7.9, для отримання зображення килима Серпінського необхідно на кожну точку $\left(x_i^{(0)}, y_i^{(0)}\right)$, розміщену всередині вихідного трикутника S_0, окремо подіяти кожним з афінних перетворень T_1, T_2, T_3:

$$T_1\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i1}^{(0)} \\ y_{i1}^{(0)} \end{bmatrix}, \quad T_2\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i2}^{(0)} \\ y_{i2}^{(0)} \end{bmatrix}, \quad T_3\begin{bmatrix} x_i^{(0)} \\ y_i^{(0)} \end{bmatrix} = \begin{bmatrix} x_{i3}^{(0)} \\ y_{i3}^{(0)} \end{bmatrix}.$$

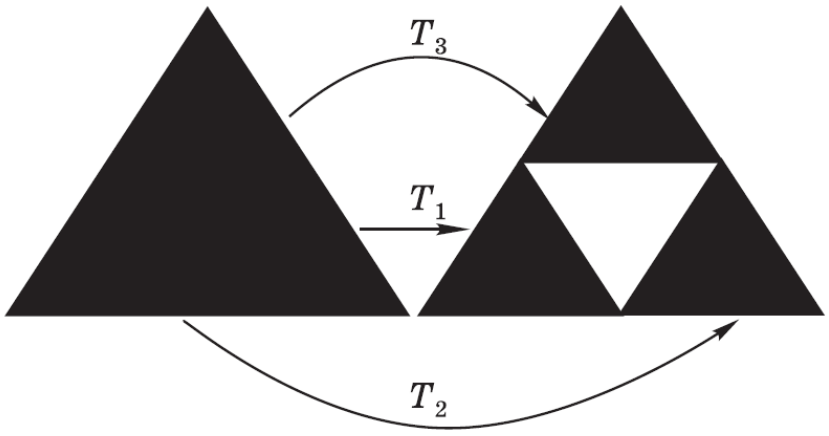


Рисунок 7.9 – Побудова килима Серпінського за допомогою DIFS

Отже, кожна точка $\left(x_i^{(0)}, y_i^{(0)}\right)$ на першому кроці ітерації породжує три нові точки: $\left(x_{i1}^{(0)}, y_{i1}^{(0)}\right)$, $\left(x_{i2}^{(0)}, y_{i2}^{(0)}\right)$, $\left(x_{i3}^{(0)}, y_{i3}^{(0)}\right)$, на кожну з цих точок на другому кроці ітерації знову слід подіяти афінними перетвореннями T_1, T_2, T_3.

В результаті кожна з трьох точок килима S_1 знову породить три точки килима S_2 тощо. Описаний процес зручно зобразити у вигляді наступного графа (рис.7.10).

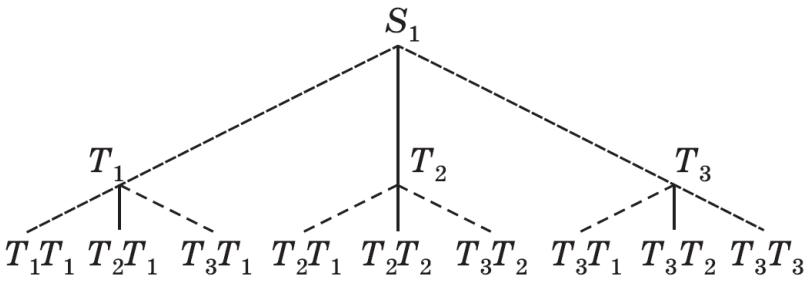


Рисунок 7.10 – Граф побудови килима Серпінського за допомогою DIFS

З рис.7.10 видно, що на другому кроці ітерації існує 9 ($3^n = 9$, де $n = 2$) правил, за якими кожній початковій точці $\left(x_i^{(0)}, y_i^{(0)}\right)$ ставляться у відповідність 9 точок килима Серпінського S_1.

Відповідно, на третьому кроці ітерації таких правил буде $3^3 = 27$:

$$
\begin{array}{lll}
T_1T_1T_1 & T_1T_2T_1 & T_1T_3T_1 \\
T_1T_1T_2 & T_1T_2T_2 & T_1T_3T_2 \\
T_1T_1T_3 & T_1T_2T_3 & T_1T_3T_3 \\
T_2T_1T_1 & T_2T_2T_1 & T_2T_3T_1 \\
T_2T_1T_2 & T_2T_2T_2 & T_2T_3T_2 \\
T_2T_1T_3 & T_2T_2T_3 & T_2T_3T_3 \\
T_3T_1T_1 & T_3T_2T_1 & T_3T_3T_1 \\
T_3T_1T_2 & T_3T_2T_2 & T_3T_3T_2 \\
T_3T_1T_3 & T_3T_2T_3 & T_3T_3T_3
\end{array}
$$

Таким чином, для побудови килима Серпінського *n*-го рівня за допомогою DIFS необхідно навчитися генерувати 3^n правил, за якими кожній точці $\left(x_i^{(0)}, y_i^{(0)}\right)$ ставиться у відповідність 3^n точок $\left(x_{ij}^{(n)}, y_{ij}^{(n)}\right)$, $j = 1, 2, \ldots, 3^n$.

