# 海龜製圖法（ Turtle Graphics ）

南台科大 電子系　黎靖

圖形可視為線段的集合，繪圖的順序可視為這些線段的一種排序，若繪圖的線段順序具有此特性: 除了第 1 條與最後 1 條線段以外，其他任何一條線段的起點是前一條線段的終點，則此繪圖法稱為一筆畫或稱為海龜繪圖法，以頭尾相連繪圖= $\bigcup_{1\le i<n}\overline{p_i p_{i+1}}$ 。

遞迴繪圖的基本原則，就是以遞迴函式呼叫的繪圖為基礎，每次遞迴時依同樣的原則繪製圖形。結合遞迴與海龜製圖法，我們可以用很短的程式得到很多有趣的圖形。

## 一、海龜繪圖法

海龜繪圖法取名海龜並不是因為它的速度像烏龜，而是它繪圖的方式就像海龜前進的方式。在 2D 的情況下，也就是海龜只在一個 XY 平面運動的情況下，海龜的運動基本上只有前進與轉彎兩個動作，由此開始聯想，定出海龜繪圖法的幾個基本方法：

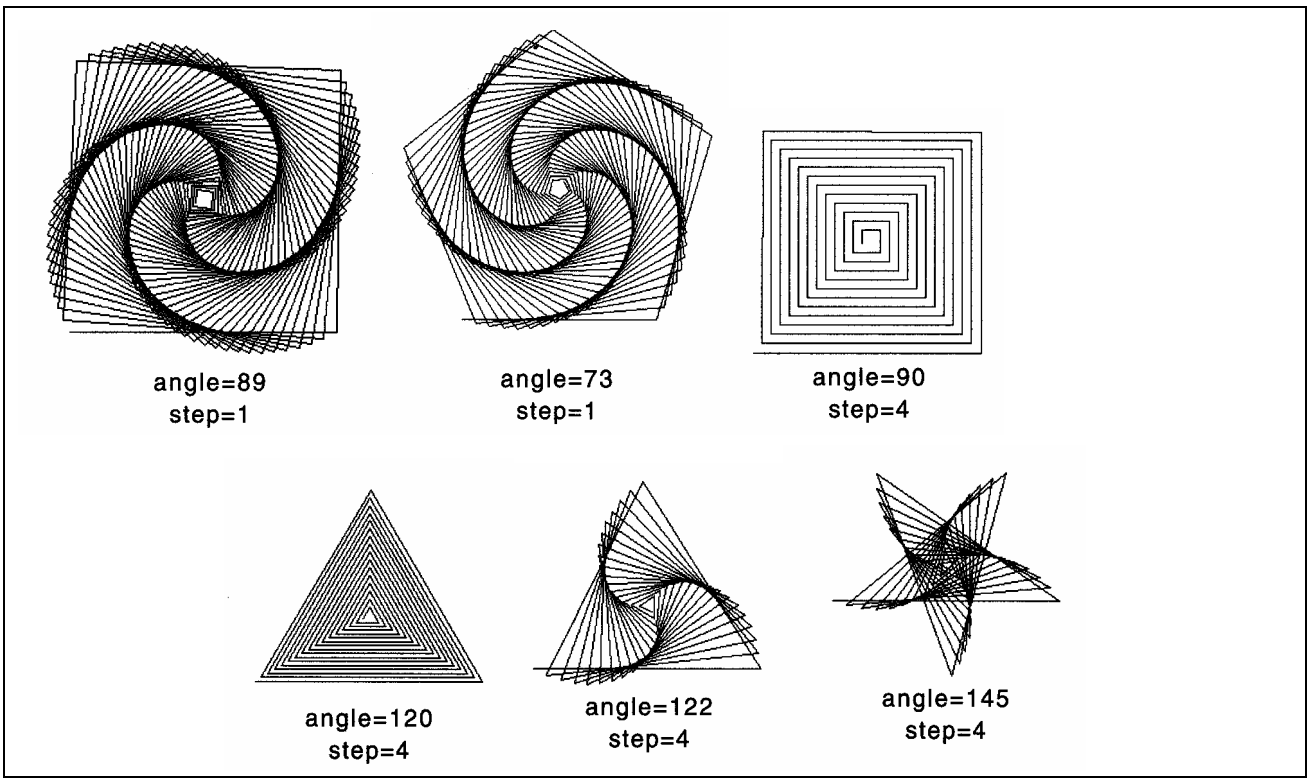
```
struct POSI { double x, y ; };
const float D2R = 0.0174533;
#define SIN(x)                  (sin((x)*D2R))
#define COS(x)                  (cos((x)*D2R))

// p1 為起點，a 是畫線的角度，L 是畫線的長度。
POSI TurtleDraw(POSI p1, const float a, const float L, shapeColor c)
{
      static POSI p2;
      p2.x = p1.x + L*COS(a);       p2.y = p1.y + L*SIN(a);
      DrawLine(p1, p2, c);    return p2;
}

void DrawLine(const POSI p1, const POSI p2, shapeColor c)
{
      lineShape Line(p1.x, p1.y, p2.x, p2.y, c);
      Line.draw();
}

POSI MoveTo(POSI base, float length, float deg)   //以 base 為起點，沿 deg 角度移動 L 距離
{
      POSI vp = {base.x + length*COS(deg), base.y + length*SIN(deg)};
      return vp;
```

```
}
```

範例 1:

```
// drawline.cpp
#include "stdio.h"
#include <math.h>
#include "d_draw.h"              // graphics library
#include "d_linesh.h"     // lineShape class
#define SIN(x)                   (sin((x)*D2R))
#define COS(x)                  (cos((x)*D2R))
const float D2R = 0.0174533;
struct POSI { double x, y ; };

void DrawLine(const POSI pA, const POSI pB, const float width, shapeColor color)
{
      if (fabs(width) < 1e-5)
      {
            lineShape Line(pA.x, pA.y, pB.x, pB.y, color);
            Line.draw();
      }
      else
      {
            static POSI p[4];
            if (fabs(pA.x - pB.x) < 1e-5)
            {
                  p[0].x = p[1].x = pA.x - 0.5*width;    p[0].y = p[3].y = pA.y;
                  p[2].x = p[3].x = p[0].x + width;       p[1].y = p[2].y = pB.y;
            }
            else if (fabs(pB.y - pA.y) < 1e-5)
            {
                  p[0].x = p[1].x = pA.x;    p[0].y = p[3].y = pA.y - 0.5*width;
                  p[2].x = p[3].x = pB.x;    p[1].y = p[2].y = pA.y + width;
            }
            else
            {
                  double m = (pA.x - pB.x)/(pB.y - pA.y);      // slope of the normal line

                  p[0].x = pA.x - sqrt(0.5*width*width/(1+m*m));
                  p[0].y = pA.y + m*(p[0].x - pA.x);
                  p[1].x = 2*pA.x-p[0].x;
                  p[1].y = 2*pA.y-p[0].y;
```

```
                p[2].x = pB.x + sqrt(0.5*width*width/(1+m*m));
                p[2].y = pB.y + m*(p[2].x - pB.x);
                p[3].x = 2*pB.x-p[2].x;
                p[3].y = 2*pB.y-p[2].y;
            }
            const double X4[4] = {p[0].x, p[1].x, p[2].x, p[3].x};
            const double Y4[4] = {p[0].y, p[1].y, p[2].y, p[3].y};
            ezdSetColor(color.convertToEzdColor());
            ezdDrawPolygon(4, &X4[0], &Y4[0]);
        }
}

POSI TurtleDraw(POSI p1, const float a, const float L, shapeColor c)
{
    POSI p2;
    p2.x = p1.x + L*COS(a);       p2.y = p1.y + L*SIN(a);
    DrawLine(p1, p2, 0.03, c);
    delayWindow(0.2);
    return p2;
}

void main()
{
    float length =5, angle = 0,    step = 0.1, turnAngle;
    printf("Enter the turn Angle.\n");
    printf("ex. : 73, 89, 90, 120, 122, 145\n");
    scanf("%f", &turnAngle);
    POSI p1={2, 0.5};
    openWindow();   // 視窗區間，預設為 10×8
    while (length > 0.05)
    {
        p1 = TurtleDraw(p1, angle, length, black);
        angle += turnAngle;    // deg
        length -= step;
    }
    viewWindow();    closeWindow();
}
```




範例 2: 樹木曲線

樹木曲線的描繪規則如下：

- 0 次的樹木曲線為長度為 $l$ 的直線
- 1 次的樹木曲線為 2 支長度為 $l/2$ 的分支，兩線的夾角為 $90^\circ$
- 2 次的樹木曲線為 2 支長度為 $l/4$ 的分支，兩線的夾角為 $90^\circ$。各個主幹分出兩個分支，所以樹木的分支共有 4 支。

此外分支的縮小率、伸展角度不一定非得 $l/2$ 與 $90^\circ$ 不可。

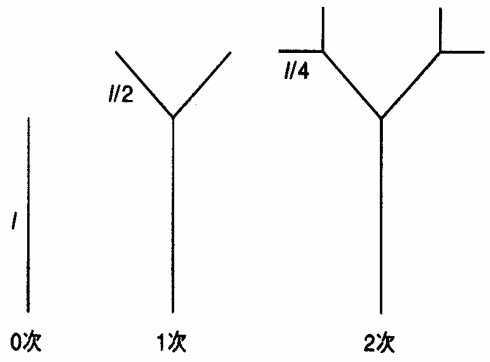



圖 8.42　樹木曲線

由於樹木曲線的主幹分出 2 支的分支，因此可將樹木曲線視為 2 元樹的樹狀結構。

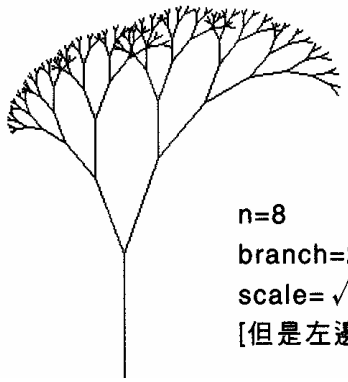
若以圖 8.43 所示的前序追蹤方式查訪這顆樹，則其分支的描繪順序為 ① → ② → ③...。

