Université Gustave Eiffel — INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Master Informatique 1

TD Infographie n°5

Transformations & Coordonnées Homogènes –suite

Présentation des matrices homogènes et routines de tranformation proposées par OpenGl[a] *et* libg3x.

[a] cf. https://www.khronos.org/opengl/wiki/Viewing_and_Transformations

① La méthode OpenGl.

OpenGl propose nativement un système de gestion des transformations très complet, mais assez complexe. Ne seront présentés ici que les éléments indispensables. En particulier l'impasse sera faite sur la question des matrices de projection (`glMatrixMode(GL_PROJECTION)`, pour les curieux), pour se concentrer sur les transformations associées aux objets (`glMatrixMode(GL_MODELVIEW)`).

C'est la sur-couche libg3x qui gère en amont la question des projections et de la caméra.

En outre le mode de gestion de ces transformations à considérablement évolué dans les versions plus récentes d'OpenGl. Les éléments présentés ici correspondent à la version "primaire" d'OpenGl, proposée par freeglut3, bien suffisante pour en comprendre le principe.

Dans la plupart des cas simples, on ne manipule pas directment les matrices OpenGl. On fait appel directement aux routines de transformation. Il faut néanmoins en connaître le principe.

ⓐ le format :

Tout d'abord, comme pour les points, les vecteurs ou les couleurs, OpenGl travaille directement avec des tableaux de réels (`float` ou `double`).

Une matrice OpenGl est donc brutalement un tableau de $16 = (3+1)*(3+1)$ réels (`double glmat[16];`) stockés par colonnes.

Les 4 premières valeurs correspondent à la première colonne, etc... :

$$\left|\begin{array}{cccc} m_{00} & m_{04} & m_{08} & m_{12} \\ m_{01} & m_{05} & m_{09} & m_{13} \\ m_{02} & m_{06} & m_{10} & m_{14} \\ \hline m_{03} & m_{07} & m_{11} & m_{15} \end{array}\right|$$

Ainsi, la matrice d'une translation de vecteur $\overrightarrow{\mathsf{T}}(\mathsf{a},\mathsf{b},\mathsf{c})$, serait

$$\left|\begin{array}{ccc|c} 1 & 0 & 0 & a \\ 0 & 1 & 0 & c \\ 0 & 0 & 1 & c \\ \hline 0 & 0 & 0 & 1 \end{array}\right| \Leftrightarrow \texttt{double Tmat[16]=\{1,0,0,0,0,1,0,0,0,0,1,0,a,b,c,1\};}$$

ⓑ la pile :

OpenGl gère l'enchaînement des transformations via une pile de matrices (nous, nous utiliserons plutot des arbres...).

☞ la matrice courante est toujours celle constituée par la multiplication (à gauche) de toutes les matrices de la pile.

☞ si la pile contient 5 matrices (du fond M1 vers le sommet M5) la matrice courante est le produit $\mathsf{C} = \mathsf{M5} * \mathsf{M4} * \mathsf{M3} * \mathsf{M2} * \mathsf{M1}$.

☞ par défaut, la pile contient toujours au moins la matrice idéntité. Pour la réinitialiser : `glLoadIdentity()`.

☞ en mode `MODEL_VIEW`, la pile supporte jusqu'à 32 matrices.

☞ appliquer une nouvelle transformation localement sur un objet revient donc à

① "ouvrir" la pile - `glPushMatrix()`,
② charger la transformation voulue (via la fonction OpenGl adéquate),
③ appeler la routine d'affichage de l'objet,
③ et enfin dépiler cette dernière matrice `glPopMatrix()`.
☞ la pile retrouve alors son état d'avant la transformation.

