TP 7

Site: lms.univ-cotedazur.fr
Cours: Realite virtuelle - EIMAD919
Livre: TP 7

Imprimé par: Theo bonnet
Date: vendredi 28 février 2020, 14:59

Description

Table des matières

1. Caméra subjective

Présentation

Dans une caméra subjective, la position de la caméra représente la position du joueur et le joueur oriente la direction vers laquelle il regarde (gauche/droite ou haut/bas) c'est à dire qu'il définit la position de la cible de la caméra. Ensuite s'il décide d'avancer, le mouvement de la caméra se fait dans la direction de la cible.
Concrètement on crée une classe `RVSubjectiveCamera` qui hérite de `RVStereoCamera` :

- a deux variables membres de type float appelées `m_yaw` (pour *lacet*) qui est un angle autour de l'axe *y* et correspond au mouvement du corps gauche/droit et `m_pitch` (pour le *tangage*) qui est l'angle haut/bas
- ces deux variables (initialisées à 0) dans le constructeur, servent dans une méthode privée `updateTarget()` qui recalcule la valeur de `m_target` en fonction de la position de la caméra subjective (un peu comme dans la caméra sphérique c'était la position qui était recalculée en fonction du changement des angles et à partir de la cible).

```
void RVSubjectiveCamera::updateTarget()
{
    float x = qCos(m_pitch) * qCos(m_yaw);
    float y = qSin(m_pitch);
    float z = - qCos(m_pitch) * qSin(m_yaw);
    this->setTarget(m_position + QVector3D(x,y,z));
}
```

- ajouter les accesseurs et les mutateurs (en faisant en sorte que `updateTarget()` soit appelé).
- méthode publique `move(float d)` qui met à jour la variable `m_position` en se déplaçant d'une distance `d` dans la direction de `m_target` ; si P est la position et T est la cible :

$$P = P + d \cdot \overrightarrow{PT}$$

  Sauf que, comme on ne veut pas s'envoler, on laisse constante la composante y de P même si on regarde vers le haut. Puis il faut appeler `updateTarget()`.
- méthode publique `turn(float angle)` qui change la variable de lacet de la caméra avec `setYaw`,
- méthode publique `tilt(float angle)` qui change la variable de tangage de la caméra avec `setPitch`,
- enfin il faut déclarer la méthode `setPosition` comme étant *virtuelle* dans `RVCamera` pour pouvoir la surcharger ici en appelant aussi `updateTarget()` (de même que dans la caméra sphérique on avait fait virtuelle `setTarget` pour ouvoir appeler `updatePosition()`).

Utilisation de la caméra

Déplacement dans l'IUT : dans `RVWidget` :

- Ajouter une `RVSubjectiveCamera` appelée `m_camera2`
- Initialiser cette camera dans `initializeGL()` avec les attributs suivants :
  - zMax à 2000
  - position à (0, 50, -100)
  - fov à 45 degrés
- Passer `m_camera2` a `m_batiment` et aux autres objets de la scène au lieu de `m_camera`
- Dans `mouseMoveEvent` utiliser dx et dy pour incrémenter/décrémenter les paramètres tangage et lacet de la caméra subjective avec `turn` et `tilt`.
- Dans `keyPressEvent` utiliser les touches Up et Down pour avancer et reculer (ou d'autres touches si vous préférez) en utilisant les méthodes `move` et `turn`. Eventuellement vous pouvez utiliser la manette pour faire tous ces mouvements.

