Algorithmes fondamentaux 2D

Christian Nguyen

Département d'informatique
Université de Toulon

Introduction

Lorsqu'une primitive graphique doit être affichée, elle subit un certain nombre de transformations :

1. elle est fenêtrée,
2. elle est projetée,
3. elle est discrétisée,
4. elle est "coloriée".

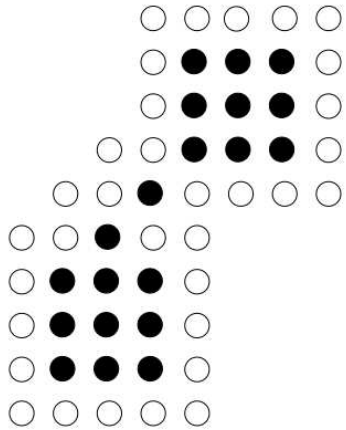
A chaque fois qu'une image est créée ou modifiée, ces algorithmes sont invoqués. Ils doivent donc être particulièrement efficaces.

Introduction
Objets manipulés

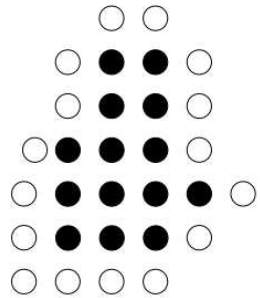
On se limite aux polygones : dans l'espace discret tout objet peut être approximé par un polygone.

Défini par une suite des sommets, suivant un sens de parcours trigonométrique ou horaire qui permet d'orienter la frontière du polygone.

Afin de pouvoir distinguer intérieur et extérieur d'un polygone, il faut préciser le type de connexité : *d* ou *i*.



région 8, frontière 4



région 4, frontière 8

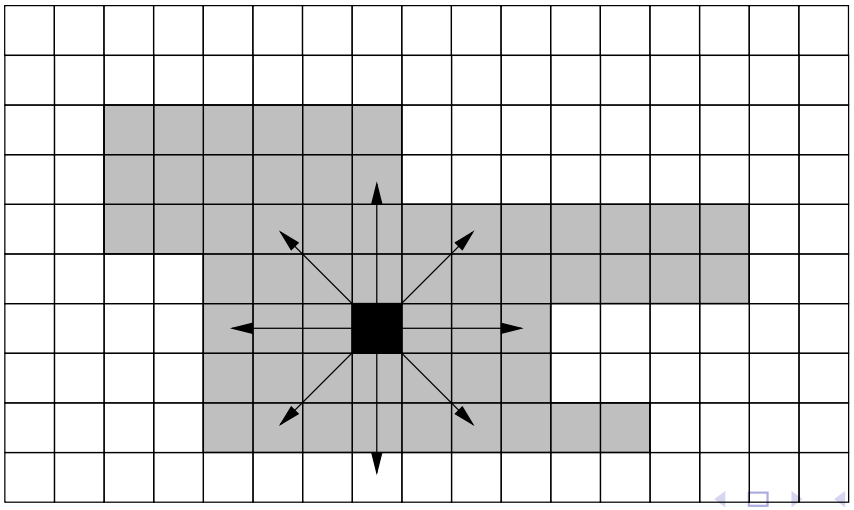
Plan

Remplissage
Dans l'espace discret

Zone définie par un ensemble de pixels.

Ces pixels définissent soit une surface (coloriage), soit un contour fermé (remplissage).

Coloriage : pixel initial (appelé *germe*) appartenant à la surface, puis application d'un algorithme récursif qui examine les 8 pixels voisins, détermine si la limite de zone est atteinte et applique la couleur choisie (*backtracking*).



Remplissage
Dans l'espace discret

Remplissage : germe, exploration des voisins droites et gauches du germe afin de parvenir au contour.

Une fois la frontière G-D déterminée, tracé d'un segment horizontal et poursuite de l'algorithme pour tous les pixels situés au dessus et en dessous de la ligne qui vient d'être tracée.

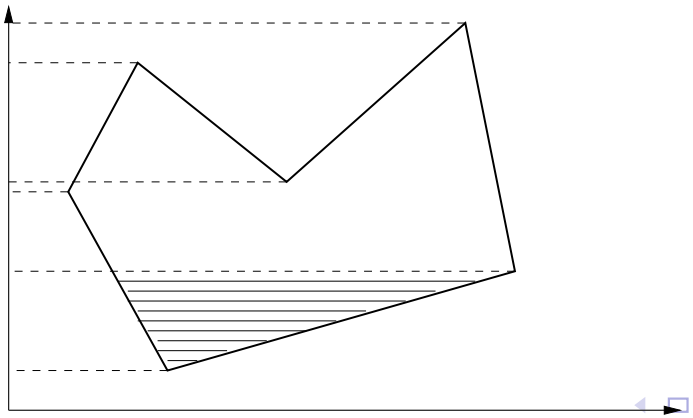
Le processus se termine lorsque tous les pixels examinés sont situés sur le contour.

Remplissage
Approche géométrique

Définition dans l'espace objet (polygones décrits par leurs sommets).

Calculs géométriques dans l'espace discret dans lesquels les objets sont représentés :

- parcours du polygone par une ligne de balayage (*scan-line*),
- calcul des intersections (de la ligne avec les arêtes du polygone) en arithmétique entière,
- incrémental : cohérence spatiale (d'arêtes) et temporelle (de lignes).



Remplissage
Approche géométrique

Algorithme pour une ligne de balayage :

1. trouver les intersections de la ligne de balayage avec les arêtes concernées,
2. trier les points d'intersection par abscisses croissantes ; cas particuliers :
   - coordonnées d'une intersection non entières : parité du nombre d'intersections
   - coordonnées d'une intersection entières mais sommet partagé par deux arêtes : pris en compte si c'est le sommet "bas" d'une arête,
   - sommets d'une arête horizontale non pris en compte.
3. tracer une ligne de pixels entre chaque couple de points d'intersection.

Remplissage géométrique
Cohérence d'arête

Amélioration sensible des calculs des points d'intersection ligne-arête :

- une partie des arêtes est concernée par ce calcul d'intersection à chaque ligne de balayage,
- beaucoup d'arêtes intersectées par la ligne de balayage à l'étape courante le seront encore à l'étape suivante.

Calcul d'intersection de la nouvelle ligne avec une arête :

$$x_{i+1} = x_i + \frac{1}{p},\ y_{i+1} = y_i + 1$$

Pour éviter les calculs fractionnaires on conserve l'expression de la pente sous la forme $(\Delta y, \Delta x)$ et on compare les termes pour savoir quand et comment changer d'abscisse.

Remplissage géométrique
Cohérence de ligne de balayage

Utilisation d'une "table des arêtes actives" (TAA), liste d'arêtes, triées par abscisses d'intersection croissantes et modifiée en fonction de la ligne de balayage :

$$\begin{cases} \text{si } y = y_{max} \text{ alors l'arête est enlevée} \\ \text{si } y = y_{min} - 1 \text{ alors l'arête est ajoutée} \end{cases}$$

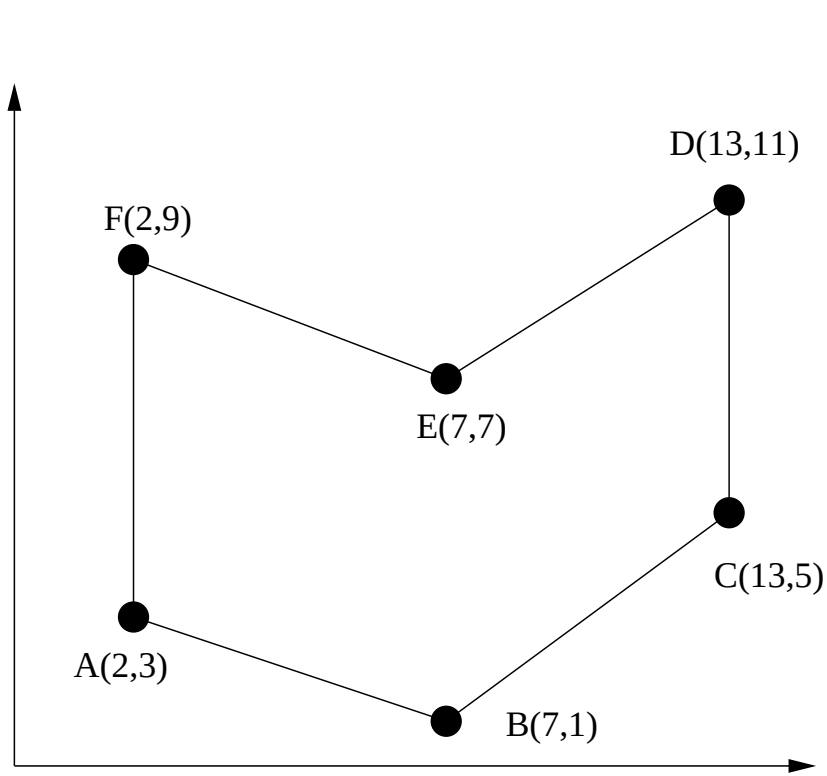
Chaque élément de cette liste sera de la forme :

x_{inter}	y_{max}	pente (N, D)	suivant→

Mise à jour efficace avec une "table des arêtes" (TA), dont les éléments sont triés par y_{min} croissante (sans les arêtes horizontales) :

