TP4

Site: lms.univ-cotedazur.fr
Cours: Realite virtuelle - EIMAD919
Livre: TP4

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

Table des matières

1. Mise en place du TP4

2. Classe RVLight

3. Eclairage des cubes

4. Lumière dans IHM

5. Environment Mapping

6. Bonus

Mise en place

Copier le TP3 dans un nouveau dossier TP4 et changer le nom de TP3.pro en TP4.pro.
La première étape consiste à intégrer dans le TP4, ce qui a été développé dans le tuto4 :

- La version 4 de `RVBody` qui intègre le nouveau `RVVertex` avec vecteur normal.
- Les nouveaux shaders par défaut de `RVBody` qui intègrent l'éclairage et qui sont `VS_lit_texture.vsh` et `FS_lit_texture.fsh`, qu'il faut ajouter aux ressources du projet.
- La version 4 de `RVPlane` qui utilise la tetxure "wood.png" et les nouveaux shaders avec éclairage.
- La version 4 de `RVSurface` avec calcul du vectuer normal sur chaque sommet.
- La version 4 de `RVSphere` avec la texture de la terre qui en hérite.
- La version 4 de `RVTorus` qui en hérite et qui utilise un fragent shader spécifique `FS_lit_damier.fsh`.

Malgré la nouvelle version de `RVBody` on peut, pour l'instant, continuer à utiliser les différents types du cube texturés et la skybox pourvu de bien spécifier les anciens sahders sans éclairage. On peut aussi continuer à utiliser la caméra sphérique et donc placer dans le `RVScene` un certains nombre d'objets tout autour de l'observateur.

Classe RVLight

Ajouter au projet une nouvelle classe RVLight (sans classe parent pour l'instant) qui va encapsuler les données relatives à la source luineuse ; ses variables membres sont donc :

- une position `m_position` de type `RVVector3D` ;
- trois `QColor` pour les 3 composantes de la lumière, `m_ambient, m_diffuse et m_specular`.

Donner des accesseurs et des mutateurs à ces variables ainsi que des valeurs par défaut (tirées du Tuto4) dans son constructeur.

Ajouter alors à la classe `RVBody` une variable `m_light` de type pointeur sur `RVLight` , ainsi qu'un float `m_specStrength` qui va spécifier le coefficient de réflexion spéculaire à utiliser dans le shader. Ces deux variables doivent aussi avoir mutateurs et accesseurs.
Vous pouvez donc modifier les méthodes `draw()` de `RVPlane` et `RVSurface` pour qu'elles utilisent directement ces nouvelles variables pour donner une valeur aux variables uniformes du shader.
Il faut bien sûr que `RVWidget` ait aussi une instance de `RVLight` qu'elle la passe au plan, à la sphère et à tore.

Eclairage des cubes

Pour tous les 3 types de cube, `RVCuve`, `RVTexCube`, `RVDice` , il faut mettre à jour le vertex buffer pour ajouter les vecteurs perpendicualires aux faces. Ainsi, on pourra les éclairer avec les nouveaux shaders.
Ceci dit les shaders d'éclairage que l'on a utilisés dans le tuto4 ne peuvent pas être utilisés tels quels pour les cubes pour deux raisons :

- dans les cubes l'attribut `rv_TexCoord` est un `vec3` et pas un `vec2`
- les cubes utilisent aussi un attribut `rv_Color` qui n'existe pas dans les autres classes qui héritent de `RVBody` et qui structurent leur VBO en utilsant la structure `RVVertex` .

Pour cela il fait modifier le vertex shader "VS_lit_texture.vsh" en "VS_lit_texture_cube.vsh" pour qu'il s'adapte aux spécificités du vertex buffer des cubes.

Puisque le coefficient de réflection spéculaire est un attribut de chaque instance de `RVBody` on peut modifier sa valeur d'un objet à l'autre...

La lumière dans IHM

Modifier l'IHM de votre application afin que l'on puisse modifier la position de la source lumineuse ainsi que la teinte de ses trois composantes.
Pour modifier la couleur de fond et la forme des PushButton il faut utiliser la variable styleSheet de QWidget en lui passant une css comme en html. Par exemple

```
background-color: rgb(128, 128, 128);
border: 1px solid black;
border-radius: 5px;
```

Pour modifier une composante de la couleur de la lumière, le slot associé à l'action d'appuyer sur le bouton s'écrit de la façon suivante :

```
void MainWindow::changeAmbientLight()
{
    RVLight* lumiere = ui->widgetRV->light();
    QColor col = QColorDialog::getColor(lumiere->ambient(), this);
    if (col.isValid()) {
        QString qss = QString("background-color: %1;\n border: 1px solid black;\n border-radius: 5px;").arg(col.name());
        ui->ambientButton->setStyleSheet(qss);
        lumiere->setAmbient(col);
    }
}
```

Rendre la source lumineuse visible

Si l'on veut que la source lumineuse apparaisse dans la scène, par exemple comme une petite sphère blanche ou avec une texture de soleil, on peut ajouter à la scène une petite sphère qui a la même couleur et la même position de la `RVLight`.
On peut imaginer aussi de lui ajouter un booléen qui indique si l'on veut ou pas qu'il soit rendu dans la scène (et du coup une case à cocher dans l'IHM).

