Moteurs de jeux

Organisation de scènes

Transformation spatiales 3D

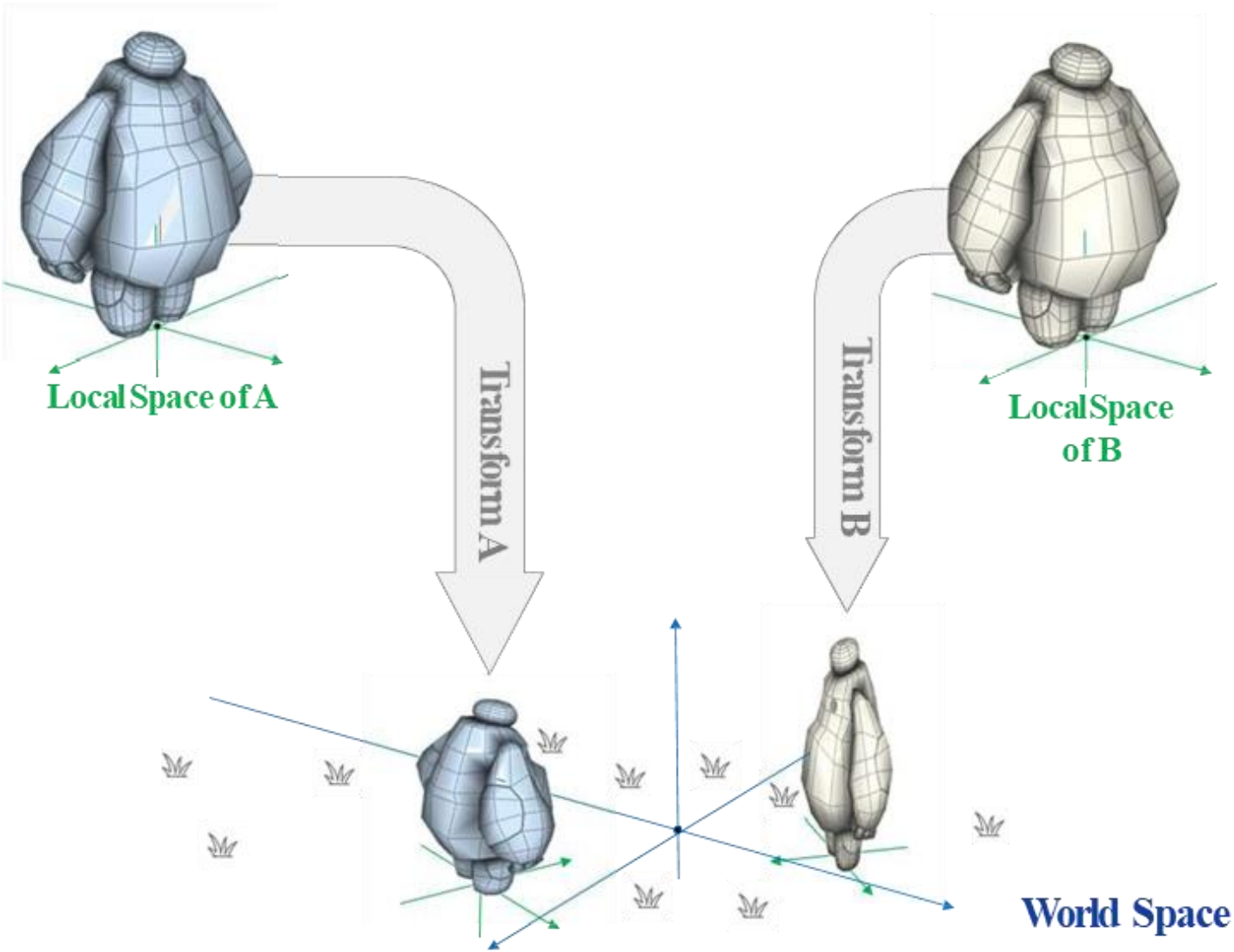
- Fonctions mathématiques :
  - Entrée : point / vecteurs
  - Sortie : point / vecteurs

} Peuvent être appliquées à des modèles 3D (aux positions des sommets et aux normales)

- Typiquement :
  - Mise à l'échelle + rotation + translation

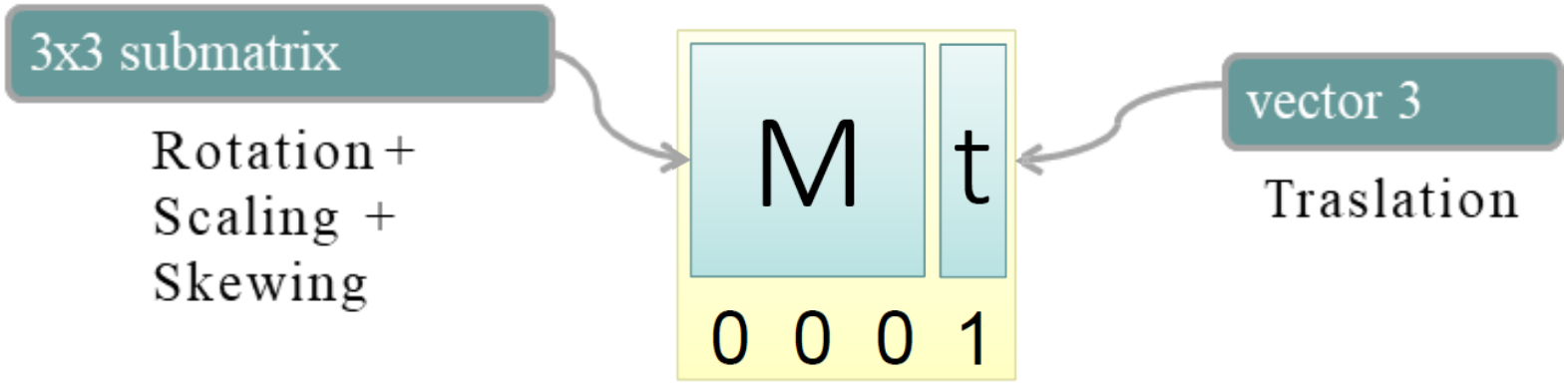
- Modélise :
  - L'agrandissement ou rétrécissement / taille
    - Isotrope (uniforme)
    - Anisotrope (déformation
  - Orientation dans l'espace / rotation
  - Mouvement / position (translation)

Example


Local Space of A
Transform A
Transform B
Local Space of B
World Space

Transformation affine 3D en une matrice 4x4

- Cas général :



- Can be stored as:
  Mat3x3 + Vec3

Représentation de points/vecteurs en coordonnées affines

POSITIONS

- point **p** of Cartesian coords (x , y , z)
  - affine coords of **p** : (x , y , z , **1**)

a Vec3

a Vec4

DIRECTIONS / DISPLACEMENTS

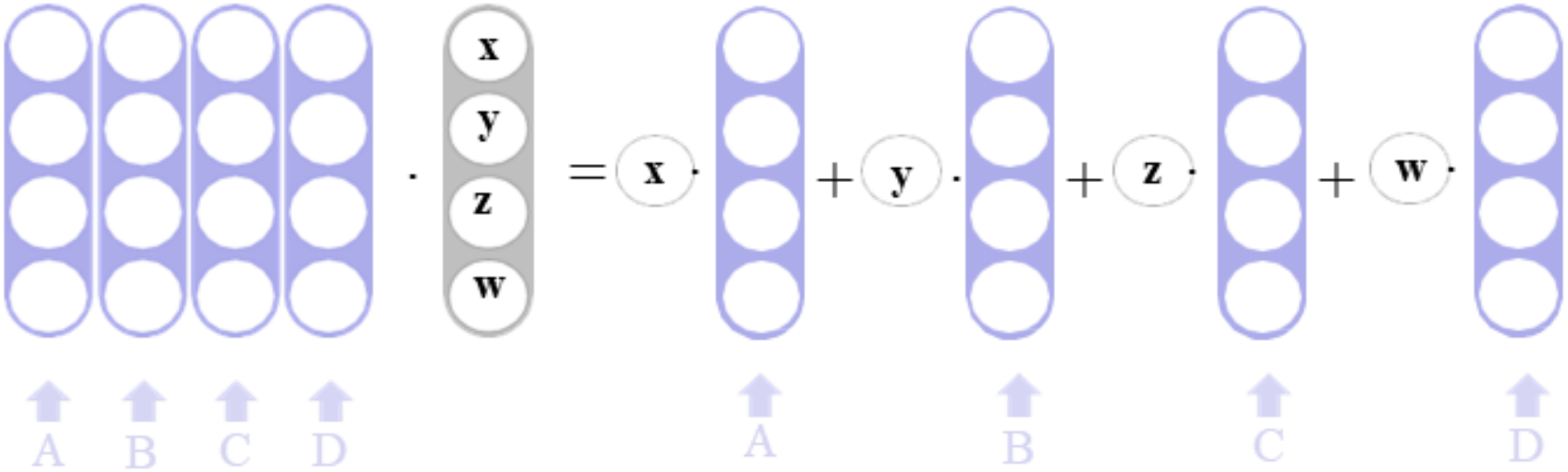
- vector **v** of Cartesian coords (x , y , z)
  - affine coords of **v** : (x , y , z , **0**)

a Vec3

a Vec4

Produit-matrice vecteur

- Une combinaison linéaire des colonnes



Représentations internes des transformations spatiales 3D

- Par exemple une matrice 4x4

```
class Transform {
  // fields:
  Mat4x4 m;

  // methods:
  Vec4 apply( Vec4 p ); // p is in affine coords

  Vec3 applyToPoint( Vec3 p ); // p in Cartesian coords.
  Vec3 applyToVector( Vec3 v ); // v in Cartesian coords.
  Vec3 applyToVersor( Vec3 v ); // v in Cartesian coords.

  ...
}
```

Représentations internes des transformations spatiales 3D

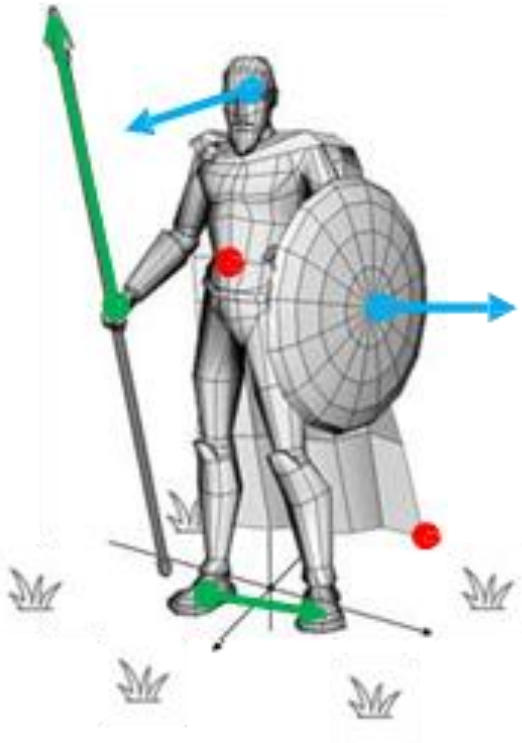
- Variante : une matrice 3x3 plus un vecteur de translation

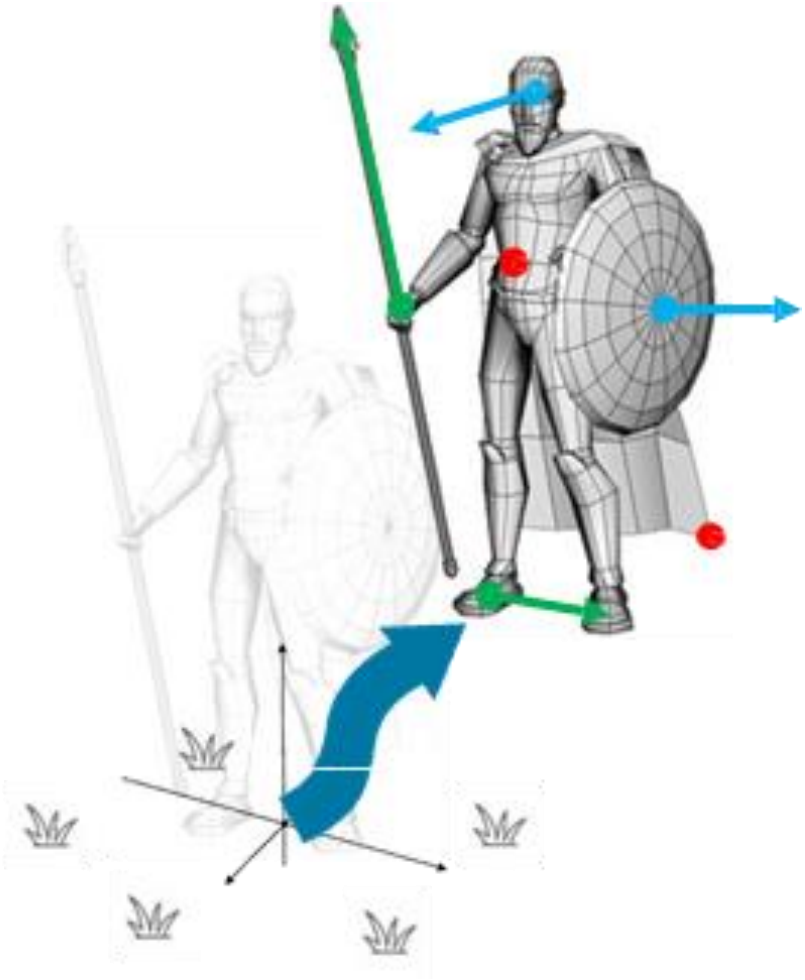
```
class Transform {
  // fields:
  Mat3x3 m; // rotation + skew + scale
  Vec3 t; // translation

  // methods:
  Vec3 applyToPoint( Vec3 p ){
      return m* p + t;
  }
  Vec3 applyToVector( Vec3 v ){
      return m* p;
  }
  ...
}
```

Effets d'une transformation, sur différentes choses

- Translation

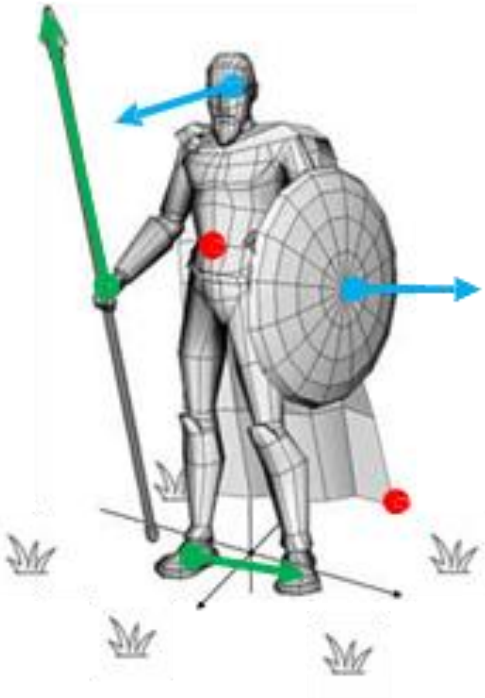


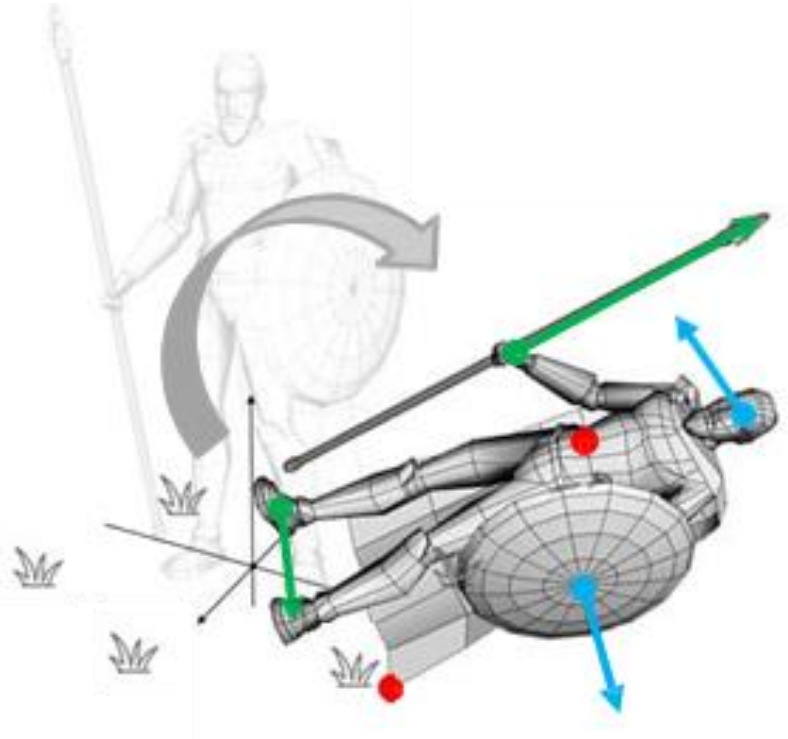


points? vectors? versors?

Effets d'une transformation, sur différentes choses

- Rotations

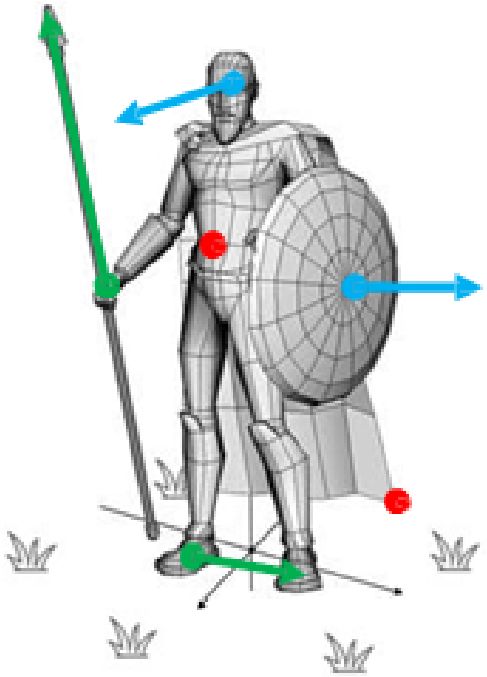


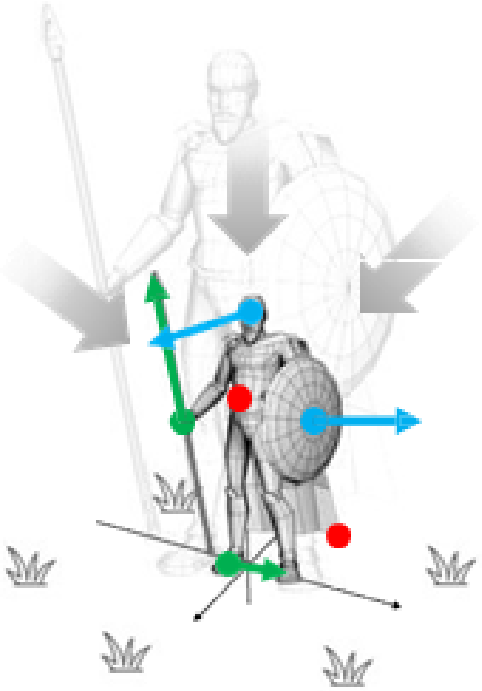


points? vectors? versors?

