université de BORDEAUX

RAPPORT DE PROJET DE FIN D'ÉTUDES

IMPLÉMENTATION ET COMPARAISON DE MODÈLES D'APPARENCE DE FEUILLES

Étudiants : DUBOIS Gabriel, GUEVARA GARBAN Manuel, LACHIVER Loïc, MORISSET Nicolas

Encadrant : Pascal DESBARATS

Client : Pascal BARLA

Année universitaire : 2021 - 2022

Remerciements

Nous tenons à remercier Pascal BARLA pour son suivi tout au long de ce projet et pour les connaissances qu'il nous a transmises lors de ce projet de fin d'études,

Pierre BÉNARD pour sa disponibilité et la rapidité de ses réponses,

Pascal DESBARATS pour ses conseils sur l'organisation du projet.

Résumé

L'apparence des feuilles est importante dans des domaines tels que la synthèse d'image (jeux vidéo et effets spéciaux) ou l'écologie (télédétection de canopées, photo morphogenèse, éthologie).

Une feuille pouvant être apparentée à une surface mince, ses propriétés de réflexion et de transmission peuvent être caractérisées par la Bidirectional scattering distribution function (BSDF) [1.2.2]. Nous avons implémenté des modèles de feuilles en nous basant sur la littérature existante, et nous nous sommes servis du logiciel BRDFExplorer, développé par Disney, pour les visualiser. BRDFExplorer étant conçu uniquement pour observer des BRDF (Bidirectional reflectance distribution function), nous avons dû étendre ses fonctionnalités pour permettre la visualisation de BSDF.

Nos modèles de feuilles sont écrits en GLSL, et les modifications apportées à BRDFExplorer sont en C++.

Mots-clés : Bidirectional scattering distribution function (BSDF), BTDF, modèles de feuilles, BRDFExplorer

Sommaire

1 Contexte

1.1 Introduction

Ce projet de fin d'étude est constitué de deux axes de travail majeurs : étendre le logiciel BRDF-Explorer afin de visualiser des BSDF et étudier des articles afin d'en tirer des modèles de BSDF et de les implémenter dans BRDFExplorer.

Dans ce rapport, nous traitons en premier lieu des besoins formulés par M. Pascal BARLA. Nous abordons ensuite comment nous nous sommes organisés pour réaliser ce projet. Les parties suivantes traitent du cœur technique du sujet à savoir l'architecture du logiciel, les modifications que nous avons apportées puis une discussion sur les résultats et les limites de notre travail.

1.2 Définition des termes

Nous utilisons, tout au long de ce rapport, certains termes techniques que nous définissons ci-dessous.

1.2.1 BRDF

Une BRDF (Bidirectional Reflectance Distribution Function) [7] est une fonction à 4 dimensions sous la forme :

$$f_r(\omega_i, \omega_r) : \Omega_N \times \Omega_N \Rightarrow \mathbb{R}_+ \tag{1}$$

Cette fonction [1], utilisée dans les études d'optique du monde réel ainsi qu'en synthèse d'image, sert à définir comment la lumière est réfléchie sur une surface opaque. Elle prend en paramètre ω_i, la direction de lumière incidente et ω_r, la direction de lumière réfléchie par rapport à la normale de la surface **n**. Ω représente l'ensemble de directions possibles dans un hémisphère.

Source : [6]

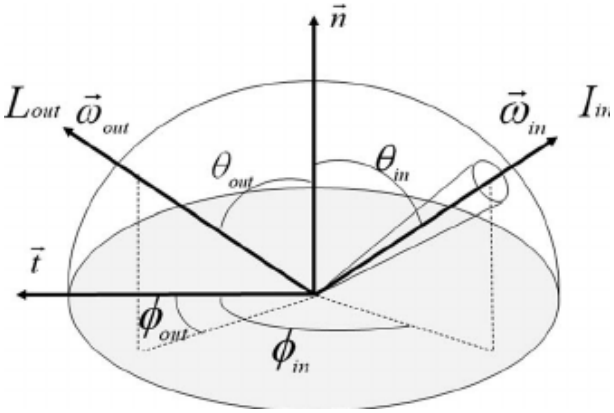


Figure 1: Schéma d'une BRDF

La fonction retourne le ratio de radiance réfléchie sur la direction ω_r. Comme on peut voir dans la figure 1, chaque direction est paramétrée par un angle azimutal ϕ et un angle zénithal θ. Une BRDF est mesurée en stéradian^{-1}.

Pour qu'une BRDF soit considérée physiquement réaliste [3] il faut qu'elle respecte les propriétés suivantes :

- Positive
$$f_r(\omega_i, \omega_r) \geq 0$$
- Respecter la réciprocité de Helmholtz, les directions ω_i et ω_r peuvent être inversées.
$$f_r(\omega_i, \omega_r) = f_r(\omega_r, \omega_i)$$
- Conservation de l'énergie, la puissance de lumière totale réfléchie doit être inférieure ou égale à la puissance de lumière incidente.
$$\forall \omega_i, \int_{\Omega_N} f_r(\omega_i, \omega_r) cos\theta_i d\omega_i \leq 1$$

1.2.2 BTDF et BSDF

Une **BTDF** (Bidirectional Transmittance Distribution Function) est similaire à la fonction décrite en 1.2.1, cependant une BTDF cherche à évaluer le ratio de radiance transmise au travers d'une surface.

$$f_t(\omega_i, \omega_t) : \Omega_N \times \Omega_N \Rightarrow \mathbb{R}_+ \tag{2}$$

Avec ω_i, la direction de lumière incidente et ω_t, la direction de lumière transmise par rapport à la normale de la surface **n**.

Une **BSDF**, (Bidirectional scattering distribution function) [2] représente une généralisation des termes BRDF et BTDF, où on cherche à évaluer la contribution de la **BRDF** et de la **BTDF** en même temps.

$$f_s(\omega_i, \omega_r, \omega_t) = f_r(\omega_i, \omega_r) + f_t(\omega_i, \omega_t) \tag{3}$$

Source : https://upload.wikimedia.org/wikipedia/commons/d/d8/BSDF05_800.png

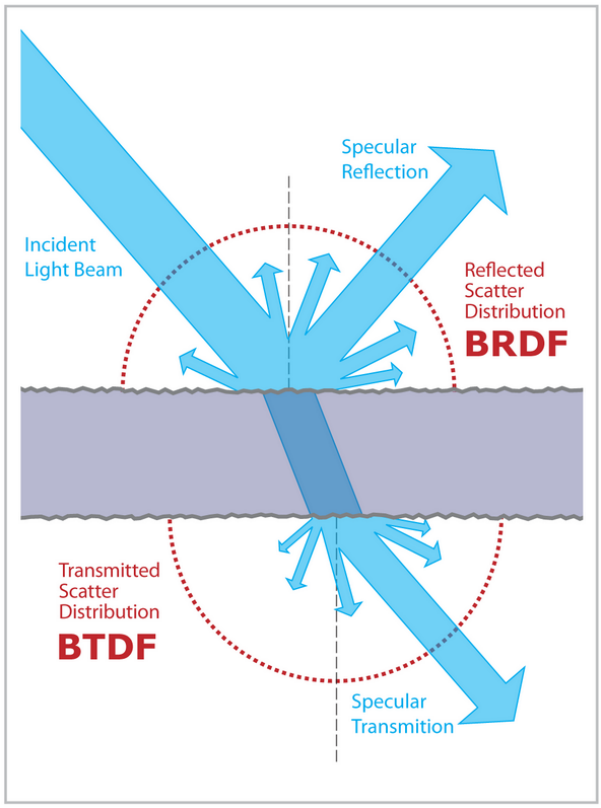


Figure 2: Schéma d'une BSDF

Comme on peut voir dans la figure 2, une BSDF évalue la distribution des rayons réfléchis ainsi que les rayons transmis en fonction d'un rayon de lumière incident.

1.3 Analyse de l'existant

1.3.1 Le logiciel BRDFExplorer

BRDFExplorer est un logiciel développé initialement par Walt Disney Animation Studios. En 2012, lors d'une conférence SIGGRAPH, le studio de Disney annonce le lancement d'un ensemble d'outils permettant l'analyse, la visualisation et la création de BRDF : BRDFExplorer.
Développé d'abord pour la création de leurs films d'animation, le logiciel est passé open-source (licence MS-PL) après la conférence SIGGRAPH, ce qui pour l'époque était l'un des seuls logiciels proposant gratuitement de tels outils.
Plus récemment, le logiciel BRDFLab, développé à partir de l'article "BRDFLab: A general system for designing BRDFs" [4], est lancé en open-source en 2013 et propose des outils similaires.

L'équipe de l'INRIA a récupéré BRDFExplorer dès la sortie afin de l'utiliser en interne. Au fil des années, de nombreuses modifications et extensions ont été apportées.

BRDFExplorer contient deux types de vues : les graphiques ("Plot" dans l'application), qui permettent de visualiser en 2D ou en 3D les lobes des BRDF, et les rendus, qui permettent de visualiser

le comportement de la BRDF sur un objet 3D. Le logiciel propose également de visualiser, pour les graphiques seulement, plusieurs BRDF/BTDF simultanément, laissant à l'utilisateur la possibilité de les comparer. BRDFExplorer met en place un système de chargement de BRDF créées par l'utilisateur. L'utilisateur peut donc implémenter une BRDF de son choix et la tester directement dans le logiciel. Par défaut, l'utilisateur dispose de deux paramètres importants : les angles incidents *phi* et *theta* qui sont essentiels pour visualiser et faire varier les valeurs de la BRDF. Chaque vue dispose de paramètres spéciaux qui permettent une visualisation et une analyse plus poussé de la BRDF.

1.3.2 Les articles scientifiques

Une partie de notre projet fut d'implémenter des modèles d'apparence des feuilles existants et de les comparer. Nous avons notamment travaillé sur les articles "Microfacet Models for Refraction through Rough Surfaces" par Walter et al.[9] et "The Dual-microfacet Model for Capturing Thin Transparent Slabs" par Dai et al. [1], qui proposent chacun un modèle de BSDF basé sur la théorie des micro-facettes.

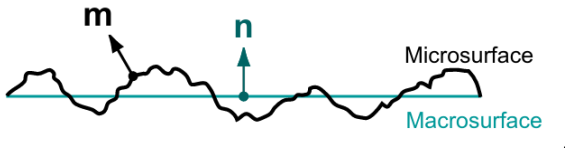


Figure 3: Schéma micro-facettes de l'article [9]

Afin de modéliser une surface rugueuse, Walter et al. en [9] considère les possibles perturbations d'une surface avec leurs normales. Sur la figure 3 **m** représente la normale de la micro-surface et **n** la normale de la macro-surface.

La BSDF sur l'article de Walter et al. [9] est présentée sous la forme:

$$f_s(\mathbf{i},\mathbf{o},\mathbf{m}) = f_r(\mathbf{i},\mathbf{o},\mathbf{m}) + f_t(\mathbf{i},\mathbf{o},\mathbf{m}) \tag{4}$$

Où **i** est le rayon de lumière incident, **o** est le rayon de lumière sortant et **m** est la normale de la micro-surface.

