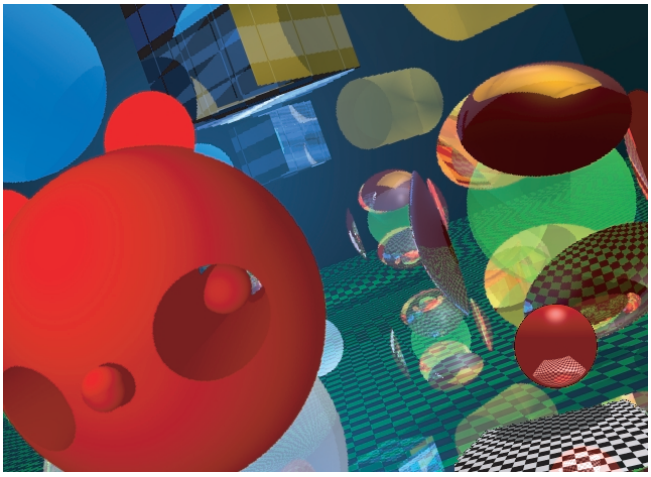
Université de REIMS
Champagne-Ardenne
UFR de Sciences
Exactes et Naturelles

Moteur de Ray-Tracing

Projet - Module Synthèse d'image 2

GRUSON Sébastien / RABAT Cyril



Responsable : P. Mignot

Maîtrise d'Informatique
Synthèse d'image 2
2002/2003

Table des matières

Table des figures

Introduction

Ce projet a pour but de réaliser un moteur de Ray-Tracing, c'est à dire un programme capable d'afficher une scène composée d'objets simples (sphères, plans, ...) en couleur. Pour cela, on "lance un rayon" depuis l'observateur vers chaque point visible de la scène et on évalue, pour ceux-ci, la luminance observée sur les différents objets rencontrés.

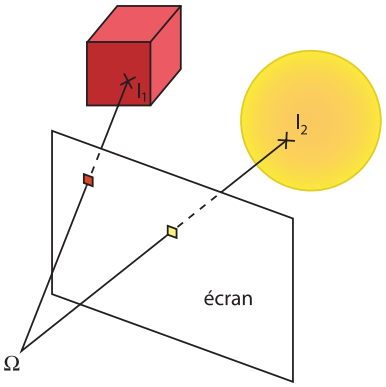
Tout d'abord, le moteur devra lire la scène à réaliser dans un fichier de configuration donné puis il effectuera les calculs nécessaires pour finalement produire une image enregistrée au format *ppm*.

Les différentes fonctionnalités gérées sont :
- les objets : sphères, plan infinis et contraints, cylindres, slabs, facettes
- les objets CSG (compositions d'objets),
- la couleur suivant le modèle RGB
- le traitement de l'aliasing avec l'utilisation de filtres gaussien ou l'échantillonnage aléatoire de type jittered
- les lumières : sources ponctuelles multiples et spots, gestion des ombres, reflets et transparence
- gestion des ombres colorées
- la texture de type damier pouvant être appliquée sur tous les types d'objets

Outre le moteur de Ray-Tracing, un moteur de Ray-Casting a été ajouté (même procédé que le Ray-Tracing mais sans les lumières) ainsi que des options permettant d'effectuer une égalisation et un recadrage de l'image de sortie si nécessaire (pour régler les éventuels problèmes de saturation) et, en parallèle, une application permettant de gérer le moteur en réseau (pour diminuer les temps de calcul).

1 Principe général

Un moteur de Ray-tracing permet d'effectuer un rendu en image de synthèse d'une scène constituée d'objets en envoyant des *rayons* dans la scène. Lorsqu'un rayon touche un des objets, il retourne la couleur de celui-ci.



Afin d'ajouter du réalisme, au contact d'un objet, chaque rayon peut être renvoyé dans une autre direction. Ceci dépend de la *récursivité* que l'on a choisi et cela permet de simuler la réflexion, la réfraction, la transparence, etc...

Dans le moteur, on différencie les composants suivants :

- la caméra : définie par sa position, son angle de vision et son sens de direction,
- la scène, constituée :
  - d'objets caractérisés par leur équation géométrique, leur couleur et leur matière,
  - des lumières caractérisées par leur couleur, leur intensité et leur direction.
- l'image finale : ce n'est autre qu'un tableau de couleurs à deux dimensions qui est rempli au fur et à mesure.

2 Opérations de pré-traitements

2.1 Lecture du fichier de configuration

On y trouve toutes les informations dont le moteur a besoin pour calculer la scène. Cela permet de régler tous les paramètres de la caméra mais on y trouve aussi tous les objets qui composent la scène. Le nom de fichier est donné en paramètre lors du lancement du programme (par défaut, c'est le fichier *config.cfg*) et lu par la fonction `charge` située dans *moteur.cpp*.

Le fichier de configuration est constitué d'une liste de mots-clés et de paramètres divers (voir la liste complète en annexe au chapitre **??** p. **??**). Ces mots-clés se rangent en trois catégories :

- Les options globales : elles influent directement sur le rendu total comme le réglage de la récursivité, de la taille de l'image, etc...
- Les objets : ils constituent la scène globale et on doit préciser leur caractéristiques (le centre et le rayon pour une sphère, par exemple),
- Les propriétés : elles affectent les objets comme la couleur, les propriétés matérielles ou bien la texture,
- Les lumières : que ce soient la lumière ambiante, les spots ou les lumières ponctuelles,
- Les réglages de la caméra : position, angle de vision, ...

Tous ces mots clés, suivis de leur paramètres, sont mis dans n'importe quel ordre dans le fichier de configuration, sauf les propriétés. En effet, cela fonctionne comme des états. Une couleur, par exemple, est un état. Si on déclare une couleur bleue, alors tant que l'on ne redéfinit pas une autre couleur, tous les objets que l'on crée après sont bleus.

2.2 Réglages de la caméra

Par défaut, l'utilisateur doit donner :

- la position de la caméra Ω,
- la direction du regard $\overrightarrow{U}$,
- la direction approximative du haut $\overrightarrow{V'}$,
- la largeur du champ de vision φ.

Avant de commencer le rendu, il est essentiel de calculer le repère de la caméra. On doit tout d'abord calculer la direction du haut $\overrightarrow{V}$ qui se trouve avec cette formule :

$$\overrightarrow{V} = \overrightarrow{V'} - (\overrightarrow{U}.\overrightarrow{V'}).\overrightarrow{U}$$

Il reste à calculer le troisième vecteur du repère : $\overrightarrow{W}$. On a alors la formule suivante :

$$\overrightarrow{W} = \overrightarrow{V} \wedge \overrightarrow{U}$$

Une fois que l'on a le repère de la caméra, on doit déterminer les différents composants de la grille :

- le rayon de l'image : on le fixe arbitrairement à 1,
- la distance focale d : on sait que $tan(\frac{\varphi}{2}) = \frac{r}{d}$. On fixe alors $r = 1$ et on trouve $d = \frac{r}{tan(\frac{\varphi}{2})}$.

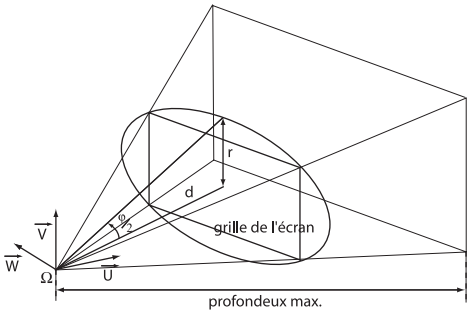


FIG. 1: Le modèle de la caméra

3 Le lancer de rayon

3.1 Algorithme

Tous les rayons partent de la caméra vers la scène. Le centre de tous les rayons est donc Ω. Par contre, leur direction dépend du pixel de la grille que l'on veut calculer. On a le schéma suivant :

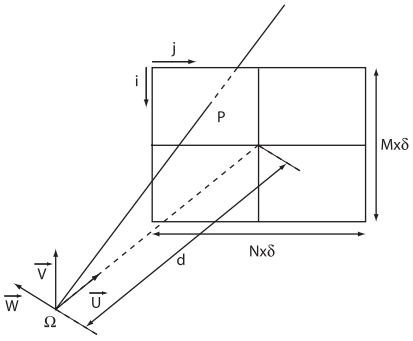


FIG. 2: Calcul d'un rayon

Le δ est la taille d'un carré de la grille. Il est égal à : $\delta = \frac{2.r}{min(M,N)}$, où M est la taille horizontale de la grille et N la taille verticale.
On veut calculer le point (i, j) de l'image. La direction du rayon est donc :

$$\overrightarrow{\Omega.P} = d\overrightarrow{U} + \delta(\frac{M}{2} - i)\overrightarrow{V} + \delta(\frac{N}{2} - j)\overrightarrow{W}$$

Ainsi, pour remplir toute la grille, on a l'algorithme suivant :

```
fonction raytracing()
Pour i=0 à MAXX faire
    Pour j=0 à MAXY faire
        pile.empile(1.0002926) ; // On empile le milieu global qui est l'air
        A ← Ω ;
        u⃗ ← d.U⃗ + V⃗.δ(MAXX/2 − i) + W⃗.δ(MAXY/2 − j) ;
        coul ← calcul_luminance(r,pile)
        image[i,j] ← coul ;
    Fin pour
Fin pour
```

3.2 Gestion des calculs d'intersections

3.2.1 Objets simples

Description :

Les objets simples implémentés sont les suivants :

- **Le plan :** Il est construit avec un point, une normale et trois paramètres pour les contraintes. Les contraintes permettent de créer des plans contraints en fonctions de l'un des trois axes, voire de deux ou de trois. Si une des contraintes est mise à 0, le plan est infini dans cet axe.

