TP4 : Texture Mapping.

L'objectif de ce TP est d'apprendre l'utilisation basique des textures avec OpenGL permettant déjà d'avoir des résultats intéressants. Les possibilités d'utilisation des textures sont immenses et découvrir l'ensemble de ces possibilitées dépasse le cadre du TP mais il est évident que n'importe qui de passionné trouvera son bonheur en cherchant par lui-même. Le TP se fera en binôme, un compte rendu comportant des captures d'écran des différents résultats, des commentaires ainsi que les réponses aux questions posées est à rendre.

Important : Dans la première partie du TP, seul le simple maillage d'un triangle sera utilisé. Vous utiliserez les fichiers `meshLoader.*` fournies pour écrire vos coordonnées textures ainsi que le nouveau CMakeList.txt

Comment utiliser une texture avec *OpenGL* ?

Exercice 1 : Le chargement des coordonnées de texture

Comme pour les positions et normales des sommets, créer un *VBO* pour les coordonnées de texture. Les coordonnées de texture sont accessible depuis l'attribut `uvcoord` du maillage.

Ajouter les coordonnées de texture dans le VBO et envoyer les au shader

1. Déclarer un nouveau VBO ainsi qu'un nouvel attribut pour les coordonnées dans *glwidget.h*.
2. N'oublier pas de détruire le buffer une fois que le programme est terminé.
3. A l'initialisation générer, lier le VBO et envoyer les coordonnées à la mémoire GPU.
4. Pendant le dessin activer/désactiver le buffer et faites le lien avec le shader.

Modifier les shaders

1. Déclarer les coordonnées de texture (`vec2`)
2. Interpoler les coordonnées du vertex shader au fragment shader (`varying`)

Tester si les coordonnées sont accessibles en les affichant

```
gl_FragColor = vec4( uv, vec2(1) );
```

1. Observer les couleurs, pourquoi bleu, magenta et blanc ?

Exercice 2 : Création d'une texture

Comme avec les autres objets OpenGL, une texture est controlée via un entier unique.

– Créer une fonction `createTexture` qui sera appelée à l'initialisation.

Compléter la fonction `createTexture` en utilisant les opérations suivantes

```
#include <QGLWidget>

QImage img = QGLWidget::convertToGLFormat( QImage("crate.jpg") );
if( img.isNull() )
{
        std::cerr << "Error Loading Texture" << std::endl;
        exit(EXIT_FAILURE);
}
//Generate a texture buffer in GPU memory controlled by an id
glGenTextures(...);
```

