2015

Rapport de TER

Créer un shader – Modèle de lumière Cook-Torrance & Physically Based Rendering

13 Avril – 3 Juin

Bernard Yannick
L3 INFORMATIQUE
ANNÉE 2014-2015

Vanderhaeghe David
VORTEX / IRIT
UNIVERSITE PAUL SABATIER

Remerciements

Je tiens à remercier toutes les personnes qui m'ont apporté leur aide et qui m'ont généreusement fait partager leurs expériences tout au long de ce TER. Je remercie plus particulièrement :

- **Mon tuteur, David Vanderhaeghe :** pour son rôle d'encadrant et toute l'aide apporté à l'avancement de ce TER.
- **Les enseignants de la licence informatique :** pour leurs enseignements et leurs expériences qu'ils ont su nous faire partager.

Table des matières

I – Introduction :

De nombreuses recherches sont effectuées depuis plusieurs années dans le domaine de la 3D, pour des rendus de scènes toujours plus réalistes. La technologie ne cesse de s'améliorer. Aujourd'hui, nous produisons des rendus physiquement réalistes en temps réel. L'univers de la 3D interagit au gré de l'utilisateur, généralement un artiste, définissant plusieurs paramètres de matériaux et/ou de textures sur une forme 3D (fig. 1). Une des applications principales de ce genre de rendu temps réel est le jeu vidéo.

Pour ce TER, il m'a été donné comme référence le cours « Moving Frostbite to Physically Based Rendering » [1] co-écrit par Sébastien Lagarde et Charles de Rousiers en 2014, pour la SIGGRAPH.

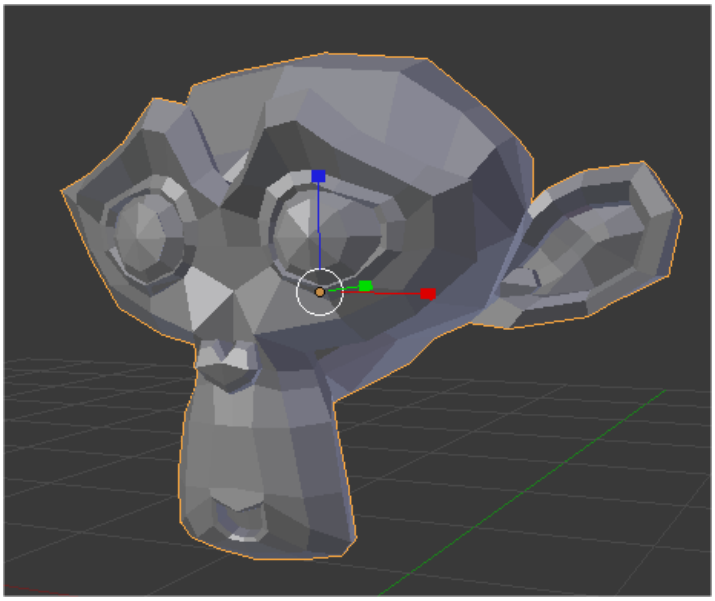

Figure 1- Exemple de modèle 3D

C'est dans le domaine de l'informatique appliquée à la 3D essentiellement, que reposent les recherches de l'équipe VORTEX de l'IRIT. En utilisant leur moteur de rendu 3D à temps réel, il m'a été proposé de créer un shader basé sur le cours cité précédemment [1].

Dans un premier temps, je vais présenter le contexte dans lequel le TER a été réalisé. Dans un second temps une présentation plus précise du sujet, mon organisation de travail tout au long de ce TER ainsi la manière dont j'ai abordé le problème.

II – Présentation du contexte :

1- L'IRIT :

L'Institut de Recherche en Informatique de Toulouse (IRIT) a été créé en 1990 en partenariat entre l'Université Toulouse 3 - Paul-Sabatier (UT3), le Centre national de la recherche scientifique (CNRS), l'Institut national polytechnique de Toulouse et l'Université des Sciences Sociales de Toulouse (UT1) devenue en 2009 Université Toulouse 1 Capitole.

L'IRIT en chiffres (en date du 26 Mai 2015) :

- 241 enseignants-chercheurs
- 27 chercheurs
- 44 ITA – ITAOS
- 220 doctorants
- 155 postdoc, invités, contractuels

L'équipe VORTEX[1] fait partie du thème de recherche : « Analyse et synthèse de l'information ». Elle a été créée en fin 2006.

Et plus particulièrement le groupe de recherche AGGA[2] mène des recherches sur le rendu temps réel, la synthèse d'images et la modélisation. [13]

Ces compétences permettent au groupe d'aborder les différentes problématiques liées aux mondes 3D et 2D.

2- Contexte :

A – Définitions :

Un moteur de rendu 3D à temps réel est un outil permettant de créer des images 2D à partir d'informations tridimensionnelles pour pouvoir les afficher sur un écran. Le terme « temps réel » signifie que chaque image est calculée et affichée immédiatement après la précédente. Il existe deux méthodes de rendu : le lancer de rayons et la rastérisation. [3]

Le pipeline de rendu 3D est une suite d'opérations réalisées sur une carte graphique pour effectuer un rendu 3D sur un écran 2D. Les shaders s'exécutent à certaines étapes de ce pipeline. [4]

Un shader est un programme, utilisé par la carte graphique, permettant le rendu de « scènes » essentiellement 3D. Il va permettre le calcul de la position à l'écran, des couleurs des pixels, etc. De par son utilisation par la carte graphique, il permet des calculs en parallèle sur celle-ci.

---

[1] **V**isual **O**bjects : from **R**eality **T**o **EX**pression

[2] **A**ppearance of **G**eometry and **G**eometry of **A**ppearance

Il est essentiellement composé de plusieurs fichiers (sous-programmes compris dans l'ensemble) : un fichier pour le calcul des vertices et un ou plusieurs fichier(s) pour le calcul des fragments. Il est parfois aussi composé d'un fichier, facultatif, de calcul de la géométrie. [7]

Une scène 3D est généralement créée dans un logiciel de modélisation 3D. C'est là que l'on va placer les éléments comme les formes géométriques (plus ou moins complexes), la ou les lumière(s) et la caméra. Les formes géométriques et la lumière contiennent divers paramètres. On définira la couleur diffuse et spéculaire pour la lumière, ainsi que son intensité pour une paramétrisation simple. Pour les formes géométriques, nous appliquerons des matériaux et/ou des textures.

