TD 4 - Questions de visibilité

Xavier Décoret

Résumé

Le but de ce TD est d'implémenter du Hierarchical Frustum Culling

1 Squelette de programme

Récupérez le squelette de programme suivant (si `wget` ne marche pas à cause d'un proxy, tapez l'url dans votre navigateur préféré)

```
mkdir TD4
cd TD4
wget http://artis.imag.fr/~Xavier.Decoret/teaching/inf583/td3/template/A/A.tar.gz
tar zxf A.tar.gz
rm -f A.tar.gz
```

Ce code permet d'obtenir deux viewer qui affichent un modèle simplissime de ville. Le premier (dont la barre de titre de fenêtre marque "Observer's View") affiche la vue d'un observeur qui se balade dans la ville. Appuyez sur la barre d'espace pour utiliser un mode de navigation *fly* plutôt que *trackball*.

Le deuxième viewer affiche une vue du dessus, et montre en jaune le *view frustum* de l'observateur. Ce view frustum est parfois tronqué. Pouvez vous expliquer pourquoi ? Que constatez vous également sur les *near* et *far planes* de la caméra de l'observateur ? Savez vous ce qui se passe ?

2 Exercices

2.1 Retrouver les équations des plans du frustum

Le *view frustum* est définie comme l'intersection de 6 demi-espaces. Dans cette partie, on cherche les équations des plans délimitant ces 6 demi-espaces. Pour cela, il faut retrouver les coordonnées des 8 points définissant la pyramide tronquée qu'est le *view frustum*. Plusieurs facons pour cela :

– OpenGL pur : on utilise *gluUnproject()* sur les points extrêmes du viewport [(0,0,0.0f) à (width(),height(),1.0f)]

– QGLViewer : on utilise les méthodes `inverseCoordinatesOf()` du `camera()->frame()` pour trouver les coordonnées dans le monde des 8 points exprimés dans le repère caméra (ces 8 points y ont des coordonnées simples à base de znear/zfar,aspect et fovy).
– il y a une solution encore plus simple pour les fainéants (ou les malins)...

Pour débugguer cette partie, inspirez vous du code ci-dessous pour visualiser les points :

```
QVector<Vec> points(8);
glPointSize(10.0f);
glBegin(GL_POINTS);
foreach(Vec p,points)
{
  glVertex3fv(p); // la classe Vec sait se comporter comme un const float*
}
glEnd();
```

et utiliser les clippings planes supplémentaires OpenGL pour vérifier les équations.

```
glEnable(GL_CLIP_PLANE0);
glClipPlane(GL_CLIP_PLANE0,...);
```

2.2 Calcul des bounding box

Étendez la classe `City` pour pouvoir obtenir la bounding box de cchaque bâtiment.

2.3 Culling simple

Utiliser les équations des plans calculées précédemment, et les bounding box, pour pour ne pas dessiner les buildings qui sont en dehors du *view frustum*.

2.4 Culling hiérarchique

Utilisez maintenant une version hiérarchique. Pour cela, prenez un nombre d'avenues/rues qui soit dyadique (puissance de 2) et construisez un arbre de boîtes englobantes.