Chapitre 3 : remplissage de régions et de polygones

1 Présentation

Le remplissage d'une zone est le "coloriage" de cette zone ou région.

Les zones peuvent être définies au niveau des pixels ou au niveau géométriques.

1) Au niveau des pixels : la zone est décrite
   a) soit par l'ensemble des pixels qui la composent, la zone est dite surfacique.
   Utilisation d'un algorithme de remplissage = algorithme de coloriage
   b) soit par les pixels qui la bordent, la région est un contour.

2) Au niveau géométrique : la zone est décrite en terme d'objets,
segments, polygones, circonférences.

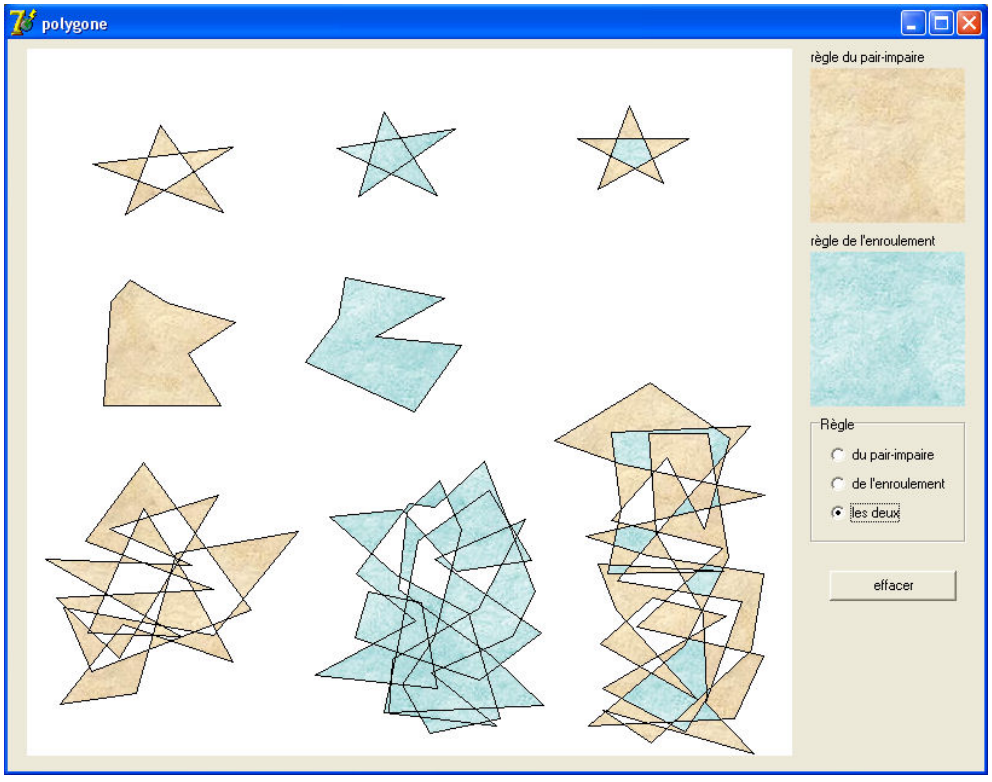


FIGURE 1 – Remplissage de polygones

Il existe deux règles pour déterminer si un point est dans un polygone. On considère une demi-droite partant du point et allant vers l'infini dans n'importe quelle direction et on s'intéresse aux intersections de cette demi-droite et des arêtes du polygone.

1. Dans la première règle, on se contente de compter les intersections. Si le compte est pair, le point est à l'extérieur du polygone, sinon, il est à l'intérieur (règle pair/impair). Voir cours Cyrus Beck.
2. Dans la seconde règle, on oriente les arêtes du polygone et on ajoute 1 chaque fois qu'une arête coupe la demi-droite dans le sens des aiguille d'une montre et -1 dans l'autre sens. Si la somme est nulle, le point est à l'extérieur, sinon, il est à l'intérieur (règle du nombre d'enroulement non nul).

