TP Caml 7: Fractales

lionel.rieg@ens-lyon.fr
d'après un sujet de Jean-Baptiste Rouquier

27 janvier & 3 février 2009

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 voit la fractale comme le point fixe d'un ensemble de fonctions contractantes du plan. Les fonctions de tests sont fournies dans le fichier de modèle template.ml, situé dans le répertoire R:\TP07\ comme d'habitude. Les graphiques ne peuvent être réalisés que sous le *toplevel Caml Light* et non depuis un éditeur. Il vous faudra donc copier votre travail, par exemple à l'aide d'un include "nom−de−fichier";;.

1 Géométrie en Caml Light

Ne pas oublier au début de votre fichier la commande

```
#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 puis 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";;
```

► **Question 2** Définir pi (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. En revanche, 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 qu'on veut que ce soit l'intervalle $\left[k - \frac{1}{2}, k + \frac{1}{2}\right[$ qui soit envoyé sur $\{k\}$.

► **Question 5** Écrire des fonctions

```
draw_line : point → point → unit
plot_pt : point → unit
```

qui respectivement trace un segment entre deux points et affiche un point.

2 Construction par étapes

2.1 La courbe de von Koch

Cette fractale 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.

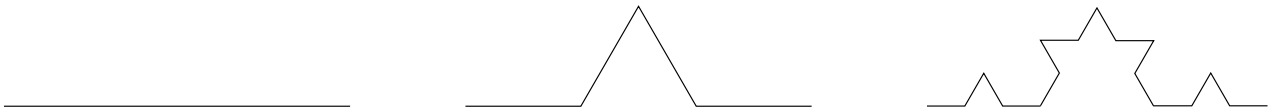

FIGURE 1 – Les trois premières étapes de la courbe de von Koch

► **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 Autres fractales

► **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, la fonction de découpe pour obtenir le triangle de 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. (penser à décomposer la courbe en parties qui soient l'image de la courbe totale par des applications contractantes à déterminer)

► **Question 13** Ajouter la gestion des niveaux de gris : à chaque fonction de $\mathcal{F}$ 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.

4 Solutions

► **Question 1**

```
let o f g x = f (g x);;
# infix "o";;
```

► **Question 2**

```
let pi = acos (-. 1.);;
```

► **Question 3**

```
let minus p =
  { x = -. p.x ; y = -. p.y };;
let translation v p =
  { x = v.x +. p.x ; y = v.y +. p.y };;
let symmetry c p =
  { x = 2. *. c.x -. p.x ; y = 2. *. c.x -. p.y };;
let rotation_origin angle p =
  { x = p.x *. cos angle -. p.y *. sin angle;
    y = p.y *. cos angle +. p.x *. sin angle };;
let rotation c angle =
  (translation c) o (rotation_origin angle) o (translation (minus c));;
let homothety k p =
  { x = k *. p.x ; y = k *. p.y };;
```

► **Question 4**

```
let round f = int_of_float (f +. 0.5);;
```

► **Question 5**

```
let draw_line a b =
  moveto (round a.x) (round a.y);
  lineto (round b.x) (round b.y);;

let plot_pt p = plot (round p.x) (round p.y);;
```

► **Question 6**

```
let koch_transformation a b =
  let m = ((homothety (1. /. 3.)) o (translation (minus a))) b in
  let pi_3 = pi /. 3. in
  let pt_2 = translation m a in
  let pt_4 = translation m pt_2 in
  let pt_3 = rotation pt_2 pi_3 pt_4 in
    [| a; pt_2; pt_3; pt_4; b |];;
```

► **Question 7**

```
let rec koch_draw a b n =
  if n = 0 then draw_line a b
  else
    let step = koch_transformation a b in
      for i = 0 to vect_length step - 2 do
        koch_draw step.(i) step.(i + 1) (n - 1)
      done;;
```

▸ **Question 8**

```
let rec draw transfo a b n =
  if n = 0 then draw_line a b
  else
    let step = transfo a b in
      for i = 0 to vect_length step - 2 do
        draw transfo step.(i) step.(i + 1) (n - 1)
      done;;
```

▸ **Question 9**

```
let rec bdraw transfo a b revd n =
  if n = 0 then draw_line a b
  else
    let step, reversed = if revd then transfo b a else transfo a b in
      for i = 0 to vect_length step - 2 do
        bdraw transfo step.(i) step.(i + 1) reversed.(i) (n - 1);
      done;;
```

▸ **Question 10**

```
let new_point f_array p =
  f_array.(random__int (vect_length f_array)) p;;
```

▸ **Question 11**

```
let draw_ifs f_array n =
  let p = ref { x = float_of_int (random__int (size_x ())) ;
                y = float_of_int (random__int (size_y ())) } in
    for i = 1 to n do
      p := new_point f_array !p;
      plot_pt !p
    done;;
```

▸ **Question 12**

```
let koch_ifs width height = (* Courbe de von Koch *)
  let size = min (float_of_int width) (float_of_int (height * 6) /. sqrt 3.) in
  let h = homothety (1. /. 3.) in
  let side = size /. 3.2 in
    [| h;
       translation {x = side; y = 0.} o rotation_origin (pi /. 3.) o h;
       translation {x = side *. 1.5; y = side *. sqrt 3. /. 2.}
         o rotation_origin (-. pi /. 3.) o h;
       translation {x = 2.*. side; y = 0.} o h |];;
```

▸ **Question 13**

```
let proba_point f_array p =
  let r = random__float 1. in
  let i = ref 0 in
    while r > snd f_array.(!i) do incr i done;
    let f, proba = f_array.(!i) in
      f p, proba;;

let sum_array f_array =
```

```
  let new_farray = copy_vect f_array in
    for i = 1 to vect_length f_array - 1 do
      let f, p = f_array.(i) in
        new_farray.(i) <- f, p +. snd new_farray.(i - 1)
    done;
    new_farray;;

let proba_ifs mult f_array n =
  let new_farray = sum_array f_array in
  let p = ref { x = float_of_int (random__int (size_x ())) ;
                y = float_of_int (random__int (size_y ())) } in
    for i = 1 to n do
      let new_p,prob = proba_point new_farray !p in
        darken (round (mult *. prob)) new_p;
        p := new_p
    done;;
```

Et en bonus, une fonction qui affiche successivement tous les ifs :

```
let full_test_ifs width height =
  let f_width = float_of_int width in
  let test_ifs draw ifs iter =
    clear_graph ();
    draw ifs iter ;
    ignore (read_key ())
  in
    open_graph ( string_of_int width ^ "x" ^ string_of_int height );
    ignore (read_key ());
     test_ifs draw_ifs (square_ifs width height) 100000;
     test_ifs draw_ifs ( sierpinski_ifs width height) 100000;
     test_ifs draw_ifs (koch_ifs width height) 50000;
     test_ifs draw_ifs (dragon_ifs width height) 150000;
     test_ifs draw_ifs (maple_ifs width height) 150000;
     test_ifs draw_ifs ( translate 300. 20. (pentagon_ifs width height)) 100000;
     test_ifs (proba_ifs 30.)
       ( proba_translate (f_width /. 2.) 20. (fern_proba width height)) 200000;
     test_ifs (proba_ifs 30.) (maple_proba width height) 200000;
    close_graph ();;
```