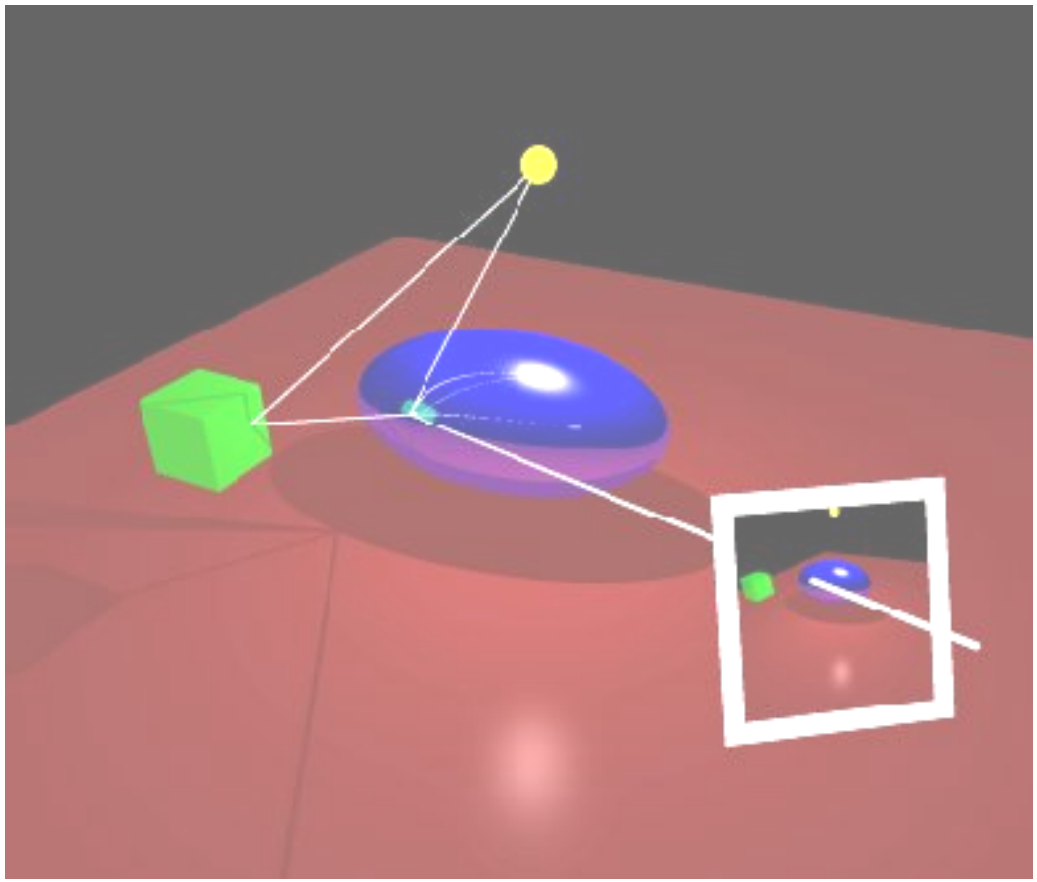
IFT3355: Infographie
Imagerie 2D

© Pierre Poulin, Derek Nowrouzezahrai

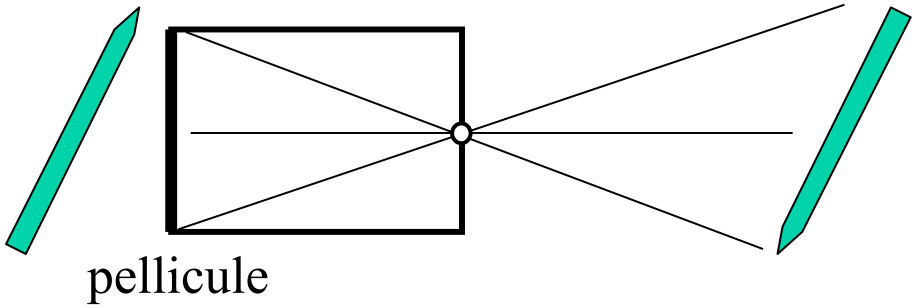
Dép. I.R.O.

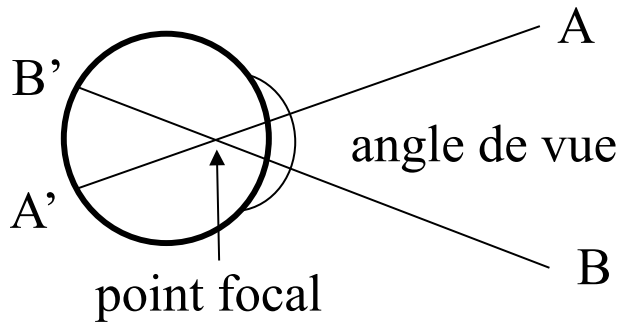
Université de Montréal

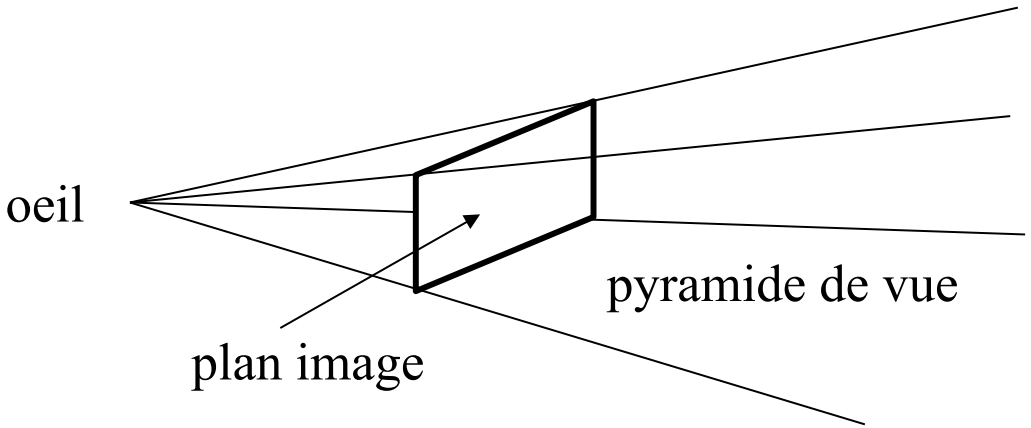
Scène 3D vs. Image 2D



Caméra obscure (*Pinhole Camera*)

