CHAPTER : III
FRACTAL GRAPHICS
&
RECURSIVE PROGRAMMING

C H A P T E R -III :: FRACTAL GRAPHICS AND RECURSIVE PROGRAMMING ::

3.1 CREATING NATURAL PATTERNS :

Suppose we want to draw a fine scenery consisting of many thousand blades of grass. One approach is to write a program that produces a single blade of grass and call the program at different locations of screen, where you want to produce the blades of grass. (Note that the commands in the single blade program must be relative rather than absolute to make the duplication successful). So a lawn will be produced surely.But the lawn would look utterly unreal, because in nature no two grass blades are alike. So a regular art of the geometrical shapes with which we have dealt in chapter II will not help here. A kind of irregular art is, required to draw natural elements such as landscapes. To introduce randomness in natural elements, one can, for example, write a separate program to produce different grass blades with their varying height and orientation. The result will be an unmanageable large data base. The compiling time for the program will also be large. One method to simulate the randomness of natural objects is the technique of 'fractals'. This techique is devised by Benoît Mandelbrot of IBM and is popularly known as 'Mandelbrot Graphics,'[2,1]

Another major tool to imitate natural patterns is recursion[3,1]. In this chapter programs are developed that illustrate recursion and fractals.

3.2 RECURSIVE PROGRAMMING :

A recurisive procedure is one which calls itsef . Certain type of problems such as choosing the next move in a computer chess program, can best be solved with a recursive procedure. Recursion technique has a great value in the field of 'Artificial Intelligence". By definition , A.I. involves improving the computer performance in the fields where human is better. So A.I. always tries to copy human problem solving activities. Many times human problem solving activities involve recursion. This is a very classic 'divide and conquer' approach of problem solving. One example of recursive problem solving by human is quoted in reference[3]. It is the problem of planning a route for walking through London from Trafalgar square to the British Museum. One way of solving this problem might be to pick an intermediate landmark such as Covent Garden, and break the original problem down into the problem of getting from Trafalgar Square to Covent Garden and from Covent Garden to the British Museum. Extreme care has to be taken while designing recursive procedure. Recursion must be terminated at a level of detail appropriate both to the problem being solved and to the size of the computer available.

To become a successful recursive programmer, you must recognize when a problem can be broken down into easier problems. There must be some tests, which will tell whether the problem is further divisible or not. The exact sequence of operations is not important in a recursive subroutine.

3.3 RECURSION AND COMPLEXITY :

There are two types of complexities, extensive complexity Extensive complexity includes repeating the same designs or again and again order and over again and generating forms of different size, position and even shape from the same function and it does appear to represent the way reality is structured. In intensive complexity the parts reflect the whole and the details are produced by the same function which is used to organize the overall structure. The essence of intensive complexity is nesting a function within the same function, the sequence of nesting reflects successive levels of detail. The principle behind drawing landscapes is the complexity results from the repeated application of simpler processes and structures.

3.4 RECURSIVE SQUARES :

There are many complex patterns and curves that can easily be described recursively. So recursion is an useful tool in computer graphics and computer generated art (computer aided drawing). The principle of self-similarity is used in drawing

the recursive shape i.e. the same shape is repeated at different scale. Here a program is developed that creates a pattern of recursive squares ..

The pattern consists of a square, together with a recursive half-size copy of the complete pattern centred on each corner of the main square.

3.4.1 DESCRIPTION OF THE PROGRAM :

At the start of the program, the required files are included. Then there is a declaration of a function censqr which draws a square.

The main module first sets the drive argument to the costant DETECT. It sets the mode to 1 and then it calls initgraph. The initgraph will automatically determine the type of graphics adapter installed and will load the appropriate driver. Then the censqr function is called to draw the centre square. Before shuting down the graphics system, the main module also checks whether key 'p' is pressed, if 'yes' ! then it calls the 'ghardpict' function which is a prototype in "gpmt.h" .