Effets d'une transformation, sur différentes choses

- Mise à l'échelle (scaling)





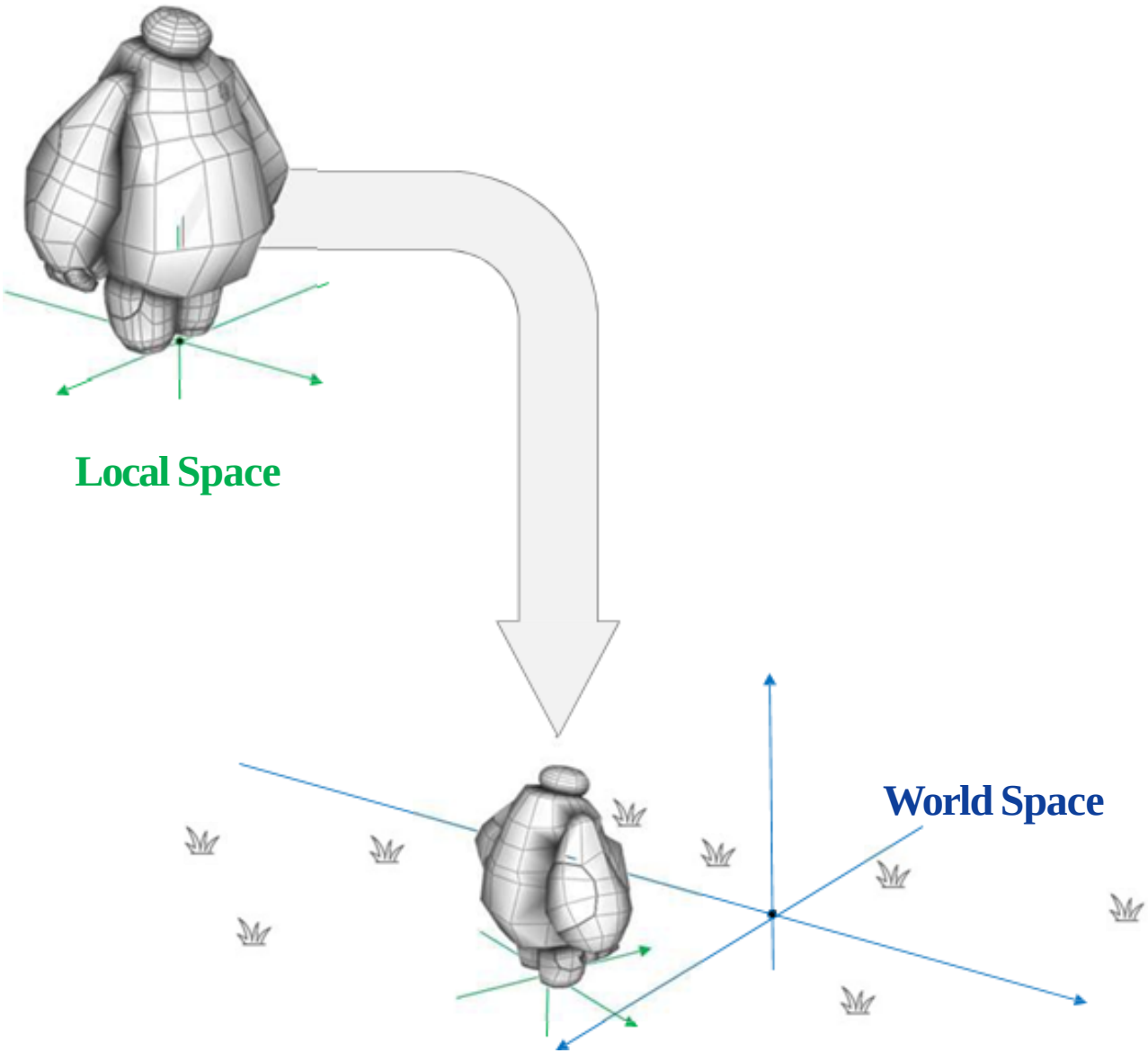
points? vectors? versors?

Effets d'une transformation, sur différentes choses

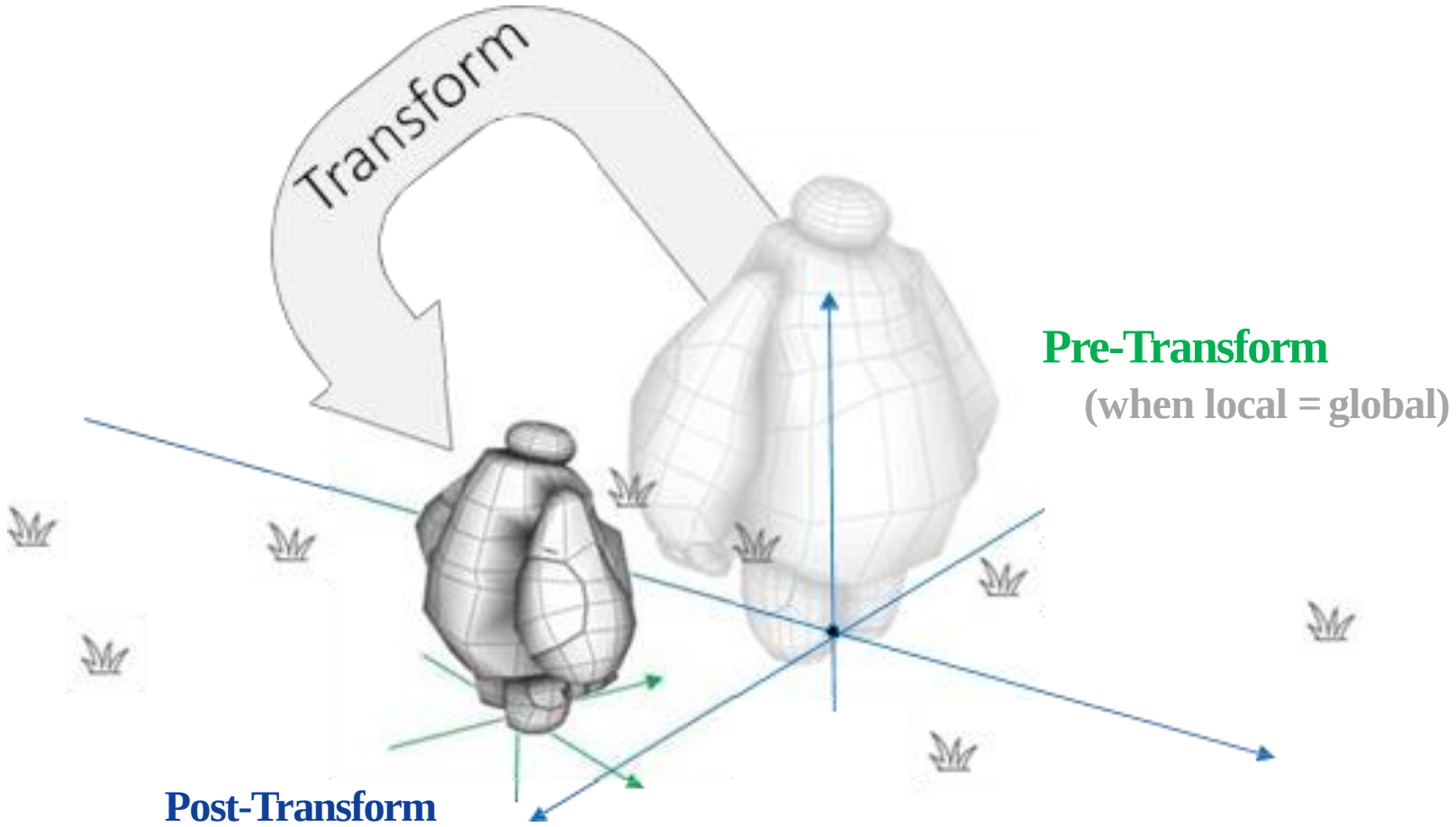
- **Rotation** :
  - S'applique aux **Points, Vecteurs, Versors** (de la même façon)

- **Scaling** Uniforme :
  - S'applique aux **Points, Vecteurs** (de la même façon)
  - N'affecte pas les **Versors**

- **Translation** :
  - S'applique seulement aux **Points**
  - N'affecte pas les **Vecteurs** et **Versors**


Local Space
World Space

Ou



2 equivalent ways to see transformations

Translation the vector of which the object is displaced	OR	**Position** where the object currently is
Rotation by how much the object is spun, re-orienting it	OR	**Orientation** where the object is currently facing toward
Scaling by how much the object is enlarged or shrunk	OR	**Size** how big the object currently is (1 = original size)

Choisir les transformations supportées par le moteur

- **Rotation + translation + mise à l'échelle (non uniforme) + skews**
  - c'est-à-dire la classe des «transformations affines»
  - c'est-à-dire toutes les transformations linéaires
  - communément exprimé sous forme de matrices 4x4 dans CG

- **Rot + Transl + Mise à l'échelle uniforme**
  - terme mathématique: vous obtenez la classe des «similitudes», aussi appelée «transformations similaires»,
  - aka «transformation conforme»
  - préservent les angles, donc les «formes»

- **Rot + Transl**
  - terme mathématique: vous obtenez la classe des «isométries»
  - «Transformations isométriques», «Transformations rigides», «roto-translations»
  - remarque: chacun préserve les distances, donc les angles, les surfaces, les volumes…

Représentation

- On a besoin des méthodes telles que :

```
class Transform {
  // fields:
  ...
  // methods:
  Vec3 applyToPoint( Vec3 p );
  Vec3 applyToVector( Vec3 v );
  Vec3 applyToVersor( Vec3 v );

  Transform combine_with( Transform&t );
  Transform inverse();
  Transform interpolate_with( Transform&t , float k );
}
```

Code d'exemple : interpolation (ou blend, mix, lerp…)

```
class Transform {
  // fields:
  float s;     // uniform
  scale Rotation r; //
  rotation Vec3 t;
                //
  translation

  Transform mix_with( Transform b, float k
      ){ Transform result;
      result.s =this.s * k +b.s * (1-k);
      result.r =this.r.mix_with( b.r , k
      ); result.t =this.t * k +b.t * (1-
      k); return result;
  }

}
```

Transformation ne sont pas (toujours) des matrices 4x4

- Une matrice 4x4 est un moyen de représenter un type de transformation 3D de
  - spécifiquement: **transformations affines**

- Type utile, et c'est un bon moyen
  - élégant, pratique…
  - en CG, manière si courante que «matrice» est synonyme de «transformation». Exemple: la « matrice de vues »

- Pour les jeux cette méthode n'est pas idéale
  - Cela ne correspond pas particulièrement à tous les critères dont nous avons besoin…

On a besoin les transformations

- **Compact**
  - empreinte mémoire pour une transformation?
- **Rapide à appliquer**
  - à quelle vitesse l'applique-t-il à un (ou 80.000) points / vecteurs ?
- **Facile à interpoler**
  - étant donné 2 transformations, est-il possible / facile de les interpoler?
  - si oui, quel est le «bon» résultat?
- **Rapide à combiner**
  - étant donné que N se transforme, est-il facile de trouver la transformation combinée?
  - (nb: la combinaison n'est pas commutative!)
- **Rapide à inverser**
  - est-il facile ou rapide de trouver la transformation inverse?
- **Intuitif**
  - est-il facile d’utiliser les modélisateurs / scénographes / animateurs / etc.?

Garder les composants de la transformation séparés

une Transformation = { une Rotation
+une Mise à l'échelle (uniforme ou non)
+une Translation

important!
L'**ordre est fixe** (typiquement : scaling, rotation, translation)

Avantages de les stocker séparément

- **Nous pouvons choisir uniquement les composants dont nous avons réellement besoin**
  - (économise de l'espace et simplifie toutes les opérations)
- **L'inversion est plus rapide**
  - (N'oubliez pas de compenser pour l'ordre fixe)
- **Possibilité d'appliquer uniquement les composants pertinents à chaque entité**
  - (plus rapide, c'est-à-dire aux points, vecteurs)
- **Intuitif à mettre en place**
  - (chaque composant a un sens très intuitif)

Représentations internes des transformations spatiales 3D

- Beaucoup de possibilités, mais typiquement:
  - translation (vec3) + échelle (float ou vec3)
    + rotation (matrice 3x3 / quaternion / axe + angle / angles d'Euler)

- Nous supposons qu'une transformation est :
  - Légère en mémoire (quelques dizaines d'octets)
  - Rapide à appliquer (même pour les grands modèles complexes)
    - à la volée lors du rendu sur le GPU
  - Rapide à:
    - Interpoler avec une autre (aka "blend" / "mix" / "lerp")
    - inverser (trouver la transformation opposée)
    - cumuler (trouver A * B)

- Toujours interprétable comme un changement de référentiel (vrai pour les «affines»)

Transformations associées à un objet de la scène

- Chaque objet associé à une position spatiale dans le jeu a une transformation associée :

- De :
  - **espace local** a.k.a.
  - **espace objet** a.k.a.
  - espace de **pre-transformation** a.k.a.
  - espace du « héro » / espace « caméra « / espace « bazooka » etc
- Vers :
  - **espace global** a.k.a.
  - **espace monde** a.k.a.
  - espace **post-transformation**

Scène composées : transformations hiérarchiques

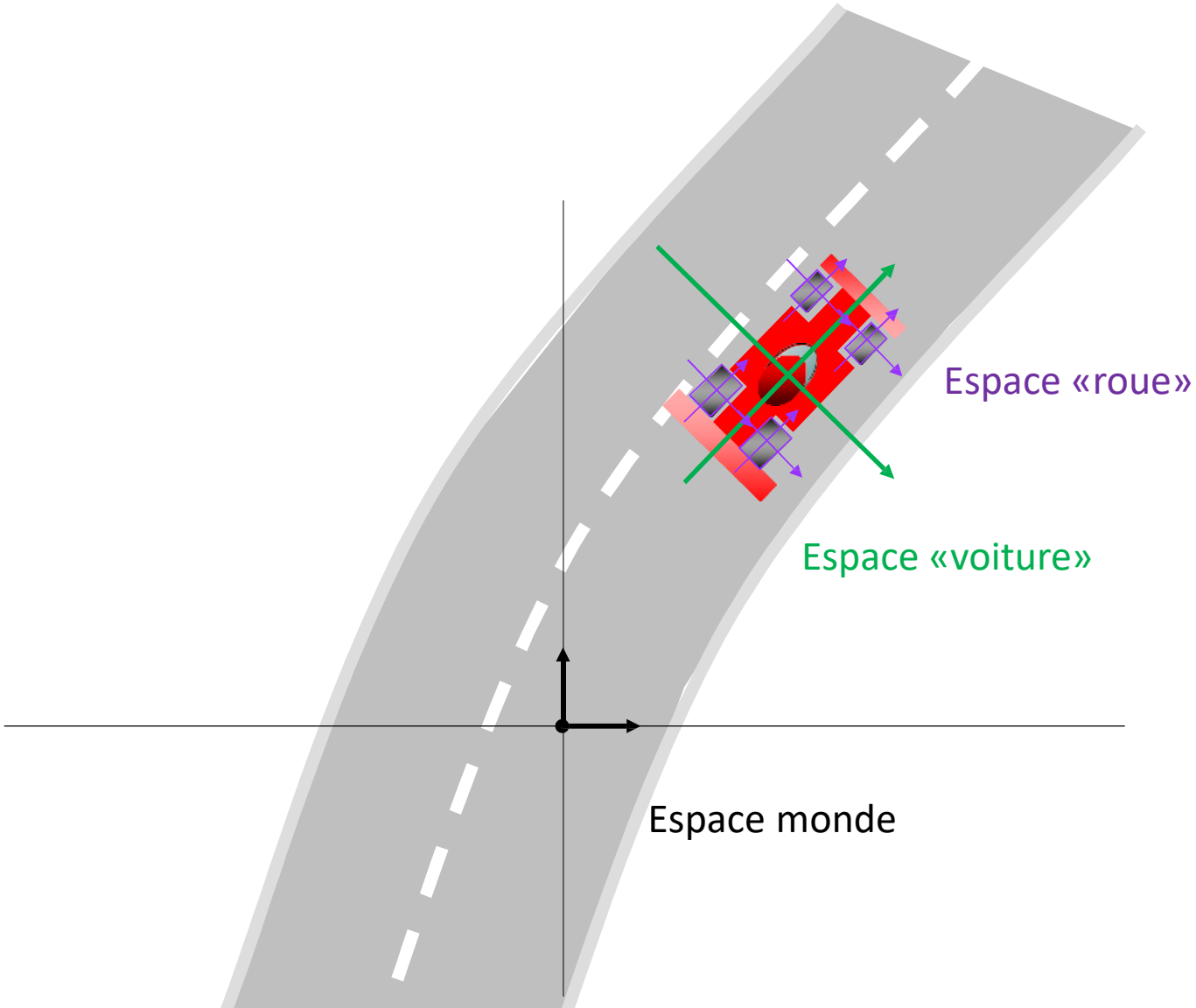
- Jusqu'ici, nous avons supposé que la transformation de chaque objet passe du local au global en une étape

- En réalité, la **scène est construite hiérarchiquement**

- Les **objets constitués de sous-objets**
  - ville faite de maisons faites de murs de briques
  - «chapeau» assis sur une «tête» reposant sur un «personnage» assis dans un «vaisseau spatial» se déplaçant à travers la «scène»

- Aussi: différentes instances d'un même objet peuvent apparaître à plusieurs endroits de la scène

Scène composée

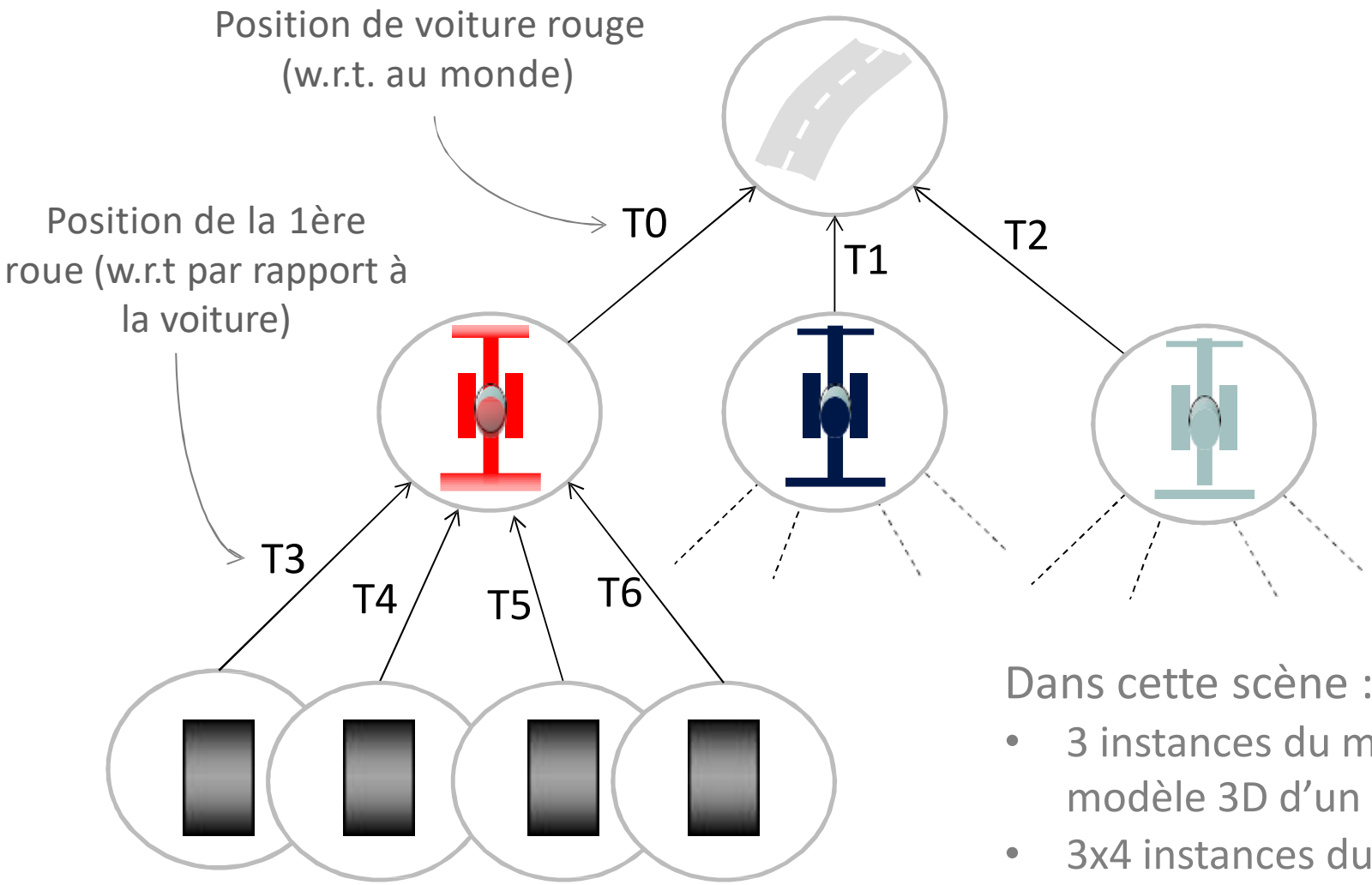


Graphe de scène

- **Un arbre** (i.e. structure hiérarchique)

- Chaque nœud : un espace (un repère)
- Associé à chaque nœud :
  - Des instances de choses (modèles 3D, lumières, caméras, point de résurrection, explosions etc…)
- Nœud racine : espace monde
- Sur les arêtes : **la transformation locale**

