

GMIN 317 – Moteur de Jeux

Gestion de scènes

Université Montpellier 2

Rémi Ronfard

remi.ronfard@inria.fr

<https://team.inria.fr/imagine/remi-ronfard/>

Ce cours est très largement inspiré des cours de ***Marc Moulis et Benoit Lange.***

Le but de cette présentation est de vous amener les briques nécessaire pour résoudre la complexité de certain calculs. Ainsi, nous allons principalement nous intéresser à plusieurs problématiques:

- La gestion de scène au travers d'un graphe de scène
- Les enveloppes
- Le partitionnement de l'espace
- Les optimisations spatiales
- Les optimisations géométriques

A quoi sert un graphe de scène et comment l'utiliser ?

Quels sont les avantages des enveloppes ?

Comment décomposer efficacement la scène ?

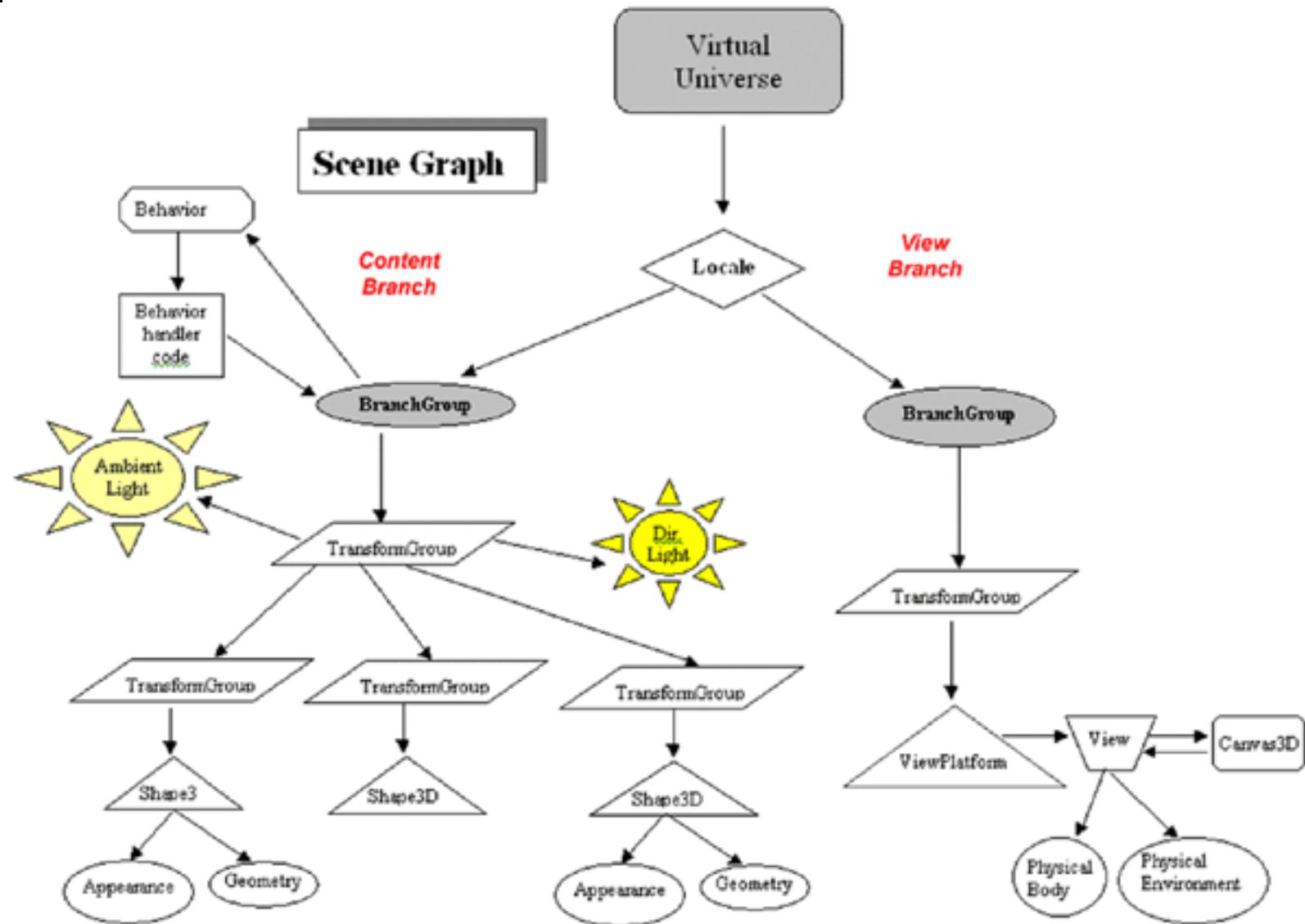
Comment accélérer les calculs ?

Le graphe de scène

- Il s'agit de l'outil le plus important pour gérer votre scène 3D
- Il gère tout ce qui est *mis* dans votre scène.

- Il est basé sur une arborescence
- Il n'est pas implémenté dans OpenGL (API de rendu)
- Vous devez donc construire votre propre graphe de scène ...

- Un graphe de scène est un arbre !
 - Une seule racine
 - Tous les fils héritent des propriétés spatiales de leur père
 - Lorsque une matrice associée change, tous les fils sont impactés
 - Il n'est pas limité en nombre de feuilles
(ex à opposer à un arbre binaire)
 - Il contient toute votre scène et pas seulement la géométrie
 - Camera
 - Transformations
 - ...
- Implémenter une structure d'arbre.
 - Composé de nœuds, père, fils, ...
- Le principe est d'architecturer sa pensée
- Réduire les accès aux éléments



Les données géométriques représentant les objets de la scène peuvent être très complexes, il faut donc leur associer une représentation simplifiée pour maximiser les performances lors :

- des tests de visibilité
- des tests de collision

Pour résoudre ce problème, on utilise en général une approche basée sur un volume englobant convexe de la géométrie, permettant d'accélérer les traitements en réalisant certaines approximations.

Dans l'idéal, un volume englobant doit:

- permettre des tests de collisions très rapides
- approximer au mieux le volume réel de l'objet
- se calculer très rapidement
- pouvoir être transformé rapidement (ex: rotation)
- avoir une empreinte mémoire réduite

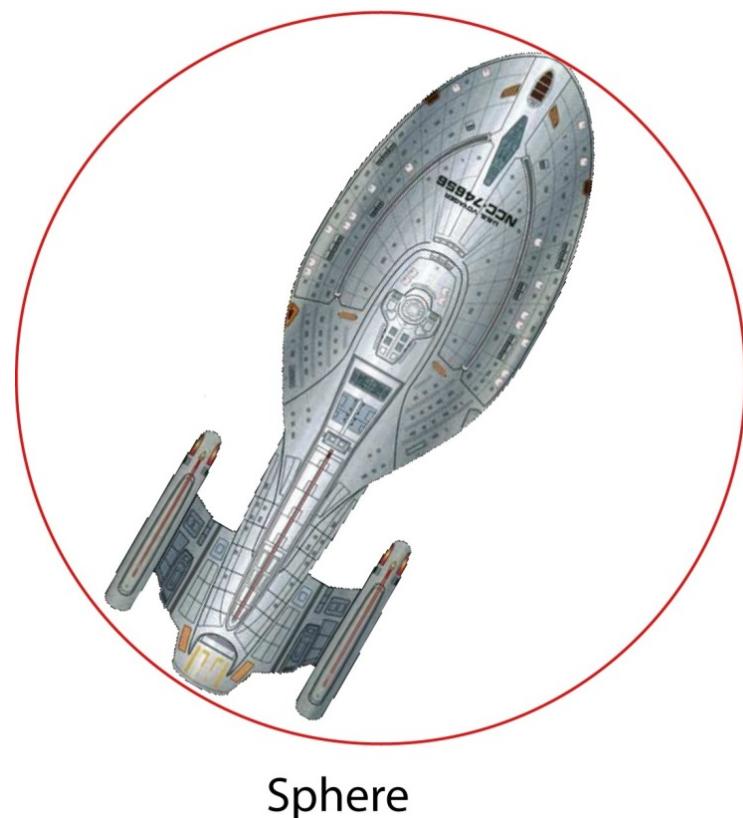
En général, les caractéristiques ci-dessus sont opposées les unes aux autres

- Les volumes englobant sont en général exprimés dans le repère local de l'objet. Afin de pouvoir réaliser les tests de collision, les volumes doivent cependant être transformés dans un repère commun.
- Il est possible d'utiliser l'espace global pour cela, mais c'est en général plus rapide (1 seule transformation) et plus précis d'utiliser le repère local d'un des deux objets à tester.

Sphère englobante

- Avantages
 - Empreinte mémoire réduite (origine, rayon)
 - Tests d'intersection rapides (temps constant)
 - Mises à jour rapides
- Inconvénients
 - Approximation souvent faible du véritable volume de l'objet

Deux sphères s'intersectent seulement si la distance entre les centres est inférieure à la somme des rayons.

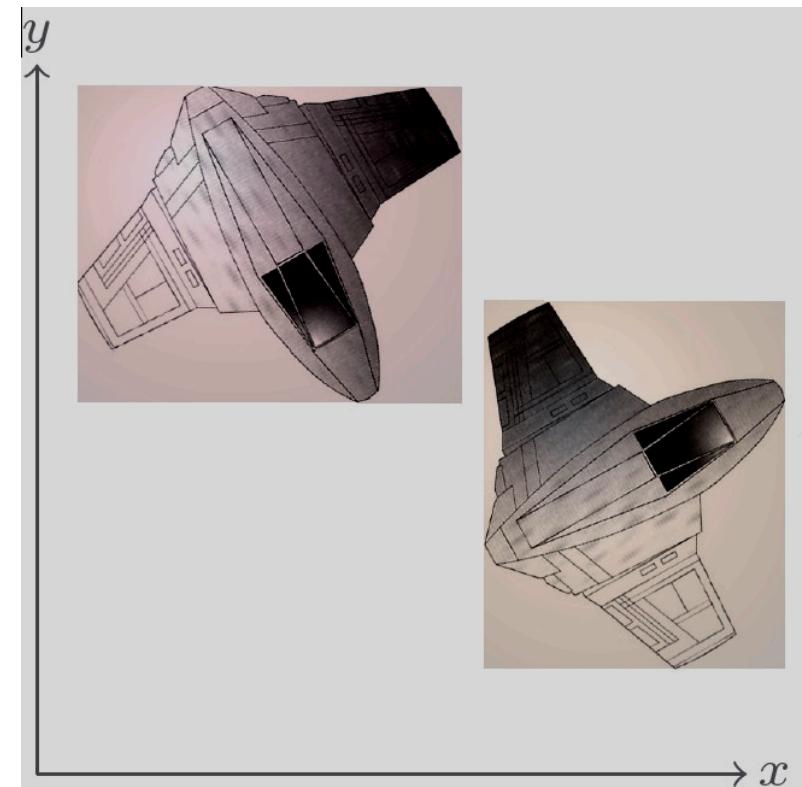


Boîte AABB

L'Axis-Aligned Bounding Box (AABB) est une boîte à 6 faces (en 3D), dont les normales des faces sont alignées avec les axes du système de coordonnées.

- Avantages
 - Empreinte mémoire réduite
 - Tests d'intersection rapides (temps constant)
 - Mises à jour relativement rapides (temps constant)
- Inconvénients
 - Approximation faible du véritable volume de l'objet

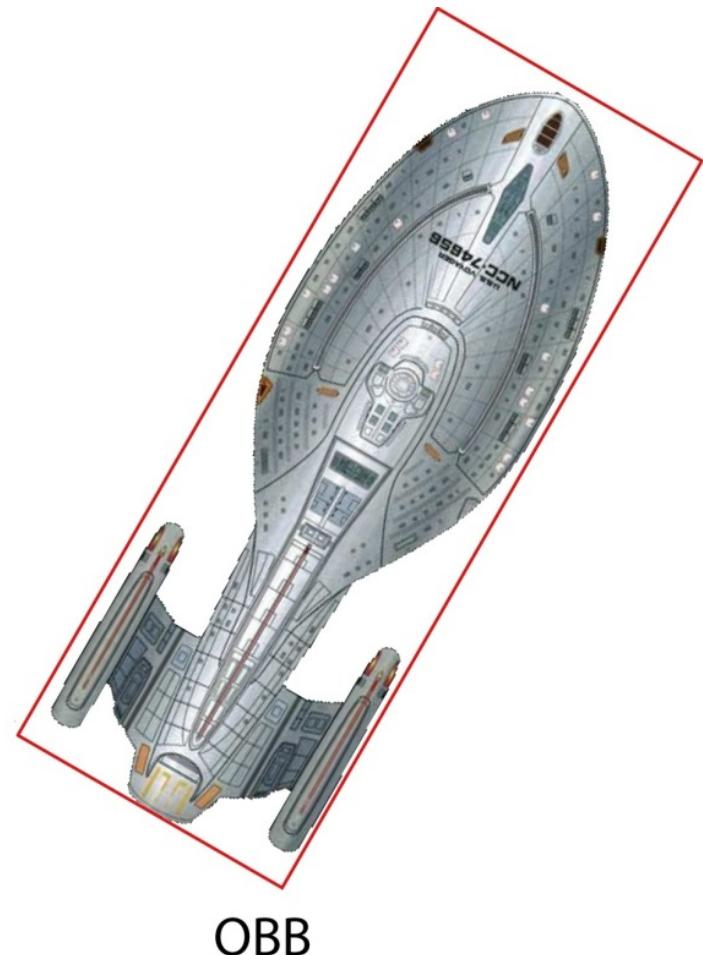
Deux AABB s'intersectent seulement si les boîtes ont un recouvrement sur tous les axes du système de coordonnées.



Boite OBB

L'Object-Oriented Bounding Box (OOBB) est une boîte à 6 faces (en 3D), dont les normales des faces sont alignées avec les axes du système de coordonnées local à l'objet.

