

M2-Images

Rendu temps réel - OpenGL 3

J.C. lehl

April 11, 2012

résumé des épisodes précédents . . .

openGL

- ▶ affichage 2D de primitives géométriques simples,
- ▶ opérations.

détails . . .

Présentation de l'api

openGL est un automate, l'api n'est pas basée objet.

sélecteurs :

- ▶ activer un " mode " (l'objet que l'on veut paramétrer),
- ▶ puis modifier les propriétés associées (le contexte),
- ▶ les " objets " sont manipulés à travers un identifiant opaque (pas de pointeur).

exemple :

```
GLuint buffer;  
glGenBuffers(1, &buffer);  
glBindBuffer(GL_ARRAY_BUFFER, buffer);  
glBufferData(GL_ARRAY_BUFFER, ...);
```

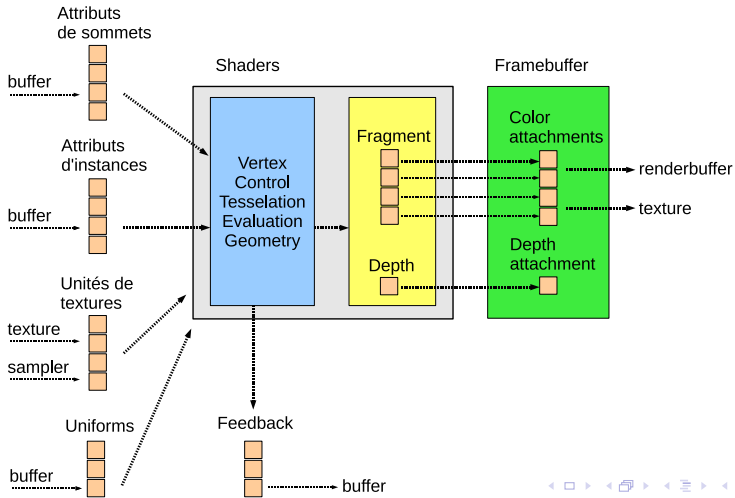
Présentation de l'api

les fonctions de l'api sont nommées en fonction du type de leur paramètres. (pas de surcharge en C...)

exemple :

```
glUniform1i( ... GLint v );  
glUniform3f( ... GLfloat x, GLfloat y, GLfloat z );
```

Présentation de l'api



Transformations

afficher un objet :

- ▶ les (positions des) sommets des primitives sont définies dans un repère lié à l'objet,
- ▶ il faut placer et orienter chaque objet dans la scène,
- ▶ et changer encore de repère pour placer les objets dans le repère de la caméra,
- ▶ + les projeter sur le plan image 2d pour les dessiner,
- ▶ + mettre à l'échelle de la fenêtre !

== enchainement de changement de repères.

Transformations ...

utilisation de matrices ...

pour représenter tous les types de changements de repères.

une api 3d utilise des matrices homogènes 4×4 ...

Matrices homogènes

représenter une rotation :

- ▶ R : matrice 3×3 et p : vecteur 3,
- ▶ changement de repère : $p' = Rp$.

mais pour représenter une translation ?

- ▶ changement de repère : $p' = Tp$. comment définir T ?

une matrice homogène encode les translations dans une colonne supplémentaire.

Matrices homogènes

translation par un vecteur \vec{t} :

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrices homogènes

d'autres transformations :

- ▶ rotationX, rotationY, rotationZ, rotation,
- ▶ translation,
- ▶ mise à l'échelle,
- ▶ symétrie, miroir / reflet,
- ▶ projections orthographique et perspective.

