Fiches de techniques graphiques

Thierry EXCOFFIER

8 octobre 1998

Les techniques graphiques appelées aussi informatique graphique ou en raccourci : infographie. Ce cours comprend beaucoup d'algorithmie.

Bibliographie :

"Computer graphics : Principes and Practice", J.-D. FOLEY A. VAN DAM, The systems programming series, ADDISON WESLEY, (plus de 1000 pages : la bible)

"La synthèse d'images", B. PEROCHE D.GHAZANFARPOUR J. ARGENCE D. MICHELUCCI, Traité des Nouvelles technologies, HERMES, (300 pages 300 francs)

Disponibles à la B.U.

Objectif des cours et TP : savoir créer une image et un programme interactif.

Table des matières

1 Historique des techniques graphiques

L'utilisation de l'image est foncdamentale comme moyen de communication homme/machine. Ceci pour plusieurs raisons :

- L'image contient beaucoup d'informations.
  (Une image remplace un long discours).
- La vue est le sens de l'homme qui permet le plus grand débit d'informations.
- L'informatique permet de représenter sous forme de graphismes de nombreuses données.

On a donc donné des dispositifs graphiques aux ordinateurs.
Les types d'image :

- Représentation de données 2D et 3D (ou plus).
- Tracé de fonctions, graphiques de présentation.
- Affichage d'image traitées.
- Création d'images artistiques.
- Création de plans.
- Interface utilisateur. (bureautique, jeux, EAO)

Historique :

1950 MIT, Ecran vectoriel (traits) + light pen (photostyle)

alentours 1960 Traceurs de courbes.

1963 I.SUTHERLAND, système de dessin SKETCHPAD.
DAO chez GM, Lockheed Aircrafts et Bell Telephone.

196? Tektronix 4012 + tablette de digitalisation

1970 Baisse prix mémoire → Réalisation d'écrans matriciels (Raster Display, CRT) + souris

1977 Micro-ordinateur + cartes graphiques
Bibliothèques graphiques CORE, GKS.

1982 Station de travail.

1984 Normalisation GKS.

1985 Processeurs graphiques spécialisés.

1988 X Consortium (Début X en 84).
PHIGS

198? PostScript, imprimantes Laser.

1990 PHIGS+.

1991 Les réalités virtuelles.

1992 PEX 5.0.

2 Technologie graphique

La technologie graphique est en fait l'ensemble des moyens permettant d'afficher ou d'imprimer des images. Mais aussi les dispositifs d'interaction.

Pour les dispositif d'affichages, ceux-ci peuvent être :

- Vectoriel ou matriciel.
- Plus ou moins rapide.
- A rafraichissement ou à mémoire.
- Monochrome ou couleur.
- 2D ou 3D
- Directionnel ou non.

Les plus connus sont l'écran matriciel à balayage de trame (écran TV) à cristaux liquides et plasma.

Les dispositifs de sortie "matériel", peuvent :

- Sortir sur papier, papier spéciaux, films, résine.
- Vectoriel ou matriciel.
- Plus ou moins rapide.
- Monochrome ou couleur.
- 2D ou 3D
- Supportent ou non le passage des années.

Les plus connus sont l'imprimante matriciel à impact, à jet d'encre ou laser. Ainsi que les traceurs de courbes à plume.

Les dispositifs d'entrée :

- Entrée d'un ou plusieur choix d'un ou plusieur nombres flottants et d'images.
- Entrée absolue ou relative.
- Entrée avec le doigt, la main, la voix, l'oeil, le corps.
- Dispositif d'entrée fixe, mobile, déplaçable.
- Avec ou sans contact physique.

Le clavier et la souris sont universellement connus on trouve aussi les boîtes à boutons, les potentiomètres, les tablettes à digitaliser, les scanners, écrans tactiles, data glove, ...

Nous allons étudier les dispositifs les plus employés mais aussi certains qui sont au stade expérimental.

2.1 Ecrans vectoriels

Un écran vectoriel n'est capable d'afficher que des segments de droites. C'est la première technologie a avoir été utilisée car elle était facilement réalisable physiquement et ne demandait pas du tout ou peu de mémoire. On les appel aussi écran à balayage cavalier.

2.1.1 Ecrans vectoriels avec tube à mémoire

Cette technologie a tout d'abord fonctionné sur des écrans à mémoire. L'avantage de cette approche était le fait que l'image était stockée par la surface d'affichage elle même et non par une mémoire informatique.

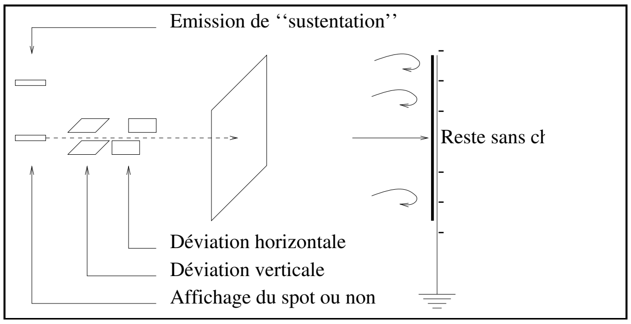


FIG. 1 – *Tube à mémoire*

Pour tracer un segment de droite, il faut faire une interpolation linéaire entre 2*deux tensions électriques. Ceci est facilement fait analogiquement.

Le problème de cet écran est l'effacement, en effet tous ce qui est affiché le reste. Pour effacer, il faut tout effacer (un grand flash sur l'écran) Il est donc difficile de faire des programmes interactifs. La qualité d'affichage pour des segments de droites est parfaite, il n'y a pas de problème de marches d'escalier. La vitesse est très grande car le processus d'affichage est analogique.

La méthode n'est pas applicable à la couleur.

2.1.2 Ecrans vectoriels à rafraichissement

Sur ces écrans la surface d'affichage ne mémorise rien, il faut donc retracer constament l'image. L'écran doit donc mémoriser les segments à retracer.

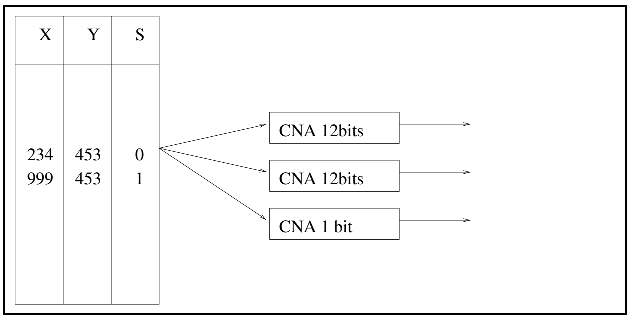


FIG. 2 – *Balayage cavalier rafraichie*

Ce type d'écran était souvent utilisé pour faire de l'animation en filaire. Il est très rapide et nécessite peu de mémoire, il peut être en couleur (3 canons à electrons). L'inconvénient de cet écran est la limite de complexité de l'image, si il y a trop de vecteurs, l'image "flicke" car l'écran n'a pas le temps de tout afficher en 1/25 de seconde.

L'affichage vectoriel est utiliser de nos jours pour afficher sur de grands support en utilisant un laser. La déflection du laser étant mécanique (miroir), leur vitesse d'affichage est beaucoup plus faible.

2.2 Différents écrans matriciels

Les écrans matriciels non à balayage de trames sont composés d'une surface composées de conducteur verticaux parallèles et d'une autre d'horizontaux. A chaque croisement de conducteur correspond un pixel.

De nombreux types existent :

- Cristaux liquides, la tension electrique oriente les cristaux.
  Inconvénients :
  - Le temps d'orientation est long.
  - Il est nécessaire de rafraichir cet écran.

  Avantages :
  - Très faible consommation.
  - On peut faire de la couleur en superposant des matrices colorées (génération soustractive de couleurs)
- Matrice active. A chaque pixel correspond un transistor qui maintient l'état du pixel.
  Inconvénients :
  - Consommation plus importante.
  - Couteux.

  Avantages :
  - Grande rapidité,
  - Niveaux de gris ou couleurs précis.
  - Rafraichissement peut être évité.
- Ecran plasma. (analogue aux néon) Inconvénients :
  - Consommation plus importante.
  - De couleur orange.
  - Faible résolution.

  Avantages :
  - Grande rapidité,
  - Rafraichissement est évité.
- Electroluminescent. Emet de la lumière si grande tension.
- Electrophorique. Particules colorées en suspension, ...

Pour la projection sur grands écrans des écrans basés sur des cristaux liquides sont de plus en plus utilisés.

La vitesse d'affichage est importante dans le cas d'un curseur se déplaçant sur l'écran (celui de la souris notamment) car si l'affichage est trop lent, il disparaît complètement.

2.3 Les écrans 3D

Les écrans 3D sont de différents types :

- L'affichage stéréoscopique (fixe ou lié à l'observateur).
- L'affichage 3D réel.
- L'écran holographique (science-fiction pour le moment)

2.3.1 Ecrans stéréoscopiques

Montrer une image différente à l'oeil gauche et au droit. Différentes technologies sont possibles :

- L'écran rouge/vert, la fameuse technique du cinéma 3D. L'observateur à des lunettes rouge/vert.
  Avantages :
  - Pas cher.
  - Accessible à plusieurs personnes en même temps.

  Inconvénients :
  - Les couleurs sont fausses.
- L'ecran polarisé. Il y a un filtre polarisant devant l'écran qui polarise dans un sens les images paires (gauche par exemple) et dans l'autre les impaires. L'observateur à des lunettes polarisées.
  Avantages :
  - Accessible à plusieurs personnes en même temps.
  - Couleur possible.

  Inconvénients :
  - On doit avoir un filtre devant l'écran.
  - Si le spectateur penche la tête...
- Lunettes intelligentes. Les verres contiennent des cristaux liquides qui sont commandé par l'écran. Ils deviennent alternativement opaques.
  Avantages :
  - Couleur possible.
  - Si le spectateur penche la tête il n'y a pas de problème car pas de lumière polarisée.

  Inconvénients :
  - Accessible seulement à peu de personnes.
- Un écran devant chaque oeil. Les écrans doivent être petit, léger. Les inconvénients sont la faible maniabilité et le prix. De plus cela est limité à une seule personne. Cette méthode est utilisée pour les réalités virtuelles car l'utilisateur doit pouvoir se déplacer.

2.3.2 Ecrans 3D réels

Le problème principal est la quantité d'information à traiter, elle est énorme, le seul écran 3D qui a été commercialisé fonctionne dans les aéroports. Il permet d'afficher un nuage de points dans l'espace. Plusieurs personnes peuvent le regarder en même temps et l'on peut tourner autour.

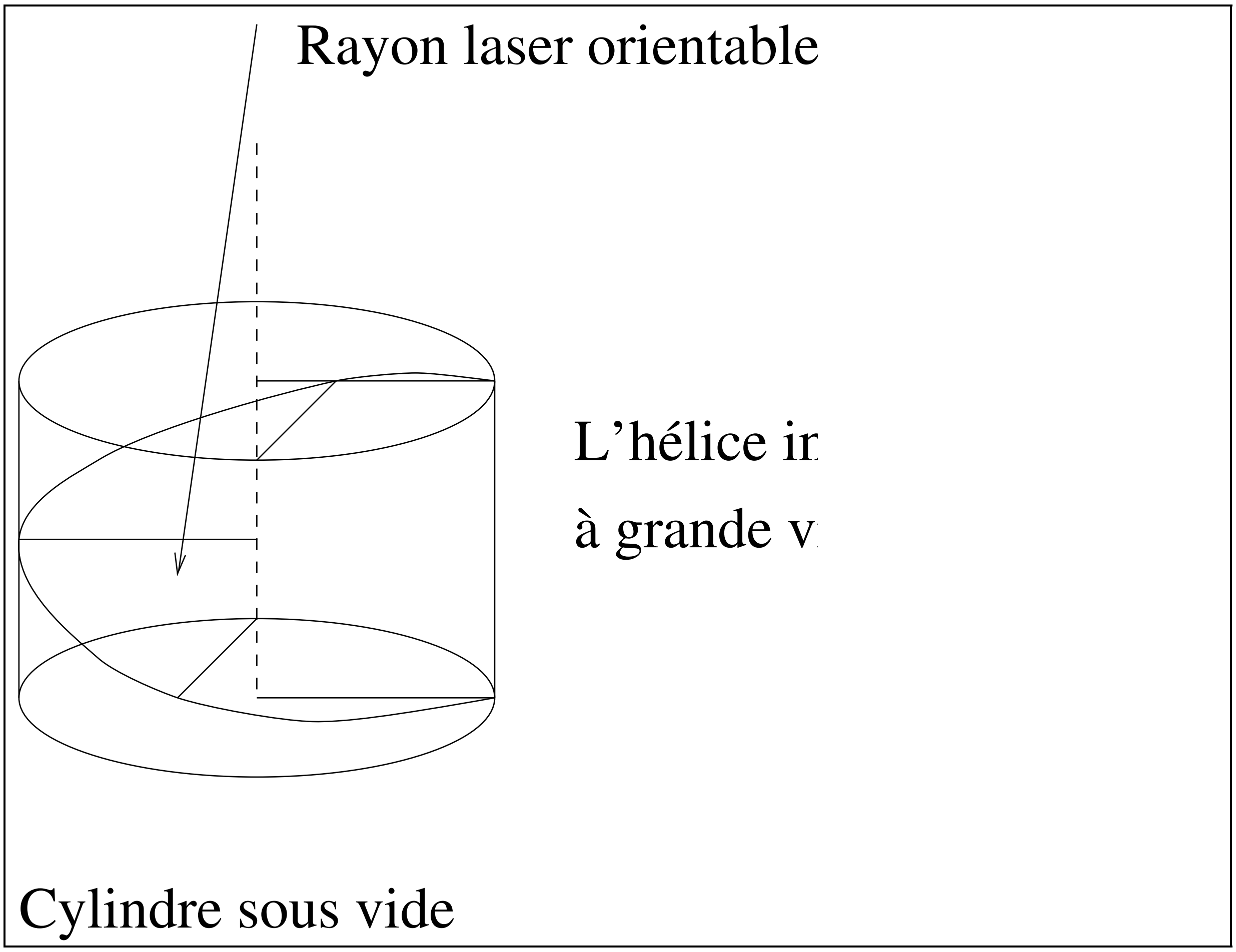


Fig. 3 – *Ecran 3D ponctuel*

Une autre possibilité expérimentée est de projeter une image sur une écran qui vibre; L'épaisseur de l'écran est l'amplitude de la vibration. Le problème est bien sùr le débit d'information qui est énorme dans le cas de matrices de pixel. On doit aussi travailler avec des phosphores très peu réémanents.

2.3.3 Ecrans holographiques

Ils seront possibles dès que l'on pourra afficher un résolution de 4000 points (à vérifier) par mm en cristaux liquides. Ceci permet en effet de faire des interférences.

2.4 Ecran matriciel (à balayage de trame)

L'image est découpée en lignes, et les lignes en pixels. Elle est affichée pixel par pixel. Différentes technologies sont utilisées pour l'affichage :

- Tubes à rayons cathodiques (CRT : Cathodic Ray Tube).
- Ecran à plasma. (mémoire)
- Diodes electroluminescentes.
- Cristaux liquides.
- Matrice active. (mémoire)

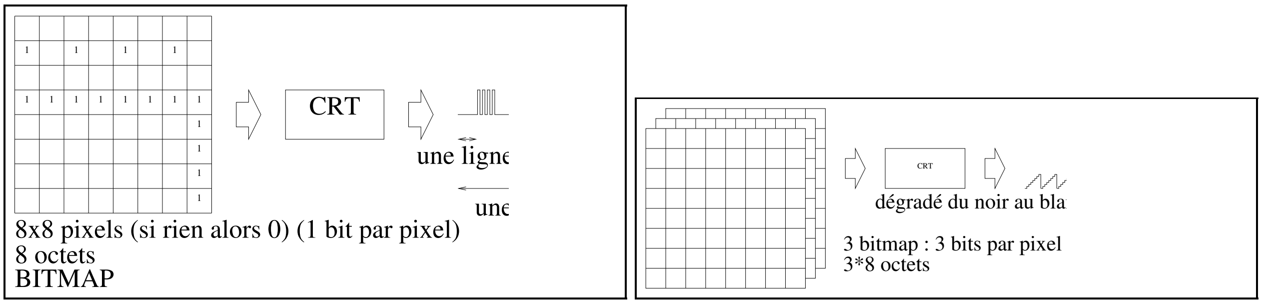


Fig. 4 – *Ecrans monochromes (noir et blanc et niveaux de gris)*

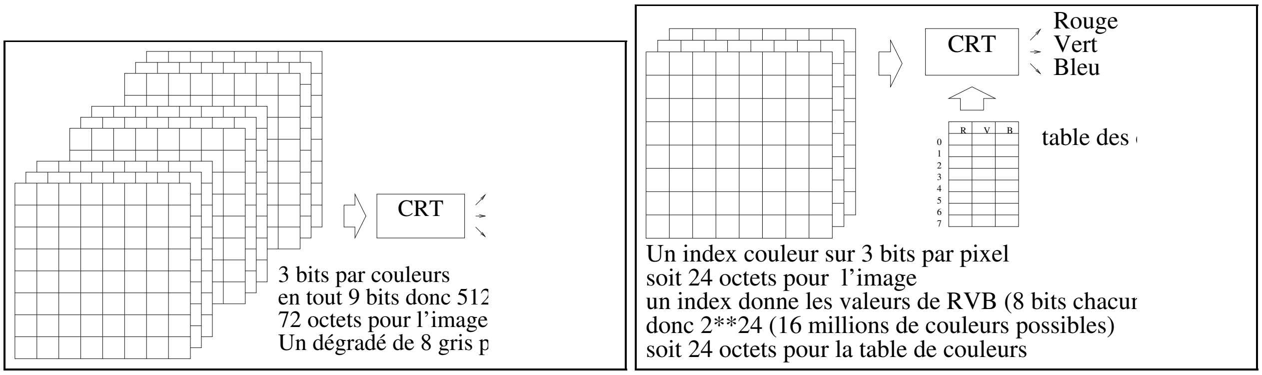


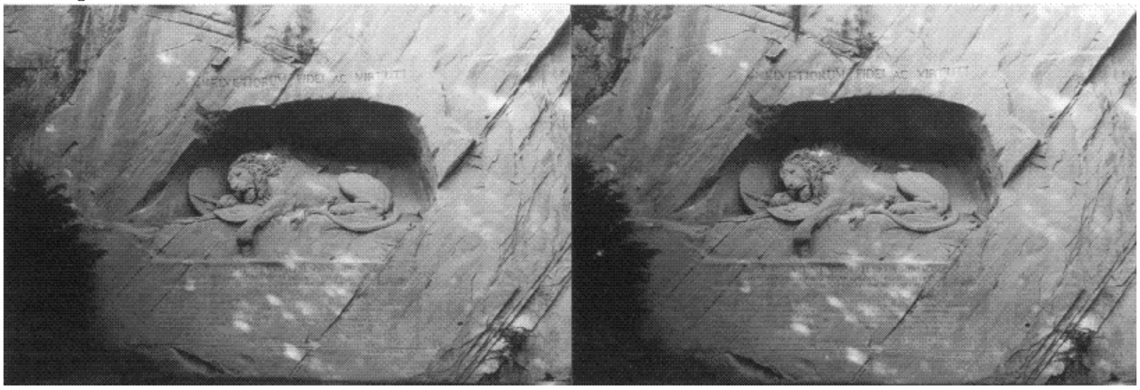
Fig. 5 – *Ecrans couleurs (directes ou par table de couleurs)*

Maximum 1991 (1x3500x2500 et 12x1600x1280). Ecran classique haute définition couleur :

- Taille écran 19 pouces : 35x27 cm
- Résolution 1280x1024 pixels.
- Taille pixel 0.27 millimètre soit approximativement 100dpi.
- L'oeil séparant des points avec une angulaire de 4.10^{-4} radian. On doit être à une distance de $\frac{0.27}{4.10^{-4}}$mm soit 67cm pour ne pas voir les pixels.
- 256 couleurs par pixel (un octet par pixel).
- Taille mémoire écran : 1280*1024=1310720
- Choix des 256 couleurs parmis 2^{24} soit 768 octets de table de couleur.
- Rafraichissement 66Hz.
- Débit d'information :
  1310720*66 = 86Mo par seconde, pour l'image.
  3*1310720*66 = 258Mo par seconde pour la table des couleurs.