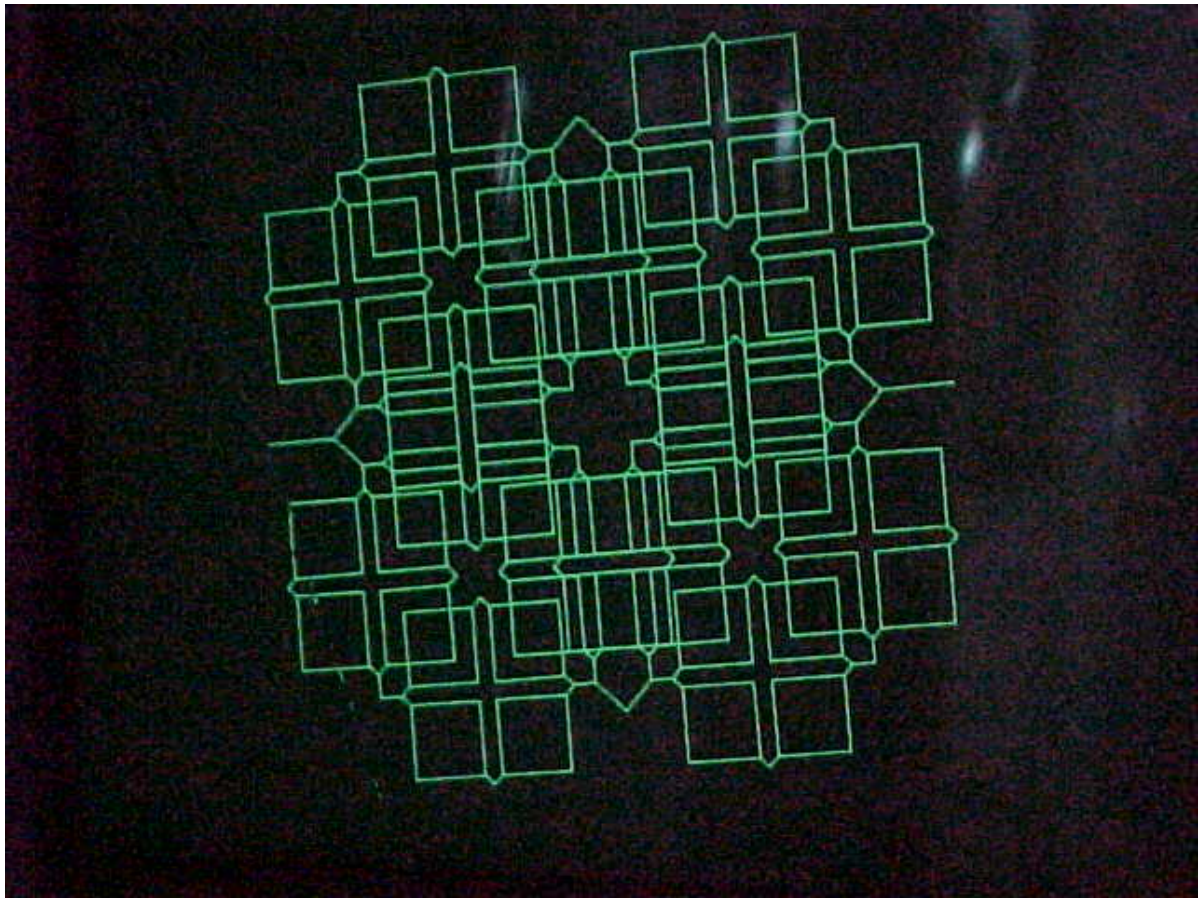






Vector Graphics

Tektronix
Evans & Sutherland

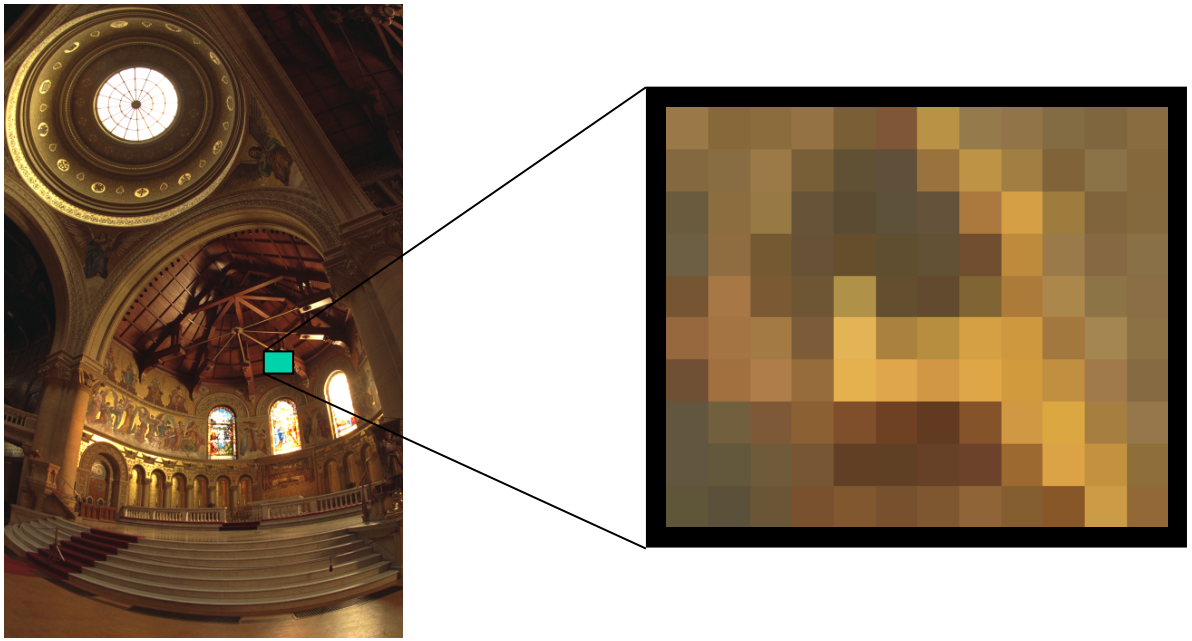


Vol de rêve, 1982

www.cca.org/vector

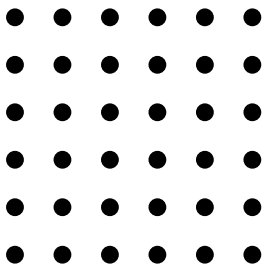
Représentations d'une image (*raster*)

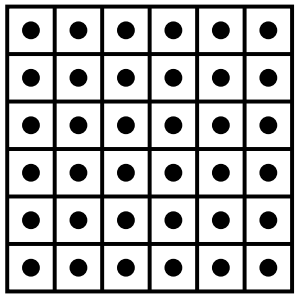
- Signal couleur continu sur un espace 2D
- Ensemble de pixels avec leurs couleurs respectives
- Ensemble de primitives transformées en un ensemble de pixels avec leurs couleurs respectives

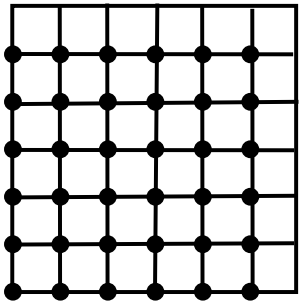


Pixel (*picture element*)

- Quadrillage
  - approximation à des positions discrètes
  - aliassages sous forme d'*escaliers* et de *Moirés*
  - aliassage peut être *réduit* en utilisant des intensités variables en fonction du recouvrement des pixels
- Tableau rectangulaire de points

