Infographie - M1

Chapitre 4 - Remplissage

V.Padovani
Equipe Preuves, Programmes et Systèmes, Université Paris 7

1 Remplissage de polygones

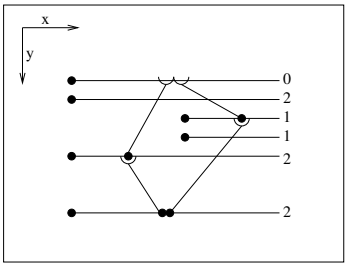
On considère le problème du remplissage d'un polygone du plan $(P_0, \ldots, P_{n-1})$, non nécessairement convexe. Noter que ce problème est distinct de l'affichage du contour du polygone : les algorithmes présentés allument correctement les pixels intérieurs au polygone et une partie des pixels de son bord, mais n'allument pas nécessairement intégralement son bord, qui doit être dessiné indépendamment.

1.1 Méthode naïve

La méthode la plus simple pour remplir un polygone est d'implémenter un test permettant de déterminer, pour chaque pixel (x, y) de l'écran, s'il est intérieur au polygone, et dans ce cas, à l'allumer.

1.1.1 Test d'intériorité

Pour déterminer si un point est intérieur ou extérieur au polygone, on considère la demi-droite horizontale Δ partant de (x, y) vers les abscisses positives. On compte le nombre d'intersections de Δ avec chaque arête non horizontale privée de son sommet d'ordonnée minimale (pour éviter de compter *deux* intersections lorsque cette demi-droite passe par un sommet du polygone).



On peut vérifier qu'au bout de ce traitement :

- Si le nombre total d'intersections comptées est *impair* alors (x, y) est intérieur au polygone, ou bien sur le bord du polygone.
- Si le nombre total d'intersections comptées est *pair* alors (x, y) est extérieur au polygone, ou bien sur le bord du polygone.

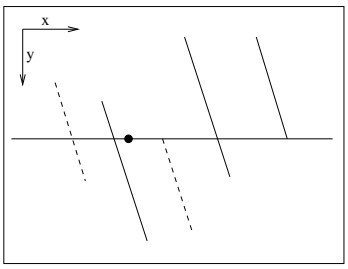
Pour les points situés sur le bord du polygone, le test n'est pas significatif, et peut donner un nombre d'intersections paire ou impaire, en fonction de leur placement. Pour être sûr de dessiner totalement le polygone rempli, il faut donc aussi dessiner son contour.

1.1.2 Description

Pour chaque arête $A_i = [P_i, P_{i+1 \bmod n}] = [(x_1, y_1), (x_2, y_2)]$:

1. Si A_i est horizontale, elle est ignorée : on passe à l'arête suivante.
2. Sinon, on vérifie que :
   (a) $\min(y_1, y_2) < y$
   (b) $y \leq \max(y_1, y_2)$
   (c) $x \leq \max(x_1, x_2)$

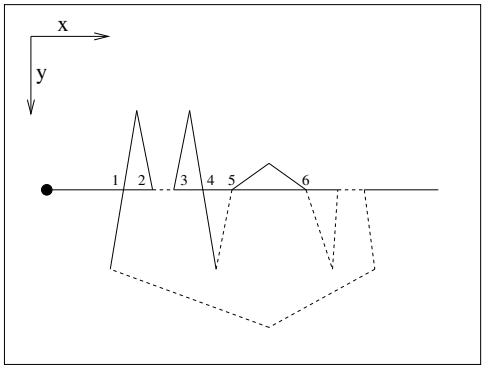
   Si l'une de ces conditions n'est pas vérifiée, A_i est ignorée : on passe à l'arête suivante. Noter que si $\min(y_1, y_2) \leq y \leq \max(y_1, y_2)$, alors A_i intersecte la droite horizontale d'ordonnée y, mais le test (a) élimine cette arête si le point d'intersection est le sommet de A_i d'abscisse minimale. Le test (c) garantit l'un au moins des deux sommets de A_i est à droite de (x, y). La figure ci-dessous indique en pointillés les arêtes rejetées par ce test.



