Projet de Synthèse d'images

Maîtriser le lancer de rayons simple dans une scène hiérarchisée

Présentation du sujet :

La technique du lancer de rayon a été vue en cours, les slides sont disponibles sur ma page web. Par ailleurs, de nombreuses documentations sont disponibles sur Internet. Voir par exemple la page wikipedia sur le lancer de rayon (attention la partie 'Futur et prospectives' est largement dépassée). Il existe pas mal de code disponible mais pas forcément adapté à votre projet.

L'objectif du projet sera de réaliser, en binôme, un lancer de rayon en C/C++ mais qui travaillerait sur une structure accélératrice de données permettant un rendu efficace.

Travail demandé :

Niveau	Travail demandé	« Module » informatique à réaliser	Note max
1	Lancer de rayon simple	- chargement de la scène par fichier - caméra statique - restitution d'une image finale - lancer de rayon en *flat painting*	4
2	Lancer de rayon optimisé	- création de la structure de donnée hiérarchique - lancer de rayon exploitant la structure de données - création d'une scène de plus de 100 objets canoniques - envoie de plusieurs rayons par pixel	8
3	Lancer de rayon optimisé interactif	- restitution de l'image dans une fenêtre OpenGL - déplacement de la caméra - lancer de rayon « progressif » - insertion des lumières et calcul du modèle de Blinn-Phong	12
4	Extensions	- Calcul de l'ombrage - Calcul des reflets & de la transparence - Extensions des formats d'entrée et de sortie	+4

De plus, 4 points sont attribués au rapport final & à la qualité du rendu : commentaire dans le code, qualité de la soutenance, facilité d'installation et respect des spécifications demandées.

Détails des modules informatiques à réaliser

Chaque niveau doit aboutir à une application « standalone » en C/C++ qui permettra de réaliser les fonctionnalités demandées et détaillées ci-dessous.

Niveau 1 : le lancer de rayon simple

Format et chargement de la scène

En lancer de rayon (comme en rasterisation d'ailleurs), les scènes virtuelles à rendre sont constituées d'objet canonique. Ce sont les éléments « unitaire » constituant la scène sur lesquels les tests d'intersection avec les rayons sont fait. Dans notre cas, la scène sera constituée des objets canoniques suivant : sphère, rectangle, triangle et cylindre. Pour le niveau 1, vous rendrez une scène constituée d'au moins un objet de chacun de ces types.

La construction de la scène doit se faire via le chargement d'un fichier au format que vous désirez parmi les possibilités suivantes : simple fichier texte (txt), json, XML. Le format « interne » de ces fichiers est laissé à votre appréciation mais chaque objet devra comporter les informations suivantes :
- Ces caractéristiques intrinsèques : coordonnées des 3 points pour un triangle, centre et rayon pour une sphère…
- Des informations sur sa couleur ou si extensions ses propriétés radiométriques (coefficient de réflexion diffuse par exemple)

Éventuellement, pour pouvez associer à chaque objet une matrice de transformation mais cela est laissé à votre appréciation.

Le fichier contiendra également la position et l'orientation de la caméra. Enfin, vous pourrez définir des propriétés supplémentaires comme la position & l'intensité de sources de lumière ou d'autres points de vue.

Votre application devra être capable de lire ce fichier et de réagir en conséquence.

L'application du lancer de rayon

Pour le niveau 1, vous devrez donc réaliser un exécutable qui prend en entrée un fichier au format défini par vos soins (voir ci-dessus) décrivant la scène et qui doit restituer une image en sortie.

L'exécutable, appelé lray, donc pouvoir répondre à la commande suivante :

```
lray -n 1 -i <mon_fichier.format> -o image.ppm
```

L'option -n défini le « niveau » du lancer de rayon exécuté. L'option -i défini le fichier d'entrée. L'option -o défini l'image de sortie.

Ainsi le fichier mon_fichier.format contient la scène et la position caméra et le programme génère une image au format PPM contenu une vue de la scène depuis la caméra. Ce lancer de rayon enverra un rayon par pixel et renverra une image en *flat painting* (i.e. une couleur par objet) de la scène.

