Algorithmes des vus et caches

Christian NGUYEN

Departement d'informatique

Universite de Toulon et du Var

Introduction

Différences et similitudes

On distingue ces algorithmes principalement par l'espace dans lequel ils travaillent : objet ou image (mais aussi : modèle de représentation, périphérique de sortie, . . .).

Notions communes :

– le tri géométrique dont le critère est une quantité géométrique et qui donne un ordre de priorité sur les objets,
– la cohérence d'image : deux représentations d'un même objet sont peu différentes si certains paramètres ont peu variés.

Cohérence d’image

Cohérence de faces : faces petites vis à vis de l’écran.

Cohérence d’arêtes (intersectantes et non intersectantes)

Cohérence de ligne de balayage : un segment visible sur une ligne l’est sur la ligne suivante.

Cohérence de profondeur : des faces se superposant dans le plan de projection ont des profondeurs différentes.

Cohérence de pages écrans : en animation les images successives d’une scène sont peu différentes les unes des autres.

Classification

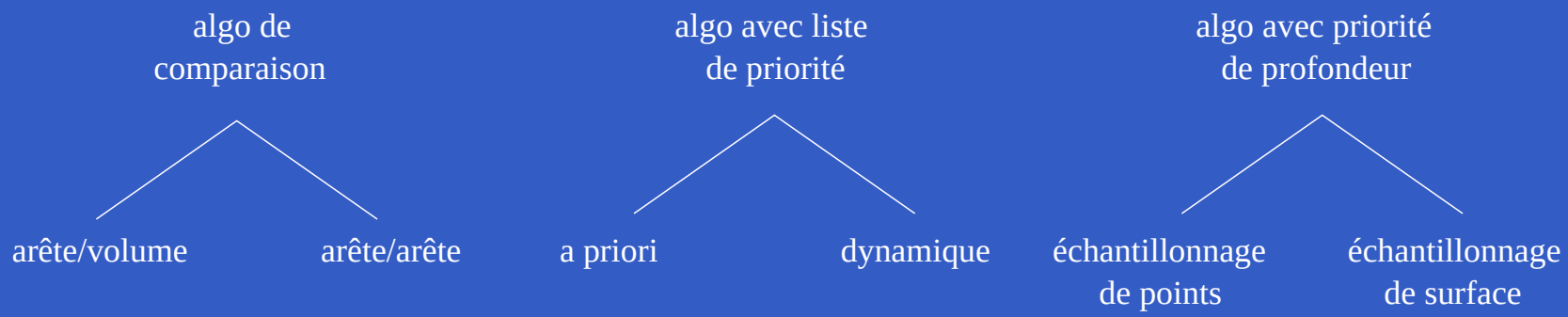
Trois catégories [Sutherland, Sproull, Schumacker 74] :

- ceux qui travaillent dans l'espace objet ($\theta(n^2)$),
- ceux qui travaillent dans l'espace image par une projection sur le plan image qui entraîne une moins grande précision dans les calculs et un traitement plus rapide de l'image ($\theta(n)$ fonction de la résolution),
- ceux qui travaillent de manière hybride : pré-traitement dans l'espace objet (liste de priorité des faces, . . .) puis traitement dans l'espace image.

Classification

espace objet
résolution exacte

espace image
résolution limitée

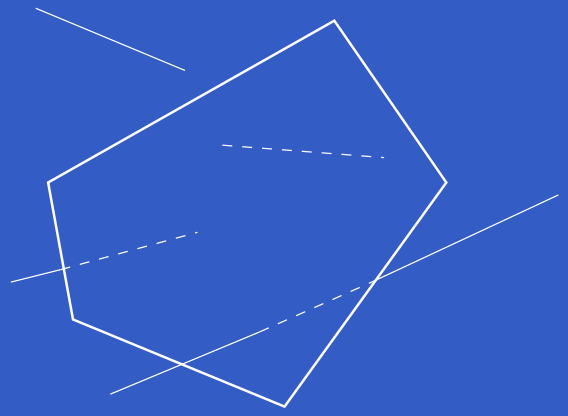


Algorithmes caractéristiques

Arête/volume [Roberts 63]

Scène uniquement constituée de polyèdres convexes. A partir de la définition paramétrique d'une arête, on distingue les cas suivants par rapport à un solide convexe :

1. l'arête est complétement visible,
2. l'arête est complétement cachée,
3. l'arête est partiellement visible (1 portion),
4. l'arête est partiellement visible (2 portions).



