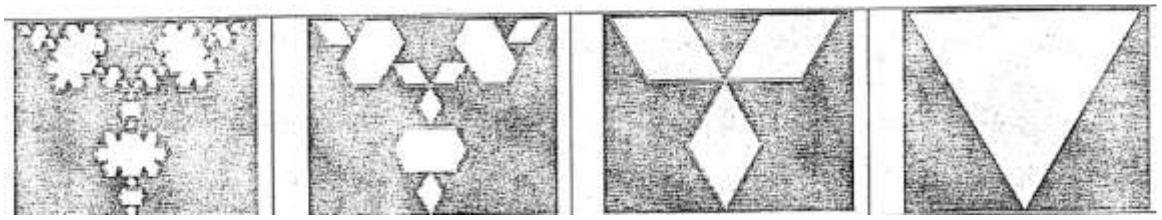
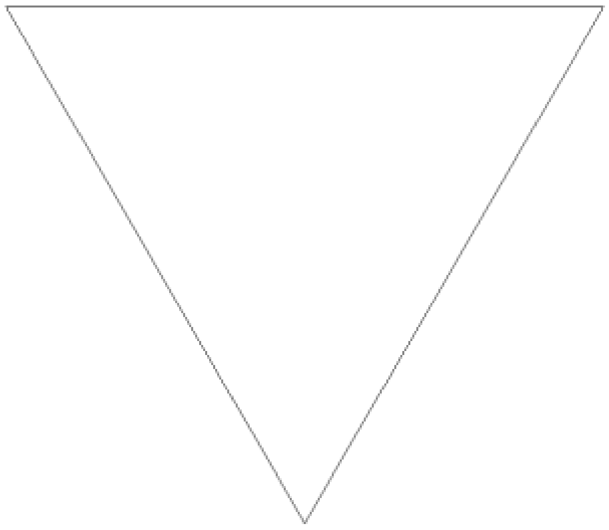
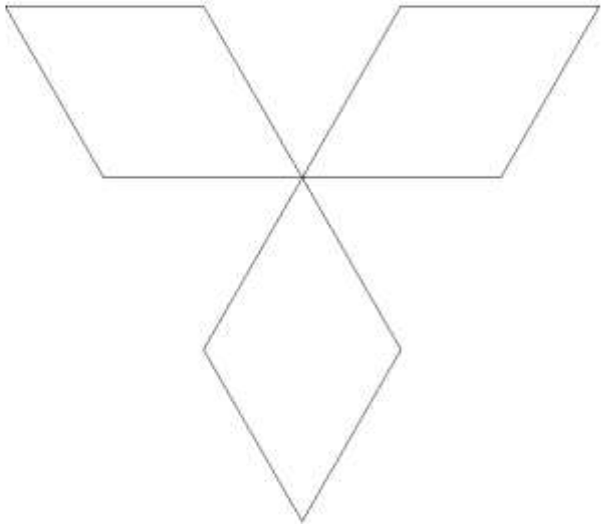
How to draw this:

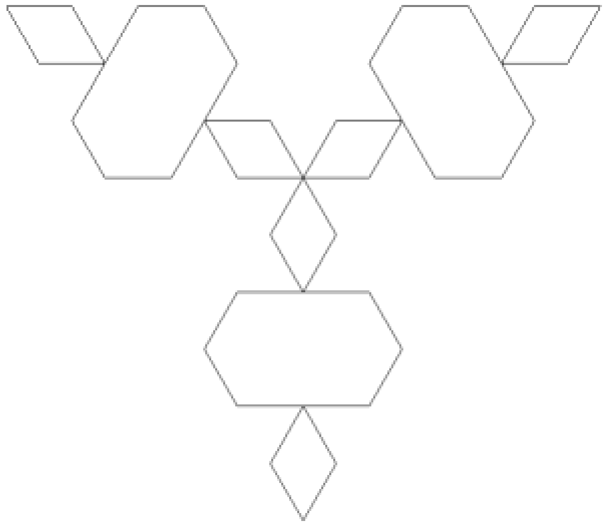


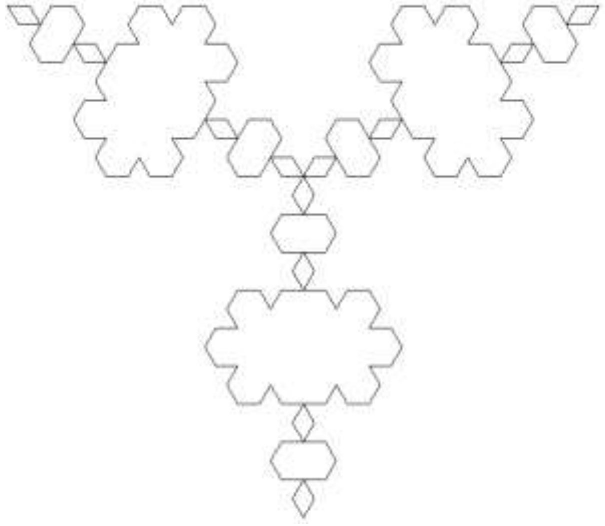
Intro:

Implementation of all next projects was done in Java (JDK1.4), it’s a modular OOP language and easy to draw with.







