B4 - OBJECT-ORIENTED PROGRAMMING

RAYTRACER

DOCUMENTATION

Periode: Apr 15, 2024 - May 12, 2024

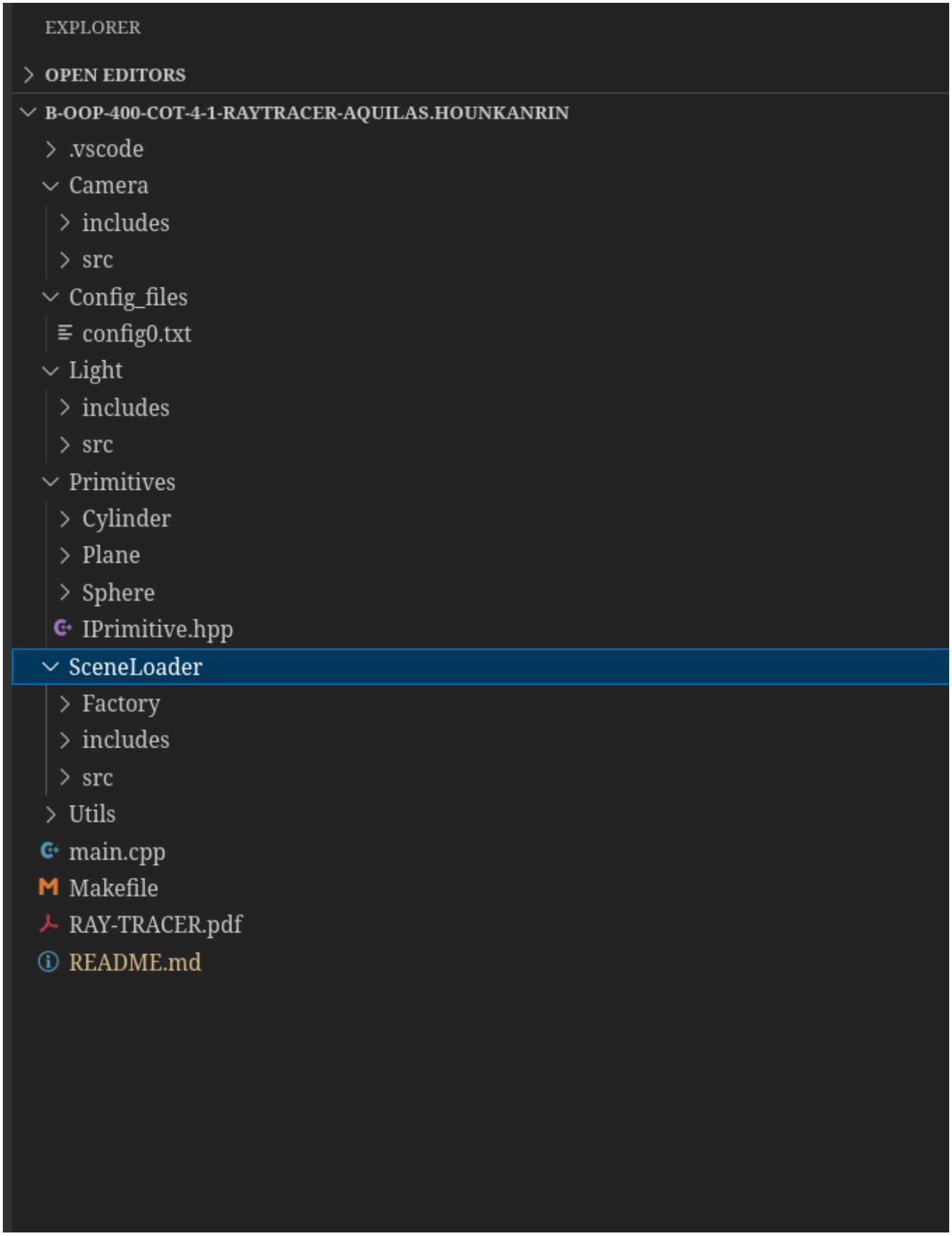
Membres du groupe

Jaures Adehossi	Aquilas Hounkanrin	Seth Djenontin	Ronel Hounkpatin

LE BUT DU PROJET

Le projet consiste à développer un programme en C++ appelé "raytracer" qui utilise la technique du ray tracing pour générer des images numériques réalistes en simulant le trajet inverse de la lumière. L'objectif est de créer un programme capable de générer une image à partir d'un fichier décrivant la scène à représenter.

