Élimination des parties cachées

Contenu

Étant donné une scène 3D constituée de divers objets, il s'agit de déterminer quelles parties de ces objets seront visibles (et donc à afficher) dans une image, après projection selon un centre de projection ou suivant une direction. C'est un problème fondamental de la synthèse d'images (*hidden surface removal* en Anglais) dont les aspects importants sont :

- processus en ligne,
- rapidité d'exécution,
- mémoire mise en œuvre,
- indépendance vis à vis du matériel,

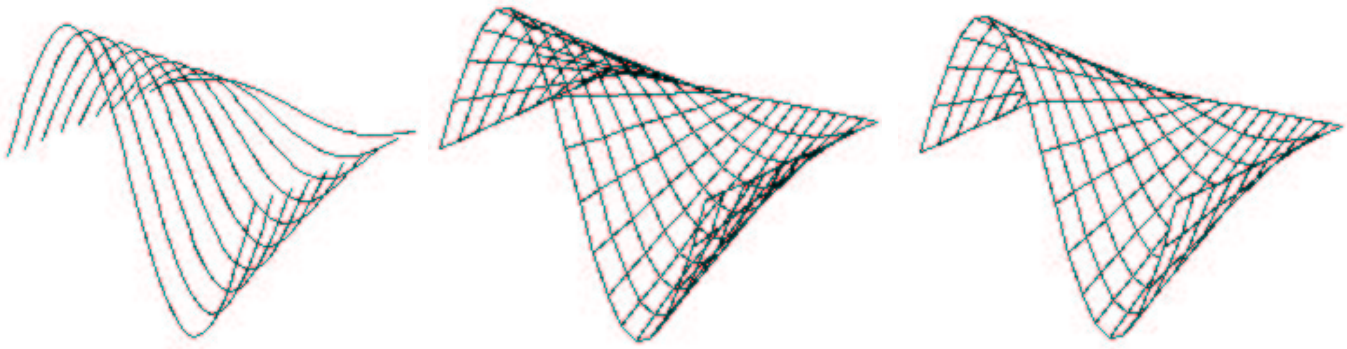
Un certains nombre de solutions ont été développées pour répondre à ces besoins. On peut classer ces approches de manière grossière dans deux catégories :

1. les approches à précision image :
   pour chaque pixel, on détermine l'objet le plus proche. Ces approches sont dépendantes du matériel (la taille de l'image).
2. les approches à précision objet :
   pour chaque objet, on détermine la partie visible. Approches indépendantes du matériel.

1 Les iso-courbes

Les algorithmes présentés ici traitent les surfaces définies par des fonctions continues de deux variables : $z = f(x, y)$. Ces surfaces représentent un cas particulier pour l'élimination des parties cachées ; cas pour lequel des solutions efficaces et rapides existent.

Ce type de surface peut être représentée sous une forme *fil de fer* par un ensemble de courbes à x constant : les *iso-x*, à y constant : *les iso-y*, ou bien les deux. Un algorithme de suppression des parties cachées consiste à éliminer les parties des courbes qui sont occultées par la surface elle-même au point d'observation.



(a) iso-x

(b) iso-x et iso-y

(c) iso-x, iso-y et élimination des parties cachées

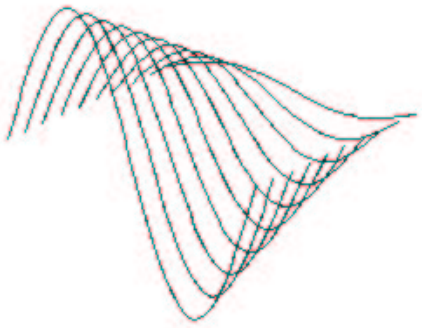
Figure 1: *Différentes représentations*

1.1 Les courbes iso-x

Nous traitons dans un premier temps le cas des courbes à x constant. Chaque courbe appartient à un plan à x constant et ne peut donc occultée par un plan à x constant plus éloigné du point d'observation. L'affichage des courbes se fera donc suivant un ordre déterminé par la distance au point d'observation, du plus proche au plus éloigné.

On définit la silhouette de la surface comme étant les extrémités dans l'image de la surface observée. Le principe ensuite d'élimination repose sur le fait que :

- Pour chaque courbe tracée, les parties visibles sont celles qui sont soit au-dessus soit en-dessous de la silhouette.