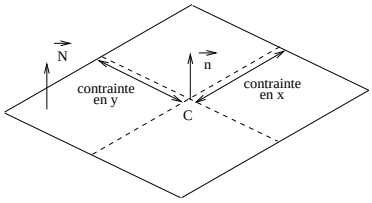


FIG. 3: Le plan

- **La sphère :** L'objet le plus simple, une sphère n'est constituée que d'un centre et d'un rayon.

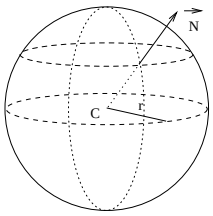


FIG. 4: La sphère

- **Le cylindre :** Pour définir un cylindre, il faut fixer un centre, une hauteur, un rayon et une direction. On remarquera que tous les cylindres sont forcement finis et les plans qui les coupent sont parallèles.
- **Les slabs :** Ce sont des unions de plans parallèles. Pour en définir un, on doit préciser son centre, le nombre de couple de plans et pour chacun des couples, la normale d'un des plans ainsi que la distance qui sépare les deux plans.
- **Les facettes :** On doit préciser les trois points de l'espace qui constituent la facette. On remarque qu'il est essentiel de faire attention à l'ordre dans lequel on met les points puisque cela joue sur la normale.

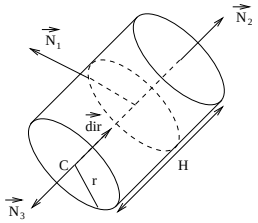


FIG. 5: Le cylindre contraint par deux plans parallèles

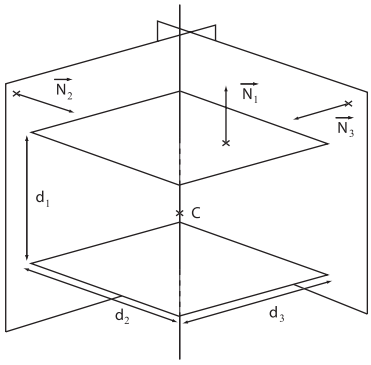


FIG. 6: Le slab

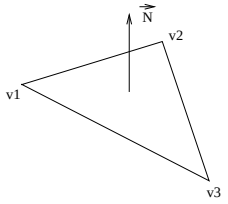


FIG. 7: La facette

Le calcul des intersections :

Pour chaque objet, la fonction `inter` permet de calculer l'intersection entre l'objet et le rayon. Le retour est donc un entier a tel que : $P = C + a.\overrightarrow{r}$ où C est le centre du rayon, $\overrightarrow{n}$ est sa direction. Le point P est le point d'intersection dans le repère global.

La fonction prend un second paramètre. Celui-ci n'est utilisé que pour les slabs ou les cylindres finis. En effet, il indique sur quelle face de l'objet se trouve le point d'intersection.

Le calcul de la normale :

Lorsque l'on cherche les intersections, la normale n'est pas calculée en même temps. Cela évite de faire des calculs superflus puisque l'on trouvera plusieurs intersections dont une seule sera utilisée, et donc, un seul calcul de normale sera nécessaire.

La fonction `normale_surface` permet de calculer la normale. Elle prend comme paramètres le point en lequel la normale doit être calculée et un second paramètre qui est fourni par la fonction `inter` et qui correspond au numéro de la face de l'objet touché.

En effet, pour les objets simples tels que la sphère ou le plan, quelque soit l'intersection, la normale sera calculée toujours de la même manière. Par contre, pour les slabs ou les cylindres, en fonction de la surface d'intersection, la normale sera différente.

3.2.2 Objets CSG

Description :

Les objets CSG sont des compositions d'autres objets : unions, intersections et différences. Ces objets peuvent être ceux décrits dans la section précédente mais aussi des objets CSG. Pour simplifier, ils peuvent se représenter sous forme d'arbres binaires. Voici un exemple avec l'arbre associé :

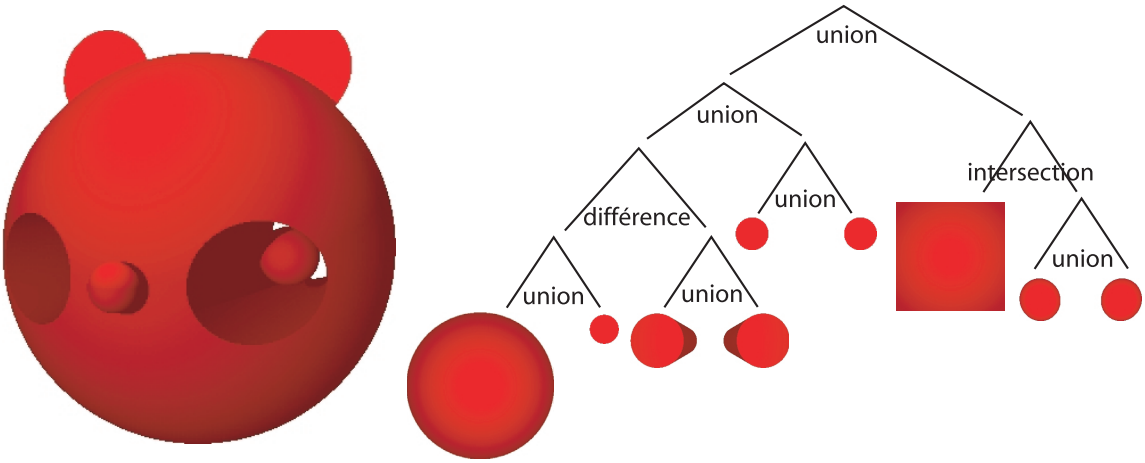


FIG. 8: Un CSG avec son arbre binaire associé

Il est évident que la matière et la couleur du CSG est identique dans tous les objets qui le compose. Il serait difficile de dire quel serait le milieu de l'union d'une sphère en verre rouge et d'une sphère d'eau verte.

Calcul des intersections :

Afin d'implémenter les CSG, il a fallut modifier les calculs d'intersections des objets simples. En effet, il faut traiter des listes d'intersections au lieu d'intersections simples.