Vecteurs et points homogènes

plus qu'à calculer le produit matrice \times vecteur ...
euh ... il manque une ligne ?

$$\vec{u}^h = \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_w = 0 \end{bmatrix} \text{ et } p^h = \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w = 1 \end{bmatrix}$$

pourquoi ? on peut simplifier en disant :
"un vecteur ne subit pas de translation, donc $u_w = 0$ "

Vecteurs et points réels

certaines transformations modifient la composante w

- ▶ une projection, par exemple.
- ▶ comment retrouver le point "réel" correspondant ?

$$p = p^h / p_w^h = \begin{bmatrix} p_x^h / p_w^h \\ p_y^h / p_w^h \\ p_z^h / p_w^h \\ p_w^h / p_w^h = 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Compositions de transformations

seulement 3 matrices :

- ▶ ModelView : objet \rightarrow scène \rightarrow caméra,
- ▶ Projection : définit la projection de la caméra,
- ▶ Viewport(x, y, w, h) : définit les dimensions de la fenêtre.

succession de plusieurs repères : objet, scène, caméra, projection, fenêtre.

”grouper” plusieurs transformations dans une seule matrice ?

Compositions de transformations

repère 1 $\xrightarrow{T_{12}}$ repère 2 $\xrightarrow{T_{23}}$ repère 3

on connaît p_1 , les coordonnées de p dans le repère 1, quelles sont ses coordonnées dans le repère 3 ?

$$\begin{aligned}
 p_2 &= T_{12}p \\
 p_3 &= T_{23}p_2 \\
 p_3 &= T_{23}(T_{12}p) \\
 p_3 &= T_{23}T_{12}p \\
 p_3 &= Tp \text{ avec } T = T_{23}T_{12}
 \end{aligned}$$

Compositions de transformations

et dans l'autre sens ?

on connaît p_3 , et on veut les coordonnées de p dans le repère 1 ?

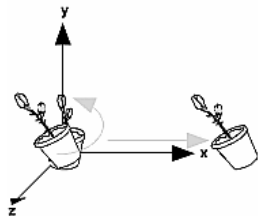
repère 1 ← repère 2 ← repère 3
 $(T_{12})^{-1}$ $(T_{23})^{-1}$

$$p_3 = T_{13}p \text{ et } p = (T_{13})^{-1}p_3$$

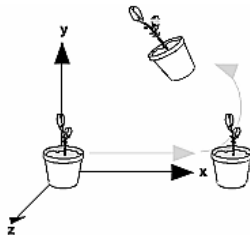
Transformations : Bilan

l'ordre des transformations est important :

- ▶ le produit de matrices n'est pas commutatif,
- ▶ calculs dans l'ordre inverse des changements de repères.



Rotate then Translate



Translate then Rotate

Afficher un objet

en résumé :

- ▶ définir les dimensions de la fenêtre (`glViewport`),
- ▶ définir la projection de la caméra (`Projection`),
- ▶ placer et orienter l'objet dans (le repère de) la scène (`Model`),
- ▶ placer et orienter la caméra dans (le repère de) la scène (`View`),
- ▶ paramétrer le shader avec les transformations.

Affichage

allouer des buffers sur le gpu et transférer les données.

plusieurs représentations :

- ▶ vertex buffer : efficace,
- ▶ vertex buffer + index buffer : plus performant,
- ▶ réordonner les sommets pour exploiter les caches du gpu.

Vertex Buffer Object

- ▶ créer les buffers,
- ▶ allouer les buffers,
- ▶ transférer les données,
- ▶ activer et associer les buffers,
- ▶ afficher avec `glDrawArrays(...)`.
- ▶ afficher avec `glDrawElements(... 0)`.

Vertex Buffer Object

créer un buffer et transférer les données :

- ▶ créer : `glGenBuffers()`,
- ▶ activer : `glBindBuffer(buffer)`,
- ▶ allouer : `glBufferData(length)`,
- ▶ transfert : `glBufferSubData(offset, length)`,
- ▶ `glMapBuffer[Range](offset, length)`,
- ▶ `glUnmapBuffer()`.

Vertex Buffer Object

- ▶ créer un buffer et transférer les positions des sommets,
- ▶ activer le buffer sur `GL_ARRAY_BUFFER`,
- ▶ associer le tableau de sommets `glVertexAttribPointer()`,
- ▶ activer l'utilisation d'un tableau de sommets, `glEnableVertexAttribArray()`,
- ▶ afficher les primitives, `glDrawArrays()`.