La recherche de l'inclusion d'un point dans un volume convexe est très coûteuse ($\Theta(n^2)$).

Arête/arête [Appel 67]

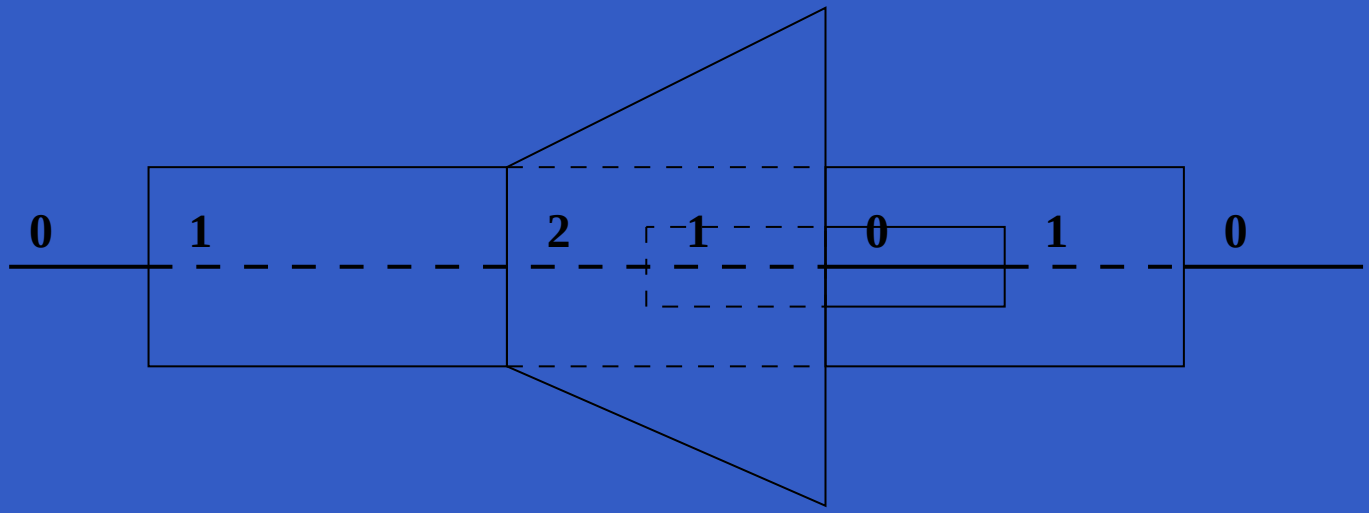
Polyèdres dont les faces sont décrites par une liste de sommets dans le sens trigonométrique.

1. élimination des faces arrières (*culling*),
2. comparaison des arêtes potentiellement visibles avec les faces avants du polyèdre : quantité d'invisibilité le long de l'arête.

Utilisation de la cohérence d'arêtes pour limiter les temps de calcul : à chaque intersection QI = QI $\pm$ 1. Si une arête du contour polygonal traverse le triangle formé de l'arête à tester et de l'œil, elle coupe l'arête en projection et son polygone l'obscurcit.

Arête/arête [Appel 67]