Graphe de scène



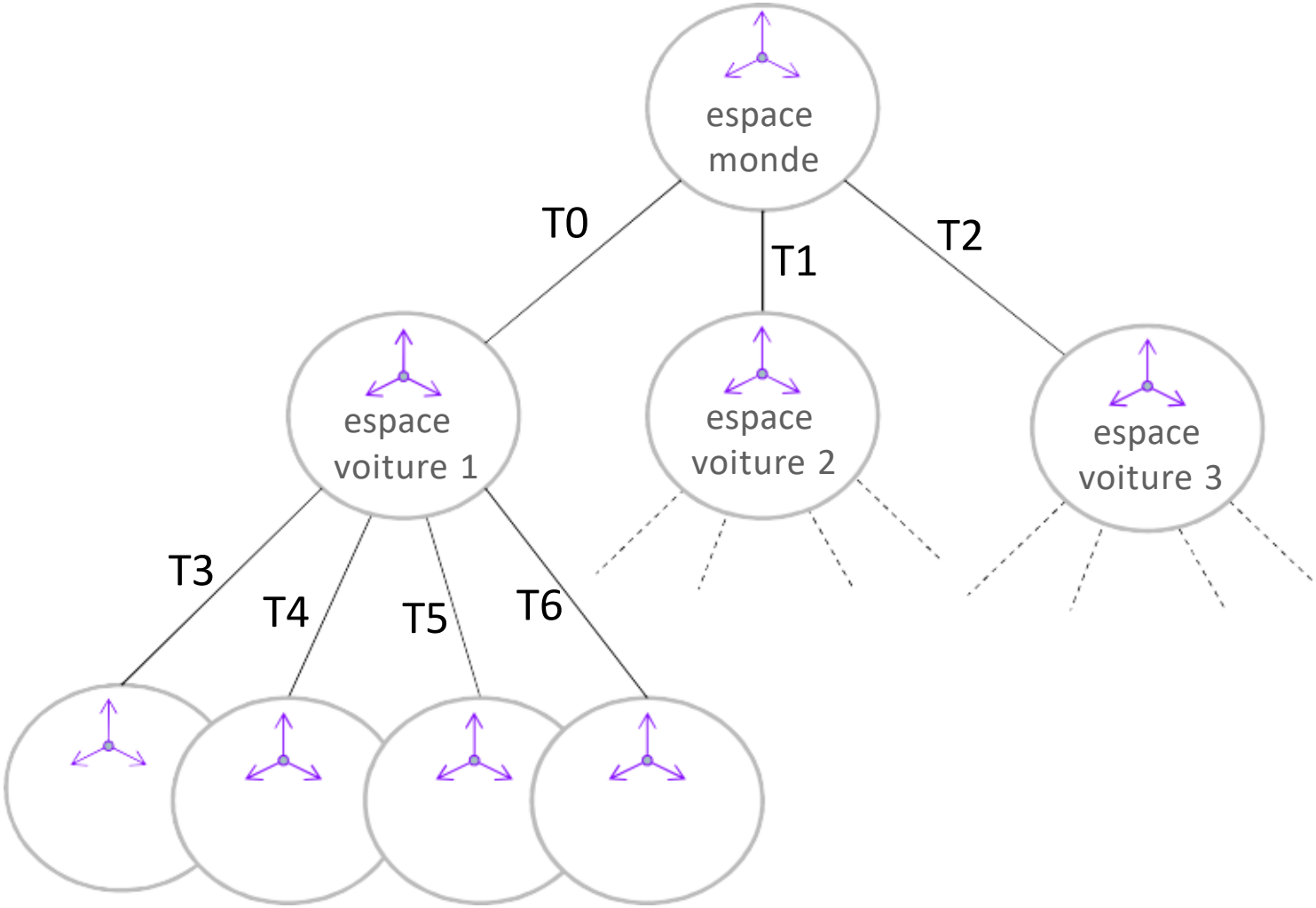
Dans cette scène :

- 3 instances du même modèle 3D d'un véhicule
- 3x4 instances du même modèle 3D de roue

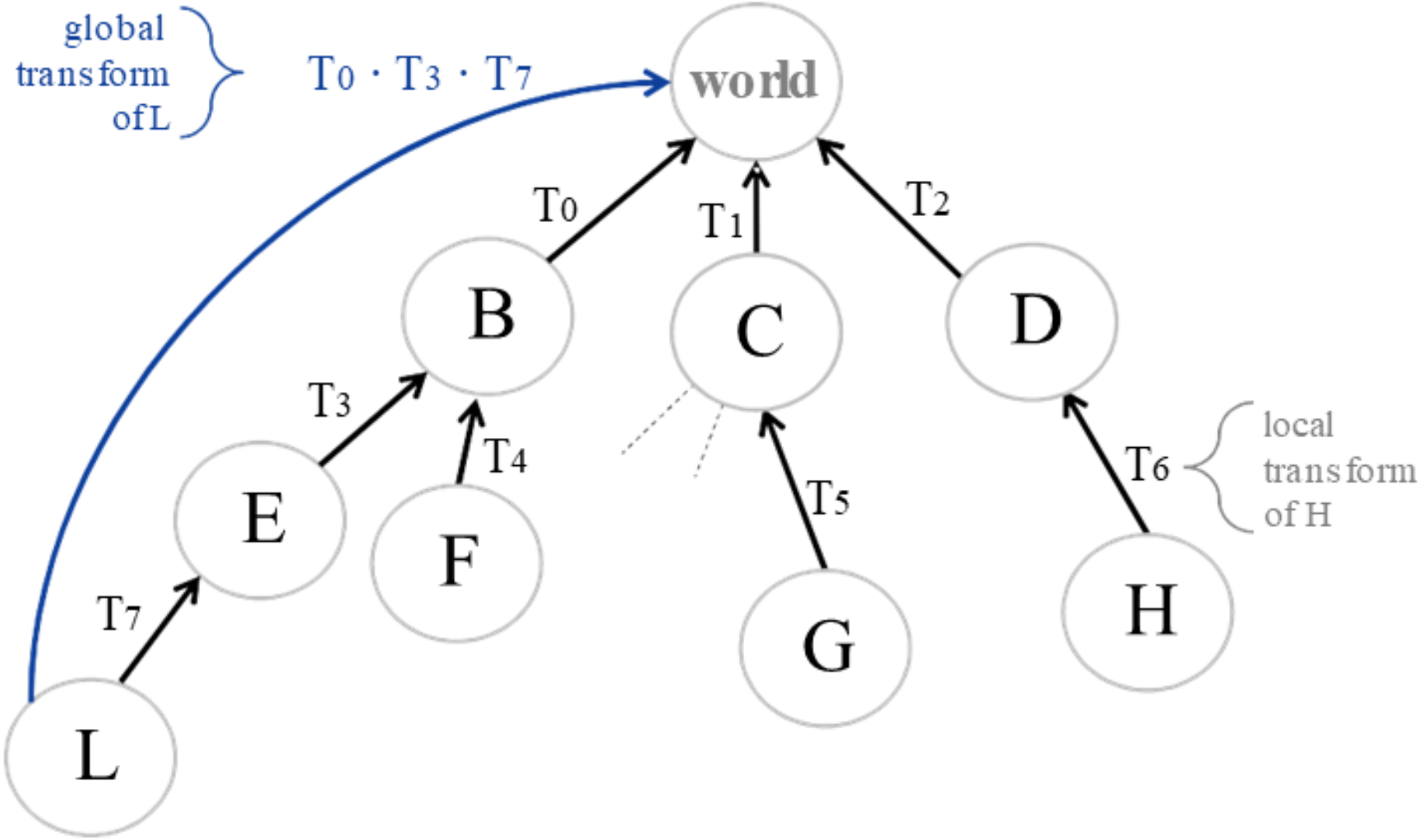
Transformation **local vs globale**

- **Transformation locale** (transformation «relative»)
  - d'un espace de nœud à l'espace de son parent
- **Transformée globale** (transformée «absolue»)
  - d'un espace de nœud à l'espace monde
  - obtenue en **cumulant les transformations locales**
- Avantage: changer les transformations d'un nœud affecte tout le sous-arbre

Graphe de scène



Example


global
transform
of L
T0 · T3 · T7
world
T0
T1
T2
B
C
D
T3
T4
T5
T6
local
transform
of H
E
F
G
H
T7
L

Caméra dans le graphe de scène

- **Caméra** :
  - juste un élément quelque part la scène graphique
  - Rendu de la scène : il doit y avoir une caméra dans le graphe

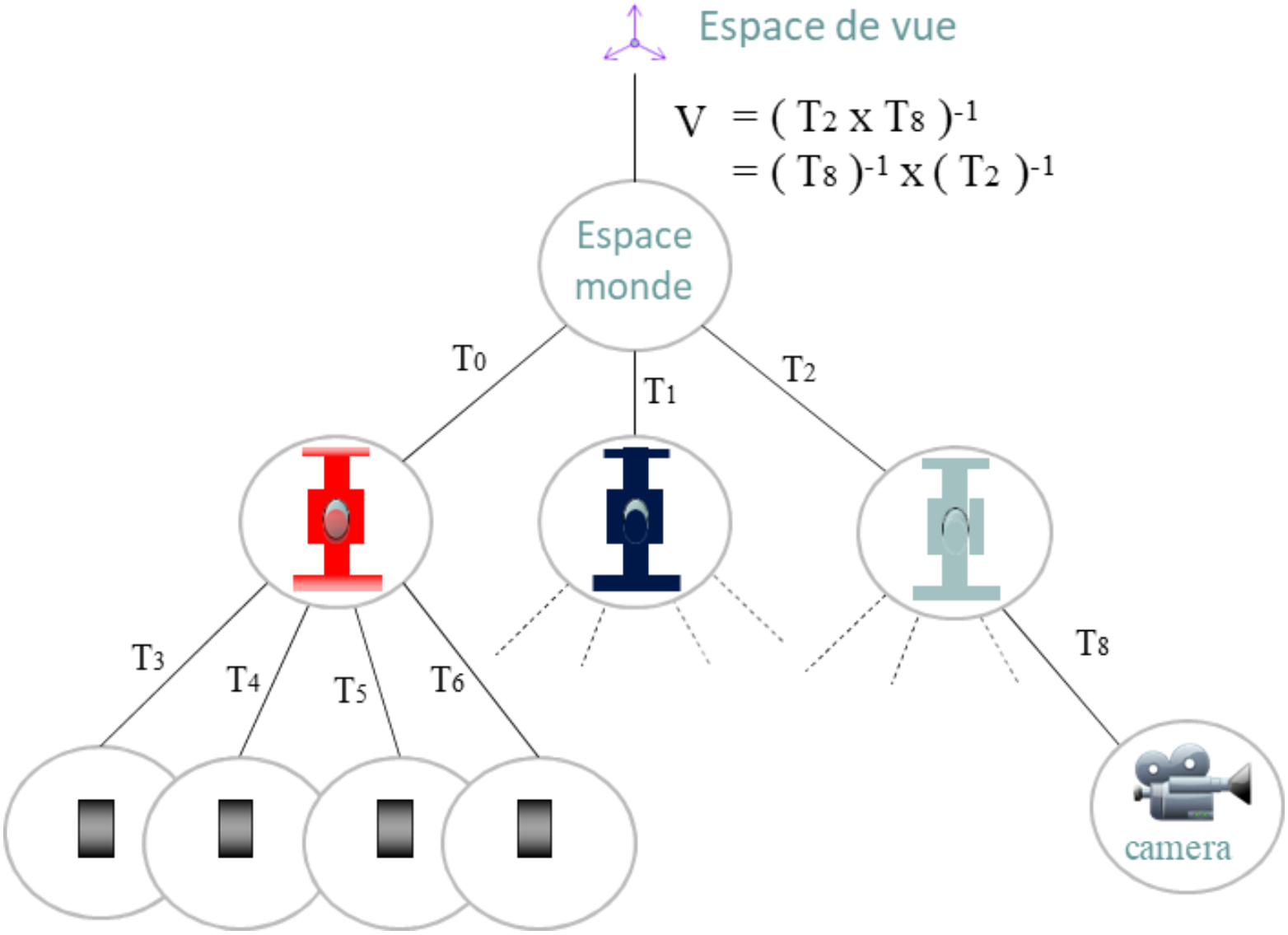
- L'«**espace caméra**» (espace objet de caméra) est spécial
  - aka Espace de vue
  - en CG, la transformation de vue V

  = **inverse de la transformation globale du nœud de la caméra**

  = de l'espace-monde à l'espace-vu

- V est utilisé dans le rendu pour déterminer où les objets se retrouvent à l'écran.
- Animations de la caméra = déplacer la caméra = changer V (e.g. par script)

Caméra dans le graphe de scène



Graphe de scène comme structure de données

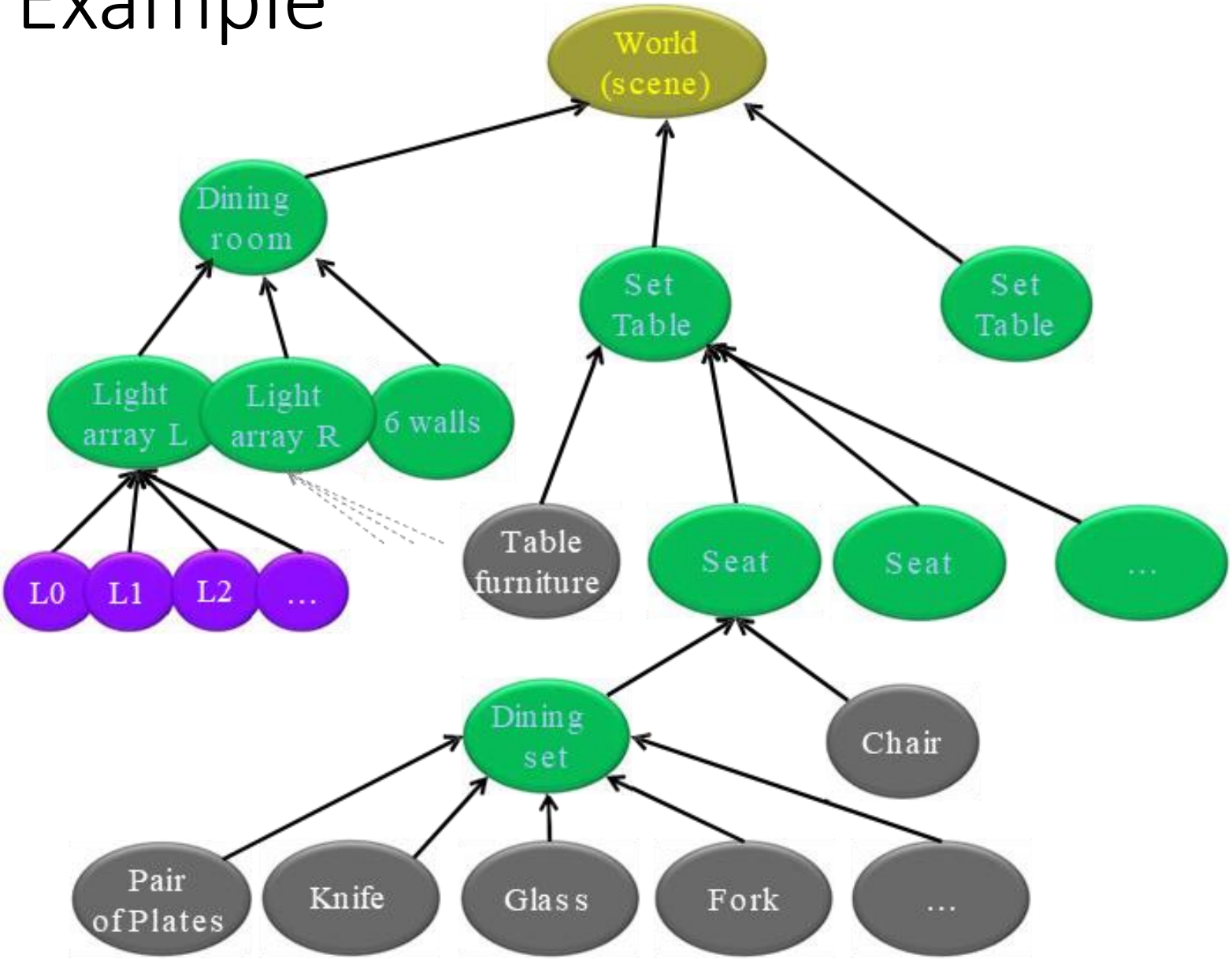
- Chaque moteur / bibliothèque adopte sa propre solution

- Pas de standards
  - certains formats de fichier existants peuvent inclure un graphique de scène (COLLADA)

- Concepts typiques:
  - la classe Node conserve la **transformation (locale) vers le nœud père** (et / ou les enfants, les sibilings…)
  - des liens (e.g. des pointeurs) vers des instances / assets
  - les **transformations globales sont calculées** à la demande
  - le nœud de la caméra est un nœud spécial
  - un mécanisme est nécessaire pour les **sous-arbres répétés**

Example :

Example


World (scene)
Dining room
Set Table
Set Table
Light array L
Light array R
6 walls
L0
L1
L2
…
Table furniture
Seat
Seat
…
Dining set
Chair
Pair of Plates
Knife
Glass
Fork
…

Example

Nœud d'un graphe de scène GameObjects & Transforms

- Un nœud = un **GameObject** avec

- un champ de **transformation** contenant
  - sa transformation locale
  - liens vers les parents, les enfants (et les frères et sœurs) - qui sont transformés

- un nombre quelconque de **«composants»** associés, représentant tout ce qui réside dans ce nœud, comme
  - Maillages (à afficher sur ces nœuds)
  - Caméras : la (ou les) active(s) produisant le rendu
  - « RigidBodies » : objets contrôlés par la physique
  - « Colliders »: les proxies géométriques utilisés pour les collisions
  - « Systèmes de particules » : (« émetteurs » de particules)
  - Producteurs / Récepteurs Sonores
  - Scripts…
  - fondamentalement n'importe quel atout (asset)

Enveloppes

Les données géométriques représentant peuvent être très complexes → représentation simplifiée pour maximiser les performances lors :

- des tests de visibilité
- des tests de collision

Approche basée sur un **volume englobant convexe** de la géométrie, permettant d'accélérer les traitements en réalisant certaines approximations

Enveloppes

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**

Enveloppes

- Les volumes englobant
  - exprimés dans le repère local de l'objet
  - afin de pouvoir réaliser les tests de collision,
  - être transformés dans un repère commun.

- Possible d'utiliser l'espace global pour les tests,

- Mais 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.

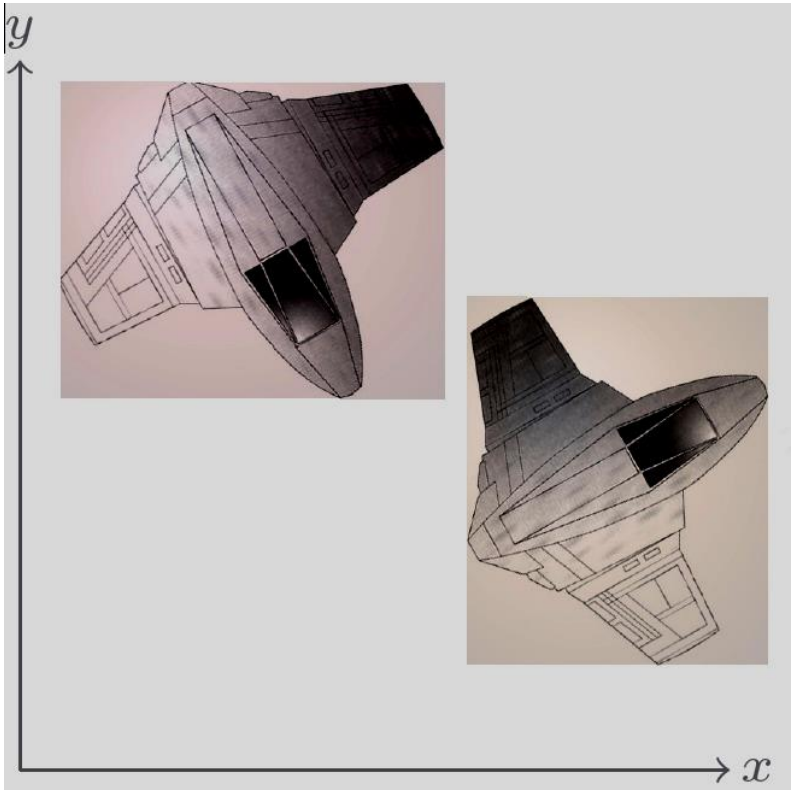
Boite AABB

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.

