Université Gustave Eiffel — INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Master Informatique 1

Correction TD Infographie n°1

RayTracer 1.0 - utilitaires

Quelques algorithmes de base du RayTracer à tester à part

Préambule : Produit scalaire ($\vec{u}\bullet\vec{v}$) et produit vectoriel ($\vec{u}\wedge\vec{v}$)

On travaille en dimension 3, dans le repère canonique de $\mathbb{R}^3$ $\mathcal{R}(O,x,y,z)$.
Les produits scalaire et vectoriel sont les 2 opérateurs fondamentaux sur les vecteurs.

- **Produit scalaire** $\vec{u}\bullet\vec{v} = (x_u.x_v) + (y_u.y_v) + (z_u.z_v) \quad \in \mathbb{R}$
  Propriétés :
  $\vec{u}\bullet\vec{v} = \vec{v}\bullet\vec{u}$ (symétrie)
  $\vec{u}\bullet\vec{u} = {x_u}^2 + {y_u}^2 + {z_u}^2 = \|\vec{u}\|^2$
  $\vec{u}\bullet(\vec{v}+\vec{w}) = \vec{u}\bullet\vec{v} + \vec{u}\bullet\vec{w}$ (distributivité)
  $\vec{u}\bullet\vec{v} = 0 \Leftrightarrow$ les vecteurs sont orthogonaux
  $\vec{u}\bullet\vec{v} = \|\vec{u}\|.\|\vec{v}\|.\cos(\widehat{\vec{u},\vec{v}})$

- **Produit vectoriel** $\vec{u}\wedge\vec{v} = \begin{vmatrix} x_u \\ y_u \\ z_u \end{vmatrix} \wedge \begin{vmatrix} x_v \\ y_v \\ z_v \end{vmatrix} = \begin{vmatrix} y_u.z_v - z_u.y_v \\ -x_u.z_v + z_u.x_v \\ x_u.y_v - y_u.x_v \end{vmatrix} \quad \in \mathbb{R}^3$

  Propriétés (3D) :
  $\vec{u}\wedge\vec{v} = -\vec{v}\wedge\vec{u}$ (attention : anti-symétrie)
  $\vec{u}\wedge(\vec{v}+\vec{w}) = \vec{u}\wedge\vec{v} + \vec{u}\wedge\vec{w}$ (distributivité)
  $\vec{u}\wedge\vec{v} = \vec{0} \Leftrightarrow$ les vecteurs sont colinéaires
  $\|\vec{u}\wedge\vec{v}\| = \|\vec{u}\|.\|\vec{v}\|.\sin(\widehat{\vec{u},\vec{v}})$

  Propriétés (2D) : le produit vectoriel en dimension 2 n'est pas défini, mais...

  $\vec{u}\wedge\vec{v} = \begin{vmatrix} x_u \\ y_u \\ 0 \end{vmatrix} \wedge \begin{vmatrix} x_v \\ y_v \\ 0 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ x_u.y_v - y_u.x_v \end{vmatrix} \quad \in \mathbb{R}^3$

  On peut donc considérer que, en dimension 2 : $\vec{u}\wedge\vec{v} = \begin{vmatrix} x_u \\ y_u \end{vmatrix} \wedge \begin{vmatrix} x_v \\ y_v \end{vmatrix} = x_u.y_v - y_u.x_v \in \mathbb{R}$

- Vecteur "normés" (i.e. de norme 1)
  en 2D ($\vec{u}\begin{vmatrix} x_u \\ y_u \end{vmatrix}$ et $\vec{v}\begin{vmatrix} x_v \\ y_v \end{vmatrix}$) : $\vec{u}\bullet\vec{v} = \cos(\widehat{\vec{u},\vec{v}})$ et $\vec{u}\wedge\vec{v} = \sin(\widehat{\vec{u},\vec{v}})$

