Création d'images virtuelles

Nicolas Holzschuch
iMAGIS/GRAVIR-IMAG

*i*MAGIS is a joint project of CNRS - INPG - INRIA - UJF

iMAGIS

Création d'images virtuelles

- Introduction à la création d'images virtuelles
- Techniques de base :
  - Modèle géométrique
  - Affichage
  - Animation
  - Un peu de réalisme
- Techniques avancées au deuxième semestre

iMAGIS

Création d'images virtuelles

- Pipeline graphiqe
- Élimination des parties cachées
- Modèles de couleur
- Modèles d'illumination locale
- Textures
- Modélisation géométrique
- Animation
- Éclairage local : ombres, reflets
- Éclairage global
- Rendu volumique

iMAGIS

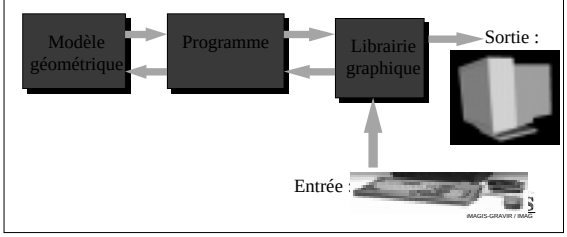
Livre de cours ?

- Les diapos de chaque cours sont (seront) sur le site web du DEA
- Si vous voulez en savoir plus :
  - Foley, vanDam, Feiner et Hughes
  - *Computer Graphics, Principles and Practice*
    - Gros, complet, cher.
  - Ou bien : *Introduction to Computer Graphics*
    - Moins gros, moins complet, moins cher
    - Traduit en français : *Introduction à l'infographie*

iMAGIS

Plan général

- Pipeline graphique



iMAGIS

Modèle géométrique

- Description des objets
  - Forme
  - attributs
  - Comportement, propriététs
- Stocké en mémoire par l'application
- Modifiable par l'application
- Envoyé à l'affichage

iMAGIS

Affichage

- Conversion :
  - Représentation interne au programme
  - Transformer en primitives graphiques standard
  - lignes, cercles, polygones
  - Transformations géométriques
  - À l'aide de la librairie graphique
- Interactions:
  - Boucle événements/call-backs
  - File d'événements à traiter

iMAGIS

Modèle géométrique

- Représentation surfacique
- Représentation volumique, etc
- 2 cours spécifiques, 6-13 novembre

iMAGIS

Plan

- Transformations géométriques
- Projections, caméra, perspective
- Élimination des parties cachées

iMAGIS

Transformations géométriques

- Représentation vectorielle des points
  - Points attachés aux primitives graphiques
  - Sommets, centres, données volumiques…
- Transformations sur ces données
  - Translation, rotation, changement d'échelle…
  - Projections :
    - Perspective, parallèle…
  - Notation unifiée ?

iMAGIS

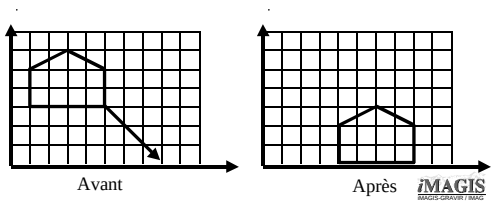
En 2 dimensions

- On commence en 2D
  - Plus facile à représenter
- Chaque point est transformé:
  - $x' = f(x,y)$
  - $y' = g(x,y)$
- Comment représenter la transformation ?

iMAGIS

Translations

- Modification simple :
  - $x' = x+t_x$
  - $y' = y+t_y$



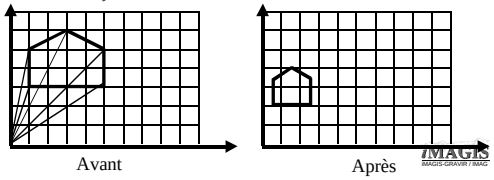
iMAGIS

Notation vectorielle

- On utilise des vecteurs pour la représentation
  - C'est plus simple
- Un point est un vecteur : $\begin{bmatrix} x \\ y \end{bmatrix}$
- Une translation est une somme vectorielle :
  $P' = P + T$