*n* 次的樹木曲線之描繪演算法的大致內容如下：

① 從$(x_0,y_0)$位置起以角度 *a* 畫出長度為 *leng* 的分支，並將畫好以後的終點座標設為新的$(x_0,y_0)$位置
② 將 *n*-1 次的右邊的樹遞迴呼叫出來
③ 將 *n*-1 次的左邊的樹遞迴呼叫出來

例題 63 的執行結果為左右對稱的圖形，左右的分支的伸展率如果不同，則會描繪出下列的變形樹。這個範例將左邊分支的伸展率設為右邊分支的伸展率的 0.8 倍。也就是將例題 63 的呼叫左邊分支的部分更改如下：

```
tree(n-1,x0,y0,leng/scale*0.8,angle+branch)
```

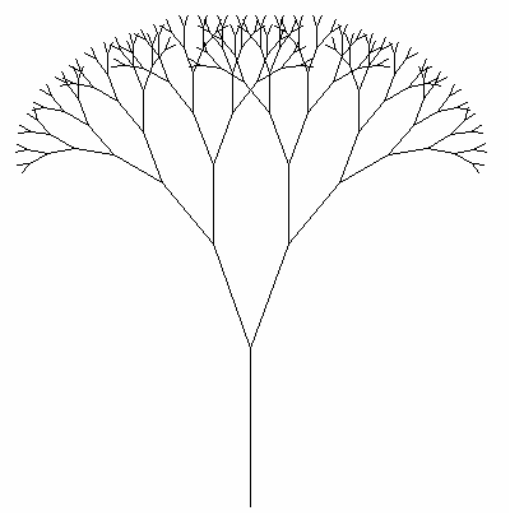


n=8
branch=20
scale= $\sqrt{2}$
[但是左邊分支的伸展率於 0.8 倍]

```
// Tree2.cpp        Recursive Tree
#include <math.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_linesh.h"     // lineShape class


struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

#define RD (0.0174533)    // RD = 3.14159/180
#define scale (sqrt(2))
```

```
#define branch_angle (20)
#define SIN(x)          (sin((x)*RD))
#define COS(x)          (cos((x)*RD))
#define VSx(x)     (((x)-WS.x1)*mfx)
#define VSy(y)     ((WS.y2-(y))*mfy)

void tree1(int branch, POSI p1, float length, float angle);
void DrawLine(POSI, POSI, shapeColor);
void setWindow(float x1, float y1, float x2, float y2);
POSI TurtleDraw(POSI p1, const float a, const float L, shapeColor c);
float mfx, mfy;

void main()
{
    int branch = 8;
    float length = 100, angle = 90;
    POSI p1 = {200, 50};
    setWindow(0, 0, 600, 480);    // x1, y1, x2, y2
    tree1(branch,p1,length,angle);
    viewWindow();    closeWindow();
}

void tree1(int branch, POSI p1, float L, float a)
{
    if (0 == branch)    return ;
    p1 = TurtleDraw(p1, a, L, black);
    tree1(branch-1, p1, L/scale, a - branch_angle);
    tree1(branch-1, p1, L/scale, a + branch_angle);
}

POSI TurtleDraw(POSI p1, const float a, const float L, shapeColor c)
{
    static POSI p2;
    p2.x = p1.x + L*COS(a);       p2.y = p1.y + L*SIN(a);
    DrawLine(p1, p2, c);
    delayWindow(0.2);
    return p2;
}

void DrawLine(POSI p1, POSI p2, shapeColor c)
```

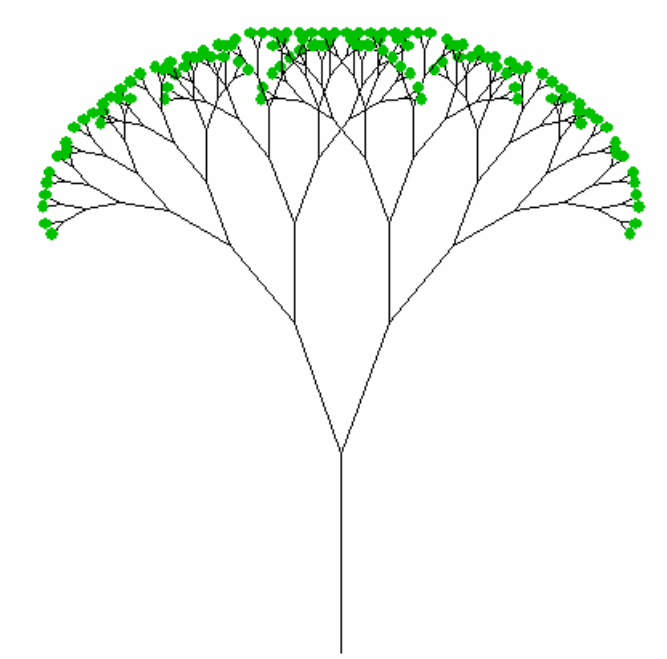
```
{
    lineShape Line(VSx(p1.x), VSy(p1.y),VSx(p2.x), VSy(p2.y), c) ;
    Line.draw();
}

void setWindow(float x1, float y1, float x2, float y2)
{
    WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
    openWindow();
    mfx = 10./(WS.x2-WS.x1);
    mfy = 8./(WS.y2-WS.y1);
    mfx = mfy = (mfx < mfy? mfx: mfy);
}
```



```
// Tree3.cpp        Recursive Tree
#include <math.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_linesh.h"     // lineShape class


struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

#define RD (0.0174533)    // RD = 3.14159/180
#define scale (sqrt(2))
#define branch_angle (20)
#define SIN(x)             (sin((x)*RD))
#define COS(x)              (cos((x)*RD))
#define VSx(x)      (((x)-WS.x1)*mfx)
#define VSy(y)      ((WS.y2-(y))*mfy)
```

```
void tree1(int branch, POSI p1, float length, float angle);
void DrawLine(POSI, POSI, shapeColor);
POSI TurtleDraw(POSI p1, const float a, const float L, shapeColor c);
void setWindow(float x1, float y1, float x2, float y2);
float mfx, mfy;

void main()
{
	int branch = 8;
	float length = 100, angle = 90;
	POSI p1 = {200, 50};
	setWindow(0, 0, 600, 480);    // x1, y1, x2, y2
	tree1(branch,p1,length,angle);
	viewWindow();    closeWindow();
}

void tree1(int branch, POSI p1, float L, float a)
{
	if (0 == branch)
	{
		ezdSetColor(ezdGreen);
		ezdDrawPoint(VSx(p1.x), VSy(p1.y));
		return ;
	}
	p1 = TurtleDraw(p1, a, L, black);
	tree1(branch-1, p1, L/scale, a - branch_angle);
	tree1(branch-1, p1, L/scale, a + branch_angle);
}

POSI TurtleDraw(POSI p1, const float a, const float L, shapeColor c)
{
	static POSI p2;
	p2.x = p1.x + L*COS(a);	p2.y = p1.y + L*SIN(a);
	DrawLine(p1, p2, c);
	delayWindow(0.2);
	return p2;
}
```

```
void DrawLine(POSI p1, POSI p2, shapeColor c)
{
	lineShape Line(VSx(p1.x), VSy(p1.y),VSx(p2.x), VSy(p2.y), c) ;
	Line.draw();
}

void setWindow(float x1, float y1, float x2, float y2)
{
	WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
	openWindow();
	mfx = 10./(WS.x2-WS.x1);
	mfy = 8./(WS.y2-WS.y1);
	mfx = mfy = (mfx < mfy? mfx: mfy);
}
```