2.5 Images stéreo

Stéréogrammes



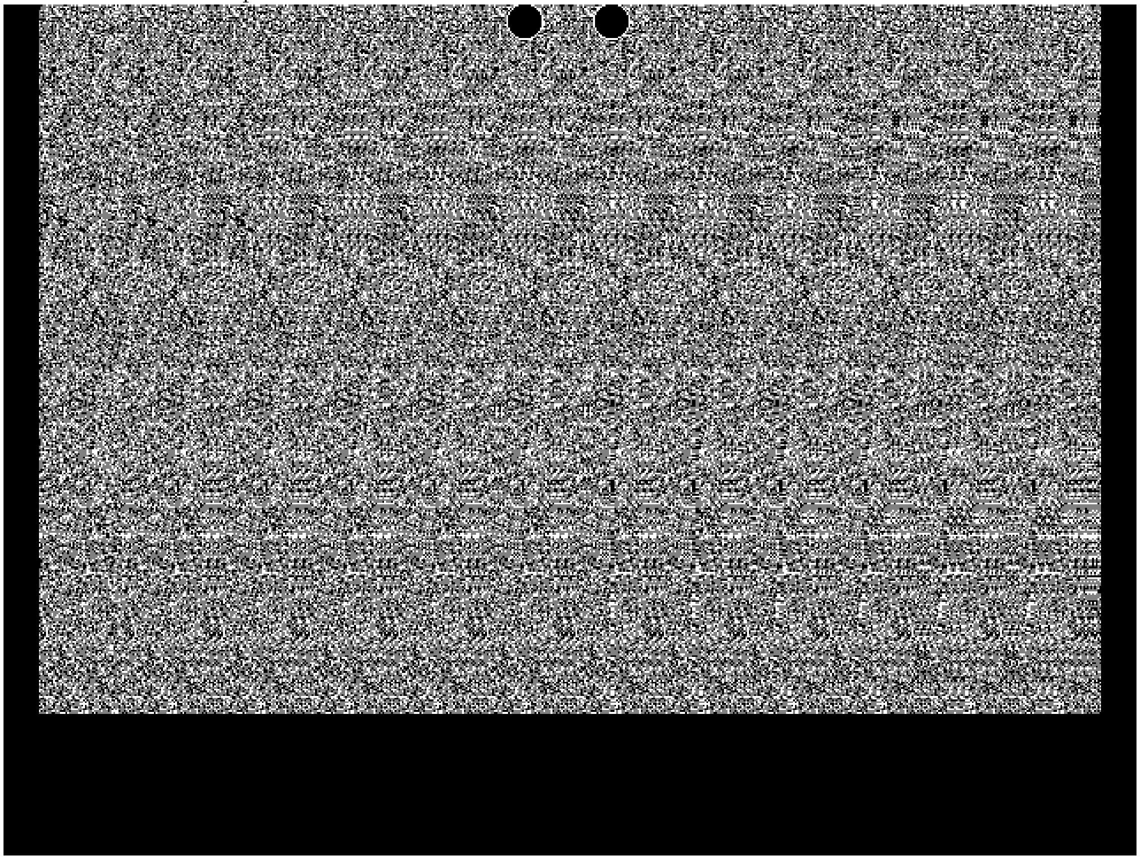
Des SIRDs (Single Image Random Dots) avec des caractères

```
      0   0
        n   n   n   n   n   n   n   n   n   n   n   n   n   n   n   n
        f    f    f    f    f    f    f    f    f    f    f    f    f
        e   e   e   e   e   e   e   e   e   e   e   e   e   e   e   e
        a    a    a    a    a    a    a    a    a    a    a    a    a
        a   a   a   a   a   a   a   a   a   a   a   a   a   a   a   a
        r    r    r    r    r    r    r    r    r    r    r    r    r
        r   r   r   r   r   r   r   r   r   r   r   r   r   r   r   r

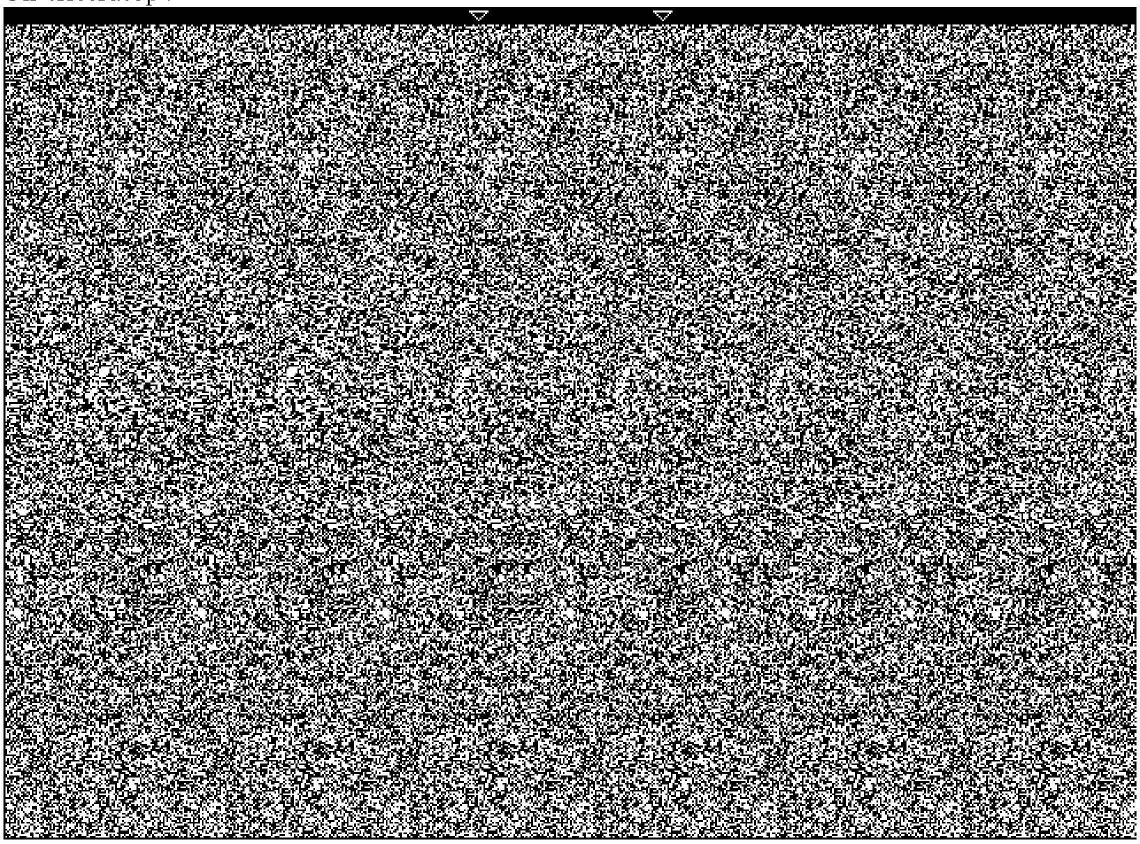
 g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g  g
  r   r   r   r   r   r   r   r   r   r   r   r   r   r   r   r   r   r
    e    e    e    e    e    e    e    e    e    e    e    e    e    e
    a     a     a     a     a     a     a     a     a     a     a     a
      t      t      t      t      t      t      t      t      t      t
<<>>>><<<<>>>><<<<>>>><<<<>>>><<<<>>>><<<<>>>><<<<>>>><<<<>>>><<<<>>>>
      d      d      d      d      d      d      d      d      d      d
    e     e     e     e     e     e     e     e     e     e     e     e
    p    p    p    p    p    p    p    p    p    p    p    p    p    p
  t   t   t   t   t   t   t   t   t   t   t   t   t   t   t   t   t  t
 h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h  h

Techniques Techniques Techniques Techniques Techniques Techniques Techniques
Graphiques  Graphiques  Graphiques  Graphiques  Graphiques  Graphiques
Techniques Techniques Techniques Techniques Techniques Techniques Techniques
Graphiques  Graphiques  Graphiques  Graphiques  Graphiques  Graphiques
Techniques Techniques Techniques Techniques Techniques Techniques Techniques
Graphiques  Graphiques  Graphiques  Graphiques  Graphiques  Graphiques
```

Des SIRDs avec des points aléatoires



Un triceratop :



2.6 Dispositifs d'impression

Synthèse soustractive des couleurs. On ne présentera pas les traceurs de courbes à plat ou à rouleau.

2.6.1 Impression matricielle

Utilisée pour la copie d'image ou d'écran.

- Impact.
  Avantages :
  - Pas cher.

  Inconvénients :
  - Bruyant.
  - Résolution faible à moyenne.
- Jet d'encre.
  Inconvénients :
  - Entretiens plus difficile.
  - Résolution faible à moyenne.
- Thermique.
  Inconvénients :
  - Le papier supporte mal le passage du temps.
  - Résolution faible à moyenne.
  - Noir & blanc seulement.
- Laser et electrostatique.
  Avantages :
  - Haute résolution. $>$ 1200dpi

  Inconvénients :
  - Cher (de moins en moins).
- Sublimation.
  Avantages :
  - Haute résolution.
  - Niveau de gris et couleur sans dithering.

  Inconvénients :
  - Cher (de moins en moins).
- Photographique.
  Avantages :
  - TRES Haute résolution.
  - Niveau de gris et couleur.

  Inconvénients :
  - TRES Cher.
- Stéréolithographie.

2.7 Dispositifs de saisie

Ils permettent de faires des saisies de différentes choses.

2.7.1 Locator device

Ce sont des dispositifs permettant de saisir directement des coordonnées.

- La tablette à digitaliser.
  Le support contient deux ensembles de conducteurs, le stylo émet un signal electro-magnétique. La pointe du stylo est en plus un bouton.
  Avantage :

  - Permet la digitalisation de documents.

  Inconvénients :

  - Encombrant.
  - Sensible aux champs magnétiques
  - Fragile.

  Au point de vue réalisation, d'autre techniques sont utilisables, les plus intéressant n'utilisent pas de stylo.

  - Par détection de pression.
  - Par interception de rayons lumineux.
  - Par changement de capacité.

  La technique du stylo optique a été abondonnée.

- Déplacements relatifs.
  La souris et le trackball permettent d'avoir un vecteur déplacement relatif. La souris peut être mécanique ou optique. La souris mécanique pose plus de problèmes mécaniques et l'optique limite les déplacements.

- Directions et pressions.
  Tout ce qui est du genre manche à balais. Le plus connu est le joystick. On peut arriver à 6 degrés de liberté. L'accès à une position donnée est plus long, on ne fait pas d'accès direct.

2.7.2 Touches

Le clavier et les boîtes à boutons. Il y a souvent des LED au dessus des boutons. Ou encore mieux : un petit afficheur.

2.7.3 Valuateur

Les périphériques qui permettent de saisir une valeur flottante : des potentiomètres avec aussi peut-être un affichage de la valeur courante.

2.7.4 Data glove, ...

Différents dispositif permettant une saisie de la position du corps humain, notamment les mains et la tête.

2.7.5 scanner

Saisie d'une image avec une caméra ou bien un dispositif moins couteux si l'image est sur papier.

3 Matrice de pixel

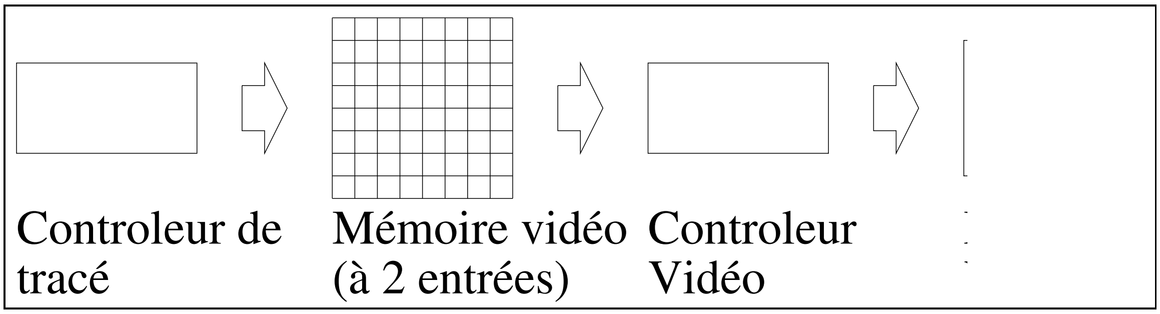


FIG. 6 – *La matrice de pixel et ses accès*

Pour dessiner sur l'écran il faut donc modifier les valeurs de la matrice de pixel. C'est le contrôleur de tracé qui s'occupe généralement de cela. Il faut donc faire des programmes pour :

- Afficher un pixel.
- Tracer un segment de droite.
- Tracer du texte.
- Tracer un rectangle plein.
- Un polygone plein.
- Un arc de cercle.
- Un disque.

La couleur de tracé dépend de :

- La couleur avec laquelle on affiche.
- La couleur de ce qui est déjà affiché.
- La couleur du "fond" (pointillés, texte, ...)

De nombreux modes possibles. Dans les tables suivantes, *src* est la couleur dans laquelle on veut afficher et *dst* la couleur déjà présente sur le pixel. On donne un tracé de pointillé en exemple. La couleur de tracé est 1 et 0 pour la couleur de fond. Dans la réalité, on ne trace pas tous les pixels du pointillé.

SET
res=1

dst / src	0	1
0	1	1
1	1	1

CLEAR
res=0

dst / src	0	1
0	0	0
1	0	0

COPY
res=src

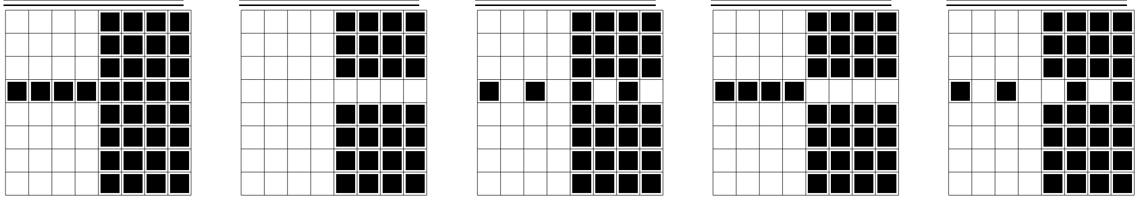
dst / src	0	1
0	0	0
1	1	1

INVERT
res=NOT dst

dst / src	0	1
0	1	0
1	1	0

XOR
res=src XOR dst

dst / src	0	1
0	0	1
1	1	0



16 tables sont possibles, il y a donc 16 modes d'affichage. En couleur la même opération s'applique à chacun des bits séparément.

3.1 Affichage d'un pixel

On suppose que lorsque l'on travaille en couleur les opérations sont faites simultanéments dans les différents plans. Pour la suite on se considère donc en noir et blanc.

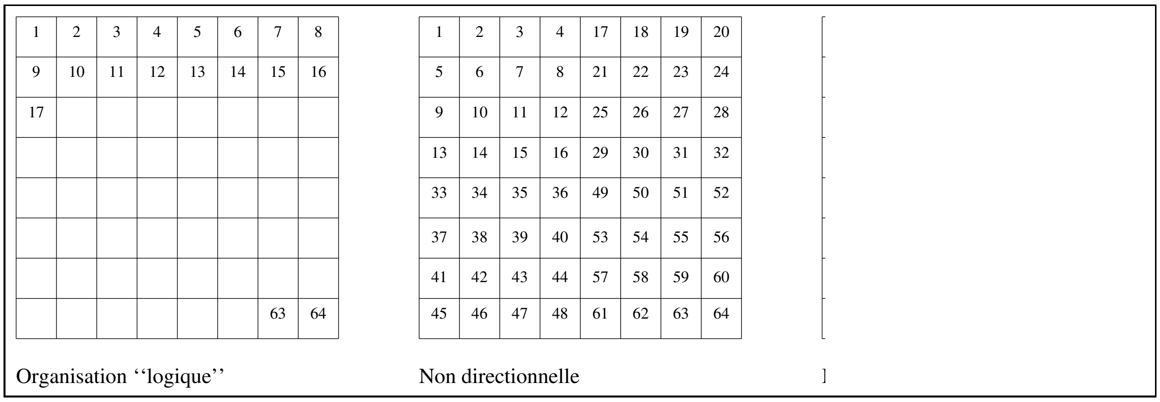


FIG. 7 – *Quelques organisations possible d'une bitmap*

Si l'on est dans le premier cas et que l'on se limite à l'opération SET et si l'origine (0,0) est en haut à gauche on obtient le programme suivant :

```
set(x,y,ecran,largeur,hauteur)
int x,y,largeur,hauteur ;
unsigned char ecran[] ;
{ ecran[x/8 + y*(largeur/8)] |= 128>>(x%8) ; }
```

Plus simple de faire plusieurs bitmap pour une pixmap que de mettre les bits d'un même index les uns à coté des autres. Le hardware en est simplifiée.

Pour accélérer les calculs : supprimer division et modulo. Utiliser un état courant. Le concept de tortue.

```
#define LAR 80
#define HAU 25
static unsigned char ecran[LAR/8*HAU] ;
static int octet_courant ;
static unsigned char masque ;
positionne(x,y)
int x,y ;
{ octet_courant = x/8 + y*(LAR/8) ;
  masque        = 128>>(x%8) ;
}
a_droite()
{ if ( masque==1 ) { masque = 128 ;
                     octet_courant++ ;
                   }
              else   masque >>= 1 ;
}
en_bas()
{ octet_courant += LAR/8 ; }
```

Ces exemples sont bien entendue simplifié. La programmation nécessite des pointeurs et structures afin d'être plus propre. Aucun test de débordement n'est fait.

3.2 Tracé de droites

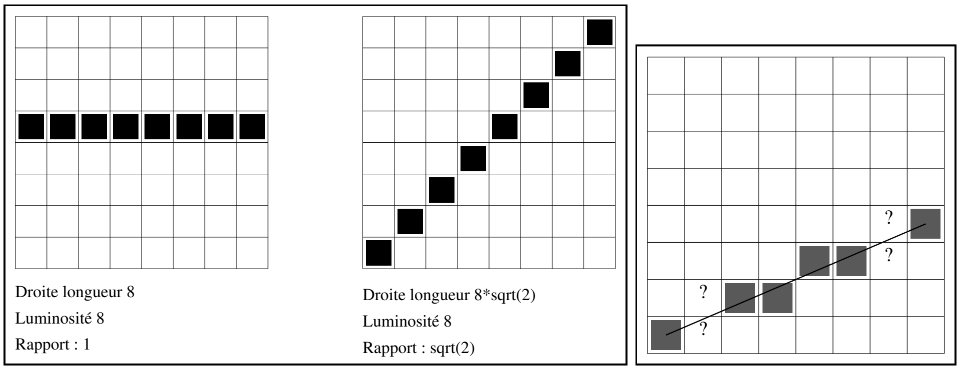


Fig. 8 – *Problème de luminosité et de choix*

Différentes façons de tracer une droite (ou tout autre courbe) :

- Utilisation directe de la fonction.
- Méthode de l'Analyseur Différentiel Numérique (calcul tangente).
- Algorithme incrémental.
- Algorithme incrémental optimisé (nombres entiers sans * ou /).
- Algorithme incrémental optimisé spécifique au hard.
- Algorithme récursif.

Utilisation direct de la fonction.

L'équation de la droite est $y = ax + b$ (avec transposition si nécessaire). La procédure suivante est déjà améliorée, elle calcule un seul pixel par ligne ou colonne. elle ne trace que dans le premier octant.

```
void droite(x1,y1,x2,y2)
float x1,y1,x2,y2 ;
{
float dx,dy ;
float a,b ;
int x ;
dx = x2-x1 ;
dy = y2-y1 ;
a = dy/dx ;     /* /0 si droite degeneree */
b = y1 - a*x1 ; /* Triangles semblables */
for(x=x1;x<=x2;x++)
      set( x,(int)(a*x+b) ) ;
}
```

3.2.1 Tracé de droite par MADN

Soit le segment de droite reliant $(x1, y1)$ à $(x2, y2)$.
Soit (x, y) un point sur la droite à l'abscisse curviligne t.
Nous avons :

$$x = x1 + t * (x2 - x1)$$

$$y = y1 + t * (y2 - y1)$$

Avec $t \in [0 \ldots 1]$
La tangente nous ai donnée en dérivant.

$$\frac{\partial x}{\partial t} = x2 - x1$$

$$\frac{\partial y}{\partial t} = y2 - y1$$

Nous pouvons donc écrire :

$$x_{i+1} = x_i + \Delta x$$

$$y_{i+1} = y_i + \Delta y$$

Pour implanter cet algorithme, il faut déterminer le nombre de pixel à afficher. Il faudra diviser la tangente par ce nombre. Normalement le nombre de pixel est la longueur de la droite, dans la réalité c'est autre chose...

```
droite(x1,y1,x2,y2)
float x1,y1,x2,y2 ;
{
float dx,dy ;
int longueur ;

longueur = max( abs(x2-x1) , abs(y2-y1) ) ;
dx = (float)(x2-x1)/longueur ;
dy = (float)(y2-y1)/longueur ;

do
     {
     set((int)x1,(int)y1) ;
     x1 += dx ; /* Addition FLOTTANTE */
     y1 += dy ; /* Addition FLOTTANTE */
     }
while(longueur--) ;
}
```

Il faut remarquer que cet algorithme est incrémental.
Il est en nombres flottants.
Il nécessite 2 divisions.
On peut le modifier de manière plus ou moins malpropre pour travailler en entier, le premier algorithme propre a été fait par BRESENHAM

3.2.2 Algorithme de BRESENHAM (1965)

On suppose pour la démonstration que le segment de droite commence en $(0,0)$, est dans le premier octant et de pente a On ne détaille l'algorithme de BRESENHAM que dans ce cas. Pour les autres, on s'y ramène par translation, symétries et transposition.

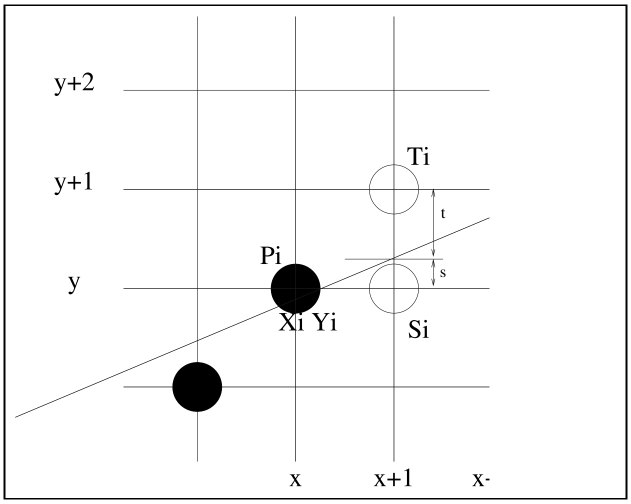


FIG. 9 – *Le problème du choix (les pixels sont les cercles)*

L'algorithme vient d'allumer le point P_i. Le pixel suivant est forcément à droite car l'on est dans le premier octant. Il peut-être sur la même ligne ou bien celle du dessus. Si $t > s$ alors on choisi S_i sinon T_i.

Soit a la pente de la droite $(0 \leq \frac{dy}{dx} \leq 1)$

L'ordonnée de l'intersection est : $(x_i + 1) * a$

Donc :

$$s = (x_i + 1) * a - y_i$$

$$t = y_i + 1 - (x_i + 1) * a$$

Soit $C_i = s - t$

$$C_i = s - t = 2 * a * (x_i + 1) - 2 * y_i - 1$$

Si $C_i < 0$ alors S_i sinon T_i.

On peut maintenant appliquer une astuce pour passer en entier, le test se faisant par rapport au signe, on peut multiplier par un nombre positif, par exemple dx

$$C'_i = dx * C_i = 2 * dy * (x_i + 1) - 2 * y_i * dx - dx$$

On peut d'ores et déjà écrire un algo en nombres entiers mais avec multiplications. Passons en incrémentale pour supprimer les multiplications.

$$C'_0 = 2 * dy - dx$$

$$C'_{i+1} - C'_i = 2 * dy * (x_{i+1} - x_i) - 2 * (y_{i+1} - y_i) * dx$$

Or $x_{i+1} - x_i = 1$

$$C'_{i+1} - C'_i = 2 * dy - 2 * (y_{i+1} - y_i) * dx$$

Si $C'_i < 0$ alors $y_{i+1} - y_i = 0$ sinon 1

Si $C'_i < 0$ alors $C'_{i+1} - C'_i = 2 * dy$ sinon $2 * dy - 2 * dx$

On peut donc calculer le suivant à partir du précédent uniquement par additions.

