Raytracing - TD 03

Éclairage

L'objectif de ce TD est d'ajouter des lumières à votre scène et de calculer leur influence sur les objets intersectés par les rayons.

Exercice 01 – Structure pour les lumières

Vous allez avoir besoin de représenter des lumières dans votre programme.

A faire :
01. Dans le fichier *raytracer.h* , créez une structure `Light` , représentant une lumière. Une lumière sera définie par une position dans l'espace et une couleur.

02a. Toujours dans *raytracer.h* , ajoutez une fonction `createLight` , prenant en paramètre une position et une couleur, et retournant une `Light` .
02b. Implémentez cette fonction dans *raytracer.c* .

03. Dans votre structure `Scene` , ajoutez un tableau (de taille fixe pour le moment, ex : 10) permettant de stocker des `Light` , et un compteur pour connaître le nombre de lumières. (Pensez à initialiser ce compteur lors de la création de la scène.)

04. Ajoutez et implémentez la fonction `addLightToScene(Scene* scene, Light l);` permettant d'ajouter une lumière dans votre scène.

Exercice 02 : Structure pour les matériaux

Pour calculer les effets d'une lumière sur un objet, il est nécessaire de connaître le matériau constituant cet objet et certaines propriétés qui lui sont associées.

Pour ce TD, nous chercherons à traiter le cas des réflexions diffuses et spéculaires.

Jusqu'ici vous n'aviez besoin de ne donner qu'une seule couleur à vos objets et intersections, il vous faudra par suite manipuler deux composantes de couleur : diffuse et spéculaire, auxquelles s'ajoutent un coefficient de brillance.

Par conséquent il est nécessaire de repenser la représentation des couleurs, et la manière dont elle s'applique aux objets. Pour ce faire, nous introduisons la notion de matériau.

A faire :

01. Dans un nouveau fichier *shading.h* , crée une structure `Material` .
Cette structure devra contenir des attributs pour une couleur diffuse et une couleur spéculaire, ainsi qu'un attribut `shininess` correspondant à la brillance de la surface.

02a. Toujours dans *shaing.h* , ajoutez une fonction `createMaterial` .
02b. Implémentez cette fonction dans *shading.c* .

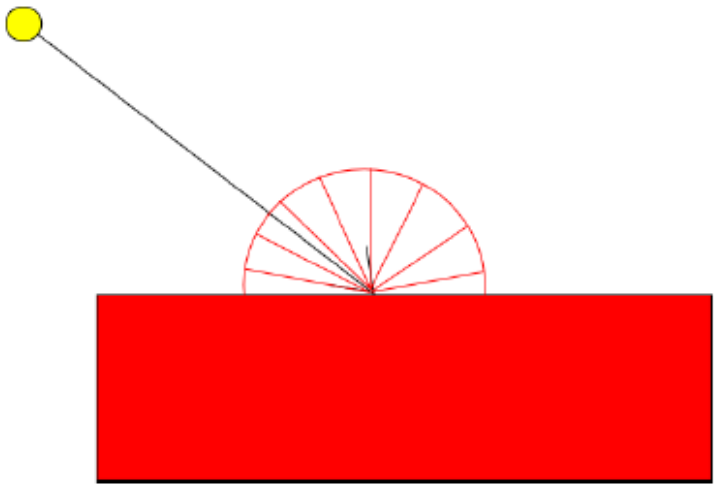
03a. Modifiez vos structures `Intersection` et `Sphere` pour qu'elles utilisent un `Material` en lieu et place d'une `ColorRGB` .
03b. De même, modifiez les fonctions de votre raytracer qui utilisent où créent des `Color` , pour qu'elles manipulent des `Material` à la place.

Exercice 03a : Modèle de Lambert – Réflexion diffuse

L'étude des échanges d'énergie lumineuse entre surfaces relève d'une branche de la physique nommée radiométrie. Plusieurs modèles ont étés établis afin de décrire ces interactions.

Nous allons ici commencer par implémenter un modèle simple, voire extrêmement basique, ne prenant en compte que les réflexions diffuses de la lumière sur les surfaces des objets.

Une surface est dite diffuse si elle renvoie de la lumière dans toutes les directions :



De ce fait, quelque soit votre position, vous verrez un point de l'objet de la même manière. La couleur ne dépend uniquement de la position de l'objet par rapport à la source lumineuse.

Pour afficher la couleur d'un objet en suivant le modèle de Lambert,
vous allez devoir faire un calcul qui dépend :
- de la lumière
- de la position de l'intersection
- de la couleur diffuse au point d'intersection
- de la normale au point d'intersection (i.e. vecteur unitaire orthogonal à la surface)

Vous devez par conséquent apporter quelques modifications pour déterminer la normale lorsque vous effectuez vos tests d'intersection.

