SI – TP projet : rendu par lancer de rayon

L'objectif de ce TP est de comprendre et programmer les principaux éléments d'un moteur de lancer de rayon et implémenter une structure accélératrice pour le calcul d'intersection rayon/scène, ainsi que gérer les textures.

1. IMPORTANT

Ce TP constitue la base du TP de rendu de SIA au second semestre, vous devrez donc le conserver. En SIA, nous vous présenteront des techniques de rendu plus avancées (Monte Carlo par exemple) qui nécessitent un nombre important de calculs, plus précisément un nombre important de lancers de rayon. Avoir une structure accélératrice de calcul d'intersection rayon / scène fonctionnelle est donc nécessaire (voir 3.b).

2. L'ENVIRONNEMENT DE TP

a. RECUPERATION DU TP, STRUCTURE DU REPERTOIRE ET CONFIGURATION

Vous trouverez les sources du TP sur le répertoire share/esir2, dans le dossier SyntheseImages. Ce dossier contient les répertoires suivants :

- dependencies_ima. Ce répertoire contient des librairies précompilées utiles pour le TP : SOIL, lib3ds, tbb, SDL...
- Models. Ce répertoire contient un ensemble de modèles exemples au format 3DS (seul format chargé par la plateforme) et un ensemble de textures associées.
- RayCasting. Ce répertoire contient les sources du TP de lancer de rayon. Les sous répertoires sont organisés comme suit :
    - Doxygen. Ce répertoire contient le fichier de configuration pour la génération de la documentation au format doxygen.
    - src. Ce répertoire contient les sources du TP. L'organisation est faite de telle manières à ce que chaque sous répertoire corresponde à un espace de nommage et que chaque fichier porte le nom de la classe qu'il contient. Si des fichiers .cpp sont associés à des fichiers .h, ils se trouvent dans un sous répertoire src.
    - project 2010. Ce répertoire contient un projet configuré pour visual studio 2010.
    - project 2015. Ce répertoire contient un projet configuré pour visual studio 2015.
    - Project 2017. Ce répertoire contient un projet configuré pour visual studio 2017 (version actuellement installée sur les ordinateurs des salles de TP).

Vous devrez recopier les répertoires *RayCasting* et *Models* et les placer dans le même répertoire pour pouvoir réaliser le TP.

Attention. Avant d'ouvrir le projet Visual Studio, vous devez lancez le script setup.bat situé dans le répertoire *dependencies_ima* et ce, à chaque fois que vous vous connectez sur votre compte. Ce fichier paramètre des variables d'environnement qui sont utilisées dans les configurations des projets.

b. LES FICHIERS PRINCIPAUX ET POINTS D'ENTREE

main.cpp. Dans le répertoire src du projet, vous trouverez le fichier main.cpp. Ce fichier vous permet de configurer le rendu en fournissant la scène à calculer (voir le nom des méthodes associées dans la section suivante), le nombre de rebonds à prendre en compte lors du calcul des réflexions, la taille de la fenêtre de rendu etc… L'ensemble de ces réglages s'effectuent dans la fonction main du programme.

Il vous est conseillé, lors des phases de mise au point d'utiliser une petite fenêtre de rendu afin de limiter les temps de calcul tout en pouvant observer les résultats, tout particulièrement lorsque vous prendrez en compte l'éclairage indirect. Dans la fonction main, trois tailles vous sont proposées : 1000x1000, 500x500 et 300x300. Attention, il vous faut avoir des fenêtres de rendu carrées, la gestion du facteur de forme n'est pas implémentée pour le moment.

Geometry/scene.h. La classe Geometry::Scene est le point d'entrée du moteur de rendu. Elle propose une méthode nommée *compute* qui calcule le rendu de toute la scène. Cette dernière prend trois paramètres :

- *maxDepth* : correspond au nombre maximum de rebonds de rayons à considérer,
- *subPixelSubdivision* : correspond à la racine carrée du nombre de rayon primaires à lancer par pixel (cela correspond à l'antialiasing),
- *passPerPixel*, correspond au nombre de passes de rendu à effectuer par pixel. Ce paramètre n'est utile que si vous utilisez une méthode de rendu de type Monte Carlo. Dans le cas contraire, vous pouvez fournir une valeur de 1.

