DM n° 3 — Plaquage de textures en lancer de rayons

Antoine de Roquemaurel (G1.1)

1 Avant-propos

L'archive que vous avez obtenu sur Moodle contient plusieurs fichiers organisés comme suit :

`src/` Contient uniquement les fichiers modifiés, c'est-à-dire `main.cpp`, `texturefetch.cpp` et `color.hpp`.

`images/` Contient les différentes images produites lors de ce projet

`report_AntoinedeRoquemaurel.pdf` Le présent rapport que vous êtes en train de lire

Comme vous avez pu le voir, j'ai choisi de modifier la classe `Color`. Ces modifications n'étaient pas indispensables, mais permettait d'améliorer la lecture du code. Ainsi j'ai ajouté 3 méthodes :

- Opérateur `+=`, permettant d'éviter de faire `color = color + other`, le `+=` est plus facile à lire.
- De la même manière, l'opérateur `*=`.
- Méthode `reset()` qui remet toutes les composantes à `0.f` évitant de créer une nouvelle instance de `Color`.

2 Opérateur `getTextel()`

Une fois les textures intégrés, on peut observer un effet « d'escalier » sur le quadrillage de l'image, principalement au premier plan, comme le montre la figure 5.

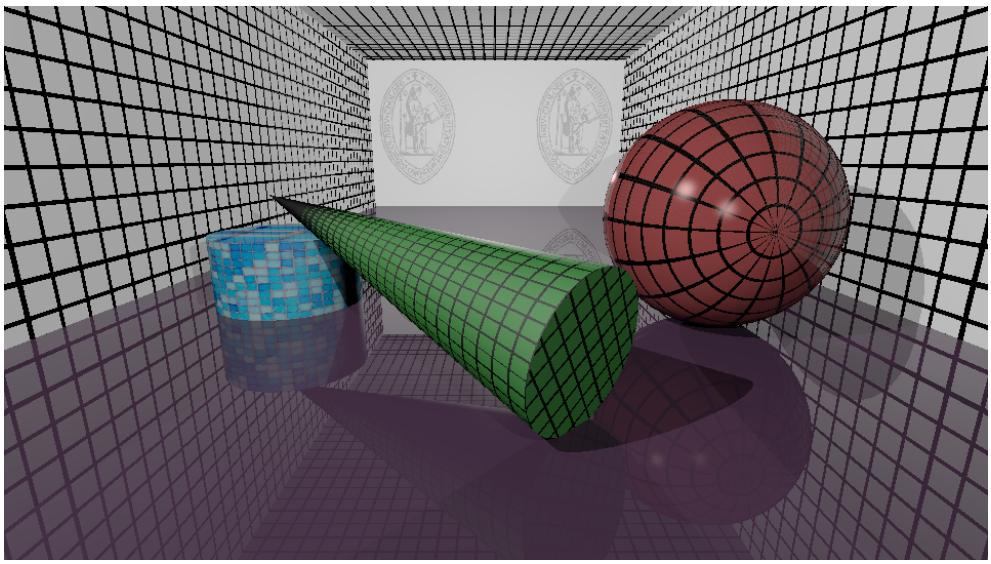

Figure 1 – Image obtenue après le développement de `getTextel()`

Cela vient du fait que nous essayons d'intégrer une texture 2D sur des figures 3D. Afin de bien représenter une profondeur sur nos textures 2D, nous les avons déformés afin que sa taille en premier plan soit plus grande que la taille en arrière plan. C'est cette transformation qui créer un effet de *crênelage* sur le devant de la scène.

3 Opérateur d'interpolation linéaire pour l'accès à un élément de texture

Afin d'améliorer la qualité de notre image, nous avons effectué une interpolation linéaire des pixels voisins. Le but de ce calcul est de prendre en compte les voisins pour afficher notre pixel, ainsi cela va créer un effet « d'adoucissement » dans les zones à fort contraste : le quadrillage par exemple.

C'est ainsi que notre quadrillage va maintenant avoir un effet de « flou » cachant le crénelage précédent.