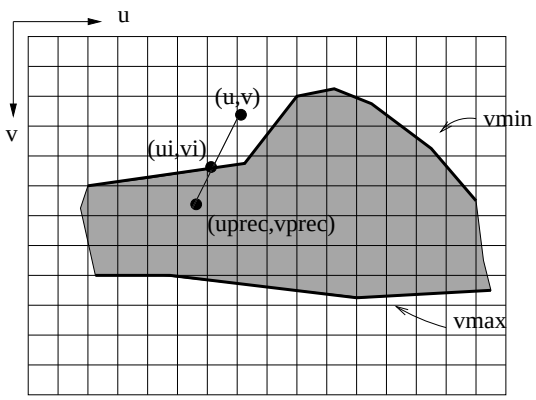


L'algorithme s'écrit donc de manière schématique :

```
pour x de xmin a xmax
 yprec = ymin
 pour y de (ymin+yprecision)  a ymax
   (uprec, vprec) = projection de (x, yprec,f(x,yprec))
   (u,v) = projection de (x, y,f(x,y))
   si (uprec, vprec, u,v) a l'exterieur de la silhouette alors
     tracer (uprec,vprec,u,v)
     mettre a jour la silhouette
   si (uprec, vprec, u,v) intersecte la silhouette
     calculer le point intersection (ui ,vi)
     si (uprec, vprec, ui,vi) a l'exterieur de la silhouette alors
        tracer (uprec,vprec,ui,vi)
        mettre a jour la silhouette
     sinon
        tracer (ui,vi,u,v)
        mettre a jour la silhouette
    yprec = y
  finpour
finpour
```

Les différents algorithmes existants varient sur la manière de représenter les segments de courbes constituant la silhouette :

1. sous la forme de pixels,
   - ☞ problème si la résolution pour le tableau de la silhouette est inférieure à celle du tracé de segment de droites.
   - ☞ dépendant de la résolution de l'écran.
2. sous la forme de points exprimés dans le repère de la scène.



- ☞ indépendant de la résolution de l'écran.
- ☞ mémoire nécessaire dépendante de la précision du tracé (xprecision, yprecision).

Un cas simple est celui où la silhouette de la surface peut être représentée sous la forme de deux fonctions de u dans l'image : vmax(u) et vmin(u). Il suffit alors de stocker deux tableaux à une dimension des valeurs vmax et vmin.

Dans le cas général, il faut des structures de données plus complexes pour stocker les différents segments constituant la silhouette.

1.2 Courbes iso-x et iso-y simultanément

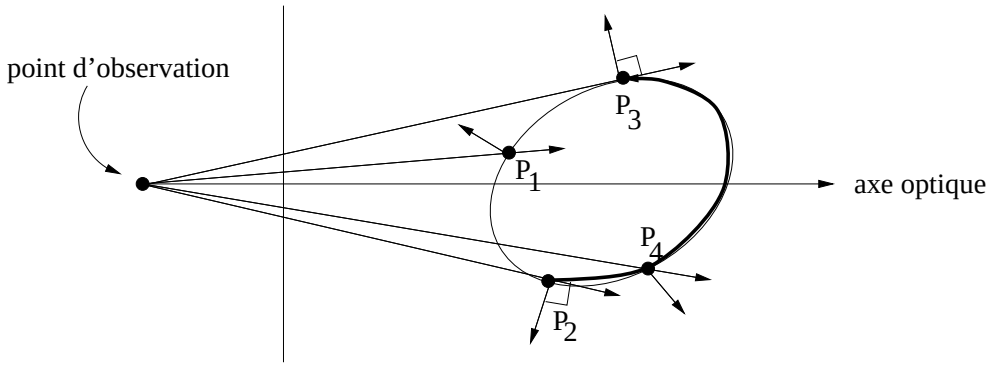
La méthode précédente s'applique indifféremment aux courbes à x constant et aux courbes à y constant. Par contre, pour obtenir une surface représentée sous la forme d'une grille de points et des courbes iso-x et iso-y, il ne suffit pas de superposer les courbes tracées avec la méthode précédente en x puis en y. Les unes peuvent en effet occulter les autres.

La solution consiste à intercaler le tracé des courbes. Après le tracé d'une courbe en iso-x (respectivement iso-y), on trace suivant l'algorithme précédent tous les segments de courbes en iso-y (respectivement iso-x) qui sont compris entre la courbe iso-x courante et la courbe iso-x suivante. La silhouette utilisée doit être la même pour les segments iso-x et les courbes iso-y.

2 Backface culling

Le backface culling (élimination des faces arrières) repose sur une idée simple. Si l'on connaît les normales à la surface de l'objet observé, il est alors possible

d'éliminer les parties de la surface pour lesquelles la normale pointe dans la direction opposée au point d'observation. Ces parties, vues du point d'observation, sont en effet occultées par l'objet lui-même.