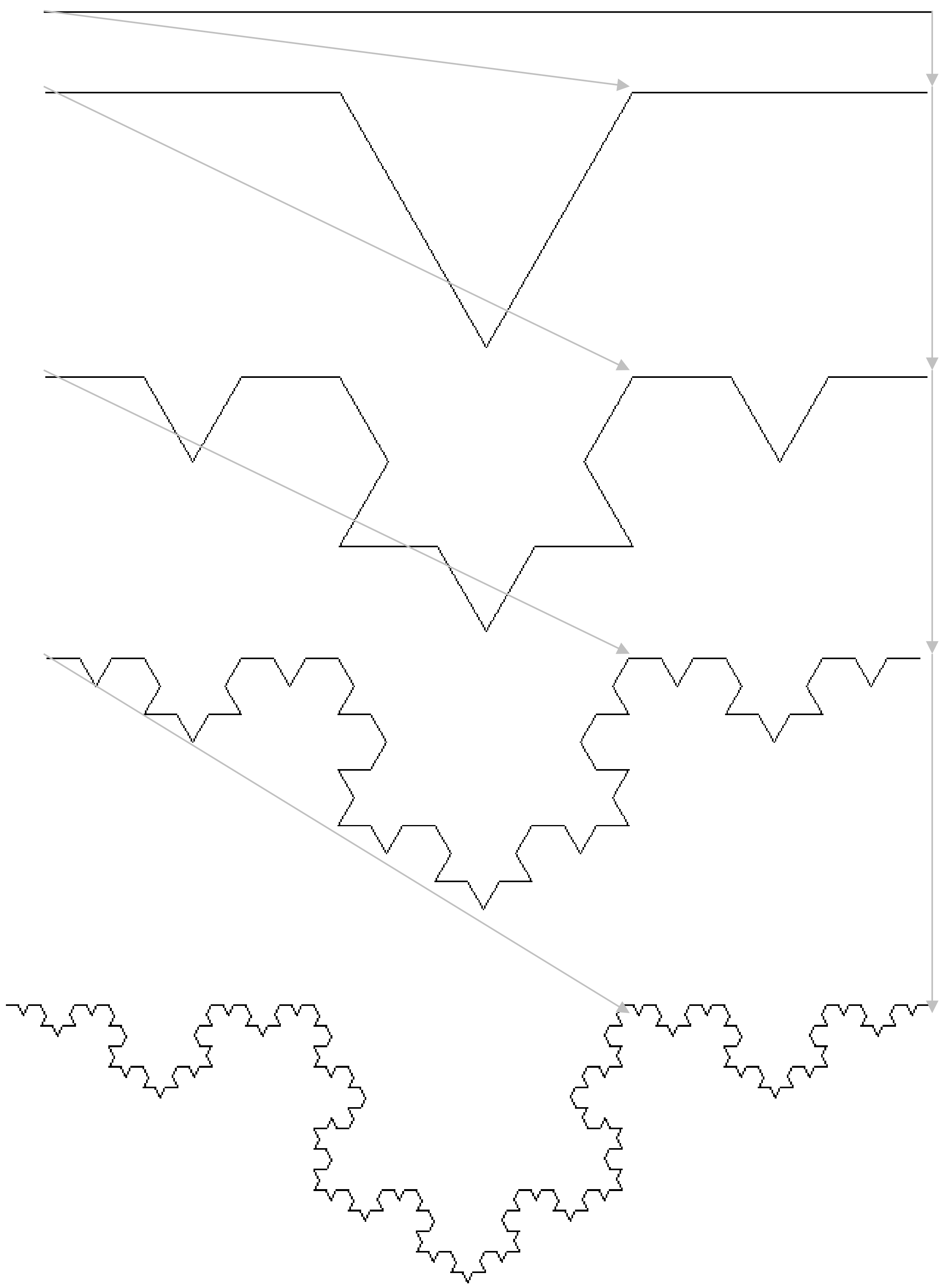


To find out the recursive in the previous figures we need to analyze the figures first to understand how they were from out of each other then to write down the drawing algorithm.

1- First of all:

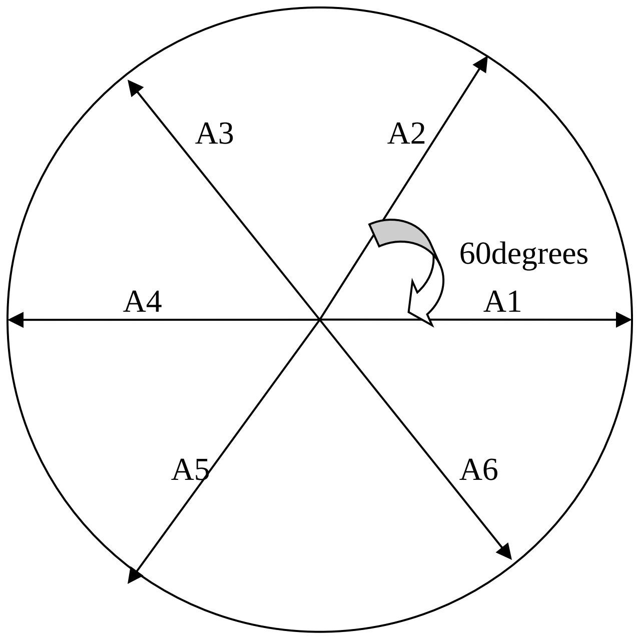
As we see there are no real intersected lines that forms an internal shapes [like triangles, or something] so we can say that the above figure is a collection of connected lines

2- the four levels have the general shape of a triangle of equal sides
3- lets take the side to above:



We see that each level consists of 4 parts of the level before, with different rotations, and the basic figure is a line at the first level, so we can say that:

For a 60 degrees angle we've got 6 different rotations with different directions which could be represented in this way:



So a line has a start point + direction , and that discriminates the all 6 rotations of the figure, considering the start point is the top-right corner we get:

A4 . A6. A2   is the basic figure

And the figures with all of their mutations will be

A1:	A2 . A6 . A2 . A1
A2:	A2 . A3 . A1 .A2
A3:	A3 . A4 . A2 . A3
A4:	A4 . A5 . A3 . A4
A5:	A5 . A6 . A4 . A5
A6:	A6 . A1 . A5 . A6

To form the final result.

Sure each call to a minor level will be in the form:

Ai(h/3 , level -1)

The final piece is: when reaching the level 1 we draw a line, else we call the parts of the lower level.

The position of the new point depends on the direction and such: sin and cos of the angle used depending on the similarity to the circle.

