TP8

Site: lms.univ-cotedazur.fr
Cours: Realite virtuelle - EIMAD919
Livre: TP8

Imprimé par: Theo bonnet
Date: vendredi 28 février 2020, 15:00

Table des matières

1. Masque

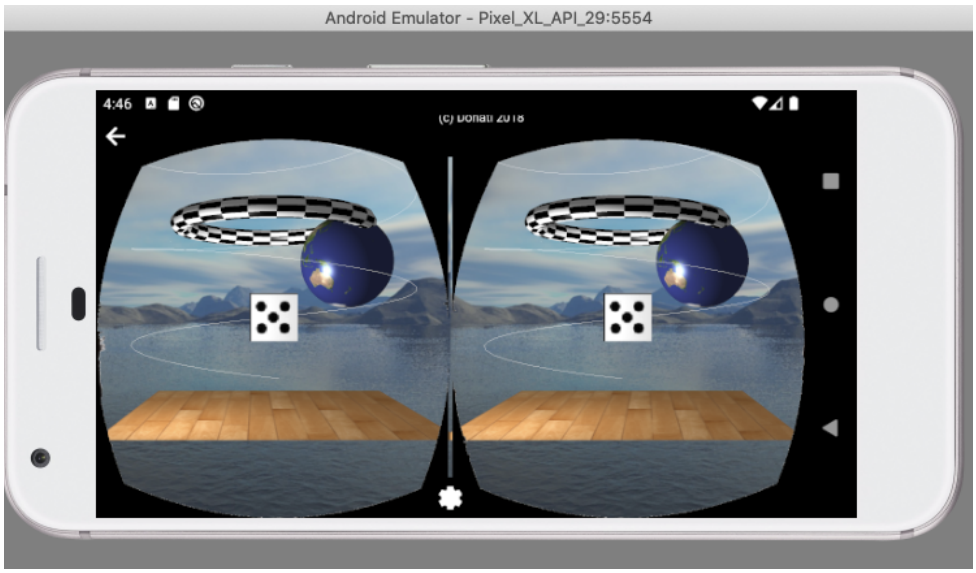
A partir du Tuto8, plus les différents objets que vous voulez ajouter à la scène, nous allons faire une application mobile destinée à être intégrée dans un casque de réalité virtuelle.

Mise en place du masque

Le principe d'un casque de réalité virtuelle est de donner des images différentes à chacun des deux yeux : au lieu de mélanger les deux images à l'aide de filtres rouge et cyan qui nécessitent des lunettes colorées, on va séparer l'écran en deux zones de même taille et on va rendre à gauche avec la caméra gauche et à droite avec la caméra droite (en manipulant le view port comme on a déjà fait dans le TP2).
Ensuite avec `QPainter` on va superposer aux deux images un bitmap qui représente un *cache* noir qui laisse seulement passer les images pour chaque oeil.

- Ajouter l'image `ecranRV.png` aux ressources du projet dans le groupe `textures` ;
- Ajouter à la classe `RVWidget` une variable membre de type `QImage` appelée `m_screen` et initialisez-la dans le constructeur à partir de la ressource ;
- Dans `mainwindow.ui` mettre à 0 les marges du *central widget* dans les propriétés *layout* ; de cette façon le widget occupe vraiment la totalité de l'écran ;
- Dans `RVWidget::paintGL()` :
  - J'ai besoin, comme pour afficher le fps à l'écran, d'une instance de QPainter et tout le code OpenGL sera à l'intérieur d'un bloc délimité par `beginNativePainting` et `endNativePainting` ;
  - Je vais diviser l'écran en deux parties, chacune d'une hauteur égale à `this->height()` mais d'une largeurégale à `this->with()/2`. Donc il faut en tenir compte pour donner la bonne valeur `aspect` à la caméra.
  - Pour l'oeil gauche, on met le viewport à la moitié gauche de l'écran, on active la caméra guache et on fait le rendu de la skybox et de la scène ;
  - la même chose pour l'oeil droit.
  - Après les commandes OpenGL, j'utilise la `QPainter` pour afficher le cache et écrire un petit texte.

```
void RVWidget::paintGL()
{
    QPainter p(this);
    p.beginNativePainting();

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    m_camera->setAspect(float(width())/(2*height()));
    m_skybox->setPosition(m_camera->position());

    //Rendu oeil gauche
    glViewport(0, 0, width()/2, height());
    m_camera->setCameraType(RV_CAMERA_LEFT);
    m_skybox->draw();
    m_scene.draw();

    //Rendu oeil droit
    glViewport(width()/2, 0, width()/2, height());
    m_camera->setCameraType(RV_CAMERA_RIGHT);
    m_skybox->draw();
    m_scene.draw();

    glDisable(GL_DEPTH_TEST);
    glDisable(GL_CULL_FACE);

    p.endNativePainting();
    p.setPen(Qt::white);
    p.setFont(QFont("Arial", 12));

    p.drawImage(QRect(0, 0, width(), height()), m_screen);
    p.drawText(width()/2-40, 25, "(c) Donati 2020" );
}
```



2. IHM

Pour implémenter des interactions avec l'application en plein écran, nous devons nous baser sur les actions liés à la souris (et donc au fait d'appuyer sur l'écran du smartphone).

