Tuto 5

Site: lms.univ-cotedazur.fr
Cours: Realite virtuelle - EIMAD919
Livre: Tuto 5

Imprimé par: Theo bonnet
Date: vendredi 28 février 2020, 14:55

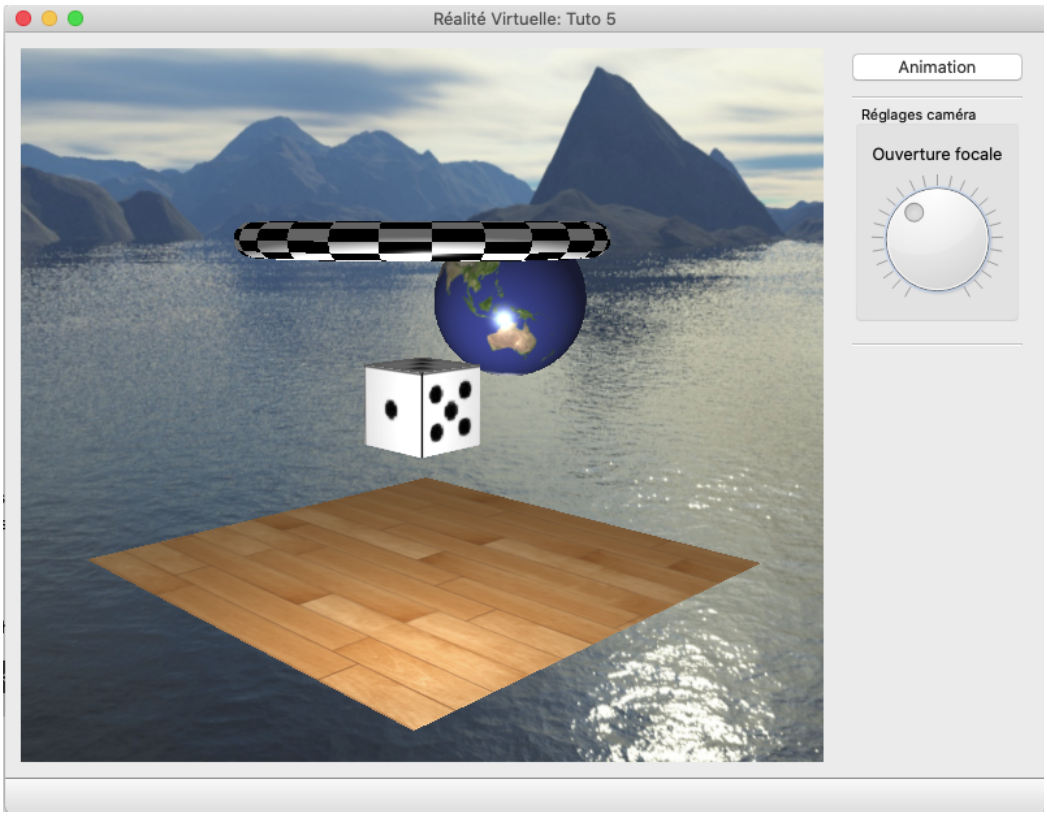
Table des matières

1. Animation

Animation

Le point de départ du Tuto5 est un projet qui intègre

- plan
- sphère avec texture de la terre
- tore avec texture procédurale à damier
- dé à jouer
- skybox
- éclairage
- caméra sphérique



2. Animation

Actuellement, il y a dans l'interafce utilisateur de notre programme un bouton `Animation` qui fait tourner tous les objets dela scène. Tous les objets ont la même vitesse de rotation et le même axe de rotation. Si l'on veut que chaque objet puisse avoir une animation et un mouvement différents, il faut faire évolaure notre modèle.

Méthode update(float time)

La façon la plus logique consiste à ajouter à la classe `RVBody` une méthode publique `void update(float time)` qui est destinée à recevoir le code nécessaire à faire évoluer l'objet au fil du temps ; soit en modifient sa position, soit en modifiant son *attitude*, soit en modifiant n'importe quel autre de ses paramètres (taille, opacité, couleur, etc...). L'argument de `update` est le temps (en millisecondes) soit qu'il s'est écoulé depuis le dernier appel de update (temps relatif), soit depuis le lancement de l'application (temps absolu). C'est à vous de choisir comment vous vouler utiliser cette méthode.
Dans la classe abstraite `RVBody` nous pourrions faire de `update` une méthode abstraite, mais cela oblige toutes les classes filles à l'implémenter (même les objets qui ne bougent pas come la skybosx ou le plan). Donc nous allons plutôt définir une méthode qui ne fait rien.

```
    virtual void update(float time){}
```