Введемо позначення: $T_1 \Leftrightarrow 0$, $T_2 \Leftrightarrow 1$, $T_3 \Leftrightarrow 2$. Тоді у вибраних позначеннях правила перетворення на третьому кроці ітерації набудуть вигляду

000	010	020
001	011	021
002	012	022
100	110	120
101	111	121
102	112	122
200	210	220
201	211	221
202	212	222

Аналіз таблиці закодованих правил перетворень показує, що назви правил є нічим іншим, як множиною натуральних чисел 1, 2,..., 27, записаних у трійковій системі числення. При цьому для подання коду кожного правила використовується число цифр, що збігається з порядком килима *n*.

Відповідно, для випадку $n = 4$ маємо множину, що складається з $3^4 = 81$ правила, назви яких є множиною чисел 1, 2,..., 81, записаних у трійковій системі числення, при цьому для подання кожного числа використовуються 4 цифри. Очевидно, що для зберігання назви правил найзручніше використовувати масив рядкових змінних довжиною *n*, число елементів якого дорівнює $3^4 = 81$.

Таким чином, для побудови килима Серпінського в MatLab за допомогою DIFS можна використовувати такий алгоритм:

1. Задати порядок килима Серпінського *n*.

2. Задати число точок початкової конфігурації *m*.

3. Задати координати *i* точок (*i*=1, 2,..., *m*), що заповнюють початкову множину.

4. Перевести кожне з чисел $1, 2, \ldots, 3^n$ в трійкову систему числення.

5. Сформувати масив, що складається з 3^n рядків, довжиною *n* символів.

6. Задати афінні перетворення.

7. Для *i*-тої точки початкової конфігурації послідовно застосувати кожне з $j = 1, 2, \ldots, 3^n$ ітераційних правил і відобразити в графічному вікні отримані образи кожної початкової точки.

Нижче наводиться функція, що повертає зображення килима Серпінського.

```
function z=SerpDSIF(Niter,NPoints)
% Функція, що повертає зображення килима Серпінського,
% створюваного за допомогою алгоритму DSIF
% Niter - порядок килима,   NPoints - число точок початкової конфігурації
x=zeros(NPoints,1);        y=zeros(NPoints,1);
% задання координат точок початкової конфігурації
x1=0;   y1=0;  x2=1;  y2=0;  x3=1/2;  y3=sin(pi/3);
j=1;
while j<=NPoints
    tmpx=rand(1,1);       tmpy=sqrt(3)/2*rand(1,1);
    if(-sqrt(3)*tmpx+tmpy<=0)&&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
      x(j)=tmpx;        y(j)=tmpy;
      j=j+1;
    end
end
% формування масиву, що містить правила ітерації
for i=1:3^Niter % переведення числа з десяткової у трійкову систему числення
    Tmp(i)=system2(i-1,3);
end
n=1;
s='0';
while n<Niter     s=strcat(s,'0');   n=n+1;    end
for i=1:3^Niter
   tmp=num2str(Tmp(i));tmp1=s;
   for m=1:length(tmp)
     tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
   end
   Cod(i,1:Niter)=tmp1;
end
% задання афінних перетворень
a1=[0;0];    a2=[1/2;0];    a3=[1/4;sqrt(3)/4];    A=[1/2,0;0,1/2];
figure;    hold on;
set(gca,'xtick',[],'ytick',[]);     set(gca,'XColor','w','YColor','w');    fill([x1 x2 x3],[y1 y2 y3],'w');
% виклик функції, що будує килим Серпінського
GosperDraw(Niter,NPoints,x,y,A,a1,a2,a3,Cod);
hold off
end

function z=system2(D,m)
% Функція, що реалізує перетворення числа D у системі числення з основою m
n=1;
while D>=m^n         n=n+1;      end
if n>1
  a=floor(D/m^(n-1))*10^(n-1);        b=mod(D,m^(n-1));
  if b>=m
```

```
    b=system2(b,m);
  end
  z=a+b;
else     z=D;
end
end

function z=GosperDraw(Niter,NPoints,x,y,A,a1,a2,a3,Cod)
% Функція, що створює зображення килима Серпінського
for m=1:3^Niter
   X=x;      Y=y;
   Rule=Cod(m,:);
   for i=1:Niter
      tmp=Rule(Niter+1-i);
      if tmp=='0' % перше афінне перетворення
         [X Y]=aff(NPoints,X,Y,A,a1);
      end
      if tmp=='1' % друге афінне перетворення
        [X Y]=aff(NPoints,X,Y,A,a2);
      end
      if tmp=='2' % третє афінне перетворення
        [X Y]=aff(NPoints,X,Y,A,a3);
      end
   end
   plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','k');   % відображення результатів ітерації
end
end
function [X,Y]=aff(NPoints,x,y,A,a)
% Функція, що реалізує афінні перетворення координат
X=zeros(NPoints,1);     Y=zeros(NPoints,1);
for i=1:NPoints
   R=[x(i);y(i)];      R=A*R+a;       X(i)=R(1);         Y(i)=R(2);
end
end
```

Вочевидь, результат виконання команди >> SerpDSIF(6,100) ідентичний килиму Серпінського 6-го порядку, наведеному на рис.7.3.

