Feuille d'exercices n°12 de l'option d'informatique.

Autour de la 3D.

Le but de ce TD est d'avoir une représentation d'objets 3D définis par des facettes planes, de pouvoir faire tourner ces objets autour de l'origine et de les dessiner. Pour décrire de tels objets, on dispose d'un tableau de points à coordonnées réelles qui se présente comme un vecteur de triplets de réels `(float * float * float) vect`. Un objet est représenté par la liste de ses facettes et chaque facette par une liste d'entiers[1], chaque entier faisant référence à un élément du tableau de points précédemment décrit. Un objet sera donc du type `int list list`. Par exemple, dans le cas d'un cube centré en l'origine :

```
let points =
[|-.100.,-.100.,-.100.;100.,-.100.,-.100.;-.100.,100.,-.100.;100.,100.,-.100.;
-.100.,-.100.,100.;-.100.,100.,100.;100.,-.100.,100.;100.,100.,100.|];;
let cube =[[0;1;3;2;0];[0;2;5;4;0];[4;5;7;6;4];[2;3;7;5;2];[0;1;6;4;0];[1;3;7;6;1]];;
```

Le nombre π est obtenu de la manière suivante : `let Pi = atan2 0. (-.1.);;`

1. Écrire une fonction `convert` qui à partir d'un tableau de points du type `(float * float * float) vect` retourne un vecteur dont les composantes sont les vecteurs coordonnées des points sous forme de matrices à trois lignes et une colonne.

```
#convert points;;
- : float vect vect vect =
 [|[|[|-100.0|]; [|-100.0|]; [|-100.0|]|]; [|[|100.0|]; [|-100.0|]; [|-100.0|]|];
   [|[|-100.0|]; [|100.0|]; [|-100.0|]|];  [|[|100.0|]; [|100.0|]; [|-100.0|]|];
   [|[|-100.0|]; [|-100.0|]; [|100.0|]|]; [|[|-100.0|]; [|100.0|]; [|100.0|]|];
   [|[|100.0|]; [|-100.0|]; [|100.0|]|]; [|[|100.0|]; [|100.0|]; [|100.0|]|]|]
```

2. Écrire une fonction `dessine tpoints objet ymin ymax zmin zmax` qui affiche le projeté orthogonal d'un objet sur le plan Oyz. Le point $(ymin, zmin)$ correspond au point $(0, 0)$ de l'écran et le point $(ymax, zmax)$ doit correspondre au point `(size_x(),size_y())` de l'écran. `dessine` est du type `float vect vect vect -> int list list -> float -> float -> float -> float -> unit = <fun>`.

3. Écrire une fonction Caml `rotationx` qui, étant donné un angle θ, retourne la matrice de la rotation d'axe Ox et d'angle θ.

```
#rotationx (Pi/.3.);;
- : float vect vect =
 [|[|1.0; 0.0; 0.0|]; [|0.0; 0.5; -0.866025403784|]; [|0.0; 0.866025403784; 0.5|]|]
```

4. Écrire une fonction Caml `rotationz` qui, étant donné un angle ϕ, retourne la matrice de la rotation d'axe Oz et d'angle ϕ.

1. Pour simplifier, on travaille avec des polygones fermés, c'est à dire que systématiquement le premier point de la liste est égal au dernier.

5. Écrire une fonction `produit` qui, étant données deux matrices M et N respectivement de type (n, p) et (p, q), retourne la matrice produit $M.N$ de type (n, q).

6. Écrire une fonction `produit_vect M [|P0;...;Pn|]` qui, étant donné une matrice M et un tableau de points $[|P_0; ...; P_n|]$, retourne le tableau de points $[|M.P_0; ...; M.P_n|]$.
On pourra visualiser l'ensemble des fonctions précédentes (après avoir chargé la bibliothèque graphique et ouvert une fenêtre graphique) avec les commandes suivantes :

```
let tpoint = convert points;;
for i = 0 to 1000 do
   let I = float_of_int i in
   let M = produit (rotationz (0.01 *. I)) (rotationx (0.015 *. I)) in
    clear_graph();
    dessine (produit_vect M tpoints) cube (-200.) (200.) (-.150.) (150.);
done;;
```

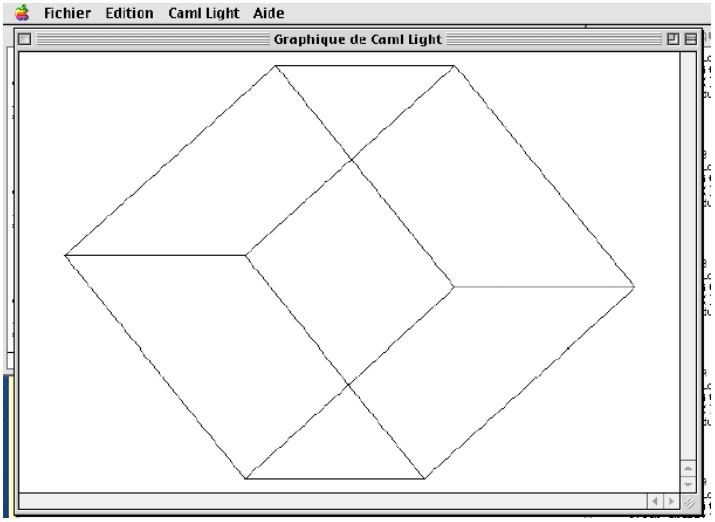

FIG. 1 – *Image du cube*

7. Pour donner un effet plus réaliste, on veut représenter un objet à faces pleines et donc dans notre visualisation, ne pas voir les arêtes cachées. Dans le cas d'un objet convexe comme pour le cube, il est facile de différencier les faces cachées de celles qui ne le sont pas. En effet, il suffit de regarder si la normale à la face[2] appartient au demi-plan des x positifs ou des x négatifs. Écrire une fonction «`vu`» qui, à partir d'un polygone, renvoie un booléen précisant si la face est vu ou non. (le vecteur normal à la face sera calculé par le produit vectoriel des vecteurs définissant les deux premières arêtes du polygone).

Réécrire une fonction d'affichage, n'affichant que les facettes visibles d'un objet.

2. Cela suppose que les faces définissant notre objet sont bien planes. Pour éviter d'avoir des faces gauches, bien souvent les modeleurs n'acceptent que des faces triangulaires ou bien décomposent les polygones en triangles. De plus, il faut faire attention à l'orientation de la surface pour que le vecteur normal soit dirigé vers l'extérieur de l'objet, ce qui a une influence sur l'ordre des points pour définir le polygone d'une facette de l'objet.