M2PRO-Images

Images Immersives - Rendu Temps Réel

J.C. Iehl

January 10, 2008

Plan

- Cartes graphiques 3D
- Opérations
- API 3D (DirectX / OpenGL)
- Pipeline graphique et shaders
- Utilisation Efficace
- Effets / Rendu multi-passes

Cartes graphiques 3D

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

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, texture, etc.).
4. décrire la lumière qui éclaire les objets (+ astuces)
5. décrire la projection (passage 3D vers 2D)

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 !

Qu'est ce que c'est ?

une librairie / API 3D

permet à l'application d'utiliser les fonctionalités de la carte graphiqe 3D,

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).

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.

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 polygones,
- rendu interactif !
- calculs génériques ?

Ce qu'une API 3D ne sait pas faire

- OpenGL est une librarie graphique,
- on ne l'utilise jamais seul !
- idem pour DirectX Graphics

OpenGL : Développement

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 ?

DirectX : Développement

Portabilité

- non portable,
- DirectX Graphics + librairie utilitaire D3DX

DirectX : Développement

Jeux video

- affichage : DirectX Graphics,
- audio : DirectX Audio : DirectSound, X3DAudio,
- animations : D3DX (base)
- physique ?
- comportement ? intelligence artificielle (path finding, etc.) ?
- multi-joueurs ?
- chargement des données ?

Opérations

OpenGL et DirextX (Graphics) exposent les mêmes fonctionnalités

- initialisation
- description de la caméra
- desciption des sources de lumières
- description des objets (primitives) + attributs (matières . . .)
- affichage des objets (primitives) + paramètres
- 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);
```

Caméra

- position
- orientation
- projection 2D / orthographique, 3D / perspective, centrée, ...

SDL : exemple

```
// selectionne la matrice de visualisation
glMatrixMode(GL_PROJECTION);
// reinitialise les transformations
glLoadIdentity();

// definit une projection perspective
gluPerspective(50., 1., 1., 1000.);
```

Lumière

Energie

- ambiente
- diffuse, spéculaire, réfléchissant (glossy / phong)

Type de source

- ponctuelle, directionnelle,
- spot, ...

Objets (forme) + Matières

Forme

- placée et orienté devant la camera
- description par des primitives simples

Matière : interagir avec la lumière

- ambiente
- diffuse, spéculaire, réfléchissant (glossy / phong)
- détails : texture, shaders

Primitives

primitives : points, droites, triangles, quadrangles, polygones convexes

Décomposition 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

ou se trouve l'objet ?

Faces

Faces

- liste de sommets
- liste d'indices + liste de sommets partagés
- description sous forme de tableaux (sommets, sommets + indices)
- stockage par l'application (conséquences ?)
- stockage sur la carte graphique (conséquences ?)

Primitives indexées

un cube : 8 sommets, 6 faces.

description par sommet

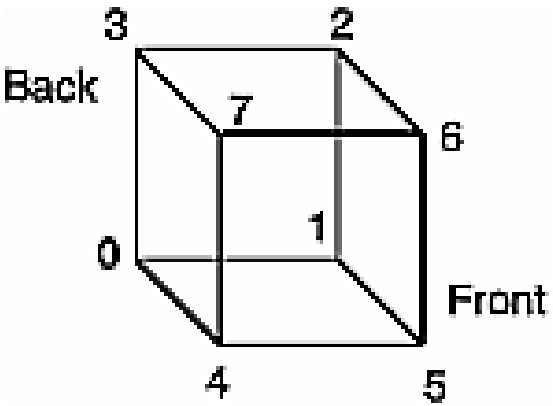
- 24 positions
- 24*float[4]

description indexée

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

Quelle est la meilleure solution (résultat identique) ?

SDL : exemple



SDL : exemple

```
int frontIndices= {4, 5, 6, 7};
int rightIndices= {1, 2, 6, 5};
int bottomIndices= {0, 1, 5, 4};
int backIndices= {0, 3, 2, 1};
int leftIndices= {0, 4, 7, 3};
int topIndices= {2, 3, 7, 6};

// dessiner face par face
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, frontIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, rightIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, bottomIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, backIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, leftIndices);
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_INT, topIndices);

// dessiner les 6 faces directement
static GLubyte allIndices = {
    4, 5, 6, 7, 1, 2, 6, 5,
    0, 1, 5, 4, 0, 3, 2, 1,
    0, 4, 7, 3, 2, 3, 7, 6
};

glDrawElements(GL_QUADS, 24, GL_UNSIGNED_INT, allIndices);
```

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 devant la caméra (matrice MODELVIEW)
- activer le type de primitive (triangles)
- activer le format des sommets : position, + couleur, + normale, + textures
- activer les tableaux de sommets et d'indices
- activer les textures utilisées
- (activer les shaders)
- draw()

SDL : exemple

```
// place et oriente l'objet
glRotatef(rotation_x, 1.f, 0.f, 0.f);
glRotatef(rotation_y, 0.f, 1.f, 0.f);
glRotatef(rotation_z, 0.f, 0.f, 1.f);
glTranslatef(position_x, position_y, position_z);

// active le tableau de positions
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, sizeof(VERTEX), mesh->vertices);

// dessine (mesh->indices_n / 3) triangles
glDrawElements(GL_TRIANGLES,
    mesh->indices_n, GL_UNSIGNED_INT, mesh->indices);
```

Présentation

boucle d'affichage

- effacer les buffers utilisés (color, z-buffer, etc.)
- placer / orienter la caméra
- afficher les objets
- échanger les buffers de dessin et de présentation

SDL : exemple

```
// efface "l'ecran"
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// selectionne la matrice de la scene
glMatrixMode(GL_MODELVIEW);
// reinitialise les transformations
glLoadIdentity();

// oriente la scene par rapport a la camera qui est restee en 0,0,0
glRotatef(rotation_x, 1.f, 0.f, 0.f);
glRotatef(rotation_y, 0.f, 1.f, 0.f);
glTranslatef(camera_x, camera_y, camera_z);

// desssine la scene
display( ... );

SDL_GL_SwapBuffers();
```