Le nombre de rayons primaires lancés par pixel est de subPixelSubdivision^2 * passPerPixel.

La méthode *sendRay* est appelée par la méthode *compute* une fois par rayon primaire. Cette méthode a pour rôle de retourner la couleur du pixel associé à ce rayon primaire. Cette méthode prend 5 paramètres :

- *ray* : le rayon pour lequel nous souhaitons effectuer le calcul.
- *depth* : la profondeur courante de récursion (nombre de rebonds du rayon depuis que le rayon primaire correspondant a été lancé)
- *maxDepth* : la profondeur maximale de récursion (nombre maximum de rebonds du rayon)
- *diffuseSamples* : uniquement utile si vous implémentez une méthode de type Monte Carlo pour connaitre le nombre de rayons à lancer pour estimer la composante diffuse indirecte (utile pour le TP de SIA au second semestre).
- *specularSamples* : uniquement utile si vous implémentez une méthode de type Monte Carlo pour connaitre le nombre de rayons à lancer pour estimer la composante spéculaire indirecte (utile pour le TP de SIA au second semestre).

Comme vous pourrez le remarquer, cette méthode est vide. Il s'agit de votre point d'entrée dans le moteur et de la méthode que vous allez devoir programmer. Attention, au cours de ce TP, pensez au découpage fonctionnel, ce dernier est important si vous souhaitez vous retrouver dans vos différents calculs.

C. LES SCENES FOURNIES

Le tableau ci-dessous résume les caractéristiques principales des scènes qui vous sont proposées dans l'application (elles sont initialisées dans la fonction main). La première colonne correspond au nom de la méthode à appeler pour initialiser la scène correspondante.

Fonction d'initialisation de scène	Type de scène	Nombre de triangles	Utilisation de textures	Lumières ponctuelles	Lumières surfaciques
initDiffuse	Boite de Cornell (diffus)	36	Non	Oui	Non
initSpecular	Boite de Cornel (spéculaire)	36	Non	Oui	Non
initDiffuseSpecular	Boite de Cornel (diffus et spéculaire)	36	Non	Oui	Non
initGuitar	Une guitare	45399	Non	Oui	Non
initDog	Un jouet représentant un chien	132034	Non	Oui	Non

initGarage	Un garage	219152	Non	Oui	Non
initTemple	Temple de Séraphin de Sarov	603111	Non	Oui	Oui
initRobot	Un robot	126944	Non	Oui	Non
initGraveStone	Une tombe avec feuillage	540902	Oui	Oui	Non
initBoat	Un bateau	389554	Non	Oui	Non
initSombrero	Un sombrero	2024	Oui	Oui	Non
initTibetHouse	Une maison tibétaine	22251	Oui	Oui	Oui
initTibetHouseInside	Une maison tibétaine	22251	Oui	Oui	Oui
initMedievalCity	Une ville médiévale	774644	Oui	Oui	Non

3. LES ETAPES DU PROJET

Dans cette section, il vous est proposé un ensemble d'étapes à suivre pour réaliser votre projet. Les parties a) et b) sont obligatoires (il s'agit de l'algorithme de base du lancer de rayon et de la structure accélératrice). La partie c) est une extension qu'il vous est recommandé de programmer.

a. L'ALGORITHME DE LANCER DE RAYON

Lors de l'implémentation de votre algorithme, il est très fortement conseillé de penser au découpage fonctionnel. Une proposition est la suivante : une fonction pour calculer l'éclairage direct, une fonction pour calculer l'éclairage indirect et une fonction pour vérifier si un point est éclairé ou non par une source lumineuse ponctuelle. Ce découpage n'est pas exhaustif mais vous permettra de mieux vous repérer dans votre code.