- Avantages
 - Empreinte mémoire réduite
 - Mises à jour relativement rapides
 - Approximation améliorée du véritable volume de l'objet
- Inconvénients
 - Tests d'intersection de 2 OOB plus complexes

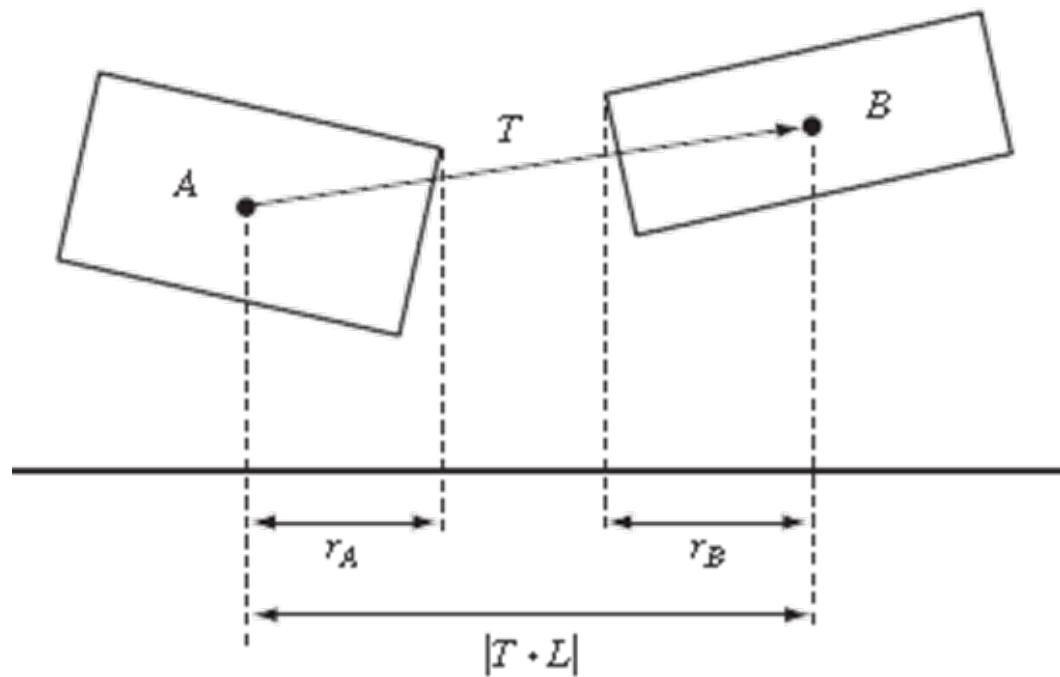


Intersections

- Pour tester l'intersection de deux OOOB, on utilise généralement un test d'*axe séparateur* :
- Les OOOB ne s'intersectent pas s'il existe un axe sur lequel la somme de leurs rayons projetés sur cet axe est inférieure à la distance des centres des OOOB projetés sur cet axe.

Au plus, 15 axes nécessitent d'être testés:

- Les 3 axes de chacune des OOOB (ax, ay, az, bx, by, bz)
- Les 9 axes perpendiculaires à chaque axe ($ax \times bx, ax \times by, ax \times bz, ay \times bx, ay \times by, ay \times bz, az \times bx, az \times by, az \times bz$)

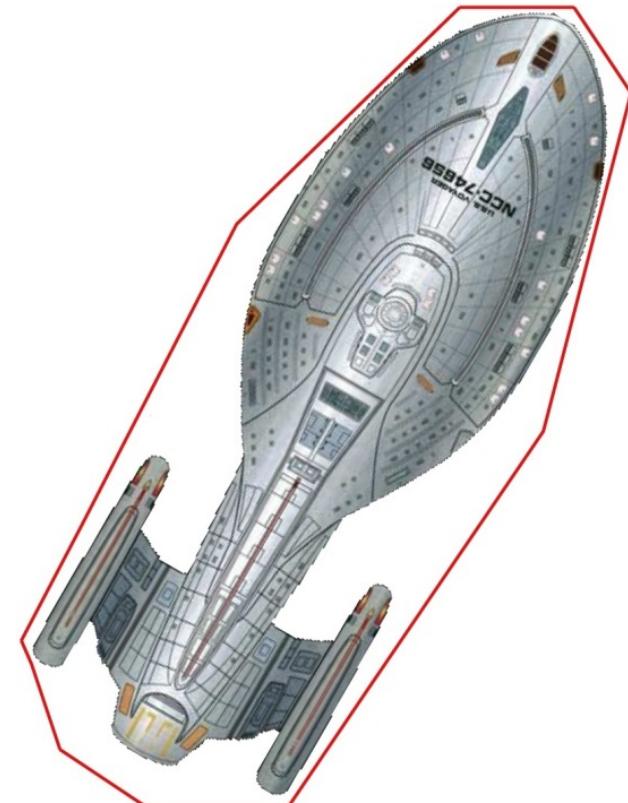


Discrete Oriented Polytopes (K-DOP)

Un k-DOP est un polytope convexe, pour lequel les normales appartiennent à un (petit) ensemble de k axes, partagés entre tous les volumes k-DOPs.

- Avantages
 - Empreinte mémoire relativement réduite
 - Tests d'intersection relativement rapides
 - Bonne approximation du véritable volume de l'objet

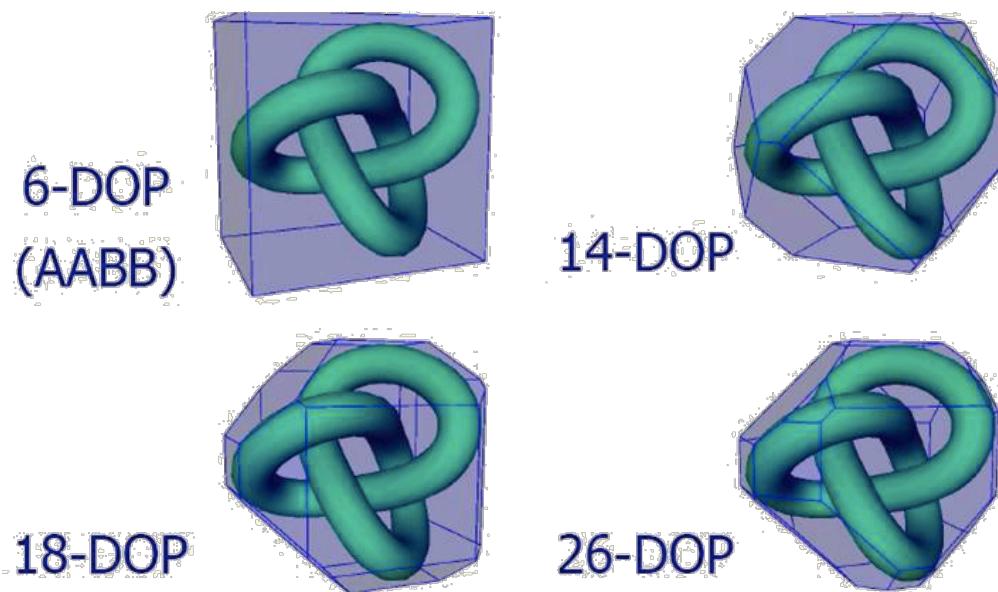
- Inconvénients
 - Mises à jour moyennement rapides



kDOP

Ensemble des normales pour différents k-DOPs:

- 6-DOP : $\{1, 0, 0\}, \{-1, 0, 0\}, \{0, 1, 0\}, \{0, -1, 0\}, \{0, 0, 1\}, \{0, 0, -1\}$
- 8-DOP : $\{1, 1, 1\}, \{-1, 1, 1\}, \{1, -1, 1\}, \{-1, -1, 1\}, \{1, 1, -1\}, \{-1, 1, -1\}, \{1, -1, -1\}, \{-1, -1, -1\}$
- 12-DOP : $\{1, 1, 0\}, \{-1, 1, 0\}, \{1, -1, 0\}, \{-1, -1, 0\}, \{1, 0, 1\}, \{-1, 0, 1\}, \{1, 0, -1\}, \{-1, 0, -1\}, \{0, 1, 1\}, \{0, -1, 1\}, \{0, 1, -1\}, \{0, -1, -1\}$
- 18-DOP : 12-DOP U 6 DOP



Construction d'un k-DOP ($O(kn)$, quasi-linéaire). Seuls les intervalles min/max sur chaque direction D de k sont stockés

Example for 12-DOP generation:

```
{1, 1, 0}, {-1, 1, 0}, {1, -1, 0}, {-1, -1, 0}, {1, 0, 1}, {-1, 0, 1}, {1, 0, -1}, {-1, 0, -1}, {0, 1, 1}, {0, -1, 1}, {0, 1, -1}, {0, -1, -1}
```

```
// Loop over model points table V, update object's DOP Axis
```

```
for (int i=0; i<V.Size(); i++)
{
    // First axis (1, 1, 0)
    fValue = V[i].x + V[i].y;
    Axis[0].min = MIN(Axis[0].min, fValue);
    Axis[0].max = MAX(Axis[0].max, fValue);

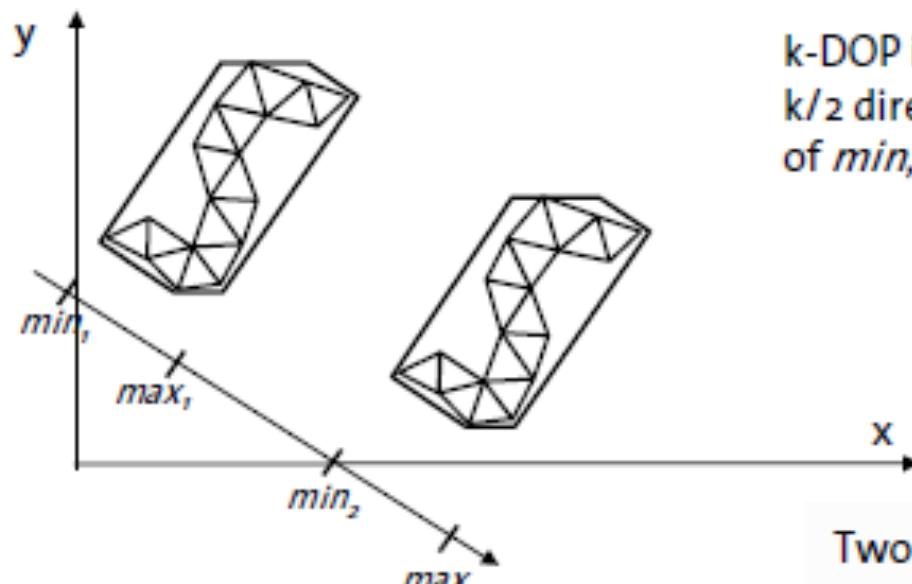
    ...
    // Tenth axis (0, -1, 1)
    fValue = -V[i].y + V[i].z;
    Axis[9].min = MIN(Axis[9].min, fValue);
    Axis[9].max = MAX(Axis[9].max, fValue);

    ...
}
```

Intersections

- Intersection de 2 k-DOPs : il y a superposition seulement si tous les $k/2$ intervalles se superposent ($O(k/2)$, quasi-constant)
- Inconvénient : la mise à jour du k-DOP après rotation de l'objet peut être coûteuse

A k-DOP is “a convex polytope whose facets are determined by halfspaces whose outward normals come from a small **fixed** set of k orientations.” [Klosowski]



Examples:

6-, 14-, 18-, 26-DOPs

k-DOP is represented by $k/2$ directions and $k/2$ pairs of *min, max* values.

Two k-DOPs do not overlap, if at least the intervals in one direction do not overlap.

Enveloppe convexe

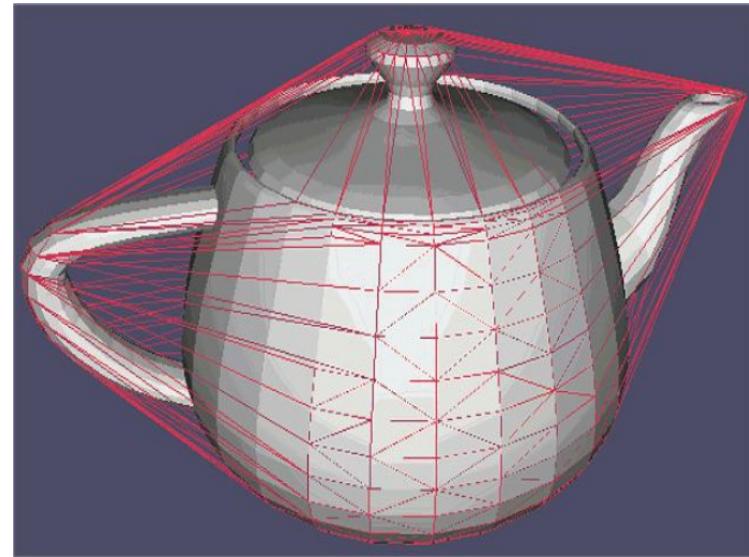
L'enveloppe convexe d'un objet est le plus petit polyèdre convexe qui contient cet objet.

