ELIMINATION DES FACES CACHÉES

- Plusieurs algorithmes
  - tampon de profondeur (z-buffer, Catmull 1975)
  - Warnock 1969
  - méthode de la liste de priorité (l'algorithme du peintre, Newell-Newell-Sancha 1972)
- Tous possèdent plusieurs versions plus ou moins sophistiquées
- Tous travaillent dans l'espace image: chaque point est connu par sa projection sur l'écran et son éloignement peut être calculé

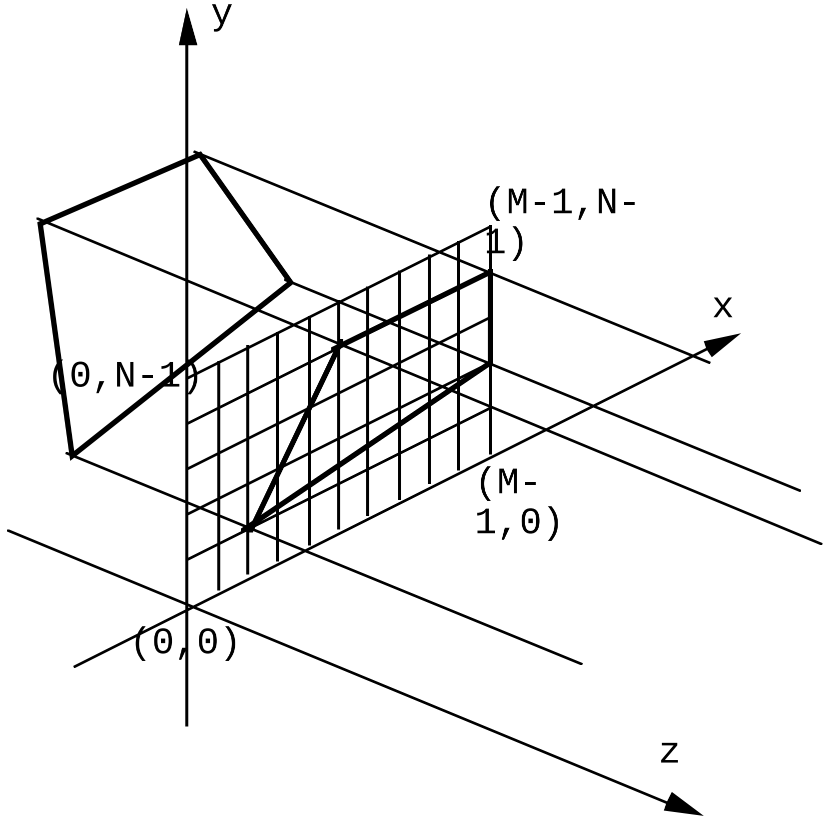
HYPOTHÈSES

(A comparer à l'élimination de lignes cachées)

- **l'écran a dimension $M \times N$ où**
  - M: **nombre de colonnes de l'écran**
  - N: **nombre de lignes de l'écran**
- **la fenêtre (espace objet) coïncide avec l'écran (on peut s'y ramener par changement d'échelle)**
- **les objets sont en général modélisés par des polyèdres non nécessairement convexes**
- **pour chaque face on connaît**
  - **la liste de ses sommets donnés par leurs coordonnées** x, y, z
  - **une équation du plan qui la contient**

$$ax + by + cz + d = 0$$

  - **sa couleur ou son niveau de gris**
- **une projection orthogonale avec observateur à l'infini dans la direction des** z **(la projection du point** (x, y, z) **est le point** (x, y)**)**


y
(M-1,N-1)
x
(0,N-1)
(M-1,0)
(0,0)
z

TAMPON DE PROFONDEUR (z-BUFFER)

Caractéristiques

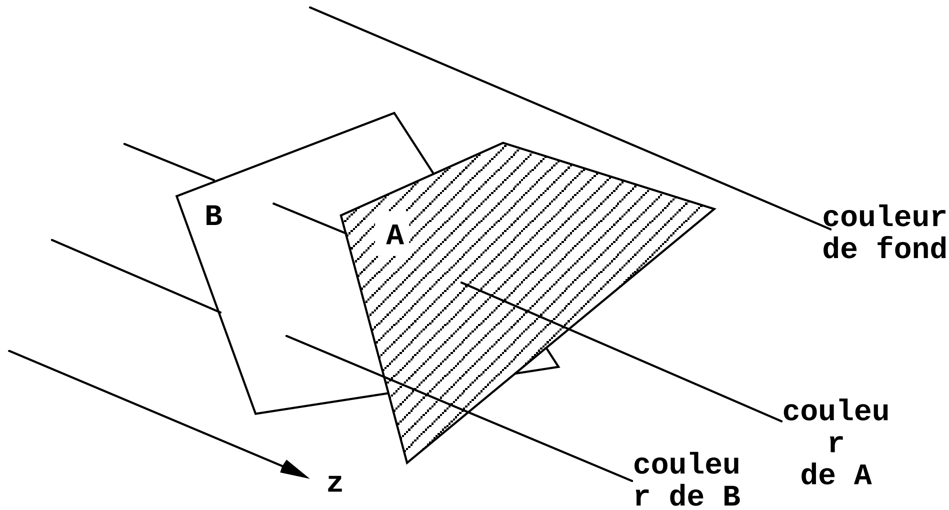
- **la méthode est brutale mais facile à implémenter**
  - **l'image est calculée pixel par pixel sans tri**
  - **le problème de l'intersection d'objets quelconques est résolu trivialement**
- **sa complexité dépend seulement linéairement de la taille de l'image**
- **son inconvénient: couteuse en espace mémoire**

$$M \times N(B_c + B_p)$$

où

  - B_c: **nombre de bits par couleur**
  - B_z: **nombre de bits par composante** z

