Introduction to Programming

— Practical #7 —

1 First objects and fractals

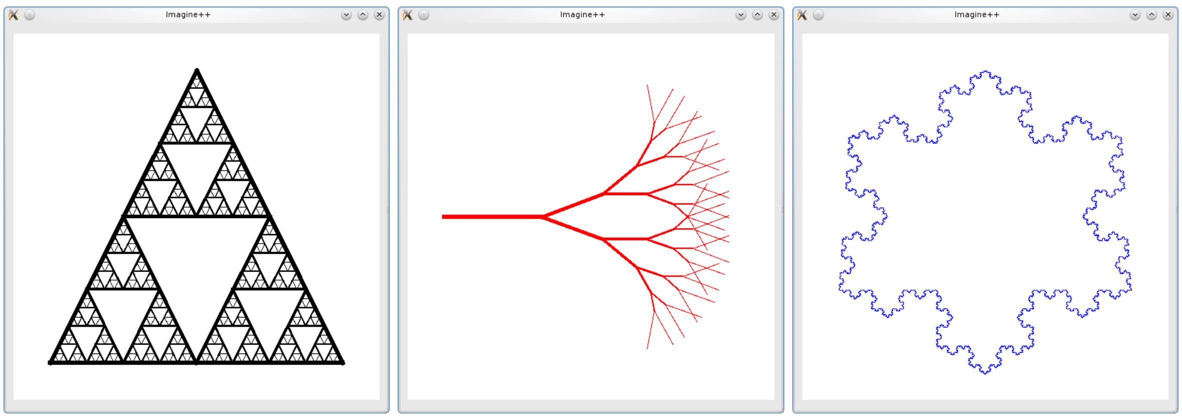

Figure 1 – Fractals...

In this practical, we will try some modest object oriented programming. We will transform a structure for a vector in a class and use it to draw fractals (figure 1).

1.1 Sierpinski triangle

1. *Get the project :*
   Download the file `Tp7_Initial.zip`, decompress it and launch your IDE. Study the structure `Vector` defined in files `Vector.cpp` and `Vector.h`.
2. *Interface with Imagine++ :*
   The structure `Vector` does not provide any graphic display. Add in `main.cpp` functions `drawLine` and `drawTriangle` taking some `Vector` as arguments (here, a vector is used to note a plane point). Just call the regular function

   ```
   void drawLine(int x1, int y1, int x2, int y2,
                 const Color& c, int pen_w)
   ```

   from Imagine++. The last argument controls the thickness of the pencil.
3. *Sierpinski Triangle :*
   This is the figure chosen by ENPC for its logo. Figure 2 illustrates its construction.
   Write a recursive function to draw the Sierpinski triangle. This functions takes as arguments the three

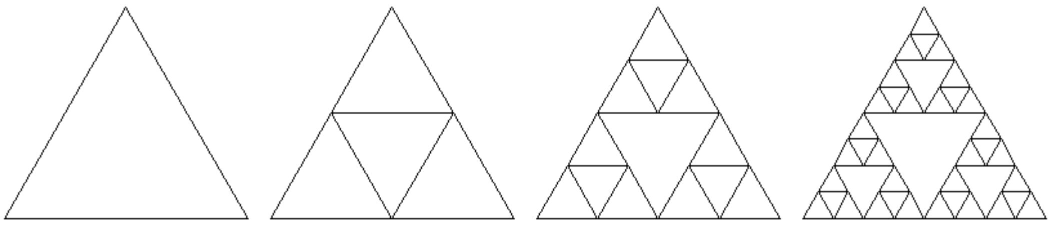

Figure 2 – Construction of Sierpinski triangle.

   points of the current triangle and the thickness of the pencil. The three subtriangles are drawn with a pencil less thick. **Do not forget the break condition for the recursion !**
   Use this function in main by giving it an initial equilateral triangle of thickness 6 pixels.

1.2 A class rather than a structure

4. *Class vector :*
   Transform the structure `Vector` into a class. Incorporate all functions and operators. Put in public only what is necessary, do the required modifications in `main.cpp`.
5. *Accessors for members :*
   Add accessors for reading and writing members, and use them systematically in the main program. The idea is to hide from the user of the class `Vector` the details of its implementation.
6. *Recursive drawing of a tree :*
   We are now going to draw a tree. For that, we start from a trunk and replace the second half of each branch by two branches of same length with an angle of 20 degrees with the originating branch. Figure 3 illustrates the result for different depths of recursion.
   Write a recursive function to draw such a curve. You will use the method `rotate` of class `Vector`.

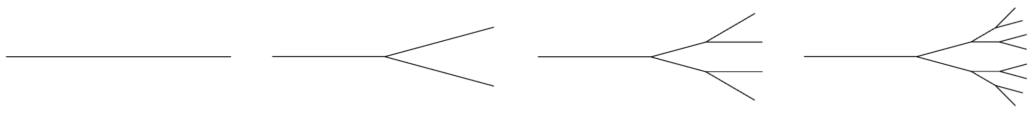

FIGURE 3 – Tree construction.

It is natural to think of erasing the half of the preceding segment to draw the new branches, but it is not a great idea : the segments are discretized (in pixels), and a discretized subsegment may not go exactly on the same pixels as the master segment, hence some residual stains after erasing. The same remark will apply to the Koch curve below.

1.3 Change the implementation

7. *Second implementation :*
   Modify the implementation of class `Vector` by replacing the members double x,y; by an array double coord[2];. What modifications should be applied in `main.cpp` ?
8. *Vectors of higher dimension :*
   The advantage of the latest implementaion is that it can be generalized to vectors of higher dimension. Put a global constant `DIM` of value 2 at the top of file `vector.h` and make the class `Vector` independent of the dimension.
   NB : The method `rotate` and the accessors we defined cannot be generalized directly to higher dimension. Leave them unchanged but reserve their use to dimension 2 by putting an assert...

1.4 The snowflake

9. *Koch curve :*
   This fractal curve can be built by starting from a segment and replacing the second third of each segment by two segments so as to get an equilateral triangle.
   Write a recursive function to draw this curve.

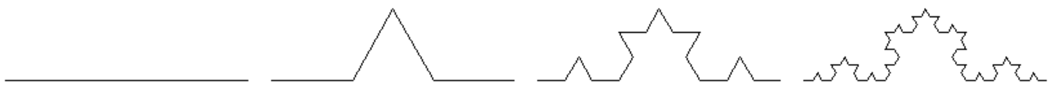

FIGURE 4 – Construction of the Koch curve.

10. *Snowflake :*
    It is obtained by building three Koch curves from each of the sides of an equilateral triangle.