P_1 est visible, P_2et P_3 sont à la limite de visibilité, P_4 n'est pas visible.

Pour les points de la figure ci-dessus, on vérifie aisément que :

$$N_1 \cdot V_1 < 0,\ N_2 \cdot V_2 = N_3 \cdot V_3 = 0,\ N_4 \cdot V_4 > 0,$$

où V_i est la direction de la ligne de vue au point P_i. On en déduit que les parties pour lesquelles $N \cdot V > 0$ ne sont pas visibles et peuvent être éliminées.

- ☞ Dans le cas d'une projection orthographique (le centre de projection est rejeté à l'infini), la direction de la ligne de vue est constante en tout point de l'image et correspond à la direction de projection ; à savoir $(0, 0, 1)$ dans le repère de la caméra.
- ☞ pour un seul objet convexe, l'élimination des parties cachées se résume au backface culling.

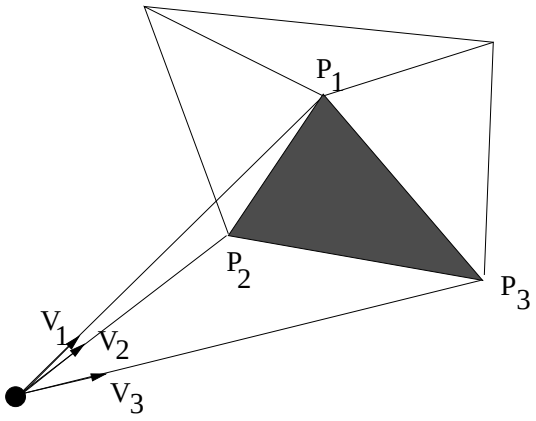
2.1 Application au cas des facettes triangulaires

Nous supposons ici que les objets observés sont constitués de facettes triangulaires. Nous supposons de plus que les facettes sont orientées de manière cohérente. Dans ce cas, la normale à une facette s'écrit :

$$N = P_1P_2 \wedge P_1P_3,$$

et le critère de visibilité pour une facette :

$$V_{1,2,3} \cdot N \leq 0.$$



Soit :

$$V_1 \cdot (P_1P_2 \wedge P_1P_3) \leq 0,$$

et :

$$V_1 \cdot ((V_2 - V_1) \wedge (V_3 - V_1)) \leq 0,$$

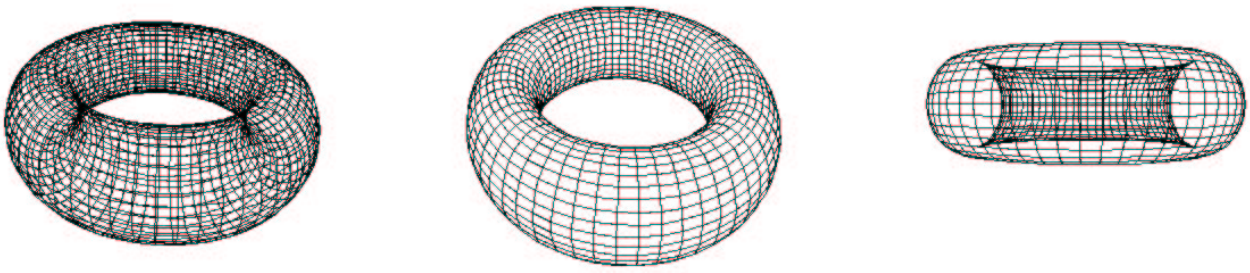
d'où:

$$V_1 \cdot (V_2 \wedge V_3) \leq 0.$$

Le critère de visibilité consiste donc ici à déterminer le signe du déterminant correspondant au produit mixte de l'expression précédente :

$$\begin{vmatrix} X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \\ Z_1 & Z_2 & Z_3 \end{vmatrix} \leq 0.$$