PRINCIPE



- **pour tout** $(x, y), 0 \leq x \leq M - 1,\ 0 \leq y \leq N - 1$ **calculer, pour chaque face de la scène, la valeur en** z **du point, si il existe, ayant** x, y **comme deux premières coordonnées.**
- **afficher en position** (x, y)**, la couleur du point de la face la plus proche, si une telle face existe, ou sinon, la couleur de fond**

STRUCTURES DE DONNÉES

- **un tableau profondeur**$[0..M-1, 0..N-1]$ **enregistre la valeur courante de la composante en** z **de la face la plus proche de l'observateur (donc la valeur maximale en** z**).**
- **un tableau couleur**$[0..M-1, 0..N-1]$ **enregistre la couleur de la face la plus proche.**

LA PROCÉDURE

{*Initialisation*}
pour tout x, y **faire début**
    profondeur$[x, y] := -\infty$**;**
    couleur$[x, y]$ **:=couleur_de_fond**
    fin;
{*Boucle*}
pour tout polygone P projection d'une face F faire
    pour chaque pixel (x, y) **intérieur à P faire début**
        calculer la profondeur z **de F au point** (x, y)**;**
        si $z >$**profondeur**$[x, y]$ **alors début**
            profondeur$[x, y] := z$**;**
            couleur$[x, y]$**:=couleur(P);**
        fin
    fin

VERSION PLUS PERFORMANTE

(Méthode de la ligne de balayage, voir Géométrie Algorithmique)

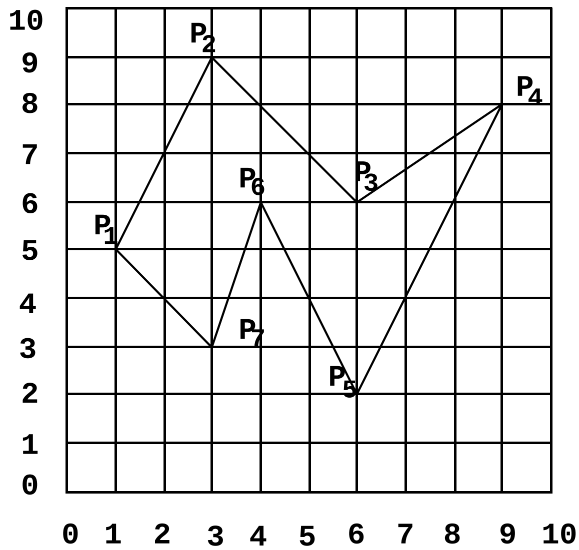
But: étant donnés la projection P d'une face et des valeurs $0 \leq x \leq M-1, 0 \leq y \leq N-1$, **accélérer les opérations**

- **tester si la droite parallèle à l'axe** z **de coordonnées** x, y **intersecte l'intérieur de P**
- **si oui, déterminer la coordonnée en** z **de ce point**

LES STRUCTURES DE DONNÉES

- **une structure *statique* $B[0..N-1]$: elle contient, pour chaque $0 \leq y \leq N-1$, la liste des arêtes dont l'extrémité supérieure a une ordonnée égale à y. On ignore les arêtes horizontales**

- **une structure *dynamique* ListeActive: imaginer une ligne horizontale balayant l'écran de haut en bas et s'arrêtant dans les N positions possibles. A chaque position de cette ligne, la structure décrit la liste des arêtes qui sont intersectées par la ligne**



10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10
P_1
P_2
P_3
P_4
P_5
P_6
P_7