For example: to draw A1:

```
drawA1(h,level) {
    if (level == 1) {
        x = x0 + h;
        plot(x0, y0, x, y);
        x0 = x;
    } else {
        drawA1(h / 3, level - 1, g);
        drawA2(h / 3, level - 1, g);
        drawA6(h / 3, level - 1, g);
        drawA1(h / 3, level - 1, g);
    }
}
```

Implementation Notes: Graphic (like in the first assignments):
drawing is done using double buffering technique, with partial changes taken into account:
So all drawing is done off-screen, and flipped back onto screen at once, so no direct change to the graphic device happens only when copying off-screen onto screen.

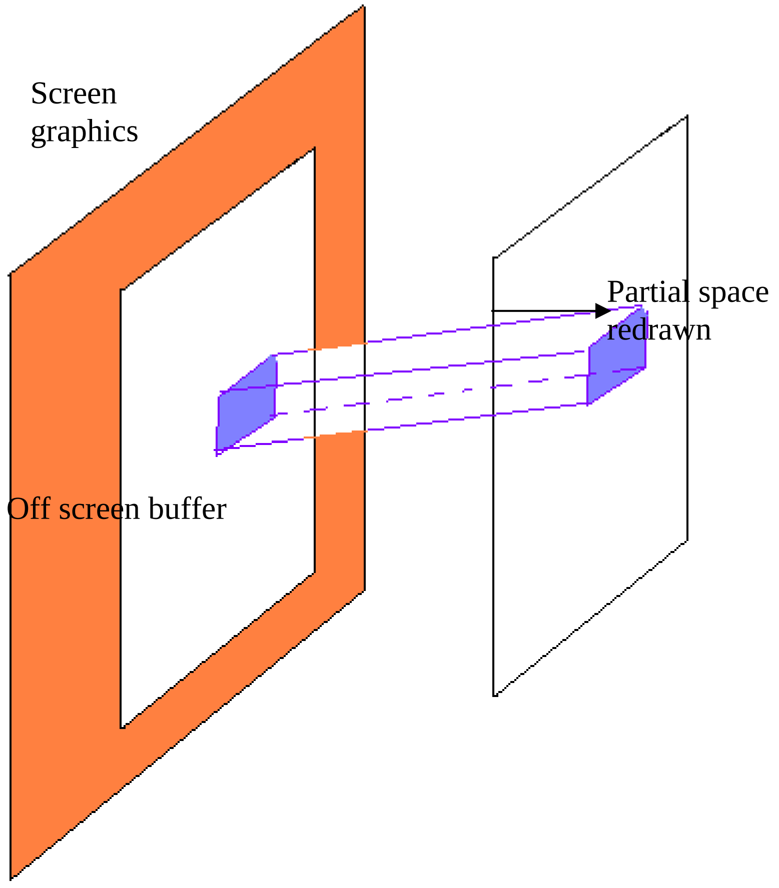
In java we use a BufferedImage object for this purpose.

