Corrigé feuille d'exercices n°12 de l'option d'informatique.

Autour de la 3D

Exercices n°1

Pour convertir le tableau de points en tableau de points sous forme matricielle

```
#let convert  p  =
let matrice_of_point (x,y,z) =
let P = make_matrix 3 1 x in
P.(1).(0) <- y;
P.(2).(0) <- z;
P in
let n = vect_length p in
let Pc = make_vect n [||] in
for i = 0 to n-1 do
  Pc.(i) <- matrice_of_point p.(i);
done;
Pc;;
```

Exercices n°2

La fonction `projete` calcule les coordonnées du projeté du point dans la fenêtre graphique.

```
#let projete point ymin ymax zmin zmax =
let y = point.(1).(0) and z = point.(2).(0) in
   let u = int_of_float (((y -. ymin) /. (ymax -. ymin)) *. (float_of_int larg)) and
   v = int_of_float (((z -. zmin) /. (zmax -. zmin)) *. (float_of_int haut)) in
(u,v);;
projete : float vect vect -> float -> float -> float -> float -> int * int = <fun>
#let dessine tpoint objet ymin ymax zmin zmax =
let rec trace_poly = function
  [] -> ()
| x::xs -> let (u,v) =   projete tpoint.(x) ymin ymax zmin zmax in
                lineto u v; trace_poly xs
 in
let rec affiche_polygone = function
 [] -> ()
| (p::ps) as polygone-> let (u,v) = projete (tpoint.(hd p)) ymin ymax zmin zmax in
                moveto u v; trace_poly p; affiche_polygone ps;
in
affiche_polygone objet
;;
dessine :
 float vect vect vect ->
  int list list -> float -> float -> float -> float -> unit = <fun>
```

Exercice n° 3

Pour créer une matrice de rotation d'axe $0x$ et d'angle «`theta`» en radian.

```
#let rotationx theta =
let M = make_matrix 3 3 0. in
M.(1).(1) <- cos(theta);
M.(2).(1) <- sin(theta);
M.(1).(2) <- -.sin(theta);
M.(2).(2) <- cos(theta);
M.(0).(0) <- 1.;
M;;

rotationx : float -> float vect vect = <fun>
```

Exercice n° 4

Pour créer une matrice de rotation d'axe $0z$ et d'angle «`phi`» en radian.

```
#let rotationz phi =
let M = make_matrix 3 3 0. in
M.(0).(0) <- cos(phi);
M.(1).(0) <- sin(phi);
M.(0).(1) <- -.sin(phi);
M.(1).(1) <- cos(phi);
M.(2).(2) <- 1.;
M;;

rotationz : float -> float vect vect = <fun>
```

Exercice n° 5

```
#let dimension M =
let n = vect_length M and p = vect_length M.(0) in
  (n,p);;
dimension : 'a vect vect -> int * int = <fun>
#let produit M N =
let (n,p) = dimension M and (r,q) = dimension N in
if p <> r then failwith "produit : les matrices n'ont pas la bonne dimension";
let P = make_matrix n q 0. in
for i = 0 to n-1 do
  for j = 0 to q-1 do
      let c = ref 0. in
         for k = 0 to p-1 do
             c := !c +. M.(i).(k) *. N.(k).(j)
         done;
       P.(i).(j) <- !c;
   done;
done;
P;;
produit : float vect vect -> float vect vect -> float vect vect = <fun>
```

Exercice n° 6

Pour faire le produit d'une matrice avec un tableau de points.

```
#let rec produit_vect M tpoints =
let n = vect_length tpoints in
let res = make_vect n tpoints.(0) in
for i = 0 to n-1 do
   res.(i) <- produit M tpoints.(i)
done;
res;;
produit_vect : float vect vect -> float vect vect vect -> float vect vect vect = <fun>
```

Exercice n° 7

```
#let moins P1 P2 =
let R = make_matrix 3 1 0. in
R.(0).(0) <- P1.(0).(0) -. P2.(0).(0);
R.(1).(0) <- P1.(1).(0) -. P2.(1).(0);
R.(2).(0) <- P1.(2).(0) -. P2.(2).(0);
R;;
moins : float vect vect -> float vect vect -> float vect vect = <fun>
#let prodvect P1 P2 =
let R = make_matrix 3 1 0. in
R.(0).(0) <- P1.(1).(0) *. P2.(2).(0) -. P1.(2).(0) *. P2.(1).(0);
R.(1).(0) <- P1.(2).(0) *. P2.(0).(0) -. P1.(0).(0) *. P2.(2).(0);
R.(2).(0) <- P1.(0).(0) *. P2.(1).(0) -. P1.(1).(0) *. P2.(0).(0);
R;;
prodvect : float vect vect -> float vect vect -> float vect vect = <fun>
#let normal points = function
 p1::p2::p3::ps -> prodvect (moins points.(p2) points.(p1))
                                (moins points.(p2) points.(p3))
| _ -> failwith "erreur dans normal";;
normal : float vect vect vect -> int list -> float vect vect = <fun>
#let vu points = function
 p1::p2::p3::ps as poly -> let r = normal points poly in
                          r.(0).(0) >= 0.;
| _ -> failwith "erreur dans vu";;

vu : float vect vect vect -> int list -> bool = <fun>
#let affiche_cache tpoint objet ymin ymax zmin zmax =
let rec trace_poly = function
  [] -> ()
| x::xs -> let (u,v) =   projete tpoint.(x) ymin ymax zmin zmax in
                lineto u v; trace_poly xs
 in
let rec affiche_polygone = function
 [] -> ()
| (p::ps) as polygone when vu tpoint p ->
                let (u,v) = projete (tpoint.(hd p)) ymin ymax zmin zmax in
                moveto u v; trace_poly p; affiche_polygone ps;
| p::ps -> affiche_polygone ps
in
affiche_polygone objet
;;

affiche_cache :
 float vect vect vect -> int list list
                -> float -> float -> float -> float -> unit = <fun>
```