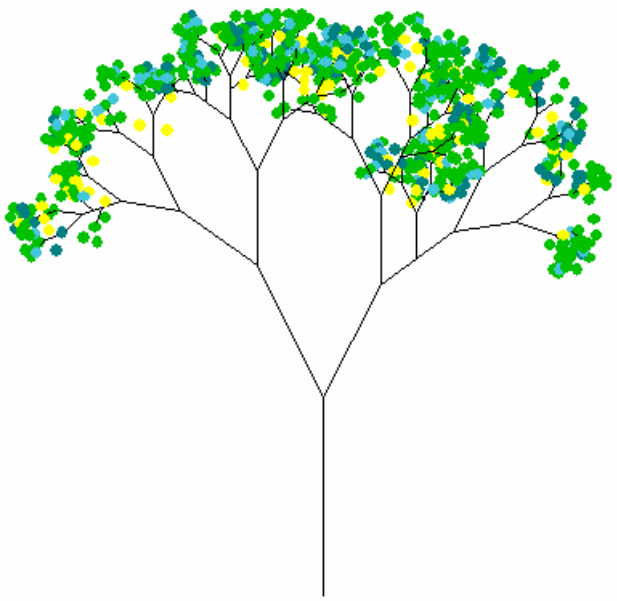


```
// Tree4.cpp		Recursive Tree
#include <time.h>
#include <math.h>
#include "d_draw.h"		// openWindow(); viewWindow();	closeWindow();
#include "d_linesh.h"	// lineShape class
double Rand_0_1(long int *seed);

struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

#define RD (0.0174533)	// RD = 3.14159/180
```

```
#define scale (sqrt(2))
#define branch_angle (27)
#define SIN(x)              (sin((x)*RD))
#define COS(x)              (cos((x)*RD))
#define VSx(x)      (((x)-WS.x1)*mfx)
#define VSy(y)      ((WS.y2-(y))*mfy)

void tree1(int branch, POSI p1, float length, float angle);
void DrawLine(POSI, POSI, shapeColor);
void setWindow(float x1, float y1, float x2, float y2);
float mfx, mfy;
long int seed;
void main()
{

      time(&seed);
      int branch = 8;
      float length = 100, angle = 90;
      POSI p1 = {200, 50};
      setWindow(0, 0, 600, 480);    // x1, y1, x2, y2
      tree1(branch,p1,length, angle);
      viewWindow();    closeWindow();
}

void tree1(int branch, POSI p1, float L, float a)
{
      POSI p2, p3;
      float factor=1;
      p2.x = p1.x + L*COS(a);
      p2.y = p1.y + L*SIN(a);
      ezdSetColor(ezdGreen);
      if (0 == branch || (branch < 3 && Rand_0_1(&seed)> 0.5))
      {
            p3.x = p1.x - L*COS(a);
            p3.y = p1.y - L*SIN(a);
            if(Rand_0_1(&seed) > 0.2) ezdSetColor(ezdGreen);
            else ezdSetColor(ezdTeal);
            ezdDrawPoint(VSx(p1.x), VSy(p1.y));
            if(Rand_0_1(&seed) > 0.2) ezdDrawPoint(VSx(p2.x), VSy(p2.y));
            if(Rand_0_1(&seed)> 0.5) ezdDrawPoint(VSx(p3.x), VSy(p3.y));
```

```
            p3.x = p1.x - 2*L*COS(a);
            p3.y = p1.y - 2*L*SIN(a);
            if(Rand_0_1(&seed)> 0.7)
            {
                  ezdSetColor(ezdYellow);
                  ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            }
            p3.x = p1.x - 0.5*L*COS(a);
            p3.y = p1.y + 0.5*L*SIN(a);
            if(Rand_0_1(&seed)> 0.7)
            {
                  ezdSetColor(ezdTurquoise);
                  ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            }
            p3.x = p1.x + 0.5*L*COS(a);
            p3.y = p1.y - 0.5*L*SIN(a);
            if(Rand_0_1(&seed)> 0.8) ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            return ;
      }
      p2.x = p1.x + L*COS(a);
      p2.y = p1.y + L*SIN(a);

      DrawLine(p1, p2, black);
//       delayWindow(0.2);
      if(Rand_0_1(&seed)> 0.7)    factor = 0.9;
      tree1(branch-1, p2, L/scale*factor, a - branch_angle);
      if(Rand_0_1(&seed)> 0.8)    factor = 1.05;
      tree1(branch-1, p2, L/scale*factor, a + branch_angle);
      p3.x = p1.x + 0.5*L*COS(a);
      p3.y = p1.y + 0.5*L*SIN(a);
      if (branch == 8)    return;
      if(Rand_0_1(&seed)> 0.85)    tree1(branch-1, p3, L/scale*0.7, a + 2*branch_angle);
      p3.x = p1.x + 0.7*L*COS(a);
      p3.y = p1.y + 0.7*L*SIN(a);
      if(Rand_0_1(&seed)> 0.8)    tree1(branch-1, p3, L/scale*0.7, a - 2*branch_angle);
}

void DrawLine(POSI p1, POSI p2, shapeColor c)
{
      lineShape Line(VSx(p1.x), VSy(p1.y),VSx(p2.x), VSy(p2.y), c) ;
```

```
    Line.draw();
}

void setWindow(float x1, float y1, float x2, float y2)
{
    WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
    openWindow();
    mfx = 10./(WS.x2-WS.x1);
    mfy = 8./(WS.y2-WS.y1);
    mfx = mfy = (mfx < mfy? mfx: mfy);
}

double Rand_0_1(long int *seed)     //高精密度之亂數函數產生器
{
    *seed = 16807 * (*seed % 127773) - 2836 * (*seed/127773);
    if (*seed < 0)     *seed += 2147483647;
    return (double) *seed / (double) 2147483647;
}
```



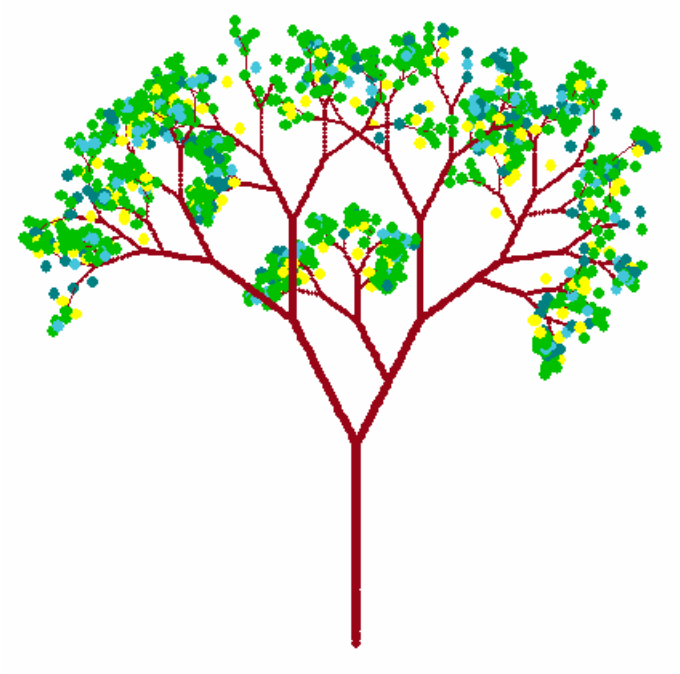
```
// Tree5.cpp       Recursive Tree
#include <time.h>
#include <math.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_linesh.h"     // lineShape class
#include "d_circsh.h"

struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;
```

```
double Rand_0_1(long int *seed);

#define RD (0.0174533)    // RD = 3.14159/180
#define scale (sqrt(2))
#define branch_angle (27)
#define SIN(x)            (sin((x)*RD))
#define COS(x)            (cos((x)*RD))
#define VSx(x)      (((x)-WS.x1)*mfx)
#define VSy(y)      ((WS.y2-(y))*mfy)

void tree1(int branch, POSI p1, float length, float angle);
void DrawBranch(int, POSI, POSI, shapeColor);
void DrawLine(const POSI pA, const POSI pB, const float width, shapeColor color);
void setWindow(float x1, float y1, float x2, float y2);
float mfx, mfy;
long int seed;
void main()
{

      time(&seed);

      int branch = 8;
      float length = 100, angle = 90;
      POSI p1 = {200, 50};
      setWindow(0, 0, 600, 480);    // x1, y1, x2, y2
      tree1(branch,p1,length, angle);
      viewWindow();    closeWindow();
}

void tree1(int branch, POSI p1, float L, float a)
{
      POSI p2, p3;
      float factor=1;
      p2.x = p1.x + L*COS(a);
      p2.y = p1.y + L*SIN(a);
      ezdSetColor(ezdGreen);
      if (0 == branch || (branch < 3 && Rand_0_1(&seed)> 0.5))
      {
            p3.x = p1.x - L*COS(a);
```

```
            p3.y = p1.y - L*SIN(a);
            if(Rand_0_1(&seed) > 0.2) ezdSetColor(ezdGreen);
            else ezdSetColor(ezdTeal);
            ezdDrawPoint(VSx(p1.x), VSy(p1.y));
            if(Rand_0_1(&seed) > 0.2) ezdDrawPoint(VSx(p2.x), VSy(p2.y));
            if(Rand_0_1(&seed)> 0.5) ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            p3.x = p1.x - 2*L*COS(a);
            p3.y = p1.y - 2*L*SIN(a);
            if(Rand_0_1(&seed)> 0.7)
            {
                  ezdSetColor(ezdYellow);
                  ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            }
            p3.x = p1.x - 0.5*L*COS(a);
            p3.y = p1.y + 0.5*L*SIN(a);
            if(Rand_0_1(&seed)> 0.7)
            {
                  ezdSetColor(ezdTurquoise);
                  ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            }
            p3.x = p1.x + 0.5*L*COS(a);
            p3.y = p1.y - 0.5*L*SIN(a);
            if(Rand_0_1(&seed)> 0.8) ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            return ;
      }
      p2.x = p1.x + L*COS(a);
      p2.y = p1.y + L*SIN(a);

      DrawBranch(branch, p1, p2, brown);
//       delayWindow(0.2);
      if(Rand_0_1(&seed)> 0.7)    factor = 0.9;
      tree1(branch-1, p2, L/scale*factor, a - branch_angle);
      if(Rand_0_1(&seed)> 0.8)    factor = 1.05;
      tree1(branch-1, p2, L/scale*factor, a + branch_angle);
      p3.x = p1.x + 0.5*L*COS(a);
      p3.y = p1.y + 0.5*L*SIN(a);
      if (branch == 8)    return;
      if(Rand_0_1(&seed)> 0.85)    tree1(branch-1, p3, L/scale*0.7, a + 2*branch_angle);
      p3.x = p1.x + 0.7*L*COS(a);
      p3.y = p1.y + 0.7*L*SIN(a);
```

```
    if(Rand_0_1(&seed)> 0.8)    tree1(branch-1, p3, L/scale*0.7, a - 2*branch_angle);
}



void DrawBranch(int branch, POSI p1, POSI p2, shapeColor c)
{
    if(branch < 3)
    {
        lineShape Line(VSx(p1.x), VSy(p1.y),VSx(p2.x), VSy(p2.y), c) ;
        Line.draw();
    }
    else  DrawLine(p1, p2,    0.6*branch,    c);
}

void setWindow(float x1, float y1, float x2, float y2)
{
    WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
    openWindow();
    mfx = 10./(WS.x2-WS.x1);
    mfy = 8./(WS.y2-WS.y1);
    mfx = mfy = (mfx < mfy? mfx: mfy);
}

double Rand_0_1(long int *seed)      //高精密度之亂數函數產生器
{
    *seed = 16807 * (*seed % 127773) - 2836 * (*seed/127773);
    if (*seed < 0)      *seed += 2147483647;
    return (double) *seed / (double) 2147483647;
}

void DrawLine(const POSI pA, const POSI pB, const float width, shapeColor color)
{

    if (fabs(width) < 1e-5)
    {
        lineShape Line(pA.x, pA.y, pB.x, pB.y, color);
        Line.draw();
    }
    else
    {
```

```
            static POSI p[4];
            if (fabs(pA.x - pB.x) < 1e-5)
            {
                  p[0].x = p[1].x = pA.x - 0.5*width;    p[0].y = p[3].y = pA.y;
                  p[2].x = p[3].x = p[0].x + width;       p[1].y = p[2].y = pB.y;
            }
            else if (fabs(pB.y - pA.y) < 1e-5)
            {
                  p[0].x = p[1].x = pA.x;    p[0].y = p[3].y = pA.y - 0.5*width;
                  p[2].x = p[3].x = pB.x;    p[1].y = p[2].y = pA.y + width;
            }
            else
            {
                  double m = (pA.x - pB.x)/(pB.y - pA.y);      // slope of the normal line

                  p[0].x = pA.x - sqrt(0.5*width*width/(1+m*m));
                  p[0].y = pA.y + m*(p[0].x - pA.x);
                  p[1].x = 2*pA.x-p[0].x;
                  p[1].y = 2*pA.y-p[0].y;

                  p[2].x = pB.x + sqrt(0.5*width*width/(1+m*m));
                  p[2].y = pB.y + m*(p[2].x - pB.x);
                  p[3].x = 2*pB.x-p[2].x;
                  p[3].y = 2*pB.y-p[2].y;

            }
            const double X4[4] = {VSx(p[0].x), VSx(p[1].x), VSx(p[2].x), VSx(p[3].x)};
            const double Y4[4] = {VSy(p[0].y), VSy(p[1].y), VSy(p[2].y), VSy(p[3].y)};
            ezdSetColor(color.convertToEzdColor());
            ezdDrawPolygon(4, &X4[0], &Y4[0]);
      }
}
```