Avantages

- Très bonne approximation du véritable volume de l'objet

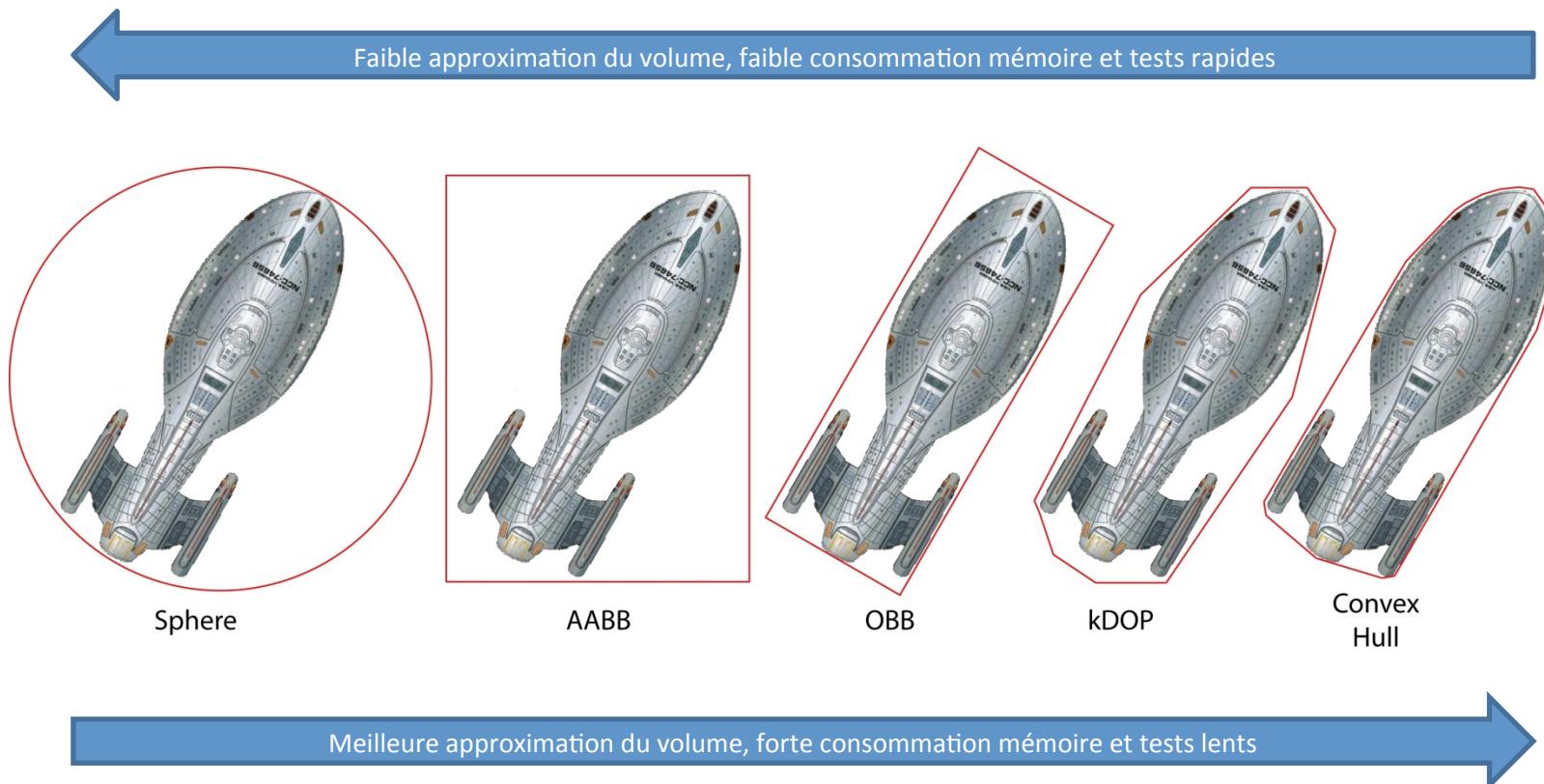
Inconvénients

- Empreinte mémoire importante
- Tests d'intersection lents
- Mises à jour en général non-dynamiques (l'enveloppe est précalculée)

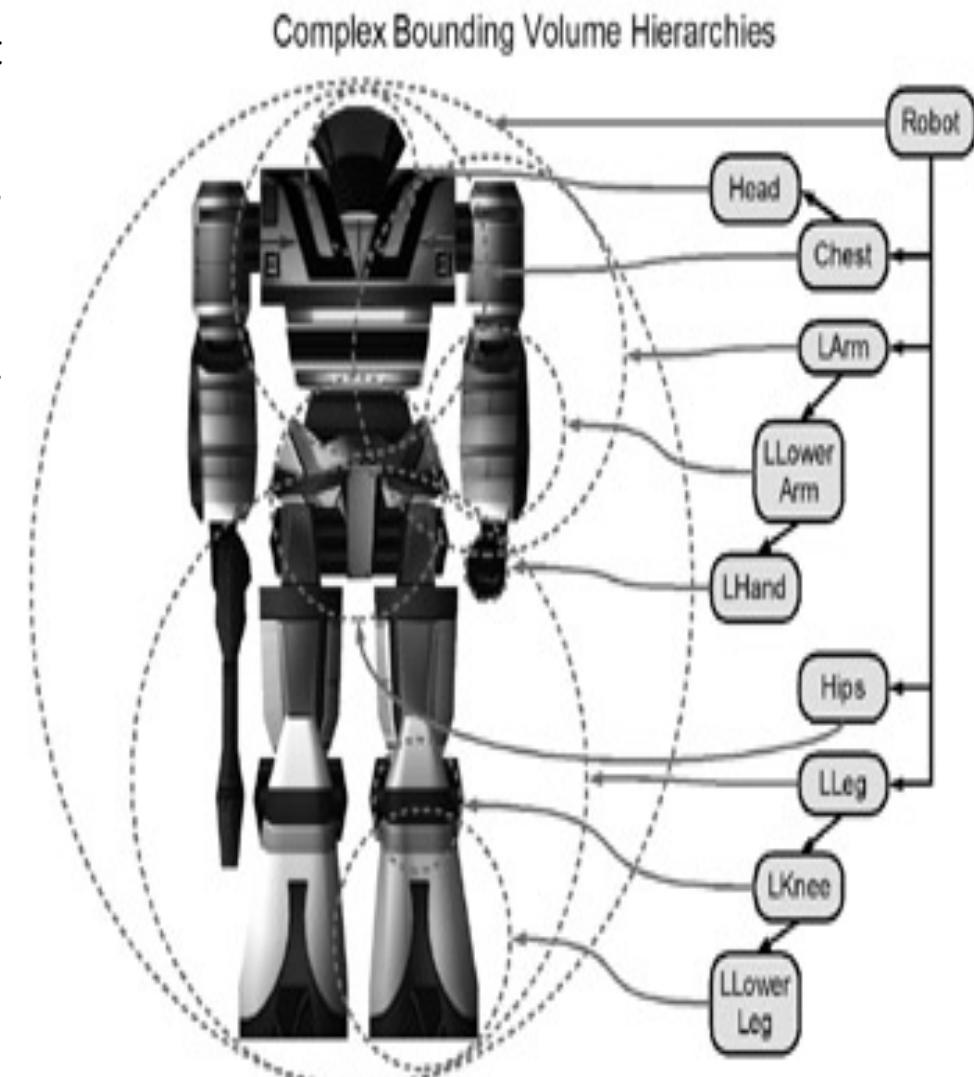
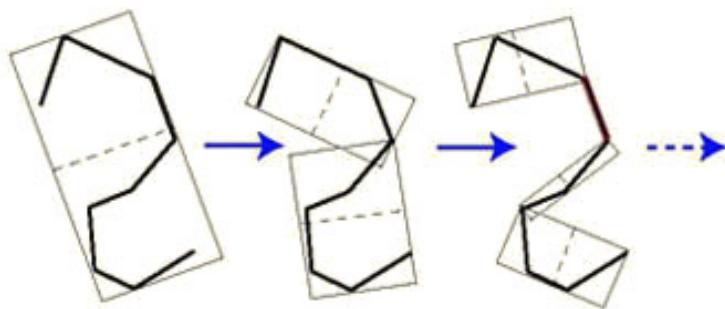


Stratégies de tests d'intersection : soient P et Q deux polyèdres convexes

- Intersection de Moore ($O(n^2)$)
 - Tester l'intersection de chaque segment de face de P avec Q. Si un segment est partiellement ou totalement inclus, renvoyer l'intersection.
 - Tester l'intersection de chaque segment de face de Q avec P. Si un segment est partiellement ou totalement inclus, renvoyer l'intersection.
 - Tester le centre de chacune des faces de Q relativement aux faces de P. Si un centre est contenu dans une face, renvoyer l'intersection.
- Intersection de Chung-Wang ($O(kn)$)
 - Utilisation d'axes de séparation pour tester l'intersection



- Le concept de volume englobant peut être poussé plus loin en mettant en place pour un objet non plus un, mais une hiérarchie de volumes englobant.
- La racine de la hiérarchie fournit un test rapide mais très approximatif, et chaque niveau de hiérarchie successif permet de raffiner les tests.
- Le concept est applicable à tous les types de volumes englobant, voire permet même de mixer différents types de volumes englobant au sein



Partitionner l'espace, dans quel but ?

La machine n'est (pour l'instant) pas capable de:

- réaliser un rendu interactif d'un trop grand nombre de données
- réaliser la gestion des interactions entre un trop grand nombre d'entités en même temps

La stratégie de partitionnement de l'espace s'appuie sur le constat: « *ce qui coûte le moins de cycles machine est ce que l'on ne calcule pas* ».

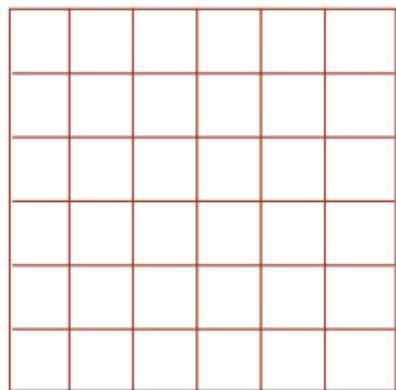
Le partitionnement de l'espace va donc permettre de:

- ne pas afficher les zones de la scène non visibles (décor, et objets contenus dans ces zones)
- limiter le traitement des interactions objets/objets et objets/scène

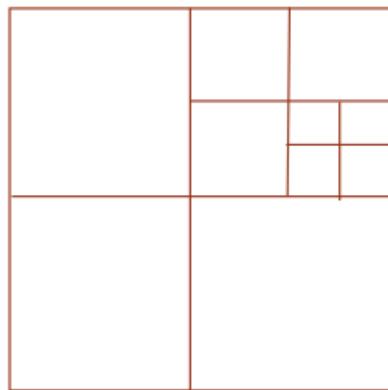
Un partitionnement de l'espace est généralement une structure de données hiérarchique qui assure une propriété monotone: si un test de collision pour un nœud est négatif, alors il sera aussi négatif pour tous les enfants de ce nœud.

Il existe différentes stratégies de découpage de l'espace, dont les plus connues:

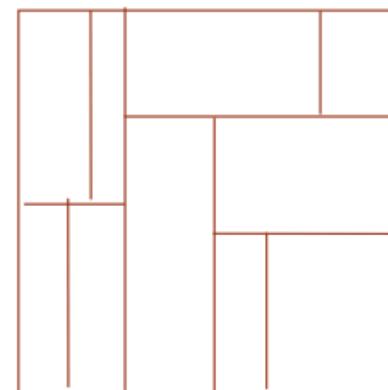
- Quadtrees (2D) et octrees (3D)
- K-d trees
- BSP (Binary Space Partitionning)
- Regular grids, hierarchical uniform grids, recursive grids, loose octrees,...



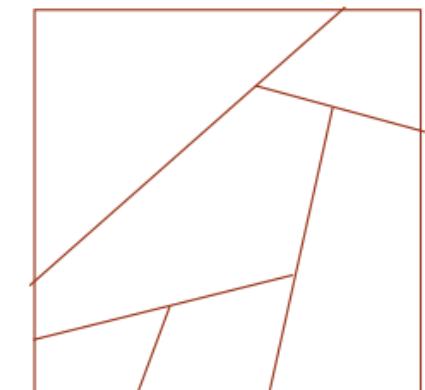
uniform grid



Quadtree/Octree



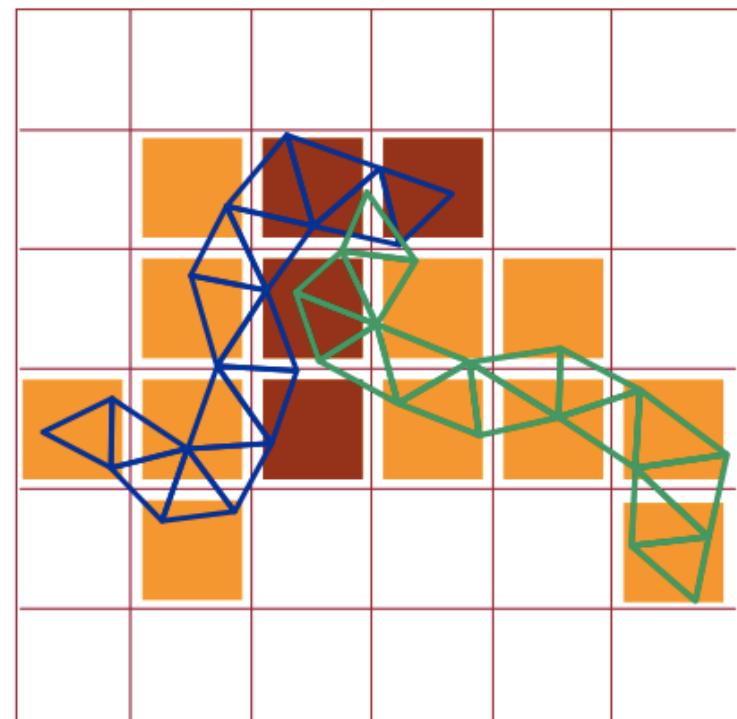
k-d tree



BSP tree

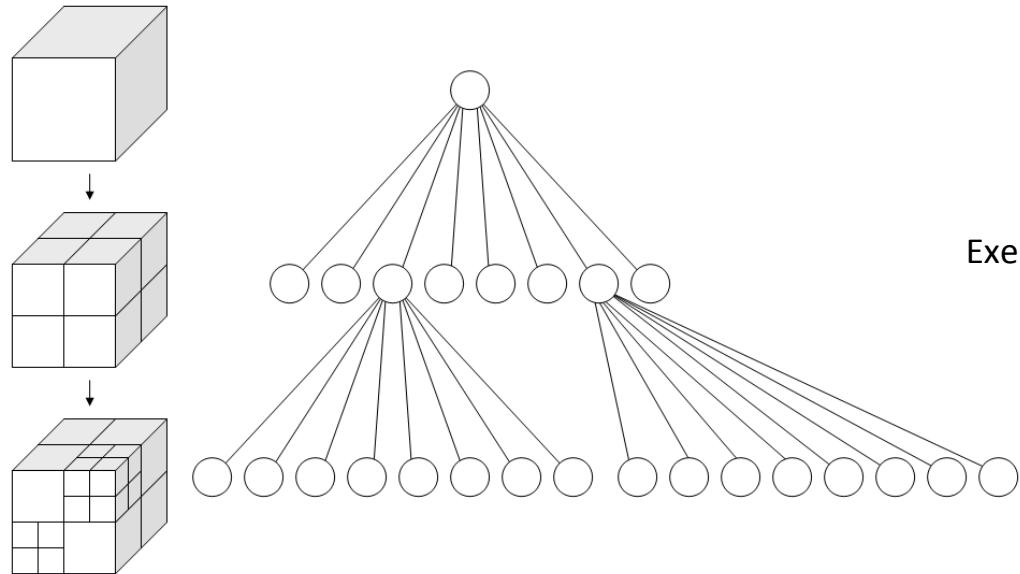
Principe de la subdivision

- L'espace est divisé en cellules
- Les primitives des objets sont placés dans les cellules
- Tous les objets dans les même cellules sont alors analysé (pour déterminer si il y a collision ou non)
- Les parties des objets qui sont seules, sont alors rejetés.