LA STRUCTURE STATIQUE

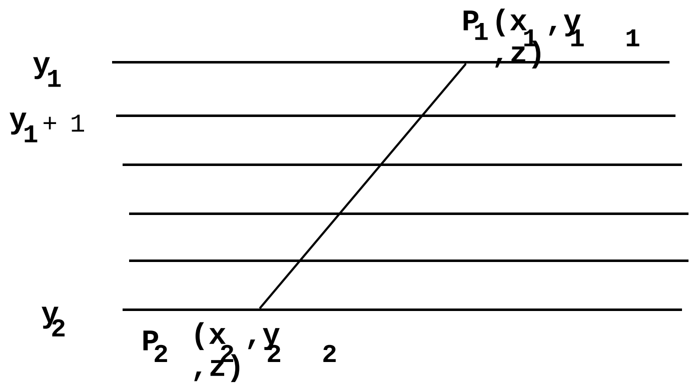
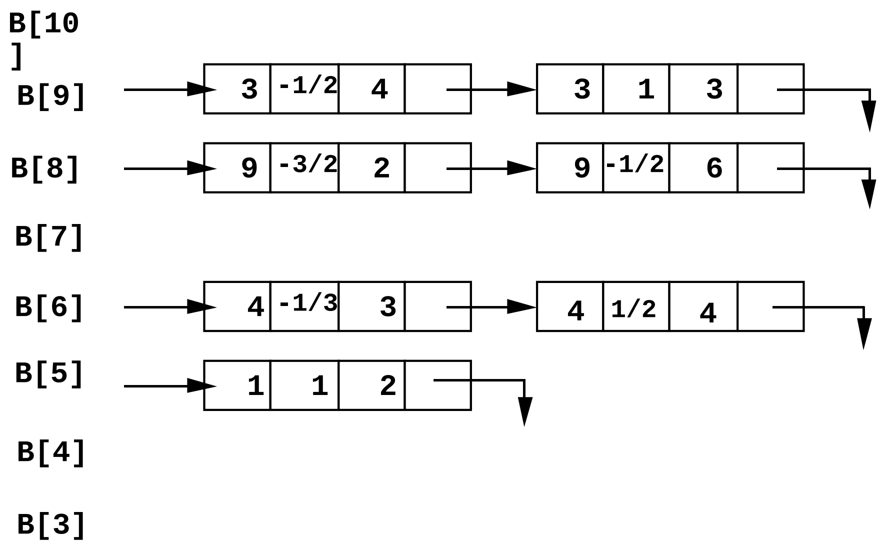


Tableau $B[0..N-1]$: mémorise, pour chaque valeur $0 \leq y \leq N-1$, les arêtes pour lesquelles l'extrémité la plus élevée a pour ordonnée y.

Ces arêtes sont rangés sous la forme de triplets $(x, \Delta x, \Delta y)$ triés dans l'ordre *alphabétique* (d'abord la première composante, puis la seconde)

- $x = x_1$: **abscisse du sommet le plus élevé (P_1 sur la figure)**
- $\Delta x = (x_2 - x_1)/(y_1 - y_2)$: **incrément en x quand on passe d'une position à une autre de la ligne de balayage (on suppose $y_2 < y_1$)**
- $\Delta y = y_1 - y_2$: **hauteur de l'arête**

EXEMPLE



LA STRUCTURE DYNAMIQUE

ListeActive: les intersections de la ligne de balayage en position y avec les arêtes du polyèdre sont rangées dans une structure de type list, sous la forme de quadruplets

$$(x_c, \Delta x, \Delta y_c, z_c)$$

- x_c **est l'abscisse de l'intersection courante de la ligne de balayage en position** y **avec l'arête en question**
- $\Delta x = (x_2 - x_1)/(y_1 - y_2)$**: voir la structure statique**
- $\Delta y_c = y - y_2$ **est le nombre de lignes horizontales restantes qui intersectent l'arête**
- z_c**: si le rang du quadruplet dans la liste est impair, c'est la profondeur de l'intersection courante de la ligne de balayage en position** y **avec l'arête en question sinon c'est** $\perp$

NOUVELLE PROCÉDURE

{*Boucle*}
pour tout polygone P projection d'une face F faire
    créer la structure B **associée à P**
    traiter le polygone P

Traiter le polygone P

$y = N - 1$
créer ListeActive
tant que $y >= 0$ **faire début**
    traiter les segments intersectés
    modifier ListeActive
    $y := y - 1$
fin

MODIFICATION DE LA LISTE

Passage de la position $y-1$ à la position y (l'équation du plan de la face est $ax+by+cz+d=0$)

pour chaque arête $(x_c, \Delta x, \Delta y_c, z_c)$ **de ListeActive faire**
  $x_c := x_c + \Delta x;$
  $\Delta y_c := \Delta y_c - 1;$
  $z_c := z_c - \frac{a\Delta x + b}{c};$
supprimer toutes les arêtes pour lesquelles $\Delta y_c < 0$
supprimer toutes les arêtes pour lesquelles $\Delta y_c = 0$
  et pour lesquelles il existe dans $B[y]$ **un élément**
  dont la composante en x **est égale à** x_c
insérer les éléments de $B[y]$ **dans ListeActive,**
  en respectant l'ordre alphabétique,
  et en posant $z_c = -\frac{ax+by+d}{c}$ **s'il a rang impair**
  dans la liste et $z_c = \perp$ **sinon**

ILLUSTRATION

10				
9	$(3, -0.5, 4, -12)$	$(3, 1, 3, \perp)$		
8	$(2.5, -0.5, 3, -10.5)$	$(4, 1, 2, \perp)$	$(9, -1.5, 2, -17)$	$(9, -0.5, 6, \perp)$
7	$(2, -0.5, 2, -9)$	$(5, 1, 1, \perp)$	$(7.5, -1.5, 1, -14.5)$	$(8.5, -0.5, 5, \perp)$
6	$(1.5, -0.5, 1, -7.5)$	$(4, -0.33, 3, \perp)$	$(4, 0.5, 4, -10)$	$(6, 1, 0, \perp)$
	$(6, -1.5, 0, -12)$	$(8, -0.5, 4, \perp)$		
5	$(1, 1, 2, -6)$	$(3.67, -0.33, 2, \perp)$	$(4.5, 0.5, 3, -9.5)$	$(7.5, -0.5, 3, \perp)$
4	$(2, 1, 1, -6)$	$(3.33, -0.33, 1, \perp)$	$(5, 0.5, 2, -9)$	$(7, -0.5, 2, \perp)$
3	$(3, 1, 0, -6)$	$(3, -0.33, 0, \perp)$	$(5.5, 0.5, 1, -8.5)$	$(6.5, -0.5, 1, \perp)$
2	$(6, 0.5, 0, -8)$	$(6, -0.5, 0, \perp)$		
1				
0				

