6. Elimination des parties cachées

6.1 Introduction

Etant donné un ensemble d'objets 3D et un point de vue désigné par la position de l'observateur, on s'intéresse à la détermination des parties susceptibles d'être éliminées (non visibles) dans le cas des projections parallèles ou perspectives.

- Formulation du problème simple
- Pas d 'algorithme optimal

1

3 catégories

- Algorithme opérant dans l 'espace objet
  - n objets, $O(n^2)$
  - précision dépend de la résolution des objets
  - algorithmes complexes
- Algorithme opérant dans l'espace image
  - n objets, p pixels, O(np)
  - précision dépend de la résolution de l'image
  - algorithmes + simples
- Algorithme hybride
  - combine les 2 techniques
  - pré-tri dans l'espace objets

2

Dans l'espace objet :

- **algorithme :**

```
Pour chaque objet de la scène faire
    { déterminer les parties de l'objet dont la vue n'est
     occultée par aucune partie des autres objets;
     tracer (ou afficher) ces parties avec la couleur
     appropriée;
    }
```

- **exemple : algorithme du Peintre**

Dans l 'espace image :

- **algorithme**

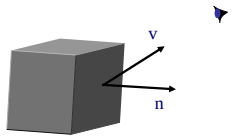
```
Pour chaque pixel de l 'image
    {déterminer l'objet le plus proche de l 'observateur qui
     est traversé par le projecteur passant par le pixel;
     afficher le pixel avec la couleur appropriée;
    }
```

- **exemple : le Z-buffer, lancer de rayon**

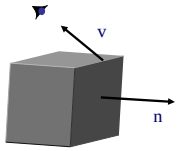
3

Test de visibilité

- Elimination des faces arrières
  - faces visibles



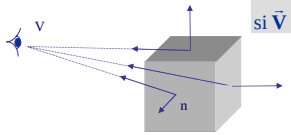
  - faces invisibles



4

Elimination des faces arrières

- **Les faces arrières sont les faces qui ne peuvent en aucun cas être visibles par l'observateur.**
- **On se place dans le cas d'objets convexes à facettes planaires.**
- **une face est visible par rapport à l'objet si le produit scalaire entre la normale sortante de la face et le vecteur vision est positif.**
- **Attention au sens de V**



si $\vec{\mathbf{V}} \cdot \vec{\mathbf{n}} > 0$ alors la face est visible de l'observateur

si $\vec{\mathbf{V}} \cdot \vec{\mathbf{n}} < 0$ alors la face est invisible

Parties cachées

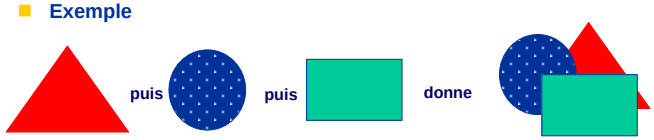
- Test de visibilité
  - normales sortantes
  - objets convexes
  - surfaces opaques
  - objet-observateur
- Parties cachées
  - occultation des objets entre eux
  - partielle ou totale
  - parfois complexe

6.2 Algorithme du peintre

- **Principe**

Comme le peintre, on commence par dessiner (afficher) l'arrière-plan et ensuite on recouvre les couches précédentes par une nouvelle couche. Ainsi, la dernière couche et les parties non-recouvertes sont alors visibles.

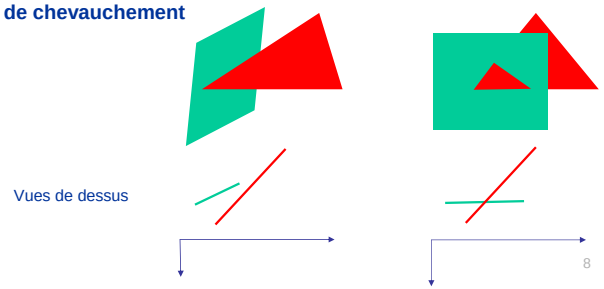
- **Exemple**



Etapes de l 'algorithme

- Élimination des faces arrières
- Tri des faces restantes en fonction de leur z_{min}
- Levée des ambiguïtés : études des chevauchement en x,y,z
- Afficher dans l'ordre les faces et les remplir avec la couleur appropriée.