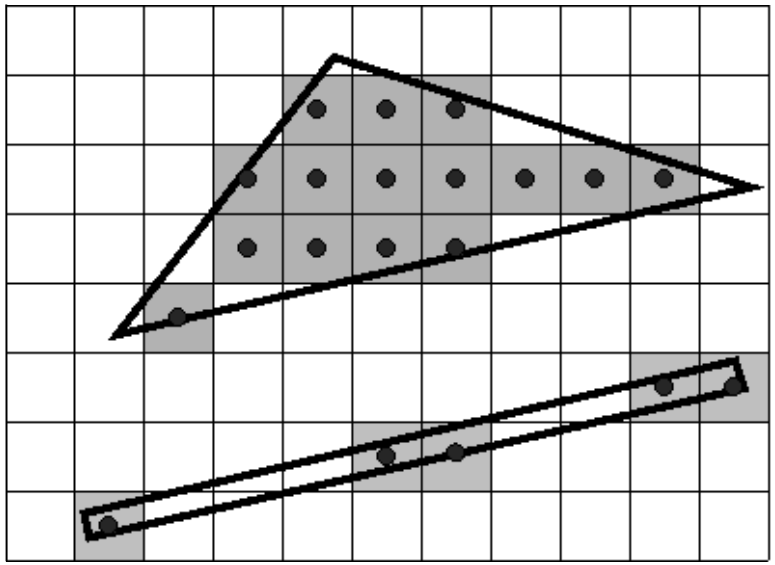




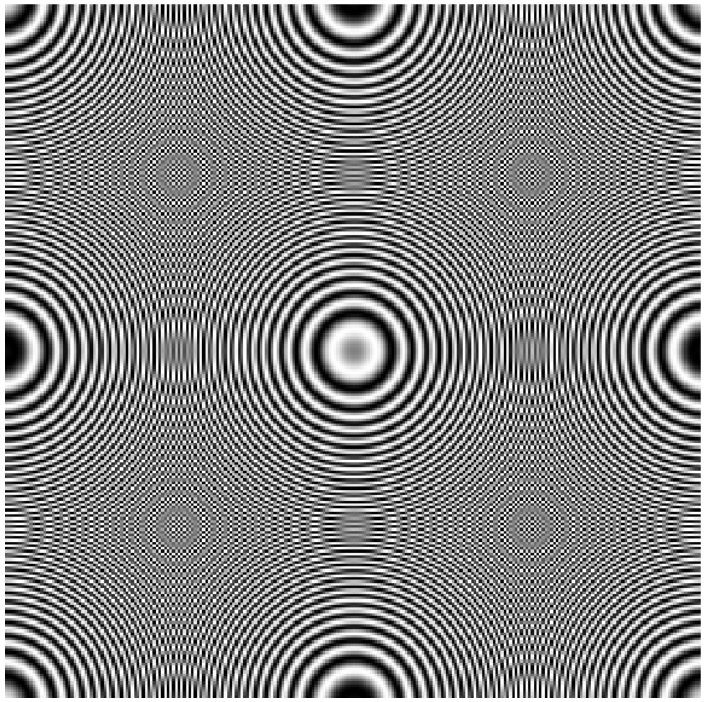


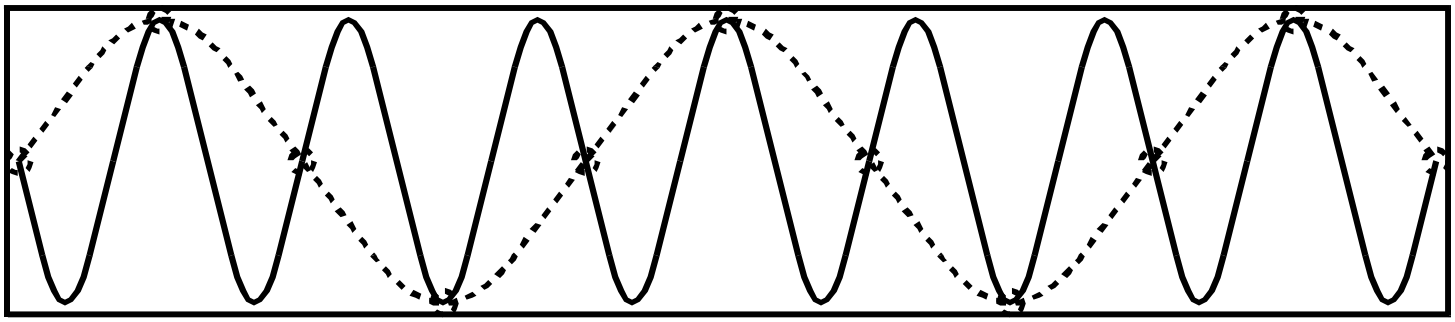
Aliassage

escaliers

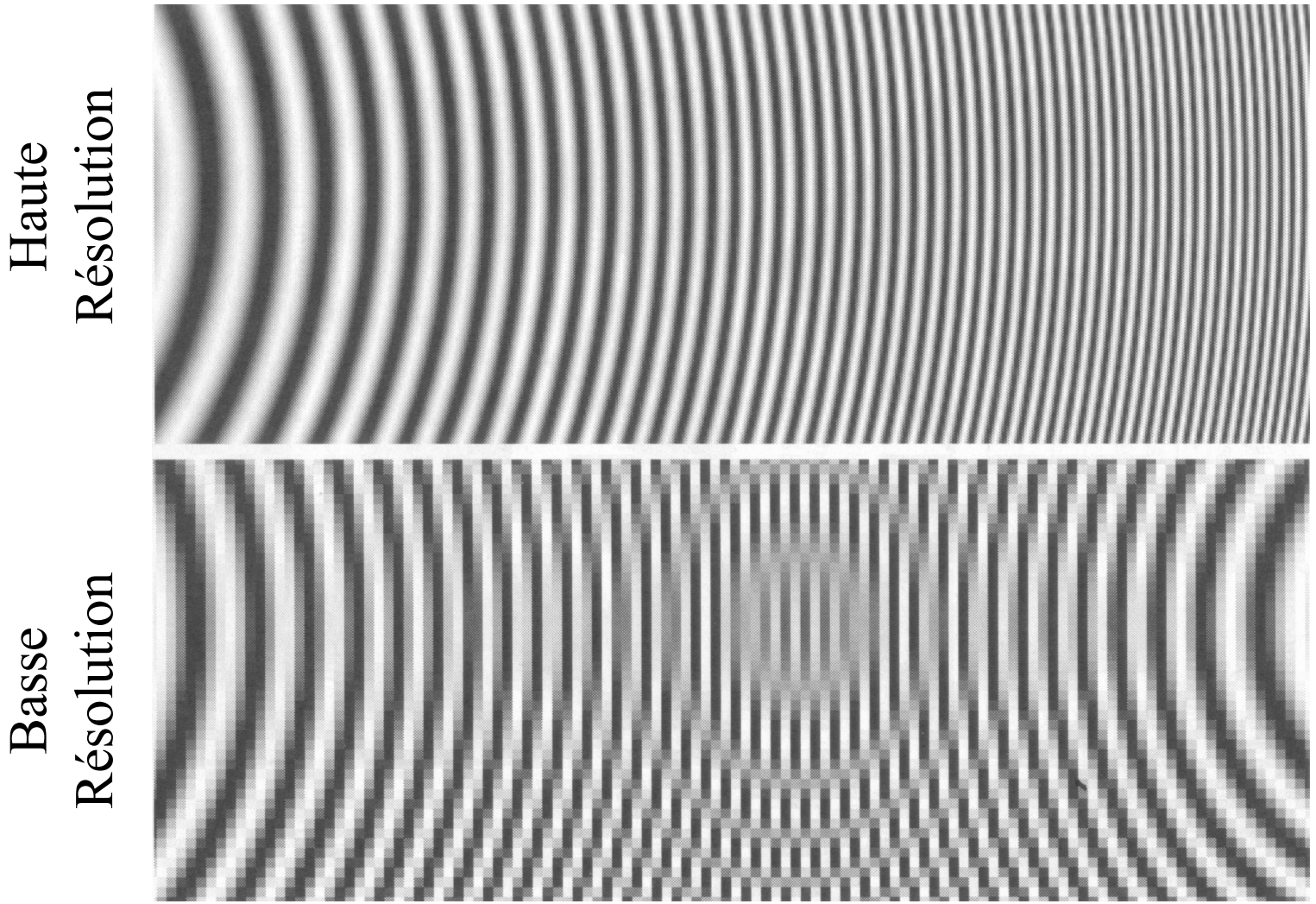


Moirés





Aliassage



Pixels sur un moniteur (CRT)

Deux pixels blancs séparés par 8 pixels noirs
Ecran Sony Trinitron 17”

Note: sur un LCD, le pixel est proche d’un filtre carré

Quadrillage (*raster graphics*)