Нижче наводяться функції, що здійснюють побудову інших фрактальних об’єктів (листок та папороть) за допомогою алгоритму DIFS, а також відповідні результати їх виконання (рис.7.11): >> MapleS(14,100); Paporotnic(7,100);

```
function z=MapleS(Niter,NPoints)
% Функція, що повертає зображення фрактального об'єкту Листок,
% створюваного за допомогою алгоритму DSIF
```

```
% Niter - порядок об'єкту,  NPoints - число точок початкової конфігурації
% задання координат точок початкової конфігурації
x1=0;  y1=0;  x2=1;  y2=0;  x3=1/2;  y3=sin(pi/3);
x=zeros(NPoints,1);    y=zeros(NPoints,1);
j=1;
while j<=NPoints
    tmpx=rand(1,1);       tmpy=sqrt(3)/2*rand(1,1);
    if (-sqrt(3)*tmpx+tmpy<=0)&&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
      x(j)=tmpx;       y(j)=tmpy;
      j=j+1;
    end
end
for i=1:2^Niter  % формування масиву, що містить правила ітерації
    Tmp(i)=system2(i-1,2);
end
n=1;   s='0';
while n<Niter
    s=strcat(s,'0');      n=n+1;
end
for i=1:2^Niter
  tmp=num2str(Tmp(i));    tmp1=s;
  for m=1:length(tmp)
    tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
  end
  Cod(i,1:Niter)=tmp1;
end
% задання афінних перетворень
A1=[0.4,-0.3733;0.0600,0.6000];   A2=[-0.8000,-0.1867;0.1371,0.8000];
a1=[0.3533;0.000];      a2=[1.1000;0.1000];
figure;    hold on;
set(gca,'xtick',[],'ytick',[]);    set(gca,'XColor','w','YColor','w');
% виклик функції, що повертає зображення фрактального об'єкту
Simplexf(Niter,NPoints,x,y,A1,A2,a1,a2,Cod);    hold off
end

function z=Simplexf(Niter,NPoints,x,y,A1,A2,a1,a2,Cod)
% Функція, що повертає зображення фрактального об'єкту
for m=1:2^Niter
  X=x;    Y=y;
  Rule=Cod(m,:);
  for i=1:Niter
    tmp=Rule(Niter+1-i);
    if tmp=='0'          [X Y]=aff(NPoints,X,Y,A1,a1);       end
    if tmp=='1'          [X Y]=aff(NPoints,X,Y,A2,a2);       end
  end
```