3. Si A_i est verticale, c'est-à-dire si $x_1 = x_2$, alors A_i intersecte Δ,
   on compte cette intersection et on passe à l'arête suivante.
4. Sinon, on calcule explicitement l'abscisse x_i du point d'intersection de A_i avec la droite horizontale d'ordonnée y.
   (a) si $x_i \geq x$, alors A_i intersecte Δ,
   on compte cette intersection et on passe à l'arête suivante.
   (b) sinon, A_i n'intersecte pas Δ, et on passe à l'arête suivante.

1.1.3 Exemple

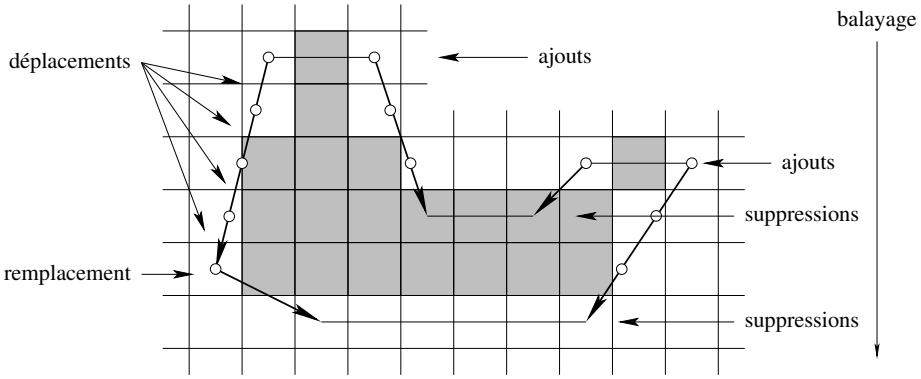
Voici un exemple du décompte des intersections de Δ avec les arêtes d'un polygone, pour un point (x, y) extérieur. Les arêtes pour lequelles aucune intersection n'est comptée sont en pointillés. Noter que le cas (2.b) ainsi que la condition (3.a) évitent de compter deux fois l'intersection de Δ avec deux arêtes si celles-ci ont un sommet commun situé sur Δ.



1.2 Algorithme de la ligne de balayage

L'algorithme de la ligne de balayage consiste à allumer, pour chaque ligne de l'écran, les portions de cette ligne intérieures au polygone. Les extrémités de ces portions sont les intersections de la ligne courante avec les arêtes du polygone, stockées dans une liste appelée *liste dynamique*.

La liste dynamique est mise à jour à chaque passage d'une ligne à la suivante, par déplacement ou suppression des extrémités courantes, et par ajout de nouvelles extrémités. Comme précédemment, cette méthode ne garantit pas l'affichage correct du bord.



La liste des extrémités à ajouter à une ligne donnée est précalculée. Elle contient, pour chaque extrémité, son abscisse initiale couplée à sa durée de vie (le nombre de lignes au bout duquel elle doit être supprimée) ainsi qu'à la valeur à ajouter à son abscisse lors du passage d'une ligne à l'autre. Les listes associées à chaque ligne sont stockées dans un tableau de listes appelé *tableau statique*.

1.3 Construction du tableau statique

On suppose le nombre de lignes de l'écran égal à H, et le polygone totalement inscrit dans l'écran. Les ordonnées des sommets du polygone sont supposées

entières. Le tableau statique est un tableau de listes $T[0, \ldots, H[$ initialement toutes vides, et remplies comme suit :

- Pour chaque arête A_i du polygone, soit (x_1, y_1) son sommet d'ordonnée minimale, soit (x_2, y_2) son sommet d'ordonnée maximale.
  - si $y_1 < y_2$, on ajoute à $T[y_1]$ le triplet (x_1, δ_x, h) où :
    $\delta_x = (x_2 - x_1)/(y_2 - y_1) \in \mathbb{R}$ (incrément)
    $h = y_2 - y_1$ (durée de vie de l'entrée)
  - sinon, c'est-à dire si $y_1 = y_2$, l'arête est ignorée.

