Synthèse d'Image - TD04

Texturing

Dans ce TD, nous allons apprendre à charger des textures dans OpenGL et à les afficher à l'écran.

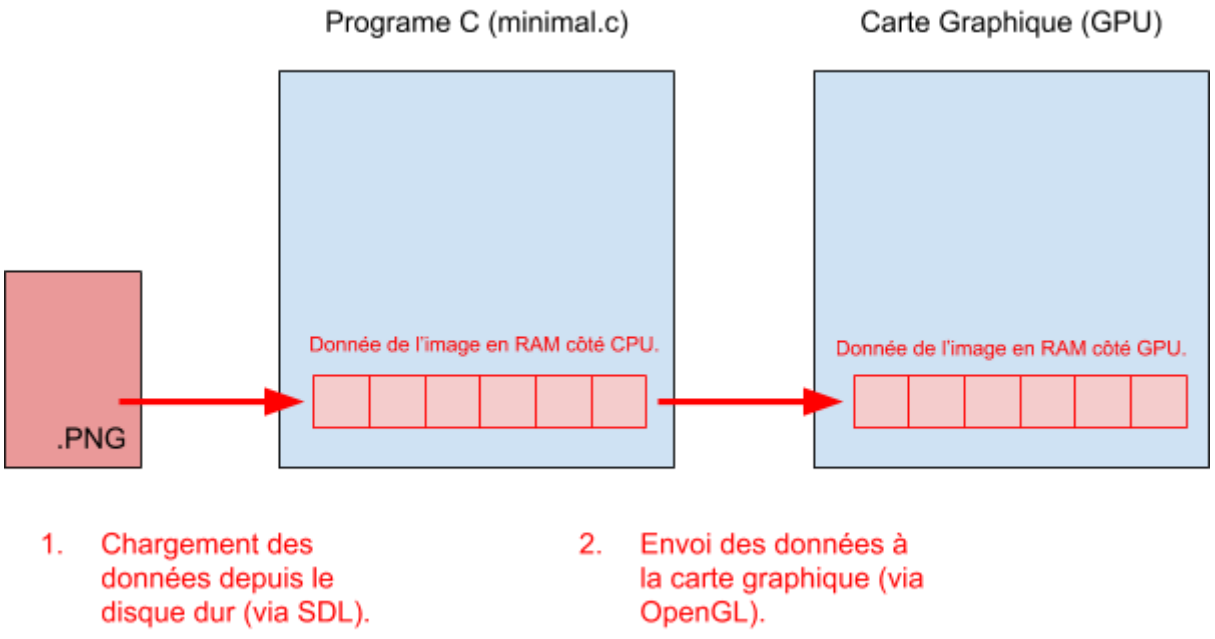
Principe de base

En infographie, Le **texturing** concerne tout ce qui touche à la création et à l'affichage d'images dans une scène 3D. Plus spécifiquement, une texture est une image que l'on va "plaquer" sur un objet 3D qui sera rendu à l'écran.

Pour pouvoir afficher une texture en OpenGL, il faut :

1. Charger cette image dans votre programme.
2. Créer un objet texture en OpenGL (représenté par un entier GLuint).
3. Envoyer les données de la texture chargée à la carte graphique.
4. Afficher la texture sur un objet (par exemple un quad).
5. A la fin du programme, libérer la mémoire qu'OpenGL a alloué pour la texture.

Il faudra bien faire la différence entre le **chargement des données côté CPU** (c'est à dire ouvrir l'image et récupérer un pointeur sur les données), et **chargement des données côté GPU** (c'est à dire l'envoi des données à la carte graphique).



Pour ce TD, vous repartirez avec le nouveau fichier minimal.c et le nouveau Makefile.

EXERCICE 01 : Chargement des données de textures en RAM

La première étape consiste à charger la texture dans notre programme. Plus exactement, cela consiste à lire les données de l'image depuis le disque dur et à recréer un tableau de données identiques dans le programme.

Fort heureusement, vous n'aurez pas à développer une telle fonction car de nombreux outils existent. Dans notre cas, nous allons utiliser le module **SDL_image** de la librairie SDL qui fournit différents outils pour manipuler les images.

Prérequis : Vous devez au choix :
- utiliser les nouveaux fichiers minimal.c et Makefile
- ajouter l'import <SDL/SDL_image.h> à votre fichier minimal.c et la librairie -lSDL_image à votre Makefile

Nous allons utiliser la fonction **`IMG_Load`** fournie par SDL_image. La signature de la fonction est la suivante :

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

Le type de retour **SDL_Surface** est une structure contenant diverses informations sur l'image (taille, format, etc.) ainsi qu'un pointeur sur le tableau de données chargées. Nous en auront besoin pour préciser à OpenGL le types de données à utiliser.