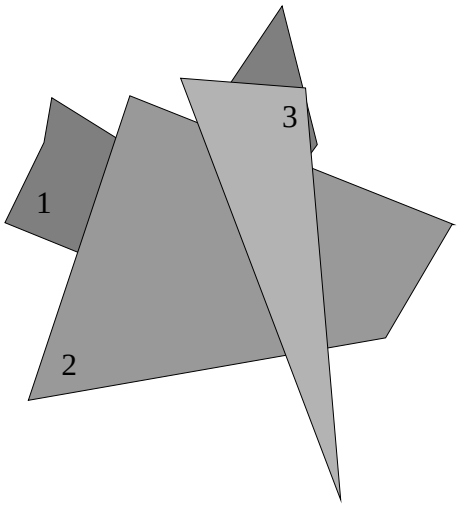
- ☞ Cette approche peut être facilement généralisée au facettes polygonales.
- ☞ Le backface culling n'est pas suffisant pour éliminer l'ensemble des parties cachées mais il permet d'éliminer un nombre important de primitives (la moitié des facettes présentes dans la scène). C'est donc une première étape d'un processus plus globale qui permet de réduire de manière significative la complexité.
- ☞ C'est un algorithme qui se classe dans les approches à précision objet.
- ☞ Algorithme qui peut facilement être implémenté matériellement.



3 Algorithme du peintre

L'idée directrice de l'algorithme du peintre [1972] est de *peindre* les facettes polygonales dans la mémoire vidéo suivant un ordre de distance décroissante au point d'observation. Le processus nécessite trois étapes principales :

1. Trier les facettes suivant les z décroissants dans le repère de la caméra.
2. Résoudre les ambiguïtés dans la liste lorsque les facettes se recouvrent.
3. Projeter les facettes et remplir les polygones suivant la liste.



Le cas trivial

☞ Le cas trivial concerne plusieurs applications : cartographie, tracé de circuits, gestion de fenêtres.

Les ambiguïtés qui peuvent se présenter sont de différents types :

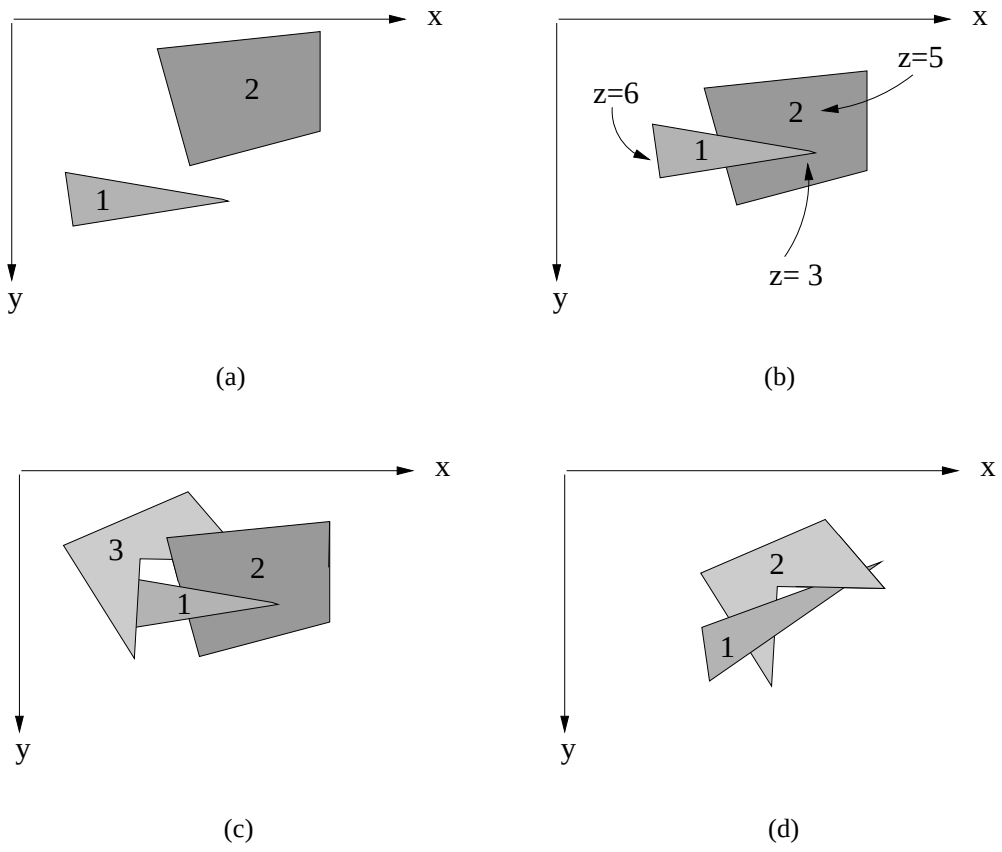


Figure 2: *algorithme du peintre : différentes situations.*

Le cas (a) de la figure peut être résolu en calculant les projections dans le plan des facettes et en déterminant leurs intersections. Le cas (b) peut être résolu en remarquant que tous les points de la facette 2 sont du du même côté du plan de la facette 1. Les cas (c) et (d) nécessitent le calcul explicite d'intersections (difficile).