Ces 2 opérateurs, très économiques, donnent donc toutes les informations nécessaires : longueur, orientation, direction, angles ....
☞ quel que soit le langage de prog., les fonctions `cos` et `sin` sont TRES coûteuses (basées sur des opérations polynômiales). Il faut limiter leur utilisation.
☞ à l'inverse les opérateurs $\bullet$ et $\wedge$ sont très économique : 2 multiplications et 1 addition en 2D.
☞ Lorsqu'on a besoin d'un angle (valeur, orientation...) : passer par ces 2 opérateurs

▸ Exercice 1. (Projection d'un point sur un segment de $\mathbb{R}^2$)

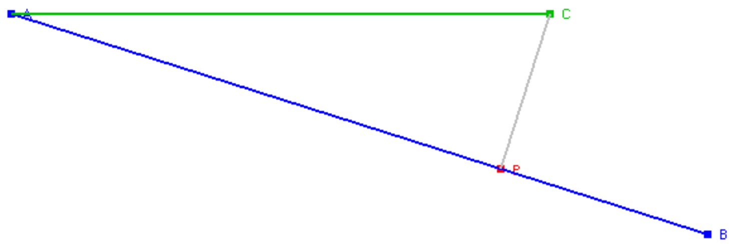
On considère trois points de $\mathbb{R}^2$ $\{A,\ B,\ C\}$

1. En utilisant uniquement le Produit Scalaire 2D, déterminer la position de la projection orthogonale P de C sur (AB)

2. Comment déterminer, en plus si P est entre A et B, avant A ou après B

3. Que devient ce calcul si la droite est définie par le point A et un vecteur $\overrightarrow{u}$ normé ($\overrightarrow{u} = \overrightarrow{AB}/\|\overrightarrow{AB}\|$) ?

4. Ecrire une fonction `Projection` qui renvoie la projection d'un point C sur une droite AB
   Intégrer cette fonction dans un programme graphique complet, permettant d'illustrer cet algorithme.



1. $P(x,y) \in (A,B) \Leftrightarrow \exists! t \in \mathbb{R}$ *tel que* $\overrightarrow{AP} = t \times \overrightarrow{AB}$
2. $\overrightarrow{AP} = t.\overrightarrow{AB} = \overrightarrow{AC} + \overrightarrow{CP} \quad \Rightarrow \quad t.\overrightarrow{AB}{\bullet}\overrightarrow{AB} = \overrightarrow{AB}{\bullet}\overrightarrow{AC} + \overrightarrow{AB}{\bullet}\overrightarrow{CP}$

   mais (CP) orthog. à (AB) par construction $\Leftrightarrow \overrightarrow{AB}{\bullet}\overrightarrow{CP} = 0$ donc $t = \dfrac{\overrightarrow{AB}{\bullet}\overrightarrow{AC}}{\overrightarrow{AB}{\bullet}\overrightarrow{AB}}$

3. il suffit de regarder la valeur de t : $t < 0$ ou $t \in [0,1]$ ou $t > 1$

4. $t = \overrightarrow{u}{\bullet}\overrightarrow{AC}$ soit $P = A + (\overrightarrow{u}{\bullet}\overrightarrow{AC}).\overrightarrow{u}$

```
G2Xpoint Projection(G2Xpoint A, G2Xpoint B, G2Xpoint C)
{
  G2Xvector AB = g2x_SetVect(A,B);
  G2Xvector AC = g2x_SetVect(A,C);
  double    t  = g2x_ProdScal(AB,AC)/g2x_ProdScal(AB,AB);
  return (G2Xpoint){A.x+t*AB.x,A.y+t*AB.y};
}
```

▸ Exercice 2. (Intersection de deux segments de $\mathbb{R}^2$)

On considère quatre points de $\mathbb{R}^2$ $\{A,\ B,\ C,\ D\}$.

1. Donner l'équation de la droite support du segment $[A,B]$.

2. On veut déterminer la valeur de t telle que $P(t)$ soit le point d'intersection de $[A,B]$ et $[C,D]$.

   - Quelle condition doit vérifier t pour que l'intersection entre les droites (A,B) et (C,D) existe et soit située sur le segment $[C,D]$?
   - En utilisant uniquement le Produit Vectoriel 2D, déterminer le point d'intersection des deux segments, s'il existe.

