M2-Images

J.C. Iehl

January 29, 2007

OpenGL

- A quoi ça sert ?
- Qu'est ce que c'est ?
- Comment ça marche ?

OpenGL : A quoi ça sert ?

A dessiner des polygones 3D sur une image 2D !

1. décrire les objets à afficher sous forme d'un ensemble de faces polygonales,
2. décrire les positions des sommets des faces,
3. décrire la matière des faces (couleur, aspect mat ou réfléchissant, etc.).
4. décrire la lumière qui éclaire les objets (+ astuces)
5. décrire la projection (passage 3D vers 2D)

OpenGL : A quoi ça sert ?

Conséquences

1. description hiérarchique des repères de modélisation,
2. positionnement des objets devant la camera,
3. toutes ces transformations sont composées dans une seule matrice `MODELVIEW`,
4. description de la projection par une matrice `PROJECTION`.
5. la notion de camera n'existe pas !

OpenGL : Qu'est ce que c'est ?

une librairie

permet à l'application d'utiliser les fonctionalités OpenGL,

un driver

permet à la librairie de transmettre les données au matériel et de réaliser l'affichage demandé par l'application,

du matériel spécialisé

réalise l'affichage le plus vite possible (G80 600M triangles / seconde).

OpenGL : Comment ça marche ?

dessine les primitives une par une, dans l'ordre

plusieurs paramètres disponibles selon le type de primitive (point, ligne, polyogne).

le contexte

permet de stocker l'ensemble des paramètres d'affichage.

OpenGL : Comment ça marche ?

la librairie

- vérifie que l'application utilise correctement les commandes OpenGL,
- prépare les données et les paramètres pour simplifier leur utilisation par le matériel.

le driver

- construit le contexte,
- transmet le contexte, les données et les commandes au matériel.

OpenGL : Comment ça marche ?

le matériel

- récupère les données,
- récupère les commandes,
- récupère le contexte.

utilise les paramètres du contexte et les données mises en forme par la librairie et / ou le driver pour réaliser les opérations demandées par l'application.

modèle client-serveur

- le client : l'application, la librairie et le driver,
- le serveur : le matériel (et le driver dans certains cas).

OpenGL : Mais à quoi ça sert (réellement) ?

Résumé

- afficher des polygones,
- rendu interactif !
- calculs génériques ?

Ce que OpenGL ne sait pas faire

- OpenGL est une librarie graphique,
- on ne l'utilise jamais seul !

OpenGL : Développement d'applications graphiques

Portabilité

- OpenGL est disponible plusieurs plateformes,
- utiliser des librairies "annexes" disponibles sur les mêmes plateformes,
- libSDL (images, textes, plugins, threads, réseaux, timers, audio, joystick, évènements, etc.),
- OpenAL (audio 3D),
- ...

OpenGL : Développement

Jeux video

- affichage : OpenGL,
- audio : OpenAL,
- animations ?
- physique ?
- comportement ? intelligence artificielle (path finding, etc.) ?
- multi-joueurs ?
- chargement des données ?

OpenGL : Utilisation efficace

bien utiliser la librairie

- pas de `glVertex` !
- utiliser les `vertex array` et les `buffer objects`.

limiter la quantité de géométrie à dessiner

- ne dessiner que ce qui est visible ! (*"culling"*)
- et dans le bon ordre ! (en s'éloignant de la camera) (*"Z-Buffer"*)

limiter les changements de contextes

changement de type de primitive, de type d'affichage, de texture, de shader ...

OpenGL : Bien utiliser la librairie

limiter les transferts de données entre CPU et GPU

- `vertex arrays` : primitives indexées, strips, fans.
- `vertex buffers`.

Primitives indexées

un cube : 8 sommets, 6 faces.