Exemple : création vertex buffer

```
Point sommets[n]= { ... };  
  
GLuint vertex_buffer;  
glGenBuffers(1, &vertex_buffer);  
  
glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer);  
glBufferData(GL_ARRAY_BUFFER, sizeof(Point)*n,  
             sommets, GL_STATIC_DRAW);
```

Exemple : dessiner avec un vertex buffer

```
glBindBuffer(GL_ARRAY_BUFFER, buffer);  
glEnableVertexAttribArray( attribut );  
glVertexAttribPointer( attribut, 3, GL_FLOAT,  
    sizeof(Points), 0 );  
glDrawArrays( primitive, 0, n );
```

Index Buffer

- ▶ créer un buffer, transférer les indices des sommets de chaque primitive,
- ▶ activer le buffer sur `GL_ELEMENT_ARRAY_BUFFER`,
- ▶ afficher les primitives indexées, `glDrawElements(0)`.

Exemple : creation index buffer

```
unsigned int indices[m]= { ... };  
  
GLuint index_buffer;  
glGenBuffers(1, &index_buffer);  
  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,  
             index_buffer);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(  
            unsigned int)*m, indices, GL_STATIC_DRAW);
```

Exemple : dessiner avec un index buffer

```
glBindBuffer( GL_ARRAY_BUFFER, buffer );  
glEnableVertexAttribArray( attribut );  
glVertexAttribPointer( attribut, 3, GL_FLOAT,  
    sizeof(Points), 0 );  
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER,  
    index_buffer );  
glDrawElements( primitive, 24, GL_UNSIGNED_INT,  
    0 );
```

Buffers : résumé

résumé :

- ▶ un buffer est un bloc de données,
- ▶ on peut stocker n'importe quoi dedans,
- ▶ interpretation du contenu lors de l'utilisation, cf. `glDrawElements`, `glVertexAttribPointer`,
- ▶ l'organisation des données est très souple : cf. paramètres `offset`, `stride`.

Buffers : résumé

exercice :

- ▶ construire un seul buffer contenant les positions et les autres attributs de chaque sommet,
- ▶ construire un seul buffer contenant les attributs de sommets et les indices,
- ▶ vérifier que l'affichage est correct.

Visibilité

rappel : pipeline graphique

- ▶ lancer de rayons,
- ▶ REYES (Renderman),
- ▶ rasterisation (OpenGL / DirectX).

Pipeline graphique : rasterisation

visibilité :

- ▶ pour chaque objet :
- ▶ pour chaque primitive de la surface de l'objet :
- ▶ déterminer les pixels sur lesquels se projette la primitive,
- ▶ + étapes suivantes du pipeline.

ne conserver que la couleur de la primitive la plus proche de l'observateur ?

Pipeline graphique : rasterisation

Z-buffer :

image de profondeur pour conserver le point de l'objet le plus proche de l'observateur.

les objets sont dessinés un par un, dans un ordre quelconque, mais l'image et le Z-buffer conservent la couleur et la profondeur du point (de l'objet) le plus proche (vu à travers le pixel).

Z-Buffer

utiliser un Z-Buffer :

- ▶ le z-buffer est créé en même temps que la fenêtre d'affichage,
- ▶ cf. paramètres de création du contexte.

avec SDL :

```
SDL_Init(SDL_INIT_VIDEO);  
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);  
  
screen= SDL_SetVideoMode(width, height, 0, ...  
    SDL_OPENGL);
```


Z-Buffer

activer / désactiver le test de visibilité :

- ▶ `glEnable(GL_DEPTH_TEST);`
- ▶ `glDisable(GL_DEPTH_TEST);`

modifier le test

```
glDepthFunc( );
```

Rasterisation

quelles transformations subit un sommet ? une primitive ?

comment déterminer si un sommet $v(x, y, z, 1)$ est visible ?