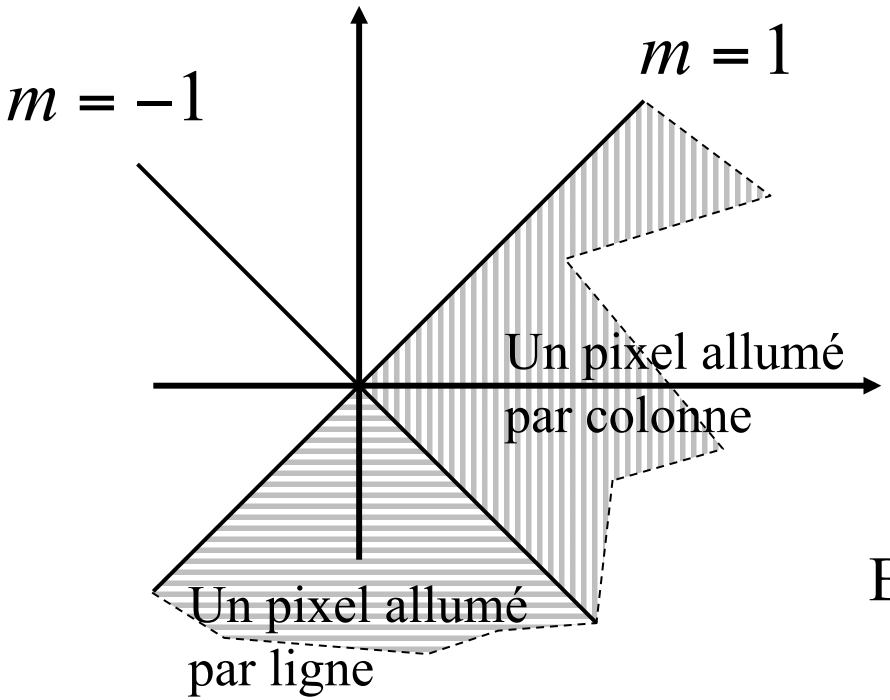
- Balayage ou conversion ponctuelle (*scan-conversion*)
  - allume les pixels couverts par la primitive
  - visuellement satisfaisant à haute résolution
  - rapide grâce aux méthodes incrémentales et au parallélisme

- Clippage ou Fenêtrage
  - les pixels couverts par la primitive mais hors de la fenêtre ne sont pas affichés

Lignes 2D

- Pixels noirs ou blancs:

  on allume les pixels sur ou les plus près d'une ligne infiniment étroite



$$y = mx + b$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - mx_1$$

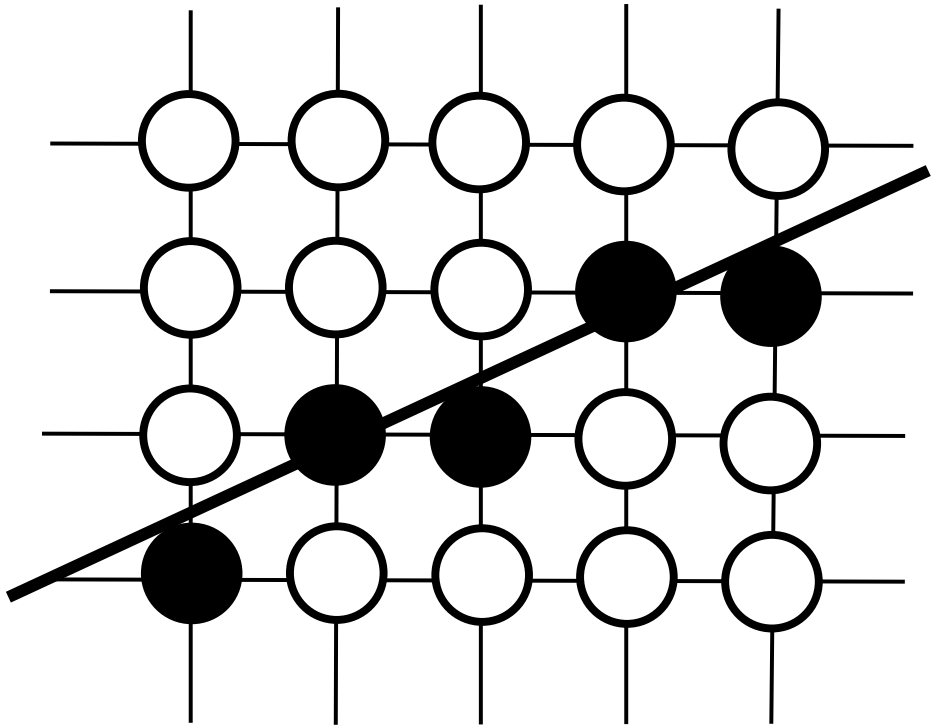
intersection axe Y

Beaucoup de symétrie

$$\pm x \leftrightarrow \pm y$$

Lignes 2D

- Supposition: -1 < m < 1 (incrément en x)
- Sinon, inverser x et y



DDA (*Digital Differential Analyzer*)

(1) $y = mx + b \quad m < 1 \Rightarrow$ incrémente x

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

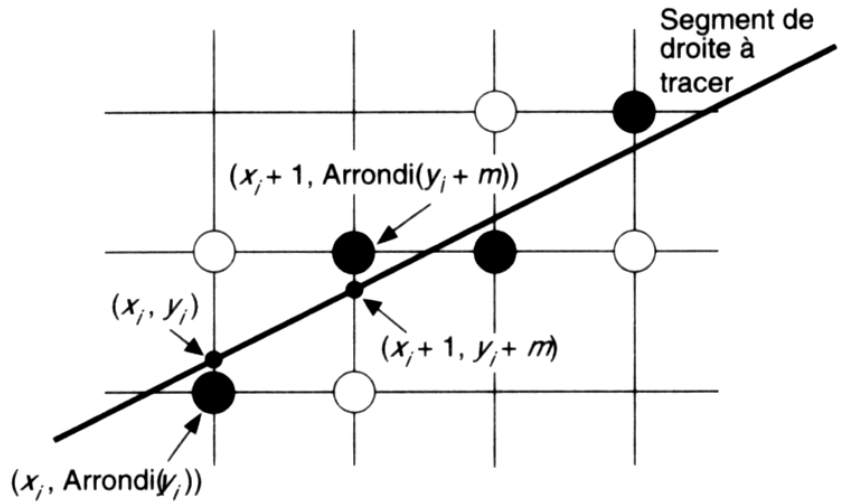
$$y_{i+1} = mx_{i+1} + b$$
$$= m(x_i + \Delta x) + b$$
$$= (mx_i + b) + m\Delta x$$
$$= y_i + m\Delta x \quad (\Delta x = 1)$$
$$= y_i + m$$

Allume :

$(x_i, round(y_i))$ ou $(x_i, trunc(y_i + 0.5))$

$(x_i + 1, round(y_i + m))$

(2) Si $m > 1$ alors $x_{i+1} = x_i + \dfrac{1}{m}$



- *round*()
- y et m doivent être des réels

Algorithme du DDA

```
void DDALine (int x0, int y0, int x1, int y1, int color)
//  suppose –1 <= m <= 1,   x0 < x1
{
   int x;                  // x est incrémenté de x0 à x1 par pas d'une unité
   float y, dx, dy, m;
   dx = x1 – x0;    dy = y1 – y0;
   m = dy / dx;
   y = y0;
   for (x = x0; x <= x1; x++)  {
      WritePixel (x, (int) floor (y + 0.5), color); // pixel plus proche
       y += m;       // avance y d'un pas de m
   }
}
```