y_{min}	x_{min}	y_{max}	pente (N, D)	suivant→

Remplissage géométrique
Cohérence de ligne de balayage



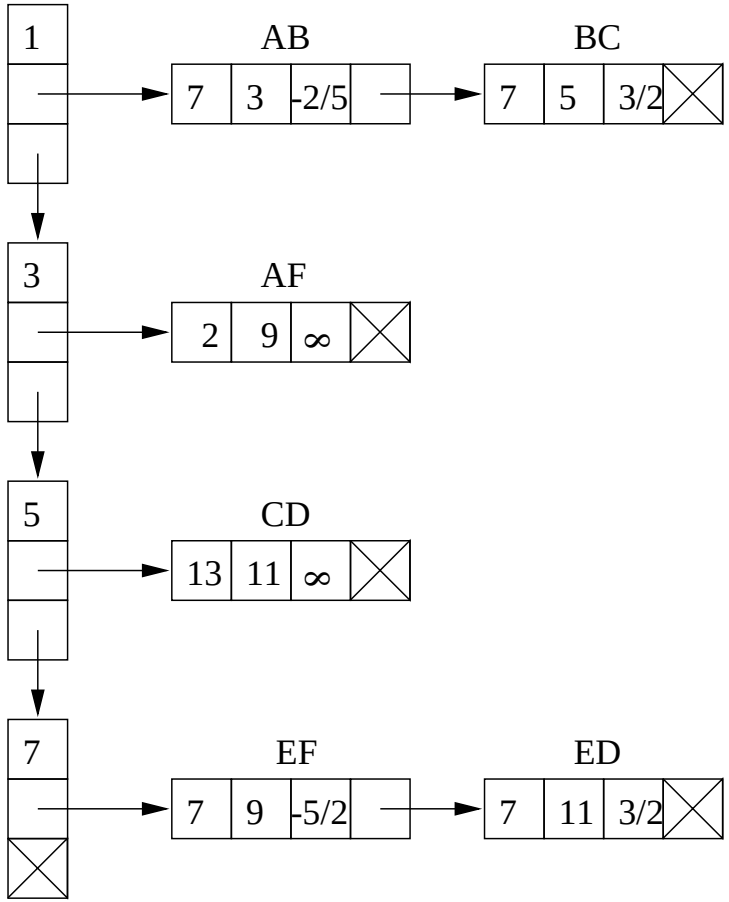


FIGURE : Un polygone et sa TA

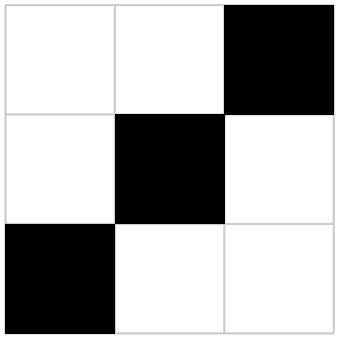
Remplissage géométrique
Algorithme

Algorithme Remplissage Geometrique(*pol*)
données
pol : polygone
variables
TA, *TAA* : listes d'arêtes
y, Ymax : entiers
ajout : booléen

Remplissage
Remplissage par motif

On distingue deux méthodes :

- pendant la discrétisation par balayage : dans l'espace réel par rapport à la primitive graphique ou dans l'espace écran auquel cas la primitive joue le rôle de région transparente laissant apparaître le motif,

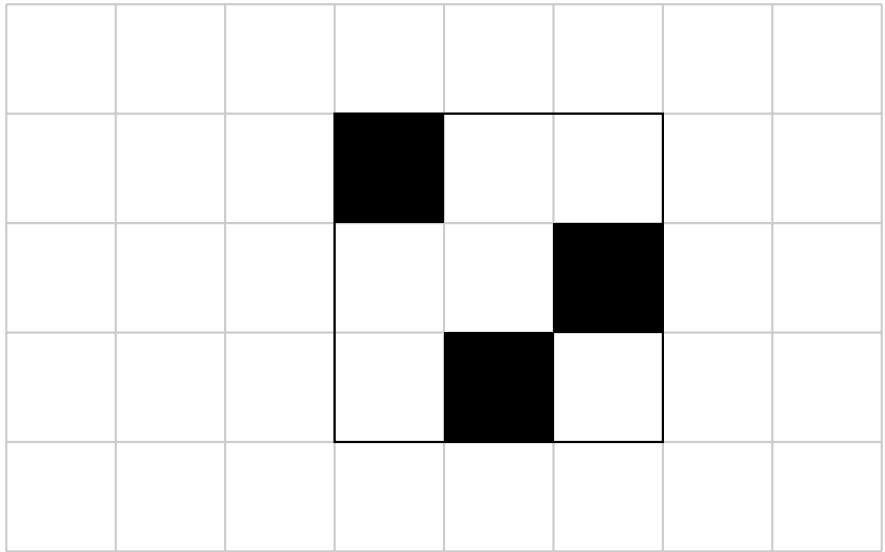


arithmétique modulaire

$3\%3 = 0$

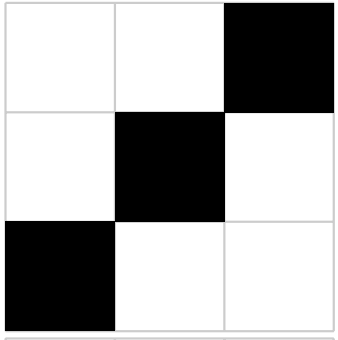
$1\%3 = 1$

,

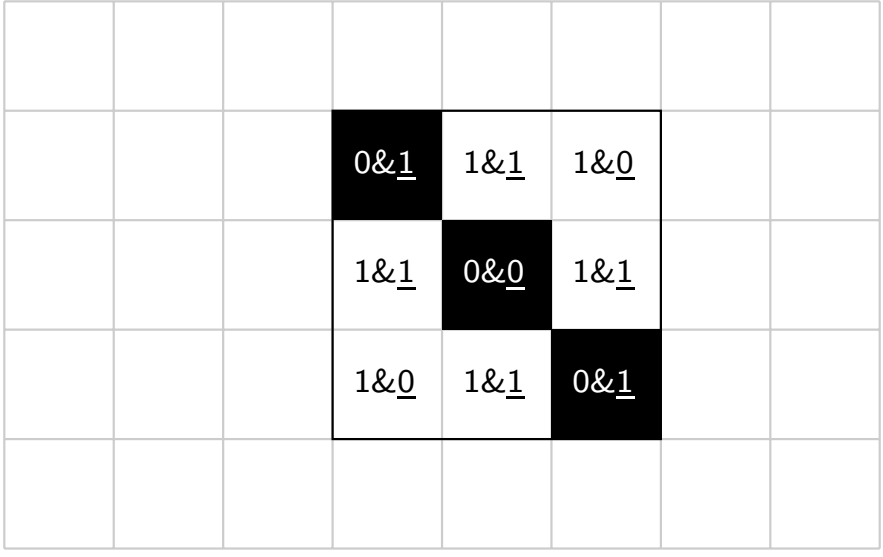


Remplissage
Remplissage par motif

- avant la discrétisation par balayage : définition d'un masque rectangulaire associée à la primitive (0 pour la primitive, 1 pour le reste), opérateur `ET` pour le masque puis opérateur `OU` pour la primitive.

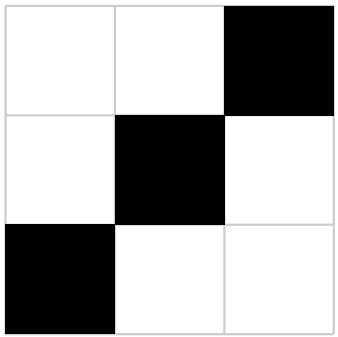


1	1	0
1	0	1
0	1	1

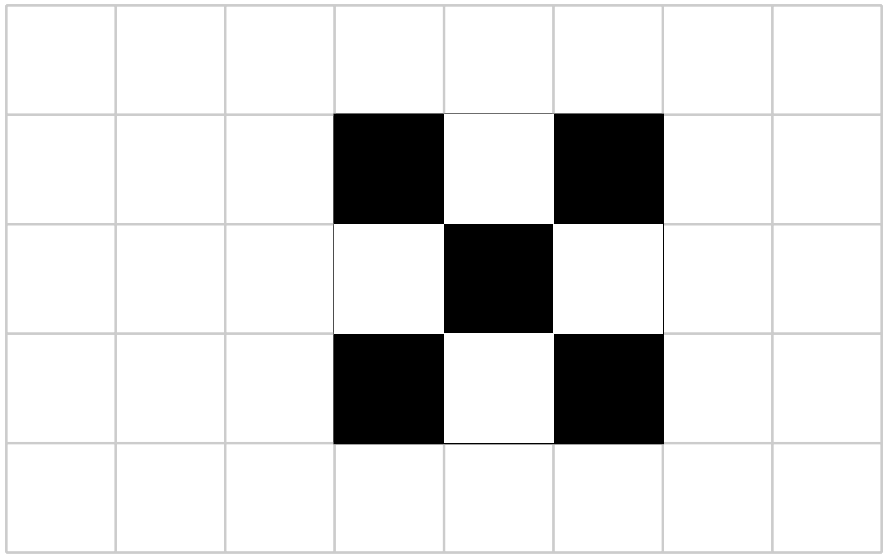


Remplissage
Remplissage par motif

- avant la discrétisation par balayage : définition d'un masque rectangulaire associée à la primitive (0 pour la primitive, 1 pour le reste), opérateur `ET` pour le masque puis opérateur `OU` pour la primitive.