Ensuite le terme de réflexion f_r est défini :

$$f_r(\mathbf{i},\mathbf{o},\mathbf{m}) = \frac{F(\mathbf{i},\mathbf{h_r})G(\mathbf{i},\mathbf{o},\mathbf{h_r})D(\mathbf{h_r})}{4|\mathbf{i}\cdot\mathbf{n}||\mathbf{o}\cdot\mathbf{n}|} \tag{5}$$

avec

$$\vec{\mathbf{h_r}} = (\mathbf{i}+\mathbf{o})$$

Où F est le terme de Fresnel qui sert à définir la répartition des reflets aux angles rasants, D la fonction qui définit la distribution des micro-facettes est G appelée terme de masquage. Elle modélise le fait que certains rayons réfléchis par une micro-facette peuvent êtres cachés par d'autres micro-facettes, comme présenté dans la figure 4.

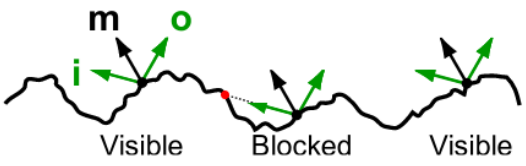


Figure 4: Schéma du terme de masquage de l'article [9]

D'autre part, la définition du terme f_t varie selon les articles. Pour nos expérimentations nous nous sommes basés sur la définition décrite sur [1] qui définit une surface rugueuse comme une fine couche avec deux interfaces de rugosités différentes, comme montré dans la figure 5.

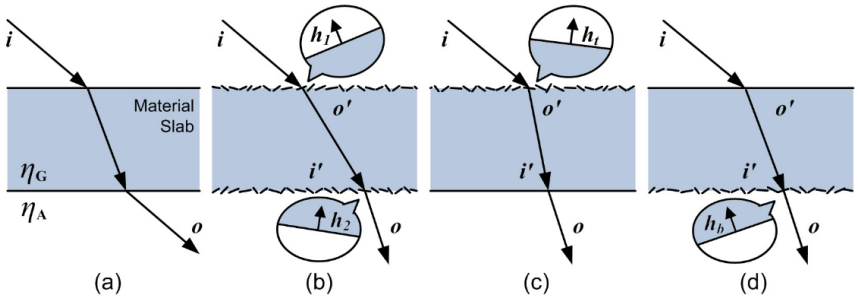


Figure 5: 4 types de surfaces présentées sur [1]

Comme montré dans la figure 6, dans le cas d'une feuille, la partie transmise sera essentiellement diffuse, vu que la structure interne de la feuille fait que le rayon incident d'origine est décomposé à l'intérieur du matériau.

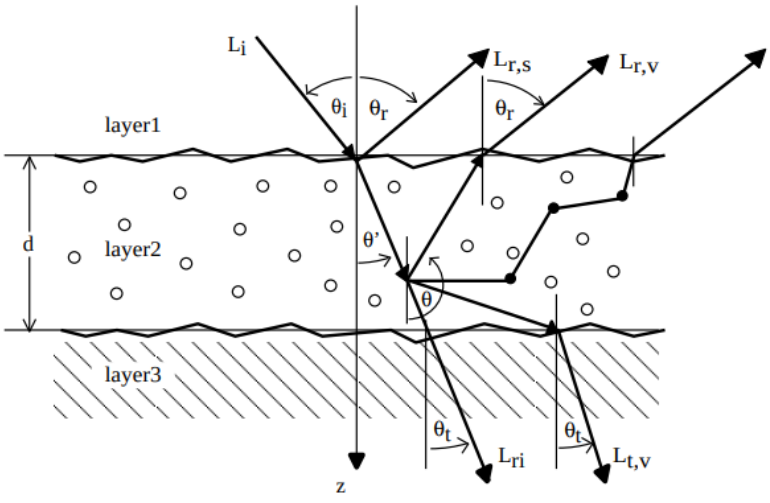


Figure 6: Schéma d'une BSDF d'un matériau multi-couches de l'article [5]

1.3.3 Autres définitions

Certains rendus présentés dans ce rapport portent le nom de rendu **IBL**, qui signifie Image-Based Lighting. Cette technique de rendu utilise une image comme source de lumière. Cette image est appelée ici "carte d'environnement".

2 Analyse des besoins

2.1 User stories

Voici, ci-dessous, la liste des Users Stories que nous avons pu dégager suite aux premières réunions avec notre client. Nous avons décidé de garder seulement les Users Stories dans leur version finale.

En tant qu'utilisateur je veux :

- Implémenter un shader au format .bsdf avec une fonction BTDF.
- Ouvrir avec BRDFExplorer un fichier au format .bsdf.
- Visualiser les paramètres associés à ma BSDF sur l'interface graphique de BRDFExplorer.
- Visualiser une BSDF dans les widgets de rendu.
- Ne visualiser que la BRDF dans les différents widgets du logiciel lorsque je charge une BRDF et non une BSDF.
- Avoir accès à l'implémentation de quelques BSDF afin de les ouvrir, de les visualiser et de les analyser au sein du logiciel BRDFExplorer.

2.2 Besoins fonctionnels

Dans cette partie, nous allons détailler l'ensemble des besoins fonctionnels implémentés dans le cadre de ce projet. Les termes techniques sont tous détaillés dans la partie 3 de ce rapport.

Nous utilisons ici un ordre de priorité afin d'indiquer si un besoin est plus important qu'un autre. Les priorités sont représentées par un ordre numérique s'étendant de 1 (essentiel) à 3 (moins important).

- 1 PolarPlot : afficher le cercle unitaire complet et tracer le lobe de transmittance sous le lobe de réflectance lorsqu'une BSDF est chargée.
- 1 LitSphere : Faire le rendu de la scène sur un hémisphère pour pouvoir visualiser la lumière transmise et avoir un bouton à cocher pour changer le sens des normales des faces.
- 1 LitObject : Faire le rendu de la scène en prenant en compte la BTDF en plus de la BRDF.
- 1 Implémenter un modèle de BSDF Lambertien.
- 2 3D Plot : Tracer le lobe de transmittance sous le lobe de réflectance lorsqu'une BSDF est chargée et modifier les mouvements de la scène pour pouvoir voir sous le plan.

- 2 Theta H/D/V: Tracer les courbes de BTDF et BRDF sur le même graphique avec une en pointillés pour pouvoir les différencier. Implémenter un bouton pour changer quelle fonction est affichée en pointillés.
- 2 Implémenter des modèles de BSDF plus complexes (Dai [1], Walter [9]).
- 3 ImageSlice : Ajouter un bouton pour changer le sens des normales des faces.
- 3 LobeSlice : Ajouter un bouton pour choisir si on veut rendre la BTDF ou la BRDF.
- 3 LitObject : Adapter la fonctionnalité d'importance sampling pour le calcul des BSDFs.
- 3 Implémenter des modèles de BSDF asymétriques (BRDF ou BTDF différentes d'un côté et de l'autre de la surface).

2.3 Besoins non fonctionnels

- L'extension de BRDFExplorer aux BSDF ne doit pas altérer le fonctionnement des BRDF existantes.
- Minimiser les modifications du code source de BRDFExplorer afin de faciliter l'intégration de nos fonctionnalités.
- Éviter les calculs superflus (appelle de fonction BTDF quand cette dernière ne doit pas être évaluée).

3 Architecture

Dans cette partie, nous présentons l'architecture de notre projet, avec nos modifications permettant la visualisation de BSDF.

3.1 Arborescence

```
+- BRDFExplorer
   +- src
   |  +- brdf
   |  +- shaderTemplates
   |  +- brdfs
   |  +- bsdfs
   |  +- cmake
   |  +- data
   |  +- probe
   |  ...
   ...
```

Figure 7: Arborescence de notre projet

En figure 7 est présentée l'arborescence de notre projet, qui est l'arborescence du logiciel BRDFExplorer.

Le dossier "src" contient le code source de BRDFExplorer (répertoire "brdf"), ainsi que d'autres répertoires nécessaires à son fonctionnement :

- "shaderTemplates" contient des modèles de shader 2D, 3D, IBL... utilisés pour faire les rendus.
- "brdfs" contient des modèles de BRDF sous forme de shaders modifiables par l'utilisateur et possèdent l'extension .brdf.
- "bsdfs" contient des modèles de BSDF sous forme de shaders modifiables par l'utilisateur et possèdent l'extension .bsdf. Nous détaillons ces shaders dans la partie 4.
- "data" contient les modèles 3D que l'utilisateur peut charger pour les rendus IBL.
- "probe" contient des cartes d'environnement utilisées pour les rendus IBL.

La visualisation de BSDF repose donc sur deux parties : le logiciel BRDFExplorer et les shaders ("shader templates" et shaders utilisateur contenant chacun une définition de BSDF).

3.2 Compilation

Pour compiler avec CMake sur Linux, exécuter cette commande depuis le dossier "BRDFExplorer" :

```
path/to/BRDFExplorer$ mkdir build && cd build && cmake -S ../src -B ./
```

Le dossier "build" contient maintenant l'exécutable "BRDFExplorer". **Attention :** il doit être lancé depuis le dossier "src". Le cas échéant, BRDFExplorer ne pourra pas accéder aux autres répertoires.

Pour lancer le logiciel, il faut donc exécuter cette commande :

```
path/to/BRDFExplorer/src$ ../build/BRDFExplorer
```

3.3 Le logiciel, BRDFExplorer

BRDFExplorer est codé intégralement en C++ et utilise la librairie Qt pour son interface graphique. Les shaders, détaillés dans la partie suivante de ce rapport, sont codés en GLSL, le langage de programmation de shader d'OpenGL.

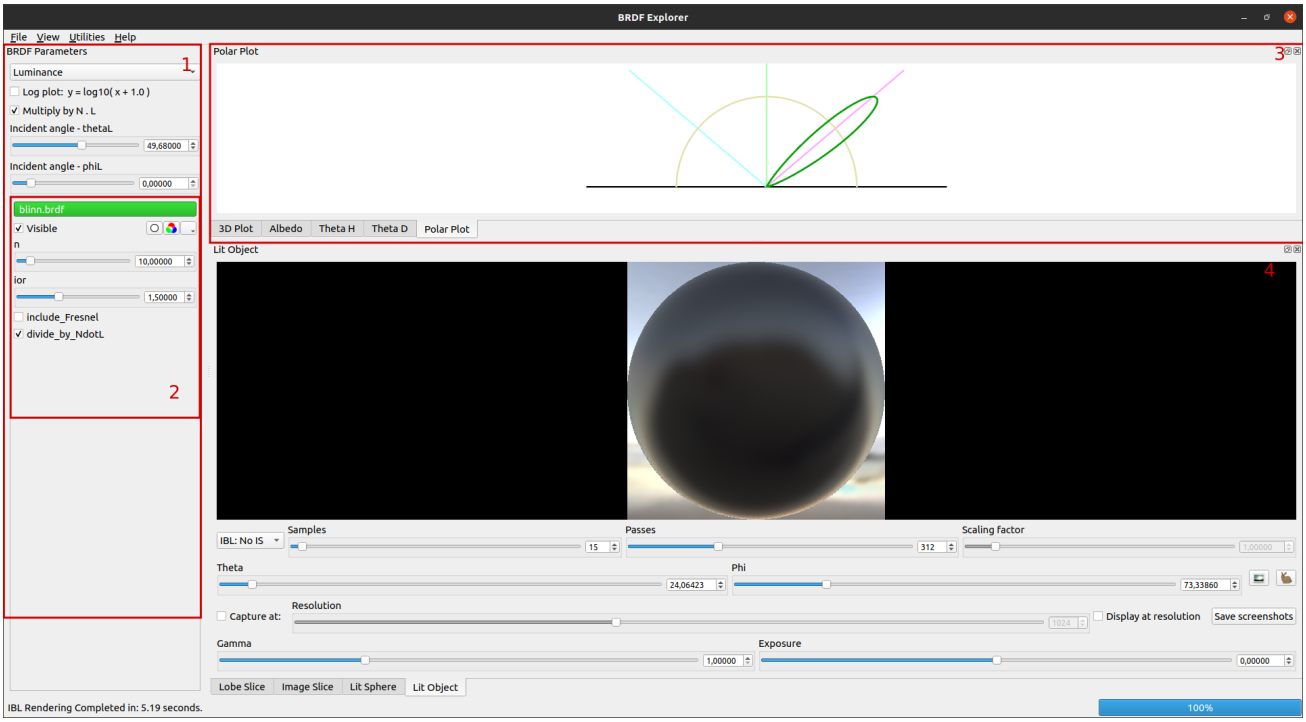


Figure 8: Capture d'écran du logiciel BRDFExplorer après avoir chargé une BRDF et présentation des classes QT. (1) : ParameterWindow, (2) : ParameterGroupWidget. (3) et (4) correspondent aux "Vues" (<Vue>Window contenant <Vue>Widget). (3) présente les "Plots" et (4) les rendus.

