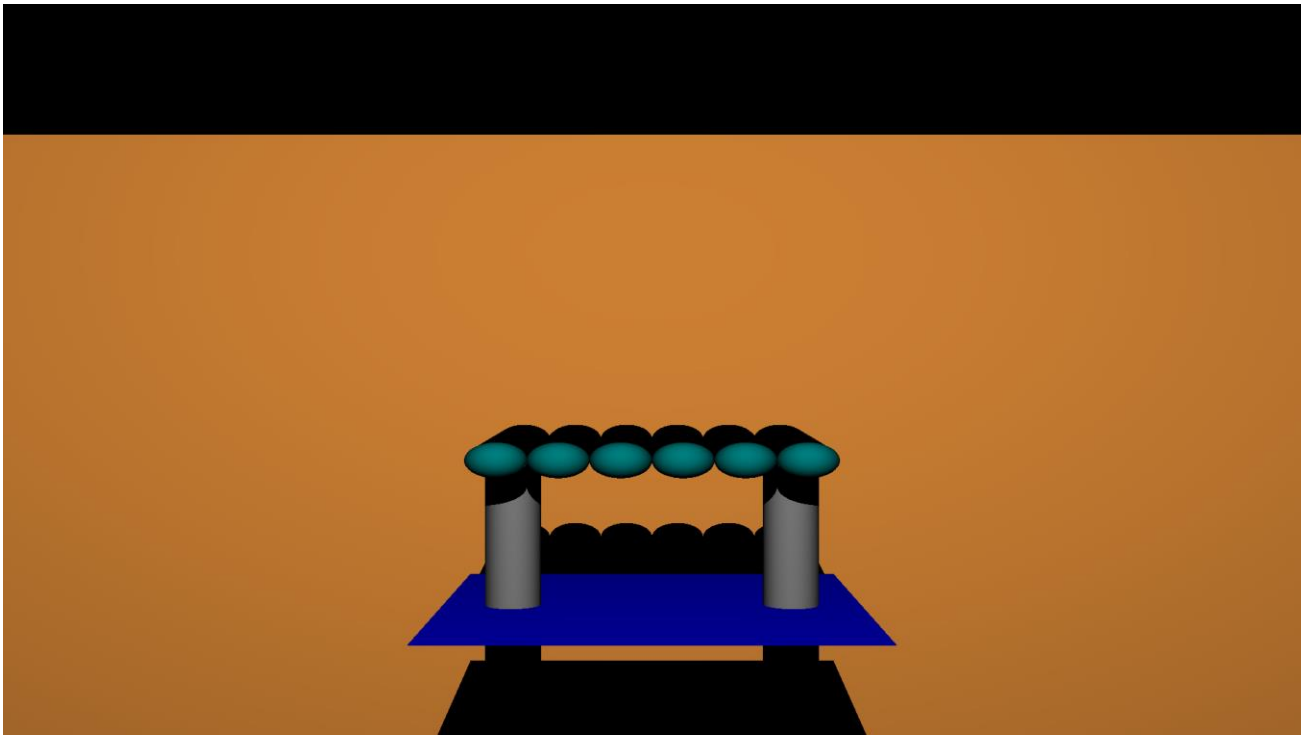
Documentation RayTracer

Introduction du projet

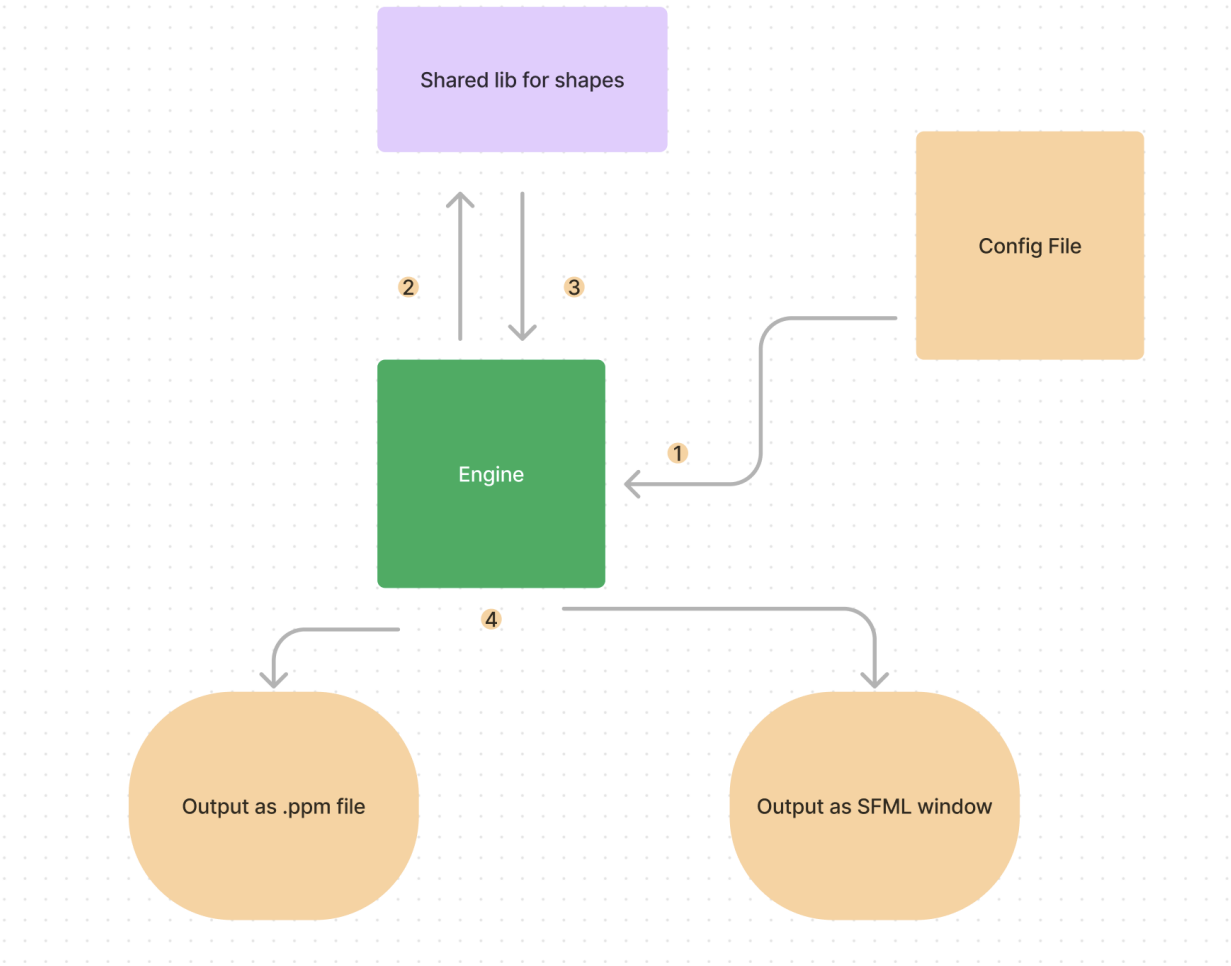
Le projet Raytracer consiste à créer un programme en C++ capable de générer des formes géométriques (tel que des sphères, des cônes ou des cylindres) et de jouer avec les lumières en affichant le tout en SFML (une librairie graphique). Le programme d'effectuer des transformations sur les formes géométriques tel que des translations, des rotations. Le programme doit également permettre de configurer la caméra et la scène ainsi que de sauvegarder l'image dans un fichier .ppm.



1) Engine

L'Engine est le moteur de ce RayTracer, il permet de faire tourner les affichages en SFML et de parser les fichiers .cfg qui contiennent

toutes les informations nécessaires à la création des objets qui figureront dans le RayTracer.



Au démarrage du programme, la fonction “loadConfig” permet la création des objets qui seront créer (les formes géométriques et les lumières). Elle appelle aussi une fonction qui permet de configurer la caméra grâce au fichier configuration.cfg.

```cpp
void Engine::loadConfig(const std::string &configFile) {
    libconfig::Config cfg;
    try {
        cfg.readFile(configFile.c_str());
        const libconfig::Setting &root = cfg.getRoot();
        loadLightConfig(root["light"]);