Last part of the program is 'square' module. Basically a rectangle is defined of which size depends upon reduction factor r. It takes the coordinates of the square defined in the main module and reduces them by r. Delay is set so that

one can observe the pattern while building up. The stopping condition is triggered by the statement (r < 20) and the program is terminated without drawing square. Next part of the module is calculation of proper locations where, the copy of the square is to be placed. Then a copy of reduced size square is placed at each corner of main square and the process continues.

Refer to listing of program on page number 80

3.5 SNOWFLAKES :

The algorithm for constructing continuous, non-rectifiable, recursive curves was first presented by Helge Von Koch[1] in 1904. He described the process in terms of initiators generator and cascade. The overall process of drawing is referred to as Koch recursion. The significance of Koch recursion is that the same recursive process, can be used to generate a variety of curves different from Koch's original curve.

In the Koch or Snowflake curve the initiation is a star of David which is shown in figure number 1 . Star of David consists of two equilateral triangle drawn in a fashion shown in figure 1 . The generator is a similar smaller triangle which is recursively applied to each of the sides of the previous pattern.

Many interesting variations can be tried with this pattern e.g. you can vary the reduction factor that determines the size of smaller flakes. By using different reduction factor

for the central flake also gives effective picture. One can also use two different reduction factors, one for the small flakes on the points of one of the triangles, and the other for the small flakes on the points of the other triangle. Here, in this work we have kept restriction on number of patterns and we have defined it as order of the shape.

3.5.1 PROGRAM DESCRIPTION :

The function 'David' is declared after include part of the program, which draws the basic star of david pattern. Another function 'Rekoch' is also declared. Rekoch function draws the primitive David pattern at a reduced scale at each corner of the David pattern drawn in earlier iteration . The degree to which the recursive drawing continues is denoted by 'k'.

The main module of the program asks the user about the degree of the snowflakes. Then it initializes graphics mode. Rekoch function is called with x = 350 ,y =150 and r =150 to draw the initial star of David pattern. Here r denotes the reduction factor i.e. in second iteration the basic star of David pattern will be reduced by this much amount. Rest part of the main module is to look for key 'p' or key 'q'. With key 'p' you can take the hardcopy and with 'q' you can exit the program. By pressing any other key you can continue with the pattern.

Description of David Function :

The task of David function is to draw the basic star of David pattern. This star is drawn with linesegements with proper endpoints. This function in its first iteration takes the value of x , y and r as x,y and r (i.e. 320,150,150) and in successive it erations x,y and r decrease by the amount of reduction factor.

Description of Rekoch Function :

Rekoch function first calls the David function. Then there is a delay so that one could observe the pattern while building up. Then the stopping condition is checked. If reduction factor is less than 'degree, the program is terminated. Rest of the statements in this module are the calculation of the location of corners. Then by proper coordinate pairing the same Rekoch function is recursively called to place the reduced copies of the star at each corner.

Refer to listing of program on page number 81.

3.6 FRACTAL GEOMETRY :

In Latin language fractus means irregular. So the fractal geometry deals with the structures which are self similar in nature but irregular in form. Typical objects of this form are coastlines, rock formations and trees. Normally such natural forms have the property best summarized by Mandelbrot (1982) as follows :"When each piece of the shape is geometrically similar to the whole, both the shape and the

cascade that generates it are called ' Self Similar' (quoted in reference 1).

A deep study of the structure of a coastline was done by Mandelbrot[1] (at that time he was a research fellow at IBM). He concluded with three points as follows :

1) The length of the coastline varies with scale. As scale becomes finer and finer, more and more detailed features, headland, a cosy, sheltered recess and crannies emerged which increase the length of the coastline. So Mandelbrot concluded that the length of coastline might be considered "Infinite or rather, undefinable".

2) Curves like coastlines exhibit an irregularily which is not smooth, hence can't be studied using the calculus. Though continuous the curves are non-rectifiable (i.e. they cannot be differentiated).

3) So the basis of the structure of coastline is irregularity and infinite length. But when the patterns are drawn they seem to span space. Now the curves are one dimensional and space is two dimensional. How the shape can be both one-and two-dimensional ?