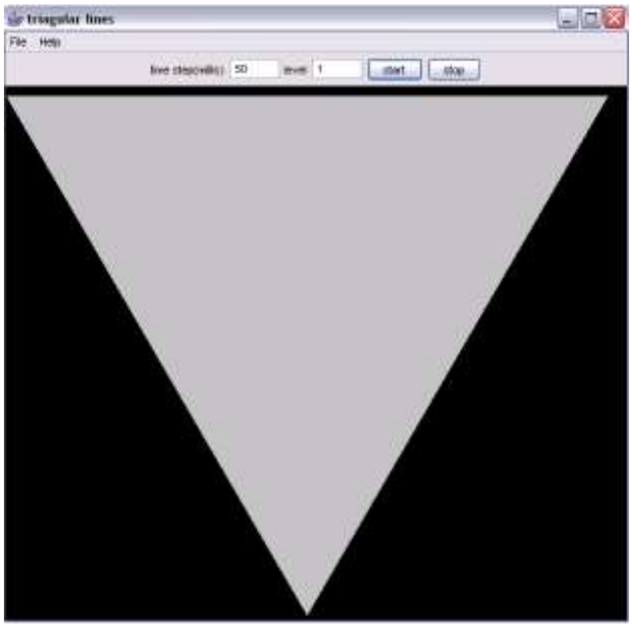
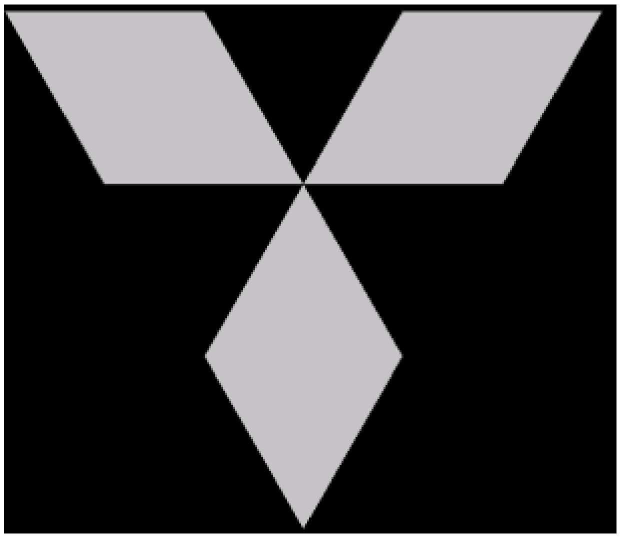
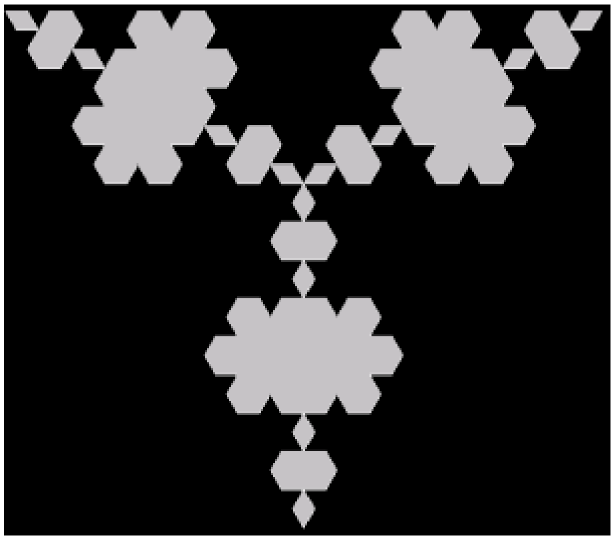
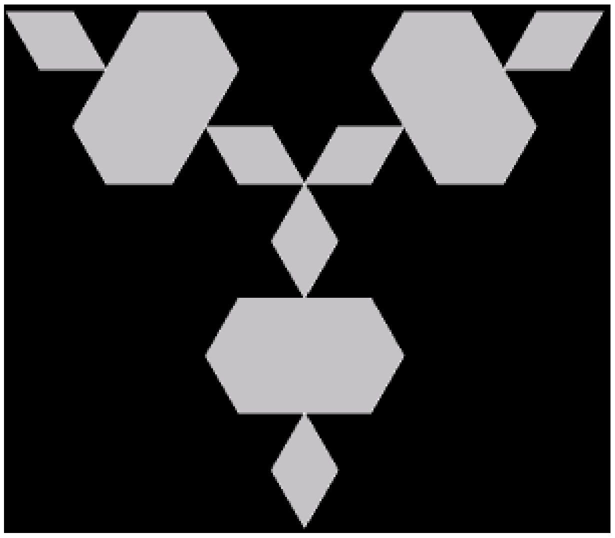
Drawing is done using steps inside a single thread, and thus prevents the UI from locking as we said before, and is done every step, which can be initially controlled by the user by setting it's value.

So you'll experience a smooth
Animation with no flickering
Or anything.