        loadObjectConfig(root["objects"]);
        setupCamera(root["camera"]);
    } catch (const libconfig::FileIOException &fioex) {
        throw ConfigException("Erreur lors de la lecture du fichier de configuration");
    } catch (const libconfig::ParseException &pex) {
        throw ConfigException(
                "Erreur de syntaxe dans le fichier de configuration à la ligne " + std::to_string(pex.getLine()));
    } catch (const libconfig::SettingNotFoundException &nfex) {
        throw ConfigException("Erreur de configuration : " + std::string(nfex.getPath()));
    }
}
```

- Run:

La fonction “run” va durer indéfiniment tant que la fenêtre est ouverte. Elle va ensuite regarder si la configuration du fichier .cfg reste inchangé à l’aide de la fonction “checkConfigChanged”. Puis Elle va s’occuper de l’affichage grâce à la fonction “render” et gérer les évènements de la SFML grâce à la fonction “processEvents”. A la fin du programme la dernière image se sauvegarde dans un .ppm dans “saveToPPM”.

```cpp
void Engine::run() {
    while (window.isOpen()) {
        if (checkConfigChanged()) {
            std::cout << "Configuration file has changed. Reloading and restarting rendering..." << std::endl;
            reloadConfig();
        }
        render();
        processEvents();
    }
    saveToPPM(outputFileName);
}
```

