

M2-Images

Rendu temps réel - OpenGL 3

J.C. Iehl

April 11, 2012

Introduction

OpenGL et la 3D :

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

OpenGL

OpenGL :

- ▶ api 3d ?
- ▶ expose les fonctionnalités d'un pipeline 3d ?
- ▶ dessiner des objets sur l'écran.

une api 3d est l'ensemble de fonctions permettant de paramétrer toutes les opérations nécessaires à l'affichage d'objets.

une api 3d est l'ensemble de fonctions permettant de paramétrer un pipeline 3d...

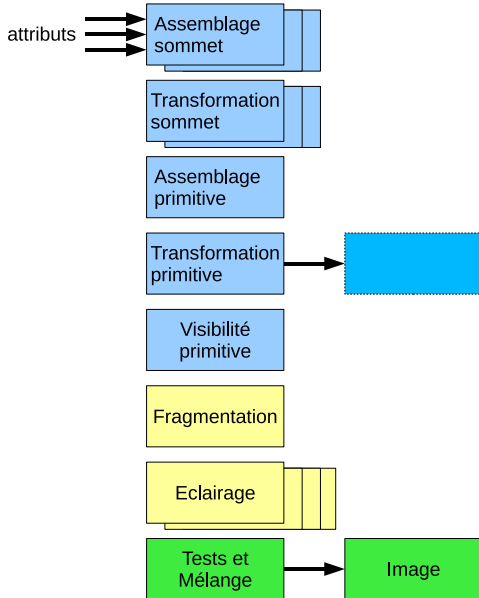
Rappel: Pipeline graphique

pipeline graphique :

organisation des opérations nécessaires à l'affichage d'objets.

3 types de pipelines :

- ▶ lancer de rayons,
- ▶ reyes / renderman,
- ▶ carte graphique.



Pipeline carte graphique

données :

- ▶ ensemble de primitives :
- ▶ ensemble de sommets attribués (vertex),
- ▶ + topologie,

traitements :

- ▶ transformations géométriques + coloriage,
- ▶ configuration des fonctions "fixes",
- ▶ ou programmes (shaders) + paramètres,

résultats :

image couleur, et / ou ensemble de primitives transformées.

Afficher un objet 3d

forme :

- ▶ placée et orienté par rapport à la caméra (comment ?),
- ▶ description par des primitives simples (triangles ?),

aspect, matière, interaction avec la lumière :

- ▶ couleur uniforme,
- ▶ diffuse, spéculaire, réfléchissante (glossy / phong),
- ▶ détails : textures,
- ▶ paramètres du shader associé à la matière.

Objets et primitives d'affichage

une carte graphique est spécialisée pour afficher des points, droites, triangles (quadrangles, polygones convexes).

afficher un objet == décomposer la forme de l'objet en primitives :

- ▶ un objet est un ensemble de faces (triangles, quadrangles),
- ▶ une face est un ensemble de sommets (3 ou 4),
- ▶ un sommet est un ensemble d'attributs.

une face est donc l'interpolation des attributs de ses sommets. . .

ou se trouve l'objet ? (par rapport à la camera ?)

Objets

décrire la forme d'un objet :

- ▶ ensemble de sommets des primitives,
- ▶ ensemble d'indices + ensemble de sommets (partagés),
- ▶ description sous forme de tableaux (sommets, sommets + indices),

stocker la description :

- ▶ stockage par l'application (conséquences ?)
- ▶ stockage sur la carte graphique (conséquences ?)

Primitives indexées

un cube : 8 sommets, 6 faces.

description par sommet :

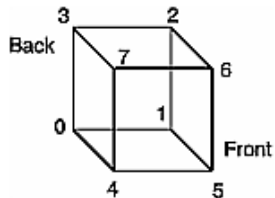
- ▶ 24 positions (6 faces de 4 sommets)
- ▶ $24 * \text{float}[4]$ (position == VEC4)

description indexée :

- ▶ 8 positions + 24 indices
- ▶ $8 * \text{float}[4] + 24 * \text{uint8}$

quelle est la meilleure solution (résultat identique) ?

