



RTv1

Mon premier Ray Tracer

Pedago team pedago@42.fr

Résumé: *Ce mini-projet est une première étape dans la réalisation d'un programme de Raytracing, en vue de calculer entièrement des images de synthèse.*

Table des matières

I	Préambule	2
II	Introduction	3
III	Objectifs	4
IV	Consignes générales	5
V	Partie obligatoire	7
VI	Partie Bonus	12
VII	Rendu et peer-évaluation	13

Chapitre I

Préambule

“Il semble qu’en Europe, la recherche soit d’un bon niveau. Mais les faiblesses viennent des applications concrètes, une étape qui est en elle même une source d’innovation. Cela je pense vient d’un manque de sociétés prêtes à risquer pour entreprendre. Ce que nous appelons le développement c’est rarement les grandes entreprises qui le font, c’est plus les petites ou les moyennes entreprises. Alors ce qu’il faut c’est beaucoup de petites entreprises, avec des étudiants doués, des capitaux à risque, plus efficaces entre les mains du secteur privé, et aussi des champions que l’on prenne pour modèle, en disant l’innovation c’est ça. Mais il y a quelque chose de plus subtil : c’est le facteur culturel. En Europe, l’échec c’est très grave. Si en sortant de l’université vous loupez votre coup, cela vous suit toute votre vie. Alors qu’en Amérique, à Silicon Valley, on passe son temps à échouer. Quand on se casse la figure, on se relève et on recommence. Ce qu’il faut pour que l’industrie informatique se développe en Europe et en France, c’est une solide industrie du logiciel. Parce que le logiciel c’est le pétrole des années 80 et 90, de cette révolution informatique. Il faut des centaines de mini entreprises du logiciel, et la France pourrait dominer l’Europe dans le logiciel. Elle a les étudiants les plus brillants, une bonne maîtrise de la technologie, ce que nous devons faire c’est encourager les jeunes à créer des sociétés de logiciel. Nous, nous ne voulons pas mettre la main dessus, le gouvernement ne doit pas non plus tenter de le faire. Elles doivent appartenir à ceux qui prennent des risques.”

Steve Jobs, 1984, interview accordée à Antenne 2

Chapitre II

Introduction

Pour une fois, le sujet est en rapport avec le préambule. Pour réussir, vous devez échouer. C'est pour cela que vous allez commencer par une sorte de proof-of-concept du principe du Raytracing. C'est l'objet du RTv1 : vous familiariser avec le lancer de rayon, les éléments géométriques à manipuler, la description de la scène... . Vous allez donc réussir, ou échouer, ce projet dans le but de le faire par la suite plus gros, plus grand, plus bôôôooo, avec tout plein d'options (ça sera le RT tout court - ne commencez donc pas le RT sans avoir fait le RTv1, c'est pas raisonnable).

Voyez la vidéo avec la démo pour comprendre ce que l'on a au départ, et ce que doit faire votre programme. Les ressources sur le net sont plutôt importantes en matière d'explication du Raytracing. Les approches sont parfois variées, trouvez celle qui vous convient. Le RTv1 reste une version simple, voyez ce qui est demandé pour ne pas vous perdre dans des méandres des très nombreuses fonctionnalités que peut contenir un tel programme et qu'il vous faut de toute façon conserver pour le RT.

Chapitre III

Objectifs

Quand il s'agit de générer des images de synthèse en 3 dimensions, il existe en gros deux approches : la “Rasterisation”, qui est le procédé utilisé par l'écrasante majorité des moteurs de rendus pour son efficacité, et le “Ray Tracing”, approche beaucoup plus lourde en terme de calculs, donc peu adapté au temps réel, mais proposant un réalisme impressionnant.



FIGURE III.1 – Aucune de ces images n'est une photo, elles ont toutes été générées à l'aide d'un ray tracer. Ca pique, non ?

Mais avant d'atteindre un tel niveau de qualité graphique, commençons par le commencement : RTv1, votre premier ray tracer en C, à la norme, humble, mais fonctionnel.

Et puis ça fait toujours des jolies images à montrer à votre chéri(e) pour justifier vos longues heures passées à l'école plutôt que dans ses bras :).

