DEVOIR MAISON II

# Fractales en OCaml

Ceci est un DM **optionnel** ; il n'y a rien à rendre. Son but est de faire de jolis dessins : des fractales. Une fractale est un dessin qui se répète dans lui-même ; on appelle **profondeur** le nombre maximal de répétitions imbriquées.

Le triangle de Sierpinski est une exemple célèbre, vous pouvez aller en voir des dessins en section II.

## I - Un peu d'aide

### 1) Compléments de OCaml

En OCaml, si x et y sont des expressions, `(x,y)` est la paire (le 2-uplet) composée de x et y.

Réciproquement, si `p` est une paire, `let (x,y) = p in ...` permet d'accéder aux deux coordonnées de la paire. On parle de **let destructurant**. Voici par exemple des fonctions qui renvoient respectivement la première et la seconde coordonnée d'une paire :

```
let fst = fun paire ->
  let (x,_) = paire in x
```

```
let snd = fun paire ->
  let (_,y) = paire in y
```

Pour plus de confort :

*Remarque.*

- Au lieu d'écrire `let f = fun x ->...` , on peut écrire `let f x = ...` . De même à deux arguments : `let f = fun x y -> ...` se réécrit `let f x y = ...` , et ainsi de suite.
- On peut destructurer une paire « dès qu'on la prend en argument ». Ainsi, les deux fonctions ci-dessus sont équivalentes à :

```
let fst (x,_) =
  x
```

```
let snd (_,y) =
  y
```

### 2) Fonctions pré-fournies

#### 2.i) `Graphics`

Pour faire une fenêtre graphique et des dessins, j'utilise la librairie `Graphics`. J'en donne ici les grandes lignes.

`Graphics` permet de faire une fenêtre graphique. Les pixels y sont indicés comme en maths : $(0,0)$ est le pixel en bas à gauche, l'abscisse est croissante vers la droite et l'ordonnée croissante vers le haut. Un **point** est donc une paire (`abscisse`, `ordonnee`) d'entiers ; par exemple `(990, 512)` .

Le module contient des couleurs pré-codées, que l'on pourra assigner aux pixels : j'utilise `Graphics.black` et `Graphics.white` . La fonction `Graphics.set_color` prend en argument une couleur et la définit comme « couleur par défaut ».

La fonction `Graphics.fill_poly` prend en argument un tableau de points qui réprésente les sommets d'un polygone et remplit le polygone de la couleur par défaut. Vous n'avez pas besoin de savoir manipuler les tableaux ; sachez juste que `[|a;b|]` est le tableau de longueur 2 contenant `a` et `b` ; `[|a;b;c|]` un tableau de longueur 3, etc. L'ordre des éléments dans le tableau ne compte pas pour `Graphcis.fill_poly` .

Et c'est à peu près tout ce qu'il faut savoir. [1]

1. La documentation officielle est `https://ocaml.github.io/graphics/graphics/Graphics/index.html` ; mais vous ne devriez pas avoir besoin d'aller la lire.

### 2.ii) Fonction précodées

- `init ()` . Elle ouvre une fenêtre, la redimensionne pour qu'elle soit de longueur horizontale `x_max` et verticale `y_max` , puis remplit le triangle `[|a; b; c|]` . `x_max` , `y_max` , `a` , `b` et `c` sont des variables globales.
- `colorie_creux u v w` colorie le triangle `u--v--w` avec la couleur `couleur_creux` . Elle fait appel à `Graphics.fill_poly`, puis utilise `Unix.sleepf` pour attendre 0,5s .
- `wait_keys` prend en argument une liste de caractères (par exemple `['s'; 'S']` ) et met en pause le programme jusqu'à ce que la touche correspondante soit appuyée sur le clavier. Cela permet par exemple d'attendre le signal de début ou d'attendre le signal de fin.

Ces 3 fonctions ne renvoient rien (c'est à dire renvoient `()` ), mais ont des effets secondaires (ceux décrit ci-dessus).

