Université de Limoges
Faculté des Sciences & Techniques
Faculté des Sciences et Techniques de Limoges

ISIR - TP3 - Sources lumineuses surfaciques et ombres douces

Maxime MARIA

2022-2023

Dans ce TP, vous allez prendre en compte des sources lumineuses surfaciques et calculer des ombres douces. Dans un premier temps, vous allez paralléliser votre boucle de rendu pour diminuer les temps de calcul.

1 Réduisons les temps de calcul

Les pixels étant indépendant entre eux, la boucle de rendu est facilement parallélisable. Ici nous allons utiliser OpenMP [1] (de manière naïve).

1. Il faut tout d'abord indiquer au compilateur que nous allons utiliser OpenMP. Pour cela, allez dans les propriétés de votre projet (`Project` > `RT_ISICG Properties`). Dans `Configuration Properties` > `C/C++` > `Language`, mettez l'option `Open MP Support` à `Yes (/openmp)`.
2. Pour activer le traitement parallèle de la boucle de rendu, il suffit d'écrire `#pragma omp parallel for` au dessus de la boucle traitant les lignes de pixels.
3. Exécutez votre code et comparez les temps de calcul.

2 Sources lumineuses surfaciques : le quad

Dans le TP précédent, la scène était uniquement éclairée par une source lumineuse ponctuelle : celle-ci était soit visible depuis le point observé (qui était donc éclairé), soit invisible (qui était donc dans l'ombre). Les ombres calculées ainsi étaient donc des ombres dures, ou franches (*cf.* Figure 1(a)). Dans le monde réel, les sources ponctuelles n'existe pas, les ombres dures produisent donc une image visuellement peu réaliste.

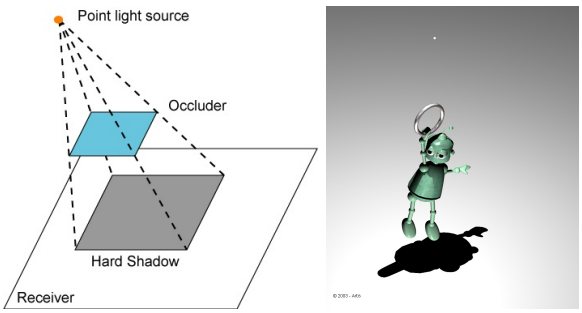
Pour une source lumineuse surfacique (possédant une aire), il est possible que seule une partie de celle-ci soit visible depuis le point observé. La proportion de la source visible depuis le point de vue va donc influencer l'éclairage (*cf.* Figure 1(b)). Il y aura des zones d'ombre complète (la source n'est totalement pas visible), des zones de pénombre (la source est partiellement visible) et des zones éclairées (la source est entièrement visible).

Dans cet exercice, vous allez ajouter un nouveau type de source lumineuse ayant la forme d'un quadrilatère (quad).

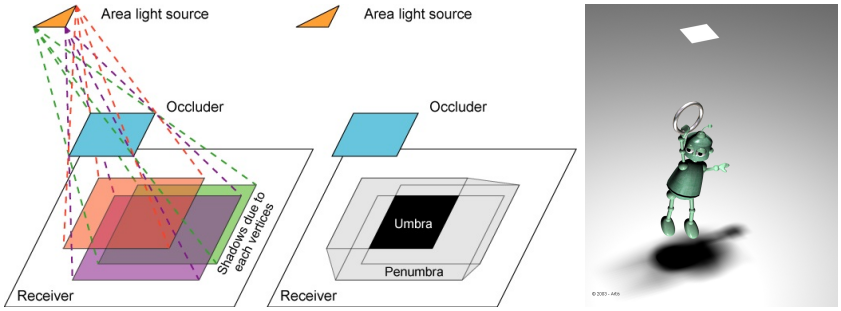
1. Ajoutez une classe `QuadLight` héritant de `BaseLight`. Cette classe possède cinq attributs (*cf.* Figure 2) :
   — la position d'un de ses coins (`_position`) ;
   — deux vecteurs représentant les deux arêtes partant du coin donné (`_u` et `_v`) ;
   — sa normale (`_n`) ;
   — son aire (`_area`).
2. Définissez un constructeur prenant en paramètre la position, les vecteurs d'arêtes, la couleur et la puissance de la source lumineuse :

```
1 QuadLight( const Vec3f & p_position,
2            const Vec3f & p_u,
3            const Vec3f & p_v,
4            const Vec3f & p_color,
5            const float   p_power );
```

1. https://www.openmp.org/



(a)



(b)

FIGURE 1 – Ombres dures et ombres douces : (a) ombre dure produite par une source lumineuse ponctuelle ; (b) ombre douce produite par une source lumineuse surfacique (un quad). *Source* : [HLHS03].

