Graphisme

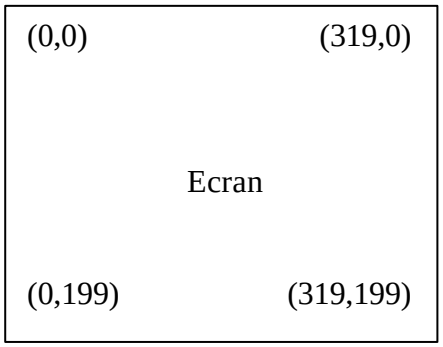
2D - 3D

- **L'écran est constitué de plusieurs pixels qui sont assimilables a des points, chaque pixel s'identifier par son numéro.**
- **Mode nh : définit le mode vidéo qu'on va utiliser, c-â-d le mode vidéo qui décompose l'écran en X*Y pixels en associant à chaque pixel un nombre fini d'octets.**

Notre 1ère mode sera le mode 13h (320 * 200) en associant 1octet à chaque pixel, c.a.d. 8 bits, ce qui donne 256 colore.
On ne peut repérer un pixel que par son numéro dans l'écran, sachant que dans le mode 13h on a 320*200=64000 pixels et que le nombre max en 16 bits est 65536 (11111111 en binaire), donc on ne trouvera pas de problèmes puisque 64000<56535.
Mais si on veut introduire les notions de géométrie mathématiques, on va assimiler l'écran a un repère 2D pour repérer chaque pixel, par leur coordonnées cartésiennes, pour cela il faut construire une relation entre ces coordonnées et le numéro de pixel, mathématiquement c'est :

$$N = y*320 + x$$

Maintenant on va identifier chaque pixel par leur coordonnées (x , y), on dit que l'écran est un espace 2D, dont l'origine est en haut et a gauche de l'écran, de coordonnées (0,0) , avec :



$x \in [0,319]$ *et* $y \in [0,199]$

Donc dessiner un point sur l'écran, c'est allumer un pixel avec une couleur bien définit, ça se fait avec un code qui dépend de langage de programmation utilisé.

Exemple **:**

- **Putpixel(x , y , col) ; // En C/C++**
- **Pset(x , y), col ' En Qbasic**

Le langage de programmation permet aussi de dessiner une droite, un carre, un cercle, tout ça dans le plan (oxy), le seul plan en 2D.

Comment donc dessiner un point repéré par leur coordonnées (x , y , z) en 3D ?

Je vais utiliser un peut de géométrie perspective.

- **Je vais choisir l'origine du repère 3D (x_{or} , y_{or}).**
- **l'axe $\overrightarrow{O_0Y_0}$ parallèle à $\overrightarrow{OX}$ et ont le même sens.**
- **l'axe $\overrightarrow{O_0Z_0}$ parallèle à $\overrightarrow{OY}$ et ont des sens opposés.**
- **l'axe $\overrightarrow{O_0X_0}$ fait un angle $t=\frac{p}{4}$ avec $\overrightarrow{O_0Y_0}$. On pose $\text{pi}=p=3.14$**

Voir figure 1 :

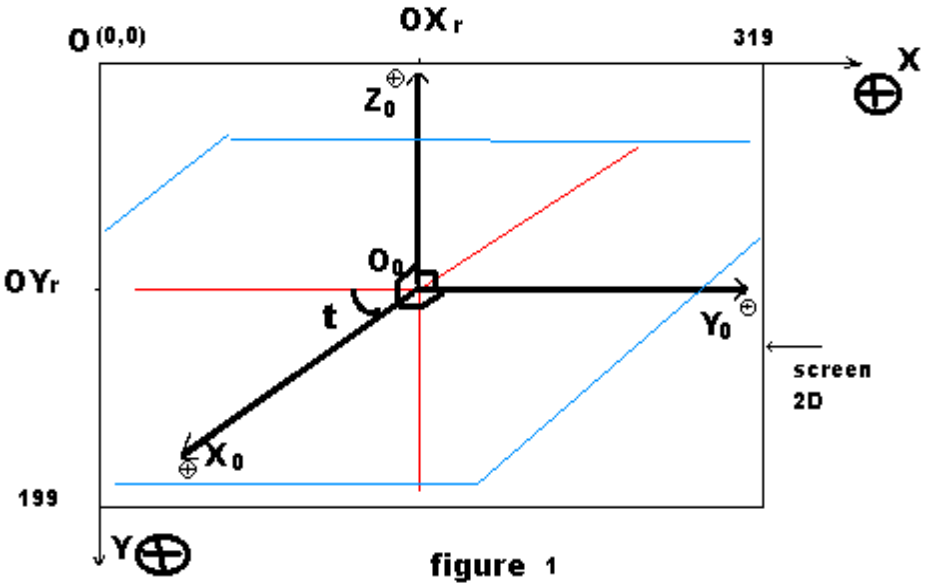


figure 1

Il reste le problème de transformer les cordonnées 3D en 2D, c.a.d si on veut allumer le pixel (x_0 , y_0 ,z_0) avec la couleur c_0, il faut chercher une application et après on utilise putpixel (x,y).