- **Cas de chevauchement**



Algorithme de Newell-Newell-Sancha

(cas où l 'observateur est à l'infini sur l'axe des z positifs)

- Tri des polygones suivant les z_{min} croissants (le plut petit est le plus éloigné)

  soit P et Q : 2 polygones dans la liste tels que $z_{min}(P) < z_{min}(Q)$

- Test sur les chevauchements (boites limites)

  Si $zmax(P) < zmin(Q)$
  pas de chevauchement : le tri est correct.
  Sinon **Si** $[xmin(P)\ xmax\ (P)] \cap [xmin(Q)\ xmax\ (Q)] = \varnothing$
  pas de chevauchement sur l'écran
  Sinon **Si** $[ymin(P)\ ymax\ (P)] \cap [ymin(Q)\ ymax\ (Q)] = \varnothing$
  pas de chevauchement sur l'écran
  Sinon **Test des plans de coupe**

Si pas de chevauchement : tri entre P et Q est correct, sinon il faut lever les ambiguïtés

Algorithme de Newell-Newell-Sancha (suite)

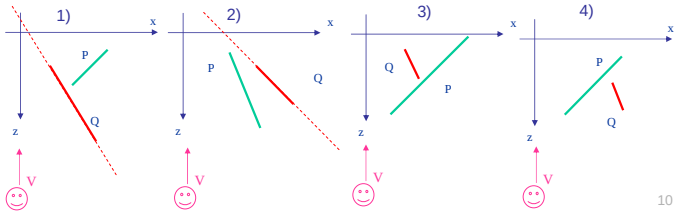
- Test sur les plans de coupe

Equation du plan de Q :

1) **si P est entièrement du coté du plan de Q, le plus éloigné du point de vue**
P ne peut occulter Q, donc dessiner P puis

2) **si P est entièrement du coté du plan de Q, le plus proche du point de vue**
Q ne peut occulter P, donc dessiner Q puis P

sinon refaire les tests sur les plans de coupe en inversant P et Q (cas 3 et 4)



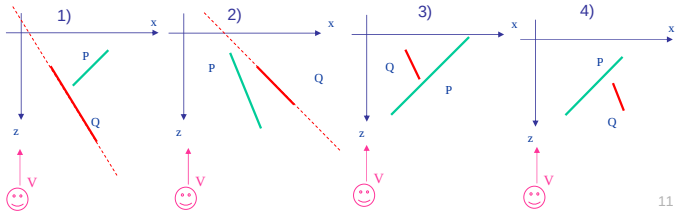
Algorithme de Newell-Newell-Sancha (suite)

- Test sur les plans de coupe

Equation du plan de P :

3) **si Q est entièrement du coté du plan de P, le plus éloigné du point de vue**
Q ne peut occulter P, donc dessiner Q puis P

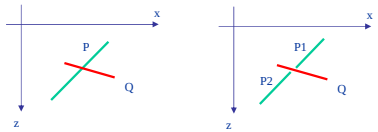
4) **si Q est entièrement du coté du plan de P, le plus proche du point de vue**
P ne peut occulter Q, donc dessiner P puis Q



Algorithme de Newell-Newell-Sancha (suite)

si P peut occulter Q et si Q peut occulter P alors il y a recouvrement cyclique

- **Recouvrement cyclique**
  si inter pénétration, on coupe P le long du plan de coupe de Q et on obtient 2 polygones P1 et P2 que l'on replace judicieusement dans la liste.



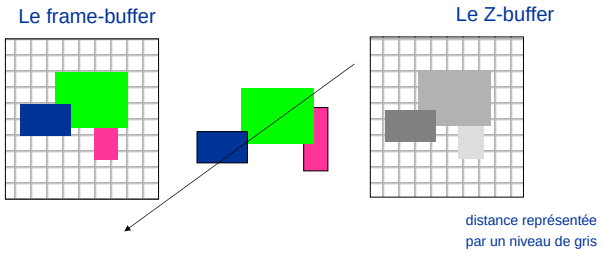
6.3 Le Z-buffer

- très simple à implémenter
- tampon d'écran de même taille que la mémoire d 'image
- on stocke les profondeurs (valeur en z) de chaque pixel représentant la scène

```
initialisation  /* l'observateur regarde vers les z négatifs */
   z-buffer initialisé à une valeur minimum  (éloigné de l'obs.)
   vidéo buffer initialisé à la couleur de fond

Principe
   Pour chaque polygone :
     pour chaque pixel (x,y) du polygone :
     • calculer la profondeur z(x,y) pour ce pixel
     • si z(x,y) > Zbuffer(x,y)  (objet + proche de l'observateur)
        {
        écrire les attributs du polygone dans la mémoire image
        remplacer Zbuffer(x,y) par z(x,y)
        }
```