En figure 8 est présentée une capture d'écran du logiciel BRDFExplorer.

- MainWindow est la fenêtre mère qui va instancier les éléments principaux de l'application.

- ParameterWindow, le volet de gauche ((1) sur la figure 8), affiche les paramètres communs à toutes les BRDF/BSDF, par exemple l'orientation du vecteur incident de lumière. A chaque chargement de BRDF ou BSDF un Widget ParameterGroupWidget est créé et ajouté en tant qu'enfant de cette fenêtre.
- ParameterGroupWidget ((2) sur la figure 8) est la classe qui affiche les paramètres spécifiques à chaque BRDF/BSDF.
- Chaque vue permettant de visualiser les BRDF ((3) et (4) sur la figure 8) sont implémentées dans deux types de classes :
  - <Vue>Window contient les sliders et boutons contrôlant des paramètres spécifiques au rendu.
  - <Vue>Widget effectue le rendu en implémentant les fonctions InitializeGL() et PaintGL() de OpenGL.

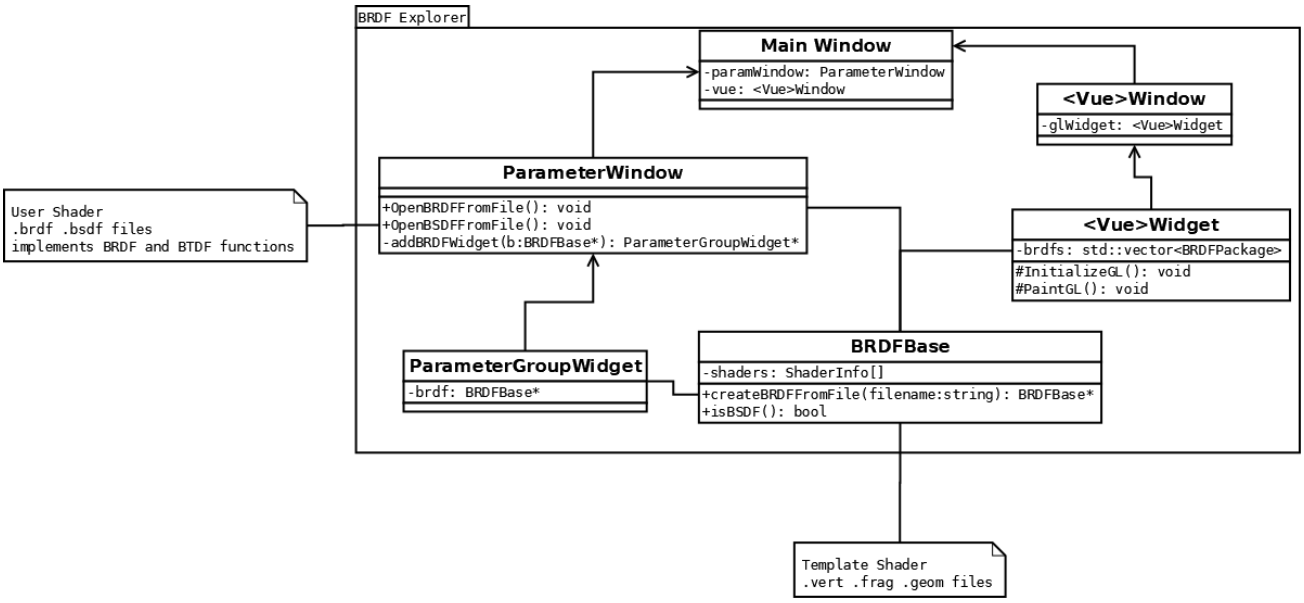


Figure 9: Architecture simplifiée du logiciel BRDF Explorer

On peut voir sur la figure 9 un diagramme UML simplifié de l'architecture de BRDFExplorer.

BRDFBase est la classe qui contient tous les shaders complets utilisés pour tous les types de rendus. Ainsi, chaque Widget contient une liste d'objets de type brdfPackage contenant chacun une BRDF-Base. Cette liste contient les BRDF et BSDF actuellement chargées (voir partie 4 de ce rapport pour les détails d'implémentation des BSDF).

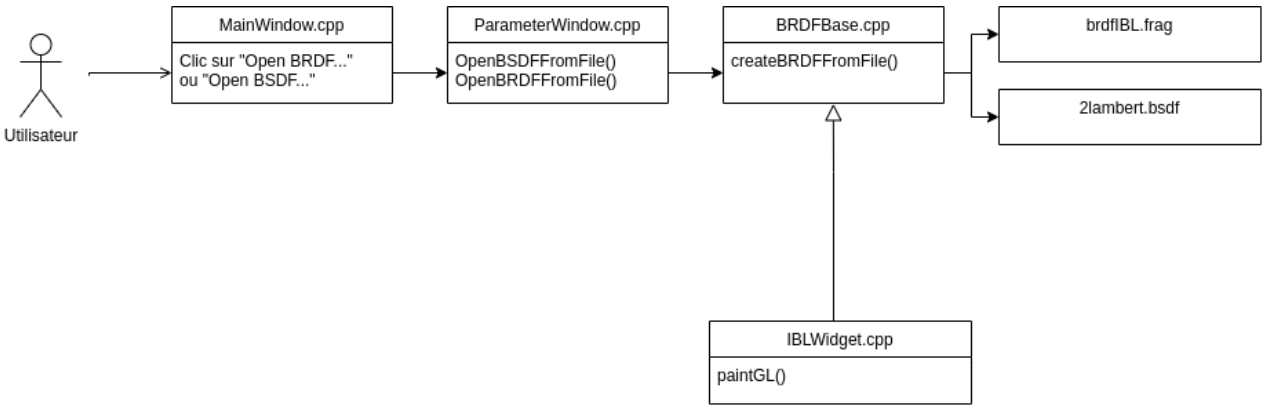


Figure 10: Scénario d'utilisation de BRDFExplorer : l'utilisateur charge le shader utilisateur "2lambert.bsdf" et BRDFExplorer effectue le rendu IBL

Pour résumer, nous présentons en figure 10 les interactions en jeu quand un utilisateur charge un shader utilisateur (ici 2lambert.bsdf) et que BRDFExplorer effectue un rendu IBL.

3.4 Les shaders

Les shaders se divisent en deux catégories : les "shader templates" internes au logiciel utilisés pour faire le rendu des différentes vues, et les shader utilisateurs qui contiennent les modèles de BRDF/BSDF. Tous sont écrits en GLSL.

Un point important est que le shader final utilisé pour le rendu d'une BRDF/BSDF est généré à la volée par le logiciel : le shader utilisateur est injecté dans shader template afin pouvoir faire le rendu.

3.4.1 Shaders utilisateurs

Les shaders utilisateur implémentent des modèles de BRDF et BSDF qui sont des fichiers au format .brdf et .bsdf.
Pour les BRDF, l'utilisateur doit implémenter au minimum la fonction BRDF(). Pour les BSDF, l'utilisateur doit en plus implémenter la fonction BTDF(). Les fonctions BRDF() et BTDF() sont appelées dans les shaders templates.

L'utilisateur peut spécifier des paramètres contrôlables depuis l'interface graphique (couleur, flottant, entier, booléen) pour faire varier les résultats des BRDF et BSDF.

```
analytic

::begin parameters
color diffuse 0.0 1.0 0.5
float reflectance 0.0 1.0 0.5
::end parameters

::begin shader

vec3 BRDF( vec3 L, vec3 V, vec3 N, vec3 X, vec3 Y )
{
    return vec3(diffuse / PI) * reflectance;
}

vec3 BTDF( vec3 L, vec3 V, vec3 N, vec3 X, vec3 Y )
{
    float transmittance = 1 - reflectance;
    return vec3(diffuse / PI) * transmittance;
}

::end shader
```

Figure 11: Exemple d'un de nos shaders utilisateur décrivant une bsdf : "2lambert.bsdf"

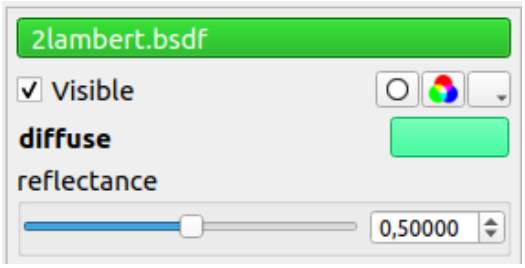


Figure 12: Fenêtre de paramètres de 2lambert.bsdf visible sur le volet de gauche de BRDFExplorer. Le paramètre contrôlable "reflectance" est défini dans le shader.

En figure 11 est présenté le code de 2lambert.bsdf. On observe les deux méthodes BRDF() et BTDF() appelées par les "shader templates". Au début du fichier sont spécifiés les paramètres contrôlables qui apparaissent sur le logiciel (voir figure 12).

3.4.2 Shader templates

Chaque vue dispose de son propre "shader template" suivant la nature du rendu (2D, 3D, IBL, etc.).

Les shader templates sont initialement incomplets : ils font appel aux fonctions BRDF et BTDF sans les définir. Ces fonctions sont injectées dans le shader template lors du chargement du shader utilisateur (.brdf ou .bsdf).

Voici les shader templates dont nous nous servons dans ce projet :

Plots :

- `brdftemplate2D.vert`, chargé de dessiner la vue "Polar Plot".
- `brdftemplateAnglePlot<ThetaD | ThetaH | Albedo>.vert`, chargés de dessiner "Theta D", "Theta H", et "Albedo".
- `brdftemplate3D.vert`, chargé de dessiner "3D plot".

Rendus :

- `brdfIBL.frag`, chargé de faire le rendu "Lit Object" qui est un rendu IBL (image-based lighting).
- `brdftemplatesphere.frag`, chargé de faire le rendu "Lit Sphere".
- `brdftemplateImageSlice.frag`, chargé de faire le rendu "Image Slice".
- `brdftemplateLobeSlice.frag`, chargé de faire le rendu "Lobe Slice".