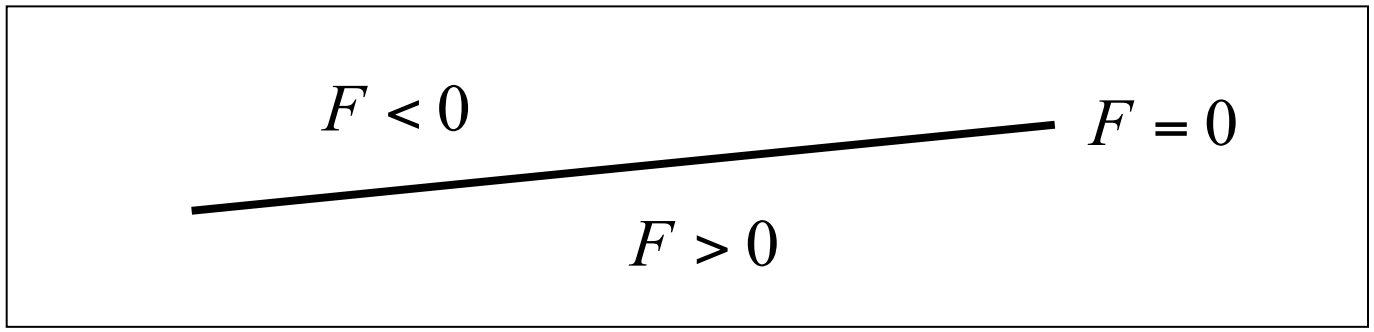
Bresenham / Point milieu

$$y = mx + b = \frac{y_2 - y_1}{x_2 - x_1} x + b = \frac{dy}{dx} x + b$$

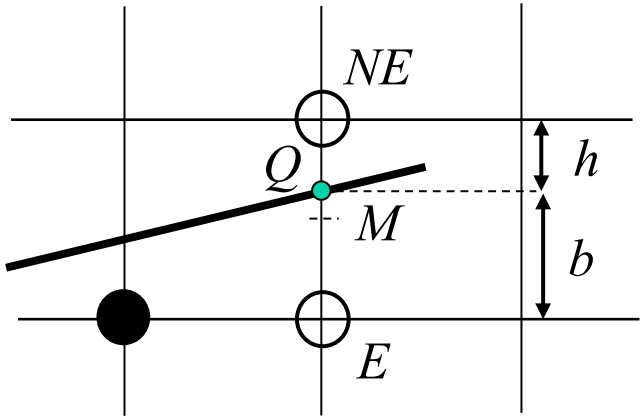
$$\frac{dy}{dx} x - y + b = 0$$

$$dy \cdot x - dx \cdot y + dx \cdot b = 0$$

$$F(x, y) = Ax + By + C = 0 \qquad \text{Forme implicite}$$



Bresenham / Point milieu



Bresenham	Point milieu
if ($h - b < 0$)	if ($F(M) > 0$)
NE	NE
else	else
E	E

- Signe de *F(M)* détermine si on choisit NE ou E
- Pas de *round()*
- Arithmétique entière

Bresenham / Point milieu

$$F_1(M_i) = dy(x_p + 1) - dx(y_p + 0.5) + b \cdot dx$$

Si E : $F_2(M_{i+1}) = dy(x_p + 2) - dx(y_p + 0.5) + b \cdot dx$

$$\Delta E = F_2(M_{i+1}) - F_1(M_i) = dy$$

Si NE : $F_2(M_{i+1}) = dy(x_p + 2) - dx(y_p + 1.5) + b \cdot dx$

$$\Delta NE = F_2(M_{i+1}) - F_1(M_i) = dy - dx$$

Bresenham / Point milieu

Initialisation

$$F_0(x_0, y_0) = dy(x_0) - dx(y_0) + b \cdot dx = 0$$
$$F_1(M_i) = dy(x_0 + 1) - dx(y_0 + 0.5) + b \cdot dx$$
$$= F_0(x_0, y_0) + dy - 0.5dx$$

Comme on ne regarde que le signe, *signe(F(M)) = signe(2F(M))*

$$F(M) = 2dy - dx$$
$$\Delta E = 2dy$$
$$\Delta NE = 2(dy - dx)$$

Algorithme de Bresenham / Point milieu

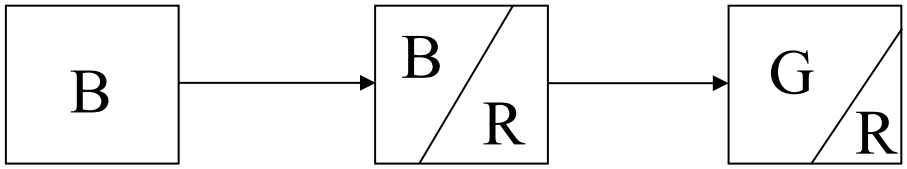
```
void MidpointLine (int x0, int y0, int x1, int y1, int color)
{
   int dx, dy, incrE, incrNE, x, y, d;   // d: variable de décision

   dx = x1 – x0;    dy = y1 – y0;
   d = 2 * dy – dx;                      // valeur initiale de d
   incrE = 2 * dy;                       // incrément en cas E
   incrNE = 2 * (dy - dx);               // incrément en cas NE
   x = x0;  y = y0;
   WritePixel (x, y, color);             // pixel initial
   while (x < x1)  {
      if (d <= 0) {                      // choix E
         d += incrE;   x++;
      } else {                           // choix NE
          d += incrNE;   x++; y++;
      }
      WritePixel (x, y, color);          // pixel plus proche
   }
}
```

Bresenham / Point milieu

- Inverser l'ordre des sommets
  - Surveiller lorsque $F(M)=0$
  - Ré-ordonner les sommets
    - Difficultés avec les lignes pointillées
- Sommet hors de la fenêtre
  - Côté vertical
    - Même calcul excepté pour l'initialisation
    - Attention de ne pas changer la pente (en changeant l'origine)
  - Côté horizontal
    - Intersecte avec $y=y-0.5$ au lieu de y

Bresenham / Point milieu

- Changement d'intensité avec la pente *m*
  - Espacement différent entre les pixels
    - Antialiassage / recouvrement
- Lignes multiples
  - Si combine avec la couleur précédemment dans le pixel, attention aux sommets de segments joints (deux fois le même pixel)
  - Danger du *compositing alpha*



