

# Synthèse d'image

---

<b>I. PROJET</b>	<b>2</b>
a. Description	2
b. Librairies	2
c. Références	2
<b>II. ORGANISATION DU PIPELINE</b>	<b>3</b>
<b>III. RESULTAT</b>	<b>4</b>
<b>IV. PROBLEMES</b>	<b>6</b>
a. Normales	6
b. Camera	6

## I. Projet

### a. Description

Le but du projet de synthèse d'image est de simuler l'action d'un pipeline graphique personnalisable. Pour résumer, il nous faut réaliser ce que permettent les API tels que OpenGL et DirectX : en entrée, des sommets, sommets qui seront transformés puis projetés sur l'écran, les faces (composées de trois sommets) seront alors remplies et l'ombrage calculé pour avoir en sortie, un rendu. Le choix était donné entre partir d'une base ou non, j'ai choisi de repartir de zéro.

### b. Librairies

J'ai utilisé principalement deux librairies. La première est surtout utilisée pour l'interface utilisateur. Il s'agit de QT, QT est une librairie de Trolltech, en C++ et orienté objet, offrant beaucoup de possibilités. Ici, les fonctionnalités d'enregistrement d'images et d'interfaces sont exploitées.

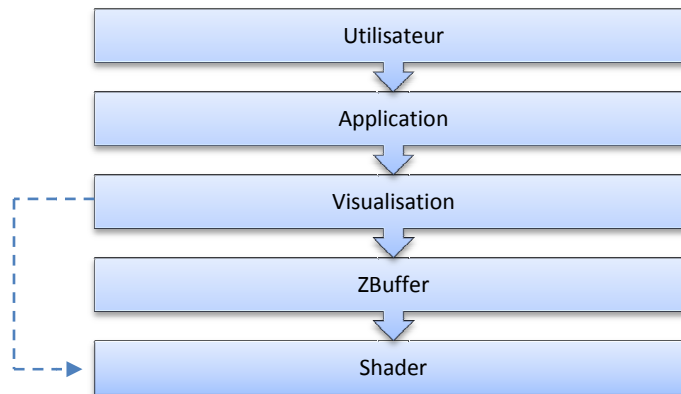
La seconde librairie tend à devenir un standard : Boost. Boost est connu pour ses fonctionnalités en programmation système, mais j'ai trouvé son application par la librairie d'algèbre linéaire « uBlas » que j'ai encapsulée dans des classes Matrix4x4 et Vector3 pour simplifier son utilisation.

### c. Références

- *Initiation à la synthèse d'images* de Pascal Mignot pour la plupart des algorithmes
- *Algorithme pour la synthèse d'image et l'animation 3D* de Rémy Malgouyres pour l'algorithme de remplissage
- *GPU Gem* de Randima Fernando pour les Shadow Map

## II. Organisation du pipeline

Comme il est courant en programmation, il est possible de décrire l'organisation du programme comme un modèle en couche.



En partant du haut, l'utilisateur demande à la classe Application de rendre à partir de fichiers PLY. L'application envoie alors la demande de créer un nouveau « Mesh » à partir d'un fichier PLY, la classe Visualisation lui renvoie un identifiant que la classe CApplication utilisera si elle souhaite modifier un matériel par exemple.

Au moment du rendu, la classe Application fournit une zone de rendu à Visualisation et demande de rendre un objet avec identifiant particulier. La classe Visualisation configure la classe ZBuffer qui allouera une zone pour le tampon de profondeur. La classe Zbuffer s'occupera d'appeler pour chaque vertex, un VertexShader précédemment fourni par la classe Visualisation et ensuite, après rasterisation et interpolation, pour chaque pixel, un PixelShader aussi fourni par la classe Visualisation.

Pour faire l'analogie avec une application désignée pour les jeux vidéo, la classe CApplication s'occupe des contrôles mais est aussi l'équivalent du « World » c'est-à-dire, l'ensemble des objets abstraits (car représentés par des identifiants) de la scène. La classe Visualisation utilise OpenGL ou DirectX. Le ZBuffer correspondrait alors à ce genre d'API. Les shaders sont fournis par la Visualisation et manipulés de manière transparente (pour faciliter la personnalisation) par le ZBuffer.