Remarque : Une région n'est pas forcément définie par un polygone

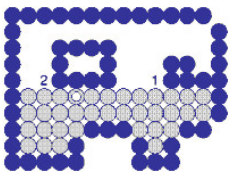


FIGURE 2 – Début d'un remplissage d'une région

2 Remplissage d'une région donnée par son contour

2.1 Principe

Ce type d'algorithme traite les régions définies par une frontière.
A partir d'un pixel (germe) intérieur à la région, on propage récursivement la couleur de remplissage au voisinage de ce pixel jusqu'à atteindre la frontière.

Remarque : ce traitement récursif (très lourd) pourra être traduit en un algorithme itératif (plus efficace) à l'aide d'une gestion de piles. Pour chaque itération, on empilera chaque nouveau germe trouvé et on procédera à un remplissage par balayage ligne par ligne de la région jusqu'au dernier germe de la pile.

2.2 Remplissage récursif : algorithme à germes

2.2.1 Présentation

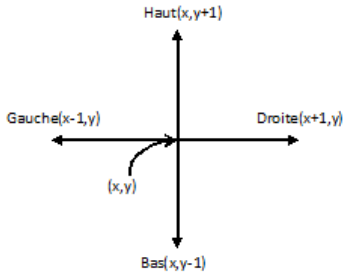
On souhaite remplir avec la couleur de remplissage (CR) une région dont le contour est formé par des pixels de couleur contour (CC).

Si des points de la surface sont de couleur CR ou CC, ils ne seront pas modifiés.

Cette propriété permet de définir des régions avec trous à l'aide de contours intérieurs de la région.

2.2.2 Algorithme de remplissage de connexité 4

Connexité 4 : deux pixels de la région peuvent être joints par une séquence de déplacements dans les quatre directions.



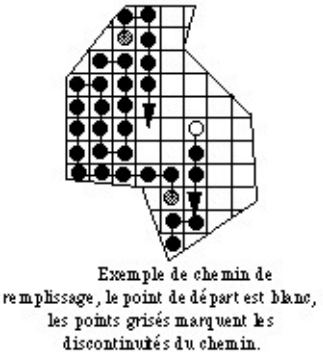


FIGURE 3 – Parcours récursif des germes remplissant une région de connexité 4

RemplissageRégionConnexité4(x, y,CC,CR)
PF :
CC : couleur du contour
CR : couleur du remplissage (couleur de la région)
x, y : coordonnées du px initial = germe
VI :
CP : couleur du pixel courant
Début
CP⟵CouleurPixel(x,y)
Si ((CP $\neq$ CC) et (CP $\neq$ CR)) **Alors**
AffichePixel(x, y,CR)
RemplissageRégionConnexité4($x, y-1$,CC,CR) /* Bas */
RemplissageRégionConnexité4($x-1, y$,CC,CR) /* Gauche */
RemplissageRégionConnexité4($x, y+1$,CC,CR) /* Haut */
RemplissageRégionConnexité4($x+1, y$,CC,CR) /* Droite */
FinSi
Fin

Remarques :

- L'utilisation de la récursivité entraîne très vite des problèmes de dépassements de capacités de la pile si les zones remplies sont trop grande.
  Pile d'appel (ou pile, ou pile système) : dans l'architecture des microprocesseur, pile particulière dans laquelle sont poussés les paramètres d'appels des procédures/fonctions ainsi que l'adresse de retour. On y stocke aussi les variables locales des fonctions.
- Si le contour de la forme n'est pas fermé, le remplissage déborde
- Avec 8 connexités, on effectue 8 appels récursifs (Est, Nord Est, Nord, Nord Ouest, Ouest, ...).