Une liste d'intersection d'un objet est la liste de toutes les intersections avec un rayon donné. Ces intersections sont des réels, mais ne sont pas nécessairement positives. Lorsque l'on calcule une intersection d'un rayon avec un CSG, il s'agit en fait, d'une manipulation de ces listes.
L'algorithme pour une union est le suivant :

---

liste li ; *// Liste d'intersection temporaire*
booléen entre1=FAUX, entre2=FAUX ; *// Pour savoir si on est dans l'objet 1 ou l'objet 2*
entier i ;

li ← obj1.interl(r) ; *// On met les intersections de l'objet 1 dans la liste*
li ← obj2.interl(r) ; *// On met les intersections de l'objet 2 dans la liste*
// La liste obtenue est rangée par ordre croissant

Si li.taille()=0 **alors**
  pas d'intersection ;

i ← 1 ;

// On recherche la première intersection positive (donc devant le rayon)
// On change aussi les valeurs des booléens (pour savoir si on est dans un objet)
Tant que (i<li.taille())∧(li[i]<0) **faire**

```
    Si (li[i]=objet1) alors
        entre1 ← ¬ entre1 ;
    sinon
        entre2 ← ¬ entre2 ;
    i ← i + 1 ;
fin tant que

// On ignore toutes les intersections comprises à l'intérieur de l'un des deux objets
Tant que (i≤ li.taille())
    Si (entre1=FAUX) ∧ (entre2=FAUX)
        retourne(li[i]) ;
    Si (li[i]=objet1)
        entre1 ← ¬ entre1 ;
    Sinon
        entre2 ← ¬ entre2 ;
    Si (entre1=FAUX) ∧ (entre2=FAUX)
        retourne(li[i]) ;
    i ← i + 1 ;
fin tant que
```

Pour le cas de l'intersection et de la différence, on ne change uniquement que les tests avec $entre1$ et $entre2$:

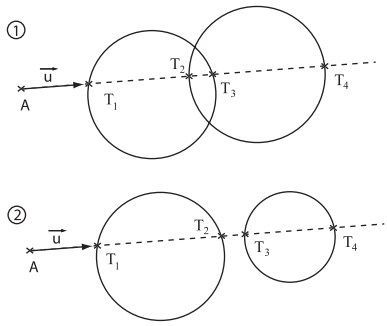
- pour l'intersection, on ignore toutes les intersections extérieures des objets, donc $entre1 = VRAI$ et $entre2 = VRAI$.
- pour la différence, on ignore toutes les intersections du premier objet qui se trouve entre deux intersections du second objet, donc $entre1 = VRAI$ et $entre2 = FAUX$.

En résumé, on peut rencontrer différents cas. Sur le schéma 1, on lance un rayon du point A dans la direction $\overrightarrow{v}$. En fonction des objets CSG, on trouvera comme intersections :

- Union : intersections $\in \{T_1, T_4\}$
- Intersection : intersections $\in \{T_2, T_3\}$
- Différence : intersections $\in \{T_1, T_2\}$

Pour le deuxième schéma, par contre, on trouvera :

- Union : intersections $\in \{T_1, T_2, T_3, T_4\}$
- Intersection : intersections $\in \emptyset$
- Différence : intersections $\in \{T_1, T_2\}$



Calcul de la normale :

Comme on a vu pour les objets simples, on utilise la fonction `normale_surface` qui prend comme paramètre un point et un numéro de face, pour calculer la normale. Seulement, pour les CSG, il faut savoir sur quel objet on doit faire le calcul.

On utilise alors une numérotation binaire : l'objet 1 est nommé 1 et l'objet 2 est nommé 2. Lorsque l'on descend d'un niveau dans l'arbre, on multiplie ce nombre par 10. Les objets seront alors appelés : 11 et 12 pour ceux de gauche, 21 et 22 pour ceux de droite. Cette numérotation permet de déterminer n'importe quel objet du CSG de manière unique et surtout, permet de retrouver très simplement un objet. Pour cela, il suffit de partir de la racine et le premier chiffre du numéro détermine la direction.

On a donc modifié la fonction `inter` qui retourne le paramètre de la face, mais aussi le numéro de l'objet. Bien entendu, pour les objets simples, le numéro de l'objet sera inutile.

3.3 Calcul de la luminance

C'est le corps principal du moteur. C'est dans cette fonction (`calcul_luminance` dans *moteur.cpp*) qu'on cherche les intersections entre un rayon et tous les objets. On doit calculer la couleur au point d'intersection : la luminance.

On utilise dans le moteur, le modèle de Hall. La luminance est la somme de plusieurs contributions :

- la contribution de l'objet en lui-même (sa couleur, ses propriétés matérielles, sa texture, etc...),
- la contribution de la lumière ambiante L_r^a,
- la contributions des lumières ponctuelles de la scène,
- la contributions des autres objets de la scène (par réflexion, réfraction, transparence, etc...).

La formule générale est la suivante :

$$L = L_e + L_r^a + \underbrace{\sum_j (L_{r,d}^j + L_{r,s}^j + L_{t,s}^j + L_{t,d}^j)}_{\text{contribution des sources ponctuelles}} + \underbrace{\sum_O (L_r^O + L_t^O)}_{\text{contribution des autres objets}}$$

avec :

- L_e : lumière émise par l'objet. Elle vaut toujours 0.
- L_r^a : contribution de la lumière ambiante,
- $L_{r,d}^j$: contribution de la j^{ime} source de lumière ponctuelle visible par réflexion diffuse,
- $L_{r,s}^j$: contribution de la j^{ime} source de lumière ponctuelle visible par réflexion spéculaire,
- $L_{t,d}^j$: contribution de la j^{ime} source de lumière ponctuelle visible par réfraction diffuse,
- $L_{t,s}^j$: contribution de la j^{ime} source de lumière ponctuelle visible par réfraction spéculaire,
- L_r^O : contribution du O^{ime} objet par réflexion,
- L_t^O : contribution du O^{ime} objet de l'autre côté de la surface.

Toutes ses contributions sont détaillées dans les paragraphes suivants.

3.3.1 Contributions de la lumière ambiante

La lumière ambiante est définie par une couleur C_a et une intensité I_a [1]. La contribution de la lumière ambiante est identique, quelque soit l'endroit de la scène et de l'objet.
Elle se calcule, suivant le modèle de Hall, comme suit :

$$L_r^a = k_d.L_a.\rho_d$$

avec :

- L_a : la luminance qui vaut $I_a.C_a$,
- k_d : le coefficient de réflexion diffuse,
- ρ_d : la couleur de l'objet (voir la section 3.3.7 p. 13).

3.3.2 Contribution des lumières ponctuelles par réflexion

Soit un rayon avec son centre Ω, sa direction $\overrightarrow{u}$ (voir figure 9). Il coupe un objet au point P et on note $\overrightarrow{V}$ son vecteur d'arrivée. Il possible alors de calculer la normale à la surface $\overrightarrow{N}$. La lumière

[1] En fait, comme l'intensité n'est utilisée qu'avec la couleur, on multiplie ces deux composantes dès le début.

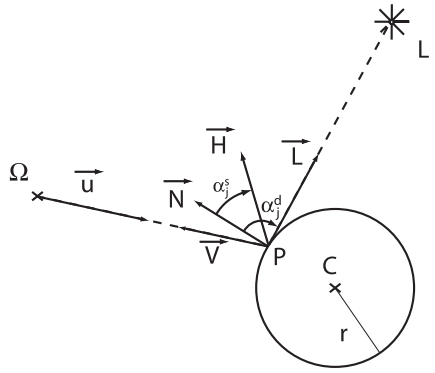


FIG. 9: La réflexion sur un objet

ponctuelle L_j a pour centre L, on calcule alors sa direction $\overrightarrow{L}$. Pour calculer la contribution par réflexion, on a besoin aussi de $\overrightarrow{H}$ qui est la normale à la micro-facette que l'on trouve par :

$$\overrightarrow{H} = \frac{\overrightarrow{L} + \overrightarrow{V}}{|\overrightarrow{L} + \overrightarrow{V}|}$$

C'est le milieu des deux vecteurs $\overrightarrow{L}$ et $\overrightarrow{V}$ que l'on normalise.

On a deux types de réflexion : la réflexion diffuse et la réflexion spéculaire.

La réflexion diffuse

La contribution des lumières ponctuelles par réflexion diffuse, pour le modèle de Hall, se calcule comme suit :

$$L^j_{r,d} = s^j.k_d.L_j.\rho_d.\cos\alpha^d_j$$

avec :

- s^j : terme de visibilité toujours égal à 1,
- k_d : coefficient de réflexion diffuse,
- L_j : luminance de la lumière ponctuelle j (voir section 3.3.4 ci-après),
- ρ_d : couleur de l'objet (voir la section 3.3.7 p. 13),
- α^d_j : c'est l'angle entre $\overrightarrow{N}$ et $\overrightarrow{L}$.

La réflexion spéculaire

La contribution des lumières ponctuelles par réflexion spéculaire se calcule comme suit :

$$L^j_{r,s} = s^j.k_s.L_j.\rho'(\alpha^d_j).(\cos\alpha^s_j)^{n_s}$$

avec :

- k_s : coefficient de réflexion spéculaire,
- $\rho'(\alpha^d_j)$: fonction de Fresnel (réflectance) non implémentée ici donc ce coefficient vaut toujours 1,
- α^s_j : c'est l'angle entre $\overrightarrow{N}$ et $\overrightarrow{H}$,
- n_s : exposant de réflexion spéculaire.