### III. Résultat

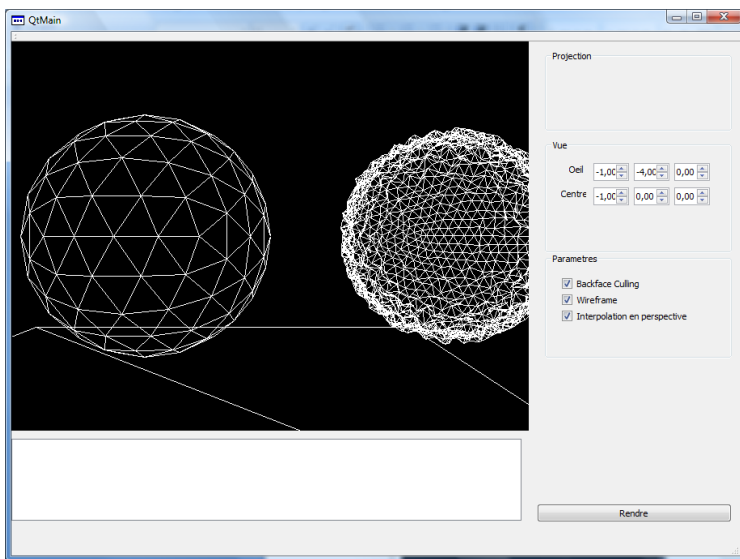


Figure 1 : Aperçu de l'interface

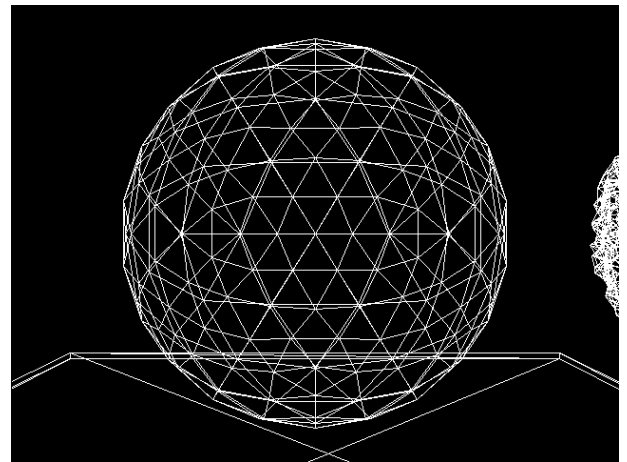


Figure 2 : Un rendu filaire simple

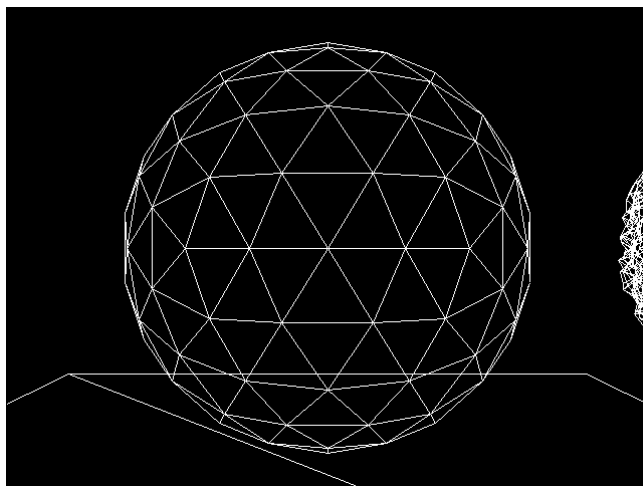


Figure 4 : Filaire avec rejet trivial des facettes arrière

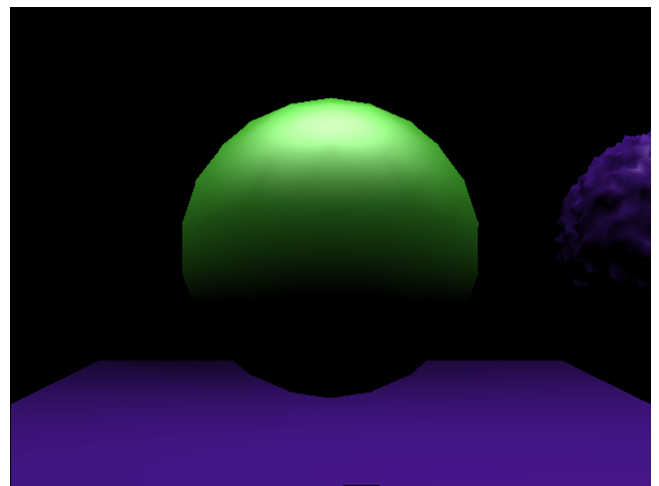


Figure 3 : Facettes remplies et éclairées

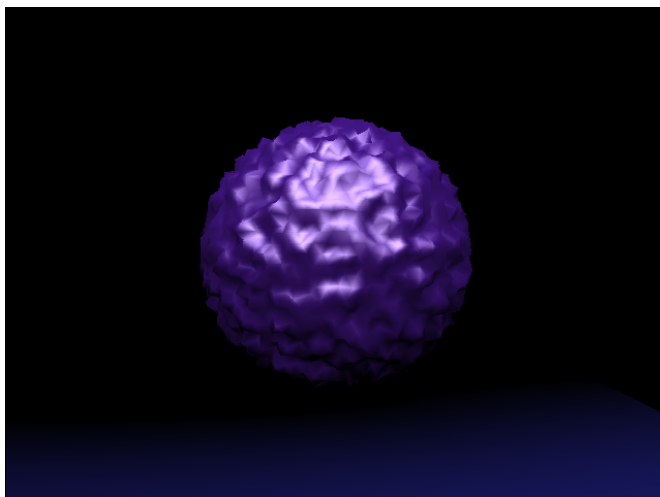


Figure 6: Autre objet, autre position de la lumière, autre matériel

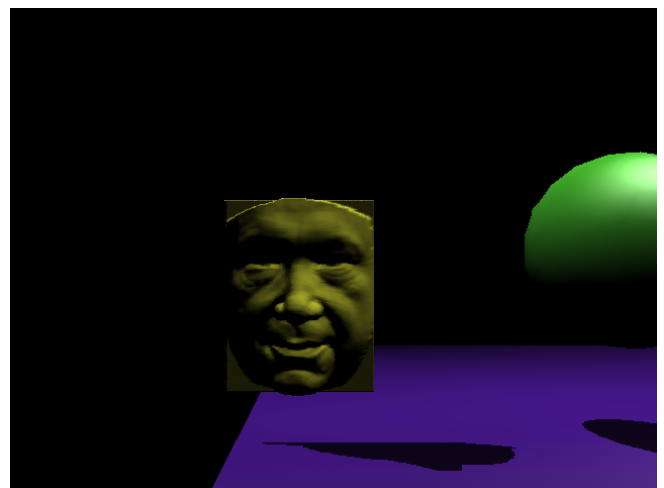


Figure 5 : Autre objet



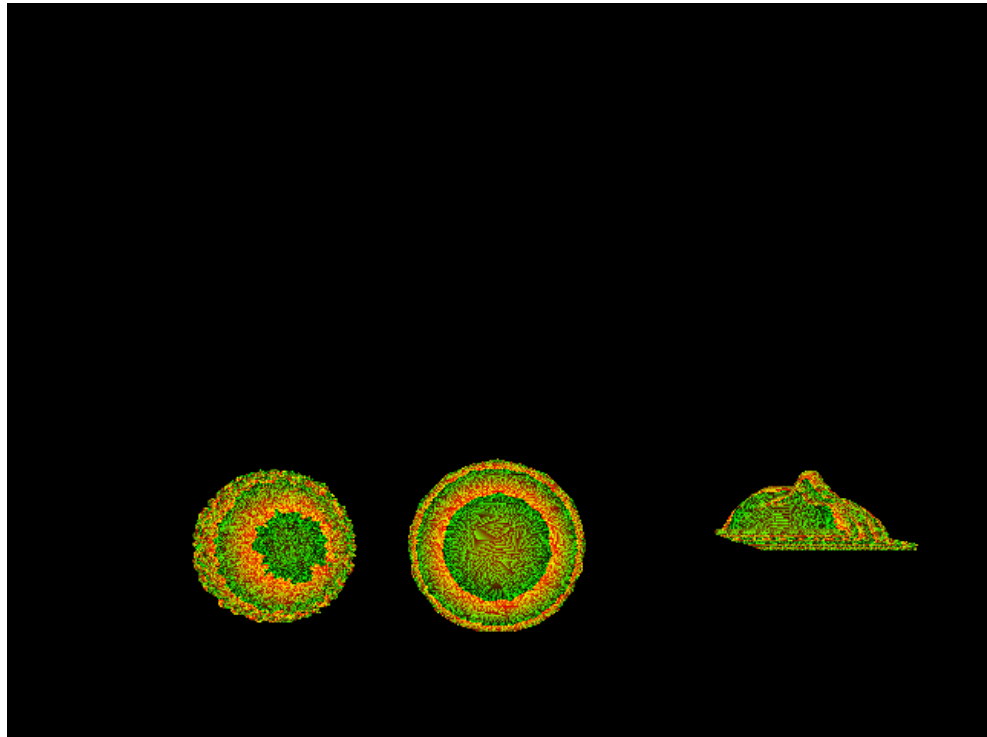