2.3 Première version itérative

RemplissageRégionConnexité4(x, y,CC,CR)
PF :
CC : couleur du contour
CR : couleur du remplissage (couleur de la région)
x, y : coordonnées du px initial = germe
VI :
CP : couleur du pixel courant
p : pile
Début
$p \longleftarrow$initPile() /* initialisation de la pile à vide */
empiler($(x, y), p$) /* on empile (x, y) dans p */
Tant que (non(pileVide(p))) **Faire** /* un germe à traiter */
$(x, y) \longleftarrow$ sommetPile(p) /* stockage du sommet */
dépiler(p)
CP$\longleftarrow$CouleurPixel(x,y)
Si ((CP $\neq$ CC) et (CP $\neq$ CR)) **Alors**
AffichePixel(x, y,CR)
FinSi
CP$\longleftarrow$CouleurPixel(x,y-1)
Si ((CP $\neq$ CC) et (CP $\neq$ CR)) **Alors**
empiler($(x, y-1), p$)
FinSi
CP$\longleftarrow$CouleurPixel(x-1,y)
Si ((CP $\neq$ CC) et (CP $\neq$ CR)) **Alors**
empiler($(x-1, y), p$)
FinSi
CP$\longleftarrow$CouleurPixel(x,y+1)
Si ((CP $\neq$ CC) et (CP $\neq$ CR)) **Alors**
empiler($(x, y+1), p$)
FinSi
CP$\longleftarrow$CouleurPixel(x+1,y)
Si ((CP $\neq$ CC) et (CP $\neq$ CR)) **Alors**
empiler($(x+1, y), p$)
FinSi
FinTQ
Fin

Remarque : le stockage des germes dans la pile de traitement peut être réduit. Le remplissage par ligne va permettre de diminuer la charge de la pile et ainsi d'optimiser l'algorithme.

2.4 Seconde version itérative : remplissage par ligne

Principe :

- Détection d'un pixel initial = germe (x, y), empilé dans la pile de traitement
- Pour chaque pixel (x, y) de la pile
  - · Calculer les pixels frontières (x_g, y) et (x_d, y) pour la ligne y
  - · Afficher tous les pixels Pi connectant (x_g, y) et (x_d, y)
  - · Parmi les pixels du dessus $(x, y+1)$ ou du dessous $(x, y-1)$ des Pi , avec $x_g \leqslant x \leqslant x_d$, on recherche ceux qui sont le plus à droite de chaque segment horizontal à afficher.
  - · On empile ces pixels qui seront traités dans les itérations suivantes

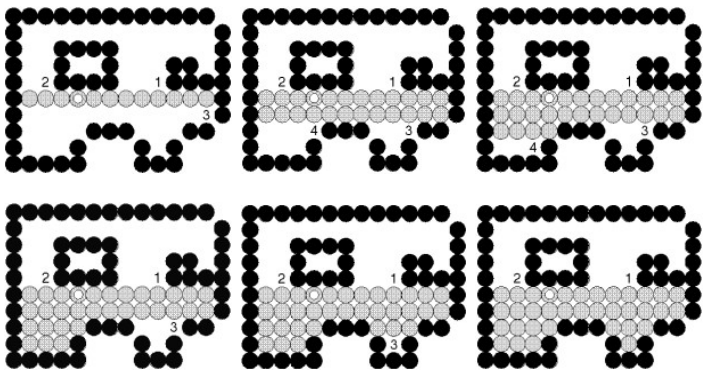


FIGURE 4 – Remplissage à l'aide des pixels du dessous empilés en premier

RemplissageLigne(x,y,CC,CR)
PF :
CC : couleur du contour
CR : couleur du remplissage (couleur de la région)
x, y : coordonnées du px initial = germe
VI :
CP, CPd, CPg : couleur pixel /* courant, droite, gauche */
p : pile
xd, xg : entiers
Début
p⟵initPile() /* initialisation de la pile à vide */
empiler((x,y),p) /* on empile (x,y) dans p */