Les matériaux des objets sont ceux qui le définissent. On peut créer un matériau suivant des paramètres de couleurs diffuses, spéculaires, rugosité, indice de réfraction, etc pour simuler du bois, ou du linoléum par exemple. Bien que l'on puisse aussi appliquer une texture à l'objet 3D pour en simuler l'apparence. Mais le rendu en sera légèrement différent. Avec la technologie actuelle, les artistes préféreront utilisés une paramétrisation d'un objet pour simuler un matériau pour un rendu physiquement réaliste. Ce sera justement le moteur de rendu 3D via le pipeline 3D, en chargeant la scène, qui s'occupera de ce rendu en appliquant des formules programmées dans les shaders.

B – Outils mis à disposition :

À ma disposition, il m'a été donné le cours « Moving Frostbite to Physically Based Rendering » [1] écrit par Sébastien Lagarde et Charles de Rousiers, en 2014 pour la SIGGRAPH. Mais aussi le « vortex renderer », le moteur de rendu 3D à temps réel de l'équipe VORTEX. Et aussi de nombreux liens internet dédiés à l'apprentissage du C++, de l'OpenGL et du GLSL.

Le moteur « vortex renderer » contenait un projet nommé « basic » avec ce qu'il faut de base pour charger une scène 3D en appliquant le shader Phonng.

III – Organisation du travail :

1- Mise en place des outils nécessaires :

Liste exhaustive des outils qui ont été utilisés pour développer ce TER :

- Ubuntu 14.04 LTS 64 bits :

Ubuntu est un système d'exploitation open source commandité par la société Canonical en 2004. Il est basé sur le système d'exploitation Debian mais se veut plus intuitif et plus simple d'utilisation.

- Qt Creator :

Qt Creator est un environnement multiplateforme faisant partie du framework Qt. De ce fait, il est fait pour être utilisé avec le C++. Il permet l'édition simple de fenêtre avec Qt et d'organiser son travail comme tout IDE. Il possède aussi tous les outils nécessaires au développement.

- OpenGL Shading Language :

OpenGL Shading Language, plus communément appelé GLSL, est un langage de programmation de shader. C'est avec lui que je vais créer le shader. Il a été développé par l'OpenGL Architecture Review Board afin de faciliter la création de shaders avec OpenGL.

- Langage C++ :

Le langage C++ est un langage de programmation compilé, développé par Bjarne Stroustrup au cours des années 80. Il a depuis fait office de nombreuses normes, la dernière datant de 2014. C'est l'un des langages le plus populaire de par sa variété de plateformes matérielles et de système d'exploitation.

2- Mise en place du planning prévisionnel :

Après une réunion de présentation du sujet et de l'article, trois parties distinctes sur le travail à effectuer ont été identifiées :

- Cook-Torrance avec un ou plusieurs spot light(s) :

Le but de cette première partie est de mettre en place un shader simple, calculant les composante diffuses et composante spéculaire d'un matériau. Pour cela, le shader reçoit en entrée les paramètres de chaque matériau. Il effectue le calcul de la composante diffuse et le calcul de la composante spéculaire, renvoie la couleur résultante de ces calculs. Et ce, suivant un ou plusieurs spot light(s) placé dans la scène.

- Cook-Torrance avec rendu dans une « skybox » de forme cubique :

L'objectif de cette deuxième partie est de charger une skybox dans l'API et d'effectuer un rendu suivant les lumières de la skybox et non plus des spot lights. Des pré-calculs sont effectués au niveau de l'API et stockés dans une texture 2D et 3D (ces textures contiennent les résultats calculés par le CPU). De ce fait, le shader reçoit ces deux textures en paramètres d'entrée et effectue le calcul de la composante diffuse et spéculaire suivant de nouveaux vecteurs de lumière et du niveau de mipmap.

- Cook-Torrance avec un/plusieurs area light(s) :

L'objectif de cette troisième partie est similaire à la première. Le changement est la source de lumière qui sera de forme géométrique : sphère, plan, cylindre, disque. Le fait que la source de lumière soit sous différentes formes demande des calculs supplémentaires pour définir la composante diffuse et composante spéculaire d'un matériau.

Nous disposions de sept semaines au total pour effectuer ce TER (13 Avril au 31 Mai). J'avais planifié mon temps selon mes estimations, de manière à pouvoir travailler sur les trois parties, sans oublier les documents intermédiaires et finaux à rendre.

➢ Ci-après, le planning prévisionnel :

Tâches	Semaine 1 Début : 13 Avril	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7 Fin : 3 Juin
1							
2							
3							
4		26/04					
5				10/05			
6						23/05 au	26/05
7							27/05
8							31/05
9							31/05
10							31/05
11						30/05 au	02/06
12							03/06

Tâches :

1) Cook-Torrance avec un/plusieurs spot light(s)
2) Cook-Torrance avec rendu dans une « skybox »
3) Cook-Torrance avec un/plusieurs area light(s)
4) Rendu 1ière partie du carnet de bord
5) Rendu 2ième partie du carnet de bord
6) Rédaction pré-rapport
7) Rendu pré-rapport
8) Rendu 3ième partie du carnet de bord
9) Rendu fiche PEC
10) Rendu résumé du TER
11) Rédaction rapport de TER final
12) Rendu rapport de TER final

IV – Présentation détaillée du travail :

En se servant du shader Phonng fourni dans le « vortex renderer », nous pouvons extraire le fichier de calcul de vertices. Ainsi que les fonctions permettant de récupérer la couleur diffuse et spéculaire d'un matériau et les lignes de calcul pour l'atténuation de la lumière.