Figure 8: Tampon de profondeur rendu depuis la position de la lumière

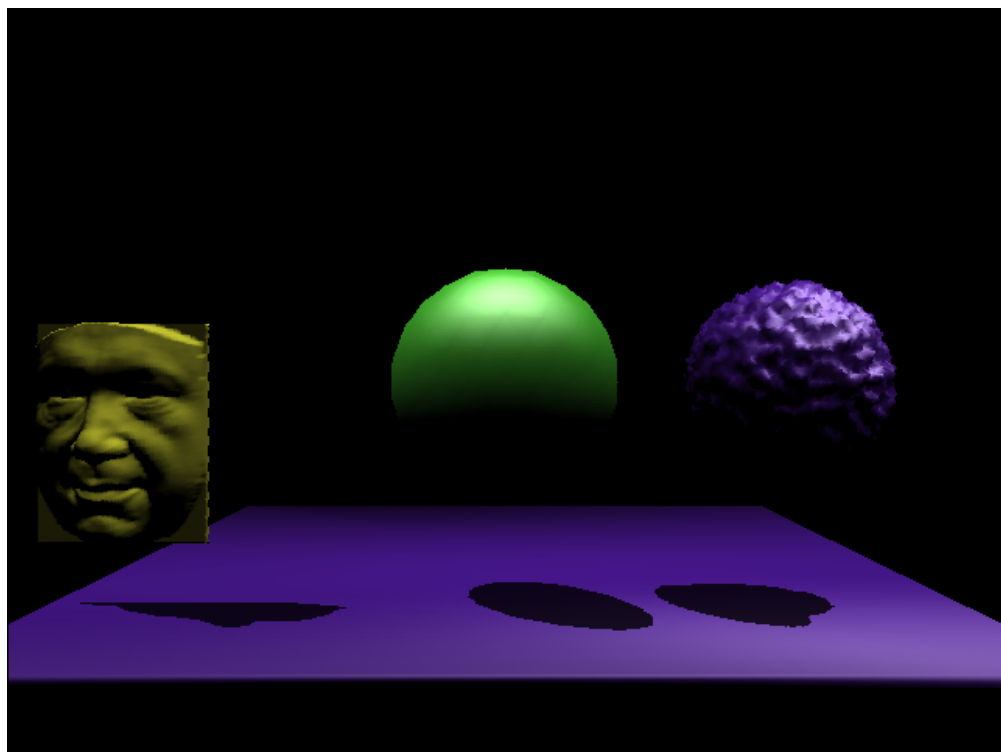


Figure 7: Rendu avec *tentative* de shadow map

## IV. Problèmes

### a. Normales

Les normales ne sont pas toujours correctes. Elles sont calculées pour chaque vertex en moyennant les faces voisines. Par ailleurs, elles ne subissent pas les transformations des objets, les rotations sont pourtant isolées mais le temps m'a manqué pour identifier le problème.

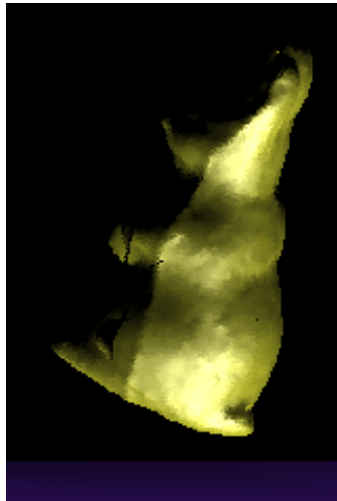