1	1	0
1	0	1
0	1	1



Plan

Classification 2D

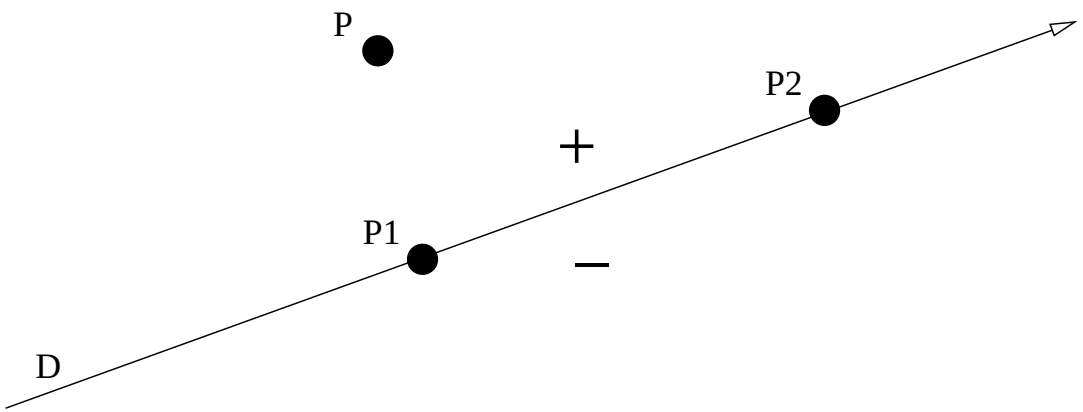
Classification point/droite

Soient $P1(x_1, y_1, h_1)$ et $P2(x_2, y_2, h_2)$. Une droite δ passant par ces deux points est donnée par :

$$\delta : det(P, P_1, P_2) = \begin{vmatrix} x & x_1 & x_2 \\ y & y_1 & y_2 \\ h & h_1 & h_2 \end{vmatrix} = 0$$

$$x(y_1 h_2 - h_1 y_2) + y(h_1 x_2 - h_2 x_1) + h(x_1 y_2 - y_1 x_2) = 0$$

Classification 2D
Classification point/droite



Un point sera à gauche (resp. à droite) d'un segment orienté joignant deux points P_1 et P_2 ssi $det(P, P_1, P_2) > 0$ (resp. $det(P, P_1, P_2) < 0$).

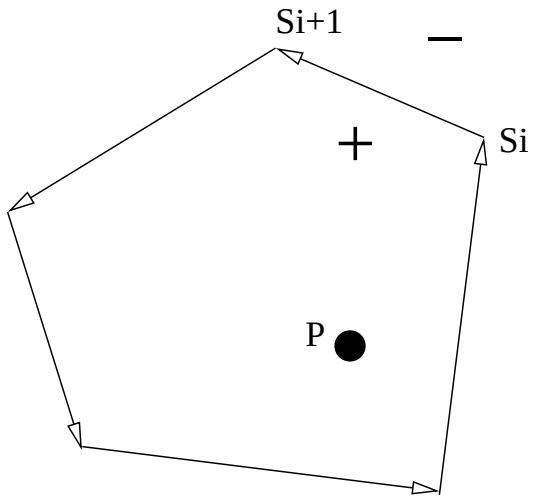
Un point appartiendra au contour si $det(P, P_1, P_2) = 0$.

Classification 2D

Classification point/contour

Dans le cas des polygones *convexes*, on se ramène à une classification point/droite.

P est à l'intérieur (resp. à l'extérieur) du polygone si $\forall i$ (resp. $\exists i$), $det(P, P_1, P_2) > 0$ (resp. $det(P, P_1, P_2) < 0$).



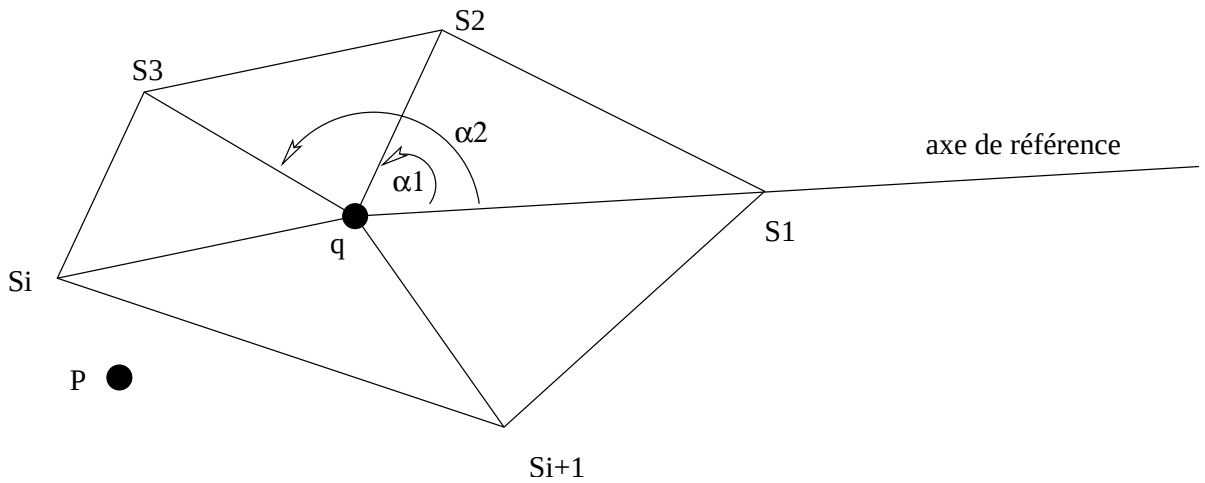
Classification 2D
Algorithme

Algorithme ClassificationPointPolygone(*pt*, *poly*)
données
pt : point
poly : polygone oriente
variables
continuer : booleen
i : entier
pos : reel

Classification 2D

L'algorithme de *Shamos*

Classement d'un grand nombre de points par rapport à un même contour. Complexité est en $O(\log n)$ mais avec un prétraitement en $O(n)$ qui consiste à décomposer le polygone en secteurs angulaires :



Calcul de l'angle $\Theta = \widehat{S_1 q P}$ puis recherche dichotomique du secteur auquel appartient P. Soit (S_i, S_{i+1}) l'arête de ce secteur, il ne reste plus qu'à classer P par rapport à cette arête.

Classification 2D

Point/polygone concave

L'algorithme des "angles capables" permet de déterminer la position d'un point sauf si celui-ci appartient au contour.

Calcul de tous les angles $\Theta_i = \widehat{S_i P S_{i+1}}$ et de leur somme. Si $\sum_{i=1}^{N} \Theta_i \simeq 2\Pi$ alors P se situe à l'intérieur du polygone sinon si $\sum_{i=1}^{N} \Theta_i \simeq 0$ alors P est à l'extérieur.

Coûteux du fait de l'utilisation de fonctions trigonométriques.

Polygones concaves
Algorithme des “angles capables”

Algorithme ClassificationPointPolygone(*pt*, *poly*)
données
    pt : point
    poly : polygone oriente
variables
    i : entiers
    α : reel
constantes
    PRECISION = 1.0e-1

---

i $\longleftarrow$ 0
tantque ($i < poly.nb\_som$) **faire**
    $\alpha \longleftarrow \alpha$ + CalculAngle(pt, poly.pt[i], poly.pt[(i+1)%poly.nb_som])
    i $\longleftarrow$ i+1
ftq
si ($-PRECISION < \alpha < PRECISION$) **alors**
    retourner EXT
sinon
    retourner INT
fsi

Classification 2D
Point/polygone concave

Autre solution : adapter l'algorithme "scan-line".

Calcul de la parité du nombre d'intersections entre le point considéré et la table des arêtes.

Si ce nombre est pair alors le point est à l'extérieur sinon il est à l'intérieur.

Cas particuliers (point appartenant à la droite support d'une arête, ...) augmentant la complexité initiale, en $O(n)$.

Classification 2D

Segment/segment

Méthode paramétrique naturelle et efficace. Soient deux segments de droites définis par (A, B) et (C, D). Un point intersection de ces deux segments sera solution du système :

$$\begin{cases} x_A + t_1(x_B - x_A) = x_C + t_2(x_D - x_C) \\ y_A + t_1(y_B - y_A) = y_C + t_2(y_D - y_C) \end{cases}$$

soit

$$\begin{cases} t_1(x_B - x_A) + t_2(x_C - x_D) = x_C - x_A \\ t_1(y_B - y_A) + t_2(y_C - y_D) = y_C - y_A \end{cases}$$

Le déterminant de ce système (de Cramer) est : $\det = (x_B - x_A)(y_C - y_D) - (x_C - x_D)(y_B - y_A)$.