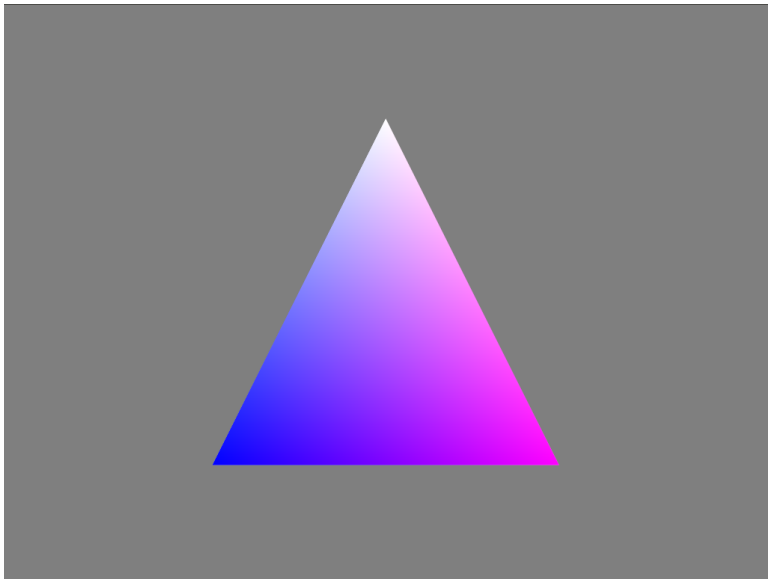


Figure 1: Visualisation des coordonnées uv

```
//Activate the texture in the machine state
//All following functions will operate on this texture
glBindTexture(GL_TEXTURE_2D, ...);
//Choose filtering mode for minification
glTextParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
//Choose filtering mode for magnification
glTextParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAX_FILTER, GL_NEAREST);
//Choose the border mode in the U direction
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
//Choose the border mode int the V direction
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
//Load the image into GPU memory
glTexImage2D(GL_TEXTURE_2D,...);
//Release the texture once we have done what we wanted
glBindTexture(GL_TEXTURE_2D, 0);
```

Note : Nous reviendrons sur les options de répétitions de texture et de filtrage

- Vous choisirez le format interne `GL_RGBA32F`.
- Le format de l'image Qt est `GL_RGBA`.
- Le type de l'image Qt est `GL_UNSIGNED_BYTE`.
- Accéder aux données de l'image Qt peut être fait via la fonction `image.bits()`

Enfin, n'oubliez par de détruire la texture une fois le programme terminé en utilisant

```
glDeleteTextures(...)
```

Exercice 3 : Utiliser la texture dans le shader

Comme pour les variables uniformes on lie l'objet texture avec le shader.

Faire le lien entre la texture et le shader

1 Comme pour les positions et les normales, obtenez une *location* pour la texture.

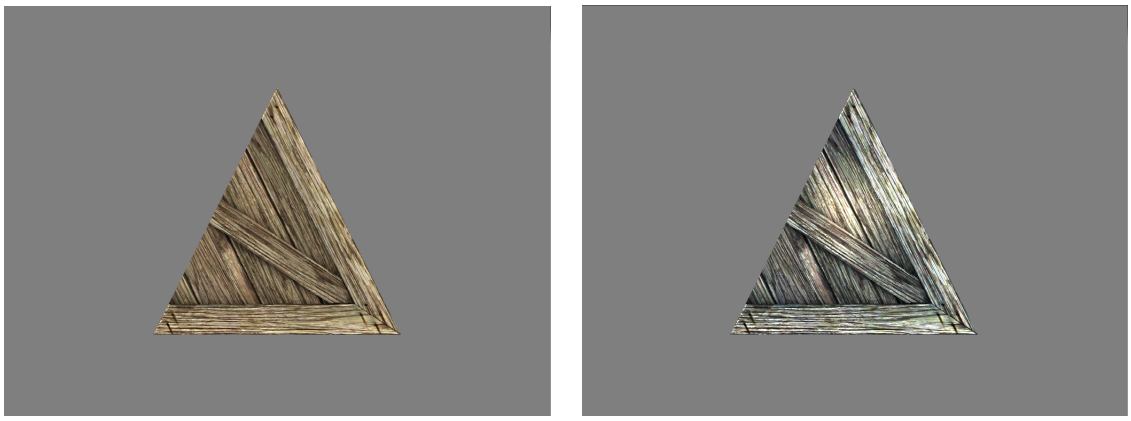
2 Les textures ont déjà été envoyées au GPU, il suffit de spécifier à la machine à état quelle texture est utilisée à l'aide de la *texture location*. Au moment du rendu :

```
//Activate texture unit 0
glActiveTexture(GL_TEXTURE0);
//Activate the texture (in the texture unit 0)
glBindTexture(GL_TEXTURE_2D, ...);
//Tell the shader to use the texture in the texture unit 0
glUniform1i(textureLocation, 0);
```

Note : N'oubliez pas le `release` de la texture une fois l'appel au dessin terminé.

Utiliser la texture dans le *fragment shader*

- Déclarer la texture (type = `sampler2D`).
- Accéder à la texture à l'aide de la fonction GLSL texture2D(...).
- Note : Utiliser les coordonnées uv pour accéder à la texture.
- Afficher la couleur obtenue dans la texture.
- Combiner la texture avec l'éclairage de Phong en utilisant une multiplication.