- Render:

La fonction “render” genere l’affichage des pixels en les créant en appelant la fonction “renderPixel” qui va s’occuper de la création des pixels des formes géometrique. Ensuite elle s’occupe d’afficher en SFML tous les pixels créer.

```
void Engine::render() {
    unsigned num_threads = std::thread::hardware_concurrency();
    std::vector<std::thread> threads(num_threads);
    const unsigned int num_pixels = camera.imageWidth * camera.imageHeight;
    const unsigned int pixels_per_thread = num_pixels / num_threads;

    for (unsigned int t = 0; t < num_threads; ++t) {
        unsigned int start = t * pixels_per_thread;
        unsigned int end = (t + 1) * pixels_per_thread;
        if (t == num_threads - 1) {
            end = num_pixels;
        }
        threads[t] = std::thread(&Engine::renderPixel, this, start, end);
    }
    for (auto &thread: threads) {
        thread.join();
    }
    window.clear();
    window.draw(pixels);
    window.display();
}
```

- RenderPixel:

Cette fonction permet de créer les pixels et d’appeler la fonction render2 qui permettra la création des lumières et des objets (les formes géométriques). La fonction render2 appellera la fonction RayTrace qui va calculer la couleur de chaque pixel en fonction de la lumière.

- SaveToPPM:

Cette fonction va créer un fichier .ppm pour y sauvegarder la dernière image afficher par le programme.

2) Shapes

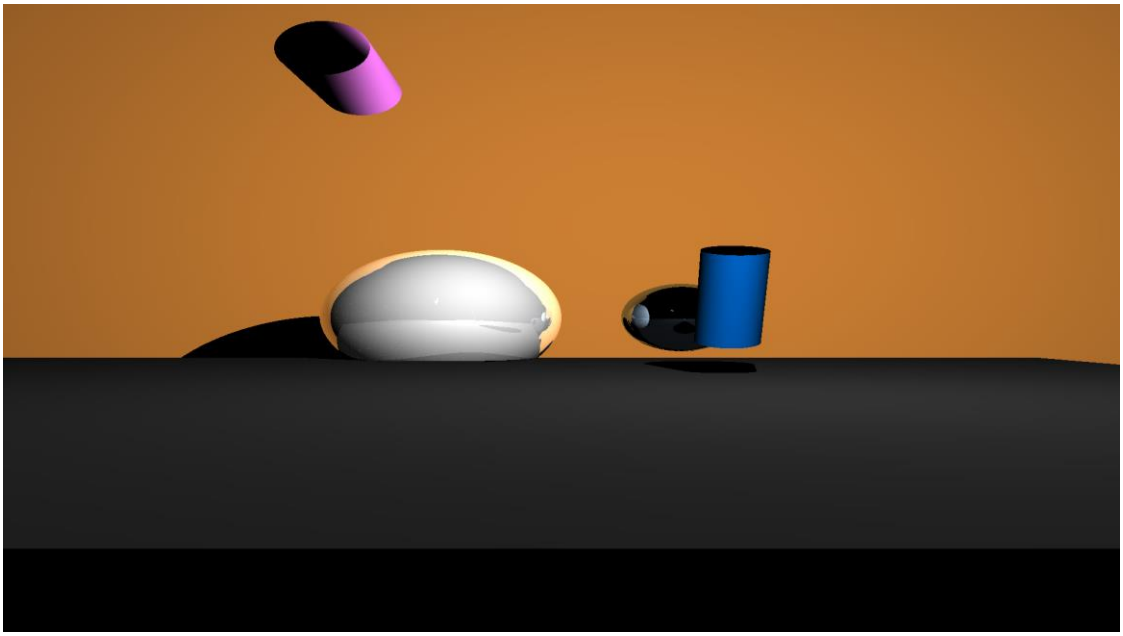
Pour créer les formes géométriques, chaque classe va hériter de l'interface "IShape" qui contient une fonction nommé "hit". Cette fonction est utilisée pour déterminer si un rayon lumineux lancé depuis la caméra traverse ou touche une forme particulière dans la scène. Pour chaque forme créer, la fonction hit sera appeler mais effectuera des calculs diffèrent selon la forme géométrique.

```
namespace Raytracer {
    class IShape {
    public:
        virtual bool hit(const Ray &r, Interval ray_t, double &t_hit, vec3 &hit_normal, Color &hit_color, HitMat &rec) const = 0;

        virtual ~IShape() {}
    };
}
```

Voici les formes géométriques qui figurent sur ce RayTracer:

- La sphère
- Le cylindre
- Le triangle
- Le cône
- Le plan
- Le cube



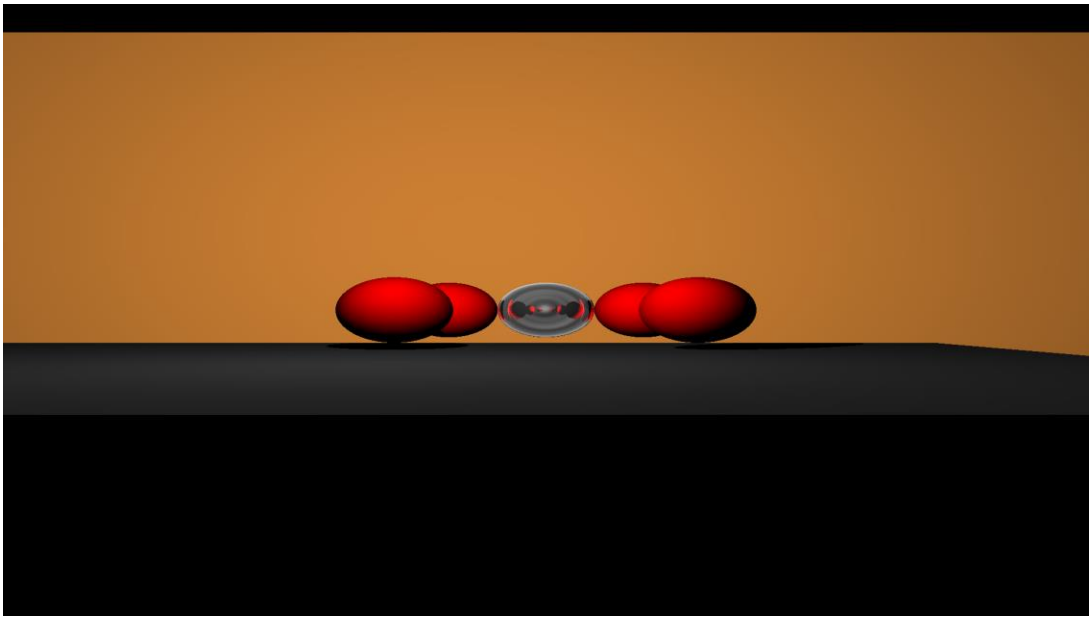
3) Lights

Nous avons créé deux types différents d’éclairage. Il y a la “PointLight”, un éclairage qui se fais à partir d’un point. Il y a aussi la “FrontLight”, un éclairage qui se fais sur toute une ligne et qui éclair en dessous de lui. C’est deux classes héritent de l’interface “ILight”. Elle permet de donner la direction et l’intensité d’une lumière selon un point donné.

```
namespace Raytracer {
    class ILight {
    public:
        virtual vec3 getDirection(const point3 &hit_point) const = 0;

        virtual Color getIntensity(const point3 &hit_point) const = 0;

        virtual ~ILight() {}
    };
}
```



4) Configuration du RayTracer

Pour configurer le RayTracer, nous utilisons un fichier .cfg qui va permettrent de recuperer les informations de tout ce qui sera afficher dans le RayTracer. Ce fichier comporte trois parties distinctes:

- Les objets (les formes géométriques)

Ici seront créer un plan, un triangle et un cylindre avec leurs propres caractéristiques :

```
objects = (
    {
        type = "Plane",
        size = 20.0,
        color = (0.8, 0.5, 0.2),
        position = (0.0, 0.0, -10.0),
        normal = (0.0, 0.0, 1.0)
    },
    {
        type = "Triangle",
        color = (0.8, 0.5, 0.2),
        position = (0.0, 0.0, -10.0),
        v0 = (0.0, 0.0, 0.0),
        v1 = (1.0, 0.0, 0.0),
        v2 = (0.0, 1.0, 0.0)
    },
    {
        type = "Cylinder",
        position = (0.0, -1.0, -2.0),
        axis = (0.0, 1.0, 0.0),
        radius = 0.2,
        height = 1.0,
        color = (1.0, 0.5, 1.0)
        rotation = (90.0, 0.0, 0.0)
    }
);
```

- Les lumières

Ici le fichier de configuration va créer une frontLight avec sa position et sa couleur :

```
light = (
    {
        type = "front",
        position = (0.0, 2.0, 0.0),
        color = (0.5, 0.5, 0.5)
    },
);
```

- La caméra

Le fichier de configuration va créer la résolution, la position et le champ de vision de la caméra :

```
camera = (
    {
        type = "camera",
        resolution = { width = 1920, height = 1080 },
        position = { x = 1.0, y = 0.0, z = 5.0 },
        rotation = { x = 0.0, y = 0.0, z = 0.0 },
        fieldOfView = 50.0
    }
);
```