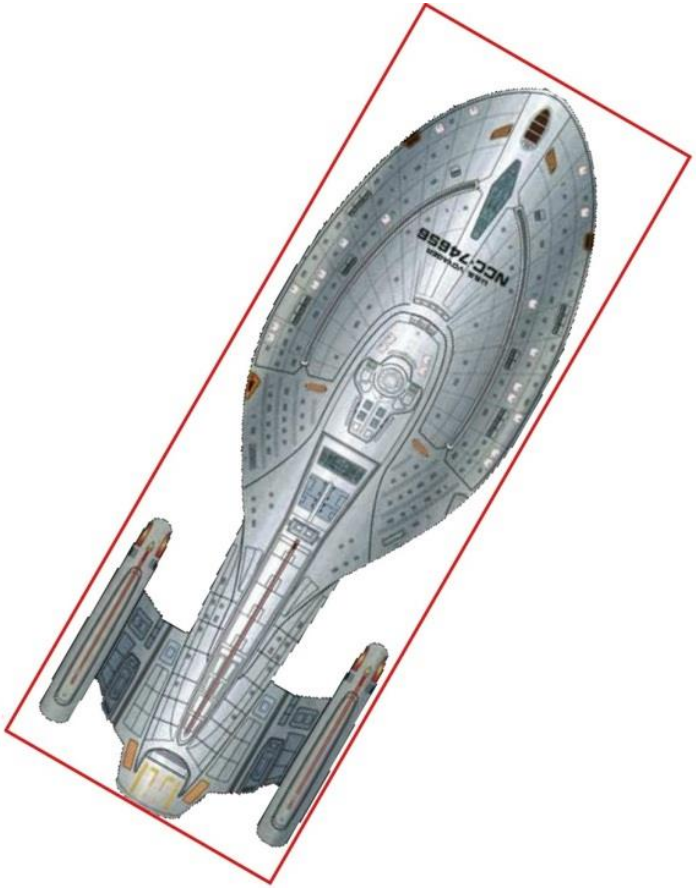


Boite **OOBB**

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 OOBB plus complexes



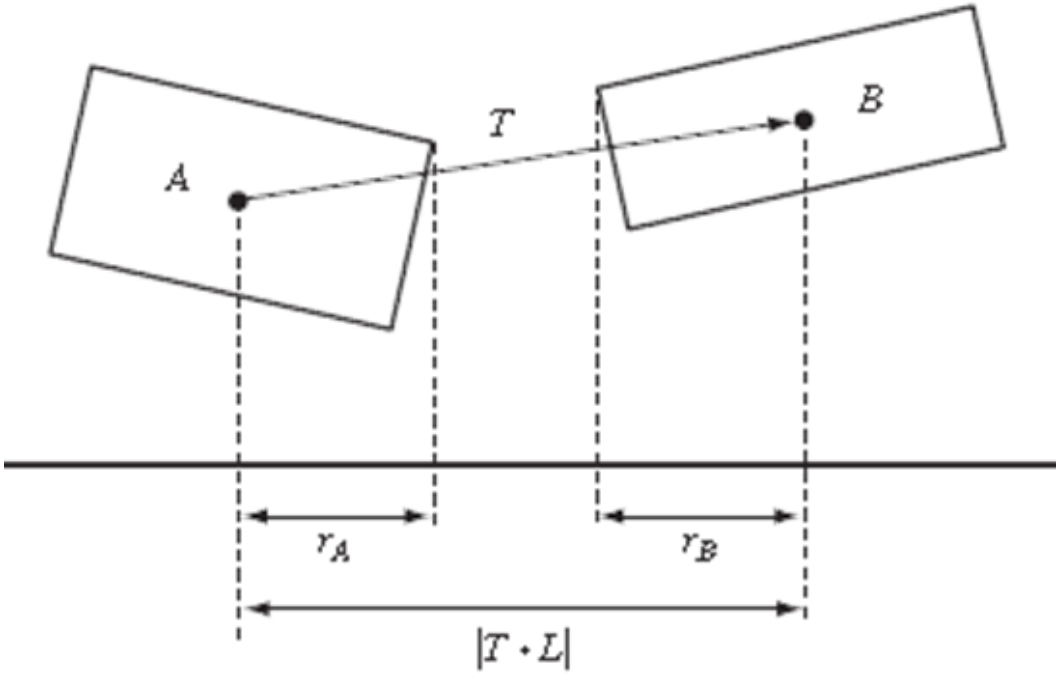
OBB

Intersections

- Pour tester l'intersection de deux OOBB, on utilise généralement un test d'*axe séparateur* :
- Les OOBB 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 OOBB projetés sur cet axe.

Au plus, 15 axes nécessitent d'être testés:

- Les 6 axes des deux OOBB (a_x, a_y , a_z , b_x , b_y , b_z)
- Les 9 axes perpendiculaires à chaque axe ($a_x \times b_x$, $a_x \times b_y$, $a_x \times b_z$, $a_y \times b_x$, $a_y \times b_y$, $a_y \times b_z$, $a_z \times b_x$, $a_z \times b_y$, $a_z \times b_z$)

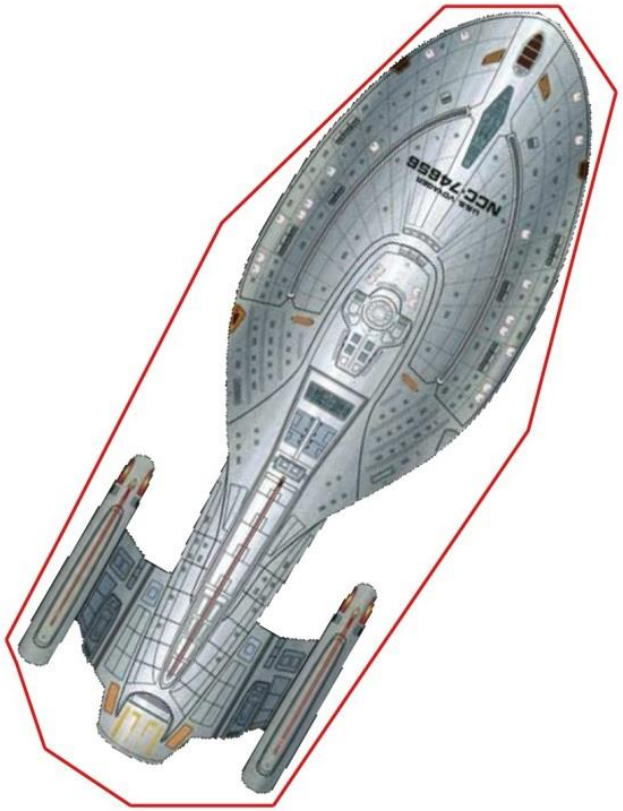


Discrete Oriented Polytopes (**K-DOP**)

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

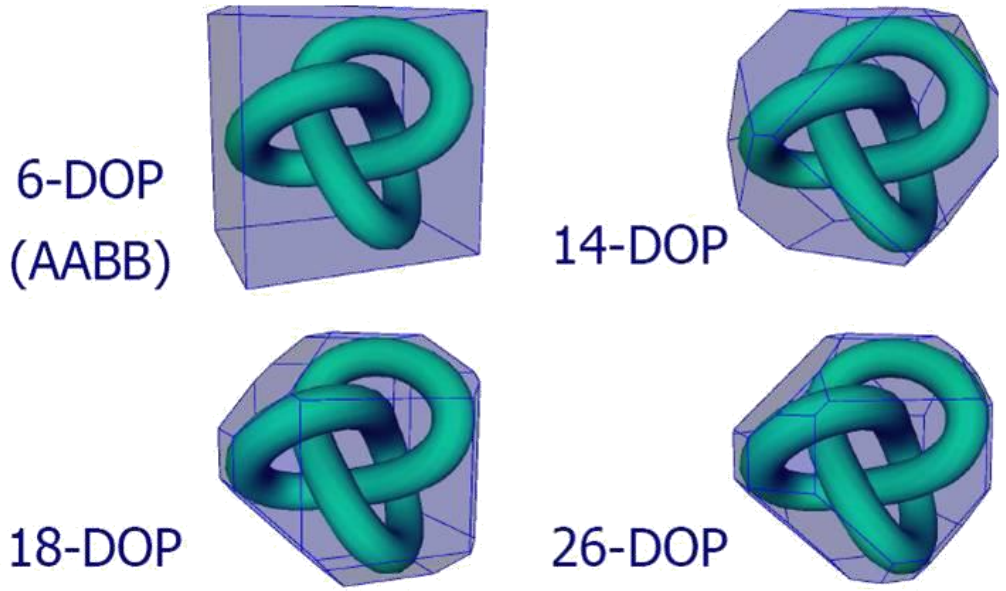
- Inconvénients
  - Mises à jour moyennement rapides



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

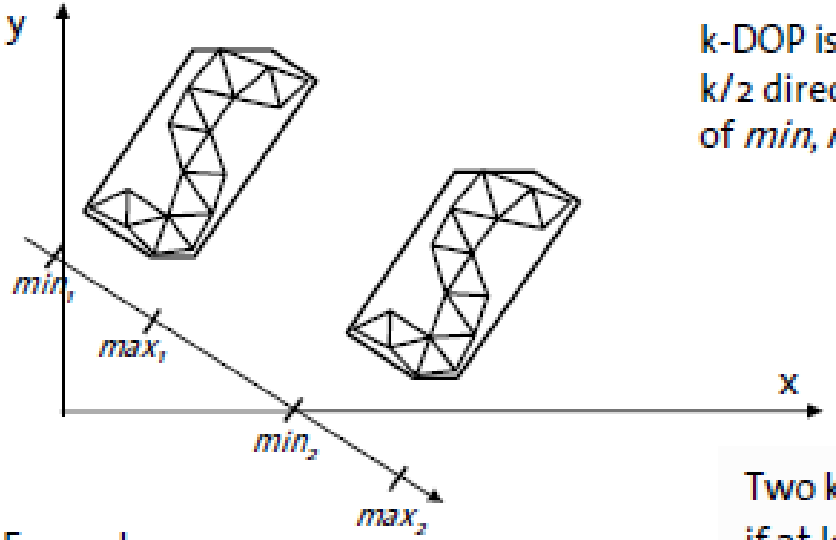


Intersections

- 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]



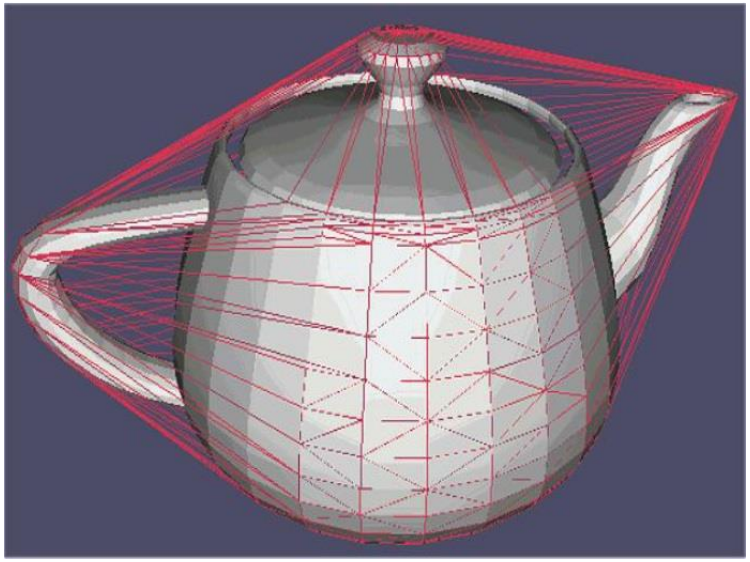
k-DOP is represented by k/2 directions and k/2 pairs of *min*, *max* values.

Examples:
6-, 14-, 18-, 26-DOPs

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)

Intersections

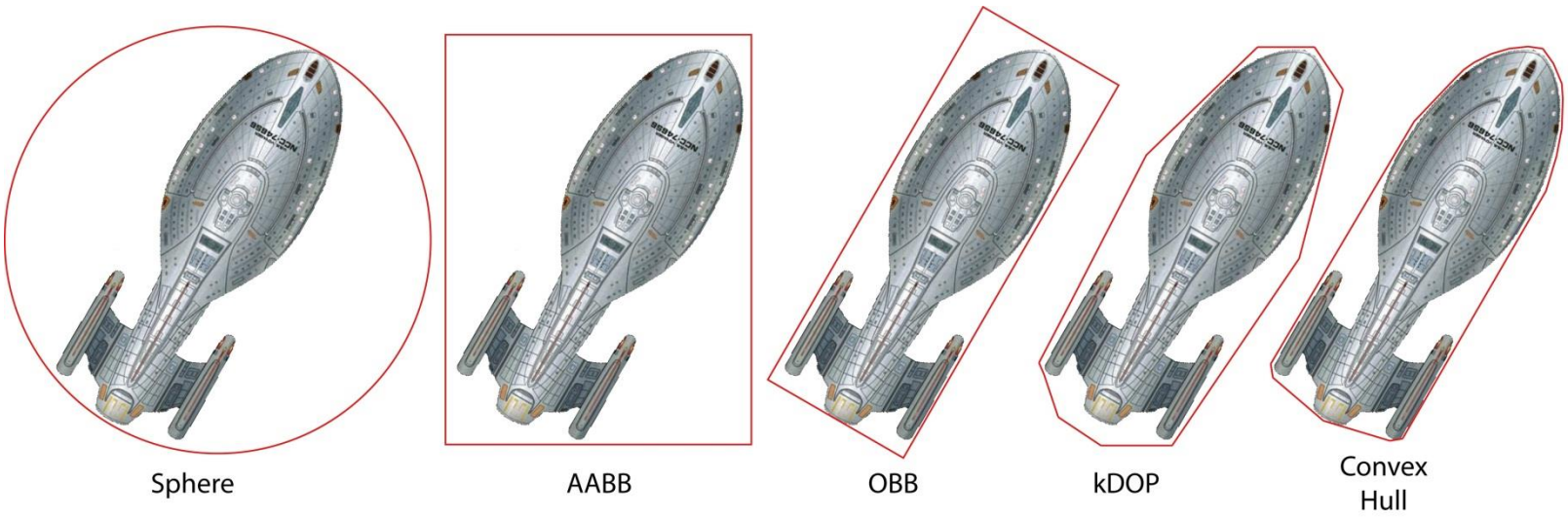
Stratégies de tests d'intersection : soient P et Q deux polyèdres convexes

- 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

Volumes englobants

Faible approximation du volume, faible consommation mémoire et tests rapides



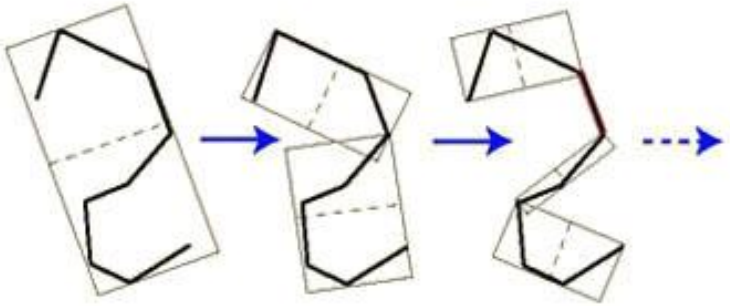
Meilleure approximation du volume, forte consommation mémoire et tests lents

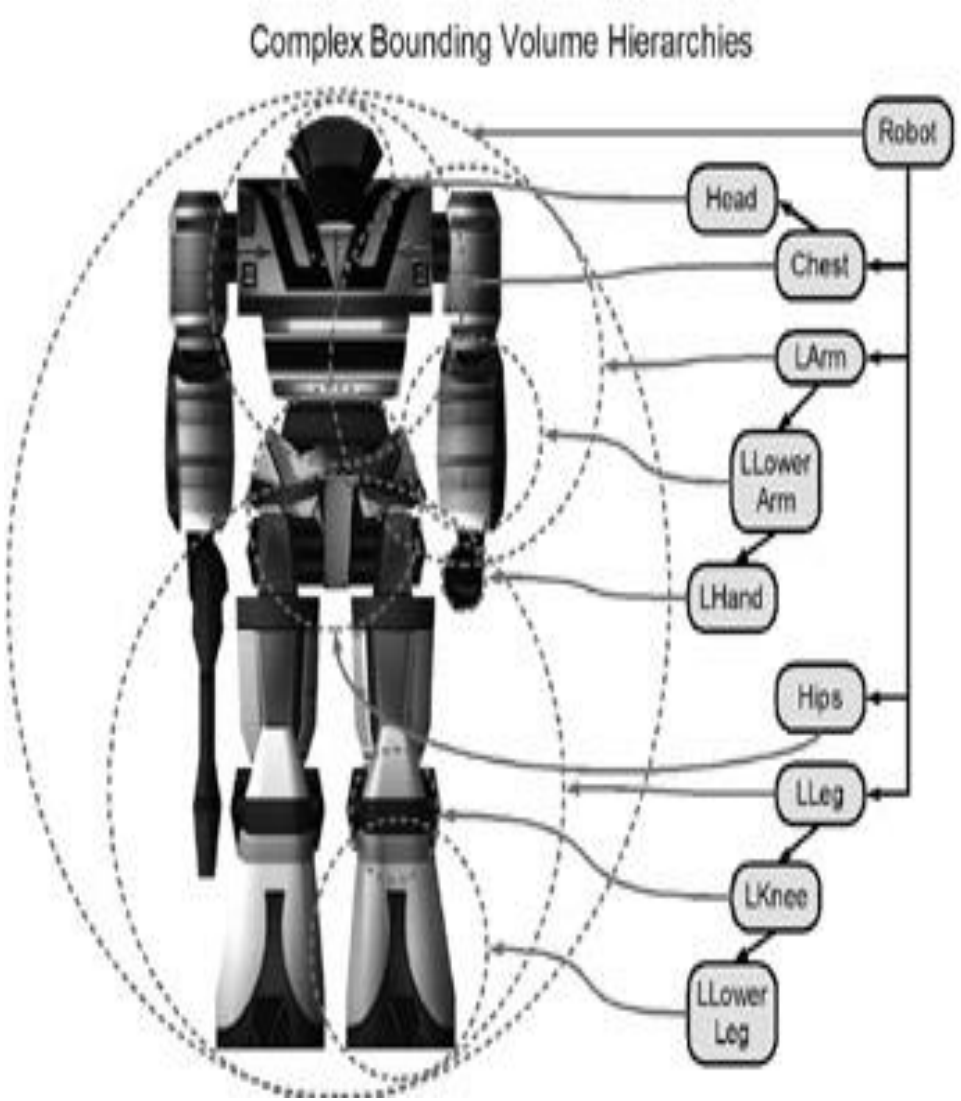
Volumes englobants

- 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 d'une hiérarchie.





Partitionner l'espace, dans quel but ?

La machine (grand publique) 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

« 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

Partitionner l'espace, dans quel but ?

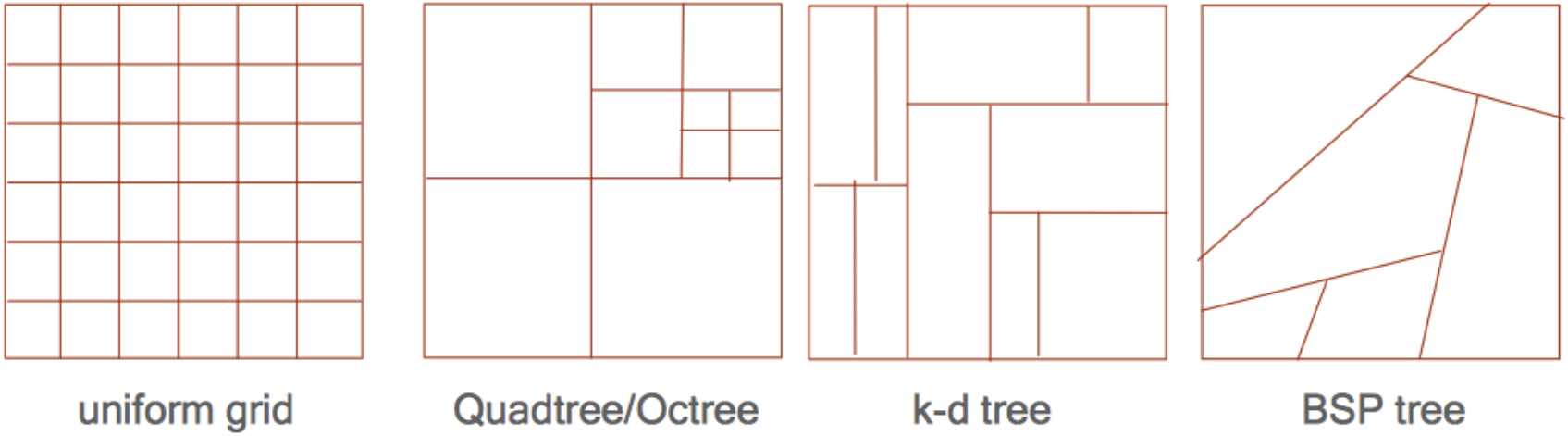
Partitionnement de l'espace :

- généralement une **structure de données hiérarchique**
- 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,…

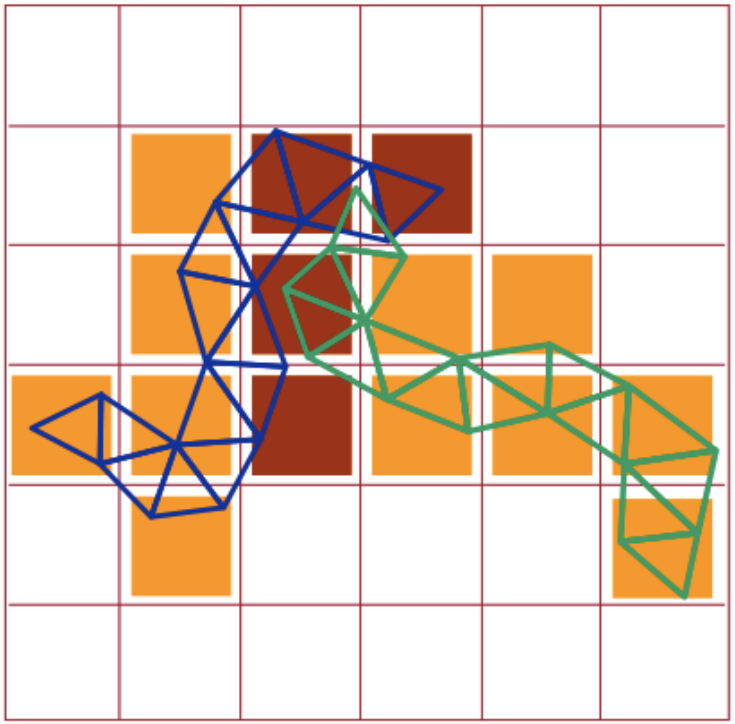
Partitionnement



Partitionnement

Principe de la sudivision

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



Quadtree et Octree

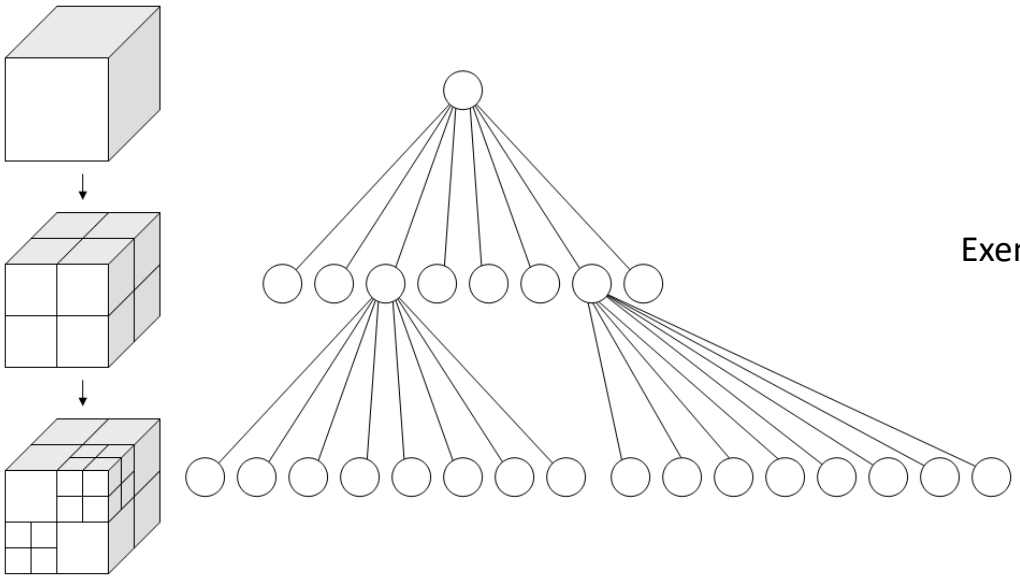
- Un **quadtree** (2D) ou **octree** (3D) est une structure de données hiérarchique de type arbre, qui subdivise l'espace selon des volumes englobants de type **AABB**.

