TP6

Site: lms.univ-cotedazur.fr
Cours: Realite virtuelle - EIMAD919
Livre: TP6

Imprimé par: Theo bonnet
Date: vendredi 28 février 2020, 14:57

Table des matières

1. Mise ne place

2. Classe RVAirplane

3. Caméra qui suit l'avion

4. Sol en relief

5. Amélioration du pilotage

6. Utilisation d'une manette (optionnel)

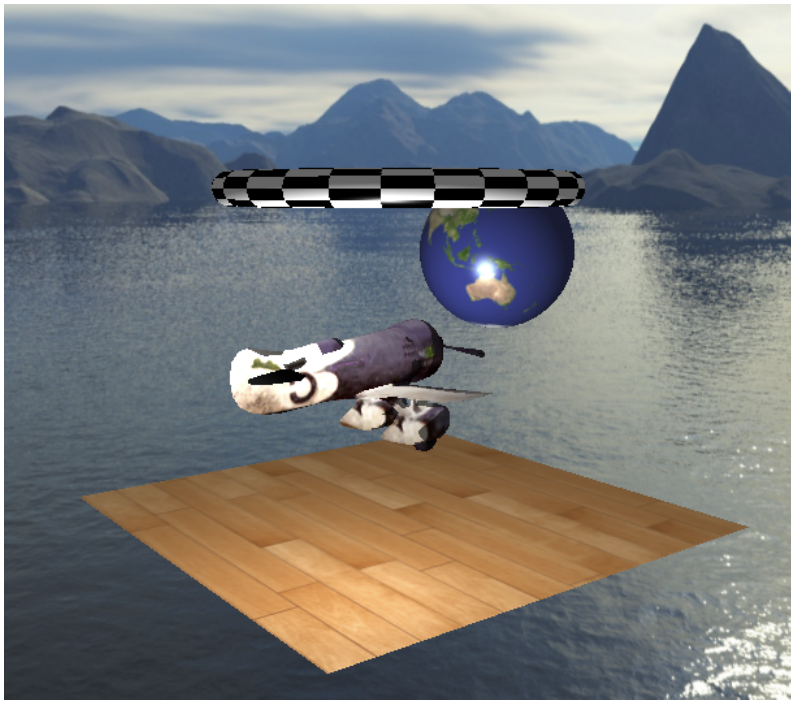
7. Idées de bonus

1. Mise ne place

Soit vous commencez là où s'est arrêté le tuto6, soit vous intégrez dans votre copie de TP5 ce qui a été ajouté dans le tuto6, c'est à dire :

- la classe `RVMesh`
- la classe `RVModel`
- quelques modifications à `RVBody` : certaines méthodes ont été déclarées virtuelles, et la méthode `setTexture` a été modifiée pour permettre le renversement vertical.
- la caméra a été remise en mode MONO

2. Classe RVAirplane



Pour piloter l'avion il faut créer une classe `RVAirplane` qui hérite de `RVModel` et qui est initialisé à partir de modèle 3D utilisé dans le tuto6.

1. le constructeur appelle celui de `RVModel` en lui passant le nom du fichier du modèle ;
2. variables membres (avec accesseurs et mutateurs)
   - `m_velocity` pour la vitesse (un float)
   - `m_acceleration`pour l'accélération (un float aussi)
3. la surcharge de la méthode `update(float time)` doit donc modifier la position de l'avion (`m_position`) en le faisant avancer (avec la bonne vitesse (qui est elle-même modifiée par l'accélération)) dans la direction du devant de l'avion. Et donc on doit tenir compte de son orientation pour modifier la position : si l'avion pointe vers le haut, l'avion doit monter, etc... Tout cela, evidement en tenant compte du temps qui s'est écoulé depuis le dernier appel de `update`
4. Des contrôles clavier doivent permettre d'accélérer ou freiner l'avion (sans toutefois jamais le faire reculer...) ainsi que de changer d'orientation (tangage = haut/bas et lacet = gauche/droite).
5. Faire tourner l'hélice : c'est le maillage n°2 de l'avion. Donc il suffit donc d'appeler sa méthode `rotate` avec le bon axe de rotation. Et ce serait bien que la vitesse de rotation de l'hélice dépende la vitesse de l'avion.