A faire :

01a. Ajoutez le champ `normal` à votre structure `Intersection`.

01b. Modifiez votre fonction `intersectsSphere` (et optionnellement `intersectsCube`) de manière à calculer la normale dans le nouveau champ de la structure `Intersection`. Attention : la normale sera différente si l'origine du rayon est à l'intérieur de la `Sphere`.

01c. Modifiez également la fonction `throwRayThroughScene` pour remplir correctement les champs de l'`Intersection` passée en paramètre.

Votre programme est désormais prêt pour accueillir le calcul de la composante diffuse.
Pour chacune de vos `Light`, le calcul de la couleur du pixel est le suivant :

```
Ic * Lc * dot(N, normalize(IL)) / norm(IL) ^ 2
```

avec :
- `Ic` : couleur diffuse au point d'intersection
- `Lc` : couleur de la lumière
- `N` : vecteur normal au point d'intersection
- `IL` : vecteur reliant la position d'intersection et la position de la lumière

Si il y a plusieurs lumières, il suffit d'additionner leurs résultats (contributions) une par une, pour obtenir la couleur finale du pixel.

A faire :

02. Créez une fonction `void lambertRaytracing(Scene scene,SDL_Surface* framebuffer)`, similaire à la fonction `simpleRaytracing`, mais qui à chaque intersection détectée, itère sur les `Light` présentes dans la scène pour calculer la couleur finale d'un pixel.

03a. Dans *colors.h* et *colors.c* , ajoutez une fonction `ColorRGB clampColor(Color3f c)`, qui retourne une couleur avec ses 3 composantes clampées entre 0 et 1.

03b. Dans votre fonction `lambertRaytracing`, une fois la couleur finale du pixel obtenue, clampez cette couleur et servez vous en comme paramètre de la fonction `PutPixel`.

04. Dans votre fichier *main.c* , créez une scène avec :
- une sphère de rayon 1,5 à la position (0, 0, -5) et de couleur diffuse (1, 0, 0)
- une lumière située en (0, 0, 0) et de couleur (10, 10, 10)

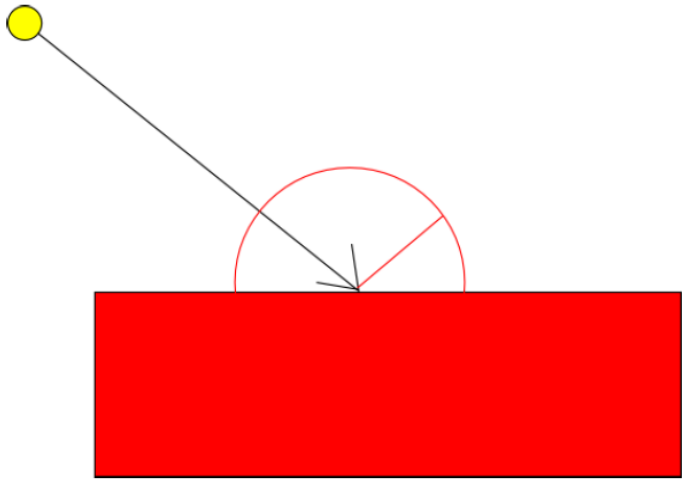
Optionnel :

05. Ajoutez d'autres sphères et lumières a votre scène et observez les résultats.

Exercice 03b : Modèle de Phong – Réflexions spéculaires

Le modèle de Lambert permet de distinguer la forme d'un objet en 3D.
Néanmoins il ne permet pas de faire le rendu d'objets brillants, tel qu' vase en porcelaine.
Le modèle de Phong va vous permettre de palier à cela.

Les surfaces dites spéculaires, telles qu'un miroir ou de l'eau, renvoient la lumière dans une direction privilégiée par rapport à la source lumineuse, contrairement à la réflexion diffuse.



La direction privilégiée est le symétrique de la direction d'incidence par rapport à la normale.

Dans le cas d'une réflexion spéculaire pure (ex : miroir), la lumière est totalement réfléchie selon cette direction privilégiée. Cependant, il existe de nombreux matériaux brillants, mais sans jusqu'à être purement spéculaires. Pour ces derniers, on utilise le modèle de Phong qui associe une composante diffuse à une composante spéculaire :

```
Ic * Lc * dot(N, normalize(IL)) / norm(IL) ^ 2
   + Ispec * Lc * specFactor / norm(IL) ^ 2
```

avec :
- `Ic` : couleur diffuse au point d'intersection
- `Lc` : couleur de la lumière
- `N` : vecteur normal au point d'intersection
- `IL` : vecteur reliant la position d'intersection et la position de la lumière
- `Ispec` : couleur spéculaire au point d'intersection