Niveau 2 : le lancer de rayon optimisé

Le lancer de rayon « niveau 1 » est trop inefficace pour gérer correctement une scène massive (i.e. constituée de centaines d'objet canonique). Il est donc nécessaire de créer une structure de données de briser la dépendance au nombre de ces objets. Nous utiliserons, pour ce faire, une structure de données appelée *Boundary Volume Hierarchy* ou, en Français, hiérarchie de volume englobant. L'idée est de calculer, pour chaque objet canonique, un volume englobant, puis de construire un arbre binaire où chaque nœud contient un volume englobant les deux fils (eux même ayant des volumes englobants). Les feuilles de l'arbre contiennent un lien vers les objets canoniques pour assurer l'intersection entre un rayon et ces objets canoniques.

Chargement d'une scène massive

La première difficulté réside dans la création de la scène complexe. Impensable en effet d'écrire à la main une scène constituée de plus de 100 éléments. Deux solutions s'offre donc à vous :

- Créer des objets canoniques automatiquement, procéduralement, par exemple en discrétisant des surfaces (un rectangle transformé en une myriade de triangles) ou en utilisant des algorithmes de génération de géométrie procédurale (comme la création d'un terrain par fractales)
- Charger un objet 3D complexe via un format de fichier comme .obj ou, plus facile encore, le format .ply. Pour ce faire, vous avez le loisir d'utiliser des bibliothèques dédiées comme ASSIMP par exemple. Dans ce cadre, le fichier texte descriptif de la scène peut contenir des noms de fichiers à charger.

Une fois la géométrie créée ou chargée, vous pourrez remplir la structure de données accélératrice.

Remplissage de la structure de données

L'algorithme général construit la hiérarchie en partant de la racine, c'est à dire le volume occupé par l'ensemble des objets de la scène. Ensuite, il faut trouver une coupe dans ce volume qui permette de répartir les objets entre les fils gauche et droit. La construction est récursive et se termine lorsqu'il n'est plus possible de séparer les objets (il n'en reste plus que 1 ou 0). Dans notre cas, **nous utiliserons des sphères englobantes**.

Le problème le plus délicat est de choisir où couper un volume englobant afin de répartir les objets entre les fils gauche et droit. La solution simple consiste à répartir "équitablement" les objets entre les deux fils et permet d'obtenir un arbre binaire équilibré. Malheureusement, cette "stratégie" est une des pires pour le calcul d'intersections. En effet, un arbre équilibré ne garanti pas la meilleure séparation des objets : c'est à dire éliminer les nœuds dont le volume n'est pas traversé par le rayon.

Il faut donc utiliser un autre critère de séparation qui tient compte du rayon. Le plus répandu utilise la probabilité qu'un rayon traverse le fils gauche (ou droit) sachant que le père est lui aussi traversé par le rayon. Cette probabilité est proportionnelle au rapport de l'aire des surfaces des volumes englobants des deux nœuds : A(fils) / A(père), avec A(nœud), l'aire de la surface du volume englobant.

Ce critère estime le temps d'intersection du rayon avec les objets associés au nœud sachant que l'un des fils sera également intersecté. Il suffit alors de trouver la répartition entre les fils gauche et droit qui minimise le temps d'intersection T :

$T = T_{gauche} + T_{droit}$

avec $T_{fils} = T_{englobant} + A(fils) / A(père) \,.\, N(fils)\, T_{objet}$

avec :

- $T_{englobant}$: temps d'intersection du rayon avec le volume englobant d'un nœud,
- T_{objet} : temps d'intersection du rayon avec un objet,
- A(nœud) : aire de la surface du volume englobant du nœud (rappel : les volumes englobants sont des cubes dans notre cas),
- N(nœud) : nombre d'objets "affectés" au nœud.

Pour T_{objet} et $T_{englobant}$, vous pourrez prendre une constante comme la valeur 1.

On peut interpréter les différentes parties de cette expression de T_{fils} :

- A(fils) / A(père) : est proportionnel à la probabilité d'intersecter les objets d'un fils connaissant le volume englobant du père,
- N(nœud) T_{objet} : représente le temps d'intersection du rayon avec les objets associés au noeud.

L'algorithme de construction ne teste pas tous les plans de coupe possibles, mais uniquement les plans principaux alignés sur le repère de la scène et passant par le centre du volume englobant de chaque objet.