Le Z-buffer



- couleur du pixel (RGB)
- taille de l'image
- visualisé à l'écran

- profondeur de l'objet le plus proche
- taille de l'image
- non visualisé à l'écran

Remarques sur le Z-buffer

- On peut préalablement supprimer les faces arrières
- La précision dépend de la taille du z-buffer (nombre de bits/pixel).
- L'ordre des polygones est sans importance
- Dans le cas de scène décrites par les sommets de facettes, la valeur de profondeur n'est pas définie de manière explicite pour toutes les "cases" du z-buffer

Remarques sur le Z-buffer

- Le calcul de z en tout point du polygone est très simple dans le cas de projection parallèle :

Si on connaît l 'équation du plan du polygone on peut simplifier le calcul de z de chaque point de la ligne de balayage

$$Ax + By + Cz + D = 0$$

$$z = -(Ax + By + D) / C$$

si on connaît z **en** $(x,y) = z_0$

alors z **en** $(x+1, y) = z_0 - A/C$ $\quad -A/C$ **constante**

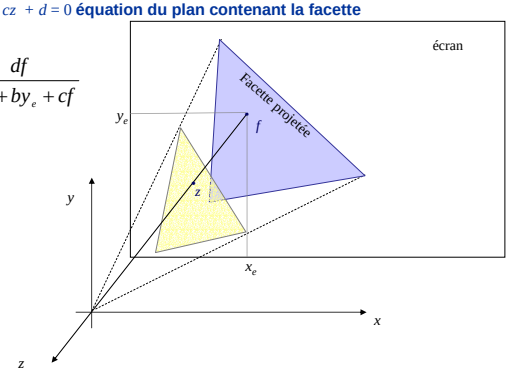
cas de projection perspective

- Le calcul de z en tout point du polygone est moins simple

avec f **= distance de l'écran à l 'observateur**

et $ax + by + cz + d = 0$ **équation du plan contenant la facette**

$$z = -\frac{df}{ax_e + by_e + cf}$$



OpenGL

- **Z-buffer**
  - Le nom du z-buffer est le `depth buffer`
  - Il faut l'activer de manière explicite
  - Possibilité de modifier la fonction de test (par défaut `GL_LESS`)
  - Rafraîchir le buffer entre 2 affichages.
  - Initialisation avec la valeur de `zfar`

6.4 Le lancer de rayon (version primaire)

En anglais : ray-tracing

Les algorithmes précédents utilisent la cohérence (liée à la scène)

Le ray-tracing est une technique simple de type z-buffer et qui permet d'intégrer très facilement les notions de transparence, et certains modèles d'éclairement de scènes, d'ombrages, etc...

- Le principe

On suit la trajectoire du rayon qui part de l'observateur, qui passe par le centre d'un pixel. Dans le cas de surfaces opaques : la première surface rencontrée est la bonne.

Ceci consiste donc à calculer des intersections entre les objets et le rayon

Pour accélérer on peut définir des volume englobants

bien adapté pour des surfaces gauches définies par leurs équations

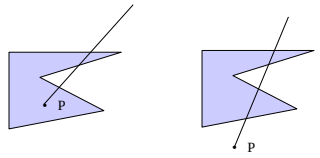
6.5 Remplissage et parties cachées

6.5.1 Introduction

Contours fermés les plus simples : polygones
test l'appartenance d'un pixel de l'image au polygone.
Généralement dans l'espace image (2D)

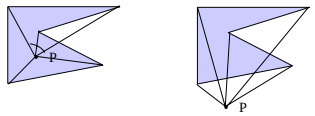
- **test de parité**

si le nombre d'intersection est impair
alors P est à l'intérieur



- **test des angles**

si somme des angles = 2π
alors P est à l'intérieur



6.5.2 Algorithmes par balayage

- **Principe**

Basé sur le balayage ligne par ligne (scan line algorithm)