The objects like coastline, seem to lie between dimensions. They are not bounded by scale but are referred to as scaling. Coastline seem to be more than one dimensional but not exactly two. So the dimensions of these typical objects

appear to be fractional or nonintegral. Such objects are called "fractals" by Mandelbrot. Study of fractals has initiated a new branch of mathematics of major relevance to computer graphics called as "fractal geometry" by Batty .

By-the-way, Fractal Graphics introduced new words to English language viz.

"FRACTOPHILIC " : Meaning a lover of fractals

"FRACTALICIOUS" : Meaning a delicious fractal.

3.7 APPLICATIONS OF FRACTALS[4]:

Fractals have many important applications[4] in chaos theory and materials research. Certain fractal systems implemented on computer closely resemble grains and dust particles. Lucas used fractal logic to create out the world of landscapes for the movie "The Return of the Jedi" After observing the simulation of fractal logic on screen some pathologists believe that modelling fractal systems on a computer may help in understanding how cancers develop.

With the aid of a simple fractal program, a computer can be converted into a kind of microscope for viewing the boundary of the Mandelbrot set. Zooming in one sees an infinite regress of detail that astonishes us with its complexity. A modified fractal technique, computer bimorphing is used in the recent record breaking movie "Jurasic park".

Now-a-days, fractals are being used to produce still paintings and sculptures as well as animation. This raises the interesting question of fractal landscapes should be judged on the same criteria as the works of conventional landscape painters. Paul Brown[2] thinks that upto a point they should but other artists disagree strongly.

3.8 SI ERPINSKI CURVES :

The curve is developed by Waraclaw Sierpinski.[3,5] Sirepinski curves exists from order 0 to order 4. It is convenient to define a Sirepinski curve of order 0 which consists of a diamond (or a square rotated through 45^{o}).

The characteristics of the Sirepinski curve is that it can be drawn as continuous line, without lifting pencil from paper. There are two ways of drawing the curve.

First method of drawing is an easy one. Here we must recognize that the Sierpinski curve of order 1 consists of four order 0 curves "Joined" at the centre. The order 2 curve consists of four order 1 curves joined at the centre. So generalizing, an order N curve consists of four order N-1 curves joined at the centre when four subcurves are joined. We have to delete four diagonal lines from the subcuves and join the subcurves with two horizontal and vertical lines.

Another appraoch of drawing Sierpinski curves is suggested by Wirth (the inventor of programming language PASCAL). In this approach one can draw the curve as a continuous line. If you want to take hardcopy of the Sierpinski curves, you have to implement this approach (because on printer you can't delete lines).

Any Sierpinski curve of order N can be divided into four components connected at the corners, a left component, a top componet, a right component, a bottom component.

Now an order N component is made up of a sequence of order N-1 components joined in a well-defined way. e.g. A left component of order N consists of :

a left component of order N-1.

a diagonal line.

a top component of order N-1.

a vertical line.

a bottom component of order N-1.

a diagonal line.

a left component of order N-1.

This principle of drawing is shown in figure 1. The same principle is worked out in the program listings on page number 83. Refer to listing of program on page number 83.

3.10 DIRECTIONS FOR EXTENSION :

Colour has the power to attract attention and evoke complex feelings. Our world is a colourful world. Coolour gives new dimensions to the output pattern implemented in this chapter. Unfortunately we are using HGA for better resolution purposes . But one can implement these patterns with colours. If different colours are used at adjacent levels, successively smaller copies of the shape will appear in different colours over the larger copies that have already been plotted in another colour. So one can introduce the colour intelligence in the recursive square, Snowflakes and Sierpinski curves.

Here we have implemented the Sierpinski curve as a representative element of space filling curves (i.e. nonrandom fractals). Using the same principle one can implement Peano type curves, W curves, C curves and dragon curve.

The first variant on the scheme of Koch recursion is the construction of Koch landscape or a Koch forest. Here the star of David pattern is not used as initiator as a whole. But a horizontal line is drawn which provides a base for the forest.

The second possible variant of the Koch recursion involves sweeping around the plane and generating a design in which the Koch forest results as the complement of the two dimensional sweep. The result is Peano-Cesaro triangle sweep.

