Raytracer Documentation

Gwenaël HUBLER - Roman LOPES - Stéphane FIEVEZ - Alexandre DE FREITAS MARTINS

Présentation du projet

La présente documentation technique introduit le projet Raytracer en C++, réalisé pour l'école Epitech. Ce projet vise à développer un moteur de rendu utilisant le raytracing pour générer des images réalistes en simulant le comportement des rayons lumineux dans une scène 3D. Cette documentation fournira une vue d'ensemble de l'architecture logicielle, des fonctionnalités et des techniques utilisées dans le projet.

L'architecture modulaire du code du projet Raytracer en C++ offre aux développeurs une grande flexibilité pour étendre ses fonctionnalités. Grâce à une conception soigneusement pensée, il est aisé d'ajouter de nouveaux éléments tels que des primitives géométriques, des sources lumineuses, des matériaux ou même des bibliothèques graphiques supplémentaires.

Chaque composant est indépendant et peut être intégré de manière transparente dans le moteur de rendu, permettant ainsi aux développeurs d'explorer différentes approches et d'enrichir facilement les capacités du Raytracer selon leurs besoins spécifiques.

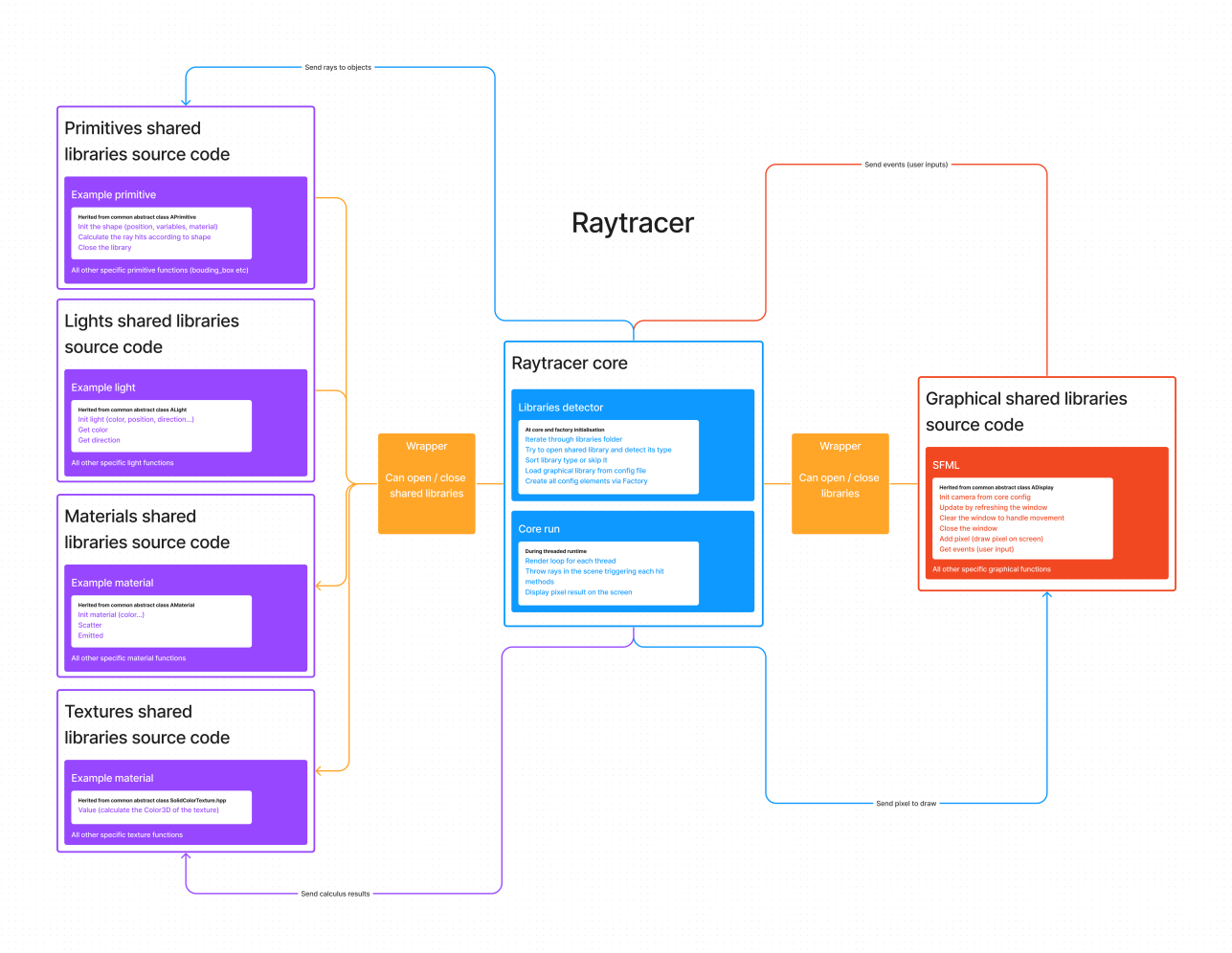

SCHÉMA TECHNIQUE DU FONCTIONNEMENT DU PROGRAMME

Installation des pré-requis (Ubuntu)

