# Intersections

Christian Nguyen

Département d'informatique
Université de Toulon

# Plan

Soient les vecteurs u, v dans le plan euclidien, l'*expression analytique* de leur déterminant est :

$$det(u, v) = \begin{vmatrix} x & x' \\ y & y' \end{vmatrix} = xy' - yx'$$

L'*expression géométrique* équivalente est :

$$det(u, v) = \|u\|\|v\| \sin(u, v)$$

Propriétés :

- la valeur absolue du déterminant est égale à l'aire du parallélogramme défini par u et v,
- le déterminant est nul ssi les deux vecteurs sont colinéaires,
- son signe est strictement positif ssi la mesure de l'angle (u, v) est comprise dans l'intervalle $]0, \pi[$.

Déterminant de trois vecteurs dans l'espace euclidien (ou produit mixte) :

$$det(u,v,w) = \begin{vmatrix} x & x' & x'' \\ y & y' & y'' \\ z & z' & z'' \end{vmatrix} = x \begin{vmatrix} y' & y'' \\ z' & z'' \end{vmatrix} - y \begin{vmatrix} x' & x'' \\ z' & z'' \end{vmatrix} + z \begin{vmatrix} x' & x'' \\ y' & y'' \end{vmatrix}$$

Propriétés :

- la valeur absolue du déterminant est égale au volume du parallélépipède défini par les trois vecteurs,
- le déterminant est nul ssi les trois vecteurs sont contenus dans un même plan (coplanaires).

Soit une droite $\delta : ax + by + c = 0$, celle-ci peut s'écrire $axh + byh + ch = 0$ ou bien encore :

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} xh \\ yh \\ h \end{pmatrix} = 0$$

Remarque : dans le système de coordonnées homogènes, les points et les droites ont une représentation analogue et leurs rôles peuvent être échangés.

Soient $\vec{u} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}$, $\vec{v} \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}$ vecteurs directeurs des droites $\delta$ et $\delta'$.

Le point d'intersection de ces deux droites peut être déterminé grâce au produit vectoriel par un calcul en composantes :

$$\vec{u} \wedge \vec{v} = \begin{pmatrix} b_1 c_2 - b_2 c_1 \\ a_2 c_1 - c_2 a_1 \\ a_1 b_2 - b_1 a_2 \end{pmatrix}$$

Méthode 1 :

1. Déterminer les équations des deux supports des segments.
2. Résoudre le système (2 équations à 2 inconnues réelles).

Les deux segments se coupent ssi les deux paramètres appartiennent simultanément à $[0, 1]$.

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

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

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

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

**Algorithme 1 :** `classification_segment_segment`$(A, B, C, D)$

Méthode 2 : utilisation de la notion de *puissance d'un point*, les deux segments se coupent en un point distinct de leur quatre extrémités ssi :

- A, B sont situés de part et d'autre de la droite (CD),
- C, D sont situés de part et d'autre de la droite (AB).

En géométrie euclidienne du plan, la *puissance d'un point* $M$ par rapport à un cercle de centre $O$ et de rayon $R$ est un nombre qui indique la position de $M$ par rapport à ce cercle. Elle peut être définie comme $P(M) = OM^2 - R^2$.

La droite (AB) définit 3 régions :

- un demi-plan formé de tous les points de puissance positive,
- un demi-plan formé de tous les points de puissance négative,
- la droite elle-même dont tous les points ont une puissance nulle.

Tous les points ayant une puissance positive (resp. négative, nulle) sont "à gauche de" (resp. "à droite de", "sur") la droite orientée $\overrightarrow{AB}$.

En *coordonnées homogènes*, raisonnement en deux temps pour la méthode 2 :

1. Les sommets définissant l'un des segments doivent être de part et d'autre de l'autre segment considéré :

$$\begin{cases} det(A, C, D) \cdot det(B, C, D) < 0 \\ det(C, A, B) \cdot det(D, A, B) < 0 \end{cases} \quad \text{(signes opposés)}$$

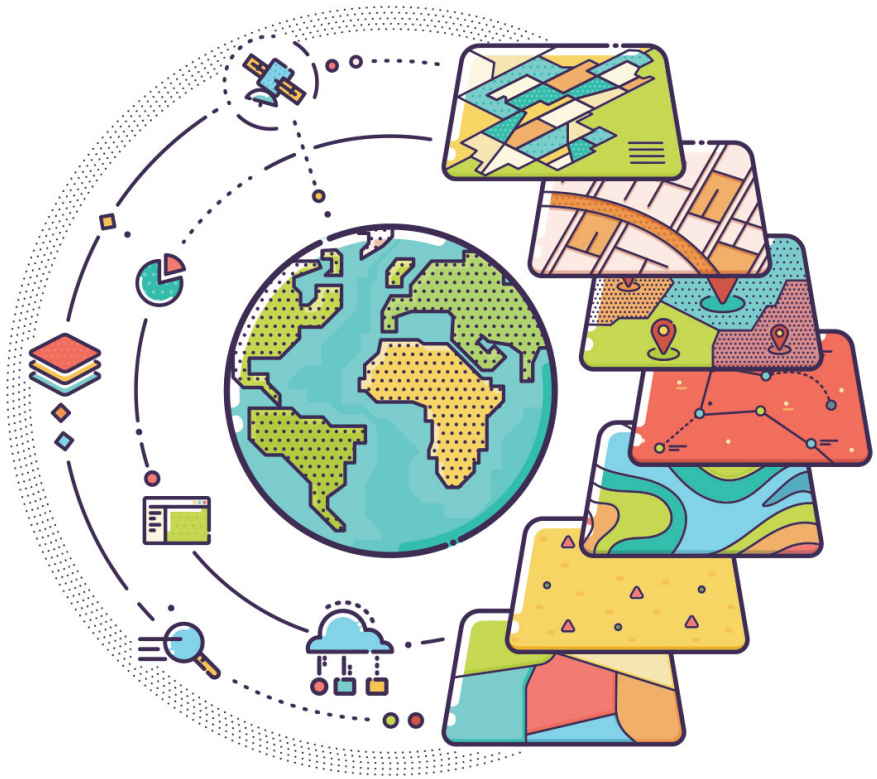
2. Si ces conditions sont remplies, alors le point d'intersection est solution de :

$$\begin{cases} det(M, A, B) = 0 \\ det(M, C, D) = 0 \end{cases}$$

# Plan

# Introduction

Les SIG sont des sources d'informations concrètes : routes, lignes de chemin de fer, attractions touristiques, lacs, . . . ou plus abstraites : densité de population, précipitations moyennes, types de végétations, . . .



# Introduction

La géométrie décrivant chaque information peut elle-même être variée : points localisant les villes, lignes brisées (ou courbes) décrivant les routes, régions, ...

Pour rendre les cartes plus lisibles, les SIG les décomposent en plusieurs couches, chaque couche correspondant à un type d'information.

Obtention d'information pertinente par intersection de plusieurs couches du SIG. Par exemple : intersection des routes et des rivières, intersection de régions de végétations et de précipitations, ...

# Recouvrement simple

Cas simple d'extraction d'informations par recouvrement de couches : chaque couche contient des collections de segments de droites.

Problème géométrique : soient deux ensembles de segments, calculer toutes les intersections.

Un algorithme brute-force analyse chaque paire de segments et détermine leur éventuelle intersection. Il est en $O(n^2)$, ce qui est optimal ... si tous les segments s'intersectent.

En pratique, la plupart des segments intersectent très peu, voire pas du tout, les autres segments.

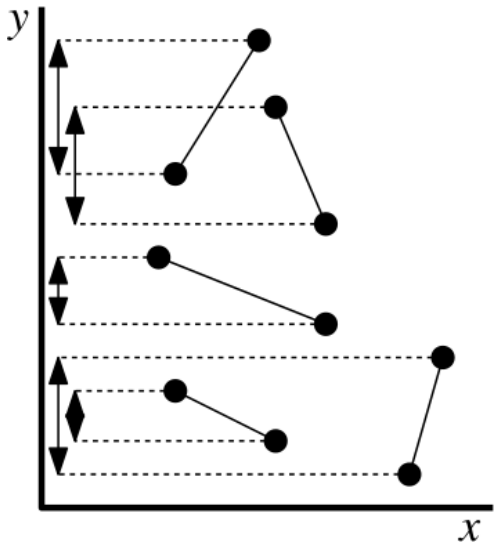
# Intersection de segments

## Bentley et Ottmann (1979)

Algorithme *output-sensitive* : tient compte non seulement du nombre de segments mais aussi du nombre d'intersections.

Les segments proches les uns des autres sont susceptibles de s'intersecter contrairement aux segments éloignés.

Une façon élégante de déterminer quels segments sont proches est de les projeter tous orthogonalement sur l'axe des ordonnées.



# Intersection de segments

La comparaison des intervalles de valeurs fait apparaître deux cas :

1. S'ils ne se recouvrent pas, c'est que les segments sont éloignés verticalement et ne s'intersectent pas,
2. S'ils se recouvrent, il faut tester si des *paires* de segments appartiennent à la même droite d'intersection horizontale.