Chapitre IV

Consignes générales

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et de nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- L'exécutable doit s'appeler `rtv1`.
- Vous devez rendre un `Makefile`. Ce `Makefile` doit compiler le projet, et doit contenir les règles habituelles. Il ne doit recompiler et relinker le programme qu'en cas de nécessité.
- Si vous êtes malin et que vous utilisez votre bibliothèque `libft` pour votre `rtv1`, vous devez en copier les sources et le `Makefile` associé dans un dossier nommé `libft` qui devra être à la racine de votre dépôt de rendu. Votre `Makefile` devra compiler la bibliothèque, en appelant son `Makefile`, puis compiler votre projet.
- Les variables globales sont interdites.
- Votre projet doit être en C et à la Norme. La norminette fait foi.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (segmentation fault, bus error, double free, etc...).
- Votre programme ne doit pas avoir de fuites mémoire.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```
- Vous pouvez utiliser la `MinilibX` native `MacOS` qui est sur les dumps, ou alors, vous pouvez utiliser la `MinilibX` à partir de ses sources qui seront à intégrer de la même manière que celles de la `libft`. Dernière possibilité, vous pouvez aussi utiliser d'autres bibliothèques d'affichage (`X11`, `SDL`, etc...). Si la bibliothèque que vous utilisez n'est pas installée sur les machines de cluster, vous devrez fournir les sources de cette bibliothèque dans votre rendu, et elle devra se compiler automatiquement exactement comme la `MinilibX` ou votre `libft`, sans autre action que de compiler votre rendu. Quelle que soit la bibliothèque d'affichage, vous ne devez utiliser que ses fonctions de dessin basiques similaires à celles de la `MinilibX` : ouvrir une

fenêtre, allumer un pixel et gérer les évènements.

- Dans le cadre de votre partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes de la libc :
 - `open`
 - `read`
 - `write`
 - `close`
 - `malloc`
 - `free`
 - `perror`
 - `strerror`
 - `exit`
 - Toutes les fonctions de la lib math (`-lm` et `man 3 math`)
 - Toutes les fonctions de la `MinilibX` ou leurs équivalents dans une autre bibliothèque d'affichage.
- Vous avez l'autorisation d'utiliser d'autres fonctions, voire d'autres bibliothèques, dans le cadre de vos bonus à condition que leur utilisation soit dûment justifiée lors de votre correction. Soyez malins.
- Vous pouvez poser vos questions sur le forum, sur slack, ...

Chapitre V

Partie obligatoire

Votre objectif est donc d'être capable, à l'aide de votre programme, de générer des images de synthèse selon la méthode du Ray-Tracing.

Ces images de synthèse représentent chacune une scène, vue d'une position et d'un angle spécifiques, définie par des objets géométriques simples, et disposant d'un système d'éclairage.

Les éléments à réaliser sont les suivants :

- Implémenter la méthode du lancer de rayon (le raytracing quoi..) pour obtenir une image de synthèse
- Avoir obligatoirement au moins 4 objets géométriques simples comme objets de base (non composés) : plan, sphère, cylindre et cône.
- Possibilité d'appliquer des transformations de translation et de rotations aux objets avant de les afficher. Par exemple une sphère déclarée en $(0, 0, 0)$ doit pouvoir subir une translation en $(42, 42, 42)$.
- Gérer le réaffichage sans re-calculation. En gros, avec la MinilibX, on gère l'exposition correctement. Si une partie de la fenêtre doit être redessinée à l'écran, les calculs ne doivent pas être refaits.
- Positionner et diriger le point de vision ainsi que des objets simples en spécifiant ces informations par un fichier de configuration ou, moins pratique, par les arguments du programme.
- Gérer la lumière : luminosité et ombres. Et ceci n'est PAS une option.