Primitives indexées : exemple



Vertex

sommets attribués (Vertex) :

- ▶ position 3d,
- ▶ matière : couleurs ambiente, diffuse, etc.
- ▶ normales,
- ▶ textures + coordonnées,
- ▶ paramètres supplémentaires (shaders).

Affichage des objets

pour chaque objet :

- ▶ placer et orienter l'objet par rapport à la caméra (transformation),
- ▶ activer le type de primitive (triangles),
- ▶ activer l'utilisation des attributs de sommets : position, + couleur, + normale, + textures,
- ▶ activer les tableaux des attributs de sommets (+ indices),
- ▶ activer les textures utilisées,
- ▶ activer et paramétrer les shaders,
- ▶ draw().

Présentation du résultat

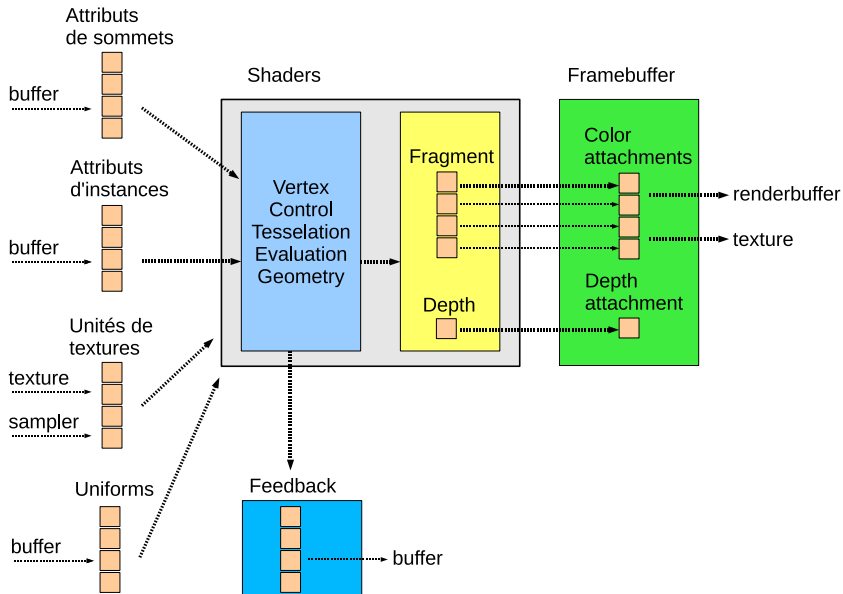
affichage d'une image :

- ▶ effacer l'image,
- ▶ placer / orienter la caméra,
- ▶ définir la projection de la caméra (passage 3d vers 2d),
- ▶ dessiner les objets,
- ▶ présenter l'image résultat.

Pipeline graphique : résumé

plusieurs étapes :

- ▶ attributs des sommets,
- ▶ topologie des primitives à dessiner,
- ▶ transformations géométriques :
passage repère local des sommets vers repère projectif de la caméra,
- ▶ dessin des primitives dans l'image :
déterminer sur quels pixels dessiner chaque primitive
- ▶ écrire les pixels dans l'image résultat.



Qu'est ce que c'est ?

une librairie / api 3d :

permet à l'application de paramétrer le pipeline 3d implémenté par la carte graphique,

un driver :

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

du matériel spécialisé :

- ▶ accélération des opérations du pipeline 3d,
- ▶ réalise l'affichage le plus vite possible (geforce 680gtx 4G triangles / seconde).

Comment ça marche ?

dessine les primitives une par une, dans l'ordre :
plusieurs paramètres disponibles selon le type de primitive (point,
ligne, triangle).

le contexte :

permet de stocker l'ensemble des paramètres.

Comment ça marche ?

la librairie / api 3d :

- ▶ vérifie que l'application utilise correctement l'api,
- ▶ 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.

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).