3. Caméra qui suit l'avion

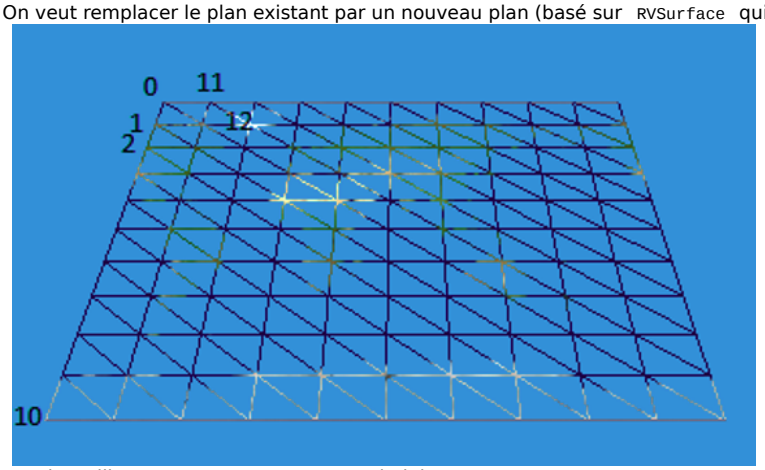
Pour empêcher que l'avion disparaisse de la vue de la caméra, on doit faire en sorte que la camera le suive.
Concrètement, cela signifie que la cible (`m_target`) de la caméra doit toujours être égale à la position de l'avion.
Pour faire ça, il faut que :

- dans la classe `RVCamera` la méthode `setTarget` soit marquée `virtual` ;
- dans la classe `RVSphericalCamera` on surcharge la méthode `setTarget`, pour qu'après la modification de `m_target` on appelle `updatePosition()`.

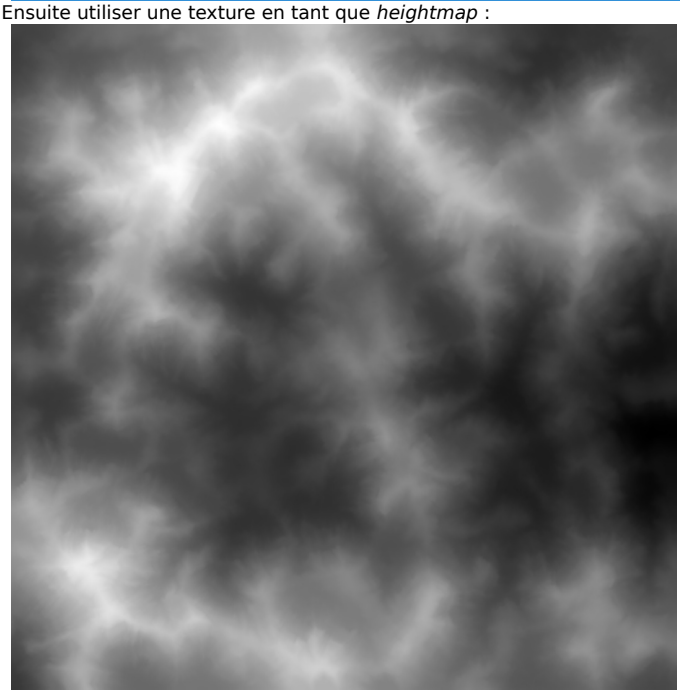
En fait, si l'on veut donner la même sensation que dans les jeux vidéo, il faut qua la caméra reste toujours derrière l'avion, et donc que lorsque l'avion tourne, la caméra (éventuellement avec un peu de retard) amorce le même mouvement pour se replacer à l'arrière.
Un effet intéressant qui peut constituer un bonus est l'effet **élastique** typique des jeux de course de voiture : lorsque la voiture accélère, la caméra démarre un peu plus tard et donc sa distance à la cible augmente, pour revenir normale un peu plus tard et lorsque l'on freine c'est l'effet inverse.
Remarquez que pour percevoir que l'avion avance, vous avez intérêt de rendre très très grand la parquet qui forme le sol.

4. Sol en relief

On veut remplacer le plan existant par un nouveau plan (basé sur `RVSurface` qui beaucoup plus de vertex.