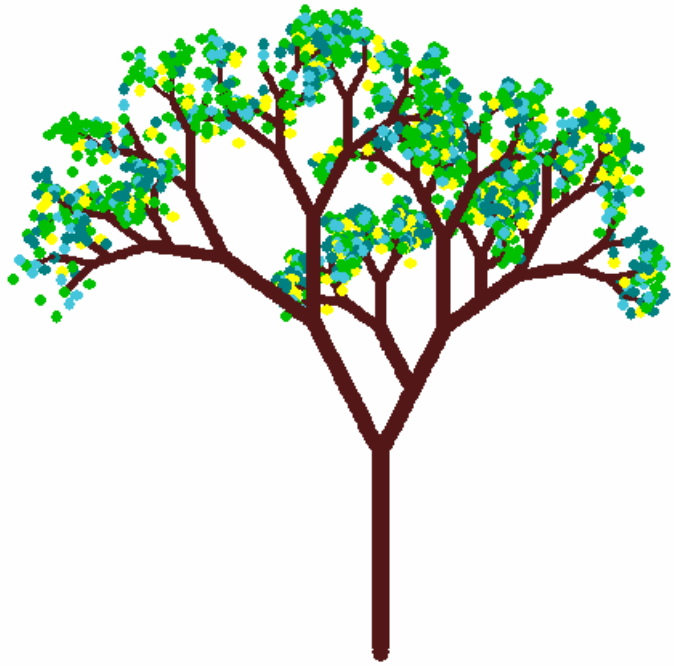
```
// Tree6.cpp        Recursive Tree
#include <time.h>
#include <math.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_linesh.h"     // lineShape class

struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

double Rand_0_1(long int *seed);

#define RD (0.0174533)    // RD = 3.14159/180
#define scale (sqrt(2))
#define branch_angle (27)
#define SIN(x)              (sin((x)*RD))
#define COS(x)              (cos((x)*RD))
#define VSx(x)      (((x)-WS.x1)*mfx)
#define VSy(y)      ((WS.y2-(y))*mfy)

void tree1(int branch, POSI p1, float length, float angle);
void DrawBranch(int, POSI, POSI, shapeColor);
void DrawLine(const POSI pA, const POSI pB, const float width, shapeColor color);
void setWindow(float x1, float y1, float x2, float y2);
float mfx, mfy;
long int seed;
```

```
void main()
{

	time(&seed);

	int branch = 8;
	float length = 100, angle = 90;
	POSI p1 = {200, 50};
	setWindow(0, 0, 600, 480);    // x1, y1, x2, y2
	tree1(branch,p1,length, angle);
	viewWindow();    closeWindow();
}

void tree1(int branch, POSI p1, float L, float a)
{
	POSI p2, p3;
	float factor=1;
	p2.x = p1.x + L*COS(a);
	p2.y = p1.y + L*SIN(a);
	ezdSetColor(ezdGreen);
	if (0 == branch || (branch < 3 && Rand_0_1(&seed)> 0.5))
	{
		p3.x = p1.x - L*COS(a);
		p3.y = p1.y - L*SIN(a);
		if(Rand_0_1(&seed) > 0.2) ezdSetColor(ezdGreen);
		else ezdSetColor(ezdTeal);
		ezdDrawPoint(VSx(p1.x), VSy(p1.y));
		if(Rand_0_1(&seed) > 0.2) ezdDrawPoint(VSx(p2.x), VSy(p2.y));
		if(Rand_0_1(&seed)> 0.5) ezdDrawPoint(VSx(p3.x), VSy(p3.y));
		p3.x = p1.x - 2*L*COS(a);
		p3.y = p1.y - 2*L*SIN(a);
		if(Rand_0_1(&seed)> 0.7)
		{
			ezdSetColor(ezdYellow);
			ezdDrawPoint(VSx(p3.x), VSy(p3.y));
		}
		p3.x = p1.x - 0.5*L*COS(a);
		p3.y = p1.y + 0.5*L*SIN(a);
		if(Rand_0_1(&seed)> 0.7)
		{
```

```
                ezdSetColor(ezdTurquoise);
                ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            }
            p3.x = p1.x + 0.5*L*COS(a);
            p3.y = p1.y - 0.5*L*SIN(a);
            if(Rand_0_1(&seed)> 0.8) ezdDrawPoint(VSx(p3.x), VSy(p3.y));
            return ;
        }
        p2.x = p1.x + L*COS(a);
        p2.y = p1.y + L*SIN(a);

        DrawBranch(branch, p1, p2, brown);
//        delayWindow(0.2);
        if(Rand_0_1(&seed)> 0.7)    factor = 0.9;
        tree1(branch-1, p2, L/scale*factor, a - branch_angle);
        if(Rand_0_1(&seed)> 0.8)    factor = 1.05;
        tree1(branch-1, p2, L/scale*factor, a + branch_angle);
        p3.x = p1.x + 0.5*L*COS(a);
        p3.y = p1.y + 0.5*L*SIN(a);
        if (branch == 8)    return;
        if(Rand_0_1(&seed)> 0.85)    tree1(branch-1, p3, L/scale*0.7, a + 2*branch_angle);
        p3.x = p1.x + 0.7*L*COS(a);
        p3.y = p1.y + 0.7*L*SIN(a);
        if(Rand_0_1(&seed)> 0.8)    tree1(branch-1, p3, L/scale*0.7, a - 2*branch_angle);
}

void DrawBranch(int branch, POSI p1, POSI p2, shapeColor color)
{
        if(branch < 3)
        {
                ezdSetColor(color.convertToEzdColor()); ;
                ezdDrawLine(VSx(p1.x), VSy(p1.y),VSx(p2.x), VSy(p2.y));
        }
        else  DrawLine(p1, p2,    0.6*branch,    color);
}

void setWindow(float x1, float y1, float x2, float y2)
{
        WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
        openWindow();
```

```
	mfx = 10./(WS.x2-WS.x1);
	mfy = 8./(WS.y2-WS.y1);
	mfx = mfy = (mfx < mfy? mfx: mfy);
}

double Rand_0_1(long int *seed)	//高精密度之亂數函數產生器
{
	*seed = 16807 * (*seed % 127773) - 2836 * (*seed/127773);
	if (*seed < 0)	*seed += 2147483647;
	return (double) *seed / (double) 2147483647;
}

void DrawLine(const POSI pA, const POSI pB, const float width, shapeColor color)
{
	ezdSetColor(EZDCOLOR(83,23,23));	//設定畫筆的顏色爲 sky blue
	if (fabs(width) < 1e-5)
		ezdDrawLine(VSx(pA.x), VSy(pA.y),VSx(pB.x), VSy(pB.y));
	else
	{
		static POSI p[4];
		if (fabs(pA.x - pB.x) < 1e-5)
		{
			p[0].x = p[1].x = pA.x - 0.5*width;	p[0].y = p[3].y = pA.y;
			p[2].x = p[3].x = p[0].x + width;	p[1].y = p[2].y = pB.y;
		}
		else if (fabs(pB.y - pA.y) < 1e-5)
		{
			p[0].x = p[1].x = pA.x;	p[0].y = p[3].y = pA.y - 0.5*width;
			p[2].x = p[3].x = pB.x;	p[1].y = p[2].y = pA.y + width;
		}
		else
		{
			double m = (pA.x - pB.x)/(pB.y - pA.y);	// slope of the normal line

			p[0].x = pA.x - sqrt(0.5*width*width/(1+m*m));
			p[0].y = pA.y + m*(p[0].x - pA.x);
			p[1].x = 2*pA.x-p[0].x;
			p[1].y = 2*pA.y-p[0].y;

			p[2].x = pB.x + sqrt(0.5*width*width/(1+m*m));
```

```
            p[2].y = pB.y + m*(p[2].x - pB.x);
            p[3].x = 2*pB.x-p[2].x;
            p[3].y = 2*pB.y-p[2].y;
        }
        const double X4[4] = {VSx(p[0].x), VSx(p[1].x), VSx(p[2].x), VSx(p[3].x)};
        const double Y4[4] = {VSy(p[0].y), VSy(p[1].y), VSy(p[2].y), VSy(p[3].y)};
        ezdDrawPolygon(4, &X4[0], &Y4[0]);
    }
}
```



範例 3: Koch curve

科赫曲線是由數學家科赫(H.von Koch)所發現的，如圖 8.35 所示， 0 次的科赫曲線為長度為 $l$ 的直線。 1 次的科赫曲線為 1 邊的長為 $l/3$ 大的正三角形狀。 2 次的科赫曲線相對於 1 次科赫曲線的各邊（4 個），為 $l/9$ 大的正三角形的角。若無限的反覆操作下去，就成為由無數條的無限小長度的線條所組成的曲線。