Changement d'échelle

- Les coordonnées sont multipliées par le facteur de changement d'échelle :
  - $x' = s_x\, x$
  - $y' = s_y\, y$



Notation matricielle

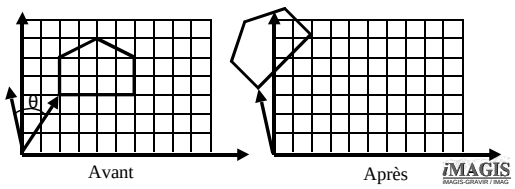
- C'est une multiplication matricielle :

$$P' = \mathbf{S}P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation

- Rotation en 2D :
  - $x' = \cos\theta\, x - \sin\theta\, y$
  - $y' = \sin\theta\, x + \cos\theta\, y$



Notation matricielle

- Rotation = multiplication matricielle :

$$P' = \mathbf{R}P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Unification

- Notation simple, concise
- Mais pas vraiment unifiée
  - Addition ou bien multiplication
  - Comment faire pour concaténer plusieurs transformations ?
- On veut une notation unique
  - Qui permette de noter aussi les combinaisons de transformations
  - Comment faire ?

Coordonnées homogènes

- Outil géométrique très puissant :
  - Utilisé partout (Image, Vision, Robotique)
  - Sera étudié en détail dans le cours de géométrie projective
- On ajoute une troisième coordonnée, *w*
- Un point 2D devient un vecteur à 3 coordonnées :

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Coordonnées homogènes

- Deux points sont égaux si et seulement si :
  - $x'/w' = x/w$ et $y'/w' = y/w$
- *w=0:* points « à l'infini »
  - Très utile pour les projections, et pour certaines splines

Translations en c. homogènes

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \begin{cases} \frac{x'}{w'} = \frac{x}{w} + t_x \\ \frac{y'}{w'} = \frac{y}{w} + t_y \end{cases}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ w' = w \end{cases}$$

Changement d'échelle

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \begin{cases} \frac{x'}{w'} = s_x \frac{x}{w} \\ \frac{y'}{w'} = s_y \frac{y}{w} \end{cases}$$

$$\begin{cases} x' = s_x x \\ y' = s_y y \\ w' = w \end{cases}$$

Rotation

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \begin{cases} \frac{x'}{w'} = \cos\theta \frac{x}{w} - \sin\theta \frac{y}{w} \\ \frac{y'}{w'} = \sin\theta \frac{x}{w} + \cos\theta \frac{y}{w} \end{cases}$$

$$\begin{cases} x' = \cos\theta x - \sin\theta y \\ y' = \sin\theta x + \cos\theta y \\ w' = w \end{cases}$$

Composition des transformations

- Il suffit de multiplier les matrices :
  - composition d'une rotation et d'une translation:
    M = **RT**
- Toutes les transformations 2D peuvent être exprimées comme des matrices en coord. homogènes
  - Notation très générale

Rotation autour d'un point Q

- Rotation autour d'un point Q:
  - Translater Q à l'origine ($\mathbf{T}_Q$),
  - Rotation autour de l'origine ($\mathbf{R}_\Theta$)
  - Translater en retour vers Q ($-\mathbf{T}_Q$).

$$P' = (-\mathbf{T}_Q)\mathbf{R}_\Theta\mathbf{T}_Q\,P$$

Et en 3 dimensions ?

- C'est pareil
- On introduit une quatrième coordonnée, w
  - Deux vecteurs sont égaux si : $x/w = x'/w'$, $y/w = y'/w'$ et $z/w = z'/w'$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- Toutes les transformations sont des matrices 4x4

Translations en 3D

$$T(t_x,t_y,t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ z' = z + wt_z \\ w' = w \end{cases}$$

Changement d'échelle en 3D

$$S(s_x,s_y,s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} x' = s_x x \\ y' = s_y y \\ z' = s_z z \\ w' = w \end{cases}$$

Rotations en 3D

- Rotation : un axe et un angle
- La matrice dépend de l'axe et de l'angle
- Expression directe possible, en partant de l'axe et de l'angle, et quelques produits vectoriels
  - Passage par les quaternions
- Les rotations autour d'un axe de coordonnées ont une expression simple
  - Les autres rotations s'expriment comme combinaison de ces rotations simples