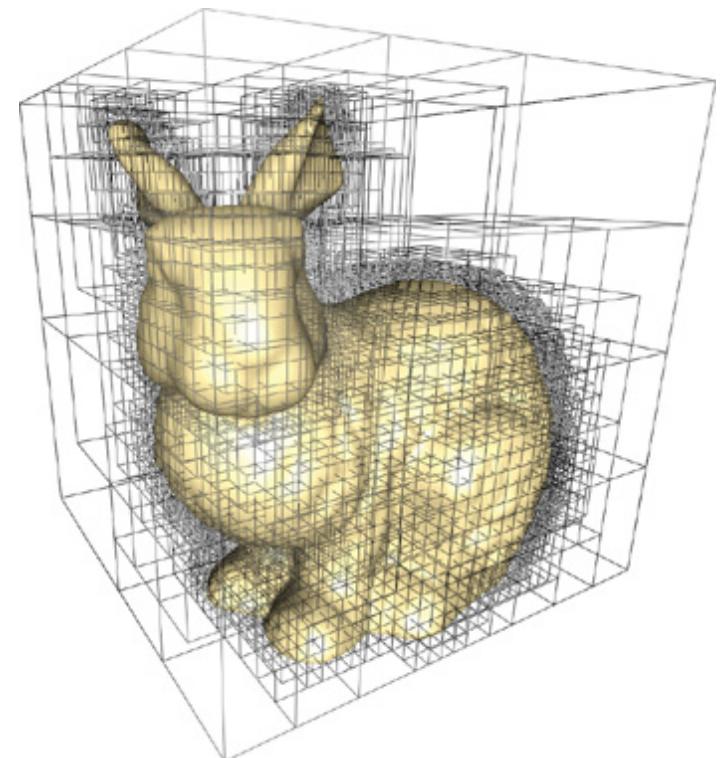
Quadtree et Octree

- Un quadtree (2D) ou octree (3D) est une structure de données hiérarchique de type arbre, qui subdivise l'espace selon des volumes englobants de type AABB.
- Chaque nœud de la hiérarchie est une AABB, qui peut à son tour contenir jusqu'à 4 (ou 8 en 3D) AABB de tailles identiques.
- La construction de l'arbre s'arrête lorsqu'un nœud est vide (ou sous un certain seuil de remplissage), et/ou lorsque la taille des nœuds est en-dessous d'une limite fixée.
- Chaque nœud de l'arbre va contenir, en plus des informations de filiation, des références sur les géométries (ou toutes autres données utiles) contenues dans le volume du nœud.



Génération d'un octree, et visualisation de l'arbre associé.

Exemple d'octree généré à partir d'une géométrie (ici, le lapin de Stanford).



KD-tree

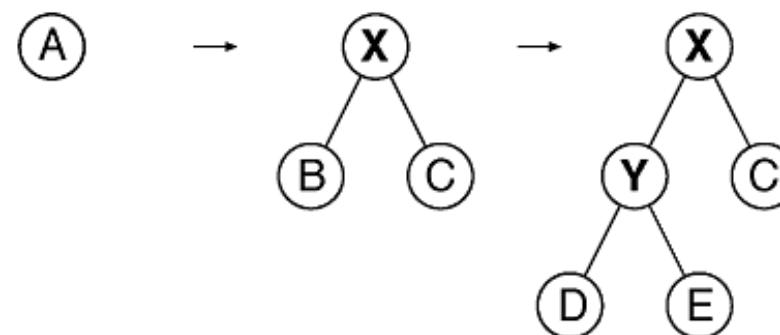
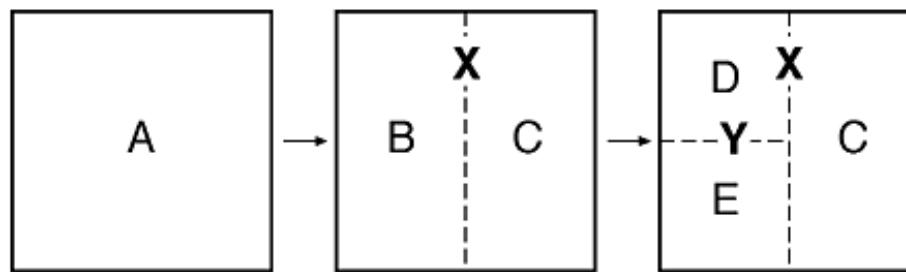
La méthode de sélection de l'hyperplan peut grandement varier, on retiendra cependant les contraintes suivantes pour décrire une méthode de construction canonique:

- La sélection du plan séparateur cycle à travers l'ensemble des axes au fur et à mesure de la descente dans le graphe (racine : dimension X, profondeur 1 : dimension Y, profondeur 2 : dimension Z, profondeur 3 : dimension X, etc)
- La position de l'hyperplan séparateur se base sur le point médian (i.e. qui sépare de manière homogène les données contenues dans le sous-espace) du sous-espace à subdiviser, par rapport à ses coordonnées vis-à-vis de la normale à l'hyperplan

Si ces 2 contraintes sont respectées, la construction permet de générer un arbre équilibré, dans lequel chacune des feuilles est à égale profondeur de la racine.

BSP-tree

- Un arbre BSP (Binary Space Partitioning) est un arbre binaire qui partitionne récursivement l'espace en 2 sous-espaces, par rapport à un hyperplan dont la position et l'orientation sont arbitrairement choisies.



Construction d'un arbre BSP

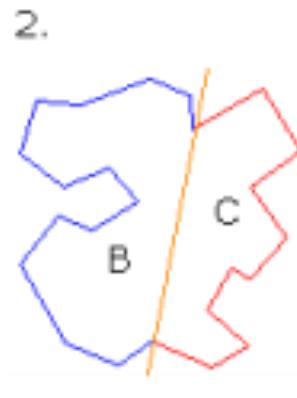
Choisir un hyperplan séparateur, et lui associer un nœud de l'arbre

Placer tous les éléments géométriques situés « devant » l'hyperplan dans un des deux fils du nœud, et tous les éléments géométriques situés « derrière » l'hyperplan dans l'autre nœud fils

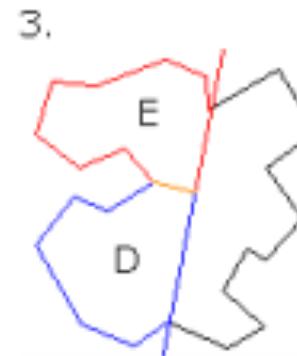
Répéter récursivement le processus de subdivision sur chacun des nœuds fils en choisissant à chaque fois un nouvel hyperplan séparateur



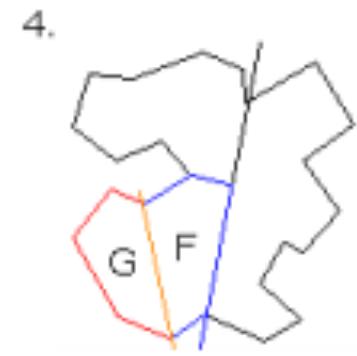
(A)



(B)
(C)



(D)
(E)



(F)
(G)

Optimisation

Objectif : mettre en place des méthodes permettant de n'afficher que ce qui est visible dans la scène à un instant donné.

Différents types de stratégies existent, qui sont en général cumulables:

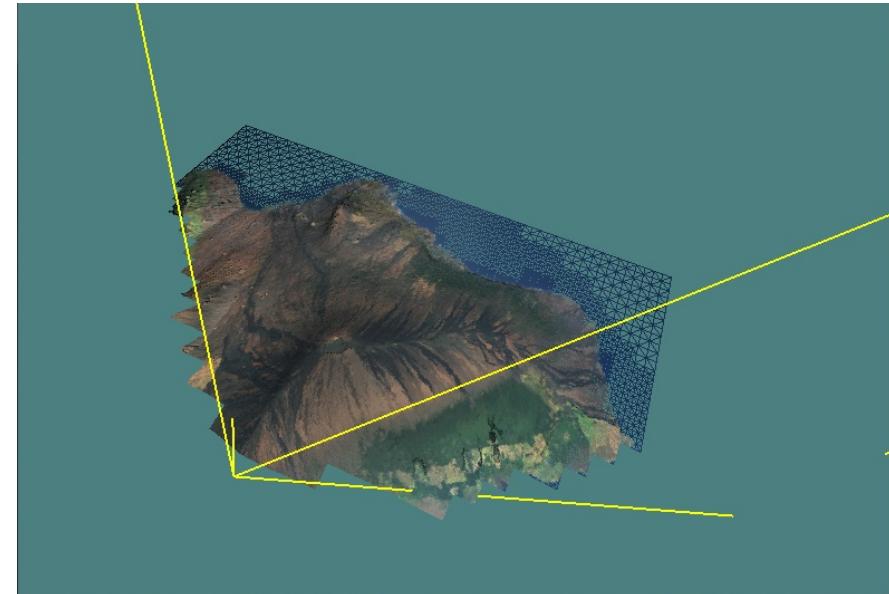
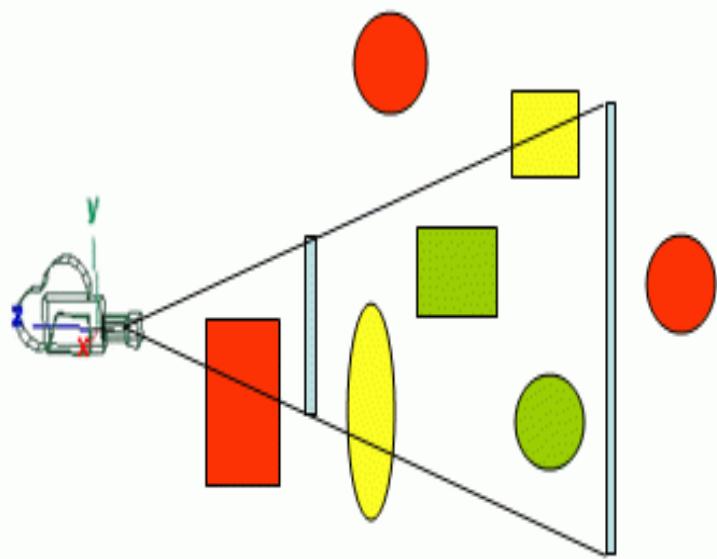
- Ne pas afficher les primitives géométriques dont la normale ne fait pas face à la caméra (backface culling)
- Ne pas afficher ce qui n'est pas dans le champ de vue (geometry ou frustum culling)
- Ne pas afficher ce qui est dans le champ de vue, mais masqué (occlusion culling)

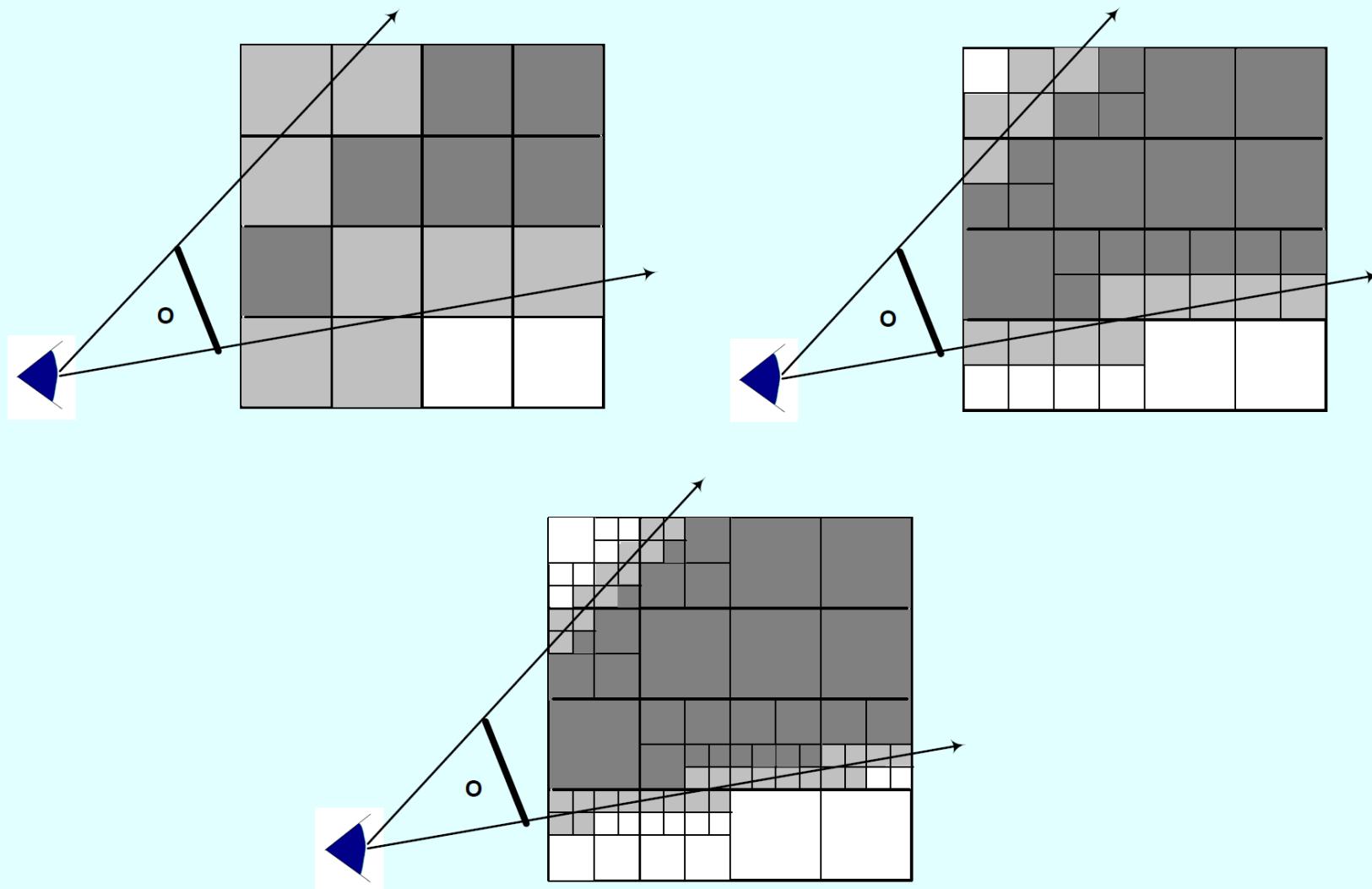
Les optimisations spatiales vont tirer parti des stratégies de subdivision spatiale mises en place pour représenter la scène (BSP, octree, ...)

