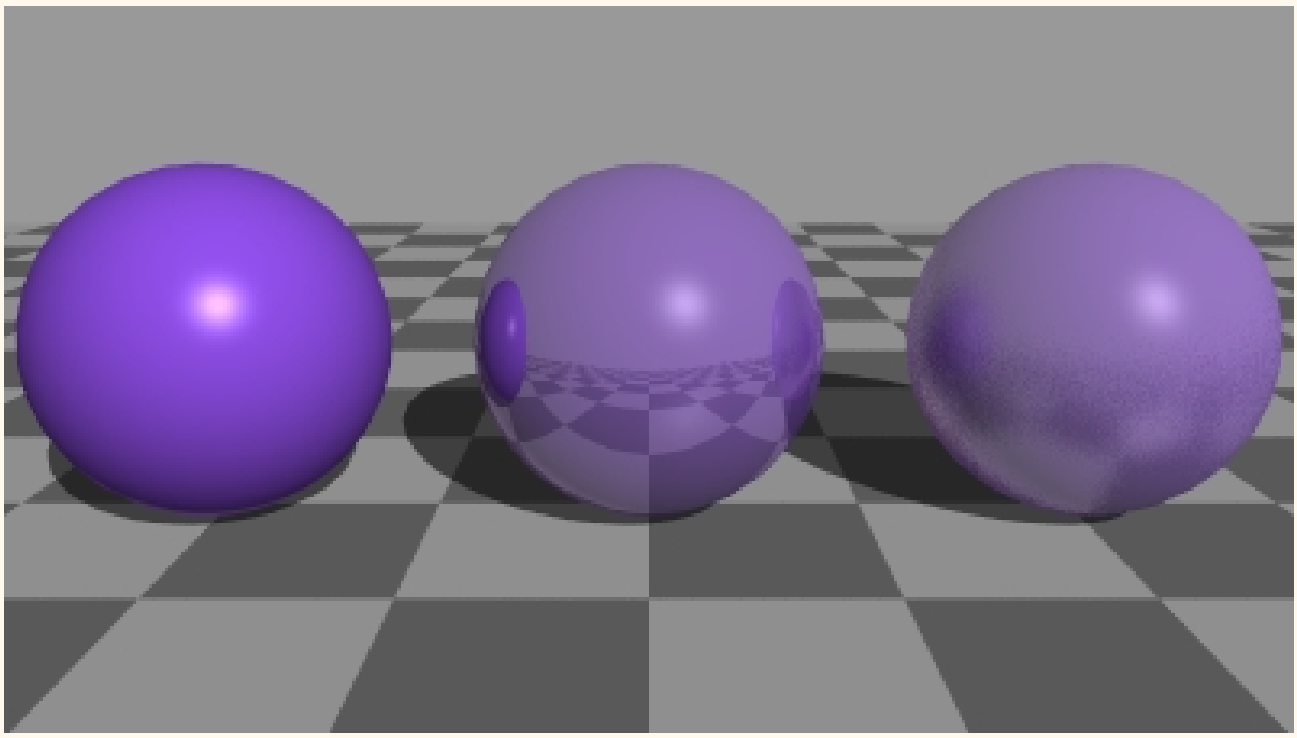

COMMENT IMPLÉMENTER DE NOUVEAUX OBJETS MATERIAL

Projet RayTracer : Marius Pain, Landry Gigant, Thomas Boué & Aubane Nourry

Introduction

L'intégration de nouveaux objets materials pour le projet RayTracer d'Epitech requiert une approche méthodique et structurée, étroitement alignée avec une architecture spécifique afin de garantir sa compatibilité avec le corps du projet, notamment à travers l'utilisation d’interfaces préétablies. Dans ce document, nous explorerons les étapes essentielles pour implémenter de nouveaux objets

materials dans le cadre du projet RayTracer. Nous mettrons en lumière l'importance de respecter les interfaces et conventions établies dans le but de garantir la fonctionnalité et la compatibilité de ces objets avec notre projet, tout en assurant une expérience de développement harmonieuse.

Créer un nouveau objet matérial

Une fois l'objet sélectionné, il est temps de procéder à son intégration dans l'architecture existante du projet RayTracer.

1. Les Prérequis

Vous devrez dans un premier temps :

- Créer une classe principale pour votre objet, celle-ci devra hériter de la classe abstraite : AMaterial.