Rotation around ***x***-axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x-axis is unmodified

Sanity check: a rotation of $\pi/2$ should change y in z, and z in $-y$

$$R_x(\tfrac{\pi}{2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation around ***y***-axis

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

y-axis is unmodified

Sanity check: a rotation of $\pi/2$ should change *z* in *x*, and *x* in *-z*

$$R_y(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

iMAGIS

Rotation about ***z***-axis

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

z-axis is unmodified

Sanity check: a rotation of $\pi/2$ should change *x* in *y*, and *y* in *-x*

$$R_z(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

iMAGIS

Toutes les transformations 3D

- Toute transformation 3D s'exprime comme combinaison de translations, rotations, changement d'échelle
  - Et donc comme une matrice en coordonnées homogènes

iMAGIS

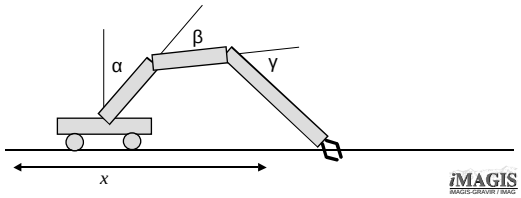
Warning !

- La multiplication de matrice n'est pas commutative
- L'ordre des transformations est important
  - Rotation puis translation aura un effet très différent de translation puis rotation
  - L'ordre des angles d'Euler est un grand classique
  - Une source de bugs **très** courante

iMAGIS

Definition d'objets complexes

Notre problème :



iMAGIS

Definir un objet complexe

- L'objet est défini comme une combinaison d'objets plus petites
  - Exemples : robots, voiture, roue…
- On veut un comportement normal :
  - L'objet reste connecté : si je bouge le bras, la main suit
  - Utiliser les paramètres naturels : *x*,α,β,γ

iMAGIS

Comment faire ?

- Coordonnées relatives
  - La position de la roue par rapport à la voiture
  - La position des boulons par rapport à la roue
- Personne n'utilise des coordonnées absolues dans la vie

Coordonnées relatives

- Utiliser les coordonnées relatives :
  - La position de l'avant-bras est donné en fonction du bras
- Mais il faut bien revenir aux coordonnées absolues :
  - On utilise une concaténation des transformations :
    - Translation sur la position du bras
    - Dessiner le bras
    - Translation sur la position de l' avant-bras par rapport au bras
    - Rotation
    - Dessiner l'avant-bras
  - Je veux revenir aux coordonnées du bras, ou de l'épaule
    - Que faire ?

Pile de transformations

- Garder l'information sur les transformations successives
  - initially = **M**, from model to viewport
  - **M'=MT** (translation by x)
  - draw robot body
  - **M"=M'T1** (translation to center of 1st wheel)
  - draw first wheel as circle of center (0,0)
  - return to **M'**
  - **M'''=M'T2** (translation to center of 2nd wheel)
  - draw second wheel

Pile de transformations

- OpenGL:
  - `popmatrix(), pushmatrix()`
- SPHIGS:
  - `openStructure(), closeStructure()`
- Postscript:
  - `gsave, grestore`

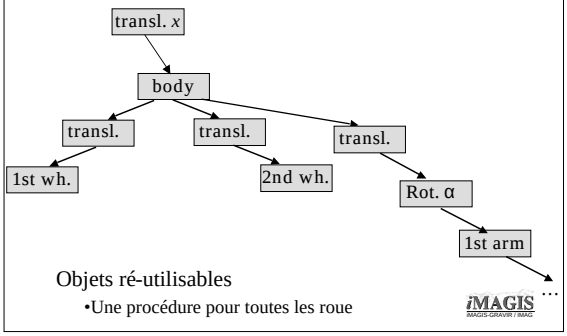
Exemple d'implémentation

- Set transformation as projection matrix
- translate by x (concatenate translation matrix with transformation matrix)
- draw car body
- save transformation matrix
  - translate+rotate
  - draw first wheel
- restore transformation matrix
- save transformation matrix
  - translate+rotate
  - draw second wheel
- restore transformation matrix