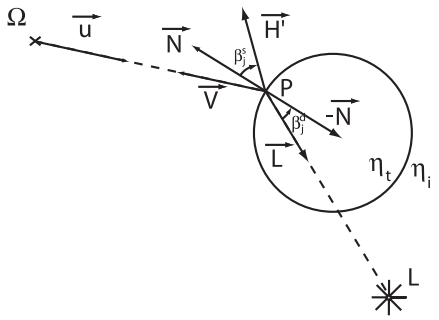


FIG. 10: La réfraction sur un objet

3.3.3 Contribution des lumières ponctuelles par réfraction

Pour calculer la contribution par réflexion, on a besoin de $\overrightarrow{H'}$ qui est la normale à la micro-facette que l'on trouve par :

$$\overrightarrow{H'} = \frac{\overrightarrow{L} + \eta.\overrightarrow{V}}{\eta - 1}, \text{ avec } \eta = \frac{n_t}{n_i}$$

Comme pour la réflexion, on trouve la réfraction diffuse et la réfraction spéculaire.

La réfraction diffuse

La contribution des lumières ponctuelles par réfraction diffuse, pour le modèle de Hall, se calcule comme suit :

$$L^j_{t,d} = s^j.k^t_d.L_j.\rho^t_d.\cos\beta^d_j$$

avec :
- k^t_d : coefficient de réfraction diffuse,
- ρ^t_d : la couleur par transparence de la surface,
- β^d_j : c'est l'angle entre $\overrightarrow{L}$ et $-\overrightarrow{N}$.

La réfraction spéculaire

La contribution des lumières ponctuelles par réfraction spéculaire, pour le modèle de Hall, se calcule comme suit :

$$L^j_{t,s} = s^j.k_t.L_j.\rho'(\beta^d_j).(\cos\beta^s_j)^{n_t}$$

avec :
- k_t : coefficient de réfraction spéculaire,
- $\rho'(\beta^d_j)$: fonction de Fresnel (réfringence avec $\rho'_t = 1 - \rho'$) non implémentée ici donc ce coefficient vaut toujours 1,
- β^s_j : l'angle entre $\overrightarrow{N}$ et $\overrightarrow{H'}$,
- n_t : exposant de réfraction spéculaire.

3.3.4 Calcul de la luminance pour les lumières ponctuelles

Lumières ponctuelles :

Les lumières ponctuelles sont des sources disséminées dans la scène. Tout comme la lumière ambiante, elles ont une position (dans le repère global), une intensité I_a et une couleur C_a.

Comme pour la lumière ponctuelle, La luminance est tout simplement : $I_a.C_a$.

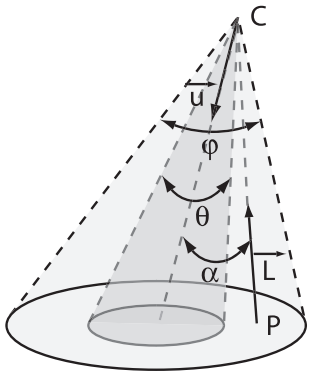
Spots :

Les spots sont des lumières ponctuelles mais directionnelles.

Elles ont un centre C, une direction $\overrightarrow{u}$ et sont définies par deux angles Φ et θ, un exposant p de directionnalité de la source, ainsi que par leur intensité I_a et leur couleur C_a. Les deux angles permettent de simuler le comportement de halo lumineux.

Sur la figure ci-contre, $\overrightarrow{L}$ est le vecteur de direction de la source, autrement dit la direction de la lumière vue depuis le point d'intersection P. On mesure alors l'angle α qui est l'angle entre $\overrightarrow{L}$ et $\overrightarrow{u}$. On a alors trois cas :

- Si $\alpha > \Phi$, la contribution du spot est nulle,
- Si $\alpha < \theta$, la contribution du spot est maximale, autrement dit $C_a.I_a$,
- Si $\Phi < \alpha < \theta$, la contribution diminue progressivement plus θ se rapproche de Φ.



La luminance du spot se calcule donc ainsi :

$$L_a = C_a.I_a.\left[min\left(1, max\left(0, \frac{\cos\alpha - \cos\Phi}{\cos\theta - cos\Phi}\right)\right)\right]^p$$

3.3.5 Contribution des autres objets

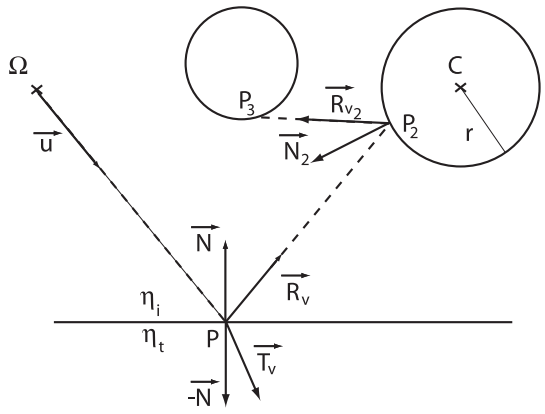


FIG. 11: La récursivité des rayons

Lorsqu'un objet est réfléchissant ou transparent, il est possible de voir les autres objets dedans. En Ray-Tracing, ceci se fait naturellement en envoyant un nouveau rayon dans la direction de réflexion et un autre dans la direction de réfraction, ceux-ci pouvant créer à leur tour d'autres rayons : c'est la récursivité. Elle est réglée une fois pour toute et permet de déterminer le nombre maximum de rayons renvoyés. Évidemment, plus elle est importante, plus il y a de calculs à faire, surtout dans le cas des objets transparents.

Les deux vecteurs que l'on a besoin sont :

- $\overrightarrow{R_v}$: le vecteur réfléchi

$$\overrightarrow{R_v} = 2.\overrightarrow{N}(\overrightarrow{N}.\overrightarrow{L}) - \overrightarrow{L}$$

– $\overrightarrow{T_v}$: le vecteur réfracté

$$\overrightarrow{T_v} = \frac{1}{\eta}\left[\left(\overrightarrow{N}.\overrightarrow{L} - \sqrt{\eta^2 - 1 + (\overrightarrow{N}.\overrightarrow{L})^2}\right).\overrightarrow{N} - \overrightarrow{L}\right], \text{ avec } \eta = \frac{n_t}{n_i}$$

Contribution dans la direction de la réflexion

On la calcule suivant la formule :

$$L_r^O = s^s.k_s.L(\overrightarrow{R_v}).\rho'(\theta_i).(T_s)^{\delta_r}$$

avec :
- s^s : terme de visibilité toujours égal à 1,
- $L(\overrightarrow{R_v})$: la luminance dans la direction $\overrightarrow{R_v}$,
- T_s : transmitivité du milieu dans la direction $\overrightarrow{R_v}$,
- δ_r : la distance parcourue par le rayon.

Contribution des objets de l'autre côté de la surface

On la calcule suivant la formule :

$$L_t^O = s^t.k_t.L(\overrightarrow{T_v}).\rho'_t(\theta_i).(T_t)^{\delta_t}$$

avec :
- s^t : terme de visibilité toujours égal à 1,
- $L(\overrightarrow{T_v})$: la luminance dans la direction $\overrightarrow{T_v}$,
- T_t : transmitivité du milieu dans la direction $\overrightarrow{T_v}$,
- δ_t : la distance parcourue par le rayon.

3.3.6 Les ombres colorées (des objets transparents)

Les ombres ou les ombres colorées jouent directement sur la luminance d'une lumière. Pour déterminer si une lumière est visible depuis un point d'un objet, il faut d'abord savoir s'il n'y a pas d'objet opaque situé entre les deux. Si c'est le cas, la lumière n'est pas visible, on voit alors l'apparition d'ombres.

Pour les ombres colorées, on s'intéresse à la présence des objets transparents situés entre le point et la lumière. On envoie un rayon dans la direction de la lumière et on récupère toutes les intersections. La luminance de la lumière sera alors modifiée suivant la distance parcourue par le rayon dans les objets transparents et la couleur de transparence de ceux-ci.