Pour déterminer ces paires, on fait descendre une *droite de balayage* (scan-line ou sweep-line) depuis une position « dominante ».

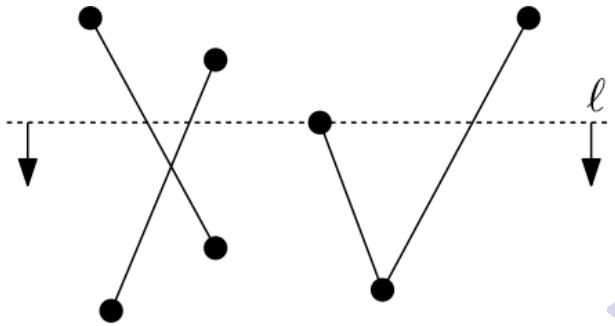
# Intersection de segments

On a éliminé les segments qui ne se recouvrent pas verticalement et qui donc ne s'intersectent pas.

On associe à la ligne de balayage l'ensemble des segments qui se recouvrent verticalement, information que l'on met à jour de façon *contextuelle* (et non continue).

Seuls certains points nécessitent une mise à jour : les points d'évènement (PE), qui sont ici les sommets extrémités des segments :

- lorsque la ligne de balayage atteint le sommet d'ordonnée maximale d'un segment, ce dernier est ajouté,
- lorsque la ligne de balayage atteint le sommet d'ordonnée minimale d'un segment, ce dernier est retiré.



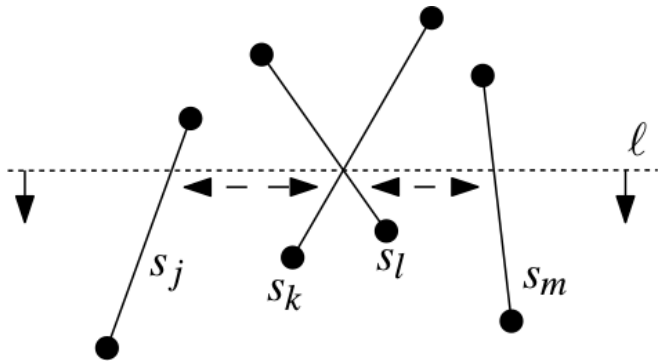
# Intersection de segments

Malheureusement, des cas particuliers entraînent une détermination quadratique des intersections d'un grand ensemble de segments.

Il est nécessaire de trier les segments intersectés par la ligne de balayage, par exemple par abscisse croissante.

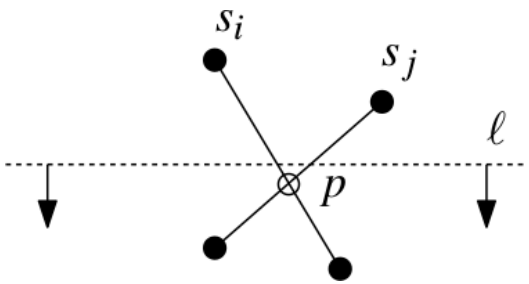
- cela limite la comparaison des paires de segments aux voisins droites et gauches d'un segment donné.

Par contre, la mise à jour des segments intersectés par la ligne de balayage prend maintenant en compte les points d'intersection (nouveau PE) entre segments.

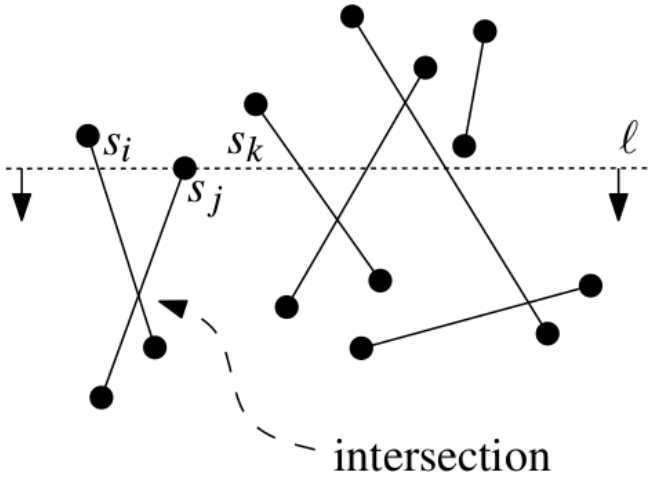


Sans tenir compte des cas particuliers (pas de segments horizontaux, pas de segments portés par la même droite, pas plus de deux segments par point d'intersection), les intersections sont-elles toujours détectées ?

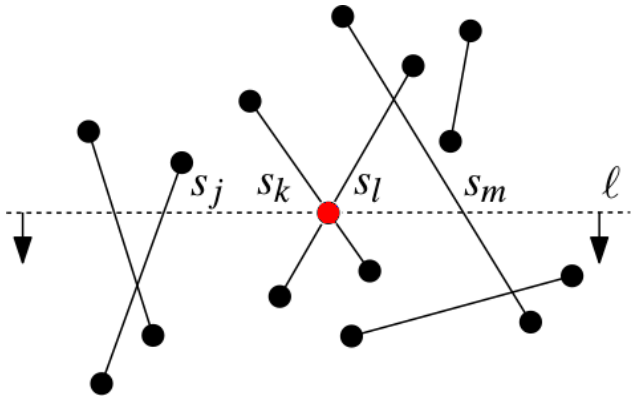
Soient $s_i$ et $s_j$ deux segments non horizontaux, soit $p$ l'intersection de leur ouvert, $p$ n'appartenant à aucun autre segment, alors il y a un PE avant $p$ pour lequel $s_i$ et $s_j$ sont adjacents et dont on teste l'intersection.



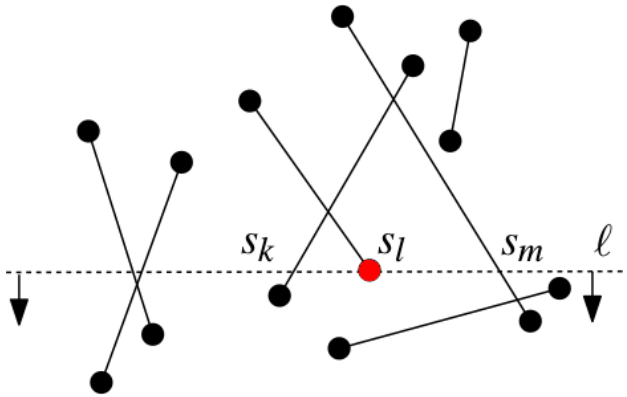
- Une demi-droite balaye le plan du haut vers le bas,
- Elle permet de maintenir un ensemble de PE triés suivant leur abscisse :
  - point haut : insertion d'un nouveau segment $s_j$ et tests d'intersection (en dessous de la ligne de balayage) avec ses voisins $s_i$ et $s_k$,



- point intersection : inversion des segments concernés ($s_k$, $s_l$) et tests d'intersection avec les nouveaux voisins ($s_l/s_j$ et $s_k/s_m$).




- point bas : suppression du segment associé et tests d'intersection des nouveaux voisins ($s_k$ et $s_m$).



On note $\Omega$ la queue des (points) évènements.

La fonction associée doit prendre la prochain évènement et le retourner pour être traité.

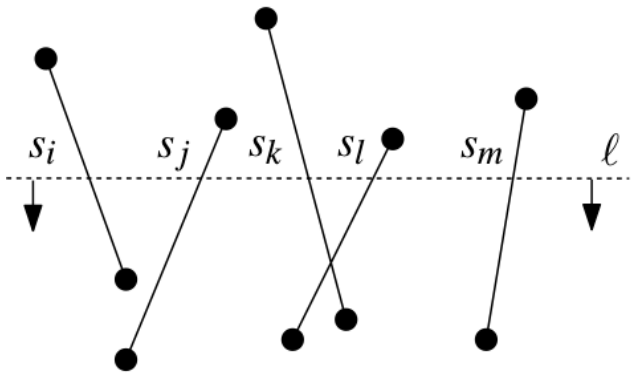
On définit un ordre $\prec$ sur les PE : soient $p$ et $q$ deux PE, $p \prec q$ ssi $p_y > q_y$ (ou $p_x < q_x$ si $p_y = q_y$).

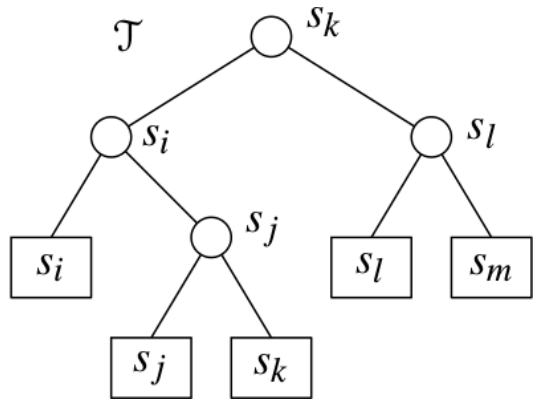
Il peut y avoir plusieurs PE qui coïncident, appartenant à des segments différents mais ayant un sommet commun (durant une insertion, il doit donc y avoir vérification d'existence).