Tant que (non(pileVide(p))) **Faire** /* un germe à traiter */
(x,y)⟵ sommetPile(p) /* stockage du sommet */
dépiler(p)
CP⟵CouleurPixel(x,y)

```
/* On détermine les abscisses extrêmes xg et xd de la ligne de balayage (y) du germe courant */
/* Recherche de xd : extrême à droite */
xd⟵x+1
CPd⟵CP
Tant que (CPd≠CC) Faire
        xd⟵xd+1
        CPd⟵couleurPixel(xd,y)
FinTQ
xd⟵xd-1
/* Recherche de xg : extrême à gauche */
xg⟵x-1
CPg⟵CP
Tant que (CPg≠CC) Faire
        xg⟵xg-1
        CPg⟵couleurPixel(xg,y)
FinTQ
xg⟵xg+1
/* Affichage de la ligne de balayage de xg à xd, avec la couleur CR */
afficheLigne(xg,y,xd,y,CR)

*/ Recherche de nouveaux germes sur la ligne de balayage au-dessus :
la recherche s'effectue entre xg et xd */
x⟵xd
CP⟵couleurPixel(x,y+1)
Tant que (x⩾xg) Faire
        Tant que (((CP=CC) ou (CP=CR)) et (x⩾xg)) Faire
                x⟵x-1
                CP⟵couleurPixel(x,y+1)
        FinTQ
        Si ((x⩾ xg) et (CP≠CC) et (CP≠CR)) Alors
                /* On empile le nouveau germe au-dessus trouvé */
                empile((x,y+1),p)
        FinSi
        Tant que ((CP≠CC) et (x⩾xg)) Faire
                x⟵x-1
                CP⟵couleurPixel(x,y+1)
        FinTQ
FinTQ

*/ Recherche de nouveaux germes sur la ligne de balayage au-dessous :
la recherche s'effectue entre xg et xd */
x⟵xd
CP⟵couleurPixel(x,y-1)
Tant que (x⩾xg) Faire
        Tant que (((CP=CC) ou (CP=CR)) et (x⩾xg)) Faire
```

```
                    x←—x-1
                    CP←—couleurPixel(x,y-1)
                FinTQ
                Si ((x⩾ xg) et (CP≠CC) et (CP≠CR)) Alors
                    /* On empile le nouveau germe au-dessous trouvé */
                    empile((x,y-1),p)
                FinSi
                Tant que ((CP≠CC) et (x⩾xg)) Faire
                    x←—x-1
                    CP←—couleurPixel(x,y-1)
                FinTQ
            FinTQ
        FinTQ
Fin
```

3 Remplissage de polygones

3.1 Méthode du rectangle englobant

3.1.1 Présentation

La méthode la plus simple pour remplir un polygone consiste à examiner chaque pixel de la fenêtre graphique d'affichage pour savoir s'il se trouve à l'intérieur de ce polygone.

Caractéristiques :

- simplicité de l'algorithme
- conduit à un gaspillage

Amélioration :

Réduire le nombre de pixels à traiter en déterminant un rectangle englobant le polygone. Le rectangle englobant est le plus petit rectangle qui contient le polygone à remplir ; de plus, ses côtés sont parallèles aux axes des coordonnées du repère. Ainsi, le test d'appartenance s'effectuera uniquement sur les points du rectangle englobant.

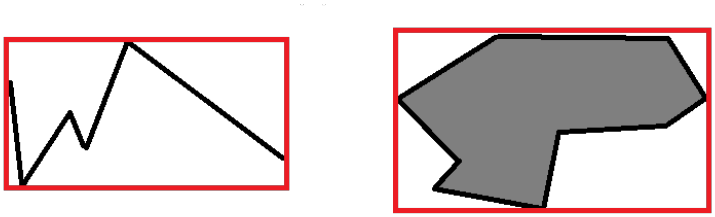


FIGURE 5 – Exemples de rectangles englobants

3.1.2 Algorithme

On se munit de deux fonctions :

1. **rectangleEnglobant**(Poly, RectEG), qui prend comme entrée Poly : le tableau (ou liste) des points du polygone et qui retourne RectEG : le tableau (ou liste) des deux points $P_1(x_{\min}, y_{\min})$, $P_2(x_{\max}, y_{\max})$ minimisant et maximisant les abscisses et les ordonnées des points de Poly.

2. **intérieur**$(x, y,$ Poly), qui prend comme entrée x, y : les coordonnées du point à traiter et Poly : le tableau (ou liste) des points du polygone. **intérieur**$(x, y,$ Poly) retourne Vrai si (x, y) est intérieur à Poly, Faux sinon.