L'algorithme, présenté en détail dans l'article de référence, parcours l'ensemble d'objets dans les deux sens (du min vers le max et réciproquement, le long de chaque axe) et construit de manière incrémentale A(gauche), N(gauche), A(droit), N(droit) pour chaque plan de coupe candidat.

La première étape consiste à calculer, pour chaque objet canonique, la sphère englobante. Le plus simple est d'utiliser une structure de données ad-hoc contenant le centre de la sphère et son rayon.

```
struct SBBOX {
       float rayon_sphere;
       VEC centre;
};
```

La seconde étape est l'algorithme récursif de construction de l'arbre :

```
algo : construction(S : ensemble des objets canoniques)
  // construire l'ensemble de plans de coupe candidats pour chaque axe
  pour axe = X, Y, Z
       P = S
       trier l'ensemble P par position du centre des sphères englobantes croissant,
         c'est à dire P[i].centre[axe] croissant

       // énumérer les plans de coupe, tous les P[i].centre[axe]
       pour i = [ 0 .. N(P) [
              répartir les primitives placées avant P[i].centre[axe] dans S1
              répartir les primitives placées après P[i].centre[axe] dans S2

              // la position d'une primitive P[j] est déterminée par
                 la position du centre de sa boite englobante, P[j].centre :
              // si P[j].centre[axe] < P[i].centre[axe] alors ajouter la primitive P[j] à S1
              // si P[j].centre[axe] > P[i].centre[axe] alors ajouter la primitive P[j] à S2

              calculer la boite englobante de S1 et de S2
              calculer l'aire des englobants des nœuds : A(S1), A(S2) et A(S)
              calculer T, le coût du plan candidat

              // conserver le candidat de coût minimum
              si T < Tmin
                     Tmin = T
                     imin = i
                     axemin = axe
              fin si
       fin pour
  fin pour

  construire les fils gauche et droit avec la répartition déterminée pour Tmin,
       dans le fils gauche : ensemble S1, les objets P[j]
                             telles que P[j].centre[axemin] < P[imin].centre[axemin]
       dans le fils droit : ensemble S2, les autres primitives P[j]
                             telles que P[j].centre[axemin] > P[imin].centre[axemin]

  // continuer récursivement la construction des fils gauche et droit
  construction(S1)
  construction(S2)
fin
```

Il faut s'arrêter lorsqu'un nœud contient entre 1 & 4 objets canoniques

Exploitation de la structure de données

Maintenant, lors du lancer de rayon, il s'agit de faire un parcours de l'arbre binaire avec le rayon. Le principe est simple. Si le nœud est une feuille, il faut renvoyer l'intersection du rayon avec le ou les objets associés. Sinon, il faut regarder l'intersection avec les 2 fils. Dès qu'il y a intersection, on fait l'appel récursif sur le fils correspondant.

Ceci est l'algorithme « simple ». Pour améliorer la rapidité, il faut essayer de faire les appels récursifs dans l'ordre. Nous ne sommes intéressés que par l'intersection la plus proche de l'origine du rayon. Dans le cas ou les deux fils sont visités par le parcours, il est préférable de tester en premier les intersections avec le fils le plus proche de l'origine du rayon. L'autre fils ne sera testé que si le rayon traverse le premier fils sans intersection. Cela dit, ça n'est pas demandé donc si vous optez pour cette technique, vous devrez le signaler dans votre rapport.

L’application du lancer de rayon optimisé et le *pixel sampling*

Pour le niveau 2, vous devrez donc réaliser, comme pour le niveau 1, un exécutable qui prend en entrée un fichier au format défini par vos soins (voir ci-dessus) décrivant la scène et qui doit restituer une image en sortie avec toujours l’option -n.

Vous devrez rajouter en plus une option -ps (pour pixel sampling) permettant de lancer plusieurs rayons par pixels. Ainsi l’exécutable, appelé lray, donc pouvoir répondre à la commande suivante :

```
lray -n 2 -ps 16 -i <mon_fichier.format> -o image.ppm
```

Pour le pixel sampling, au lieu de tirer un seul rayon au milieu, il faut maintenant tirer N rayons (N donné dans la ligne de commande après l’option -ps). Chaque rayon doit alors passer par un point tiré aléatoirement dans le pixel. Chaque rayon renvoie une couleur et l’ensemble de ces couleurs est moyenné pour obtenir la couleur finale du pixel.