En pratique, on stocke les PE dans un arbre binaire de recherche équilibré. Pour tout PE $p \in \Omega$, on stocke les segments commençant par $p$. Chercher ou insérer un PE a donc un coût en $O(\log m)$ avec $m$ le nombre de PE.

De même, on doit mettre en place la SD qui tient à jour la liste des segments intersectés par la ligne de balayage.

Là encore on s'appuiera sur un arbre binaire de recherche équilibré $\tau$ qui tient à jour au niveau des feuilles la position relative, de gauche à droite, des segments le long de la ligne de balayage.




Remarque : chaque nœud stocke le segment de la feuille la plus à droite de son sous-arbre gauche pour guider la recherche.

**Entrées :** un ensemble $S$ de segments du plan
**Output :** l'ensemble des intersections des segments de $S$, avec pour chaque intersection les segments qui la contiennent

initialiser la queue d'évènement $\Omega$ (avec les sommets)
initialiser le status de la ligne de balayage $\tau$ à vide
**tant que** $\Omega$ *n'est pas vide* **faire**
    déterminer le prochain PE $p \in \Omega$ et le supprimer
    `traiter_evnmt`($p$)
**fin**

**Algorithme 2 :** `trouver_intersections`($S$)

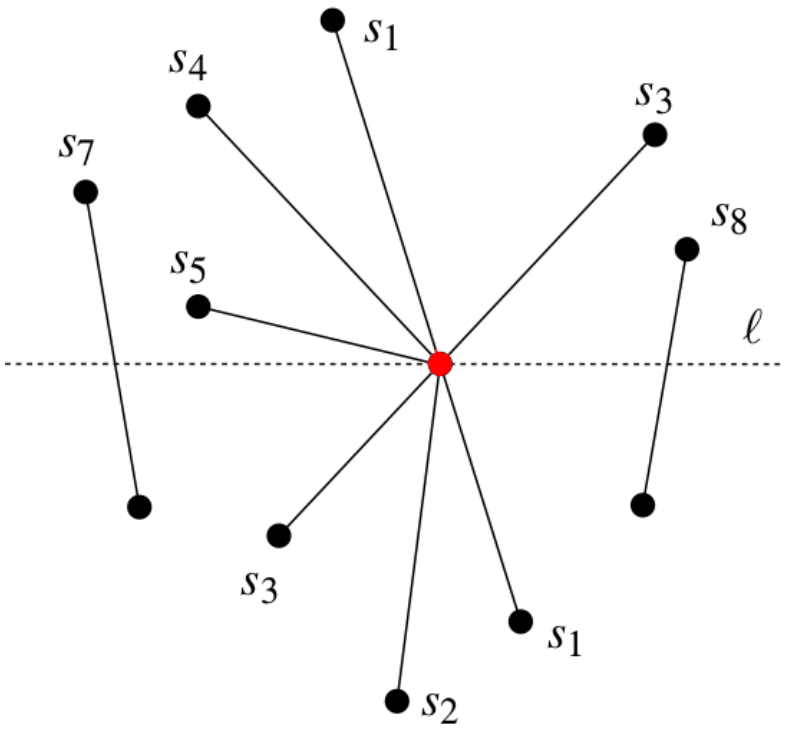
**Entrées :** PE $p \in \Omega$
**Output :** $\tau$ mis à jour
soit $U(p)$ l'ensemble des segments ayant $p \in \Omega$ comme sommet haut (si horizontaux alors sommet gauche)
trouver tous les segments dans $\tau$ contenant $p$ (ils sont adjacents)
soient $L(p)$ (resp. $C(p)$) le sous-ensemble des segments trouvés dont $p$ est le sommet bas (resp. contenant $p$)
**si** $U(p) \cup L(p) \cup C(p)$ *contient plus d'un segment* **alors**
    $p$ est un point d'intersection
**fin**
effacer les segments de $L(p) \cup C(p)$ de $\tau$
insérer les segments de $U(p) \cup C(p)$ dans $\tau$ dans l'ordre dans lequel ils *seront* intersectés par la ligne de balayage
// `effacer puis insérer = réordonner segments de` $C(p)$

**si** $U(p) \cup C(p) = \emptyset$ **alors** // segment(s) de $p$ supprimé(s)
  intersection($s_l$, $s_r$, $p$) avec $s_l$, $s_r$ voisins gauche et droite de $p$
**sinon**
  intersection($s_l$, $s'$, $p$) avec $s'$ le segment le plus à gauche de $U(p) \cup C(p)$ dans $\tau$ et $s_l$ le voisin gauche de $s' \in \tau$
  intersection($s''$, $s_r$, $p$) avec $s''$ le segment le plus à droite de $U(p) \cup C(p)$ dans $\tau$ et $s_r$ le voisin droite de $s'' \in \tau$
**fin**

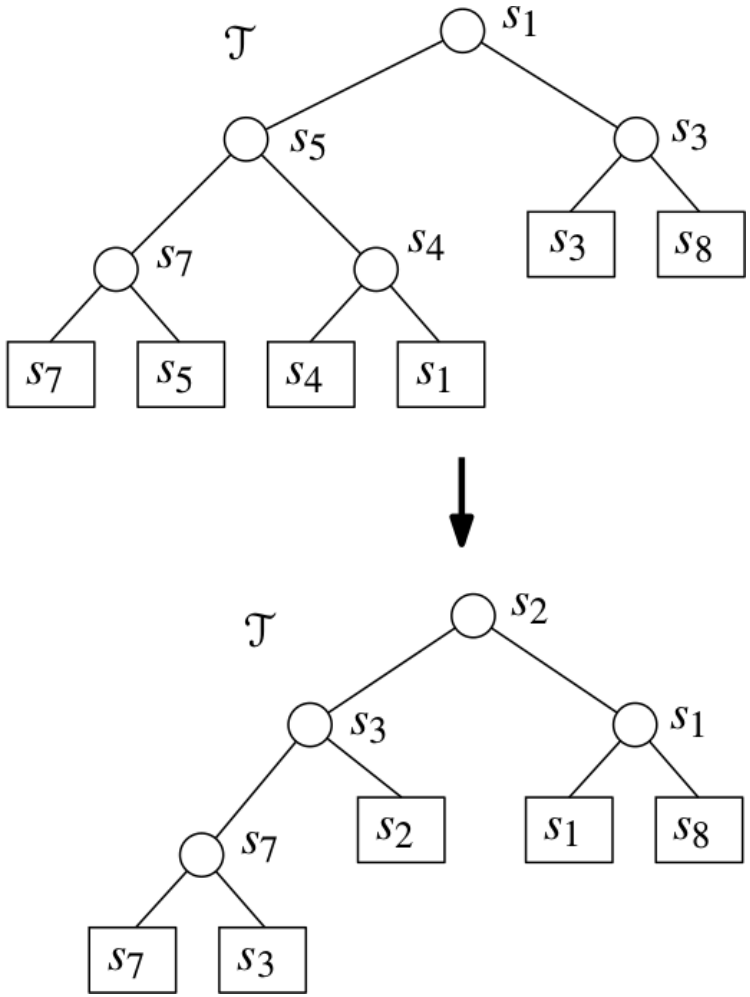
**Algorithme 3 :** traiter_evnmt($p$)

# Intersection de segments

## Algorithme `traiter_evnmt`




$U(p) = \{?\}$
$C(p) = \{?\}$
$L(p) = \{?\}$



Il ne faut traiter que les nouveaux points d'intersection et si les segments sont horizontaux ceux à droite du point $p$ (par convention).

