Université de REIMS
Champagne-Ardenne
UFR de Sciences
Exactes et Naturelles

Rendu d'objets 3D – Création d'un Z-Buffer

Projet - Module Synthèse d'image

Boris Brugevin / Edouard Roge
Master I STIC - Informatique

Responsable : P. Mignot

Table des matières

Introduction

Dans le cadre de notre année de Master, nous sommes amenés à suivre des cours d'imagerie numérique. Durant ce premier semestre, nous avons particulièrement travaillé sur le traitement d'image* ainsi que la synthèse d'image*. Cette dernière consiste à créer une image 2D à partir d'une scène 3D par ordinateur. C'est ce que réalisent les API* graphiques telles que DirectX* ou OpenGL*. Les deux différents sujets proposés dans ce module étaient : création d'un Z-Buffer* ou création d'un jeu en OpenGL. Venant tous deux de section d'imagerie numérique (IUT Imagerie numérique de Reims & Arles), nous connaissons parfaitement OpenGL. Nous avons donc relevé le défi de créer notre propre API !

1 Description du sujet

Le but du projet est de réaliser un programme faisant des rendus de scènes 3D. Pour ce faire, il nous faut modéliser une structure de programme réalisant les différents traitements similaires à ceux que font nos cartes graphiques. Ce projet a été découpé en sous étapes, différents paliers qui sont les suivants :

- lecture et affichage d'un modèle en filaire sans face cachée
- remplissage de facettes avec une couleur constante et rejet trivial
- Z-Buffer
- ombrage constant avec le modèle de rendu de Phong*
- modèle d'ombrage de Gouraud*
- modèle d'ombrage de Phong
- calcul des ombres

Nous avons réalisé l'ensemble de ces étapes, et même un peu plus !

2 Réflexion & structure du programme

2.1 Réflexion

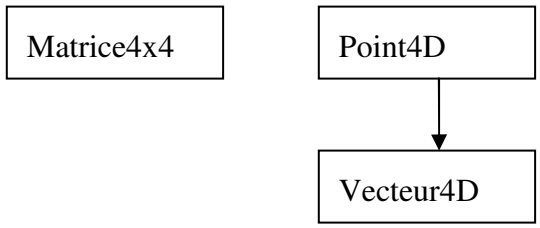
Avant de se lancer dans un tel projet, il nous faut réfléchir à ce qu'est la synthèse d'image, de ce que nous avons besoin, et de l'imbrication des traitements. Nous avons donc pensé à une structure de programme similaire à une API graphique, qui se gère sur les états.

Notre programme est écrit en C++ et est compatible Windows&Linux. Nous avons fait le choix de le faire en C++ pour avoir la possibilité d'utiliser la POO*. Celle-ci nous a facilité le travail grâce à l'héritage et à la surcharge des opérateurs.

2.2 Structure du programme

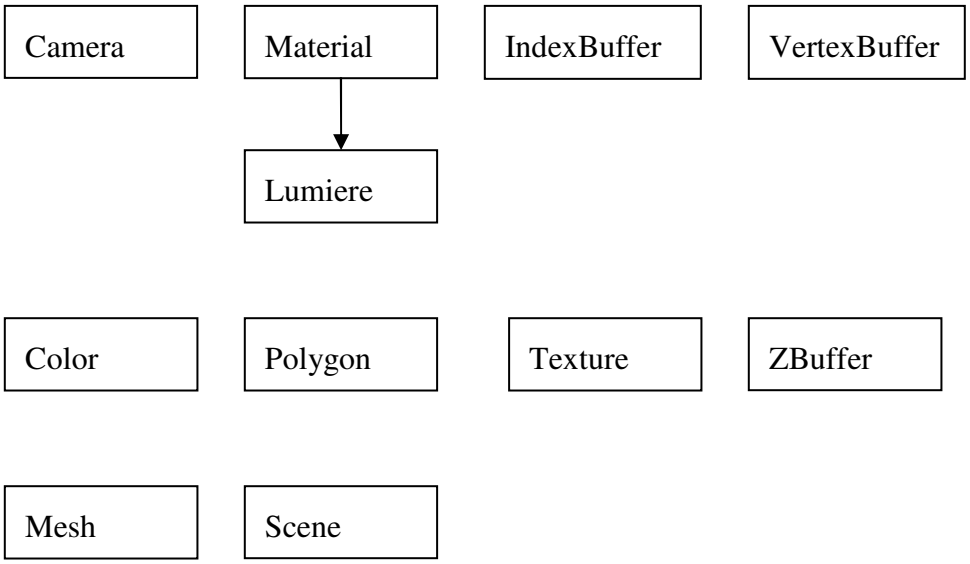
Tout d'abord, pour réaliser un moteur 3D, il nous faut la base : les mathématiques. Avant de commencer à programmer la synthèse, nous sommes donc passés par la rédaction de classes nécessaires telles que : Point4D, Vecteur4D et Matrice4x4. Grâce à celles-ci et aux différentes surcharges d'opérateurs (+, -, *, /, ^, %, [], …) nous avons allégé les lignes de codes qu'auraient été les classes de synthèse.