Des améliorations ont étés faites : Calcul par les 2 extrémités à la fois, calcul d'une condition pour les 2 pixels suivants, détection de la périodicité.

3.2.2.1 Implémentation de l'algorithme de BRESENHAM :

Ceci est une implantation lisible mais non optimale. Dans tous les octants.

```
droite(x1,y1,x2,y2)
int x1,y1,x2,y2 ;
{
int dx,dy ;
int dirx,diry ;
int c,c1,c2 ;
int principale[2] ; /* Vecteur direction principale */
int nb ;
```

```
dx = x2-x1 ;
if ( dx<0 ) { dirx = -1 ; dx = -dx ; }
       else { dirx =  1 ; }
dy = y2-y1 ;
if ( dy<0 ) { diry = -1 ; dy = -dy ; }
       else { diry =  1 ; }
if ( dx>dy ) {
             principale[0] = dirx ;
             principale[1] = 0 ;
             nb = dx ;
             }
        else {
             principale[0] = 0 ;
             principale[1] = diry ;
             nb = dy ;
             dy = dx ;
             dx = nb ;
             }
c  = 2*dy-dx ;
c1 = 2*dy ;
c2 = 2*dy-2*dx ;
do
   {
   set(x1,y1) ;
   if ( c<0 )
              {
              x1 += principale[0] ;
              y1 += principale[1] ;
              c  += c1 ;
              }
         else {
              x1 += dirx ;
              y1 += diry ;
              c  += c2 ;
              }
   }
while(x1!=x2) ;
set(x2,y2) ;
}
```

3.2.3 Diverses folies

3.2.3.1 Récursif : Pour afficher un segment de 1 pixel de long pas de problème! Si ils sont plus long : on les découpe. Attention l'algo est approximatif en nombres entiers.

```
droite(x1,y1,x2,y2) int x1,y1,x2,y2 ;
{ int x,y ;
  x = (x1+x2)/2 ;
  y = (y1+y2)/2 ;
  if ( x==x1 && y==y1 ) { set(x,y) ; return ; }
  if ( x==x2 && y==y2 ) { set(x,y) ; return ; }
  droite(x1,y1,x,y) ;
  droite(x,y,x2,y2) ;
}
```

3.2.3.2 Par paliers : Autre possibilité : on recherche la longueur du palier avant le prochain décrochement. Cette méthode est rapide pour des droites proches des axes. Pour une droite dans le premier octant. La longueur du palier est $\frac{dx}{dy}$

```
droite(x1,y1,x2,y2) int x1,y1,x2,y2 ;      /* Primier octant */
{ float m,e ; int dx=x2-x1,dy=y2-y1 ;
  e = 0 ;                                  /* Erreur d'arrondie */
  m = dx/(float)dy ;                       /* Inverse de la pente */
  do                                       /* Calcul le pas horizontale */
     { e += m ;                            /* Calcul longueur palier */
       palier(x1,y1,(int)e) ;              /* Affiche le palier horizontal */
       x1 += (int)e ;                      /* Calcul le dernier point */
       y1++ ;                              /* Ligne suivante */
       e -= (int)e ;                       /* Calcul l'erreur */
     }
  while(x1<=x2) ;
}
```

3.2.3.3 Horizontale : Optimisée pour le hardware. `tablebit` contient 11111111, 01111111, 00111111, 00011111, 00001111, 00000111, 00000011, 00000001, 00000000. Avec une organisation normale des pixels :

```
horiz(x,y,longueur) int x,y,longueur ;
{ int x8 ;
  int octet,octetfin ;
  x8 = x%8 ;
  octet = x/8+(LAR/8)*y ;
  octetfin = (x+longueur)/8+(LAR/8)*y ;
  if ( x8+longueur>=8 ) /* Deborde de l'octet */
     { ecran[octet] |= tablebit[x8] ;
       octet++ ;
     }
  else
     { ecran[octet] |= tablebit[x8] & ~tablebit[x8+longueur] ;
       return ;
     }
  while( octet<octetfin ) ecran[octet++] = 255 ;
  ecran[octet] |= ~tablebit[(x8+longueur)%8] ;
}
```

3.3 Remplissage d'un polygone

C'est une des opération les plus courante de l'affichage. Les polygones peuvent être convexes, avoir des concavités ou bien être auto-intersectant. Les méthodes de remplissage sont diverses et variées, elles dépendent du type de polygone. Sur les cartes graphiques, on se limite à des trapèze, tout polygone qui ne s'intersecte pas lui-même peut facilement est découpé en trapezes.

Méthode initiale: pour chacun des points de l'écran (ou du plus petit rectangle contenant le polygone) on regarde si le point appartient au polygone ou non. Pour le savoir plusieurs méthodes existent : la somme des angles ou le comptage des intersections avec une demi-droite issu du point considéré. Cette méthode de test est appelé suivi de contour. Elle est d'une programmation complexe (erreurs numériques)

Méthode "riche" : On trace le contour du polygone (un pixel par ligne) sur une bitmap (le contour peut alors avoir n'importe quelle forme) et l'on parcour cette bitmap ligne par ligne en traçant lorsque le nombre de contour traversé est paire.

Méthode optimale (incrémentale optimisé) : On trie les points (bas en haut, gauche droite). On part du bas en remplissant ligne par ligne. C'est l'algorithme du scan-line. La première intersection avec un segment est un point d'extrémité. Les intersections suivantes sont calculées avec un BRESENHAM. On trace donc plusieurs droites à la fois. On n'utilise que des entiers, on ne fait ni multiplication ni divisions.

```
Pour tous les segments du contour
    Mettre le point le plus bas en premier

Trier les points de l'ordonnee la plus petite a la plus grande.
Enlever les segmetns horizontaux.
Pour l'ordonnee la plus petite a la plus grande (pixel par pixel)
    Pour tous les BRESENHAM se terminant
         Enleve le BRESENHAM de la liste
    Pour tous les BRESENHAM en cours
         Passe a l'etape suivante du BRESENHAM
    Pour tous segments commencant a cette ordonnee
         Initialise un nouveau BRESENHAM
         Remarque : si une pointe v alors 2 nouveaux BRESENHAM
    Trier les points d'intersection (points courants des BRESENHAMS)
    Tracer des lignes horizontales entre les points pris 2 par 2
```

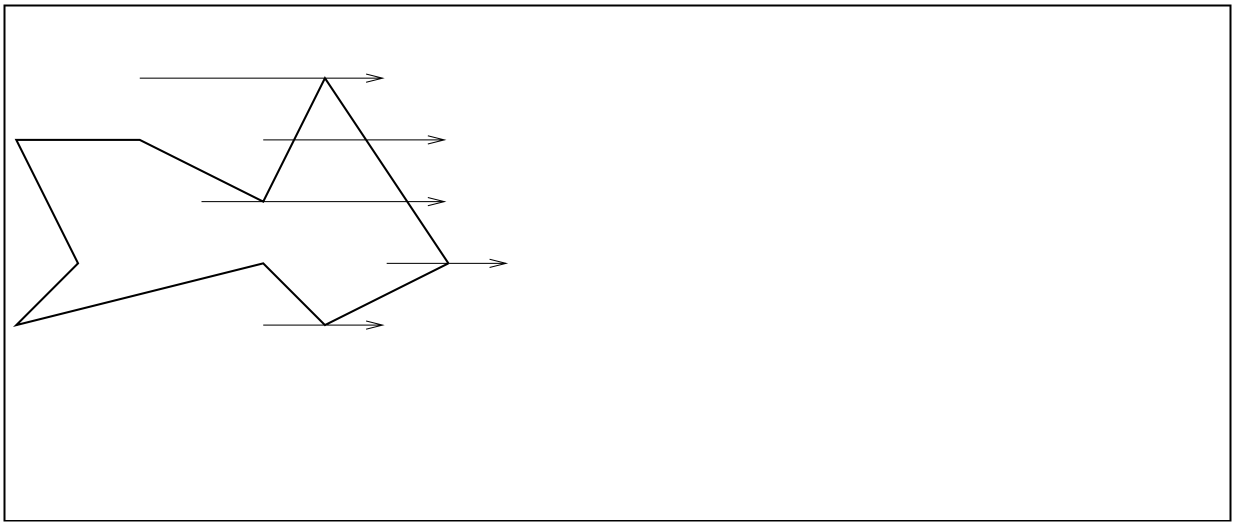


FIG. 10 – *Point dans polygone et remplissage de polygone*

3.4 Tracé de cercle et d'arc

Pour les ellipses on utilise des transformations. On retrouve les différentes méthodes :

3.4.1 Utilisation directe de la fonction

```
cercle(cx,cy,r) int cx,cy,r ;
{ float teta ;
int lx,ly,nx,ny ;
lx = x + r ;
ly = y ;
teta = 0 ;
do
nx = cx + (int)(r*cos(teta)) ;
ny = cy + (int)(r*sin(teta)) ) ;
droite(lx,ly,cx,cy) ;
lx = cx ;
ly = cy ;
teta += 2*M_PI/200 ;
while ( teta <= 2*M_PI+1e-6 ) ;
}
```

Les inconvénients :

- Calcul de `sin` et `cos` qui sont long.
- Calcul en flottant.
- Affichage peut convaincant (pas régulier.)
- On doit pouvoir tracer une droite rapidement.

3.4.2 MADN

Nous avons :

$$x(\theta) = c_x + r\cos(\theta)$$
$$y(\theta) = c_y + r\sin(\theta)$$

Donc :

$$\frac{\partial x(\theta)}{\partial\theta} = -r\sin(\theta)$$
$$\frac{\partial y(\theta)}{\partial\theta} = r\cos(\theta)$$

En remettant en forme :

$$\frac{\partial x(\theta)}{\partial\theta} = c_y - y(\theta)$$
$$\frac{\partial y(\theta)}{\partial\theta} = x(\theta) - c_x$$

En supposant le cercle centré en $(0,0)$:

$$x_{i+1} = x_i - \epsilon y_i$$
$$y_{i+1} = y_i + \epsilon x_i$$

Ceci revient en fait à appliquer une matrice de rotation d'un angle θ avec $\sin(\theta) = \epsilon$

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\epsilon \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

Problèmes : la valeur de ϵ et le déterminant.
Si ϵ est trop petit, on allume plusieurs fois le même point sinon il y a des trous dans le tracé du cercle. On peut prendre $\epsilon = 2^{-n}$ si $2^{n-1} \leq r < 2^n$ ATTENTION c'est un exemple, cela permet de travailler en entiers.
Le déterminant est égale à $1+\epsilon^2$, on va donc tracer une spirale au lieu d'un cercle. Pour résoudre ce problème on prend comme matrice :

$$\begin{pmatrix} 1 & -\epsilon \\ \epsilon & 1-\epsilon^2 \end{pmatrix}$$

On peut utiliser un nombre à virgule fixe '*256'

```
float epsilon ;
x = r ;
y = 0 ;
epsilon = 1./r ;
do {
set(x+cx,y+cy) ;
x = x - epsilon*y ;
y = y + epsilon*x ;
} while(x!=r) ;
```

3.4.3 Algorithme incrémental optimal

On ne considère que le tracé dans le $2^{\text{ème}}$ octant dans le sens inverse trigo. On considère que le cercle est de centre $(0,0)$, une translation suffit pour le repositionner. Pour tracer complètement le cercle, on fait les 8 symétries.

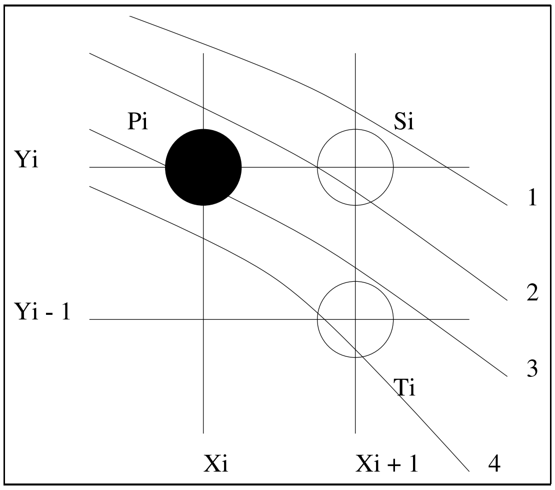


FIG. 11 – *Le problème du choix (les pixels sont les cercles)*

Le point P_i a été choisi.
Le prochain point ne peut être que S_i ou T_i
On calcule la distance du point au centre du cercle, on calcule en fait son carré.

$$d^2(S_i) = (x_i+1)^2 + y_i^2$$

$$d^2(T_i) = (x_i+1)^2 + (y_i-1)^2$$

La distance (carré) d'un point sur le cercle devrait être : r^2 On prend donc le point le plus proche de cette distance,

$$e(S_i) = d^2(S_i) - r^2$$

$$e(T_i) = d^2(T_i) - r^2$$

le critère de choix est donc :

$$C_i = |e(S_i)| - |e(T_i)|$$

Si $C_i \leq 0$ on choisit S_i sinon T_i.

Cas 1 $e(S_i) < 0$ et $e(T_i) < 0$: $C_i = -e(S_i) + e(T_i) = -2y_i + 1 < 0$
or $0 > -e(S_i) + e(T_i) > e(S_i) + e(T_i)$

Cas 2 et 3 $e(S_i) > 0$ et $e(T_i) < 0$: $C_i = e(S_i) + e(T_i) = 2(x_i+1)^2 + 2y_i^2 - 2y_i + 1 - 2r^2$

Cas 4 $e(S_i) > 0$ et $e(T_i) > 0$ $C_i = e(S_i) - e(T_i) = 2y_i - 1 > 0$ or $0 < e(S_i) - e(T_i) < e(S_i) + e(T_i)$

On fait donc comme si l'on était dans le cas 2 ou 3. Les autres cas sont pris en compte quand même par effet de "bord".

$$C_0 = 3 - 2r$$

$$C_{i+1} - C_i = \begin{cases} 6 + 4x_i & \text{si } C_i \leq 0 \\ 10 + 4x_i - 4y_i & \text{si } C_i > 0 \end{cases}$$

3.5 Remplissage d'une région

On veut remplir une zone de l'écran délimité par des pixels d'une même couleur ou bien changer la couleur d'une zone contiguë de pixel de même couleur. Remplissage connexité 4 ou 8. Voici l'algo initial (On ne tient pas compte des bords) :

```
remplissage_bord(x,y,couleur_bord)
int x,y,couleur_bord ;
{ if ( couleur(x,y)==couleur_bord ) return ;
  set(x,y) ;
  remplissage_bord(x-1,y,couleur_bord) ; remplissage_bord(x+1,y,couleur_bord);
  remplissage_bord(x,y-1,couleur_bord) ; remplissage_bord(x,y+1,couleur_bord);
}
```

Cet algorithme est inapplicable car la profondeur de récursion peut-être égale au nombre de pixels. On l'améliore donc en utilisant 2 buffers de taille certaine :-)

```
Initialise un tableau A avec le point germe
Initialise un tableau B de coordonnees a vide
Tant que A n'est pas vide
    Pour tous les points de A qui ne sont pas contour
          Afficher le point
          Ajouter les 4 voisins a B
    Vider A
    Echanger A et B
```

Problème restant la possibilité de remplir avec un motif. D'autre part la méthode test plusieur fois chaque pixel, le tracé se fait pixel par pixel. Voici un algorithme pour remplir par ligne et qui évite les retours en arrière. On appel axe un segment horizontal.

```
On empile le point de depart comme un axe de developpement.
Tant que la pile n'est pas vide
    Prendre un axe de developpement
    Remplir l'axe sur sa longueur.
    Si on remplit des axes existants dans la pile, on les detruit.
    On empile les futurs axes de developpement (sans revenir en arriere).
```

Une dernière méthode utilise une bitmap intermédaire dans laquelle on marque la zone déja rempli. On balaye l'écran successivement dans les 4 directions jusqu'à atteindre le contour (algo long si spirale mais très facilement cablable).

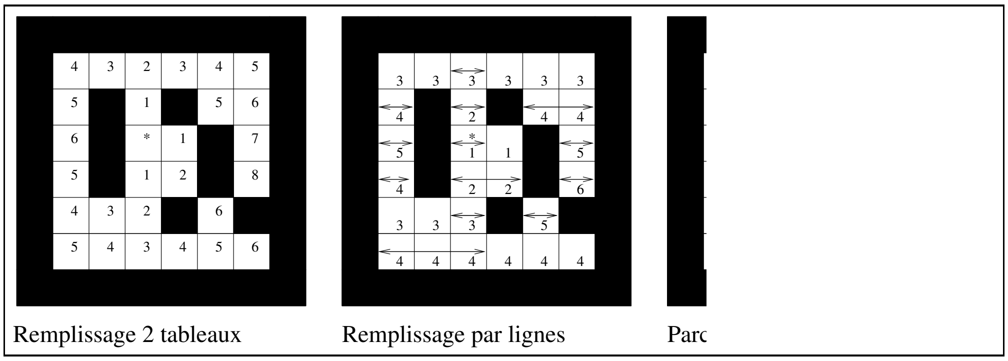


FIG. 12 – *Différentes méthodes de remplissage*

3.6 Réduction de couleur

Si l'on veut afficher une image comportant un certain nombre de couleurs sur un support (écran ou papier) en contenant moins,

Si l'image d'arrivé est en noir et blanc et l'image de départ en couleur on applique la transformation suivante : niveau = .3*R + .11*B + .59*V Puis on travaille sur l'image en niveaux de gris.

Une première possibilité consiste à trouver les couleurs les plus utilisées de l'image de départ. Cette technique est plus liées au traitement d'image.

Dans la plupart des cas, on a droit à une palette limitée de couleur et l'on doit transcrire l'image dans cette pallette.

Plusieurs méthodes :

- On utilise un "pattern" pour simuler les niveaux de gris. Le nombre de couleurs simulables dépend de la taille du pattern. Méthodes simple et rapide car les patterns sont calculés à l'avance.
- On décide de la couleur du pixel au coups par coups, en utilisant des méthodes plus raffinées.

3.6.1 Substitution pixel par pattern de pixel

Les exemples suivant sont en noir et blancs.

```
#define TAILLE_PAT 3
#define GRIS (TAILLE_PAT*TAILLE_PAT+1)
int pattern[GRIS][TAILLE_PAT][TAILLE_PAT] ;

expand256vers2(ancien,nouveau,lar,hau)
unsigned char **ancien,**nouveau ;
{ int x,y,px,py,gris ;
  for(x=0;x<lar;x++)
   for(y=0;y<hau;y++)
    { gris = (ancien[x][y]*(GRIS-1))/255 ;
      for(px=0;px<TAILLE_PAT;px++)
       for(py=0;py<TAILLE_PAT;py++)
         nouveau[x*TAILLE_PAT+px][y*TAILLE_PAT+py] = pattern[gris][px][py] ;
    }
}
```

L'inconvénient est l'augmentation de la taille de l'image.

3.6.2 Substitution du pixel par un pixel du pattern

```
  for(x=0;x<lar;x++)
   for(y=0;y<hau;y++)
    { gris = (ancien[x][y]*(GRIS-1))/255 ;
         nouveau[x][y] = pattern[gris][x%TAILLE_PAT][y%TAILLE_PAT] ;
    }
```

Dans les 2 cas l'image est tramée. L'algorithme pour ne pas faire de trame est à base de diffusion d'erreurs.

3.6.3 Réduction de couleur par diffusion d'erreur

L'algorithme est des plus simple :

- On prend un pixel de l'image de départ.
- On choisi la couleur de la pallette la plus proche de la couleur du pixel.
- On affiche la couleur choisi.
- On modifie les pixels voisins pour corriger l'erreur commise.

Problèmes de cette méthode : Lentes et dépendante de l'ordre de choix des pixels.
Voici un algo non optimisé :

```
for(x=0;x<lar;x++)
 for(y=0;y<hau;y++)
   {
   if ( image[x][y]>128 )
      {
      erreur = image[x][y]-255 ;
      image[x][y] = 255 ;
      }
   else
      {
      erreur = image[x][y] ;
      image[x][y] = 0 ;
      }
   image[x+1][y] += (6*erreur)/16 ; /* En evitant les debordements */
   image[x][y+1] += (6*erreur)/16 ;
   image[x+1][y+1] += (4*erreur)/16 ;
   }
```

Problèmes :

- Pour la programmation : on doit stocker les calculs intermédiaires dans un entier signé plus long (dépassements)
- On suppose les pixels indépendants (ne se recouvrent pas).
- Temps long de conversion.

3.7 Découpage ou "clipping"

Le but : afficher une figure en la découpant suivant un contour. Le contour est une surface fermée.

Première possibilité pour le découpage (pour des contours complexes). On trace la surface sur une bitmap. On trace le contour en se servant de la bitmap comme masque. On doit tester pixel par pixel et avoir une bitmap de libre.

Il est aussi possible de travailler en faisant des calculs d'intersections. Le découpage le plus courant est celui du à l'affichage dans une fenêtre rectangulaire.

Exemple récursif (d'école) pour un affichage sur une matrice de pixel. On découpe la droite en 2 (calculs entiers) récursivement jusqu'à tomber sur une droite évidente : complètement dedans, dehors ou de deux pixels de long.