Voici la table des différentes notations mathématiques (fig. 2) pour comprendre les équations :

$\boldsymbol{v}$	Vecteur vue
$\boldsymbol{l}$	Vecteur incident de la lumière
$\boldsymbol{n}$	Normale
$\boldsymbol{h}$	« half vector »
$\boldsymbol{f_d}$	Composante diffuse du BRDF
$\boldsymbol{f_r}$	Composante spéculaire du BRDF
$\boldsymbol{\alpha}$	« Roughness »
$\boldsymbol{\alpha_{lin}}$	« linearRoughness »
.	Produit scalaire
$\langle\ .\ \rangle$	Produit scalaire clampé entre 0 et 1
$\boldsymbol{\rho}$	Couleur diffuse

Figure 2 - Notations mathématiques

1- Cook-Torrance avec un ou plusieurs spot light(s) :

D'après le cours [1], pour calculer le BRDF, il faut calculer une composante spéculaire suivant Cook-Torrance [11] :

$$f_r(v) = \frac{F(v,h,f_0) \times G(v,l,h) \times D(h,\alpha)}{4\langle n.v\rangle\langle n.l\rangle}$$

Equation 1 – Composante spéculaire – Cook-Torrance

Il faut aussi la composante diffuse suivant Burt Burley [2] :

$$f_d(v) = \frac{\rho}{\pi}(1 + F_{D90}(1 - \langle n.l\rangle)^5)(1 + F_{D90}(1 - \langle n.v\rangle)^5)\ where\ F_{D90} = 0.5 + \cos(\theta_d)^2\alpha$$

Equation 2 – Composante diffuse sans conservation d'énergie – Burt Burley

A – Partie diffuse :

Pour la composante diffuse, les auteurs du cours [1] fournissent une explication sur la conservation d'énergie et un code fonctionnel pour la prendre en compte (fig. 3) :

```
89  vec3 getFr_Burley(float LdotH, float NdotL, float NdotV, float linearRoughness){
90      float energyBias = mix(0.0, 0.5f, linearRoughness);
91      float energyFactor = mix(1.f, (1.f / 1.51f), linearRoughness);
92      float fd90 = energyBias + (2.f * LdotH * LdotH * linearRoughness);
93      vec3 f0 = vec3(1.f);
94      float lightScatter = getFresnelSchlick(f0, fd90, NdotL).x;
95      float viewScatter = getFresnelSchlick(f0, fd90, NdotV).x;
96      return vec3(lightScatter * viewScatter * energyFactor);
97  }
```

Figure 3 - Calcul de la composante diffuse avec conservation d'énergie

Cette fonction fait intervenir un roughness élevé au carré (remapping). Il s'agit d'un choix artistique pour les artistes travaillant sur le Frostbite. (Page 14 du cours [1])

> *[…] we have chosen the "squaring" remapping which seems the most pleasing one for our artists. […]*

La méthode pour calculer le « linearRoughness » est la suivante :

$$\alpha_{lin} = \alpha^2$$

Equation 3 - Calcul du « linearRoughness »

Voici un comparatif entre trois rendus (fig. 4). Utilisation de l'équation de Burley (équation 2) sans conservation d'énergie en haut à gauche. Renormalisation pour la conservation d'énergie à droite (fig. 3). Et enfin, une simple implémentation du modèle lambertian en bas à gauche.

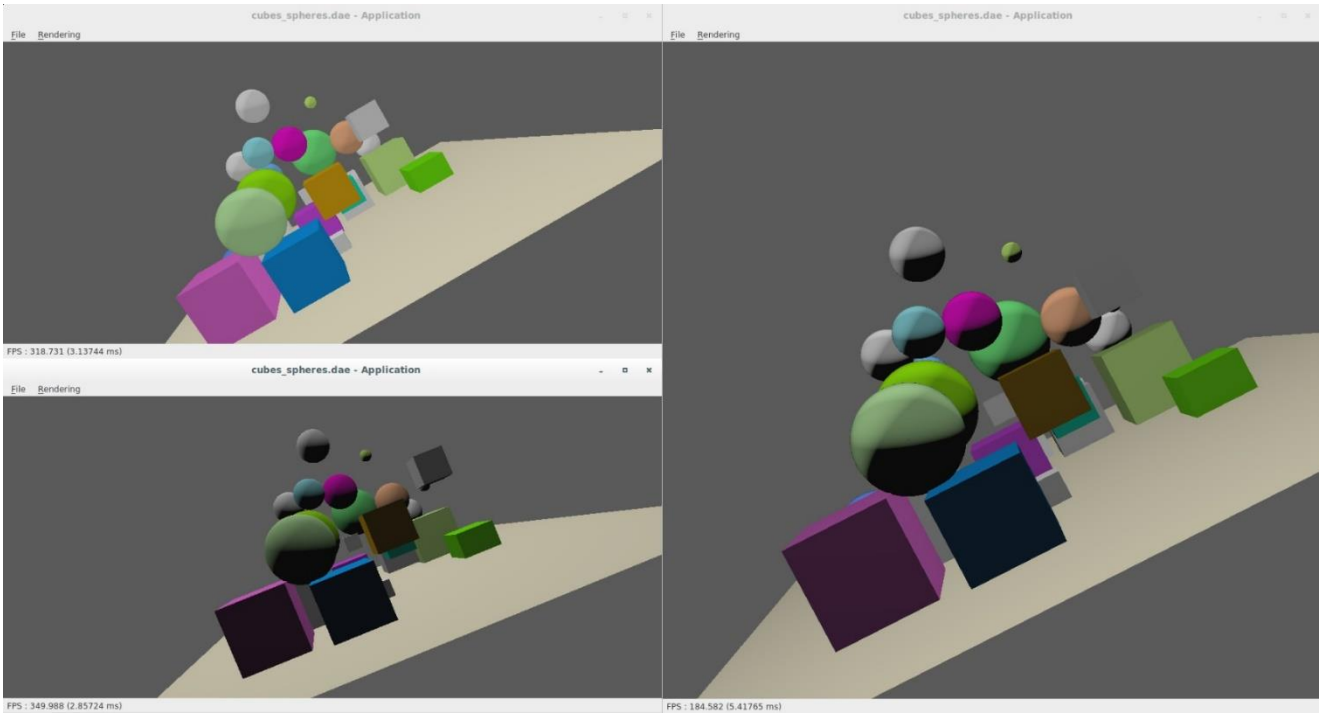

Figure 4 - Rendus Burley avec et sans conservation d'énergie et Lambert

B – Partie spéculaire :