- 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.

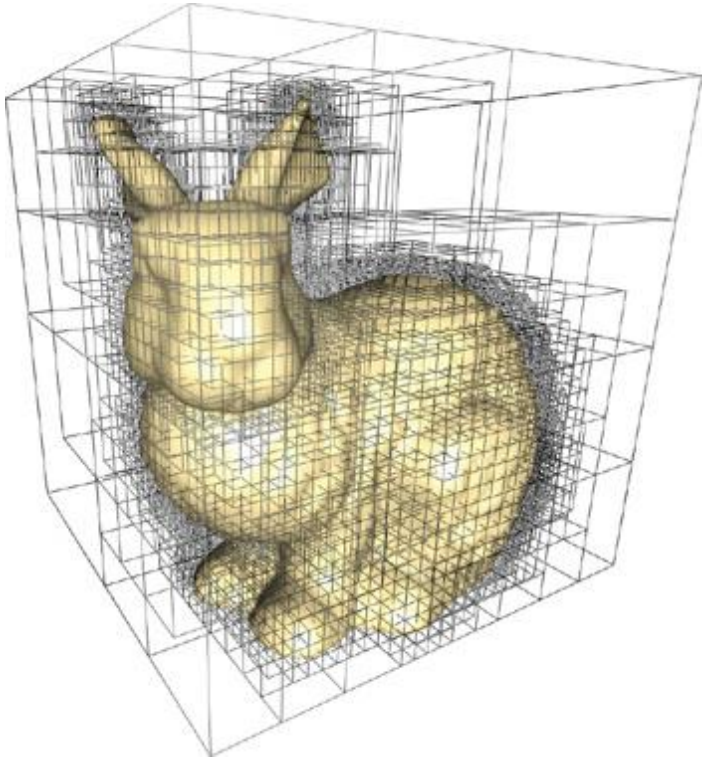
- Chaque nœud de l'arbre va contenir, en plus des informations de filiation, des références sur les géométries (ou toutes autres données utiles) contenues dans le volume du nœud.

Octree



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

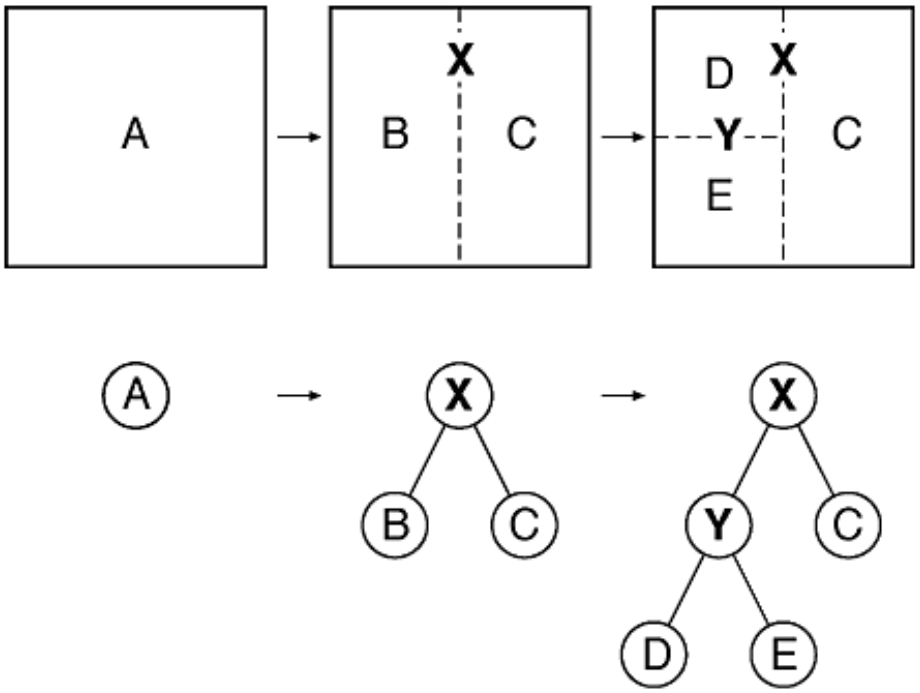
La méthode de sélection de l'hyperplan peut grandement varier, on retiendra cependant les contraintes suivantes pour décrire une méthode de construction canonique:

- 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.

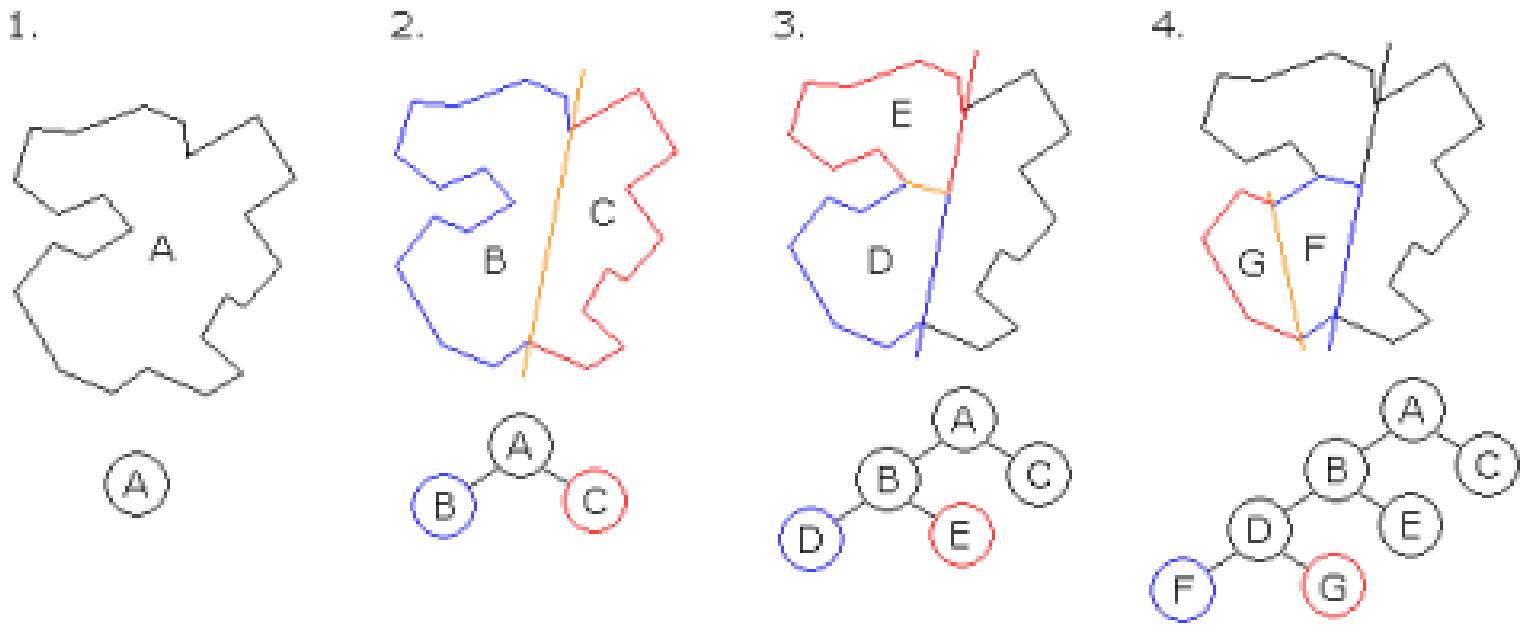


Construction d'un arbre BSP

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



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, 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**)

Optimisation

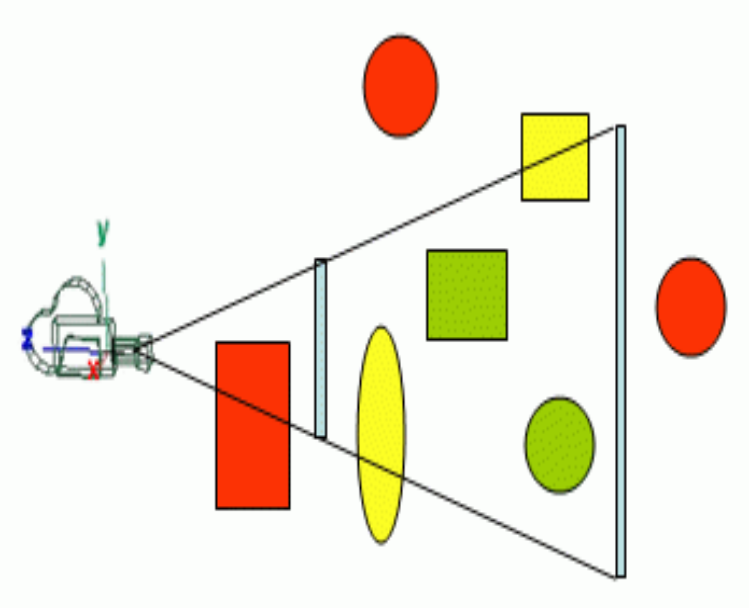
Les optimisations spatiales vont tirer parti des stratégies de subdivision spatiale mises en place pour représenter la scène (BSP, octree, …)

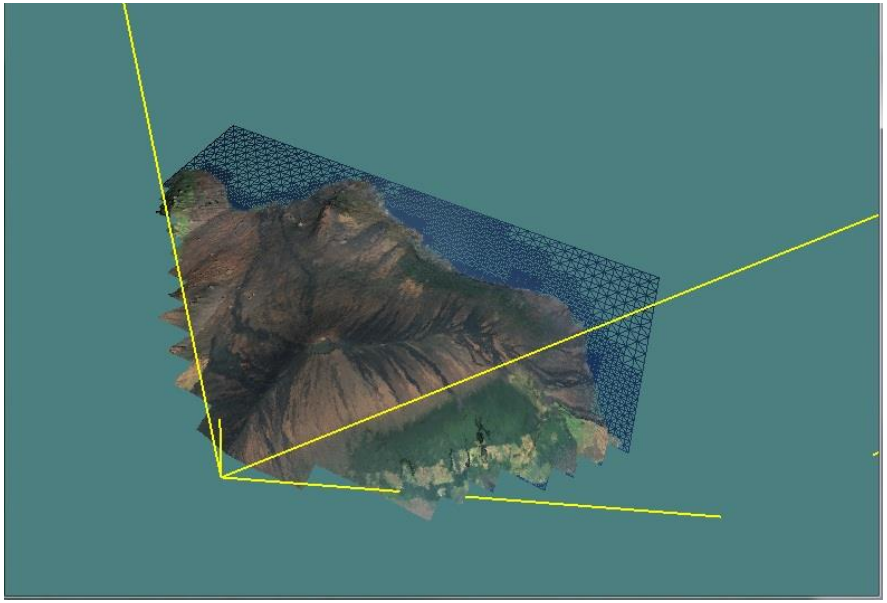
- Frustum (ou geometry) culling
- PVS (Potentially Visible Set ou Potential Visibility Set)
- Occlusion culling

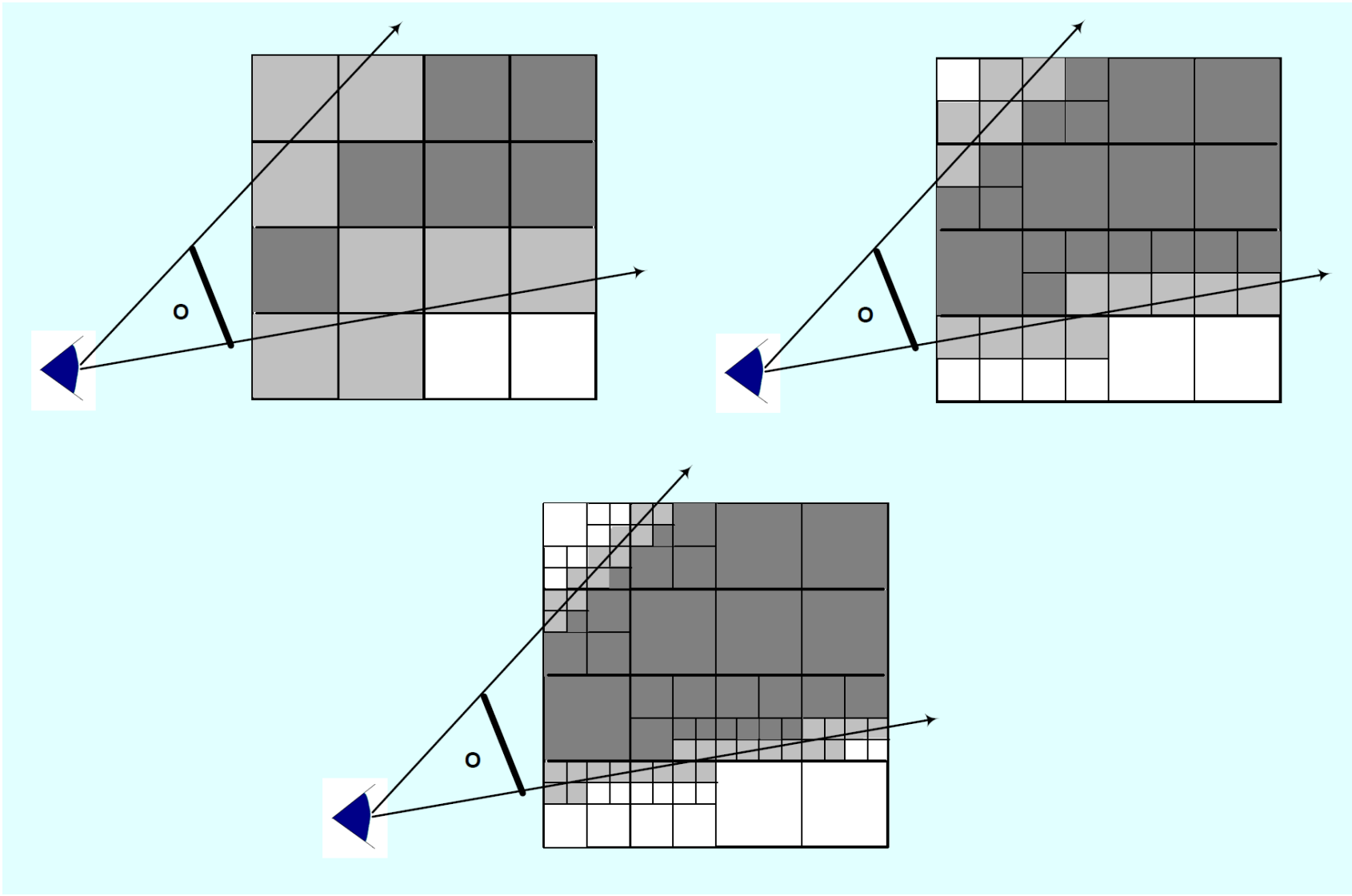
Ref: Game Engine Architecture by Jason Gregory

Frustrum culling

- Ne pas afficher ce qui est **en dehors du champ de vue** (frustum) de la caméra.
- Test est en général réalisé à partir des **volumes englobants** des objets :
  - si tous les points du volume englobant sont du coté 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.
- Principe est applicable aux **objets de la scène** (trivial) et **la scène** elle-même (un peu moins trivial selon les cas), si découpée par une stratégie de subdivision spatiale (BSP, octree, etc).



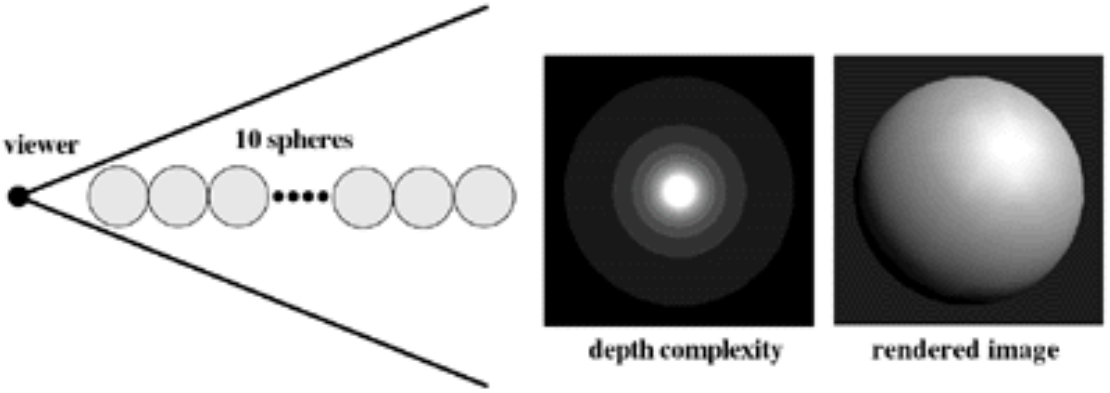




Démonstration de l’intérêt d’un test de visibilité récursif sur une structure spatiale de type octree.