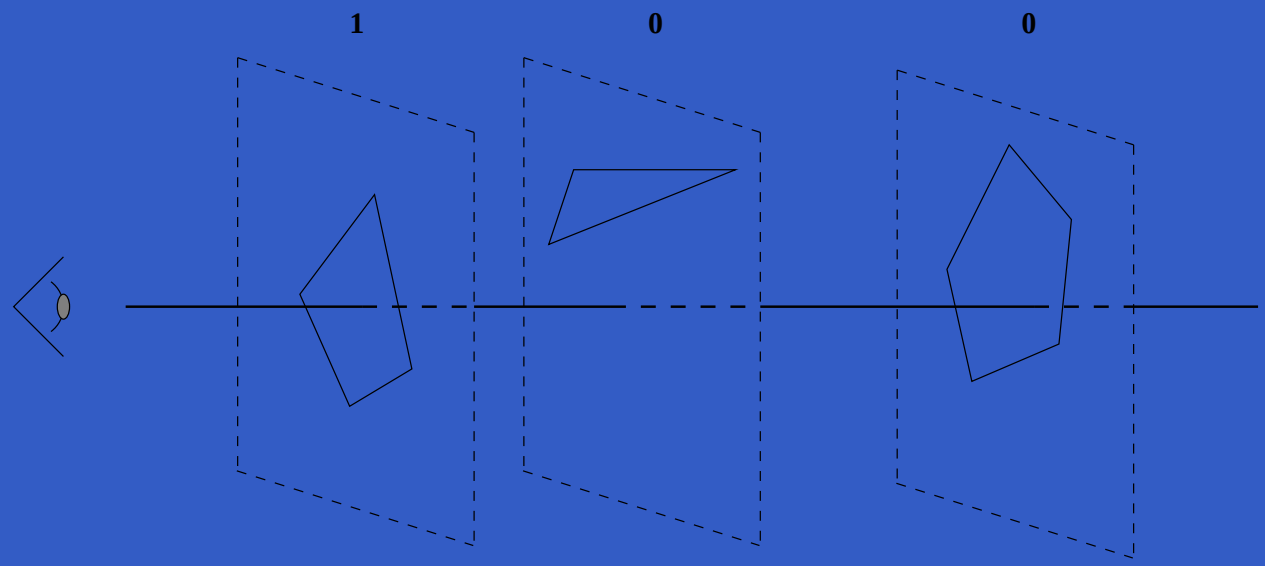
Evolution de la quantité d'invisibilité le long d'une arête, par rapport à un point de vue donné.



Arête/arête [Appel 67]

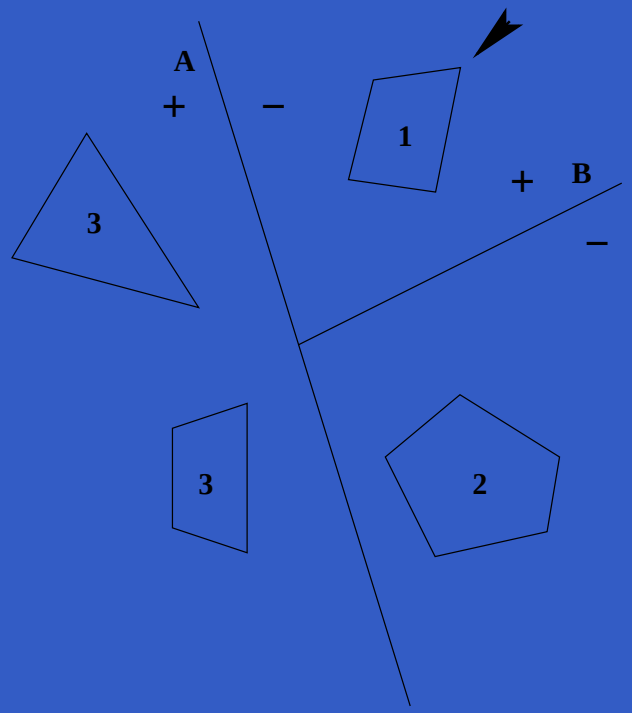
La QI du sommet initial est déterminée par une comparaison systématique de celui-ci avec l'ensemble des faces. Le dénombrement se fait suivant deux conditions :

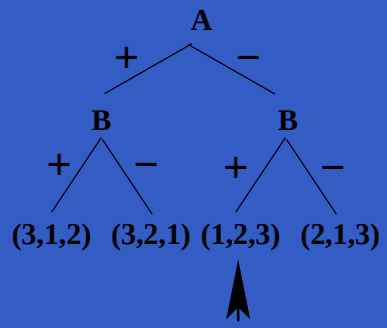
– la droite reliant le sommet à l'œil doit couper le plan contenant la face *entre* l'œil et le sommet,
– le point d'intersection appartient à la face (contour polygonal).