UML du « namespace » MATH3D :



Ensuite, nous nous sommes lancés dans la programmation du moteur en lui-même.

UML du « namespace » SYNTHESE :



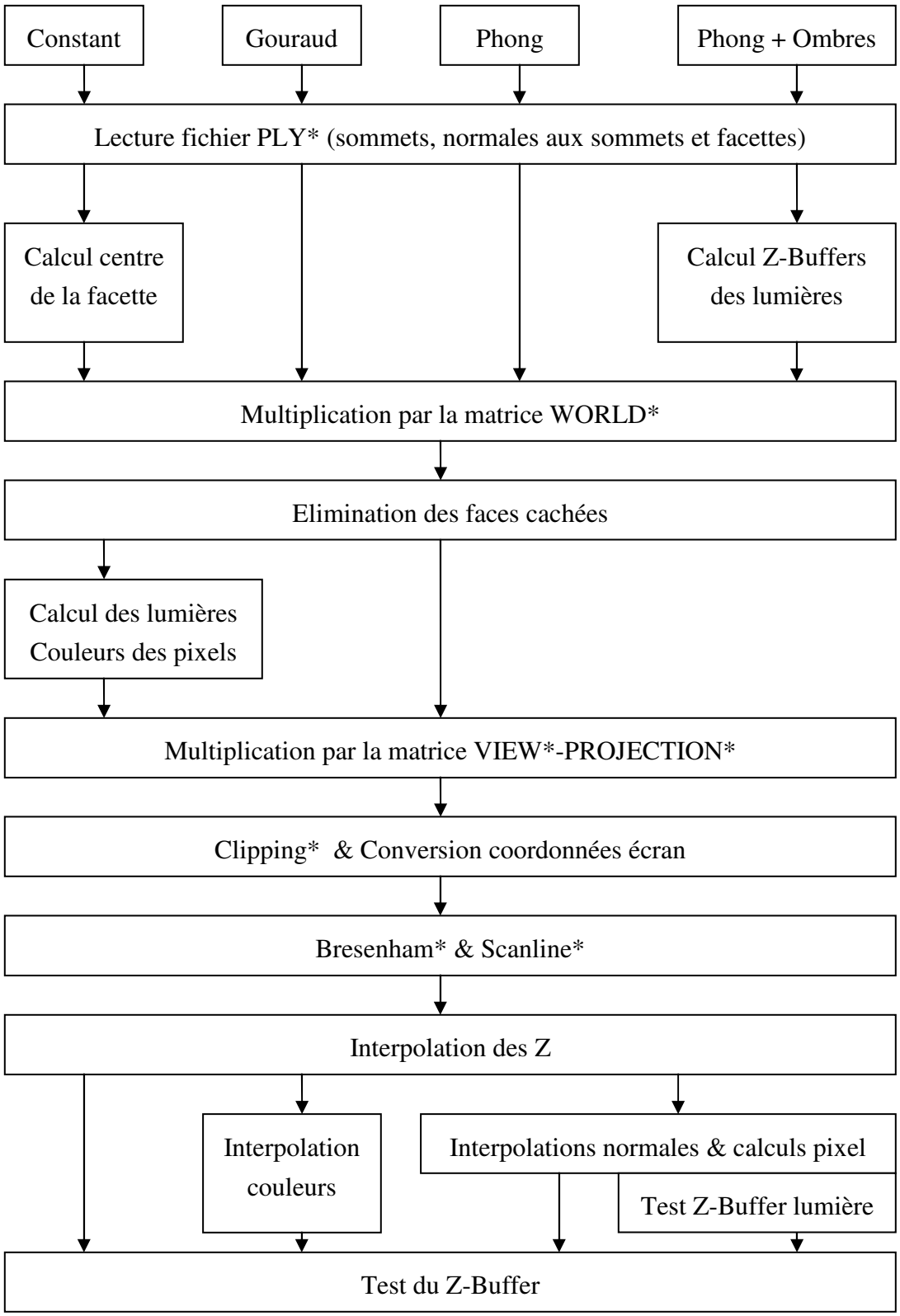
3 Pipeline de notre moteur

3.1 Imbrication des traitements

Pour chacun des rendus (Gouraud, Phong, …), les traitements ne sont pas les mêmes. Il faut donc suivre un ordre logique et ne pas calculer des choses qui n'ont rien à voir avec le model de rendu traité. Par exemple : pour Gouraud, l'interpolation des normales n'est pas souhaitable.

Les traitements réalisés sont :

- Ombrage constant
- Ombrage de Gouraud
- Ombrage de Phong
- Ombrage de Phong et gestion des ombres portées


Constant
Gouraud
Phong
Phong + Ombres
Lecture fichier PLY* (sommets, normales aux sommets et facettes)
Calcul centre de la facette
Calcul Z-Buffers des lumières
Multiplication par la matrice WORLD*
Elimination des faces cachées
Calcul des lumières Couleurs des pixels
Multiplication par la matrice VIEW*-PROJECTION*
Clipping* & Conversion coordonnées écran
