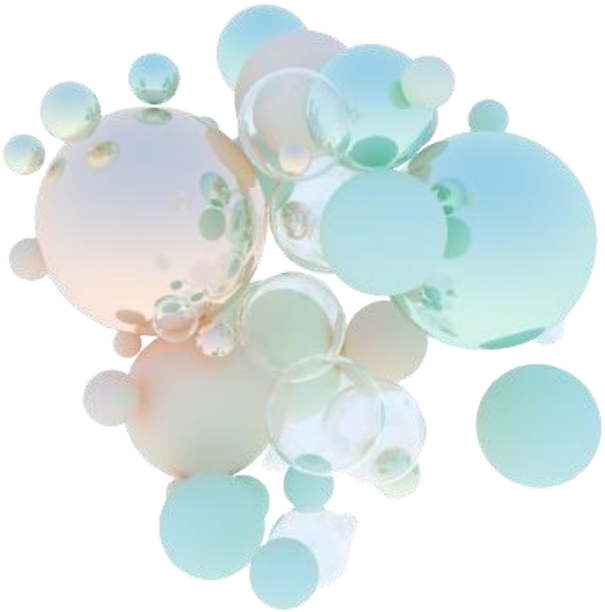
Documentation Raytracer



Raytracer est un **programme capable** de générer une image à partir d'un fichier de configuration décrivant une scène.

Ce fichier contient toutes les informations nécessaires pour construire **l'environnement 3D**, y compris la **position** et la **forme** des objets, la disposition de la **lumière**, et les **matériaux** des objets.

Le programme utilise ces informations pour simuler comment **la lumière se propagerait** à travers cet **environnement**, en suivant le chemin inverse de la lumière depuis l'œil de l'observateur jusqu'à **chaque pixel de l'image finale**.

Cette documentation s'adresse à tout développeur désirant ajouter une nouvelle composante, primitive, lumière ou bien matériel, à ce programme.

Alexandre Ghreagajlou Arnaud Niclot Caroline Boilly Clément Piasco

EPITECH Nice | Promotion 2027 - 2ème année

Sommaire

I. Installation du projet

Étape 1

Cloner le repository

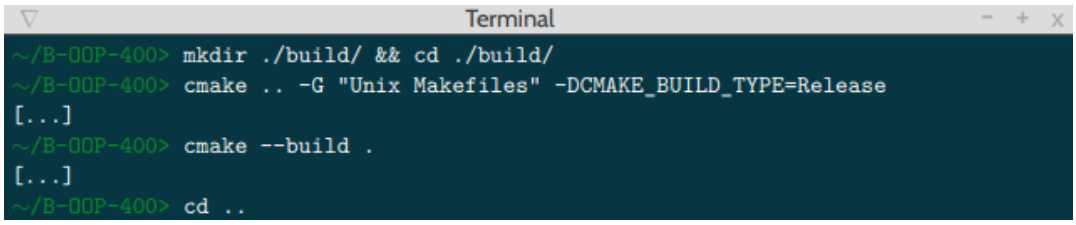
Étape 2

Installer les différents prérequis du projet

- GCC (https://gcc.gnu.org/install/)
- CMake (https://cmake.org/download/)
- SDL2 (https://github.com/libsdl-org/SDL/releases/tag/release-2.30.2)
- SDL-image (https://github.com/libsdl-org/SDL_image/releases)

