OpenGL - TD 04

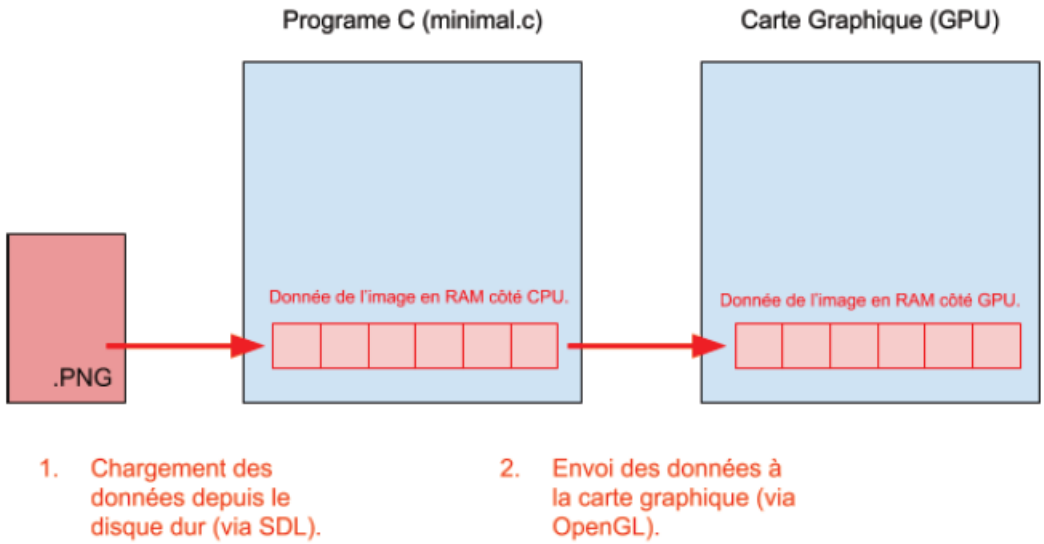
Textures

Lors de cette séance, nous apprendrons à manipuler des textures via OpenGL.

En infographie, une texture désigne une image 2D qui vient s'appliquer sur la surface d'une géométrie rendue dans une scène 3D.

Pour pouvoir afficher une texture en OpenGL, il faut :
- charger l'image voulue en mémoire (en RAM, coté CPU)
- réserver la mémoire pour une texture (sur GPU)
- envoyer les données de l'image chargée sur le CPU vers la carte graphique
- utiliser la texture au moment d'afficher un objet (ex : le quad mentionné plus haut)
- penser à libérer la mémoire allouée sur le CPU et le GPU.

Il est important ici de bien faire la distinction entre :
- les données coté CPU (fichier image ouvert et pointeur sur les données qu'il contient)
- les données envoyées sur le GPU



Prérequis :
Pour ce TP vous aurez besoin soit :
- de travailler à partir des nouveaux fichier *makefile* et *minimal.c* fournis avec ce TD
- d'ajouter `<SDL2/SDL_image.h>` dans les fichiers sources que vous réutilisez,
et d'ajouter le flag `-lSDL2_image` dans votre *makefile*

Exercice 01 – Chargement d'une image via SDL_image

Le module SDL_image de la librairie SDL permet de charger en RAM une image depuis le disque dur, via la fonction `IMG_Load` . Sa signature est la suivante :

```
SDL_Surface* IMG_Load(const char* filename)
```

Le type `SDL_Surface` est une structure contenant diverses informations sur l'image chargée, ainsi qu'un pointeur sur un tableau contenant les données composant cette image (i.e. :pixels). C'est le contenu de ce tableau qui devra être envoyé via OpenGL sur la carte graphique.

Vous pourrez trouver une description détaillée de la structure `SDL_Surface` sur cette page : https://wiki.libsdl.org/SDL_Surface

La fonction `SDL_FreeSurface` permet de libérer la mémoire allouée pour une image.
Sa signature est la suivante :

```
void SDL_FreeSurface(SDL_Surface* surface)
```

A faire :
01a. Utilisez les fonctions ci dessus pour charger l'image */doc/logo_imac.jpg*
01b. Pensez à libérer la mémoire allouée, avant de quitter le programme.

02. Vérifiez que l'image à bien été chargée.
Pour cela, assurez vous que le pointeur renvoyé par `IMG_Load` est différent de `NULL` ,
et dans le cas contraire, affichez un message d'erreur.