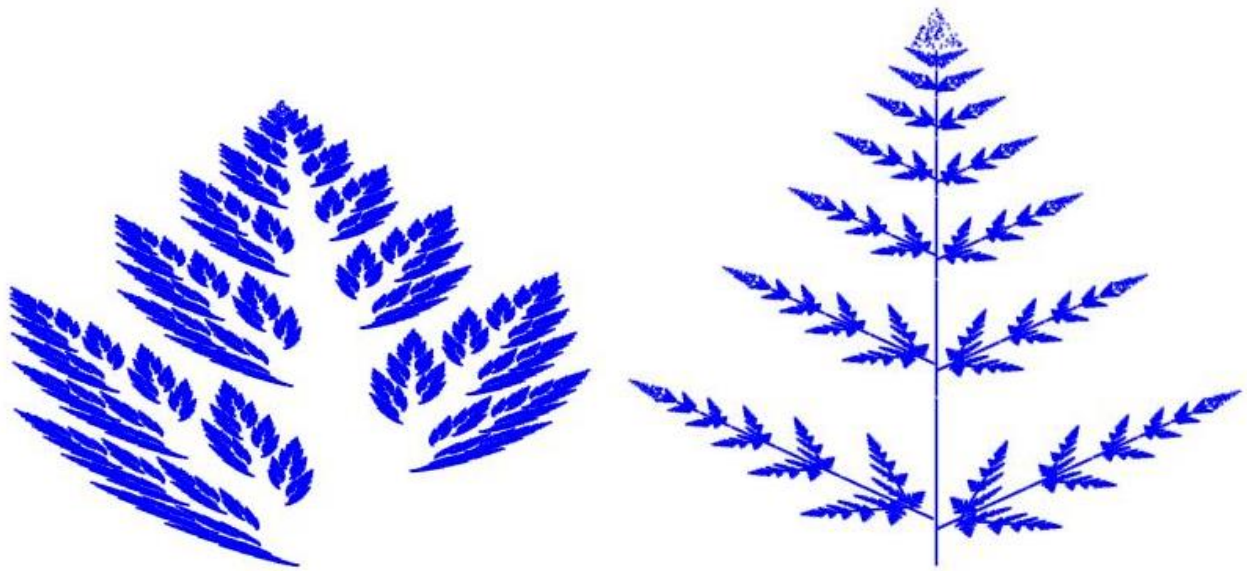
```
   plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','b');
end
end

function z=Paporotnic(Niter,NPoints)
% Функція, що повертає зображення фрактального об’єкту Папороть,
% створюваного за допомогою алгоритму DSIF
% Niter - порядок об'єкту,    NPoints - число точок початкової конфігурації
x1=0;   y1=0;   x2=1;   y2=0;   x3=1/2;   y3=sin(pi/3);
x=zeros(NPoints,1);    y=zeros(NPoints,1);
j=1;
while j<=NPoints
     tmpx=rand(1,1);      tmpy=sqrt(3)/2*rand(1,1);
     if(-sqrt(3)*tmpx+tmpy<=0)&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
       x(j)=tmpx;        y(j)=tmpy;         j=j+1;
     end
end
for i=1:4^Niter  % формування масиву, що містить правила ітерації
   Tmp(i)=system2(i-1,4);
end
n=1;     s='0';
while n<Niter
     s=strcat(s,'0');       n=n+1;
end
for i=1:4^Niter
   tmp=num2str(Tmp(i));     tmp1=s;
   for m=1:length(tmp)
      tmp1(Niter-m+1)=tmp(length(tmp)-m+1);
   end
   Cod(i,1:Niter)=tmp1;
end
% задання афінних перетворень
A1=[0.7000,0;0,0.7000];                          A2=[0.1000,-0.4330;0.1732,0.2500];
A3=[0.1000,0.4330;-0.1732,0.2500];          A4=[0,0;0,0.3000];
a1=[0.1496;0.2962];   a2=[0.4478;0.0014];  a3=[0.4450;0.1559];   a4=[0.4987;0.0070];
figure;    hold on;
set(gca,'xtick',[],'ytick',[]);     set(gca,'XColor','w','YColor','w');
% виклик функції, що будує зображення фрактального об'єкту
Simplexf1(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod);      hold off
end

function z=Simplexf1(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
% Функція, що повертає зображення фрактального об'єкту
for m=1:4^Niter
   X=x;    Y=y;
```

```
   Rule=Cod(m,:);
   for i=1:Niter
      tmp=Rule(Niter+1-i);
      if tmp=='0'        [X Y]=aff(NPoints,X,Y,A1,a1);       end
      if tmp=='1'        [X Y]=aff(NPoints,X,Y,A2,a2);       end
      if tmp=='2'        [X Y]=aff(NPoints,X,Y,A3,a3);       end
      if tmp=='3'        [X Y]=aff(NPoints,X,Y,A4,a4);       end
   end
   plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','b');
end
end

function z=Simplexf1(Niter,NPoints,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
% Функція, що повертає зображення фрактального об'єкту
for m=1:4^Niter
   X=x;      Y=y;
   Rule=Cod(m,:);
   for i=1:Niter
      tmp=Rule(Niter+1-i);
      if tmp=='0'        [X Y]=aff(NPoints,X,Y,A1,a1);      end
      if tmp=='1'        [X Y]=aff(NPoints,X,Y,A2,a2);      end
      if tmp=='2'        [X Y]=aff(NPoints,X,Y,A3,a3);      end
      if tmp=='3'        [X Y]=aff(NPoints,X,Y,A4,a4);      end
   end
   plot(X,Y,'.','MarkerSize',1,'MarkerEdgeColor','b');
end
end
```



а) б)

Рисунок 7.11 – Фрактальні об’єкти Листок (а) та Папороть (б)

На відміну від DIFS, у рандомізованому алгоритмі (RIFS) початкова множина S_0 складається з однієї точки (x_0, y_0), а правило, за яким точці (x_0, y_0) ставиться у відповідність точка (x_i, y_i), де i – номер правила, обирається випадковим чином з набору, який містить всі можливі правила афінних перетворень. Наприклад, у застосуванні до килима Серпінського це означає, що при побудові килима 2-го порядку перетворення повинне випадковим чином обиратися з такої множини перетворень: $\{T_1, T_2, T_3, T_1T_1, T_1T_2, T_1T_3, T_2T_3, T_3T_1, T_3T_2, T_3T_3\}$, число елементів N якого, очевидно, дорівнює $N = \sum_{k=1}^{n} m^k$.

Таким чином, для побудови килима Серпінського в MatLab за допомогою RIFS можна використовувати такий алгоритм:

1. Задати порядок килима Серпінського.
2. Задати кількість випробувань NTrial.
3. Задати число афінних перетворень $m = 3$.
4. Сформувати масив, що містить набір правил для афінних перетворень.
5. Задати координати початкової точки (x_0, y_0).
6. Перевести кожне з чисел $1, 2, \ldots, 3^n$ в трійкову систему числення.
7. Задати афінні перетворення.
8. Для заданого числа випробувань послідовно, починаючи з початкової точки, згідно з правилами афінних перетворень, які обирають випадковим чином, обчислити точки ітераційної послідовності.
9. Відобразити обчислену множину точок в графічному вікні.

У загальному випадку для фрактальних об’єктів n-го порядку, зображення яких створюється за допомогою афінних перетворень, як і при використанні DIFS, необхідно переводити числа $1, 2, \ldots, m^n$ в m-кову систему числення.

