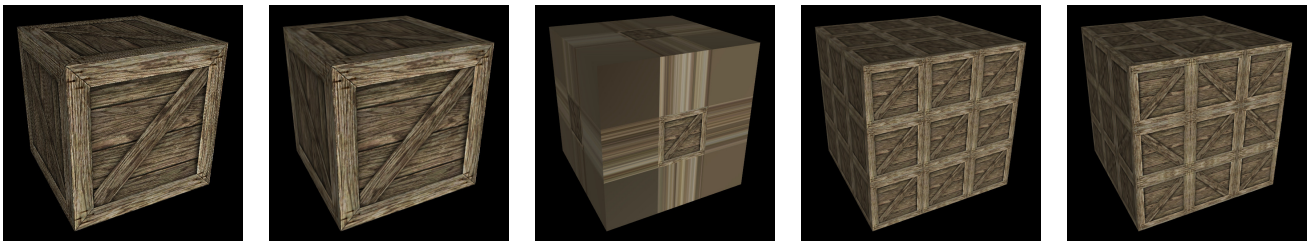
Modélisation et Synthèse d'Image
TP7 : Echantillonnage de texture



Introduction

Dans ce TP, nous allons voir quelles sont les limites associées à l'utilisation des textures. Plus particulièrement les problèmes d'échantillonnage et de coordonnées de textures.

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 TP7.zip -d ~/TP3D/`)
- accédez au dossier du TP (`cd ~/TP3D/TP7/`)
- 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_tp7`)

Vous devriez avoir la hiérarchie de fichiers ci-contre.

```
TP3D
├── external
│   └── ...
├── models
│   └── ...
├── textures
│   └── ...
├── TP1
│   └── ...
├── TP2
│   └── ...
├── TP3
│   └── ...
├── TP3 bis
│   └── ...
├── TP4
│   └── ...
├── TP5
│   └── ...
├── TP6
│   └── ...
└── TP7
    ├── external [-> ../external/]
    ├── models [-> ../models/]
    ├── textures [-> ../textures/]
    ├── build
    │   └── ...
    ├── shader
    │   └── ...
    ├── src
    │   └── ...
    └── CMakeLists.txt
```

Traitement des coordonnées de texture

Pour rappel, les coordonnées de textures sont des `vec2` associé à chaque sommet indiquant où aller lire dans une texture.

Les coordonnées peuvent comprendre n'importe quelle valeur, couvrant ainsi un domaine 2D infini. En revanche la texture est définie dans un domaine bien particulier : $C = [0, 1] \times [0, 1]$.

Pour commencer, modifiez les coordonnées de textures associées aux sommets du cube dans la fonction `create_cube` pour les faire sortir de l'intervalle C. Par exemple, vous pouvez leur faire parcourir $C' = [-1, 2] \times [-1, 3]$. Observez le résultat, et déduisez en le traitement par défaut des coordonnées sortant de C.

OpenGL propose différente manière d'interpréter les coordonnées de textures sortant de C. La fonction permettant de manipuler les paramètres associés aux textures s'appelle `glTexParameteri`. Elle prend toujours trois argument, dont le premier sera systématiquement `GL_TEXTURE_2D` au cours de ce TP. Le second argument désigne le paramètre à manipuler, et le troisième la valeur à lui associer.

Par ailleurs, pour spécifier la texture à laquelle la modification de paramètre fait appel, il faut appeler `glTexParameteri` après `glBindTexture`.

L'interprétation des coordonnées de textures est dictée par deux paramètres :

- `GL_TEXTURE_WRAP_S`, qui contrôle le comportement selon l'axe X
- `GL_TEXTURE_WRAP_T`, qui contrôle le comportement selon l'axe Y

Les valeurs que l'on peut leur associer sont :

- `GL_CLAMP_TO_EDGE` (valeur par défaut)
- `GL_REPEAT`
- `GL_MIRRORED_REPEAT`

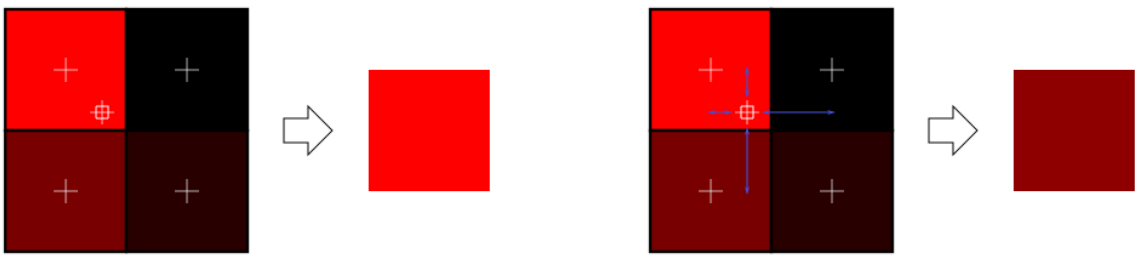
Testez ces différents paramètres, et déduisez en le comportement qu'ils définissent.