Lorsque vous aurez terminé d'utiliser les données de l'image, il faudra libérer la mémoire allouée via la fonction `SDL_FreeSurface(SDL_Surface* surface)`.

- **Utilisez les deux fonctions ci dessus pour charger l'image *logo_imac_400x400.jpg* dans votre programme, puis libérer les ressources juste après.**
- **Testez si le pointeur retourné par `IMG_Load` n'est pas NULL, auquel cas cela serait synonyme d'une erreur.**

EXERCICE 02 : Envoi des données à la carte graphique

Maintenant que les données sont chargées en RAM, nous allons les envoyer à la carte graphique dans l'optique de les afficher à l'écran. Pour cela, il faut suivre la procédure suivante :

1. Créer une texture OpenGL via la fonction `glGenTextures`
2. Configurer les paramètres de la texture via `glTexParameter`
3. Envoyer les données de texture à la carte graphique via `glTexImage2D`

Création d'une texture OpenGL

En OpenGL, les textures sont représentées par un identifiant de type GLuint (comme beaucoup d'autres objets). Pour initialiser une texture, il faut faire appel à la fonction :

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

- le paramètre *n* correspond au nombre de textures qu'on veut générer (ici, 1)
- *textures* est un pointeur sur l'identifiant de la première texture à générer (dans le cas où on voudrait plus d'une texture, il faudrait que ce pointeur soit le début d'un tableau de variables de type GLuint).

Notre appel à fonction devient donc :

```
GLuint textureID;
glGenTextures(1, &textureID);
```

1. **Ajoutez ces deux lignes de code après le chargement de la texture dans votre programme.**
2. **Après le code de votre boucle de rendu (code commenté pour le moment), libérez l'espace alloué pour la texture via la fonction `glDeleteTextures(1, &textureID)`.**

Configuration des paramètres de texture

En plus d'un tableau de données, une texture OpenGL possède de nombreux paramètres. Certains de ces paramètres ont des valeurs par défaut, et d'autres doivent être fixés par le développeur. C'est le cas du paramètre de **filtre de minification**. Le filtre de minification est un filtre qui s'applique à la texture lorsque sa taille à l'écran est plus petite que sa taille originelle.

Pour pouvoir modifier ce paramètre, il faut d'abord **attacher notre texture à un "point de bind"**, puis dire à OpenGL de changer l'attribut de la texture attachée à ce

point de bind. Il existe plusieurs "points de bind" concernant les textures et dans notre cas nous utiliserons le point **GL_TEXTURE_2D**.

Pour attacher une texture à un point de bind, il faut utiliser la fonction :

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

- *target* correspond au point de bind (ici GL_TEXTURE_2D)
- *texture* correspond à l'ID de notre texture

Une fois notre texture **bindée**, nous pouvons alors changer ses paramètres via la fonction :

```
glTexParameteri(GLenum target, GLenum pname, GLint param);
```

- *target* correspond au point de bind utilisé (GL_TEXTURE_2D)
- *pname* est le nom de l'attribut à changer (pour le filtre de minification, il faut utiliser GL_TEXTURE_MIN_FILTER)
- *param* correspond à la valeur à donner à l'attribut (nous utiliserons GL_LINEAR)

2. **Dans votre programme, modifiez le paramètre de filtre de minification de votre texture.**

Envoi des données à la carte graphique

Il est maintenant temps d'envoyer les données de texture à la carte graphique, sans quoi il sera impossible de l'afficher. L'envoi des données se fait via la fonction :

```
gltexImage2D(
      Glenum target,
      GLint level,
      GLint internalFormat,
      GLsizei width,
      GLsizei height,
      GLint border,
      GLenum format,
      GLenum type,
      const GLvoid* data);
```

- *target* correspond au point de bind sur lequel envoyer les données
- *level* est un paramètre de **mipmap**, une technique d'optimisation que vous verrez en IMAC2 ou 3. Nous mettrons la valeur **0**.
- *internalFormat* correspond au format qu'auront les données OpenGL. Nous utiliserons dans un premier temps **GL_RGB**.
- *width* et *height* sont les dimensions de la texture (disponibles dans la structure SDL_Surface).