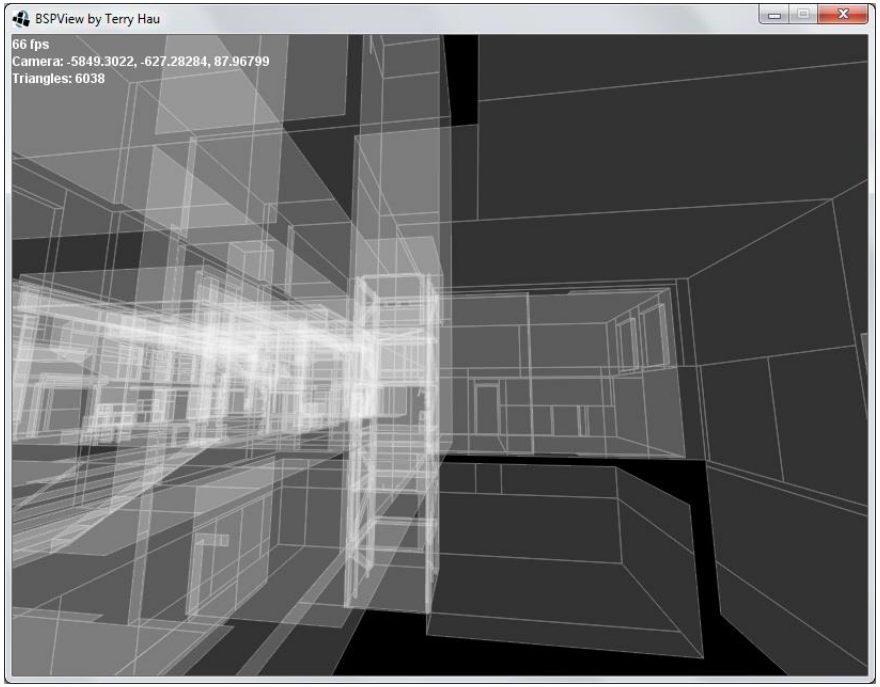
Potentially visible set (PVS)

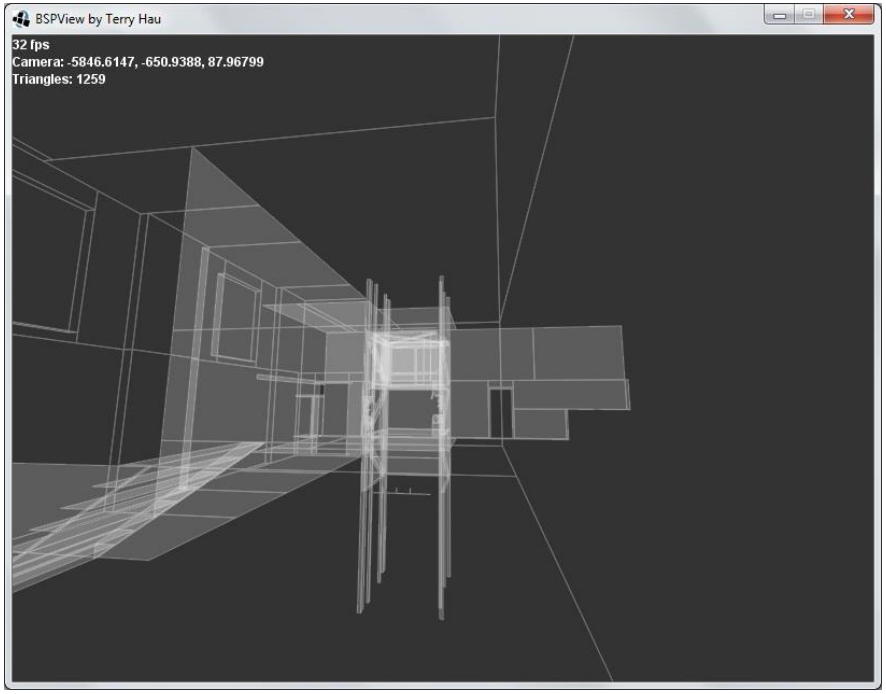
- Pour les scènes contenant un **grand nombre d'occludeurs potentiels** (en général scènes d'intérieur), générer un PVS, dont le but est de définir les zones de la scène qui seront visibles, à partir de n'importe quelle position.
- On simplifie le problème en **stockant pour chaque zone de la scène la liste des zones visibles**.
- 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 ».



Depth complexity.
Lorsque la scène est rendue en back to front, certains pixels sont rendus plusieurs fois.

Depth complexity.



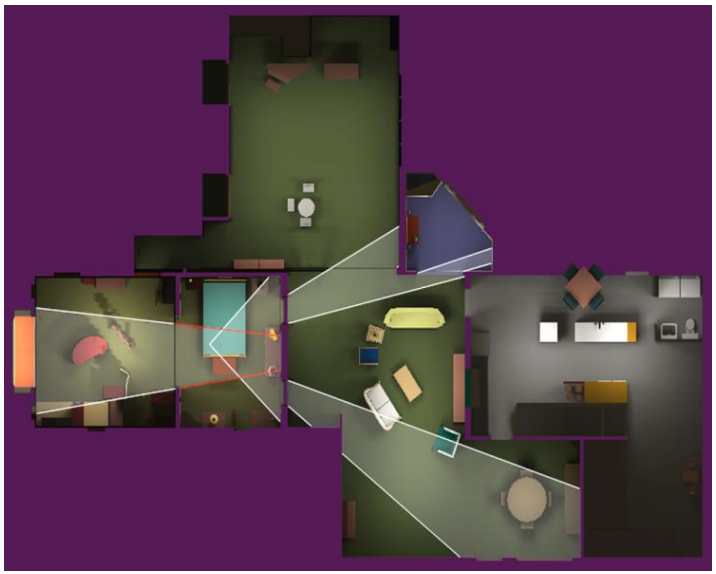


Affichage de la depth complexity d'une scène:

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

Portals

- Une approche classique de construction automatique d'un PVS se base sur le mécanisme des « portals ».
- Un **portal** est un **ensemble de polygones (invisibles)**, permettant 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).

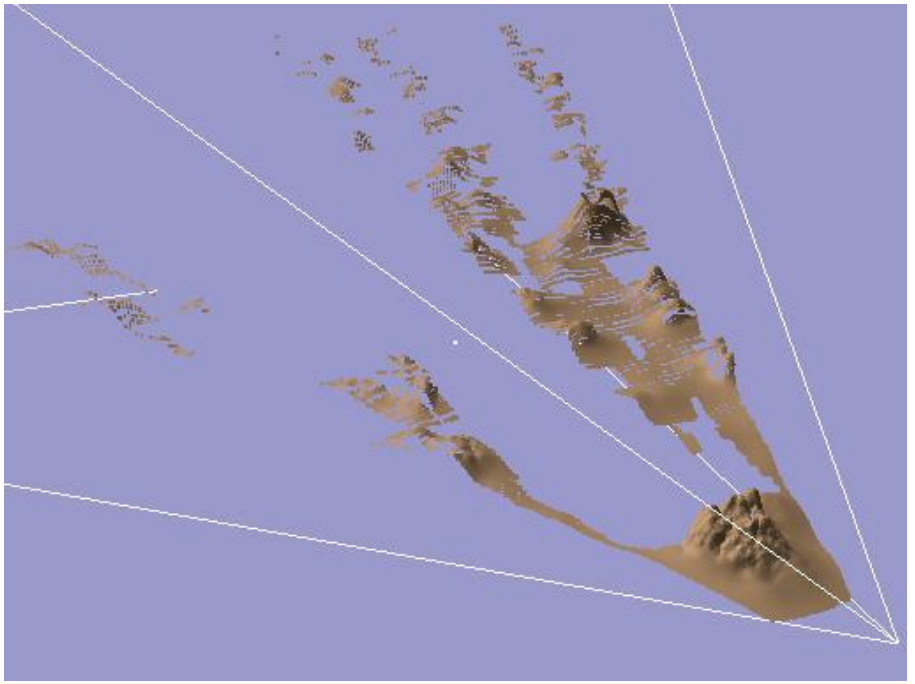


Occlusion culling

Réduire la quantité de primitives rendues, en identifiant les objets **occludeurs** : masquant de grosses portions de la scène.

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

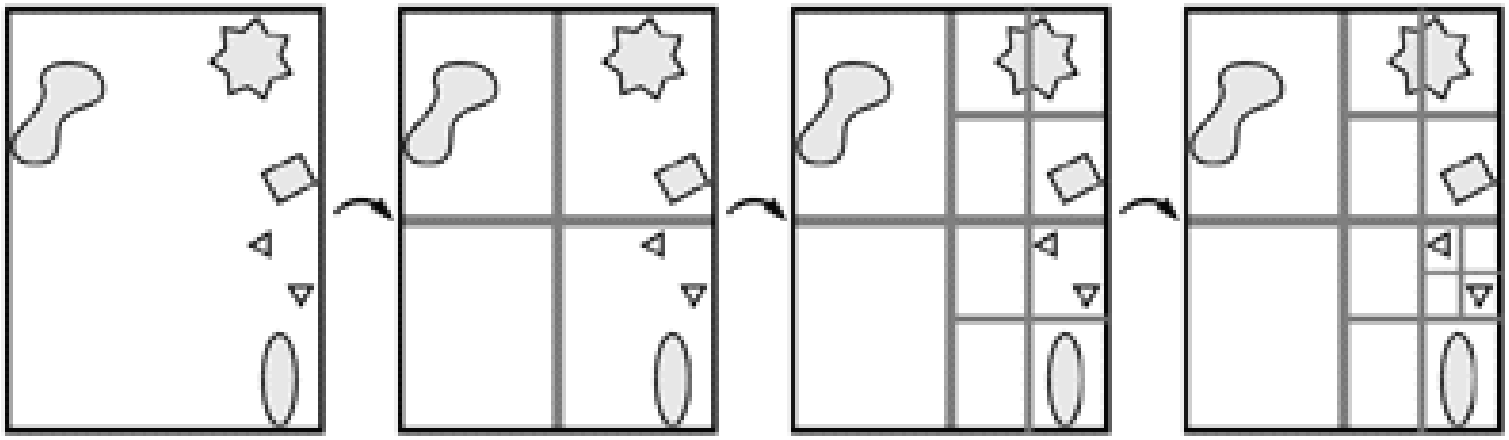
- Hierarchical Z-buffer Visibility
- Hierarchical Occlusion Maps



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.



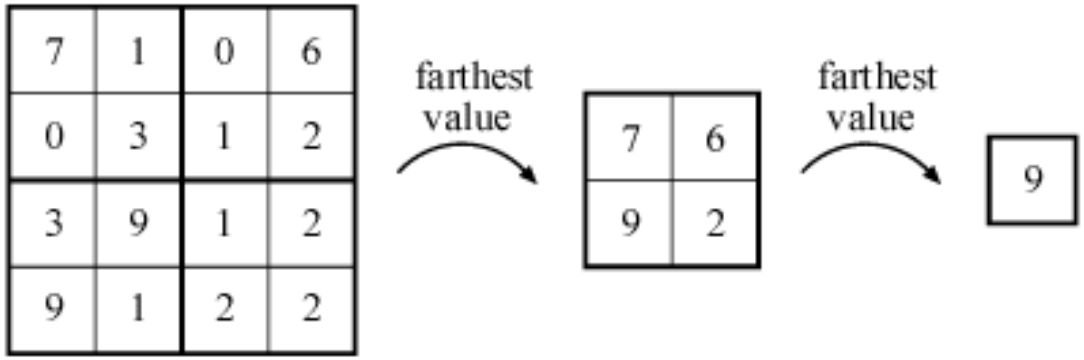
L'algorithme exploite le fait que, si **aucune des faces de la boîte** associée à un nœud de l'octree **n'est visible relativement au Z-buffer**, alors aucune des géométries associées à ce **nœud n'est visible**.

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.

Hierarchical z-buffer visibility

Tester de visibilité des faces de la boîte d'un nœud :

- **Pyramide d'images du Z-buffer**
- Chaque niveau : dimension de moitié celle du niveau supérieur
- Chaque groupe de 2x2 pixels = max des 4 valeurs de profondeur.
- Niveau le plus bas de la pyramide 1 seul pixel = valeur la plus éloignée dans le Z-buffer.

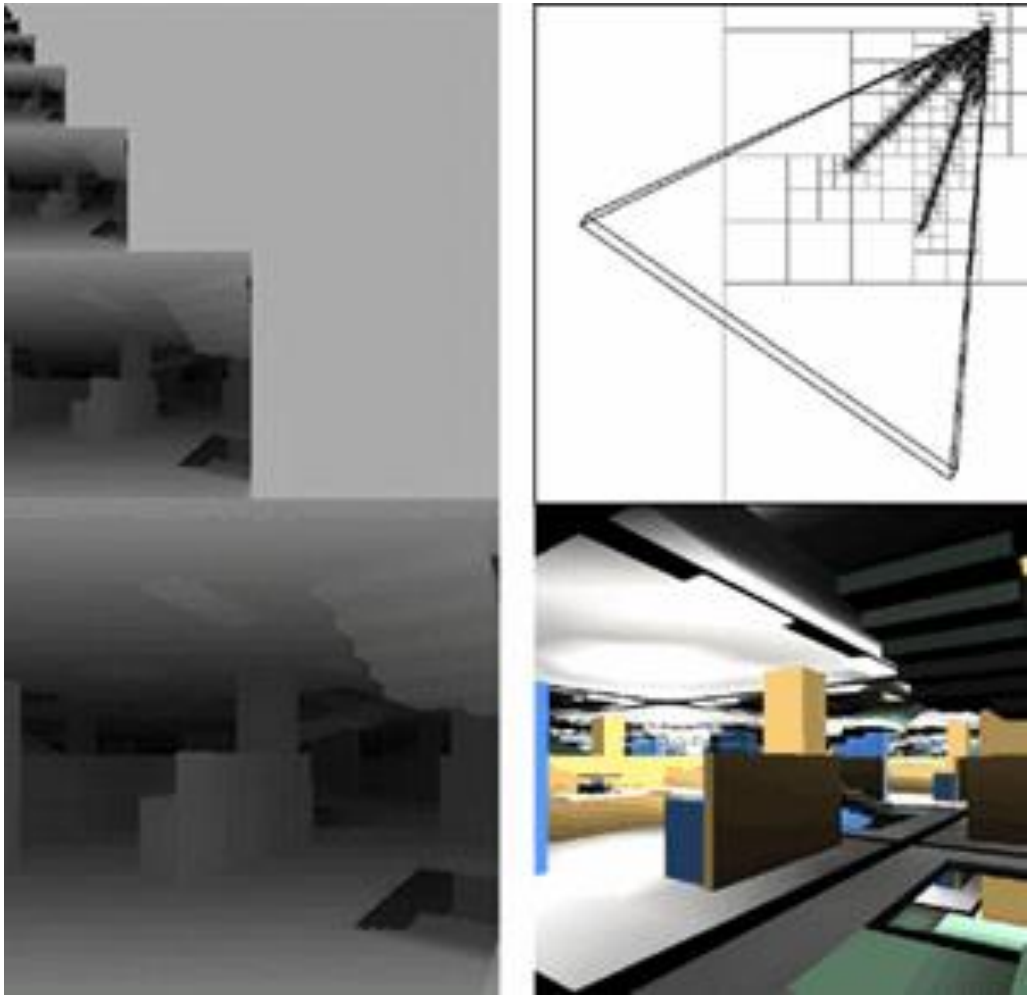


En pratique, pyramide Z non reconstruite chaque mise à jour du Z-buffer.
→ la cohérence dans l'espace image : générée à partir des objets marqués comme visibles dans l'image précédente.

Hierarchical z-buffer visibility

Pour rejeter rapidement les faces de la boîte englobante :

- chercher 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**
- comparer la valeur du Z-buffer avec la plus petite profondeur de la face.
- Si profondeur de la face > à la profondeur stockée → la face est masquée.



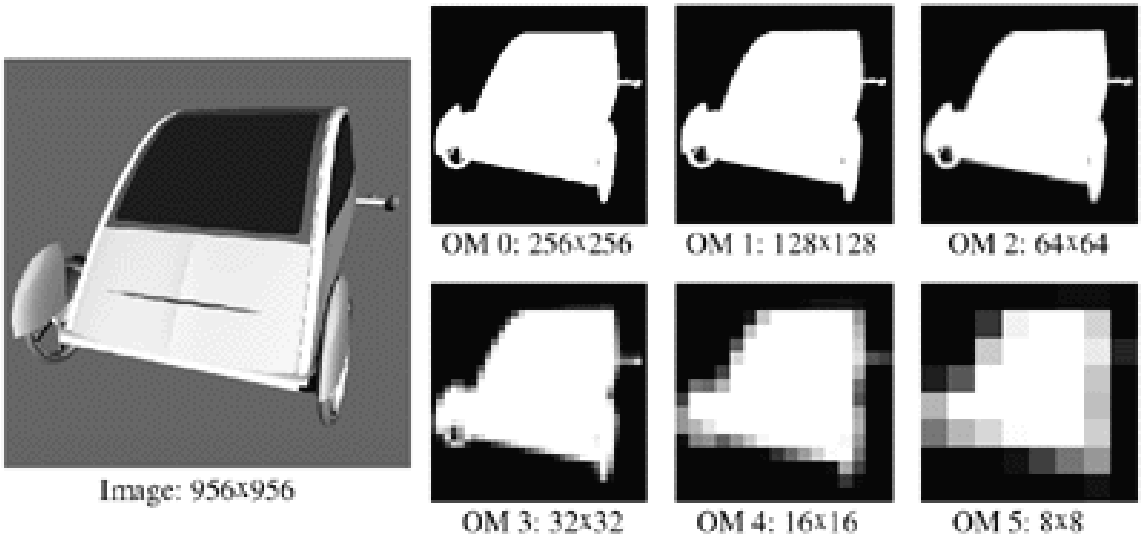
A gauche, pyramide du Z-Buffer. En haut à droite, champ visuel de la caméra représenté dans l'octree de la scène. En bas à droite, rendu de la scène.

Hierarchical occlusion maps

Similaire à « Hierarchical Z buffer » avec une pyramide d'images contenant les **occludeurs**

→ 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

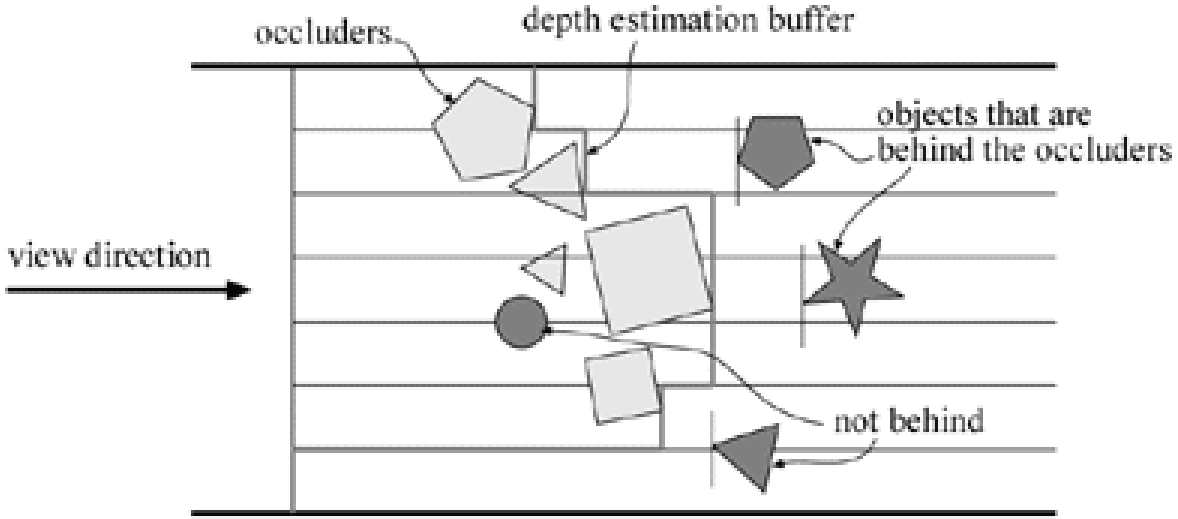


Hierarchical occlusion maps

Puis les autres objets sont rendus :

- **projection image de leur boîte englobante** 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.

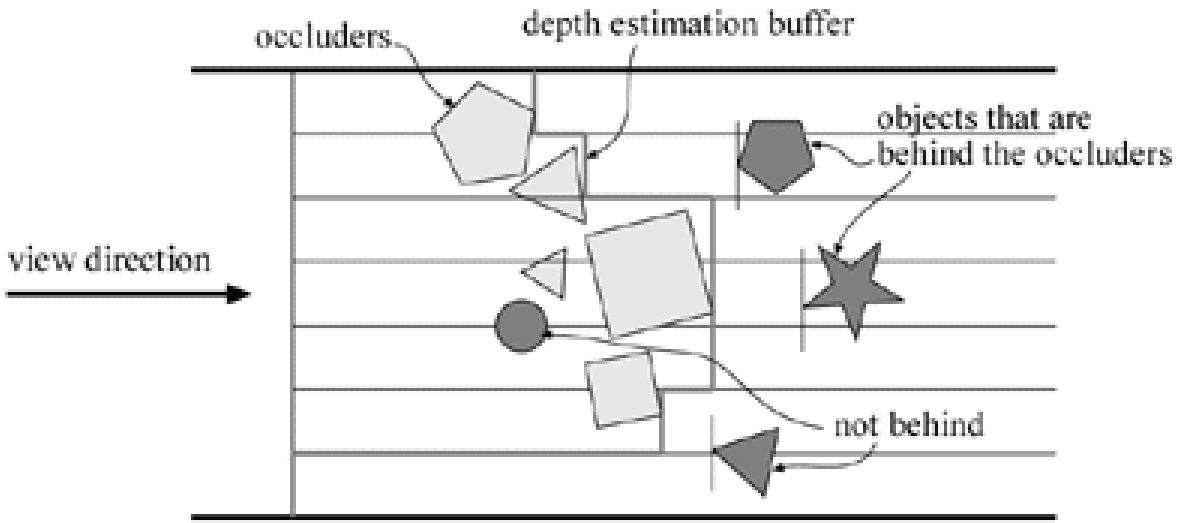
Z-buffer standard, ou image d'estimation de la profondeur (depth estimation buffer) pour déterminer la profondeur de l'objet par rapport aux occludeurs.