4 Fonctionnement, réalisation et tests

Dans cette partie, nous détaillerons l'ensemble des extensions ajoutées à BRDFExplorer permettant la visualisation de BSDF, ainsi que nos modèles.

Nous avons fait en sorte que ces extensions n'aient pas d'impact pour un utilisateur souhaitant seulement visualiser des BRDF.

Pour chaque partie, nous annoncerons le/les fichier(s) concerné(s) par nos modifications.

4.1 Extension des affichages et des rendus

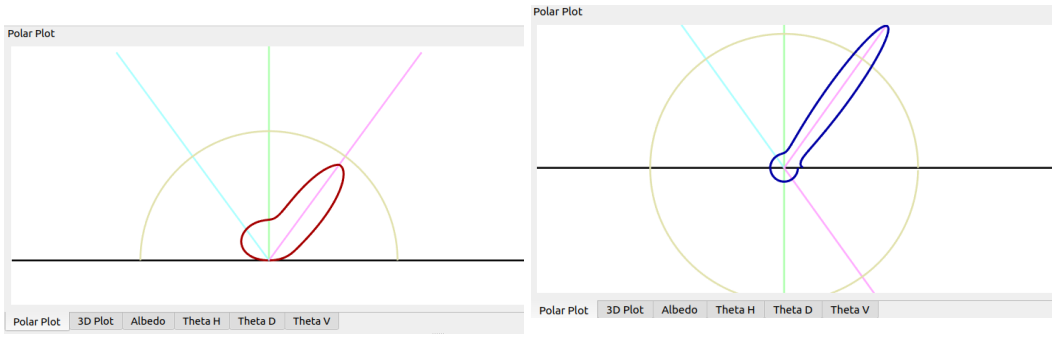
Pour être capable d'utiliser, de visualiser et d'analyser les BSDF implémentées, nous avons modifié les affichages et les rendus suivants :

4.1.1 Polar plot

Fichier(s) concerné(s) :
`src/brdf/PlotPolarWidget.h|.cpp`
`src/shaderTemplates/brdftemplate2D.vert`

Comme affiché dans la figure 13, lorsqu'une BSDF est chargée, on affiche le lobe de transmittance (en dessous) en plus du lobe de réflectance (au dessus).

On effectue une passe de shading par lobe (réflectance et transmittance) en envoyant un booléen au shader qui va déterminer si on trace la BRDF ou la BTDF. Le première passe crée des points au-dessus et la seconde en dessous de l'axe horizontal.



(a) ashikhman shirley.brdf

(b) leaf_walter asymetric.bsdf vu depuis le polar plot

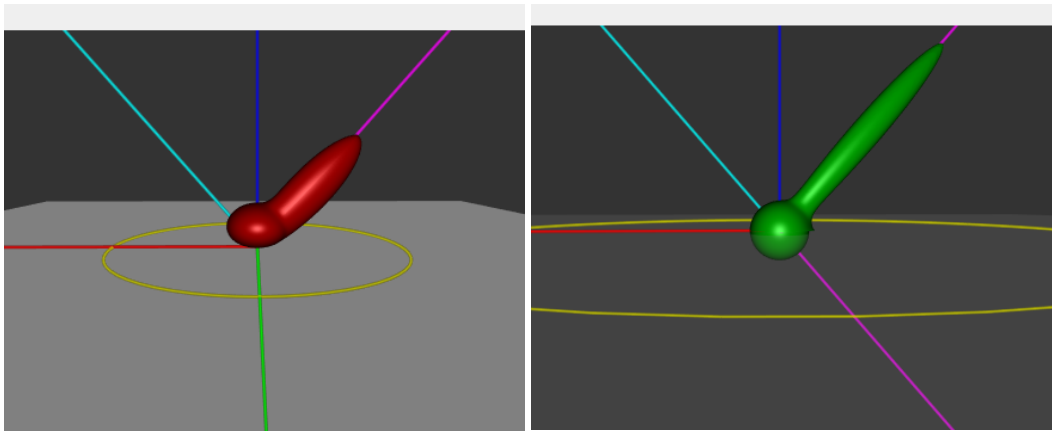
Figure 13: Comparaison du widget Polar Plot avec un fichier .brdf et un fichier .bsdf

4.1.2 3D Plot

Fichier(s) concerné(s) :
`src/brdf/Plot3DWidget.h|.cpp`

`src/shaderTemplates/brdftemplate3D.vert`

L'affichage des lobes de réflectance et de transmittance en 3D a nécessité les mêmes modifications que le Polar Plot. Nous avons également donné plus de liberté à la caméra afin de pouvoir passer sous le plan, et nous avons diminué l'opacité de celui-ci pour mieux visualiser le lobe (voir figure 14).



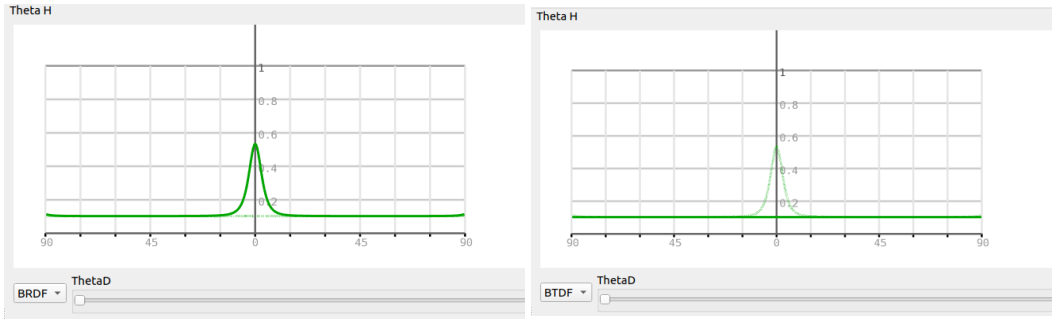
(a) ashikhman shirley.brdf (b) leaf_walter asymetric.bsdf

Figure 14: Comparaison du widget Plot 3D avec un fichier .brdf et un fichier .bsdf

4.1.3 Plots angulaires

Fichier(s) concerné(s) :
`src/brdf/PlotCartesianWindow.h| .cpp`
`src/brdf/PlotCartesianWidget.h| .cpp`
`src/shaderTemplates/brdftemplateAnglePlot.vert`
`src/shaderTemplates/brdftemplateAnglePlotThetaH.vert`
`src/shaderTemplates/brdftemplateAnglePlotThetaD.vert`

Les graphiques angulaires affichent la BRDF et la BTDF simultanément. Un menu déroulant BRDF/BTDF permet à l'utilisateur de préciser laquelle afficher en trait complet (l'autre fonction est alors tracée en pointillés). Comme précédemment, pour tracer les deux courbes, nous effectuons deux passes en précisant quelle fonction utiliser (BRDF ou BTDF) à l'aide d'un booléen envoyé au shader. Voir figure 15.



(a) leaf_walter asymetric.bsdf BRDF en trait plein

(b) leaf_walter asymetric.bsdf BTDF en trait plein

Figure 15: Comparaison du widget Theta H

4.1.4 Rendus des BSDF

Contrairement aux graphiques où l'on effectuait deux passes de shading pour calculer la BTDF et la BRDF, Pour le cas des rendus, nous évaluons la BRDF et la BTDF en faisant une seule passe (voir 5.1.3).

Afin de savoir quelle fonction on doit évaluer (BRDF ou BTDF) en fonction des paramètres (V, L, N) nous faisons le test :

$$(V \cdot N) \cdot (L \cdot N) > 0 \mid BRDF(L,V,N) \; sinon \; BTDF(L,V,N)$$

Où V est le vecteur vue, L est le vecteur lumière et N est la normale.

Ce test est réalisé dans la fonction *evaluate_BRDF(vec3 L, vec3 V, vec3 N)* dans le fragment shader des "shaders templates" (3.4.2).

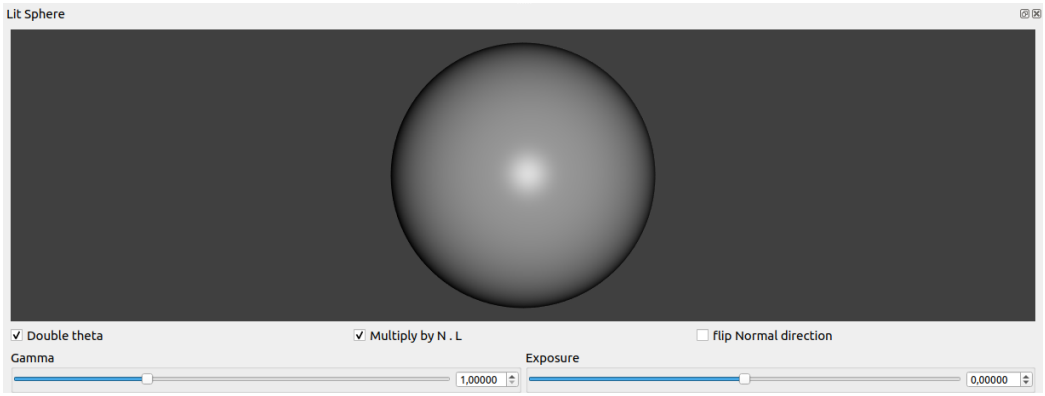
Dans le but de modéliser des BRDFs/BSDFs asymétriques, où le comportement de la fonction BRDF/BTDF peut changer selon l'orientation de la normale, **l'utilisateur doit tester l'orientation du fragment dans le shader utilisateur .brdf/.bsdf** (3.4.1).

Sur le rendu LitSphere 4.1.5, une checkbox "flip normal direction" a été ajoutée afin de pouvoir tester le comportement des BRDFs/BSDFs asymétriques (voir figure 25 pour example).

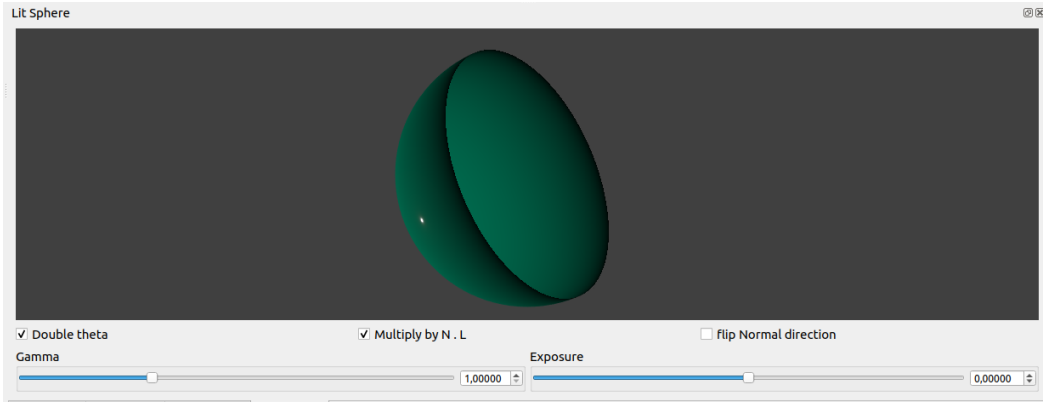
4.1.5 Rendu LitSphere

Fichier(s) concerné(s) :
`src/brdf/LitSphereWindow.h| .cpp`
`src/brdf/LitSphereWidget.h| .cpp`
`src/shaderTemplates/brdftemplatesphere.frag`

Lorsqu'une BSDF est rendue dans le widget, on affiche seulement un hémisphère pour pouvoir visualiser la couleur transmise sur la partie intérieure de celui-ci. L'hémisphère a été obtenu directement depuis le fragment shader en rejetant les fragments dont la normale est dans une direction opposée au vecteur incident de lumière. On dispose aussi d'un bouton "flip Normal direction" pour pouvoir tester les quatre cas de la BSDF. Bien que ce bouton soit présent au démarrage de l'application sans qu'aucune BRDF ou BSDF ne soit chargée, il ne changera le sens des normales que si une BSDF est chargée.



