```
open Graphics
#load "graphics.cma";;

type point = {x:float; y:float}

let (%) f g x = f (g x)

let translation vect point =
  {x = point.x +. vect.x; y = point.y +. vect.y}

let rotation_origine angle point =
  let cosa = cos angle and sina = sin angle in
  {x = cosa *. point.x  -.  sina *. point.y;
   y = cosa *. point.y  +.  sina *. point.x}

let oppose {x=x; y=y} = {x= -.x; y = -.y}

let rotation centre angle =
  (translation centre) % (rotation_origine angle) % (translation (oppose centre))

let homothetie r {x=x; y=y} = {x= x*.r; y= y*.r}

let distance a b =
  let ab = translation (oppose a) b in
  sqrt (ab.x *. ab.x  +.  ab.y *. ab.y)

let pi = 2. *. acos 0.

(* ********************************************************************** *)
let iof = int_of_float

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

let decoupe a b =
  let c = (translation a % homothetie (1./.3.) % translation (oppose a)) b in
  let e = (translation a % homothetie (2./.3.) % translation (oppose a)) b in
  let d = rotation c (pi/.3.) e in
  [|a; c; d; e; b|]

let rec draw_koch profondeur a b =
  if profondeur = 0 then draw_line a b
  else
    let [|a;b;c;d;e|] = decoupe a b in
    let p = pred profondeur in
    draw_koch p a b;
    draw_koch p b c;
    draw_koch p c d;
    draw_koch p d e

let () =
  open_graph "";
  let width = size_x () in
  let a,b = {x = 0.; y = 0.}, {x=float width; y = 0.} in
  for profondeur = 1 to 8 do
    draw_koch profondeur a b;
    ignore (Graphics.read_key ());
    clear_graph ();
  done;
  close_graph ()

let rec draw decoupe profondeur a b =
  if profondeur = 0 then draw_line a b
  else
    let points, retournes = decoupe a b in
    let p = pred profondeur in
    (for i=1 to Array.length points -1 do
       if retournes.(pred i)
       then draw decoupe p points.(i) points.(pred i)
       else draw decoupe p points.(pred i) points.(i)
     done)

let decoupe_sierpinski a b =
  let m = (homothetie 0.5 % translation (oppose a)) b in
  let c = (translation a %  rotation_origine (pi/.3.)) m in
  let d = translation m c in
  [|a; c; d; b |], [|true; false; true|]

let decoupe_dragon a b =
  let c = (translation a % rotation_origine (pi/.4.) % homothetie (sqrt 2. /.2.) % translation
 (oppose a)) b in
  [|a; c; b|],
  [|false; true|]
```

```
let decoupe_3 a b =
  let m = translation (oppose a) b in
  let c = (translation a % homothetie 0.47) m in
  let milieu = homothetie 0.5 m in
  let d = (translation a % translation milieu % rotation_origine (pi/.2.) % homothetie 0.95) m
ilieu in
  let e = (translation a % homothetie 0.53) m in
  [|a; c; d; e; b|], [|false; false; false; false|]

let main decoupe profondeur_max =
  open_graph "";
  let width = size_x () in
  let a,b = {x = 0.; y = 0.}, {x=float width; y = 0.} in
  for profondeur = 1 to profondeur_max do
    draw decoupe profondeur a b;
    ignore (Graphics.read_key ());
    clear_graph ();
  done;
  close_graph ()

let () =
  main decoupe_sierpinski 13;
  main decoupe_dragon 18;
  main decoupe_3 10

(* ********************************************************************** *)

let etape point transformations =
  transformations.(Random.int (Array.length transformations)) point

let main transformations iterations =
  clear_graph ();
  let point = ref {x= float (size_x ()) /. 2.; y= float (size_y ()) /.2.} in
  for i=1 to iterations do
    point := etape !point transformations;
    plot (int_of_float !point.x) (int_of_float !point.y);
  done;
  ignore (read_key ())

let () =
  open_graph "";
  let width = float (size_x ()) and height = float (size_y ()) in
  let f = homothetie 0.45 in
  let cote = 0.55 *. height in
  main
    [| f;
       translation {x=cote;y=0.} % f;
       translation {x=cote;y=cote} % f;
       translation {x=0.;y=cote} % f|]
    100_000;
  let f = homothetie 0.5 in
  main (* Triangle de Sierpinski *)
    [| f;
       translation {x=0.5 *. width;  y=0.} % f;
       translation {x=0.25 *. width;  y= width *. sqrt 3. /. 4.} % f|]
    100_000;
  let f = homothetie (1./.3.) in
  let cote = width /. 3. in
  main (* courbe de Von Koch *)
  [| f;
     translation {x=cote; y=0.} % rotation_origine (pi /. 3.) % f;
     translation {x=cote *. 1.5; y=cote*.sqrt 3. /. 2.} % rotation_origine (-. pi /. 3.) % f;
     translation {x=2.*.cote; y=0.} % f
  |]
  100_000;
  close_graph ()
```