M2-Images

Rendu Temps Réel - Gestion de Scènes

J.C. Iehl

December 1, 2009

résumé des épisodes précédents

contexte (paramètres) :

- décrire la matrice projection,
- pour chaque objet :
- décrire la matrice modelview,
- activer tableaux de positions + indexation,
- décrire la matière,
- décrire les sources de lumières,
- . . . draw()
- présenter le résultat.

résumé des épisodes précédents

pipeline graphique :

- récupérer modelview, projection + viewport,
- pour chaque primitive :
- récupérer les indices des sommets de la primitive,
- récupérer les sommets,
- projetter les sommets,
- éliminer les primitives de dos (back face culling),
- dessiner / fragmenter la primitive (rasterization),
- pour chaque fragment :
- calculer sa couleur + profondeur,
- écrire la couleur et la profondeur si profondeur $<$ zbuffer,

Affichage efficace

ça marche, mais :

toutes les primitives seront affichées, même celles qui ne sont pas visibles.

comment gagner du temps ?

idée :

ne pas afficher les objets qui ne sont pas visibles !

Visibilité

rappel :

un point p n'est visible que s'il se projette dans le cube unitaire du repère projectif de la caméra.

comment déterminer qu'un objet complet est visible ?

Visibilité

déterminer qu'un objet est visible :

plus rapidement que la carte graphique ?

objectif :

gagner du temps au total, les tests de visibilité doivent être simples.

ne pas tester tous les sommets de l'objet ?

Visibilité : boite englobante

idée :
utiliser la boite englobante de l'objet.

comment déterminer qu'une boite alignée sur les axes est visible ?

tester ses 8 sommets.

Visibilité : encore plus vite ?

c'est mieux, mais :

comment éliminer plusieurs objets en même temps ?

idée :

grouper les objets proches, et tester la boite englobante du groupe.

Visibilité hiérarchique

construire une hiérarchie d'englobants :

cf. structures accélératrices en lancer de rayons, BVH.

mais :

en général, les scènes sont dynamiques.

utiliser un arbre avec une mise à jour rapide lorsque les objets se déplacent.

Visibilité hiérarchique

BVH fonctionne très bien pour des scènes statiques mais la mise à jour est souvent trop lente pour une scène dynamique.

aller plus vite ?

- utiliser un octree pour insérer les objets :
- mais construire la boite englobante des objets de chaque noeud,
- construire la hiérarchie des englobants des objets.

remarque :

insérer les objets en fonction du centre de leur boite englobante.

Visibilité hiérarchique

afficher en s'éloignant de la caméra ?

pourquoi : pour aider le z-buffer.

parcours cohérent :

parcourir les fils de chaque noeud en s'éloignant de la camera.

facile pour un BVH, pour un octree ?

Visibilité globale

encore mieux ?

ne pas afficher les objets cachés derrière d'autres objets ?

idée :

si un objet est dans l'"ombre" d'un autre, il n'est pas visible.

Visibilité globale

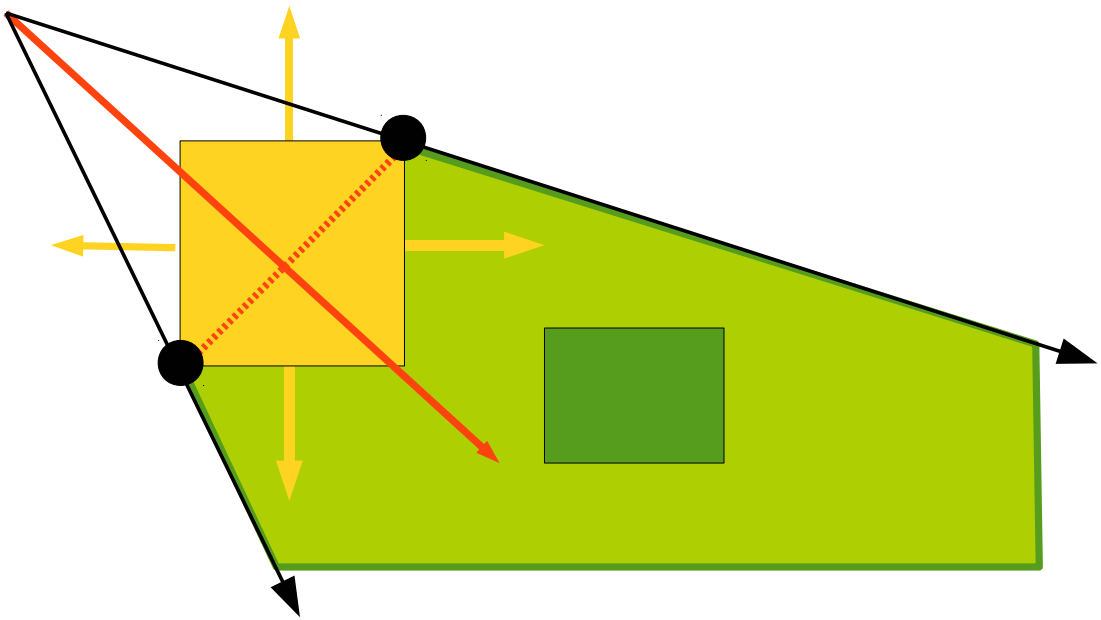
2 types de méthodes :