(a) ashikhman shirley.brdf



(b) leaf_walter asymetric.bsdf

Figure 16: Comparaison du widget LitSphere avec un fichier .brdf et un fichier .bsdf

4.1.6 Rendu LitObject

Fichier(s) concerné(s) :
`src/brdf/IBLWidget.cpp`
`src/shaderTemplates/brdfIBL.frag`

Ce rendu IBL (1.3.3) consiste à charger un carte d'environnement (envmap) avec un objet 3D de type wavefront (.obj) (voir figure 17).

De même que les autres shaders, nous envoyons la valeur de isBSDF() dans la variable isBSDF du shader brdfIBL.frag afin de déterminer si l'on doit ou non évaluer la BTDF.

Les modifications les plus importantes se trouvent dans la fonction `envMapSample(uv, pdf)` de brdfIBL.frag, qui sert à échantillonner la carte d'environnement.

A la base, `envMapSample()` ne retournait rien si la fonction `evaluate\_brdf()` (qui calcule si le point évalué est visible ou non par la caméra) renvoyait faux. Dans le cas des BSDF, ce n'est pas ce que l'on souhaite : si le point évalué est occulté, on calcule la BTDF.

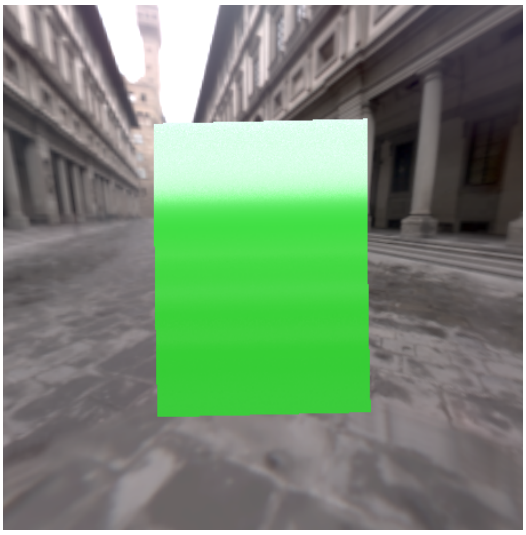
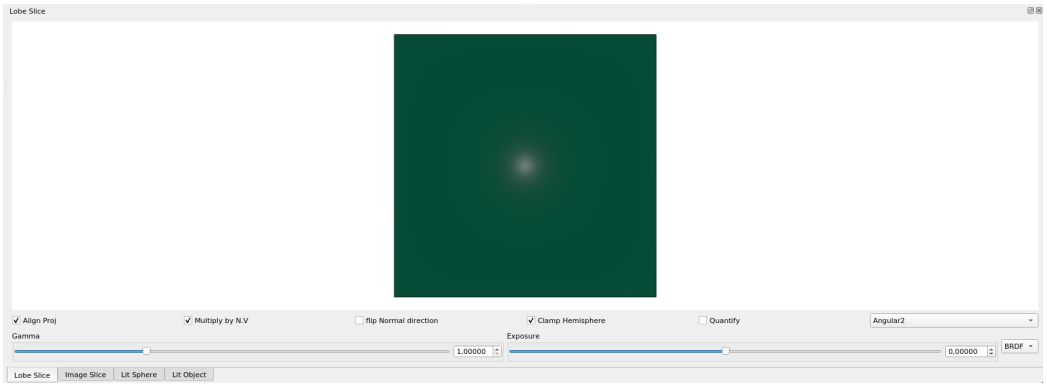


Figure 17: Exemple de rendu de lit object

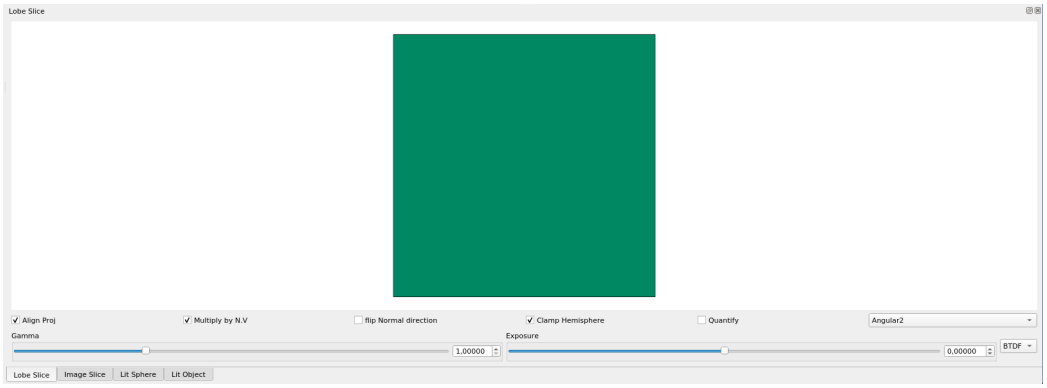
4.1.7 Rendu Lobe Slice

Fichier(s) concerné(s) :
`src/brdf/LobeSliceWindow.h| .cpp`
`src/brdf/LobeSliceWidget.h| .cpp`
`src/shaderTemplates/brdftemplateLobeSlice.frag`

Le fragment shader a été modifié comme décrit en 4.1.4 pour rendre ce widget compatible avec les BSDF. Un bouton permettant de choisir la fonction affichée a été ajouté (voir figure 18).



(a) Côté BRDF du shader leaf_walter_asymetric.bsdf



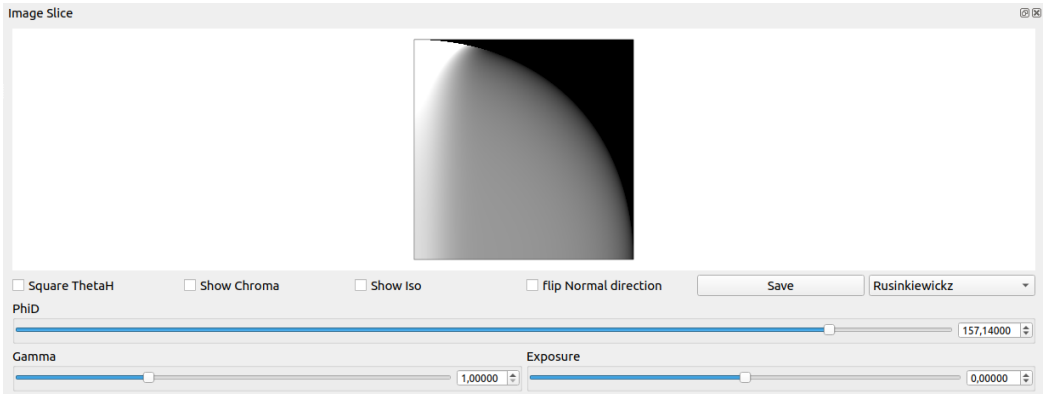
(b) Côté BTDF du shader leaf_walter_asymetric.bsdf

Figure 18: Comparaison du widget LobeSlice avec un fichier .brdf et un fichier .bsdf

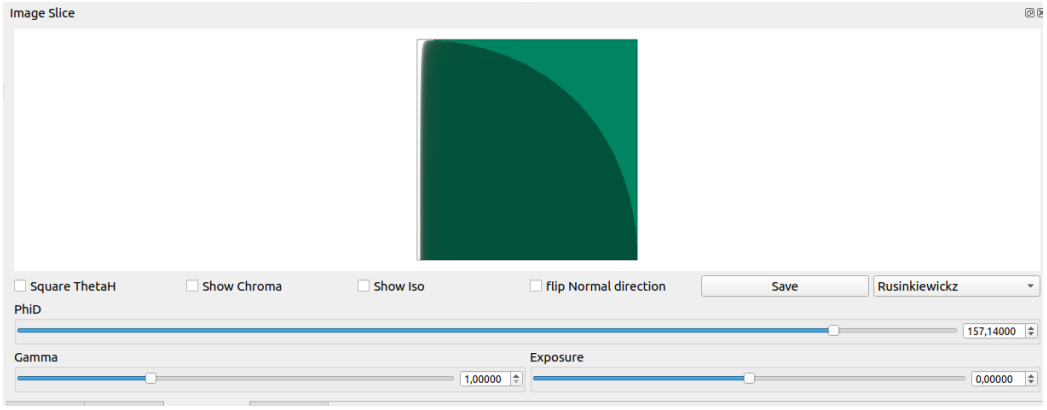
4.1.8 Rendu Image Slice

Fichier(s) concerné(s) :
`src/brdf/ImageSliceWindow.h| .cpp`
`src/brdf/ImageSliceWidget.h| .cpp`
`src/shaderTemplates/brdftemplateImageSlice.frag`

Le bouton permettant de choisir si on veut rendre la BRDF ou la BTDF n'a pas été ajouté dans ce widget (voir figure 19), car on peut déjà voir la BTDF et la BRDF sur la projection du lobe en faisant varier le slider phiD. Un bouton flip normal direction a aussi été ajouté pour pouvoir tester les quatre cas de la BSDF.



(a) ashikhman shirley.brdf



(b) leaf_walter asymetric.bsdf

Figure 19: Comparaison du widget ImageSlice avec un fichier .brdf et un fichier .bsdf

4.2 Les fichiers .bsdf

Fichier(s) concerné(s) :
`src/bsdfs/*`

Les fichiers .brdf existant contiennent uniquement une définition de BRDF et ses paramètres modifiables par l'utilisateur. Pour étendre BRDFExplorer au chargement de BSDF, nous avons fait le choix

de créer un nouveau type de shader utilisateur définissant une BRDF **et** une BTDF. Afin de les différencier des shaders existant, nous leur avons donné l'extension **.bsdf**. Ces shaders sont également écrits en GLSL et décrivent nos modèles de feuilles.

De même que pour les fichiers .brdf, nous pouvons ajouter des paramètres contrôlables par l'utilisateur qui modifient le retour de la fonction BTDF. La seule différence est donc l'ajout d'une fonction BTDF() qui prend les mêmes arguments que la fonction BRDF().

```
vec3 BTDF( vec3 L, vec3 V, vec3 N, vec3 X, vec3 Y )
{
    ... // définition de la BTDF
}
```

Figure 20: Fonction BTDF présente dans les fichiers .bsdf

A l'instar des fichiers .brdf, nous avons choisi de créer un répertoire "src/bsdfs", qui contiendra tous les fichiers .bsdf.

4.2.1 Chargement de fichiers .bsdf

Fichier(s) concerné(s) :
`src/brdf/MainWindow.cpp`
`src/brdf/ParameterWindow.h|.cpp`

Pour charger une BSDF au format .bsdf, l'utilisateur clique sur le bouton "File -> Open BSDF ...".