rappel :

$$p_h = T v_h = P(V(Mv))$$

p_h est visible si :

$$-w_h < x_h < w_h$$

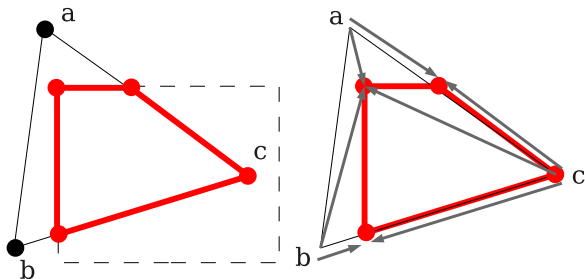
$$-w_h < y_h < w_h$$

$$-w_h < z_h < w_h$$

Rasterisation

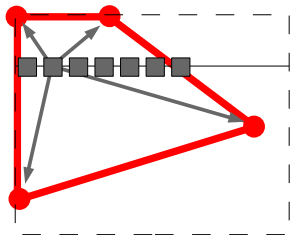
et si un sommet d'une primitive n'est pas visible ?

OpenGL crée un nouveau sommet visible et interpole tous les attributs.



Rasterisation

remplissage de l'intérieur de la primitive par interpolation des attributs des sommets, couleur, normale, etc.



Rasterisation - Algorithme

comment dessiner un triangle ?

Incremental and Hierarchical Hilbert Order Edge Equation Polygon
Rasterization

intuition :

- ▶ construire un tétraèdre avec le triangle comme base et la caméra comme origine,
- ▶ un pixel appartient au triangle si le rayon associé est à l'intérieur des 4 plans portants les faces du tétraèdre,
- ▶ calcul incrémental simple pour évaluer rapidement les 4 équations de plans pour des pixels voisins ?

comment ne pas dessiner un triangle ?

Rasterisation - Algorithme

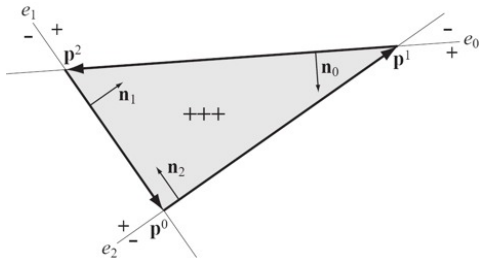
après simplification, 3 équations par triangle :

- ▶ $E_i(x, y) = a_i x + b_i y + c_i$
- ▶ le pixel (x, y) est à l'intérieur du triangle :
- ▶ si $E_i(x, y) > 0$ pour $i = 0, 1, 2$.

les équations correspondent aux arêtes du triangle projeté.

Rasterisation - Algorithme

$$E_i(x, y) = a_i x + b_i y + c_i = \vec{n}_i \cdot (x, y) + c_i$$

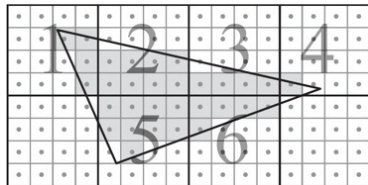
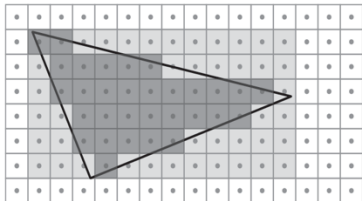


pour 2 points $p(p_x, p_y)$ et $q(q_x, q_y)$:

$$E_{pq}(x, y) = -(q_y - p_y)(x - p_x) + (q_x - p_x)(y - p_y)$$

Rasterisation - Algorithme

pour quels pixels de l'image évaluer les 3 équations ?



évaluation incrémentale :

- ▶ $E_i(0, 0) = c_i$,
- ▶ $E_i(x + s, y + t) = E_i(x, y) + sa_i + tb_i$.

comment déterminer qu'un bloc intersecte le triangle ?

Rasterisation - Algorithme

comment interpoler les attributs des sommets ?



détails dans l'article.