TRAITER LES SEGMENTS INTERSECTÉS

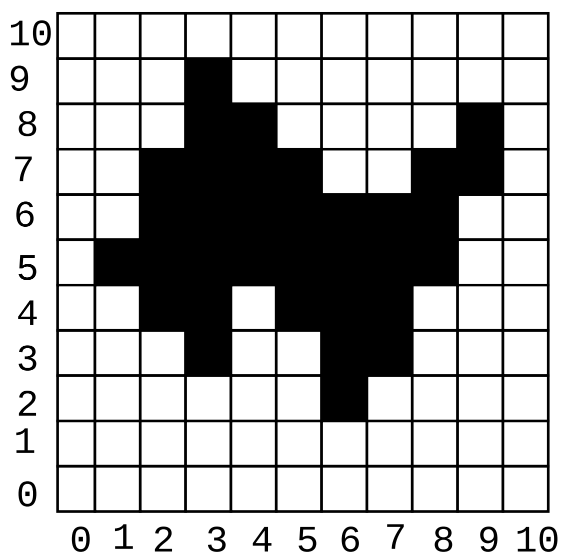
tant que ListeActive non vide faire début

retirer les deux premiers éléments

$(x_1, \Delta x_1, \Delta y_1, z_1), (x_2, \Delta x_2, \Delta y_2, \perp)$

mettre à jour les tableaux profondeur et couleur

pour tous les pixels entre (x_1, y) **et** (x_2, y)



ALGORITHME DE WARNOCK

Etant donné un ensemble de faces dans l'espace, on applique récursivement la procédure suivante à l'image (initialement l'image est l'écran complet).

1. **si aucune face ne se trouve dans l'image, alors l'image prend la couleur du fond (ou son intensité)**
2. **si toute la surface de l'image est occupée par une face qui recouvre toutes les autres (c'est alors la face la plus proche) alors l'image prend la couleur de la face (ou son intensité)**
3. **dans tous les autres cas on subdivise l'image en 4 sous-images de taille égale (voir l'arbre quartique), pour lesquelles on appelle la procédure.**

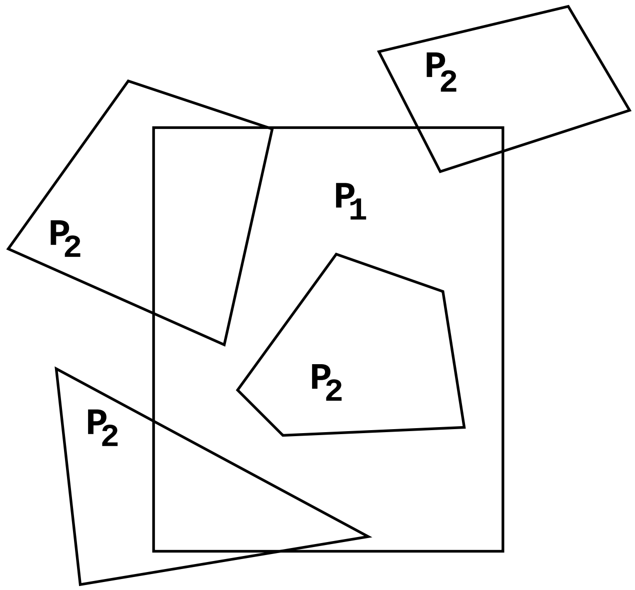
INTERSECTION DE POLYGONES

X une région quelconque du plan

- **l'*intérieur* de X est l'ensemble $\overset{\circ}{X}$ de ses points autour desquels on peut tracer un disque de rayon non nul entièrement compris dans X**

P_1 et P_2 deux polygones

- **P_1 et P_2 sont *disjoints* si les intérieurs de P_1 et P_2 sont eux-mêmes disjoints.**
- **P_1 est *contenu* dans P_2 ou P_2 *englobe* (*enveloppe*) P_1 si la région définie par P_1 est contenue dans celle définie par P_2.**
- **P_1 et P_2 s'intersectent dans les autres cas.**


P2
P1
P2
P2
P2

HYPOTHÈSES

- **les polygones sont orientés dans le sens des aiguilles d'une montre.**
- **l'un des polygones est un rectangle régulier F (la *fenêtre*) et l'autre est un polygone P (une *face* du polyèdre)**

On veut déterminer si

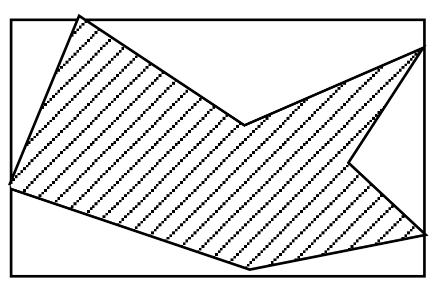
- **la fenêtre est contenue dans la face**
- **la face est contenue dans la fenêtre**
- **la fenêtre et la face sont disjointes**
- **la fenêtre et la face se coupent**