S'ouvre ensuite une boîte de dialogue invitant l'utilisateur à choisir un ou plusieurs fichiers .bsdf présent dans le dossier bsdfs/.

Pour ajouter le bouton "Open BSDF...", nous avons ajouté un objet de type `QAction` dans `MainWindow.cpp` qui exécute `openBSDFFromFile()` de la classe `ParameterWindow` lorsque l'on clique dessus. `openBSDFFromFile()` instancie un file dialog différent (nommé `fileDialogBSDF`) que nous avons également ajouté dans ParameterWindow.

4.2.2 Gestion des BSDF dans le code de BRDFExplorer

Fichier(s) concerné(s) :
`src/brdf/BRDFBase.h|.cpp`

Pour gérer les BSDF dans le code de BRDFExplorer, nous sommes partis de l'observation qu'un grand nombre de méthodes de la classe BRDFBase seraient partagées par les BSDF.

Ainsi, nous avons ajouté à la classe BRDFBase un booléen "isBSDF" qui définira si le fichier chargé par l'utilisateur est une BRDF ou une BSDF. Nous sommes conscients que le nom de "BRDFBase" est rendu moins cohérent par cet ajout, mais cela évite la duplication de code en nous dispensant de la création d'une classe BSDFBase, en plus de minimiser les modifications faites au code source original de BRDFExplorer.

4.2.3 Gestion des BSDF dans la génération du shader de rendu

Fichier(s) concerné(s) :
`src/brdf/BRDFAnalytic.cpp`

Comme nous l'avons expliqué dans la partie 3, le code final du shader de rendu est généré à la volée à partir du shaderTemplate et du fichier .brdf.

Ainsi, étant donné que le shaderTemplate est amené à dessiner des BTDF (si un .bsdf est chargé), nous devons faire en sorte qu'une fonction BTDF soit définie dans tous les cas, et ce même si c'est un fichier .brdf qui est chargé.

Pour ce faire, nous avons choisi d'injecter à la volée une fonction BTDF retournant zéro dans le shader utilisateur **uniquement** s'il s'agit d'un fichier .brdf (car dans le cas contraire, une fonction BTDF est déjà définie).

Cette injection est faite grâce à la méthode `beginSection()` de BRDFAnalytic : lorsque l'analyse syntaxique du shader utilisateur tombe sur le mot-clé "shader", cela signifie qu'on rentre dans la section où est normalement définie la BRDF. C'est donc au début de cette section que l'on ajoute une fonction BTDF retournant 0, seulement s'il s'agit d'un fichier .brdf.

4.3 Implémentation de modèles de feuilles

Cette sous-partie traite des modèles de feuille que nous avons implémenté en nous inspirant de la littérature existante (voir partie 5 pour visualiser les rendus des modèles).

4.3.1 Lambertien

Fichier(s) concerné(s) :
`src/bsdfs/2lambert.bsdf`

Le modèle le plus simple est le modèle lambertien, qui diffuse uniformément la lumière dans toutes les directions. Le retour de la BRDF lambertienne est k_d/π peu importe l'angle incident, avec k_d la couleur diffuse propre au matériau, et pi la pdf associée à l'échantillonnage uniforme sur tout l'hémisphère unitaire.

Si l'on souhaite ajouter une transmission lambertienne, on doit s'assurer que la conservation d'énergie est bien respectée, autrement dit que $\rho_r + \rho_t <= 1$, avec p_r la BRDF et p_t la BTDF ($<= 1$ car il peut y avoir de l'absorption). Pour cela, nous avons ajouté un slider "reflectance" qui permet à l'utilisateur de contrôler le ratio réflectance/transmittance.

Les retours des fonctions BRDF et BTDF sont donc tous deux k_d/π mais pondérés par un coefficient de réflectance et de transmittance.

4.3.2 Walter et lambert

Fichier(s) concerné(s) :
`src/bsdfs/leaf_walter.bsdf`

L'article [9] présente un modèle de BTDF basé sur le modèle des microfacettes ([**?**]). Un modèle de BRDF extrait de [9] existe déjà dans le répertoire brdfs. En utilisant Walter pour la BRDF et Lambert pour la BTDF, on peut conserver les reflets spéculaires tout en gérant la transmittance.

4.3.3 BSDF asymétriques

Fichier(s) concerné(s) :
`src/bsdfs/leaf_walter_asymetric.bsdf`

Une feuille présente rarement les mêmes propriétés sur le dessus que sur le dessous de sa surface. Notre modèle `leaf_walter_asymetric.bsdf` permet de faire varier la rugosité du dessus et du dessous grâce aux paramètres `alphaG_top` et `alphaG_bottom`.

4.3.4 Modèle de Dai

Fichier(s) concerné(s) :
`src/bsdfs/dai.bsdf`

L'article [1] propose une méthode utilisant les microfacettes pour la BRDF et pour la BTDF afin de modéliser des fines parois semi-transparentes. Il se sert de SVBTDF (spatially-variant BTDF), mais nous utiliserons ici de simples BTDF.

4.4 Tests

Tester de façon automatique du code GLSL qui est exécuté sur GPU (Graphics processing unit) est une tâche assez laborieuse, notre client nous a conseillé de faire des "shaders de test" qui serviront de vérification à notre implémentation.

Nous avons implémenté deux shaders permettant de vérifier, que les BRDF/BTDF sont évaluées correctement. Le premier shader, debug.bsdf permet de vérifier que la BTDF et la BRDF sont appelées pour les bons fragments dans un shader de rendu. La figure 21 dans la vue Lit Sphere met en évidence la couleur réfléchie depuis l'extérieur de l'hémisphère par la BRDF en rouge et la couleur transmise par la BTDF en cyan à l'intérieur de l'hémisphère.

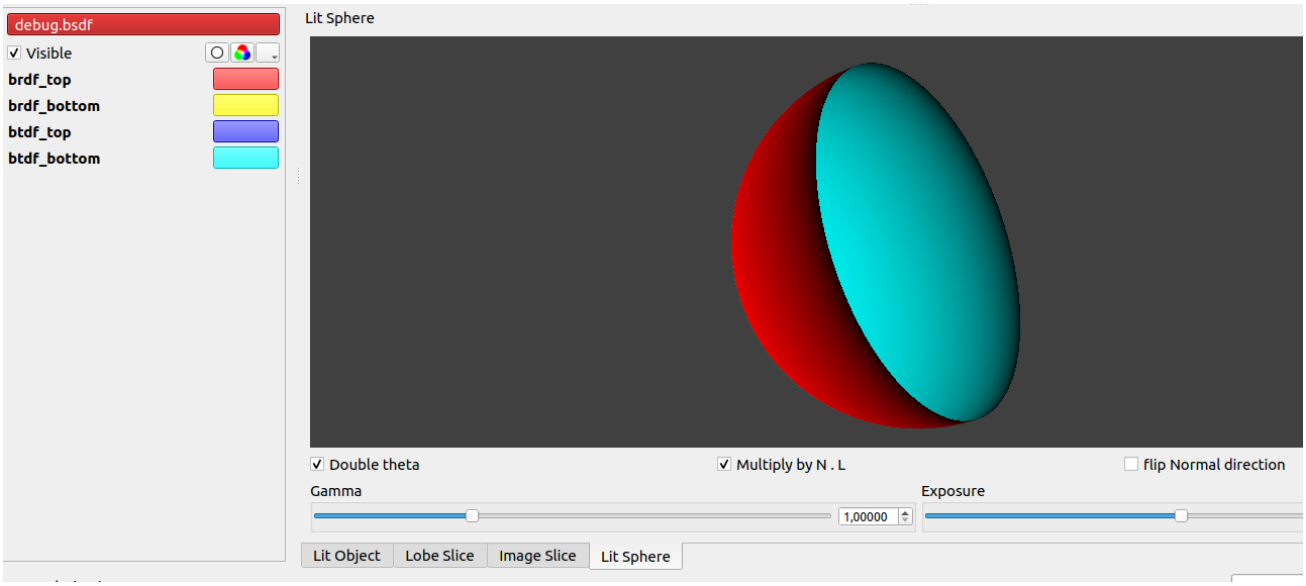


Figure 21: Shader debug.bsdf dans la vue LitSphere

La figure 22 met en évidence le cas où la BSDF est asymétrique en utilisant le bouton flip normal direction, c'est-à-dire lorsque le matériau a une BRDF ou une BTDF différente selon s'il est vu par-dessus ou par-dessous.

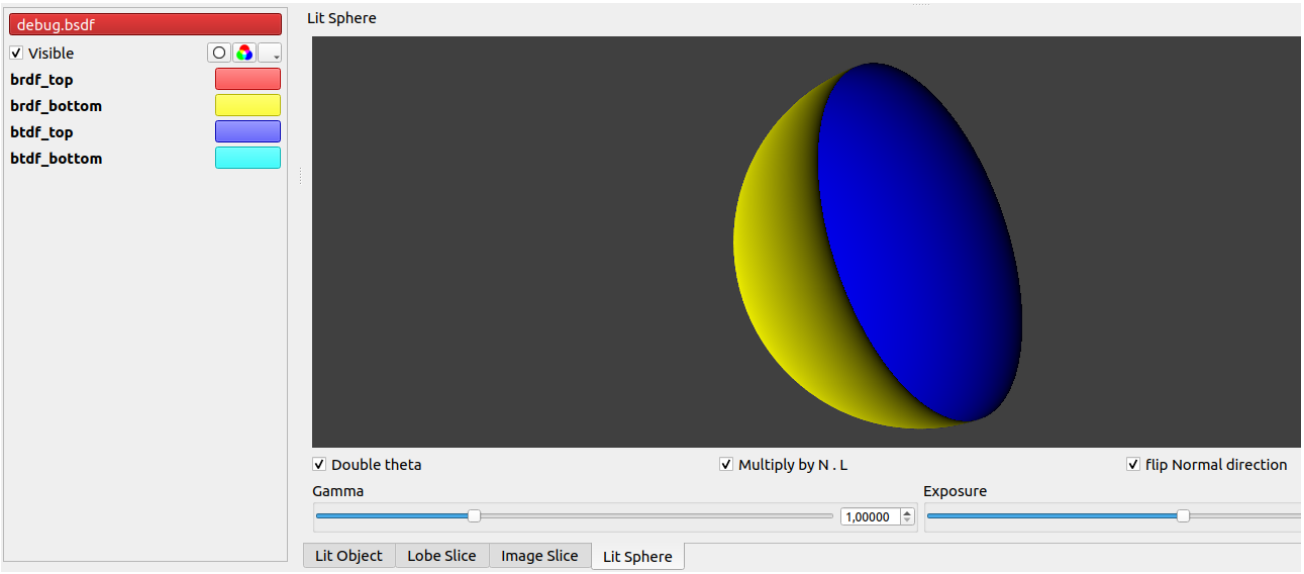


Figure 22: Shader debug.bsdf dans la vue LitSphere avec normales des faces dans le sens opposé

Le deuxième shader de test cone.bsdf, affiché dans la figure 23 est utilisé dans le graphique Polar Plot pour vérifier que les rayons de transmittance affichées correspond bien à la taille d'un cône.

On supprime toute valeur en réfléctance en mettant la valeur de la *BRDF* à 0 ensuite, dans la fonction *BTDF()* on fait le test

$$(V \cdot -L) > 1 - \varepsilon \ : \ \textit{return RED else BLACK}$$

Où ε est un réel proche de 0.

