VIII.
Images 2D

Laboratoire des Sciences de l'Information et des Systèmes
UMR CNRS 6168

1. Projection

gluOrtho2D(*xmin, xmax, ymin, ymax*)

Exemple :

```
void monCadrage(int large, int haut)
{ glViewport(0, 0, large, haut) ;
  glMatrixMode(GL_PROJECTION) ;
  glLoadIdentity() ;
  gluPerspective(90, 16./9., 5, 20) ;
  gluOrtho2D(0, large, 0, haut) ;
  glMatrixMode(GL_MODELVIEW) ;
}
```

2. Format d'une image

Format d'un fichier pgm :

- image en niveaux de gris
- entête au format texte
- pixel représenté par un octet
- image stockée ligne par ligne

```
P5
# commentaires
hauteur largeur
# commentaires
maxValue
OctetOctetOctet...
```

Format d'un fichier ppm :

- image couleur
- entête au format texte
- pixel RGB représenté par 3 octets
- image stockée ligne par ligne

```
P6
# commentaires
hauteur largeur
RGBRGBRGNRGB...
```

3. Affichage d'une image

Position dans la fenêtre du coin inférieur gauche *(x0, y0)* de l'image :

glRasterPos2i(*x0, y0*) ;

OpenGL propose trois commandes de base pour manipuler des images :

- `glDrawPixels` tableau -> tampon image
- `glReadPixels` partie du tampon image -> tableau
- `glCopyPixels` recopie une zone à l'intérieur du tampon image

glDrawPixels(*largeur, hauteur, format, type, tableau*)
glReadPixels(*largeur, hauteur, format, type, tableau*)

largeur, hauteur : taille de l'image en terme de pixels,
format : **GL_RGB** ou **GL_LUMINANCE**,
type : on se limitera à **GL_UNSIGNED_BYTE**
tableau : adresse du tableau respectant le type

4. Primitives graphiques 2D

`glVertex2i(x, y)`

Algorithme du peintre :

1. affichage de l'image
2. tracé par dessus l'image

5. Image et processeur

`glDrawPixels` et `glReadPixels` transfèrent des mots machines …

-> risque de décalage des lignes de l'image à l'affichage (si le nbre d'octets d'une ligne n'est pas un multiple d'un mot machine)

Solutions :

- définir des images de largeur multiple du mot machine
- contrôler les transferts avec :

`glPixelStorei(GL_UNPACK_ALIGNMENT,`*`taille`*`)pour glDrawPixels`
`glPixelStorei(GL_PACK_ALIGNMENT,`*`taille`*`)  pour glReadPixels`

Moins rapide mais sûr :

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glPixelStorei(GL_PACK_ALIGNMENT, 1);
```

VIII.
Algorithmes fondamentaux
(quelques compléments)

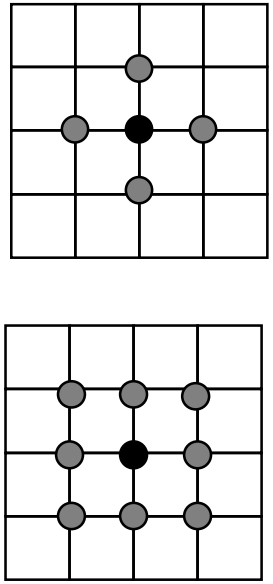

Laboratoire des Sciences de l'Information et des Systèmes
UMR CNRS 6168

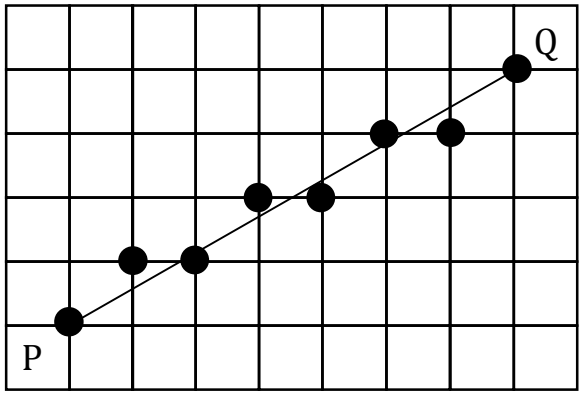
1. Génération de vecteurs

Plusieurs effets visuels peuvent être envisagés :

- **aspect le plus fin possible (« épaisseur » de 1 pixel),**
- chercher à rendre une épaisseur,
- style de trait (pointillé, motifs, …),
- limiter l'effet « marches d'escalier » (anti-aliasing, …)

4 et 8-connexité





Représentation discrète d'une droite « fine »

2. Algorithme incrémental de base

Discrétisation de la droite $y = a\,x + b$ à partir du point (x_0 , y_0) :

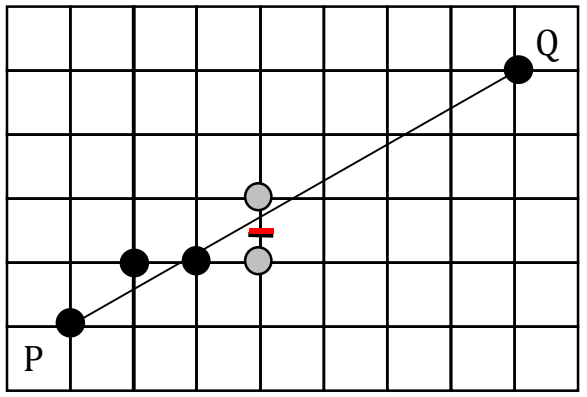
$x_{i+1} = x_i + 1$ et $y_{i+1} = y_i + a$ en coordonnées réelles

Remarque : une pente >1 génère des trous