Bresenham* & Scanline*
Interpolation des Z
Interpolation couleurs
Interpolations normales & calculs pixel
Test Z-Buffer lumière
Test du Z-Buffer

3.2 Explications des traitements

a) Lecture des fichiers PLY*

Cette partie du programme n'a pas été réalisée par nos propres moyens. Nous avons utilisé les fichiers C : readPLY.h et readPLY.c fournis sur la page web de Mr Mignot. Grâce à ces fonctions nous avons pu facilement lire un fichier PLY et ainsi récupérer les sommets, normales aux sommets et nous calculons les normales aux facettes des objets 3D. Nous n'avions plus qu'à stocker ces informations dans notre système de structures afin de pouvoir effectuer les traitements suivants.

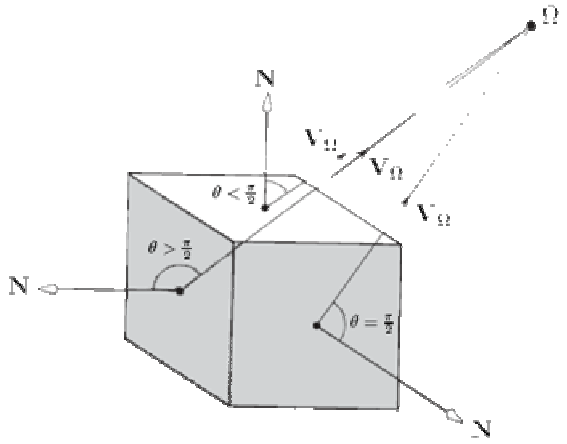
Exemple d'un fichier PLY (vert : sommets, bleu : facettes):

```
ply
format ascii 1.0
element vertex 4
property float32 x
property float32 y
property float32 z
element face 2
property list uint8 int32 vertex_indices
end_header
-1 -1 0
1 -1 0
-1 1 0
1 1 0
3 3 0 1
3 0 3 2
```