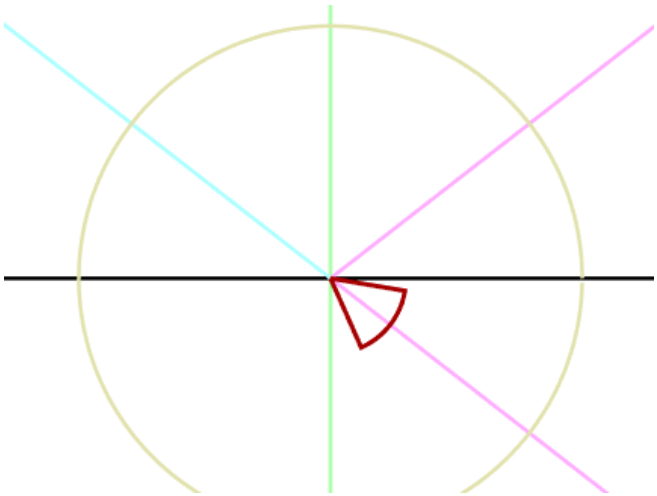


Figure 23: Shader cone.bsdf dans la vue Polar Plot

Avec ce test avons pu détecter plusieurs bugs liées à l'affichage sur les vues PolarPlot 4.1.1 et 3dPlot 4.1.2.

5 Résultats

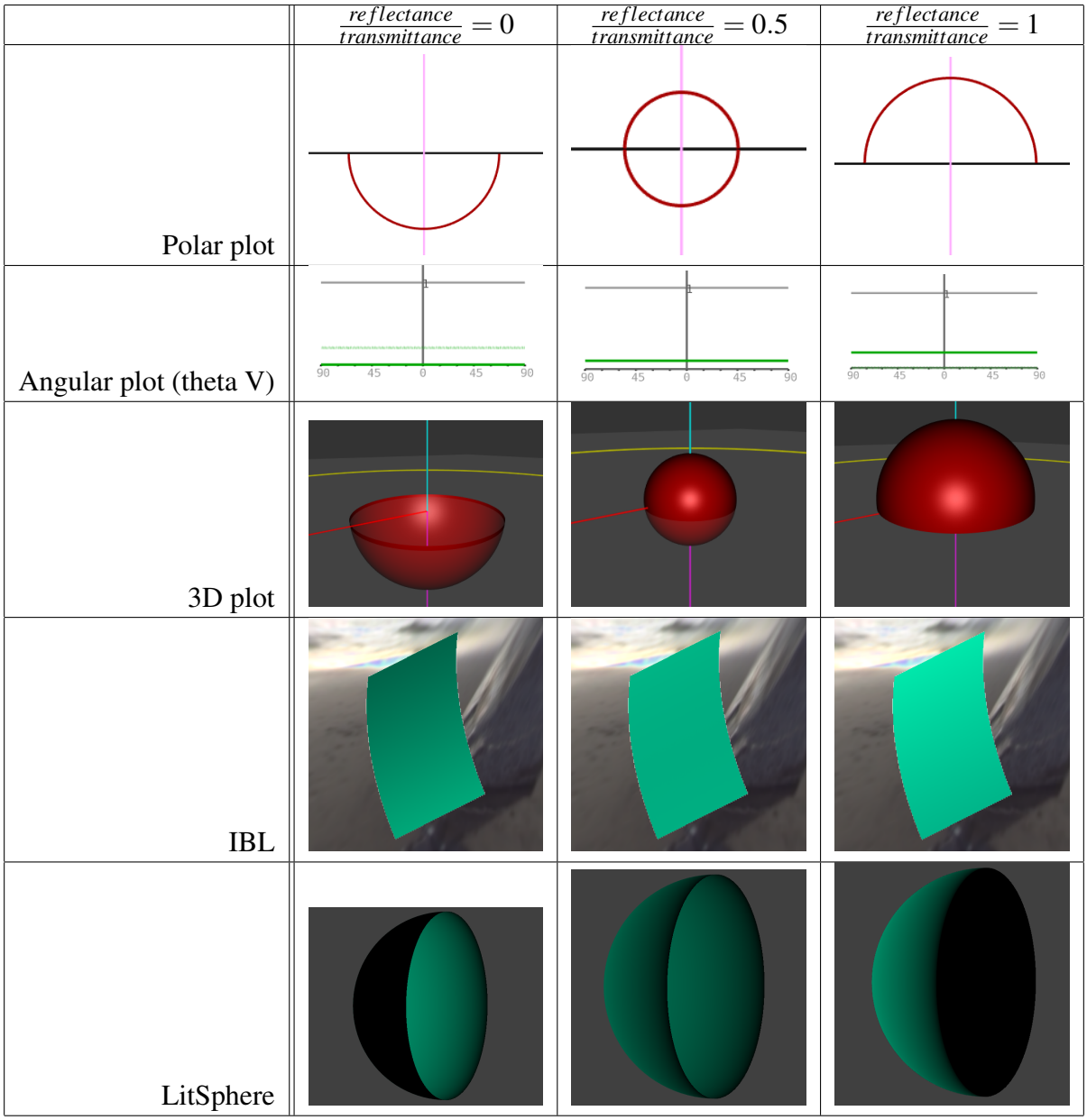


Figure 24: Résultats obtenus avec le modèle 2lambert.bsdf

En figure 25 sont présentés les différentes vues pour 2lambert.bsdf. Comme dit précédemment, nous avons ajouté un slider contrôlant le ratio réflectance/transmittance. litSphere présente un résultat cohérent : la lumière se trouvant à gauche, la partie noire se trouve côté lumière quand le ratio est à 0, et elle se trouve à l'intérieur de l'hémisphère quand le ratio est à 1 (aucune lumière n'est transmise).

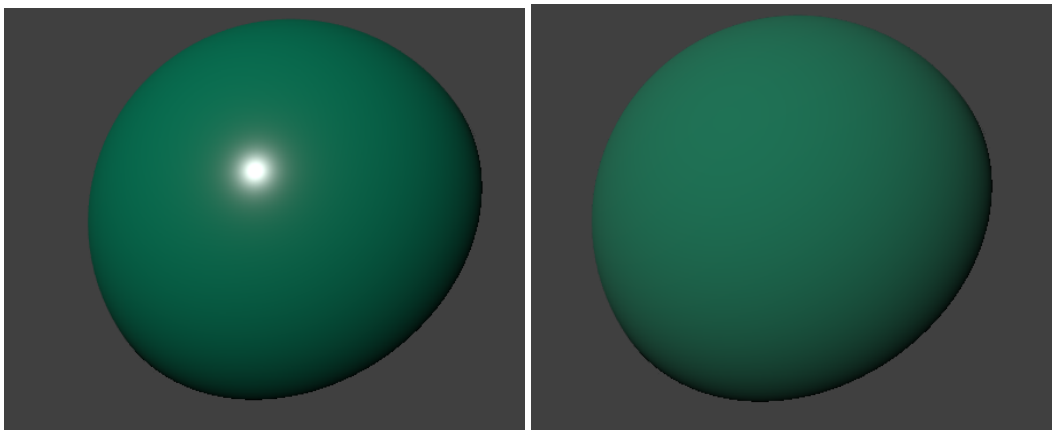


Figure 25: Résultats obtenus avec le modèle leaf_walter_asymmetric.bsdf sur la vue "LitSphere". La rugosité du la BRDF du dessus est à 0.05, celle du dessous à 1. Sur l'image de droite, les normales sont inversées : le rendu est donc fait avec la BRDF du dessous.

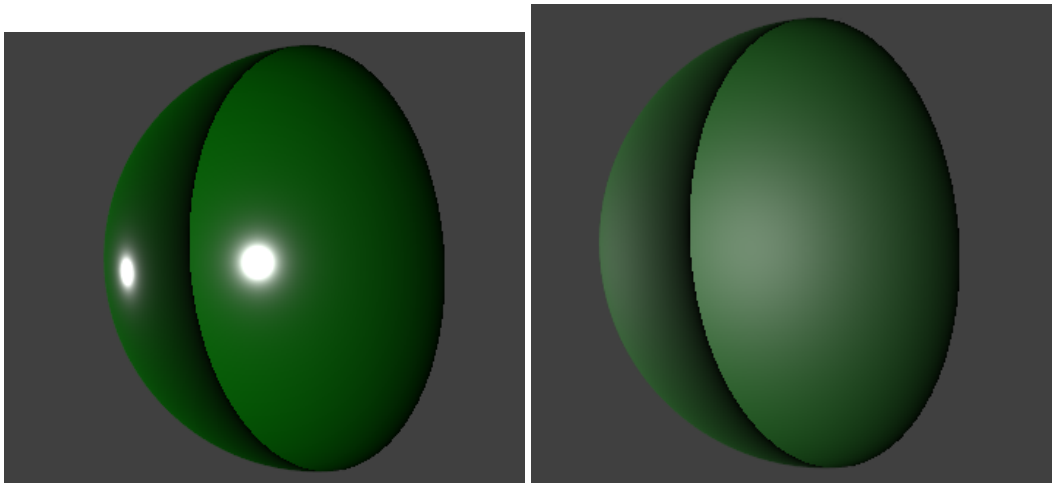


Figure 26: Résultats obtenus avec le modèle dai.bsdf sur la vue "LitSphere". On observe à gauche que les reflets spéculaires traversent le matériau. En augmentant la rugosité de la surface du dessus, les reflets spéculaires transmis deviennent également plus diffus (à droite).

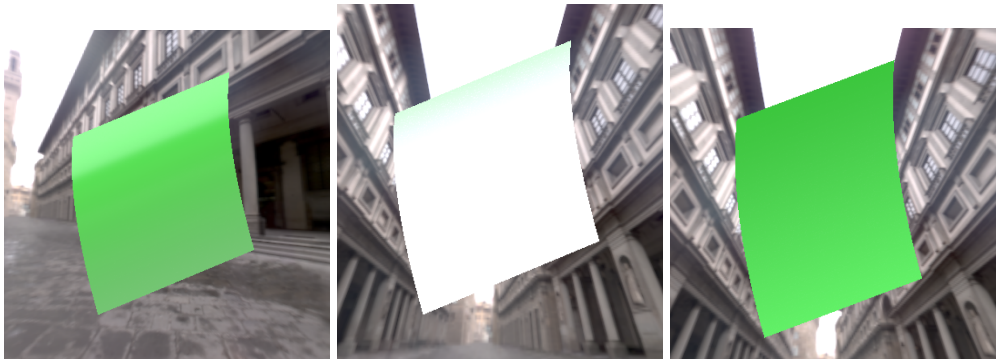


Figure 27: Rendus IBL du modèle dai.bsdf avec une rugosité à 0 pour la surface du dessous. A gauche et au milieu, on se rend compte que la lumière passe à travers. En changeant la rugosité du dessous à 1, la lumière transmise devient diffuse, d'où le fait que la couleur redevienne verte.