```
void traceLigne(int x0, int y0, int xn, int yn)
{ /* Pour le 1er octant */
  int x;
  float y, a;
  x = x0; y = y0;
  a = (yn-y0)/(xn-x0);
  while(x <= xn)
  { AffPixel (x,int(y+0.5));
    x++; y += a;
  }
}
```

3. Optimisation : algorithme de Bresenham

```
void traceLigne(int x0, int y0, int xn, int yn)
{ int x, y;
  float a, d;
  a = (yn-y0)/(float)(xn-x0);
  x=x0; y=y0;
  d=0; /* partie décimale de y */
  while(x <= xn)
  { AffPixel(x,y);
    x++
    d += a;
    if (d>0.5)
    { d--;
      y++;
    }
  }
}
```



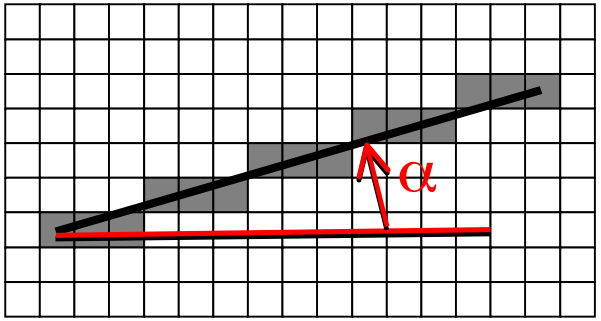
Pour calculer en entier : on multiplie `a` par 2*(xn-x0)

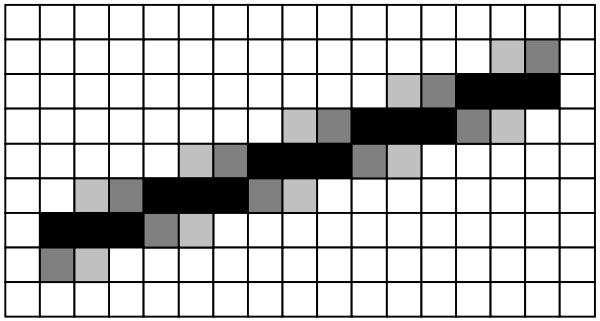
```
void traceLigne(int x0, int y0, int xn, int yn)
{ int x, y;
  int   a, d, demiPt, dx;
  a = (yn-y0)*2;
  x=x0; y=y0; demiPt= xn-x0; delta = 2*demiPt;
  d=0;
  while(x <= xn)
  { AffPixel(x,y);
    x++
    d += a;
    if (d>demiPt)           // if (d>0.5)
    { d -= delta ;          // d--;
      y++;
    }
  }
}
```

Remarque : L'intensité du segment diminue près des diagonales.

Deux solution pour compenser :

- ➔ intensité lumineuse des points proportionnelle à $1/\cos(\alpha)$
- ➔ **anti-aliasing** (trait « d'épaisseur »).



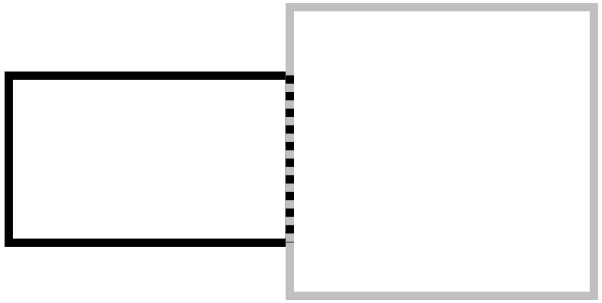


4. Remplissage de polygones

Deux problèmes :

- Décision quant aux pixels qu'il faut modifier
- Valeur à affecter (couleur de remplissage)

Exemple élémentaire : partage d'une arête



Solution arbitraire partielle :

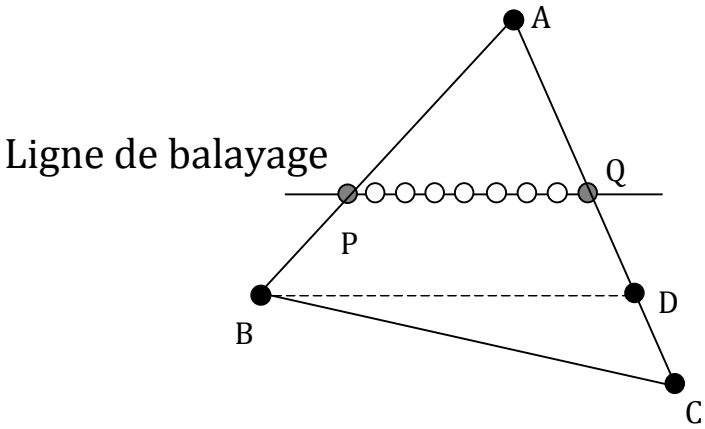
les contours supérieur et droit
ne font pas partie de la primitive.

Que faire pour les rotations ?

Remplissage d'un triangle par ligne de balayage horizontale :

Remplir(A, B, C) :

- découper en deux triangles avec un côté horizontal (A, B, D) et (B, C, D)
- tracer(A, B, D)
- tracer(B, C, D)



tracer(A, B, D) :

- avec Bresenham, générer les sommets P et Q des arêtes AB et AC
- remplir le segment horizontal (P, Q)

Remplissage d'un polygone par ligne de balayage horizontale.

La décision de coloriage est prise pour chaque pixel en fonction du nombre d'arêtes qui ont été coupées.

Algorithme :

- stocker dans une table les intersections avec une ligne de balayage,
- ordonner cette table dans l'ordre croissant des numéros de colonne,
- il y a un nombre pair d'extrémités,
- grouper les extrémités par paires,
- pour chaque paire, tracer le segment horizontal reliant ces deux extrémités.
- considérer toutes les arêtes du polygone,