**si** *$s_l$ et $s_r$ s'intersectent en $i$ sous la ligne de balayage ou sur la ligne à droite de $p$ et $i \notin \Omega$* **alors**
    insérer $i$ dans $\Omega$;
**fin**

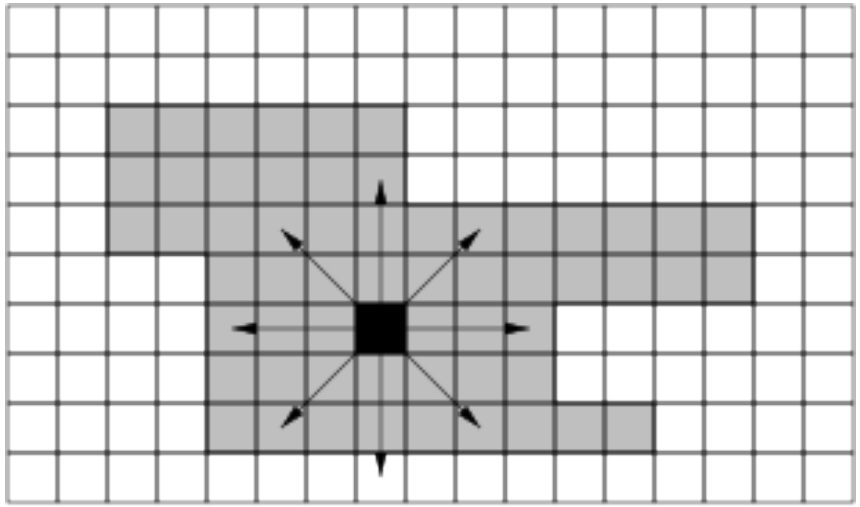
**Algorithme 4 :** `intersection`$(s_l, s_r, p)$

# Plan

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

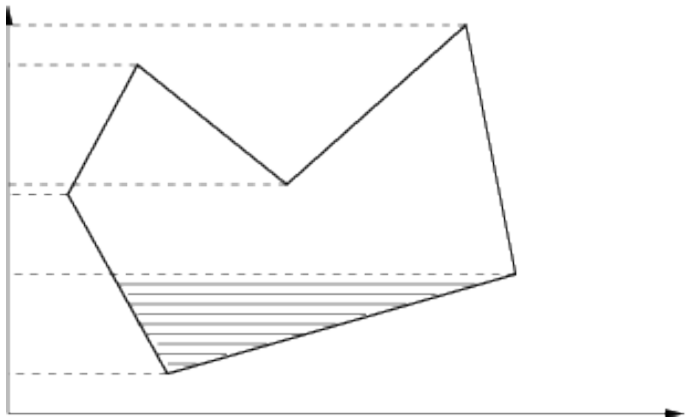
Une fois la frontière G-D déterminée, tracé d'un segment horizontal et poursuite de l'algorithme pour tous les pixels situés au dessus et en dessous de la ligne qui vient d'être tracée.

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

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).



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.

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.

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→ |
|---|---|---|---|---|

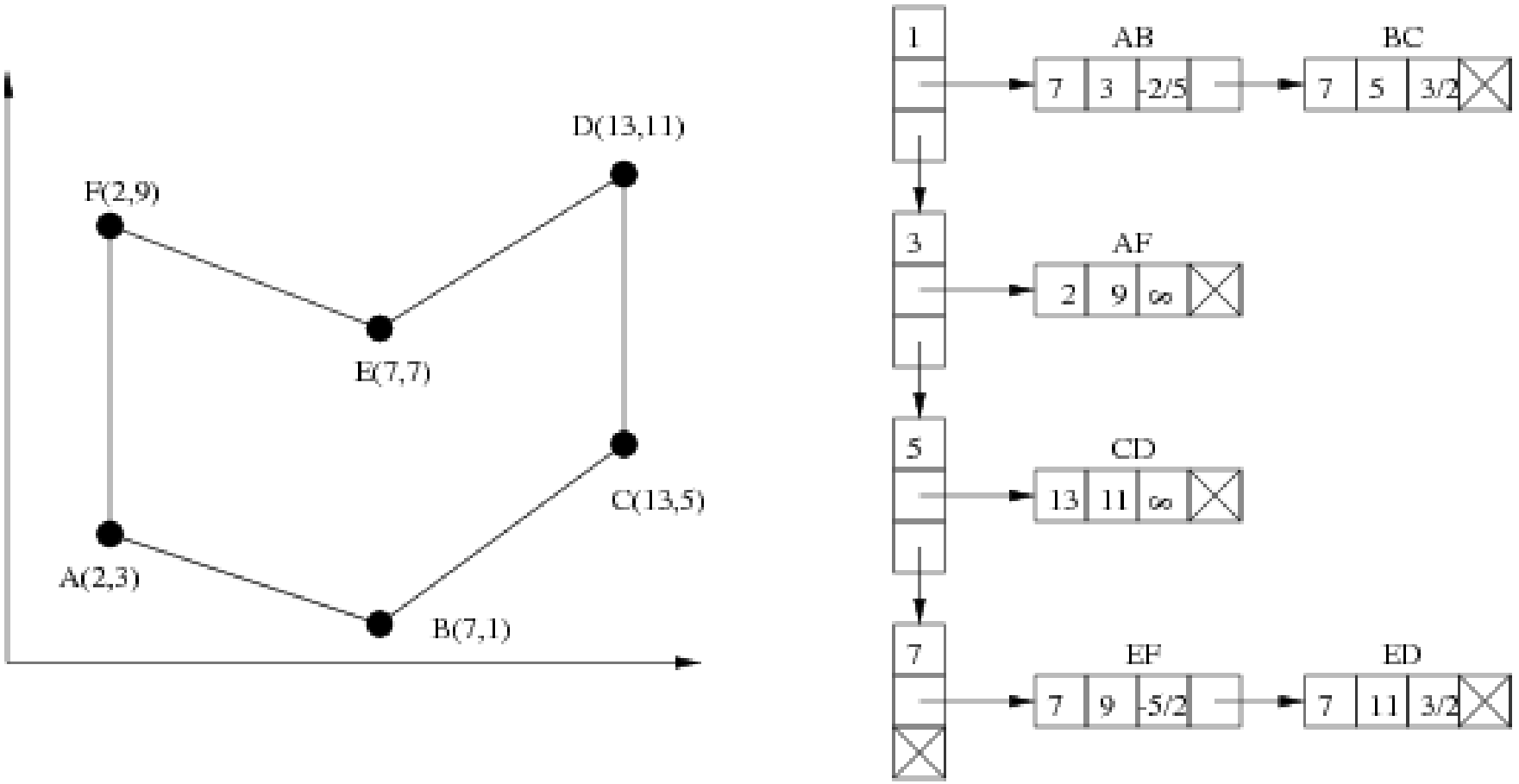


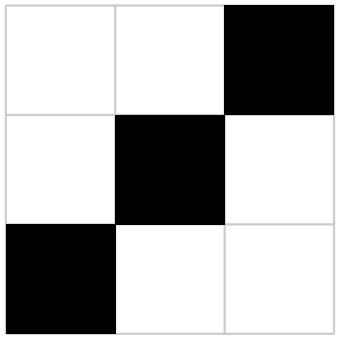

FIGURE – Un polygone et sa TA

**Entrées :** *pol* : polygone
**Données :** *TA*, *TAA* : listes d'arêtes
y, Ymax : entiers
ajout : booléen

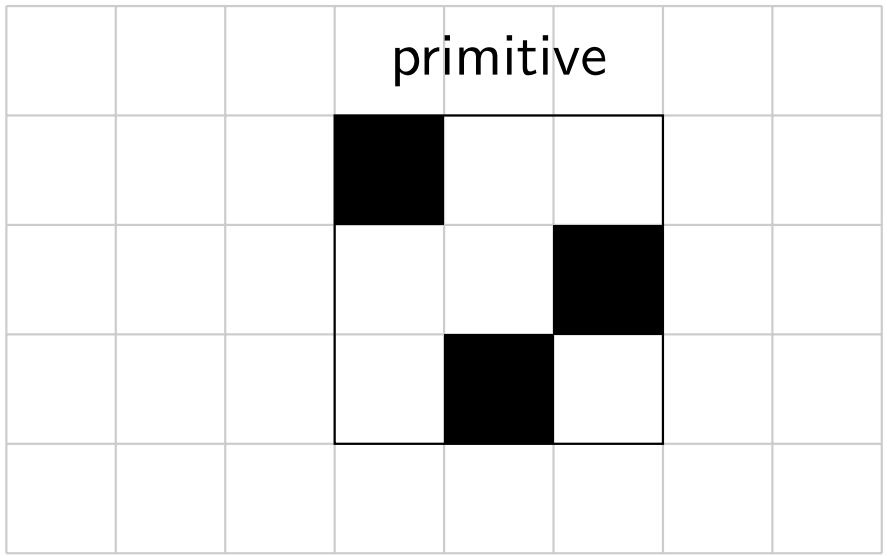
**Algorithme 5 :** Remplissage Geometrique

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 (ex : $x_0 = 3\%3 = 0$, $y_0 = 1\%3 = 1$),



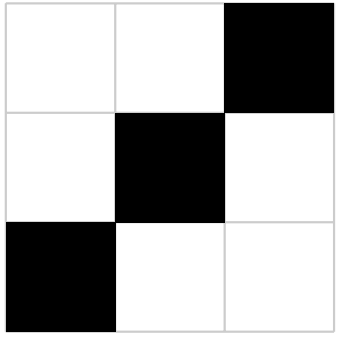
motif




écran

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

motif



masque

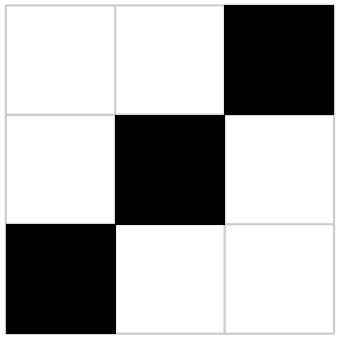
| | | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

primitive

| | | |
|---|---|---|
| 0&1 | 1&1 | 1&0 |
| 1&1 | 0&0 | 1&1 |
| 1&0 | 1&1 | 0&1 |