Ce type de fichier peut aussi contenir d'autres informations, telles que les normales, les couleurs aux sommets ou les coordonnées de textures.

b) Calcul du centre de la facette

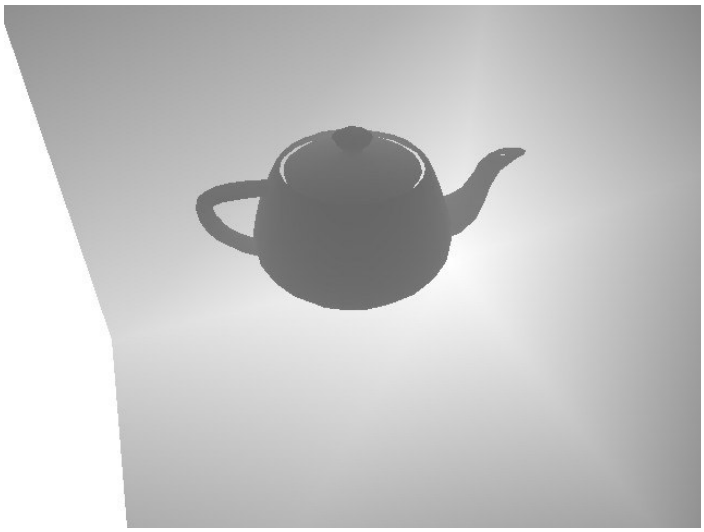
Ce calcul est nécessaire dans le cas du rendu constant. Il nous permet de calculer l'intensité lumineuse de Phong au centre de la facette.



c) Calcul des Z-Buffers des lumières

Nous aurons besoin des Z-Buffers des lumières dans le cas du calcul des ombres projetées par les objets 3D. En effet, grâce à ces textures de Z nous pourrons déterminer si un pixel est éclairé ou non par une source lumineuse.

Exemple de texture de Z :



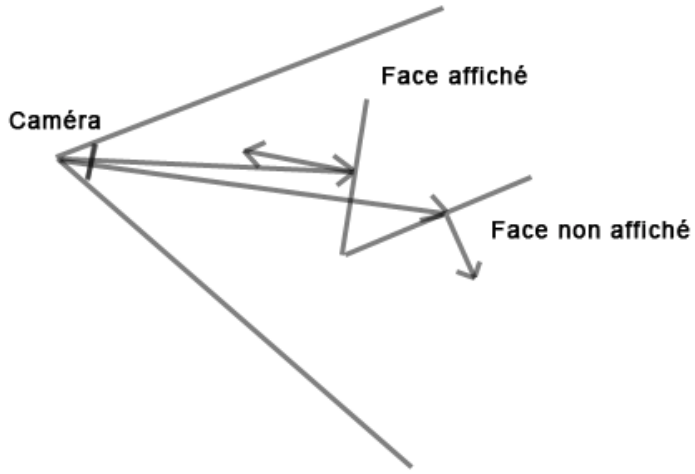
d) Multiplication par la matrice WORLD

Afin de placer correctement l'objet dans une scène, nous pouvons lui appliquer une matrice, la matrice : WORLD. Elle permet d'effectuer des rotations, mises à l'échelle et

des translations sur l'objet souhaité. Grâce à elle, nous pouvons déterminer les positions réelles des sommets ainsi que l'orientation des normales.

e) Elimination des faces cachées

Ce traitement porte aussi le nom de : rejet trivial. Il permet d'éliminer les faces qui ne seront pas visibles par l'observateur en fonction de la direction de la normale. Cela évite donc de calculer pour rien et de perdre du temps.



f) Calcul des lumières & couleurs des pixels

Dans notre programme nous effectuons les calculs des intensités lumineuses et des couleurs des sommets dans le repère global. Cela évite de transformer les lumières par la matrice de vue.

g) Multiplication par la matrice VIEW PROJECTION

Maintenant, il faut placer les objets dans le repère de l'observateur et calculer leurs projections sur un plan 2D. Après la division par W, nous obtiendrons des coordonnées x et y comprises entre [-1, 1] pour chaque sommet de l'objet.