Pour la recherche de cette application on fait les schémas suivants :

$$f:\ \Re^3 \longrightarrow \Re^2$$
$$(Y_0, Y_0, Z_0) \longrightarrow (x, y)$$

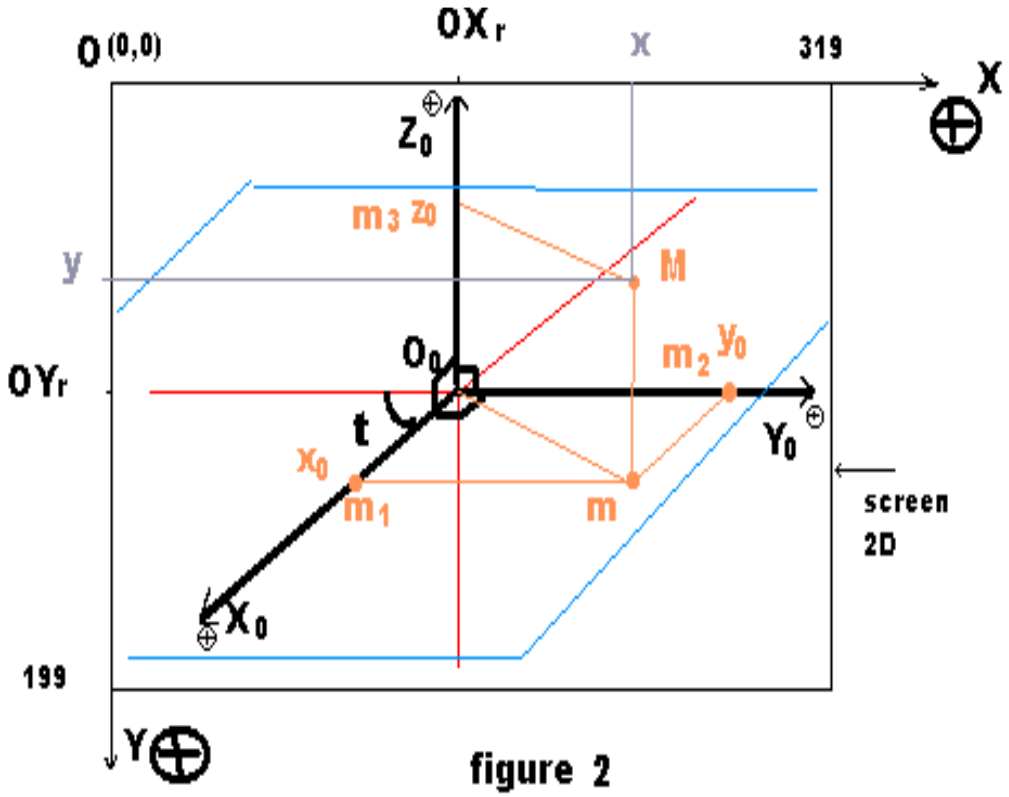


figure 2

On a $\overrightarrow{OM}\ (X,Y)$: les cordonnées réelles du point M dans l'écran 2D.

donc $\overrightarrow{OM} = \overrightarrow{OO_0} + \overrightarrow{O_0M}$

$\overrightarrow{OO_0}\ (X_{or}, Y_{or})$

$\overrightarrow{O_0M} = \overrightarrow{O_0m_1} + \overrightarrow{m_1m} + \overrightarrow{m\ M}$

$\overrightarrow{O_0m_1}$ **$(-X_0\cos(t)\ ,\ X_0\sin(t)) = (Xm_1, Ym_1)$**

$\overrightarrow{m_1m}$ **$(Xm_1 + Y_0, Ym_1)$**

$\overrightarrow{m\ M}$ **$(Xm_1\ ,\ Ym_1 - Z_0)$**

Donc $\overrightarrow{O\ M}$ **(X, Y) tel que :**

$X = X_{or} - X_0\cos(t) + Y_0$

$Y = Y_{or} + X_0\sin(t) - Z_0$

Puis on fabriquera une fonction (putpixel3D) :

- **en qb :**

```
declare sub putpixel3D( X0 as double, Y0 as double ,Z0 as double ,col)

    sub putpixel3D( X0 as double,Y0 as double ,Z0 as double ,col as integer)

        pset ( Xor -X0*cos(t) + Y0 , Yor + X0 *sin(t) - Z0 ), col

    end sub
```