Нижче наводиться приклад функції, що будує зображення фрактального об’єкту Кристал за допомогою RIFS.

```
function z=Cristal(Niter,NTrial)
% Функція, що повертає зображення кристалу, створеного згідно з RIFS
% Niter - порядок кристалу,   NTrial - кількість випробувань
% задання координат початкової точки
x1=0;  y1=0;  x2=1;  y2=0;  x3=1/2;  y3=sin(pi/3);  j=1;  Flag=0;
```

```
while Flag==0
    tmpx=rand(1,1);     tmpy=sqrt(3)/2*rand(1,1);
    if(-sqrt(3)*tmpx+tmpy<=0)&&(sqrt(3)*tmpx+tmpy-sqrt(3)<=0)
     x=tmpx;       y=tmpy;       Flag=1;
    end
end
% створення системи породжуючих правил
k=1;
for m=1:Niter
  for i=1:4^m        Tmp(k)=system2(i-1,4);        k=k+1;       end
end
Q(1)=4;
for m=2:Niter        Q(m)=Q(m-1)+4^m;     end
n=1;  s='0';  M=1;
while n<=length(Tmp)
    m=1;
    while n>Q(m)            m=m+1;       end
    if m==1
      S(n,1:1)=s;
    else
      S(n,1:1)=s;
      for i=2:m             S(n,1:i)=strcat(S(n,:),s);          end
    end
n=n+1;
end
for i=1:k-1
  tmp=num2str(Tmp(i));    m=1;
  while i>Q(m)          m=m+1;      end
  tmp1(1:m)=S(i,1:m);
  for m=1:length(tmp)
    tmp1(length(tmp1)-m+1:length(tmp1)-m+1)=tmp(length(tmp)-m+1:length(tmp)-m+1);
  end
  Cod(i,1:length(tmp1))=tmp1;
end
% задання параметрів афінних перетворень
A1=[0.2550,0.0000;0.0000,0.2550];              A2=[0.2550,0.0000;0.0000,0.2550];
A3=[0.2550,0.0000;0.0000,0.2550];              A4=[0.3700,-0.6420;0.6420,0.3700];
a1=[0.3726;0.6714];   a2=[0.1146;0.2232];   a3=[0.6306;0.2232];   a4=[0.6356;-0.0061];
% ініціалізація графічного вікна
figure;   hold on;
set(gca,'xtick',[],'ytick',[]);   set(gca,'XColor','w','YColor','w');
% виклик функції, що повертає зображення фрактального об'єкту
DrawFractal(NTrial,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod);  hold off
end
```

```
function z=DrawFractal(NTrial,x,y,A1,A2,A3,A4,a1,a2,a3,a4,Cod)
% Функція, що повертає зображення фрактального об'єкту
X1=zeros(NTrial,1);    Y1=zeros(NTrial,1);   X=x;   Y=y;
for m=1:NTrial
   Np=1+round((size(Cod,1)-1)*rand(1,1));       Rule=Cod(Np,:);
   for i=1:length(Rule)
      tmp=Rule(length(Rule)+1-i);
      if tmp=='0'          [X Y]=affnew(X,Y,A1,a1);       end
      if tmp=='1'          [X Y]=affnew(X,Y,A2,a2);       end
      if tmp=='2'          [X Y]=affnew(X,Y,A3,a3);       end
      if tmp=='3'          [X Y]=affnew(X,Y,A4,a4);       end
   end
X1(m)=X;       Y1(m)=Y;
end
plot(X1,Y1,'.','MarkerSize',1,'MarkerEdgeColor','k');
end

function [X,Y]=affnew(x,y,A,a)
% Функція, що повертає результат афінного перетворення
R=[x;y];   R=A*R+a;  X=R(1);   Y=R(2);
end
```

Результат виконання команди >> Cristal(4,500000) наведений на рис.7.12.

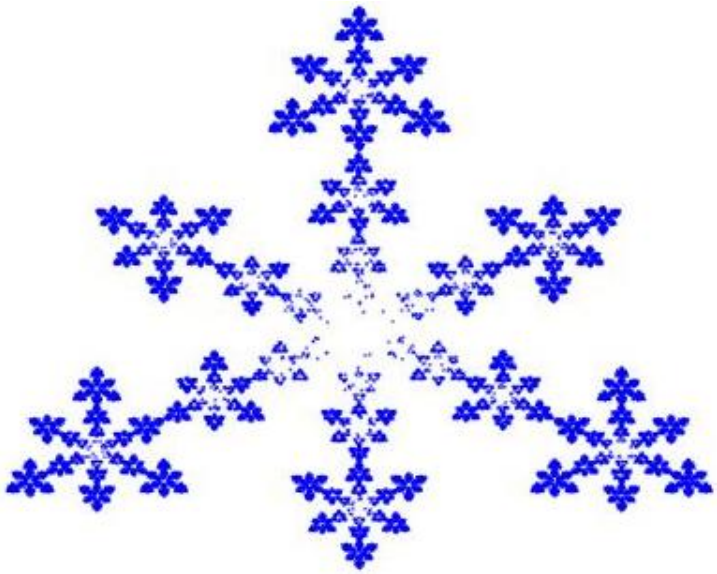


Рисунок 7.12 – Фрактальний об’єкт Кристал

7.2.3 Алгебраїчні фрактальні структури

Свою назву вони отримали за те, що їх будують, на основі алгебраїчних формул, іноді досить простих.