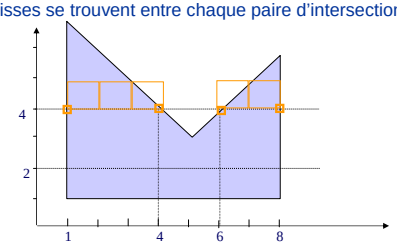
droite de balayage horizontale parallèle à l'axe des x

calculer des intersections entre cette droite et les arêtes du polygone : arêtes actives.

trier ces intersections (valeurs de x)

remplir et afficher tous les pixels se trouvant sur la ligne de balayage et dont les abscisses se trouvent entre chaque paire d'intersection

- **Exemple**



Algorithme par balayage

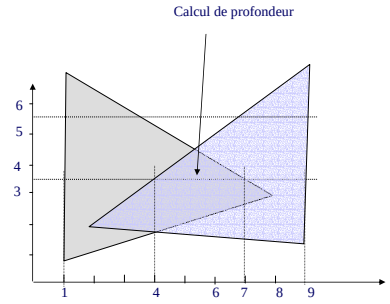
- Problèmes
  - valeurs non entières
  - intersection sur un sommet commun à plusieurs arêtes
  - arêtes parallèles à la ligne de balayage
- Eléments de solution
  - système de coordonnées modifié
  - minimum ou maximum local : double l'intersection
  - on ne traite pas les segments horizontaux

Algorithme de la liste triée des cotés actifs

- Déterminer l'intersection de chaque arête avec le milieu de la largeur des lignes de balayages et stocker chaque intersection $(x, y+ ½)$ dans une liste
- Pour chaque ligne de balayage, trier dans l'ordre croissant des abscisses la liste des intersections. (Choisir un algorithme de tri efficace)
- Extraire de chaque liste triée des paires $(x_1, y+ ½)$ $(x_2, y+ ½)$ $x_1 \leq x_2$
- Allumer les pixels de la ligne de balayage y pour les valeurs entières de x / $x_1 \leq x+½ < x_2$

6.5.3 Combinaison avec parties cachées

- Principe : Utiliser le même algorithme et traiter les cas de chevauchement en détectant les paires non uniformes (arêtes n'appartenant pas au même polygone)
- Exemple :



Algorithme des cotés actifs et parties cachées

- Gestion de 3 tables (initialisation)

TP : table des polygones, chaque polygone est décrit par

Id : identificateur du polygone
Eq. du plan : cœfficients ABCD
couleur: couleur de remplissage
flag : polygone actif ou non

Id	équation du plan	couleur	flag

TA: table des arêtes, chaque arête est décrite par

x initialisé pour $y_{min}+1/2$
y_{max}: hauteur max de l'arête
Δx : inverse de la pente
Id : identificateur du polygone

x	y_{max}	Δx	Id	

TAA : table des arêtes actives

y : ligne de balayage

Initialisée avec les arêtes dont le $y_{min} = y$

y	
y+1	
y+2	
y+3	
...	

Liste des cotés actifs

Algorithme des cotés actifs et parties cachées (suite)

- Générer le tracé (ligne y)
  - On procède comme pour l'algorithme de cotés actifs mais on détecte les "mauvaises paires" en modifiant le flag (bouléen) de chaque polygone à chaque fois qu'on traite un coté de celui ci :
  - Si le flag de 2 polygones sont à 1 (vrai) simultanément , il faut calculer les profondeurs et on allume le pixel avec la couleur du polygone correspondant au z_{min}
  - Sinon on traite les paires sans calculer la profondeur et on allume le pixel avec la couleur du polygone correspondant au coté actif.

Algorithme des cotés actifs et parties cachées (suite)

- Propager à la ligne suivante (ligne $y+1$)
  - Si $y+1 > y_{min}$, l'arête n'est pas propagée (elle n'est plus active)
  - Si non,
    - modifier x en le remplaçant par $x+\Delta x$
    - ajouter l'arête à la liste de la ligne ($y+1$)
  - Générer le tracé en ($y+1$)