L'algorithme est le suivant :

---

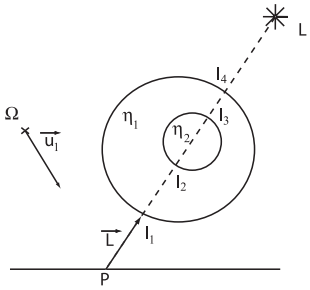
int obstacle(**rayon** r(A,$\overrightarrow{u}$),**float** d,**couleur** &c)

pile milieux ;
pile couleur ;
liste L_i ; *// La liste d'intersections*
booléen *enu*[nb_objets] ; *// Tableau pour savoir si on est en dehors ou dans un objet*

// Création de la liste d'intersection
Pour (i=0) **jusqu'à** nombre d'objets **faire** *// On parcourt toute la liste d'objets*
    $L_i \leftarrow L_i$ + inter(r,objets[i]) ;
    enu[i]←FAUX ; *// On initialise le tableau de booléens*
Fin pour
// On recherche la première intersection positive
i ← 0 ;
Tant que (i<L_i →taille ∧ L_i[i]→inter<0) **faire**

```
    Si (enu[L_i[i]→objet]=FAUX) alors // On entre dans un objet
        milieux→ empile(L_i[i]→milieu); // On empile le milieu de l'objet dans lequel on va entrer
        couleur→ empile(L_i[i]→couleur); // On empile la couleur de l'objet dans lequel on va entrer
        enu[L_i[i]→objet]=VRAI;
    Sinon
        milieux→dépile;
        couleur→dépile;
        enu[L_i[i]→objet]=FAUX;
    Fin si
Fin tant que
// On est bien devant le rayon
Tant que (i<L_i →taille) faire
    Si (L_i[i]→objet.k_t=0 ∧ L_i[i]→inter<d) alors // Il y a un objet opaque devant la lumière
        retourner 0;
    Fin si
    Si (L_i[i]→inter>d) alors // On est derrière la lumière
        retourner -1;
    Fin si
    Si (milieux ¬ vide) alors // On est dans un objet
        Si (L_i[i-1]→inter<0) alors // Première intersection positive + dans un objet ⇒ l'origine était dans un objet
            c ← c * couleur→tête() * milieux→tête()^{L_i[i]→inter};
        Sinon
            c ← c * couleur→tête() * milieux→tête()^{L_i[i-1]→inter − L_i[i]→inter};
        Fin si
    Fin si
    Si (enu[L_i[i]→objet]=FAUX) alors // On entre dans un objet
        milieux→ empile(L_i[i]→milieu); // On empile le milieu de l'objet dans lequel on va entrer
        couleur→ empile(L_i[i]→couleur); // On empile la couleur de l'objet dans lequel on va entrer
        enu[L_i[i]→objet]←VRAI;
    Sinon
        milieux→dépile;
        couleur→dépile;
        enu[L_i[i]→objet]←FAUX;
    Fin si
Fin tant que
retourner -1;
```

On voit sur la figure deux objets transparents (dont la couleur de transparence est C_1^t et C_2^t et une source de lumière L de luminance L_a . On lance un rayon de centre ω dans la direction $\overrightarrow{u}$. Il intersecte le plan au point P. On *lance* alors un nouveau rayon dans la direction $\overrightarrow{L}$. On trouve les intersections suivantes : $\{I_1, I_2, I_3, I_4\}$.



Le rayon parcourt les distances $[I_1, I_2]$ et $[I_3, I_4]$ dans l'objet 1 et les distances $[I_2, I_3]$ dans l'objet 2. La luminance de la lumière est modifiée alors avec :

$$L' = C_1^t.\eta_1^{d(I_1,I_2)}.C_2^t.\eta_2^{d(I_2,I_3)}.C_1^t.\eta_1^{d(I_3,I_4)}.L_a$$

3.3.7 La couleur des objets

La couleur des objets est définie au début, dans le fichier de configuration. Il est possible d'appliquer des textures sur les objets, ce qui change leur couleur.

Les textures procédurales déterministes

Il y a 2 modèles implémentés :

- type damier : ce type de texture prend 3 paramètres : t_x,t_y,t_z. Au point $P(x, y, z)$ sur la surface, la couleur est :

$$C_i \text{ avec } i = \frac{x.t_x + y.t_y + z.t_z}{n_c}, \text{ où } n_c = \text{nombre de couleurs de l'objet}$$

Les paramètres définissent la taille des carreaux. Si l'un est mis à 0, la direction correspondante est ignorée (pour faire des rayures, deux paramètres doivent être mis à 0),
- la distance aux axes : la texture dépend de la distance aux axes. Elle prend deux paramètres : t_1,t_2. Ainsi, par rapport à l'axe des z, la couleur au point $P(x, y, z)$ est :

$$C_i \text{ avec } i = \frac{\sqrt{x^2.t_1 + y^2.t_2}}{n_c}$$

Les deux paramètres indiquent la taille des rayures.

Le plaquage de textures

Le principe est de réussir à plaquer sur une surface, une image au format *ppm*. Il faut réussir à transformer des coordonnées 3D du repère local de l'objet en coordonnées 2D dans l'image. Pour cela, on utilise la fonction `coord_cart` qui réalise l'opération pour chaque objet. Pour la sphère, par exemple, on prend tout simplement les coordonnées cartésiennes.

Dans le matériel de l'objet, on charge l'image en mémoire au début, lors de la lecture du fichier de configuration. Ensuite, lorsque l'on trouve une intersection au point P, on utilise la fonction `coord_cart` qui va renvoyer les coordonnées (i, j) associées. La couleur est alors le pixel (i, j) dans l'image en mémoire.

3.4 Anti-aliasing

L'anti-aliasing permet d'éliminer le phénomène de lignes crénelées sur une image de synthèse. Le principe ici a été de choisir un suréchantillonnage sur chaque pixel. Au lieu d'envoyer un seul rayon par pixel, on en envoie plusieurs autour.

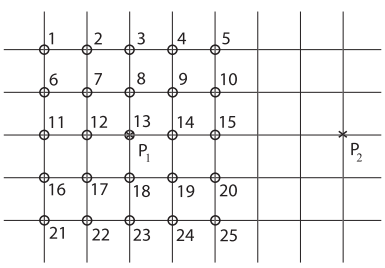
Deux types d'anti-aliasing ont été implémentées :
- le traitement par filtre gaussien,
- l'échantillonnage aléatoire de type jittered.

3.4.1 Les filtres gaussiens

Lorsque l'on choisi l'anti-aliasing de ce type, on a le choix entre plusieurs filtres différents :
- le filtre moyenneur : $\frac{1}{9} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- les trois filtres résultants de convolution de filtres :

$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \frac{1}{256} * \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \text{ et } \frac{1}{81} * \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Sur la figure ci-contre, on a les deux pixels P_1 et P_2. On choisit un filtre 5 par 5. On envoie alors 25 rayons autour de chaque pixel. Suivant le filtre choisi, on pondère plus ou moins certains rayons et on fait la moyenne. On sait que sur la grille, la distance qui sépare deux points est δ. Ainsi, pour calculer chaque rayon, il suffit de décaler de $\frac{\delta}{n_d}$ où n_d est le nombre de divisions pour un pixel (dans le cas d'un filtre 5 par 5, $n_d = 5$).



Les deux premiers filtres (ceux à neuf rayons) enlèvent beaucoup d'aliasing sur l'image. Les deux autres sont (beaucoup) plus

long mais ne laissent que très peu d'aliasing. Ils ont une tendance à *flouter* l'image, surtout au loin (comme sur les plans infinis).

3.4.2 L'échantillonnage aléatoire de type jittered

L'autre méthode d'anti-aliasing est l'échantillonnage aléatoire du type jittered. Cette méthode consiste aussi à envoyer plusieurs rayons pour un même pixel mais ceux-ci sont envoyés aléatoirement autour du pixel. C'est-à-dire que si on veut envoyer neuf rayons par pixel, on divise alors celui-ci en neuf carrés égaux. Ensuite, au lieu d'envoyer un rayon au centre de chacun de ces carrés, on calcule un *déplacement* de ce carré suivant la formule : $P = P + \alpha * (2 * U() - 1)/(nbdiv - 1)$ ou P est le centre du carré, α est la constante de déviation relative par rapport au centre du pixel donné (sachant que si $\alpha = 0$ alors le point sera toujours au centre et si $\alpha = 1$, alors le rayon pourra être lancé au bord du carré), *nbdiv* est le nombre de division du pixel et $U()$ est la fonction renvoyant un nombre aléatoire entre 0 et 1 (on utilise *rand()* avec une graine initialisée au temps CPU du démarrage du programme).

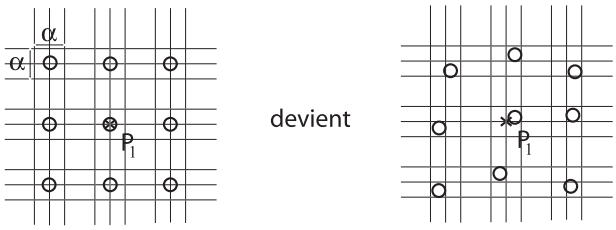


FIG. 12: L'anti-aliasing de type jittered

On calcule ainsi pour chaque pixel n rayons tous répartis aléatoirement sur le pixel à calculer, sachant que chaque répartition est différente sur les pixels voisins.
La différence par rapport à la technique précédente c'est que le côté aléatoire permet de remplacer l'aliasing par du bruit.