4 Algorithmes de subdivision de l'image

Ces algorithmes exploitent une stratégie de division du plan image et de résolution de sous-problèmes de visibilité. Chaque subdivision du plan image est examinée, si la visibilité ne pose pas de problèmes spécifiques dans cette région (les cas (a) et (b) de l'algorithme du peintre par exemple) alors l'affichage est réalisé dans cette région de l'image. Dans le cas contraire, la région est divisée de nouveau en sous-régions et on réitère le processus de manière récursive. La subdivision s'arrête lorsque la région est de la taille du pixel. L'affichage concerne alors le polygone qui contient le pixel.

4.1 L'algorithme de Warnock

L'algorithme de Warnock [1969] subdivise l'image en quatre régions égales. À chaque itération, on teste pour une région si :

1. Tous les polygones sont à l'extérieur de la région. Pas d'affichage.
2. Un et un seul polygone intersecte la région, ou est contenu dans la région, ou contient la région. La partie du polygone correspondante est affiché.
3. Plus d'un polygone intersecte, ou est contenu, ou contient la région. Dans ce cas, on détermine si un polygone contenant la région correspond à une facette qui est devant toutes les autres, auquel cas le polygone est affiché. Autrement, une subdivision est effectuée.

Pour déterminer si un facette polygonale est devant toutes les autres, on calcule la profondeur (ici le z) de l'intersection de la ligne de vue en chaque points extrémités de la région avec tous les plans des facettes. Si les quatre z calculés pour une facette sont supérieurs à l'ensemble des z déterminés, alors cette facette est devant les autres.

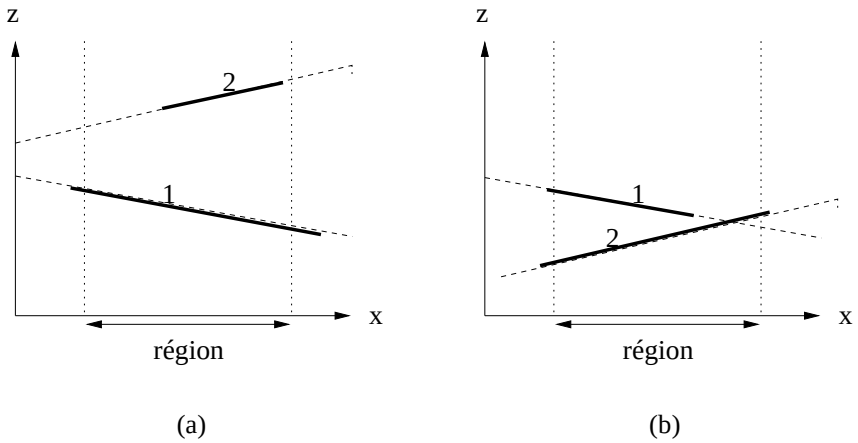


Figure 3: *Warnock : test si un polygone est devant les autres. (a) le polygone 1 est affiché ; (b) la région est divisée.*

L'algorithme :

```
Warnock(Liste_Poly, R)
Liste_Poly : liste de polygones
R : région de l'image

si (Liste_poly a 0 polygone dans la region courante) alors
          afficher(couleur_fond)
si  (Liste_poly a un seul polygone dans la region courante
     ou un polygone devant les autres) alors
        afficher(polygone)
sinon
     diviser région en R1, R2, R3, R4
     Warnock(List_Poly, R1)
     Warnock(List_Poly, R2)
     Warnock(List_Poly, R3)
     Warnock(List_Poly, R4)
finsi
```