L'algorithme classique de découpage d'un segment de droite par un rectangle

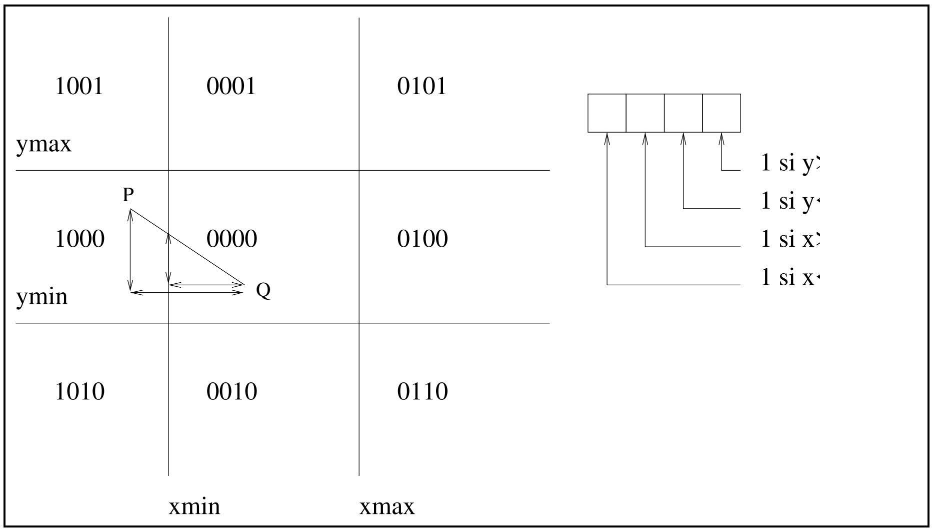


FIG. 13 – *Découpage de Evan et Sutherland*

Soit le segment allant de P à Q :
On calcule le code de P et Q : C_P,C_Q
Pour i=1 a 5
Si $C_P|C_Q = 0$ Tracer PQ et terminer.
Si $C_P\&C_Q \neq 0$ terminer : rien à voir.
Si le bit i de $(C_P \wedge C_Q) = 0$ continuer la boucle.
Calculer l'intersection avec l'axe i (xmin,xmax,ymin,ymax)
(IL Y A FORCEMENT UNE INTERSECTION) recalcuer le code du point deéplacé.
Recalculer le code du point déplacé.
fin pour

L'intersection avec la droite verticale d'équation $x = xmin$. On utilise les triangles semblables.

Le point d'intersection a pour coordonnées : $\begin{pmatrix} xmin \\ y \end{pmatrix}$ Nous avons :

$$\frac{x_Q - x_P}{y_P - y_Q} = \frac{x_Q - xmin}{y - y_Q}$$

Donc :

$$y = \frac{(x_Q - xmin)(y_P - y_Q)}{x_Q - x_P}$$

On peut améliorer en mémorisant la pente... La réalisation hard prévoit un composant par direction de découpage. Il y a aussi un découpage en profondeur.

4 Les bibliothèques graphiques

Ensemble de fonctions permettant de faire des sorties graphiques ainsi que des interactions.

Standard : Approuvé par un organisme de normalisation.
ISO International Standard Organisation,
AFNOR Association Française de NORmalisation,
DIN Deutch Institute Norma...,
ANSI American National Standards Institute
Sans parler des standards de fait.

Avantages :

- Portabilité.
- Indépendant des périphériques.
- Plus haut niveau d'abstraction. Pas le bête tracé de droites.
- Nombreuses fonctions écrites. Ne pas réinventer la roue.

Inconvénients :

- Moins performant en tout, moins adapté au matériel.
- Lourd. Trop de fonction, Trop compliqué.
- Pas forcément du bon niveau d'abstraction.

Quelques bibliothèques graphiques :

- GKS, 1984, 2D, ISO, ANSI, AFNOR, ... à l'origine Allemand.
- PHIGS, 1988, 3D, ISO.
- PHIGS+, 1991, 3D, ISO, Extension du rendu de PHIGS.
- X, 1988, pixel.
- PEX, 1992, 3D, Extension 3D de X.
- OPEN GL, 1992, 3D synthèse d'image, Celle de Silicon Graphique.
- PostScript, 198?, 2D, Mise en page : C'est un langage.

Une bibliothèque graphique minimale pour un écran RASTER avec LUT :

- set(x,y,index)
- get(x,y)
- setlut(index,r,v,b)
- getlut(index,r,v,b)
- inquire(lar,hau,nbc,ratio)
- mouse(x,y) position souris et non curseur.

4.1 Introduction à PHIGS

PHIGS : Programmer's Hierarchical Interactive Graphic System.
Création d'un modèle puis association de ce modèle à un dispositif d'affichage.

Workstation (Input, Output, Input+Output, Metafile IN et OUT) un poste de travail, une fenêtre, un traceur de courbes...

OutPut Primitives POLYLINE, POLYMARKER, POLYGON, PIXEL, ANNOTATION TEXT, GEOMETRIC TEXT.

Attributes Ce qui contrôle l'apparence des primitives. Couleur, fonte, ...

Structure modèle pour l'affichage, primitives graphiques, appel de structure, changement d'état courant...

View la position de l'observateur. (une transformation)

Window délimite ce qui est vu.

Viewport position de la vue sur l'écran normé.

Bundle table Table de groupement d'attributs par primitive. Chaque attribut peut être individuel ou groupé.

Modeling Transformation Transformation des primitives pour leur placement.

Workstation Transformation Passage de l'écran normé à l'écran physique.

Application Data Données liées à l'application stockées dans les structures.

Class Chaque primitive est associée avec un ensemble de classes. par exemple : tuyauterie, 1er etage, plastique.

Hightlight Mise en évidence d'une ou plusieurs primitives.

Associate Mettre un contenu (structure) dans une vue.

Structure traversal Parcour de toutes les structures de toutes les vues pour générer l'image.

Interaction mode Event, Sample, Request.

Input device Choice, Locator, Pick, Stroke, String, Valuator.

Inquire Récupération de différentes infos.

State Etat de PHIGS, Workstation, Structure, Archive.

Structure d'un simple programme d'affichage :

- Ouverture de PHIGS.
- Ouverture d'une workstation (ou plusieurs).
- Ouverture structure.
- Primitives graphiques, changements attributs.
- Fermeture structure.
- Association sur l'écran.
- Affichage.
- Fermeture des workstations.
- Fermeture de PHIGS.

4.1.1 Les transformations

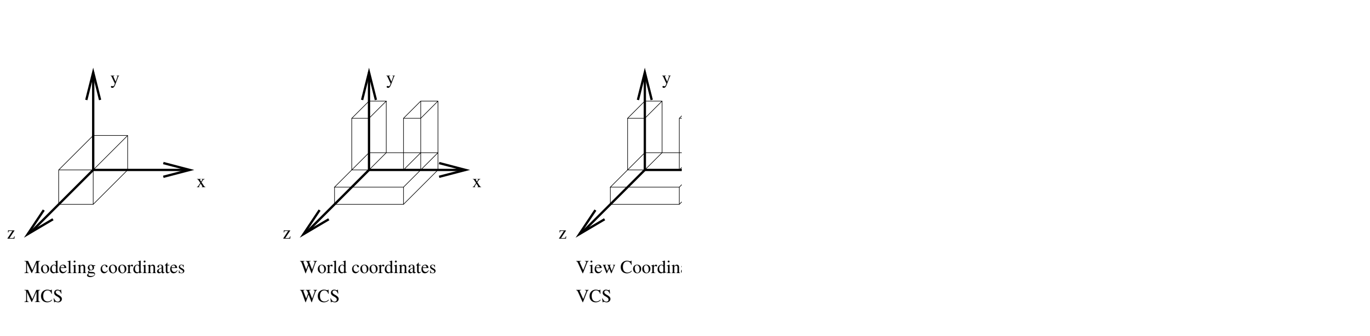


Fig. 14 – *Pipe de visualisation PHIGS*

On décrit les objets dans un système de coordonnées propre (MCS).
On place les objets dans le monde 3D (WCS)
On choisit un point de vue (VCS)
On choisit une fenêtre et une clôture (NPC)
On choisit la position sur l'écran (DC)
Pour passer d'un système de coordonnées à un autre, on utilise une transformation.

- 2D → 2D avec une matrice 2x2
- 3D → 3D avec une matrice 3x3 si "normale"
- 3D → 3D avec une matrice 4x4 si coordonnées homogènes (points à l'infini et projection perspective).

Création de matrices par des fonctions utilitaires :

- Rotate X,Y,Z
- Scale XYZ
- Translate XYZ
- Set View plane normal
- Set View reference point
- Compose Matrix (produit de matrices)
- Define Coordinate system (normal + point + up → matrice)

Modification des transformations courantes :

- Set global transformation (rarement utile: met à la poubelle)
- Set local transformation (PRECONCATENATE,POSTCONCATENATE)
- Set view mapping (window, viewport, ...)
- Set workstation transformation.

4.1.2 Transformations de modélisation

Soit G la transformation globale de modélisation et L la transformation locale de modélisation.
Au départ (structure associée à la vue) $G = Id$ et $L = Id$. Les primitives subissent la transformation $G * L$.
L'opération SET GLOBAL TRANSFORMATION modifie la valeur de G par un remplacement ou un produit. Cette opération est rarement employée. En effet on préfère conserver dans G la transformation qui était courante au moment de l'exécution de la structure.
L'opération SET LOCAL TRANSFORMATION fait la même chose sur L. Celle-ci est par contre plus utilisée.
L'opération EXECUTE STRUCTURE sauvegarde les valeurs de G et L (ainsi que les attributs courants) et fait $G = G * L$ ainsi que $L = Id$.
Voici un exemple simple, on appel `carre` la structure contenant un carré de coins opposés (0,0) et (1,1). On travaille en 2D.

```
OPEN STRUCTURE( quatre_carres )
```

```
SET LOCAL TRANSFORMATION : homothetie de 1/3 (postconcatenate)
SET LOCAL TRANSFORMATION : rotation de PI/2 (postconcatenate)
EXECUTE STRUCTURE( carre )
SET LOCAL TRANSFORMATION : une translation de 2 a droite (postconcatenate)
EXECUTE STRUCTURE( carre )
SET LOCAL TRANSFORMATION : une translation de 2 en haut (postconcatenate)
EXECUTE STRUCTURE( carre )
SET LOCAL TRANSFORMATION : rotation de PI/2 (postconcatenate)
EXECUTE STRUCTURE( carre )
CLOSE STRUCTURE( quatre_carres )
```

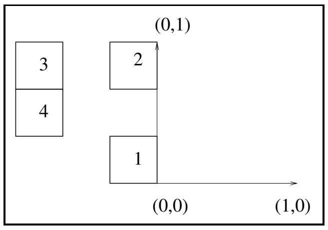


FIG. 15 – *Composition de transformations*

Si on fait la structure `carre4x4` identique à la précédente excepté un appel à `carre4` au lieu de `carré` on obtient :

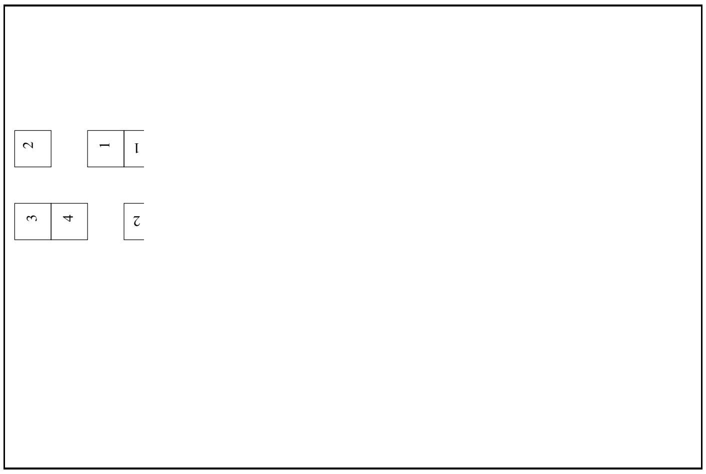


FIG. 16 – *Plusieurs niveaux de hiérarchie de structures*

4.1.3 Les fonctions de sorties de PHIGS

Elles sont minimales mais complètes, pour chacune des fonctions, il y a une version 2D et 3D. La version 2D appelle la 3D avec une composante Z nulle.

ANNOTATION TEXT Texte n'étant pas affecté par les diverses transformations géométriques sauf pour son point d'accrochage.

POLYGON Une facette (points coplanaires).

POLYLINE Ligne brisée.

POLYMARKER On donne une liste de points, il y aura une marque sur chaque point.

TEXT Texte géométrique subissant toutes les transformations.

gdp PIXEL Affichage d'une matrice de pixel (transformations possibles).

gdp CIRCLE Affichage d'un cercle.

gdp ARC Affichage d'un arc de cercle.

gdp DISJOINT POLYLINE Liste de POLYLINE.

Les primitives GDP (Generalized Drawing Primitives) ne sont pas standard. Avant de les utiliser, il faut demander à PHIGS si elles existent sur le poste de travail.

Ces différentes primitives n'affichent rien. Lors de leur appel, elles sont insérées dans la structure ouverte. L'affichage est fait lors du retraçage de l'écran.

Une fois la primitive insérée, on peut réutiliser les structures de données, elle sont recopiées par PHIGS.

Il faut donc stocker les primitives dans les structures.

OPEN STRUCTURE On lui donne le numéro de la structure à ouvrir. Il la crée si elle n'existe pas, sinon on est positionné à la fin.

CLOSE STRUCTURE Ferme la structure courante.

DELETE STRUCTURE Destruction de la structure considérée.

EXECUTE STRUCTURE Lors du "structure traversal" la structure indiquée est exécutée. On forme donc un graphe de structure (sans boucle). L'exécution d'une structure ne modifie pas l'état courant. Quelque soit les modifications apportées par la structure appel'ee (couleur, transformation, ...) ces modifications disparaissent après l'exécution.

Edition Copy Structure, Delete Element between labels, Delete Element, Delete Element Range, Set Element Pointer, Set Element Pointer at Label, Set Element Pointer at Pick Identifier, Empty Structure, Insert Label, Offset Element Pointer.

Destructions Delete all Structures, Delete Structure Network Conditionnaly, Delete Structure Network?

Les attributs des primitives sont les attributs courants. Il existe différentes fonctions qui modifient les attributs courants.

4.1.4 Les attributs des primitives

Toutes les primitives de sortie ont les attributs suivants :

- Un index de table de groupage (différent suivant le type de primitive).
- Un pick identifier.
- Une liste de classes.
- Un View index qui détermine sur quelles vues elle s'affiche.

L'utilisation des attributs est basée sur un état courant. Les attributs sont soit individuels soit groupés (dans une BUNDLE TABLE). Pour chaque type de primitive on a une table qui regroupe un certain nombre d'attributs de la primitive, il y a une table par STATION. Le choix final de l'attribut est fait selon l'algorithme suivant.

```
Soit P le type de primitive
Soit A le type de l'attribut
SI l'attribut A de P est de type INDIVIDUEL
              Tracer P avec la valeur courante de A (pour P)
SINON
              Soit W la station ou l'on doit tracer
              Soit I l'index courant de table pour P
              Soit T la table de P dans W
              Tracer P avec la valeur de A prise en position I de T
```

Chacune des primitives a un index courant de table de groupage. La table de groupage contient les attributs précédés d'une *. Pour chacune des primitives on indique si l'attribut est individuel ou à rechercher dans la table. Attributs pour les primitives :

Polyline *Line type, *Linewidth scale factor, *Polyline color index, Polyline end type.

Polymarker *Polymark type, *Polymark size, *Polymark color.

Polygon *Edge flag, *Edge line type, *Edge scale factor, *Edge color index, *Interior style, *Interior color index, *Interior style index, Pattern size, Pattern reference point, Pattern reference vectors.

Texte Text character set, *Text font, *Text color index, Text path, Char height, *Char spacing, *Char expansion factor, Char up vector, Text alignment, *Text precision.

Annotation text Idem.

Valeurs possibles des attributs :

Line type Solid, Dashed, Dotted, Dash-dotted, Double-dotted, long-dashed, Dash-double-dot.

End type Flat, Round, Square (droites épaisses à angle droit).

Marker type Dot, Plus sign, Asterisk, Circle, Diagonal cross.

Interior style Hollow, Solid, Pattern, Hatch.

Text Path Up, Down, Right, Left.

Text precision String, Character, Stroke.

Text alignment Top, Half, Base, Left, Center, Right.

4.1.5 Exemple simple d'affichage

Les noms de fonction commencent par “p” et ceux des `define` par “P”. Il est possible de programmer avec des noms longs.

Il est plus rapide de faire une recherche dans le fichier “/usr/include/phigs.h” pour les structures et les `define` et dans le fichier “/usr/include/phigsproto.h” pour les prototypes que dans la documentation.

```
#include <phigs.h>                      /* On inclu les definitions */
main()
{
int i;
static Ppoint3 pt={0.1,0.66,0.0};      /* Un point */
static Ppoint_list3 pt_l={1,&pt};      /* Une table de points */
int type;
int stat,vie;

printf("Workstation type ?:");          /* On recupere le type de station */
scanf("%d",&type);
popph(PDEF_ERR_FILE,PDEF_MEM_SIZE);    /* On ouvre PHIGS */
popwk(1,"file.dat",type);              /* On ouvre la station */

popst(1);                               /* On ouvre une structure */
for (i=1;i<6;i++)
  {
    psmk(i);                            /* Choisit la forme du marqueur */
    pspmci(i);                          /* Choisit la couleur du marqueur */
    ppm3(&pt_l);                        /* Creation de la marque */
    pt.x +=0.2;                         /* Depalce le point de la marque */
  }

pt.x = 0.1;                             /* Donne un autre point */
pt.y = 0.33;
for (i=1;i<6;i++)
  {
    psmk(-i);                           /* Test les marques negatives */
    pspmci(i);
    ppm3(&pt_l);
    pt.x +=0.2;
  }
pclst();                                /* Ferme la structure */

ppost(1,1,1.);                          /* Met la strucuture 1 sur la station 1 */
prst(1,PFLAG_ALWAYS);                   /* Retrace les structures */
prqlc(1,1,&stat,&vie,&pt);              /* Fait une requete de point */
pclwk(1);                               /* Ferme la station */
pclph();                                /* Ferme PHIGS */
}
```

4.1.6 Fonctions d'entrées de PHIGS

Les types de dispositifs d'entrées :

- Locator. Des coordonnées WC, et un index de vue.
- Stroke. Une séquence de coordonnées WC.
- Choice. Numéro de la touche appuyée.
- Valuator. Une valeur réelle.
- Pick. Un chemin menant à la primitive désignée.
- String. Une chaîne de caractères.

Les modes d'acquisition :

- Request
- Sample
- Event

Il y a donc en tout 3*6=18 fonctions d'acquisition.
Les fonctions de préparation à la saisie :

Initialise Valeurs initiales, valeurs possibles, type de prompt, ...

Mode (Req,Samp,Event) et si oui ou non echo.

Il y a donc en tout 2*6=12 fonctions qui préparent à l'acquisition, sans compter quelques particulières.
Il peut y avoir plusieurs dispositifs d'entrée du même type sur une workstation.
On peut se mettre en attente d'un événement quelconque.
Si l'on veut attendre plusieurs types d'entrée simultanément (un Pick et un Locator) On ne peut utiliser Request, Sample est complexe par contre Event est parfait. L'exemple typique : Saisie d'une ligne brisée et accès à un menu possible simultanément.
Dans le cas de la désignation, on peut filtrer les primitives que l'on peut désigner (On n'autorise à désigner que de la plomberie...)

4.1.7 Les filtres et les classes

Une `classe` est un entier. Chaque primitive “appartient” à un certain nombre de classes. On dispose des fonctions ADD NAMES TO SET et REMOVES NAMES FROM SET pour changer la liste courante des `classes`.

Chaque `workstation` contient des filtres d'inclusion ou d'exclusion.

- 2 filtres d'`highlightning` qui indiquent quelles sont les primitives à mettre en évidence. Différents moyens sont possibles : augmenter la luminosité, faire clignoter ou bien épaissir les traits. Cela dépend de la station employée.
- 2 filtres d'`invisibility` qui indiquent quelles sont les primitives à ne pas afficher.
- 2 filtres de `detectabiblity` pour indiquer les primitives que l'on peut désigner par un `pick`.

L'intersection entre le filtre d'inclusion et d'exclusion doit être nulle.

Une primitive est choisie si une de ses classes appartient aux filtre d'inclusion et qu'aucune n'appartienne au filtre d'exclusion.

4.1.8 Les inquires

Ce sont toutes des fonctions permettant de récupérer des informations sur les structures de données PHIGS.

- L'état courant de : PHIGS, Structures, Archives, Workstation.
- Les limites de : PHIGS, ...
- Des chemins dans les structures.

4.1.9 Mode d'affichage

La mise à jour de l'écran peut être faite suivant différents modes :

ASAP As At Possible.

BNIG Before Next Interaction Global.

BNIL Before Next Interaction Local.

ASTI Quand il veut.

WAIT Ne pas afficher.

La mise à jour est de haut niveau. En effet le contenu des vues est retracé complètement. Si une primitive disparaît d'une structure, elle disparaîtra de l'écran sans que le programmeur n'ait à se poser de question. LA contrepartie est bien sur la perte de performance puisque tout est retracé.

4.1.10 Archives

On peut sauvegarder et récupérer des structures dans des fichiers. Le problème de ces commandes est le conflit de noms de structures.