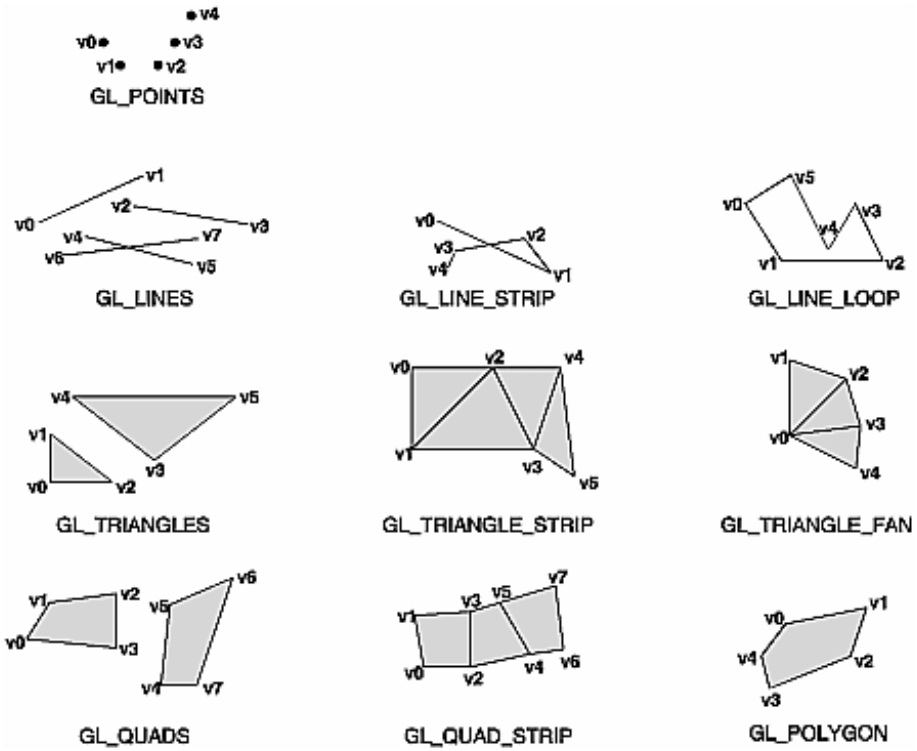
glVertex

- 24 `glVertex()` !
- 24*float[4]

vertex arrays

- 8 positions + 24 indices
- 8*float[4] + 24*uint8

Primitives indexées compactes



Primitives indexées compactes

Comment transformer un objet quelconque en description compacte ?

Exemple : le cube

2 strips de 3 quads, 1 strip de 6 quads ou trianguler ...

En général :

problème NP complet ... cf. page du cours,

- *"Optimization of Mesh Locality for Transparent Vertex Caching"*
  H. Hoppe
- *"Fast and Effective Striping"*
  J. Behr and M. Alexa
- ATI Tootle, nvStripLib

Vertex Arrays / Vertex Buffers Objects

- les vertex array sont dans la mémoire du client (l'application et / ou la librairie)
- les vertex buffers objects sont stockés sur le serveur (mémoire de la carte graphique) ...
- exemple d'utilisation : cf. page du cours,
- utiliser un *vertex shader* pour déformer / animer les objets (plus de transmission des nouvelles positions à chaque image).

Utilisation efficace : Demo !

source disponible sur la page du cours :
"An Improved Adjacency Data Structure for Fast Triangle Stripping"
P. Reuter and J. Behr and M. Alexa

OpenGL : Ne dessiner que ce qui est visible

OpenGL affiche scrupuleusement tous les polygones ...

limiter l'affichage à la partie visible depuis la camera

- aller plus vite que le GPU
  (G80 : 1 triangle / cycle == 600M triangles / seconde)
- seule solution : éliminer de gros "blocs" de géométrie,
- utiliser une partition spatiale grossière de la scène ...
- et laisser le GPU finir le travail
  (sur les blocs partiellement visibles).

Partition spatiale

Découpage hiérarchique de la boîte englobante de la scène

- octree
- BSP,
- kdtree,
- BVH ...

Partition spatiale : Parcours

Comment déterminer les noeuds visibles ?