Définition hiérarchique

- Comment savoir quelle est la bonne transformation ?
- Comment savoir qu'il est temps de revenir à la transformation précédente ?
- Définition hiérarchique des objets
- Dessiner l'objet = traversée de la hiérarchie

Objet défini hiérarchiquement



Objets ré-utilisables

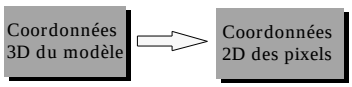
- Une procédure pour toutes les roue

Plan

- Transformations géométriques
- Projections, caméra, perspective
- Élimination des parties cachées

Projections et perspectives

- Affichage d'un modèle 3D
- Fenêtre 2D
- Transformation des coordonnées 3D du modèle vers les coordonnées 2D des pixels



Projections

- Représentation d'une scène 3D, pour un observateur virtuel
- Un point de vue :
  - position de l'observateur
- Une direction de visée :
  - direction vers laquelle est tourné l'observateur
- Une direction « en haut » :
  - la verticale pour l'observateur

Différent types de projections

- Projections parallèles :
  - Pas de rétrécissement des objets dans le lointain
  - Plusieurs types de projections parallèles :
    - isométrique, cavalière,…
- Projection perspective :
  - Les objets lointains sont plus petits
  - Plusieurs types :
    - Un, deux ou trois points de fuite

Projection parallèle

- Projection parallèle sur le plan $z=0$:
  - $x'=x, y'=y, w'=w$
- Matrice de projection :

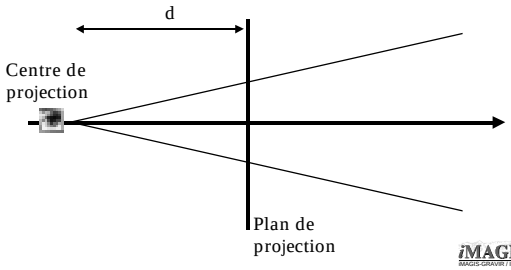
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection perspective

- Projection sur le plan $z=0$, avec le centre de projection placé à $z=-d$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

Distance focale

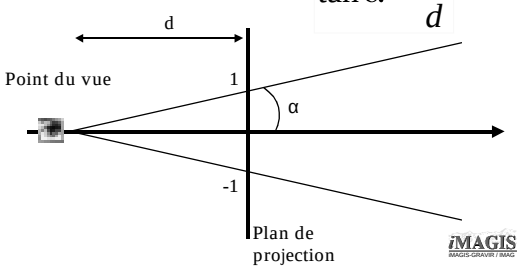


Ouverture de vue

- Le distance focale n'est pas un paramètre intuitif
- Ouverture de vue : plus simple
  - Angle
  - Exprime la largeur du champ visuel

Ouverture

$$\tan\alpha = \frac{1}{d}$$



Coordonnées homogènes

- Essentielles pour la perspective
- La rétrécissement des objets utilise w

$$w' = \frac{z}{d} + w$$

$$\frac{x'}{w'} = \frac{x}{\frac{z}{d} + w}$$

- Impossible sans coordonnées homogènes

Autres positions de la caméra

- Position de la caméra ultra-simple
  - Plan image sur $z=0$, vision suivant z
- Comment passer à un modèle général ?
  - Point de vue quelconque, direction quelconque…
- Translations, rotations
  - On est ramené au cas précédent

Plan

- Transformations géométriques
- Projections, caméra, perspective
- Élimination des parties cachées

Élimination des parties cachées

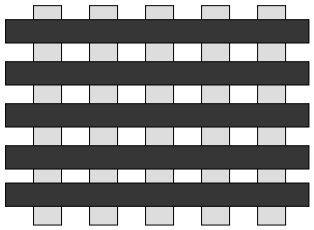
- Espace image contre espace objet
- Algorithmes « objet »:
  - Backface culling
  - Tri par la profondeur
  - BSP-trees
- Algorithmes « image » :
  - Area-subdivision
  - Scan-line algorithm
  - Z-buffer
  - Coûts comparés, choix optimal

Élimination des parties cachées

- Détermination des surfaces visibles
- Complexité : équivalente au tri
  - $O(n \log n)$
- Résolution dans l'espace image ou dans l'espace objet
  - Espace image : précision limitée, $O(np)$ (p = nombre de pixels, 10^6)
  - Espace objet : précision infinie, $O(n^2)$