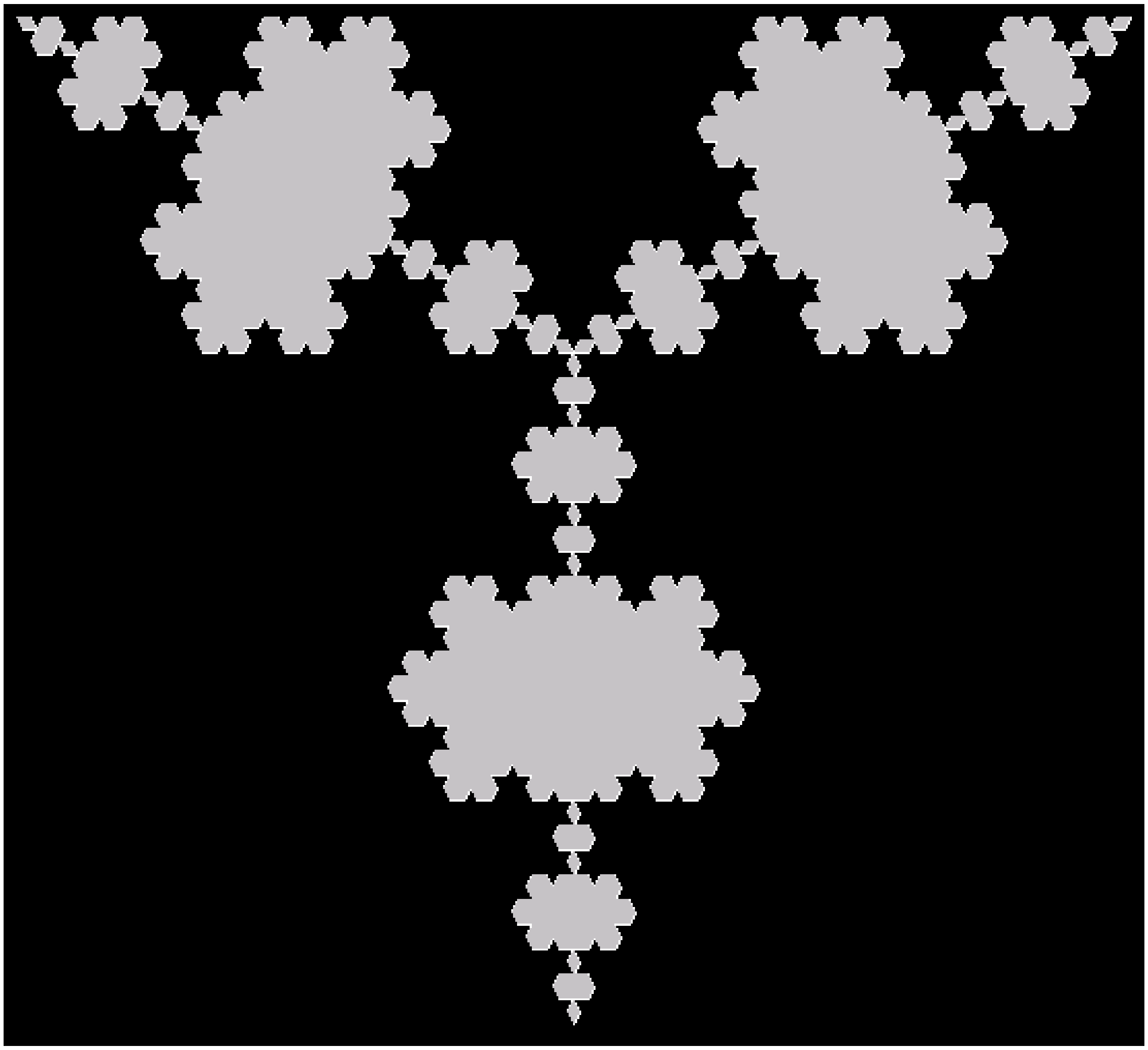
step of line drawing is
entered by the user, so the
level too.

Next we'll see a sample
Of the UI



Leve1 (shows the time step and level choice)	Level 2
 time step(millis) 50 level 1	
Level3	Level4
	

Level 5



The shading of the figure was done using path drawing:
A GeneralPath object of the drawn lines and closed when finished and filled with a light-gray color (filling after the whole shape was drawn, so animation over the lines stands still).