Experiment No-11

***Aim:*** *To* *Implement* *Fractals* *Algorithm.*

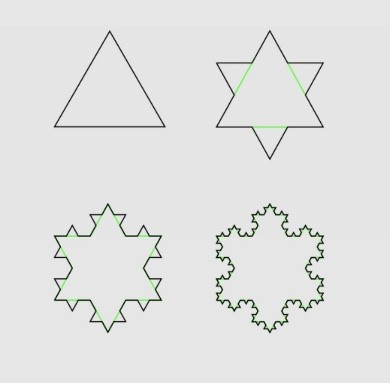
# Theory:

***Fractal*** ***Dimension-***

Fractals are complex geometric shapes that exhibit self-similarity at different scales. They are characterized by their fractal dimension, which can be a non-integer value. For example, the coastline of a country is a fractal because as you zoom in, you will continue to see irregularities at smaller scales.

# The Koch Snowflake-

The Koch snowflake is a classic fractal that can help illustrate the concept of fractal dimension. It's constructed using a recursive algorithm, and each iteration adds more detail to the snowflake



# Algorithm for Koch curve-

1. Start with two points: Define the starting point (let’s call it `P1`) and the ending point (let’s call it `P2`) of your line segment. These two points represent the initial straight line.
2. Calculate the length: Calculate the distance between `P1` and `P2`. This will be the length of your line segment.
3. Divide the line into three equal parts: Calculate two additional points:

`P3`, one-third of the way from `P1` to `P2`.

`P4`, two-thirds of the way from `P1` to `P2`.

1. Draw the first segment: Draw a line segment from `P1` to `P3`.
2. Calculate the height of the equilateral triangle: Calculate the height (`h`) of an equilateral triangle with a side length equal to one-third of the original line segment. The formula for the height of an equilateral triangle is `(length / 2) \* sqrt(3)`.
3. Calculate the midpoint of the triangle: Calculate the midpoint of the line segment from

`P3` to `P4`. This will be the top vertex of the equilateral triangle. Let’s call this point

`P5`.

1. Calculate the two remaining vertices of the equilateral triangle: Calculate two more points:

`P6`, which is `h` units vertically above `P5`.

`P7`, which is the reflection of `P6` with respect to `P5`. You can calculate `P7` by mirroring `P6` horizontally across `P5`.

1. Draw the equilateral triangle: Draw lines to connect `P5`, `P6`, and `P7` to form an equilateral triangle.
2. Draw the last segment: Draw a line segment from `P4` to `P2`.
3. Repeat the process: You can repeat steps 3 to 9 for each of the four line segments created in the previous iteration. You can control the depth of recursion to control the level of detail in your Koch curve.
4. Stop at a certain iteration: You can choose to stop the process after a certain number of iterations to control the level of detail in your Koch curve.

You can see in the above figure that only the segment-1st shape as we have only changed the control point P1, and the shape of segment-2nd remains intact.

# Program:

#include <graphics.h> #include <stdlib.h> #include <stdio.h> #include <math.h>

#define pi M\_PI typedef *struct* {

*double* x, y;

} point;

*void* kochCurve(point *p1*, point *p2*, *int* *times*) { point p3, p4, p5;

*double* theta = pi / 3;

if (*times* > 0) {

p3 = (point)((2 \* *p1*.x + *p2*.x) / 3, (2 \* *p1*.y + *p2*.y) / 3);

p5 = (point)((2 \* *p2*.x + *p1*.x) / 3, (2 \* *p2*.y + *p1*.y) / 3);

p4 = (point)(p3.x + (p5.x - p3.x) \* cos(theta) + (p5.y - p3.y)

\* sin(theta), p3.y - (p5.x - p3.x) \* sin(theta) + (p5.y - p3.y) \* cos(theta));

kochCurve(*p1*, p3, *times* - 1); kochCurve(p3, p4, *times* - 1); kochCurve(p4, p5, *times* - 1); kochCurve(p5, *p2*, *times* - 1);

}

else {

line(*p1*.x, *p1*.y, *p2*.x, *p2*.y);

}

}

*int* main(*int* *argC*, *char* \*\**argV*) {

*int* w, h, r; point p1, p2;

if (*argC* != 4) {

printf("Usage : %s <window width> <window height> <recursion level>", *argV*[0]);

}

else {

w = atoi(*argV*[1]);

h = atoi(*argV*[2]);

r = atoi(*argV*[3]);

initwindow(w, h, "Koch Curve"); p1 = (point){10, h - 10};

p2 = (point){w - 10, h - 10};

kochCurve(p1, p2, r);

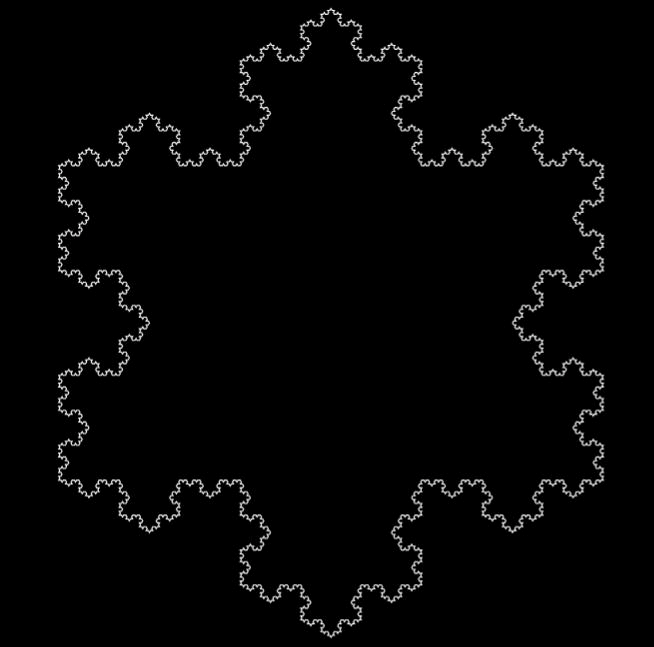
getch(); closegraph();

}

return 0;

}

# Output:



***Conclusion:***

I have understood how to implement fractals in CG