- **en c/c++ :**

```
void putpixel3D(flot X0 , flot Y0 , flot Z0 ,int col)

    {

        putpixel( Xor -X0*cos(t) + Y0 , Yor + X0 *sin(t) -Z0 ,col) ;

    }
```

Pour une ligne on va faire une fonction line3D

```
void line3D ( float x1 , float y1, float z1, float x2, float y2, float z2,int col)

    {

        float xx1,xx2,yy1,yy2 ;

        xx1= xor -x1*cos(t) + y1 ; yy1=yor + x1 *sin(t) -z1 ;
        xx2= xor -x2*cos(t) + y2 ; yy2=yor + x2 *sin(t) -z2 ;
        setcolor( col ) ;
        line(xx1,yy1,xx2,yy2);

    }
```

en qb :

```
sub line3D ( double x1 , double y1, double z1, double x2, double y2, double
z2, integer col)

    xx1= xor -x1*cos(t) + y1 ; yy1=yor + x1 *sin(t) -z1 ;
    xx2= xor -x2*cos(t) + y2 ; yy2=yor + x2 *sin(t) -z2 ;
    line(xx1,yy1)-(xx2,yy2),col

end sub
```

Sachant que un cercle est un ensemble de points satisfont a l'équation suivant :

$$(x - x_0)^2+(y-y_0)^2=r^2$$

ou

$x= x_0+r \cos(fi)$

$y= y_0+r \sin(fi)$

Dans un plan (oxy), et puisque l'écran est un plan 2D alors la fonction circle est équivalant a :

$$x = x_0 + r \cos(fi)$$
$$y = y_0 + r \sin(fi)$$
putpixel(x,y,col) ;

Or en 3D on a 3 cas particuliers des plans :

- **plan oxy :**

  $x = x_0 + r \cos(fi)$
  $y = y_0 + r \sin(fi)$ **, équation 1**

- **plan oyz**

  $z = x_0 + r \cos(fi)$
  $y = y_0 + r \sin(fi)$ **, équation 2**

- **plan oxz**

  $x = x_0 + r \cos(fi)$
  $z = y_0 + r \sin(fi)$ **, équation 3**

Si on utilise notre fonction putpixel3D pour dessiner un cercle de coordonnées x_0,y_0,z_0, ce cercle sera dans un plan parallèle au plan oxy

Voir figure 3 :

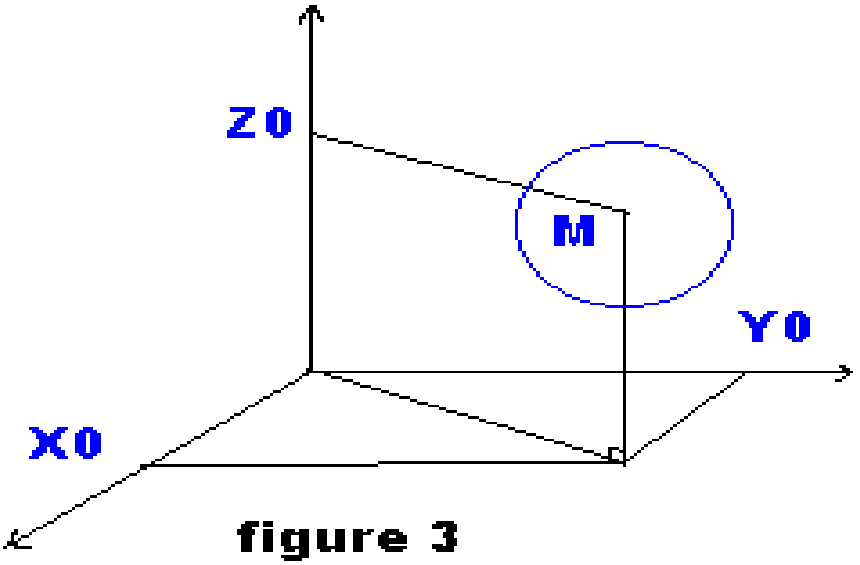


figure 3

Car putpixel3D transforme seulement les coordonnées cartésiennes 3D du centre en coordonnées cartésiennes 2D. Donc pour dessiner un cercle dans chacun des plans précédents, on va construire 3 fonctions circle3D :

- **circle3Doxy(float x0, float y0 ,float z 0, int r ,int col)**

```
{float fi ; for (fi=0 ;fi<=6.28 ;fi+=0.05 )
{
x=x0+r*cos(fi) ;
y=y0+r*sin(fi) ;
```

```
                z=z0 ;
                putpixel3D(x,y,z,c) ;

            }

        }
```