5.1 Limitations

5.1.1 bugs connus

- Le projet initial comportait un bug qui empêchait le widgetIBL de fonctionner : la variable `viewportSize` dans la fonction `InitializeGL()` était toujours égale à zéro à l'issue de son initialisation. Nous avons donc introduit une condition permettant de donner à `currentSize` une taille par défaut si `viewportSize` est nulle. Cependant, sur certains ordinateurs, ce changement entraîne le dysfonctionnement du widget 3D Plot qui ne dessine plus les lobes de réflectance et de transmittance. Ce bug est difficile à régler, car nous n'arrivons pas à identifier quel est l'origine de ce dysfonctionnement. La version de QT ou le système d'exploitation ne semblent pas être l'origine du problème.

```
viewportSize = width() < height() ? width()*devicePixelRatio()
    : height()*devicePixelRatio();
    if(!shownResolution || viewportSize == 0)
        currentSize = fboSize;
    else
        currentSize = viewportSize;
```

- L'utilisation simultanée de IBLWidget et 3D Plot déclenche parfois un bug d'affichage sur l'affichage de IBLWidget créant de nombreux artefacts dans ce widget. Nous n'avons pas réussi identifier les étapes pour reproduire ce bug.

5.1.2 Limitations de l'expérience utilisateur

L'interface graphique de BRDF Explorer est déjà chargée avec de nombreux paramètres de contrôles. Il n'est pas intuitif pour un utilisateur qui n'a pas lu ce rapport de savoir quelles sont les

modifications qui ont été apportées au logiciel et les options qui sont disponibles lorsqu'on charge une BSDF. De plus, certains boutons comme flip Normal Direction ou celui pour changer quelle fonction afficher entre la BTDF et la BRDF sont présent dès le lancement de l'application même s'ils n'ont pas d'effet tant qu'une BSDF n'est pas chargée.

5.1.3 Incohérence entre calcul de BSDF entre différentes widgets

Comme mentionné dans la partie 4.1.4, nous calculons les BSDF des affichages de type "PolarPlot" et "AngularPlot" en faisant deux passes au shader, en envoyant une variable booléenne au shader pour indiquer si on doit évaluer la BRDF ou la BTDF. Alors que pour le reste des rendus nous faisons uniquement une seule passe au shader et dans le shader, on décide laquelle des deux afficher.

Cette incohérence est due à une décision logicielle prématurée au début du projet, laquelle nous n'avons pas eu le temps de changer, **en effet, une opération de refactoring** est à considérer afin de garder la même logique de traitement dans les différents widgets du logiciel.

5.2 Tests unitaires

Malgré avoir mis en place un mécanisme de test avec des shaders de debug (voir 4.4). Il y a quelques éléments de l'application qui auraient pu être testés de façon automatique comme le chargement des fichiers .bsdf ainsi que le bon fonctionnement des éléments de l'interface graphique. Cependant, nous avons choisi de tester en priorité les fonctionnalités critiques du logiciel, telles que la bonne évaluation des fonctions BRDF et BTDF.

6 Gestion du travail

6.1 Organisation et avancement du travail

Nous avons essayé respecter au maximum le processus agile pour développer de nouvelles fonctionnalités dans BRDF Explorer. Lors de la première réunion de présentation du sujet, nous avons identifié les axes principaux de travail : mettre à jour chaque vue dans le logiciel pour visualiser les bsdf et implémenter des modèles de bsdf à l'aide de la littérature.
Toutes les semaines ou deux semaines, nous avons fait une réunion avec notre client M. Pascal Barla pour présenter notre avancée, poser des questions et essayer de déterminer quelles étaient les tâches prioritaires. La figure 28 montre les périodes de travail pour chaque tâche. La majeure partie du travail du projet a été de comprendre l'existant et les articles sur les modèles de BSDF. Une fois le fonctionnement du logiciel bien assimilé, il a été plus facile de finaliser toutes les vues vers la fin du projet.

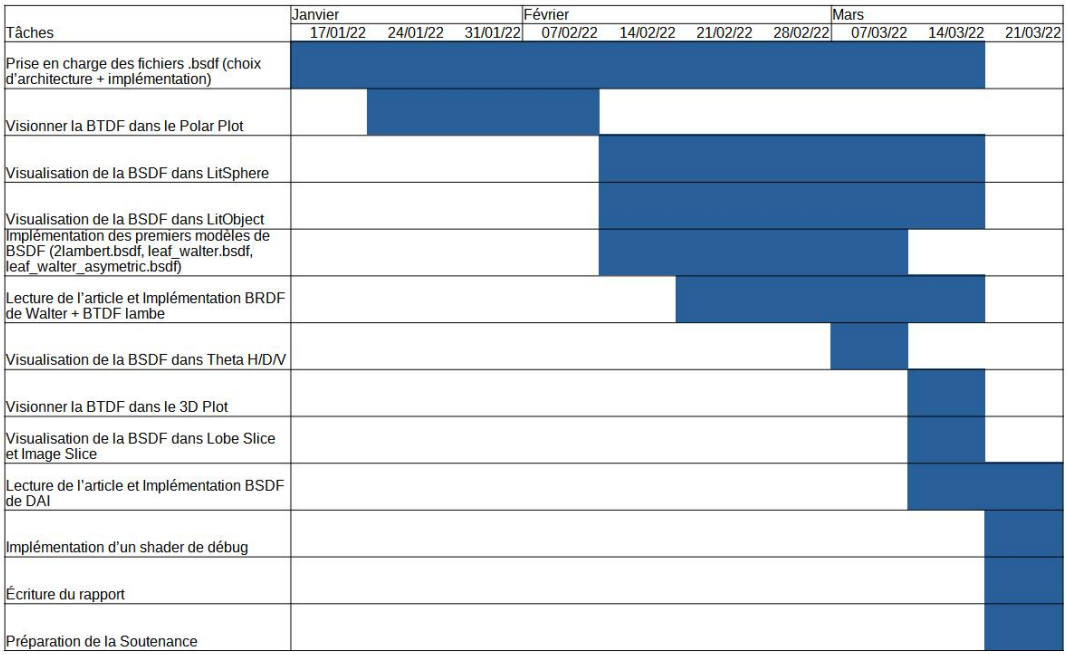


Figure 28: Diagramme de Gantt

6.2 Les outils utilisés

Tout le long du projet, nous avons utilisé principalement trois outils :

- Trello est un site internet de gestion de projet. Il nous a permis de gérer, à travers des tâches et des sprints d'une semaine, l'avancement du projet. Pour mettre en œuvre cette gestion, nous avons utilisé plusieurs tableaux.
  Un premier tableau "Backlog" qui contient les Users Stories et l'ensemble des tâches à faire.
  Un second tableau "Sprint" assez standard (Il contient 7 colonnes : "To do", ""In progress", "To test", "Done", "Blocked" – pour les tâches bloquées par un bug, une feature obligatoire non terminée, un problème de conception, etc. – et "Abandoned" – pour les tâches devenus inutiles ou trop compliquées à réaliser –).
  Un dernier tableau a été utilisé pour archiver, semaine après semaine, la colonne "Done" du tableau "Sprint", cela dans le but d'avoir un regard sur l'avancement du projet au fil du temps.

- Nous avons beaucoup utilisé Discord pour la communication. Cela nous a permis de faire régulièrement des réunions à distance et des séances de travail en commun avec le partage d'écran. Nous avons également pu centraliser les informations importantes (les liens, les dates, les comptes-rendus de réunion avec le client). Le serveur que nous utilisons est découpé en plusieurs canaux, afin de faciliter les échanges (un canal pour ce rapport, un autre pour le développement, un autre pour les articles, les liens, etc.).

- Gitlab, avec directement l'utilisation du git interne du projet de l'INRIA. Une branche a été créée spécialement pour notre projet. Nous avons développé les fonctionnalités dans des branches parallèles que nous avons ensuite intégrées dans la branche principale (notamment pour les extensions des différentes vues du logiciel).

7 Bilan et perspectives

Dans ce projet de fin d'études, nous avons essayé d'adapter un logiciel existant en le rendant compatible avec autre modèle physique pour visualiser l'interaction entre la lumière et les matériaux : la BSDF. Dans le cadre du Master, ce projet nous a permis de mettre en application des notions vues dans le cours de Synthèse d'image avancée ainsi que celles du cours Méthodes et outils pour la conduite de projet informatique. Une grande partie de notre travail a été consacrée à la compréhension du logiciel en profondeur, notamment sur la génération des shaders de chaque vue en utilisant les templates shader et les shaders utilisateur.

Ce projet présente encore des axes d'améliorations. Nous nous sommes focalisés sur les graphiques Polar Plot et les rendu IBL et LitSphere (4.1.4) qui avaient plus d'importance pour notre client. Nous avons apporté des modifications basiques sur les autres vues pour qu'elles soient compatibles, mais elles méritent plus d'attention en terme de design pour permettre une meilleure expérience utilisateur du logiciel. De plus, nous aurions souhaité aller plus loin dans les améliorations pour que les BSDF fonctionnent dans la vue IBL avec l'importance sampling et le multiple importance sampling. Pour cela, il faut rajouter l'échantillonnage de la BTDF. Enfin, une autre fonctionnalité intéressante aurait été d'implémenter les rebonds des rayons dans les widget IBL et LitSphere qui pour pouvoir visualiser la transmittance dans un objet fermé.

Notre travail a été de rendre le logiciel plus souple, plus générique, mais le logiciel a été conçu pour les BRDF. Un axe d'amélioration du logiciel serait de faire une refonte complète pour pouvoir refactoriser tout le code en renommant les classes adéquatement et en fournissant une base permettant d'implémenter facilement de nombreux modèles de BSDF et proposer une interface graphique plus flexible selon le modèle chargé.

Ce projet de fin d'études nous a permis d'approfondir nos connaissances en synthèse d'images tout en travaillant sur un domaine qui nous sort de nos habitudes.

References

[1] Qiang Dai, Jiaping Wang, Yiming Liu, John Snyder, Enhua Wu, and Baining Guo. The Dual-microfacet Model for Capturing Thin Transparent Slabs. Computer Graphics Forum, 2009.

[2] Eustace L. Dereniak, Langford G. Brod, and John E. Hubbs. Bidirectional transmittance distribution function measurements on znse. Appl. Opt., 21(24):4421–4425, Dec 1982.

[3] B. Duvenhage, K. Bouatouch, and D. G. Kourie. Numerical verification of bidirectional reflectance distribution functions for physical plausibility. SAICSIT '13, page 200–208, New York, NY, USA, 2013. Association for Computing Machinery.

[4] Adrià Forés, Sumanta N. Pattanaik, Carles Bosch, and Xavier Pueyo. Brdflab: A general system for designing brdfs. In Carlos Andújar and Javier Lluch, editors, XIX Spanish Computer Graphics Conference, CEIG 2009, San Sebastián, Spain, September 9-11, 2009, pages 153–160. Eurographics Association, 2009.

[5] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '93, page 165–174, New York, NY, USA, 1993. Association for Computing Machinery.

[6] Duck Bong Kim, Kang Park, Kang Kim, Myoung Seo, and Hiu Kwan Lee. High-dynamic-range camera-based bidirectional reflectance distribution function measurement system for isotropic materials. Optical Engineering - OPT ENG, 48, 09 2009.

[7] Fred E. Nicodemus. Directional reflectance and emissivity of an opaque surface. Appl. Opt., 4(7):767–775, Jul 1965.

[8] Matt Pharr, Wenzel Jakob, and Greg Humphreys. Physically Based Rendering: From Theory to Implementation (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, November 2016.

[9] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In Proceedings of the 18th Eurographics Conference on Rendering Techniques, EGSR'07, pages 195–206, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.