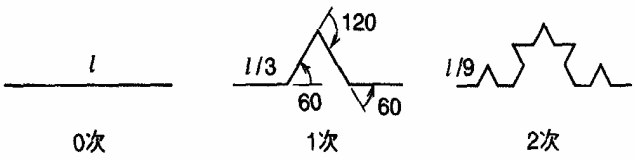
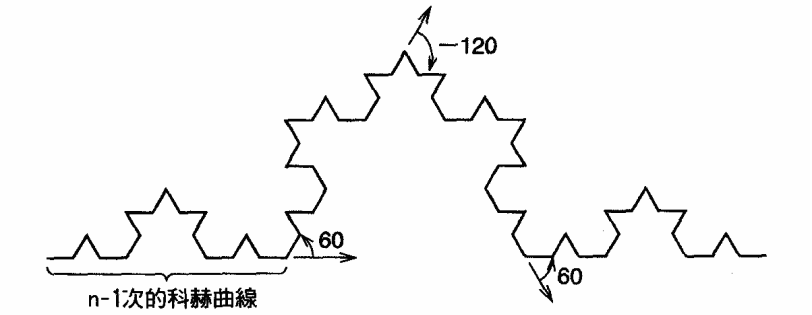



圖 8.35 科赫曲線

描繪 $n$ 次科赫曲線的演算法之大致內容為，如要描繪 $n$ 次科赫曲線，則要按下面方式描繪 4 個 $n$-1 次的科赫曲線。這裡將科赫曲線的 1 邊的長度一直固定為 *leng* 。

① 描繪 1 個 $n$-1 次的科赫曲線
② 將角度改成 60 度，描繪 1 個 $n$-1 次的科赫曲線
③ 將角度改成 -120 度，描繪 1 個 $n$-1 次的科赫曲線
④ 將角度改成 60 度，描繪 1 個 $n$-1 次的科赫曲線




將 3 個科赫曲線以 -120° 的角度連接起來，就形成 1 個白雪結晶狀的科赫島圖案。

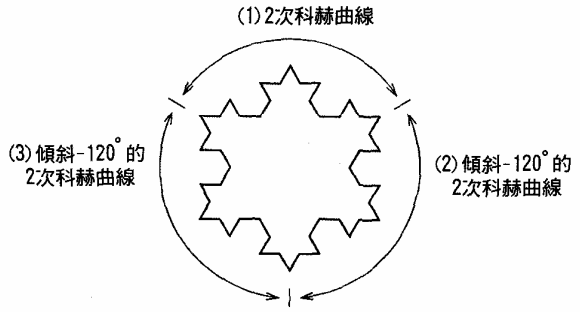



圖 8.37　科赫島

此圖形有其特別的性質，如周長為無限大，但所圍面積卻為有限。周長無限大的性質可很快的看出來。因為我們對正三角形每邊所做的步驟均一樣，因此只須考慮其中一邊的情況即可。若此正三角形的邊長為 1，則經過一個製作步驟後，此邊的邊長變成 4/3，且原來的線段變成四段小線段，再對現有的四個小線段作一次製作步驟，則每一個新的邊皆為原來邊的 4/3 倍長，也因此其總長為 $\left(\frac{4}{3}\right)^2$ ，持續不斷的作下去，可推出，當我們做了 n 個步驟後，其邊長為 $\left(\frac{4}{3}\right)^n$ ，(詳細的證明可很簡單的利用數學歸納法來證得)，當 n 趨近到無窮大時，邊長也會趨近無窮大。

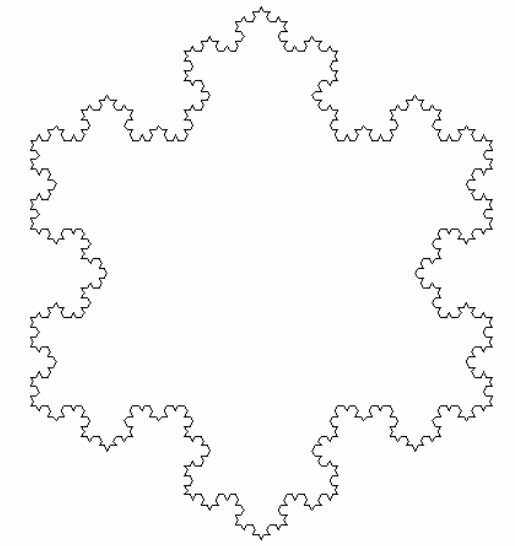
對於其所圍面積的值，我們可觀察圖 4 變化的情形。設一開始的三角形邊長為 1，則在完成第一次製作步驟後，增加了一個正三角形，其邊長為 1/3，面積為 $\frac{\sqrt{3}}{4}\left(\frac{1}{3}\right)^2$。完成第二次製作步驟後，增加了四個正三角形，每個邊長為 $\left(\frac{1}{3}\right)^2$，總面積為 $4\times\frac{\sqrt{3}}{4}\left(\frac{1}{3}\right)^4$。完成第三次製作步驟後，增加了 $4^2$ 個正三角形，每個邊長為 $\left(\frac{1}{3}\right)^3$，總面積為 $4^2\times\frac{\sqrt{3}}{4}\left(\frac{1}{3}\right)^6$。依此，利用數學歸納法可證得：當我們完成第 n 次製作步驟後，可再增加的三角形面積為 $4^{n-1}\times\frac{\sqrt{3}}{4}\left(\frac{1}{3}\right)^{2n}$。因此可得，Koch Snowflake 圖形所圍的面積為 $\frac{\sqrt{3}}{4}\cdot 1^2+3\sum_{n=1}^{\infty}\left(4^{n-1}\cdot\frac{\sqrt{3}}{4}\left(\frac{1}{3}\right)^{2n}\right)=\frac{2\sqrt{3}}{5}$，這是一個有限值，而且為原三角形面積的 8/5 倍。

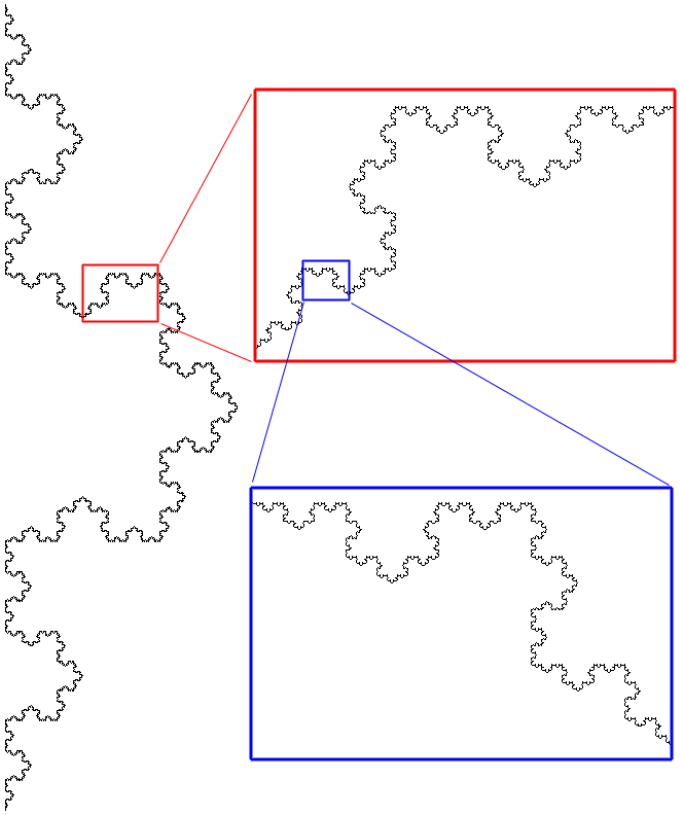
不管一開始的正三角形有多小，所做出的 Koch Snowflake 的周長皆為無窮大，而所圍面積都是原三角形的 8/5 倍，換句話說，我們可造出一個所圍面積任意小而周長無窮大的封閉圖形。發揮一下想像力，現在若有一個 Koch Snowflake 在你面前，拿個剪刀將其任意邊剪斷，開始把它拉直，你將發現，永遠都不可能把圖形拉直；反之，我們也可將一個無窮長的直線圍成一個任意小的區域。

```
// Koch.cpp       Koch curve
#include <math.h>
#include "d_draw.h"    // openWindow(); viewWindow();    closeWindow();
#include "d_linesh.h"    // lineShape class

struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

#define RD (0.0174533)    // RD = 3.14159/180
#define SIN(x)            (sin((x)*RD))
#define COS(x)            (cos((x)*RD))
#define VSx(x)     (((x)-WS.x1)*mfx)
#define VSy(y)     ((WS.y2-(y))*mfy)

void koch(int n, POSI*, float L, float);
void TurtleDraw(POSI* p1, const float a, const float L, shapeColor c);
void DrawLine(const POSI p1, const POSI p2, shapeColor c);
```

```
void setWindow(float x1, float y1, float x2, float y2);

float mfx, mfy;
void main()
{
    int i, n = 4; float length = 4, angle = 0;
    POSI p0 = {150, 300};
    setWindow(0, 0, 640, 480);    // x1, y1, x2, y2
    for (i=0; i < 3; i++)
    {
        koch(n, &p0, length, angle);angle -= 120;
    }
    viewWindow();    closeWindow();
}

void koch (int n, POSI *p1, float length, float a)
{
    if (0 == n)    TurtleDraw(p1, a, length, black);
    else
    {
        koch (n-1, p1, length, a);          a += 60;
        koch (n-1, p1, length, a);          a -= 120;
        koch (n-1, p1, length, a);          a += 60;
        koch (n-1, p1, length, a);
    }
}

void setWindow(float x1, float y1, float x2, float y2)
{
    WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
    openWindow();
    mfx = 10./(WS.x2-WS.x1);
    mfy = 8./(WS.y2-WS.y1);
    mfx = mfy = (mfx < mfy? mfx: mfy);
}

void DrawLine(const POSI p1, const POSI p2, shapeColor c)
{
    lineShape Line(VSx(p1.x), VSy(p1.y), VSx(p2.x), VSy(p2.y), c);
    Line.draw();
```

```
}

void TurtleDraw(POSI *p1, const float a, const float L, shapeColor c)
{
    static POSI p2;
    p2.x = p1->x + L*COS(a);                p2.y = p1->y + L*SIN(a);
    DrawLine(*p1, p2, c);    *p1 = p2;
}
```