Étape 3

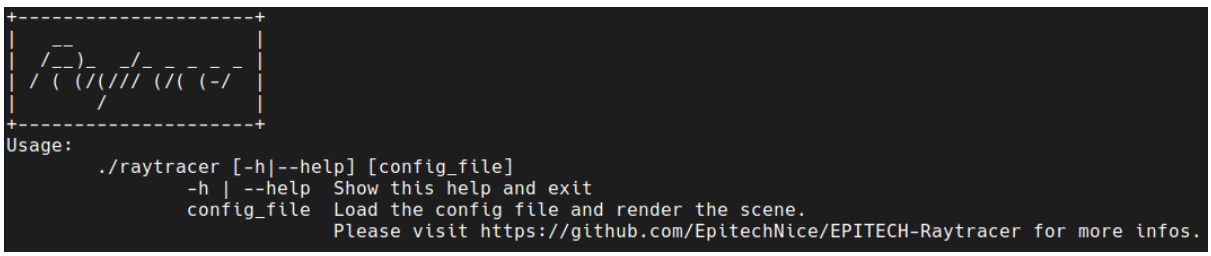
Compiler le projet



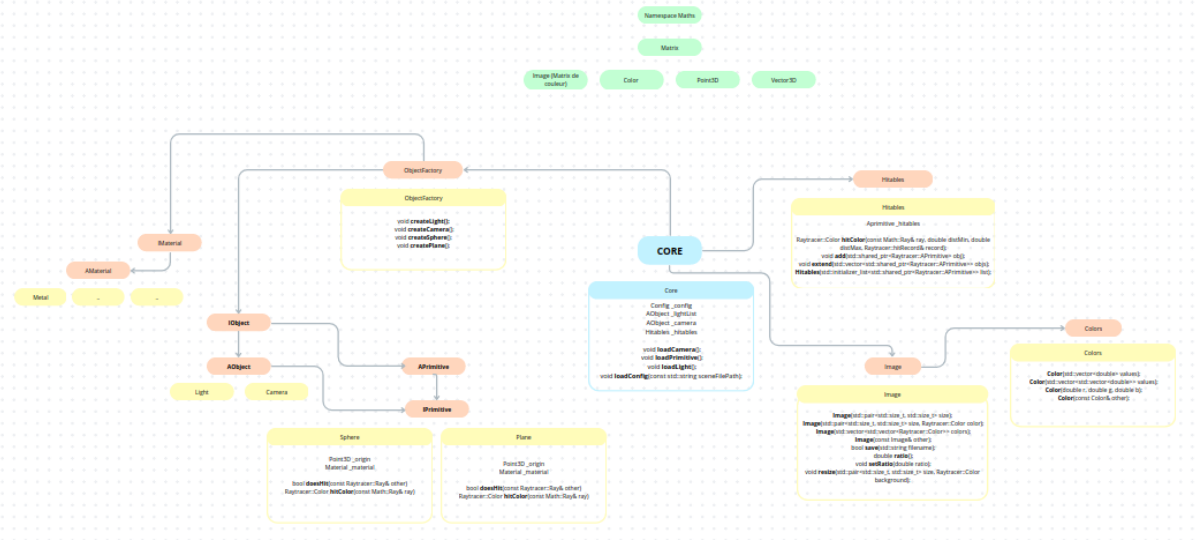
puis **`make`**

Étape 4

Lancer le projet



II. **Architecture**



III. **Fichier de configuration**

Pour créer une scène, il faut ajouter des objets à un fichier de configuration *.conf* que l'on inclue au lancement du programme. Pour qu'une scène s'affiche, il faut au minimum une caméra et une lumière.

```
# Configuration de la caméra      You, 7 days ago • [~] Two perfect balls (todo : improve code for …
camera:
{
    resolution = { width = 800, height = 400 };
    position = { x = 0.0, y = 0.0, z = 0.0 };
    rotation = { x = 0.0, y = 0.0, z = 0.0 };
    fieldOfView = 72.0;
};

# Primitives dans la scène
primitives:
{
    # Liste des sphères
    spheres = (
        { position = {x = 0.0, y = 0.0, z = -1.0}, radius = 0.5, color = { r = 255, g = 255, b = 64 }, material = "matte" },
        { position = {x = 0.0, y = -100.5, z = -1.0}, radius = 100.0, color = { r = 127, g = 255, b = 122 }, material = "matte" },
        { position = {x = 1.0, y = 0.0, z = -1.0}, radius = 0.5, color = { r = 151, g = 122, b = 255 }, material = "metal" },
        { position = {x = -1.0, y = 0.0, z = -1.0}, radius = 0.5, color = { r = 251, g = 122, b = 255 }, material = "metal" }
    );

    planes = (
        # { position = {x = 0.0, y = -200.0, z = -1.0}, size = 0.8, color = { r = 255, g = 200, b = 200 }, material = "matte" }
        # { position = {x = 20.0, y = 55.0, z = 62.0}, size = 6.0, color = { r = 255, g = 200, b = 200 } }
    );

};

# Configuration des lumières
lights:
{
    ambient = 0.4;
    diffuse = 0.6;

    # Liste des lumières ponctuelles
    light = (
        { point = {x = 400.0, y = 100.0, z = 500.0}, direction = {x = 360.0, y = 360.0, z = 360.0}, angle = 90.0 }
    );
};
```