Mais le pixel conservera une portion de B…

Cercles

$$x^2 + y^2 = r^2$$

$$y = \pm\sqrt{r^2 - x^2}$$

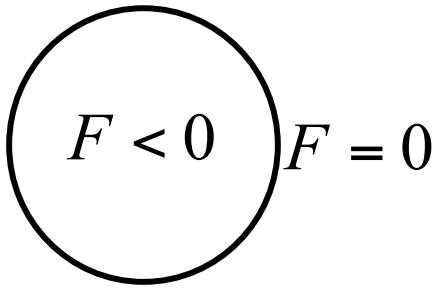
$$r\cos\theta$$

$$x : 0..r$$

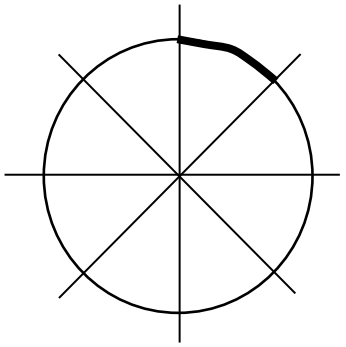
$$\theta : 0..\frac{\pi}{2}$$

Inefficace

$$F(x, y) = x^2 + y^2 - r^2$$

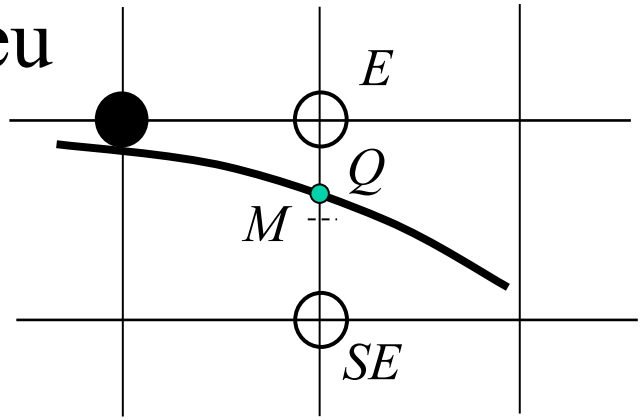


8-symétrie



$$\pm x \leftrightarrow \pm y$$

Cercles: Bresenham / Point milieu



$$F_1(M_i) = (x_p + 1)^2 + (y_p - 0.5)^2 - r^2$$

Si E : $F_2(M_{i+1}) = (x_p + 2)^2 + (y_p - 0.5)^2 - r^2$

$$\Delta E = 2x_p + 3$$

Si SE : $F_2(M_{i+1}) = (x_p + 2)^2 + (y_p - 1.5)^2 - r^2$

$$\Delta SE = 2x_p - 2y_p + 5$$

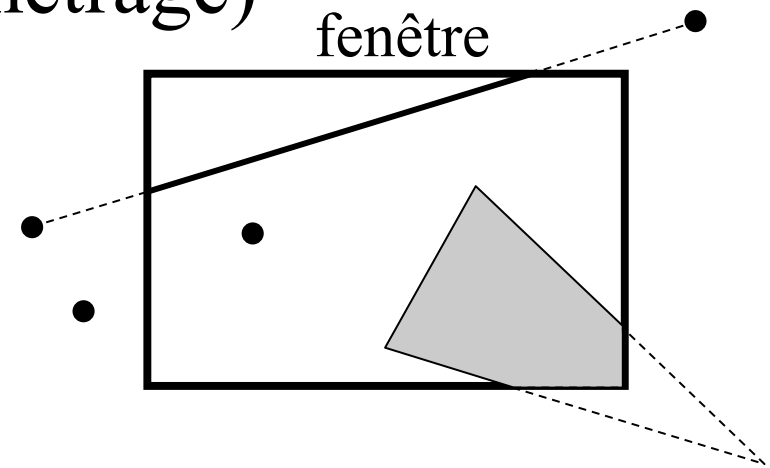
origine $(0, r)$

$$F_1(M) = (0+1)^2 + (r - 0.5)^2 - r^2 = 1.25 - r$$

Parallélisme

- Divise un segment en *n* segments
- Assigne un processeur par pixel qui calcule la distance à la ligne
  - Optimisations: boîte englobante, ligne/colonne de pixels

Clippage (Fenêtrage)



- Point: $x_{\min} \le x \le x_{\max}$ et $y_{\min} \le y \le y_{\max}$
- Segment:

  compare les deux sommets avec la fenêtre rectangulaire

  – intérieurs: affiche tout le segment

  – intérieur-extérieur: calcule l'intersection et affiche

  – extérieurs: teste/calcule pour les intersections et affiche

Clippage de segments - Cohen-Sutherland

- Si les deux sommets sont à l'intérieur

  => acceptation triviale

- Si les deux sommets respectent la condition

  $x_{1\wedge 2} < x_{\min}$ ou $x_{1\wedge 2} > x_{\max}$ ou $y_{1\wedge 2} < y_{\min}$ ou $y_{1\wedge 2} > y_{\max}$

  => rejet trivial

- Sinon, divise en deux segments avec la ligne de support d'un côté de la fenêtre et teste chaque sous-segment récursivement

+ grosse fenêtre: beaucoup d'acceptations triviales

+ petite fenêtre: beaucoup de rejets triviaux

Clippage de segments - Cohen-Sutherland

Code 4 bits $T_{\text{op}}B_{\text{ottom}}R_{\text{ight}}L_{\text{eft}}$ 0:intérieur 1:extérieur

	1001	1000	1010
T	0001	0000	0010
B	0101	0100	0110
	L		R

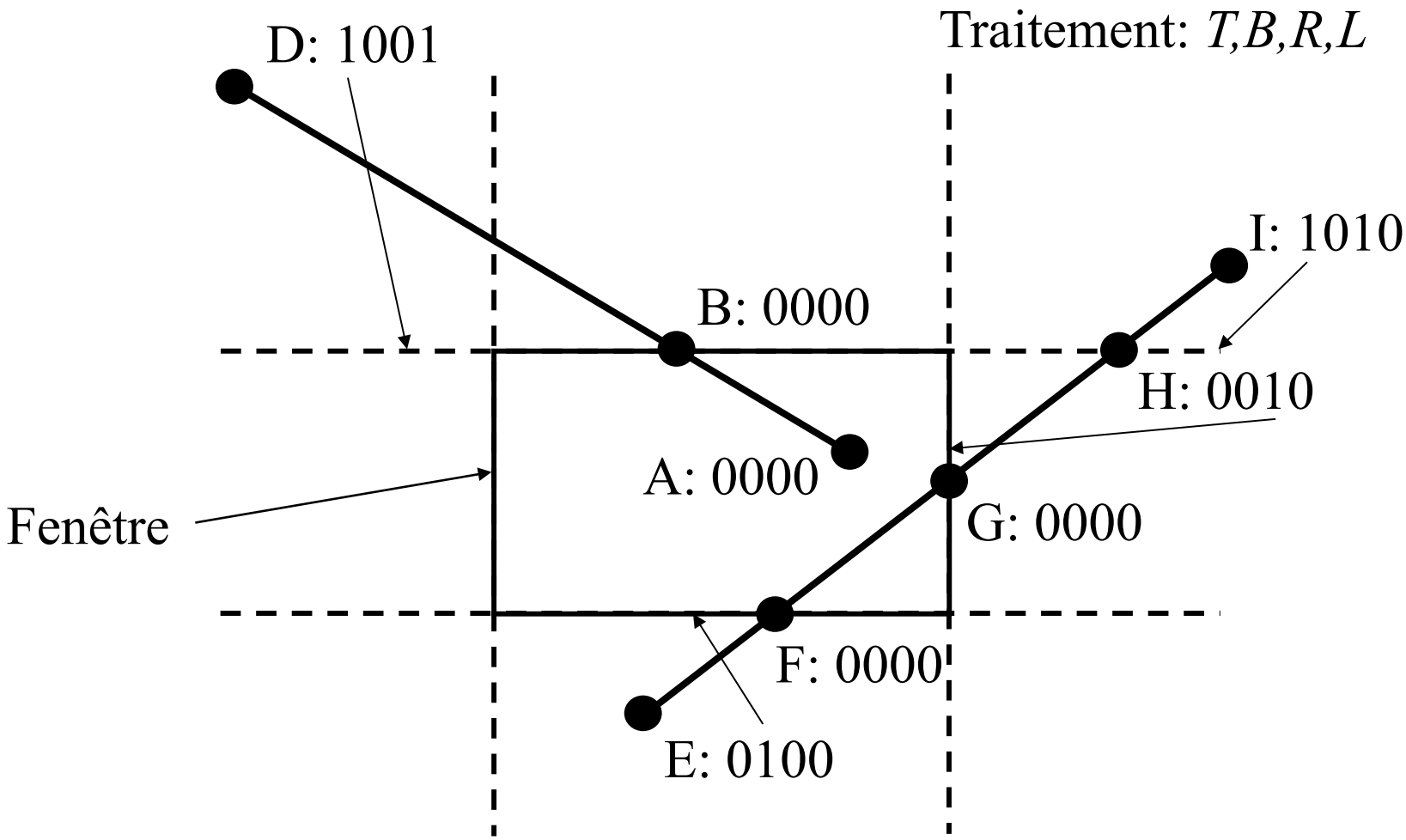
T: bit signe($y_{\max} - y$)
B: bit signe($y - y_{\min}$)
R: bit signe($x_{\max} - x$)
L: bit signe($x - x_{\min}$)