自我模仿

範例 4: Hilbert curve

1891 年的 David Hilbert 提出了一種能夠填滿平面的曲線，我們稱作 Hilbert Curve，這個曲線能夠不相交錯的方式通過平面每一個分割單元，這種特性被用來處理影像分割的問題。

產生 Hilbert Curve 的方法，如下圖所示：

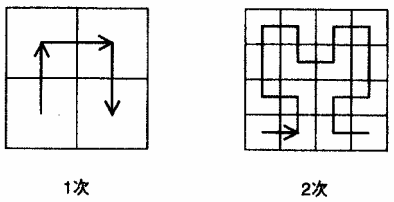



圖 8.52 希爾伯特曲線

在 2 次的希爾伯特曲線方面，①②③ 的 move(1)所描繪的情形如圖 8.53 所示。

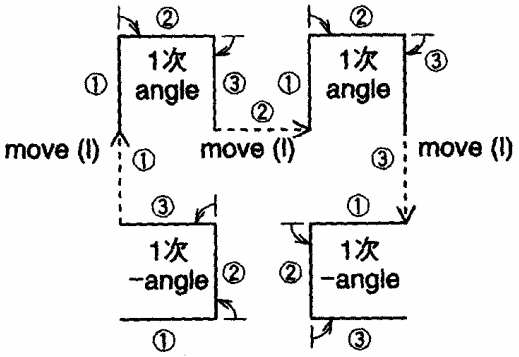
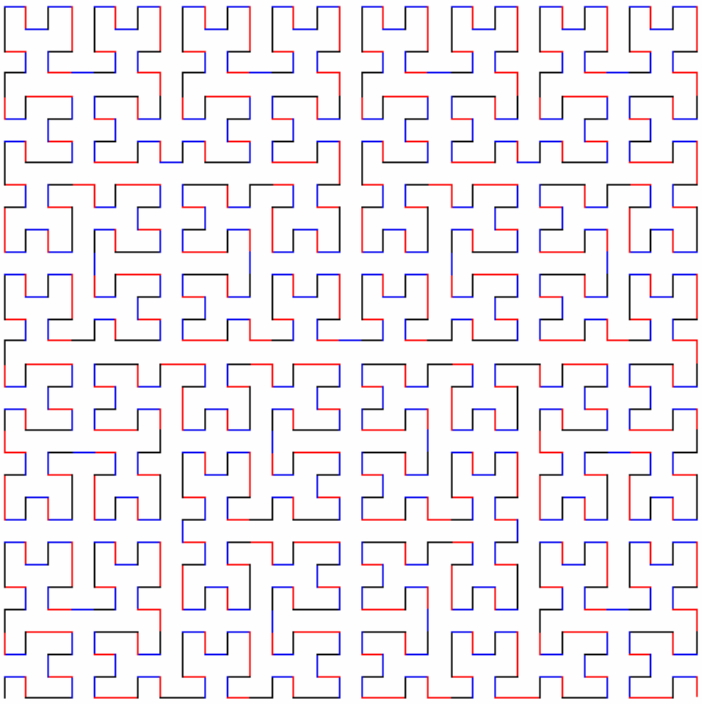



圖 8.53 2 次希爾伯特曲線的結構

```
// hilbert.cpp
#include <math.h>
#include "d_draw.h"              // graphics library
#include "d_linesh.h"     // lineShape class
#define SIN(x)                  (sin((x)*D2R))
#define COS(x)                  (cos((x)*D2R))
const float D2R = 0.0174533;
struct POSI { double x, y ; };
void DrawLine(const POSI p1, const POSI p2, shapeColor c);
void TurtleDraw(POSI *p1, const float a, const float L, shapeColor c);
void hilbert(int n, POSI*, float, float);
struct WorkingSpace { float x1, y1, x2, y2; } WS;

void     setWindow(float, float, float, float);
#define VSx(x)     (((x)-WS.x1)*mfx)
```

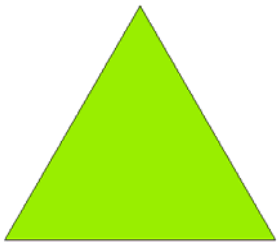
```
#define VSy(y)      ((WS.y2-(y))*mfy)
#define ROUND(x)            ((x)>0 ? (int)((x)+0.5): -(int)(0.5-(x)))
float mfx, mfy;

int ga;
void main()
{
      int    n = 5;      ga = 0;
      POSI p0 = {100, 50};
      setWindow(0, 0, 640, 480);    // x1, y1, x2, y2
      hilbert(n, &p0, 12, 90);
      viewWindow();    closeWindow();
}

void hilbert(int n, POSI *p1, float length, float a)
{
      if (0 == n)         return;
      ga += a; hilbert(n-1, p1, length, -a);
      TurtleDraw(p1, ga, length, black);
      ga -= a; hilbert(n-1, p1, length, a);
      TurtleDraw(p1, ga, length, blue);
      hilbert(n-1, p1, length, a); ga -= a;
      TurtleDraw(p1, ga, length, red);
      hilbert(n-1, p1, length, -a); ga += a;
}

void DrawLine(const POSI p1, const POSI p2, shapeColor c)
{
      lineShape Line(VSx(p1.x), VSy(p1.y), VSx(p2.x), VSy(p2.y), c);
      Line.draw();
}

void TurtleDraw(POSI *p1, const float a, const float L, shapeColor c)
{
      static POSI p2;
      p2.x = p1->x + L*COS(a);     p2.y = p1->y + L*SIN(a);
      DrawLine(*p1, p2, c);    *p1 = p2;
      delayWindow(0.1);
}
```

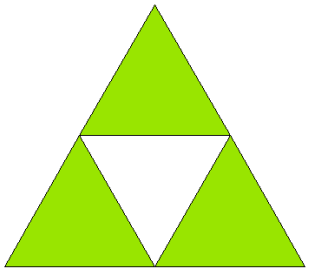
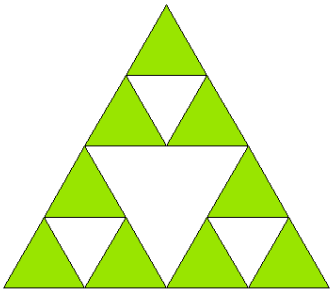
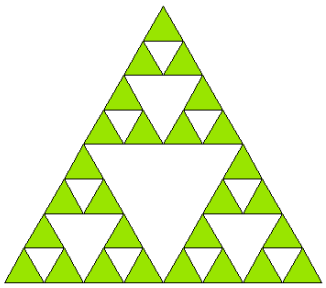
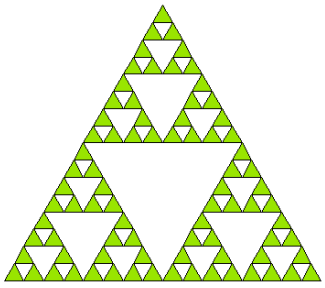
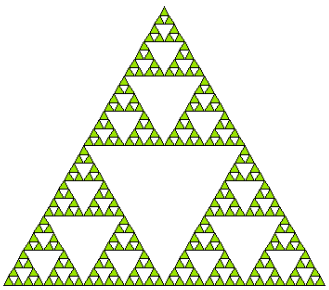
```
void setWindow(float x1, float y1, float x2, float y2)
{
      WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
      openWindow();
      mfx = 10./(WS.x2-WS.x1);    mfy = 8./(WS.y2-WS.y1);
      mfx = mfy = (mfx < mfy ? mfx : mfy);
}
```



範例 5: **Sierpinski gasket (船帆)**

這個有名的三角形，製作的方式如圖，首先，給定一正三角形，取各邊中點，挖掉中間那塊正三角形(其邊界需留下)，剩下三個相同的正三角形，接下來對剩下的每個三角形做同樣的步驟，挖掉其各邊中點連線所成正三角形，重複此步驟無窮次，即形成 Sierpinski Triangle。

| | |
|---|---|
| 拿一個正三角形假設它的面積是 1 |  |

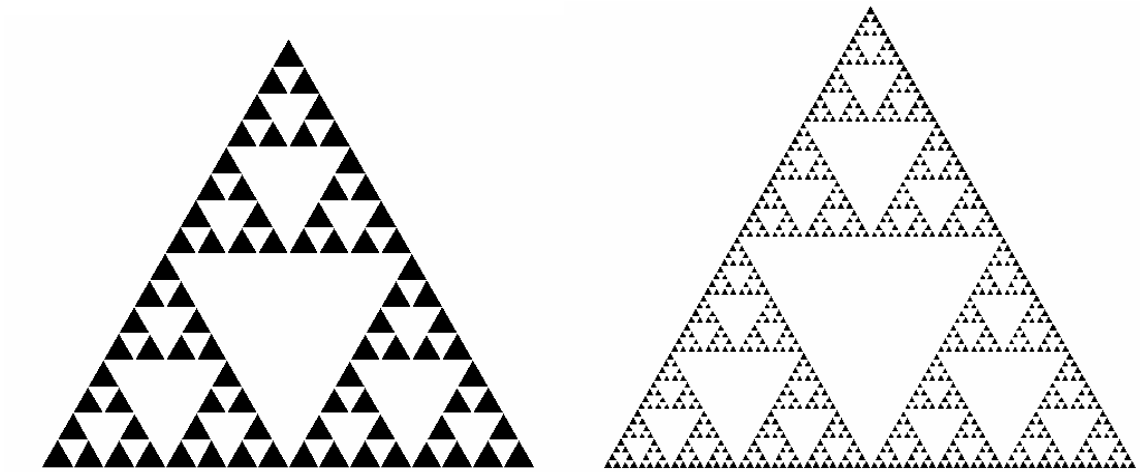
| | |
|---|---|
| 挖掉中間 1/4，這時候的面積是 3/4 |  |
| 對每一片剩下的三角形，挖掉它中間 1/4，這時候的面積是 9/16 |  |
| 挖第三遍，現在的面積是 $(3/4)^3$ |  |
| 挖第四遍，現在的面積是 $(3/4)^4$ |  |
| 挖第五遍，現在的面積是 $(3/4)^5$ |  |