- Frustum (ou geometry) culling
- PVS (Potentially Visible Set ou Potential Visibility Set)
- Occlusion culling

Frustum culling

- Le principe est de ne pas afficher ce qui est en dehors du champ de vue (frustum) de la caméra.
- Le test est en général réalisé à partir des volumes englobants des objets: si tous les points du volume englobant sont du côté extérieur d'au moins un des plans de clipping de la caméra, alors le volume est en dehors du champ de vue de la caméra.
- Le principe est aussi bien applicable aux objets de la scène (trivial) qu'à la scène elle-même (un peu moins trivial selon les cas), pour peu qu'elle ait été découpée par une stratégie de subdivision spatiale (BSP, octree, etc).

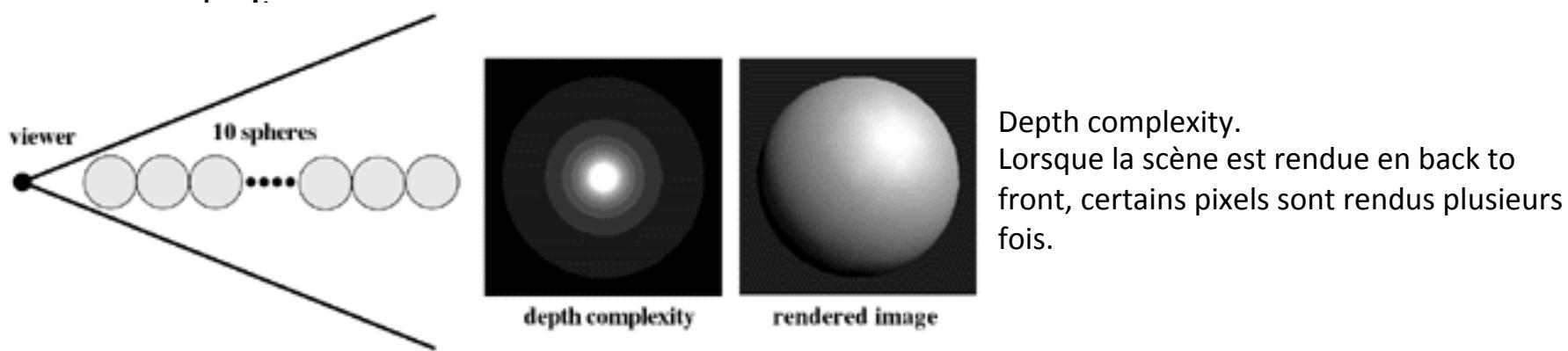


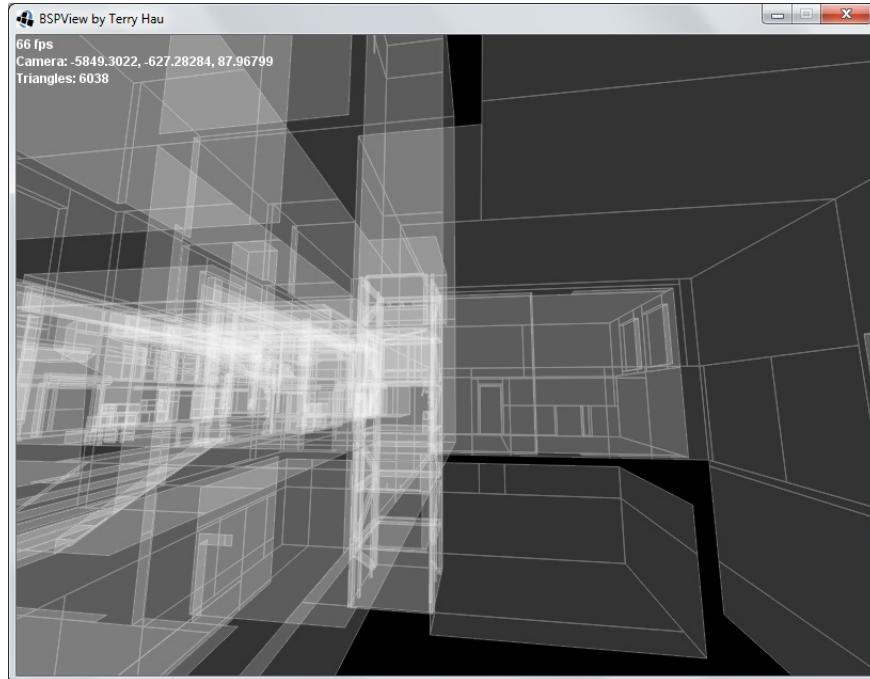


Démonstration de l'intérêt d'un test de visibilité récursif sur une structure spatiale de type octree.

Potentially visible set

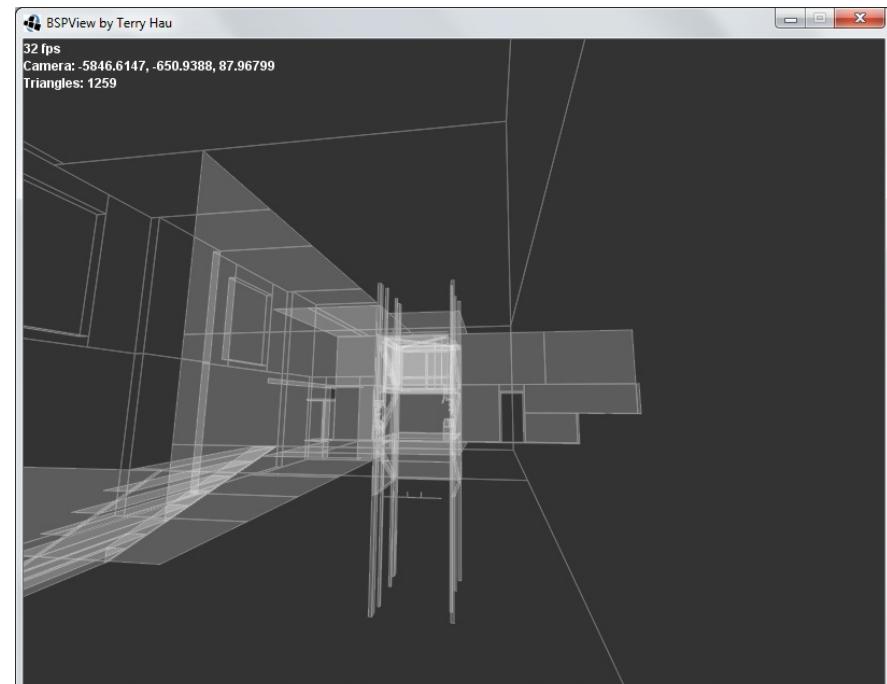
- Pour les scènes contenant un grand nombre d'occludeurs potentiels (en général scènes d'intérieur), on peut générer un PVS, dont le but est de définir les zones de la scène qui seront visibles, et ce à partir de n'importe quelle position.
- On simplifie le problème en stockant pour chaque zone de la scène la liste des zones visibles.
- Habituellement, la génération du PVS est effectuée offline (pré-processing). Elle peut être manuelle (fastidieux, nécessite un éditeur), ou automatique (en général pas optimale).
- L'utilisation d'un PVS permet de répondre en partie au problème de la « depth complexity ».





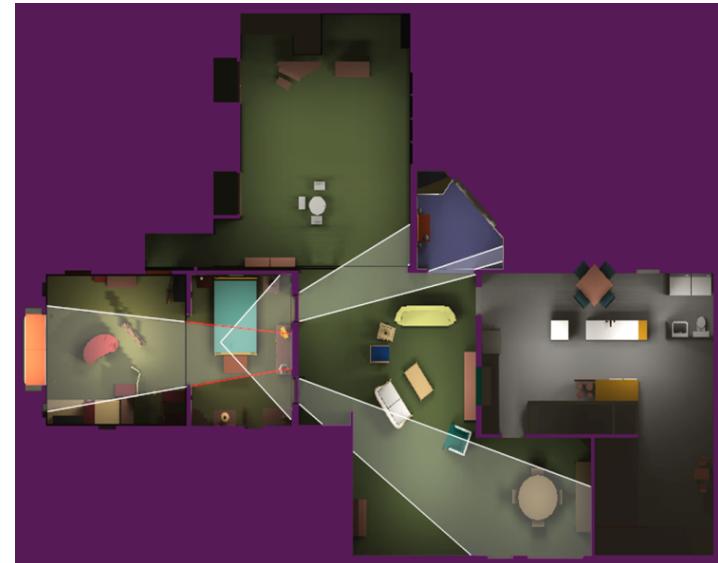
Affichage de la depth complexity d'une scène:

sans utilisation de PVS (image de gauche, 6038 triangles)
avec utilisation de PVS (image de droite, 1259 triangles).



Potentially Visible Sets

- Une approche classique de construction automatique d'un PVS se base sur le mécanisme des « portals ».
- Un portal est un ensemble de polygones (invisibles), qui permettent de faire la transition entre deux zones (volumiques) contigües (ou non !) d'une scène. Un bon exemple est d'imaginer les portes d'un bâtiment, qui relient les différentes pièces.
- A chaque franchissement de portal, le champ visuel (frustum) est réduit relativement à la géométrie du portal (propriété exploitable à la génération du PVS, et pendant la phase de rendu).

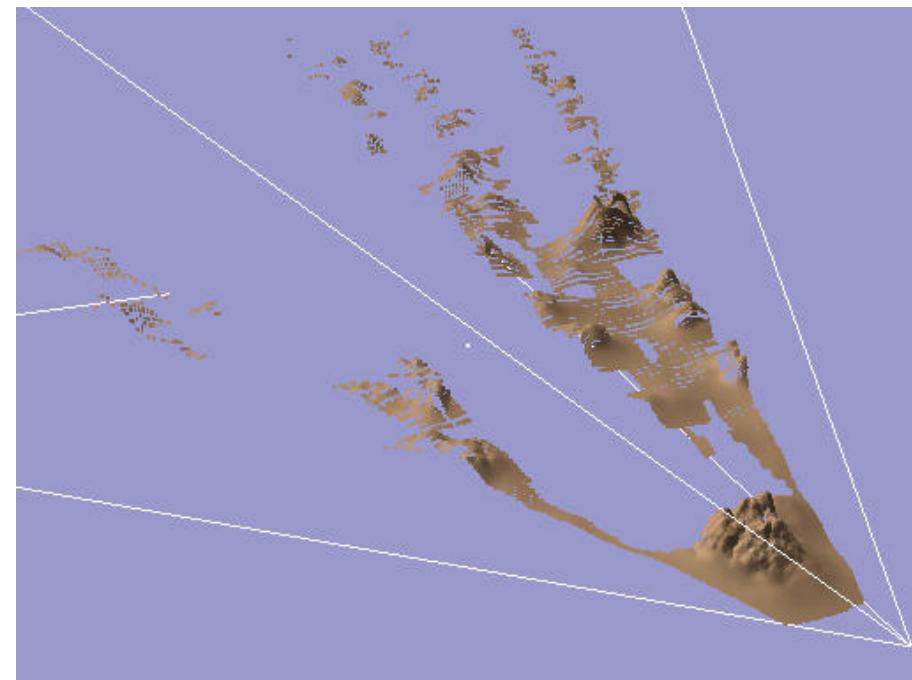


Occlusion culling

Le principe de l'occlusion culling est de tenter de réduire encore plus la quantité de primitives rendues, en identifiant les objets occludeurs qui vont masquer de grosses portions de la scène.

Parmi les méthodes les plus courantes (travaillant dans l'espace image):

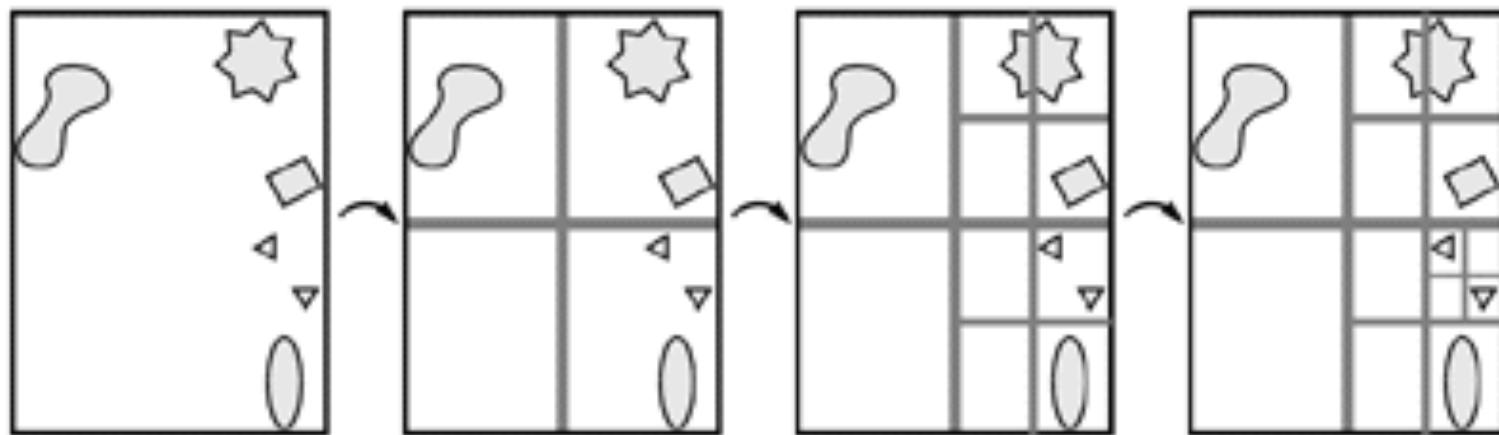
- Hierarchical Z-buffer Visibility
- Hierarchical Occlusion Maps