Classification 2D
Segment/segment

Algorithme Classification segment/segment(A, B, C, D)
données
    A, B, C, D : points
variables
    det : réel

$det \longleftarrow (x_B - x_A)(y_C - y_D) - (x_C - x_D)(y_B - y_A)$
si $(det = 0)$ **alors**
    segments parallèles ou colinéaires
sinon
    $t_1 = ((x_C - x_A)(y_C - y_D) - (x_C - x_D)(y_C - y_A))/det$
    $t_2 = ((x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A))/det$
    si ($t1 > 1$ *ou* $t1 < 0$ *ou* $t2 > 1$ *ou* $t2 < 0$) **alors**
        pas d'intersection
    sinon
        si ($t1 = 0$ (resp. $t1 = 1$)) **alors**
            intersection au point A (resp. B)
        sinon
            si ($t2 = 0$ (resp. $t2 = 1$)) **alors**
                intersection au point C (resp. D)
            sinon
                $x_i = x_A + t_1(x_B - x_A)$
                $y_i = y_A + t_1(y_B - y_A)$
            fsi
        fsi
    fsi
fsi

Classification 2D
D'un ensemble de segments

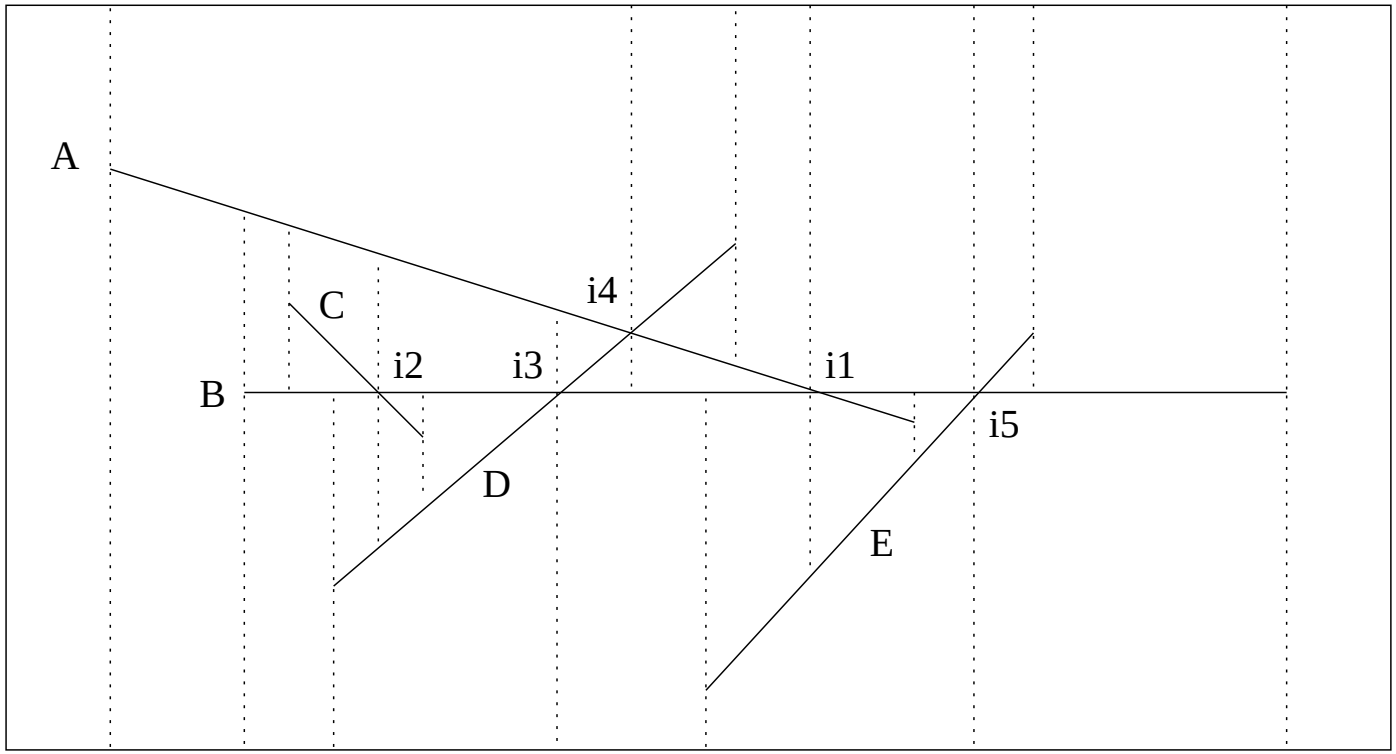
La classification d'un ensemble de segments nécessite une méthode efficace.

La méthode de Bentley et Ottmann recherche dans une famille finie de segments, par balayage du plan, les paires qui se coupent. Toutes les insertions effectuées dans l'algorithme se font par abscisse ou ordonnée croissante.

À noter cependant que cet algorithme est bien connu pour être très sensible aux erreurs numériques.

Classification 2D

D'un ensemble de segments



lPtx = A_0, B_0, C_0, D_0, i_2, C_1, i_3, i_4, E_0, D_1, i_1, A_1, i_5, E_1, B_1

lSeg = A → BA → BCA → DBCA → DCBA → DBA → BDA → BAD → EBAD → EBA → EAB → EB → BE → B

Classification d'un ensemble de segments

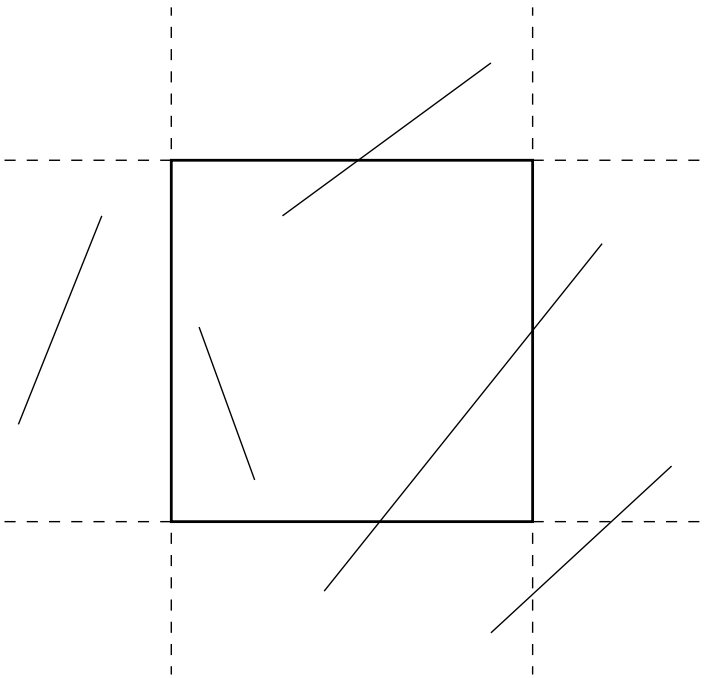
Algorithme ClassificationEnsembleSegments(*tseg*)
données
tseg : tableau des N segments
variables
lPtx : liste de points (associés à un type)
lSeg : liste de liste d'identificateurs des segments ▷ *triés par ord. croissantes*
*x*0 : couple (abscisse, type)

lPtx ⟵ tri par abscisses croissantes des 2N sommets
initialiser lSeg avec le 1er segment correspondant au 1er point de lPtx
pt ⟵ 1er point de lPtx
tantque (non fin(*lPtx*)) **faire**
cas (type(pt)) **de**
`DEBUT` :
insérer dans lSeg par ord. croissante le segment de sommet pt
▷ *par intersections des segments de lSeg avec x = pt.x*
si (dans lSeg ce segment coupe ses voisins) **alors**
insérer le(s) point(s) d'intersection dans lPtx (par abs. croissantes)
fsi
`FIN` :
si (dans lSeg les voisins de ce segment se coupent) **alors**
insérer le point d'intersection dans lPtx (par abs. croissantes)
fsi
enlever ce segment de lSeg
`INTERSECTION` :
échanger les positions des segments concernés dans lSeg
si (les segments nouvellement adjacents dans lSeg se coupent) **alors**
insérer dans lPtx les points d'intersection (par abs. croissantes)
fsi

Fenêtrage (clipping)

De segments

Plusieurs cas se présentent :



Le fenêtrage des points extrémités répondent au critère :

$$\begin{cases} x_{min} \leq x \leq x_{max} \\ y_{min} \leq y \leq y_{max} \end{cases}$$

Clipping
De segments

Résolution d'équations : comporte trop de calculs et de tests (test des points extrémités, intersection avec les 4 droites définissant la fenêtre, test d'appartenance des points d'intersection à la fenêtre, ...).

Fenêtrage de Cohen-Sutherland (1970) : basé sur la notion de "contrôle de régions", test les points extrémités pour déterminer si le segment peut être accepté, retraité ou rejeté.

Le plan de fenêtrage est divisé en 9 régions, chacune d'elles comportant un code sur 4 bits, correspondant à des inégalités strictes et autorisant des tests d'appartenance efficaces.

Clipping
De segments

N	S	E	O

- si les deux extrémités sont à 0000 alors le segment est totalement visible.
- si le `ET` logique appliqué aux deux points extrémités d'un segment est différent de 0 alors le segment peut être rejeté.
- sinon, on prend l'un des sommets *extérieur* (il y en a au moins un) et à partir du premier bit rencontré à 1, on détermine le point d'intersection avec un côté de la fenêtre.
- on réitère ainsi le processus jusqu'à ce que les deux codes de chacun des sommets soient à 0000.

