UJF - M1 Informatique 2005-2006 : Infographie
TP4 : Les repères

21 octobre 2005

Aujourd'hui nous allons étudier openGL et ses fonctions de transformations de repères dans le cadre d'une animation d'un système solaire.

1 Transformations de repères

Récupérez le nouveau fichier `main.cpp`, `Astre.h`, `systeme.h` et `systeme.cpp`.

- `Astre.h` : défini un astre (rayon, couleur, vitesse de rotation, les satellites, etc.)
- `systeme.cpp` : défini un système solaire simple formé d'un soleil et d'une planète
- `main.cpp` : les procédures importantes sont dessinerSysteme, dessinerAstre et animer

1.1 Repère en openGL

Dans ce TP nous utiliserons une petite partie des fonctionnalités des transformations des repères offertes par openGL. La procédure dessiner est appelée à chaque fois qu'openGL redessiner la fenêtre.

A l'entrée dans cette fonction, la caméra vise le centre de la scène (le point 3D `(0,0,0)` en absolu) avec un peu de recul.

1. Complétez la procédure `dessinerRepere` qui dessine 3 vecteurs de couleur différentes (pour les différencier) représentant le repère du monde
2. Dans la suite du TP, pour vous aider, vous pourrez dessiner des repères car ils donnent directement la position du repère courant dans l'espace.

1.2 Transformations

En openGL, vous pouvez utiliser 3 types transformations de repères :

1. Homotétie : `glScalef( cx, cy, cz )`
2. Translation : `glTranslatef( tx, ty, tz )`

3. Rotation : `glRotatef( a, x, y, z )` (fait tourner le repère d'un angle a (en degrés) autours des vecteurs désirés : `glRotatef( 45, 1, -1, 0 )` tourne de 45 degrés autours de x et de -45 autours de y)

Il est **indispensable** d'utiliser une feuille de brouillon pour représenter les repères et les transformations. Pour ceci vous pouvez représenter les transformations comme des transitions entre repères en y indiquant les paramètres et le type de transformation.

Dans la procédure dessinereTranfs, décommentez les 2 lignes pour dessiner un second repère plus loin.

1. Essayez les 2 autres transformations
2. Combinez des transformations (par exemple une rotation et une translation) : l'ordre importe t-il ?

Si vous voulez dessinere un troisième repère après le deuxième, openGL "partira" bien sûr du dernier repère, vous pouvez ainsi appliquer une suite de transformations.

1. Comment pourrait-on faire pour revenir en arrière, dans le repère d'origine ? Est-ce pratique ?

Pour palier cette lourdeur, vous pouvez (et devez !) utiliser une hiérarchie de repères grâce aux procédures `glPushMatrix()` et `glPopMatrix()`. De manière simplifiée, `glPushMatrix` "enregistre" (empile) le repère courant (au moment où l'on y fait appel. Après avoir fait plusieurs transformations, `glPopMatrix` raménera openGL au **dernier** repère enregistré. OpenGL empile tous ces repères, ainsi vous pouvez cumuler les `glPushMatrix` pour enregistrer et dépiler au fur et à mesure vos différents repères ; et ainsi créer une structure complexe de hiérarchie de repères.

1.3 Dessiner le Soleil, Mars et la Terre - hiérarchie de repères

La procédure `glutSolidSphere` dessine une sphère au centre du repère (en (0,0,0)) avec pour rayon le premier paramètre.

1. Compilez le programme et exécutez le avec le paramètre "-t 1" pour afficher le système solaire simplifié. Pour l'instant seulement le soleil est affiché.
2. Dans `dessinerSysteme`, complétez la partie "Méthode brute" sans utiliser les Push/PopMatrix pour placer la Terre au bon endroit. La procédure `dessinerAstre` dessine l'astre en fonction de son rayon, vous n'avez plus qu'à placer la Terre à une distance terre->d par exemple sur l'axe des x (idem pour Mars).

Maintenant passons à une méthode plus classe, complétez la procédure `dessinereAstre` pour qu'elle dessine (récursivement grâce à une hiérarchie de repères) ses satellites (considérés comme des astres). Les pointeurs vers les satelittes sont stockés dans le tableau a->sat.

1. Dessinez la hiérarchie des repères. Utilisez les Push/PopMatrix pour la traduire en openGL.
2. La caméra est un peu proche par défaut, en utilisant simplement les repères, comment pouvez faire que globalement le système soit plus petit ?

La procédure `animer` fonctionne de la même manière qu'`animerSinus` des TP précédents. Elle fait tourner les astres en fonction du temps en modifiant `selfa` et `a` qui représente respectivement l'angle de rotation sur elle même de l'astre et l'angle de rotation de l'astre autours de son "père" (le père de la Terre est le Soleil) - par rapport à l'instant de départ.

1. Implémentez la rotation des astres autours de leur père.
2. La rotation propre d'un astre n'influe pas sur ses satellites (càd que la rotation propre du Soleil n'influe pas sur la rotation de la Terre autours de lui). Comment traduisez vous cette contrainte en openGL ?

1.4 Système plus complexe

Ajoutez d'autres planètes et la Lune (cf. `systeme.cpp`) qui possède un angle `a2` non nul (c'est-à-dire qu'elle n'est pas dans le plan de l'écliptique). Il est à noter que les vitesses angulaires peuvent être négatives, ainsi l'astre tournera dans le sens contraire des aiguilles d'une montre.

2 Simuler un changement de caméra

Nous souhaitons placer la caméra à la surface de la Terre, sur l'équateur. Cela signifie que le repère originel est maintenant situé à la surface de la Terre.

1. Imaginez vous comme étant ce repère : comment (par quelles transformations) pouvez vous vous déplacer jusqu'au repère du Soleil ?
2. Visualisez le repère de la future caméra (où elle se trouvera) dans l'animation.
3. En utilisant "./run -t 2" nous définissons par défaut une caméra fixe. La position de cette caméra est définie dans la procédure `placerCaméra`. La caméra openGL regarde toujours vers les Z négatifs, et est au départ placée en (0,0,0) : c'est pour cela que nous reculons de 5 pour nous placer dans le repère du Soleil : nous pouvons ainsi le voir.
4. Commentez le glTranslatef. Ecrivez la suite de transformation nécessaire pour vous placer sur la surface de la Terre (n'oubliez pas que la caméra regarde vers les Z négatifs !).