Dans l'équation 1, on peut clairement identifier 3 composantes distinctes sur le nominateur :

- Fresnel : F
- Atténuation géométrique : G
- Fonction de distribution : D

Pour G et D, on utilise le modèle GGX : GGX est un modèle de micro-facettes utilisé pour les rendus de lumières sur les surfaces.

a. Calcul de Fresnel :

Pour Fresnel, l'article parle de l'approximation de Schlick mais avec un f0 remappé :

$$f_0 = 0.16 \times reflectance^2$$

Equation 4 - Calcul de f0 remappé

Ce remapping de f0 provient du modèle de matériau utilisé dans le Frostbite [1]. La réflectance varie entre 0 et 1. Cette valeur est donc remappée pour simuler plusieurs comportements :

- entre [0 ; 0.2] pour pouvoir simuler une micro occlusion faisant intervenir le terme f90.
- entre]0.2 ; 1] : remapper sur les matériaux diélectriques ayant un indice de réfraction entre 1.2 et 2.x

Et un terme f90 pour les micro-occlusions seulement si f0 est compris entre 0 et 0.2 :

$$f_{90} = \langle 50 \times (f_0 \,.\, 0.33) \rangle$$

Equation 5 - Calcul de f90

Sinon f90 vaut 1 et f0 vaut la couleur spéculaire. Ce qui nous donne comme code (fig. 5) :

```
166     if(IoR >= 1.2f && IoR < 3.f) {
167         f0 = vec3(0.16f * (ks * ks)); // Page 14
168         if(f0.x >= 0.0 && f0.x <= 0.2f) // For micro-occlusion
169             f90 = clamp(50.f * dot(f0, vec3(0.33)), 0.0, 1.f); // Page 79
170         else
171             f90 = 1.f;
172     } else {
173         f0 = ks;
174         f90 = 1.f;
175     }
176
177     vec3 FS = getFresnelSchlick(f0, f90, LdotH);
```

Figure 5 - Détermination de f0 et f90

Enfin, le calcul de Fresnel avec l'approximation de Schlick [8] (fig. 6) :

```
64  vec3 getFresnelSchlick(vec3 f0, float f90, float cosT) {
65      return f0 + (vec3(f90) - f0) * clamp(pow(1.f - cosT, 5.f), 0.f, 1.f);
66  }
```

Figure 6 - Calcul de l'approximation de Schlick pour Fresnel

Le clamp au niveau de (1 – cosT) à la puissance 5 permet de régler le problème si f0 est égal à 0, ce qui fait apparaître une tâche noire suivant l'angle où l'on observe la scène.

b. Calcul de G :

Le terme G joue un important rôle pour le rendu sur des valeurs élevées de roughness. Les auteurs [1] utilisent une fonction de Smith corrélée, basée sur la distribution GGX. Ils fournissent un code optimisé fonctionnel. Ce qui donne en GLSL (fig. 7) :

```
132 // Geometric attenuation
133 float V_SmithGGXCorrelated(float NdotL, float NdotV, float alphaG) {
134     float alphaG2 = alphaG * alphaG;

144     // This is the optimize version
145     // Caution : the "NdotL *" and "NdotV *" are explicitely inversed, this is not a mistake.
146     float Lambda_GGXV = NdotL * sqrt((-NdotV * alphaG2 + NdotV) * NdotV + alphaG2);
147     float Lambda_GGXL = NdotV * sqrt((-NdotL * alphaG2 + NdotL) * NdotL + alphaG2);
148
149     return 0.5f / (Lambda_GGXV + Lambda_GGXL);
150 }
```

Figure 7 - Calcul du terme G

c. Calcul de D :

Le terme D joue aussi un rôle pour le rendu des micro-facettes et par extension, l'apparence des surfaces. Aussi basé sur la distribution GGX, les auteurs [1] fournissent à nouveau un code fonctionnel. Ce qui donne en GLSL (fig. 8) :

```
152 float D_GGX(float NdotH, float m) {
153     // Divide by PI is apply later
154     float m2 = m * m;
155     float f = (NdotH * m2 - NdotH) * NdotH + 1;
156
157     return m2 / (f * f);
158 }
```

Figure 8 - Calcul du terme D

Pour le dénominateur, le chiffre 4 représente PI arrondi au supérieur de l'équation originelle de Cook-Torrance. NdotL sert à corriger des problèmes d'artefacts blancs que fait apparaître G et NdotV accentue la couleur.

Le code final de la composante spéculaire en utilisant Cook-Torrance est (fig. 9) :

```
vec3 cooktorrance(float NdotV, float NdotH, float LdotH, float NdotL, float roughness) {
    float f90;
    vec3 f0;
    vec3 ks = getKs();

    // After send an email to Charles De Rousiers
    if(IoR >= 1.2f && IoR < 3.f) {
        f0 = vec3(0.16f * (ks * ks)); // Page 14
        if(f0.x >= 0.0 && f0.x <= 0.2f) // For micro-occlusion
            f90 = clamp(50.f * dot(f0, vec3(0.33)), 0.0, 1.f); // Page 79
        else
            f90 = 1.f;
    } else {
        f0 = ks;
        f90 = 1.f;
    }

    vec3 FS = getFresnelSchlick(f0, f90, LdotH);
    float G = V_SmithGGXCorrelated(NdotL, NdotV, roughness);
    float D = D_GGX(NdotH, roughness);

    vec3 Fr = vec3((FS * G * D) / PI * NdotV * NdotL); // Page 13 - Equation (2) page 8

    return Fr;
}
```

Figure 9 - Fonction de calcul de la composante spéculaire

C – Paramètres des matériaux :

Pour calculer un matérial de façon physiquement réaliste, nous n'avons pas besoin du « shininess » fourni avec le shader Phonng. Par contre, nous avons besoin de ces paramètres :