Résolution dans l'espace objet

- Parfois nécessaire, mais complexité la pire :

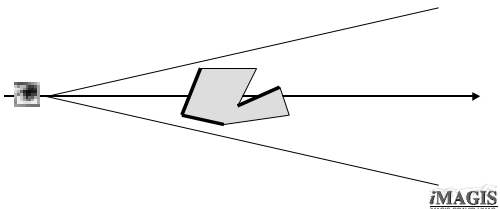


Résolution dans l'espace image

- Permet d'utiliser la cohérence:
  - L'objet visible à un pixel sera sans doute visible sur les pixels voisins
- Plus rapide en moyenne
- Cas le pire : $O(np)$

Back Face Culling

- Éliminer tous les polygones qui ne sont pas tournés vers la caméra :



Back Face Culling : algorithme

- Si le point de vue n'est pas devant le polygone, on n'affiche pas le polygone
- Produit scalaire :
  - (Sommet-PointDeVue)*normale
  - > 0 : on garde le polygone
  - < 0 : on l'élimine

Back Face Culling

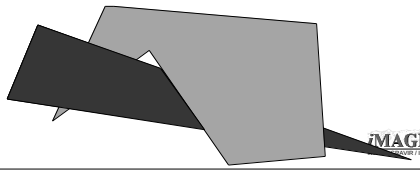
- Économise 50 % du temps de calcul
  - En moyenne
- Faible coût par polygone
- Étape préliminaire pour les autres algorithmes
- Suffisant pour un seul objet convexe

Tri par la profondeur

- Dit aussi « algorithme du peintre »
- On dessine d'abord les polygones les plus lointains, puis les polygones proches de l'œil
- Les polygones plus proches cachent les polygones lointains
- Comme un peintre dessine d'abord l'horizon, puis l'arrière-plan, puis le premier plan
- Problème : définir un ordre sur les polygones
  - Complet, non ambigu

Algorithme simplifié

- Trier les polygones en fonction de la distance à l'œil
- Ordre incomplet
- Ambiguïtés à résoudre :



Algorithme complet

- Trier les polygones en fonction de leur plus grande coordonnée en *z*
  - *z* distance à la caméra suivant la direction de visée
- Si deux polygones ont des étendues en *z* qui se recouvrent :
  - On teste :
    - Si les boites englobantes de leurs projections sont séparées
    - Si l'un est complètement derrière l'autre
    - Si leurs projections sont séparées
  - Si rien ne marche, on coupe l'un des polygones par le plan de l'autre

Pour ou contre

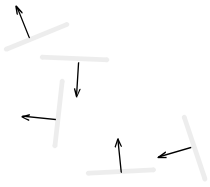
- Le plus intuitif des algorithmes
- Coût en mémoire :
  - Affichage direct à l'écran : O(*p*)
  - Il faut trier les polygones : O(*n*log*n*)
- Temps de calcul :
  - On affiche toute la scène
  - Efficace surtout sur des petites scènes

BSP-Trees

- On construit un BSP-Tree 3D pour toute la scène
  - En subdivisant les polygones qui sont intersectés par le plan du nœud
- Affichage des polygones par un parcous de l'arbre :
  - En premier les polygones qui sont derrière le nœud courant (par rapport à la caméra)
  - Puis le nœud courant
  - Puis, les polygones qui sont devant le nœud courant
- Sorte d'algorithme du peintre
  - Avec un ordre partiel
  - Mais suffisant

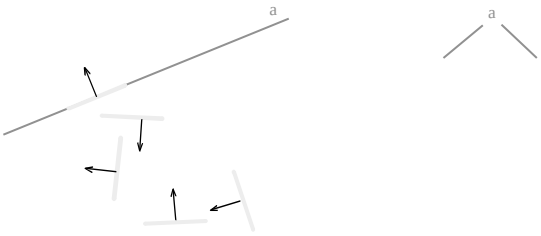
iMAGIS

Construire un BSP-Tree (1)