Ensuite utiliser une texture en tant que *heightmap* :



On va utiliser le *vertex shader* pour lire la *heightmap* et interpréter l'information de couleur du pixel comme une information sur la hauteur du sommet (composante y). Donc là où la texture est claire, il y aura un fort dénivelé et là où la texture est sombre, au contraire, le dénivelé sera quasiment nul.

Classe RVTerrain

`RVTerrain` hérite de `RVSurface` et a deux variables membres :

- `m_heightmap` qui est un pointeur sur un `QOpenGLTexture` qui sera la texture utilisée pour lire l'information sur la hauteur; son mutateur est basé sur `RVBody::setTexture` :

```
void RVTerrain::setHeightmap(QString textureFilename)
{
    m_heightmap = new QOpenGLTexture(QImage(textureFilename).mirrored());
}
```

- un float `m_heightFactor` qui est la facteur multiplicatif à utiliser (avec son accesseur et son mutateur).

Le constructeur de `RVTerrain` prend un argument de type *double* qui est la taille du plan. Cette taille est utilisée pour renseigner les intervalles utilisés pour les paramètres `s et t` de la paramétrisation.

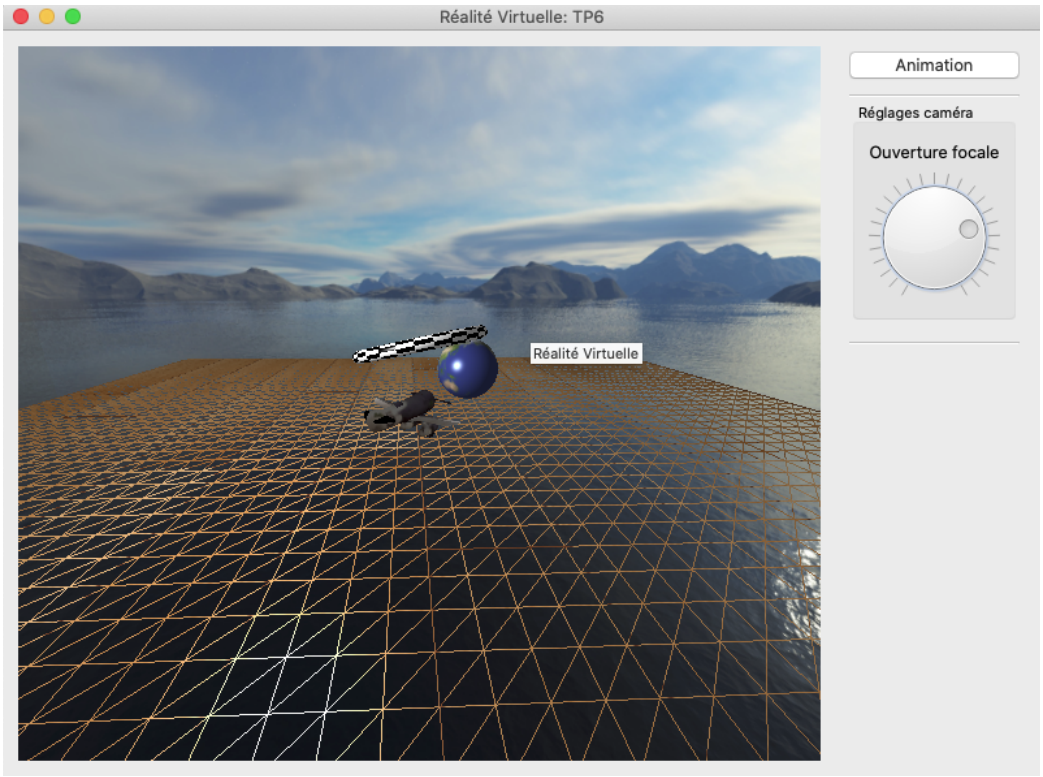
```
    RVTerrain::RVTerrain(double width)
    :RVSurface()
{
    m_minS = -width/2;
    m_maxS = +width/2;
    m_minT = -width/2;
    m_maxT = +width/2;
    m_numSegS = 50;
    m_numSegT = 50;

    m_VSFileName = ":/shaders/VS_heightmap.vsh";
}
```

Dans le code ci-dessus, on a fixé le nombre de segments à 50 sur chacune des deux dimensions. PLus ce nombre est grand plus le maillage est fin. Pour le vertex shader a utiliser on verra ci-dessous.
Les fonctions qui donnent l'équation paramètrée du plan horizontal sont très simple, puisque

$$x(s,t) = s,\ \ y(s,t) = 0\ \ z(s,t) = t$$

Ce qui donne un plan avec beaucoup de triangles.