Priorité a priori [Schumacker 69]

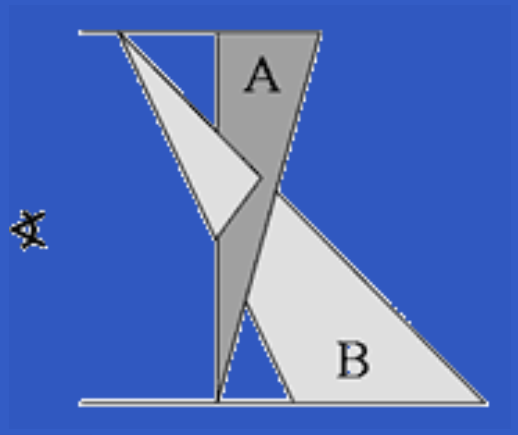
Scène composée d'objets convexes et partitionnable par des plans (*clusters*). On peut assigner une priorité aux faces indépendamment du point de vue.





Priorité dynamique [Newell 72]

Algorithme du peintre : élimination des faces arrières, tri des faces restantes (suivant Z), résolution des ambiguïtés en cas de chevauchements, affichage des polygones dans l'ordre calculé.
Le tri des faces suivant leur profondeur se fait en calculant la côte max. des sommets constituant chaque polygone.
Levés des ambiguïtés : série de tests (comparaison de la face courante de profondeur Z_c avec celles dont $Z_{max} \leq Z_c$) de plus rapide au plus lent. Test réussi : face courante plus éloignée que face testée.

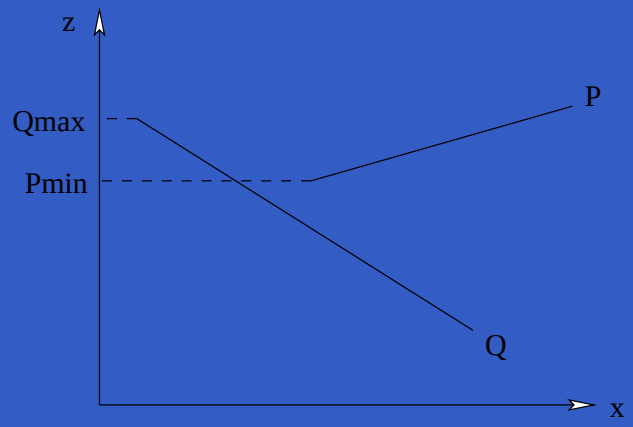


Priorité dynamique [Newell 72]

Test 1 : priorité suivant Z. Test réussi si $Z_{max}(Q) \leq Z_{min}(P)$ (face suivante).

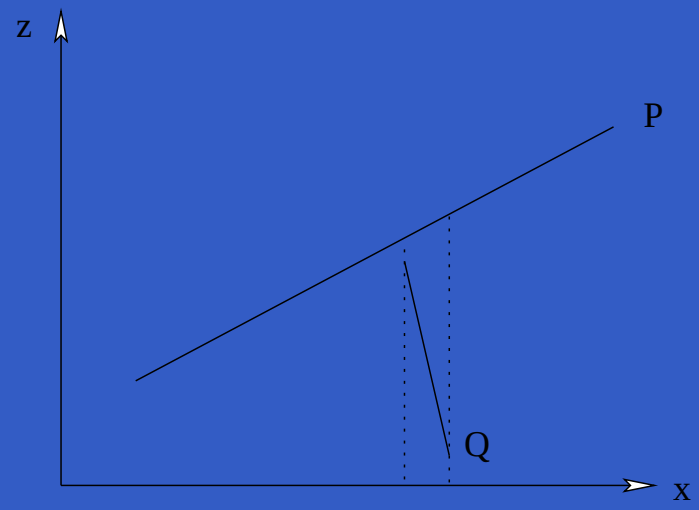
Test 2 : rectangles englobants (suivant x puis y). Test réussi s'il n'y a pas chevauchement.

Test 3 : substitution des coord. des sommets de la face P dans l'équation du plan support de la face Q. Test réussi si tous les sommets donnent un résultat de même signe (P d'un seul côté de Q).