Mountains and coastlines are the structures which are moduled throughout the ages and have multiple influences. There is no way to record and reconstitute the multiple influences. So for modelling such objects we have to turn to the concept of random fractals. Their details and irregularity can't be described in any deterministic way. One can modify the Koch recursion program by making the displacement random but witnin certain limits of the midpoint interms of its orientation and magnitude.

One more direction of extension in recursive programming is the implementation of recursively generated trees. It is truly said that,

"A tree is a branch with a tree on the end of it'

A simple subroutine that draws a branch of tree can be called recursively to draw a full tree with many branches. With the facility of random number generation of many languages, randomness can be introduced in tree drawing.

You will come to know many more directions of extension of the work on fractals if you just go through the twenty-two unique fractal programs found on the diskette into the back cover of Judd Robin's book "Fun with fractals".[6] This is a software of about 3 Megabytes. The book also explains how the program works and how to use them for greatest enjoyment. It also gives to create infinite possibilities of fractal programming.

```
/*ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ
  Æ PROGRAM       DEMONSTRATING      FRACTAL          Æ
  Æ SIMPLE  RECURSIVE  SQUARES  ARE  IMPLEMENTED.     Æ
  Æ DATE     : 29-02-1994                             Æ
  ÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆÆ*/

     */
     --------------------^^^^^------------------------------


     #include <stdio.h>
     #include <graphics.h>
     #include <stdlib.h>
     #include <conio.h>
     #include <dos.h>
     #include "gprnt.h"


 void censqr(int,int,int);          /*  draws the square */

  main()
    æ
     int drive = DETECT;
        int mode = 1;

        initgraph(&drive,&mode,"");
        censqr(320,220,100);
        if(getch() == 'p') ghardpict(50,50,600,300);
        closegraph();

    å

         void censqr(int xc,int yc,int r)
                      /*  r reduction factor */
    æ
    int x1,y1,x2,y2;
    rectangle(xc-r*.5,(yc-r*.5)*.7,xc+r*.5,(yc+r*.5)*.7);
             /* draws the rectangle */
    delay(100);
       if(r < 20) return;
       x1 = xc - r/2;
       y1 = yc - r/2;
       x2 = xc + r/2;
       y2 = yc + r/2;


        censqr(x1,y1,r/2);
        censqr(x2,y1,r/2);
        censqr(x1,y2,r/2);
        censqr(x2,y2,r/2);
        return;
    å
```

```
/* PROGRAM  DEMONSTRATING  FRACTAL

 PROGRAM   DEMONSTRATING  SIRPINSKI  CURVES
 DATE      : 29-06-1994    */

/* X?0???? PROGRAM  STARTS  HERE  ++++++++++*/

#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "gprnt.h"
#include <math.h>
void   spinki(int,int,int,int);