4.1.11 Liste des fonctions PHIGS

/* CONTROL FUNCTION */

popen_phigs(`char` *err_file,`unsigned int` mem_units)
pclose_phigs(void)
popen_ws(`Pint` ws_id,`char` *conn_id,`Pint` ws_type)
pclose_ws(`Pint` ws_id)
predraw_all_structs(`Pint` ws_id,`Pctrl_flag` control_flag)
pupd_ws(`Pint` ws_id,`Pregen_flag` regen_flag)
pset_disp_upd_st(`Pint` ws_id,`Pdefer_mode` def_mode,`Pmod_mode` mod_mode)
pmessage(`Pint` ws_id,`char` *msg)

/* GRAPHIC PRIMITIVES */

ppolyline3(`Ppoint_list3` *point_list)
ppolyline(`Ppoint_list` *point_list)
ppolymarker3(`Ppoint_list3` *point_list)
ppolymarker(`Ppoint_list` *point_list)
ptext3(`Ppoint3` *text_pos,`Pvec3` dir[2],`char` *char_string)
ptext(`Ppoint` *text_pos,`char` *char_string)
panno_text_rel3(`Ppoint3` *ref_pt,`Pvec3` *offset,`char` *char_string)
panno_text_rel(`Ppoint` *ref_pt,`Pvec` *offset,`char` *char_string)
pfill_area3(`Ppoint_list3` *point_list)
pfill_area(`Ppoint_list` *point_list)
pfill_area_set3(`Ppoint_list_list3` *point_list_list)
pfill_area_set(`Ppoint_list_list` *point_list_list)
pcell_array3(`Pparal` *paral,`Ppat_rep` *colr_array)
pcell_array(`Prect` *rect,`Ppat_rep` *colr_array)

/* GRAPHIC ATTRIBUTES */

pset_line_ind(`Pint` index)
pset_marker_ind(`Pint` index)
pset_text_ind(`Pint` index)
pset_int_ind(`Pint` index)
pset_edge_ind(`Pint` index)
pset_linetype(`Pint` linetype)
pset_linewidth(`Pfloat` scale)
pset_line_colr_ind(`Pint` index)
pset_marker_type(`Pint` markertype)
pset_marker_size(`Pfloat` scale)
pset_marker_colr_ind(`Pint` index)
pset_text_font(`Pint` font)
pset_text_prec(`Ptext_prec` precision)
pset_char_expan(`Pfloat` exp_factor)
pset_char_space(`Pfloat` spacing)
pset_text_colr_ind(`Pint` index)
pset_char_ht(`Pfloat` height)
pset_char_up_vec(`Pvec` *up_vect)
pset_text_path(`Ptext_path` path)
pset_text_align(`Ptext_align` *align)
pset_anno_char_ht(`Pfloat` height)
pset_anno_char_up_vec(`Pvec` *up_vect)
pset_anno_path(`Ptext_path` path)
pset_anno_align(`Ptext_align` *align)
pset_anno_style(`Pint` style)
pset_int_style(`Pint_style` style)
pset_int_style_ind(`Pint` index)
pset_int_colr_ind(`Pint` index)
pset_edge_flag(`Pedge_flag` edge_flag)
pset_edgetype(`Pint` edgetype)
pset_edgewidth(`Pfloat` scale)
pset_edge_colr_ind(`Pint` index)
pset_pat_size(`Pfloat_size` *size)
pset_pat_ref_point_vecs(`Ppoint3` *pat_ref_point,`Pvec3` pat_ref_vec[2])
pset_pat_ref_point(`Ppoint` *pat_ref_point)

```
padd_names_set(Pint_list *names)
premove_names_set(Pint_list *names)
pset_indiv_asf(Paspect asf_id,Pasf asf_source)
pset_line_rep(Pint ws_id,Pint index,Pline_bundle *rep)
pset_marker_rep(Pint ws_id,Pint index,Pmarker_bundle *rep)
pset_text_rep(Pint ws_id,Pint index,Ptext_bundle *rep)
pset_int_rep(Pint ws_id,Pint index,Pint_bundle *rep)
pset_edge_rep(Pint ws_id,Pint index,Pedge_bundle *rep)
pset_pat_rep(Pint ws_id,Pint index,Ppat_rep *rep)
pset_colr_rep(Pint ws_id,Pint index,Pcolr_rep *colr_rep)
pset_highl_filter(Pint ws_id,Pfilter *filter)
pset_invis_filter(Pint ws_id,Pfilter *filter)
pset_colr_model(Pint ws_id,Pint model)
pset_hlhsr_id(Pint id)
pset_hlhsr_mode(Pint ws_id,Pint mode)

    /* TRANSFORMATION FUNCTIONS */
pset_local_tran3(Pmatrix3 xform,Pcompose_type compose_type)
pset_local_tran(Pmatrix xform,Pcompose_type compose_type)
pset_global_tran3(Pmatrix3 xform)
pset_global_tran(Pmatrix xform)
pset_model_clip_vol3(Pint op,Phalf_space_list3 *hf_spaces)
pset_model_clip_vol(Pint op,Phalf_space_list *half_spaces)
pset_model_clip_ind(Pclip_ind ind)
prestore_model_clip_vol(void)
pset_view_ind(Pint index)
pset_view_rep3(Pint ws_id,Pint index,Pview_rep3 *rep)
pset_view_rep(Pint ws_id,Pint index,Pview_rep *rep)
pset_view_tran_in_pri(Pint ws_id,Pint index,Pint ref_index,Prel_pri priority)
pset_ws_win3(Pint ws_id,Plimit3 *window)
pset_ws_win(Pint ws_id,Plimit *window)
pset_ws_vp3(Pint ws_id,Plimit3 *viewport)
pset_ws_vp(Pint ws_id,Plimit *viewport)
ptranslate3(Pvec3 *trans,Pint *err_ind,Pmatrix3 mat_trans)
ptranslate(Pvec *trans,Pint *err_ind,Pmatrix mat_trans)
pscale3(Pvec3 *scale,Pint *err_ind,Pmatrix3 mat_scale)
pscale(Pvec *scale,Pint *err_ind,Pmatrix mat_scale)
protate_x(Pfloat ang,Pint *err_ind,Pmatrix3 mat_rotx)
protate_y(Pfloat ang,Pint *err_ind,Pmatrix3 mat_roty)
protate_z(Pfloat ang,Pint *err_ind,Pmatrix3 mat_rotz)
protate(Pfloat ang,Pint *err_ind,Pmatrix mat_rotx)
pcompose_matrix3(Pmatrix3 matA,Pmatrix3 matB,Pint *err_ind,Pmatrix3 matR)
pcompose_matrix(Pmatrix matA,Pmatrix matB,Pint *err_ind,Pmatrix matR)
ptran_point3(Ppoint3 *point3D,Pmatrix3 mat_trans,Pint *err_ind,Ppoint3 *point_TR)
ptran_point(Ppoint *pt,Pmatrix mat_trans,Pint *err_ind,Ppoint *point_TR)
pbuild_tran_matrix3(Ppoint3 *pt,Pvec3 *shift_vec,Pfloat x_angle,
.                          Pfloat y_angle,Pfloat z_angle,Pvec3 *scale_pt,Pint *err_ind,Pmatrix3 trans_mat)
pbuild_tran_matrix(Ppoint *pt,Pvec *shift,Pfloat angle,Pvec *scale,
.                                                            Pint *err_ind,Pmatrix result)
pcompose_tran_matrix3(Pmatrix3 matrix3,Ppoint3 *point3D,Pvec3 *shift_vec,
.                                               Pfloat x_angle,Pfloat y_angle,Pfloat z_angle,
.                                            Pvec3 *scale_pt,Pint *err_ind,Pmatrix3 trans_mat)
pcompose_tran_matrix(Pmatrix matrix,Ppoint *point2D,Pvec *shift_vec,
.                                      Pfloat angle,Pvec *scale_pt,Pint *err_ind,Pmatrix trans_mat)
peval_view_ori_matrix3(Ppoint3 *view_ref,Pvec3 *view_norm,Pvec3 *view_up,
.                                                           Pint *err_ind,Pmatrix3 result)
peval_view_ori_matrix(Ppoint *view_ref,Pvec *view_up,Pint *err_ind,Pmatrix result)
peval_view_map_matrix3(Pview_map3 *mapping,Pint *error_ind,Pmatrix3 result)
peval_view_map_matrix(Pview_map *mapping,Pint *error_ind,Pmatrix result)

    /* STRUCTURE ELEMENT DEFINITION */
popen_struct(Pint struct_id)
pclose_struct(void)
pexec_struct(Pint struct_id)
```

```
plabel(Pint label_id)
pappl_data(Pdata *data)
pgse(Pint id,Pgse_data *data)
pset_edit_mode(Pedit_mode mode)
pcopy_all_elems_struct(Pint struct_id)
pset_elem_ptr(Pint el_value)
poffset_elem_ptr(Pint ep_offset)
pset_elem_ptr_label(Pint label_id)
pdel_elem(void)
pdel_elem_range(Pint elem1_pt,Pint elem2_pt)
pdel_elems_labels(Pint label1_id,Pint label2_id)
pempty_struct(Pint struct_id)

   /* STRUCTURE HANDLING */
pdel_struct(Pint struct_id)
pdel_struct_net(Pint struct_id,Pref_flag ref_flag)
pdel_all_structs(void)
pchange_struct_id(Pint orig_struct_id,Pint result_struct_id)
pchange_struct_refs(Pint orig_struct_id,Pint result_struct_id)
pchange_struct_id_refs(Pint oldsid,Pint newsid)

   /* STRUCTURE DISPLAY */
ppost_struct(Pint ws_id,Pint struct_id,Pfloat prior)
punpost_struct(Pint ws_ident,Pint struct_id)
punpost_all_structs(Pint ws_ident)

   /* ARCHIVING FUNCTION */
popen_ar_file(Pint Pfile_id,char *Pfile_name)
pclose_ar_file(Pint Pfile_id)
par_structs(Pint archive_id,Pint_list *struct_ids)
par_struct_nets(Pint archive_id,Pint_list *struct_ids)
par_all_structs(Pint archive_id)
pset_conf_res(Pconf_res archival_res,Pconf_res retrieval_res)
pret_struct_ids(Pint archive_id,Pint max_ids,Pint start,Pint_list *ids,Pint *actual_ids)
pret_structs(Pint archive_id,Pint_list *struct_ids)
pret_struct_nets(Pint archive_id,Pint_list *struct_ids)
pret_all_structs(Pint archive_id)
pret_paths_ances(Pint ar_id,Pint struct_id,Ppath_order order,Pint depth,
.                Pint size,char *buffer,Pelem_ref_list_list *paths,Pint *total_size,Pint *mem_status)
pret_paths_descs(Pint ar_id,Pint struct_id,Ppath_order order,Pint depth,
.                Pint size,char *buffer,Pelem_ref_list_list *paths,Pint *total_size,Pint *mem_status)
pdel_structs_ar(Pint archive_id,Pint_list *struct_ids)
pdel_struct_nets_ar(Pint archive_id,Pint_list *struct_ids)
pdel_all_structs_ar(Pint archive_id)

   /* INPUT FUNCTIONS */
pset_pick_id(Pint pick_id)
pset_pick_filter(Pint ws_id,Pint pick_dev,Pfilter *filter)
pinit_loc3(Pint ws_id,Pint loc_dev,Pint init_view_index,
.                        Ppoint3 *init_loc_pos,Pint pet,Plimit3 *echo_volume,Ploc_data3 *loc_data_rec)
pinit_loc(Pint ws_id,Pint loc_dev,Pint init_view_index,
.                          Ppoint *init_loc_pos,Pint pet,Plimit *echo_area,Ploc_data *loc_data_rec)
pinit_stroke3(Pint ws_id,Pint stroke_dev,Pint init_view_index,
.                                 Ppoint_list3 *init_stroke,Pint pet,Plimit3 *echo_volume,
.                                                           Pstroke_data3 *stroke_data_rec)
pinit_stroke(Pint ws_id,Pint stroke_dev,Pint init_view_index,
.                Ppoint_list *init_stroke,Pint pet,Plimit *echo_volume,Pstroke_data *stroke_data_rec)
pinit_val3(Pint ws_id,Pint val_dev,Pfloat init_value,Pint pet,
.                                                   Plimit3 *echo_volume,Pval_data3 *val_data)
pinit_val(Pint ws_id,Pint val_dev,Pfloat init_value,Pint pet,
.                                                      Plimit *echo_area,Pval_data *val_data)
pinit_choice3(Pint ws_id,Pint choice_dev,Pin_status init_status,
.                      Pint init_choice,Pint pet,Plimit3 *echo_volume,Pchoice_data3 *choice_data_rec)
pinit_choice(Pint ws_id,Pint choice_dev,Pin_status init_status,
```

. Pint init_choice,Pint pet,Plimit *echo_area,Pchoice_data *choice_data_rec)
pinit_pick3(Pint ws_id,Pint pick_dev,Pin_status init_status,Ppick_path *init_pick,
. Pint pet,Plimit3 *echo_volume,Ppick_data3 *pick_data_rec,Ppath_order order)
pinit_pick(Pint ws_id,Pint pick_dev,Pin_status init_status,Ppick_path *init_pick,
. Pint pet,Plimit *echo_area,Ppick_data *pick_data_rec,Ppath_order order)
pinit_string3(Pint ws_id,Pint string_dev,char *init_string,Pint pet,
. Plimit3 *echo_vol,Pstring_data3 *string_data)
pinit_string(Pint ws_id,Pint string_dev,char *init_string,Pint pet,
. Plimit *echo_area,Pstring_data *string_data)
pset_loc_mode(Pint ws_id,Pint loc_dev,Pop_mode mode,Pecho_switch echo_switch)
pset_stroke_mode(Pint ws_id,Pint stroke_dev,Pop_mode mode,Pecho_switch echo_switch)
pset_val_mode(Pint ws_id,Pint val_dev,Pop_mode mode,Pecho_switch echo_switch)
pset_choice_mode(Pint ws_id,Pint choice_dev,Pop_mode mode,Pecho_switch echo_switch)
pset_pick_mode(Pint ws_id,Pint pick_dev,Pop_mode mode,Pecho_switch echo_switch)
pset_string_mode(Pint ws_id,Pint string_dev,Pop_mode mode,Pecho_switch echo_switch)
preq_stroke3(Pint ws_id,Pint stroke_dev,Pin_status *status,
. Pint *view_index,Ppoint_list3 *stroke)
preq_stroke(Pint ws_id,Pint stroke_dev,Pin_status *status,
. Pint *view_index,Ppoint_list *stroke)
preq_val(Pint ws_id,Pint val_dev,Pin_status *stat,Pfloat *value)
preq_choice(Pint ws_id,Pint choice_dev,Pin_status *status,Pint *choice)
preq_pick(Pint ws_id,Pint pick_dev,Pint depth,Pin_status *status,Ppick_path *pick)
preq_string(Pint ws_id,Pint string_dev,Pin_status *status,char *string)
preq_loc3(Pint ws_id,Pint loc_dev,Pin_status *status,
. Pint *view_index,Ppoint3 *loc_pos)
preq_loc(Pint ws_id,Pint loc_dev,Pin_status *status,Pint *view_index,Ppoint *loc_pos)
psample_loc3(Pint ws_id,Pint loc_dev,Pint *view_index,Ppoint3 *loc_pos)
psample_loc(Pint ws_id,Pint loc_dev,Pint *view_index,Ppoint *loc_pos)
psample_stroke3(Pint ws_id,Pint stroke_dev,Pint *view_index,Ppoint_list3 *stroke)
psample_stroke(Pint ws_id,Pint stroke_dev,Pint *view_index,Ppoint_list *stroke)
psample_val(Pint ws_id,Pint val_dev,Pfloat *value)
psample_choice(Pint ws_id,Pint choice_dev,Pin_status *status,Pint *choice)
psample_pick(Pint ws_id,Pint pick_dev,Pint depth,
. Pin_status *status,Ppick_path *pick)
psample_string(Pint ws_id,Pint string_dev,char *string)
pawait_event(Pfloat timeout,Pint *ws_id,Pin_class *class,Pint *input_dev)
pflush_events(Pint ws_id,Pin_class class,Pint input_dev)
pget_loc3(Pint *view_index,Ppoint3 *loc_pos)
pget_loc(Pint *view_index,Ppoint *loc_pos)
pget_stroke(Pint *view_index,Ppoint_list *stroke)
pget_stroke3(Pint *view_index,Ppoint_list3 *stroke)
pget_val(Pfloat *value)
pget_choice(Pin_status *status,Pint *choice)
pget_pick(Pint depth,Pin_status *status,Ppick_path *pick)
pget_string(char *string)

4.2 Introduction à X

X ou X11 ou X window system. Le 11 vient du fait que l'on est en version 11. La release est la sixième

X est un système de multifenêtrage réseau, il a été par le MIT, le projet a été commencé en 1985 La diffusion à grande échelle en 1988. X est diffusé gratuitement avec les sources, c'est maintenant un standard de fait.

Mot-clefs concernant le serveur :

- Architecture client/serveur, le serveur gère la ressource écran(s) et périphériques d'entrées.
- Le serveur peut communiquer avec plusieurs clients sur des machines différentes.
- L'application fonctionnera de la même manière quelque soit l'écran, qu'il soit local ou non.
- Le serveur permet toutes les manipulations bas niveaux sur l'écran. Il travaille au niveau pixel.
- Le serveur peut fonctionner sur tous les types d'écran matriciels.
- Le serveur gère une *hiérarchie de fenêtres.*
- Le *window manager* est un client comme un autre, il est indépendant du serveur.
- Le serveur gère des ressources, ces ressources sont partageables entre les clients :
  - Les fenêtres (Window).
  - Les tables de couleurs (Colormap).
  - Les contextes graphiques (GC).
  - Les matrices de bit ou de pixel (Bitmap,Pixmap).
  - Les fontes (Font).
  - Les curseurs (Cursor).
  - Des tampons de données (Cutsel).
  - Une base de données de configuration d'applications (xrdb).
  - Les symboles clavier (KeySym).
  - Type d'affichage (Visual).
- Le serveur offre des mécanismes de base pour échanger des données entre des clients.
- Le serveur peut contenir des extensions :
  - PEX pour supporter PHIGS.
  - VEX pour l'affichage d'une vidéo extérieur.
  - XIE pour le traitement d'image.
  - ...
- Le serveur peut directement être accédé via la bibliothèque `Xlib.a` celle-ci offre les fonctions élémentaires permettant de tous faire, c'est l'*assembleur X.*
- Les intrinsics (`libXt.a`) sont des fonctions permettant une approche objet de la conception d'une application.
- On appel *Widget* les objets que permet de créer `libXt.a`.
- MOTIF est un gestionnaire de fenêtres et un ensemble de *Widget* MOTIF et n'est pas gratuit.
- Athena Widget est une bibliothèque de *Widget* très employée elle est puissante et gratuite.

4.2.1 Commandes X

Si le serveur X ne fonctionne pas, il faut le lancer, la commande standard est `xinit` ou `startx` sur HP c'est `x11start` ou `vuesession` et sur SUN `openwin`.

Le nom d'un serveur X est de la forme : `nom-machine:numéro-serveur.numéro-écran` Dans le cas général les deux numéros sont nuls. Pour choisir un serveur durant toute une session, il suffit de mettre le nom du serveur dans la variable DISPLAY du shell.

```
DISPLAY=dessiipa:0
export DISPLAY
```

Cette manipulation est inutile si vous vous ête directement connecté sous X. Si vous voulez lancer un seul client sur un autre serveur il suffit d'ajouter `-display dessiipa:0` dans les arguments. Si cela n'a pas été fait il faut ajouter a votre `PATH` les directory `/usr/bin/X11` ou `/vol/X11R6/bin`.

Un des premier clients à être lancé sur un serveur est le window manager cela peut-être : `mwm twm vuewm uwm dxwm ...` Les clients les plus classiques (souvent lancé par menu) sont :

- `resize` qui écrit un programme donnant la taille de la fenêtre.
- `xhost` pour autoriser ou non certaine machine a se connecter.
- `xcalc` la calculatrice.
- `xclock` l'horloge.
- `xload` charge de la machine.
- `xbiff` indique si vous avez du courrier ou pas.
- `xfontsel` pour choisir interactivement une fonte.
- `xfd` affiche le contenu d'une fonte.
- `xmag` pour zoomer une partie de l'écran.
- `xman` pour le manuel interactif.
- `xrdb` pour charger un fichier dans la base de donnée du serveur.
- `xset` pour configurer le serveur.
- `xmodmap` change le clavier.
- `xsetroot` choix du fond d'écran.
- `xterm` pour lancer un émulateur de terminal VT100.
- `xwd` sauvegarde d'une fenêtre dans un fichier.
- `xwud` affiche le contenu d'un fichier `.xwd` dans une fenêtre.
- `bitmap` petit éditeur de pixel pour faire des icones.
- `editres` éditeur permettant de configurer une application `(Xaw) interactivement.`

Ressource :

```
*Font : "-*-times-bold-r-*-*-16-*-*-*-*-*-*-*"
HPterm*Font : "-*-*-*-r-*-*-12-*-*-*-*-*-*-*"
*MenuButton.Font : "-*-*-*-r-*-*-16-*-*-*-*-*-*-*"
*quit.foreground : Red
```

4.2.2 A l'intérieur de X

Quand on ouvre une fenêtre on peut utiliser les visuals suivants : Static color, Pseudo color, Static gray, Gray scale, True color, direct color.

Les événements : KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify, LeaveNotify, FocusIn, FocusOut, Expose, UnmapNotify, MapNotify, ConfigureNotify, ResizeRequest, Visibility, ... Si l'événement provient d'un périphérique il est envoyé à la plus petite fenêtre contenant le curseur, si celle-ci n'en veut pas, le serveur essaye avec le père et ceci jusqu'a la racine. Ceci est faux en cas de “Focus window” et de Grab sur un périphérique.

Les attributs graphiques (GC : Graphic Context) sont très nombreux : Fonction, plane_mask, foreground, background, line_width, line_style, cap_style, fill_style, tile, stipple, subwindow_mode, font, ...

Les attributs de fenêtres : background_pixmap, background_pixel, border_pixmap, border_pixel, win_gravity, backing_store, save_under, event_mask, do_not_propagate_mask, colormap, cursor, ...

Les fonctions d'affichages : Manipulations de rectangles de pixel, affichage de points, lignes et arcs de cercles ainsi que de caractères.

Communiquer avec le window manager : nom de la fenêtre, taille, icone, colormap.

Définir des `Atom` les Atoms sont l'expression sous forme de chiffre d'une chaîne de caracteres qui représente un nom. Le serveur contient la correspondance entre le numéro de l'atome et la chaîne de caractères. De nombreux atomes sont prédéfinis.

Définir des `Properties`, chaque property est une donnée typée associée à une fenêtre par exemple : la propriété WM_NAME est de type STRING elle permet au window manager de trouver le titre de la fenêtre pour l'afficher.

Une sélection est un property qui n'appartient qu'a une fenêtre, cette property appartient à la fenêtre qui détient la selection. L'ancien possesseur est prévenu par un événement si on lui arrache sa sélection. Exemple : le coupé coller de texte.

Décodage clavier, chaque touche du clavier possède un numéro appellé `KeyCode` ce numéro dépend du clavier. A chaque Keycode correspond une liste de `Keysym` qui indiquent le code à retourner en fonction du `modifier` (shift, ctrl, ...). La liste de Keysim associée à chaque Keycode peut être modifié.

Il est possible d'envoyer des événements directement à une client ou bien lui voler des événements, ceci est notamment fait par le window manager lors de l'ouverture de la fenêtre ou bien lors du changement de sa taille.

Des fonctions utilitaires travaillant sur des régions, elle permettent de faire certain calcul localement pour éviter des accès au serveur.