3.4.3 La technique accélérée

Il est possible d'améliorer la rapidité de l'anti-aliasing. En effet, on remarque que l'aliasing ne se produit que sur deux pixels voisins ayant une forte différence. Ainsi, il suffit de lancer le rayon central et si la différence entre la couleur trouvée et la couleur du pixel précédent ou situé au-dessus est grande, alors on applique le filtre. On lance alors le nombre de rayons nécessaire. Cette technique présente l'avantage d'accélérer nettement le calcul de l'image dans des zones où la couleur est uniforme comme l'arrière plan ou dans des grandes plages de couleur.

On remarque qu'il est nécessaire de calculer une ligne supplémentaire, celle du dessus de l'image, et une colonne, celle de gauche.

4 Traitements finaux et sauvegarde

L'implémentation de ceux-ci sont contenus dans la classe *image*.

4.1 Seuillage, égalisation et recadrage

Le calcul de la luminance en un point est une série d'additions de diverses intensités (cf 3.3) et on peut assez rapidement obtenir une saturation : la couleur possède l'une de ses composantes au-dessus de 1. Or, pour le codage d'une image, il est important de n'avoir que des valeurs

comprises entre 0 et 1.

Pour résoudre ce problème, on peut utiliser la fonction de seuillage qui consiste à parcourir l'image entière et, pour chaque composante trouvée supérieure à 1, on la ramène à 1. Cette méthode fonctionne bien car elle est peu coûteuse en temps mais n'offre pas de bon résultats. En effet, si une partie de l'image est très lumineuse et une autre très sombre, le seuillage va écraser la différence et le contraste va être réduit.

Pour pallier à ce problème, une seconde méthode consiste d'abord à effectuer une égalisation de l'image puis un recadrage de celle-ci.

L'égalisation consiste à rechercher la plus grande valeur de l'image et si celle-ci est supérieure à 1, alors on divise toutes les valeurs de l'image par celle-ci. Donc, à la sortie de cette opération, on aura toutes les valeurs de couleur de l'image comprises entre 0 et 1.

Ensuite, après ceci, on peut effectuer un recadrage de l'image, c'est-à-dire redistribuer toutes les valeurs dans l'intervalle [0,1]. Pour cela, on calcule d'abord l'histogramme cumulé pour chaque composante de couleur. Puis, on cherche le seuil bas (resp. haut) de l'image de manière à ce que 2.5% (resp. 97.5%) des pixels ont des valeurs inférieures (resp. supérieures). Les deux seuils seront alors les valeurs trouvées. Puis on reparcourt l'image et, pour chaque composante de couleur :

- si la valeur est inférieure eu seuil bas, alors on la remplace par 0
- si la valeur est supérieure eu seuil haut, alors on la remplace par 1
- sinon, on la remplace par une nouvelle valeur qui est :

$$\text{nouvelle valeur} = \alpha * \text{ancienne valeur} + \beta$$

où α = 1 / (seuil_haut - seuil_bas) et β = seuil_bas * 255 / (seuil_bas - seuil_haut)

On obtient donc un traitement plus long que le seuillage mais qui à l'avantage de garder les nuances car on redimensionne juste l'espace des couleurs de l'image.

4.2 Le brouillard

Le brouillard provoque une atténuation de toute la scène. Il dépend de la distance des objets par rapport à la position de l'observateur. Il ne peut malheureusement pas se calculer en même temps que l'image finale. En effet, si la couleur du brouillard est le blanc et qu'un objet est fort lumineux, les traitements détaillés dans le paragraphe suivant vont avoir un effet sur le brouillard. En particulier, dans ce cas, le brouillard va foncé. On ne peut donc pas calculer l'atténuation avant la fin.

Lorsque l'on calcule les intersections, on stocke la distance de l'intersection à l'observateur dans la matrice de profondeur. Une fois l'image calculée et les traitements faits, le brouillard peut être calculé.

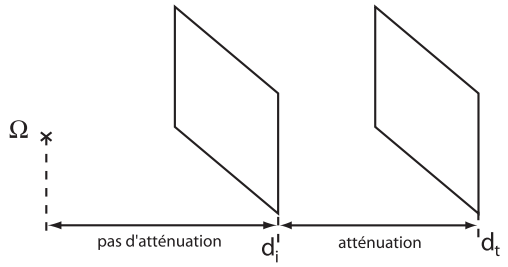


FIG. 13: Le brouillard

La méthode employée ici, est une fonction linéaire par rapport à la distance. Deux paramètres, d_i et d_f indiquent respectivement la distance minimum à partir de laquelle le brouillard commence à être visible et la distance à partir de laquelle on ne voit plus que le brouillard. On a alors la

fonction d'atténuation suivante :

$$f(d) = \begin{cases} 1 & \text{si } d < d_i, \\ \frac{d_f - d}{d_f - d_i} & \text{si } d_i < d < d_f \\ 0 & \text{si } d \geq d_f \end{cases}$$

Elle dépend évidement de la distance. La couleur C d'un pixel de l'image est modifiée ainsi :

$$C' = f(d).C + (1 - f(d)).C_b, \text{où } C_b \text{ est la couleur du brouillard}$$

4.3 Sauvegarde

Cette opération consiste juste à enregistrer l'image au format *.ppm*. Pour cela, on ouvre un fichier et on écrit *P6*, la taille de l'image et *255*. Puis, on écrit successivement les valeurs de rouge, vert et bleu de chaque pixel après les avoir multiplié pas 255 pour passer de l'espace [0,1] à l'espace [0,255] du *.ppm*.

5 Structures utilisées

Le projet a été écrit en C++. Les différentes classes implémentées sont les suivantes :
- *observateur* : c'est la caméra,
- *moteur* : le corps principale du programme,
- *scene* : comprenant tous les objets, les lumières,
- les différents objets (*objet*, *plan*, *sphere*, *cylindre*, *facette*, *slab*, *csgunion*, *csginter*, *csgdiff*,
- les lumières : *lum_ponc*, *spot*, *lum_nponc*,
- *couleur*, *materiel*,
- *erreur* qui gère toutes les erreurs,
- *image*

Des librairies de gestion des listes (*liste.hpp*), des files (*file.hpp*) et des piles (*pile.hpp*) sous forme de templates ont été implémentées pour les lumières, les objets et les milieux (pour la transparence). Des objets vecteurs ont été crées (*vecteurd.h*) pouvant définir bien entendu des vecteurs mais aussi des points de l'espace et des objets matrices (*matrice3.h*, *matrice4.h* et *matriced.h*).

5.1 Les objets

Une classe abstraite *objet* à été créée et tous les objets (simples et CSG) dérivent de celle-ci. Elle permet simplement d'associer des propriétés matérielles à chaque objet, de définir les fonctions virtuelles pures d'intersection, de liste d'intersection (cf section 3.2) et de calcul de normale en un point. L'avantage de la classe abstraite c'est qu'elle permet la création d'une liste d'objets différents.

Les différents objets implémentés se trouvent dans la section 3.2.1 pour les objets simples et dans la section 3.2.2 pour les objets CSG.

5.2 Les propriétés des objets

La classe propriété d'un objet définie toutes les propriétés matérielles de celui-ci :
- La couleur d'un objet se présente sous la forme d'un tableau, puisqu'il est possible d'associer plusieurs couleurs pour un objet dans le cas des textures par exemple.
- La couleur de transparence.

- Le type de la texture avec les différents paramètres associés. Pour le cas du plaquage de texture, on a besoin des trois matrices qui forme l'image plaquée.
- Les propriétés :
  - les coefficients diffus (*kd* et *kdt*),
  - les coefficients spéculaires (*ks* et *ns*),
  - les coefficients de transparence (*kt* et *nt*),
  - le milieu interne de l'objet (*milieu*, qui doit toujours être défini, même pour les objets non transparents.

5.3 La couleur

Le format de couleur utilisé est le type RVB. L'implémentation de ce format est dans la classe *couleur*. Chaque couleur est composée de trois flottants et, lors des opérations sur les couleurs (addition, multiplication, ...), on ne vérifie jamais si une des composantes dépasse 1 (on travaille entre 0 et 1) car les traitements nécessaires seront effectués ultérieurement (cf 4.1).

