Rappels mathématiques
TD n°0

Exercice 1 *Soit une direction normée (d_x, d_y, d_z) et une position (p_x, p_y, p_z) dans un espace 3D. Calculer les cœfficients a, b, c et d du plan P d'équation cartésienne $ax + by + cz + d = 0$ tels que la position de (p_x, p_y, p_z) soit incluse dans P et que la direction normée (d_x, d_y, d_z) soit orthogonale à P.*

Correction de l'exercice 1 – Tout vecteur du plan peut être considéré comme allant du point (p_x, p_y, p_z) à une autre point (q_x, q_y, q_z) vérifiant l'équation

$$aq_x + bq_y + cq_z + d = 0 \tag{1}$$

Ce vecteur a alors comme coordonnées $(q_x - p_x, q_y - p_y, q_z - p_z)$. Or s'il appartient au plan, alors il est orthogonal au vecteur (d_x, d_y, d_z) (qui est normal au plan) donc le produit scalaire entre les deux vecteurs est nul.

$$\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \cdot \begin{pmatrix} q_x - p_x \\ q_y - p_y \\ q_z - p_z \end{pmatrix} = 0 \tag{2}$$

$$d_x (q_x - p_x) + d_y (q_y - p_y) + d_z (q_z - p_z) = 0 \tag{3}$$

$$d_x x + d_y y + d_z z - (p_x d_x + p_y d_y + p_z d_z) = 0 \tag{4}$$

On en déduit que $a = d_x$, $b = d_y$, $c = d_z$ et $d = -(p_x d_x + p_y d_y + p_z d_z)$.

Exercice 2 *Soient deux facettes triangulaires définies individuellement par un triplet de sommets de $\mathbb{R}^3$. Comment tester le parallélisme de ces deux facettes ?*

Correction de l'exercice 2 – Soit les points A, B et C définissant la facette T_1 et L, M et N définissant la facette T_2. Si un vecteur normal à T_1 est normal à au moins deux vecteurs de T_2, alors T_1 et T_2 sont parallèles.

Pour obtenir un vecteur normal $\vec{n}$ à T_1, on utilise le produit vectoriel $\vec{n} = \vec{AB} \wedge \vec{AC}$. Il suffit de vérifier à présent que $\vec{n} \cdot \vec{LM} = 0$ et $\vec{n} \cdot \vec{LN} = 0$.

Exercice 3 *Quatre points constitue un tétraèdre dans l'espace.*

$$A = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \quad C = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} \quad D = \begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix} \tag{5}$$

avec les faces suivantes (voir figure 1 page suivante)

$$F_1 = \{A, B, D\} \quad F_2 = \{A, C, D\} \quad F_3 = \{B, C, D\} \quad F_4 = \{A, B, C\} \tag{6}$$

L'observateur, en $\vec{z}$ positif, regarde dans la direction $(d_x, d_y, d_z) = (0, 0, -1)$.

1. *Calculer le vecteur normal $\vec{n_i}$ pour chacune des faces F_i de manière à ce que le vecteur soit orienté vers l'intérieur du polyèdre.*
2. *Déduire les faces visibles pour l'observateur et les faces cachées.*

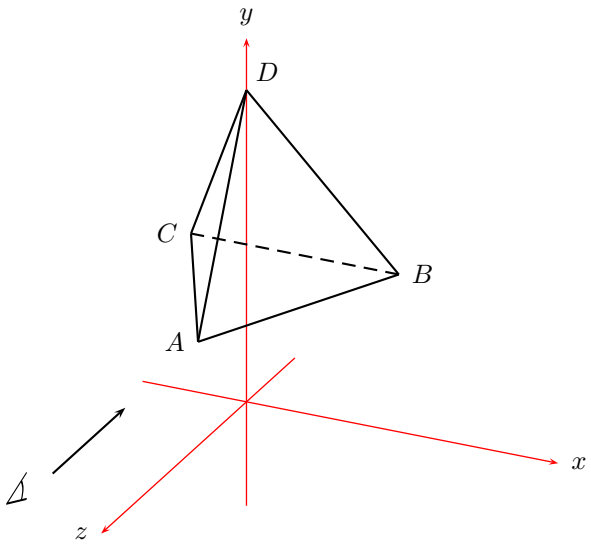

Figure 1 – Tétraèdre

Correction de l'exercice 3 – Pour obtenir les vecteurs $\vec{n_i}$, il suffit d'utiliser le produit vectoriel (attention au sens du produit).