Figure 9 : Mise en évidence de problèmes lors du calcul de normales

### b. Camera

Une classe camera est clairement manquante, elle aurait été très utile. En effet, cette lacune ne m'a pas permis de mettre en place des shadow map robustes, le développement actuel ne facilite pas l'identification des problèmes de mise en perspective et de changement de repère. Ce ne fut alors qu'une tentative de rendre des ombres projetées, le résultat n'est pas convaincant mais le principe a été implémenté.

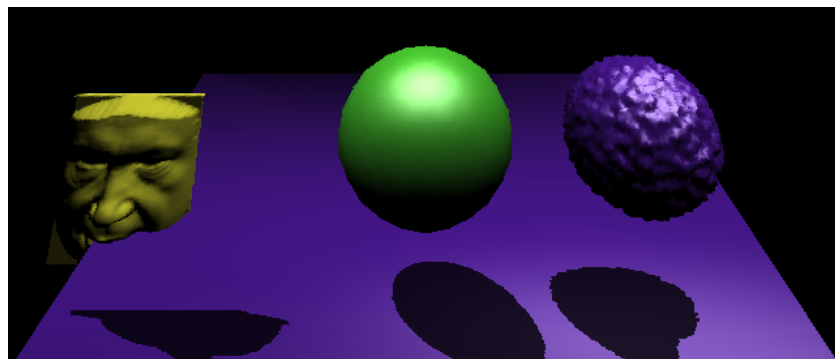


Figure 10 : Exemple de problème de projection ou caméra