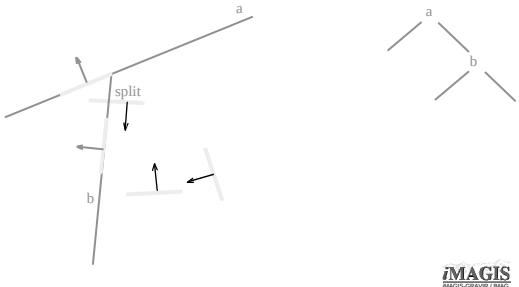
iMAGIS

Construire un BSP-Tree (2)



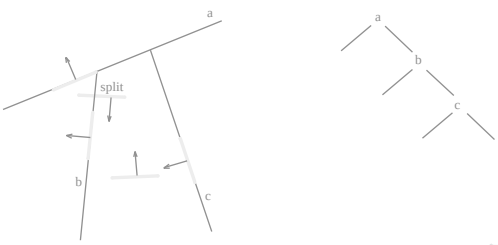
iMAGIS

Construire un BSP-Tree (3)



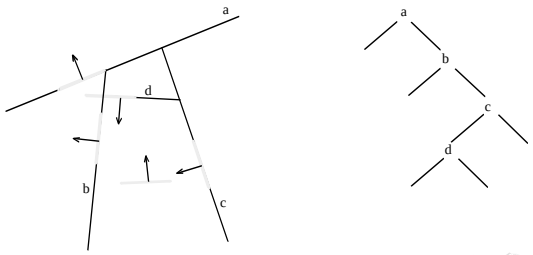
iMAGIS

Construire un BSP-Tree (4)



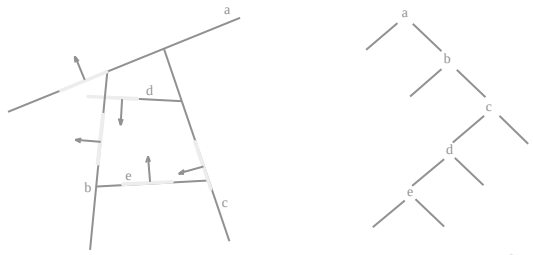
iMAGIS

Construire un BSP-Tree (5)

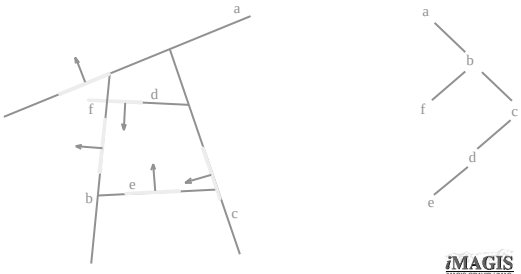


iMAGIS

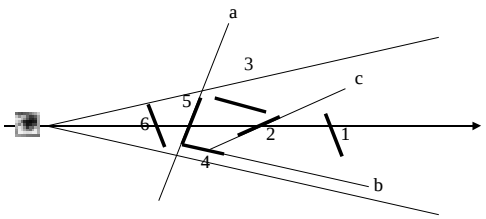
Construire un BSP-Tree (6)



Construire un BSP-Tree (7)



Utiliser un BSP-Tree



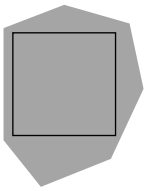
BSP Tree contre algorithme du peintre

- Le BSP-Tree fait plus de divisions de polygones
- Mais l'affichage est plus direct
- BSP-Tree:
  - Pré-traitement plus long
  - Coût mémoire plus élevé
  - Temps par requête (position de la caméra) plus petit
- Algorithme du peintre:
  - Pas de pré-traitement
  - Temps par requête plus long
- Dans les deux cas : on affiche toute la scène

Area Subdivision (Warnock)

- On utilise la cohérence spatiale
- On divise l'écran en zones de travail
- Pour chaque zone, on ne considère que les polygones qui l'intersectent
- Si la visibilité est connue, on s'arrête
- Si la visibilité est inconnue, on subdivise
- On interrompt la subdivision quand on atteint une taille limite