Методів отримання алгебраїчних фрактальних структур кілька. Одним з них є *багаторазовий (ітераційний) розрахунок функції* $Z_{n+1} = f_C(Z_n)$, де Z – комплексне число, а f – деяка функція. Розрахунок даної функції триває аж до виконання певної умови. І коли ця умова виконається – на екран виводиться точка. При цьому значення функції для різних точок комплексної площині може мати різну поведінку: 1) з часом прямує до нескінченності; 2) прямує до 0; 3) приймає кілька фіксованих значень і не виходить за їх межі; 4) поведінка хаотична, без будь-яких тенденцій.

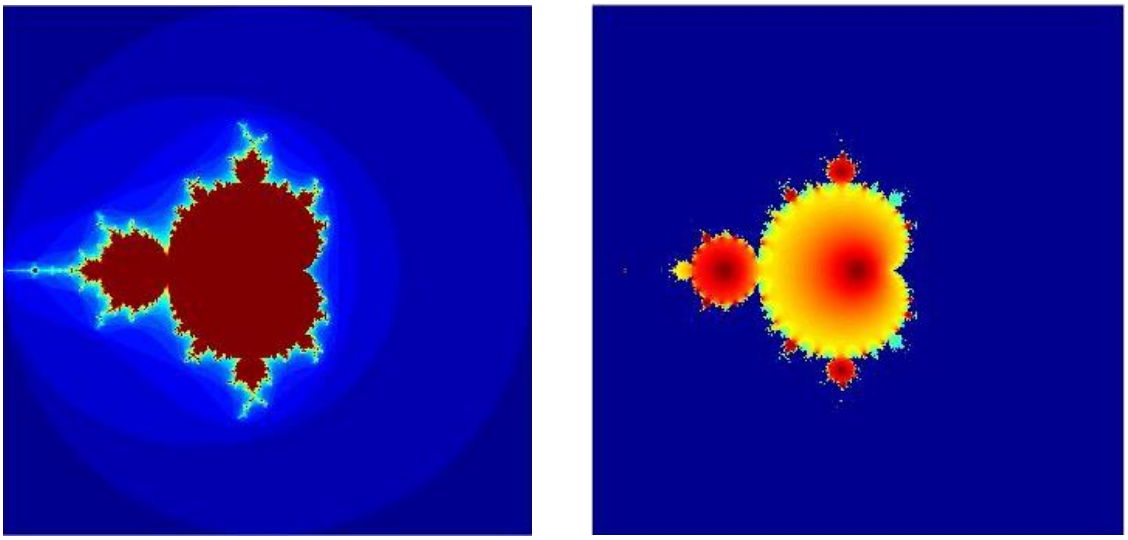
Класичною ілюстрацією алгебраїчних фрактальних структур є *множина Мандельброта*. Для її побудови нам необхідні комплексні числа. Як відомо, комплексне число – це число, що складається з двох частин – дійсної та уявної, і позначається як $a + bi$. Дійсна частина a – звичайне число, bi – уявна частина, i – уявна одиниця (якщо ми піднесемо i до квадрату, то отримаємо -1).

Геометрична інтерпретація: комплексне число зображують як точку на площині, у якої координата X – дійсна частина a, а Y – коефіцієнт при уявній частині b.

Для побудови множини Мандельброта для всіх точок на комплексній площині в інтервалі від (-2+2i) до (2+2i) виконуємо досить велику кількість разів таке: $Z_n = Z_n \cdot Z_n + C$, щоразу перевіряючи абсолютне значення Z_n. Якщо це значення $|Z_n| > 2$, то виводимо (малюємо) точку, що має колір, рівний номеру поточної ітерації, інакше виводимо точку чорного кольору.

Чорний колір всередині показує, що в цих точках функція прямує до нуля – це і є множина Мандельброта. За межами цієї множини функція прямує до нескінченності. Границі множини і є фрактальними: на границях цієї множини функція поводиться непередбачувано – хаотично.

Приклади множини Мандельброта в палітрі jet наведені на рис.7.13.



а) б)

Рисунок 7.13 – Зображення множини Мандельброта з розділенням 800×800, отримані різними функціями, за допомогою 30 ітерацій в палітрі jet

