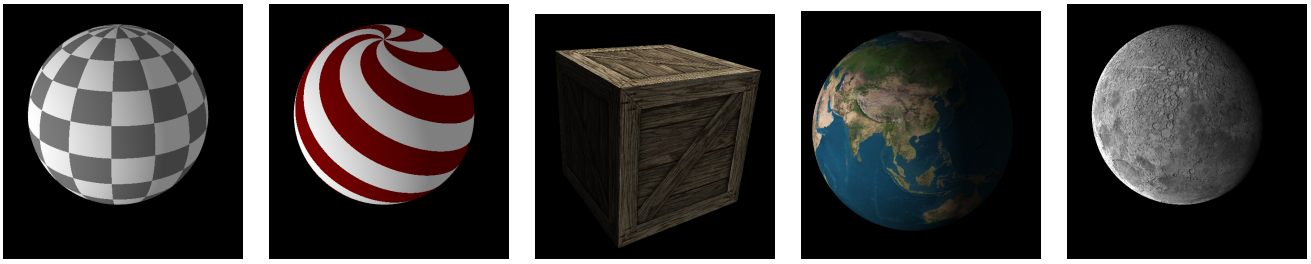
Modélisation et Synthèse d'Image
TP6 : Textures



Introduction

L'objectif du TP est d'apprendre à utiliser le fragment shader pour calculer des images complexes. L'objectif du TP est d'apprendre à utiliser les textures en OpenGL.

La base de code fournie gère l'affichage d'un objet 3D non texturé. Pour la récupérer, allez sur la page du cours : http://romain.vergne.free.fr/blog/?page_id=228.

Pour l'utiliser, suivez la démarche habituelle :

- extrayez l'archive du TP dans le dossier de votre répertoire personnel dédié aux TP de 3D (`unzip TP6.zip -d ~/TP3D/`)
- de même pour le dossier des textures (`unzip textures.zip -d ~/TP3D/`) après l'avoir téléchargé
- accédez au dossier du TP (`cd ~/TP3D/TP6/`)
- créez un lien symbolique vers le dossier `external` (`ln -s ../external/`)
- créez un lien symbolique vers le dossier `models` (`ln -s ../models/`)
- créez un lien symbolique vers le dossier `textures` (`ln -s ../textures/`)
- créez un dossier pour la compilation (`mkdir build`)
- accédez à ce dossier (`cd build`)
- lancez `cmake` (`cmake ..`)
- lancez la compilation (`make`)
- exécutez (`./polytech_ricm4_tp6`)

Vous devriez avoir la hiérarchie de fichiers ci-contre.

Note : ce TP utilise `QGLWidget` pour gérer le chargement des images et leur conversion en texture. La librairie `Qt` est déjà présente sur les machines de l'Imag, et n'est donc pas rajoutée au dossier `external`.

```
TP3D
├── external
│   └── ...
├── models
│   └── ...
├── textures
│   └── ...
├── TP1
│   └── ...
├── TP2
│   └── ...
├── TP3
│   └── ...
├── TP3 bis
│   └── ...
├── TP4
│   └── ...
├── TP5
│   └── ...
└── TP6
    ├── external [-> ../external/]
    ├── models [-> ../models/]
    ├── textures [-> ../textures/]
    ├── build
    │   └── ...
    ├── shader
    │   └── ...
    ├── src
    │   └── ...
    └── CMakeLists.txt
```

Introduction

Dans ce TP, nous allons utiliser des images pour texturer nos objets. L'avantage de cette technique est de permettre d'avoir des résultats beaucoup plus détaillés sans les sur-coûts dus à la création et à l'affichage d'une géométrie plus complexe.

1 Affichage de texture

Dans le programme initial, vous observez une sphère colorée. Les couleurs des sommets sont calculées à partir des coordonnées de textures : vous pouvez observer que celle-ci varient continûment selon des coordonnées sphériques de la position du sommet. Ces coordonnées sont stockées dans un nouveau VBO, qui est déjà correctement créé et bindé.

Afin d'utiliser ces coordonnées pour afficher une texture, il vous faut modifier l'initialisation du programme C++ :

- Charger l'image : `QImage img("../textures/super_texture.jpg");`
- La convertir à un format propre à OpenGL : `img = QGLWidget::convertToGLFormat(img);`
- Vérifier que l'image est bien chargée :

```
if(img.isNull()) {
    std::cerr << "Error Loading Texture !" << std::endl;
    exit(EXIT_FAILURE);
}
```

- Déclarer un identifiant : `GLuint textureID;`
- Allouer la texture sur le GPU : `glGenTextures(1, &textureID);`
- La définir comme texture courante : `glBindTexture(GL_TEXTURE_2D, textureID);`
- Définir des paramètres de filtre (nous reviendrons sur ces paramètres et leur signification au TP suivant) :

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

- Transmettre l'image au GPU :

```
glTexImage2D(GL_TEXTURE_2D,
     0,
     GL_RGBA32F,
     img.width(),
     img.height(),
     0,
     GL_RGBA,
     GL_UNSIGNED_BYTE,
     (const GLvoid*)img.bits());
```

- Récupérer l'identifiant de la texture dans le shader :
  `GLuint texSamplerID = glGetUniformLocation( programID, "texSampler" );`
- Dans la boucle de rendu, lier l'unité de texture 0 avec la texture :

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureID);
glUniform1i(texSamplerID, 0);
```

Puis à la fin du programme, il faut détruire la texture : `glDeleteTextures(1, &textureID);`
Dans le GLSL, il suffit maintenant de récupérer la couleur : `frag_color = texture( texSampler, vert_texCoord);`
Réalisez la procédure ci dessus pour texturer votre sphère.

2 Coordonnées de texture

Avant d'utiliser une texture, il faut créer des coordonnées de texture pour les sommets du maillage qu'on visualise. Ces coordonnées sont des `vec2` qui vont nous permettre de savoir où aller lire dans la texture pour récupérer la couleur du fragment à afficher.

Dans le cas d'un sphère, on peut comme précédemment utiliser les composantes angulaires des coordonnées sphériques pour définir les coordonnées de textures. Trouvez une ou plusieurs manières de définir les coordonnées de textures d'un cube. Pour ceci, ajoutez les dans la fonction

3 Bonus

Les textures ne servent pas uniquement à définir la couleur des fragment : elles peuvent représenter n'importe quel facteur utilisé dans l'équation de l'éclairage (cf. TP 4). Utilisez une texture pour représenter une autre caractéristique de la surface.