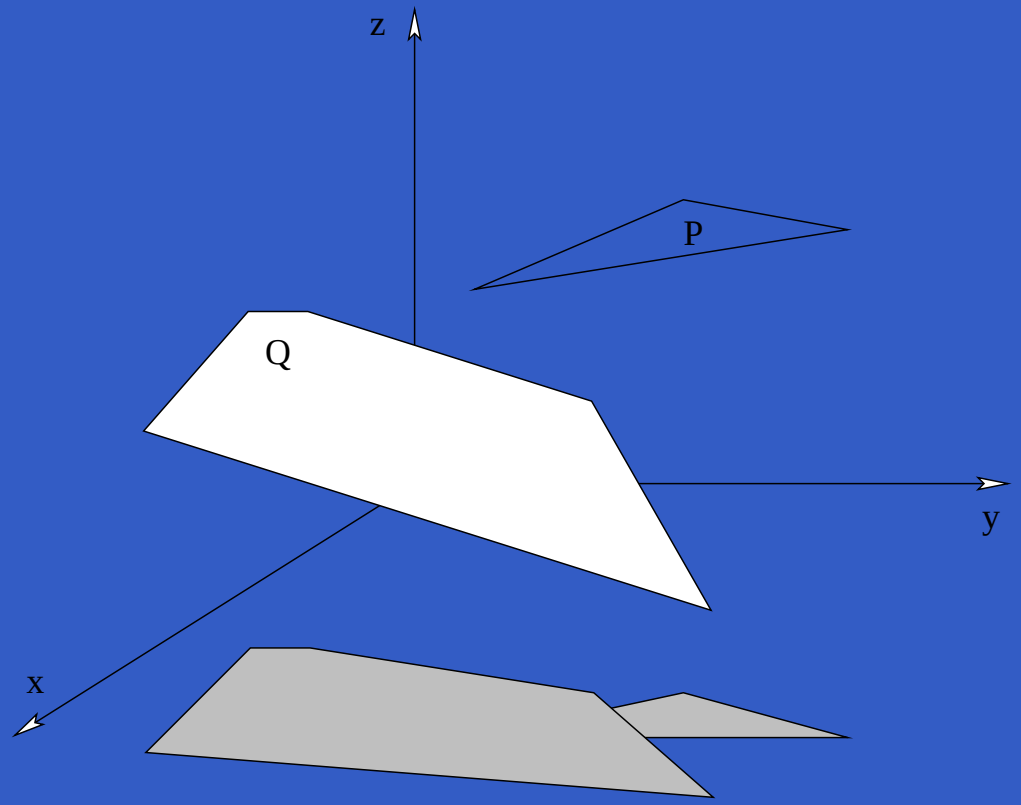
Priorité dynamique [Newell 72]

Test 4 : tous les sommets de Q sont-ils plus près du point de vue que les points de P ayant même projecteur ?



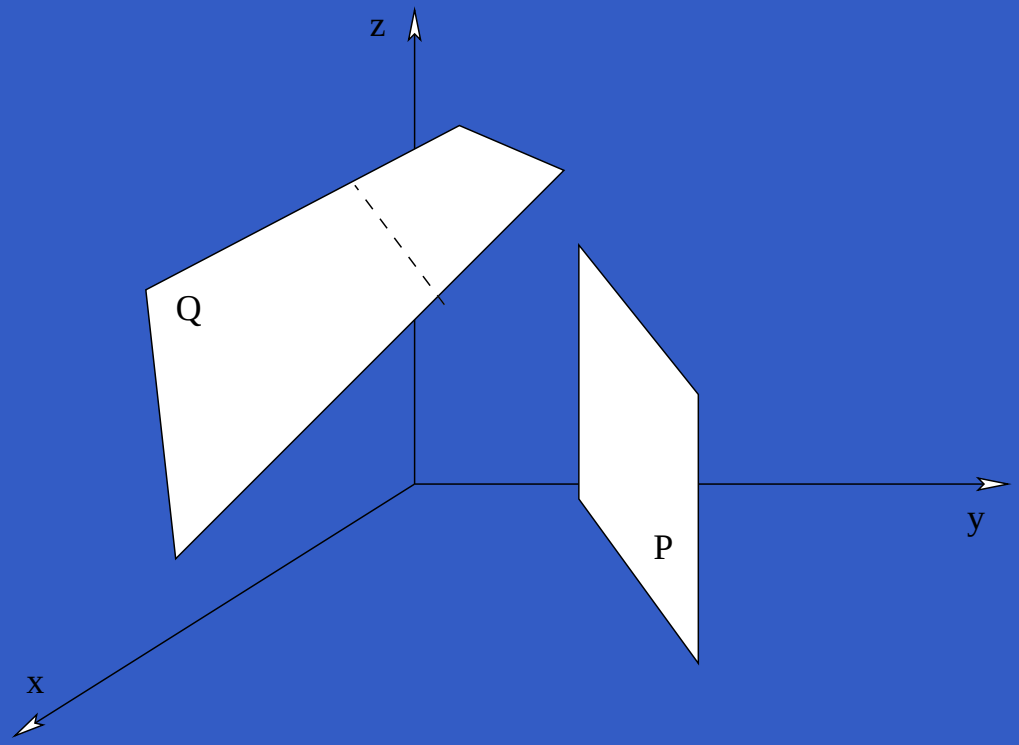
Priorité dynamique [Newell 72]

Test 5 : les projections de Q et P ne s'intersectent-elles pas ?