- Couleur spéculaire
- Couleur diffuse
- Roughness
- Indice de réfraction

Sur le format de fichier « Collada », à l'exportation d'une scène 3D, on peut tout récupérer via Assimp sauf le roughness auquel il faudra affecter une valeur par défaut différente de 0. (fig. 10)

Il faut ajouter au moteur les fonctions nécessaires pour charger et modifier le roughness, l'indice de réfraction en étendant la classe Material (fig. 11) ainsi que les charger via Assimp (fig. 10) :

- assimploader.cpp :

```
320         if(AI_SUCCESS == tmpMat->Get(AI_MATKEY_REFRACTI, f)) {
321             mat->setRefractionIndex(f);
322         }
323
324         // Set default roughness
325         mat->setRoughness(0.1f);
```

Figure 10 - Modification du code dans assimploader.cpp

Dans la fonction : void AssimpLoader::buildInternalRepresentation(const std::string &name, const aiScene* scene) qui construit le graphe interne et charge les différentes données.

```
180     float mRefractionIndex;
181     float mRoughness;
```

Figure 11 - Modification du code dans material.h : Attributs

Ajout des deux attributs pour l'indice de réfraction et le roughness. Auquel on ajoute les setters / getters correspondants.

- renderloop.cpp :

```
70      shader->setUniform("IoR", mat->getRefractionIndex());
71      shader->setUniform("Nr", mat->getRoughness());
```

Figure 12 - Modification du code dans renderloop.cpp

Pour finir, le dernier fichier à modifier dans la fonction : void MaterialState::bind(Material *mat, ShaderProgram *shader) qui permet de binder la valeur de roughness et de l'indice de réfraction au shader. (fig. 12)

Afin de pouvoir les configurer une fois le programme lancé, il faut modifier le « Material editor », fourni avec le « vortex renderer », afin de rajouter le roughness et l'indice de réfraction. (fig. 13)

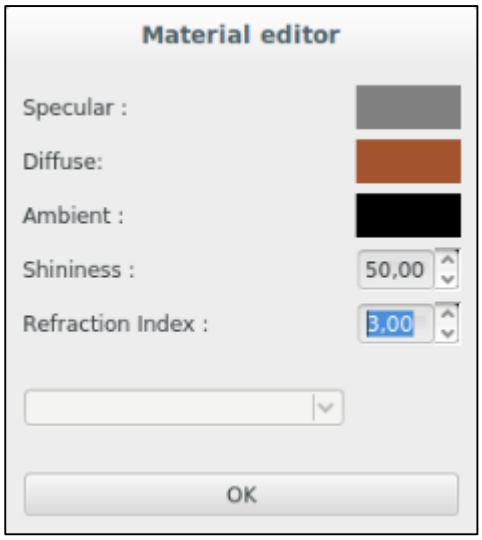

Figure 13 - Material editor modifié

D – Conclusion :

Pour conclure cette première partie de travail, déroulons un peu le shader :

En entrée, le shader récupère les paramètres suivants :

```
5   uniform vec3 uniLightSpecular;  /// light specular color
6   uniform vec3 uniLightDiffuse; /// light diffuse color
7   uniform float uniLightAngleOuterCone; /// spotlight cone size
8
9   uniform float uniLightAngleInnerCone; /// spotlight  inner cone size
10
11  uniform vec3 Ks; /// specular color
12  uniform vec3 Kd; /// diffuse color
13  uniform float Ns; /// shininess
14  uniform float IoR; /// index of refraction
15  uniform float Nr; /// roughness
16
17  /// Skybox
18  uniform int hasEnvMap;
19  uniform samplerCube uniEnvMap;
20
21  in vec3 varColor;
22  in vec3 varEyeVec;
23  in vec3 varNormal;
24  in vec3 varLightVec;
25  in vec3 varLightSpotDir;
26  in vec4 varTexCoord;
```

Figure 14 - Paramètres d'entrée du shader

Les paramètres écrits en dessous de Skybox seront expliqués dans la partie suivante. Le shader reçoit donc les vecteurs nécessaires à son fonctionnement : la normale, la caméra, la lumière. Ainsi que les différentes couleurs de la lumière et la taille du cône du spot.

De la ligne 11 à 15, il s'agit des paramètres du matériau. Il y a donc la couleur spéculaire, diffuse, le shininess (inutile pour ce shader), l'indice de réfraction et le roughness.

```
186 void main(void)
187 {
188     // Vectors
189     vec3 normal = getNormal();
190     vec3 eye = safeNormalize(varEyeVec);
191     float lightDist = length(varLightVec);
192     vec3 light = (varLightVec) / lightDist;
193
194     // half-vector
195     vec3 halfVec = normalize(eye + light);
196
197     // Angles
198     float NdotV = abs(dot(normal,eye)) + 1e-5f;
199     float NdotH = clamp(dot(normal, halfVec), 0.0, 1.f);
200     float LdotH = clamp(dot(light, halfVec), 0.0, 1.f);
201     float NdotL = clamp(dot(normal, light), 0.0, 1.f);
202
203     float roughness = getRoughness();
204
205     // Diffuse term
206     float linearRoughness = roughness * roughness; // [Bur12 - page 15] Page 14
207     vec3 Fd = getFr_Burley(LdotH, NdotL, NdotV, linearRoughness) / PI;
208     vec3 kd = getKd();
209
210     // Specular term - Cook-Torrance
211     vec3 Fr = cooktorrance(NdotV, NdotH, LdotH, NdotL, roughness);
212     vec3 ks = getKs();
213
214
215     // Lights
216     float lightAttenuation = 1.0;
217     lightAttenuation = 1.0 - smoothstep(uniLightAngleInnerCone, uniLightAngleInnerCone
218                                         + (uniLightAngleOuterCone-uniLightAngleInnerCone)/20.,
219                                         acos(dot(light, normalize(-varLightSpotDir.xyz))));
220     float spotLightCoef = lightAttenuation;
221     vec3 Ed = uniLightDiffuse*spotLightCoef;
222     vec3 Es = uniLightSpecular*spotLightCoef;
223
224     outColor = vec4(Fd*Ed*kd + Fr*Es*ks, 1.f);
```

Figure 15 - Fonction main() du shader

Ensuite, revenons à la fonction main() :

1) Tout d'abord, les vecteurs sont attribués
2) Le half-vector est ensuite calculé
3) La méthode getRoughness() permet de ne pas avoir un roughness à 0. Car sinon, le rendu ne fonctionne pas. Il est donc initialisé à 0.05 à la place.
4) Calcul de la composante diffuse du BRDF
5) Calcul de la composante spéculaire
6) Calcul de la couleur de la lumière en prenant en compte son atténuation sur les extrémités du cône du spot. (Fourni avec le shader Phonng)

7) Calcul de la couleur du fragment. Pour la partie diffuse, on multiplie chaque terme entre eux et de même pour la partie spéculaire. Enfin, on additionne les deux résultats. En n'oubliant pas de mettre la couleur alpha (transparence) à 1 car c'est un vec4 qui est retourné.