Dans la classe `RVScene` il y aura aussi une méthode `update` qui appelle la méthode `update` sur tous les objets qu'elle contient.

```
void RVScene::update(float t)
{
    foreach (RVBody* body, *this) {
       body->update(t);
    }
}
```

Gestion du temps

C'est dans `RVWidget` que l'on va gérer le passage du temps.
Il y aura une variable `m_time` de type `QTime` (ne pas confondre avec `QTimer`). Cette variable sera initialisée et démarrée dans le constructeur.

```
    m_time = QTime();
    m_time.start();
```

Ensuite dans la méthode `RVWidget::update()`

- Soit on calcule le temps qui est passé depuis le dernier appel (temps rélatif) avec la méthode `restart()` et on le passe à la méthode `update` de la scène

```
    void RVWidget::update()
    {
        int t = m_time.restart();

        if (m_animation) {
            m_scene.update(float(t));
        }

        QOpenGLWidget::update();
    }
```

- Soit on calcule le temps qui est passé depuis l'appel de `initializeGL()` (temsp absolu) avec la méthode `elapsed()`

```
void RVWidget::update()
{
    int t = m_time.elapsed();

    if (m_animation) {
        m_scene.update(float(t));
    }

    QOpenGLWidget::update();
}
```

Utilisation pour le cube

Pour animer le dé, il suffit de lui donner une méthode `update(float t)` et lui dire d'appliquer une rotation autour d'un axe quelconque.
Si par exemple on a choisi de passer un temps relatif (en millisecondes), et que l'on cible une vitesse de rotation fixe de 30 degrés par seconde alors le code est :

```
void RVDice::update(float t)
{
    int vitAngulaire = 30; //en degré par seconde
    this->rotate(t*vitAngulaire*0.001, QVector3D(1, 1, 1));
}
```

Ainsi la vitesse va être parfaitement régulière, même si l'intervalle de tmeps entre deux appels consécutifs de `update` n'est pas régulier à cause d'autres process qui tournent sur votre machine.
Remarquez que je divise t par 1000 pour convertir le temps en secondes.
Si vous utilisez le même code mais avec un temps absolu vous allez voir le cube tourner de plus en plus vite, ce qui est normal puisqu'on ajoute un angle de plus en plus grand.
Vous pouvez faire la même chose avec la terre et le tore avec des axes de rotation diférents.

Translation

Vous pouvez aussi utiliser update pour changer la position du cube soit de façon absolue (avec `setposition`)soit relative avec `traslate()`. Une façon de procéder consiste à ajouter une variable vecteru vitesse `m_velocity` (de type `QVector3D`) à `RVBody` et multiplier la vitesse par le temps écoulé pour connaître le vecteur de translation.

```
this->translate(m_velocity*t*0.001);
```

Evidemment l'objet va avoir un mouvement rectiligne et très rapidement sortir de l'écran.
Ou encore on peut intégrer une accélération de gravité (0, -10, 0) qui va courber la trajectoire rectiligne vers le bas en forme de parabole en une espèce de simulation physique.