Priorité dynamique [Newell 72]

Test 6 : partage de l'un des plans par l'autre (création d'une arête) et reprise des tests.



Echantillonnage de surf. [Warnock 68]

Subdivision de l'espace écran (*area subdivision*). Analyse du contenu de chaque fenêtre (*looker*), 3 cas :

- il n'y a rien dans la fenêtre,
- ce qu'il y a est facile à dessiner,
- ce qu'il y a est complexe : division de la fenêtre en 4 (test d'arrêt : limite de résolution).

Le *looker* détermine chaque polygone par rapport à une fenêtre : englobant, intersectant et disjoint.

Le *thinker* détermine la complexité de la fenêtre (cas simple : polygones intersectants non superposés, polygone englobant en avant-plan, . . .).

Algorithmes actuels

Méthodes les plus populaires

L’approche "polygone par polygone" (*Z-buffer*) : simple à implanter, pas de limite à la complexité d’une scène mais pas de notion de cohérence.

L’approche "ligne de balayage" (*scan-line*) : gestion plus délicate (discrétisation, ombrage et texture simultanés) mais solution matérielle immédiate (génération suivant le balayage).

L’algorithme du Z-buffer

[Catmull 75] opère dans l’espace écran : comparaison des profondeurs de tous les points ayant même coordonnées (plus complexe dans l’espace objet car il faudrait déterminer si 2 points ont le même projecteur que dans l’espace écran).

Chaque pixel d’un polygone est mémorisé dans la mémoire image si sa profondeur est plus faible que le pixel correspondant de cette mémoire image (auquel cas, la nouvelle profondeur est mémorisée). Une dilatation de la scène est conseillée pour plus de précision.

Les pixels à l’intérieur d’un polygone donné sont ombrés incrémentalement et leur profondeur est évaluée par interpolation sur les sommets constituant le polygone.

Le Z-buffer est indépendant de la représentation des objets (même s’il est le plus souvant utilisé dans l’approximation par facettes).