RemplissageRectEG(Poly, RectEG, CR)
PF :
Poly : tableau (ou liste) de points
RectEG : tableau de 2 points, le 1^{er} min de x,y et le 2^{nd} max de x,y
CR : couleur de remplissage
VI :
x, y : entiers
$x_{\min}, y_{\min}, x_{\max}, y_{\max}$: entiers
Début
$x_{\min} \longleftarrow$ RectEG[1][1]
$y_{\min} \longleftarrow$ RectEG[2][1]
$x_{\max} \longleftarrow$ RectEG[1][2]
$y_{\max} \longleftarrow$ RectEG[2][2]
Pour x variant de $x_{\min}$ à $x_{\max}$ **faire**
Pour y variant de $y_{\min}$ à $y_{\max}$ **faire**
Si (intérieur$(x, y,$ Poly)) **Alors**
affichePixel(x, y)
FinSi
FinPour
FinPour
Fin

3.2 Méthode LCA : Liste des Côtés Actifs

3.2.1 Présentation

Principe :

Ce type de traitement consiste à balayer les lignes horizontales du rectangle englobant le polygone à remplir et à afficher les pixels intérieurs au polygone sur chaque ligne.

Balayage du polygone :

Les caractéristiques des pixels d'une ligne de balayage donnée changent en fonction du nombre d'intersections entre la ligne et les côtés (minimum une, qui est double).

Ces intersections divisent la ligne de balayages en deux régions :

- région intérieure : ce sont les pixels de la ligne de balayage qui sont à l'intérieur du polygone
- région extérieure : ce sont les pixels de la ligne de balayage qui sont à l'extérieur du polygone

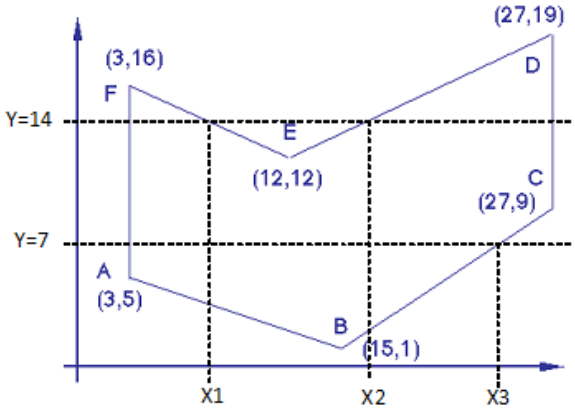


FIGURE 6 – Polygone et lignes de balayage

La ligne de balayage $Y = 7$ est divisée en 3 régions :

- $x < 3$: extérieure au polygone
- $3 \leqslant x \leqslant x_3$: intérieure au polygone
- $x > x_3$: extérieure au polygone

La ligne de balayage $Y = 14$ est divisée en 5 régions :

- $x < 3$: extérieure au polygone
- $3 \leqslant x \leqslant x_1$: intérieure au polygone
- $x_1 < x < x_2$: extérieure au polygone
- $x_2 \leqslant x \leqslant 27$: intérieure au polygone
- $x > 27$: extérieure au polygone

L'ordre des intersections de la ligne de balayage $Y = 14$ dépend de l'ordre des clicking des points du polygone.

Si le polygone est décrit par la liste ordonnée des sommets $ABCDEF$, alors :

- les intersections des côtés avec $Y = 14$ seront déterminées dans l'ordre $27, x_2, x_1, 3$
- il faudra ensuite les trier dans l'ordre des abscisses croissantes $3, x_1, x_2, 27$
- traiter les intersections par paires :
  - les pixels de 0 à 3, x_1 à x_2, 27 à 100 reçoivent la couleur du fond
  - les pixels de 3 à x_1, x_2 à 27 reçoivent la couleur de remplissage

3.2.2 Différentes étapes de l'algorithme

1. Déterminer pour chaque ligne de balayage (pixel par pixel), les points d'intersection avec le polygone.

   Rappel : pour un segment $[AB]$, de coefficient directeur m, connaissant la position du pixel $p_i(x_i, y_i)$ sur $[AB]$, on peut, par incrémentation des ordonnées, déterminer la position du pixel suivant $p_{i+1}(x_{i+1} = x_i + \frac{1}{m}, y_{i+1} = y_i + 1)$