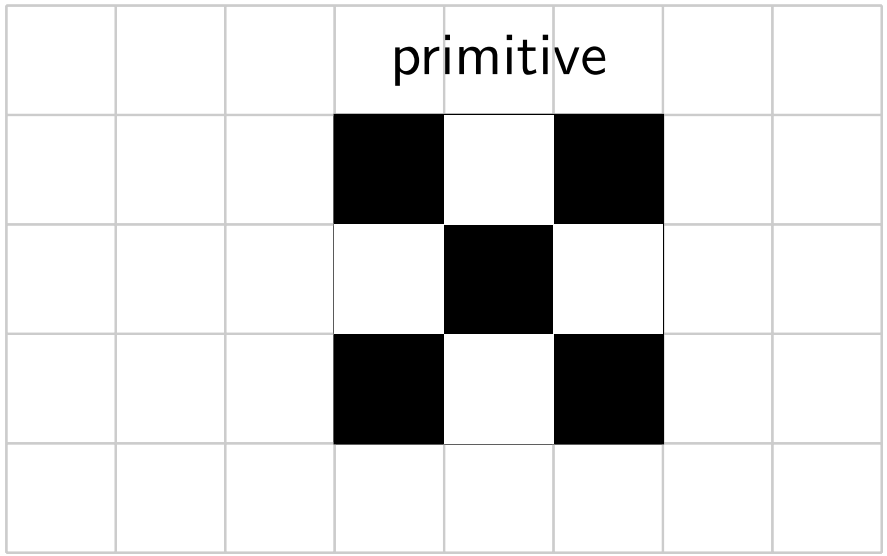
- avant la discrétisation par balayage : définition d'un masque binaire associée à la primitive (0 pour la primitive, 1 pour le reste), opérateur `ET` pour le masque puis opérateur `OU` pour la primitive.

motif



masque

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |



# Plan

# Introduction

La superposition dans les SIG de réseaux représentés par des collections de segments est le cas le plus simple à résoudre.

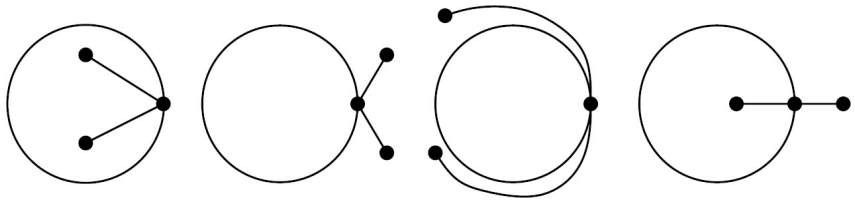
La superposition de régions (végétation, pluviométrie, . . .) est bien plus complexe et tout autant, si ce n'est plus, utilisée.

En géométrie, les graphes ne suffisent pas à capturer les propriétés des objets. Par exemple, dans le cas des maillages, il est usuel de parler de faces.

Cela nécessite une structure de données adaptée : la *structure de demi-arêtes* (half-edge), une représentation de la combinatoire des *cartes planaires*.

Étant donné un graphe $G$, une carte planaire est un plongement de $G$ dans $R^2$, qui satisfait les conditions suivantes :

1. Les sommets du graphe sont représentés par des points.
2. Les arêtes sont représentées par des arcs de courbes ne se coupant qu'aux sommets.
3. Le complément $S \setminus G$ est une union disjointe de régions appelées faces, ne contenant ni sommets ni arêtes, qui ont toutes sauf une [1] la topologie d'un disque ouvert.



Quatre plongements d'une même carte planaire

1. cette dernière (appelée face externe ou face infinie) ayant la topologie d'un disque ouvert avec un trou

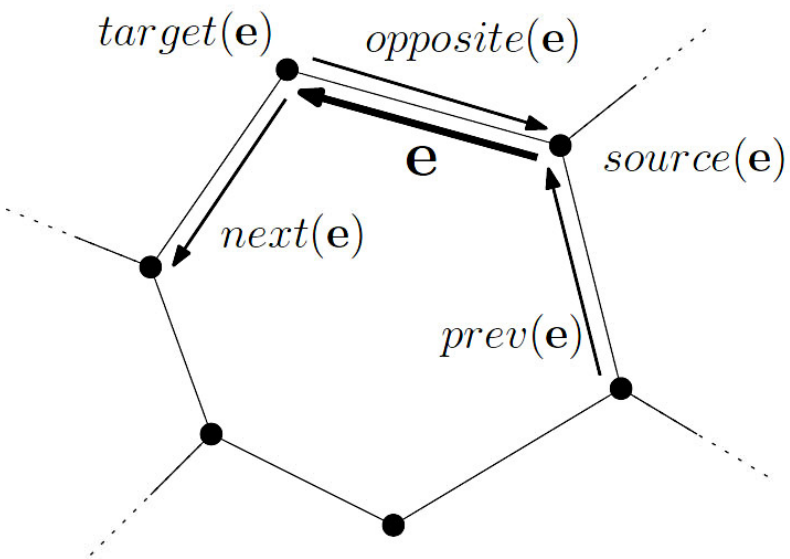
Informations topologiques : quels segments limitent une région données, quelles régions sont adjacentes, . . .

Chaque arête est décomposée en deux demi-arêtes, d'orientations opposées et incidentes à un sommet source et à un sommet destination.

Représentation d'une demi-arête :

- une référence à la face incidente, ainsi qu'aux sommets source et destination,
- une référence à la demi-arête opposée,
- une référence vers la demi-arête qui suit et celle qui précède, dans la même face incidente.

Chaque face et sommet est représenté en stockant une référence vers l'une des demi-arêtes incidentes.



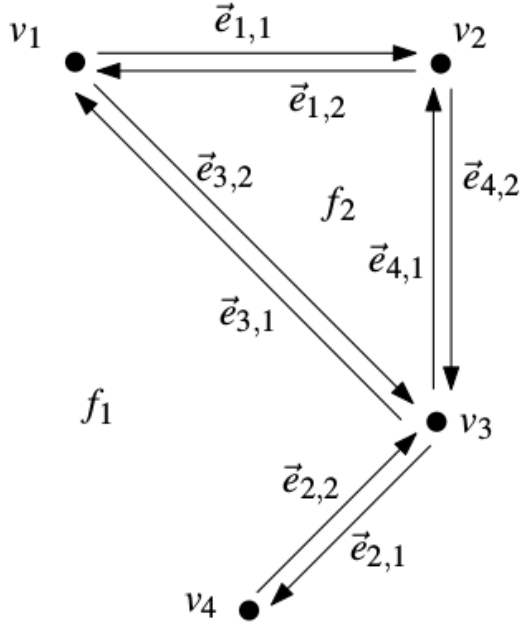

demi-arêtes

```
class Halfedge{
  Halfedge  prev, next, opposite;
  Vertex source, target;
  Face f;
}
 class Vertex{
  Halfedge  e;
  Point p;
}
 class Face{
  Halfedge  e;
}
```

information combinatoire

```
class Point{
  double x;
  double y;
}
```

information géométrique




| Vertex | Coordinates | IncidentEdge |
|---|---|---|
| $v_1$ | $(0,4)$ | $\vec{e}_{1,1}$ |
| $v_2$ | $(2,4)$ | $\vec{e}_{4,2}$ |
| $v_3$ | $(2,2)$ | $\vec{e}_{2,1}$ |
| $v_4$ | $(1,1)$ | $\vec{e}_{2,2}$ |

