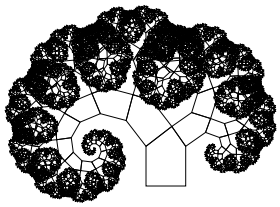
Fractales

plagia d'un sujet de Lionel Rieg

Étienne Miquey
etienne.miquey@ens-lyon.fr



Ce TP a pour objectif de présenter deux techniques différentes pour générer des fractales. La première décrit de façon explicite les étapes alors que la seconde, plus matheuse, voit la fractale comme le point fixe d'un ensemble de fonctions contractantes du plan. C'est aussi l'occasion de manipuler rapidement les bibliothèques graphiques et de faire un peu de géométrie, c'est toujours utile. Les graphiques ne pouvant être réalisés que sous le toplevel Caml Light et non depuis un éditeur (il fallait bien un jour vous avouer la petite faille d'emacs), il vous faudra copier votre travail, par exemple à l'aide d'un `include "nom-de-fichier";;`.

1 Géométrie camélienne

N'oubliez pas de débuter votre fichier par un `#open "graphics";;` qui permet d'utiliser les fonctions graphiques sans devoir les précéder d'un fort encombrant `graphics__`.
On représente les points du plan par le type suivant :

```
type point =  x : float; y : float;;
```

Les coordonnées sont définies comme en mathématiques, c'est à dire que le pixel (0, 0) est celui en bas à gauche. Les primitives graphiques dont vous aurez besoin sont :

- `plot : int -> int -> unit`, qui dessine le pixel de coordonnées données en arguments
- `moveto : int -> int -> unit`, qui déplace le point courant
- `lineto : int -> int -> unit`, qui trace une ligne depuis le point courant jusqu'au point en argument et y déplace le point courant
- `current_point : unit -> int * int`, qui renvoie la position du point courant et peut être utile pour détecter les erreurs

Question 1. Écrire une fonction `o` qui compose deux fonctions à un argument. On pourra l'infixer avec la commande suivante `#infix "o";;`. Cela nous sera utile par la suite pour pouvoir composer des transformation du plan sans s'abimer les méninges

Question 2. Définir π (de type float) à l'aide des fonctions trigonométriques de Caml, à savoir `sin, cos, tan, asin, acos` et `atan`.

Question 3. Écrire les opérations géométriques suivantes :

```
minus : point -> point
translation : point -> point -> point
symmetry : point -> point -> point
rotation_origin : float -> point -> point
rotation : point -> float -> point -> point
homothety : float -> point -> point
```

`minus` est la symétrie centrale de centre l'origine, `symmetry` est la même pour un centre quelconque et `homothety` a pour centre l'origine.

Question 4. Les coordonnées des pixels sont des entiers tandis que notre type de points manipule des nombres à virgule flottante. Écrire une fonction `round` d'arrondi entre flottants et entiers. Noter que `int_of_float` fait une troncature, i.e. si $k \in \mathbb{N}$, elle transforme l'intervalle $[k, k+1[$ en $\{k\}$ alors que l'on voudrait que ce soit l'intervalle $[k-2, k+1[$ qui soit envoyé sur $\{k\}$.

Question 5. Écrire des fonctions qui respectivement trace un segment entre deux points et affiche un point.

```
draw_line : point -> point -> unit
plot_pt : point -> unit
```

2 Construction par étapes

2.1 La courbe de von Koch

Cette fractale, peut-être la plus connue, est obtenue en remplaçant chaque segment par un « segment à pic » : on coupe le segment en tiers et on substitue la partie centrale par un pic de triangle équilatéral. On réitère ensuite sur chacun des quatre nouveaux segments.

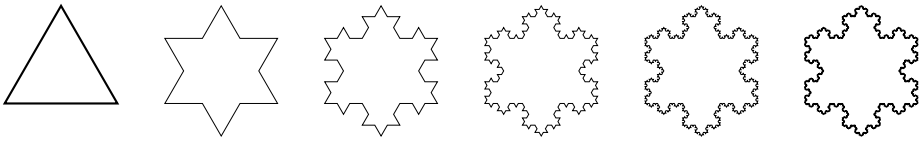


FIGURE 1 – Les premières étapes du flocon de von Koch

Question 6. Écrire une fonction de découpe qui réalise une étape de la construction de la courbe en transformant un segment en un « segment à pic ». Elle prendra en arguments les coordonnées des extrémités du segment initial et renverra le vecteur des cinq points extrémités des nouveaux segments :

```
koch_transformation : point -> point -> point vect
```

Question 7. Écrire une fonction qui dessine la n^e étape de la courbe de von Koch sur un segment délimité, de type

```
koch_draw : point -> point -> int -> unit
```

2.2 Généralisation

Question 8. Généraliser la fonction de la question précédente pour qu'elle prenne en argument supplémentaire la fonction de découpe. La tester sur `cross`.

Question 9. Généraliser à nouveau la fonction de la question 7 pour une fonction de découpe qui permet d'inverser le sens de dessin, donc de type `point -> point -> point vect * bool vect`. Les variables booléennes supplémentaires indiqueront le sens du tracé : si le i^e élément du vecteur est vrai, alors il faut réaliser la transformation sur le segment entre les i^e et $(i+1)^e$ points du $(i+1)^e$ au i^e et non l'inverse. L'essayer sur la fonction `dragon` ou `sierpinski`.

3 Iterated Function System

Le principe de cette technique est de représenter une fractale (ou plus généralement une image) comme l'unique ensemble A dont la réunion des images par un ensemble $\mathcal{F}$ de fonctions contractantes donne encore A, autrement dit $A = \bigcup_{f\in\mathcal{F}} f(A)$. Cet ensemble, appelé l'attracteur de l'IFS, est unique par théorème de point fixe.
Pour la construire, on répète le procédé suivant en partant d'un point courant p quelconque du plan :

- tirer au sort une fonction f de $\mathcal{F}$
- modifier le point courant en $f(p)$
- colorier le pixel le plus proche du point courant

Pour obtenir une image acceptable, le nombre d'itérations est en général de l'ordre de la centaine de milliers, voire du million.

Question 10. Écrire une fonction qui prend $\mathcal{F}$ et le point courant et renvoie le nouveau point courant. Elle sera de type

```
new_point : ( point -> point ) vect -> point -> point
```

Question 11. Écrire une fonction qui prend en argument $\mathcal{F}$ et le nombre d'itérations et applique l'algorithme précédent. Essayer cette technique de génération de fractales avec les fonctions `square_ifs, sierpinski_ifs, dragon_ifs, pentagon_ifs` ou `maple_ifs`.

Question 12. Trouver l'ensemble $\mathcal{F}$ de fonctions qui génère la courbe de von Koch. [1]

Question 13. Ajouter la gestion des niveaux de gris : à chaque fonction de F est associée une probabilité de tirage et plutôt que de le colorier en noir, on assombrit chaque nouveau pixel de la probabilité de la fonction tirée. On utilisera pour cela la fonction `darken : float -> point -> unit : darken col p` assombrit de `col` le pixel `p`. Tester cette version avec les IFS `maple_proba` et `fern_proba`.

1. *Pensez à décomposer la courbe en parties qui soient l'image de la courbe totale par des applications contractantes à déterminer*