h) Clipping & Conversion en coordonnées écran

Ici nous ramenons les coordonnées comprises entre [-1, 1] à la taille du Z-Buffer souhaitée. Puis nous effectuons un pseudo clipping pour éviter d'avoir à calculer ce qui sortira de l'écran (si tous les sommets sont en dehors de l'écran, on affiche pas la facette).

i) Bresenham & Scanline

A partir des sommets 2D nous pouvons maintenant tracer le contour des facettes avec la fonction de Bresenham et les remplir si on le souhaite avec la fonction de scanline. C'est aussi dans cette partie que toutes les interpolations ont lieu : Z, couleur, normales, …

Formule d'interpolation linéaire :

$$I = \lambda \, . \, I_{end} + (1 - \lambda) \, . \, I_{start} \quad \text{avec } \lambda \in [0.0, 1.0]$$

j) Interpolation de la couleur

Ce traitement a lieu pour l'ombrage de Gouraud. Il suffit de calculer l'interpolation de la couleur entre deux sommets.

k) Interpolation des normales

Ce traitement a lieu pour l'ombrage de Phong. Il faut calculer l'interpolation de la normale entre deux sommets et calculer l'intensité lumineuse en ce point interpolé. Ce type de rendu est beaucoup plus gourmand que Gouraud car les calculs d'intensités lumineuses sont assez lourds.

l) Test Z-Buffers des lumières

Ce traitement a lieu pour l'ombrage de Phong avec gestion des ombres portées. Avant d'effectuer le calcul de l'intensité lumineuse du pixel, nous regardons si le point est éclairé ou non par la source lumineuse grâce aux textures de Z calculées précédemment.

m) Z-Buffer de la scène

Enfin, nous pouvons effectuer le denier traitement de la scène : le test du Z-Buffer. Il consiste tout simplement à comparer la profondeur du Z pour un (x, y) donné afin de choisir la couleur en ce point.

4 Traitements supplémentaires

4.1 Atténuation de la lumière

Afin de gérer l'atténuation de la lumière nous avons pris exemple sur DirectX. Nous avons donc trois variables pour gérer ce phénomène et une formule à appliquer.

Formule de l'atténuation :

$$Attenuation = 1/(att0_i + att1_i * d + att2_i * d^2)$$

```
Paramètres :
att0i = Atténuation constant
att1i = Atténuation linéaire
att2i = Atténuation quadric
d = Distance du vertex à la position de la lumière
```

4.2 Gestion des couleurs et matériaux

Le rendu primaire du projet était juste composé des intensités lumineuses de chacun des pixels composant l'objet 3D. Nous avons donc voulu colorer un peu les scènes en gérant les couleurs. Nous pouvons donc définir une couleur en chaque sommet de l'objet ainsi qu'un matériel. Nous avons modifié la formule de Phong afin de pouvoir gérer ces nouveaux paramètres.

Formule de Phong avec gestion des matériaux :

$$Lr = Le + Ca.La + \sum(Ci.(Cd.(N.Li) + Cs.(Ri.V)^{ns})$$

Paramètres du matériel :
Le = luminance de l'objet si celui-ci est une source (vecteur rgb).
Ca = couleur ambiante de l'objet (vecteur rgb).
Cd = couleur diffuse de l'objet (vecteur rgb).
Cs = couleur du reflet spéculaire de l'objet (vecteur rgb).
ns = exposant spéculaire (réel).

Paramètres de la source lumineuse:
Ci = luminance de la $i^{ème}$ source (vecteur rgb).

4.3 Gestion de l'alpha

Notre application gère aussi la composante alpha. Nous pouvons donc définir un degré de transparence à nos objets. Ce traitement est effectué lors de l'écriture sur le Z-Buffer de la scène.

