Recursive Pratical
Fractals

Principle

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.

Wikipedia extract[1]:

> "A fractal is a mathematical set that has a fractal dimension that usually exceeds its topological dimension and may fall between the integers. Fractals are typically self-similar patterns, where self-similar means they are "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 definition of fractal goes beyond self-similarity per se to exclude trivial self-similarity and include the idea of a detailed pattern repeating itself."

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 graphic window:

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

Some useful functions (extract from manuel):

`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`
`rmoveto dx dy` 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 test (x,y) (z,t) =
  clear_graph ();
  set_color red;
  moveto x y ;
  lineto z t ;
  set_color blue ;
  rmoveto x y ;                    test (50,50) (50,150) ;;
  rlineto z t ;;                   test (100,100) (200,100) ;;
```

[1] http://en.wikipedia.org/wiki/Fractal

[2] http://caml.inria.fr/pub/docs/manual-ocaml/libref/Graphics.html

Curves

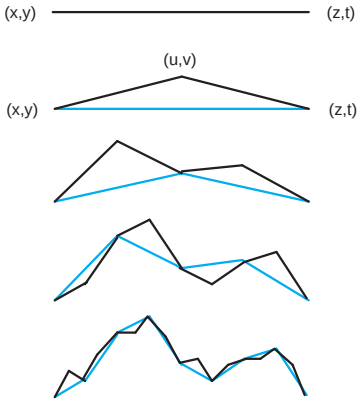
Advice for the following exercises: first try small values for curve orders...

Exercise 1 (Montagne)

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

step 0: Draw a segment between two points.
step 1: Calculates 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.

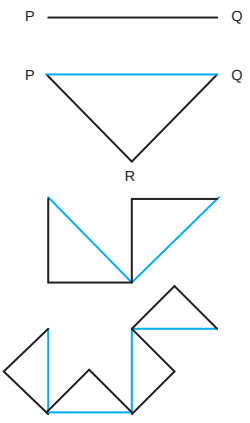
Method:
The n-order "curve" between points (x, y) and (z, t) is:



$n = 0$ 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 $e-1$):
$h = (y + t)/2 + int(10 * n)$
$h = (y + t)/2 + int(abs(z - x)/5 + 20)$

Exercise 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:

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

Application example to obtain a pretty dragon:

```
clear_graph ();
dragon 19 (150,150) (350,350) ;;
```

Some "dragon bonus" (and others) online (`http://algo-td.infoprepa.epita.fr/`)!

[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!

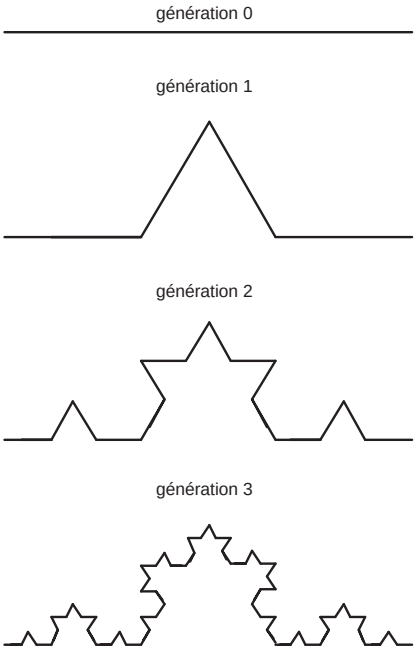
Exercise 3 (Bonus: Koch snowflake)

The Koch snowflake is defined as follows:

- The 0 order snowflake is an equilateral triangle.
- The 1^{st} order snowflake is the same triangle of which each line segment is cut into three parts and on each one, another equilateral triangle leans (in the middle).
- The $(n+1)$-order snowflake involves an n-order snowflake on which the same operation is applied to each of its sides.

Speaking in pattern terms, the snowflake can be seen as follows: it is composed of three Koch curves placed on equilateral triangle sides.

Koch curve: a segment of length d is replaced by 4 segments of length $d/3$. The angle of oblique segments is 60^o (see figure below).



First, write a function that draws the Koch curve from two integers: n the curve rank and d the segment length. Then write the function drawing a whole snowflake at given rank n.

Surfaces

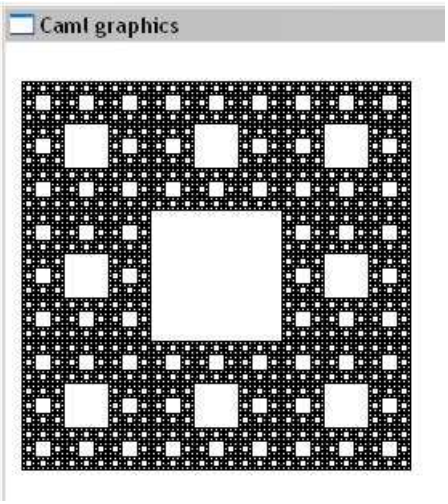
Exercise 4 (Sierpinski carpet)

A simple fractal (when you know the principle...): write a function that creates this "carpet" starting from a given point (x, y) and with a given size.

For example, the carpet opposite was obtained with:

```
let sponge (x,y) n =     (* "éponge" de Sierpinski in French *)
  clear_graph () ;
  moveto x y ;
  draw_sponge n ;;

sponge (10,10) 243 ;;
```



This function, from the manual, will be useful:

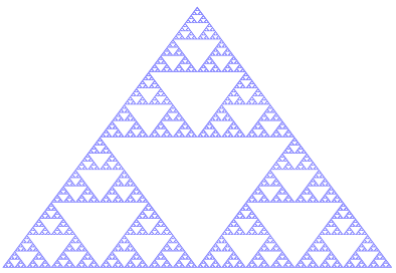
`val fill_rect : int -> int -> int -> int -> unit`

`fill_rect` x y w h fills the rectangle with lower left corner at (x, y), width w and height h, with the current color. Raise `Invalid_argument` if w or h is negative.

Exercise 5 (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.



Exercise 6 (Bonus: 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.