Clipping
De segments

Algorithme FenSeg(*seg*, *fen*)
données
    seg : @segment
    fen : fenetre
variables
    *reg*1, *reg*2 : entiers
    d : droite

---

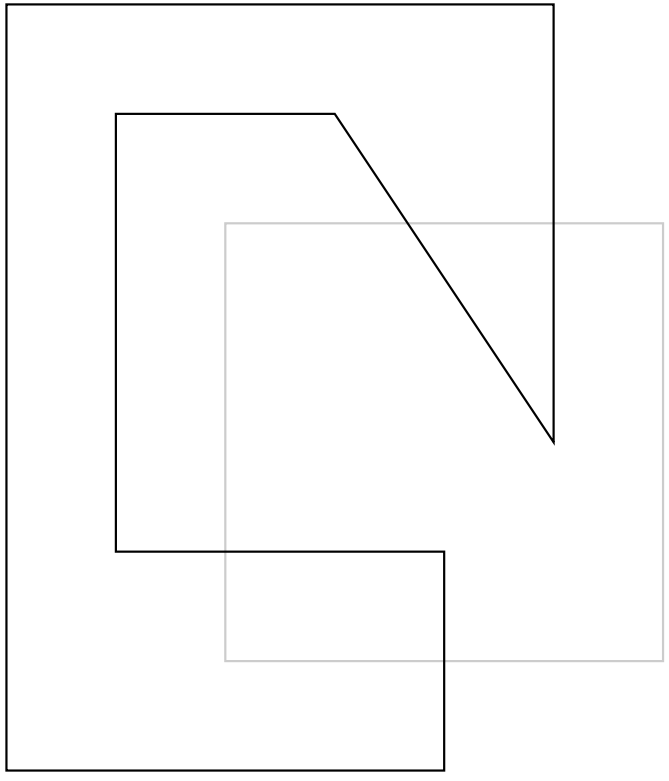
reg1 ⟵ region(seg.P1, fen)
reg2 ⟵ region(seg.P2, fen)
tantque ((*reg*1 ≠ 0) ou (*reg*2 ≠ 0) et (*reg*1 ET *reg*2 = 0)) **faire**
    si (*reg*1 ≠ 0) **alors** ▷ *P1 à l'extérieur*
        d ⟵ cote(@reg1) ▷ *on anticipe sur la modification*
        seg.P1 ⟵ intersection(seg, d)
    sinon ▷ *reg2 ≠ 0 et P2 à l'extérieur*
        d ⟵ cote(@reg2) ▷ *on anticipe sur la modification*
        seg.P2 ⟵ intersection(seg, d)
    fsi
ftq
  ▷ *cas segment rejeté*
si (*reg*1 ET *reg*2 ≠ 0)) **alors**
    seg.P1 ⟵ seg.P2 ⟵ NULL
fsi

Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` $\rightarrow$ `ext` : écarté
- `ext` $\rightarrow$ `int` : intersection
- `int` $\rightarrow$ `ext` : intersection tourner
- `int` $\rightarrow$ `int` : retenu

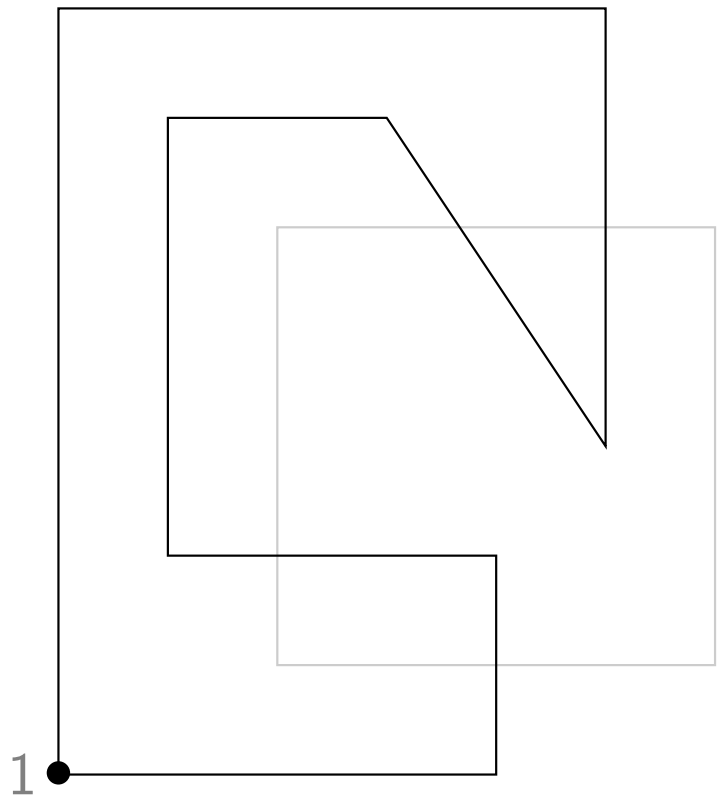


Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` $\rightarrow$ `ext` : écarté
- `ext` $\rightarrow$ `int` : intersection
- `int` $\rightarrow$ `ext` : intersection tourner
- `int` $\rightarrow$ `int` : retenu

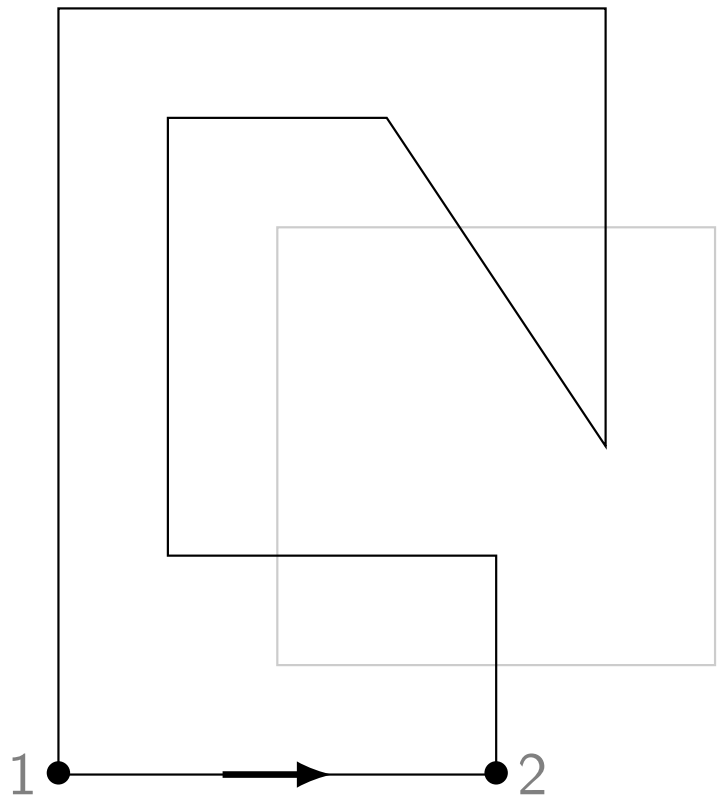


Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` $\rightarrow$ `ext` : écarté
- `ext` $\rightarrow$ `int` : intersection
- `int` $\rightarrow$ `ext` : intersection tourner
- `int` $\rightarrow$ `int` : retenu

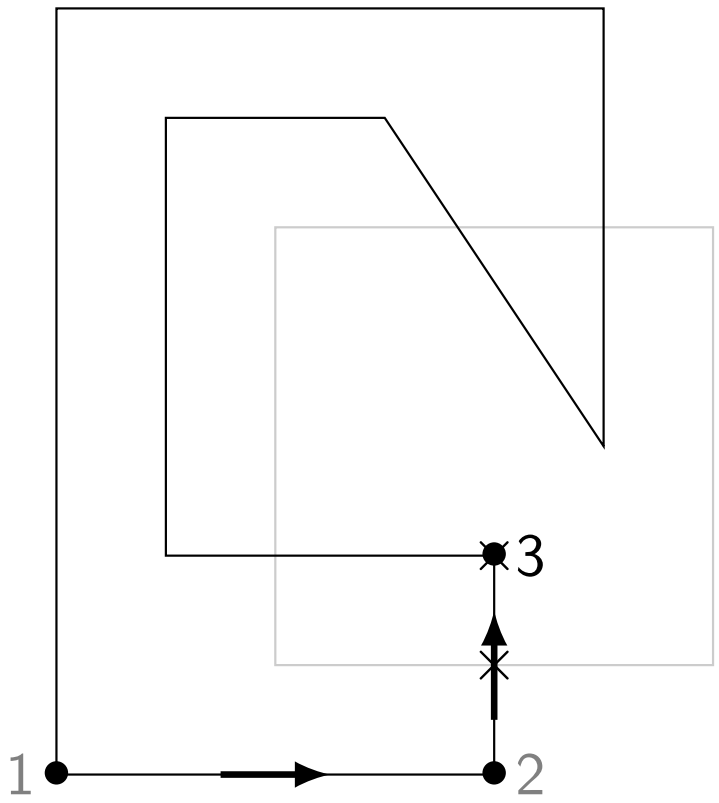


Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` $\rightarrow$ `ext` : écarté
- `ext` $\rightarrow$ `int` : intersection
- `int` $\rightarrow$ `ext` : intersection tourner
- `int` $\rightarrow$ `int` : retenu

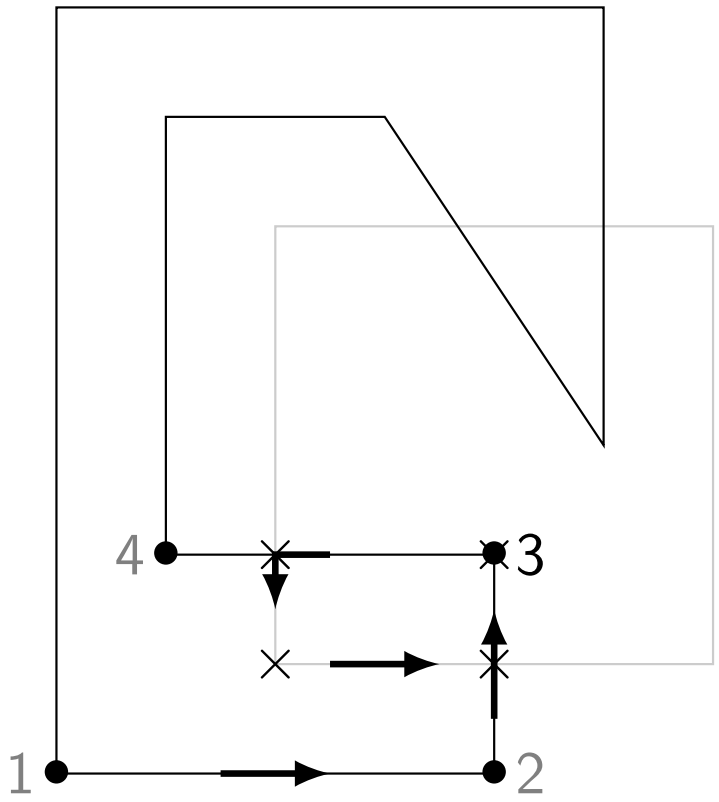


Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` → `ext` : écarté
- `ext` → `int` : intersection
- `int` → `ext` : intersection tourner
- `int` → `int` : retenu

