TP 7— Géométrie plane

http://www.gallium.inria.fr/~scherer/teaching/colles/2012/tp7.pdf

Les questions en rapport avec les TP, Caml ou la programmation en général sont les bienvenues et peuvent être envoyées à gabriel.scherer@gmail.com. N'hésitez pas !

Cette colle est un sujet donné par Victor Nicollet il y a maintenant de nombreuses années, repris intégralement. Les fautes de frappe ne sont pas de moi.

Caml Light propose une librairie graphique qui permet de dessiner à l'écran des figures géométriques simples: points, segments, polygones, cercles. L'objectif de ce TP est d'illustrer l'utilisation de certaines de ces fonctionnalités, ainsi que des problèmatiques de base en géométrie algorithmique.

```
#open "graphics";;
open_graph "640x480";;

let nettoyer () =
  set_color black; clear_graph();;
```

On utilisera le code ci-contre pour initialiser le système graphique, et définir une fonction nettoyer, utilisée pour revenir à un état propre de la librairie.

1 Couleurs

Une couleur est représentée par un ordinateur comme trois composantes distinctes: rouge, vert et bleu. Ces valeurs sont entre 0 (absent) et 255 (présence maximale). Des couleurs usuelles sont préprogrammées en Caml Light:

couleur	Caml Light	R	V	B	couleur	Caml Light	R	V	B
noir	black	0	0	0	blanc	white	255	255	255
rouge	red	255	0	0	vert	green	0	255	0
bleu	blue	0	0	255	jaune	yellow	255	255	0
magenta	magenta	255	0	255	cyan	cyan	0	255	255
orange	non défini	255	128	0	gris	non défini	128	128	128

Une couleur qui n'est pas prédéfinie peut être spécifiée à l'aide de la fonction rgb r g b, comme ci-contre. La fonction set_color permet de modifier la couleur actuellement utilisée par le langage.

```
let orange = rgb 255 128 0 in
set_color orange;;
```

▷ **Question 1.** La fonction plot x y dessine un pixel au point (x, y) avec la couleur actuelle, $(0, 0)$ étant le coin inférieur gauche.

Dessiner un carré de taille 255×255 dans ce coin, dont la couleur varie graduellement entre bleu (coin inférieur gauche), magenta (inférieur droit), vert (supérieur gauche) et jaune (supérieur droit). ◁

2 Enveloppes convexes

On considère désormais des points: on les prendra de type int * int (des couples d'entiers, représentant respectivement les coordonnées horizontales et verticales). On regroupe ces points en un nuage aléatoire (une liste de points dont les coordonnées ont été prises au hasard).

▷ **Question 2.** Écrire une fonction qui crée un nuage de points de taille n (où n est un argument de la fonction). L'utiliser pour créer un nuage de 100 points. Écrire une fonction qui dessine un nuage de points. On pourra utiliser la fonction do_list pour en simplifier l'écriture. ◁

L'enveloppe convexe d'un nuage de points est le plus petit ensemble convexe contenant tous les points du nuage. Cette enveloppe est en fait un polygone composé de segments reliant des points du nuage, dessinés avec la fonction ci-contre.

```
let segment (x1,y1) (x2,y2) =
  moveto x1 y1;
  lineto x2 y2;;
```

L'explication la plus simple est la «méthode de l'élastique»: si on étire un élastique autour du nuage de points, puis qu'on le relâche, il va se reserrer autour des points et former un polygone qui se trouve être l'enveloppe convexe du nuage. Il est bien sûr immédiat que ce polygone a pour sommets des points du nuage. Nous allons donc écrire une fonction qui examine tous les segments entre les points, et ne dessiner que les segments de l'enveloppe convexe.

▷ **Question 3.** Écrire une fonction qui prend en entrée une fonction de dessin (par exemple, la fonction `segment`) et une liste de points, et appelle la fonction une et une seule fois sur tous les couples de points de la liste. On pourra chercher une solution récursive consistant à relier un point aux autres, puis tous les autres entre eux. Complexité? ◁

Nous allons utiliser ici une caractérisation particulière des segments de l'enveloppe convexe: en effet, il s'agit des paires de points telles que tous les points du nuage se trouvent d'un seul côté du segment qui les relie.

▷ **Question 4.** Dessiner l'enveloppe convexe d'un nuage de points. On pourra, pour cela:

- Définir un type somme Gauche-Milieu-Droite pour représenter la position d'un point.
- Écrire une fonction qui, étant donnés trois points, renvoie la position du troisième par rapport au segment reliant les deux premiers.
- Écrire une fonction qui renvoie une paire d'entiers décrivant combien de points d'un nuage se trouvent à gauche et à droite d'un segment donné.
- Définir une fonction de dessin qui n'affiche un segment que s'il est dans l'enveloppe convexe.

La fonction de la question précédente pourra être utile pour traiter tous les segments possibles. Complexité? ◁

3 TRIANGULATION

Une triangulation consiste à recouvrir l'intérieur de l'enveloppe convexe à l'aide de triangles dont les sommets sont les points du nuage. Une caractérisation équivalente dit qu'une triangulation est un sous-ensemble de l'ensemble de tous les segments reliant entre eux des points du nuage. Ce sous-ensemble est obtenu à partir de l'ensemble entier en retirant des segments qui intersectent d'autres segments ailleurs qu'aux extrémités, jusqu'à ce que plus aucune intersection n'existe.