Par exemple, pour translater d'un vecteur $\overrightarrow{T}(a,b,c)$, un triangle `<P[0],P[1],P[2]>`, et uniquement celui-là, l'enchaînement serait :

```
01| glPushMatrix();                    /* ouverture de la pile                     */
02|   glTranslatef(a,b,c);             /* chargement de la matrice de translation  */
03|   glBegin(GL_TRIANGLES)            /* routine d'affichage du triangle          */
04|     glNormal3dv(N);                /* une normale globale pour tout le triangle */
05|     glVertex3dv(P[0]);             /* les 3 vertex, dans le bon ordre          */
06|     glVertex3dv(P[1]);
07|     glVertex3dv(P[2]);
08|   glEnd();                         /* fin de l'affichage                       */
09| glPopMatrix();                     /* déchargement de la translation           */
```

☞ Le code d'exemple `<demo_scene.c>` fourni avec la libg3x illustre cela avec plusieurs niveaux d'empilement sur des objets composites (tables, chaises...). Il faut aller l'étudier.

ⓒ les transformations usuelles :

Chaque transformation élémentaire possède sa propre routine de chargement :

- homothétie de rapports (h_x,h_y,h_z) : `glScalef(hx,hy,hz);` – ou `glScaled` selon le type
- rotation d'angle θ autour d'un axe $\overrightarrow{a}(x_a,y_a,z_a)$: `glRotatef(theta,xa,ya,za);` – ou `glRotated`
- translation de vecteur $\overrightarrow{T}(a,b,c)$: `glTranslatef(a,b,c);` – ou `glTranslated`

On peut appliquer plusieurs transformations à la fois. La matrice effectivement chargée est alors le produit **à gauche** des matrices dans l'ordre d'apparition.

☞ Conseil : bien que rien ne l'impose, il est fortement recommandé de procéder toujours dans le même ordre : d'abord l'homothétie, puis les rotations[(1)] (autours des axes, dans l'ordre x, y puis z) et enfin la translation.
Par exemple, pour enchaîner, sur une sphère, l'ensemble de ces transformation, on aurait :

```
01| glPushMatrix();                    /* ouverture de la pile                      */
02|   glTranslatef(tx,ty,tz);          /* la translation T en dernier               */
03|   glRotatef(dz,0.,0.,1.);          /* la rotation Rz d'angle dz autour de l'axe z */
04|   glRotatef(dy,0.,1.,0.);          /* la rotation Ry d'angle dy autour de l'axe y */
05|   glRotatef(dx,1.,0.,0.);          /* la rotation Rx d'angle dx autour de l'axe x */
06|   glScalef(hx,hy,hz);              /* l'homothétie H en premier                 */
07|   glutSolidSphere(1.,20,20);       /* une sphere glut de rayon 1.               */
08| glPopMatrix();                     /* déchargement des 5 matrices T,Rz,Ry,Rx,H  */
```

☞ si C est la matrice courante avant l'appel à `glPushMatrix()`, la matrice effectivement appliquée sur la sphère est $(C*T*Rz*Ry*Rx*H)$.

☞ Pour les 3 rotations, surtout ne pas faire `glRotatef(1,ax,ay,az);` !!

ⓓ utiliser ses propres matrices :

Jusqu'ici nous n'avons pas explicitement utilisé de matrice. Cela est possible grâce à la fonction `glMultMatrixf(const float *m)` (ou `glMultMatrixd(const double *m)`)

L'enchaînement suivant applique la matrice M prédéfinie (`double M[16];`) à une sphère.

```
01| glPushMatrix();                    /* ouverture de la pile                     */
02|   glMultMatrixd(M);                /* chargement de la matrice M               */
03|   glutSolidSphere(20,20,1.);       /* une sphere glut de rayon 1.              */
04| glPopMatrix();                     /* déchargement de la matrice M             */
```

☞ si C est la matrice courante avant l'appel à `glPushMatrix()`, la matrice effectivement appliquée sur la sphère est $(C*M)$.

(1) OpenGl exprime les rotations en degrés

ⓔ quid des normales ? :

On a vu précédemment que la gestion des normales nécessite quelques précautions lorqu'on applique des homothéties (qui ne sont pas des isométries).

☞ Pas de souci ici, OpenGl gère ça tout seul et les normales sont corretement transformées.

② La sur-couche libg3x.

OpenGl n'est pas une lib permettant de faire de l'algorithmique, c'est juste une lib permettant de faire de l'affichage très efficacement. Son utilisation doit donc se restreindre aux fonctionalités de dessin.