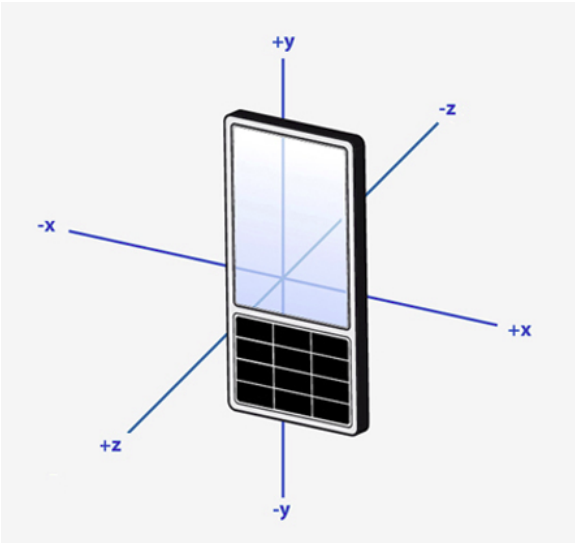
- Pour mettre en route l'animation on peut utiliser la méthode `mouseDoubleClickEvent`
- Pour afficher des messages à l'écran (comme le fps ou bien un message de *credits*) on doit dans `mousePressEvent` tester si l'endroit ou on a cliqué est proche du petit symbole de "roulette" au centre.

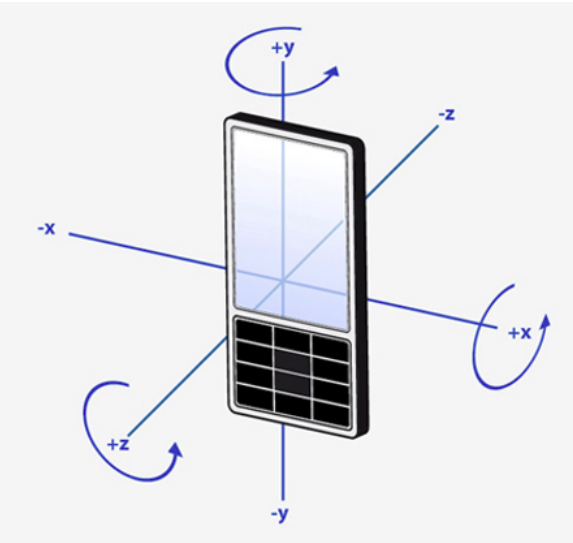
3. Senseurs

Senseurs :

Qt offre un plugin **Qt Sensors** pour gérer les différents senseurs présents sur le dispositif mobile ; la classe de base s'appelle **QSensor** qui possède des classes filles spécifiques à chaque type de senseur ; ces classes proposent une méthodes *reading*() qui renvoie la valeur du senseur sous la forme d'une instance de **QSensorReading** spécifique à chaque senseur :

- **QCompass** représente la boussole ; sa lecture est un **QCompassReading** qui a une méthode *azimuth*() qui donne l'angle en degré de l'orientation de l'axe du téléphone par rapport au Nord magnétique ;
- **QRotation** donne l'orientation du téléphone à partir des axes ; sa lecture est un **QRotationReading** qui a 3 méthodes *x*(), *y()* et *z()* qui donnent l'angle par rapport à chaque axe (angles d'Euler**)**
- **QAccelerometer** donne la mesure de l'accéléromètre ; sa lecture est un **QAccelerometerReading** qui a 3 méthodes *x(), y()* et *z()* qui donnent la valeurs de l'accélération le long de chaque axe en m/s^2. Cette accélération comporte la composante de la gravité.





4. Boussole

Ajouter à RVTP8.pro le plugin sensors ;

```
QT       += core gui sensors
```

- Ajouter à **RVWidget** une variable membre **m_compass** de type **QCompass***, initialisée dans le constructeur ;
- Dans *initializeGL* on lance la boussole avec sa méthode *start*()
- Dans *update*() on lit la boussole et on récupère l'azimuth :

```
    QCompassReading* reading = m_compass->reading();
```

```
    float azimuth = reading->azimuth();
```

- On convertit azimuth en radians et on le passe au lacet de la caméra.

Testez : ça devrait marcher.

5. Orientation

Faites la même chose que plus haut avec une variable **m_orientation** de type **QRotationSensor** et utilisez la variable y de la lecture du senseur pour modifier le tangage de la caméra.