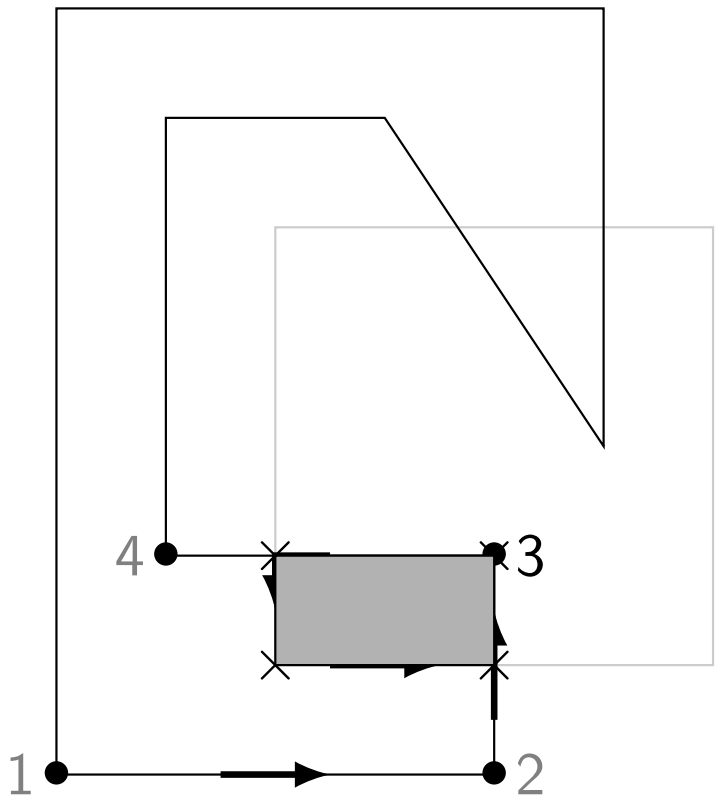


Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` $\rightarrow$ `ext` : écarté
- `ext` $\rightarrow$ `int` : intersection
- `int` $\rightarrow$ `ext` : intersection tourner
- `int` $\rightarrow$ `int` : retenu

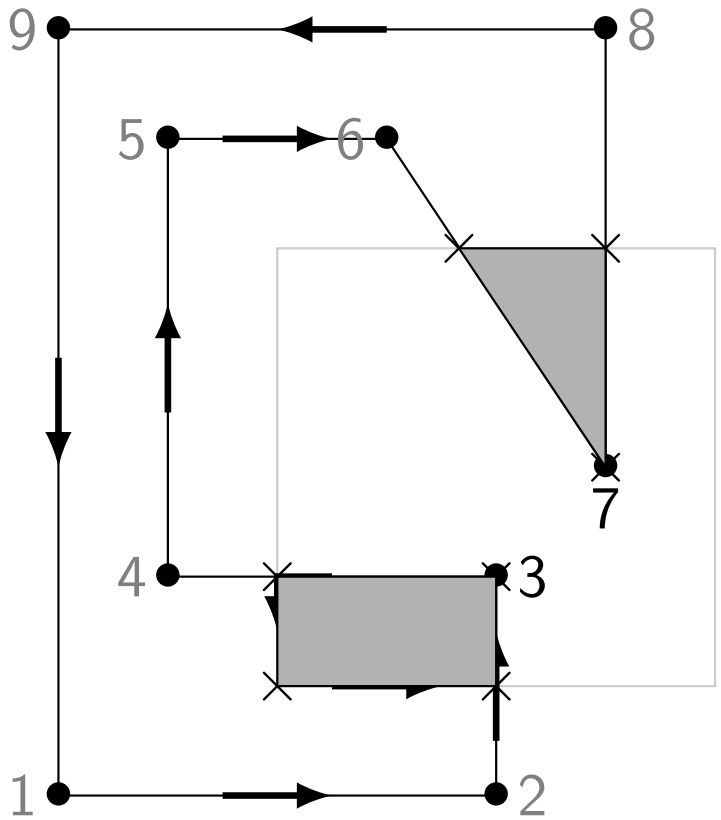


Clipping
De polygones

Algorithme de Sutherland-Hodgman (1974) : parcours d'un polygone dans le sens trigonométrique.

Règles :

- `ext` $\rightarrow$ `ext` : écarté
- `ext` $\rightarrow$ `int` : intersection
- `int` $\rightarrow$ `ext` : intersection tourner
- `int` $\rightarrow$ `int` : retenu



Antialiasing (anticrênelage)

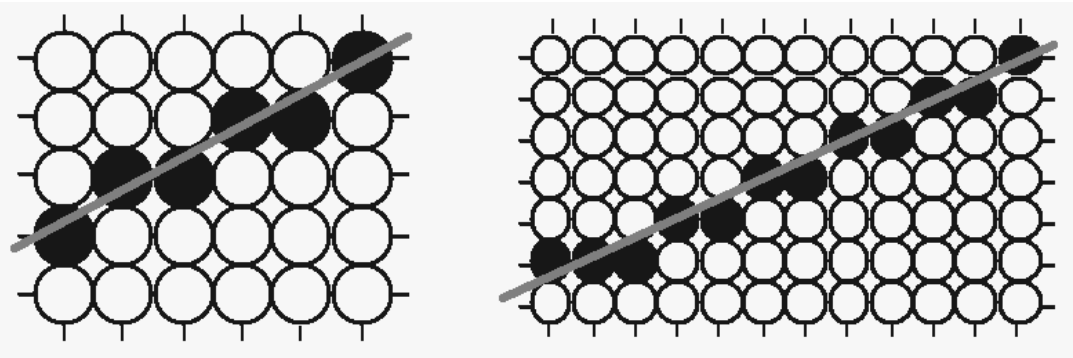
Les algorithmes de génération de tracés ont un problème commun : la création de bords crénelés.

Cet effet indésirable est dû au passage du continu au discret : c'est donc un problème d'échantillonnage (traitement du signal).

Les techniques misent en œuvre pour réduire cet effet sont connus sous le nom d'anticrênelage (*antialiasing*) et les primitives (ou les images) produites en utilisant ces techniques sont dites anticrênelées.

Antialiasing
Augmentation de la résolution

Le nombre de crénelures est deux fois plus important mais les sauts sont de taille deux fois moindre.



En multipliant par deux la résolution horizontale et vertical d'un terminal, il est nécessaire de quadrupler le coût en mémoire, la bande passante de la mémoire et le temps de discrétisation.

Antialiasing
Full Scene Antialiasing

On calcule l'image avec une définition virtuelle plus élevée que celle du dispositif d'affichage, puis on moyenne les couleurs des pixels obtenus pour l'affichage dans une résolution plus basse.

La discrétisation se fait à une fréquence supérieure, multiple de la fréquence cible, c'est le *suréchantillonnage*. Une pondération est ensuite effectuée sur les échantillons ainsi obtenus :

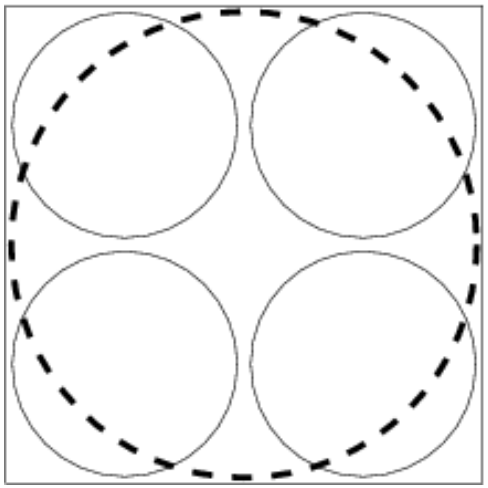
$$p(x,y) = \sum_{i=1}^{n} w_i p_i(x,y)$$

avec généralement $w_i = 1/n$ (moyenne uniforme).

Antialiasing
Full Scene Antialiasing

Application directe au cas $n = 4$:

- quatre rendus successifs (à résolution supérieure) avec un léger décalage spatial,
- puis calcul de moyenne pour chaque pixel du buffer d'accumulation.



$$p(x, y) = \frac{1}{4} \sum_i p(i, x, y)$$

Antialiasing
Full Scene Antialiasing

Moyenne avec pondération (judicieuse) [Crow] : à une résolution virtuelle triple, on utilise le filtre passe-bas bidimensionnel suivant :

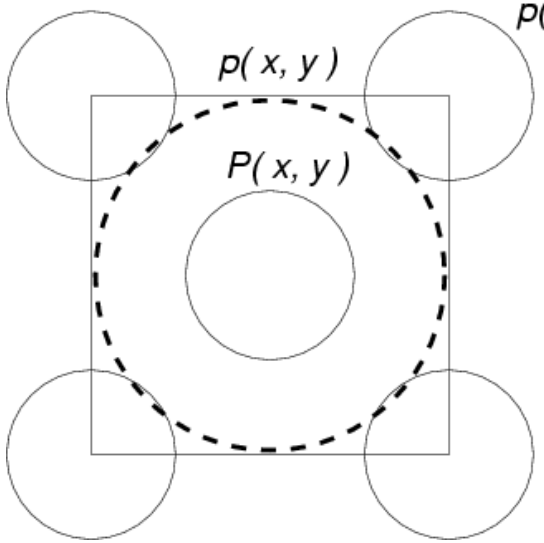
1	2	1
2	4	2
1	2	1

Ainsi, des surfaces égales donnent une intensité différente en fonction de la distance entre le centre du pixel et la surface.