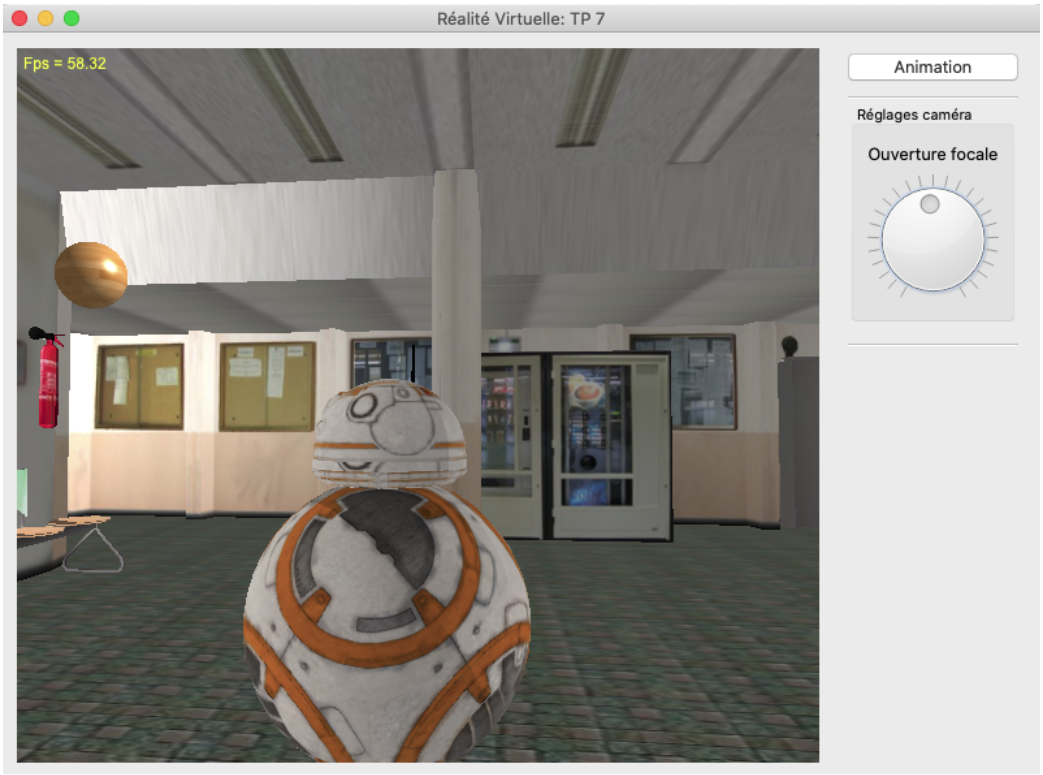
2. Déplacement

Déplacement dans le bâtiment

Pour améliorer le déplacement subjectif dans le bâtiment, on peut s'inspirer de ce qui se passe dans les jeux de type FPS (tir à la première personne) en utilisant la manette ou la combinaison clavier/souris :

- ajouter une méthode `lateral(distance d)` qui permet de se déplacer latéralement sans changer la direction de vue (comme un pas de coté) ;
- possibilité de s'accroupir ;
- possibilité de sauter.

3. BB8



BB8 est le robot qui apparaît dans l'épisode 7 de Star Wars. J'ai trouvé un modèle 3D assez bien fait sur le web (libre de droits) avec de belles textures.
Mais il a deux défauts qui font qu'on ne peut pas simplement utiliser la classe `RVModel` simplement comme le bâtiment (mais comme pour l'avion, nous allons créer une sous-classe spécifique) :

- les noms des textures ne sont pas renseignées dans le fichier `bb8.obj`
- l'origine du repère local utilisé pour modéliser les différents maillages n'est pas centré dans l'axe du robot (mais dans l'axe de l'antenne longue) et au niveau du sol (et pas du centre de la sphère). Cela va nous géner pour l'animer...

Textures

La classe `RVBB8` hérite donc de `RVModel` . Son constructeur apppelle le constructer de `RVModel` avec le chemin qui mène au fichier `bb8.obj`.

Ensuite pour chacune des 7 *meshes* de BB8 il faut appeler `setTexture` avec le bon chemin de la bonne texture pour chaque partie du maillage.

Pour bien comprendre quel *mesh* correspond à quelle partie du robot et donc à quelle texture, le mieux est de surcharger la méthode `draw()` pour pouvoir appeler un par un, les *draw* de chaque *mesh*.

Test

Pour l'ajouter à notre scène vous pouvez revenir en caméra sphérique et placer BB8 en (0, 0, -100) et le réduire de 40% pour qu'il ne soit pas disproportionné dans le hall du département Informatique.

```
m_bb8 = new RVBB8();
m_bb8->setCamera(m_camera);
m_bb8->setPosition(QVector3D(0, 0, -100));
m_bb8->setLight(m_light);
m_bb8->setScale(0.4f);
m_bb8->initialize();
```

Animation

Comme décrit plus haut l'origine du repère local est mal placée : pour s'en apercevoir, il suffit de surcharger la méthode `update(float t)` et appliquer un `rotate` sur l'un des *meshes*. Par exemple

```
void RVBB8::update(float t)
{
    m_meshes[6]->rotate(5, QVector3D(0,1,0));
}
```

Pour corriger cela, il faut donc replacer l'origine du repère (une chose qu'on aurait du faire pour que la rotation de l'hélice de l'avion soit plus jolie).

Pour cela il faut ajouter à `RVModel` une méthode virtuelle `setOrigin(QVector3D pos)` qui permet de modifier une variable membre `m_origin` (initialisée à (0, 0, 0) dans le constructeur. Cette variable `m_origin` est utilise dans la méthode `RVBody::modelMatrix()` pour appliquer une translation de **moins** `m_origin` comme première transformation (avant le scale) et donc placée en dernier dans la méthode (juste avant le `return`).
Cette méthode `setOrigin` doit être surchargée dans `RVModel` pour qu'elle s'applique à tous les *meshes*.
Enfin il faut juste trouver la bonne valeur à passer à BB8 pour que son origine soit dans l'axe et au centre de la sphère pour pouvoir l'animer proprement.

Déplacement de BB8

Si l'on veut déplacer BB8, comme l'avion, il faut qu'il ait une direction de déplacement et une vitesse. Mais il faut aussi de la sphère principale roule sans glisser sur le sol dans un axe perpendiculaire à la fois à la direction verticale et à la direction de déplacement (et donc on peut l'obtenir par un produit vectoriel). En revanche tous les autres *meshes* qui composent la tête doivent tourner tous ensemble selon des commandes clavier (ou manette).
Et évidemment la caméra sphérique est attachée à BB8.
Il faut donc prévoir une touche pour changer de caméra et passer de caméra subjective (visite libre dans le batiment) à caméra sphérique quand on pilote BB8.

4. Bonus

Idées

- Améliorer le HUD avec l'affichage du niveau de vie, projectiles, etc... Mais aussi ajouter un X de viseur au milieu de l'écran (avec la possibilité de zoomer comme avec un fusil de précision ?).
- Définir une trajectoire pour BB8 pour qu'il se déplace tout seul...
- Ajouter le petit avion du TP précédent (avec une troisième caméra).