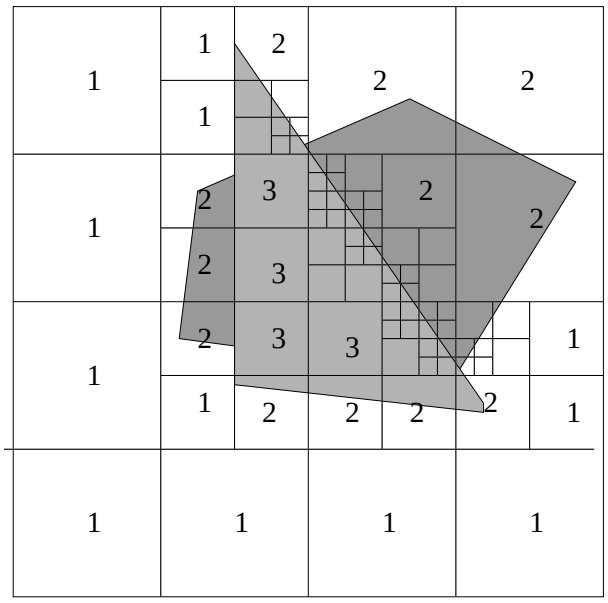


Figure 4: *Warnock : les différentes situations dans les régions.*

- ☞ L'algorithme de Warnock est à précision image.
- ☞ Il existe des améliorations possibles à cet algorithme, en particulier effectuer des subdivisions aux sommets des polygones (économie de subdivisions), ou bien effectuer des subdivisions le long des côtés des polygones (pour une précision objet).

5 z-buffer

L'algorithme du z-buffer [1974] consiste à stocker, pour chaque pixel de l'écran, non seulement les valeurs couleurs (framebuffer) mais aussi la profondeur (z-buffer). Le z-buffer est initialisé à l'infini. Durant le processus d'affichage d'un polygone, si le point traité a un z inférieur à celui correspondant dans le z-buffer, alors ce point est affiché et sa profondeur z remplace l'ancienne valeur dans le z-buffer.

L'algorithme :

```
Pour chaque polygone
  pour chaque pixel (u,v) du polygone
  z = profondeur du point du polygone correspondant au pixel (u,v)
  si z < z-buffer(u,v) alors
    afficher(u,v,couleur)  /* framebuffer(u,v) = couleur */
    z-buffer(u,v) = z
  finsi
 finpour
finpour
```

- ☞ Algorithme très simple à précision image,
- ☞ un tri suivant la profondeur n'est pas nécessaire,
- ☞ aucune comparaison entre objet,
- ☞ nécessite de l'espace mémoire.

5.1 Application dans le cas de l'algorithme de balayage de lignes

L'algorithme du z-buffer s'integre très bien dans celui de balayage de lignes pour traiter le cas des polygones projection de facettes polygonales.

Si la facette traitée est plane, alors soit :

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

l'équation du plan correspondant. Donc :

$$z = (-d - ax - by)/c.$$

et :

$$\Delta z = -\frac{a}{c}\Delta x - \frac{b}{c}\Delta y$$

L'algorithme de balayage de lignes peut donc facilement être modifié pour prendre en compte la profondeur z en chaque pixel.

L'algorithme de balayage de lignes s'écrit :

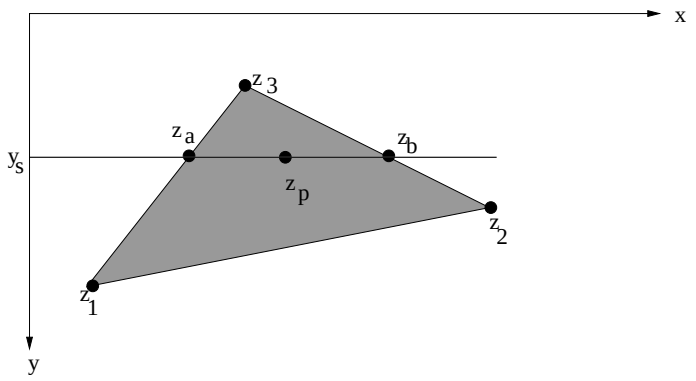
```
1. y est initialisé à la valeur de la première entrée de la table
   des côtés

2. répéter jusqu'à que la table active soit vide

  2.1 mettre dans la table active les entrées de la table des côtés
      dont le ymin est égal à y.
  2.2 enlever de la table active les entrées dont le ymax est égal
      à y.
  2.3 trier la table active suivant les xi croissants.
  2.4 remplir la ligne suivant la règle de parité en utilisant les
      xi de la table active.


  2.5 incrémenter y de 1.
  2.6 mettre à jour les xi dans la table active
```

Exercices :

1. Quelles modifications doit-on apporter à cet algorithme pour traiter le z-buffer dans le cas d'une projection orthographique.

2. Même question dans le cas d'une projection perspective.

Dans le cas d'une facette non plane ou d'une surface plus générale, il est toujours possible d'appliquer cet algorithme en déterminant les valeurs de z par interpolation des valeurs aux sommets.