Antialiasing
High Resolution Antialiasing

nVIDIA : suréchantillonnage dont les échantillons influencent plusieurs pixels.

Cette méthode est plus économique (deux échantillons en moyenne par pixel) et meilleure que la méthode précédente.



$$p(x,y) = \frac{1}{8} \sum_i p(i,x,y) + \frac{1}{2} P(x,y)$$

Antialiasing
Echantillonnage non pondéré de surface

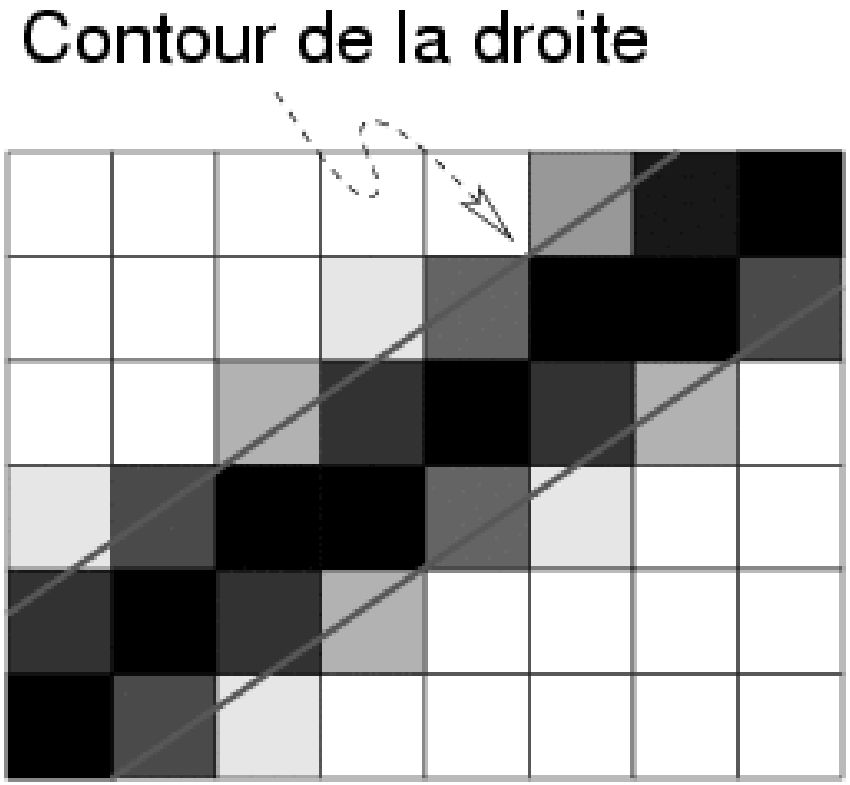
Un segment tracé à une largeur non nulle, variable le long de sa trajectoire (sauf horizontalement et verticalement).

Dans l'espace discret un segment est un rectangle (d'une certaine largeur) recouvrant partiellement de nombreux pixels, ceux-ci contribuant pour un certain pourcentage.

Du point de vue calculatoire, il est avantageux de considérer la grille comme une succession de pixels juxtaposés. L'intensité attribuée à chaque pixel dépend du pourcentage de surface récouvert par le segment.

Antialiasing

Echantillonnage non pondéré de surface



Antialiasing
Echantillonnage pondéré de surface

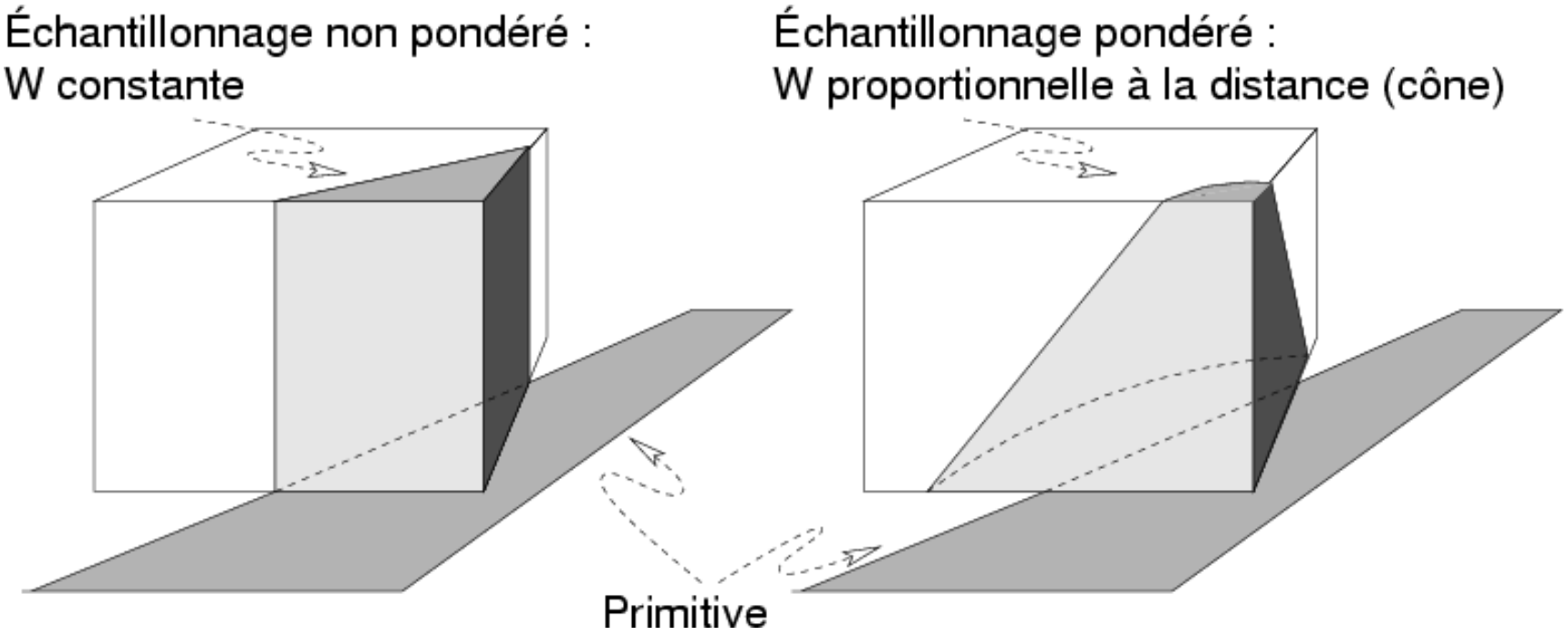
Plus une aire est près du centre de la primitive, plus elle a une contribution élevée.

Soit $W(x, y)$ une fonction de pondération, on calcule l'intensité I d'un pixel comme suit :

$$I = \int_{aire} W(x, y) dxdy$$

Rechercher l'intégrale de cette fonction de pondération revient à chercher le volume d'intersection d'un cône avec le pixel.

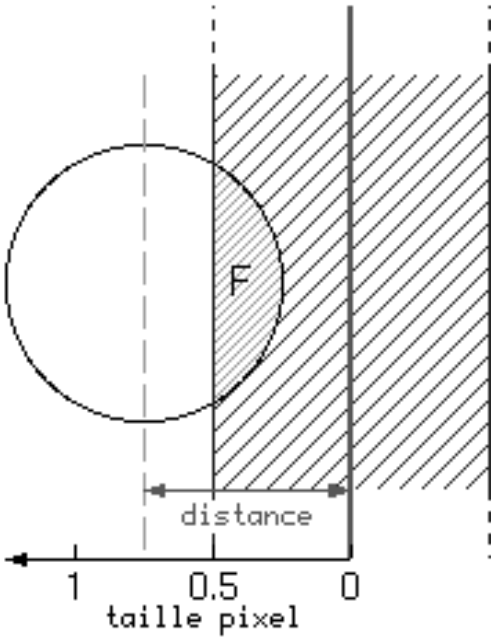
Antialiasing
Echantillonnage pondéré de surface

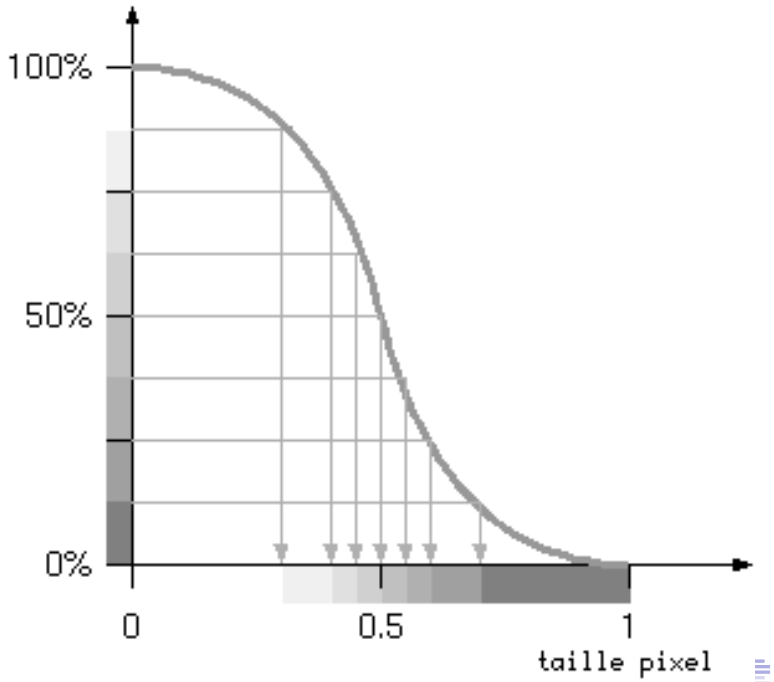


Antialiasing
Précalculé de Gupta-Sproull

Le calcul précédent peut se ramener à celui du calcul de la surface d'un cercle recouvert par une bande : table pré-calculée des valeurs d'intensité en fonction de la distance.