Pour finir, le shader fonctionne à peu près comme le montrent les images suivantes. Il est intéressant de noter qu'il est impératif de différencier les sphères par leur couleur diffuse pour un chargement sans bugs via Assimp. En effet, si les sphères ont la même couleur, quand on veut modifier un matériau, d'autres sphères changeront aussi. Toujours le même groupe où chaque membre sont choisis aléatoirement.

Mais un problème dans l'exécution du shader subsiste, l'ombre des objets est très mal gérée avec un roughness élevé. Les auteurs appellent ça : « the off-specular peak ». (fig. 16) [1]

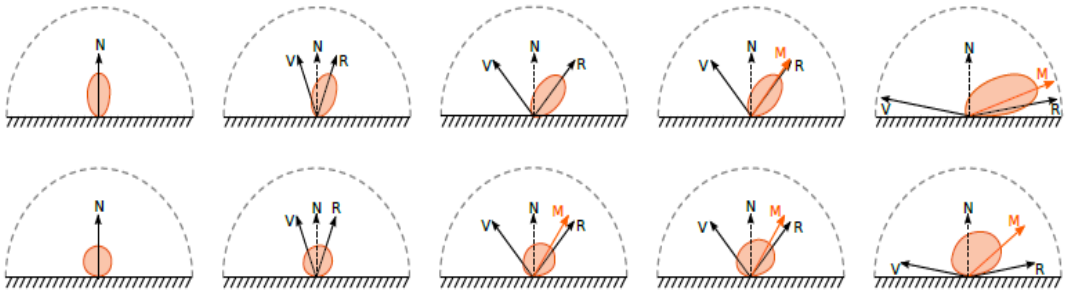

Figure 11: Example of BRDF lobe shapes for various view angles, where the dominant lobe direction is not aligned with the mirror direction R for grazing view angles, but instead with the direction M. Top row: $\alpha = 0.4$. Bottom row: $\alpha = 0.8$.

Figure 16 - Figure de l'article montrant le problème d'off-specular peak

Mise en évidence du problème d' « off-specular peak » avec 10 sphères vues de face (fig. 17) et vue de sur le côté (fig. 18):