Formule du calcul de la transparence :

$$C = (1 - \alpha)*C1 + \alpha*C2$$

Paramètres :
C = couleur résultat
C1 & C2 = couleurs de départ
α = composante alpha de la couleur

*4.4 Ombrage « Cartoon » : Cell Shading**

Ce traitement est un autre modèle d'illumination qui donne un effet de dessin animé ou de BD. Il consiste à réduire la palette des couleurs ainsi qu'à marquer les contours de l'objet pour simuler les coups de crayon. Ce nouveau modèle est beaucoup utilisé dans le domaine du jeu vidéo. Nous avons donc codé très simplement ce modèle qui donne un très beau rendu.

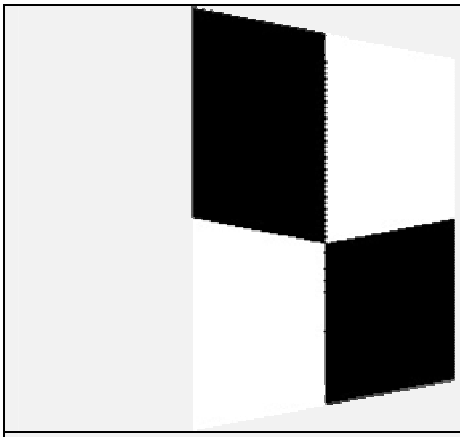
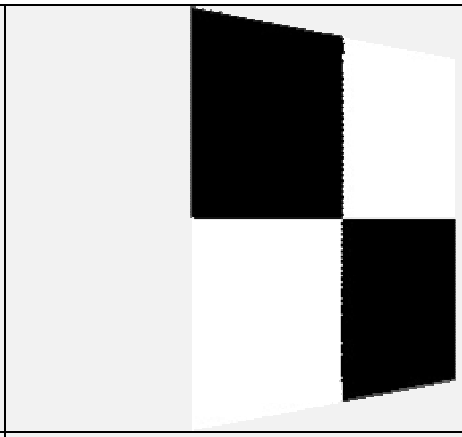
Extrait du code :

```
float4 L = lightPosition - VertexPosition;
float4 N = VertexNormal;
L.normalize();
float diffuseLight = max((N%L), 0.0f);

float4 V = cameraPosition - VertexPosition;
V.normalize();
float4 R = reflect(N,L);
float specularLight = pow(R.V, Shininess);
if((diffuseLight <= 0.0f)||(R.V<=0.0f)) specularLight = 0.0f;

float edge = max(0.0f, (V.N));

diffuseLight = (diffuseLight < 0.5f ? 0.3f : 0.7f);
specularLight = (specularLight < 0.7f ? 0.0f : 1.0f);
edge = (edge < 0.3f ? 0.0f : 1.0f);


float4 color = (Kd * diffuseLight + Ks * specularLight) * edge;
```

4.5 Application des textures avec correction de perspective

Le texturage d'un objet le rend plus réaliste. Nous nous sommes donc lancés dans ce traitement. L'application d'une texture n'est rien d'autre que l'interpolation des coordonnées u et v attribuées en chaque sommet. Mais dans le cas présent l'interpolation doit tenir compte de la perspective car le rendu en dépendra. Ce traitement ne fonctionne qu'avec le modèle d'illumination de Phong.

Exemple texturage :

	
Sans correction de perspective	Avec correction de perspective

La formule d'interpolation va tenir compte de la perspective, pour cela nous utilisons une formule en fonction de la projection des sommets (W).

Formule d'interpolation avec correction de perspective (sur une ligne) :

$$S = (\lambda * s1.W)/(s2.W + \lambda * (s1.W - s2.W))$$

variables :
λ : [0.0, 1.0]
s1 : sommet 1
s2 : sommet 2

5 Problèmes rencontrés

5.1 Scanline

Le scanline n'est pas un algorithme simple à réaliser. En effet, une seule erreur peut entraîner des rendus plus ou moins mauvais. De plus, trouver les erreurs est une tâche parfois difficile. Nous avons donc passé du temps pour réaliser un code correct.

5.2 Interpolation en perspective

Nous n'avions pas tout à fait compris la réalisation de cette interpolation lors du cours. Nous nous sommes donc beaucoup documentés sur internet pour comprendre le fonctionnement de cette interpolation.