331 fonctions, 114 structures et 349 constantes définie par `define`. Accessibles par `Xlib.a`

Une première couche au-dessus de `Xlib.a` permet de créer de `Widget` se sont les intrinsics `libXt.a`. Cette couche permet de créer des applications plus simplement, elles sont d'autre part dans le même “moule”. Cette couche n'est pratiquement jamais utilisée par le programmeur, elle programmée sous forme d'objets de classe et de méthode. Elle est très maléable et comporte de nombreuse possibilités.

La couche au-dessus `libXaw.a` comporte des instances de Widget. Elle est très facilement utilisable, on définit une hiérarchie de Widget avec leur propriétés. On appel ensuite la boucle principale, les intrinsics se chargent de tout. Les Widgets peuvents être des Label, Menu, Boutton, ... La programmation est d'un très haut niveau. L'utilisateur peut facilement configurer son application. Pour comprendre cela : lancer `editres` sur une application utilisant les Athena Widget.

4.2.3 Athena Widget

4.2.3.1 Utilisation des `widgets` :

Un `widget` est un zone rectangulaire de l'écran possédant un certain comportement.

Chaque `widget` est forcément inclus dans un autre rectangle, il possède donc un père.

Un `widget` porte un nom court et un nom long formé par la concaténation des noms de ces ancêtres et du sien. Le spéparateur est le ".".

Un `widget` *composite* peut contenir d'autres `widget`. Il dispose ses fils selon un algorithme qui lui est propre.

Une ressource est définie par un nom de `widget` suivit d'un nom de ressource et de ":" et la valeur de la ressource.

Un widget peut être complètement défini par ces ressources.

```
MonApplication.menu.quit.label: Quit
MonApplication.menu.quit.foreground: Red
MonApplication.menu.quit.font: -*-courier-*-r-*-*-12-*-*-*-*-*-iso8859-*
```

Ceci n'est qu'une petite partie des ressources.

On peut spécifier les ressources à l'aide de patterns : l'"*" remplace une chaîne `widget` et le "?" un seul. Précision et ordre des ressources sont importants.

L'application "dialogue" avec les `widget` en utlisant les fonctions "GetValues" et "SetValues" qui permettent de modifier durant l'exécution les ressources du `widget`. Il est mis à jour automatiquement sur l'écran.

Un `callback` est un fonction C qui est appelée chaque fois que le `widget` est dans un état "particulier" qui n'est pas lié à un événement précis. Par exemple pour le bouton de commande.

Une action est une fonction C dont on a permis l'appel direct par les `widget`

Un `widget` est capable d'executer certaine actions en fonction d'événements qui lui arrivent : (<Key>, <BtnDown>, <Map>, <EnterWindow>, <Leave>, <Btn1Down>, <Btn1Up>, <Btn3Motion>, ...) ou d'une suite d'événements :

```
*Text.translations : #override\
                Shift<Key>Find: search(backward)\n\
                <Key>Find:      search(forward)\n\
                <Key>Delete:    delete-next-character()\n\
                <Btn3Down>,<Leave>,<Btn3Up> :      select-all()\n\
                <Key>Next:      next-page()\n\
                <Key>Prior:     previous-page()
```

Une application définira la hiérarchie des `widget`. Elle associera des resources aux différents `widgets`.

Exemples de fichiers de ressources dans :
/vol/X11R6/lib/X11/app-defaults

Modification des ressources par `xrdb` ou `editres`. Elles sont recherchées dans : le terminal, app-defaults, le programme.

4.2.3.2 Création de nouveaux `widgets` :

Il sont organisé sous forme d'une hiérarchie avec héritage des méthodes et des attributs. Par exemple Object, Rect, Core, Simple, Label, Command, MenuButton.

On crée un nouveau `widget` en spécialisant son comportement. Il récupère tout ce que ses ancêtres savent faire.

Le nouveau `widget` peut s'ajouter de nouvelles ressources qui seront connue par sa descendance.

Il ajoute de nouveaux `callback`.

Visualisation de la hiérarchie par `viewres`.

4.2.3.3 `widgets` existant :

SIMPLE : Command, Toggle, Label, AsciiTexte, List, Panner, ScrollBar, ...
COMPOSITE : Box, Dialog, Form, Paned, Porthole, ...

4.3 Le langage PostScript

PostScript est un langage de définition de page imprimante comportant textes et graphiques.

- Langage interprété ne comportant que de l'ASCII lisible.
- Langage à pile (FORTH)
- Création de fonctions exécutées à l'intérieur de l'imprimante.
- De nombreuse fonctions graphiques (2D pour le moment)
- Envoit de programmes à l'imprimante pour limiter les transferts.

Quelques instructions PostScript :

```
newpath
x y lineto
x y moveto
x y rlineto
x y rmoveto
chaine bool charpath
closepath

n setgray
n setlinewidth
fill
stroke
clip

gsave
r rotate
x y scale
x y translate
grestore

(texte) show

/nomfonction { ... } def
dup mul add ...
n { ... } repeat
min increment max { ... } for

showpage

%% est le commentaire

%%%% est un m\'eta commentaire
```

4.4 GL

GL bibliothèque graphique Silicon. Niveau bas. Facile d'emploi. Pas indépendant du périphérique. Pour l'image de synthèse (non présenté ici: lumières, maťeriaux). Regarder dans le ".h" pour plus de précisions.

4.4.1 Simple output

- `winopen(char *nom_fenetre)` On mélange graphique/multifen.
- `color(int col)` BLACK, WHITE, RED, ... couleur courante
- `clear()` Effacement écran
- `gexit()`
- `bgnline()`, `bgnpolygon()`
- `v2i(int x, int y)` en pixel (0,0 haut gauche)
- `v3f(float pt[3])` entre -1 et 1, centre, Z vers nous ...
- `endline()`, `endpolygon()`
- `cmov(float x,y,z)`, `charstr(char*)` lettres 2D position 3D
- `gflush()`

4.4.2 Perspective

- `mmode(MVIEWING)`
- `perspective(int angle*10, x/y, zmin, zmax)` zmin max >0, zmax min
- `lookat(ox,oy,oz, px,py,pz, int angla*10 )`
- `viewport(int minx, maxx, miny, maxy)` Tout l'ecran généralement

4.4.3 Objets

Pour afficher plus vite

- `makeobj(int n)` le vide si existe
- `closeobj()`
- `callobj(int n)`

4.4.4 Repère

Pour travailler avec d'autres coordonnées et déplacer des objets

- `translate(x,y,z)`
- `scale(x,y,z)`
- `rot(int angle*10, char axe)`
- `pushmatrix()` y compris trasnformation affichage
- `popmatrix()`

4.4.5 Elimination partie cachées, double buffer

- `zbuffer(TRUE)` initialisation
- `zclear()`
- `doublebuffer()` initialisation
- `swapbuffers()`
- `gconfig()` juste après les initialisations

4.4.6 Options diverse

- `prefsize(x,y)` juste avant le `winopen`, taille fixe
- `linestyle()`
- `linewidth(int)`
- `getsize(long *lar, *hau)`
- `getorigin(long *lar, *hau)` haut gauche fenetre

4.4.7 Input Devices

- `qdevice(KEYBD|MOUSEX|MOUSE3|...)` pas besoin pour REDRAW, INPUTCHANGE
- `device = qread(short *value)` lecture blocante
- `qtest()` non bloquant
- `val = getvaluator(device)` échantillonnage
- `val = getbutton(device)` échantillonnage

La souris a des coordonnées en absolu.

4.4.8 Nommage et picking

Un nom est une pile d'entier, courant comme la couleur.

- `initnames()` Vide la pile.
- `pushname(short)` Ajoute un élément à la pile
- `loadname(short)` Chage le sommet de pile
- `popname()` Ajoute un élément à la pile
- `pick(short *buf, int len)` On préviens que l'on veut identifer.
- `nombreobjet = enpick()` retourne table de table

4.4.9 GL réaliste

En début de programme : `RGBmode() ; c3f(float t[3])`

Faire le `gconfig()` après.

Idée : On définie les données une fois pour toute en leur associant un index. Les changements suivant sont fait en indiquant l'index. `lmdef` fait la définition et `lmbind` indique que l'on veut l'utiliser.

Matériel : `lmdef( DEFMATERIAL, indexmat, 0, table )` Exemple de table:

```
static float materiel[] = {
    AMBIENT, 0.1, 0.1, 0.1,
    DIFFUSE, 0.7, 0  , 0,
    SPECULAR,0.2, 0  , 0,
    SHININESS, 4
    ALPHA, 0.1,
    LMNULL
    }
```

4.5 OPEN GL

OPEN GL :

- Vient de GL, la bibliothèque graphique Silicon.
- 3D, faces cachées, réaliste.
- Niveau très bas.
- Facile d'emploi.
- Seulement sur écrans matriciels.
- Le plus indépendant possible de X11 (ne gère pas les périphériques).

Regarder dans les ".h" pour plus de précisions dans les directories `/usr/include/GL` ou `/vol/X11R6/include/GL` les fichiers `gl.h` et `glu.h`

Des pages de manuel existent pour toutes les fonctions sur Silicon.

Les concepts :

- Des primitives graphiques simples : point, segment, facette, matrice.
- Etat courant pour tout.
- Repère courant (MODEL). Projection courante (PROJECTION).
- Définition d'objet (DISPLAY LIST).
- Couleurs. Depth. Matériaux. Lumières. Textures.
- Désignation (nommage et identification)
- Intégration au fenêtrage existant.

Les noms de fonctions OPEN GL :

- Ils commencent par `gl`
- Chaque nouveau mot commence par une majuscule.
- Le suffixe indique le nombre et type des arguments pour de nombreuses fonctions : Integer, Float, Double, ... De plus suivi de `v` si c'est un tableau.
- Les "défine" en majuscule, les mots étant séparés par des '_'

4.5.1 Affichage graphique

- `glClearColor(fr, fv, fb, fa)`
  `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`
- `glColor3f(fr, fv, fb)` Nécessite un fenêtre vrai couleur.
- `glPointSize(float s)`
- `glLineWidth(float w)`
- `glBegin(type)` ... `glEnd()`
  ou `type` peut être entre autre :
  - `GL_POINTS` des points séparés
  - `GL_LINE_STRIP` une ligne brisée
  - `GL_LINES` des segments disjoints
  - `GL_TRIANGLES` des triangles disjoints
  - `GL_QUADS` des quadrangles disjoints
- `glVertex3f(fx, fy, fz)` Position de point courant dans R3
- `glVertex4f(fx, fy, fz,fw)` R3 + homogène
- `glRasterPos3f(fx, fy, fz)` Position de point courant dans la matrice de pixel écran (opérations pixels).

4.5.2 Transformations

- `glMatrixMode(GL_MODELVIEW || GL_PROJECTION)`
- `glLoadIdentity()` La matrice est remplacée par l'identité.
- `glOrtho(xmin,xmax,ymin,ymax,zmin,zmax)` Projection orthogonale
- `gluPerspective(fangle,lar/hau,zmin,zmax)`
- `gluLookAt(xoeil,yoeil,zoeil,xvu,yvu,zvu,xvert,yvert,zvert)`
- `glViewport(GLint x, y, lar, hau)` Zone d'affichage.
- `glTranslatef(vx,vy,vz)` Translate le repère courant.
- `glScalef(sx,sy,sz)` Dilate le repère courant.
- `glRotatef(angleDegré,vx,vy,vz)` Rotation du repère courant.
- `glPushMatrix()` et `glPopMatrix()` Gestion pile de matrices et donc de repères.

4.5.3 Objets

Pour afficher plus vite un objet statique, on le mémorise une fois pour toute.

- `glNewList(numero, GL_COMPILE || GL_COMPILE_EXECUTE)`
- `glEndList()`
- `glCallList(numero)`
- `glIsList(numero)`

4.5.4 Nommage et désignation

Un nom est une pile d'entier, courant comme la couleur.

- `glInitNames()` Vide la pile.
- `glPushName(nom)` Ajoute un élément à la pile
- `glLoadName(nom)` Change le sommet de pile
- `glPopName()` Enlève le sommet de la pile
- `glSelectBuffer(BUFSIZE, selectBuf)` Indique ou stocker les objets trouvés.
- `glRenderMode(GL_SELECT)` On n'affiche pas, on cherche.
- `gluPickMatrix( xEcran, yEcran, dist, dist, viewport )` Cette fonction doit être appelée avant toute transformation de repère.
- `nb = glRenderMode(GL_RENDER)` Dans `selectBuf` on trouve alors `nb` objets, avec pour chaque objet : `zMin, zMax, tailleNom, nom, ...`

4.5.5 Lampes

On peut avoir jusqu'a 8 lampes active simultanément dans une scène.

- `glEnable(GL_LIGHT0)` Alume la lampe 0..7
- `glDisable(GL_LIGHT0)` Eteint la lampe 0..7
- `glLightfv(GL_LIGHT0, GL_POSITION , float pt[4]);` Position de la source lumineuse en coordonnées homogènes.
- `glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION , float vec[4]);` Direction d'éclairage du spot.
- `glLightf(GL_LIGHT0, GL_SPOT_EXPONENT , float);` Diminution d'éclairement en fonction de l'éloignement du centre du cone. Si 0 éclairement uniforme.
- `glLightf(GL_LIGHT0, GL_SPOT_CUTOFF , float);` Angulaire du cone entre 0 et 90 degré. 180 pour éclairer partout.
- ```
  glLightfv(GL_LIGHT0, GL_DIFFUSE , float [4]);
  glLightfv(GL_LIGHT0, GL_AMBIENT , float [4]);
  glLightfv(GL_LIGHT0, GL_SPECULAR , float [4]);
  ```
  L'énergie émise par la lampe en RVBA. On prendra la bonne valeur lorsque l'on calculera Phong. Il est préférable de prendre DIFFUSE et SPECULAR identiques.
- ```
  glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION , float);
  glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION , float);
  glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION , float);
  ```
  L' énergie est multipliée par 1/(A0 + d*A1 + d*d*A2). Dans la réalité : A0=A1=0 et A2=1 pour une source ponctuelle.

4.5.6 Matériaux

On utilise des matériaux pour avoir un effet réaliste. Ils nécessitent au moins une source lumineuse.

- `glEnable(GL_LIGHTING);` Met en route l'éclairage réaliste.
- `glNormal3f(float vx, vy, vz);` Normale à la surface.
- ```
  glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION , float emission[4]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE , float diffuse[4]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR , float speculaire[4]);
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT , float ambiante[4]);
  glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS, float n);
  ```
  Les coefficients du matériaux. Le AMBIENT tient compte de la lumière AMBIENT de la lampe et pas l'EMISSION.

4.5.7 Modèle de lumière

Ceci configure l'éclairage indépendamment des objets et des lampes. C'est la page du manuel la plus complète car elle donne la formule totale.