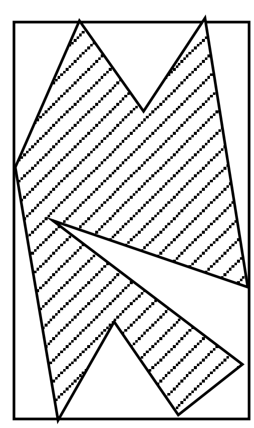
RECTANGLE ENVELOPPANT

Le *rectangle enveloppant* (*bounding box*) d'un polygone P est le rectangle $(xmin, ymin, xmax, ymax)$ **où:**

- $xmin = \min\{x_i : (x_i, y_i)$ **est un sommet de P**$\}$
- $ymin = \min\{y_i : (x_i, y_i)$ **est un sommet de P**$\}$
- $xmax = \max\{x_i : (x_i, y_i)$ **est un sommet de P**$\}$
- $ymax = \max\{y_i : (x_i, y_i)$ **est un sommet de P**$\}$

Il permet de simplifier le test d'intersection de polygone



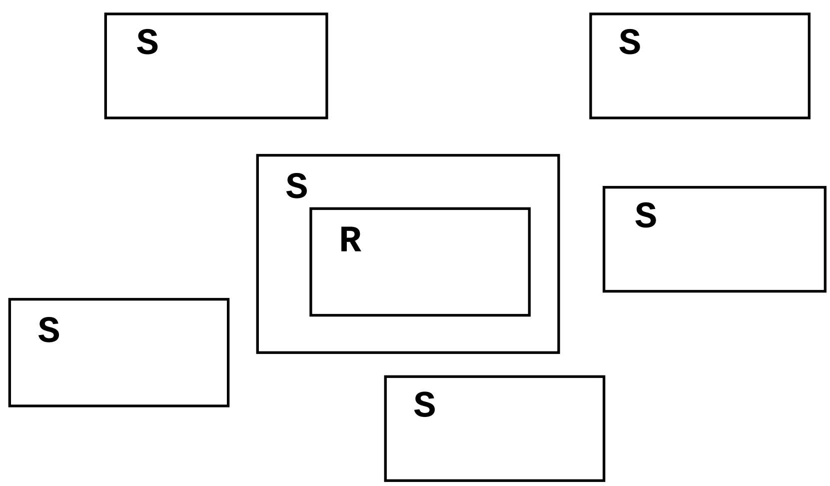


TEST SUR RECTANGLES RÉGULIERS

R=(x_{gauche}, y_{bas}, x_{droit} ,y_{haut}) **et** S=(z_{gauche}, t_{bas}, z_{droit}, t_{haut})

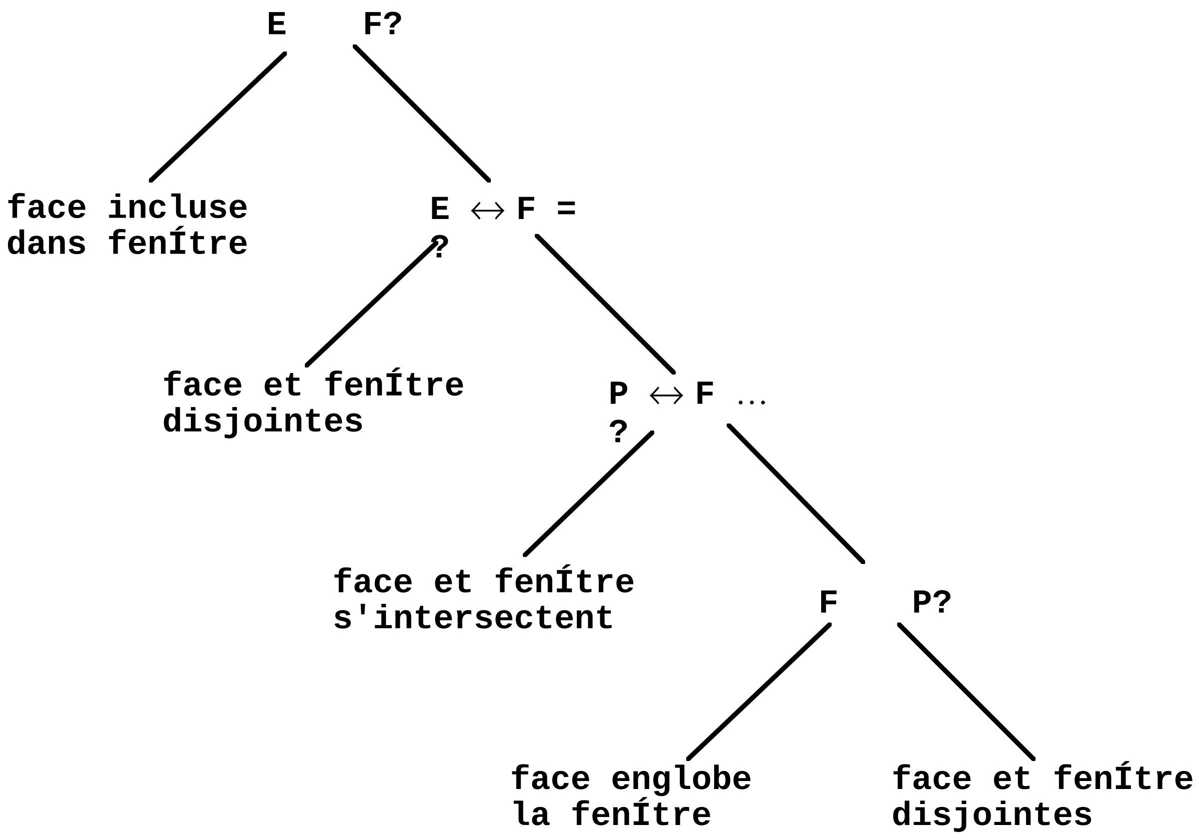
- **R et S sont disjoints si et seulement si:** $x_{droit} \leq z_{gauche}$ **ou** $y_{bas} \leq$ $\mathbf{t}_{haut}$ **ou**