- dans l'espace objet,
- dans l'espace image, éventuellement avec l'aide de la carte graphique.

Visibilité globale : espace objet

construire le volume d'ombre d'un objet :

par rapport à la camera !



Visibilité globale : espace objet

algorithme :

- pour chaque arete de l'objet :
- si les faces adjacentes ne sont pas orientées de la même manière (par rapport à la caméra),
- l'arete fait partie de la silhouette de l'objet,
- construire la face latérale du volume d'ombre, passant par la caméra et l'arete.
- construire les faces avants du volume d'ombre.

remarque :

lorsque le volume d'ombre est convexe, comment tester efficacement l'inclusion d'un objet ?

Visibilité globale : espace objet

contrôler le temps d'exécution :

- éliminer les silhouettes intérieures, utiliser un maillage convexe comme objet de test,
- choisir $n << N$ objets sur lesquels faire le test,
- et bien sûr, utiliser les boites englobantes des objets à tester.

solution courante :

- utiliser un quad comme objet de test,
- choisir les quads qui occuppent le plus de place à l'écran ($\propto$ angle solide ...).
- et, placer les quads à la main !

Visibilité globale : espace image

mêmes idées :
mais utilisées sur la projection de l'objet de test.

il faut dessiner l'objet de test !

Visibilité globale : espace image

contôler le temps d'exécution :

- dessiner logiciellement une version réduite de l'image,
- utiliser la carte graphique !

requêtes de visibilité (occlusion queries) :

- extension openGL : ARB_occlusion_query
- algorithme et utilisation efficace : GPU Gems 2, chapitre 6
  Hardware Occlusion Queries Made Useful

Requêtes de visibilité (openGL)

principe :

- activer le mode requête,
- dessiner l'objet,
- attendre le résultat de la requête,
- désactiver le mode requête,
- dessiner l'objet, s'il est visible.

Requêtes de visibilité (openGL)

simple :

- mais totalement inefficace :
- il faut dessiner 2 fois les objets !
- et attendre le résultat de la requête,
- c'est presque 3 fois plus lent que l'affichage direct !

limiter le nombre de requêtes utilisées, utiliser la boite englobante de l'objet pour "poser" la requête, exploiter le résultat à l'image suivante (pour cacher le temps de réponse).

se limiter à "quelques" objets particulièrement couteux.

Pré-calculer

pour une grande partie de la scène, les objets ne bougent pas . . .

déterminer une seule fois leur visibilité relative et re-exploiter le résultat.

PVS :
ensemble d'objets potentiellement visibles.

approximation du résultat exact : certains objets seront affichés alors qu'ils ne sont pas visibles.

PVS

plusieurs types de méthodes :

basées sur les mêmes idées : partitions spatiales, BSP, cellules + portails, etc.

utilisées dans quasiment tous les jeux : quake (BSP), doom 3 (cells & portals), unreal 3, etc.

et alors ?

plusieurs méthodes :

- pas de méthode "générale" toujours efficace,
- selon la structure de la scène certaines méthodes sont plus adaptées.

scènes d'intérieurs :

BSP, cells & portals,

scènes d'extérieurs :

- ??
- visibilité globale sur le relief, sur la végétation,
- PVS ou version spécialisée pour l'affichage de terrain.

et alors ?

en pratique :

- précalculer le PVS,
- + objets visibles par la caméra,
- + objets dans l'ombre de quads (placés à la main),
- + requêtes pour quelques objets très "lourds".

Insomniac games : occlusion systems