Mais à quoi ça sert (réellement) ?

résumé :

- ▶ afficher des primitives,
- ▶ rendu interactif !
- ▶ calculs génériques (autres que 3d).

ce qu'une api 3d ne sait pas faire :

- ▶ OpenGL est une librairie graphique,
- ▶ on ne l'utilise jamais seul !
- ▶ (idem pour DirectX Graphics)

OpenGL : Développement

portabilité :

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

OpenGL : Développement

extensions :

- ▶ introduction de nouvelles fonctionnalités,
- ▶ optimisation de fonctionnalités existantes,
- ▶ permet de tester avant d'intégrer dans la version suivante.

utiliser une librairie pour utiliser les extensions : GLEW

OpenGL 2,3,4 et OpenGL ES 1,2

fonctionnalités différentes :

- ▶ openGL 2 : carte graphique SM3 (geforce 5, radeon 9800)
- ▶ openGL 3 : carte graphique SM4 (geforce 8, radeon hd 2000)
- ▶ openGL 4 : carte graphique SM5 (geforce 400, radeon 5000)

openGL 2 + extensions : fonctionnalités SM4 et SM5 ... mais
openGL 3 et 4 : meilleure intégration dans l'api des nouvelles
fonctionnalités.

openGL ES : sous ensemble des fonctionnalités pour les systèmes
embarqués.

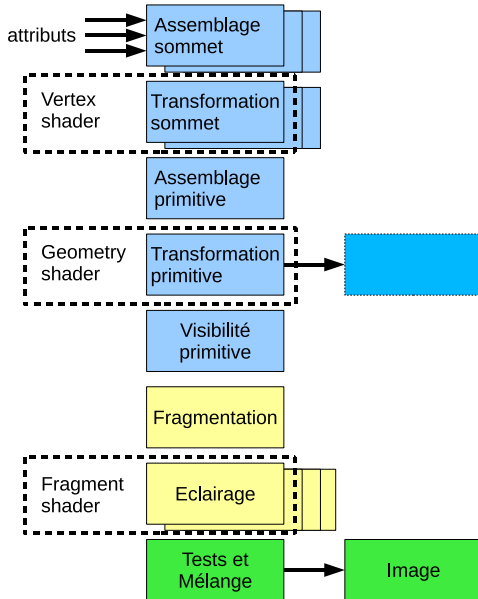
OpenGL Core Profile

évolution de l'api :

- ▶ openGL 2 et cartes $< SM3$: fonctions "cablées", non programmables, mais configurables,
- ▶ openGL 3.0 et 4.0 : transition vers un modèle entièrement basé sur les shaders, plus de fonctions "cablées".

l'api reflète ce changement matériel : la description des lumières et des matières n'existe plus, il faut écrire un shader pour obtenir le même résultat, ou autre chose (?).

mais on peut avoir les deux ! cf. compatibility profile.



Opérations

OpenGL et DirectX Graphics exposent les mêmes fonctionnalités :

- ▶ initialisation,
- ▶ description des transformations (matrice, camera, etc.),
- ▶ description des objets (forme) + attributs (matière),
- ▶ compilation et paramétrage des shaders,
- ▶ affichage des objets + paramètres d'affichage,
- ▶ présentation du résultat,
- ▶ ... recommencer.

Initialisation

créer un contexte de rendu :

- ▶ permettre à plusieurs applications / threads d'utiliser la carte graphique,
- ▶ interactions avec le système de fenêtrage.

définir comment afficher :

- ▶ dans une fenêtre / en plein écran,
- ▶ avec / sans synchronisation,
- ▶ plusieurs buffers : draw, display, color, z-buffer, stencil, ...
- ▶ format : RGB, RGBA, 8bits, float 32bits, float16 bits, ...

SDL : exemple

```
SDL_Init(SDL_INIT_VIDEO);  
info= SDL_GetVideoInfo();  
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);  
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);  
  
screen= SDL_SetVideoMode(width, height, info->vfmt->BitsPerPixel,  
    ... SDL_OPENGL);
```