- **R est contenu dans S si et seulement si:** $z_{gauche} \leq x_{gauche}$ **e** $t_{bas} \leq y_{bas}$ **e**



BATTERIE DE TESTS

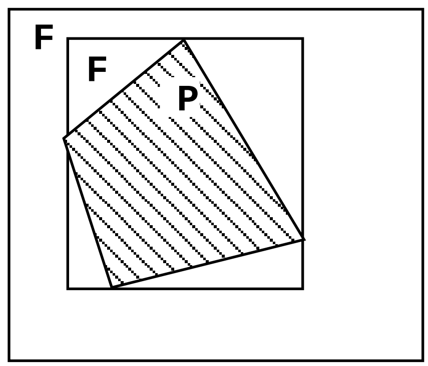
La méthode consiste à faire subir au couple polygone-fenêtre, 4 tests allant du plus simple au plus complexe



TEST1

Le polygone P est-il contenu dans la fenêtre F?

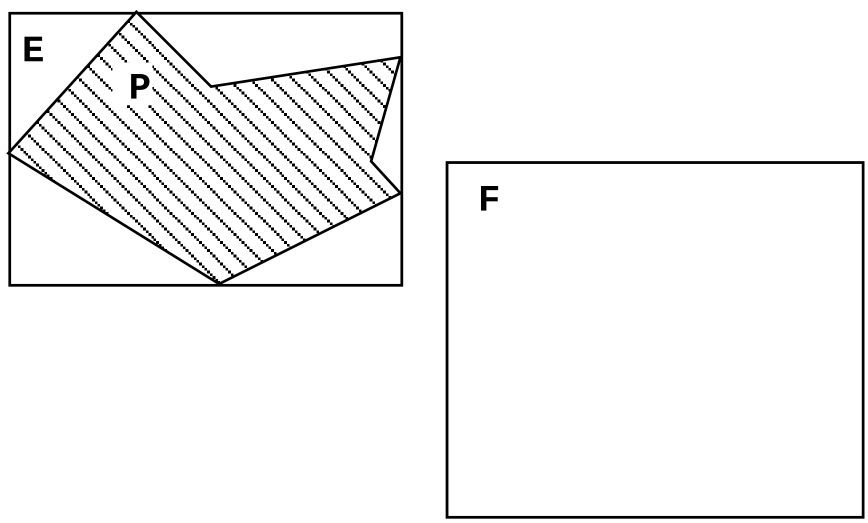
Ceci équivaut à demander si le rectangle enveloppant E est contenu dans la fenêtre?



TEST2

Le rectangle enveloppant E du polygone P est-il disjoint de la fenêtre?

Si oui, alors le polygone est a fortiori disjoint de F



TEST3

Le polygone P intersecte-t'il la fenêtre?

Ceci a lieu si et seulement si l'une des 4 conditions suivantes est satisfaite

- **Il existe une arête $\mathbf{P}_1\mathbf{P}_2$ du polygone et une arête AB de la fenêtre qui s'intersectent en leur intérieur**

  $R(x, y) = 0$ **l'équation de la droite support de AB**

  $Q(x, y) = 0$ **l'équation de la droite support de $\mathbf{P}_1\mathbf{P}_2$**

  équivaut à $R(\mathbf{P}_1)R(\mathbf{P}_2) < 0$ **et** $Q(\mathbf{A})Q(\mathbf{B}) < 0$

- **Sinon il existe une arête $\mathbf{P}_1\mathbf{P}_2$ du polygone et 3 sommets consécutifs (ordre des aiguilles d'une montre) ABC tels que B soit à l'intérieur de $\mathbf{P}_1\mathbf{P}_2$**