Niveau 3 : le lancer de rayon interactif et optimisé

Cette partie, légèrement plus technique, permet de visualiser directement le rendu que l'on obtient dans une fenêtre OpenGL. Pour ce faire, vous devrez, en *offline*, construire l'image en tirant un rayon par pixel, puis l'afficher dans la fenêtre. Vous pouvez réitérer le processus en moyennant l'image, cela vous permet ainsi d'affiner l'image finale interactivement. C'est particulièrement utilisé pour des algorithmes plus puissant comme le path tracing, mais ici cela permettra essentiellement à voir la scène directement, à position la caméra, et à « luter » contre l'aliasing. Cette partie vise aussi à permettre à l'utilisateur de déplacer la caméra, grâce à l'appui de touches au clavier.

Visualisation de l'image dans la fenêtre OpenGL et interaction utilisateur

Ici, au lieu de générer une image finale, l'idée est d'afficher cette image dans une fenêtre OpenGL. Pour ce faire, il faut donc générer une fenêtre OpenGL et, simplement, remplir cette fenêtre avec l'image (sous forme de tableau d'`unsigned char`) générée par le lancer de rayon.

Remplir la fenêtre, sous OpenGL, se fait simplement avec les instructions glDrawPixels (et éventuellement glDrawBuffers avec la cible GL_FRONT).

De plus, grâce à l'action de touches sur le clavier, vous permettrez à l'utilisateur de bouger la caméra. En avançant, en reculant et en orientant la caméra à gauche & à droite (au minimum). Le choix des touches devra être explicité, soit dans le rapport, soit explicitement via le programme comme une option d'aide en ligne de commande, ou l'action d'une touche dans l'application.

Lancer de rayon « progressif »

Tant que la caméra ne bouge pas, le programme continue à « rendre » l'image et à « moyenner » l'image finale. En gros, tant que la caméra ne bouge pas, vous demandez le rendu de l'image avec un rayon par pixel. L'image générée doit se « moyenner » avec l'image déjà calculée, pixel par pixel.

Pour « moyenner » chaque pixel, vous pouvez utiliser la formule suivante (pour chaque canal) :

$$P_{new} = \frac{1}{n} P_{arrive} + \frac{n-1}{n} P_{ancien}$$

où P_{new} sera la nouvelle valeur du pixel, n le nombre d'itération (de rendu effectué), P_{arrive} la valeur du pixel que l'on vient de calculer et P_{ancien} l'ancienne valeur du pixel

Calcul du modèle d'illumination de Blinn-Phong

Vous modifierez le format de fichier de description de la scène afin d'insérer des lumières dites ponctuelle (constituée d'un point et éclairant dans toutes les directions). Ces lumières sont donc caractérisées par une position et une intensité lumineuse.

Les lumières entraîneront un éclairage de la scène. Pour le modèle d'éclairage nous utiliserons le modèle de Blinn-Phong rappelé ci-dessous pour une unique lumière ponctuelle. Si X est le point illuminé, S la position de la source de lumière, l'illumination vue depuis le point de vue V s'écrit :

$$L(X \rightarrow V) = \frac{L}{4\pi XS^2}[\rho_d \underline{\vec{N}.\vec{I}} + \rho_s(\underline{\vec{N}.\vec{H}})^s]$$

avec :

- ρ_d est la couleur diffuse ou coefficient de réflexion diffuse de l'objet (3 composantes r,v,b)
- ρ_s est le coefficient de réflexion spéculaire de l'objet (3 composantes r,v,b)
- s est la brillance (shininess) de l'objet (usuellement comprise entre 2 & 128)
- $\vec{N}$ est la normale à la surface en X
- $\vec{I}$ est le vecteur d'illumination et vaut $\frac{\vec{XS}}{||\vec{XS}||}$

- $\vec{H}$ est le half vector et est la normalisation du vecteur $\frac{\vec{I}+\vec{V}}{2}$ avec $\vec{V}=\frac{\vec{XV}}{\|\vec{XV}\|}$
- $\underline{\vec{u}.\vec{v}}=\vec{u}.\vec{v}$ si $\vec{u}.\vec{v}\geqslant 0$ et 0 sinon