- Deux intérieurs: $(T_1B_1R_1L_1 = 0000)$ et $(T_2B_2R_2L_2 = 0000)$
  => acceptation triviale
- Deux extérieurs: $(T_1B_1R_1L_1$ et_logique $T_2B_2R_2L_2 \neq 0000)$
  => rejet trivial

Clippage de segments - Cohen-Sutherland

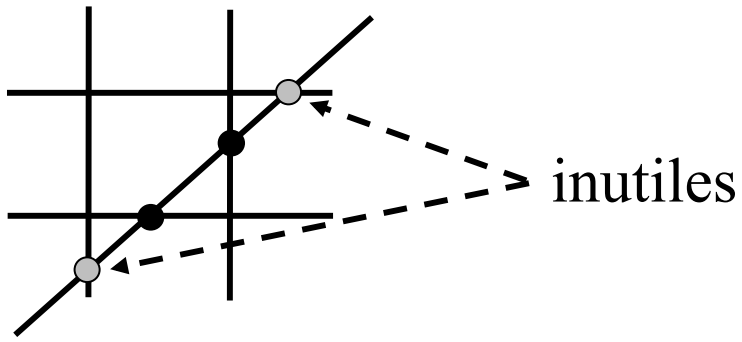
- Sinon non-trivial: résultat donne T,B,R ou $L = 1$
  - intersecte le segment avec le côté de ce bit
  - rejette la partie extérieure du segment
  - remplace ce sommet par l'intersection
  - reteste récursivement avec le segment tronqué

Clippage de segments - Cohen-Sutherland



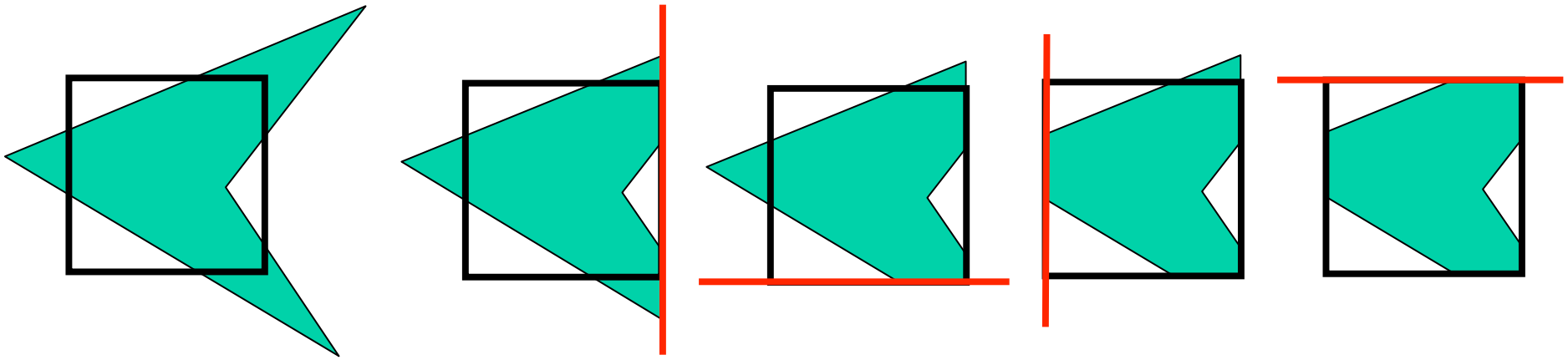
Clippage de segments - Cohen-Sutherland

- Algorithme de clippage de segments le plus utilisé
+ Extension triviale en 3D (volume orthographique)
+ Efficace en assembleur, surtout si beaucoup de cas triviaux
- Pour tout code à 1, les intersections sont calculées dans n'importe quel ordre (solution: Liang-Barsky)

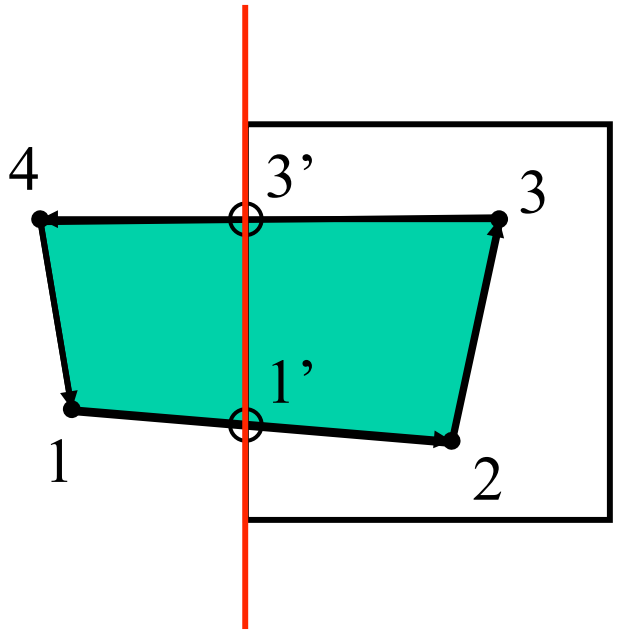


Clippage de polygones: Sutherland-Hodgeman

- Prolonge à l'infini un côté de la fenêtre
- Traverse les sommets du polygone en ordre anti-horaire et découpe chaque segment selon le côté de la fenêtre choisi
- Recommence avec la nouvelle liste de sommets et un autre côté de la fenêtre



Clippage de polygones: Sutherland-Hodgeman



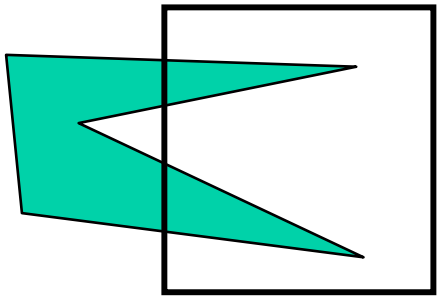
1..2: out-in => 1', 2
2..3: in-in => 3
3..4: in-out => 3'
4..1: out-out => aucun

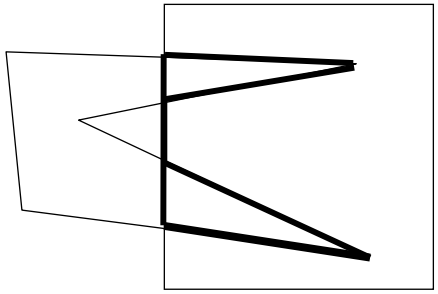
Liste clippée: 1', 2, 3, 3'