| | |
|---|---|
| | 再挖就看不到了・・・ |

Sierpinski Triangle 有點類似 Cantor Set，其面積為 0，但它卻有與平面一樣的無窮多個點。

```
// gasket.cpp
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

#define VSx(x)      (((x)-WS.x1)*mfx)
#define VSy(y)      ((WS.y2-(y))*mfy)

void Sierpinski(int order, POSI p0, POSI p1, POSI p2);
void drawTriangle(POSI p0, POSI p1, POSI p2, EZDCOLORVAL color);
void setWindow(float x1, float y1, float x2, float y2);
float mfx, mfy;

void main()
{
      int order = 7;
      POSI p0 = {0, 0}, p1={1, 1.732}, p2 = {2, 0};
      setWindow(-0.5, -0.5, 2, 2);    // x1, y1, x2, y2
      drawTriangle(p0, p1, p2, ezdBlack);
      Sierpinski(order, p0, p1, p2);
      viewWindow();    closeWindow();
}

void Sierpinski(int order, POSI p0, POSI p1, POSI p2)
{
      POSI p3, p4, p5;
      if (1 == order)    return ;
      p3.x = (p0.x + p1.x)/2., p3.y = (p0.y + p1.y)/2.;
```

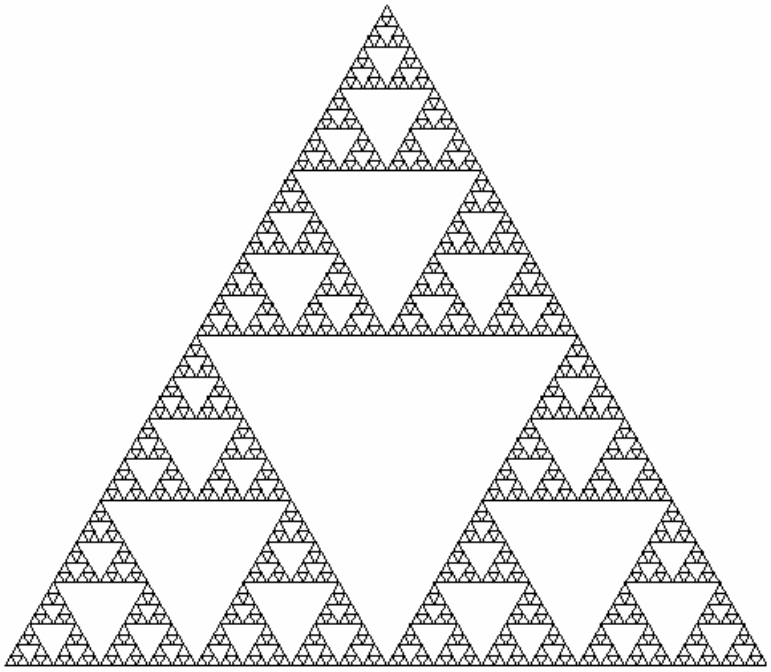
```
    p4.x = (p1.x + p2.x)/2., p4.y = (p1.y + p2.y)/2.;
    p5.x = (p0.x + p2.x)/2., p5.y = (p0.y + p2.y)/2.;
    drawTriangle(p3, p4, p5, ezdWhite);
    Sierpinski(order-1, p0, p3, p5);
    Sierpinski(order-1, p3, p1, p4);
    Sierpinski(order-1, p5, p4, p2);
}

void setWindow(float x1, float y1, float x2, float y2)
{
    WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
    openWindow();
    mfx = 10./(WS.x2-WS.x1);
    mfy = 8./(WS.y2-WS.y1);
    mfx = mfy = (mfx < mfy? mfx: mfy);
}

void drawTriangle(POSI p0, POSI p1, POSI p2, EZDCOLORVAL color)
{
    const double X3[3] = {VSx(p0.x), VSx(p1.x), VSx(p2.x)};
    const double Y3[3] = {VSy(p0.y), VSy(p1.y), VSy(p2.y)};
    ezdSetColor(color);
    ezdDrawPolygon(3, &X3[0], &Y3[0]);
    delayWindow(0.2);
}
```



```
// Sierpinski.cpp
#include <math.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_linesh.h"     // lineShape class
```

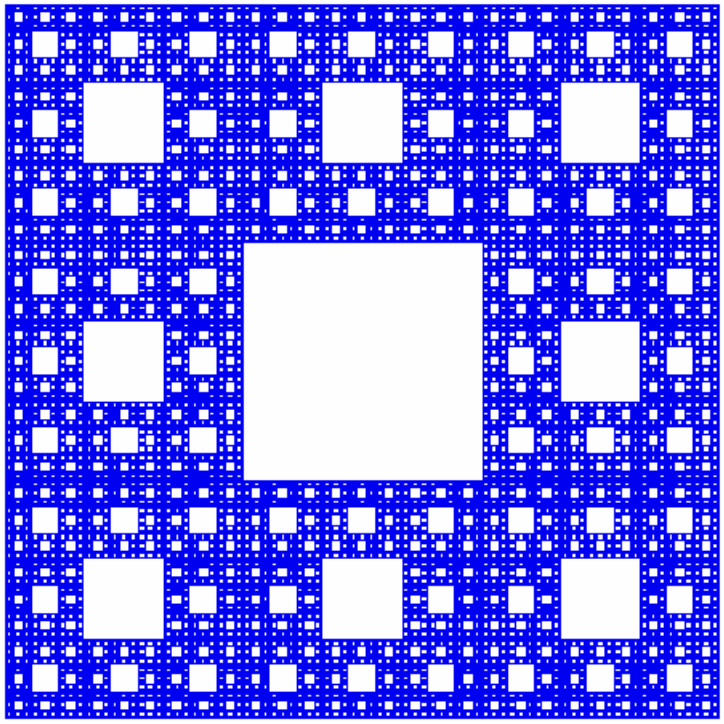
```
struct POSI { double x, y; };
struct WorkingSpace { float x1, y1, x2, y2; } WS;

#define VSx(x)      (((x)-WS.x1)*mfx)
#define VSy(y)      ((WS.y2-(y))*mfy)

void Sierpinski(int order, POSI p0, POSI p1, POSI p2);
void DrawLine(POSI, POSI, shapeColor);
void drawTriangle(POSI p0, POSI p1, POSI p2);
void setWindow(float x1, float y1, float x2, float y2);
float mfx, mfy;

void main()
{
      int order = 7;
      POSI p0 = {0, 0}, p1={1, 1.732}, p2 = {2, 0};
      setWindow(-0.5, -0.5, 2, 2);    // x1, y1, x2, y2
      drawTriangle(p0, p1, p2);
      Sierpinski(order, p0, p1, p2);
      viewWindow();    closeWindow();
}

void Sierpinski(int order, POSI p0, POSI p1, POSI p2)
{
      POSI p3, p4, p5;
      if (1 == order)    return ;
      p3.x = (p0.x + p1.x)/2., p3.y = (p0.y + p1.y)/2.;
      p4.x = (p1.x + p2.x)/2., p4.y = (p1.y + p2.y)/2.;
      p5.x = (p0.x + p2.x)/2., p5.y = (p0.y + p2.y)/2.;
      drawTriangle(p3, p4, p5);
      Sierpinski(order-1, p0, p3, p5);
      Sierpinski(order-1, p3, p1, p4);
      Sierpinski(order-1, p5, p4, p2);
}

void DrawLine(POSI p1, POSI p2, shapeColor c)
{
      lineShape Line(VSx(p1.x), VSy(p1.y),VSx(p2.x), VSy(p2.y), c) ;
      Line.draw();
```

```
}

void setWindow(float x1, float y1, float x2, float y2)
{
	WS.x1 = x1; WS.y1 = y1; WS.x2 = x2; WS.y2 = y2;
	openWindow();
	mfx = 10./(WS.x2-WS.x1);
	mfy = 8./(WS.y2-WS.y1);
	mfx = mfy = (mfx < mfy? mfx: mfy);
}

void drawTriangle(POSI p0, POSI p1, POSI p2)
{
	DrawLine(p0, p1, black);
	DrawLine(p1, p2, black);
	DrawLine(p0, p2, black);
	delayWindow(0.2);
}
```