1.3.1 Balayage effectif

On part d'une liste dynamique $\mathcal{L}$ vide, puis on effectue pour chaque numéro de ligne $y \in [0, \ldots, H[$, dans l'ordre, les opérations suivantes. A l'étape 4, on admettra le fait (vérifiable) que $\mathcal{L}$ contient toujours un nombre *pair* d'éléments.

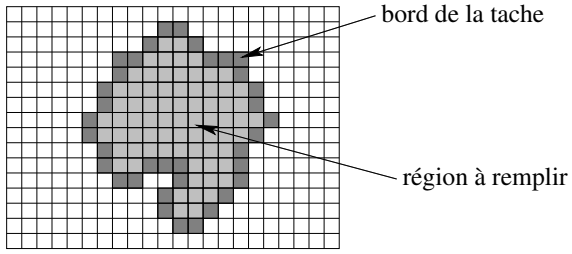
1. *Déplacement latéral de chaque entrée et mise à jour du champ hauteur*
   Pour chaque entrée $(x, \delta_x, h) \in \mathcal{L}$,
   on remplace cette entrée dans $\mathcal{L}$ par $(x + \delta_x, \delta_x, h - 1)$.
2. *Suppressions des entrées de hauteur nulle*
   Pour chaque entrée $(x, \delta_x, h) \in \mathcal{L}$,
   si $h = 0$, on supprime cette entrée de $\mathcal{L}$.
3. *Ajouts des entrées du tableau statique appartenant à la ligne courante*
   Pour chaque entrée $e = (x, \delta_x, h) \in T[y]$,
   on ajoute e à $\mathcal{L}$ de manière à ce que $\mathcal{L}$ reste trié par ordre lexicographique.
4. *Affichage de la partie intérieure du polygone sur la ligne courante*
   Soit $(e_0, e'_0, \ldots, e_{k-1}, e'_{k-1})$ la suite des éléments de $\mathcal{L}$.
   Soit $(x_0, x'_0, \ldots x_{k-1}, x'_{k-1})$ la suite des leurs premières composantes.
   Pour chaque $j \in [0, \ldots, k[$,
   on allume tous les pixels entre (x_j, y) et (x'_j, y).

1.3.2 Remarques

Les étapes (1) et (2) peuvent être faites en un seul parcours de $\mathcal{L}$. A l'étape (4), allumer les pixels entre (x_j, y) et (x'_j, y) inclus ou entre (x_j, y) et (x'_j, y) exclus ne change rien : dans les deux cas, le bord du polygone ne sera pas en général complètement affiché (cas d'une arête longue presque horizontale, ou bien horizontale d'abscisse minimale, ou encore, sommet formant un pic inférieur).

2 Remplissage de taches

On considère à présent le problème du remplissage d'une région de l'écran - la "tache" - délimitée par un bord formé d'une suite de pixels adjacents - le "bord" de cette tache. Les pixels successifs du bord de la tache peuvent être en contact direct (contact par arête verticale ou horizontale), ou indirect (contact par sommet). Le bord de la tache est supposé déjà tracé.



2.1 Méthode récursive

2.1.1 Implémentation naïve

La méthode suivante part d'un appel initial de la procédure *Remplir* sur un point (x, y) quelconque. Si ce point est strictement intérieur à la zone à remplir, le traitement allumera tous les pixels de cette zone. Si ce point est strictement extérieur à la zone, le traitement allumera tous les pixels extérieurs à cette zone.

procédure Remplir(x, y)

1. Si les conditions suivantes sont vérifiées :
   - (x, y) est dans les limites de l'écran
   - (x, y) n'est pas déjà allumé

   alors allumer (x, y) puis réinvoquer successivement :
   - *Remplir*$(x - 1, y)$,
   - *Remplir*$(x + 1, y)$,
   - *Remplir*$(x, y - 1)$,
   - *Remplir*$(x, y + 1)$,

En pratique, cette méthode est inapplicable : la profondeur d'appel peut croître jusqu'à devenir de l'ordre du nombre de pixels de l'écran, et saturer la pile d'appel de l'environnement d'exécution.