Pour l'image (dans la classe *image*), une matrice est créée pour chaque composante de couleur (donc trois matrices en tout) et c'est dans celle-ci que seront stockées les différentes luminances calculées lors du lancer de rayon.

5.4 Les lumières

Une classe abstraite *lumière* à été créée et tous les types de lumières (ponctuelles et spots) dérivent de cette classe. Les motivations de cette hiérarchie sont les mêmes que pour les objets. Les fonctions virtuelles pures ainsi définies sont les fonctions d'obtention de la couleur de la lumière, de son centre, son intensité ainsi que la distance entre la lumière et un point et pour savoir si la lumière est visible d'un point (utile pour le spot). Les différents modèles sont détailles dans la section 3.3.4.

5.5 Les autres classes

Une classe *observateur* a été créée et contient toutes les informations nécessaires à la gestion de l'observateur, à savoir :

- son centre (*centre*)
- son repère (direction du haut, du regard et le troisième vecteur dans une matrice *repere*)
- l'angle de vision (*phi*)
- la distance focale (*d*)
- le rayon de l'image (*r*)
- la taille d'un carré de la grille (*delta*)

Une classe *rayon* a été implémentée sachant qu'un rayon est caractérisé par son centre (*centre*), sa direction (*direction*) mais également un entier *r* qui est le nombre maximum que ce rayon devra rebondir pour la gestion de la récursivité.

Enfin, les classes *scene* et *moteur* contiennent, pour la première, la liste des objets et des lumières, la couleur du fond et de la lumière ambiante et les fonctions de calcul de la luminance en un point et des calculs globaux d'intersections pour l'éclairage et la transparence.
Quant à la seconde, on y trouve les fonctions de lecture du fichier de configuration et de lancer de rayon mais aussi l'observateur (*o*), la scène (*s*), l'image de sortie (*im*) et les booléen positionnant :

- l'anti-aliasing
- la récursivité
- si il y a égalisation et/ou recadrage
- si c'est du ray-tracing ou du ray-casting qui est demandé

6 Calcul parallèle

Le calcul en raytracing est lourd et donc parfois très long, surtout lorsque la taille de l'image à calculer devient importante. C'est pourquoi un programme annexe a été implémenté afin de supporter le calcul sur des ordinateurs mis en réseau.
Pour cela, le moteur a été modifié un minimum et surtout, certaines opérations ont dû être bannies lors du calcul en réseau. Les principales modifications sont les suivantes :

1. le moteur accepte le calcul sur une fenêtre donnée. Cette caractéristique est d'ailleurs intéressante puisqu'elle a facilité le débuggage au cours du développement du moteur. En effet, au lieu de calculer toute une image, on se borne uniquement aux endroits suspects.
2. le seuillage, l'égalisation et le recadrage ne doivent pas être fait. En effet, si une partie de l'image est sombre et une autre très lumineuse, on ne pourra pas faire un traitement séparé sur les deux parties de l'image.
3. la sauvegarde sous forme de fichiers de réels est obligatoire afin de pouvoir laisser le serveur réaliser les différents traitements à la fin, avec toutes les parties de l'image assemblées.

Pour mettre le programme en réseau, on a choisi de le faire sous forme de clients-serveurs comme l'indique la figure ci-dessous :

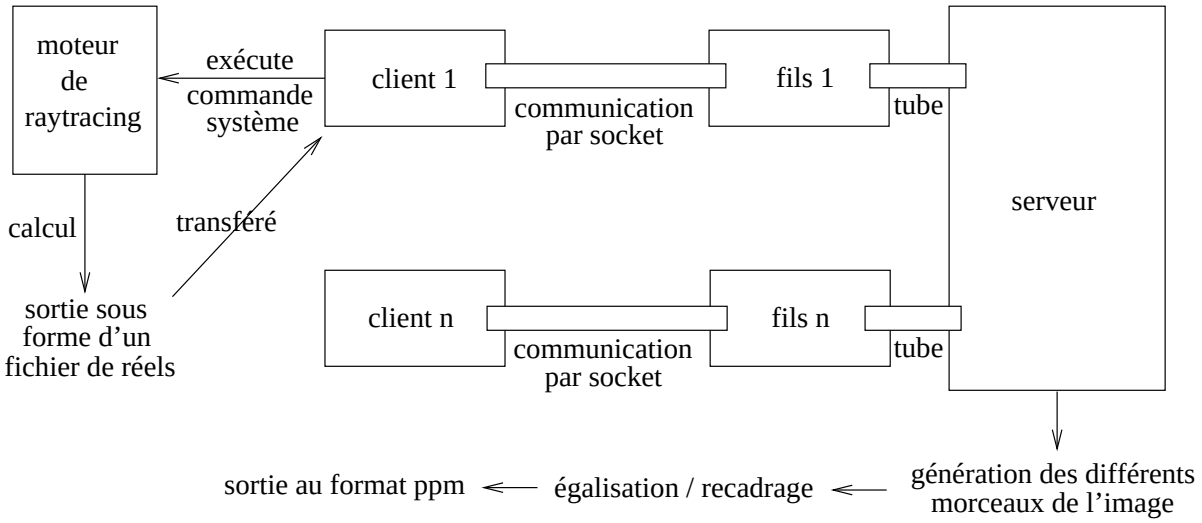


FIG. 14: Shéma de l'application en réseau

La procédure est la suivante :

1. On lance le serveur qui attend que tous les clients soient connectés,
2. Dès qu'un client est connecté et si l'option est sélectionnée, on lui envoie l'exécutable du moteur et le fichier de configuration.
3. Dès que tous les clients sont connectés, on commence le calcul.
4. Un client demande une fenêtre de calcul au serveur.
5. Le serveur calcule la fenêtre et l'envoie au client.
6. Le client exécute le moteur avec les paramètres de la fenêtre et une fois que le calcul est terminé, il envoie au serveur le fichier de sortie.
7. Le serveur réceptionne ce fichier, le stocke et renvoie la fenêtre suivante.

8. Lorsque toute l'image est calculée, le serveur indique aux clients la fin de l'opération.
9. Le serveur rassemble l'image en mémoire, fait l'égalisation et le recadrage et sauvegarde l'image obtenue au format *ppm*.

Cette technique présente des inconvénients. En effet, les temps de communication sont importants et sur une image qui se calcule en une vingtaine de secondes, on ne gagnera pas plus d'une paire de seconde. Sans parler du fait que pour l'anti-aliasing, par exemple, on est obligé de recalculer les rayons autour de la fenêtre. Cependant, la répartition dynamique va permettre d'éviter d'attendre les clients qui auraient plus à calculer que les autres et sur des images très grandes, où les zones de vide sont plus nombreuses, on gagne énormément de temps.

Des tests n'ont pas pu être réalisés sur des ordinateurs personnels en réseau. Mais rien que sur le serveur *mathinfo3*, avec deux clients, on met près de 50% de temps en moins sur des images qui mettent plus de une minute (aux alentours de 30s, le gain est très faible).

Conclusion

Le projet comporte tout ce qui avait été demandé dans le sujet :
- les objets simples : plan, plan contraints, sphères et slabs,
- la gestion des reflets, des ombres et de la transparence avec le modèle de Hall,
- le rendu en couleur,
- le traitement de l'aliasing,
- la gestion de sources lumineuses multiples et spots,
- la lecture dans un fichier de configuration,
- l'habillage de type damier.

On a ajouté les éléments suivants :
- la possibilité de faire un rendu en Ray-Casting,
- la possibilité de faire le rendu sur une fenêtre de l'image,
- les objets simples : cylindres, facettes,
- les objets CSG (union, intersection et différence),
- la transparence colorée,
- le brouillard,
- le plaquage de texture type *ppm*,
- l'application réseau.

Cependant, les améliorations suivantes auraient pu, avec du temps, être réalisées :
- l'équation de Fresnel,
- une meilleure gestion des textures plaquées avec par exemple la possibilité de faire des rotations mais surtout de texturer sur des objets autres que le plan et la sphère,
- les lumières non ponctuelles.

Annexes

A Mode d'emploi du fichier de configuration

Pour commencer, les lignes de commentaires dans le fichier de configuration sont précédées par le caractère #.