$$\vec{n_1} = \vec{AD} \wedge \vec{AB} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix} \wedge \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} -4 \\ -1 \\ -2 \end{pmatrix} \tag{7}$$

$$\vec{n_2} = \vec{AC} \wedge \vec{AD} = \begin{pmatrix} -1 \\ 0 \\ -2 \end{pmatrix} \wedge \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 4 \\ -1 \\ -2 \end{pmatrix} \tag{8}$$

$$\vec{n_3} = \vec{BD} \wedge \vec{BC} = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \wedge \begin{pmatrix} -2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -2 \\ 4 \end{pmatrix} \tag{9}$$

$$\vec{n_4} = \vec{AB} \wedge \vec{AC} = \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} \wedge \begin{pmatrix} -1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix} \tag{10}$$

Pour savoir si les faces sont visibles, il suffit de calculer le produit scalaire entre la direction du regard et les vecteurs $\vec{n_i}$.

$$\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = -n_z \tag{11}$$

Si le produit scalaire est positif (les deux vecteurs sont dans la même direction), alors les faces sont visibles. Il suffit donc de vérifier que $n_z < 0$. Les faces F_1 et F_2 sont visibles. La face F_4 est invisible. La face F_3 est parallèle à la direction du regard.

Exercice 4 *Répondre aux consignes suivantes en langage* C *:*

1. *Définir une structure* `point` *et* `vector`
2. *Calcul de la distance entre deux points*
3. *Normaliser une direction*

4. *Calcul du produit scalaire*
5. *Calcul du produit vectoriel*

Correction de l'exercice 4 –

```
typedef struct point_s
{
  float x;
  float y;
  float z;
} point;
float distance(point p1, point p2)
{
  float dx = p2.x - p1.x;
  float dy = p2.y - p1.y;
  float dz = p2.z - p1.z;
  return ((float)sqrt(dx*dx + dy*dy + dz*dz));
}

typedef struct vector_s
{
  float x;
  float y;
  float z;
} vector;
vector normalize(vector v)
{
  norm = sqrt(v.x*v.x + v.y*v.y + v.z*v.z) ;
  vector v_out ;
  v_out.x = v.x/norm;
  v_out.y = v.y/norm;
  v_out.z = v.z/norm;
  return v_out;
}
float dot_product(vector v1, vector v2)
{
  float d_out = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
  return d_out;
}
vector cross_product(vector v1, vector v2)
{
  vector v_out;
  v_out.x = v1.y*v2.z - v1.z*v2.y;
  v_out.y = v1.z*v2.x - v1.x*v2.z;
  v_out.z = v1.x*v2.y - v1.y*v2.x;
  return normalize(v_out);
}
```

Exercice 5 *Soit l'axe $\vec{d}$ défini par les deux points $(2,1,3)$ et $(4,2,-1)$. Définir la matrice de transformation de rotation de 60° autour de l'axe $\vec{d}$.*

On exprimera cette matrice sous la forme d'un produit de matrices canoniques. Il n'est pas nécessaire d'effectuer les multiplications ; nous considérons avoir les outils suffisants pour effectuer ce produit.

Correction de l'exercice 5 – Pour effectuer la transformation demandée, sept transformations géométriques successives sont nécessaires.

1. Nous ne connaissons que les matrices canoniques de rotation par rapport à l'origine. Il nous faut donc commencer par ramener à l'origine par une matrice de translation T. Les vecteurs de translation possibles sont $(-2,-1,-3)$ et $(-4,-2,1)$ ou tout point $(-2-4\lambda,-1-2\lambda,-3+\lambda)$ avec $\forall\lambda\in\mathbb{R}$.

$$T=\begin{pmatrix}1&0&0&-2\\0&1&0&-1\\0&0&1&-3\\0&0&0&1\end{pmatrix} \tag{12}$$

2. Une rotation R_1 dans le sens direct autour de l'axe $\vec{y}$ pour ramener l'axe $\vec{d}$ dans le plan (yOz). L'angle θ_y de cette rotation est tel que $\tan(\theta_y)=\frac{2}{4}$ soit $\theta_y\approx 26{,}565^\circ$.