L'ARCHITECTURE


EXPLORER
OPEN EDITORS
B-OOP-400-COT-4-1-RAYTRACER-AQUILAS.HOUNKANRIN
.vscode
Camera
includes
src
Config_files
config0.txt
Light
includes
src
Primitives
Cylinder
Plane
Sphere
IPrimitive.hpp
SceneLoader
Factory
includes
src
Utils
main.cpp
Makefile
RAY-TRACER.pdf
README.md

CAMERA

Il contient la définition de la classe Camera, comprenant des membres tels que la largeur, la hauteur, la position et la rotation de la caméra, ainsi que des fonctions pour accéder à ces valeurs. Le fichier utilise des directives de préprocesseur pour éviter les problèmes de double inclusion.

Config_files

Ce fichier d'écris une scène 3D pour un projet de rendu graphique ou de traçage de rayons. Voici une explication des différentes parties du fichier :

1. **Camera**: Cette section décrit les paramètres de la caméra utilisée pour rendre la scène. Elle spécifie la résolution de la caméra (largeur et hauteur), sa position et sa rotation dans l'espace 3D, ainsi que son champ de vision (field of view).

- **class Camera**: C'est la déclaration de la classe Camera. Elle semble représenter une caméra dans le système de traçage de rayons.
- **Camera() {};, ~Camera() {};**: Ce sont respectivement le constructeur par défaut et le destructeur de la classe Camera.
- **Camera(double width, double height, Math::Point3D position, Math::Point3D rotation, double fov);**: C'est un autre constructeur de la classe Camera qui prend des paramètres pour définir la largeur, la hauteur, la position, la rotation et le champ de vision de la caméra.
- **double get_width();, double get_height();, double get_fov();:** Ce sont des méthodes pour obtenir respectivement la largeur, la hauteur et le champ de vision de la caméra.
- **void make_ray(double u, double v);, Ray get_ray();:** Ce sont des méthodes pour créer un rayon à partir de coordonnées u et v et pour obtenir le rayon créé.
- **Math::Point3D get_position();, Math::Point3D get_rotation();:** Ce sont des méthodes pour obtenir respectivement la position et la rotation de la caméra.
- **void print_data():** C'est une méthode pour imprimer les données de la caméra.
- **double _width = 0.0;, double _height = 0.0;, double _fov = 0.0;:** Ce sont des membres de données privés de la classe Camera pour stocker respectivement la

largeur, la hauteur et le champ de vision de la caméra. Les valeurs sont initialisées à 0 par défaut.

- **Math::Point3D _position;, Math::Point3D _rotation;:** Ce sont des membres de données privés de la classe Camera pour stocker respectivement la position et la rotation de la caméra.
- **Ray ray;:** C'est un objet de type Ray qui semble être utilisé pour stocker le rayon créé par la caméra.

2. **Primitives**: Cette section décrit les objets primitifs présents dans la scène. Dans cet exemple, il y a des sphères et des plans. Chaque sphère est définie par ses coordonnées (x, y, z) du centre, son rayon (r) et sa couleur (rouge, vert, bleu). Les plans sont définis par leur axe (X, Y ou Z) et leur position par rapport à cet axe, ainsi que leur couleur.
3. **Lights**: Cette section décrit les lumières utilisées dans la scène. Il y a des lumières ambiante et diffuse, ainsi que des lumières ponctuelles et directionnelles. Les lumières ponctuelles sont définies par leurs coordonnées (x, y, z) dans l'espace 3D.

Dans l'ensemble, ce fichier décrit la configuration de base d'une scène 3D avec des objets primitifs, une caméra et des lumières, prête à être rendue ou tracée en utilisant un moteur de rendu ou un moteur de traçage de rayons.

Lights

Dans ce dossier, nous avons le fichier hpp et le cpp:

- La classe Light contient des membres privés pour stocker les valeurs ambiantes et diffuses, ainsi que des vecteurs de points et de vecteurs directionnels pour les lumières.

- Les membres publics comprennent des constructeurs, destructeurs et fonctions d'accès aux données membres.
- Le fichier utilise des directives de préprocesseur pour éviter les problèmes de double inclusion.
- Il inclut également des en-têtes nécessaires pour les dépendances de la classe Light.