  $S(x, y) = 0$ **est l'équation de la droite support de BC**

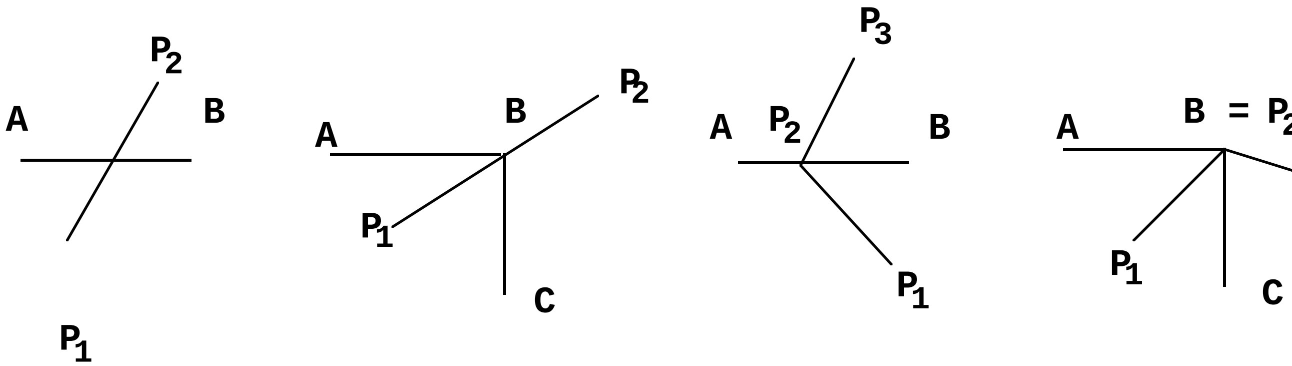
  équivaut à $R(\mathbf{P}_1)R(\mathbf{P}_2) < 0$, $Q(\mathbf{B}) = 0$ **et** $Q(\mathbf{A})Q(\mathbf{C}) < 0$

- **Sinon il existe 3 sommets consécutifs $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ du polygone et une arête AB de la fenêtre tels que $\mathbf{P}_2$ soit à l'intérieur de AB**

  équivaut à $R(\mathbf{P}_2) = 0$, $R(\mathbf{P}_1)R(\mathbf{P}_3) < 0$ **et** $Q(\mathbf{A})Q(\mathbf{B}) < 0$

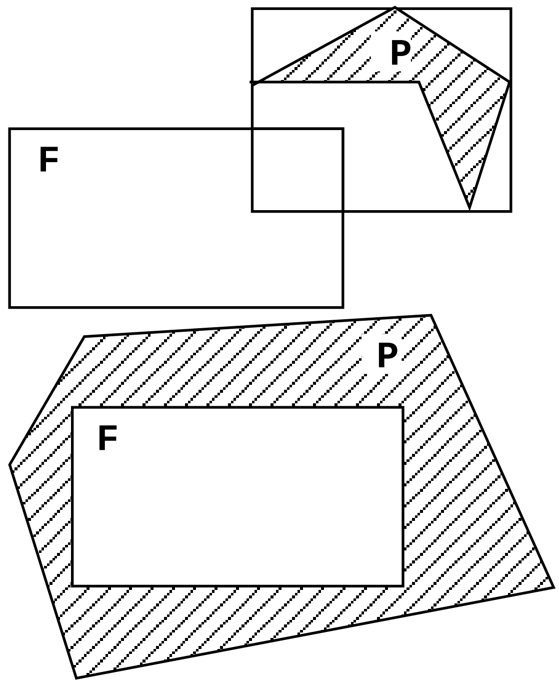
- **Sinon il existe 3 sommets consécutifs $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ du polygone et 3 sommets consécutifs ABC de la fenêtre tels que B $= \mathbf{P}_2$, et $\mathbf{P}_1$ et $\mathbf{P}_3$ sont ou bien de part et d' autre de AB ou de part et d'autre de BC**

  équivaut à $R(\mathbf{P}_2) = 0$, $Q(B) = 0$, et $R(\mathbf{P}_1)R(\mathbf{P}_3) < 0$ ou bien $S(\mathbf{P}_1)S(\mathbf{P}_3) < 0$



Situation après passage dans les 3 tests

- **ou bien le polygone et la fenêtre sont disjoints**
- **ou bien le polygone englobe la fenêtre**



TEST 4

Le polygone P enveloppe-t-il la fenêtre?

Ceci équivaut à vérifier si le sommet supérieur gauche (par exemple) de la fenêtre, est intérieur au polygone

(Il existe en fait encore quelques cas pathologiques, les trouver . . .)

COMMENTAIRES SUR L'ALGORITHME DE WARNOCK

- **condition d'arrêt: si la fenêtre a la dimension d'un pixel dont les coordonnées du centre sont x, y, on attribue au pixel la couleur (l'intensité) de la face dont le point de coordonnées x, y est le plus proche de l'œil.**
- **la condition (2) est couteûse. On peut procéder comme suit. A chaque face recouvrant l'image, on associe la plus grande profondeur (c'est-à-dire le z minimum) des 4 points de cette face dont les coordonnées (x, y) sont celles des 4 sommets de l'image. Soit $zmin$ cette valeur. Alors la face recouvre bien les autres faces si pour chacune de celles-ci le z maximum est inférieur à $zmin$. Dans les autres cas on ne peut pas conclure et l'on poursuit le raffinement.**
- **la méthode du tampon de profondeur est une variante de l'algorithme de Warnock**