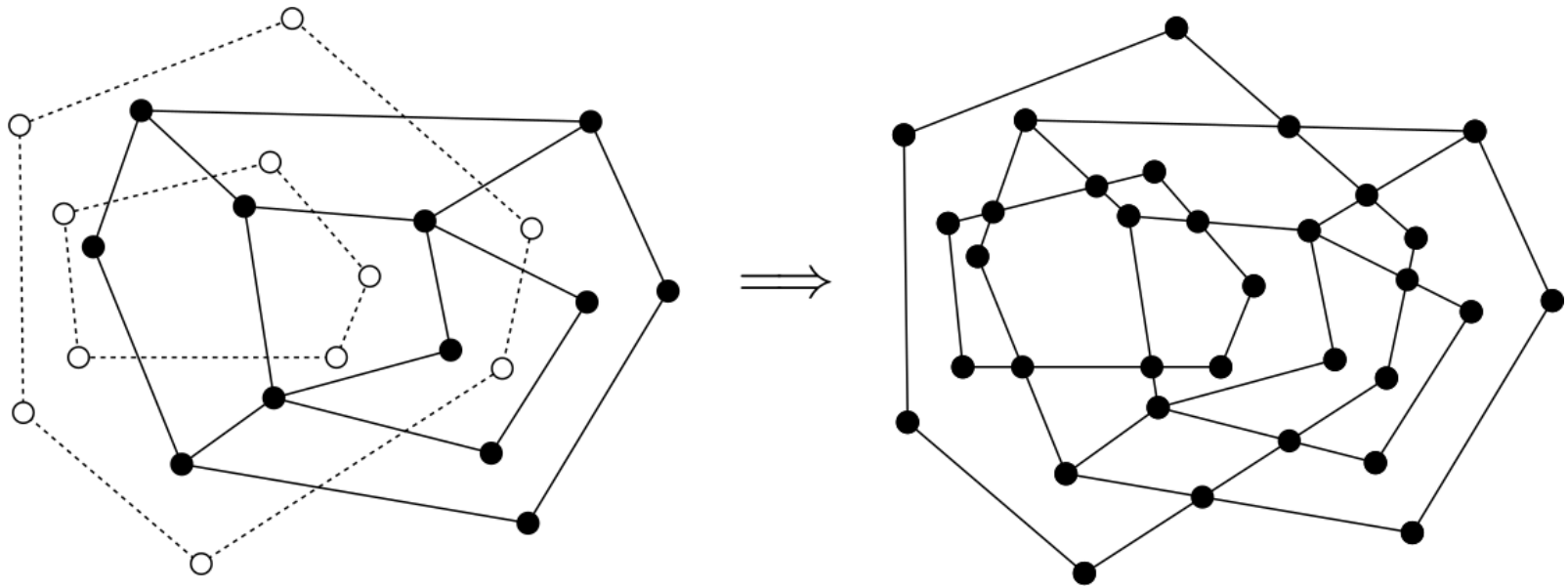
| Face | OuterComponent | InnerComponents |
|---|---|---|
| $f_1$ | **nil** | $\vec{e}_{1,1}$ |
| $f_2$ | $\vec{e}_{4,1}$ | **nil** |

| Half-edge | Origin | Twin | IncidentFace | Next | Prev |
|---|---|---|---|---|---|
| $\vec{e}_{1,1}$ | $v_1$ | $\vec{e}_{1,2}$ | $f_1$ | $\vec{e}_{4,2}$ | $\vec{e}_{3,1}$ |
| $\vec{e}_{1,2}$ | $v_2$ | $\vec{e}_{1,1}$ | $f_2$ | $\vec{e}_{3,2}$ | $\vec{e}_{4,1}$ |
| $\vec{e}_{2,1}$ | $v_3$ | $\vec{e}_{2,2}$ | $f_1$ | $\vec{e}_{2,2}$ | $\vec{e}_{4,2}$ |
| $\vec{e}_{2,2}$ | $v_4$ | $\vec{e}_{2,1}$ | $f_1$ | $\vec{e}_{3,1}$ | $\vec{e}_{2,1}$ |
| $\vec{e}_{3,1}$ | $v_3$ | $\vec{e}_{3,2}$ | $f_1$ | $\vec{e}_{1,1}$ | $\vec{e}_{2,2}$ |
| $\vec{e}_{3,2}$ | $v_1$ | $\vec{e}_{3,1}$ | $f_2$ | $\vec{e}_{4,1}$ | $\vec{e}_{1,2}$ |
| $\vec{e}_{4,1}$ | $v_3$ | $\vec{e}_{4,2}$ | $f_2$ | $\vec{e}_{1,2}$ | $\vec{e}_{3,2}$ |
| $\vec{e}_{4,2}$ | $v_2$ | $\vec{e}_{4,1}$ | $f_1$ | $\vec{e}_{2,1}$ | $\vec{e}_{1,1}$ |

# Recouvrement de deux subdivisions

Soient deux subdivisions $S_1$ et $S_2$, soient les faces $f_1 \in S_1$ et $f_2 \in S_2$, $f$ est une face du recouvrement $O(S_1, S_2)$ (overlay) ssi $f$ est le sous-ensemble maximal de $f_1 \cap f_2$.

Traduction : le recouvrement est une subdivision du plan induite par les arêtes de $S_1$ et de $S_2$.



# Recouvrement de deux subdivisions

L'objectif est de calculer une structure de demi-arêtes pour $O(S_1, S_2)$ connaissant celles de $S_1$ et de $S_2$.

Chaque face $f \in O(S_1, S_2)$ doit référencer les faces de $S_1$ et de $S_2$ qui la contiennent. Ainsi, on peut accéder aux informations associées (végétations, précipitations, ...).

La plupart des informations restent inchangées concernant :

- l'orientation (les faces restent « à gauche » des demi-arêtes qu'elles délimitent),
- les demi-arêtes qui ne s'intersectent pas dans le recouvrement.

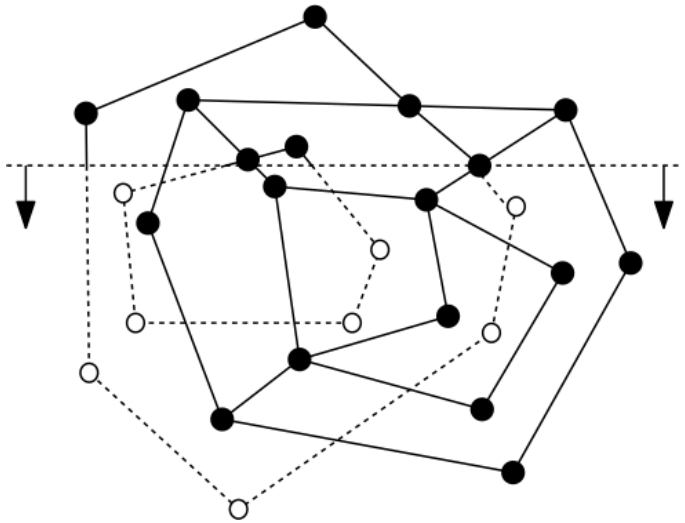
Seules les demi-arêtes incidentes à une intersection doivent être mises à jour pour le recouvrement.

# Recouvrement de deux subdivisions

Ceci suggère l'approche suivante :

1. Copier les demi-arêtes de $S_1$ et $S_2$ dans une unique structure,
2. Rendre valide cette structure pour $O(S_1, S_2)$ en calculant leurs intersections,
3. Mettre à jour les informations des faces.

On utilisera pour cela un algorithme basé sur la ligne de balayage.

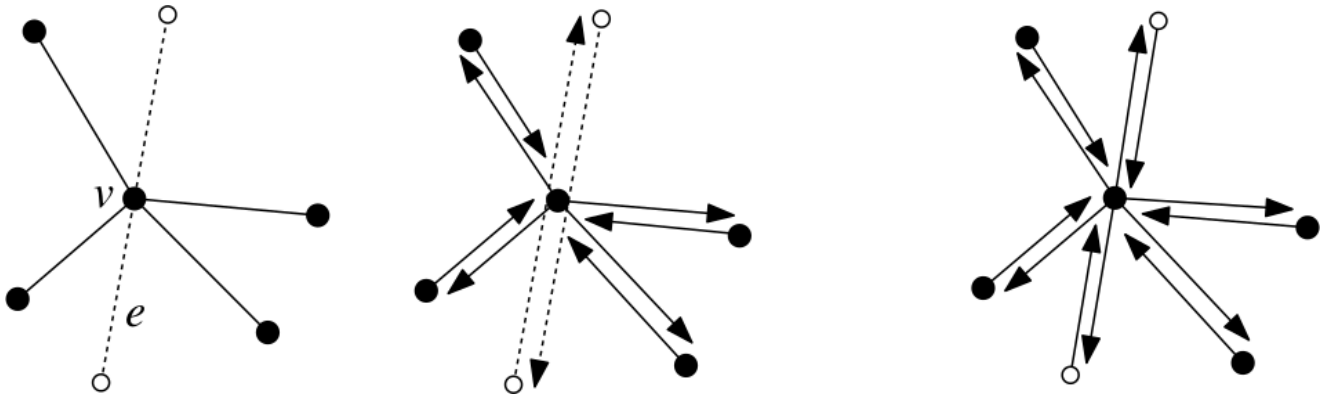


# Recouvrement de deux subdivisions

En plus des structures $\Omega$ et $\tau$, on doit gérer une structure $\mathcal{D}$ qui contient initialement l'ensemble des demi-arêtes de $S_1$ et $S_2$.

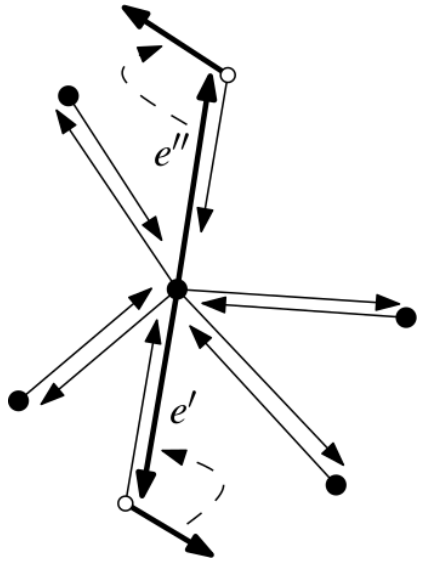
Cette structure $\mathcal{D}$ doit être mise à jour lorsque la ligne de balayage rencontre un *point d'intersection*, qui met en jeu une arête de chaque subdivision.

