```
open Constantes
open Maths
open Types

let fold_left = Array.fold_left
let mapi = Array.mapi
let map = Array.map
let map2 = Array.map2
let init = Array.init
let iteri = Array.iteri

let ( *%) q = map (fun t -> t *$ q)
let ( +%) = map2 (+$)
let ( -%) = map2 (-$)

(*la figure intiale *)
let initial () =
  Array.map
    (fun (x,y,z) ->
      let r = rayon in
        {
          pos = x*.r,y*.r,z*.r +. rayon +. hauteur_initiale;
          vit = 0.0,0.0,0.0;
          mass = mass;
        }
    )
    Icosphere.icosphere

let triangles_with i l center =
  (*renvoie la liste des aires et des vecteurs normaux des triangles voisins à i*)
  let is_in (a,b,c) = a = i || b = i || c = i in
  List.filter is_in Icosphere.indices_triangles
  |> List.map (fun (i,j,k) -> area (l.(i).pos,l.(j).pos,l.(k).pos), normal l.(i).pos
 l.(j).pos l.(k).pos center)

(*tableu de liste d'adjacence, avec la distance d'équilibre*)
let g =
  let l = initial () in
  let g = Array.make Icosphere.n [] in
  List.iter
    (fun (i,j,k) ->
      g.(i) <- j::g.(i); g.(i) <- k::g.(i);
      g.(j) <- i::g.(j); g.(j) <- k::g.(j);
      g.(k) <- i::g.(k); g.(k) <- j::g.(k)
    )
    Icosphere.indices_triangles;
  let g = Array.map Icosphere.unique g in
  Array.mapi
    (fun i l' ->
      List.map (fun j -> j, dist l.(i).pos l.(j).pos)
    l')
  g

(*renvoie les sommets adjacents à i et la longeur à vide*)
let linked_to l i = List.map (fun (j,l0) -> l.(j), l0) g.(i)

(*creation d'un graph a partir d'un tableau de positions et de vitesses*)
let to_points y y' blob =
  init Icosphere.n (fun i-> {pos=y.(i); vit = y'.(i); mass = blob.(i).mass})

(*renvoie la liste des triangles*)
let surfaces l = List.map (fun (i,j,k) -> l.(i),l.(j),l.(k)) Icosphere.indices_trian
gles
```