VARIANTES

Autres versions de l'algorithme

- **autres tests d'arrêt possibles: (par exemple) l'image est intersectée par un seul polygone ou ne contient qu'un seul polygone**
- **autres subdivisions de l'image:(par exemple) elle peut être adaptée au polygone et être définie en fonction du rectangle enveloppant**
- **autres ordres de visite des faces: on peut supposer les faces ordonnées par la coordonnée en z du sommet le plus proche de l'œil. On a aussi intérêt à maintenir 3 listes**
  - **la liste des faces qui entourent la fenêtre**
  - **la liste des faces qui lui sont disjointes**
  - **la liste des faces qui l'intersectent ou qui sont contenues par elle**

  La mise à jour au moment de la subdivision de l'image utilise des propriétés
  - **si une face entoure une image, alors elle entoure toutes ses sous-images**

– si une face est disjointe d'une image, alors elle est disjointe de toutes ses sous-images
– etc . . . ...

On retient aussi le niveau de l'arbre où les faces sont ajoutées. A chaque niveau de la subdivision, on parcourt la liste des faces entourantes pour trouver celle (si elle existe) qui est la plus proche de l'œil. On vérifie alors si elle recouvre toutes les faces de la deuxième liste. Les faces de la troisième liste sont ignorées.

LES PROCÉDURES

Hypothèses

- **les faces sont ordonnées par le** $zmax$ **de leurs sommets croissant**
- **la liste des projections des faces est représentée par une variable de type EnsemblePolygone**

EnsemblePolygone = ˆCellulePolygone;
CellulePolygone = record
p: Polygone;
r: Rect;{*rectangle englobant*}
c: couleur;
s: EnsemblePolygone
fin

3 variables globales

- **ListeEnveloppant: liste des faces qui enveloppent la fenêtre courante**
- **ListeDisjoint: liste des faces qui enveloppent la fenêtre courante**
- **ListeCoupant: liste des faces qui intersectent ou qui sont incluses dans la fenêtre courante**

Dans chaque liste les faces sont triées dans l'ordre des z_{max} décroissant

Initialisation des listes

- **ListeEnveloppant et ListeDisjoint sont initialisées à vide**
- **ListeCoupant contient toutes les faces de la scène**
- **MettreAJourListes effectue la mise à jour des listes avant l'appel récursivement la procédure Warnock**

```
procedure Warnock(r: Rect); {la fenêtre}
var
   r1 : Rect;
début
   si Pixel(r) alors AfficherPixel(r)
      sinon si Subdiviser(r) alors début
         r1:= FenêtreHautGauche(r);
         MettreAJourListes(r1);
         Warnock(r1);
         r1:= FenêtreHautDroit(r);
         MettreAJourListes(r1);
         Warnock(r1);
         r1:= FenêtreBasGauche(r);
         MettreAJourListes(r1);
         Warnock(r1);
         r1:= FenêtreBasDroit(r);
         MettreAJourListes(r1);
         Warnock(r1);
      fin;
fin;{Fin de Warnock}
```

Subdiviser détermine s'il faut décomposer la fenêtre r en quatre sous-fenêtres. Elle réalise aussi l'affichage lorsqu'elle renvoie faux.

La face F *cache fortement* la face F1 si le z_{min} de la première est supérieur au z_{max} de la seconde

```
fonction Subdiviser (r : Rect): boolean;
var b : boolean);
var
   F, F1 : Polygone;
début
   Subdiviser:=faux;
   si ListeEnveloppant = ∅ alors debut
      si ListeCoupant est réduite à une seule face F alors
      sinon si ListeCoupant est vide alors afficher couleur
         sinon Subdiviser:=vrai
      exit;
   fin;
   déterminer la face F de ListeEnveloppant la plus proch
   pour toute face F1 de ListeIntersectant faire
      si F1 non fortement cachée par F alors Subdiviser:=
   si Subdiviser alors afficher F;
fin;{Fin de Subdiviser}
```

Au passage d'une fenêtre à une sous-fenêtre on effectue la mise à jour des 3 listes. Seules les faces coupantes peuvent changer de statut.

```
procedure MiseAJourDesListes (r : Rect);
var
   r1 : Rect;
début
   pour toute face F de ListeCoupant fairedébut
   r1:= FenetreHautGauche(r);
   Ranger(r1,F);
   r1:= FenetreHautDroite(r);
   Ranger(r1,F);
   r1:= FenetreBasGauche(r);
   Ranger(r1,F);
   r1:= FenetreBasDroite(r);
   Ranger(r1,F);
fin; {Fin de MiseAJourDesListes}
```

Ranger détermine dans quelle liste associée à la fenêtre r la face F doit être rangée. On suppose implémentées les procédures RectDisjointRect, RectInclusRect, PolygoneInterRect, RectInclusPoly avec leur sens intuitif

```
procedure Ranger (r : Rect; F: Polygone);
début
   si RectDisjointRect(r, F.r) alors Inserer(F, ListeDisjoint
      sinon si RectInclusRect(F.r, r) alors
      ne rien faire, pas de changement de statut
         sinon si RectInclusPoly(r, F) alors
         Inserer(F, ListeEnveloppant);
            {sinon ne rien faire, pas de changement de st
fin; {Fin de Ranger}
```