Pour ajouter la caméra, il faut attribuer ces valeurs :

- Résolution
- Position {x, y, z} : origine spatiale de la caméra
- Rotation {x, y, z} :

Pour ajouter une primitive, il faut préciser le type (Sphere, Plane) et attribuer ces valeurs :

- Position {x, y, z} : origine spatiale de l'objet
- Radius, Size : propriété intrinsèque d'un objet
- Couleur {r, g, b} : couleur de l'objet
- Matériau : propriété physique de l'objet

IV. Interface IObject

Pour ajouter un nouvel Object (Caméra, Light...) il faudra créer une nouvelle classe héritant de AObject.

Les différentes classes d'Object existants sont :

/Object/Camera.cpp

/Object/Lights.cpp

Ci-dessous le descriptif des différentes méthodes de IObject à implémenter.

setPosition(const Math::Point3D& position) : set et modifie les valeurs de position d'un Object

setDirection(const Math::Point3D& direction) : set et modifie les valeurs de direction d'un Object

getPosition() : récupéré les valeurs de position de l'Object

getDirection() : récupéré les valeurs de direction de l'Object

getClassName() : Fonction de Debug pour récupérer le nom de classe de l'Object

Str() : Fonction de Debug pour afficher en format string les infos de l'Object

V. Interface IPrimitive

Pour ajouter un nouveau primitif, il faudra créer une nouvelle classe propre au jeu qui hérite de **APrimitive** et **IObject**.

Les différents primitifs existants sont :

`/Primitif/Plane.cpp`

`/Primitif/Sphere.cpp`

Ci-dessous le descriptif des différentes méthodes de IPrimitive à implémenter.

Plane(Math::Point3D origin, Raytracer::Material material, double size) : constructeur du Plane avec un point d'origine, une couleur, et une taille.

Sphere(Math::Point3D origin, Raytracer::Material material, double radius): constructeur du Sphere avec un point d'origine, une couleur et un radius.

DoesHit(const Math::Ray& other, double distMin, double distMax, hitRecord& record): vérifie si le rayon touche l'Object et représente la fonction centrale des lois physiques d'un objet dans l'espace.

Bouce(const Math::Ray& other): renvoie le rayon du rebond sur l'Object

GetSize(): renvoie la taille du Plane (propre a Plane)

GetRadius(): renvoie le radius de la Sphere (propre à Sphere).

VI. Hittables

Hittables est une classe qui contient un vector de Primitives. C'est à partir de cette classe que les propriétés physiques d'un objet dans l'espace seront appelées (est-ce qu'un rayon touche un objet).

Ci-dessous le descriptif des différentes méthodes de Hittables à implémenter.

DidHit(const Math::Ray& ray, double distMin, double distMax, Raytracer::hitRecord& record): check si le rayon touche l'objet

HitColor(const Math::Ray& ray, double distMin, double distMax, Raytracer::hitRecord& record) : renvoie la couleur du pixel touché par le rayon

Add(std::shared_ptr<Raytracer::APrimitive> obj): ajouter un primitif à la liste de Primitives de Hittables.

Extend(std::vector<std::shared_ptr<Raytracer::APrimitive>> objs): ajouter plusieurs primitifs à la liste de Primitives de Hittables.