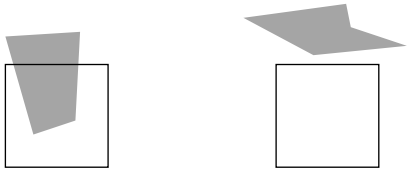
Polygones/zone de travail



Englobe

Inclus

Polygones/zone de travail



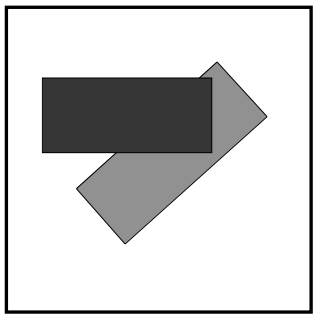
Intersecte

Disjoint

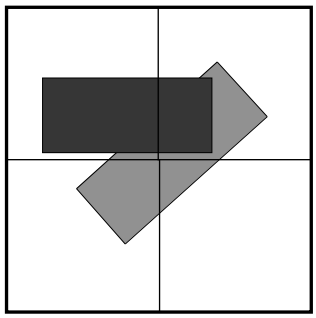
Quand est-ce que la visibilité est connue ?

- Tous les polygones sont *disjoints*
- Un seul polygone *intersecte* ou *inclus*
- Un polygone *englobant* qui est devant les autres

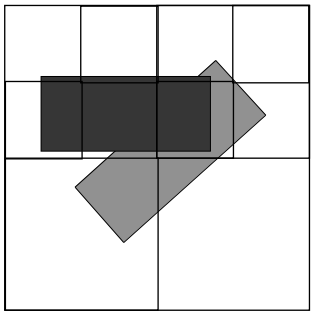
Warnock : exemple (1)



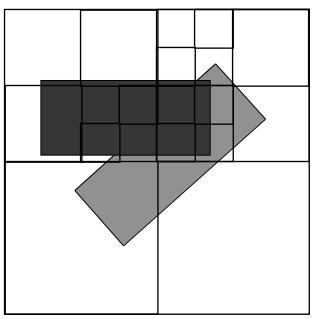
Warnock : exemple (2)



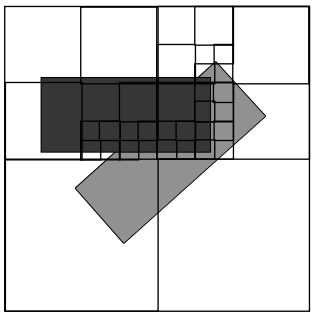
Warnock : exemple (3)



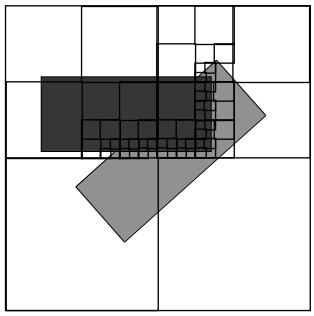
Warnock : exemple (4)



Warnock : exemple (5)



Warnock : exemple (6)



Warnock : pour ou contre

- Utilise la cohérence spatiale
- Plus efficace avec des grands polygones
- Coût mémoire parfois élevé
- Implémentation facile : appels récursifs à la même fonction

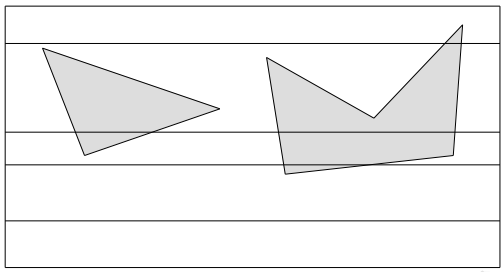
Algorithme Scan Line

- Travailler *scanline* par *scanline*
- Pour chaque *scanline*, on trouve quel polygone est devant
- On affiche la *scanline*

Algorithme Scan Line

- Tri des arrêtes des polygones :
  - Après projection sur le plan image
  - En *buckets*, triés par la coordonnée y la plus petite
  - Au sein du *bucket*, triés par la pente
- On suit la scanline :
  - Si on rencontre une arête, le polygone est *in*
    - Si on rencontre l'arête de sortie du polygone, il cesse d'être *in*
  - Tant qu'il n'y qu'un seul polygone *in* : pas de problèmes