Cas idéal d'occlusion culling : seules les portions non masquées de la scène sont envoyées au rendu

Hierarchical z-buffer visibility

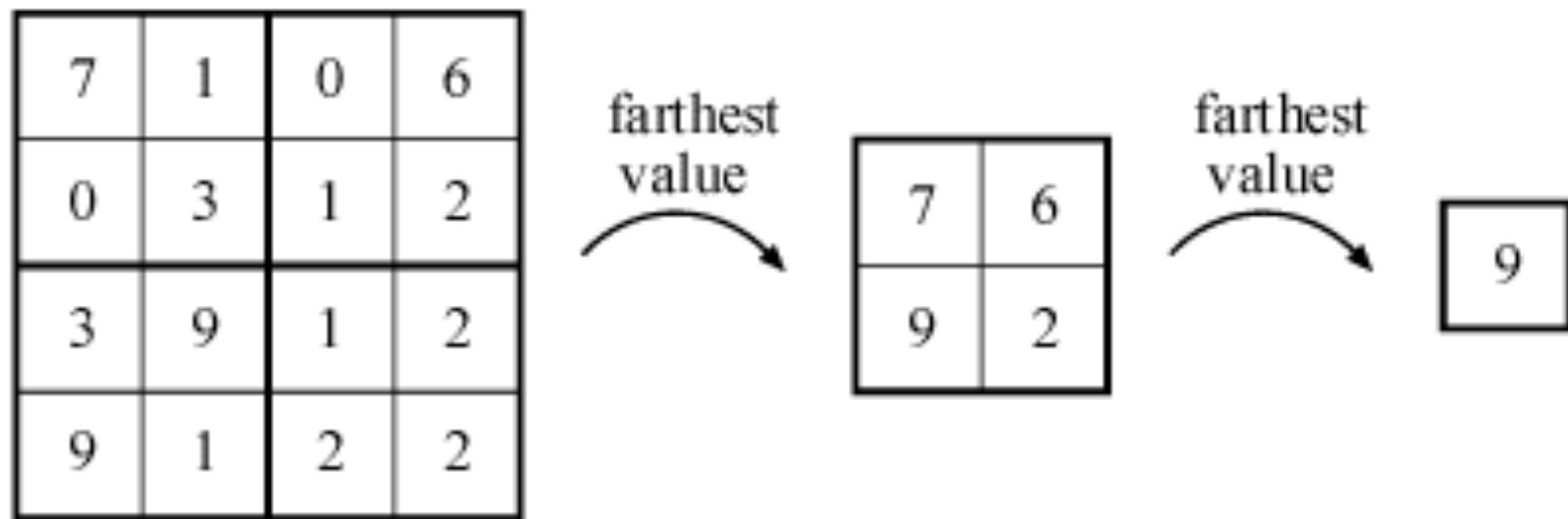
Générer un octree de la scène. Chaque géométrie est associée au plus petit nœud l'englobant dans l'octree.



L'algorithme exploite le fait que, si aucune des faces de la boîte associée à un nœud de l'octree n'est visible relativement au Z-buffer, alors aucune des géométries associées à ce nœud n'est visible.

Si la boîte d'un nœud est visible, alors les géométries associées à ce nœud sont rendues, et on boucle récursivement sur les nœuds fils.

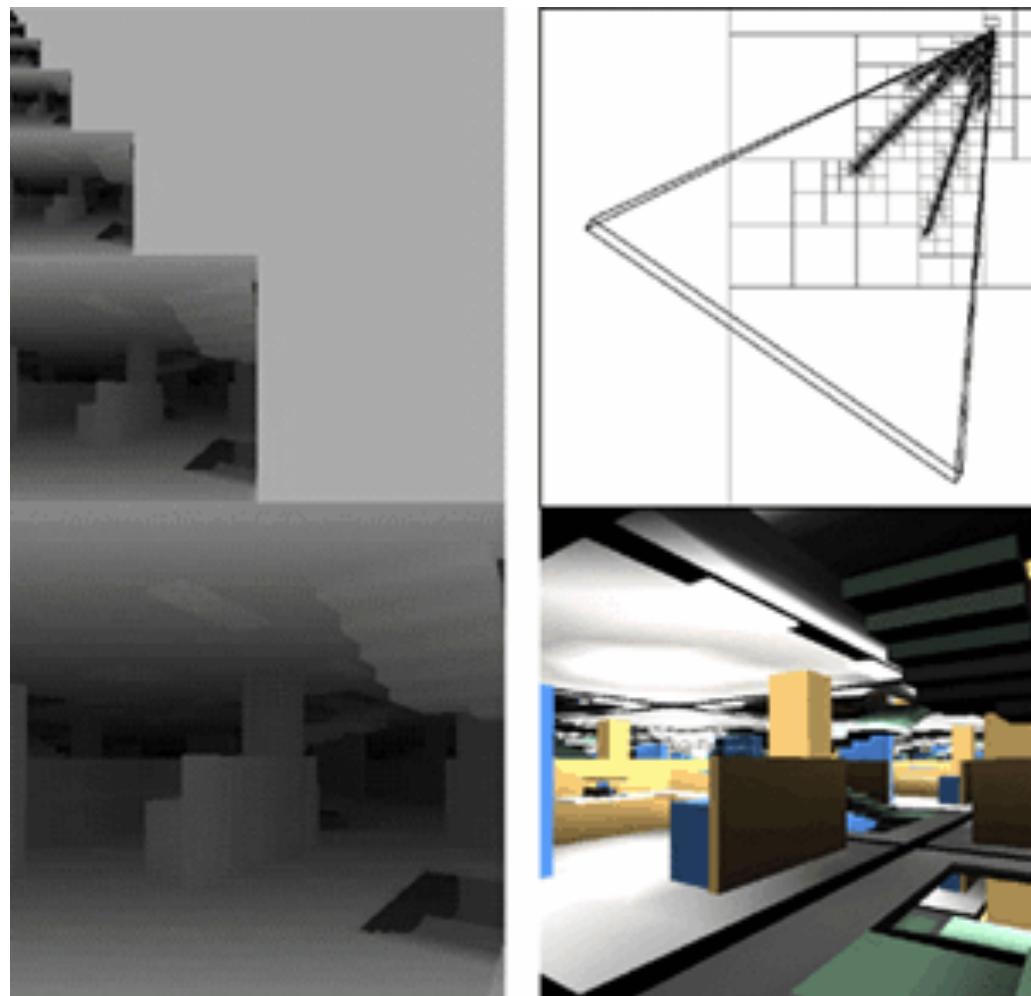
Pour tester la visibilité des faces de la boîte d'un nœud, on utilise une pyramide d'images du Z-buffer : chaque niveau de la pyramide a une dimension de moitié celle du niveau supérieur, et chaque groupe de 2x2 pixels est fusionné dans le niveau inférieur avec la plus grande des 4 valeurs de profondeur. Au niveau le plus bas de la pyramide, on obtient 1 seul pixel qui est la valeur la plus éloignée dans le Z-buffer.



En pratique, la pyramide Z n'est pas reconstruite à chaque mise à jour du Z-buffer. On exploitera plutôt la cohérence dans l'espace image: la pyramide Z est générée à partir des objets qui étaient marqués comme visibles dans l'image précédente.

Pour rejeter rapidement les faces de la boîte englobante, on cherche le niveau de pyramide le plus fin dont les dimensions d'un pixel recouvrent totalement la boîte englobante de la face projetée dans l'espace image, et on compare la valeur du Z-buffer avec la plus petite profondeur de la face.

Si la profondeur de la face est supérieure à la profondeur stockée, la face est masquée.

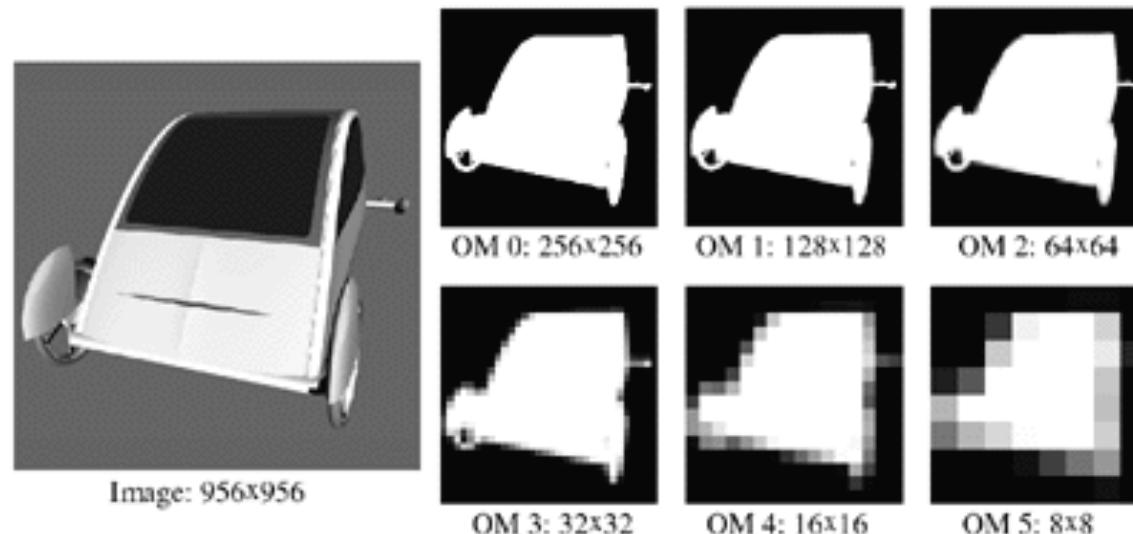


A gauche, pyramide du Z-Buffer. En haut à droite, champ visuel de la caméra représenté dans l'octree de la scène. En bas à droite, rendu de la scène.

Hierarchical occlusion maps

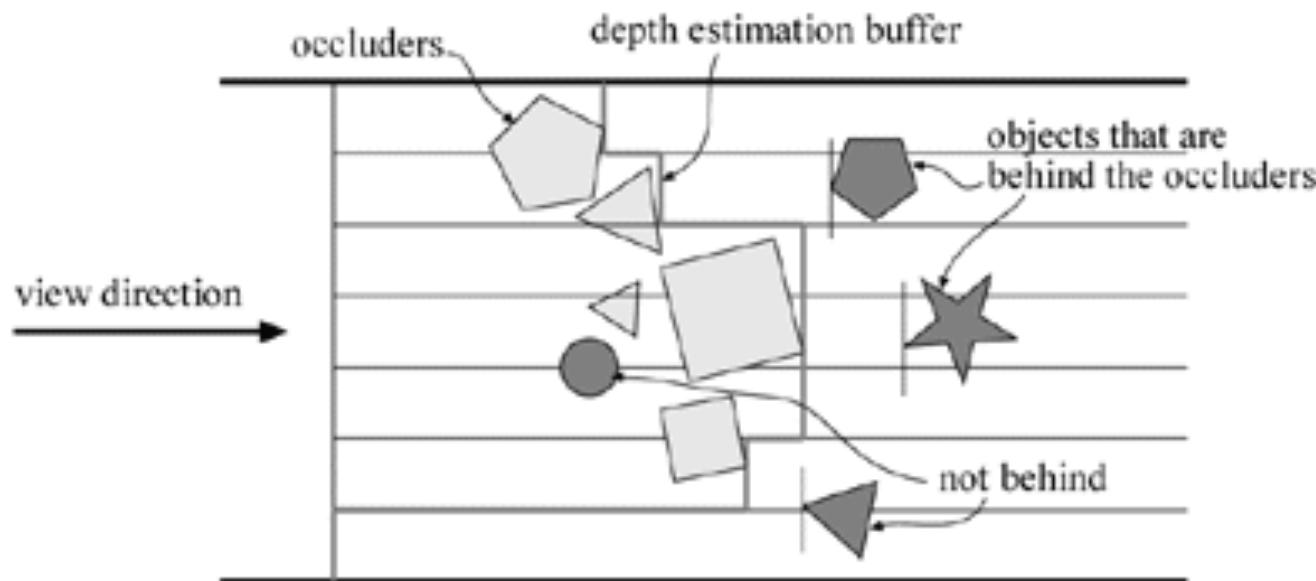
Le principe de fonctionnement est assez similaire à celui du « Hierarchical Z buffer », mais une pyramide d'images contenant les occludeurs est utilisée à la place du Z-buffer, permettant de déterminer des états « semi-visibles ».

- Sélectionner de manière heuristique un ensemble d'objets de la scène qui sont de bons candidats pour masquer partiellement ou totalement les autres objets (taille de l'objet, position relativement à la caméra, etc...)
- Faire un rendu des occlusifs en blanc sur fond noir
- Générer la pyramide d'images (mipmaps) associée à l'image des occlusifs



Puis les autres objets sont rendus, et la projection image de leur boîte englobante est comparée au bon niveau de la pyramide d'occlusion. Si la zone de recouvrement est complètement opaque, et que l'objet est derrière les occlusifs, l'objet est entièrement masqué.

Un Z-buffer standard, ou bien une image d'estimation de la profondeur (depth estimation buffer) sont utilisés pour la détermination de la profondeur de l'objet relativement aux occludeurs.



Le buffer d'estimation de la profondeur est construit en calculant la plus grande profondeur pour chacune des boîtes englobantes projetées des occludeurs. La comparaison est ensuite réalisée pendant le rendu avec la plus faible profondeur des boîtes englobantes projetées des objets de la scène

Les optimisations géométriques regroupent les stratégies mises en place pour limiter le nombre de primitives géométriques affichées pour représenter un objet 3D.

De manière générale, comme pour les optimisations spatiales, on cherche à ne pas afficher ce qui ne sera pas (ou peu) visible dans l'image finale.