- `glLightModelf(GL_LIGHT_MODEL_AMBIENT, float [4]);`
- `glLightModelf(GL_LIGHT_MODEL_LOCAL_VIEWER, float);`
- `glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, float);`
  ```

5 Interaction homme/machine

L'application intéractive (graphique le plus souvent) suplante l'application "textuelle". Les utilisateurs préfèrent l'interactivité à la puissance d'une application.

Quelle sont donc les avantages et inconvénients des applications interactives et textuelles. Dans la suite on considère que l'application interactive est forcément dans un environnement graphique.

- La facilité d'apprentissage.
  Dans une application interactive on peut très facilement rechercher la commande qui est nécessaire en parcourant les menus. Dans le cas textuel, il est possible d'avoir le même type d'interface (accélérateur clavier). Si on le possède pas, on doit disposer d'un Help.
- Désignation instinctive d'un point dans $\mathbb{R}^2$ pour une application intéractive, elle est plus difficile et moins précise pour une application textuelle.
- L'utilisation d'icone permettant de meilleur repères à l'utilisateur est bien entendu limité au graphique.
- La limitation du texte lui permet de fonctionner sur des terminaux qui ne sont pas graphiques.
- La puissance d'expression n'est pour le moment accessible que via un interface textuel ou bien un langage de programmation graphique.

Je suis contre le traitement de texte WYSIWYG et pour la PAO WYSIWYG! Et ceci pour 2 raison :

Un traitement de texte WYSIWYG ne permet pas à l'utilisateur de se concentrer sur la teneur du document, il est toujours en train d'en améliorer la présentation et ceci avant même la première lecture.

Un formateur de texte permet à l'utilisateur de prendre son éditeur de texte favori (qu'il manipule donc rapidement). Il permet d'entrer du texte aussi (sinon plus) rapidement qu'en interactif. De plus il permet généralement d'associer une sémantique plus forte au texte. L'inconvénient est bien sûr que ce que l'on voit sur l'écran n'est pas ce qui sera imprimé (mais comme le formatteur de texte est plus puissant que le traitement de texte, on n'a pas à s'en soucier).

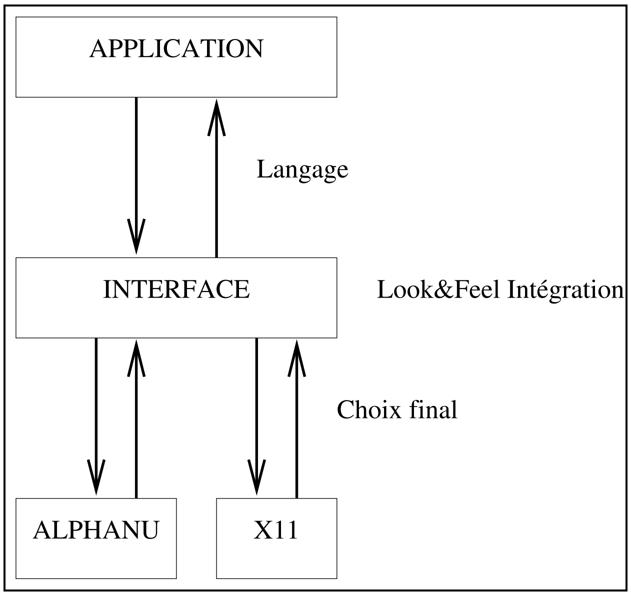


FIG. 17 – *Le découpage d'une application interactive*

5.1 Les règles d'or du programmeur d'application interactive

- Utilisateur
  - Les applications interactives doivent se ressembler.
  - On ne doit pas avoir besoin de lire la documentation.
  - On doit savoir ce que le programme est (en train/veut/essaye) de faire.
  - Le programme doit signifier la durée d'une action (interruptible).
  - La logique de l'utilisateur n'est pas celle du programmeur.
  - Adaptation du programme au niveau de l'utilisateur.
  - Pouvoir accéder à un mode commande à tous moment.
  - Mémoriser l'état courant du logiciel.
  - Mémoriser l'état de l'utilisateur.
- Conception
  - Temps de programmation est moins important que celui de l'utilisateur.
  - Minimiser le nombre d'informations affichées.
  - Minimiser le nombre de mouvement pour faire un acte.
  - Minimiser le nombre de choses qui bougent sur l'écran.
  - Minimiser le nombre de fonctions possibles.
  - Utiliser des processus déjà cablé de l'utilisateur.
  - L'utilisateur doit pouvoir personnaliser son espace de travail.
  - Pas de MODE de fonctionnement, pas de fonctions contextuelles.
  - Pas de changement d'état incongru.
  - Help interactif.
  - Messages d'erreur claire et proposition de correction.
  - Undo/Redo.
  - Toujours un feed-back.
  - Pas de PAUSE.
- Programmation
  - Séparer l'application des entrées sorties.
  - Programmation objet.
  - Multifenêtrage.
  - Interface pouvant aussi fonctionner en texte.
  - Analyse du fonctionnement de l'utilisateur.
  - Utiliser des boîtes à outils communes.
  - Utiliser un constructeur d'interface.
- Tester
  - Tester le programme par un innocent.
  - OBEIR à l'utilisateur.
  - Si l'utilisateur ne comprend pas c'est la faute de l'interface.
  - Analyser le temps d'apprentissage.
  - Analyser le temps passé à utiliser chaque fonction.

6 Synthèse d'image

La synthèse d'image permet de passer d'un modèle 3D à un image.

L'image générée doit être le plus proche possible de la réalité. Elle doit donc tenir compte de certaines choses :

- La perspective. Pour cela on plonge un oeil (virtuel) dans le modèle.
- Elimination des parties cachées. Dans la réalité les objets sont opaques.
- Eclairement, ombres portées. Pour cela on plonge des sources lumineuses dans le modèle.
- Paramètres de surfaces : transparence, textures, ...
- Le périphérique de sortie : matriciel ou vectoriel.
- Les problèmes dus au fait que l'affichage est discret.

Pour créer ces images, de nombreuses techniques sont possibles. Nous allons voir les plus répandues.

- Pour faire de la perspective on utilise les coordonnées homogènes, celle-ci permettent de combiner toutes les transformations dans une matrice. Le découpage est fait directement en coordonnées homogènes.
- La méthode la plus utilisée est le Z-buffer (tampon de profondeur) Et la plus simple l'algorithme du peintre.
- Le modèle d'éclairement le plus réaliste est PHONG, on utilise GOUREAU pour un affichage rapide (interpolation).
- La méthode permettant de simuler tous les effets optiques est le lancer de rayons, elle simule le trajet inverse de la lumière.
- Les premiers algorithme de faces cachées étaient spécifiques aux traceurs de courbes et écrans vectoriels, ils sont de moins en moins utilisés.
- Pour éliminer les marches d'escalier et les problèmes de moiré, on utilise des méthodes d'antialiassage (antialiasing).

La synthèse d'image utilise un nombre considérable d'astuces et d'approximation pour simplifier les programmes, accélérer les calculs et améliorer le rendu.

- Pour la perspective, on applique un transformation perspective, ensuite il suffit de faire une projection "orthogonale" avec l'oeil à l'infini (cela revient à supprimer la coordonnées profondeur).
- De nombreuses méthodes ne fonctionnent qu'avec des facettes planes, on subdivise donc généralement les surfaces. complexes en facettes planes.
- Pour éliminer les facettes on utilise des méthodes de lissages qui les gomment.

6.1 De l'utilité d'un modèle

Le but de la synthèse d'image est bien entendu de faire des images. Mais par n'importe comment, en effet les palettes graphiques permettent de créer des images mais ces images n'ont à voir avec les objets qu'elles représentent. On confond donc la carte et le territoire.

Les modèles permettent notamment :

- De définir des objets complètement :
  - Leur géométrie (2D, 3D ou plus).
  - Les matériaux.
  - Des attributs pour la présentation.
  - Des attributs pour la gestion de l'objet lui même.
  - ...
- D'afficher l'objet par différentes méthodes, de l'intégré dans une scène, de créer différentes vues.
- De faire un usinage de l'objet (c'est une méthode d'affichage).
- De faire des calculs de structures.
- ...

Les modèles géométriques les plus connus :

- Le modèle filaire : ensemble de lignes droites ou courbes (arc cercle, ellipse, spline, bezier, ...). Le problème : rien n'est éliminé, affichage brouillon.
- Le modèle surfacique : on définit un ensemble de facette plane ou gauches. Mais les objets réels ne sont pas des surfaces.
- Le modèle volumique :
  - BREP : Boundary Representation.
  - CSG : Constructive Solid Geometry.
  - ...

La modélisation influe sur le processus de visualisation, en effet certain modèle sont plus adapté à un algorithme de visualisation qu'a un autre.

Il est toujours possible de convertir d'un modèle à un autre mais on se heurte aux problèmes suivant :

- Perte de sémantique.
- Perte de précision.
- Impossibilité totale.

Plus l'information contenu dans un modèle est "riche" (mais non redondante) mieux on pourra la traiter.

- La hiérarchie dans le modèle permet d'accélerer l'affichage.
- Le réseau permet de ne pas répéter certain calcul, il diminue aussi la place mémoire associée au modèle.
- Associer un symbole à l'objet pour le remplacer si trop petit.
- ...

6.2 Coordonnées homogènes

Elle permettent de :

- représenter des points à l'infini.
- travailler sans faire de divisions.
- coder facilement des transformations perspectives.

On ajoute une coordonnées supplémentaire au point.

Dans le cas d'un point $\begin{pmatrix} x' \\ y' \end{pmatrix}$ de $\mathbb{R}^2$. Il est équivalent au point $\begin{pmatrix} x \\ y \\ t \end{pmatrix}$ avec $x' = \frac{x}{t}$ et $y' = \frac{y}{t}$.

Un point de $\mathbb{R}^2$ a une infinité de représentations en coordonnées homogènes. Les points homogènes avec une coordonnée homogène t nulle sont à l'infini dans l'espace normal, ils indiquent en fait une direction. On peut coder les transformations affines par une simple matrice si l'on travaille en coordonnées homogènes.

Pour une transformation classique : $\left(\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} v_x \\ v_y \end{pmatrix}\right)\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by + v_x \\ cx + dy + v_y \end{pmatrix}$

Cette transformation affine est équivalente à une simple produit matrice vecteur lorsque l'on passe en coordonnées homogène. La matrice correspondante est : $\begin{pmatrix} a & b & v_x \\ c & d & v_y \\ 0 & 0 & 1 \end{pmatrix}$

Démonstration : soit le point de coordonnées $\begin{pmatrix} \frac{x}{t} \\ \frac{y}{t} \end{pmatrix}$ Si je lui applique une transformation codée par une matrice et un vecteur j'obtient : $\begin{pmatrix} a\frac{x}{t} + b\frac{y}{t} + v_x \\ c\frac{x}{t} + d\frac{y}{t} + v_y \end{pmatrix}$

Si l'on travaille en homogène.

$$\begin{pmatrix} a & b & v_x \\ c & d & v_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ t \end{pmatrix} = \begin{pmatrix} ax + by + v_x t \\ cx + dy + v_y t \\ t \end{pmatrix}$$

Comme on peut le voir, les deux points sont équivalents. Il en ressort que le produit de transformations affines devient simplement un produit de matrices en coordonnées homogènes. A quoi servent les 2 coefficients nuls dans la matrice homogène? Tout simplement pour coder la perspective. Nous avons $\frac{p_y}{d} = \frac{y}{d+x}$ donc $p_y = y\frac{d}{d+x}$.

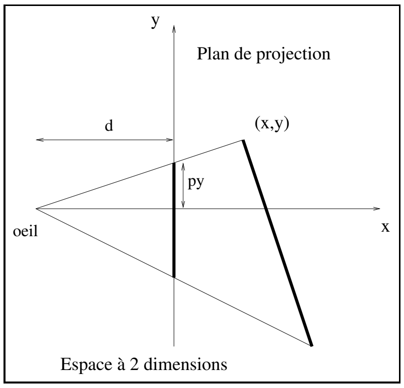


Fig. 18 – *Perspective*

il y a un problème pour $x = -d$, de plus on fait une division. Si nous travaillons en coordonnées homogènes, il suffit de trouver la matrice de projection. Dans notre cas : $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{d} & 0 & 1 \end{pmatrix}$. Il n'y a pas de division immédiate.

6.3 Les algorithmes de faces cachées

Ces algorithmes fonctionnent sur des modèles surfaciques ou volumiques. Le choix d'un algorithme d'élimination des parties cachées est dicté par :

- Le type de sortie : filaire ou pixel
- Le type de modèle en entrée.
- Si l'on va faire de l'animation.
- Si l'on privilégie le rendu ou la vitesse.

Les algorithmes d'élimination des parties cachées les plus connus sont :

- Les algorithmes travaillant dans l'espace objet, indépendamment du type et de la précision d'affichage. (Galimberti et Montanari années 60)
- Algorithme travaillant avec des priorités, celui du peintre et l'algorithme de Fuchs 1980. (limité aux facettes planes)
- Les algorithmes de balayage ligne par ligne qui transformes le problème de visibilité 3D en un problème 2D. Ils sont incrémentaux et rapide.
- Les algorithmes de subdivision qui découpe le problème jusqu'a un cas évident (espace objet ou écran)
- Des algorithmes spécialisés pour un type d'affichage. Les cas les plus connus sont les volumes numériques et les molécules (que des sphères).
- L'algorithme du tampon de profondeur qui est le plus utilisé de nos jours mais qui nécessite un hardware spécialisé.
- Le lancé de rayon qui transforme le problème 3D en un problème 1D. Pour les images réalistes, c'est l'algorithme le plus utilisé. Il travaille en volumique.

Les ombres portées sont complexe à ajouter, il est souvent nécessaire de developper un algorithme particulier. Le Z-Buffer peut être modifié pour les prendres en compte. Le lancé de rayon les prend en compte automatiquement.

6.4 Algorithme du peintre

L'algorithme du peintre est une méhode simple et efficace pour calculer les faces cachées.
Elle nécessite :

- Une matrice de pixel pour l'affichage.
- Un modèle ne contenant que des facettes.
- Un méthode rapide d'affichage de polygones.
- Une mémoire contenant toutes les facettes à afficher.
- Un tri rapide.

L'idée est simple : il suffit d'afficher les objets des plus éloignés aux plus proches. Les objets proches vont donc effacer les objets éloignés lorsqu'ils vont s'afficher devant. Pour afficher une image le processus est le suivant :

- Faire subir la transformation projective à toutes les facettes.
- Calculer la priorité de chaque facette.
- Afficher les facettes de la plus basse priorité à la plus haute.

Les inconvénients de cette méthode sont :

- On doit stocker les facettes et les trier.
- On doit tout recalculer à chaque image.
- On ne peut pas sortir sur un traceur de courbes.
- Certaines faces cachées sont mal traitées.

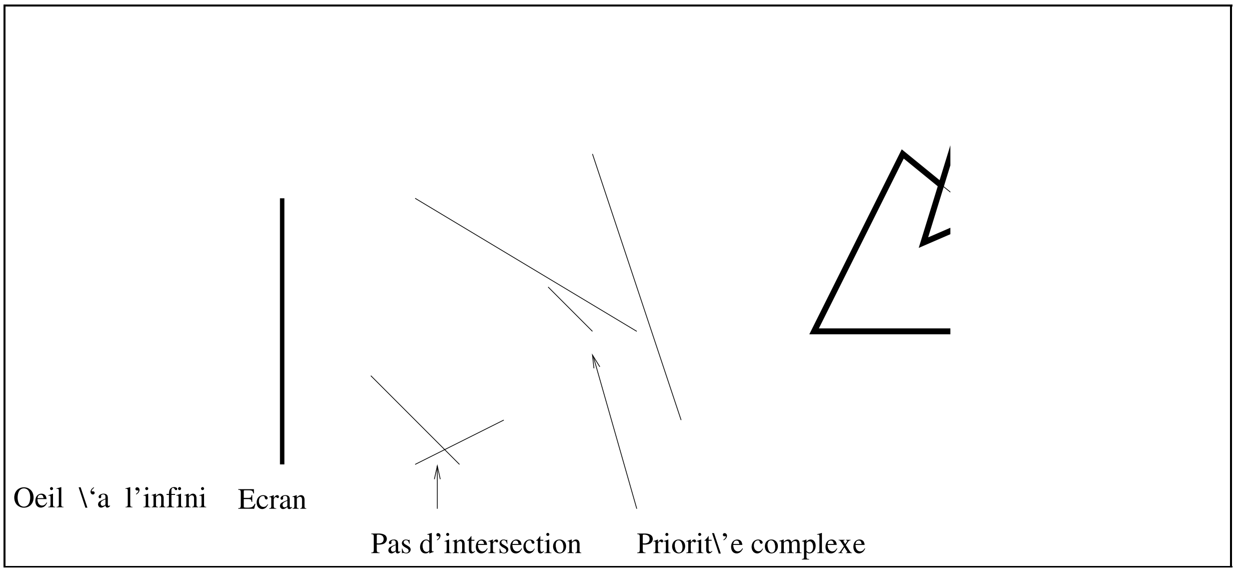


FIG. 19 – *Cas impossibles pour le peintre*

Les cas impossibles sont traités en découpant les facettes.

On peut étendre cette méthode à des objets ne s'intersectant pas. Dans ce cas, on affiche les objets du plus éloigné au plus proche, et dans chaque facette, la plus éloigné à la plus proche. Ceci permet de simplifier les tris.

Dans méthodes plus évoluées existent, elles servent à faire notamment du temps réel.

6.5 Le Z-buffer

Le problème de faces cachées peut être résolu facilement en le simplifiant. On passe d'un problème de visibilité 3D en 1 problème 1D. Pour cela on résoud le problème de visibilité pixel par pixel. Le problème de la visibilité 1D est très simple, en effet l'objet visible est celui qui est le plus proche pour ce pixel.

L'astuce est simple : pour chaque pixel affiché, on lui associe en plus de la couleur, une profondeur. L'algorithme est le suivant :

```
POUR tous les pixels P de l'ecran
   couleur(P) = couleur du fond
   profondeur(P) = infini
POUR toutes les facettes a afficher
   POUR tous les pixels P de l'ecran sur lesquels la facette se projete.
      Calculer la profondeur D de la facette pour ce pixel.
      SI D<profondeur(P)
         profondeur(P) = D
         couleur(P) = couleur de la facette
```

Avantages :

- On n'est pas obligé d'avoir l'ensemble des facettes pour afficher.
- L'ensemble des calculs peut-être fait en entier, on utilise un BRESENHAM pour la profondeur.
- L'algorithme peut-être complètement cablé.
- On utilise une place mémoire constante.
- Résoud tous les cas tordus d'élimination de parties cachées. Y compris les facettes intersectantes.
- On peut étendre la méthode à tous les types d'objets.

Inconvénients :

- La place mémoire est importante pour de petite image.
- Méthode pas très rapide si elle est faite logicielement.
- Pas très évident comme méthode pour faire les ombres portées, miroirs et autres effets optiques.

Le Z-buffer est actuellement la méthode la plus utilisée pour faire l'affichage 3D temps réel.

La méthode de GOUREAUD fait une interpolation linéaire sur la couleur de la facette à partir de ses bords et de l'orientation des facettes adjacentes. Cette interpolation permet de rendre difficilement dicernables les facettes lors de l'affichage, ceci évit de discrétiser une sphère par des centaines de facettes minuscules. Il reste néanmoins des problèmes sur les bords. La méthode de GOUREAUD consiste aussi à faire un BRESENHAM sur les intensités lumineuses, elle est donc rapide.

6.6 La couleur

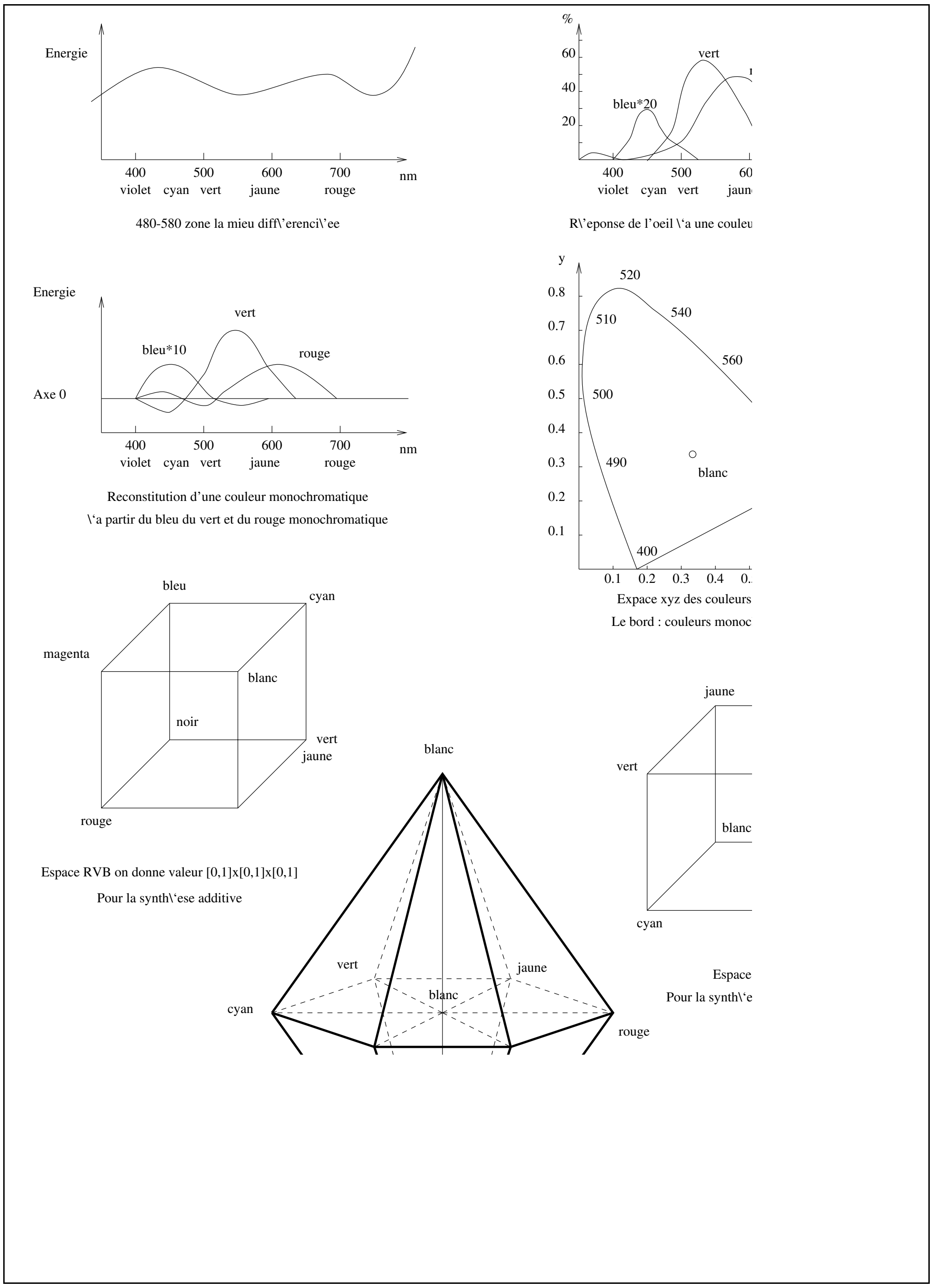


Fig. 20 – *Où est passée la couleur?*

La couleur, sait-on ce que c'est?

- Différentiation chromatique : de 1nm a 10nm (128 teintes), de saturation de 16 à 23 et de la luminosité (logarithme) sur un écran : 128 soit environ 350000.
- Une couleur verte pour l'oeil peut être un vert monochromatique ou bien un mélange. On ne pourra les distinguer qu'avec un filtre.
- Toutes les couleurs ne sont pas synthétisables avec RVB ni CMY.
- On ne peut recopier toutes les couleurs d'un écran sur une imprimante. Pixels les uns sur les autres ou supperposé.
- Réfléchir avant d'utiliser des couleurs pour un affichage. Ils faut des contrastes!

6.7 Textures

2 catégories :

- Les textures 2D que l'on plaque sur la surface des objets.
- Les textures 3D dans lesquelles on taille le volume des objets.

Dans tous les cas, une texture est une fonction de $\mathbb{R}^n \rightarrow P$ ou P est l'espace des attributs graphiques (coéfficients ambiants, diffus, spéculaires, état surface, ...).

6.7.1 Perturbation de la normale

L'un des attributs graphiques est en fait géométrique c'est la pertubation de la normale. Ceci est obligatoire si l'on veut générer des surfaces granuleuses brillante. Par exemple si 'on fait tourner une lampe autour d'une orange.

Soit $\vec{u}, \vec{v}$ le plan tangent à la surface. La normale à la surface ne sera pas $\vec{u} \cap \vec{v}$ mais $(\vec{u} \cap \vec{v}) + d_u\vec{u} + d_v\vec{v}$ ou d_u et d_v (deux scalaires) sont la pertubation de la normale.

Inconvénients :

- Si la perturbation est forte, des effets indésirables apparaisent (retournement de la normale).
- Le contour de l'objet n'a pas de petites bosses.
- Les petites bosses n'ont pas d'ombre portées.

L'avantage est bien sur que l'orange n'est qu'une sphère au lieu de nombreux objets (les bosses)

6.7.2 Accès aux textures

La texture peut être tabulée ou calculé.

Inconvénients des tabulées :

- Une texture tabulé consomme énormément de mémoire.
- Si l'on ne veut pas voir de pavés lors de l'agrandissement il faut faire une interpolation linéaire.
- Si l'on ne veut pas de moirés, il faut précalculer la texture à des résolutions inférieures. Jusqu'a une texture 1x1 pixel (MipMapping OpenGL)

Inconvénients des calculées :

- Une texture calculée est lente d'accès.
- Problème de moiré prennent du temps à résoudre car il faut faire une moyenne.

Généralement la texture tabulée est réservée aux texture 2D et la calculée au 3D.

6.7.3 Affichage des textures

Textures 3D : à partir de la position dans $\mathbb{R}^3$ du point à texturer. C'est une opération simple ne posant de pas problème.

Textures 2D : Si l'on veut une texture "raccordée" le problème est extrêmement complexe et sans solution pour les cas tordus.

Détermination de coordonnées sur la surface d'un objet.

- En CSG : on plaque chaque texture sur une primitive. Il y a de plus plusieur manière de plaquer la texture sur la primitive. De toute facon, on peut plaquer une texture sur un objet qui n'est pas plat sans la déformer. Le raccord entre primitive n'est pas fait.
- En surfacique : on peut représenter la surface par un polyèdre. Dans les bons cas, on peut projeter la texture sur le polyèdre. Ceci nous donne les coordonnées dans la texture des sommets de chaque facettes. L'afichage de facette texturé ne pose pas de problème.

  Exemple : le visage humain, même méthode pour scanner et projeter la texture.

Dans les autres cas tordus : un zèbre par exemple, il est très difficile de plaquer la texture. Dans ces cas on génère directement la texture sur le polyèdre. Il n'y a donc plus de plaquage.

6.7.4 Synthèse de textures : automates cellulaire

Le plus connu est le jeu de la vie, la matrice régulière de booleen indique si une cellule existe ou pas. Soit v le nombre de cellule 8 connexe. Pour calculer le motif suivant à partir du motif précédent :

- une cellule meure si $v < a$ ou $v > b$
- une cellule nait si $c < v < d$

En choisissant bien ces 4 valeurs on obtient des motifs intéressants, ou bien des animations.

Les automates cellulaires permettent aussi de générer des modèles complexes. Par exemple une ville.

6.7.5 Synthèse de textures : montagne, nuage, marbre

Les fractales stochastique de la famille "montagne" permettent de générer aussi du marbre, des nuages, ...

Idée 1D :