Hierarchical occlusion maps

Buffer d'estimation de la profondeur :

- construit en calculant la plus grande profondeur pour chacune des boîtes englobantes projetées des occludeurs.
- comparaison réalisée pendant le rendu avec la plus faible profondeur des boîtes englobantes projetées des objets de la scène



Optimisation

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)

Observations

- Objets détailles ! nombre de polygones élevés
- coût de rendu = f(nombre de polygones)
- tailles des polygones à l'écran = f(distance a la camera/observateur)
- Objet lointain ! nombreux polygone se projetant sur le même pixel

Idée

- Adapter la résolution (nombre de polygones) du maillage en fonction du
- point de vue

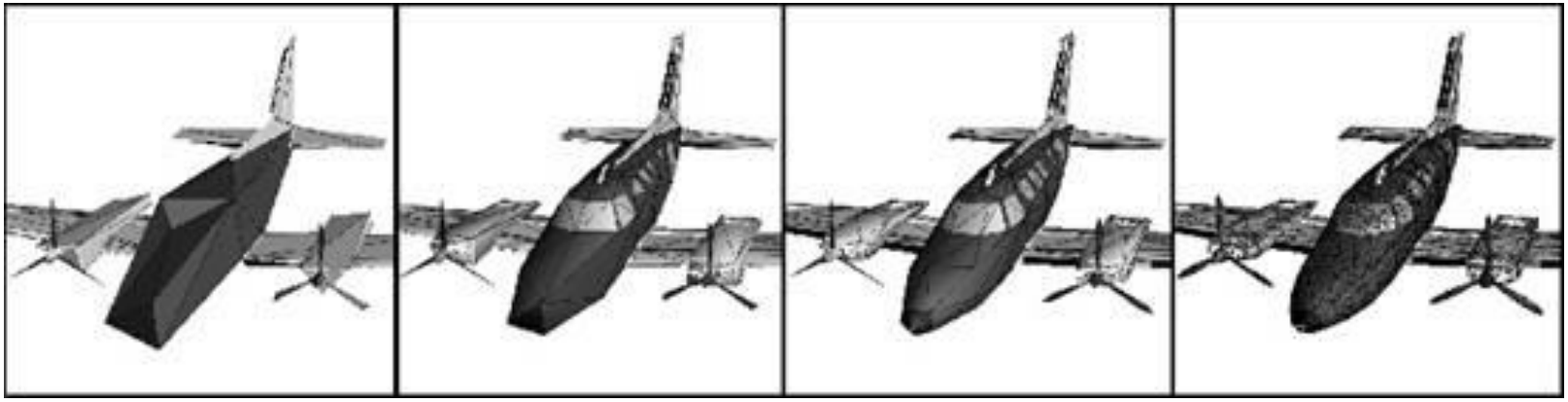
Défis

- Comment calculer des versions simplifiées du maillage ?
- Comment choisir la résolution adaptée ?
- Comment rendre cela efficace du point de vue du GPU ?

Level of details (LOD)

Utiliser un modèle géométrique simplifié lorsque loin de la caméra :

- besoin de moins de détails car plus petits à l'écran
- couvrant moins de pixels, les triangles de surface projetée inférieure au pixel ne contribuent pas à l'image finale.



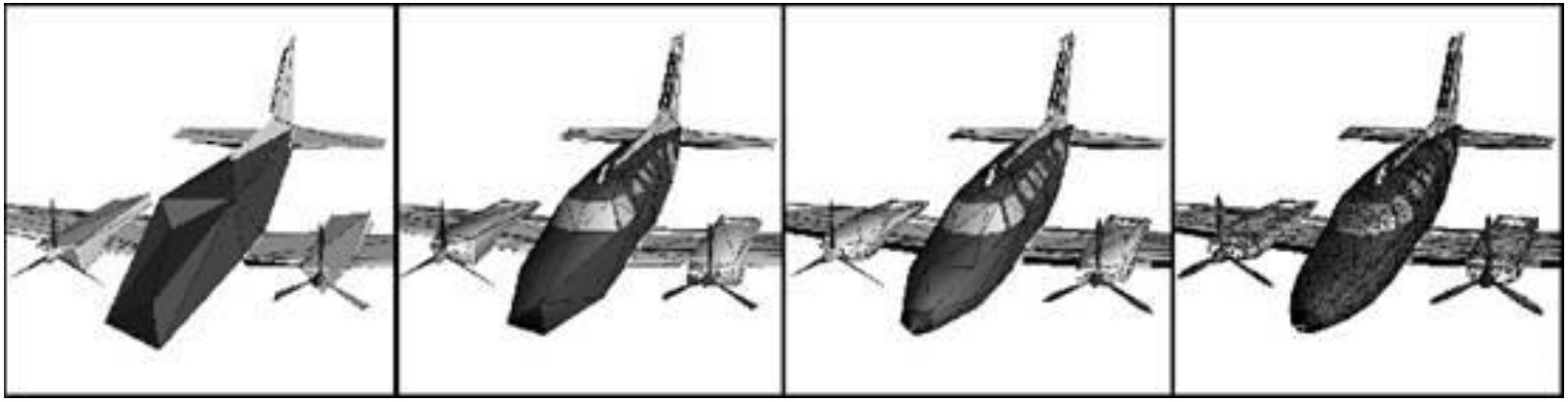
Level of details (LOD)

Plusieurs niveaux de LOD sont possibles pour un même objet :

- **transitions moins brutales**
- **distance à la caméra** critère utilisé pour sélectionner le niveau à afficher.

Mise en place de gestion de LOD :

- bonnes distances de transition basée sur la surface en pixels à l'écran de l'objet (idem pour les textures)
- chaque objet aura probablement des distances de transition différentes.



Level of details (LOD)

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

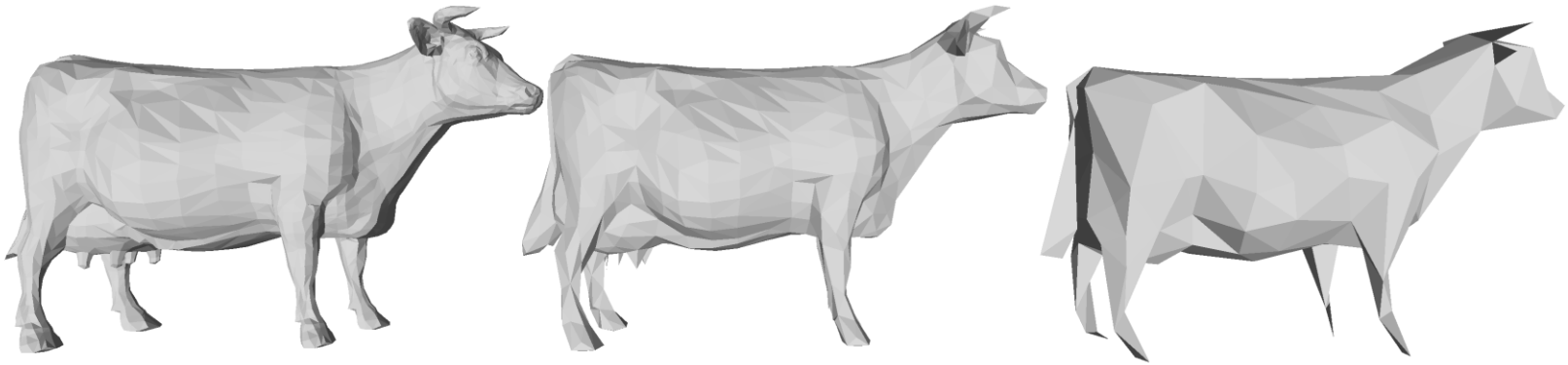
- Génération des niveaux de LOD (automatique ou manuelle : généralement meilleur résultat)
  - créer le niveau de définition maximale et supprimer progressivement les détails.
  - simplification peut être dépendante ou indépendante de la vue.

- Sélection du niveau de LOD à afficher avec une fonction d'estimation servant à sélectionner le bon niveau de LOD :
  - distance à la caméra : on définit n seuils de transition
  - projection dans l'image du volume englobant de l'objet et estimation de la surface couverte

- Implémentation de la méthode de transition entre les différents niveaux de détail

LOD discret

- Précalcule de différentes représentations du même objet
- Variation du niveau de détail
- Sélection à l’execution d'une représentation



LOD discret

Précalculer les niveaux de détails

- Surface de subdivision : trivial !
- Maillage quelconque : simplification de maillage

Critères

- Minimiser la distance entre le maillage simplifié et le maillage original
- Distance de Hausdorff
- Heuristiques...

Simplification par edge collapse

Calculer une erreur pour chaque arête

- Représente l'erreur introduite par la contraction de l’arête
- Tant que le nombre de triangles > seuil
- Contracter l’arête avec l'erreur minimale
- Mettre à jour les erreurs des arêtes voisines

Améliorations

- Contracter plusieurs arêtes en même temps
- Passe de edge-flip pour régulariser le maillage
- Prise en compte de différents critères dans le calcul de l'erreur (géométrie, normales, texture, forme des triangles, etc.)

LOD continu

Adapter la résolution

- Modification de la géométrie au cours de l'execution de l'application

Approche "top-down"

- Suppression des triangles un à un (très difficile !!)

Approche "bottom-up"

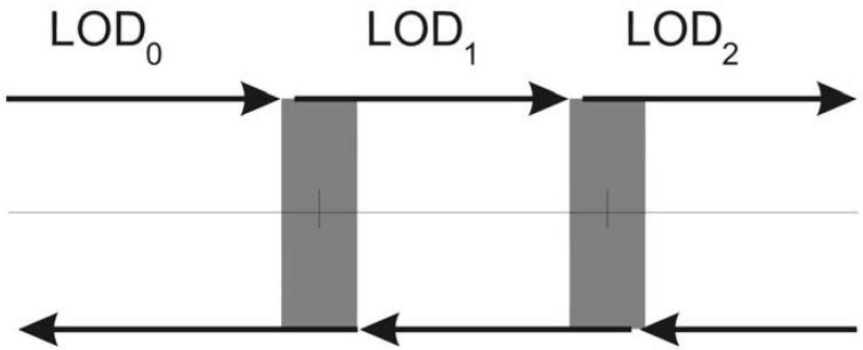
- Partir d'une surface simplifiée
- Insertion de détails stockes dans une représentation multi-résolution

Exemples

- Surface de subdivision
- progressive mesh
- quadtree pour les terrains

Level of details (LOD)

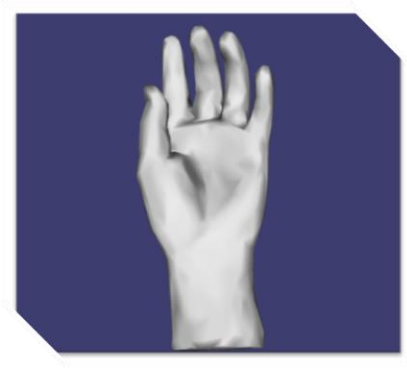
- Sélection directe du niveau. Problème : effet de « popping » et d'instabilité aux seuils de transition → **seuillage par hystérésis**



- LOD blending: 2 niveaux de LOD affichés en semi-transparence sur une zone de recouvrement (autour du seuil de transition).

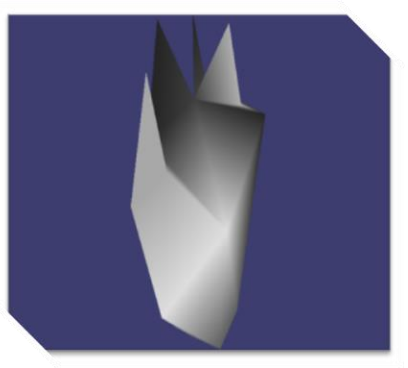
+ transition relativement fluide

- artefacts de transparence et affichage des 2 niveaux en même temps

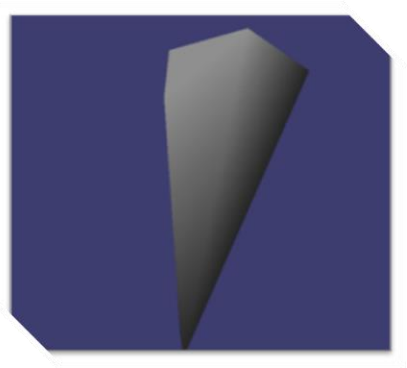


2000 Vertices
4000 Faces

52 Vertices
100 Faces

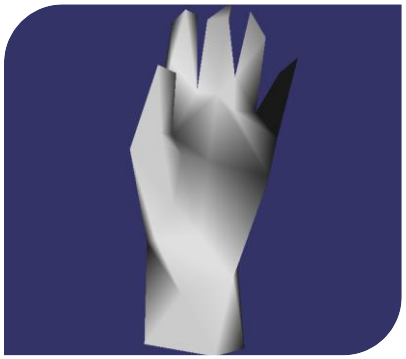


17 Vertices
30 Faces

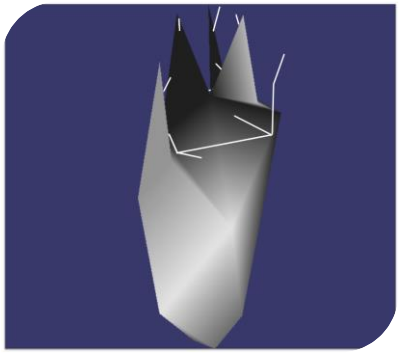


6 Vertices
8 Faces

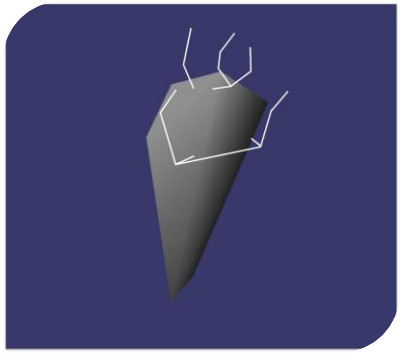
2000 Vertices
4000 Faces



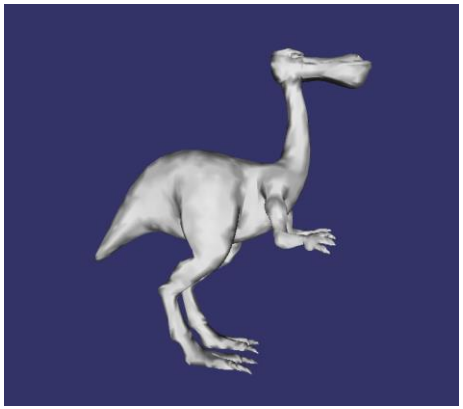
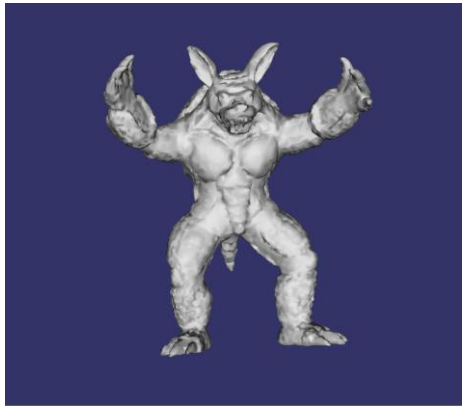
52 Vertices
100 Faces

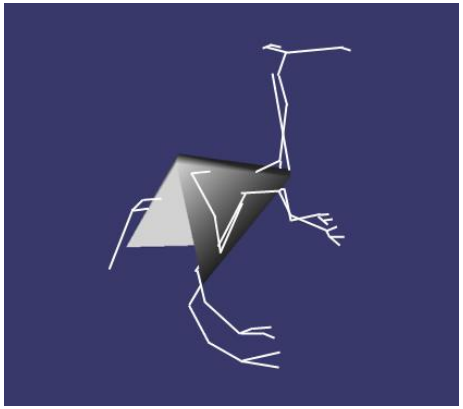


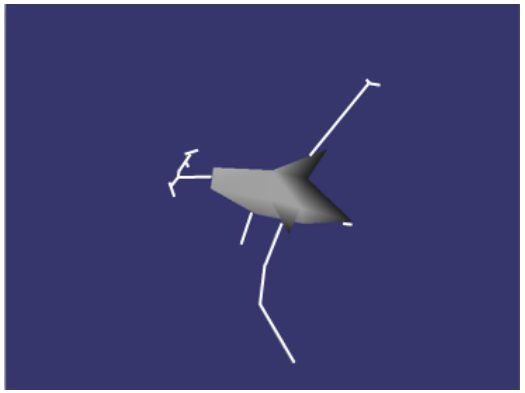
41 Vertices
30 Faces



30 Vertices
8 Faces



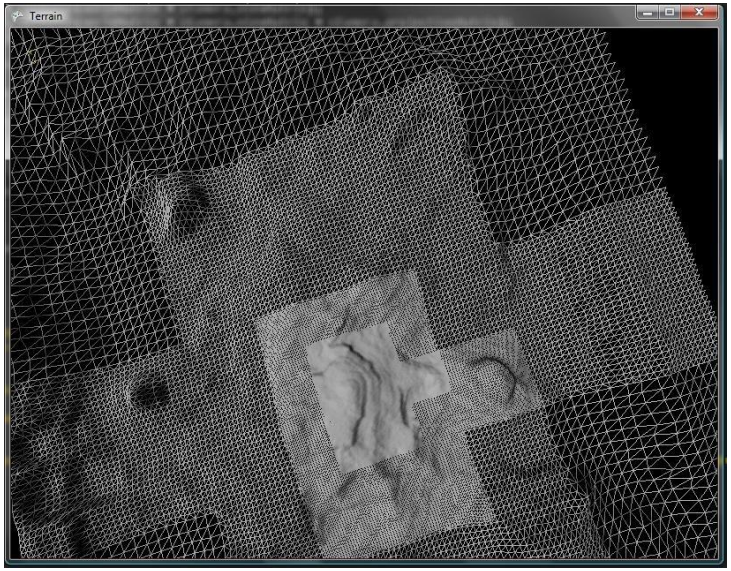


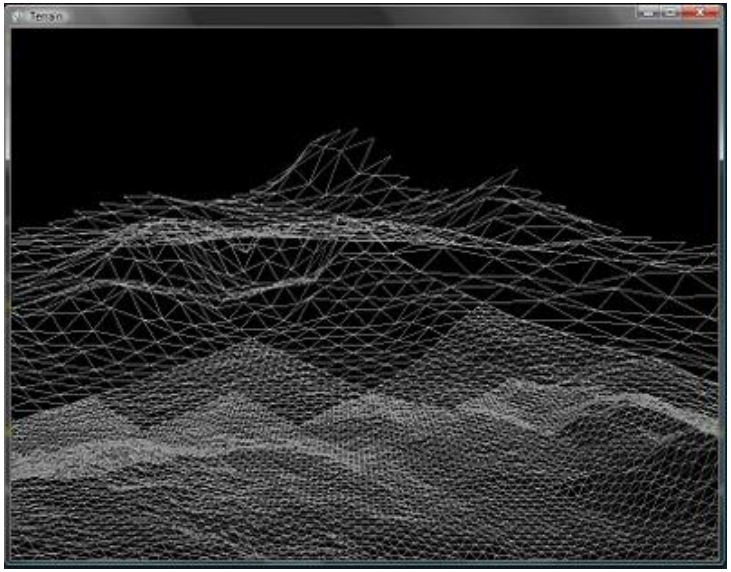
Level of details (LOD)

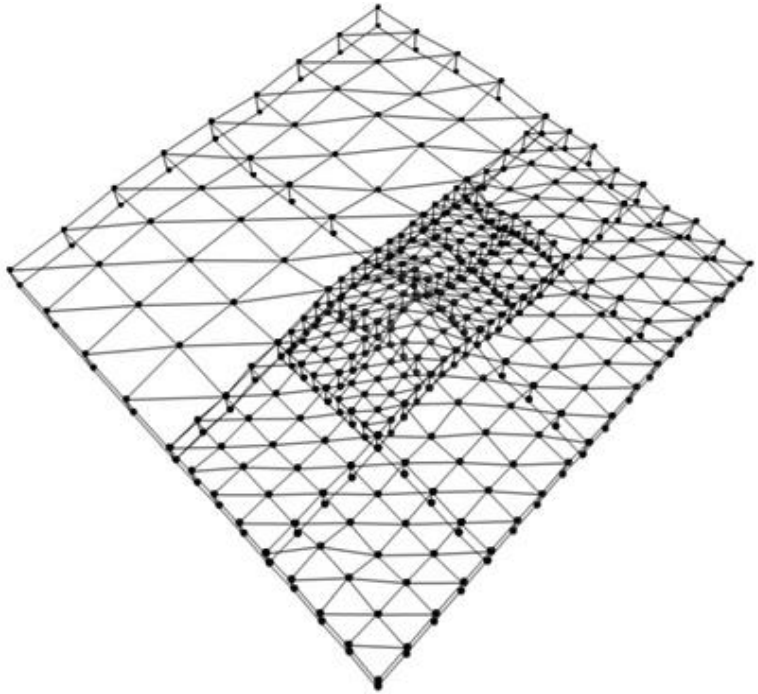
- Alpha LOD :
  - utilise plus qu'un seul niveau de géométrie,
  - faire disparaître progressivement par transparence les zones les moins importantes.
  - plus tout à fait du LOD !