2.1.2 Simulation de la pile d'appel

La méthode suivante simule l'exécution de la précédente sans récurrence, en utilisant une pile auxiliaire. Elle convient lorsque l'environnement d'exécution a une pile d'appels de taille limitée, mais est à même d'allouer dynamiquement suffisamment de mémoire pour cette pile auxiliaire.

Contenu de la pile auxiliaire

Soient N, S, E, O quatre constantes quelconques. La pile auxiliaire contient des entrées de la forme (x, y, d), où (x, y) est la coordonnée d'un pixel déjà allumé, et où d est l'une des constantes N, S, E, O.

- une entrée (x, y, d) est dite *terminée* si d vaut O.
- le *mise à jour* d'une entrée (x, y, d) non terminée est définie par :
  - (x, y, S) si $d = N$,
  - (x, y, E) si $d = S$,
  - (x, y, O) si $d = E$.
- le *successeur* d'une entrée (x, y, d) non terminée est défini par :
  - $(x, y - 1, N)$ si $d = N$,
  - $(x, y + 1, N)$ si $d = S$,
  - $(x + 1, y, N)$ si $d = E$,
  - $(x - 1, y, N)$ si $d = O$,

Description de la méthode

On part d'un pixel (x, y), et d'une pile Π initialement vide.

1. Si (x, y) est dans les limites de l'écran et n'est pas déjà allumé,
   (a) on allume (x, y),
   (b) on empile dans Π l'entrée (x, y, N).
2. Tant que Π n'est pas vide :

   On dépile le sommet e de la pile.

   Soit (x', y', N) le successeur de e,

   (a) Si e n'est pas terminée, on empile la mise à jour de e.

   (b) Si (x', y') est dans les limites de l'écran
   et n'est pas déjà allumé :
   - on allume (x', y'),
   - on empile dans Π l'entrée (x', y', N).

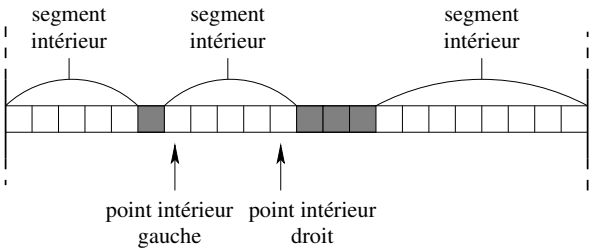
2.2 Algorithme de Smith

Un point (x, y) sera dit :

- *intérieur* si (x, y) est dans les limites de l'écran,
  et si le pixel (x, y) n'est pas déjà allumé.
- *intérieur gauche* s'il est intérieur (*i.e.* est dans l'écran et non allumé)
  et si $(x-1, y)$ n'est pas intérieur (*i.e.* est hors écran ou déjà allumé).
- *intérieur droit* s'il est intérieur (*i.e.* est dans l'écran et non allumé)
  et si $(x+1, y)$ n'est pas intérieur (*i.e.* est hors écran ou déjà allumé).

L'ensemble des points compris entre deux points (x_G, y) et (x_D, y) inclus forme un *segment intérieur* si et seulement si :

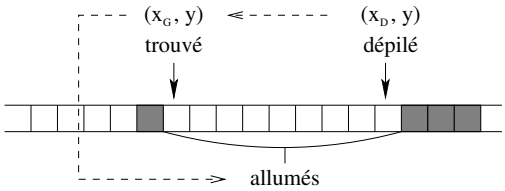
- (x_G, y) est intérieur gauche et (x_D, y) est intérieur droit,
- tous les points entre (x_G, y) et (x_D, y) sont des points intérieurs.