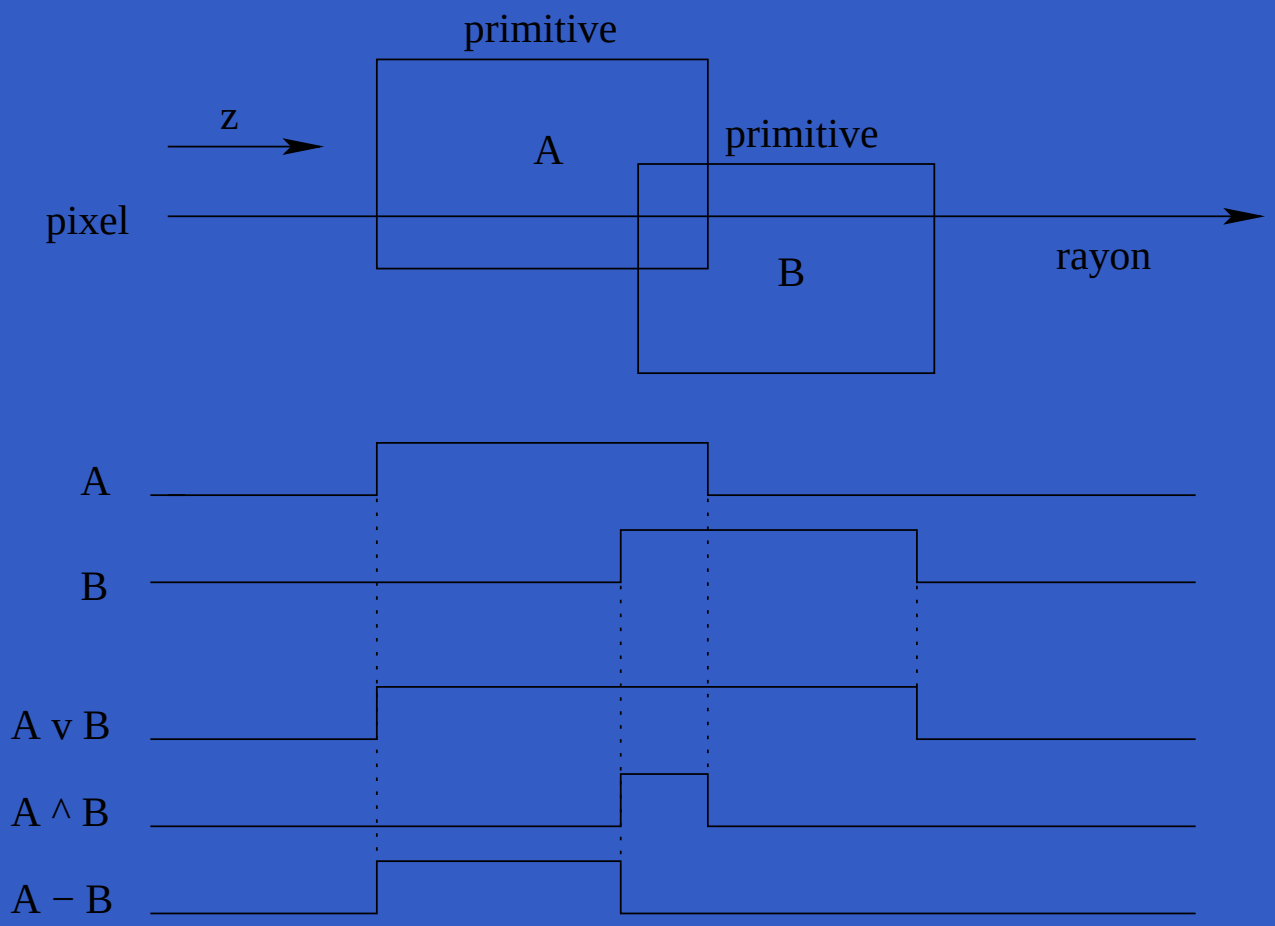
Z-buffer et CSG

Moins coûteux que le lancer de rayons : seule la 1ère intersection est à retenir (méthode dirigée par les objets et non les pixels [Rossignac, Requicha 86].

```
Algo Lancer de rayons

pour chaque pixel faire
  generer un rayon
  trouver ttes les surf. inter.
  evaluer l'arbre CSG pour
    determiner la frontiere de
    la surf. la plus proche
```

```
Algo Z-buffer

pour chaque objet faire
  pour chaque surface faire
    pour chaque "pt" de la surf. faire
      projeter P et appliquer Z-buffer
      si (visible) alors
        si en evaluant l'arbre CSG
          P est sur la frontiere alors
            m.a.j. du frame buffer
```

Z-buffer et CSG



Le rendu Z-buffer

Trois interpolations linéaires et trois boucles imbriquées :