Parcours en sens anti-horaire

Clippage de polygones: Sutherland-Hodgeman

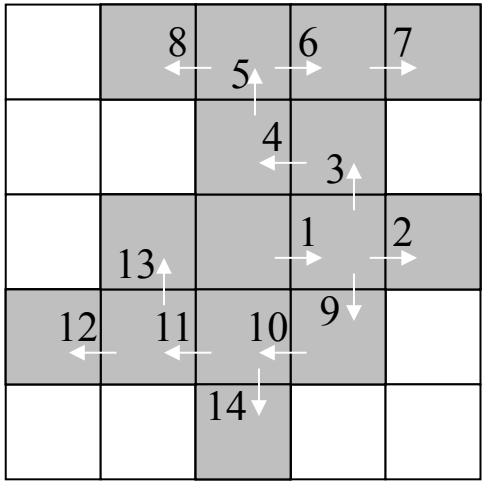
+ fenêtre peut être généralisée pour une fenêtre convexe
+ algorithme extensible en 3D pour un polyèdre convexe
- un polygone concave reste connecté (corrigé par post-traitement)





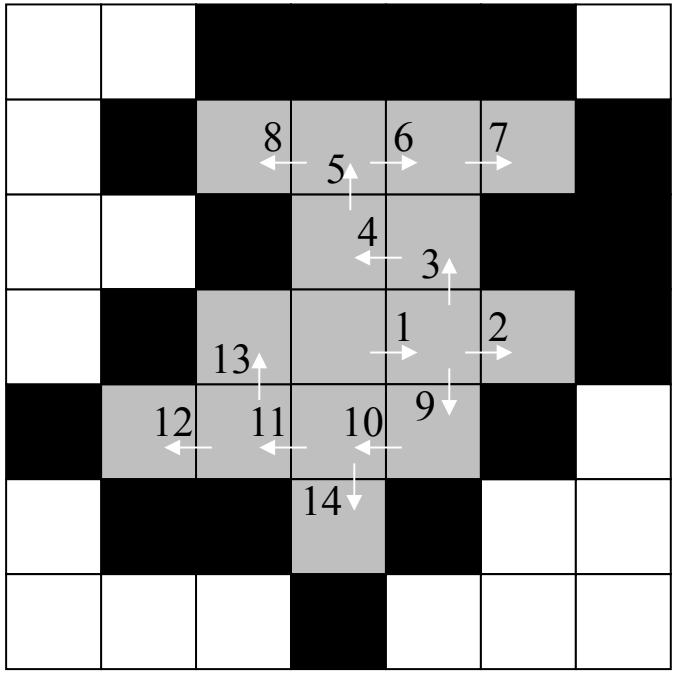
Remplissage - Algorithmes à germe ou par innondation

```
FloodFill( int x, int y )
{
   if couleur( x, y ) à changer {
      set( x, y );
      FloodFill( x+1, y );
      FloodFill( x-1, y );
      FloodFill( x, y+1 );
      FloodFill( x, y-1 );
   }
}
```

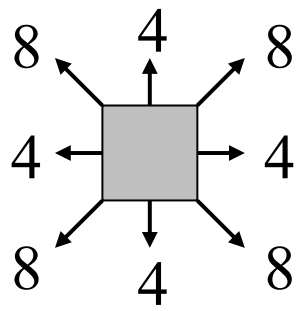


Remplissage - Algorithmes à germe

```
BoundaryFill( int x, int y )
{
   if (( x, y ) ≠  bordure et
      ( x, y ) unset) {
         set( x, y );
         BoundaryFill( x+1, y );
         BoundaryFill( x-1, y );
         BoundaryFill( x, y+1 );
         BoundaryFill( x, y-1 );
   }
}
```



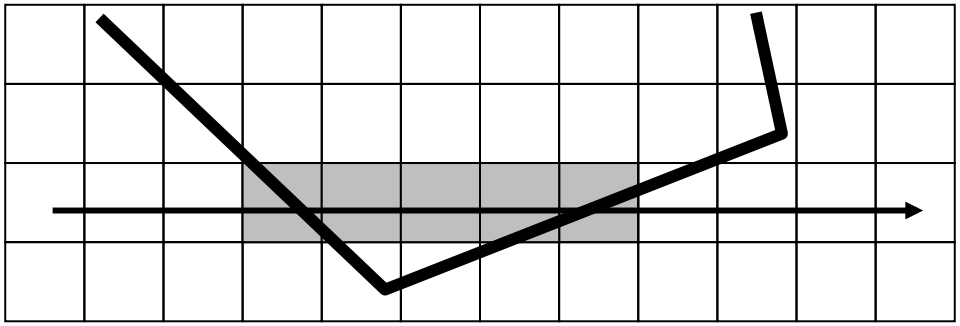
Remplissage - Algorithmes à germe



+ Très simple
- Hautement récursif (hautes piles); une version séquentielle existe mais elle est un peu plus compliquée
+ Très général (n'importe quelle forme)
- Requiert un point de départ à l'intérieur de la région (systèmes de peinture)

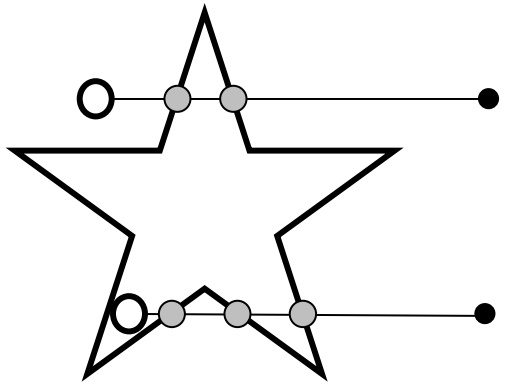
Remplissage - Algorithme par balayage

1. Calcule les points d'intersection entre la scanline et les côtés du polygone
2. Ordonne les points de gauche à droite
3. Allume les pixels entre les points d'intersection



Remplissage - Test intérieur-extérieur

- Trace une ligne du point jusqu'à un point à l'extérieur du polygone
- Si la ligne traverse un nombre impair de segments, le point est à l'intérieur; sinon il est à l'extérieur



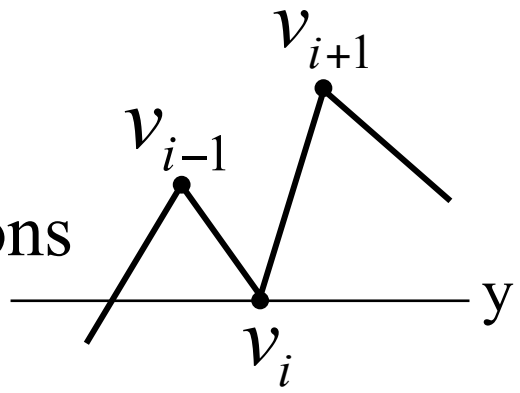
2 intersections => extérieur

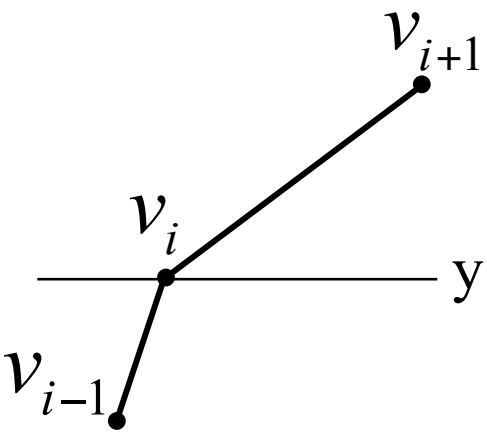
3 intersections => intérieur

Remplissage - Test intérieur-extérieur

Cas spécial: un sommet sur une scanline

$v_{i\text{-}1}$, v_{i+1} du même côté de la scanline y =>
v_i compte comme deux intersections





$v_{i\text{-}1}$, v_{i+1} de côtés différents de la scanline y =>
v_i compte comme une intersection

Cohérence de segment: $x_{i+1} = x_i + 1/m$