3. Trajectoires

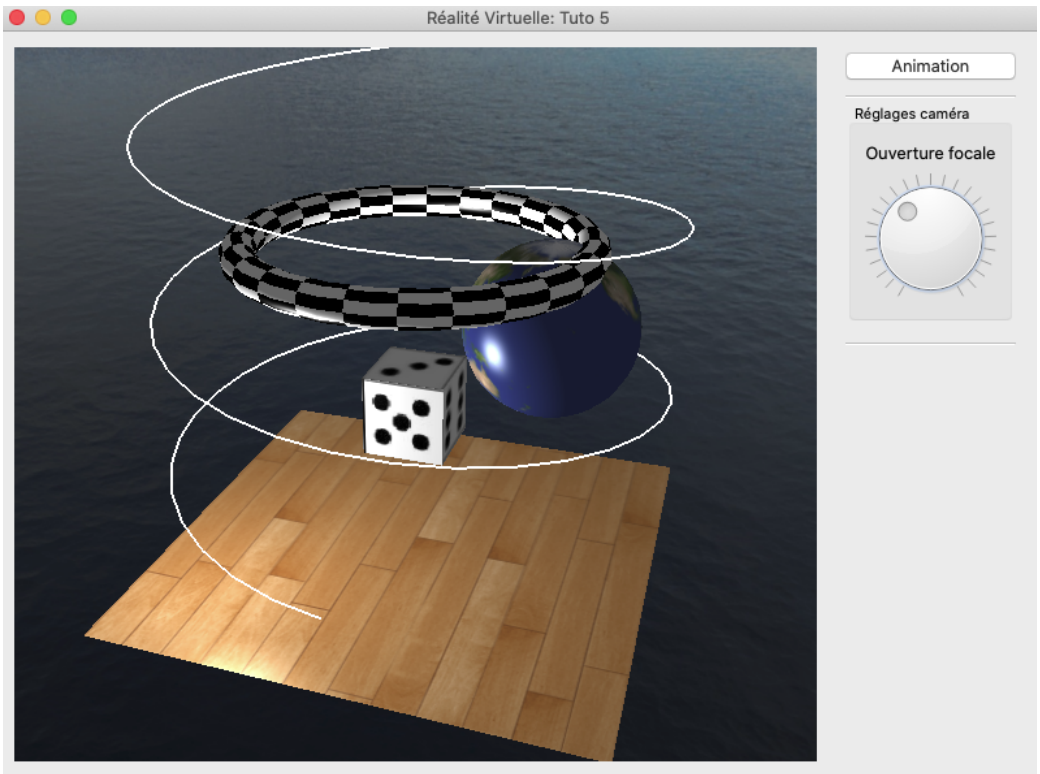
Une autre façon d'animer les objets consiste à leur associer à priori une trajectoire qu'ils vont devoir suivre : une trajectoire est une courbe de l'espace paramétrée par le temps et nous pouvons utiliser cette courbe pour fixer le centre de l'objet.
Nous avons déjà vu dans la session 2, un objet de type courbe appelé `RVCurve` et une classe-exemple qui en hérite qui est `RVHelix` .

- intégrer ces deux classes au projet
- ajouter à `RVWidget` une variable `m_trajectory` de type `RVCurve *`
- dans `RVWidget::initializeGL()` on initialise `m_trajaectory` comme les autres objets 3D et on l'ajoute à `m_scene`.

```
    m_trajectory = new RVHelix();
    m_trajectory->setCamera(m_camera);
    m_trajectory->setPosition(QVector3D(0, 0, 0));
    m_trajectory->setScale(1);
    m_trajectory->initialize();

    m_scene.append(m_trajectory);
```

Ainsi on voit clairement la courbe :

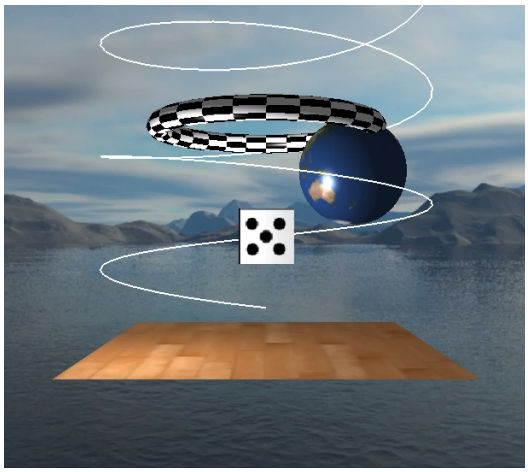


Si l'on veut que la sphère suive cette trajectoire (tout en tournant sur elle même), on pourrait dasn `RVWidget::update()` modifier la position de la sphère `m_world` pur qu'elle soit égale à la valeur donnée par la propriété `pos` de la trajectoire (ne fonctionne que avec un temps absolu) :

```
void RVWidget::update()
{
    int t = m_time.elapsed();

    if (m_animation) {
        m_scene.update(float(t));
        m_world->setPosition(m_trajectory->pos(t*0.001));
    }

    QOpenGLWidget::update();
}
```



Mais'est beaucoup plus propre d'associer à la classe `RVSphere` (ou directement `RVBody`) une variable membre `m_trajectory` de type `RVCurve*` (avec accesseurs) et que ce soit donc la méthode `update()` de la sphère qui utilise sa propre trajectoire. Ainsi on peut associer des trajectoires différentes à chaque objet dela scène.C'est ce qu eje vous conseille de faire dans le TP5 à suivre.