3. Ecrire une fonction `InterSegment` qui calcule le point d'intersection entre deux segments (4 points) passés en arguments. Cette fonction renverra par ailleurs un message booléen selon que l'intersection existe ou non.
   Intégrer cette fonction dans un programme graphique complet, permettant d'illustrer cet algorithme.

1. Equation paramétrique de (A, B) :

$$P(x,y) \in (A,B) \Leftrightarrow \exists! t \in \mathbb{R} \text{ tel que } \overrightarrow{AP} = t \times \overrightarrow{AB} \text{ soit } \begin{cases} x = x_1 + t(x_2 - x_1) \\ y = y_1 + t(y_2 - y_1) \end{cases}$$

2. On note $P(x, y)$ le point d'intersection éventuel des droites (A, B) et (C, D).

   - P *existe* $\iff \exists (t_a, t_c) \in \mathbb{R}^2$ *tels que* $\overrightarrow{AP} = t_a.\overrightarrow{AB}$ *et* $\overrightarrow{CP} = t_c.\overrightarrow{CD}$

$$\begin{cases} P \in [A,B] \Leftrightarrow t_a \in [0,1] \\ P \in [C,D] \Leftrightarrow t_c \in [0,1] \end{cases}$$

   - Si P existe, il faut alors calculer les paramètres t_a et t_c.

     Il faut commencer par éliminer le cas où les droites sont parallèles : $\overrightarrow{AB} \wedge \overrightarrow{CD} = 0$. (Ca va resservir !)

     $\overrightarrow{AC} = \overrightarrow{AP} - \overrightarrow{CP} = \overrightarrow{AP} - t_c.\overrightarrow{CD} \Rightarrow \overrightarrow{AB} \wedge \overrightarrow{AC} = \overrightarrow{AB} \wedge \overrightarrow{AP} - t_c.\overrightarrow{AB} \wedge \overrightarrow{CD}$
     mais $P \in (AB) \Leftrightarrow \overrightarrow{AB} \wedge \overrightarrow{AP} = 0$ donc $t_c = -\dfrac{\overrightarrow{AB} \wedge \overrightarrow{AC}}{\overrightarrow{AB} \wedge \overrightarrow{CD}} = \dfrac{\overrightarrow{AC} \wedge \overrightarrow{AB}}{\overrightarrow{AB} \wedge \overrightarrow{CD}}$

     Si t_c n'est pas dans l'intervalle $[0., 1.]$, inutile de continuer.
     Sinon, on fait de même avec l'autre
     $\overrightarrow{AC} = \overrightarrow{AP} - \overrightarrow{CP} = t_a.\overrightarrow{AB} - \overrightarrow{CP} \Rightarrow \overrightarrow{AC} \wedge \overrightarrow{CD} = t_a.\overrightarrow{AB} \wedge \overrightarrow{CD} - \overrightarrow{CP} \wedge \overrightarrow{CD}$
     mais $P \in (CD) \Leftrightarrow \overrightarrow{CP} \wedge \overrightarrow{CD} = 0$ donc $t_a = \dfrac{\overrightarrow{AC} \wedge \overrightarrow{CD}}{\overrightarrow{AB} \wedge \overrightarrow{CD}}$