  La classe abstraite contient toutes les méthodes communes à chaque objet primitif. Elle hérite de l'interface IMaterial.

- Créer un fichier permettant de compiler votre objet en librairie dynamique.
- Mettre à jour l'analyse syntaxique pour pouvoir créer votre objet.

2. Votre classe principale

Afin d'implémenter votre classe principale vous devrez créer les méthodes héritées via la classe abstraite, puis vous verrez ce qu'il vous faudra rajouter pour que votre bibliothèque puisse être chargée avec notre core.

a. Méthodes héritées

En héritant de la classe abstraite AMaterial**,** vous héritez des méthodes suivantes qu'il vous faudra implémenter :

- setColor : Cette méthode permet de définir la couleur de votre objet material.

  La fonction setColor prend en paramètre :

  `Math::Vector3D color` : Correspondant au code RGB de la couleur de votre objet material

  La fonction setColor ne renvoie rien.

- getColor : Cette méthode permet de récupérer la couleur de votre objet material.

  La fonction getColor ne prends rien en paramètre.

  La fonction setColor renvoie :

  `Math::Vector3D color` : Correspondant au code RGB de la couleur de votre objet material

- compute : Cette méthode renvoie la couleur de votre object matérial à un point donné. (elle se base sur sa couleur mais parfois aussi de sa position dans l'espace)

La fonction compute prend en paramètre :

`Math::Point3D point` : Correspondant au point dans l'espace du material

La fonction compute renvoie:

`Math::Vector3D` : Correspondant à la couleur du material.

- reflectLight : Cette méthode permet de savoir si votre objet material reflète la lumière.

La fonction reflectLight ne prend rien en paramètre

La fonction reflectLight renvoie:

`bool` : Correspondant à Vrai s'il reflète la lumière, Faux sinon.

b. Créer des objets material spécifiques

La classe abstraite AMaterial contient uniquement certains attributs d'un objet. En effet, elle contient:

`Math::Vector3D _color` : Correspondant au code RGB de la couleur du material.

`bool _reflectsLight` : Pour savoir si le material reflète la lumière.

c. Externe “C”

Dans votre classe principale, vous allez devoir rajouter une structure “`extern "C"`” qui contient à l’intérieur obligatoirement au moins ces deux fonctions :

- “`Primitive::NomDeVotreClasse *loadMaterialInstance(void)`” : La fonction doit créer une nouvelle instance de votre objet.

- “`const std::string &getName(void)`” : La fonction doit renvoyer le nom de l’objet material que vous avez créé.

Vous pourrez également rajouter les fonctions suivantes dans la structure, mais elles ne sont pas obligatoires :

- “`__attribute__((constructor)) void initsharedlibrary()`” : Cette fonction sera appelée lorsque votre objet sera chargé par le corps, vous ne devez mettre aucune logique nécessaire à votre objet dedans, cependant vous pouvez faire en sorte que cette fonction écrive un message pour dire que votre objet material est chargé.
- “`__attribute__((destructor)) void destroysharedlibrary()`” : Cette fonction sera appelée lorsque votre objet sera déchargé par le corps, vous ne devez mettre aucune logique nécessaire à votre objet dedans, cependant vous pouvez faire en sorte que cette fonction écrive un message pour dire que votre objet material est déchargé.

3. Créer une librairie dynamique

Après de correctement implémenter votre objet material, vous devrez pouvoir compiler le code de votre objet (La classe principale ainsi que la partie externe “C”) pour en faire un seul et même fichier au format “.so” qui va représenter une librairie dynamique.

Pour ce faire, vous allez devoir réaliser un “Makefile”. Ce Makefile est un fichier qui va simplement permettre de compiler votre code de manière automatique et plus simplifiée (pas besoin d’exécuter des lignes de commandes). Ce Makefile devra donc compiler le code, générer un fichier “.so” qui se nommera:

“`libnom_de_votre_classe.so`”: Correspondant au nom du fichier que le corps du programme devra ouvrir pour récupérer votre objet.

Votre Makefile devra également copier le fichier d’en-tête de votre classe dans le dossier adéquat.

Votre Makefile devra enfin déplacer le fichier de librairie dynamique dans le dossier adéquat.

4. Conclusion

Vous avez désormais toutes les clés en main pour implémenter un nouvel objet material pour notre RayTracer, n'hésitez pas à faire une pull-request pour ajouter votre objet si vous souhaitez contribuer au projet. Nous vous remercions par avance de votre contribution.