Primitives

Dans ce dossier nous avons la classe IPrimitive dont hérite toutes nos primitives.

PLANE	CYLINDER	SPHERE
1. **Déclaration de la classe Plane :** La classe Plane est déclarée et elle hérite de l'interface IPrimitive. La classe Plane implémente toutes les méthodes virtuelles pures définies dans l'interface IPrimitive. 2. **Constructeur :** Il prend en paramètres une chaîne de caractères axis, un double position et un objet Math::Point3D color. Ont l'utilise pour créer des instances de la classe Plane. 3. **Destructeur :** Le destructeur ~Plane() est déclaré. 4. **Méthode hits() :** La méthode hits() est déclarée. Cette méthode est responsable de déterminer si un rayon intersecte le plan et de calculer les informations sur l'intersection, telles que le point d'intersection et la normale à la surface. Dans ce cas, la méthode est définie directement dans la déclaration de la classe avec une implémentation de retour rapide, indiquant que le plan n'est pas affecté par les rayons 5. **void print_data()**; : C'est une méthode qui imprime les données du plan, comme son		1. **Déclaration de la classe Sphere :** La classe sphere est déclarée et elle hérite de l'interface IPrimitive. La classe Plane implémente toutes les méthodes virtuelles pures définies dans l'interface IPrimitive. 2. **Constructeur et destructeur** : ● **Sphere(Math::Point3D, double, Math::Point3D):** Le constructeur prend trois paramètres : la position de l'origine de la sphère (Math::Point3D), le rayon de la sphère (double) et la couleur de la sphère (Math::Point3D). ● **~Sphere():** Le destructeur de la classe. 3. **~Sphere();**: C'est le destructeur de la classe Sphere. Il est appelé lorsque l'objet Sphere est détruit, par exemple lorsque sa portée (scope) se termine. 4. **bool hits(RayTracer::Ray, double, double, Hit_info &hit_point)** C'est une méthode de la classe Sphere qui semble vérifier si un rayon donné interagit avec la sphère. Elle prend en paramètres un objet Ray, deux doubles (peut-être des bornes pour un intervalle de temps) et une référence à un objet Hit_info pour stocker les informations sur le point d'impact. 5. **void print_data();:** C'est une méthode de la classe Sphere qui semble imprimer les données de la sphère.