2. Les points obtenus doivent ensuite être triés d'abord selon Y (c'est à dire par ligne de balayage), puis selon X

3. Les segments sont alors traités par paires de points d'intersections consécutifs et affichés s'ils sont intérieurs au polygone. Utiliser un bit de parité

Remarques

- Les cotés horizontaux du polygone peuvent être ignorés car ils sont parallèles aux lignes de balayage (pas d'intersection)
- Un sommet appartenant à deux cotés suit un traitement spécial :
  - +1 si l'intersection est simple
  - +2 si l'intersection est double
- Afin d'éviter de devoir trier la liste de points, lorsqu'elle est complète, il est préférable d'utiliser :
  - un arbre binaire ordonné, dans lequel chaque point sera inséré en respectant la relation binaire $\mathcal{R}$ (ordre lexicographique inversé) sur $\mathbb{R}$ suivante :
    $(x, y)\mathcal{R}(x', y') \Longleftrightarrow ((y < y') \quad \text{ou} \quad (y = y' \quad \text{et} \quad x \leqslant x'))$
  - ou une liste chaînée

3.2.3 Méthode de l'activation des côtés

Le remplissage se fait par ligne de balayage, il est suffisant de ne stocker que les intersections à cette ligne.

On décompose le polygone en tranches horizontales dans lesquelles seuls les côtés interceptés seront considères.

On utilise deux tables :

LCA = Liste des côtés actifs
= liste chaînée pour gérer une suite de côtés actifs
= liste contenant des informations sur les côtés ayant une intersection avec la ligne de balayage en cours de traitement. Chaque côté est caractérisé par :
1) $y_{\max}$: ordonnée maximale du côté
2) x courant : abscisse de l'intersection
3) $\frac{1}{m}$: l'inverse de son coefficient directeur

- Cette liste est mise à jour à chaque changement de la ligne de balayage
- Elle est triée en permanence par ordre croissant des abscisses des points d'intersection

SI = Structure intermédiaire
= tableau de listes chaînées. Chaque case du tableau correspond à un niveau de la ligne de balayage Y. Chaque maillon d'une liste correspond à un côté du polygone intercepté par Y

- Un maillon contient les informations :
  1) $y_{\max}$: ordonnée maximale du côté
  2) $x_{\min}$: abscisse minimale du côté (correspondant à $y_{\min}$)
  3) $\frac{1}{m}$: l'inverse de son coefficient directeur
- SI permet la gestion :
  1) des entrées des côtés dans LCA pour telle ou telle ordonnée
  2) des sorties des côtés dans LCA, quand $Y = y_{\max}$
- Les listes de SI sont triées par ordre croissant de $x_{\min}$, puis pour deux $x_{\min}$ identiques, par ordre croissant de la valeur $\frac{1}{m}$. Ce tri a pour but de faciliter le déplacement des cotés vers la LCA car cet ordre de classement devra être respecté dans la LCA.

