

M2-Gamagora

Rendu temps réel - OpenGL 2 et 3

J.C. lehl

September 21, 2010

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).

exemple (transformations) :

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();
```

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();
```

Présentation de l'api

les fonctions de l'api sont nommées en fonction du type de leur paramètres.

exemple :

```
glTranslatef( GLfloat x, GLfloat y, GLfloat z );  
glTranslated( GLdouble x, GLdouble y, GLdouble z );
```

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

OpenGL utilise seulement 3 matrices :

- ▶ `GL_MODELVIEW` : objet \rightarrow scène \rightarrow caméra,
- ▶ `GL_PROJECTION` : définit la projection de la caméra,
- ▶ `glViewport(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 ?

$$p_2 = T_{12}p$$

$$p_3 = T_{23}p_2$$

$$p_3 = T_{23}(T_{12}p)$$

$$p_3 = T_{23}T_{12}p$$

$$p_3 = Tp \text{ avec } T = T_{23}T_{12}$$

Compositions de transformations

et dans l'autre sens ?

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

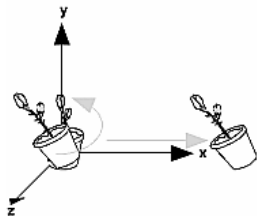
repère 1 ← repère 2 ← repère 3
 $(T_{12})^{-1}$ $(T_{23})^{-1}$

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

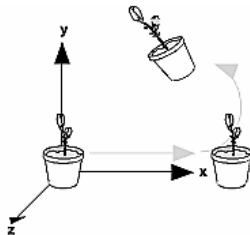
Transformations : Bilan

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 (Viewport),
- ▶ 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),

construire MODELVIEW pour placer et orienter les objets dans le repère de la caméra.

Composition de transformations

transformations :

- ▶ `glLoadIdentity();`
- ▶ `glTranslatef(x, y, z);`
- ▶ `glRotatef(angle, x, y, z);`
- ▶ `glScalef(x, y, z);`

chaque transformation T se compose avec la matrice active A :
 $A' = AT$.

Compositon : exemple

exemple :

```
glRotatef(angle_x, 1.0, 0.0, 0.0);  
glRotatef(angle_y, 0.0, 1.0, 0.0);  
glRotatef(angle_z, 0.0, 0.0, 1.0);  
glTranslatef(x, y, z);
```

quelle matrice est modifiée ?

quel est l'ordre des transformations ? XYZT ou TZYX ?

Exemple :

```

// 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);

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

```

"Charger" directement une matrice

charger une matrice :

- ▶ `glLoadTransposeMatrixf(GLfloat * m);`
- ▶ `glMultTransposeMatrixf(GLfloat * m);`

remarque :

par défaut, OpenGL manipule la transposée des matrices,
et `glLoadMatrixf(T)` attends la transposée de la matrice "directe".

donc : `glLoadTransposeMatrixf()` fait bien ce que l'on veut ...

Affichage

plusieurs solutions :

- ▶ mode immédiat : simple mais lent,
- ▶ mode "tableau" : plus efficace,
- ▶ mode "tableau" indexé : encore plus efficace,
- ▶ mode "tableau" indexé sur gpu : le plus performant.

Mode immédiat

- ▶ `glBegin(primitive);`
- ▶ `glVertex(x, y, z);`
- ▶ ...
- ▶ `glEnd();`

attributs :

- ▶ `glColor(r, g, b);`
- ▶ `glNormal(x, y, z);`
- ▶ `glTexCoord(s, t);`

Mode immédiat

attention :

- ▶ décrire tous les attributs du vertex avant le `glVertex()` ;
- ▶ sinon : le dernier attribut spécifié est utilisé.

pratique dans certains cas :

donner la même normale à tous les sommets d'un triangle, par exemple.