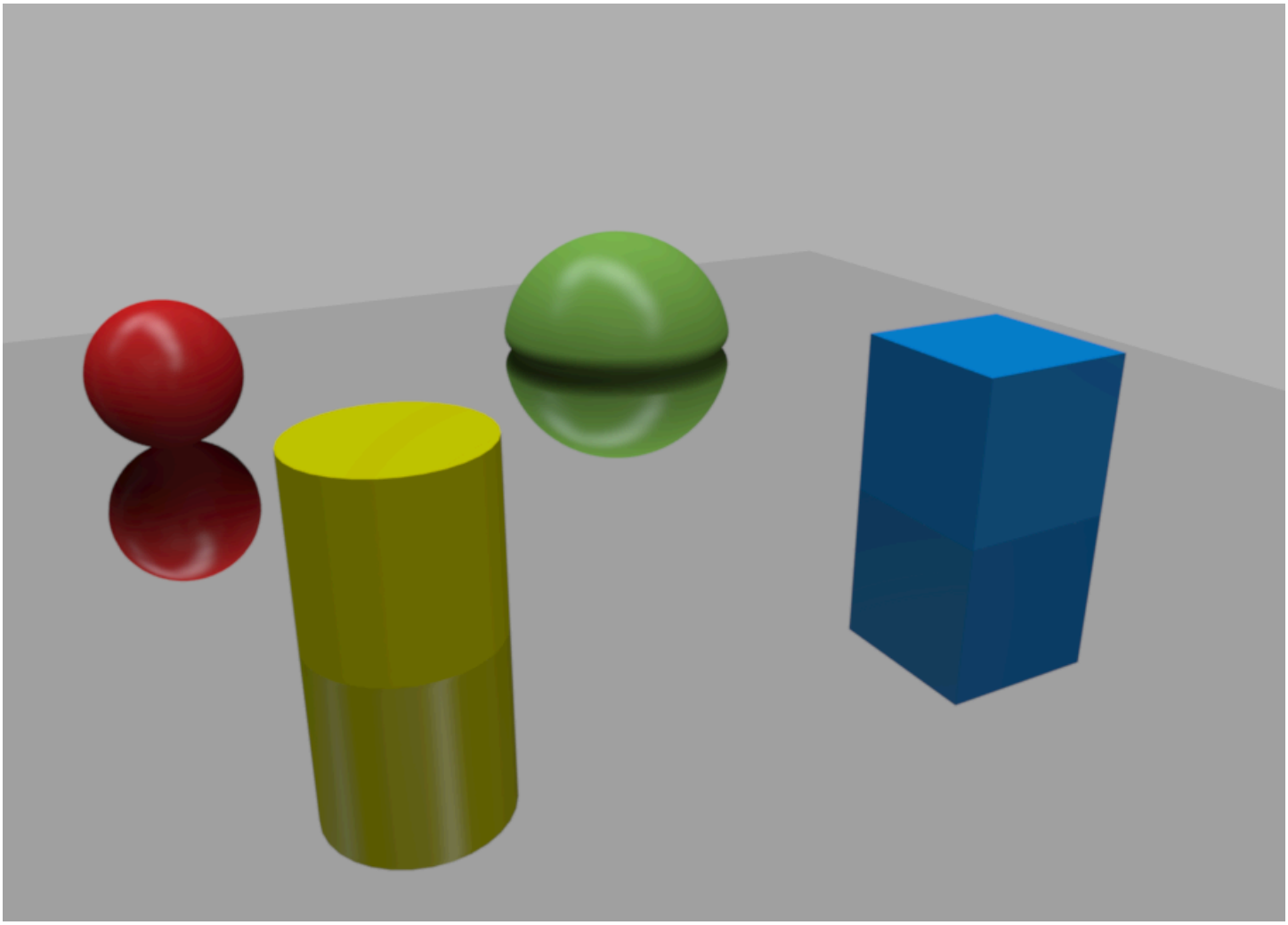
axe, sa position et sa couleur.

6. **std::string get_axis()**; : C'est une méthode qui retourne l'axe du plan (par exemple, "X", "Y" ou "Z").

7. **Double get_position(); :** C'est une méthode qui retourne la position du plan le long de son axe.

6. **Math::Point3D get_origin();, Math::Point3D get_color();, double get_radius();:** Ce sont des méthodes de la classe Sphere pour obtenir respectivement l'origine, la couleur et le rayon de la sphère.

7. **Math::Point3D origin;, Math::Point3D color;, double radius;:** Ce sont des membres de données privés de la classe Sphere qui stockent respectivement l'origine, la couleur et le rayon de la



SceneLoader

Dans ce dossier, nous avons ajouter une Factory.

Nous une classe Factory utilisée pour créer des primitives, telles que des sphères et des plans, pour un projet de traçage de rayons. Voici un résumé du contenu :

- La classe Factory contient des méthodes pour créer des primitives, notamment des sphères et des plans.
- Les données nécessaires pour créer une sphère ou un plan sont stockées dans des structures data_sphere et data_plane.
- La méthode createPrimitive prend en paramètre le type de primitive à créer et les paramètres nécessaires, puis utilise des instructions conditionnelles pour appeler les méthodes de création appropriées.
- Les méthodes createSphere et createPlane créent respectivement des sphères et des plans en utilisant les données fournies.
- Chaque méthode crée une primitive à l'aide du constructeur approprié et renvoie un pointeur unique vers cette primitive.

Ce fichier définit une usine pour créer des primitives pour une scène de traçage de rayons, en utilisant des paramètres spécifiques pour chaque type de primitive.

Parsing

Nous avons une class SceneLoader qui contient plusieurs méthodes permettant de loader la scene dont :

- load_camera qui load la camera
- load_primitive qui load toutes les primitives
- load_sphere qui load la sphere
- load_plane qui load la plane
- load_lights qui load tous les types de lumières
- load_point_light qui load les lumières ponctuelles
- load_directional_light qui load les lumières directionnelles
- read_config_file qui load le fichier
- get_camera qui retourne la camera
- get_primitives qui retourne toutes les primitives
- get_light qui retourne la lumière

Utils

Dans ce dossier, nous avons ajouté nos propres fonctions dont nous aurons besoin pour le projet