Maths pour l'image : algèbre linéaire et géométrie

Projet de programmation : transformations 3D en Java

1 But du projet

Ce projet consiste à programmer une petite application capable de gérer la visualisation de transformations géométriques 3D en utilisant des vecteurs et des matrices d'applications linéaires. En d'autres termes, il s'agit de développer un "moteur 3D" de base (à la façon de bibliothèques graphiques connues, comme OpenGL ou Direct3D), permettant de visualiser une figure 3D de base et d'y appliquer rotations, translations, etc. ainsi que de déplacer la "caméra".

2 Travail demandé

Le principe de l'application est le suivant : elle comporte une fenêtre de visualisation de la scène 3D, et un panneau de contrôle permettant de "piloter" la scène 3D.

La scène sera constituée d'un ou plusieurs solides de l'espace (cube, pyramide, prisme, polyèdre...). La fenêtre de visualisation correspond à la position de ce qu'on appelle la "caméra" : les coordonnées "réelles" de la scène 3D sont projetées sur le plan de l'écran de l'ordinateur pour être transformées en coordonnées de la fenêtre Java.

Pour cela il faudra tout d'abord implémenter ce qui est nécessaire pour gérer la représentation de la scène, de préférence dans un package spécifique. Il faut pouvoir :

- Représenter un vecteur (qui correspond à un point de l'espace avec quatre coordonnées dans le système de coordonnées homogènes)
- Représenter une transformation géométrique par une matrice 4x4 de réels. (Penser notamment aux transformations spéciales que sont la rotation, l'homothétie, la projection, la translation)
- Permettre la composition des transformations, et l'application de celles-ci à un vecteur
- Représenter un solide de l'espace. Ce n'est rien d'autre qu'un ensemble de points représentant les sommets du solide, et des couples de points représentant les arêtes du solide. Le minimum demandé est une scène constituée d'un unique solide
- Permettre de transformer un solide
- ...

Il faudra ensuite pouvoir contrôler la scène via l'interface graphique. Ceci signifie essentiellement deux choses :

- transformer le(s) solide(s) de la scène par une transformation géométrique donnée : soit par une transformation classique (translation, rotation...), soit directement via la matrice de la transformation qui est saisie à la main par l'utilisateur.
- changer le point de vue de la caméra : zoom (équivalent à une homothétie sur la scène), rotation (équivalent à une rotation d'angle opposé sur la scène), translation (équivalent à une translation de vecteur opposé sur la scène)...

Comme le but du projet n'est pas la conception d'une interface graphique Java, un exemple d'interface est fourni (à récupérer à l'adresse `www.labri.fr/~foucaud/Teaching/MathsImage/projet.tar.gz`). Pour gagner du temps, il est conseillé de l'utiliser comme base (que vous pouvez modifier librement selon vos choix et vos goûts, en le justifiant dans le rapport). Cependant, elle n'est pas forcément la plus ergonomique possible, vous pouvez donc également concevoir une nouvelle interface si vous voulez (à expliquer et justifier également dans le rapport).

La classe TestTransfos est un simple launcher avec une méthode main pour lancer l'application. L'interface donnée est composée d'un package gui (pour graphic user interface). La classe TransfosFrame contient la fenêtre principale de l'application.

La classe TransfosView est le panneau d'affichage : c'est là qu'il va falloir gérer le dessin de la scène via la méthode de base paintComponent(Graphics g) (voir la doc de la classe Graphics pour voir comment tracer des segments, changer la couleur, etc). Attention, les coordonnées dans un JComponent ont pour origine (0,0) le coin haut à gauche du composant, et les "y" s'incrémentent en descendant, les "x" en allant vers la droite.

La classe TransfosControls est la classe du panneau de contrôle. C'est là qu'il va falloir gérer les actions de l'utilisateur et l'interaction avec l'affichage. Pour assigner une action à un composant (par exemple un bouton), il faut lui ajouter un ActionListener défini par exemple de façon anonyme :

```
JButton b = new JButton("bouton");
b.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent arg0) {
    //CODE A METTRE ICI
  }
});
```

L'utilisateur doit pouvoir saisir les valeurs des cases de la matrice de transformation courante, et appliquer la transformation associée en cliquant sur un bouton. Dans l'exemple fourni, ces cases sont représentées par des champs de formulaire de type JTextField : les attributs $m00$, $m01$, ..., $m33$.

Il est de plus demandé de gérer les transformations usuelles (rotation, translation, ...) de façon spécifique. Dans l'interface fournie, à chaque transformation correspond un bouton "A" pour appliquer directement la transformation au(x) solide(s) de la scène, un bouton "C" qui compose la matrice actuelle par la transformation donnée et remplace les valeurs affichées des cases de celle-ci, et "R" qui remplace les cases de la matrice actuelle par celles de cette transformation.

Les boutons correspondant aux transformations de la scène sont gérés dans la méthode transfoPanel.
L'utilisateur doit également pouvoir déplacer l'angle de vue (la caméra) : zoom avant/arrière, rotation selon les 3 axes, translation dans 4 directions. Les boutons correspondants sont créés dans la méthode cameraPanel.
Pour gérer la caméra, il faut transformer les coordonnées de la scène en coordonnées caméra (puis, ces coordonnées doivent être ensuite transformées en coordonnées de la fenêtre Java). Pour ça on peut tenir à jour une transformation (au début, juste une translation de vecteur (0,0,10) si la caméra fait face à la scène et est positionnée à une distance de 10 sur l'axe des z). Cette transformation est ensuite mise à jour à chaque fois qu'on déplace la caméra. Il suffit ensuite d'appliquer cette transformation aux vecteurs de la scène pour obtenir leurs coordonnées dans le repère de la caméra.
Un bouton reset est également proposé, pour remettre la caméra et la figure dans leurs états initiaux.

3 Pour aller plus loin

Une fois les fonctions de base implémentées, n'hésitez pas à en rajouter de nouvelles, cela ne fera qu'augmenter votre note ;) Quelques exemples :

- dessiner les axes du repère de la scène
- gérer plusieurs solides, ajout/suppression de solides...
- zoom et/ou translation de la caméra pilotés par la souris (molette et glissement) en utilisant un MouseWheelListener et/ou un MouseMotionListener
- ...

4 Modalités

Le projet est à rendre pour le 10 décembre 2009 minuit et comptera pour 33% de la note finale. Il sera effectué par groupes de deux étudiants (un unique groupe de trois autorisé si le nombre d'étudiants est impair).

Le projet devra être rendu par e-mail à l'adresse `foucaud@labri.fr` en indiquant bien sûr les noms des étudiants. Indiquez comme objet du mail : [MATHIMAGE] projet nom1 nom2 [nom3]. Cet e-mail devra contenir un fichier archive contenant le code source, ainsi qu'un rapport au format PDF. Ces fichiers auront comme nom nom1_nom2[_nom3].extension.

Le rapport expliquera les choix d'implémentation (classes et méthodes importantes...), les fonctionnalités rajoutées, les difficultés rencontrées...