для $Z_{n+1} = f_C(Z_n) = Z_n^2 + C$

Нижче наведені два варіанти функції (mandelbro.m та mandelbrot.m), що повертає зображення множини Мандельброта (рис.7.13,а та рис.7.13,б, відповідно).

```
function mandelbro
% Варіант 1 функції, що повертає  зображення множини Мандельброта
iter=30;   % число ітерацій
npix=800; % число точок по кожній з осей координат
dl=2;    % область
x=linspace(-dl,+dl,npix);    y=linspace(-dl,+dl,npix);
% матриця  C містить всі початкові точки даної області
[X,Y] = meshgrid(x,y);      C=X+1i*Y;
% масив B міститиме всі точки, що залишились в області;
% елемент масиву дорівнює числу ітерацій, протягом яких точка залишалась в області
B = zeros(npix);      Z = B;
for l = 1:iter      Z = Z.^2 + C;       B = B + (abs(Z)<2);     end
imagesc(B);     colormap(jet);      axis ('equal','off')
end
```

```
function mandelbrot(iter,npix,palitra)
% Варіант 2 функції, що повертає зображення множини Мандельброта
% iter - число ітерацій,  palitra - кольорова палітра
% npix - число точок по кожній з осей координат (розділення)
% область
dl=2;
```

```
% формування лінійного масиву розміром 1xnpix з початковим і кінцевим елементами
% в точках -dl та +dl
x=linspace(-dl,+dl,npix);        y=linspace(-dl,+dl,npix);
% матриця C містить всі початкові точки даної області
[X,Y]=meshgrid(x,y);          C=X+1i*Y;       Z=zeros(npix);
for l=1:iter       Z=Z.^2+C;     end
W=exp(-abs(Z));   figure;   imagesc(W);    colormap(palitra);   axis ('equal','off')
end
```

Цікаво, що за тією ж формулою, що й множина Мандельброта, утворюється й інша фрактальна структура – *множина Жюліа*, отримана французьким математиком Г. Жюліа (рис.7.14). Виникає питання: якщо обидва ці фрактальні об'єкти згенеровані за однією формулою, чому ж вони такі різні?

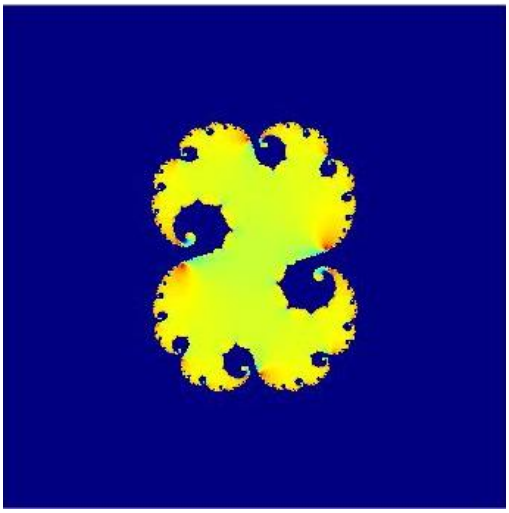


Рисунок 7.14 – Зображення множини Жюліа з розділенням 800×800, отримані за допомогою 30 ітерацій в кольоровій палітрі jet

Існують різні типи множин Жюліа. При створенні цього фрактального об'єкту з використанням різних початкових точок (щоб почати ітераційний процес), генеруються різні зображення. Це стосується лише множини Жюліа.

Хоча цього не можна побачити на рисунках, множина Мандельброта – це, насправді, множина з'єднаних фрактальних об'єктів Жюліа. Кожна точка (або координата) множини Мандельброта відповідає фрактальному об'єкту Жюліа.

Отже, взявши замість виразу $C = a + bi$ вираз $Z_0 = a + bi$ та надавши величині C довільного значення, можна отримати множину Жюліа (рис.7.14). Нижче наведена функція, що повертає зображення множини Жюліа.

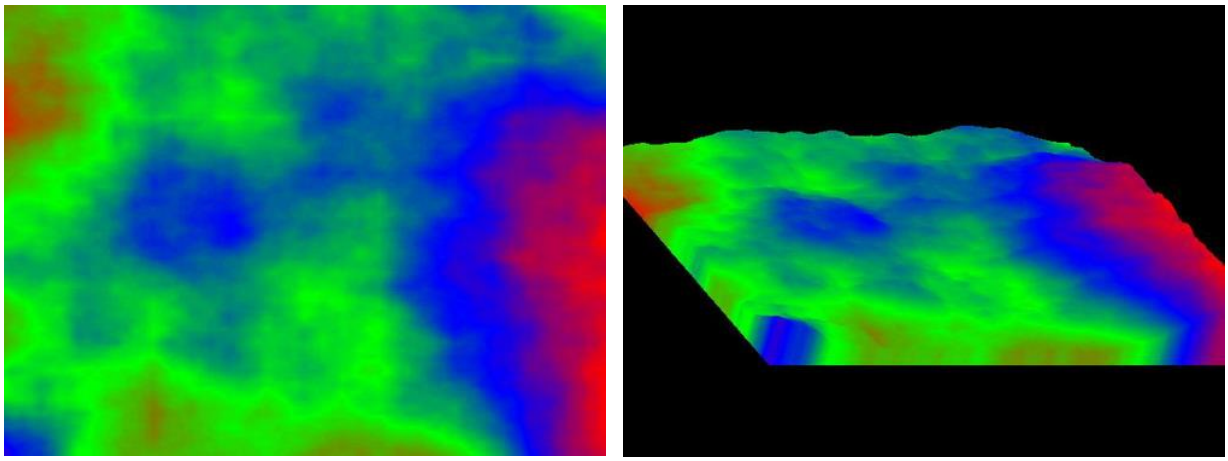
```
function julia(iter,palitra,npix)
% Функція, що повертає зображення множини Жюліа
% iter - число ітерацій, palitra - кольорова палітра
% npix - число точок по кожній з осей координат (розділення)
% область
dl=1.25;
% формування лінійного масиву розміром 1xnpix з початковим і кінцевим елементами
% в точках -dl та +dl
x=linspace(-dl,+dl,npix);        y=linspace(-dl,+dl,npix);
% матриця C містить всі початкові точки даної області
[X,Y]=meshgrid(x,y);          Z=X+1i*Y;     C=0.27334-0.00742*1i;  % початкова точка
for l=1:iter     Z=Z.^2+C;    end
figure;     W = exp(-abs(Z));  imagesc(W);     colormap(palitra);     axis ('equal','off')
end
```