Étudions l'un des cas, lorsqu'une arête $e \in S_1$ intersecte un sommet $v \in S_2$.



# Recouvrement de deux subdivisions

1. L'arête $e$ doit être divisée en deux arêtes $e'$ et $e''$ (soient quatre demi-arêtes : deux ayant les sommets de $e$ comme origine, les deux autres ayant $v$ comme origine).
2. On apparie (*opposite* ou *twin*) les demi-arêtes d'origine $e$ avec les demi-arêtes d'origine $v$.
3. Les champs *next* des demi-arêtes d'origine $v$ réfèrent les demi-arêtes initiales de $e$ (maj de leur champ *prev*).



# Recouvrement de deux subdivisions

4. Mise à jour des champs *prev* et *next* des quatre demi-arêtes $e'$, $e''$ et des huit demi-arêtes incidentes à $v$ : par rotation autour de $v$ dans le sens trigonométrique ou horaire.

Seul ce dernier cas se fait en temps linéaire, proportionnel au degré de $v$ (les autres cas sont résolus en temps constant).

Les autres situations, intersections d'arêtes ou sommets coïncidants, sont résolues en temps linéaire.

Ainsi, la mise à jour de $\mathcal{D}$ se fait-elle en $O(n \log n + k \log n)$ avec $n$ somme des composantes de $S_1$ et $S_2$ et $k$ nombre de composantes du recouvrement.

# Recouvrement de deux subdivisions

## Faces de $O(S_1, S_2)$

Pour chaque face $f$, on doit avoir :

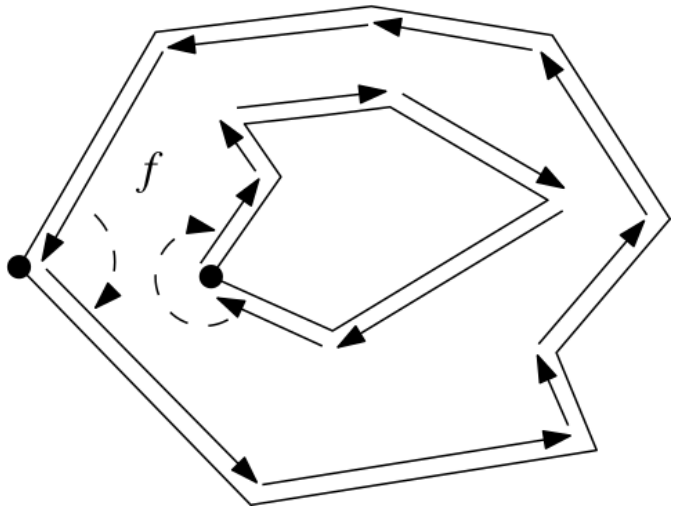
- une référence à une demi-arête de sa frontière extérieure,
- éventuellement une référence à une demi-arête de chaque frontière intérieure (ou trou),
- réciproquement la référence à la face pour chaque demi-arête de la frontière de $f$ (mise à jour).

De plus, chaque nouvelle face doit être identifiée à partir des faces de l'ancienne subdivision qui la contiennent.

# Recouvrement de deux subdivisions

## Faces de $O(S_1, S_2)$

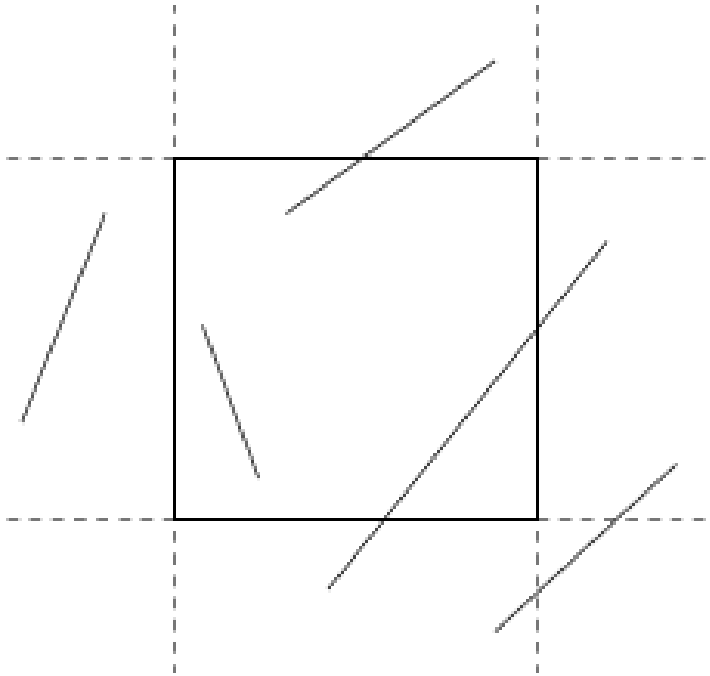
Pour distinguer les contours extérieurs et intérieurs, il suffit de considérer les deux demi-arêtes incidentes au sommet le « plus à gauche » d'un contour et l'angle qu'elles forment (inf. ou sup. à 180°) entre elles.



# Plan

# Fenêtrage de segments

Plusieurs cas se présentent :



Le fenêtrage des points extrémités répondent au critère :

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

# Fenêtrage 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.

# Fenêtrage 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.

# Fenêtrage de segments

**Entrées :** seg un segment, fen une fenêtre
reg1 $\longleftarrow$ region(seg.P1, fen)
reg2 $\longleftarrow$ region(seg.P2, fen)
**tant que** *reg*1 $\neq 0$ *ou reg*2 $\neq 0$ *et (reg1 AND reg2 = 0)* **faire**
  **si** *reg*1 $\neq 0$ **alors** `// P1 à l'extérieur`
    d $\longleftarrow$ cote(reg1) `// on anticipe sur la modification`
    seg.P1 $\longleftarrow$ intersection(seg, d)
  **sinon** `// reg2 ≠ 0 et P2 à l'extérieur`
    d $\longleftarrow$ cote(reg2) `// on anticipe sur la modification`
    seg.P2 $\longleftarrow$ intersection(seg, d)
  **fin**
**fin**
**si** *reg1 AND reg2* $\neq 0$ **alors** `// cas segment rejeté`
  seg.P1 $\longleftarrow$ seg.P2 $\longleftarrow$ None
**fin**

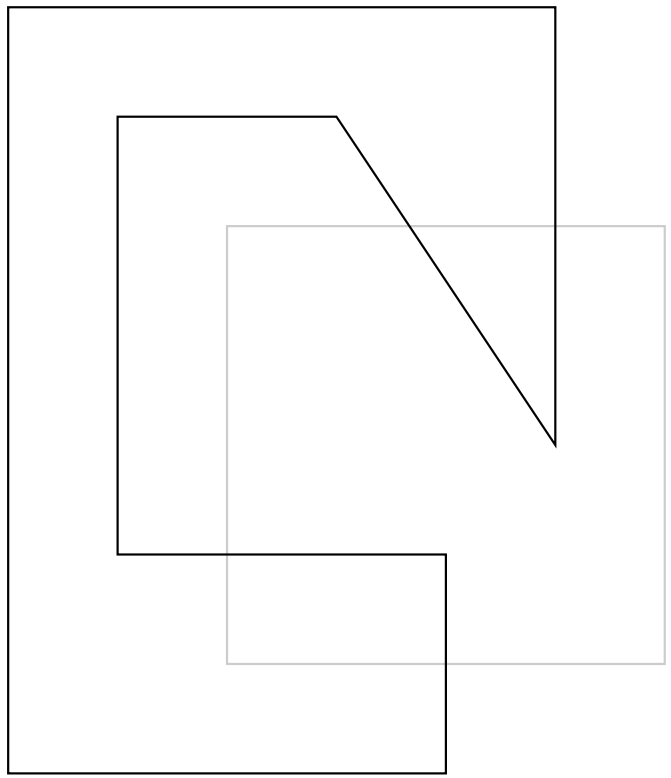
**Algorithme 6 :** `clipping_segment`(seg, fen)

