

# 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.

## Sommaire

I. Installation du projet

II. Architecture

III. Fichier de configuration

IV. Énumération et structures

V. Interface IObject

VI. Interface IPrimitive

VII. Hittables

# I. Installation du projet

## Étape 1

Cloner le repository

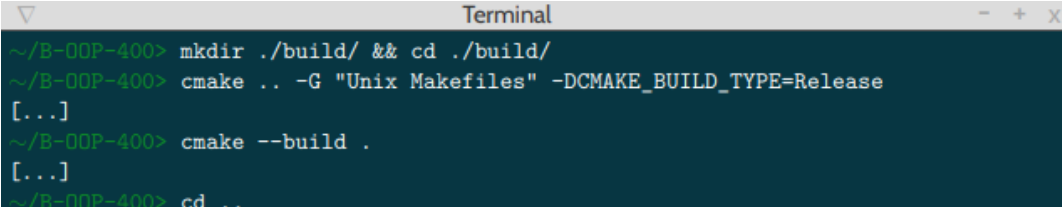
## Étape 2

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](https://github.com/libsdl-org/SDL_image/releases))

## Étape 3

Compiler le projet

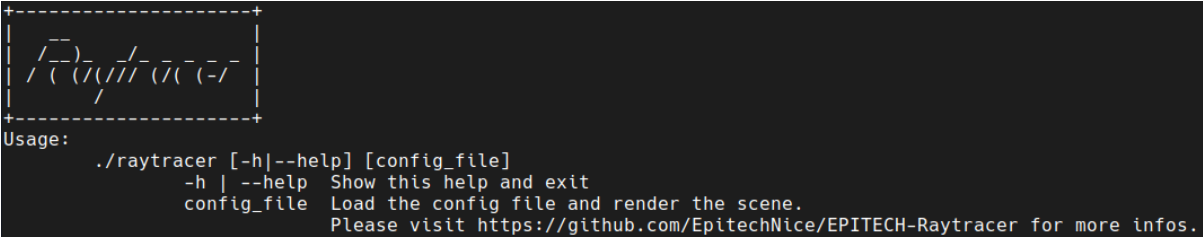
A terminal window titled "Terminal" with a dark background. It shows a series of commands being executed in a shell. The user is in a directory named ~/B-00P-400. The commands are: mkdir ./build/ && cd ./build/, cmake .. -G "Unix Makefiles" -DCMAKE\_BUILD\_TYPE=Release, cmake --build ., and cd .., with ellipses indicating intermediate output.

```
~/B-00P-400> mkdir ./build/ && cd ./build/
~/B-00P-400> cmake .. -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release
[...]
~/B-00P-400> cmake --build .
[...]
~/B-00P-400> cd ..
```

puis make

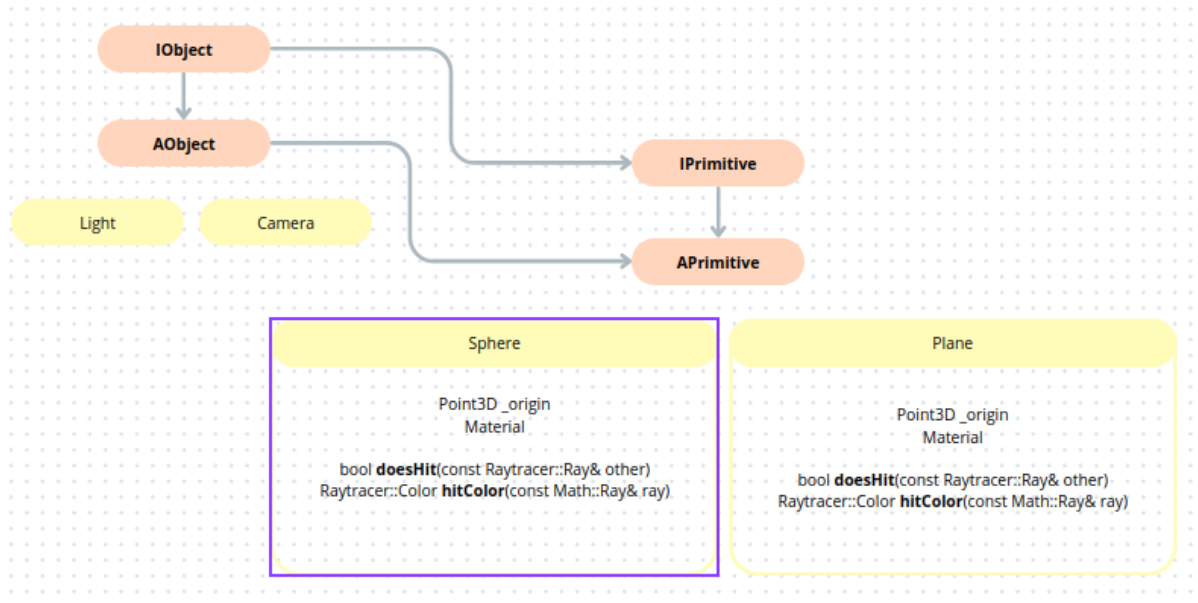
## Étape 4

Lancer le projet

A terminal window showing the output of a raytracer program. It displays a dashed box containing a simple line drawing of a sphere and a cylinder. Below the drawing, the usage instructions for the program are shown.

```
+-----+
| /--\  | /--\  | /--\  | /--\  |
| /  (  (//  (//  (//  (//  (//  |
| /    /    /    /    /    /    |
+-----+
Usage:
./raytracer [-h|--help] [config_file]
-h | --help  Show this help and exit
config_file  Load the config file and render the scene.
Please visit https://github.com/EpitechNice/EPITECH-Raytracer for more infos.
```

## 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
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 = 64, b = 64 }, material = "matte" },
        { position = {x = 0.0, y = -100.5, z = -1.0}, radius = 100.0, color = { r = 255, g = 0, b = 64 }, material = "matte" },
        { position = {x = 1.0, y = 0.0, z = -1.0}, radius = 0.5, color = { r = 255, g = 0, b = 64 }, material = "metal" }
    );

    planes = (
        { position = {x = 7.0, y = 5.0, z = 2.5}, size = 0.8, color = { r = 255, g = 200, b = 200 } },
        { 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 **nouvelle Object (caméra, light...)** il faudra créer une nouvelle classe propre à l'objet que l'on veut créé héritant de IObject.

Les différente classe d'objects existant 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 positions d'un object

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

**getPosition():** récupéré les valeurs de positions de l'object

**getDirection():** récupéré les valeurs de directions de l'object

**getClassName():** Fonction de Debug pour récupéré 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 **IObject**

Les différents primitif existant 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) :** créé un object Plane avec un point d'origine, une couleur, et une taille.

**Sphere(Math::Point3D origin, Raytracer::Material material, double radius):** créé un objet sphere avec un point d'origine, une couleur et un radius.

**DoesHit(const Math::Ray& other, double distMin, double distMax, hitRecord& record):** check si le rayon touche l'objet.

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

**GetSize():** renvoie la taille de mon plane

**GetRadius():** renvoie le radius de ma sphere

## VI. Hittables

Pour stocker les Objets physique créé (primitif), il faut les ajouter à la class Hittables qui va servir à virifier si l'objet est toucher par le rayon.

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 toucher par le ray

**Add(std::shared\_ptr<Raytracer::APrimitive> obj):** ajouter un primitif à la list

**Extend(std::vector<std::shared\_ptr<Raytracer::APrimitive>> objs):** ajouter plusieurs primitif à la list