Exemple : interpolation linéaires de valeurs aux sommets.

$$z_a = z_3 - (z_3 - z_1)(y_3 - y_s)/(y_3 - y_1),$$

$$z_b = z_3 - (z_3 - z_2)(y_3 - y_s)/(y_3 - y_2),$$

$$z_p = z_b - (z_b - z_a)(x_b - x_p)/(x_b - x_a).$$

- ☞ L'algorithme du z-buffer est le meilleur choix pour une implémentation matérielle.
- ☞ Nécessite de la mémoire, mais la mémoire n'est plus vraiment un problème.
- ☞ Il est possible de traiter les éléments de la scène en ligne, c'est à dire rajouter un objet sans retraiter l'ensemble de la scène mais juste en reprenant le z-buffer en l'état.
- ☞ C'est un algorithme à précision image.

Exercice :
Supposons que le z-buffer soit implementé avec une précision de 16 bits. Supposons par ailleurs qu'un objet soit situé à $500m$ du point d'observation. À Quelle distance minimale de cet objet doit être situé un deuxième objet pour être correctement pris en compte par le z-buffer ?

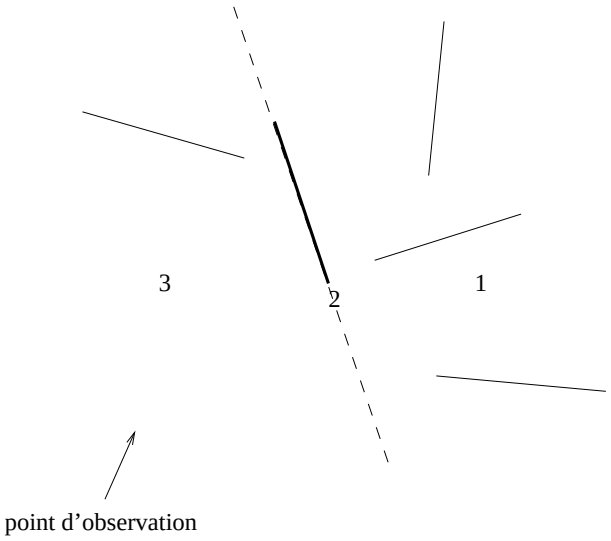
6 Les BSP-trees

Les Binary Space-Partitionning Trees [1980] servent à determiner des relations de visibilité entre les objets présents dans la scène. Ils définissent, pour une scène constituée de facettes polygonales, un ordre d'affichage de ces facettes en fonction de la position du point d'observation. Les algorithmes correspondant nécessitent un pré-calcul du BSP-tree pour chaque scène considérée et sont donc bien adaptés aux applications où le point de vue change mais non à celle où la scène observée change.

L'idée directrice des BSP-trees est de partionner l'espace en régions à l'aide des plans définis par les facettes polygonales de la scène. Cette partition de l'espace determine des ordres de visibilité pour les facettes en fonction de la région où se situe le point d'observation.

La relation d'ordre pour la visibilité repose sur le fait que pour une facette donnée, l'affichage doit se faire suivant :

1. Les facettes situées dans le 1/2 espace ne contenant pas le point d'observation.
2. La facette traitée.
3. Les facettes situées dans le 1/2 espace contenant le point d'observation.

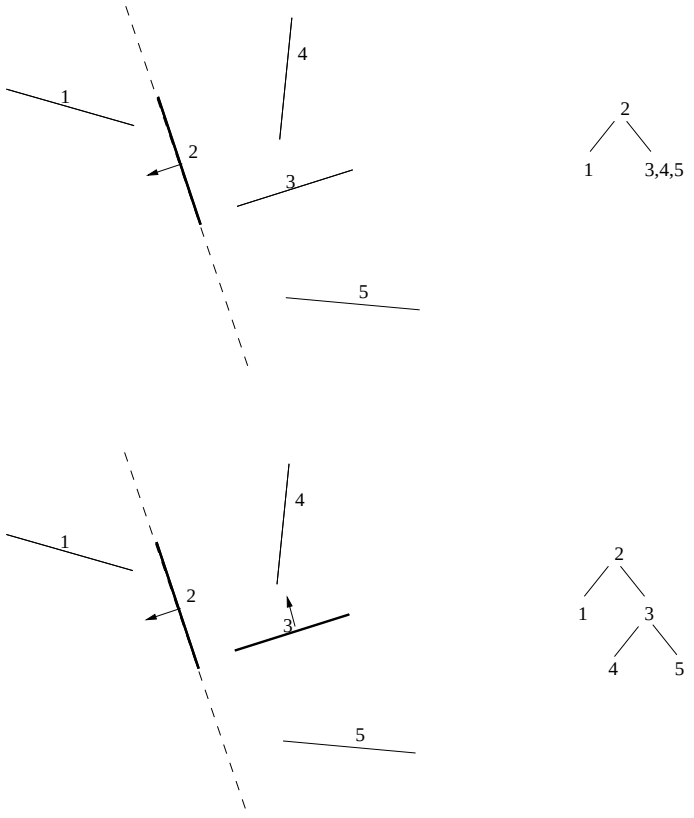


Ordre d'affichage des facettes.