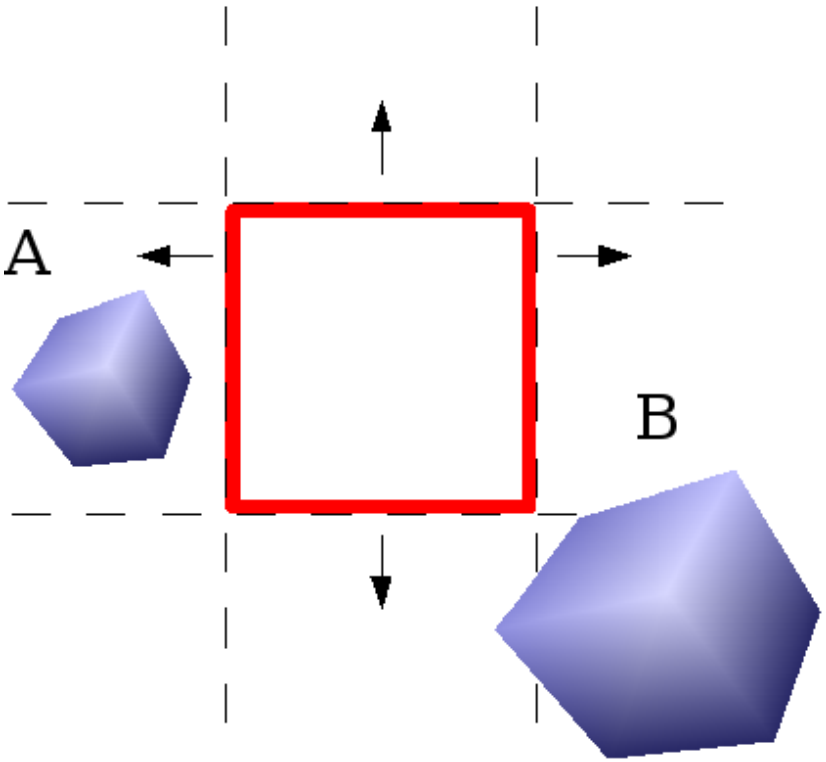
Parcours en profondeur de l'arbre

- classifier la visibilité d'un noeud :
- visible, non visible,
- partiellement visible.

Visibilité d'un noeud

- déterminer l'existance d'une intersection entre une boîte et une pyramide tronquée ...
- plus simple : trouver un plan qui rejette les sommets du noeud.

Visibilité d'un noeud : Rejet par un plan



Visibilité d'un sommet

transformation d'un sommet

$v = (\, x\, y\, z\, w\,)^t$ transformé par les matrices de modèle M, et de projection P :

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = P\,M \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

v est inclus dans le cube de vision (et la pyramide de vision), si :

$$-w_c < x_c < w_c$$

$$-w_c < y_c < w_c$$

$$-w_c < z_c < w_c$$

Partition spatiale : Parcours

- si le noeud est entièrement visible : afficher
- si le noeud est entièrement non-visible : ne pas afficher
- sinon : continuer le parcours de la hiérarchie.

Ne dessiner que ce qui est visible : Résumé

Frustum culling

- Découper la scène en "gros" blocs,
- Déterminer l'ensemble de blocs visible par la camera.
- Afficher les blocs en exploitant le *Z-Buffer*, c'est à dire en s'éloignant de la camera.
- Il suffit de descendre du "coté" de la caméra en premier, lors du parcours de la hiérarchie, et de dessiner les blocs dans cet ordre.
- Conséquence : construire un VBO par bloc,
- Ou : renuméroter les sommets de la scène pour contruire un seul VBO.

Ne dessiner que ce qui est visible : Faire mieux

ne pas dessiner les objets qui sont cachés derrières d'autres !

Occlusion Culling

- pendant le parcours de la hiérarchie, pour les noeuds (partiellement) visibles :
- dessiner leur boite englobante,
- si la boite englobante est en partie visible (rien devant), afficher les objets,
- sinon, arrêter le parcours.

Occlusion queries

OpenGL sait combien de pixels dessinés sont visibles ($Z < Z_{buffer}$) :

- glGenQueries()
- glBeginQuery()
- ... glDrawArrays()
- glEndQuery()
- glGetQueryObjectXXX()

Occlusion queries

- Problème : on dessine 2 fois la scène ... trop lent !
- Minimiser le nombre de requètes et "cacher" le temps de réponse des requètes :
- *"Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful"*
  J. Bittner, M. Wimmer, H. Piringer, W. Purgathofer
  cf. page du cours.

Faire mieux : et après ?

s'il reste encore trop de géométrie ...

Simplifier la forme des objets

plus l'objet est loin, moins les détails se voient ...
cf. page du cours

Remplacer les objets par une texture

si l'objet est suffisamment loin, quelques images bien choisies peuvent faire illusion ...
+ masquer les défauts en utilisant un brouillard atmosphérique dense.

Faire mieux : Niveaux de détails