La sur-couche libg3x sert d'interface entre les aspects algorithmiques et OpenGl, quitte à en brider de nombreuses fonctionalités (on fait de la prog. pas du graphisme).

Comme pour les points, vecteurs, couleurs, la libg3x propose un module de gestion des transformations un peu différent (mais compatible OpenGl) permettant en particulier de manipuler réellement les matrices (directes, inverses, normales) et de les utiliser pour des algorithmes graphiques plus élaborés (OpenGl ne servant que de 'moteur de rendu').

Le module `<g3x_tranfo>` regroupe ces fonctionalités.

Il définit en particulier un type `typedef struct{ double m[16]; } G3Xhmat;` (la matrice OpenGl est donc 'encapsulée' dans une structure).

Il fournit également des fonctions de construction des matrices classiques (identité, homothétie, rotation, translation) ainsi que les opération de produit classiques `G3Xhmat*G3Xpoint`, `G3Xhmat*G3Xvector` et `G3Xhmat*G3Xhmat`.

Les exemples présentés en section ① se traduiraient ainsi, avec la libg3x :

ⓐ
```
G3Xpoint  P[3];                          /* 3 vertex à construire                          */
G3Xvector N;                             /* une normale à construire                       */

G3Xhmat   Mt = g3x_Translation3d(a,b,c); /* création de la matrice de translation T(a,b,c) */
g3x_Material(G3Xwa,0.1,0.6,0.9,0.99,0.); /* couleur/matiere -- cf. TD4-illumination        */
glPushMatrix();                          /* ouverture de la pile                           */
  glMutMatrixd(Mt.m);                    /* chargement du bloc de 16 valeurs de la matrice */
  glBegin(GL_TRIANGLES)                  /* routine d'affichage du triangle                */
    g3x_Normal3dv(N);                    /* une normale globale pour tout le triangle      */
    g3x_Vertex3dv(P[0]);                 /* les 3 vertex, dans le bon ordre                */
    g3x_Vertex3dv(P[1]);
    g3x_Vertex3dv(P[2]);
  glEnd();                               /* fin de l'affichage                             */
glPopMatrix();                           /* déchargement de la translation                 */
```

ⓑ
```
G3Xhmat M;                                        /* matrice GLOBALE             */
/* ici on multiplie avec M à droite => ordre inverse par rapport à OpenGl        */
M = g3x_Identity();
M = g3x_Mat_x_Mat(g3x_Homothetie3d(hx,hy,hz) ,M); /* 1°) l'homothétie            */
M = g3x_Mat_x_Mat(g3x_RotationX(rx)          ,M); /* 2°) la rotation autour de x */
M = g3x_Mat_x_Mat(g3x_RotationY(ry)          ,M); /* 3°) la rotation autour de y */
M = g3x_Mat_x_Mat(g3x_RotationZ(rz)          ,M); /* 4°) la rotation autour de z */
M = g3x_Mat_x_Mat(g3x_Translation3d(tx,ty,tz),M); /* 5°) la translation          */

g3x_Material(G3Xwa,0.1,0.6,0.9,0.99,0.); /* couleur/matiere -- cf. TD4-illumination        */
glPushMatrix();                          /* ouverture de la pile                           */
  glMutMatrixd(M.m);                     /* chargement du bloc de 16 valeurs de la matrice */
  glutSolidSphere(1.,20,20);             /* une sphere glut de rayon 1.                    */
glPopMatrix();                           /* déchargement de la matrice                     */
```

☞ attention : libg3x exprime les rotations en radians contrairement à OpenGlqui travaille en degrés.

☞ attention : l'ordre des transformations serait le même qu'en OpenGl si on multpliait avec `M` à gauche `M = g3x_Mat_x_Mat(M,g3x_Transfo(...));`

☞ les deux codes exemples fournis `<gltransfo.c>` et `<g3xtransfo.c>` mettent en oeuvre ces 2 approches.

La libg3x ne change donc pas grand chose à la struture générale des routines d'affichage OpenGl. Mais les algorithmes en amont sont plus "confortables".