Prise en compte des calculs de distance dans l'algorithme de Bresenham.





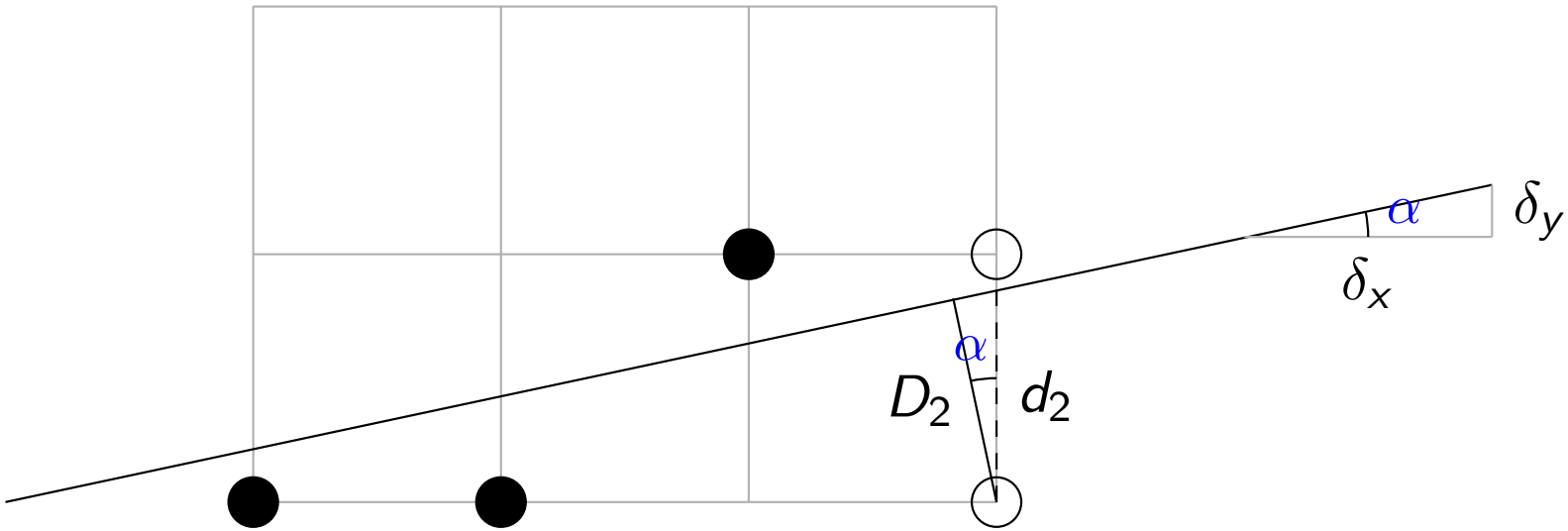
Antialiasing
Précalculé de Gupta-Sproull

Table de Gupta-Sproull pour 8 valeurs :

Pondération	Pas	Distance
$[0, 1/8]$	0 (noir)	$[1.4, 2]$
$[1/8, 2/8]$	1	$[1.2, 1.4]$
$[2/8, 3/8]$	2	$[1.08, 1.2]$
$[3/8, 4/8]$	3	$[1, 1.08]$
$[4/8, 5/8]$	4	$[0.92, 1]$
$[5/8, 6/8]$	5	$[0.8, 0.92]$
$[6/8, 7/8]$	6	$[0.6, 0.8]$
$[7/8, 1]$	7 (blanc)	$[0, 0.6]$

Antialiasing
Précalculé de Gupta-Sproull

Calcul de la distance durant le tracé de Bresenham.

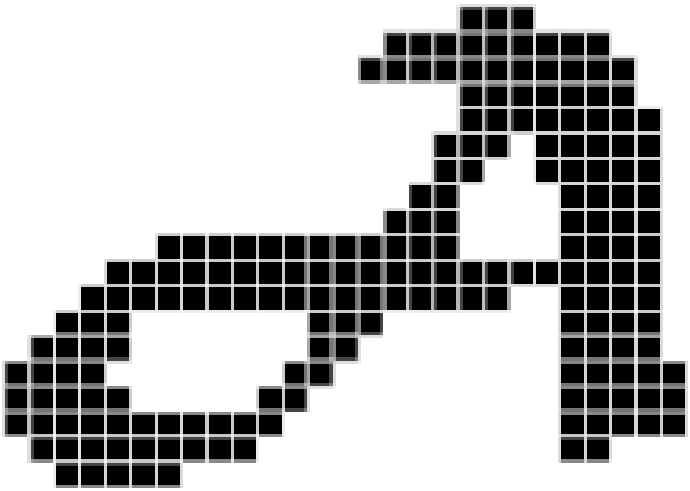


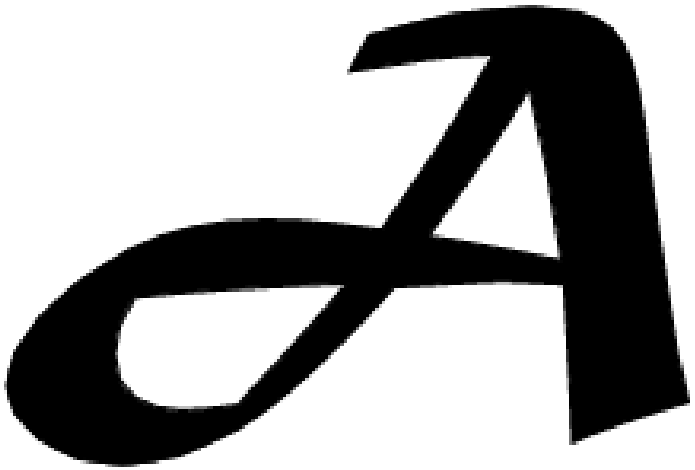
$$D_2 = d_2 \cos\alpha = \frac{d_2 \delta_x}{\sqrt{\delta_x^2 + \delta_y^2}}$$

Génération de caractères

Deux techniques complémentaires

- définition par matrice binaire,
- définition par le contour.





Génération de caractères
Matrice binaire

Chaque caractère d'une police est définie dans un petit pictogramme rectangulaire.

Fenêtrage : pixel par pixel, par caractère voire par chaîne de caractères.

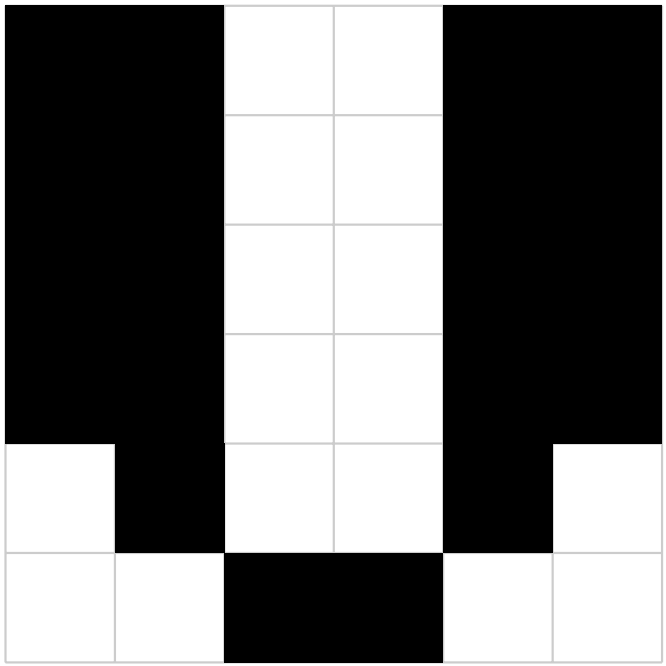
Inconvénient :

- nécessite un cache de polices pour chaque combinaison de police, de taille, d'aspect (normal, gras, italique, ...) et pour chaque définition différente,
- les transformations non rigides engendrent des déformations inesthétiques.

Génération de caractères
Matrice binaire

Description du cache de polices : descripteur mémoire, hauteur totale, hauteur du jambage, espacement (fixe ou variable) suivi de la table des caractères.

Codage ligne par ligne (1), par plages (2) ou par plages différentielles (3).



(1)	(2)	(3)
110011	1256	1246
110011	1256	0000
110011	1256	0000
110011	1256	0000
010010	2255	100-1
001100	34	100-1

Génération de caractères
Contour

Définition abstraite est indépendante du matériel, chaque caractère est défini par un *contour* curviligne (voire polygonale) :

- par interpolation grâce à des segments de droites, des arcs de cercle,
- par combinaison de primitives, barres, courbes, queues, . . .(CSD : *Character Simulated Design*),
- par codage d'un squelette (*MetaFont*),
- par des fonctions polynomiales (*splines* : courbes au moins C^1 continues).

Une seule définition du contour ne suffit pas à toutes les variations de taille.