      /*  draws the square */
void  diamond(int,int,int);

main()
  æ
    int drive = DETECT;
    int mode = 1;

    initgraph(&drive,&mode,"");
    spinki(320,140,5,4);
    if(getch() == 'p')
           ghardpict(50,50,600,300);
    closegraph();

 å



   void   spinki(int  xc,int  yc,int   h,int   n)
        /*  r reduction factor */
 æ
    int x1,y1,x2,y2,x3,y3,n1;
    setcolor(1);
    delay(100);

    if(n==0) æ
        diamond(xc,yc,h);
        return;
 å


    x1 = xc - (pow(2,n)) * h ;
    y1 = yc - (pow(2,n)) * h ;
    n1 = n - 1;

    spinki(x1,y1,h,n1);

    x1 = xc + (pow(2,n)) * h ;
    y1 = yc - (pow(2,n)) * h ;
    n1 = n - 1;
```

```
    spinki(x1,y1,h,n1);

    x1 = xc + (pow(2,n)) * h ;
    y1 = yc + (pow(2,n)) * h ;
    n1 = n - 1;

    spinki(x1,y1,h,n1);

    x1 = xc - (pow(2,n)) * h ;
    y1 = yc + (pow(2,n)) * h ;
    n1 = n - 1;

    spinki(x1,y1,h,n1);

    setcolor(0);
   line(xc-2*h,yc-h,xc-h,yc-2*h);
   line(xc+h,yc-2*h,xc+2*h,yc-h);
   line(xc+2*h,yc+h,xc+h,yc+2*h);
   line(xc-h,yc+2*h,xc-2*h,yc+h);

   setcolor(1);
   line(xc-h,yc-2*h,xc+h,yc-2*h);
   line(xc+2*h,yc-h,xc+2*h,yc+h);
   line(xc+h,yc+2*h,xc-h,yc+2*h);
   line(xc-2*h,yc+h,xc-2*h,yc-h);


   return;
   ä

void  diamond(int x,int y,int h)
   æ
     moveto(x-h, y);
     linerel(h,-h);
     linerel(h,h);
     linerel(-h,h);
     linerel(-h,-h);
   ä
```

```
/* PROGRAM DEMONSTRATING KOCH RECURSION  KOCH.C */
/* A PRIMITIVE DAVID STAR IS RECURSIVELY PLACED */
/* ON ITSELF.                                   */
/*  DATE :04-07-1994                            */

/* PROGRAM STARTS HERE          */

/*  __________ INCLUDE MODULE _________________*/
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "gprnt.h"


void  reckoch(int,int,int);
              /*  draws the square */
void david(int , int , int );

int k;

main()
{
 int drive = DETECT;
 int mode = 1;
 char op;

   do{

     printf("give degre : ");
     scanf("%d",&k);
     initgraph(&drive,&mode,"");
     reckoch(320,150,150);
     op = getch();
     if(op == 'p') ghardpict(50,50,600,300);
     closegraph();
     }while(op != 'q');

          }


     void reckoch(int xc,int yc,int r)
             /*  r reduction factor */
     {
       int x1,x2,y1,y2,y3,y4,x3;

       david(xc,yc,r);
             /* draw the rectangle */
       delay(1000);
       if(r<k) return;
       x1 = xc - r/2;
       y2 = yc - r/4;
       x3 = xc + r/2;
```

```
 y3 = yc + r/4;
 x2 = xc;
 y1 = yc - r/2;
 y4 = yc + r/2;


reckoch(x1,y2,r/2);
reckoch(x2,y1,r/2);
reckoch(x3,y2,r/2);
reckoch(x3,y3,r/2);
reckoch(x2,y4,r/2);
reckoch(x1,y3,r/2);

return;
  ä


void david(int x, int y, int r)
 æ
   line(x - r/2,y - r/4,x + r/2,y - r/4);
   line(x - r/2,y + r/4,x + r/2,y + r/4);
   line(x - r/2,y - r/4,x,y + r/2);
   line(x + r/2,y - r/4,x,y + r/2);
   line(x - r/2,y + r/4,x,y - r/2);
   line(x + r/2,y + r/4,x,y - r/2);
  ä
```