Filtrage

Lorsqu'on utilise une texture, le fait que celle-ci soit échantillonnée (ie. définit sur un ensemble discret) crée certains artefacts.

Un de ces artefact apparaît lorsqu'on s'approche très près d'un objet. Faites le test. Que constatez vous ?

Ce problème s'appelle la *magnification*. Il vient du fait que pour déterminer la couleur à associer à un pixel, OpenGL choisit le pixel de la texture (aussi appelé *texel*) le plus près de la coordonnée de texture associée au pixel. Une solution simple à ce problème est le filtrage linéaire. Dans ce cas, la couleur finale est calculée à partir des couleurs des quatre pixels les plus proches, et interpolée linéairement (cf. figure 1)



Choix de la couleur du pixel le plus près. Interpolation bilinéaire des couleurs des pixels les plus près.

Figure 1

Pour activer le filtrage linéaire de texture, appelez `glTexParameteri` pour mettre le paramètre `GL_TEXTURE_MAG_FILTER` à la valeur `GL_LINEAR` (`GL_NEAREST` pour revenir à l'état original). Testez cette fonctionnalité.

Lorsqu'on observe un objet de loin, on observe également un problème de pixelisation. On parle alors de minification. Pour réglez, tenez de mettre le paramètre `GL_TEXTURE_MIN_FILTER` à la valeur `GL_LINEAR`. Que constatez vous ?

MipMapping

Le théorème de Shannon spécifie qu'un signal doit être échantillonné à une fréquence au moins deux fois supérieure à sa fréquence propre. C'est parce que ce théorème n'est pas respecté que le problème de minification apparaît, même avec le filtrage linéaire.

Dans le cas des textures, un solution existe : créer une version basse résolution de la texture, ce qui a pour conséquence d'abaisser sa fréquence. En pratique, même avec une texture à moindre résolution, on peut encore avoir de la minification. C'est pourquoi on crée plusieurs version de la textures, chacune à différentes résolutions, comme dans la figure 2.

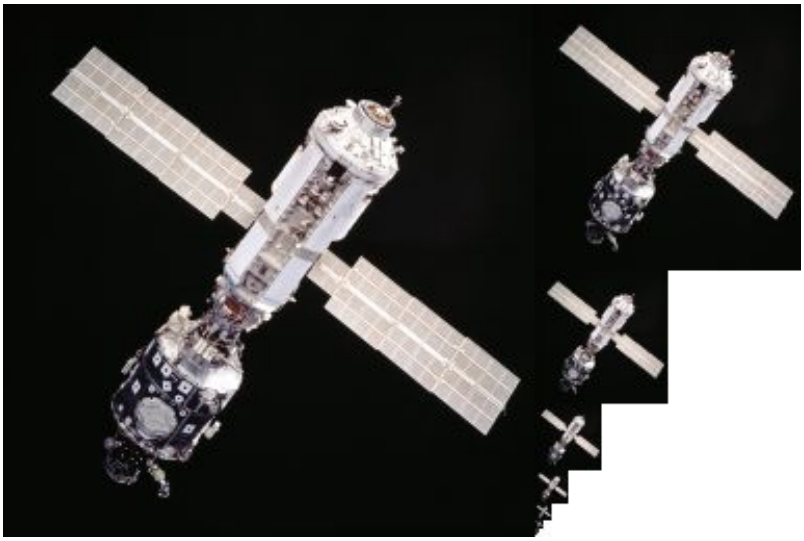


Figure 2: Exemple de mipmap

Pour créer une mipmap à partir d'un texture, il faut appeler `glGenerateMipmap(GL_TEXTURE_2D)` après avoir lié la texture.

Ensuite, pour l'utiliser, il faut mettre le paramètre `GL_TEXTURE_MIN_FILTER` à l'une des valeurs suivantes :

- `GL_NEAREST_MIPMAP_NEAREST`
- `GL_NEAREST_MIPMAP_LINEAR`
- `GL_LINEAR_MIPMAP_NEAREST`
- `GL_LINEAR_MIPMAP_LINEAR`

Testez la fonctionnalité associée à chacun de ces paramètres.

Bonus

Afin de manipuler plus facilement les différents paramètres que nous venons de voir, vous pouvez créer une fonction associant des touches (`F1`, `F2`, ... par exemple) à un appel à `glTexParameteri` avec différent paramètres. Cette fonction sera appelée en même temps que la fonction de contrôle de la vue dans la boucle de rendu.

Créer une seconde texture et combiner les 2 dans vos shaders.