5.3 Ombres portées

Les ombres portées par l'intermédiaire d'une comparaison de Z-Buffer ne sont pas difficiles à mettre en place, mais la précision des valeurs joue un rôle très important. Nous avons donc eu des problèmes de rendus à cause de ces imprécisions. Nous avons résolu le problème en fixant une marge d'erreur pour la comparaison.

6 Améliorations possibles

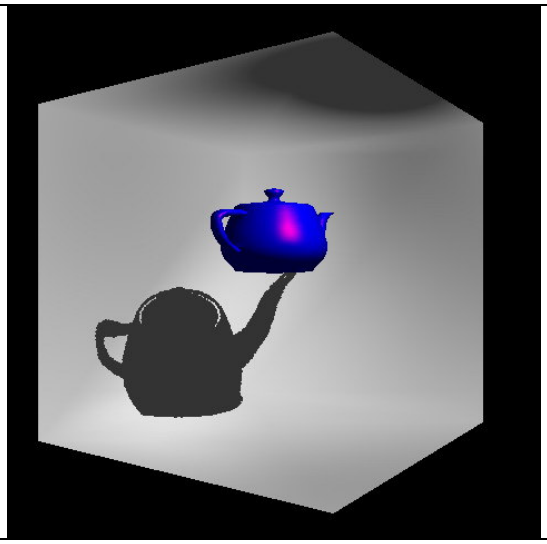
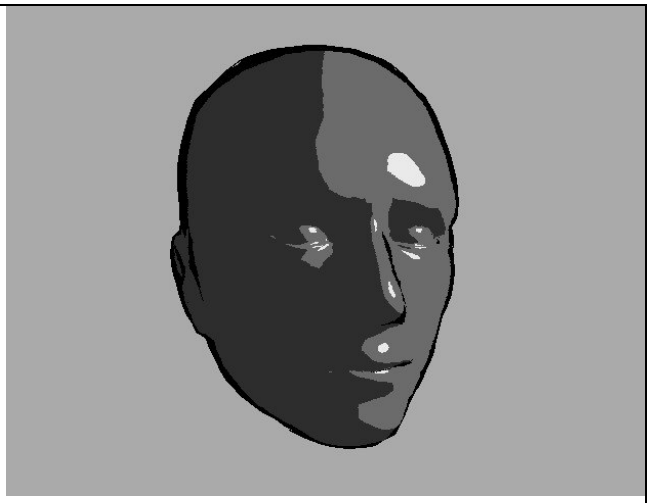
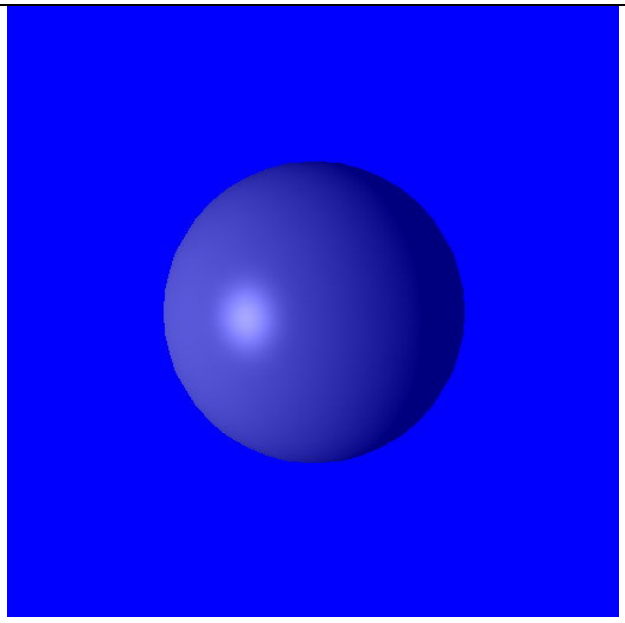
- gestion des ombres portées en gouraud
- plusieurs formats d'images pour créer des textures
- génération automatique des coordonnées de texture
- différents modes pour gérer le multi-texturage
- modes de filtrage

Conclusion

Ce projet a été très intéressant à réaliser. Nous avons découvert et codé certains traitements que réalisent nos cartes graphiques. Etant tous deux passionnés par l'imagerie numérique cela nous a beaucoup plu et beaucoup appris.