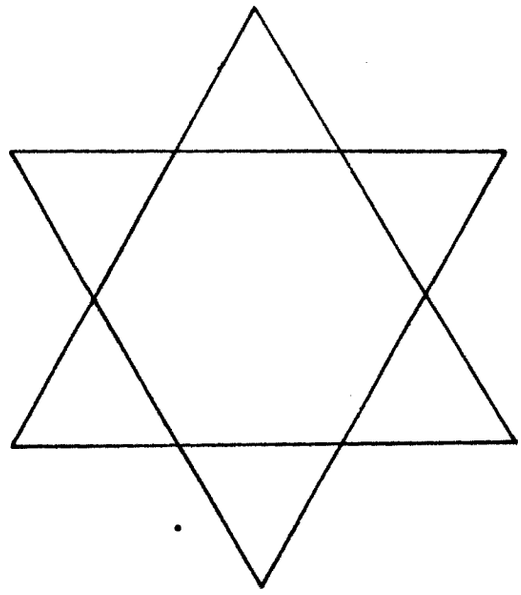


Fig.1 : Star of David Pattern. It's Recursion Gives Snowflakes

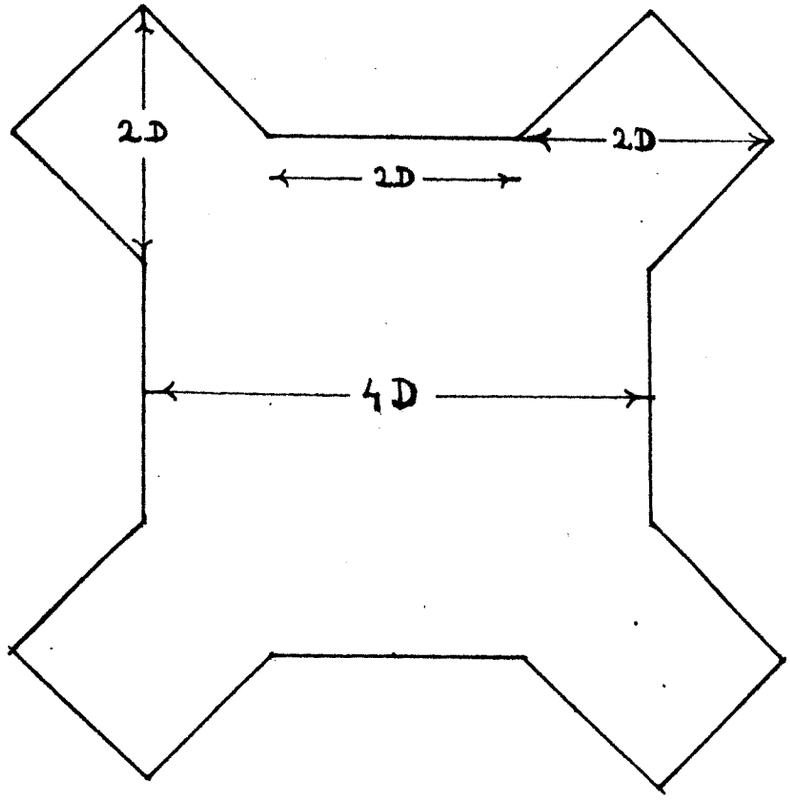
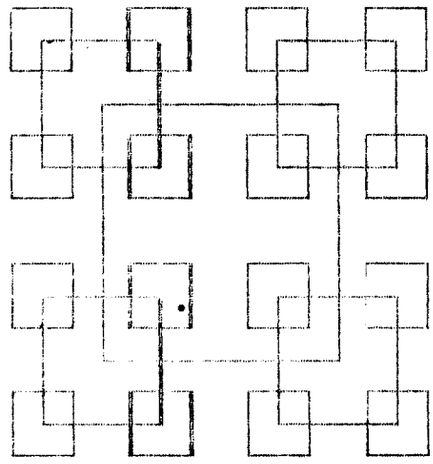
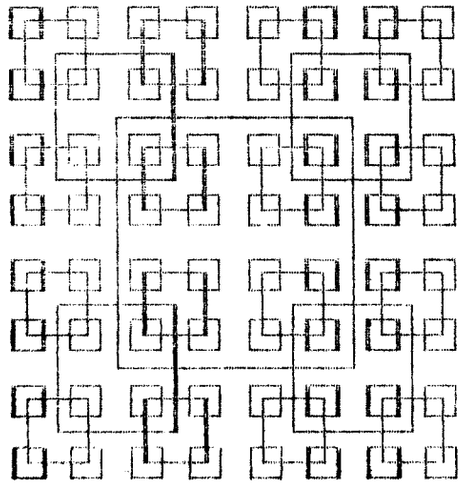


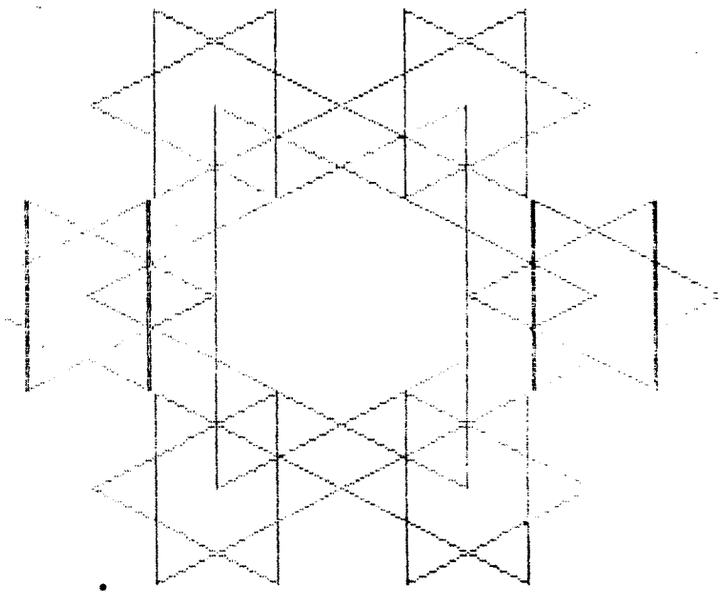
Fig.2 : Principle of Drawing Sierpinski Curve