L'application du lancer de rayon optimisé interactif

Pour le niveau 3, vous devrez donc réaliser, comme pour le niveau 2, un exécutable qui prend en entrée un fichier au format défini par vos soins (voir ci-dessus) décrivant la scène et qui doit soit restituer une image si l'option -n est suivi du nombre 1 ou 2, soit passer en mode interactif si le chiffre après l'option -n est égal à 3. Dans ce cas, il ne doit pas y avoir d'option -o (sauf éventuellement à permettre

Ainsi l'exécutable, appelé lray, donc pouvoir répondre, au niveau 3, à la commande suivante :

```
lray -n 3 -i <mon_fichier.format>
```

Extensions

La liste présentée ici n'est pas exhaustive et vous pouvez à loisir proposer d'autres extensions. Quoiqu'il en soit, **citez toutes les extensions que vous avez proposé dans votre rapport**. En particulier ces extensions peuvent modifier le format de fichier d'entrée et ceci doit être aussi explicité dans votre rapport.

Voici une liste des éléments pour les extensions.

- Calcul de l'ombrage : il faut alors tirer un rayon d'ombre entre le point sur lequel on calcule l'illumination et la position de la source de lumière. Dès que l'on a intersection, alors le point est dans l'ombre. Attention à l'intersection avec l'objet du point illuminé. C'est une extension relativement simple à implémenter lorsque l'on a déjà fait l'algorithme d'intersection rayon / objets de la scène.
- Calcul des reflets & de la transparence : Ici lors de l'intersection, on peut relancer le rayon soit dans la direction du reflet, soit en transparence pure. De plus, l'objet sur lequel on fait cette interception peut éventuellement modifier la couleur finale (verre coloré par exemple). On peut, pour ce faire, s'appuyer sur un coefficient de réflexion spéculaire. Pour la transparence, il faut rajouter aussi l'indice de refraction du matériau constituant l'objet.
- Extensions des formats d'entrée et de sortie : plutôt qu'une sortie en image au format ppm, on peut sortir des images au format jpg ou png. Pour le format d'entrée, il s'agit du chargement d'objet complexe avec des formats dédiés comme .obj ou .ply comme vu précédemment.
- Texture sur les objets : les textures peuvent être lue pour récupérer par exemple la couleur ou les coefficients de réflexion diffuse.
- D'autres types de lumière et plus particulièrement la lumière de type spot qui éclaire depuis un point vers un cône donné (axe + angle).
- D'autres modèles d'éclairage comme Ward ou Cook-Torance.

Rendu du projet

Éléments à rendre

Vous devrez rendre pour une date et des modalités qui vous seront communiquées ultérieurement les éléments suivants :

- Votre code source sous la forme d'une archive compressée au format .tgz ou .tar.gz. Le format rar n'est pas autorisé. Le code se déploiera dans un répertoire contenant le nom de famille des différents membres du binôme (ou éventuellement monôme / trinôme).
- Vous rendez un rapport **succinct** (2-3 pages max) décrivant le moyen de compiler et d'exécuter le code (et notamment toutes la partie IHM) ainsi que vos résultats (timing, screenshot).

Votre projet devra respecter les considérations techniques ci-dessous. Le non respect des considérations techniques et des éléments vu ci-dessus impactera la note sur 4 dédiée au doublon rapport/projet.
Notez qu'un tel travail entraînera une note correcte (max 16). Mais pour aller plus loin (dans le « fun » & la note), vous pourrez proposer les extensions vu dans la section ***Extensions*** ci-dessus.

Considérations techniques

- Votre code compilera sous linux avec les bibliothèques autorisées (voir ci-dessous).
- Les bibliothèque utilisables, en plus des bibliothèques standard du C & du C++, seront : GLEW, glut, GL & GLU, SDL, SFML ainsi que GLM ou, éventuellement, les outils/codes vu en TD. Il est néanmoins possible d'utiliser une autre bibliothèque mais seulement avec mon accord : il suffit alors de m'envoyer un mail (biri@u-pem.fr).