Afin de faciliter le repérage des erreurs que vous allez pouvoir commettre dans ce projet, il vous est **très fortement** conseillé de suivre les étapes suivantes lors du développement :

1. Configurez le projet de manière à initialiser la scène avec la méthode *initDiffuse*. Cette dernière crée une boite de Cornell contenant deux lumières et deux cubes ayant des matériaux diffus.
2. Calculez l'intersection entre le rayon primaire et la géométrie et retournez la composante diffuse du matériau associé au triangle intersecté. Cette étape vous permet de valider vos calculs d'intersection.
3. Calculez l'éclairage direct diffus au point d'intersection entre le rayon et la géométrie. Dans le calcul de la couleur retournée, ne prenez en compte, pour le moment, que la couleur des lumières et la composante diffuse du matériau. Ne vous souciez pas des ombres, vous les traiterez à la prochaine étape. Cette étape vous permet de valider vos calculs d'éclairage diffus.
4. L'éclairage que vous avez calculé à l'étape précédente ne prend pas en compte les ombres i.e. le fait que la lumière soit cachée par un objet situé entre cette dernière et votre point d'intersection. Ajoutez la gestion des ombres dans vos calculs.
5. Pour passer à l'étape suivante, changez de scène et utilisez la scène générée par *initDiffuseSpecular*. Cette fonction crée une boite de Cornell possédant un sol et un plafond uniquement diffus avec des cotés correspondant à des miroirs.
6. Ajoutez le calcul de la composante spéculaire directe à vos calculs d'éclairement. Pensez à réutiliser le résultat de l'étape précédente pour correctement gérer les ombres. Vous devriez voir apparaitre des reflets spéculaires sur les miroirs.
7. Prenez en compte la composante émissive des matériaux dans vos calculs (cela ne devrait faire aucune différence actuellement).
8. Comme vous le savez, une propriété de l'algorithme de lancer de rayon est de pouvoir gérer les surfaces spéculaires en prenant en compte les réflexions (éclairage indirect). Ajoutez les calculs nécessaires pour estimer l'éclairage indirect via une réflexion spéculaire dans la direction idéale (attention, vous allez avoir besoin d'utiliser les paramètres *depth* et *maxDepth* de la fonction *sendRay* ainsi qu'un peu de récursivité). Configurez votre rendu de manière à effectuer au moins un rebond

afin d'évaluer la composante spéculaire indirecte. Vous devriez voir apparaitre des reflets dans les miroirs. Augmentez le nombre de rebonds et observez le résultat.

b. ACCELERATION DU CALCUL D'INTERSECTION

En testant votre algorithme avec des scènes plus grosses (Cf. 1.c), vous devriez « rapidement » vous rendre compte que votre implémentation n'est pas assez efficace pour effectuer le rendu de telles scènes. Le problème principal en termes de performances vient du fait que vous testiez l'intersection entre le rayon et toutes les facettes de la scène, ce qui est particulièrement couteux en temps de calcul.

Implémentez une structure de données spatiale permettant d'accélérer le calcul d'intersection entre un rayon et la scène. Vous pouvez choisir d'implémenter un arbre de boites englobantes alignées sur les axes (le calcul d'une boite englobante et l'intersection boite englobante rayon vous sont fournies dans la classe *BoundingBox*) ou encore un KD-Tree. L'implémentation l'une de ces structures de données devrait vous permettre de calculer des rendus sur les scènes les plus complexes qui vous sont fournies en des temps très raisonnables (quelques secondes).

c. INTERPOLATION DES NORMALES ET GESTION DES TEXTURES

Interpolation des normales. Vous pouvez certainement remarquer que lorsque vous effectuez votre rendu, les frontières entre facettes ont tendance à être visibles. Cet effet est principalement lié au fait que vous utilisez, dans vos calculs, la normale du triangle (constante pour tous les points du triangle) pour calculer l'éclairement. La classe *Triangle* vous fournit une méthode permettant d'interpoler les normales étant donné une intersection rayon triangle. Utilisez cette méthode pour interpoler la normale au point d'intersection en fonction des normales associées à chaque sommet du triangle. Cela devrait améliorer la qualité du rendu.

Gestion des textures. La plateforme qui vous est fournie charge des fichiers 3DS ainsi que les textures associées aux matériaux utilisés. Elle est d'ailleurs robuste au fait que les textures soient absentes. Pour évaluer la couleur de la texture en un point, il vous faut calculer l'intersection entre un rayon et un triangle et utiliser les cordonnées de texture fournies par le calcul d'intersection. Pour vous simplifier la vie, la classe *Triangle* met à votre disposition la méthode *sampleTexture* qui vous permet d'évaluer la couleur de la texture en fonction des coordonnées (u, v) de l'intersection. Ajoutez la gestion des textures dans votre code.