- LOD continu (CLOD – Continuous LOD) :
  - simplification des modèles géométriques en concaténant les bords partagés par les triangles (ce qui est généralement le cas),
  - mettre en place une structure de données permettant d'interpoler en temps réel la concaténation des triangles.
  - résultat produit une transition non-brutale, mais l'objet bouge tout le temps (pas de seuils de transition).

- Geomorphs : technique proche du CLOD :
  - interpolations entre les niveaux sont réalisés dans les zones de transition.
  - transitions fluides, mais on voit les géométries bouger.
  - nécessite un gros travail préparatoire des données.
  - plutôt utilisée pour l'affichage de terrains (Black & White - Lionhead).

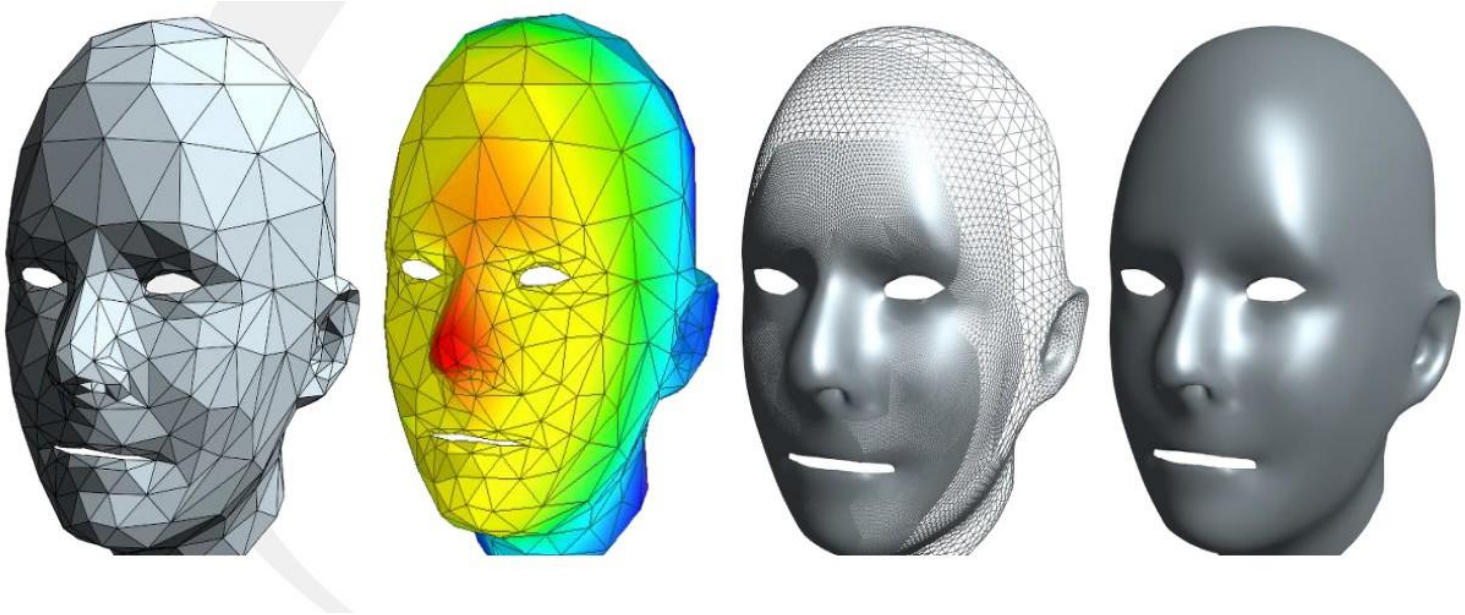






LOD dynamique pour l'affichage d'un terrain.
Le terrain est découpé en zones qui peuvent être plus ou moins raffinées selon différents critères (ici technologie ROAM – Realtime Optimally Adaptive Meshes).

LOD dépendant du point de vue

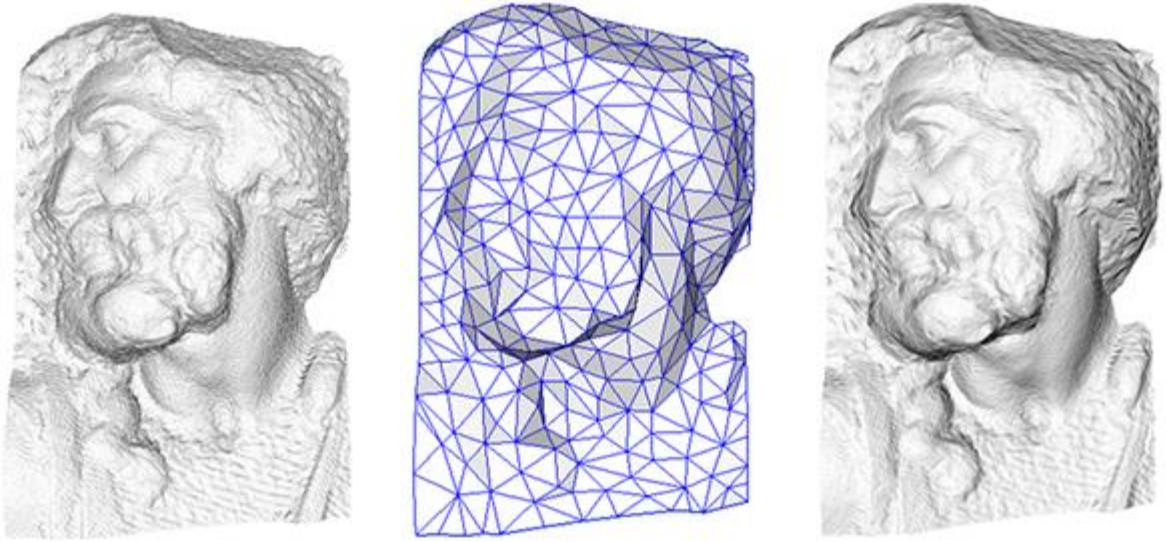


LOD hybride

Simplication de maillage + mapping

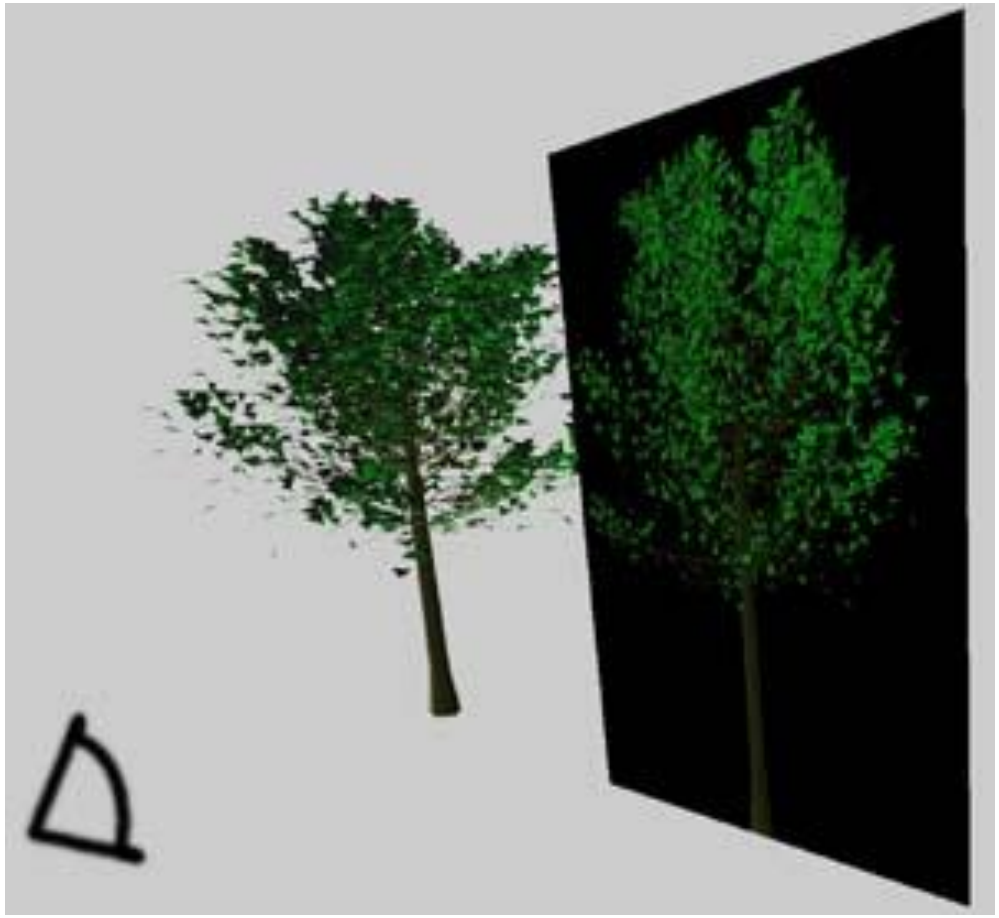
- Transfère des détails sous la forme de cartes
- normal map (carte de normale)
- displacement map (offset le long de la normale)

- Stockage compact : Pas de coordonnées 3D



Imposteurs

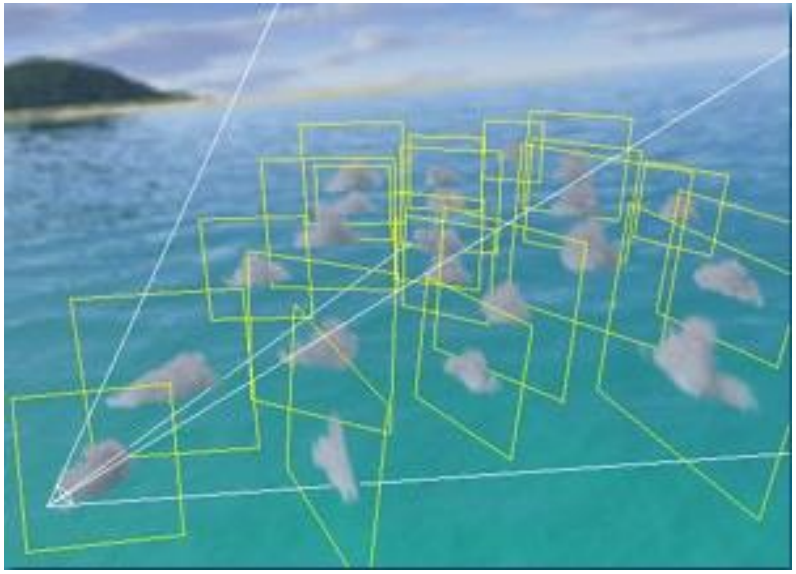
Billboard: remplacer une géométrie complète par une image de la géométrie, mappée sur une géométrie très simplifiée (en général un simple quadrilatère).

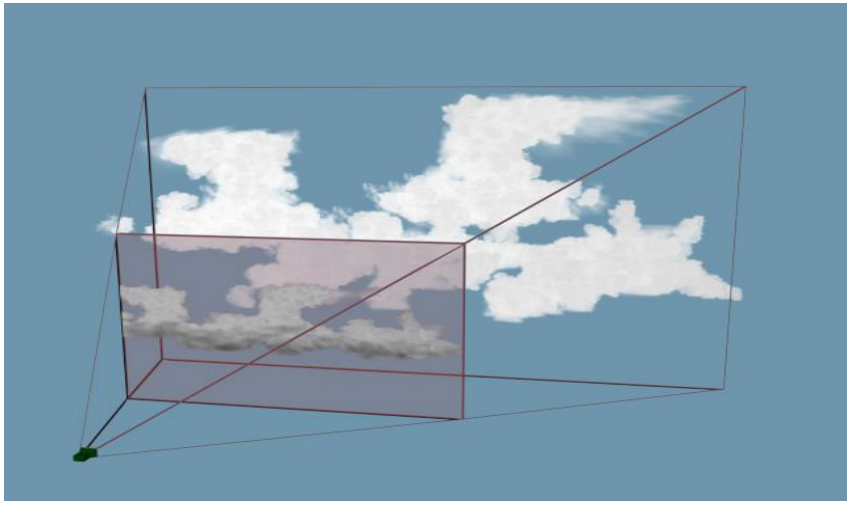


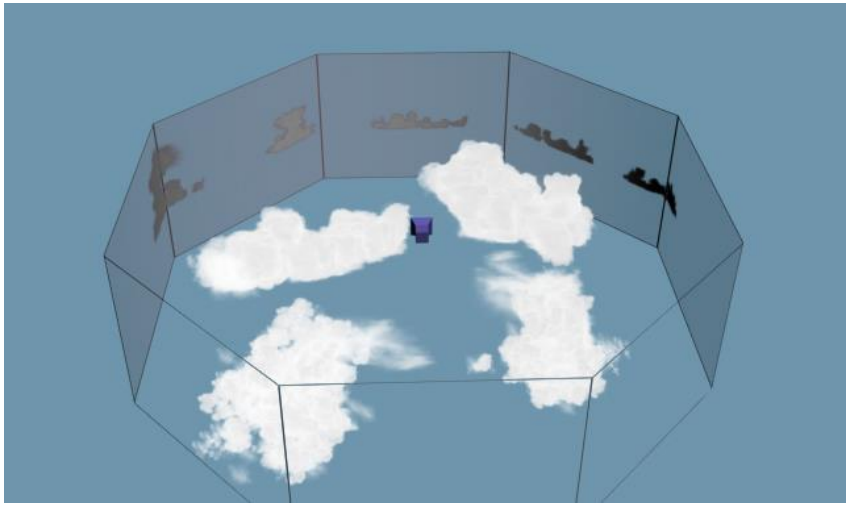
Imposteur

- 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)

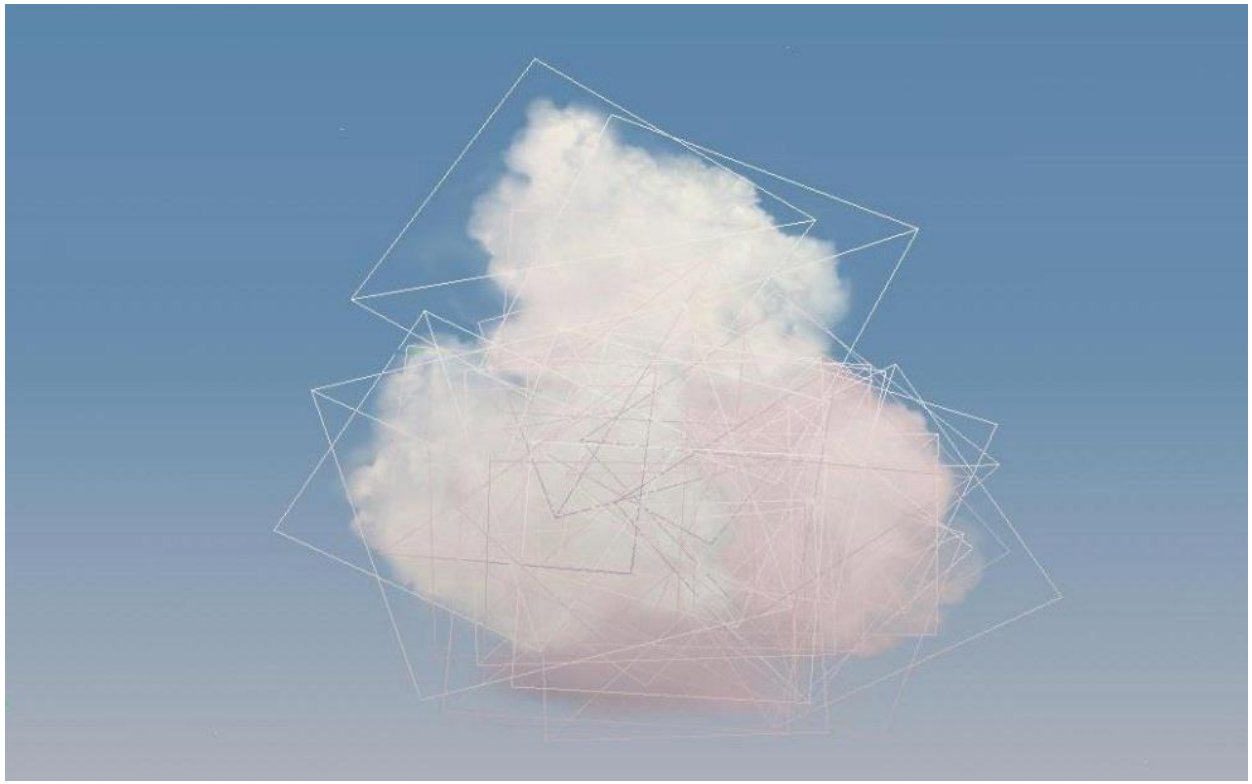






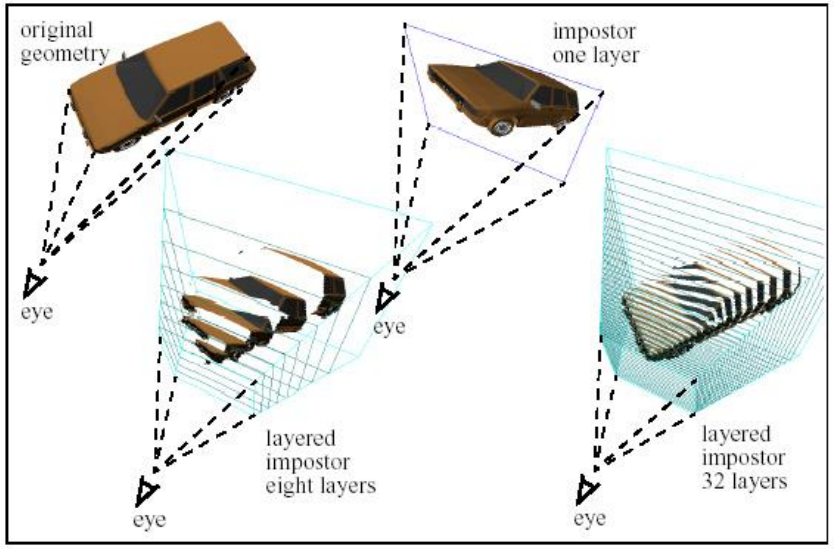
Imposteurs

- 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…

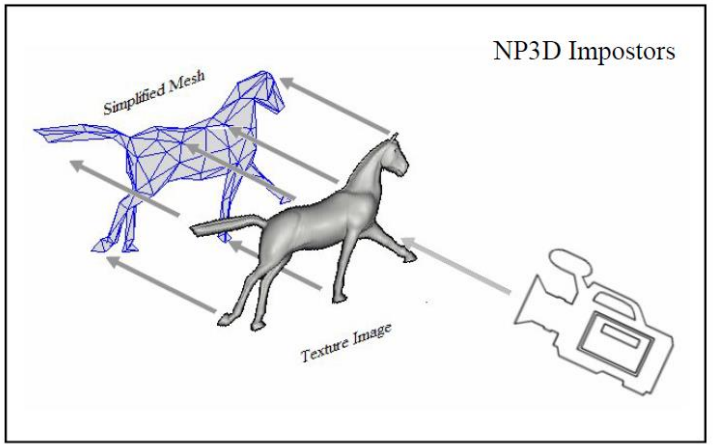


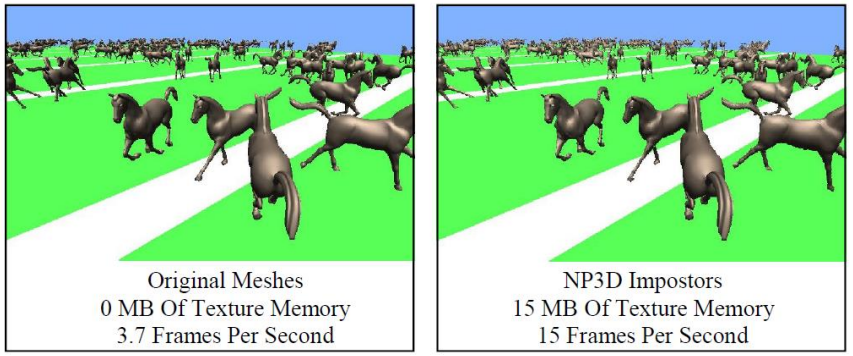
Imposteurs

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