Існують й інші приклади алгебраїчних фрактальних об’єктів, наприклад, лямбда-фрактал (варіант множин Мандельброта і Жюліа), басейни Ньютона, біоморфи, дерево Барнслі, палаючий корабель, павук тощо.

7.2.4 Стохастичні фрактальні структури

Ці фрактальні структури можна отримати в тому випадку, якщо в ітераційному процесі випадковим чином змінювати будь-які його параметри. При цьому виходять об'єкти дуже схожі на природні – несиметричні дерева, порізані берегові лінії та ін. Двовимірні стохастичні фрактальні структури використовуються при моделюванні рельєфу місцевості та поверхні моря.

Типовим представником стохастичних фрактальних структур є «Плазма» (рис.7.15,а). Для її побудови, наприклад, беруть прямокутник і для кожного його кута визначається колір. Далі знаходять центральну точку прямокутника і розфарбовують її в колір, що дорівнює середньому арифметичному кольорів по кутах прямокутника плюс деяке випадкове число. Чим більшим є випадкове число, тим більш «рваним» буде малюнок. Якщо вважати колір точки висотою над рівнем моря, замість плазми отримаємо гірський масив. Саме на цьому принципі ґрунтується моделювання гір в більшості програм. За допомогою алгоритму, схожого на плазму, будується карта висот, до неї застосовуються різні фільтри, накладається текстура – і фотореалістичні гори готові (рис.7.15,б).



а) б)

Рисунок 7.15 – Зображення фрактальної структури «Плазма» (а) та гірського масиву (б)

7.2.5 Фрактали у природі та їх застосування

У всьому, що нас оточує, ми часто бачимо хаос, але насправді це не випадковість, а ідеальна форма, розгледіти яку нам допомагають фрактальні структури. Природа – найкращий архітектор, ідеальний будівельник та інженер. Вона влаштована дуже логічно, і якщо десь ми не бачимо закономірності, це означає, що її треба шукати в іншому масштабі. Дійсно, природні об'єкти часто мають фрактальні властивості: у живій природі – це, наприклад, корали, морські зірки та їжаки, морські раковини, квіти і рослини (броколі, капуста), плоди (ананас), крони дерев і листя рослин, кровоносна система та бронхи людей і тварин; у неживій природі – границі географічних об'єктів (країн, областей, міст), берегові лінії, гірські хребти, сніжинки, хмари, блискавки, утворювані на склі візерунки, кристали, сталактити, сталагміти, геліктити тощо.

Для моделювання природних об’єктів можуть застосовуватися стохастичні (випадкові) фрактальні структури. Їх прикладами є: траєкторія броунівського руху на площині і в просторі; границя траєкторії броунівського руху на площині; еволюції Шрамма-Левнера – конформно-інваріантні фрактальні криві, що виникають у критичних двовимірних моделях статистичної механіки, наприклад, в моделі Ізінга і перколяції; різні види рандомізованих фрактальних структур, отриманих за допомогою рекурсивної процедури, в яку на кожному кроці введений випадковий параметр. Плазма – приклад використання такої структури в комп'ютерній графіці.

У фізиці фрактальні структури (об’єкти) використовують при моделюванні нелінійних процесів, таких як турбулентний плин рідини, складні процеси дифузії-адсорбції, полум'я, хмари тощо, при моделюванні властивостей поверхонь твердих тіл, для опису блискавки, при аналізі процесів втомного руйнування матеріалів, при дослідженні різних стадій зростання речовини за рахунок дифузії та подальшої агрегації, в квантовій механіці при описі геометричної структури хвильових функцій у точці переходу метал-діелектрик. В біології фрактальні структури застосовуються для моделювання популяцій і для опису систем внутрішніх органів (система кровоносних судин).

Контрольні питання

1. Наведіть алгоритми побудови випадкових систем. Які особливості розглянутих моделей зростання кластерів.
2. Визначити поняття фракталу, фрактальних структур? Наведіть приклади фрактальних структур та особливості алгоритмів побудови фрактальних об'єктів.
3. Визначити поняття L-системи. Наведіть алгоритми побудови фрактальних зображень за допомогою терл-графіки.
4. Як побудувати алгебраїчну фрактальну структуру?
5. Як побудувати стохастичну фрактальну структуру?
6. Побудова системи ітерованих функцій. Застосування афінних перетворень.
7. Які природні об’єкти мають фрактальну форму?
8. Наведіть області застосування фрактальних об’єктів.