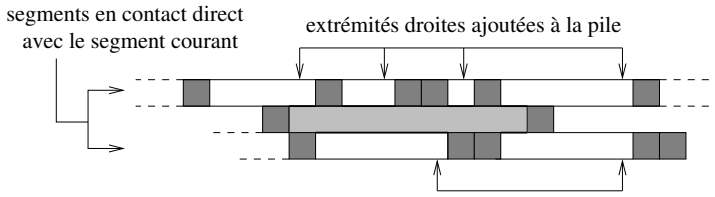
2.2.1 Principe de l'algorithme

L'algorithme de Smith utilise une pile contenant à chaque étape de sa boucle un ensemble d'extrémités droites de segments intérieurs. Lorsque (x_D, y) est dépilé, on effectue les opérations suivantes :

1. On cherche sur l'écran l'extrémité gauche (x_G, y) du segment intérieur dont (x_D, y) est l'extrémité droite. On allume tous les pixels de ce segment.



2. On ajoute à la pile toutes les extrémités droites des segments intérieurs au dessus ou en dessous du segment courant, et dont un point intérieur au moins est en contact direct avec le segment courant.



2.2.2 Description

On part d'un pixel (x_0, y_0), et d'une pile Π initialement vide. On note $y_{\max}$ l'ordonnée maximale d'un pixel de l'écran.

1. *Lancement*

   Si (x_0, y_0) n'est pas un point intérieur, l'algorithme termine.

   Sinon, soit x_D le plus petit $x \geq x_0$ tel que (x, y_0) soit intérieur droit.

   On empile (x_D, y_0) dans Π.

2. *Boucle principale*

   Tant que Π n'est pas vide :

   (a) On dépile Π. Soit (x_D, y) la valeur dépilée.

   (b) Soit x_G le plus grand $x \leq x_D$ tel que (x, y) soit intérieur gauche.

   (c) On allume tous les pixels entre (x_G, y) et (x_D, y) inclus.

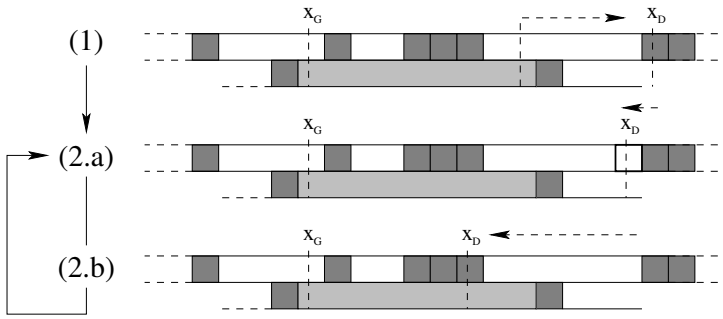
   (d) Si $y > 0$ on effectue *Ajouts* $(x_G, x_D, y-1, \Pi)$.

   (e) Si $y < y_{\max}$ on effectue *Ajouts* $(x_G, x_D, y+1, \Pi)$.

Noter que cette méthode ne fonctionne que pour le remplissage d'*intérieurs* de taches. Si l'on souhaite pouvoir remplir aussi l'extérieur d'une tache, il faut ajouter entre les étapes (a) et (b) : "si (x_D, y) n'est pas intérieur, passer à l'itération suivante". La procédure *Ajouts* est définie par :

Procédure Ajouts (x_G, x_D, y, Π)

1. tant que (x_D, y) est un point intérieur, on incrémente x_D.
2. tant que $x_G \leq x_D$:
   (a) tant que $x_G \leq x_D$ et (x_D, y) n'est pas un point intérieur, on décrémente x_D.
   (b) si $x_G \leq x_D$:
   - on empile (x_D, y) dans Π.
   - tant que $x_G \leq x_D$ et (x_D, y) est un point intérieur, on décrémente x_D.


(1)
(2.a)
(2.b)
x_G
x_D
x_G
x_D
x_G
x_D