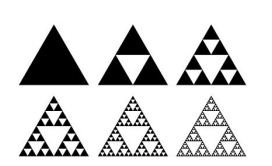
**Practical No 10.B**

**Implementation of fractal generation - Sirepenski Triangle.**

**Aim: Write a program to implement a fractal generation using Sirepenski Triangle algorithm.**

**Theory:**

Sierpinski triangle is a fractal and attractive fixed set with the overall shape of an equilateral triangle. It subdivides recursively into smaller triangles. Sierpinski Triangle is a group of multiple (or infinite) triangles. Just see the Sierpinski Triangle below to find out how infinite it may look. he concepts behind this is the fact that the filled triangle is filled by an empty equilateral triangle in the centre in such a way that this triangular space is congruent to the three triangles being formed around it.



If you see my previous article which is given here, you see that there was a triangle which I just break into further triangles without a base, in a kind of brute manner. This time it will be done in a technical manner! In other words, we don't make a triangle and then, break it into three, but we will do something else in order to produce randomness as well. The concept that we will use, is simple. It will have tiles! If there are one or more than one tiles and, not three tiles above the tile space then, we will put a tile in that space or else, no tiles!

**Approach:**

Sierpinski Triangle will be constructed from an equilateral triangle by repeated removal of triangular subsets.

**Steps for Construction:**

**Step 1:** Take any equilateral triangle.

**Step 2:** Divide it into 4 smaller congruent triangle and remove the central triangle.

**Step 3:** Repeat step 2 for each of the remaining smaller triangles forever.

**Conclusion: We have implemented a fractal generation using Sirepenski Triangle algorithm.**

**Code:**

#include <math.h>

#include <conio.h>

#include <stdlib.h>

#include <iostream.h>

#include <graphics.h>

#define Y 200

#define X 500

void triangle(float x, float y,float h) {

for (float delta = 0; delta > -5; delta -= 1) {

line(x - (h + delta) / sqrt(3),y - (h + delta) / 3, x + (h + delta) / sqrt(3),y - (h + delta) / 3);

line(x - (h + delta) / sqrt(3),y - (h + delta) / 3, x,y + 2 \* (h + delta) / 3);

line(x,y + 2 \* (h + delta) / 3,x + (h + delta) / sqrt(3),y - (h + delta) / 3); } }

void trianglev2(float x, float y, float h) {

for (float delta = 0; delta > -1 + 5; delta -= 1) {

line(x - (h + delta) / sqrt(3),y + (h + delta) / 3,x + (h + delta) / sqrt(3),y + (h + delta) / 3);

line(x - (h + delta) / sqrt(3),y + (h + delta) / 3,x,y - 2 \* (h + delta) / 3);

line(x,y - 2 \* (h + delta) / 3,x + (h + delta) / sqrt(3),y + (h + delta) / 3); } }

int drawTriangles(float x = X / 2,float y = 2 \* Y / 3,float h = Y / 2,int colorVal = 0) {

if (h < 5) {

return 0;

}

if (x > 0 && y > 0 && x < X && y < Y) {

triangle(x, y, h);

}

drawTriangles(x,y - 2 \* h / 3,h / 2,colorVal + 1);

drawTriangles(x - h / sqrt(3),y + h / 3,h / 2,colorVal + 1);

drawTriangles(x + h / sqrt(3),y + h / 3,h / 2,colorVal + 1);

return 0;

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

drawTriangles();

cout<<"\n\n\n\n\n\n\n\n\n\n\n";

cout<<"\n\n\t\t\* \* \* Sirepenski Triangle \* \* \*\n\n";

getch();

closegraph();

return 0;

}

**Output:**