INSTALLATION DE SFML

```
sudo apt-get install libsfml-dev
```

INSTALLATION DE LIBCONFIG++

```
sudo apt install libconfig++-dev
```

Compilation et exécution

COMPILATION VIA CMAKE

```
mkdir build

cd build

cmake ..

make

cd ..
```

EXÉCUTION DU BINAIRE COMPILÉ

```
./raytracer scene/DESIRED_SCENE.cfg
```

Remplacer DESIRED_SCENE.cfg par la scene désirée pour le rendu 3D. Eg. ./raytracer scene/sceneTuto.cfg

Contrôles du programme

- **T**: Screenshot -> screenshot/screenshot.png
- **R**: Rafraîchir la scene
- **SPACE**: Mettre le rendu en pause
- **ESC**: Quitter Raytracer

Documentation destinée aux développeurs

EMPLACEMENT DES SCENES
Les scenes sont des fichiers de configuration servant à définir les objets ainsi que toutes leurs propriétés au sein d'un même fichier. Elles sont situés à :

`/scenes/nom_scene.cfg`

Ajout d'une nouvelle primitive au programme

EMPLACEMENT DU CODE SOURCE
Le code source du jeu est à ajouter de la manière suivante :

`/src/Primitives/NomDeLaPrimitive.hpp`

`/src/Primitives/NomDeLaPrimitive.cpp`

CRÉATION DE LA CLASSE DE PRIMITIVES ET HÉRITAGE
Chaque primitive hérite de la classe abstraite APrimitive.hpp

La classe de la primitive nouvellement crée va hériter de fonctions de base :

void hit(const RayTracer::Ray& r, double t_min, double t_max, hit_record& rec) : Cette fonction sert gérer la manière dont la primitive va réagir en fonction de sa forme.

Ajout d'une nouvelle matière au programme

EMPLACEMENT DU CODE SOURCE
Le code source de la matière est à ajouter de la manière suivante :

`/src/Materials/NomMatière.hpp`

`/src/Materials/NomMatière.cpp`

CRÉATION DE LA CLASSE DE LA MATIÈRE ET HÉRITAGE
Chaque matière hérite de la classe abstraite AMaterial.hpp

La classe de la matière nouvellement crée va hériter de fonctions de base :

bool scatter(const RayTracer::Ray & r_in, const RayTracer::hit_record& rec, color& attenuation, RayTracer::Ray& scattered) : Cette méthode est utilisée pour simuler la diffusion des rayons de lumière lorsqu'ils interagissent avec des objets en utilisant des propriétés optiques spécifiques.

Ajout d'un nouveau type de lumière au programme

EMPLACEMENT DU CODE SOURCE
Le code source de la lumière est à ajouter de la manière suivante :

`/src/Lights/NomLumière.hpp`

`/src/Lights/NomLumière.cpp`

CRÉATION DE LA CLASSE DE LA LIBRAIRIE GRAPHIQUE ET HÉRITAGE
Chaque jeu hérite de la classe abstraite ALight.hpp

La classe du type de lumière nouvellement crée va hériter de fonctions de base :

Vector3D getPosition() : Cette fonction sert à retourner la position de la lumière.

Vector3D getDirection(const point3& point) : Cette fonction sert à retourner la direction de la lumière dans le cas applicable (ex. DirectionalLight)

color getColor() : Cette fonction permet de retourner la couleur de la lumière construite.

Ajout d'une nouvelle librairie graphique au programme

EMPLACEMENT DU CODE SOURCE
Le code source de la librairie graphique est à ajouter de la manière suivante :

`/src/Libraries/Graphical/NomLibrairie/`

`/src/Libraries/Graphical/NomLibrairie/NomLibrairie.hpp`

`/src/Libraries/Graphical/NomLibrairie/NomLibrairie.cpp`

CRÉATION DE LA CLASSE DE LA LIBRAIRIE GRAPHIQUE ET HÉRITAGE
Chaque jeu hérite de la classe abstraite AGraphical.hpp

La classe de la librairie nouvellement crée va hériter de fonctions de base :

void update() : Cette fonction sert a mettre à jour l'affichage. Peut être utilisé pour appeler this->display

void clear() : Cette fonction permet de nettoyer l'écran.

void close() : Cette fonction permet de fermer la fenêtre.

void display() : Cette fonction permet d'afficher les pixels.

bool isOpen() : Cette fonction permet de savoir si la fenêtre est ouverte ou non.

bool addPixel(int x, int y, double r, double g, double b, int samples_per_pixel) : Cette fonction permet de placer un pixel sur la fenêtre de rendu.

EventType getEvents() : Cette fonction permet de récupérer les events de la librairie.

Ajout d’une nouvelle texture au programme

EMPLACEMENT DU CODE SOURCE
Le code source de la texture est à ajouter de la manière suivante :

`/src/Texture/NomTexture.hpp`

`/src/Texture/NomTexture.cpp`

CRÉATION DE LA CLASSE DE LA TEXTURE ET HÉRITAGE
Chaque jeu hérite de la classe abstraite SolidColorTexture.hpp

La classe de la librairie nouvellement crée va hériter de fonctions de base :

Color3D value() : Cette fonction sert a calculer la Couleur 3D correspondante de la texture pour l’affichage