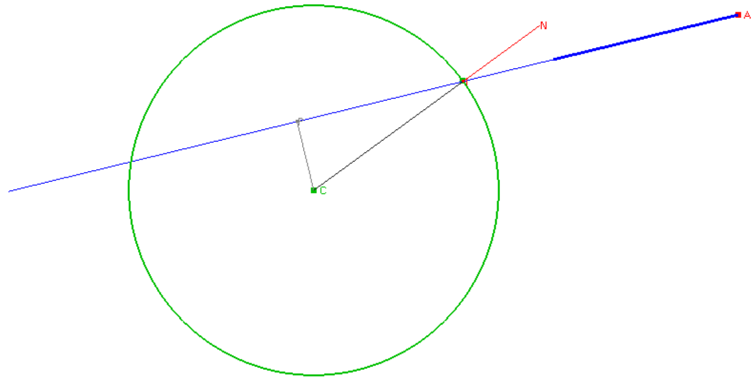
```
bool InterSegments(G2Xpoint A, G2Xpoint B, G2Xpoint C, G2Xpoint D, G2Xpoint *I)
{
  G2Xvector AB = g2x_SetVect(A,B);
  G2Xvector CD = g2x_SetVect(C,D);
  double    v  = g2x_ProdVect(AB,CD);
                                    /* si les 2 droites sont parallèles, on arrête       */
  if (G2Xiszero(v)) return false;   /* if (v==.0) TEST d'EGALITE A PROSCRIRE             */

  G2Xvector AC=g2x_SetVect(A,C);
  double t1 = g2x_ProdVect(AC,CD)/v;   /* t1 : position de I sur la droite (AB)          */
  if (t1<0. || t1 >1.) return false;   /* si I n'appartient pas à [A,B] inutile de continuer */

  double t2 = g2x_ProdVect(AC,AB)/v;   /* t2 : position de I sur la droite (CD)          */
  if (t2<0. || t2 >1.) return false;   /* si I n'appartient pas à [C,D] inutile de continuer */

  I->x = A.x+t1*AB.x;               /* si on est arrivé là, on a le point I              */
  I->y = A.y+t1*AB.y;
  return true;
}
```

▸ Exercice 3. (Intersection Rayon/Cercle dans $\mathbb{R}^2$)

On considère un cercle de centre C, de rayon r et un rayon, issu d'un point A et de direction $\vec{u}$ (normé).

1. En utilisant la fonction de Projection précédente, déterminer le premier (i.e. le plus proche de la source) point d'intersection éventuel du rayon avec le cercle.
   On s'efforcera d'éliminer le plus rapidement possible les cas sans intersection valable.

2. Ecrire une fonction `RayInterCercle` qui renvoie cette intersection, si elle existe, ainsi que la normale au cercle en ce point.
   Intégrer cette fonction dans un programme graphique complet, permettant d'illustrer cet algorithme.



3. Cet algorithme peut-il fonctionner avec une ellipse quelconque ?

On commence par éliminer les cas sans solution.

- Mauvaise direction (le rayon ne se dirige pas vers le cercle) :

  on utilise le produit scalaire $t = \overrightarrow{\mathsf{u}}\bullet\overrightarrow{\mathsf{AC}}$: si $t <= 0.$, on arrête (angle $\widehat{\overrightarrow{\mathsf{u}},\overrightarrow{\mathsf{AC}}} > 180^o$)
- Le rayon passe à côté du cerle :
  Avec t, on a la position de la proj. orthog. de C sur le rayon. $P = A + t.\overrightarrow{\mathsf{u}}$
  Si P est à l'extérieur du Cercle, on arrête.
  Il suffit de calculer $||\overrightarrow{\mathsf{CP}}||^2 = \overrightarrow{\mathsf{CP}}\bullet\overrightarrow{\mathsf{CP}}$ et de la comparer à r^2 (on calcule $d = r^2 - \overrightarrow{\mathsf{CP}}\bullet\overrightarrow{\mathsf{CP}}$).

- si P est à l'intérieur du Cercle, c'est le milieu des 2 points d'intersection I et J et les triangles (CPI) et (CPJ) sont rectangles en P. Le théorème de Pythagore donne : $||\overrightarrow{\mathsf{PI}}||^2 = ||\overrightarrow{\mathsf{PJ}}||^2 = r^2 - ||\overrightarrow{\mathsf{CP}}||^2 = d$
  Les 2 points d'intersection sont alors $I = A + (t - \sqrt{d}).\overrightarrow{\mathsf{u}}$ et $J = A + (t + \sqrt{d}).\overrightarrow{\mathsf{u}}$.
  C'est I le plus proche de la source A.