- *border* est un paramètre utilisé pour spécifier une bordure. Nous n'en voulons pas donc nous mettrons **0**.
- *format* correspond au format des données actuellement en RAM (celles côté CPU). C'est un paramètre difficile à déterminer qui peut provoquer facilement des segmentation faults si mal précisé. Dans un premier temps nous utiliserons **GL_RGB**.
- *type* correspond au type de données actuellement en RAM. La SDL utilise des données de type *unsigned char*, et un *unsigned char* est codé sur 1 octet (1 byte en anglais). La bonne valeur à passer est donc **GL_UNSIGNED_BYTE**.
- *data* correspond au pointeur sur la tableau de données actuellement en ram (disponibles dans la structure SDL_Surface).

Notre appel à fonction ressemblera donc à :

```
glTexImage2D(
      GL_TEXTURE_2D,
      0,
      GL_RGB,
      image->w,
      image->h,
      0,
      GL_RGB,
      GL_UNSIGNED_BYTE,
      image->pixels);
```

(où *image* est le pointeur sur la SDL_Surface retourné par la fonction IMG_Load)

Une fois les données chargées sur la carte graphique, il faut **détacher la texture de son point de bind** (on dit aussi **"debinder une texture"**) :

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

3. **Ajoutez à votre programme le chargement des données sur la carte graphique puis débindez la texture.**

EXERCICE 03 : Affichage de la texture (enfin!)

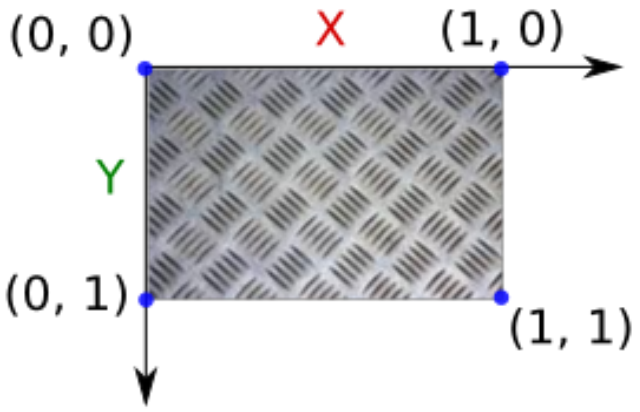
Le temps est venu d'afficher notre texture à l'écran. Comme dit dans l'introduction, une texture est toujours mappée sur un objet OpenGL. Même une texture qui semble être en arrière plan est toujours affichées sur un support (un très grand quad en fond de scène, une sphère qui englobe toute votre scène, une skybox, etc.). Dans ce TP, nous allons mapper notre texture sur un simple cube.

De la même façon que nous avons défini des couleurs pour chacun des sommets d'un quad, nous pouvons définir des coordonnées de texture pour ces sommets. Cela se fait via la fonction :

```
glTexCoord2f(u, v);
```