En réiterant de manière récursive ce raisonnement dans les 1/2 espaces précedents, on peut définir un ordre d'affichage pour l'ensemble des facettes constituant la scène. Pour cela, on construit un arbre binaire de facettes polygonales: le BSP-tree.

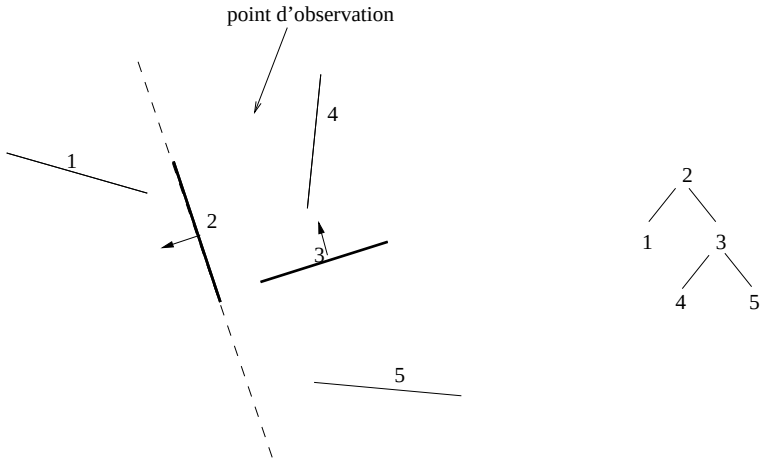
En partant d'une facette polygonale racine :

1. la scène est divisée en deux demi-espaces,
2. les facettes qui sont intersectées par le plan de la facette racine sont partagées en deux facettes suivant l'intersection.
3. pour chaque demi-espace, une nouvelle facette racine est choisie pour réiterer le processus. Cette facette constitue une nouvelle branche dans l'arbre binaire. Le processus s'arrête lorsque l'ensemble des facettes de la scène sont dans l'arbre binaire.

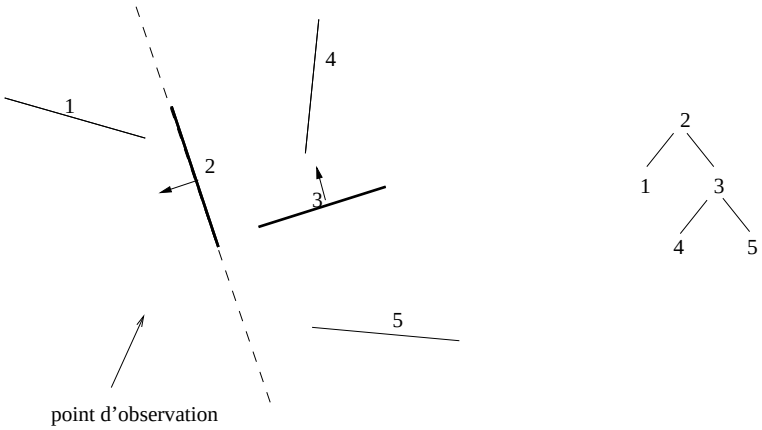


Algorithme d'affichage :

```
Affiche_BSP_Tree(BSP_Tree arbre)

si arbre non vide alors
 si le point d'observation est devant arbre->racine alors
   Affiche_BSP_Tree(arbre->filsdroit)  /* 1/2 plan arriere de arbre->ra
   Affiche_facette(arbre->racine)
   Affiche_BSP_Tree(arbre->filsgauche) /* 1/2 plan avant de arbre->raci
 sinon
   Affiche_BSP_Tree(arbre->filsgauche)
   Affiche_facette(arbre->racine)
   Affiche_BSP_Tree(arbre->filsdroit)
 finsi
finsi
```



Ordre d'affichage : 1,2,5,3,4



Ordre d'affichage : 4,3,5,2,1

- L'arbre doit être reconstruit lorsque la scène est modifiée.
- L'affichage de la scène se fait par simple parcourt de l'arbre binaire.
- Le choix des facettes polygonales racines pour la construction de l'arbre binaire se fait, pour chaque branche, en testant quelques facettes prises au hasard. Celle qui provoque le moins de subdivisions de facettes est choisie.
- L'algorithme est à précision objet.