Mode "tableau"

- ▶ stocker les positions des sommets dans un tableau,
- ▶ activer l'utilisation d'un tableau de sommets, `glEnableClientState()`,
- ▶ associer le tableau de sommets au contexte OpenGL, `glVertexPointer()`,
- ▶ afficher les primitives, `glDrawArrays()`.

```
VEC3 sommets[n] = { ... };
```

```
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer( 3, GL_FLOAT, sizeof(VEC3),  
    sommets);  
glDrawArrays( primitive, 0, n );
```


Mode "tableau" indexé

- ▶ stocker les positions et les index dans 2 tableaux,
- ▶ activer et associer les tableaux,
- ▶ afficher les primitives, `glDrawElements()`.

```
VEC3 sommets[n]= { ... };  
int indices[m]= { ... };
```

```
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer( 3, GL_FLOAT, sizeof(VEC3),  
    sommets);  
glDrawElements( primitive, m, GL_UNSIGNED_INT,  
    indices );
```

Vertex Buffer Object

même principe, mais les données sont chargées sur le gpu.

- ▶ créer les buffers (positions + indices),
- ▶ allouer les buffers,
- ▶ transférer les données,
- ▶ activer et associer les buffers,
- ▶ afficher avec `glDrawElements(... NULL)`.

Vertex Buffer

```
VEC3 sommets[n]= { ... };  
  
GLuint vertex_buffer;  
glGenBuffers(1, &vertex_buffer);  
  
glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer);  
glBufferData(GL_ARRAY_BUFFER, sizeof(VEC3)*n,  
             sommets, GL_STATIC_DRAW);
```

Index Buffer

```
int indices[m]= { ... };

GLuint index_buffer;
glGenBuffers(1, &index_buffer);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,
             index_buffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(int)
             )*m, indices, GL_STATIC_DRAW);
```

Affichage avec vertex buffer et index buffer

```
glEnableClientState(GL_VERTEX_ARRAY);  
glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer);  
glVertexPointer(3, GL_FLOAT, 0, NULL);  
  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,  
             index_buffer);  
glDrawElements(primitive, m, GL_UNSIGNED_INT,  
               NULL );
```

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 :

$$-w_h < x_h < w_h$$

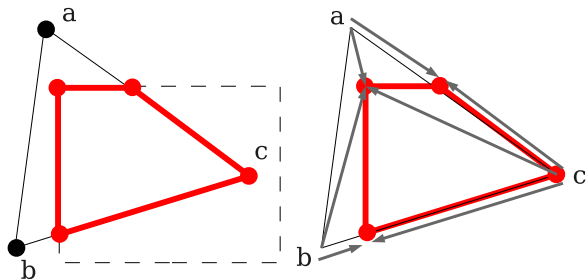
$$-w_h < y_h < w_h$$

$$-w_h < z_h < w_h$$

Rasterisation

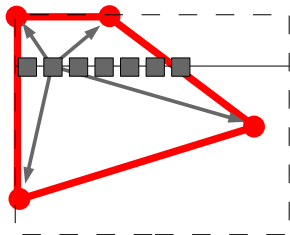
et si un sommet d'une primitive n'est pas visible ?

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

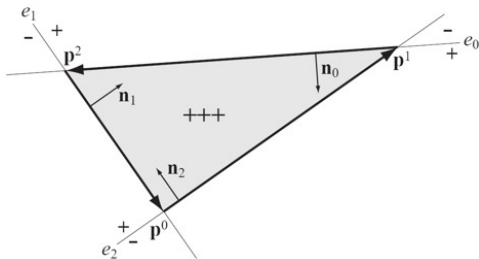
après simplification, 3 équations par triangle :

- ▶ $E_i(x, y) = a_i x + b_i y + c_i$
- ▶ 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

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

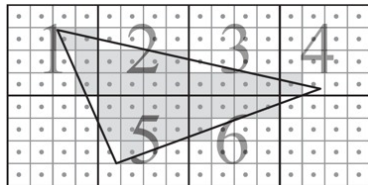
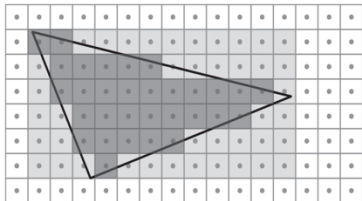


pour 2 points $p(p_x, p_y)$ et $q(q_x, q_y)$:

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

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.