# Fenêtrage 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

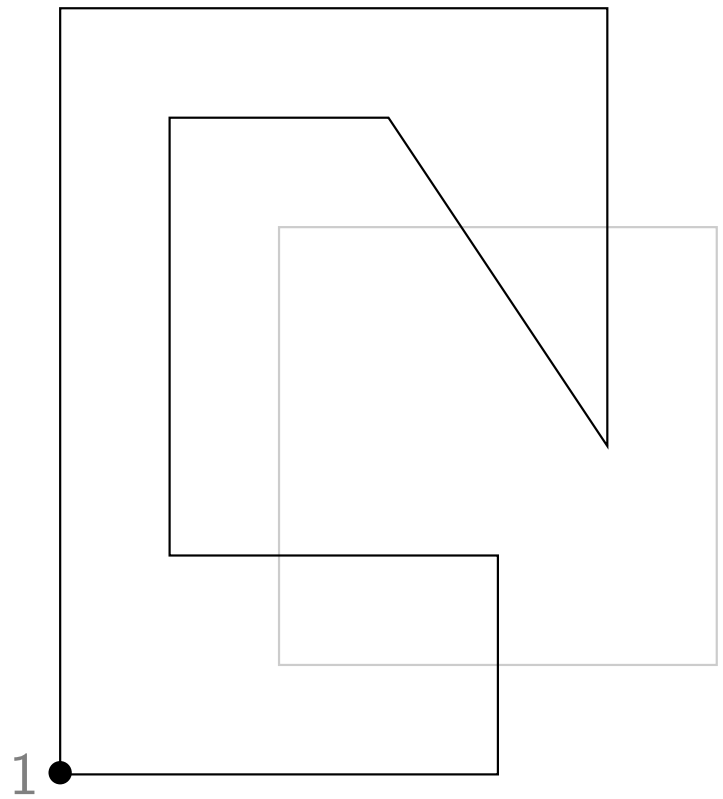


# Fenêtrage de polygones

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

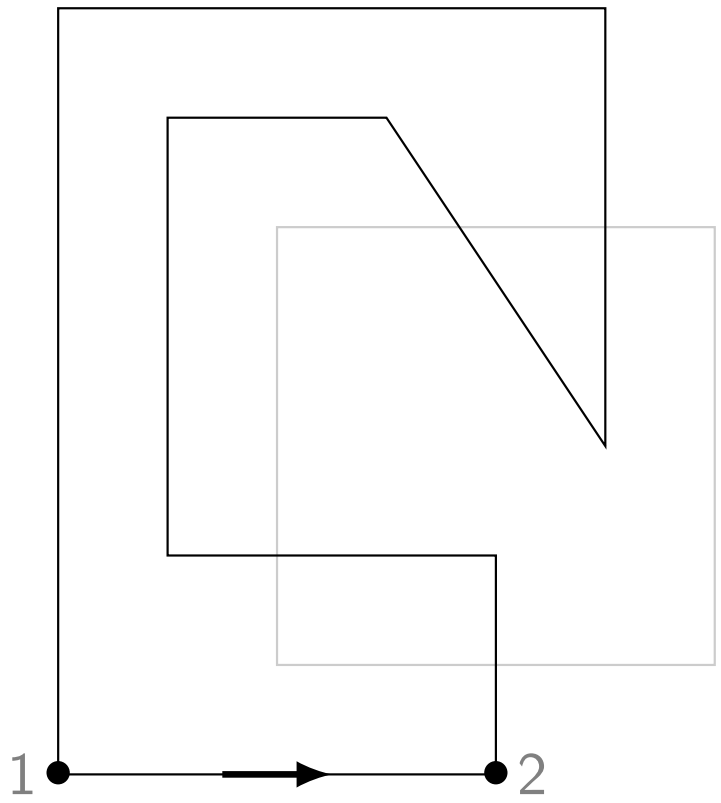


# Fenêtrage de polygones

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

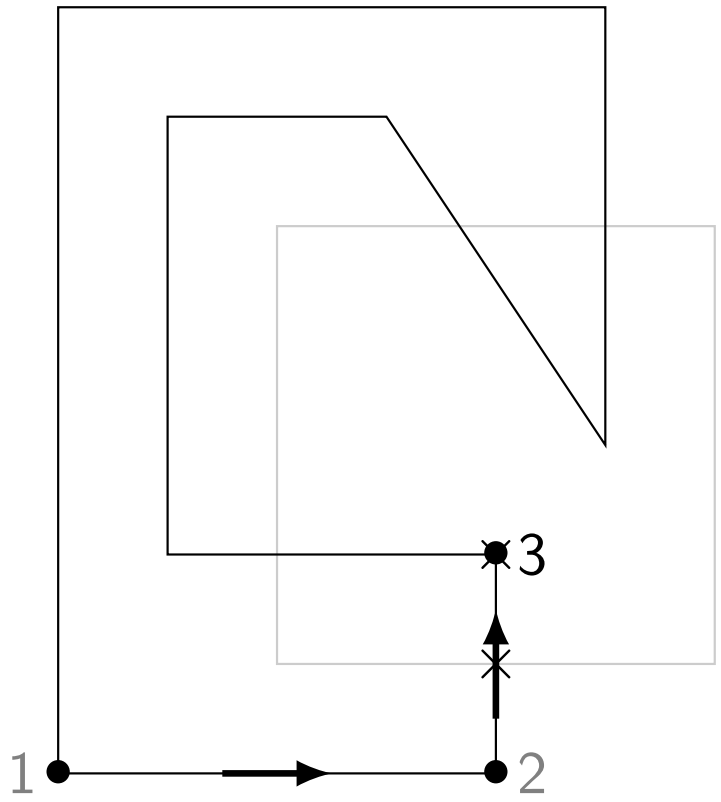


# Fenêtrage de polygones

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

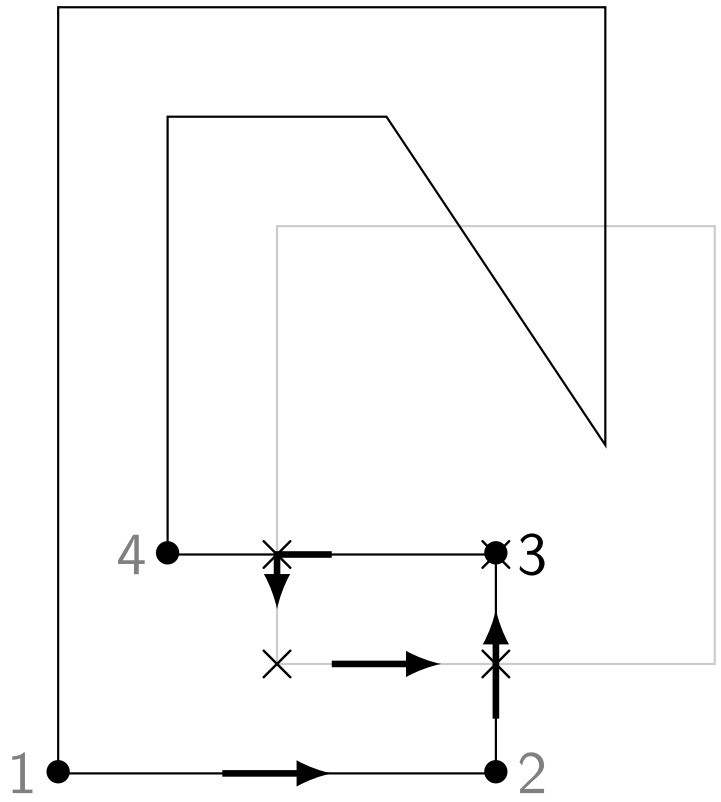


# Fenêtrage de polygones

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

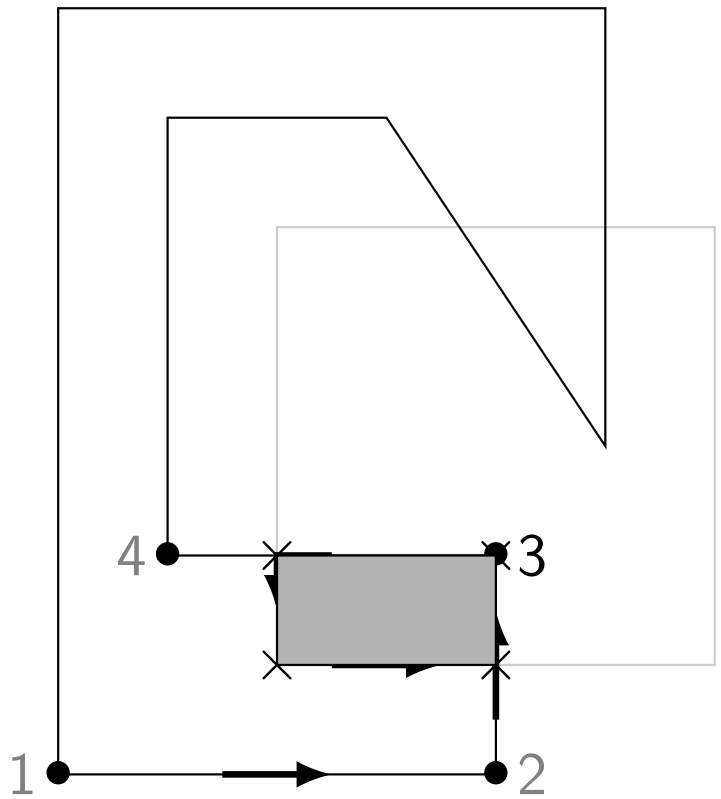


# Fenêtrage de polygones

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



# Fenêtrage 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