Recursive Pratical - Fractals

A fractal is a mathematical set that typically displays self-similar patterns, which means it is "the same from near as from far". Fractals may be exactly the same at every scale, or, [. . .], they may be nearly the same at different scales. The concept of fractal extends beyond trivial self-similarity and includes the idea of a detailed pattern repeating itself.

wikipedia[1]

Here, the fractals are based on a recursive process of building : a segment (in the first part) is replaced by a pattern composed of several lower length segments. Then, each segment is in turn replaced by the same pattern and so on. Each additional iteration is a new curve : the superior rank curve. The fractal is the set of points at the intersections of these curves. The exercises in second part are based on the same principle, but here applied to surfaces.

Tutorial

Graphics library

You will need several functions from the graphics library. [2].

First you will need to load the module (only once) and open the graphics window :

```
#load "graphics.cma" ;;     (* Load the library *)
open Graphics ;;            (* Open the module *)
open_graph "";;             (* Open the window *)
```

Some useful functions

```
val clear_graph : unit -> unit         (* erase the graphics window      *)
val moveto : x:int -> y:int -> unit    (* position the current point  *)
val rmoveto : dx:int -> dy:int -> unit (* translates the current point by the given vector *)
val lineto : x:int -> y:int -> unit    (* draw a line with endpoints the current point and
                                          the given point, and move the current point to the
                                          given point. *)
val rlineto : dx:int -> dy:int -> unit (* Draw a line with endpoints the current point and
                                          the current point translated of the given vector,
                                          and move the current point to this point.*)
```

Try out this example :

```
let draw x y l h =
    moveto x (y + h/2);
    rlineto (-l/2) (-h);
    rlineto (l) (2*h/3);
    rlineto (-l) 0;
    rlineto (l) (-2*h/3);
    rlineto (-l/2) (h);;

draw 50 50 100 100;;
```

1. *http ://en.wikipedia.org/wiki/Fractal*
2. *http ://caml.inria.fr/pub/docs/manual-ocaml/libref/Graphics.html*

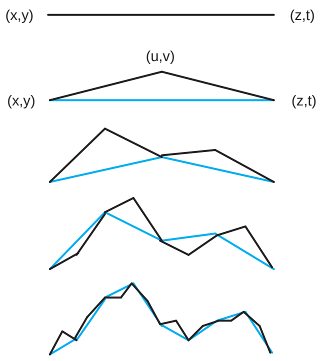
1 Mountain

Here, the aim is to randomly generate a "mountain" according to the following principle :

step 0 : Draw a segment between two points.

step 1 : Calculate a new height for the center of the segment (random[3]).

step n : The same process is applied on each new segment of the previous step.



Method :
The n-order "curve" between points (x, y) and (z, t) is :

$n = 0$ the segment $[(x, y), (z, t)]$

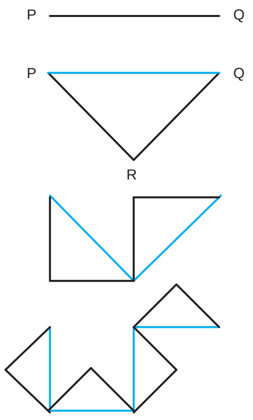
$n \neq 0$ The $(n-1)$-order "curve" between points (x, y) and (m, h) followed by the $(n-1)$-order "curve" between points (m, h) and (z, t), where m is the "center" of x and z, and h is a new height randomly calculated.

Therefore the function takes the order n and the two points (x, y) and (z, t) as parameters.
Advice : calculate the new height depending on the two points and possibly on n. As an example, we might decrease the height difference as and when the two points get closer. . .
Some examples (`int(e)` gives a random integer between 0 and $e - 1$) :
$h = (y + t)/2 + int(10 * n)$
$h = (y + t)/2 + int(abs(z - x)/5 + 20)$

2 Dragon

Write a function that draws the curve defined by :

- The 0 order curve is a segment between any two points P and Q.
- The n-order curve is the $(n-1)$-order curve between P and R followed by the same $(n-1)$-order curve between R and Q (reversed), where PRQ is an isocel triangle rectangle in R, and R is on the right of segment PQ.



That can be described this way : Starting from a base segment, replace each segment by 2 segments with a right angle and with a rotation of 45° alternatively to the right and to the left.

A little help : If P and Q are points of coordinates (x, y) and (z, t), the R coordinates (u, v) are :

$$u = (x + z)/2 + (t - y)/2$$

$$v = (y + t)/2 - (z - x)/2$$

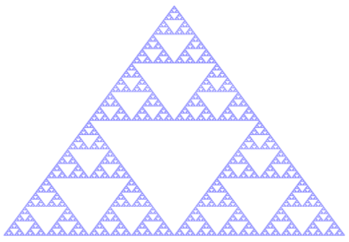
3. You can use functions from the pseudo-random number generator : the module `Random` of which you will find a description in the CAML manual. Do not forget to initialize the generator !

3 Bonus

3.1 Sierpinski triangle

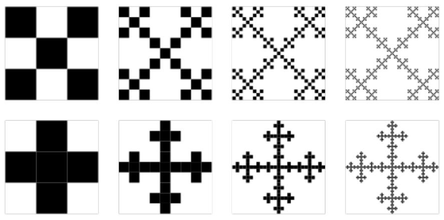
The Sierpinski triangle is defined as follows :

- The order 0 triangle is an equilateral one.
- $(n+1)$-order triangle is the n-order triangle homothety of scale factor 0.5, three times duplicated and placed in such a way that these are two by two adjacent by one vertex.



3.2 Vicsek

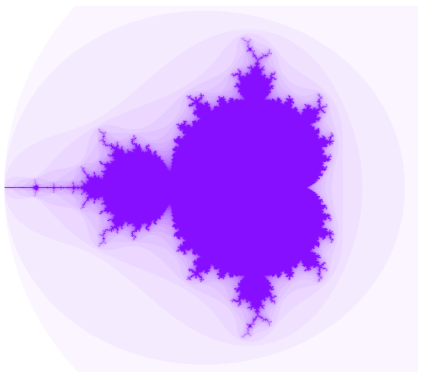
From wikipedia :
Vicsek fractal, also know as Vicsek snowflake or box fractal, is a fractal arising from a construction similar to that of the Sierpinski carpet. It is defined as follows : The basic square is decomposed into nine smaller squares in the 3-by-3 grid. The four squares at the corners and the middle square are left, the other squares being removed. The process is repeated recursively for each of the five remaining subsquares.



3.3 Mandelbrot set

To display the Mandelbrot set[4], we will use the following functions :

`rgb : int -> int -> int -> Graphics.color = <fun>` build a color using the `RGB` format
`set_color: Graphics.color -> unit = <fun>` change the current color.
`plot : int -> int -> unit = <fun>` display a point at given coordinates.



4. `http://en.wikipedia.org/wiki/Mandelbrot_set`