Algorithme Scan Line

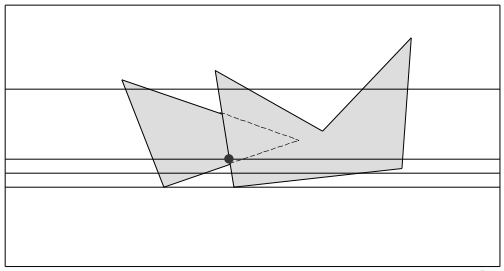


Algorithme Scan Line

- Si deux polygones sont *in* simultanément :
  - Quand on passe l'arête de début du deuxième polygone, on cherche lequel est devant
  - On n'affiche que celui qui est devant

iMAGIS

Algorithme Scan Line



iMAGIS

Scan line: pour ou contre

- Faible coût mémoire
- Utilise la cohérence spatiale
  - Dans une dimension seulement
- Quelques avantages supplémentaires :
  - Remplissage des polygones
  - Transparence
  - Ombrage (Gouraud, Phong)
  - Plaquage de textures
- Renderman (Pixar) = scan line uniquement

iMAGIS

Z-Buffer

- Un tableau, de la taille de l'écran
- On stocke la valeur maximale de z pour chaque pixel
  - z est la direction de visée, exprime la distance à l'oeil
- Initialisation : tous les pixels à moins l'infini
- Projection de tous les polygones
  - On met à jour les pixels de la projection du polygone

iMAGIS

Z-buffer : algorithme

- Pour chaque polygone :
  - Projeter le polygone sur le plan image
  - Pour chaque pixel dans la projection du polygone
    - Calculer la valeur de z pour ce pixel
    - Si z est supérieur à la valeur courant de z max
      - Changer z maximal
      - Afficher le pixel à l'écran, de la couleur du polygone

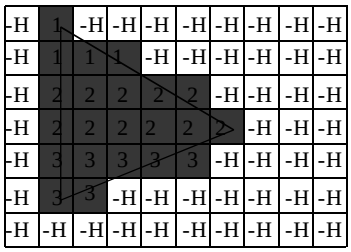
iMAGIS

Z-buffer (1)

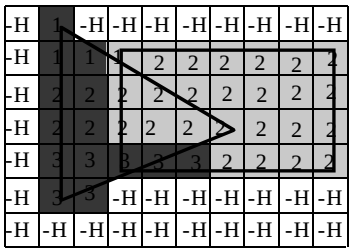
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H
-H	-H	-H	-H	-H	-H	-H	-H	-H	-H

iMAGIS

Z-buffer (2)



Z-buffer (3)



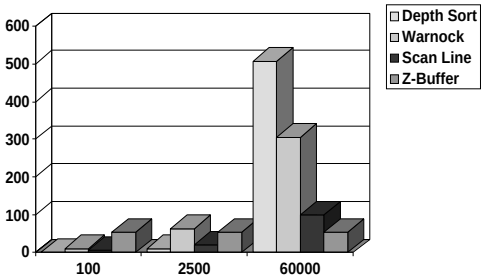
Z-Buffer : pour ou contre

- Pour :
  - Facile à implémenter
  - Travaille dans l'espace image
    - Rapide
- Contre :
  - Coût en mémoire
  - Travaille dans l'espace image
    - aliasing
    - artefacts

Z-Buffer

- Combien d'information ?
  - Combien de bits pour z max ?
  - Limité par la mémoire :
    - 8 bits, 1024x1280: 1.25 Mb
    - 16 bits, 1024x1280: 2.5 Mb
  - Nécessaire pour la séparation des objets proches :
    - 8 bits, distance minimale entre objets de 0.4 % (4mm pour 1m)
    - 16 bits, distance minimale de 0.001 % (1mm pour 1km)
    - Que se passe t-il en dessous de cette limite ?

Coûts comparés



Choix de l'algorithme ?

- L'algorithme idéal dépends :
  - De la complexité de la scène
  - Du matériel disponible
  - De ce qu'on veut faire en plus de l'affichage
- Z-Buffer en général :
  - Fourni gratuitement dans la librairie graphique
  - Pas de pré-traitement
  - Quelques effets de bord si on le connaît mal
- Scan-line pour les hackers :
  - Quand on ne peut pas utiliser la carte graphique
  - Faible coût mémoire
  - Lié à la fonction d'affichage, peut être très efficace