  L'avantage de cette méthode est qu'on élimine les cas inutiles très vite, les calculs réalisés à chaque test resservent si on doit continuer, et le calcul le plus coûteux (la racine carrée) n'est effectuée que si nécessaire.

  Et on aurait pu utiliser le fait que $d = |\overrightarrow{\mathsf{u}}\wedge\overrightarrow{\mathsf{AC}}|$ (coût de calcul sensiblement équivalent).

  Remarque : à aucune moment on n'utilise les équations du cercle... tout juste quelques propriétés simples.
- Malheureusement, ça ne fonctionne pas avec une ellipse. Mais heureusement, il y a les transformations.

```
/* Intersection d'un rayon avec un cercle quelconque */
bool rayon_inter_cercle(G2Xpoint* J, G2Xvector* N, G2Xpoint A, G2Xvector u, G2Xpoint C, double r)
{
  G2Xvector AC = g2x_SetVect(A,C);
  double     t = g2x_ProdScal(u,AC);

  if (t<0.) return false;     /* le rayon ne se dirige pas vers le cercle      */
                              /* t donne la position de la proj. de C sur [Au) */
  double d2 = g2x_ProdVect(u,AC);
         d2 = SQR(d2);        /* d donne la distance de C à sa proj. P         */
  if (d2>r*r) return false;   /* le rayon passe à côté du cercle               */

  t = t-sqrt(r*r-d2);

  J->x = A.x + t*u.x;         /* si on est arrivé là, on a le point J          */
  J->y = A.y + t*u.y;
  *N = g2x_SetVect(C,*J);     /* et la normale en J                            */
  g2x_Normalize(N);

  return true;
}
```

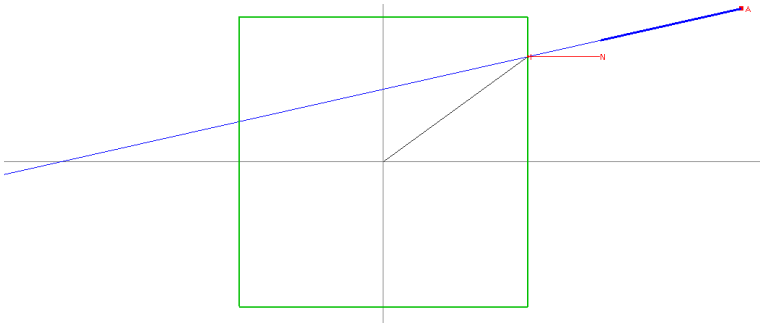
```
/* Intersection d'un rayon avec LE cercle canonique */
bool rayon_inter_cercle_canonique(G2Xpoint* J, G2Xvector* N, G2Xpoint A, G2Xvector u)
{
  /* les coord. de A sont celle du vecteur OA = -AO => inversion de signe      */
  double t = -g2x_ProdScal(u,(G2Xvector)A);
  if (t<0.) return false;     /* le rayon ne se dirige pas vers le cercle      */
                              /* t donne la position de la proj. de C sur [Au) */
  double d2 = g2x_ProdVect(u,(G2Xvector)A);
         d2 = SQR(d2);        /* d donne la distance de C à sa proj. P         */
  if (d2>1) return false;     /* le rayon passe à côté du cercle               */

  t = t-sqrt(1.-d2);
  J->x = A.x + t*u.x;         /* si on est arrivé là, on a le point J          */
  J->y = A.y + t*u.y;
  *N = (G2Xvector)(*J);       /* et la normale en J (déjà normée)              */

  return true;
}
```

▸ Exercice 4. (Intersection Rayon/Carré dans $\mathbb{R}^2$)

On considère le carré **canonique** centré sur l'origine Ω, de demi-côté 1 et un rayon $[A, \vec{u})$ (normé).