Exercice 02 – Envoi des données à la carte graphique

Maintenant que les données sont chargées en RAM, il faut les envoyer à la carte graphique.

En premier lieu, vous aurez besoin de créer un objet texture.

A faire :
01a. Allouez la mémoire texture sur la carte graphique.

En OpenGL les textures sont représentées par un identifiant de type `GLuint` , comme pour beaucoup d'autres objets. Initialiser une texture se fait via la fonction `glGenTextures` :

```
void glGenTextures(GLsizei n, Gluint* textures)
```

Le paramètre `n` correspond au nombre de textures qu'on veut générer. (1 en l'occurrence).
Le paramètre `textures` est un pointeur sur l'identifiant de la première texture à générer. (Dans le cas où on voudrait créer plusieurs textures, il faudra passer comme argument l'adresse de la première case d'un tableau de `GLuint`)

01b. Pensez à libérer l'espace alloué pour la texture, avant de quitter le programme :

```
glDeleteTextures(1, &textureID);
```

Un objet texture OpenGL ne se résume pas qu'à une zone mémoire, il s'accompagne aussi de différents paramètres concernant l'usage de la texture.
Certains de ces attributs doivent être fixés manuellement après la création d'une texture, tel que le filtre de minification utilisé pour cette texture (`GL_TEXTURE_MIN_FILTER`).
(Un filtre de minification s'applique à une texture lorsque sa taille affichée à l'écran est inférieure à a sa taille originelle)

A faire :
02. Attachez la texture au point de bind `GL_TEXTURE_2D` , en utilisant `glBindTexture` :

```
glBindTexture(GLenum target, GLuint texture)
```

Le paramètre `target` correspond au point de bind (`GL_TEXTURE_2D` en l'occurrence).
Le paramètre `texture` est l'identifiant de la texture qui doit être attachée.

Une fois la texture attachée, il est possible de modifier ses paramètres.

03. Changez le filtre de minification en utilisant la fonction `glTexParameteri` :

```
glTexParameteri(GLenum target, Glenum pname, Glint param)
```

Le paramètre `target` correspond au point de bind (`GL_TEXTURE_2D` en l'occurrence).
Le paramètre `pname` est le nom de l'attribut à modifier (`GL_TEXTURE_MIN_FILTER`)
La paramètre `param` est la valeur qu'on souhaite donner à l'attribut.

Nous utiliserons ici un filtre de minification linéaire (`GL_LINEAR`).

Il est maintenant possible d'envoyer les données d'image à la texture sur la carte graphique.

Les données sont actuellement stockées en RAM dans la `SDL_Surface` crée à l'exercice 01.
L'idée est de demander à OpenGL de copier ce bloc de données sur la carte graphique.

A faire :
04a. Envoyez les données de l'image vers le GPU, en utilisant la fonction `glTexImage2D` :

```
void glTexImage2D(
    GLenum target, GLint level, GLint internalFormat,
    GLsizei width, GLsizei height, GLint border,
    GLenum format, GLenum type, const Glvoid* data)
```

Le paramètre `target` correspond au point de bind (`GL_TEXTURE_2D` en l'occurrence).
Le paramètre `level` est en lien avec le mipmap, une technique d'optimisation abordée en 2ème et 3ème années. Nous n'utilisons pas cette fonctionnalité et mettrons cette valeur à 0.
La paramètre `internalFormat` correspond au format qu'auront les données OpenGL.
Nous utiliserons dans un premier temps `GL_RGB` .

Les paramètres `width` et `height` sont les dimensions de la texture.
Vous pourrez trouver ces valeurs dans la `SDL_Surface` crée à l'exercice 01.

Le paramètre `border` est utilisé pour spécifier une bordure.
Nous ne souhaitons pas afficher de bordure et mettrons cette valeur à 0.

Le paramètre `format` correspond au format des données de l'image stockée en RAM.
Nous utiliserons dans un premier temps `GL_RGB` .
Attention :
Une mauvaise utilisation de ce paramètre peut causer des erreurs de segmentation.

Le paramètre `type` correspond au type de données actuellement en RAM.
La SDL utilise des données de type `unsigned char` , correspondant a 1 octet.
Nous devrons donc passer la valeur `GL_UNSIGNED_BYTE` .

