Imagerie Opérationnelle – Immersion virtuelle

Cavelan Aurélien

15/12/2013

Contents

1 Objectfis

Nous disposons seulement de matériel standard. Le but est d'afficher une scène en 3D anaglyph via OpenGL en utilisant un filtre rouge pour l'oeil gauche et un filtre cyan pour l'oeil droite. La même scène est ensuite calculée puis affichée par quatre processus sur quatre fenêtres différentes comme on le ferait sur un cluster.

2 Affichage de la scène

La 3D stéréoscopique est obtenue en calculant un rendu pour l'oeil gauche et un rendu pour l'oeil droit. Afin que le cerveau puisse différencier les deux images on utilise un filtre rouge pour l'oeil gauche et un filtre cyan pour l'oeil droite. L'écartement des deux caméras doit être fixe et à l'horizontale (on suppose que la tête est droite).

La caméra tourne autour du centre de la scène sur une sphère. On peut donc calculer les coordonnées 3D de la caméra à partir de ses coordonnées polaires.

```
float camX = camR * -sinf(phi * (M_PI / 180)) * cosf(theta * (M_PI / 180));
float camY = camR * -sinf(theta * (M_PI / 180));
float camZ = -camR * cosf(phi * (M_PI / 180)) * cosf(theta * (M_PI / 180));
```

Il est aussi nécessaire de calculer le vecteur *Up* de la caméra afin de le passer à la fonction gluLookAt(). Ce vecteur représente l'orientation de la caméra (et donc la *tête* de l'observateur). Afin d'obtenir la position des deux caméra pour réaliser la 3D stéréoscopique il suffit donc de:

- Calculer la *position* de la caméra partir des coordonnées polaires
- Calculer et normaliser le vecteur *Up* de la caméra
- Calculer le vecteur *ortho* orthogonal à *Up*
- Placer la caméra de l'oeil gauche sur `-delta * ortho + position`
- Placer la caméra de l'oeil droit sur `+delta * ortho + position`

3 Affichage sur 4 fenêtres avec MPI

Au niveau de la scène, chaque processur doit effectuer le rendu d'une partie de l'écran. Il faut donc redéfinir le *frustum* avec la fonction glFrustum(). On identifie le processus maître par son rang 0.

```
if(rank == 0)       // upper left
    glFrustum(-.5, 0.0, 0.0, .5 * aspect_ratio, 0.5f, 200.0f);
else if(rank == 1)  // upper right
    glFrustum(0.0, .5, 0.0, .5 * aspect_ratio, 0.5f, 200.0f);
else if(rank == 2)  // lower left
    glFrustum(-.5, 0.0, -.5 * aspect_ratio, 0.0, 0.5f, 200.0f);
else if(rank == 3)  // lower right
    glFrustum(0.0, .5, -.5 * aspect_ratio, 0.0, 0.5f, 200.0f);
```

Afin que les rendus soient synchronisés on utilise une barrière de synchronisation (MPI_Barrier) juste avant l'appel à `glutSwapBuffers()`.

Pour ce qui est des évènements souris / clavier, on choisi un processus maître qui reçoit les entrées utilisateurs et qui les transmet aux autres processus via MPI à chaque itérations en utilisant la fonction `MPI_Bcast()`.

4 Retier les décorations des fenêtres

Il n'est actuellement pas possible de retirer les décorations des fenêtres en utilisant l'implémentation de glut sous linux. Il est possible d'utiliser directement la bibliothèque X11 (sans utiliser glutCrateWindow()), mais le plus simple est de passer par une autre bibliothèque.

La bibliothèque **glfw**: [http://www.glfw.org/] qui vient de passer en version 3 offre plus de libertées à ce niveau. Elle est open-source (license MIT), très simple (un peu à l'image de glut), moderne, bien documentée et fonctionne sous Mac, Linux et Windows.

Elle est disponible sous forme de paquet pour la plupart des distributions, sauf ubuntu où elle n'est pas encore passée en version 3.

Pour compiler la version utilisant la glfw dans le dossier canevampi:

```
Si installé sur le système:
    mpic++ -o princ glfwpinc.cpp -lX11 -lXi -lXmu -lglut -lGL -lGLU -lm -lglfw
En local dynamique:
    mpic++ -o princ glfwpinc.cpp -lX11 -lXi -lXmu -lglut -lGL -lGLU -lm -I. ./libglfw.so
En local statique (il manque peut-être des dépendances):
    mpic++ -o princ glfwpinc.cpp -lX11 -lXi -lXmu -lglut -lGL -lGLU -lm -lXrandr -I. libglfw3.a
```