- **circle3Doxz(float x0, float y0 ,float z 0, int r ,int col)**

```
        {float fi ; for (fi=0 ;fi<=6.28 ;fi+=0.05 )

            {

                x=x0+r*cos(fi) ;
                z=z0+r*sin(fi) ;
                y=y0 ;
                putpixel3D(x,y,z,c) ;

            }

        }
```

- **circle3Doyz(float x0, float y0 ,float z 0, int r ,int col)**

```
        {float fi ; for (fi=0 ;fi<=6.28 ;fi+=0.05 )

            {

                z=z0+r*cos(fi) ;
                y=y0+r*sin(fi) ;
                x=x0 ;
                putpixel3D(x,y,z,c) ;

            }

        }
```

Voire figure :

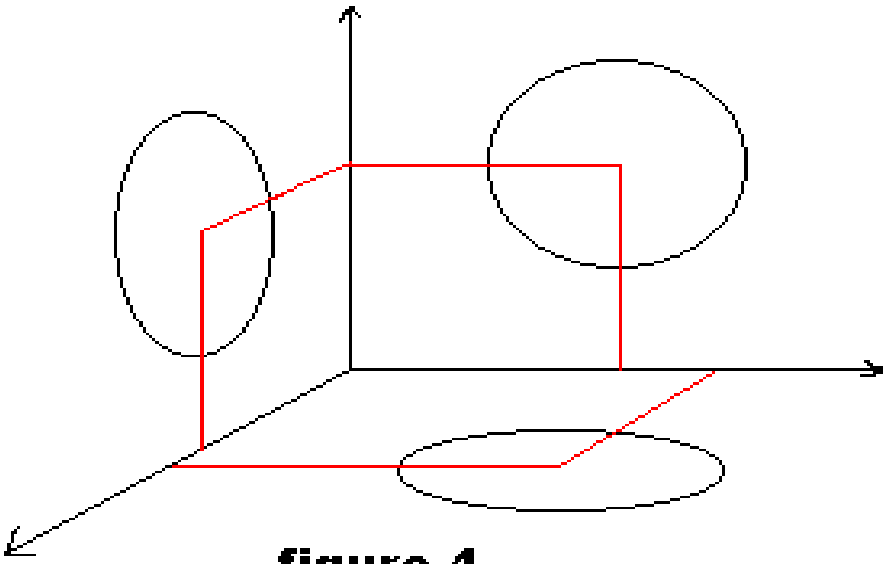


figure 4

Mais il reste un problème, c'est de dessiner un cercle dans un plan quelconque définit par sa normale (n) $\vec{n}$ de module $\left\| \vec{n} \right\|$ et des angles tta (q), ffi (j). Pour cela on va considérer un autre repère R_1 (O_1,x_1,y_1,z_1) tel que $\overrightarrow{O_1z_1}$ qui ne peut être que n. puis on va faire un changement de repère.

Voire figure :

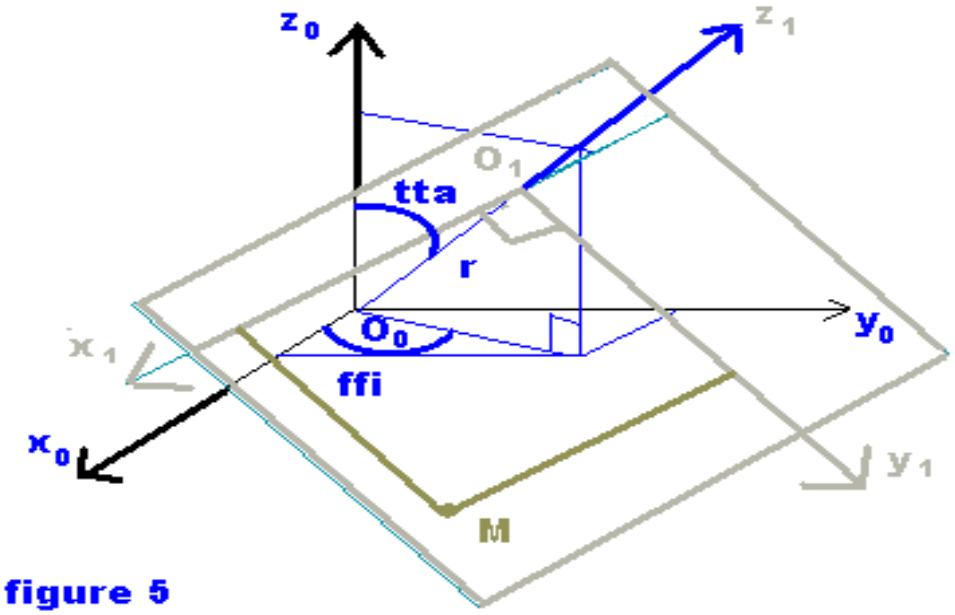


figure 5

A suivre...