h3> Réflection spéculaire de la skybox
Maintenant que chaque objet dans la scène possède un vecteur normal, on peut l'utiliser pour faire rebondir le rayon de vue sur la surface et voir où il va finir dans la skybox : ou plutôt, inversement, la skybox va se réfléchir à la surface d'un objet.
On appelle cette technique **environment mapping**
Dans notre fragment shader qui produit l'éclairage on a déjà :

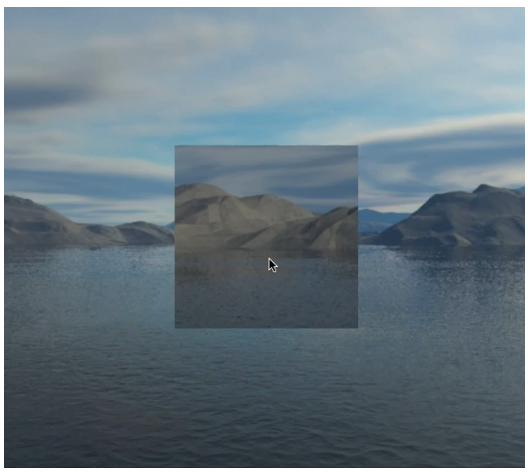
- `viewdir` qui est un vecteur unitaire qui pointe du fragment vers la caméra ;
- `reflectdir` qui est un vecteur unitaire qui donne la réflexion du premier vecteur par rapport à la normale à la surface au point où l'on est.

Donc, on n'a plus qu'à utiliser le vecteur `reflectdir` pour lire dans une texture cubique avec `textureCube` (comme pour la skybox).
Par exemple pour un environement mapping sur un cube : on crée une nouvelle classe RVSpecularCube qui hérite de RVTexCube :

- il aura une texture cubique comme `RVSkybox` et la même méthode `setCubeTexture`
- pas de redéfinition de `initializeBuffer` ni `initializeVAO`
- le vertex shader est toujours `VS_lit_texture_cube`
- c'est le fragment shader qui change : `FS_lit_specular_cube` :
  - il y a une nouvelle variable uniforme `uniform samplerCube skybox;`
  - si, à la fin, la couleur du fragment est

```
gl_FragColor = textureCube(skybox, reflectdir);
```

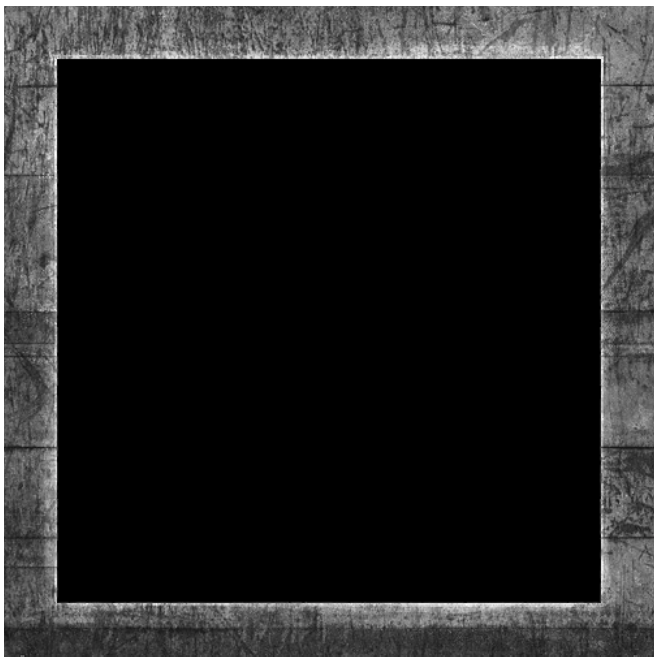
    alors on aura seulement le reflet spéculaire de la skybox.
    Si l'on veut quelque chose de plus sphistiqué il faut utiliser la couleur issue du cube comme couleur spéculaire de l'objet (à la place du blanc (1, 1, 1, 1) dans le calcul de `specular`).
- dans `draw` on reprend le code de `RVTexCuve` sauf qu'on indique que `m_texture` est une texture cubique et qu'on l'associe à la variable uniforme `skybox` du fragment shader.



Bonus

Quelques idées pour ceux qui veulent aller plus loin :

- l'*environment mapping* est crédible quand le cube réfléchissant est le seul objet dans la scène : dès qu'il est entouré d'autres objets on ne comprend pas pourquoi on ne voit pas le reflet des autres objets ! Quelle solution ? On calcule dynamiquement, via du rendu *offscreen*, les 6 images produites par 6 caméra orthogonale le long des 6 directions de tous les objets plus la *skybox* moins l'objet réfléchissant) et on utilise ces 6 images comme textures cubiques à passer au cubé réfléchissant. Ainsi on verra apparaitre dans le reflet des faces les objets autour...
- Texture de couleur spéculaire : certains objets peuvent avoir plusieurs textures en liaison avec l'éclairage : une texture pour la couleur ambiante et diffuse et une autre couleur pour la couleur spéculaire. Par exemple pour la caisse en bois vous pouvez utiliser la texture suivante pour la couleur spéculaire. Ça permet de prendre en compte que différentes parties d'un objet peuvent avoir des capacités réfléchissantes différentes.



- Plus faciles :
  - animer la source lumineuse pour qu'elle tourne autour de la scène ;
  - mettre l'effet miroir sur d'autres objets (par exemple le tore) ;
  - utiliser le rendu offscreen du tuto4 pour proposer une sauvegarde dans un fichier de l'image à l'écran.