## 3) Pour compiler

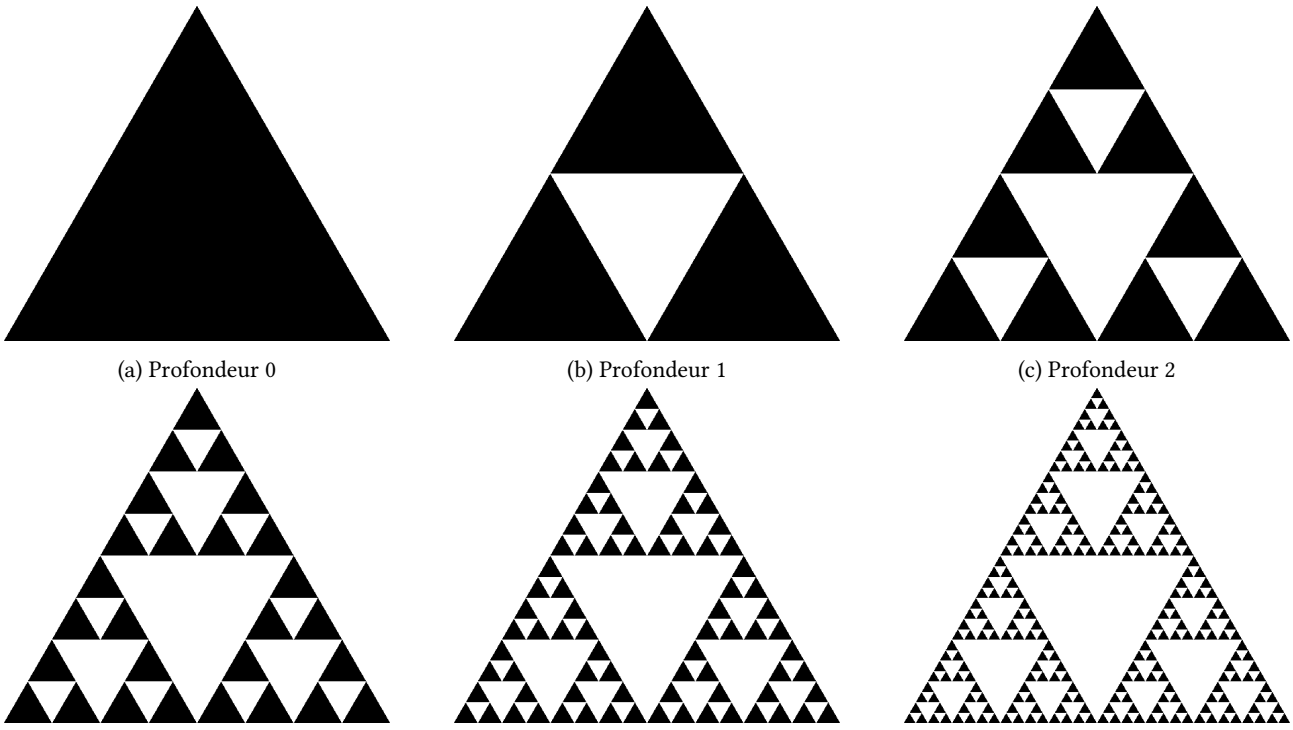
La ligne de compilation est un peu plus compliquée (car il faut appeler `Graphics` et `Unix`), la voici :

```
ocamlfind ocamlopt -package graphics,unix -linkpkg sierpinski.ml
```

# II - Ce qu'il vous reste à faire

## 1) Triangle de Sierpinski

Le triangle de Sierpinski est une fractale célèbre :



(a) Profondeur 0 (b) Profondeur 1 (c) Profondeur 2

(d) Profondeur 3 (e) Profondeur 4 (f) Profondeur 5

FIGURE II.1 – Des triangles de Sierpinski

La recette pour créer un triangle de Sierpinski de profondeur `prof` ressemble à ceci : « enlever le sous-triangle central[2], puis `prof-1` étapes sur les 3 autres sous-triangles ». Le triangle de profondeur 0 est un cas particulier sur lequel il n'y a rien à faire.

1. Essayez de formaliser un peu cette recette : en fonction des coordonnées du sommet, quelles sont les coordonnées des 4 sous-triangles utilisés ? Pour bien voir ces 4 sous-triangles, regardez le passage de la profondeur 0 à la profondeur 1.

2. On dit aussi qu'on **creuse** le sous-triangle central.

2. Quelle fonction pré-codée sera utile pour « enlever/creuser le sous-triangle central » ?

Le fichier `sierpinski.ml` contient toutes les fonctions pré-codées précédemment mentionnées. Il ne vous reste que deux fonctions à coder dedans pour que tout fonctionne.

3. Complétez le code de la fonction `milieu` . Elle prend deux points en arguments, `(x0,y0)` et `x1,y1` et doit renvoyer le point milieu du segment qui va de `(x0,y0)` à `(x1,y1)` .
   NB : on utilise des points à coordonnées entières, et c'est donc normal si votre fonction ne renvoie pas le "vrai" milieu mais uniquement une approximation à coordonnées entières de celui-ci.
4. Complétez le code de la fonction `sierpinski` . Voici sa spécification :

   - Entrées :
     `p` , `q` et `r` les sommets haut, gauche et droit d'un triangle plein. Chacune de ces 3 variables est un point c'est à dire une paire $(x, y)$ d'entiers.
     `prof` la profondeur voulue. Il s'agit d'un entier positif ou nul.
   - Sortie :
     « rien », c'est à dire `()` .
   - Effets secondaires :
     « Creuse » le triangle $PQR$ pour en faire un triangle de Sierpinski de profondeur `prof`.

   Elle prend en argument `p` , `q` et `r` les 3 sommets d'un triangle comme décrit précédemment ( `p` est le sommet du haut, `q` celui de gauche et `r` celui de droite) ainsi que `prof` la profondeur du triangle de sierpinski à créer. Elle doit « creuser » le triangle $pqr$ pour en faire un triangle de Sierpinski de profondeur `prof` .
5. Testez =) . Lancez le programme avec `./a.out profondeur_voulue` ; par exemple : `./a.out 4`
   Il faut appuyer sur la touche `s` du clavier pour lancer le dessin, et `q` pour quitter à la fin.

## 2) Tapis de Sierpinski

6. Allez voir `https://fr.wikipedia.org/wiki/Tapis_de_Sierpi%C5%84ski` , et essayez de dessiner ce tapis. Vous aurez besoin de modifier d'autres fonctions et constantes. Typiquement, `a` `b` et `c` ainsi que `init` doivent être adaptées, de même que `colorie_creux` .