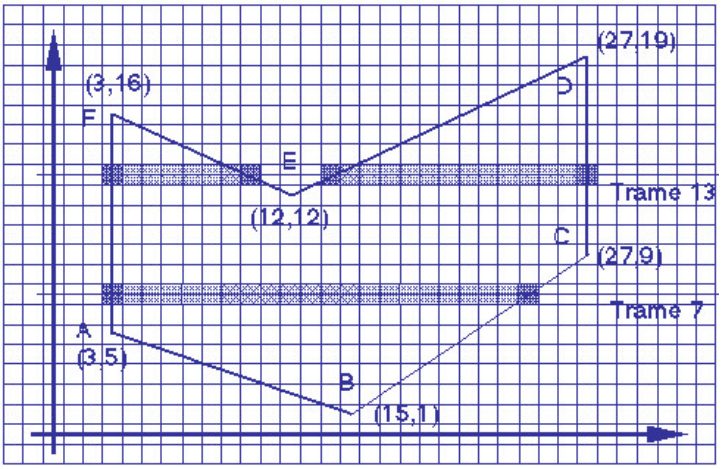


FIGURE 7 – Polygone et trames 7 et 13

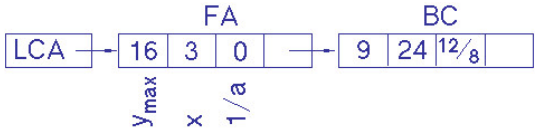


FIGURE 8 – LCA pour la trame 7

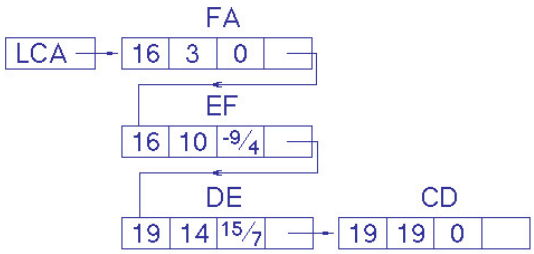


FIGURE 9 – LCA pour la trame 13

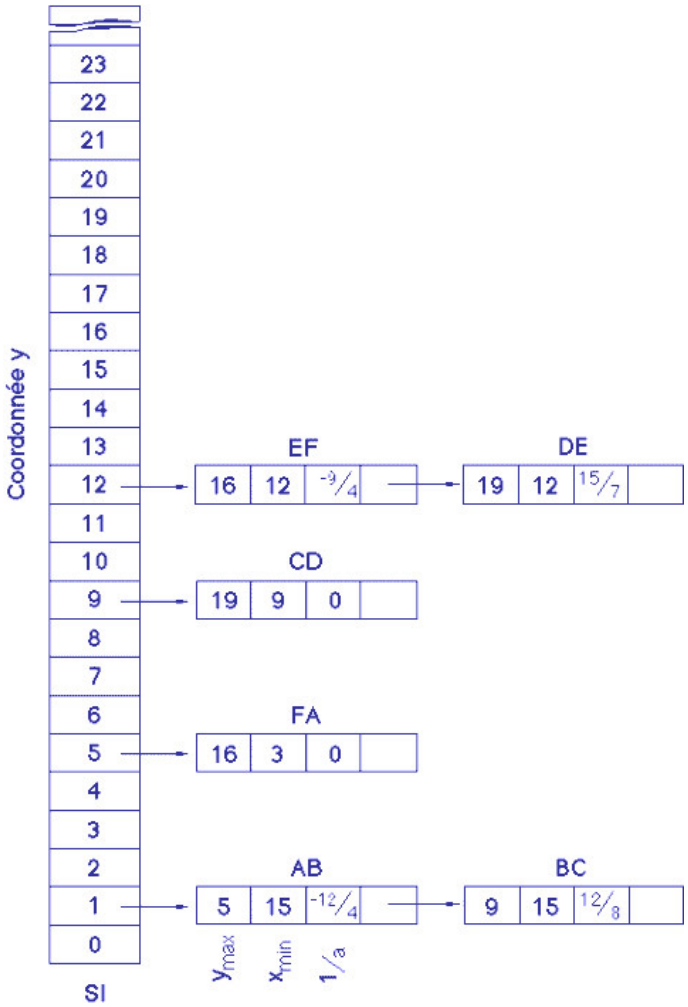


FIGURE 10 – SI

3.2.4 Algorithme

RemplissageLCA(Poly,CR)
Début
/* Création de la structure SI */
SI ⟵ CréatSI(Poly)
/* Initialisation de la structure LCA à vide */
LCA ⟵ InitLCA()
Pour chaque ligne de balayage Y interceptant le polygone **faire**
• Gérer les entrées dans LCA à partir de SI (c'est à dire mettre dans la LCA les entrées de la table SI dont le $y_{\min} = Y$
• Gérer les sorties dans LCA à partir de SI (c'est à dire enlever de la LCA les entrées da la table SI dont le $y_{\max} = Y$
• Afficher tous les morceaux de la ligne de balayage décrits dans la LCA. C'est à dire remplir tous les pixels entre deux intersections qui se trouvent à l'intérieur du polygone en utilisant la règle de parité pour déterminer si un point est à l'intérieur d'une région. La parité étant initialement paire, à chaque intersection rencontrée, on l'inverse ; ainsi les pixels sont tracés seulement quand la parité est impaire.
FinPour
Fin