Pour définir les objets, on a la syntaxe suivante :

SPHERE	$C_x\ C_y\ C_z$ Rayon
PLAN	$C_x\ C_y\ C_z\ N_x\ N_y\ N_z\ c_x\ c_y\ c_z\ D_x\ D_y\ D_z$
CYLINDRE	$C_x\ C_y\ C_z$ Rayon Hauteur $N_x\ N_y\ N_z$
SLABS	$C_x\ C_y\ C_z$ n $N_{1x}\ N_{1y}\ N_{1z}\ d_1$ ... $N_{nx}\ N_{ny}\ N_{nz}\ d_n$

Où $C_x\ C_y\ C_z$ désigne le centre de l'objet, $N_x\ N_y\ N_z$ la normale, $c_x\ c_y\ c_z$ les contraintes (uniquement pour le plan), n le nombre de couples de plans qui forment le slab avec d_i la distance séparant le centre des deux plans du couple i.

Pour les CSG, on doit préciser l'opération puis les deux objets qui forment le CSG, sachant qu'ils peuvent être eux-même des CSG.

UNION | INTERSECTION | DIFFERENCE
1^{er} objet
2^{nd} objet

Pour chaque objet, on doit définir ses propriétés matérielles comme suit :

COULEURF	R V B
COULEURF_S	R V B
COULEURF_T	R V B
COULEUR	nom_couleur
COULEUR_S	nom_couleur
COULEUR_T	nom_couleur
PROPRIETE	$k_d\ k_{dt}\ k_s\ n_s\ k_t\ n_t\ milieu$
TEXTURE	nb + paramètres

R, V et B sont des flottants tandis que nom_couleur est une chaîne de caractère définie dans *couleur.dta*. COULEURF_S et COULEUR_S servent à définir une couleur supplémentaire pour le cas du damier. Quant au milieu, celui-ci doit faire partie de la liste des milieux définie dans *milieu.dta* et il sera automatiquement remplacé par sa valeur lors de la lecture. COULEUR_T est la couleur par transparence de l'objet. Toutes les valeurs de R, V et B doivent être données entre 0 et 1. Pour la texture, nb désigne le type de texture et on met dans paramètres les indices suivants :

type	nb	paramètres
Damier	1	x y z : dilatation en x, y et z
Distance à l'axe z	2	x y : facteurs de dilatation
Distance à l'axe x	3	y z : facteurs de dilatation
Distance à l'axe y	4	x z : facteurs de dilatation
Texture	5	nom_fich x y : nom + facteurs de dilatation

Ensuite, les trois types de lumière seront crées comme suit, sachant qu'il faut d'abord définir une couleur qui lui sera attribuée.

LUMIERE	intensité
LUM_PONC	intensité C_x C_y C_z
SPOT	intensité C_x C_y C_z θ φ exposant

C_x C_y C_z est le point où se trouve la lumière, l'exposant est l'atténuation de la lumière et θ et φ sont les angles d'ouverture du spot. Les intensités doivent être comprises entre 0 et 1.
Les paramètres de l'observateur sont donnés comme suit :

OBSERVATEUR	C_x C_y C_z reg_x reg_y reg_z hau_x hau_y hau_z φ

C_x C_y C_z est la position de l'observateur,reg_x reg_y reg_z sa direction du regard, hau_x hau_y hau_z sa direction approximative du haut et φ l'angle de vision.
Les derniers réglages disponibles sont les suivants :

RECURSIVITE	n
ARRIERE_PLAN	R V B
RESOLUTION	largeur hauteur
FENETRE	min_x max_x min_y max_y
RESULTAT	nom_fichier_de_sortie
RAYCASTING | RAYTRACING	aucun
EGALISE	aucun
RECADRAGE	aucun
ANTI_ALIASING	num + paramètres + fast + ε
BROUILLARD	d_i d_t

Pour l'anti-aliasing, le paramètre *fast* permet de sélectionner la méthode accélérée s'il vaut 1. ε est le coefficient de variation pour la méthode accélérée uniquement. Les choix pour *num* sont les suivants :

nb	type	paramètres
1	filtre gaussien (grille à 9)	aucun
2	filtre gaussien (1^{ere} grille à 25)	aucun
3	filtre gaussien (2^{nde} grille à 25)	aucun
4	filtre moyenneur	aucun
5	échantillonage de type jittered	α nb_rayon

Dans le 6^{ieme} cas, α est la constante de décalage (comprise entre 0 et 1) et nb_rayon la racine du nombre de rayon à envoyer sur le pixel lors de l'échantillonage (3 et 5 sont des valeurs pour une grille de 9 et 25 rayons).

B Exemple d'anti-aliasing

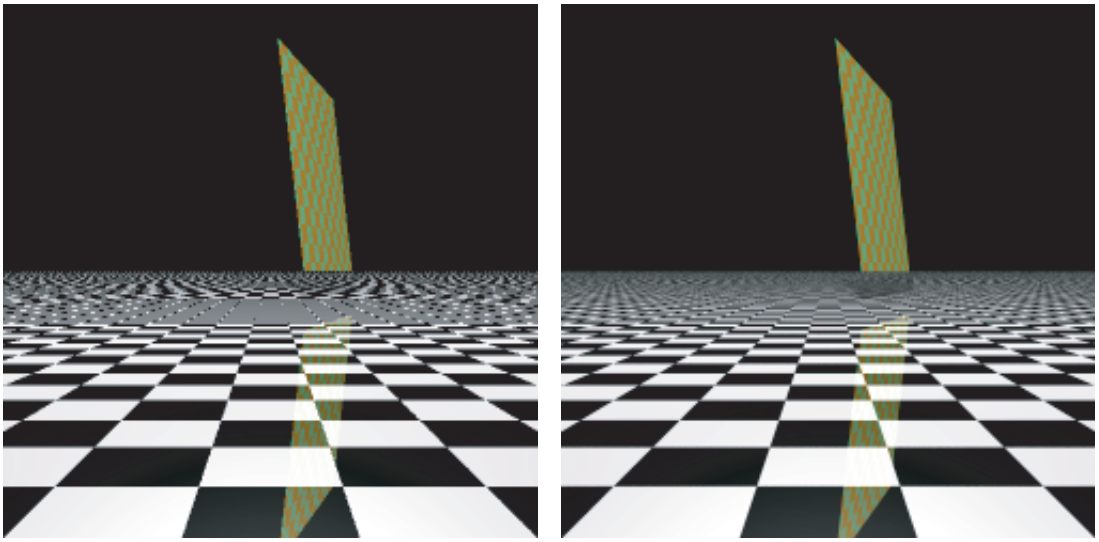


FIG. 15: Une image sans et avec anti-aliasing

C Exemple de brouillard

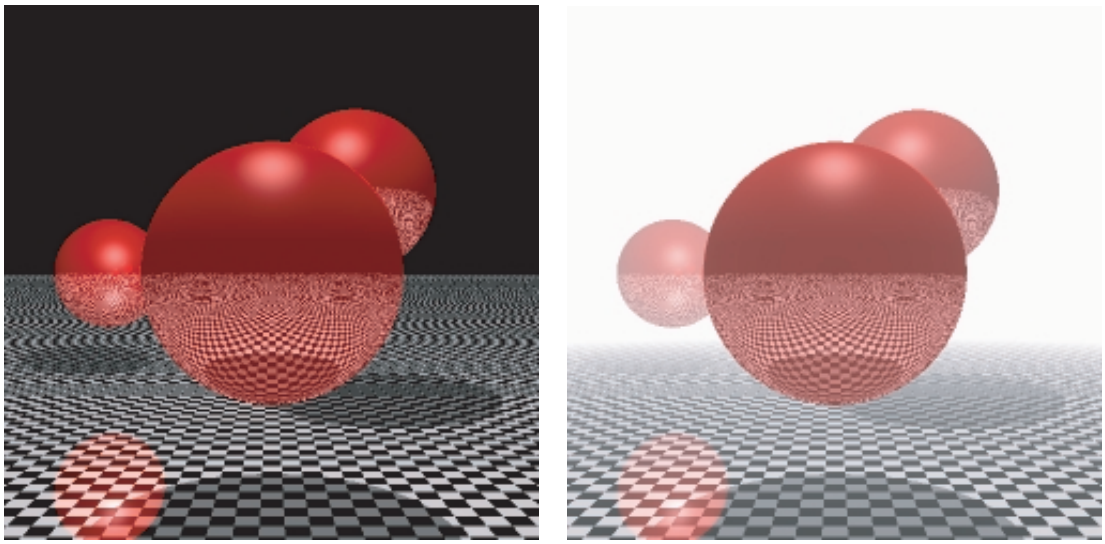


FIG. 16: Une image sans et avec brouillard