Texture sans éclairage Texture avec éclairage

Figure 2: Visualisation de la texture et combinaison

Les modes de texture

- Essayer de multiplier les coordonnées de texture par un scalaire (dans le vertex shader ou dans le fragment shader). Qu'observez-vous ?
- Essayer différents mode de **wrapping** de textures (`GL_CLAMP, GL_CLAMP_TO_EDGE, GL_REPEAT, GL_MIRRORED_REPEAT`) dans la fonction `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_*, ...)`.

Les fonctions de Mapping

Appliquer une texture à un maillage revient à spécifier à OpenGL une correspondance entre une partie de l'image et l'objet 3D. Ceci est fait à l'aide des **coordonnées UV** et s'appelle un **mapping**.

Différents type de texture existent (1D, 2D, 3D, ...), dans ce TP on s'intéresse aux texture 2D. On doit donc faire correspondre un point 3D avec un point 2D à l'aide d'un **mapping** :

$$f : (x, y, z) \rightarrow (u, v) \tag{1}$$

Il existe de nombreux fonctions de mapping pour les textures. On peut faire la distinction entre les mappings pouvant être générés automatiquement et ceux nécessitant une activité artistique.

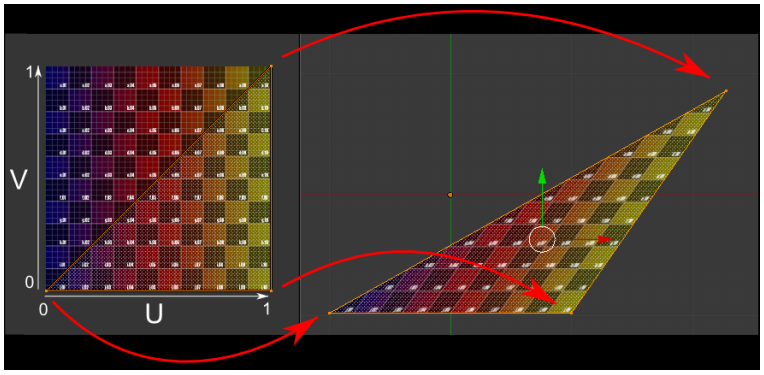
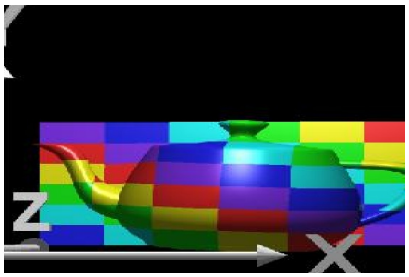


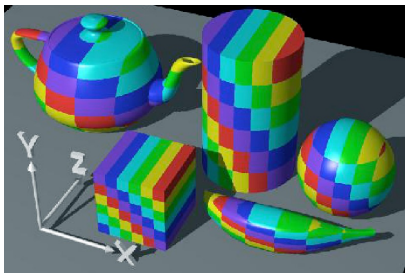
Figure 3: A chaque sommet du triangle est associé un sommet de l'image. La texture est distordue sur le triangle.

Les mapping "automatiques"

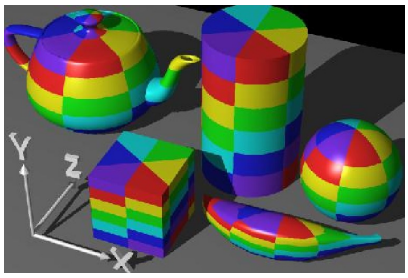
Ils permettent de calculer automatiquement les coordonnées UV. Ce type de mapping répond à des besoins simples comme le texturage d'une sphère (planète), d'un cylindre (canette). Voici quelques exemples :