Plusieurs approches de simplification géométrique existent:

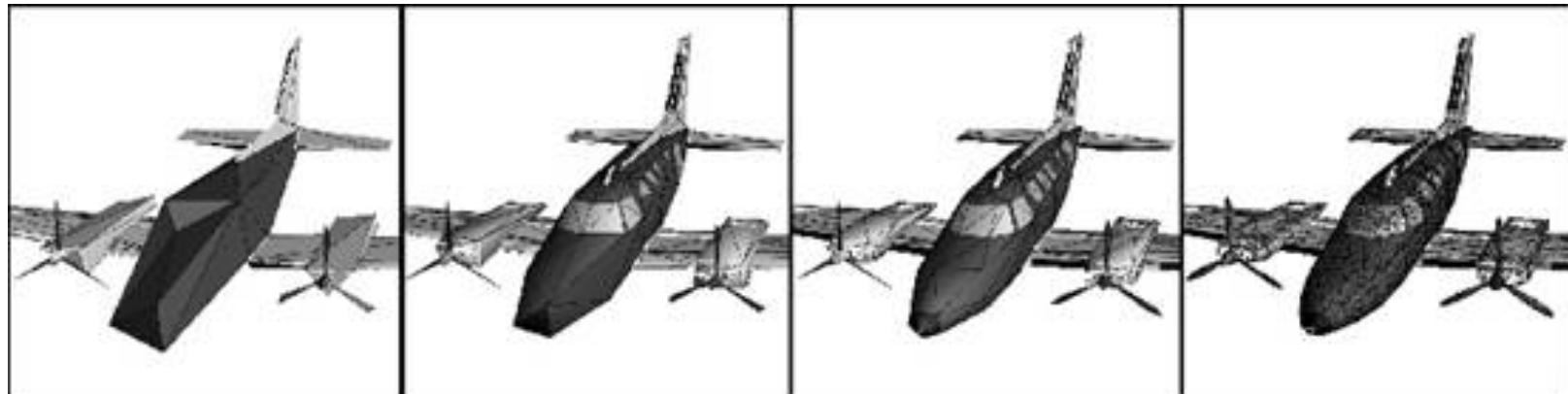
- Backface culling
- LOD (Level Of Detail)
- Imposteurs

Level of details (LOD)

L'idée du LOD est d'utiliser un modèle géométrique simplifié lorsque celui-ci est éloigné de la caméra: les objets éloignés ont besoin de moins de détails puisqu'ils sont plus petits à l'écran: couvrant moins de pixels, les triangles de surface projetée inférieure au pixel ne contribuent pas à l'image finale.

Plusieurs niveaux de LOD sont possibles pour un même objet, permettant d'obtenir des transitions moins brutales. En général, la distance à la caméra est le critère utilisé pour sélectionner le niveau à afficher.

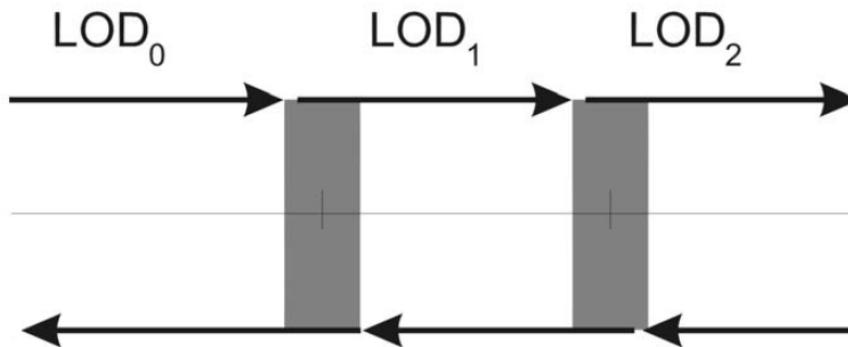
Lorsque l'on met en place une gestion de LOD, il faut toujours se baser sur la surface en pixels à l'écran de l'objet (idem pour les textures) pour déterminer les bonnes distances de transition. Cela signifie donc que chaque objet aura probablement des distances de transition différentes.



On distingue 3 étapes dans la mise en place d'une technologie de LOD:

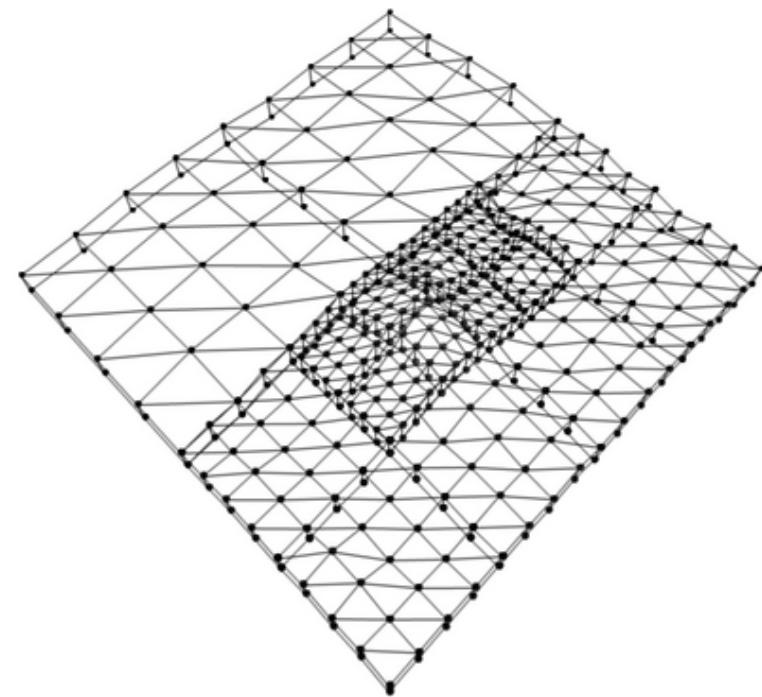
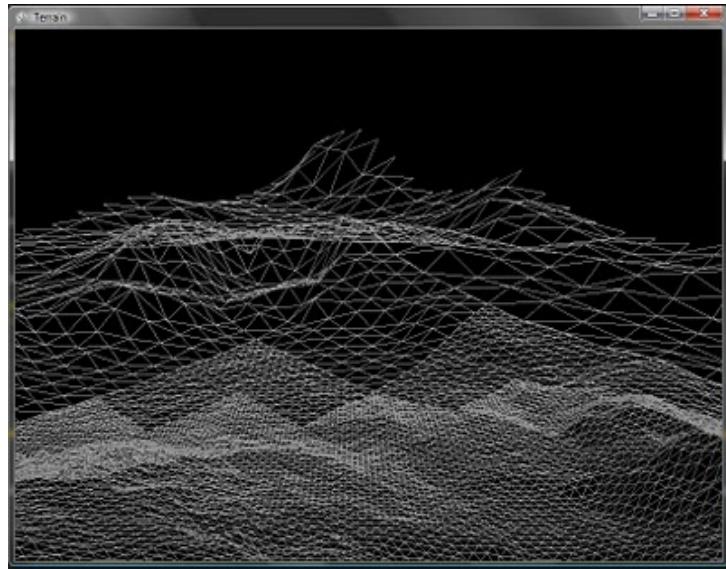
- Génération des niveaux de LOD (automatique ou manuelle). La simplification manuelle donne en général de meilleurs résultats. L'idée est de créer le niveau de définition maximale, puis de supprimer progressivement les détails. Selon les cas, la simplification peut être dépendante ou indépendante de la vue.
- Sélection du niveau de LOD à afficher: on utilise une fonction d'estimation qui servira à sélectionner le bon niveau de LOD. La fonction peut se baser sur:
 - La distance à la caméra: on définit n seuils de transition
 - La projection dans l'image du volume englobant de l'objet et l'estimation de la surface couverte
- Implémentation de la méthode de transition entre les différents niveaux de détail

- Sélection directe du niveau. Problème : génère un effet de « popping » et d'instabilité aux abords des seuils de transition. On utilisera de préférence un seuillage par hystérésis pour atténuer l'effet d'instabilité.

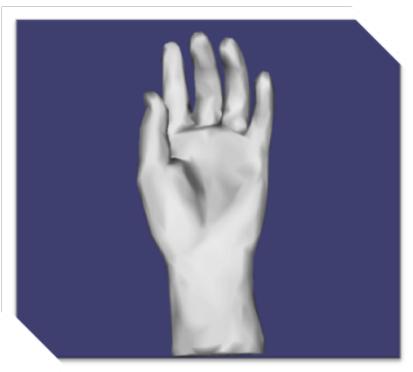


- LOD blending: les 2 niveaux de LOD sont affichés en semi-transparence sur une zone de recouvrement autour du seuil de transition. Permet une transition relativement fluide, mais peut poser des artefacts de transparence, et nécessite l'affichage des deux niveaux en même temps dans les zones de transition.

- Alpha LOD: on n'utilise plus qu'un seul niveau de géométrie, dans lequel on va faire disparaître progressivement par transparence les zones les moins importantes. Mais ce n'est plus tout à fait du LOD !
- LOD continu (CLOD – Continuous LOD). Lorsque la simplification des modèles géométriques est réalisée en concaténant les bords partagés par les triangles (ce qui est généralement le cas), on peut mettre en place une structure de données permettant d'interpoler en temps réel la concaténation des triangles. Le résultat produit une transition non-brutale, mais l'objet bouge tout le temps (pas de seuils de transition), et cette méthode ne permet pas de tirer parti du stripping de triangles.
- Geomorphs: technique proche du CLOD. Les interpolations entre les niveaux sont réalisés dans les zones de transition. Le résultat produit des transitions fluides, mais on voit les géométries bouger. La technique nécessite un gros travail préparatoire des données. On la retrouve plutôt utilisée pour l'affichage de terrains (Black & White - Lionhead).



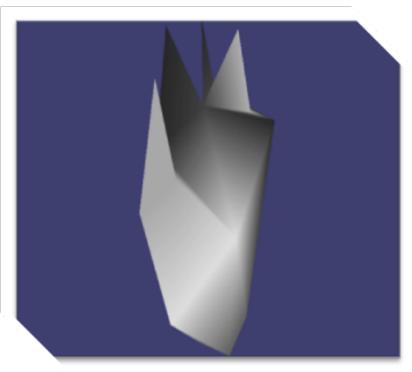
LOD dynamique pour l'affichage d'un terrain.
Le terrain est découpé en zones qui peuvent être plus ou moins raffinées selon différents critères (ici technologie ROAM – Realtime Optimally Adaptive Meshes).



2000 Vertices
4000 Faces



52 Vertices
100 Faces



17 Vertices
30 Faces



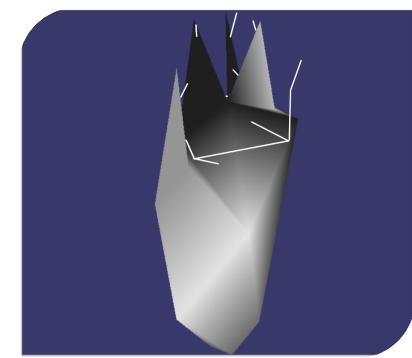
6 Vertices
8 Faces



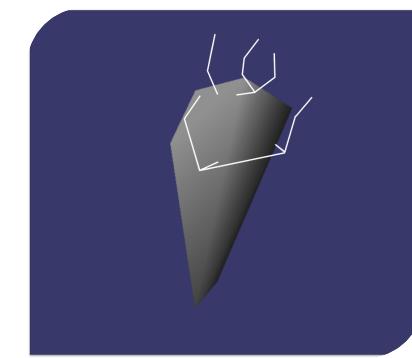
2000 Vertices
4000 Faces



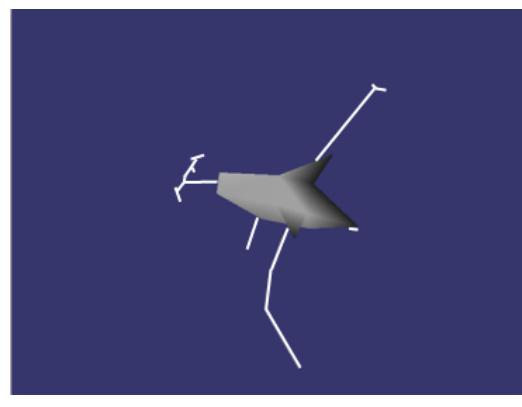
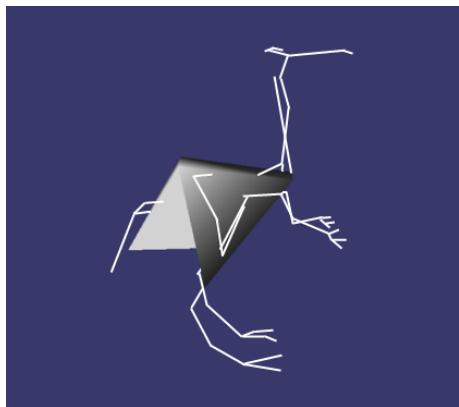
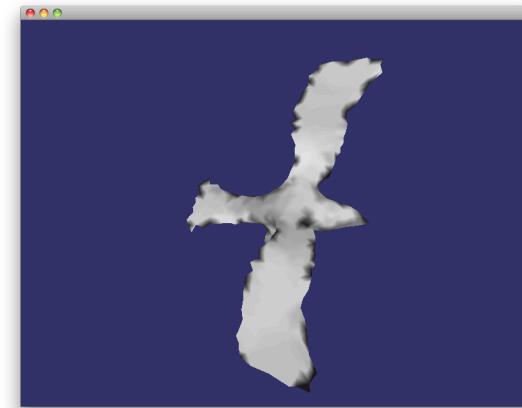
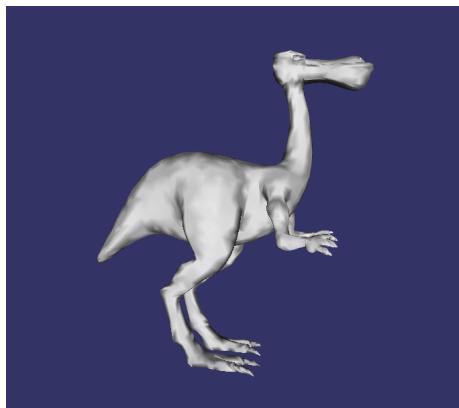
52 Vertices
100 Faces