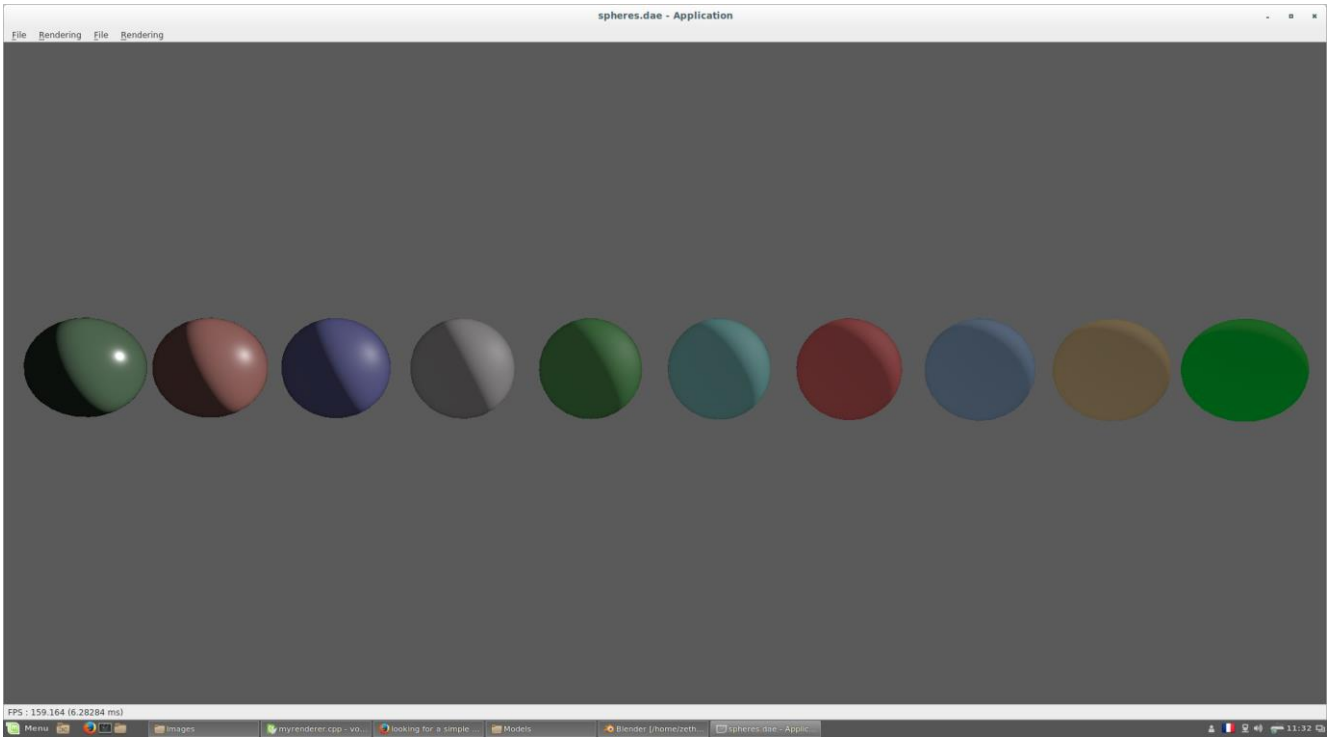

Figure 17 - Roughness de 0.1 à 1 de gauche à droite

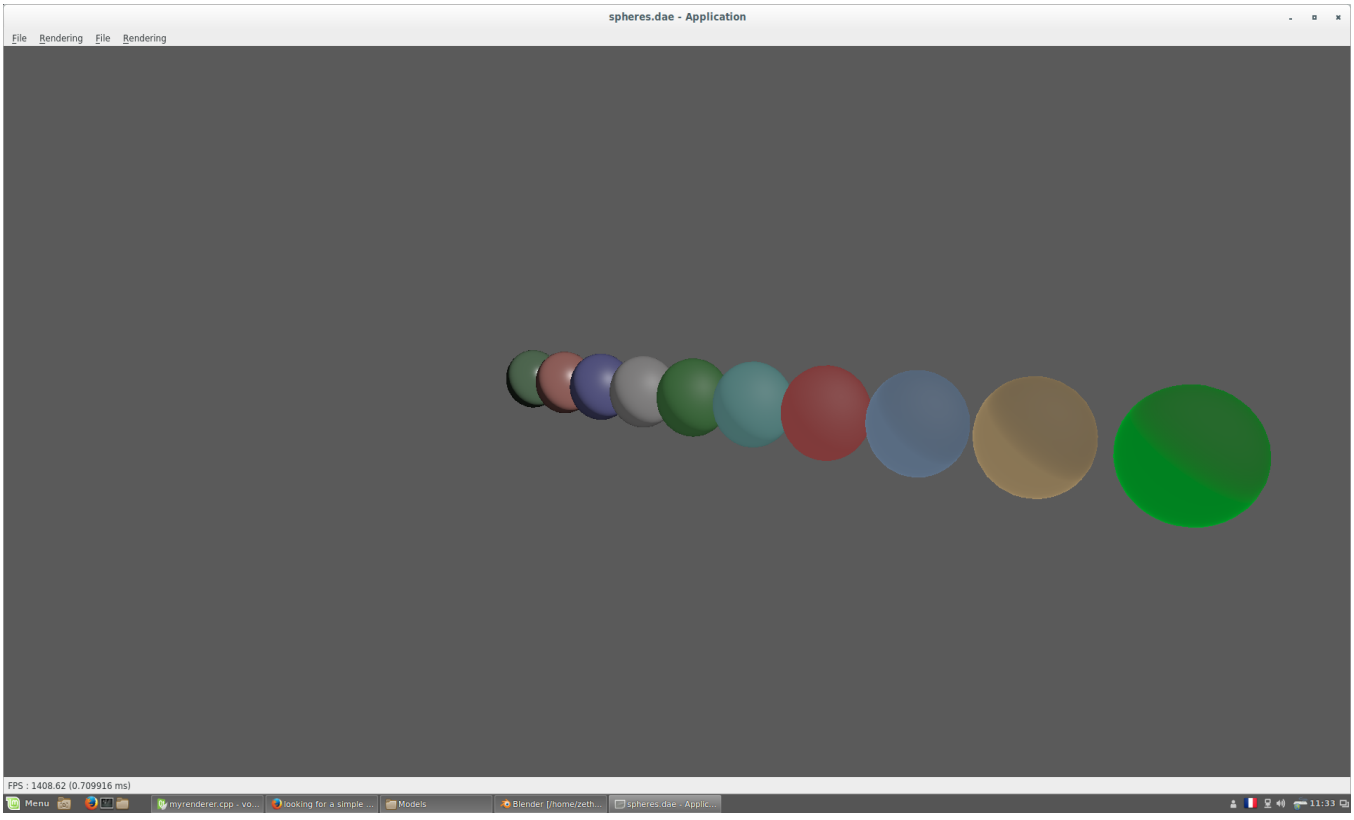

Figure 18 - Roughness de 0.1 à 1 du fond à devant

La différence entre la figure 17 et 18 est juste le point d'observation, comme nous pouvons le voir, à cause de « off-specular peak » avec un roughness élevé (>= 0.8 d'après les figures), le rendu de l'ombre sur la surface est à l'opposé de là où elle devait être.

Pour corriger ce problème, les auteurs se sont concentrés sur la correction durant un rendu dans une « Skybox », qu'ils appellent aussi Image Based Lights (IBL), section 4.9 du cours [1]. Et aussi, durant le rendu des area light(s), section 4.7 du cours [1].

2- Cook-Torrance avec rendu dans une « Skybox » de forme cubique :

Pour le second objectif fixé, la gestion d'une « Skybox » ou Image Based Lights (IBL), il devrait y avoir un rendu sans le « off-specular peak » car les auteurs développent la solution qu'ils ont apporté à la section 4.9 du cours [1].

Dans un premier temps, il faut modifier le moteur pour pouvoir charger une Skybox dans la scène. En utilisant des bouts de codes qui m'ont été fournis (fig. 19 ; fig. 20 ; fig. 21 ; fig. 22 ; fig. 23 ; fig. 24 ; fig. 25), il faut les ajoutés dans la classe MyRenderer :

```
36      mSkyBox = new vortex::SkyBox();
```

Figure 19 - Ajout dans le constructeur

```
77          if (mSkyBox->valid_){
78              lightParamameters.addParameter("uniEnvMap", mSkyBox->skyBoxTexture_);
79              lightParamameters.addParameter("hasEnvMap", 1);
80          } else
81              lightParamameters.addParameter("hasEnvMap", 0);
82          lightParamameters.addParameter("inverseViewMatrix", viewToWorldMatrix);
```

Figure 20 - Ajout dans la fonction lightPass dans le if et else

```
126     // draw skybox
127     if(drawSkyBox_ && mSkyBox->valid_){
128         glAssert( glDrawBuffers(1, bufs) );
129         drawSkyBox(mSkyShaderId, modelViewMatrix, projectionMatrix);
130     }
```

Figure 21 - Ajout dans la fonction RenderFilled()

```
285     // For skybox rendering
286     mSkyShaderId = assetManager->addShaderProgram("sky");
287     mSkyBox->setupTextures("../src/bernard/Skybox/galileo_cross.pfm", 0);
288     drawSkyBox  = true;
```