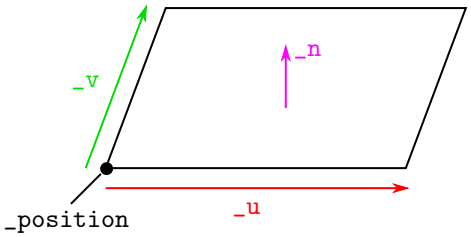


FIGURE 2 – Schéma d'une source lumineuse de type quad.

1. Définissez la méthode `QuadLight::sample` qui, à partir d'une position passée en paramètre, retourne un `LightSample`. Celle-ci est un peu plus complexe que pour la `PointLight` :
   (a) Dans un premier temps, il faut générer une position aléatoire sur le quad (utilisez la fonction `randomFloat()` disponible dans le fichier `utils/random.hpp` qui génère un nombre aléatoire entre 0 et 1).
   (b) Ensuite, il faut calculer la direction du rayon entre le point observé et la position sur le quad.
   (c) La PDF (`_pdf`, *Probability Density Function*) est égale à la PDF pour générer un point sur le quad (l'inverse de l'aire du quad) multipliée par le facteur géométrique (*i.e.* la distance au carré entre entre le point du quad et le point observé, divisée par le cosinus de l'angle entre la normale et la direction).
   (d) Pour finir, la radiance est égale à la couleur de la lumière multipliée par sa puissance, divisée par la pdf.
2. Remplacez la source ponctuelle de la scène par une `QuadLight` à la position `(1, 10, 2)`, de vecteurs `_u = (-2, 0, 0)` et `_v = (0, 0, 2)`, de couleur blanche et avec une puissance de 40.
3. Désactivez l'anti-aliasing et lancez votre programme, vous devriez obtenir l'image 3.

3 Moins de bruit svp !

La génération aléatoire d'une direction pour tester l'éclairage d'un quad induit bien évidemment du bruit dans notre image. De plus, vu que nous lançons qu'un seul rayon, il n'y a pas d'ombres douces. Pour pallier ce problème, il suffit de lancer plusieurs rayons d'ombrage...

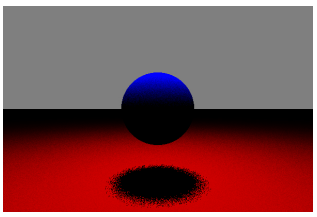


FIGURE 3 – Résultat de l'exercice 2

1. Ajoutez un attribut `bool _isSurface` à la classe `BaseLight` ainsi que son getter (l'attribut devra être égal à `true` pour toutes les sources surfaciques).
2. Modifiez la méthode `DirectLightingIntegrator::Li` pour lancer `_nbLightSamples` rayons d'ombrage en cas de source surfacique (`_nbLightSamples` étant un attribut de votre intégrateur). Pour chaque point observé, l'éclairage correspondra à la moyenne des `_nbLightSamples` contributions lumineuses. La Figure 4 montre des résultats pour plusieurs valeurs de `_nbLightSamples`.

(a) (b) (c) (d)

FIGURE 4 – Résultat de l'exercice 3

3. Réactivez l'anti-aliasing et comparez les résultats. Pourquoi y a-t-il moins de bruit dans l'ombre ?

4 D'autres sources... (Optionnel)

Vous pouvez ajouter des sources lumineuses avec d'autres formes ! Il suffit de dériver `BaseLight` comme vous l'avez fait pour le quad. Pas plus de détails dans cet exercice ! L'objectif est que vous cherchiez la théorie, que vous la compreniez puis que vous l'implémentiez dans le moteur fourni. Allez je suis gentil, voici une piste : https://pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Light_Sources.

Références

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms, dec 2003.