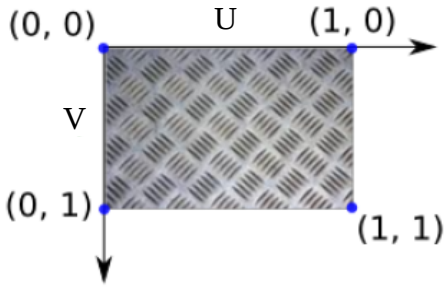
Le paramètre data est un pointeur sur les données de l'image stockée en RAM.
Vous pourrez trouver ce pointeur dans la `SDL_Surface` crée à l'exercice 01.

04b. Pensez à détacher la texture de son point de bind, une fois les données chargées.

```
glBindTexture(GL_TEXTURE_2D, 0);
```

Notice – Coordonnées de texture

Lorsque vous envoyez les données de votre image à la carte graphique, OpenGL lui associe un repère en 2D de la manière suivante :



Les couleurs et motifs présents dans la texture sont accessibles au moyen de ces coordonnées. On fait désormais abstraction des dimensions en pixels de l'image, les coordonnées de texture sont toujours comprises entre 0 et 1, OpenGL se charge de faire la correspondance.

Ces coordonnées de textures servent à définir de quelle manière la texture vient s'appliquer sur notre objet, en faisant correspondre une position dans la texture à chaque sommet.
Cette technique est parfois appelée uv mapping.

Pour définir les coordonnées de texture d'un sommet, utilisez la fonction `glTexCoord2f` , avant de faire appel à `glVertex2f` :

```
void glTexCoord2f(GLfloat u, GLfloat v)
```

Le paramètre `u` correspond à la coordonnée de texture en abscisse.
Le paramètre `v` correspond à la coordonnée de texture en ordonée.

Note :
Les coordonnées `u` et `v` sont parfois appelées respectivement `s` et `t` .

OpenGL interpole les coordonnées de textures entre les sommets pour les points se trouvant entre ces derniers, cela lui permet d'afficher le contenu de la texture de manière continue.

(Si vous avez dessiné des objets avec des sommets de différentes couleurs, vous aurez certainement remarqué qu'OpenGL interpole également les couleurs pour obtenir un dégradé)

Exercice 03 – Affichage de la texture

Il est désormais temps d'afficher la texture à l'écran.
Dans cette exercice, vous allez dessiner un quad, mais au lieu de le colorer via une `glColor` , vous lui appliquerez la texture initialisée dans les exercices 01 et 02.

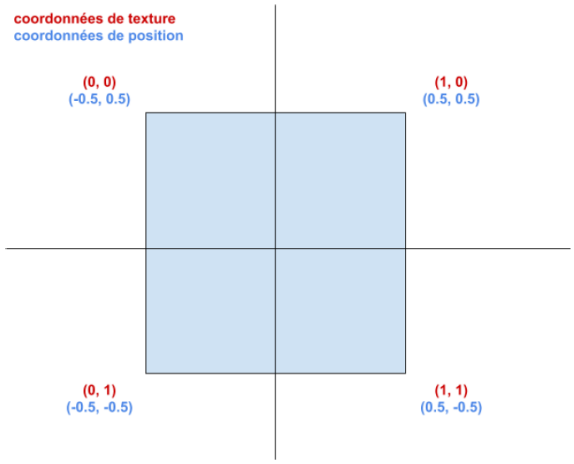
A faire :

01. Dessinez un quad dans votre boucle d'affichage.
(N'utilisez pas la fonction de dessin canonique du TD02, elles n'est pas adaptée à cet exercice)

Vous pouvez maintenant appliquer la texture sur ce quad.

02a. Activez le texturing : `glEnable(GL_TEXTURE_2D);`
02b. Attachez la texture au point de bind : `glBindTexture(GL_TEXTURE_2D, textureID);`
02c. Pensez à détacher la texture, après le dessin : `glBindTexture(GL_TEXTURE_2D, 0);`
02d. Pensez à désactiver le texturing après le dessin : `glDisable(GL_TEXTURE_2D);`

03. Appliquez les coordonnées de texture sur les sommets du quad, via `glTexCoord2f` .



Optionnel :
04. Appliquez des transformations à votre quad.
Constatez que la texture se conforme à ces transformations.