FlatMapping

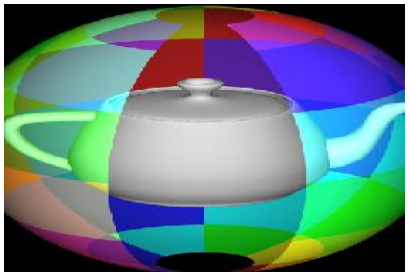
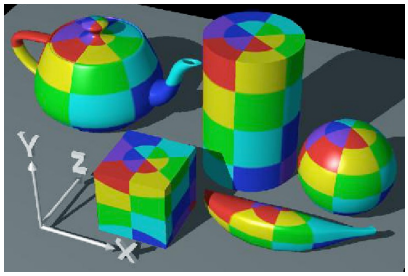




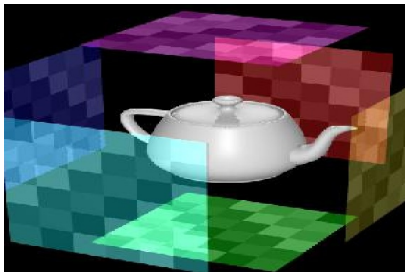
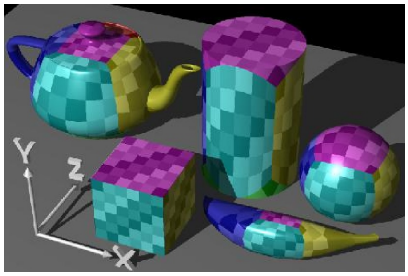
Cylindrical Mapping



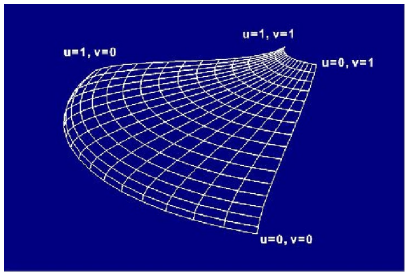
Spherical Mapping

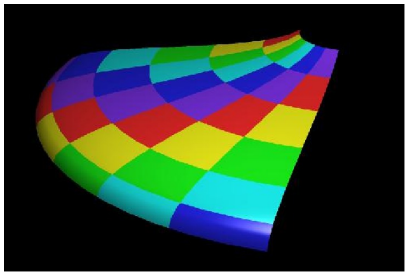



Cube Mapping

Parametric Mapping





UV Mapping

Le design et la mapping d'une texture pour un modèle 3D fait partie des compétences des infographistes. Le mapping automatique de forme complexe fait l'objet de recherche mais pour l'instant une intervention artistique reste souvent obligatoire. Ce mapping manuel a un nom, c'est l'**UV mapping**.

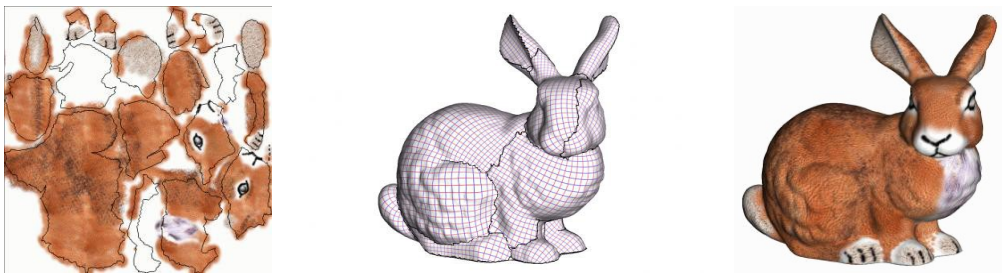


Figure 4: Exemple de UV mapping

Un designer a fait correspondre les coordonnées d'un patron, une texture à un modèle 3D.

Exercice : UV Mapping

Dans la class `MeshLoader` :

- Modifier la fonction `squareCoord()` pour texturer un carré.
- Texturer un cube : créer le maillage ainsi qu'une fonction cubeCoord() contenant les coordonnées uv du cube. Quelle est le problème ? Implémentez une solution.