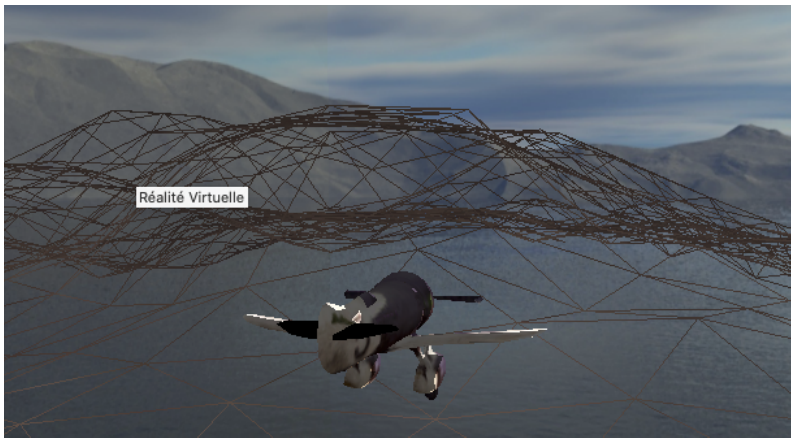


Modification de shader

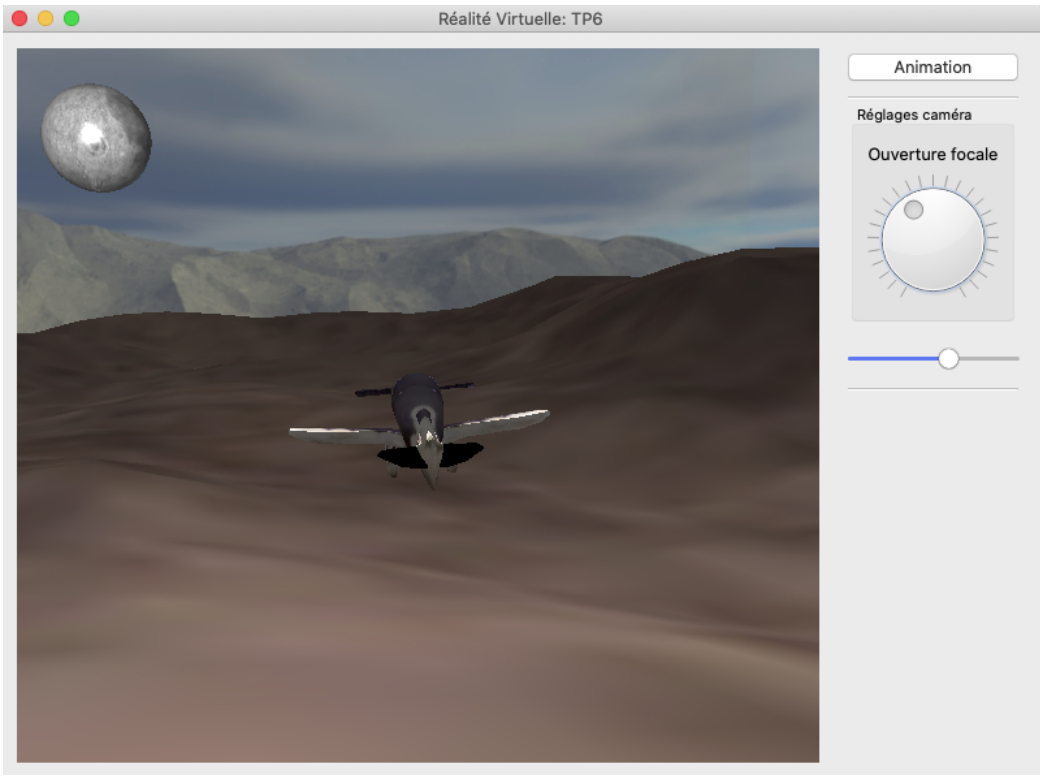
On doit seulement modifier le vertex shader. A partir de `VS_lit_texture.vsh` on crée `VS_heightmap.vsh` :

- deux nouvelles variables uniformes : la texture du heightmap et le facteur multiplicatif à appliquer ;
- on utilise les coordonnées textures pour lire dans la texture du heightmap.
- Et on utilise une composante de la couleur obtenue (par exemple le rouge) pour incrémenter (multipliée par le heightFactor) la composante y de `outPos`.