Annexe

Divers rendus de notre application

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>Rendu de Phong avec Ombres projetées et gestion des matériaux</td><td>Rendu Cell shading</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Rendu avec texture sans lumière</td><td>Rendu avec gestion de l'alpha</td></tr>
</table>

Bibliographie

Site internet :

http://www.gamedev.net/reference/articles/article667.asp
http://raphaello.univ-fcomte.fr/IG/Physique/Physique.htm
http://www.siggraph.org/education/materials/HyperGraph/illumin/specular_highlights/phong_model_specular_reflection.htm
http://en.wikipedia.org/wiki/Phong_reflection_model
http://www.supinfo-projects.com/fr/2006/remplissage_fr/4/
http://lab.erasme.org/3d/base.html
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/D3DXColorModulate.asp
http://www.mandragor.org/index
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/dx9_graphics_reference_d3dx_functions_math.asp
http://www.okino.com/conv/exp_ply.htm
http://membres.lycos.fr/heulin/3D/Plan3D.html

Cours de Mr Pascal Mignot :

http://helios.univ-reims.fr/UFR/Info/Image/PageBuilder.php?dir=Initiation&show=poly&menu=base&act=2

Glossaire

Traitement d'image : Le traitement d'images désigne en informatique l'ensemble des traitements automatisés qui permettent, à partir d'images numérisées, de produire d'autres images numériques ou d'en extraire de l'information.

Synthèse d'image : La synthèse d'images consiste en la création assistée par ordinateur d'images numériques. Ces images sont appelées images de synthèse.

API : Application programming interface, en informatique, c'est-à-dire une interface de programmation.

Bresenham: L'algorithme de tracé de segment de Bresenham est un algorithme développé par Bresenham en mai 1962.

Cell Shading: Le cell-shading (littéralement « ombrage de celluloïd »), également nommé toon-shading est un modèle d'éclairage non photo réaliste utilisé en synthèse d'image.

Clipping: Désigne le fait de borner le résultat d'un visuel.

DirectX : Microsoft DirectX est une suite d'API multimédia intégrée au système d'exploitation Windows permettant d'exploiter les capacités matérielles d'un ordinateur.

Gouraud : L'ombrage Gouraud (Gouraud shading en anglais) est une technique de rendu 3D inventée par Henri Gouraud.

OpenGL : (Open Graphics Library) est une spécification qui définit une API multi plate-forme pour la conception d'applications générant des images 3D.

Phong : L'illumination de Phong est un modèle local, c'est-à-dire qui calcule l'intensité en chaque point, et qui combine trois éléments - la lumière diffuse (modèle Lambertien),

la lumière spéculaire et la lumière ambiante.

PLY : The PLY file format is a simple object description that was designed as a convenient format for researchers who work with polygonal models.

POO : La programmation par objet (du terme anglo-saxon Object-Oriented Programming ou OOP), est un paradigme de programmation, il consiste en la définition et l'assemblage de briques logicielles appelées objets.

PROJECTION: Permet de fixer les caractéristiques optiques de la caméra de visualisation (type de projection, ouverture, ...).

Scanline: Algorithme de remplissage de polygones en effectuant une série de lignes verticales ou horizontales.

VIEW: Permet de fixer la position et l'orientation de la caméra de visualisation.

WORLD: Permet de créer la scène à afficher par création, placement et orientation des objets qui la composent.

Z-Buffer : En infographie, le Z-buffer ou tampon de profondeur est une méthode employée dans le cadre de l'affichage d'une scène 3D. Le Z-Buffer permet de gérer le problème de la visibilité qui consiste à déterminer quels éléments de la scène doivent être rendus, lesquels sont cachés par d'autres et dans quel ordre l'affichage des primitives doit se faire.