▷ **Question 5.** Dessiner une triangulation d'un ensemble de points. On pourra pour cela:

- Déterminer si deux segments s'intersectent (cela se produit lorsque les deux sommets de chaque segment sont de côtés opposés de l'autre segment).
- Écrire une fonction qui détermine si un segment donné intersecte au moins un segment d'une liste de segments donnés.
- Écrire une fonction qui mémorise les segments déjà dessinés et, au moment d'en dessiner un nouveau, vérifie qu'il n'intersecte aucun autre segment avant de le dessiner.

Cette dernière fonction sera passée en argument à la fonction de la question 3. Complexité? ◁

Malheureusement, cette approche donne un résultat assez laid: en effet, le premier point se voit relier à tous les autres, ce qui crée des triangles aplatis et difformes recouvrant l'essentiel de l'enveloppe convexe. On va donc modifier l'ordre dans lequel les segments sont examinés pour obtenir une nouvelle figure.

▷ **Question 6.** Modifier la fonction de la question 3 pour qu'elle retire un point, relie récursivement entre eux les points restants, puis relie le point retiré aux autres. En déduire une nouvelle fonction de triangulation. ◁

Le résultat est plus arrangeant, mais pas encore parfait. Une solution idéale est connue sous le nom de triangulation de Delaunay, mais est trop complexe à construire pour être présentée ici. Nous allons donc proposer une solution alternative partant de la remarque suivante: les triangles les plus dégénérés ont des côtés très longs. Ainsi, introduire en premier les sommets courts diminue la probabilité que des côtés longs apparaissent. Il y aura bien sûr des triangles aplatis, mais ceux-ci seront inévitablement plus petits.

▷ **Question 7.** Écrire une fonction qui trie une liste de segments suivant la longueur du segment (dans l'ordre croissant). Modifier alors la fonction de la question 3 pour dessiner les segments par ordre de longueur croissant, et en déduire une troisième fonction de triangulation. ◁

4 Surface liquide

Nous allons maintenant simuler la surface (unidimensionnelle) d'un fluide visqueux. Pour cela, nous allons découper cette surface en 640 points horizontaux, et stocker la hauteur de la colonne de liquide en chacun de ces points dans un tableau de réels.

Le code ci-contre définit la fonction `tourner`, qui prend en argument une fonction `evolue` utilisée pour changer l'état de la surface à chaque appel. Pour que ce code puisse fonctionner, il faut commencer par définir la taille `size` de la surface, le tableau `surface` de réels représentant les hauteurs et une fonction `dessine_surface` qui dessine la surface à l'aide de lignes verticales de hauteur correcte.

```
let last_key () =
  if (wait_next_event
    [Poll;Key_pressed]).keypressed
  then read_key ()
  else 'x';;

let rec tourner (evolue) =
  evolue ();
  dessine_surface ();
  match last_key () with
    | 'q' -> ()
    | ' ' ->
      let (x,y) = mouse_pos () in
      if x < size && x >= 0 then
      surface.(x) <- (float_of_int y);
      tourner (evolue);
    | _ ->
      tourner (evolue);;
```

Dans un premier temps, nous allons utiliser une équation simple: en considérant le tableau comme cyclique (donc les indices étant pris modulo `size`), la nouvelle valeur de chaque case après évolution est égale à la moyenne de son ancienne valeur et de l'ancienne valeur de ses deux voisines. Il s'agit donc d'un filtre de moyennage qui tend naturellement à rendre la surface aussi horizontale que possible.

▷ **Question 8.** Écrire une fonction qui prend en argument un tableau de taille `size` et lui applique un filtre de moyenne de largeur trois. L'utiliser pour définir `evolue` et tester le programme obtenu. ◁

Nous allons maintenant tenter d'obtenir un liquide un peu plus dynamique, où des masses se déplacent sous l'effet de l'inertie (créant ainsi des vagues). Pour cela, nous allons utiliser un tableau de vitesses, et faire varier les vitesses à partir des différences de hauteur, suivant des lois simples:

- On applique deux fois un filtre moyenne sur les vitesses restées de l'état précédent, pour simuler l'inertie.
- On applique le même filtre moyenne sur les hauteurs, comme précédemment, pour simuler l'effet de tension superficielle.
- Pour chaque paire de colonnes adjacentes (le tableau étant cyclique) on calcule la différence δ de hauteur entre les deux. La vitesse de vidage de la plus grande augmente de $\delta\alpha$, et la vitesse de remplissage de la plus petite augmente d'autant, pour simuler la pression.
- On multiplie les vitesses par β pour simuler la dissipation d'énergie.
- On ajoute la vitesse en chaque point à la hauteur en ce point.

On prendra pour paramètres $\alpha = 0.5$ et $\beta = 0.98$.

▷ **Question 9.** Implémenter les lois décrites ci-dessus. Que se passe-t-il lorsqu'on augmente trop les paramètres? Lorsqu'on les diminue trop? ◁

5 Questions difficiles

▷ **Question 10.** Quel est l'intérêt d'un *back-buffer* et de la synchronisation avec l'écran? ◁

▷ **Question 11.** Écrire un algorithme pour dessiner une droite passant graduellement d'une couleur à une autre, les couleurs et positions des extrémités étant passées en argument. ◁

▷ **Question 12.** Écrire un algorithme pour dessiner un triangle passant graduellement d'une couleur à une autre, les trois sommets ayant des couleurs et des positions arbitraires passées en argument. S'assurer qu'il n'y a pas de trou non dessiné entre deux triangles partageant une arête. ◁