Figure 22 - Ajout dans la fonction initRessources

```
309 void MyRenderer::drawSkyBox(int shaderId, const glm::mat4x4 &modelViewMatrix, const glm::mat4x4 &projectionMatrix)
```

Figure 23 - Ajout dans le classe MyRenderer de cette fonction

Et avec ceci, un shader « sky » et deux bouts de code pour le shader :

```
17  /// Skybox
18  uniform int hasEnvMap;
19  uniform samplerCube uniEnvMap;
```

Figure 24 - Ajout des deux paramètres d'entrée au shader

```
vec3 envMap(vec3 r, float glossy){
    float baseLod = textureQueryLod(uniEnvMap, r).x;
    return textureLod(uniEnvMap, r, baseLod+glossy).xyz;
}
```

Figure 25 - Ajout de la fonction envMap au shader

Dans la fonction envMap(vec3 r, float glossy): (fig. 25)

- r : Il représente le vecteur réfléchi que nous voulons utiliser, c'est-à-dire, la direction dans laquelle nous lisons l'IBL.
- glossy : C'est un flottant servant à passer le niveau de mipmap que nous souhaitons passer.

Ceci permet de charger simplement une « skybox » avec la scène 3D mais on ne l'utilise pas encore pour calculer le rendu (fig. 26) :

Figure 26 - Chargement d'une scène avec un Skybox

Le type « Distant Light probe » est une skybox où les bords du cube sont considérés à une distance infinie. Il faut surcharger la classe SkyBox par défaut du moteur, et dans celle-ci, la fonction :

void SkyBox::setupTextures(std::string name, int type)

En suivant le cours [1] à la section 4.9, il faut rajouter les pré-calculs des termes LD et DFG, qui seront contenus respectivement dans une texture 3D et une texture 2D. Cela me permettra de calculer le terme glossy et aussi d'avoir des vecteurs corrigés à des roughness élevés.

V – Conclusion et bilan :

En conclusion de ce TER, je tiens à souligner que le choix du sujet était directement motivé par mon projet professionnel consistant à travailler sur des moteurs de rendu 3D à temps réel appliqué au jeu-vidéo, bien que je sois conscient de la difficulté que cela incombe pour arriver à cet objectif. Je reste quand même assez ouvert sur mon futur, sans trop m'éloigner de mon but. Je souhaitais aussi conforter mon choix de spécialité de Master l'année prochaine : Image et Multimédia à l'université Paul Sabatier.

Partant du planning prévisionnel, je n'ai pas effectué entièrement la deuxième partie, Cook-Torrance avec rendu dans une « skybox ». Et je n'ai pas du tout travaillé sur la troisième partie, Cook-Torrance avec un/plusieurs area light(s). Des difficultés ont été rencontrées :

- Manque de connaissances théoriques sur le sujet
- Apprentissage en autodidacte du GLSL, C++ (bien que possédant quelques notions) et de l'informatique 3D pour pallier au manque de connaissances.

D'un point de vue plus personnel, je souhaitais tester mes capacités sur un domaine inconnu, avec deux nouveaux langages et de nouvelles notions. À noter également que la matière AII3D de l'an passé m'a assez bien aidé à appréhender le sujet grâce aux quelques notions abordées.

Malgré une certaine aisance à l'écriture du code, j'ai rencontré des difficultés en mathématiques pour la compréhension du sujet et le débogage qui en découle. Je pense notamment à l'aide apportée pour la tâche noire alors qu'il suffisait de comprendre l'approximation de Schlick et l'erreur que pouvait apporter la fonction pow(). J'ai donc pu apprendre que décomposer un problème complexe sur une feuille de papier est souvent plus efficace. Je pense avoir été, au vu de mes connaissances actuelles, au bout de mes capacités sur ce projet et je souhaite continuer sur cette lancée.

C'était une expérience qui a été très enrichissante pour moi, dans laquelle j'ai beaucoup appris sur mes limites et mes capacités. J'ai aussi pu aborder un des multiples aspects du domaine de la 3D. Il est certain que cette expérience m'a conforté dans le choix de mon projet professionnel et dans le choix de la spécialité du master.

VI – Bibliographie :

1- Sebastien Lagarde – Charles de Rousiers : « Moving Frostbite to Physically Based Rendering », ACM SIGGRAPH 2014 Courses. SIGGRAPH '14 : http://www.frostbite.com/wp-content/uploads/2014/11/course_notes_moving_frostbite_to_pbr.pdf

2- Brent Burley. "Physically Based Shading at Disney". In: Physically Based Shading in Film and Game Production, ACM SIGGRAPH 2012 Courses. SIGGRAPH '12 : http://selfshadow.com/publications/s2012-shading-course/

3- Moteur 3D : http://fr.wikipedia.org/wiki/Moteur_3D

4- Pipeline 3D : http://fr.wikipedia.org/wiki/Pipeline_3D

5- Rastérisation : http://fr.wikipedia.org/wiki/Rast%C3%A9risation

6- Skybox (Dans la 3D) : http://fr.wikipedia.org/wiki/Skybox_%28jeu_vid%C3%A9o%29

7- Shader : http://fr.wikipedia.org/wiki/Shader#Structure_pour_le_calcul_en_temps_r.C3.A9el

8- Approximation de Schlick: http://en.wikipedia.org/wiki/Schlick%27s_approximation

9- Simon's Tech Blog : Microfacet BRDF : http://simonstechblog.blogspot.fr/2011/12/microfacet-brdf.html

10- Specular highlightning : http://en.wikipedia.org/wiki/Specular_highlight#Cook.E2.80.93Torrance_model

11- Robert L. Cook – Kenneth E. Torrance : "A reflectance model for computer graphics", 1982 : http://inst.eecs.berkeley.edu/~cs283/sp13/lectures/cookpaper.pdf

12- Ressources SkyBox : http://www.pauldebevec.com/Probes/

13- IRIT – VORTEX/AGGA : http://www.irit.fr/Rendering-and-Visualization

VII - Table des illustrations :