- `specFactor` : facteur spéculaire calculé en fonction de l'angle de réflexion de la lumière avec la normale, et de la position de l'intersection par rapport à la caméra.

```
specFactor = max(0, dot(R, V) ^ shininess
```

avec :
- `shininess` : coefficient de brillance au point d'intersection
- `R` : vecteur directeur de la lumière, réfléchi par rapport à la normale au point d'intersection
- `V` : position de l'intersection par rapport à la caméra (i.e. view direction)

A faire :

01. Dans vos fichiers *geometry.h* et *geometry.c* , implémentez la fonction `Vector3D reflect(Vector3D v, Vector3D n)` , qui calcule `R` , le vecteur `V` réfléchi par rapport au vecteur `N` . Le calcul du vecteur réfléchi est le suivant :

```
R = V – N * 2.0 * dot(V, N)
```

02. Implémentez la fonction `void phongRaytracing(Scene scene,SDL_Surface* framebuffer)` , similaire à la fonction lamber`Raytracing` , mais qui calcule la couleur finale du pixel en fonction de ses composantes spéculaires et diffuses.

Exercice 04 : Ombres basiques

A faire :

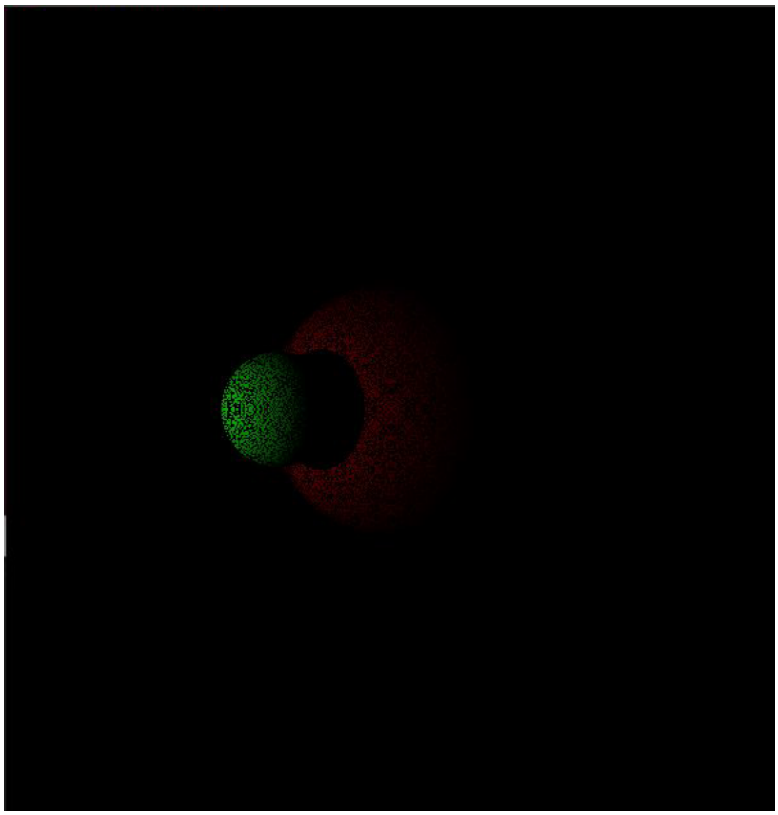
01. Construisez une scène composées de deux sphères : une rouge en (0, 0, -5) de rayon 1.5, et une verte en (-1 0, -3.5) de rayon 0.5. Placez y une lumière de couleur (5, 5, 5) en (-3, 0, 1).

En lançant votre programme sur cette scène, vous constaterez à coup sur que cette dernière manque d'une ombre projetée par la sphère verte sur la sphère rouge.

La gestion des ombres est assez simple implémenter dans un raytracer, même si quelques précautions doivent être prises. Basiquement, pour savoir si un point d'intersection est éclairé dans une ombre projetée, il suffit de lancer un rayon depuis le point d'intersection vers la source lumineuse pour laquelle veut déterminer la contribution en ce point.
Si ce rayon intersecte un objet de la scène avant d'atteindre la source lumineuse, alors le point d'intersection n'est pas (directement) éclairé par cette dernière.

02. Dans vos fonctions `simpleRaytracing` , `lambertRaytracing` et `phongRaytracing` , ajoutez les quelques lignes de code qui permettent d'ignorer la contribution d'une lumière si le point d'intersection est dans une ombre projetée par cette dernière.

Vous devriez obtenir le résultat suivant :



Question :

03. Quelle peut être la provenant de ces artefacts ? (i.e. shadow acne / noise)

A faire :

04. Proposez et implémentez une solution permettant d'éviter ou limiter leur apparition.