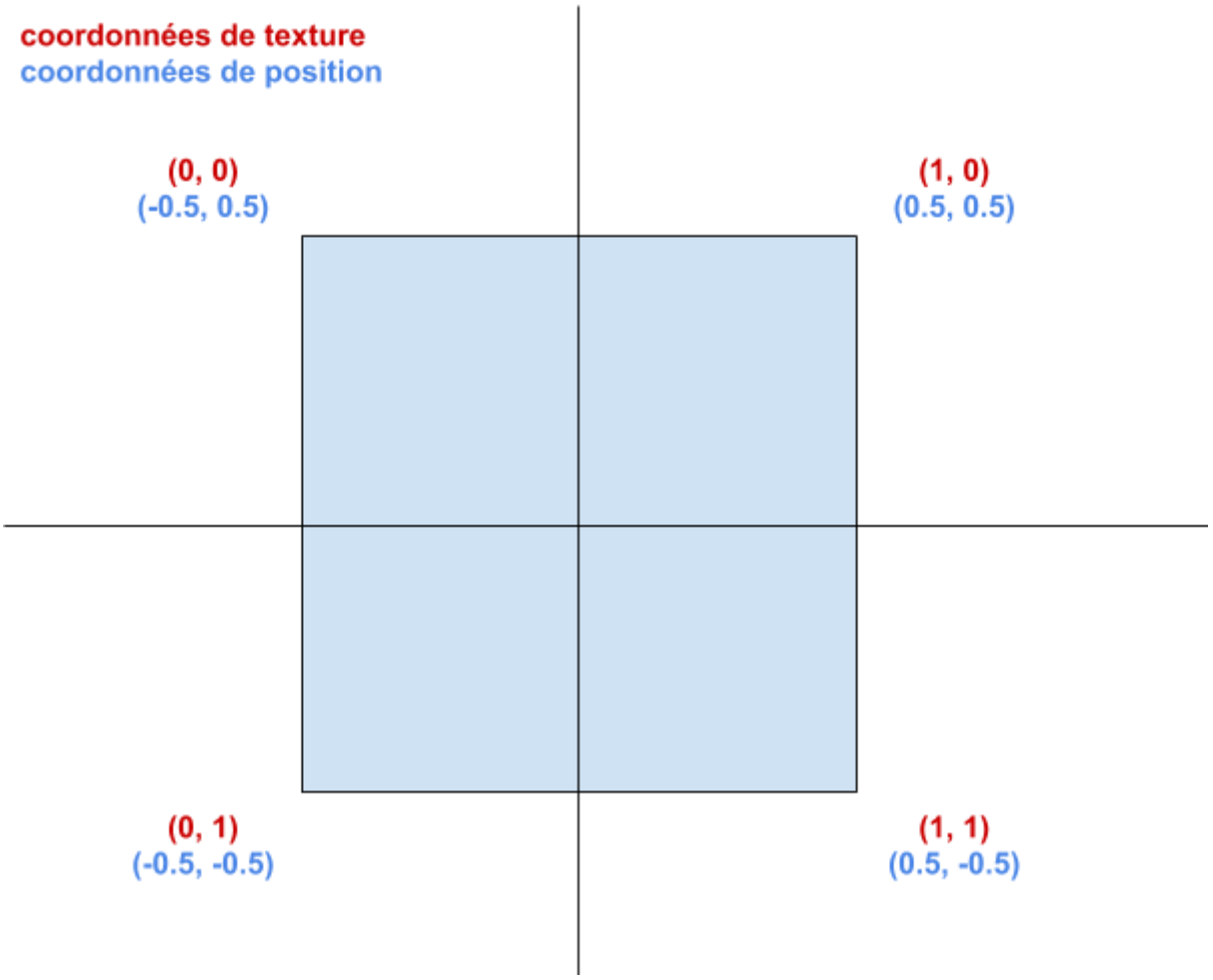
- *u* est la coordonnée de texture en abscisse (aussi appelée s)
- *v* est la coordonnée de texture en ordonnée (aussi appelée t)

Ces coordonnées de texture sont toujours comprises entre 0 et 1, et sont définies comme tel :



De même que pour la couleur, OpenGL se chargera de faire **l'interpolation** de ces coordonnées de texture pour tous les points qui se situent entre 2 sommets.

Pour afficher correctement notre image sur le quad, il faut donc utiliser les coordonnées de texture suivantes :



Il va donc falloir faire un appel à `glTexCoord2f` avant chaque appel à `glVertex2f` pour redéfinir les coordonnées de texture pour chaque sommet.

1. **Décommentez le code de la boucle de rendu.**
2. **Dans la boucle de rendu, commencez par dessiner un simple quad canonique (sans réutiliser de fonction existante).**
3. **Avant le dessin de chaque sommet, précisez les coordonnées de texture de ce sommet via la fonction `glTexCoord2f`. Exécutez votre programme. Que constatez vous ?**

A ce stade, votre cube devrait être tout blanc :/

Pas d'inquiétude, c'est tout à fait normal. Nous avons oublié de préciser à OpenGL quelle texture il fallait utiliser avant de dessiner. Pour cela, dans la boucle de rendu, commencez par appeler les deux fonctions suivantes :

```
glEnable(GL_TEXTURE_2D);  // On précise qu'on veut activer la fonctionnalité de texturing
glBindTexture(GL_TEXTURE_2D, textureID); // On bind la texture pour pouvoir l'utiliser
```

Après le dessin du quad, il faut systématiquement revenir à l'état de départ. Pour cela, appelez les deux fonctions suivantes après le code dessin.

```
glDisable(GL_TEXTURE_2D); // On désactive le sampling de texture
glBindTexture(GL_TEXTURE_2D, 0); // On débind la texture
```

4. **Faites en sorte d'afficher la texture sur le quad.**
5. **Appliquez des transformations à votre quad et constatez que la texture suit bien ces transformations.**
6. **Modifiez le paramètre du filtre de minification et appliquez un scale < 1.0 à votre quad pour voir les différences. Vous pouvez consulter ce lien pour davantage de détails : https://open.gl/textures**

EXERCICE 04 (bonus) : Une horloge

Modifiez votre programme pour qu'il affiche une horloge qui évolue en fonction du temps. Vous pouvez vous servir des images se trouvant dans le fichier .zip "numbers".

Quelques indices :

- Vous devrez générer plusieurs textures (1 par image)
- En fonction du temps, vous devrez déterminer l'heure actuelle et trouver le nombre d'heures, de minutes, et de secondes (utilisez <time.h>