$$R_1=\begin{pmatrix}\cos\theta_y&0&\sin\theta_y&0\\0&1&0&0\\-\sin\theta_y&0&\cos\theta_y&0\\0&0&0&1\end{pmatrix} \tag{13}$$

3. Une rotation R_2 dans le sens indirect autour de l'axe $\vec{x}$ pour ramener l'axe $\vec{d}$ sur l'axe $\vec{z}$. La valeur absolue de l'angle θ_x de cette rotation est tel que $\sin(\theta_x)=\frac{1}{\sqrt{2^2+(-1)^2+4^2}}$ soit $\theta_x\approx 12{,}604^\circ$.

$$R_2=\begin{pmatrix}1&0&0&0\\0&\cos\theta_x&-\sin\theta_x&0\\0&\sin\theta_x&\cos\theta_x&0\\0&0&0&1\end{pmatrix} \tag{14}$$

4. Une rotation R d'angle 60° autour de l'axe $\vec{z}$.

$$R=\begin{pmatrix}\cos 60&-\sin 60&0&0\\\sin 60&\cos 60&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix} \tag{15}$$

5. On effectue à présent les transformations inverses afin de se replacer dans le repère de départ. Pour commencer, on effectue la rotation R_2^{-1}.
6. Puis on effectue la rotation R_1^{-1}.
7. Et enfin, on effectue la translation T^{-1}.

La matrice de transformation finale M est la suivante.

$$M=T^{-1}R_1^{-1}R_2^{-1}RR_2R_1T \tag{16}$$

Exercice 6 *Comment tester la planarité d'un polygone de* $\mathbb{R}^3$.

Correction de l'exercice 6 – Pour avoir un plan, il nous faut trois points. Si le polygone possède trois points P_1, P_2 et P_3, aucun test n'est nécessaire.

Pour les polygones de plus de trois points, on calcule la normale $\vec{n}$ au plan défini par les trois premiers points P_1, P_2 et P_3. Ensuite, on testera chaque nouveau point P_i en effectuant un produit scalaire entre $\vec{P_1P_i}$ et $\vec{n}$. Si le produit scalaire est nul alors P_i est dans le plan, sinon le polygone n'est pas plan.

Exercice 7 *On veut réaliser une fonction* C *permettant d'extruder un profil contenu dans le plan* (xOy) *selon un vecteur* $\vec{v}$*. Le profil se présente sous forme d'une liste de coordonnées suivant la structure* `profil_element` *comme décrit dans le source 1.*

Source 1 – Structure `profil_element`

```c
typedef struct profil_element_s
{
  float x;
  float y;
} profil_element;
```

On considère qu'une fonction `draw_polygon` *existe et prend en argument une liste de points.*

(a) Profil **(b)** Profil extrudé

Figure 2 – Extrusion selon un profil

Correction de l'exercice 7 –

```c
typedef struct profil_element_s
{
  float x;
  float y;
} profil_element;
typedef struct point_s
{
  float x;
  float y;
  float z;
} point;
typedef struct vector_s
{
  float x;
  float y;
  float z;
} vector;
void extrude(unsigned int n, profil_element p, vector v)
{
  unsigned int i = 0;
  point profil_front[n];
  point profil_back[n];
  point quad[4];
  for(i=0; i<n; i++)
  {
    profil_front[i].x = p[i].x;
```

```
    profil_back[i].x = p[i].x;
    profil_front[i].y = p[i].y;
    profil_back[i].y = p[i].y;
    profil_front[i].z = 0;
    profil_back[i].z = 0;
  }
  for(i=0; i<n; i++)
  {
    profil[i].x += v.x;
    profil[i].y += v.y;
    profil[i].z += v.z;
  }
  draw_polygon(profil);
  draw_polygon(profil_back);
  for(i=0; i<n; i++)
  {
    quad[0].x = profil_front[i].x;
    quad[0].y = profil_front[i].y;
    quad[0].z = profil_front[i].z;
    quad[1].x = profil_back[i].x;
    quad[1].y = profil_back[i].y;
    quad[1].z = profil_back[i].z;
    quad[2].x = profil_back[(i+1)%n].x;
    quad[2].y = profil_back[(i+1)%n].y;
    quad[2].z = profil_back[(i+1)%n].z;
    quad[3].x = profil_front[(i+1)%n].x;
    quad[3].y = profil_front[(i+1)%n].y;
    quad[3].z = profil_front[(i+1)%n].z;
    draw_polygon(quad);
  }
}
```