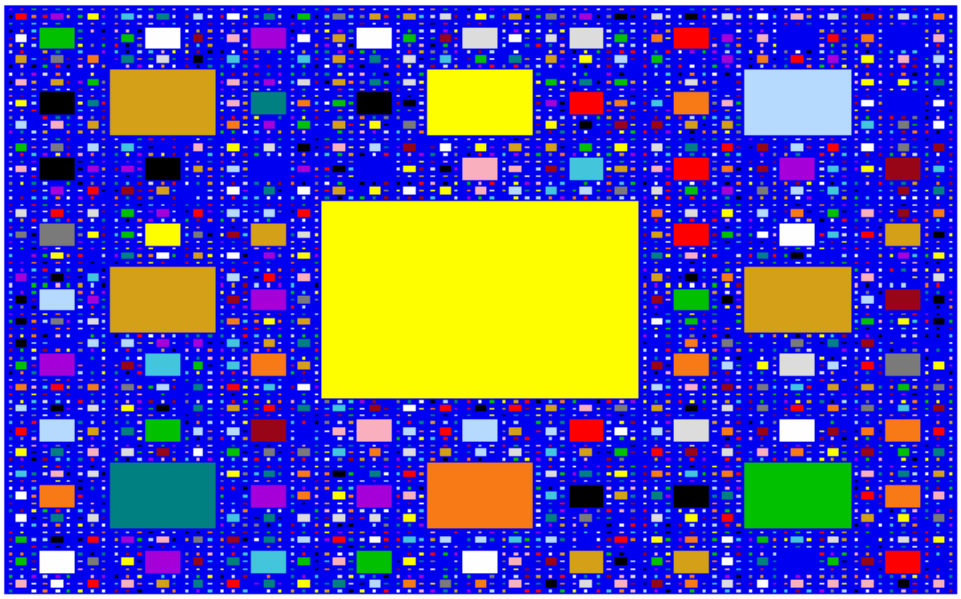
```
// Sierpinski2.cpp
#include <math.h>
#include "d_draw.h"	// openWindow(); viewWindow();	closeWindow();
#include "d_rectsh.h"	// lineShape class

struct POSI { double x, y; };
```

```
void Sierpinski(int order, POSI base, float length);

void main()
{
	int order = 5;
	POSI base = {1, 1};
	float length=6;
	openWindow();
	rectShape square(base.x, base.y, length, length, blue);
	square.draw();
	delayWindow(0.2);

	Sierpinski(order, base, length);
	viewWindow();	closeWindow();
}

void Sierpinski(int order, POSI base, float length)
{
	POSI P[8];
	P[0] = base;
	if (0 == order)	return ;
	length /= 3.;
	base.x += length,	base.y += length;
	rectShape square(base.x, base.y, length, length, white);
	square.draw();
//	delayWindow(0.2);
	P[2].x = P[1].x = P[0].x;
	P[7].x = P[3].x = P[0].x + length;
	P[6].x = P[5].x = P[4].x = P[3].x + length;
	P[7].y = P[6].y = P[0].y;
	P[5].y = P[1].y = P[0].y + length;
	P[4].y = P[3].y = P[2].y = P[1].y + length;
	for (int i=0; i < 8; i++)	Sierpinski(order-1, P[i], length);
}
```



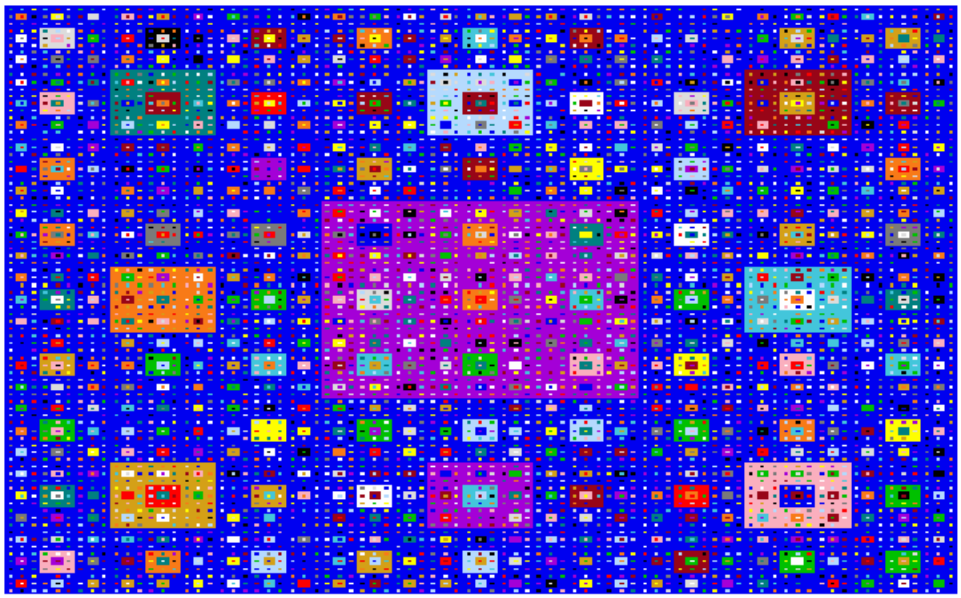
```
// Sierpinski3.cpp
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_rectsh.h"     // lineShape class
#define ROUND(x)             ((x)>0 ? (int)((x)+0.5): -(int)(0.5-(x)))
struct POSI { double x, y; };
void Sierpinski(int order, POSI base, float length, float width);
int RandomInt(int lower, int upper);
drawing_color ValueToColor(float value, float min, float max);

void main()
{
      int order = 5;
      POSI base = {1, 1};
      float length=8;
      float width = 0.618*length;
      srand(time(NULL)); /* use a random seed */
      openWindow();
      rectShape square(base.x, base.y, length, width, blue);
      square.draw();
      delayWindow(0.2);
```

```
	Sierpinski(order, base, length, width);
	viewWindow();	closeWindow();
}

void Sierpinski(int order, POSI base, float length, float width)
{
	unsigned c1 = RandomInt(0, 255);
	shapeColor color= ValueToColor(c1, 0, 16);

	POSI P[8];
	P[0] = base;
	if (0 == order)	return ;
	length /= 3.,	width /= 3.;
	base.x += length,	base.y += width;
	rectShape square(base.x, base.y, length, width, color);
	square.draw();
//	delayWindow(0.2);
	P[2].x = P[1].x = P[0].x;
	P[7].x = P[3].x = P[0].x + length;
	P[6].x = P[5].x = P[4].x = P[3].x + length;
	P[7].y = P[6].y = P[0].y;
	P[5].y = P[1].y = P[0].y + width;
	P[4].y = P[3].y = P[2].y = P[1].y + width;
	for (int i=0; i < 8; i++)	Sierpinski(order-1, P[i], length, width);
}

int RandomInt(int low, int high)
{
	if (low > high)		printf("RandomInt: low cannot be greater than high.");
	return (high - low + 1) * ((double) rand() / RAND_MAX) + low;
}

drawing_color ValueToColor(float value, float min, float max)
{
	unsigned color = 15-ROUND((value-min)*15/(max-min))%16;
	return(drawing_color(color));
}
```



```
// Sierpinski5.cpp
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "d_draw.h"               // openWindow(); viewWindow();    closeWindow();
#include "d_rectsh.h"     // lineShape class
#define ROUND(x)              ((x)>0 ? (int)((x)+0.5): -(int)(0.5-(x)))
struct POSI { double x, y; };
void Sierpinski(int order, POSI base, float length, float width);
int RandomInt(int lower, int upper);
drawing_color ValueToColor(float value, float min, float max);

void main()
{
        int order = 5;
        POSI base = {1, 1};
        float length=8;
        float width = 0.618*length;
        srand(time(NULL)); /* use a random seed */
        openWindow();
        rectShape square(base.x, base.y, length, width, blue);
        square.draw();
        delayWindow(0.2);

        Sierpinski(order, base, length, width);
```

```
	viewWindow();	closeWindow();
}

void Sierpinski(int order, POSI base, float length, float width)
{
	unsigned c1 = RandomInt(0, 255);
	shapeColor color= ValueToColor(c1, 0, 16);

	POSI P[9];
	P[0] = base;
	if (0 == order)	return ;
	length /= 3.,	width /= 3.;
	base.x += length,	base.y += width;
	rectShape square(base.x, base.y, length, width, color);
	square.draw();
//	delayWindow(0.2);
	P[2].x = P[1].x = P[0].x;
	P[7].x = P[3].x = P[0].x + length;
	P[6].x = P[5].x = P[4].x = P[3].x + length;
	P[7].y = P[6].y = P[0].y;
	P[5].y = P[1].y = P[0].y + width;
	P[4].y = P[3].y = P[2].y = P[1].y + width;
	P[8].x = P[0].x + length,	P[8].y = P[0].y + width;
	for (int i=0; i < 9; i++)	Sierpinski(order-1, P[i], length, width);
}

int RandomInt(int low, int high)
{
	if (low > high)	printf("RandomInt: low cannot be greater than high.");
	return (high - low + 1) * ((double) rand() / RAND_MAX) + low;
}

drawing_color ValueToColor(float value, float min, float max)
{
	unsigned color = 15-ROUND((value-min)*15/(max-min))%16;
	return(drawing_color(color));
}
```



```
// Sierpinski4.cpp
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "d_draw.h"              // openWindow(); viewWindow();    closeWindow();
#include "d_rectsh.h"     // lineShape class
#define ROUND(x)             ((x)>0 ? (int)((x)+0.5): -(int)(0.5-(x)))
struct POSI { double x, y; };
void Sierpinski(int order, POSI base, float length, float width);
int RandomInt(int lower, int upper);
drawing_color ValueToColor(float value, float min, float max);

void main()
{
      int order = 5;
      POSI base = {1, 1};
      float length=8;
      float width = 0.618*length;
      srand(time(NULL)); /* use a random seed */
      openWindow();
      rectShape square(base.x, base.y, length, width, blue);
      square.draw();
      delayWindow(0.2);

      Sierpinski(order, base, length, width);
```

```
	viewWindow();	closeWindow();
}

void Sierpinski(int order, POSI base, float length, float width)
{
	unsigned c1 = RandomInt(0, 255);
	float f = 0.5+ RandomInt(40, 70)/100.;
	float g = 0.5+ RandomInt(20, 55)/100.;
	shapeColor color= ValueToColor(c1, 0, 16);

	POSI P[8];
	P[0] = base;
	if (0 == order)	return ;
	length /= 3.,	width /= 3.;
	base.x += length,	base.y += width;
	rectShape square(base.x*g, base.y*g, length*f, width*f, color);
	square.draw();
//	delayWindow(0.2);
	P[2].x = P[1].x = P[0].x;
	P[7].x = P[3].x = P[0].x + length;
	P[6].x = P[5].x = P[4].x = P[3].x + length;
	P[7].y = P[6].y = P[0].y;
	P[5].y = P[1].y = P[0].y + width;
	P[4].y = P[3].y = P[2].y = P[1].y + width;
	for (int i=0; i < 8; i++)	Sierpinski(order-1, P[i], length, width);
}

int RandomInt(int low, int high)
{
	if (low > high)	printf("RandomInt: low cannot be greater than high.");
	return (high - low + 1) * ((double) rand() / RAND_MAX) + low;
}

drawing_color ValueToColor(float value, float min, float max)
{
	unsigned color = 15-ROUND((value-min)*15/(max-min))%16;
	return(drawing_color(color));
}
```