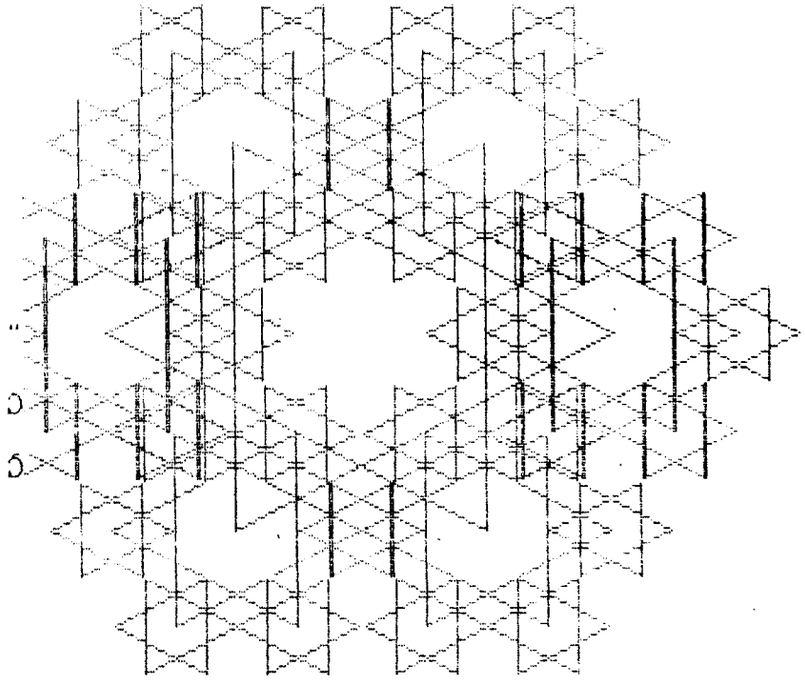
Recursive squares for r < 50



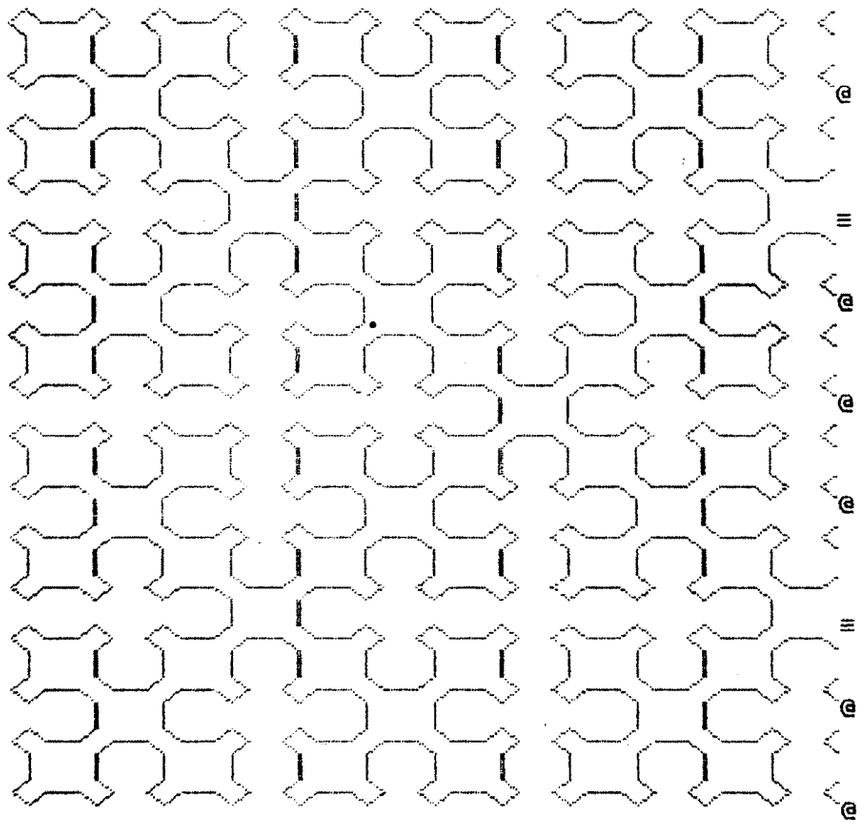
Recursive square for r < 20
Output of program for recursive squares