On surcharge la méthode `draw` de `RVSurface` en reprenant le code original mais on intégrant la nouvelle texture et les nouvelles variables uniformes.
Pour tester le résultat, dans `RVWidget` , on définit cette fois `m_plane` comme une instance de RVTerrain et on lui passe une heightmap (que vous trouvez sur internet) et un facteur multiplicatif.



On peut alouter un slider à l'IHM pour modifier `m_heightFactor` et donc augmenter le relief des montagnes.



5. Amélioration du pilotage

Dans un avion, pour changer de direction (de lacet), il faut que l'avion s'incline du coté où il veut tourner (changement de roulis).

Pour cela, il suffit que dans *keyPressEvent* on appelle rotate en modifiant à la fois le roulis et le tangage. Normalement ça marche sauf qu'après avoir tourné l'avion reste incliné sur le côté ; il faut donc lui redonner progressivement un roulis nul : par exemple en multipliant l'ancienne valeur **m_roulis** par 0.98f dans *avance,* pour l'avion et tous ses **Mesh** sauf l'hélice.

6. Utilisation d'une manette (optionnel)

Pour utiliser une manette de jeu au lieu du couple souris/clavier, on va utiliser le module **QtGamepad** de **Qt** qui fonctionne bien sous Windows et Linux mais pas encore sur Mac.

Une fois la manette branchée (par exemple la manette XBox360) vous pouvez dans l'accueil de **QtCreator** rechercher les projets gamepad et ouvrir et compiler le projet appelé « Qt GamepadQt Quick Example » qui montre à l'écran l'action des différents contrôles de la manette.

Sous Linux si la manette n'est pas reconnue, il faut suivre les instructions sur http://doc.qt.io/qt-5/qtgamepad-index.html

- Ajouter au projet le module gamepad à QT

```
QT       += core gui gamepad
```

- Dans rvwidget.h ajouter *l'include* de QtGamepad

```
#include <QtGamepad>
```

- Dans la classe **RVWidget** ajouter une variable membre **m_gamepad** de type **QGamepad***
- Initialisation dans *initializeGL* :
  - Sous Windows c'est simplement :

```
    m_gamepad = new QGamepad();
```

  - Sous Linux il faut en revanche :

```
    auto gamepads = QGamepadManager::instance()->connectedGamepads();
    if (gamepads.isEmpty()) {
        qDebug() << "No Gamepad found";
        return;
    }
    m_gamepad = new QGamepad(*gamepads.begin(), this);
```

  - Sous Mac je n'ai pas réussi avec la manette de Xbox360.

- Utilisation de la manette : c'est très simple, dans la méthode `update` on a accès via les méthodes de `m_gamepad` à la valeur de tous les boutons et *thumbsticks* de la manette ; les contrôles analogiques (comme **axisRightX** ou **axisRightY** qui représentent le déplacement en X et en Y du *thumbstick* droit) donnent un réel entre -1 et 1, les boutons en revanche (comme **buttonA**, **buttonB**) renvoient un booléen. A vous de configurer votre manette pour contrôler à la fois le déplacement de l'avion (y compris l'accélération) et la position de la caméra. On peut aussi utiliser les boutons pour sortir de l'application (ou autre) et les gâchettes pour changer le `m_fov` ou `m_rho.`

7. Idées de bonus

Quelques idées

- Enrichir l'IHM pour connaître l'altitude de l'avion, sa vitesse, son inclinaison... Mais aussi pour passer de caméra mono à caméra stéréo...
- Donner la possibilité de changer de vue en changeant de caméra : au sol, sur le cockpit, devant l'avion en vue subjective...
- Ajouter plusieurs avions avec un vol en escadrille...
- Ajouter d'autres modèles 3D trouvés sur le web...
- Disséminer des objets (des tores ou des cubes) dans la carte, et compter le nombre de fois où l'avion passe dessus...