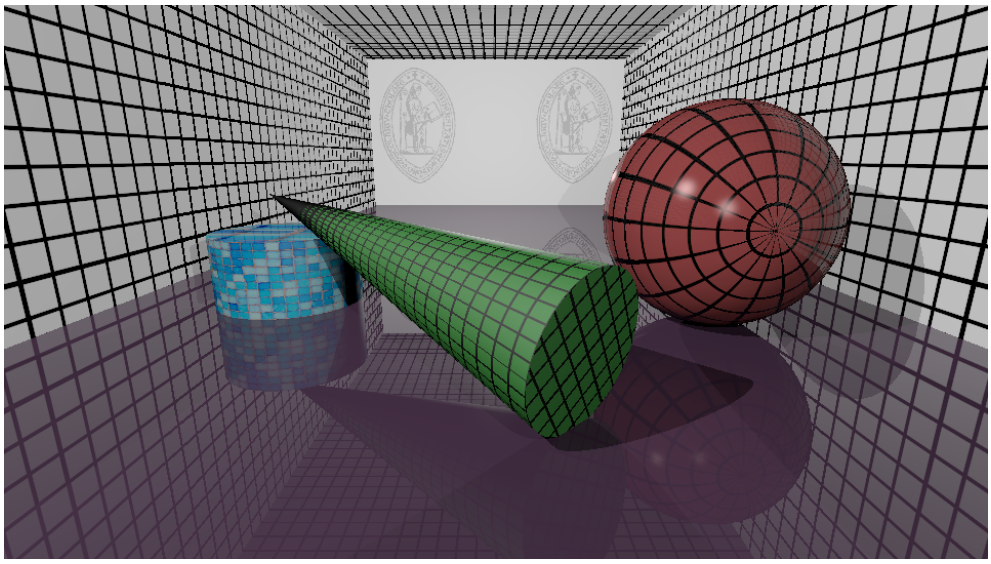

Figure 2 – Image obtenue après le développement de `interpolateTexture()`

4 Opérateur d'intégration pour le filtrage de texture

Une fois cette fonction développée, le temps d'exécution à grimpé en flèche comme le montre le listing 1.

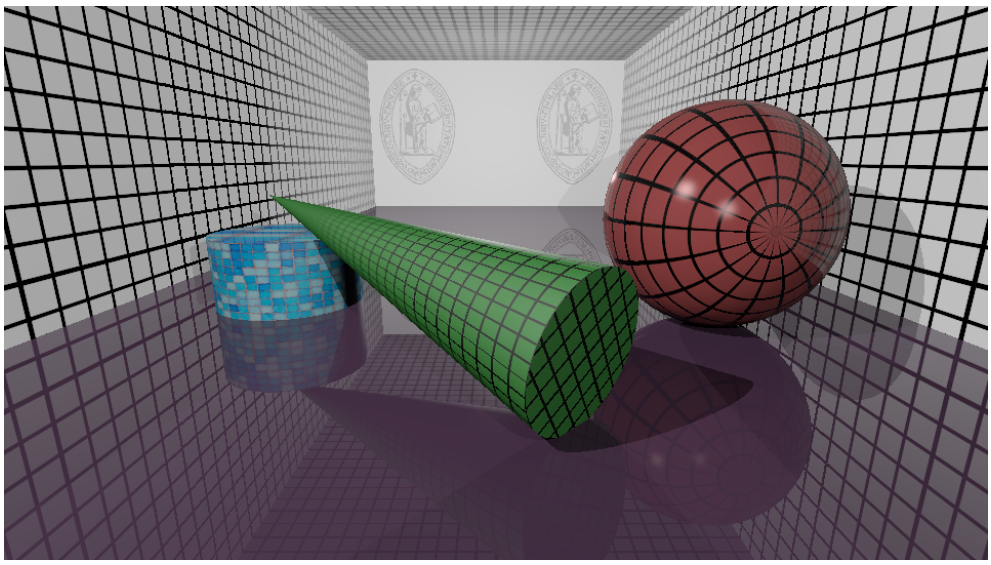

Figure 3 – Image obtenue après le développement de `integrateTexture()`

Cette méthode, par les calculs d'interpolation, permet d'effectuer un dégradé de flou au fond de l'image. Le flou s'applique ainsi en fonction de la profondeur : plus on est loin du 1^{er} plan, plus l'image sera flou, c'est ainsi que nous avons une meilleure impression de profondeur.

```
aroquemaurel@Luffy ~/projets/cpp/raytracing/bin <master> time ./main
./main  18,92s user 0,03s system 99% cpu 19,010 total
```

Listing 1 – Temps d'execution avec `interpolateTexture`

Comme dit précédemment, cette méthode à l'inconvénient de ralentir considérablement le temps d'exécution. Cela vient des calculs nécessaires pour cette interpolation, notamment un grand nombre de division.

5 Opérateur de pré-filtrage de texture

Cette fonction n'avait pas pour but d'améliorer l'image, mais uniquement de réduire le temps d'exécution. Cela a été un succès, après le développement de cette fonction, le temps d'exécution a été divisé par plus de 4 !

```
aroquemaurel@Luffy ~/projets/cpp/raytracing/bin <master> time ./main
./main  4,00s user 0,02s system 99% cpu 4,028 total
```

Grâce à cette méthode, nous avons réussi à atteindre un temps d'exécution convenable vis-à-vis de la qualité de l'image que nous avons obtenus comme le montre la courbe figure 4.