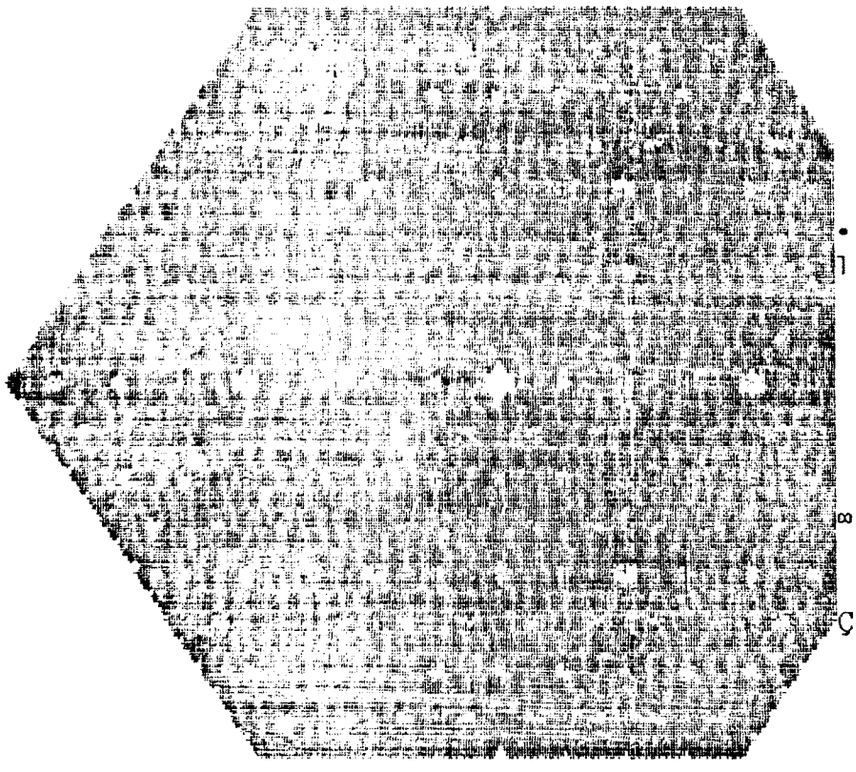
KOCH recursion for degree 100



KOCH recursion for degree 50



Implementation of Sierpinski curve printout
taken by using header file GPRNT.h



KOCH forest for degree 2
Remarkan totally different shape appears by recursion of
of star of David pattern

REFERENCES

1 Batty M.,Microcomputer Graphics , Chapman and Hall Computing, 1987 pp.144-186.

2 Michie D and Johnston R., The Creative Computer, Viking Publishing 1984.pp.150-151.

3 McGregor J. andWatt.A.,The Art of Graphics for the IBM PC, Addison-Wesley Publishing Company,1986 pp.250-283.

4 Pasupathy S.,Having fun with fractals, Online-a Publication of Birla Institute of Technology and Science, Pilani, June 1990,pp.22-23.

5 Sproull R.F.,Suytherland W.R.,Ullner M.K.,Device-Independent Graphics, McGraw Hill Book Co. 1989,pp.134-144.

6 Robbins J. , Fun with Fractals, Tech.Publications PTE Ltd.,1993.