41 Vertices
30 Faces

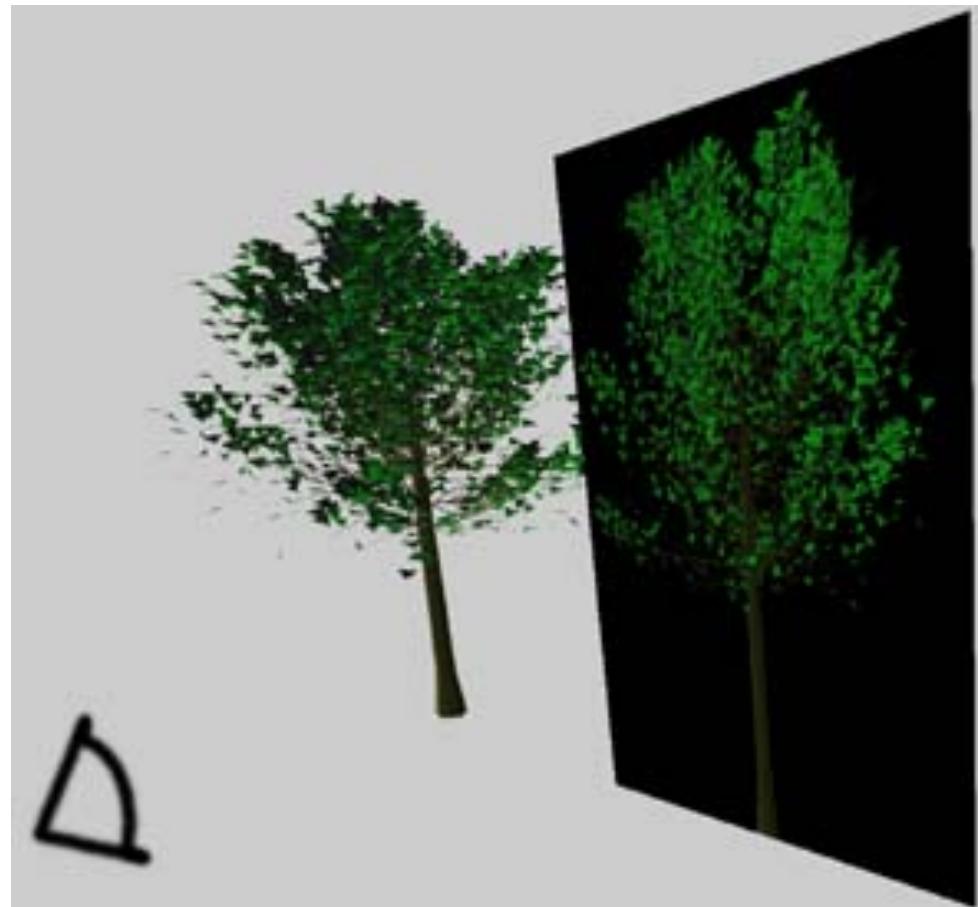


30 Vertices
8 Faces

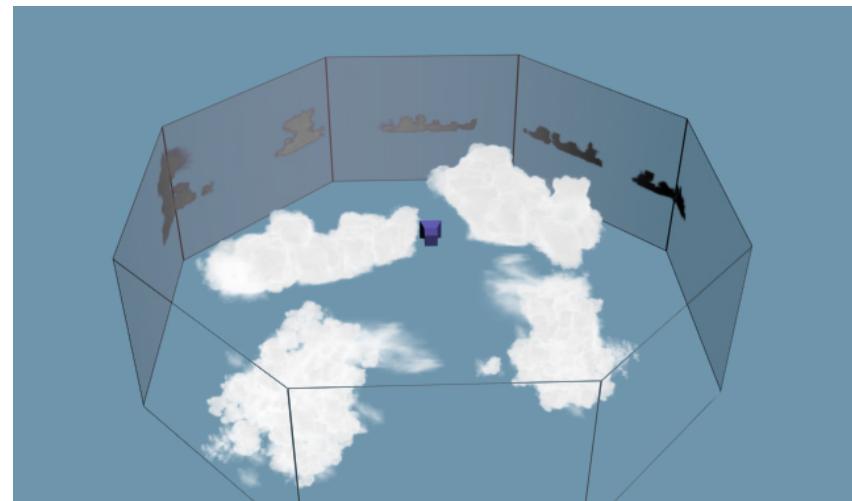
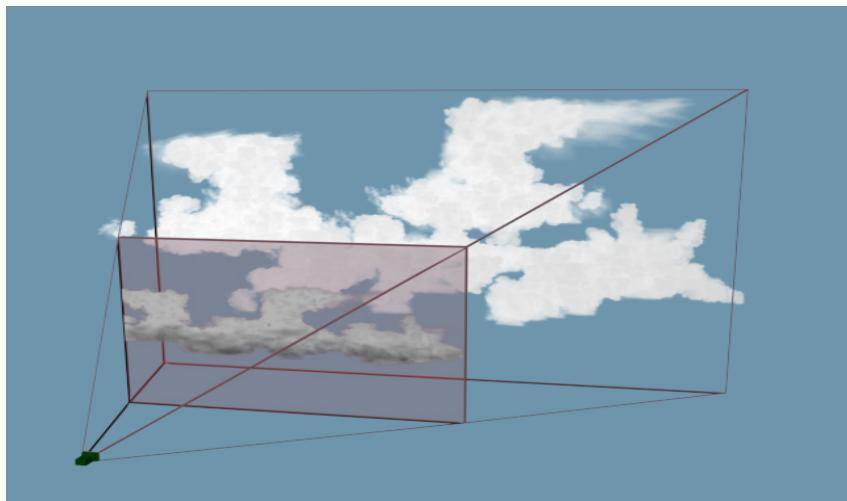
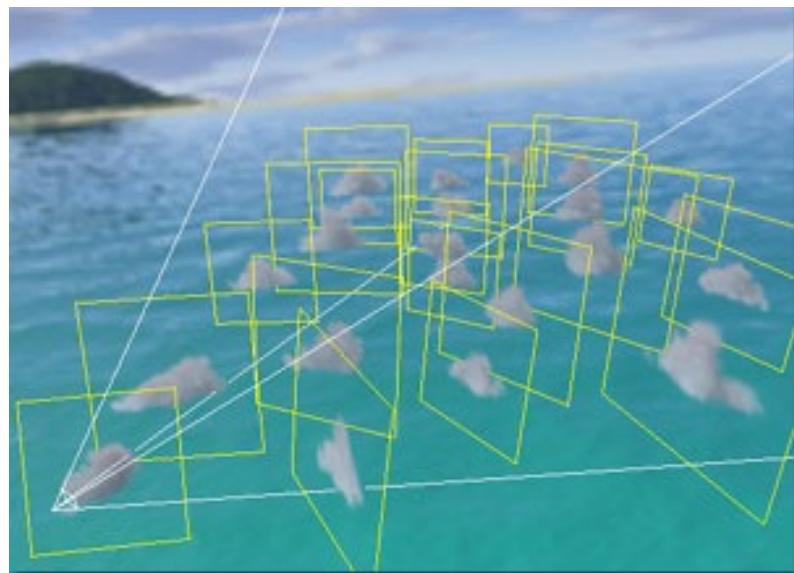


Imposteur

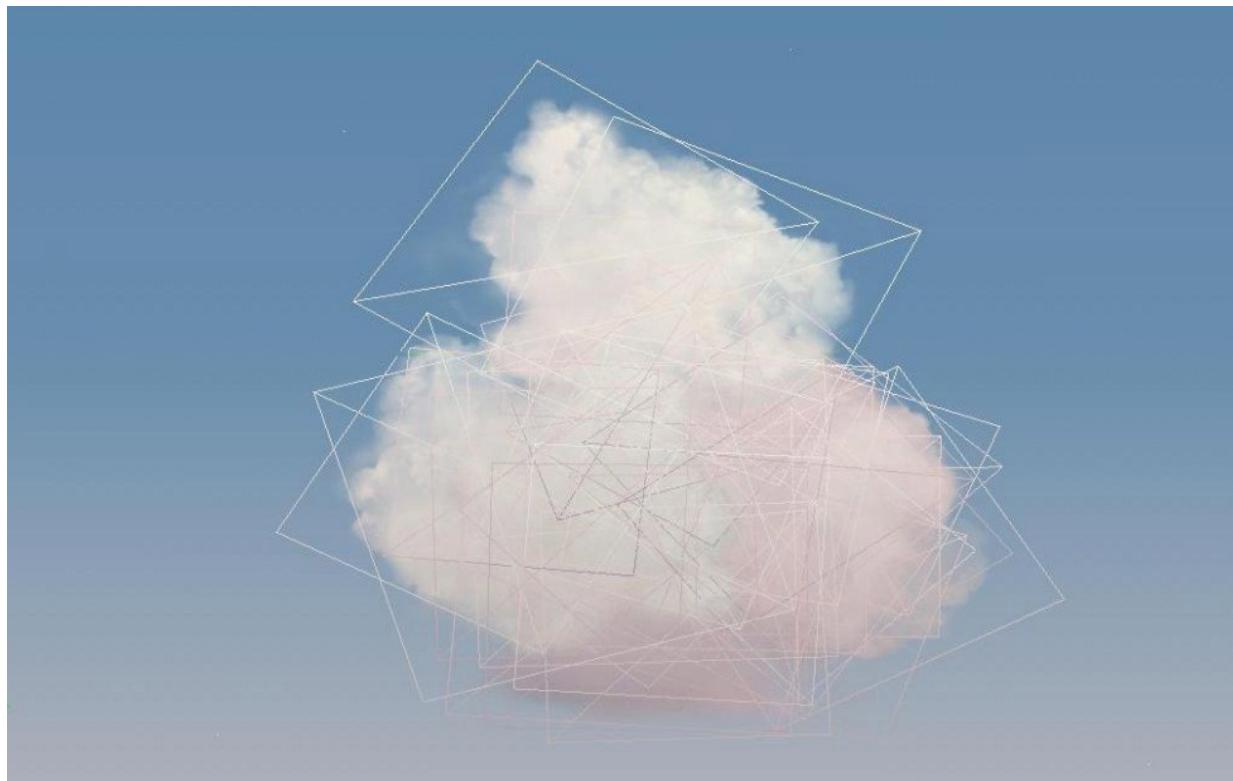
Le principe de l'imposteur (billboard) est de remplacer une géométrie complète par une image de la géométrie, mappée sur une géométrie très simplifiée (en général un simple quadrilatère).



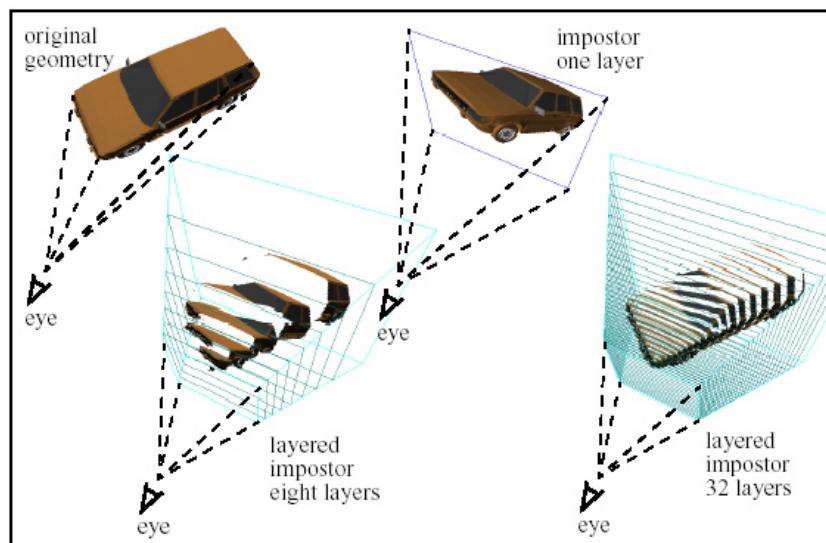
- L'imposteur est en général assez éloigné de l'observateur, et lui fait toujours face.
- La génération de l'imposteur peut se faire offline (infographistes), ou de manière dynamique (ci-dessous, le système de génération des nuages de Flight Simulator 2004)



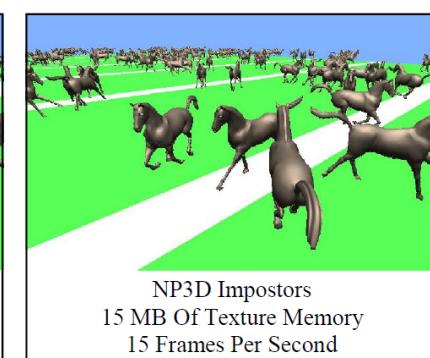
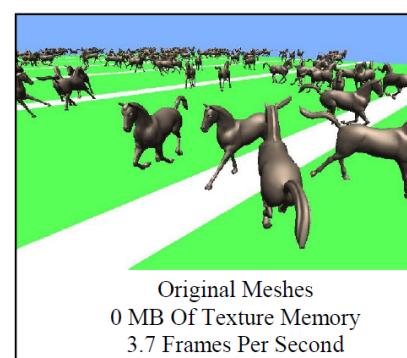
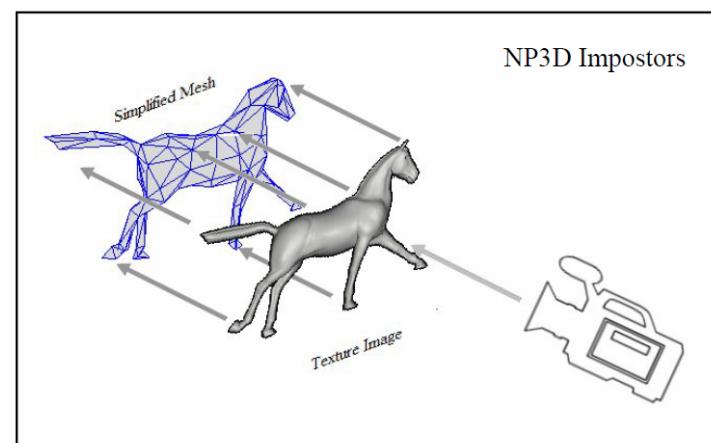
- Plusieurs imposteurs peuvent être combinés pour former un objet plus complexe. Cette technique est en général utilisée pour les effets spéciaux de particules, type fumée, feu, poussière...



Les imposteurs peuvent également être projetés sur des géométries plus complexes, pour améliorer la sensation de relief.



Affichage d'un imposteur en « coupes ». L'idée est identique à la technologie de rendu volumique, dans laquelle on échantillonne un volume de données.



Projection de l'imposteur sur une géométrie simplifiée

- Et
- Maintenant ...
- Vous pouvez réaliser votre dernier commit pour le TP précédent.

TP