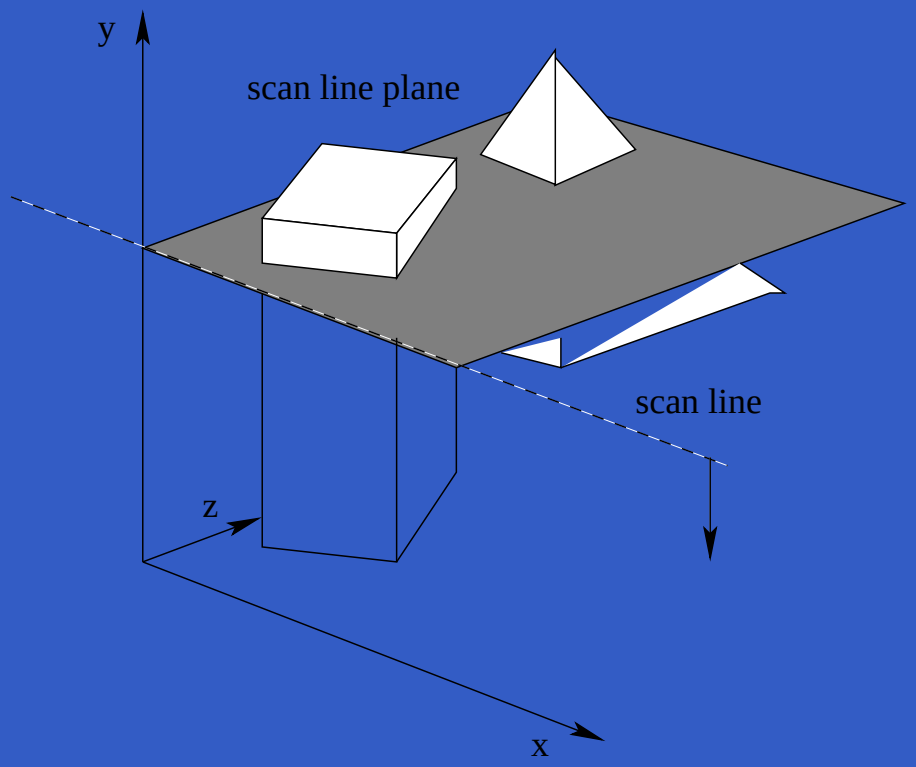
```
Algorithme Z-buffer

Initialiser le Z-buffer a la valeur MAX

pour chaque polygone faire
  Construire la liste des aretes (X, Z, I)
  pour y=Ymin a Ymax faire
    pour chaque segment de la TAA faire
      pour x=Xmin a Xmax faire
        Interpoler lineairement Z et I
        si (Z < Z-buffer[x][y]) alors
          Z-buffer[x][y] <- Z
          frame-buffer[x][y] <- I
        finsi
      finpour
    finpour
  finpour
```

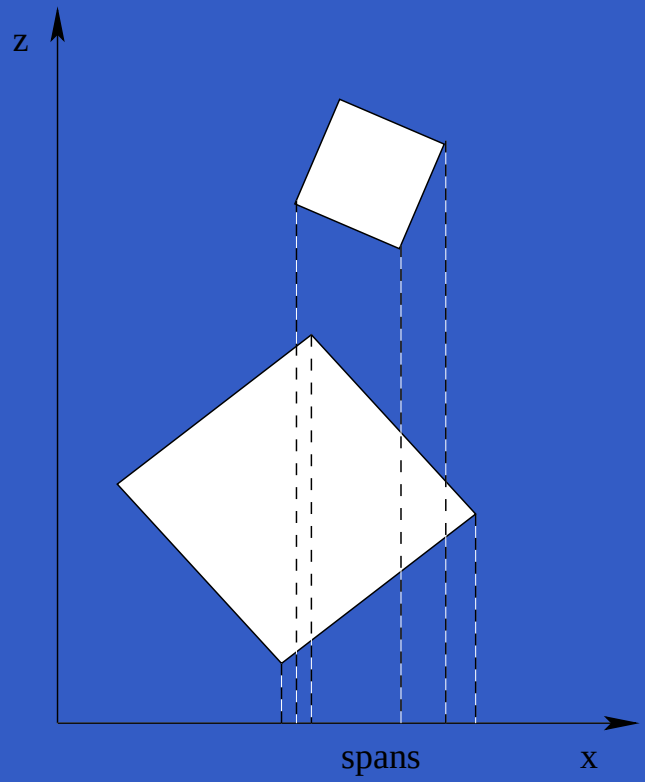
Le rendu scan-line

Philosophie des TA/TAA en 3D (tri des sommets des arêtes sur les Y, ...).

Cas simple : on suppose que les polygones résultants ne s'intersectent pas. La projection des sommets donne des *spans* (les segments de droite contenus dans les intersections d'arètes), dans lesquels il y a cohérence.

Le rendu scan-line





Le rendu scan-line

```
Algorithme scan-line

pour chaque polygone generer la TA

pour chaque plan de balayage faire
  m.a.j. des parametres d'ombrage
  Determiner les intersections plan/polygones
  Trier les AA suivant Xs
  Generer les "spans"
  pour chaque "span" faire
    Decouper les aretes suivant les "spans"
    Evaluer les portions d'aretes appartenant a un "span"
      et resoudre le probleme de profondeur
    Ombrer la portion d'arete visible (en ayant calculer
      les valeurs aux extremites)
  finpour
finpour
```