```
#define L 64

float altitude[L+1] ;
int i,j ;

altitude[0] = 0. ;
altitude[L] = 0. ;

for(i=64;i>=2;i/=2)
   {
      for(j=i/2;j<L;j+=i)
         {
         altitude[j] = (altitude[j-i/2] + altitude[j+i/2])/2 + alea(-i,i) ;
         }
   }
```

On ajoute une valeur aléatoire d'espérance plus en plus petite. C'est la variation de l'espérance qui détermine la dimension fractal de la montagne.

Pour le marbre : On reproduit le processus géologique, dépot de couches parallèles puis perturbation et compressions. Soit $m(x)$ la fonction retournant le type du marbre à plat, le marbre perturbé sera : $m(X+Y+Z+\text{altitude}(X,Y,Z))$

Ces méthodes son adaptables plus ou moins facilement à la synthèse de textures sur des polyèdres

6.7.6 Synthèse de textures : peau de bête

Issu de la chimie/biologie (Turing), schématiquement la peau contient un activiteur et un inhibiteur qui donne la couleur à la peux. Les deux composés chimiques diffusent lentement dans la peau et agissent l'un sur autre d'une certaine façon. Certaines cellules génère la substance activatrice.

Suivant les composés chimiques les formules sont différentes.

La formule proposée par Turing (étendue au 2D) était avec X l'activateur et Y l'inhibiteur :

$$\delta X_{(i,j)} = s * (X_{(i,j)}Y_{(i,j)} - Y_{(i,j)} - 12 + q) + \frac{D_x}{4}\left((X_{(i+1,j)} + X_{(i-1,j)} + X_{(i,j+1)} + X_{(i,j-1)} - 4 * X_{(i,j)}\right)$$

$$\delta Y_{(i,j)} = s * (16 - X_{(i,j)}Y_{(i,j)}) + \frac{D_y}{4}\left((Y_{(i+1,j)} + Y_{(i-1,j)} + Y_{(i,j+1)} + Y_{(i,j-1)} - 4 * Y_{(i,j)}\right)$$

Les X et Y initiaux sont à 4 environ.

Ou D_x et D_y sont les coefficients de diffusion (≤ 1) et s le coefficient de réaction ($< 1/1024$) plus il est grand plus les structures sont petites. q est une fonction aléatoire pertubant le système pour éviter une trop grande régularité.

Des effets d'une infini variété sont générés avec seulement quelque composés chimiques.

Ceci est très facilement modifiable pour générer directement une texture sur un polyèdre en évitant donc la phase de plaquage.

6.8 Aliasing et Antialiasing

L'aliasing est un phénomène qui dégrade les images crées ou traitées par les ordinateurs. Différents type d'aliasing existent :

- Les marches d'escaliers sur les droites ou les bords de polygones.
- Les moirés.
- Les roues du chariot qui tournent à l'envers.

Tous ces effets sont dus à une chose : le **sous-échantillonnage**. En effet une image numérique est une discrẽtisation de l'espace et une animation discrêtise le temps. Ceci est une conséquence du théorème du Shannon qui dit que la fréquence d'échantillonnage ne doit pas être inférieur au double de la fréquence échantillonnées si l'on ne veut pas perdre une partie importante du signal.

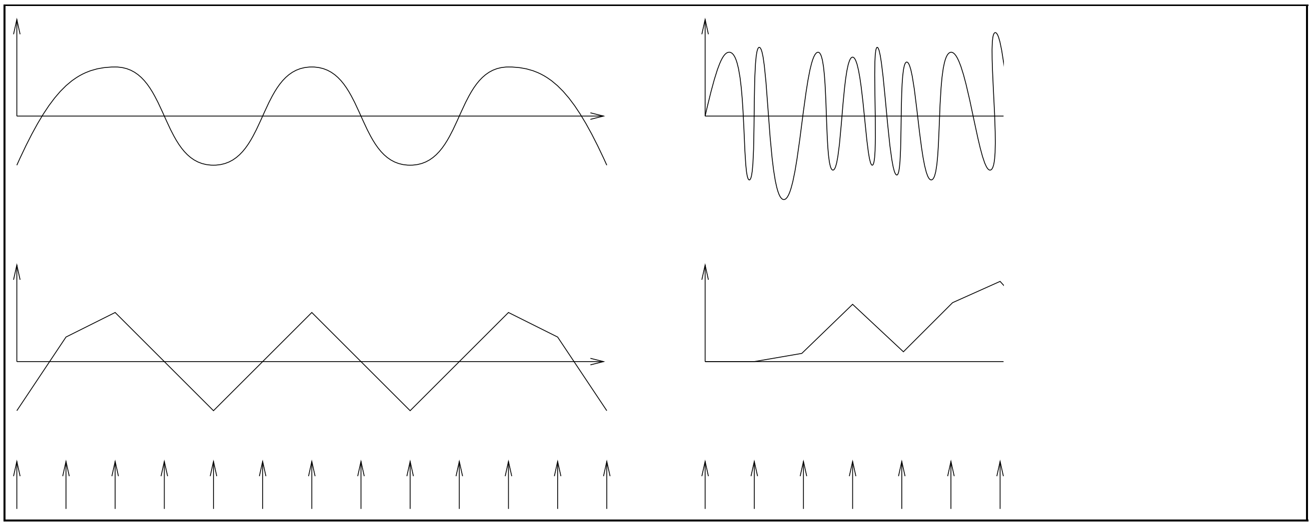


Fig. 21 – *Echantillonnage et reconstruction d'un signal*

Si le signal à échantillonner possède des fréquences trop grandes il faut le filtŕer pour éliminer les hautes fréquences avant de l'échantillonner. L'arsenal mathématique permet de faire cela en utilisant les trasnformées de Fourrier, en synthèse d'image on recherche des solutions plus simple si elles existent.

Le tracé de droites (fréquence infini) en est un bon exemple. On peut réduire la fréquence en traçant une droite épaisse ou en traçant sur un écran ayant une meilleur résolution, la deuxième méthode fonctionnant dans tous les cas.

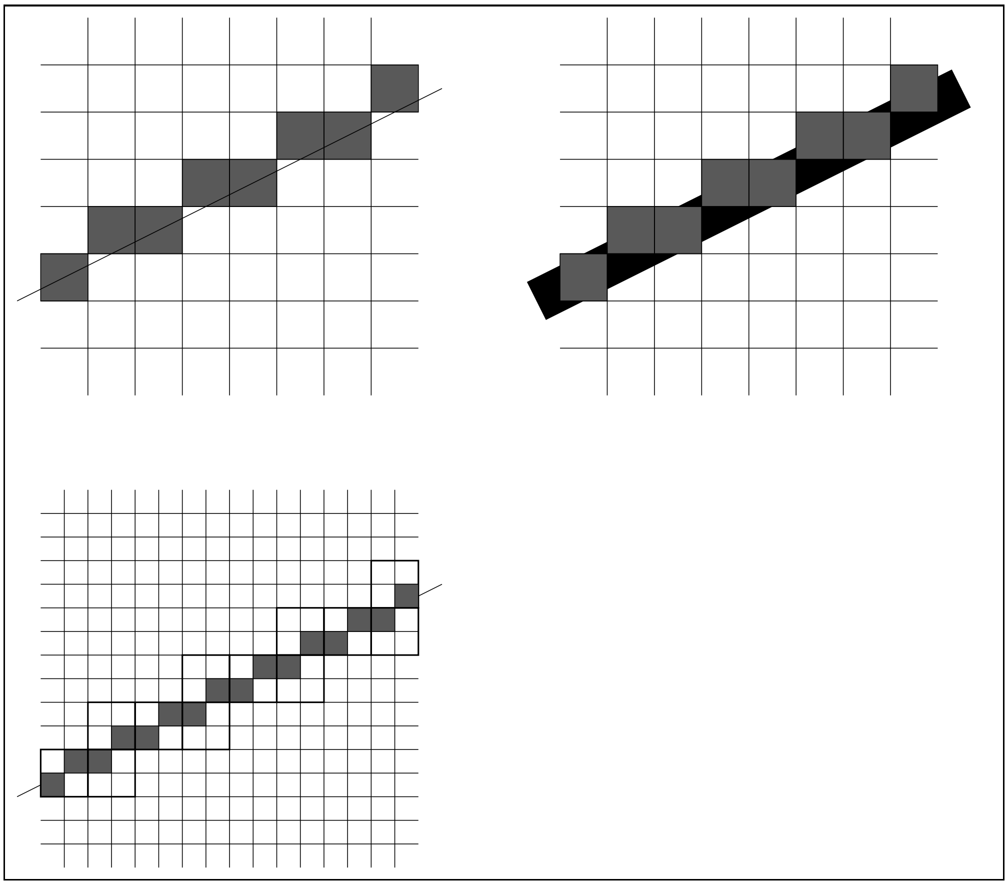


Fig. 22 – *Tracé de droites*

Le cas des moirés est plus difficile à résoudre.

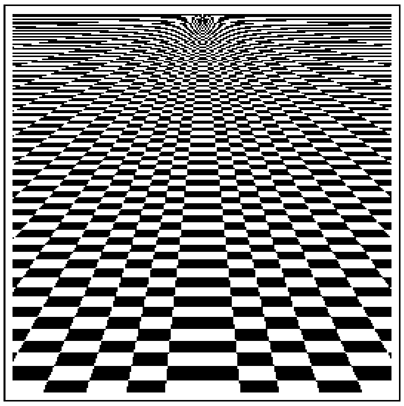


Fig. 23 – *Un damier à l'infini...*

Il est nécessaire de filtrer l'image avant de l'afficher, on obtiendra alors du gris. Une autre astuce consiste à ne pas faire un échantillonnage régulier mais aléatoire, on obtient alors des pixels aléatoires mais plus de zones incongrus.

6.9 Une image réaliste

Pour créer une image réaliste il est nécessaire de connaître parfaitement la réalité. Celle-ci est composée de volumes et des surfaces. Pour les éléments de surfaces :

- Le spectre d'émission et de réémission.
- Les directions d'émission et de réémission.
- L'état de la surface : brillant, mat.
- Le spectre d'absorption.
- Le spectre de réémission.
- Le spectre de transmission.

Pour les éléments de volume :

- Le spectre d'absorption.
- Le "spectre" d'indices de réfraction. ou les spectres dans le cas d'un objet biréfringent.
- Le spectre d'émission et de réémission.
- Les directions d'émission et de réémission.

Il faut noter qu'aucun logiciel ne tient compte de toute ces possibilités.

Si l'on veut créer l'image réaliste par excellence, il suffit de simuler le trajet des photons qui partent des sources de lumières jusqu'a ce qu'ils soient absorbés. Une surface sensible sera placée dans la scène qui comptabilisera les photons la touchant afin de créer l'image.

Estimation du temps de calcul.

- Une pièce fermée de 3x3x3 mètres.
- Une rétine de 0.01x0.01.
- Une photon subit en moyenne 5 réflexions.
- La résolution de la rétine est de 1000x1000 pixels.
- Il faut au moins que 1000 photons arrivent par pixel.
- La surface du pixel est de 10^{-10} mètre carré.
- On considère grossièrement que le nombre de photons n'arrivant pas sur l'oeil est égal au rapport de la surface des murs sur la surface de l'oeil, soit : 3.10^{11}
- Il faut donc émettre approximativement : 3.10^{14} photons.
- En lançant un photon en 10^{-3} seconde, il faut attendre 3.10^{8} secondes soit environ 10 années. On peut considérer ce chiffre correct à un facteur 100 en plus ou en moins.

Bien entendu on ne peut pas faire un tel calcul dans la réalité. On est obligé de faire des simplifications et approximer le problème. L'important est bien sur que l'utilisateur pense voir une image réaliste même si les calculs effectués son complètement faux.

La méthode élémentaire pour réaliser les images réalistes est le lancé de rayons. Cette méthode recalcule (quand c'est possible) le parcour inverse de la lumière.

6.10 Le modèle d'éclairement

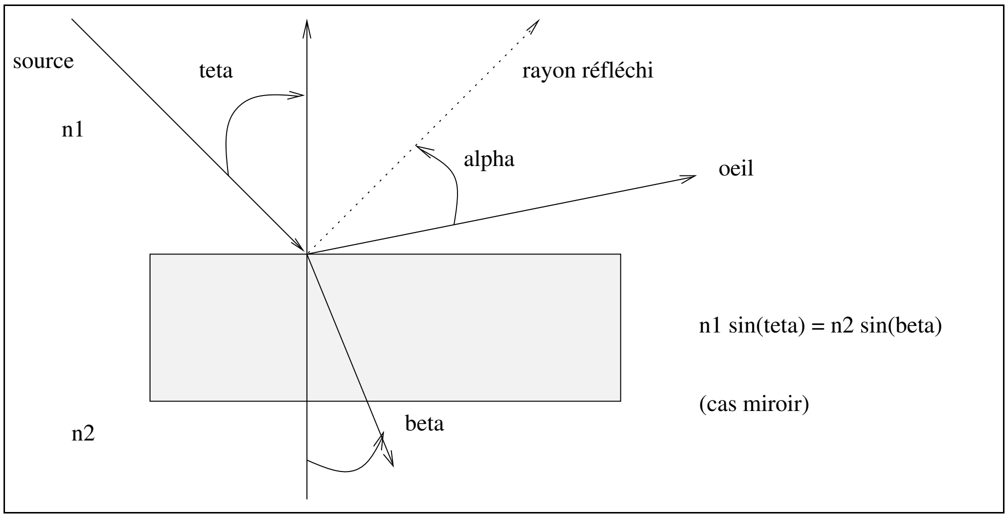


FIG. 24 – *Différent type de réémission*

Le modèle déclairement fait ces calculs en monochrome. Les calculs doivent être fait 3 fois pour obtenir une image couleur. Les coefficients sont bien entendu fonction de la longueur d'onde.

Les sources lumineuses sont considérées comme ponctuelles.

Soient :

I_p L'intensité de la source lumineuse.
R La distance de l'observateur à l'object.
r La distance de la lampe à l'object.

Le modèle de PHONG prend en compte :

- Lumière diffuse (réémise dans toutes les directions). $I_d = \frac{I_p}{R+r} k_d \cos\theta$

  k_d est la constante de diffusion.

- Lumière spéculaire (émise de préférence dans une direction) Ceci pour les surfaces légèrement réfléchissantes.

  $$I_k = \frac{I_p}{R+r} k_s \cos\alpha^n$$

  n est le coefficient d'état de surface. Plus il est grand plus la surface est brillante.

- Lumière réfléchie (miroir parfait) $I_r = k_r S$

  S est la quantité d'énergie arrivant suivant le bon angle.

- Lumière transmise (object transparent). $I_t = k_t S$

  S est la quantité d'énergie arrivant suivant le bon angle.

- Lumière ambiante :
  $I_a = 0$ Alors les ombres sont noire.
  $I_a = constante$ Alors les ombres sont plates.
  $I_a = fct(normale)$ Alors OK si fct est continue.

$$I = I_d + I_k + I_r + I_t + I_a$$

Les cosinus sont fait par produit scalaire.

6.11 Méthodes d'accélération du lancer de rayons

Le temps de calcul est égal au produit :

- Temps d'intersection d'un rayon avec un objet.
- Nombre de pixel à calculer.
- Nombre de rayons par pixel (au minimun 2)
- Nombre d'objets dans la scène.

Deux grands types d'accélérations sont possibles :

- Accélérations *hardware*
  - Composants spécialisés pour certains calculs.
    - Transformation inverse du rayon.
    - Opérations booléenne.
    - Calcul de zéros de polynomes.
  - Parrallélisme.
    - Faire calculer un rayon par processeur.
    - Associer une zone de l'espace par processeur.
- Accélérations logiques
  - Diminuer le temps de calcul d'une intersection.
    - Utilisation d'englobants pour les cas complexes.
    - Résolution de polynomes, (continuité)
  - Diminuer le nombre de pixels.
    - Définir la couleur du pixel au juger (approximatif)
    - Faire des calculs pour des groupes de pixels.
  - Diminuer le nombre d'intersections. On ne lance les rayons que sur des objets *actifs*.

Méthodes générales d'accélérations qui sont applicables dans de très nombreux cas.

- Propriété du vide : placer le plus petit objet à gauche.
- Tout objet dont la distance minimale est plus grande que la distance de la source lumineuse au point ou je veux calculer l'ombrage ne peut pas faire d'ombre.
- On suppose que l'image est continue, on peut alors faire des prédictions sur la valeur des pixels.
- Propriétés des arbres dont les noeuds supérieurs sont des UNIONs. On appelera `objet` les arbres dont la racine est différente de l'UNION et au-dessus duquel il n'y a que des UNIONs.
  - Le point d'intersection avec un objet, le plus proche est le point visible. On trie les objets par distance minimale.
  - On s'arrete au premier objet faisant de l'ombre.

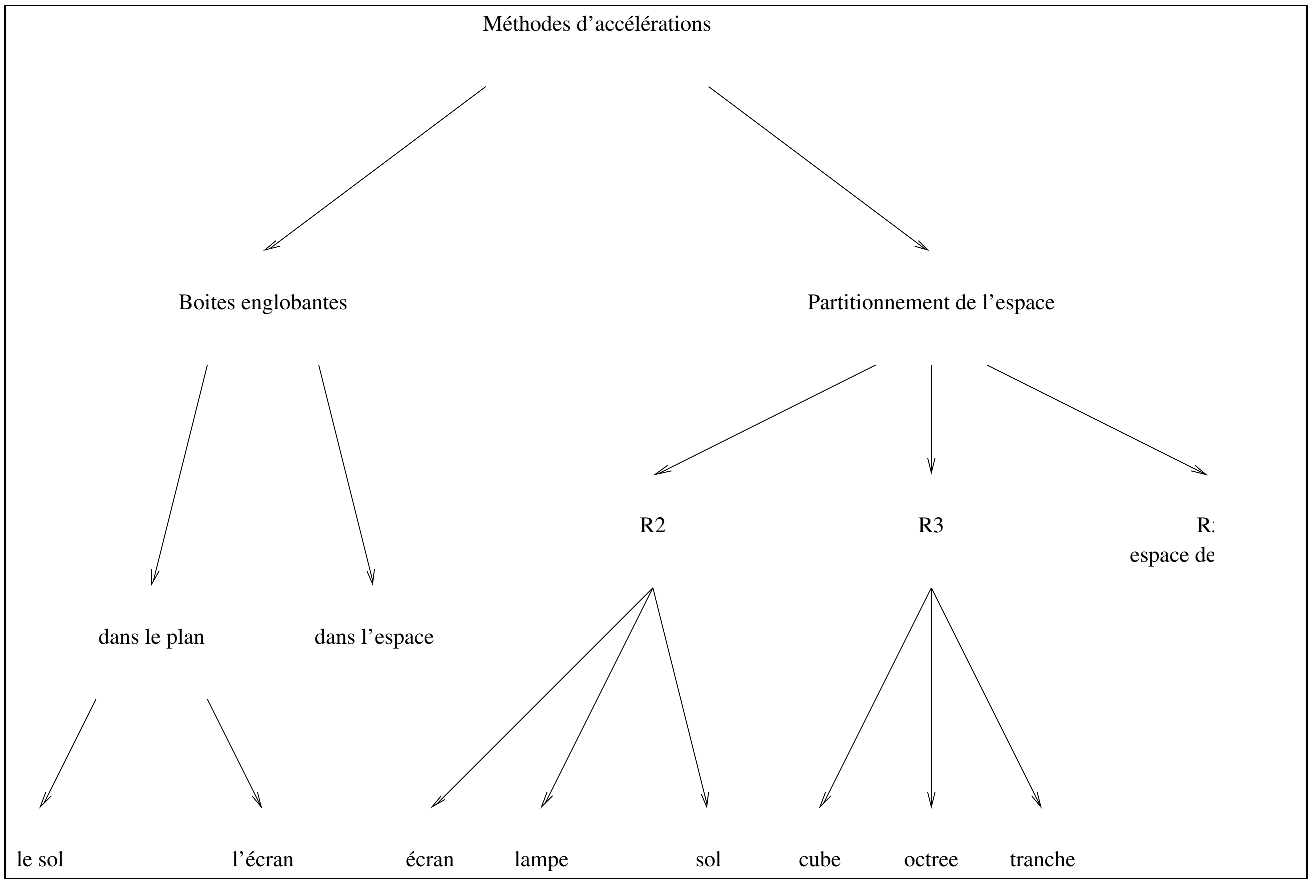


FIG. 25 – *Un classement des différentes méthodes d'accélérations*

Comment juger de ces méthodes d'accélérations, leurs performances dépendent de nombreux paramètres.

- Le précalcul est à refaire pour un changement de position.
- La méthode supporte plus ou moins les rayons secondaires.
- Elle est plus ou moins adaptée à l'animation.
- Elle est plus ou moins performante suivant la disposition des objets.
- Elle est plus ou moins adaptée à certaines formes d'objets.
- Elle peut être parrallèlisée.
- Elle peut être utilisée avec un affichage progressif.
- Elle peut utiliser un hardware déjà existant.

7 Animation

L'animation est quelque chose de complexe et couteux. L'utilisation de l'informatique permet de réduire les coûts mais aussi de créer des animations trop complexes pour être faite à la main.

Les problèmes liés à l'animation sont nombreux et sujet à de nombreuses recherches.

- L'affichage de l'animation sur un écran.
- La sortie de l'animation sur périphériques.
- La modélisation de l'animation.

La modélisation d'une animation peut-être faite de nombreuses manières.

- Système descriptif (images clefs, ...)
- Système générateur (langage)
- Système "intelligent" (acteur)
- Couplage intéractif avec l'utilisateur.
- Couplage avec simulateur physique.

La principale difficulté de l'animation est le fait qu'il est extremement difficile d'imaginer ce que va donner une animation sans l'avoir "essayée"

La réalisation d'une animation est donc liée à la puissance de la machine, en effet il est impossible de mettre au point une animation si l'on ne peut pas voir immédiatement le résulat.

Souvent les systèmes d'animation dégradent la qualité de l'affichage pour améliorer la vitesse d'affichage.

L'apport de l'informatique permet d'accélérer les méthodes de travail classique (moins cher) et de faire des animations complexes qu'il aurait été impossible de faire à la main. La technique de travail habituelle comportait les étapes suivantes :

- Le SCENARIO lieux, personnages, dialogue, plans, *story board*
- L'ANIMATEUR qui dessine les images clefs (1 à 2 images secondes).
- Les ASSISTANTS qui complètent les actions (6 à 10 images secondes).
- Le GOUACHEURS qui font le coloriage.
- Le PHOTOGRAPHE qui supperpose, compose et photographie

Cette technique est classique pour les dessins animées. Mais de nombreuses autres techniques existent :

- La pate à modeler.
- Le papier.
- Le objets courants y compris le corps humain (pixelisation) et poupées.
- Le tableau d'épingles.
- Le dessin sur pellicule (et bande son).
- ...

7.1 Affichage d'une animation

Pour l'affichage d'une animation, il faut que l'oeil ne puisse pas voir de mouvement saccadé. Il ne doit pas différencier les images, pour cela il est nécessaire d'avoir au moins 20 images par seconde (25 en france et 30 aux USA). En fait l'oeil est sensible à des vibrations de plus haute fréquence, jusqu'a environ 70Hz mais cela dépend des personnes.

Dans la réalité l'oeil moyenne l'intensité lumineuse dans le temps. Lorsqu'on fait une animation on échantillonne des images. Cela fait apparaître des problèmes de sous-échantillonnages. Il peut-être nécessaire de faire de l'antialiassing temporel.

La réalisation d'une animation différée ne pose pas de problèmes, on calcul l'image avec le temps que l'on veut et on stocke l'image résultat sur la pellicule ou sur disque.

L'animation temps réelle sur l'écran est beaucoup complexe à réaliser techniquement. Elle pose plusieurs problèmes :

- La vitesse d'affichage qui doit être inférieur au 1/25 de seconde.
- Le processus qui doit être temps réel (cela ne doit pas coincer).
- L'utilisateur ne doit pas voir la prochaine image s'afficher.

Ces problèmes sont résolus et utilisant des cartes graphiques rapides et un système d'exploitation temps réel de préférence mono-utilisateur. Le problème du calcul de l'image suivante alors q'une autre image est affichée est un problème de *double buffering*

Les cartes graphiques peuvent posséder directement le Z-buffer ou bien il faut le simuler par différents moyens.

7.1.1 Double buffering