1. En utilisant la fonction d'intersection de segments précédente (ou une variante simplifiée adaptée au cas du carré canonique), déterminer le premier (i.e. le plus proche de la source) point d'intersection éventuel du rayon avec le carré.
   On s'efforcera d'éliminer le plus rapidement possible les cas sans intersection valable.

2. Ecrire une fonction `RayInterCarre` qui renvoie cette intersection, si elle existe, ainsi que la normale au carré en ce point.
   Intégrer cette fonction dans un programme graphique complet, permettant d'illustrer cet algorithme.



3. Cet algorithme peut-il fonctionner avec un rectangle quelconque ?

On commence par éliminer les cas sans solution.

- Mauvaise direction (le rayon ne se dirige pas vers le cercle) :
  on utilise le produit scalaire $t = \vec{u} \bullet \overrightarrow{AC}$: si $t <= 0.$, on arrête (angle $\widehat{\vec{u}, \overrightarrow{AC}} > 180^o$)

Mêm si le carré peut être défini par une équation presque aussi simple que le cercle (laquelle....?), il est surtout défini par 4 côtés qui sont des segments très simples, portés par les droites $(x = +1); (y = +1); (x = -1); (y = -1)$.

D'autre part, quelle que soit la position de l'origine du rayon on est sûr que l'on peut élminer 2 de ces côtés. Reste à déterminer, simplement, lesquels.

On définit 4 valeurs booléenes, une pour chaque côté et on teste où se trouve le point A par rapport à ces 4 droites :

si $x_A > -1.$ on élimine le côté ($x = -1.$)
sinon, si $x_A < +1.$ on élimine le côté ($x = +1.$)
si $y_A > -1.$ on élimine le côté ($y = -1.$)
sinon, si $y_A < +1.$ on élimine le côté ($y = +1.$)

On définit également les 4 jeux de coordonnées (à la fois points et vecteurs) : $(+1., 0.), (0., +1.), (-1., 0.), (0., -1.)$

Elles représentent à la fois les centres (points) des 4 côtés et les normales (vecteurs) au carré sur ces côtés.

Pour chacun des côtés restant (2 au maximum) :

- on détermine l'intersection du rayon et de la droite support (`InterSegment` simplifiée) en utilisant le centre et la normale du côté.
- on vérifie si le point $I(x_I, y_I)$ trouvé est bien sur le segment.

Cet algo ce généralise très simplement en 3D avec le cube canonique (6 faces).

```
/* Intersection d'un rayon avec LE carré canonique (même prototype que le pour le cercle) */
bool rayon_inter_carre_canonique(G2Xpoint* J, G2Xvector* N, G2Xpoint A, G2Xvector u)
{
  bool flag[4]={false,false,false,false}; /* flags de positionnement de la source du rayon par rapport aux 4 faces */
  static G2Xcoord  CN[4]={{+1.,0.},{0.,+1.},{-1.,0.},{0.,-1.}};   /* les 4 centres de face et normales exterieures */

  flag[0]=(bool)(A.x>+1.);  /* au max. 2 valeurs vraies sur les 4 (0 si A est dans le carré) */
  flag[1]=(bool)(A.y>+1.);
  flag[2]=(bool)(A.x<-1.);
  flag[3]=(bool)(A.y<-1.);

  int i;
  for (i=0;i<4;i++)
  { /* on ne traite que les faces "visibles" depuis [A,u) */
    if (flag[i]==false) continue;

    double ps1 = g2x_ProdScal(g2x_SetVect(A,CN[i]),CN[i]);  if (ps1>=0.) continue;
    double ps2 = g2x_ProdScal(u                   ,CN[i]);  if (ps2>=0.) continue;

    double t=ps1/ps2;
    J->x = A.x+t*u.x;
    J->y = A.y+t*u.y;
    if (MAX(fabs(J->x),fabs(J->y)>1.) continue; /* si J n'est pas dans le carré, ce n'est pas la bonne, on continue */
    *N=CN[i];                                   /* sinon, on récupère la normale courante et on s'en va             */
    return true;
  }
  return false;
}
```