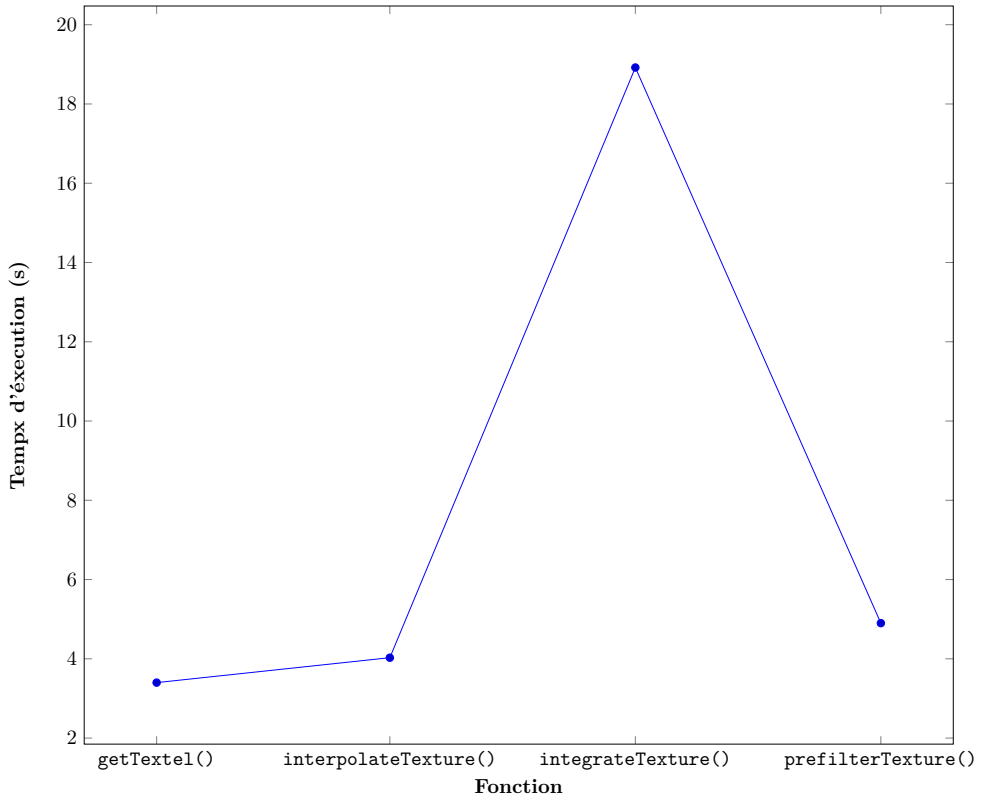

Figure 4 – Temps d'exécution des techniques d'améliorations de l'image

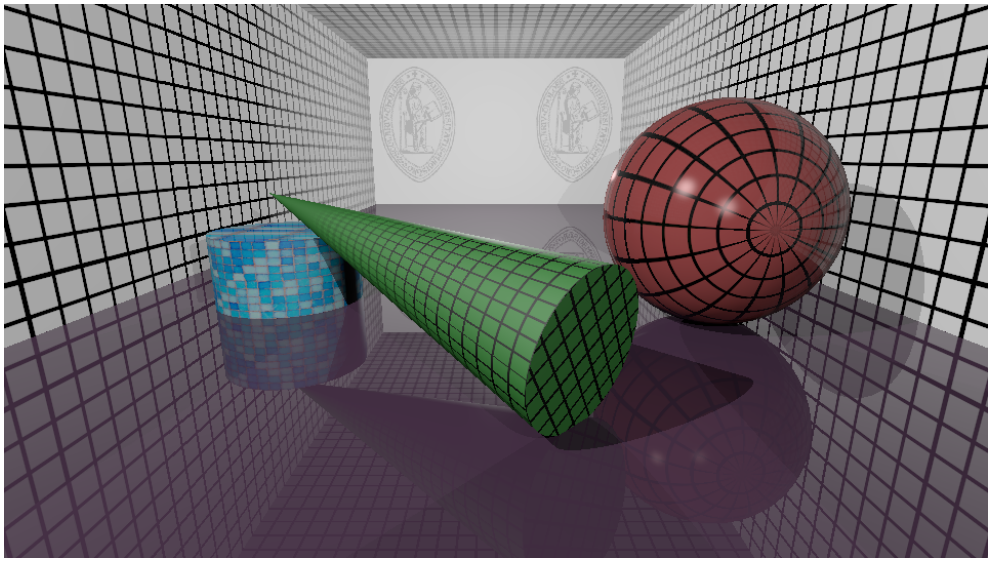

Figure 5 – Image obtenue après le développement de `prefilterTexture()`

6 Analyse et estimation des limites des techniques de filtrage implantées

La méthode que nous avons implémenté à permis d'améliorer considérablement l'image, ceci en créant des effets de flous et en lisant les effets de crénelage.

Cependant, cette méthode est loin d'être parfaite, en effet en premier lieu, il y a une perte d'information, notamment avec la moyenne des pixels voisins. Également, le temps d'exécution est assez important, et très gourmand en ressource, particulièrement pour de grandes images en HD. Ce temps de calcul a été diminué grâce à la section 5, mais cela reste assez important.