Pour la soutenance, vous devez préparer un jeu de scènes mettant en avant ce qui est fonctionnel, facilitant ainsi le contrôle des éléments à réaliser. Ces scènes doivent pouvoir être générées facilement durant la soutenance. Par exemple :

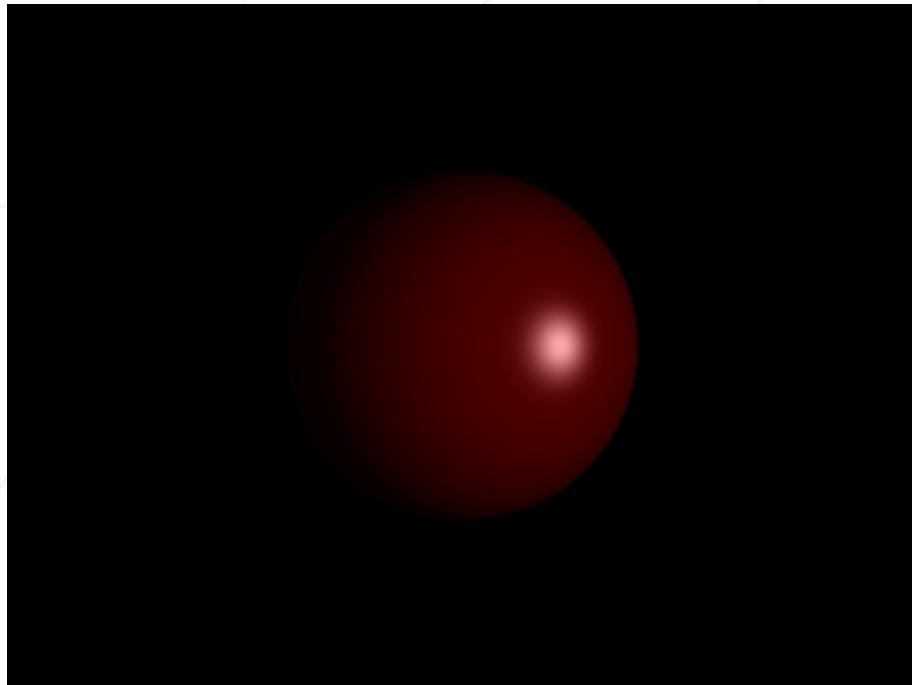


FIGURE V.1 – Une sphère, un spot, de la brillance (option)



FIGURE V.2 – Un cylindre, un spot

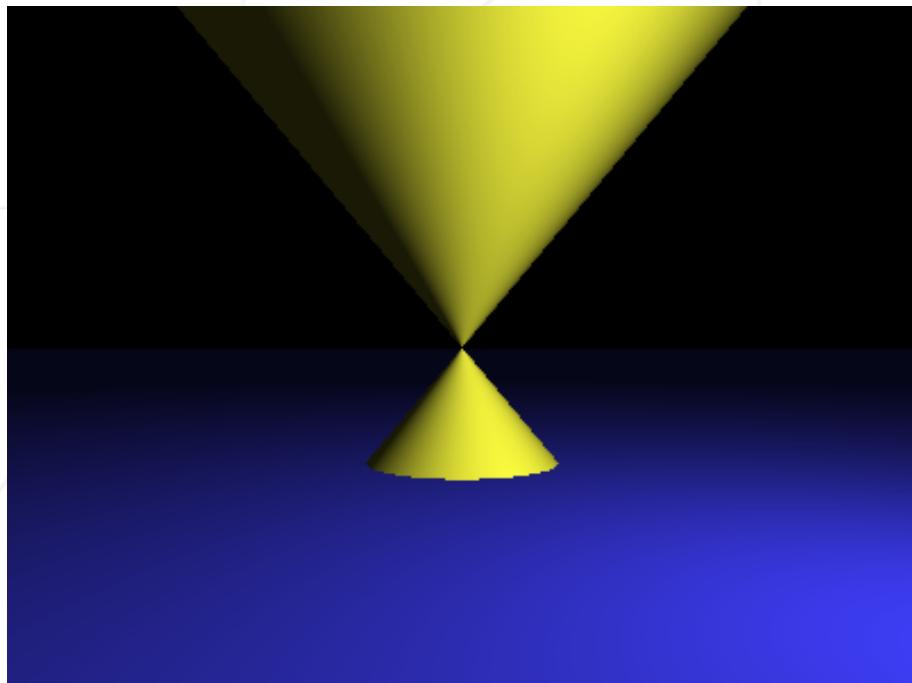


FIGURE V.3 – Un cone, un plan, un spot

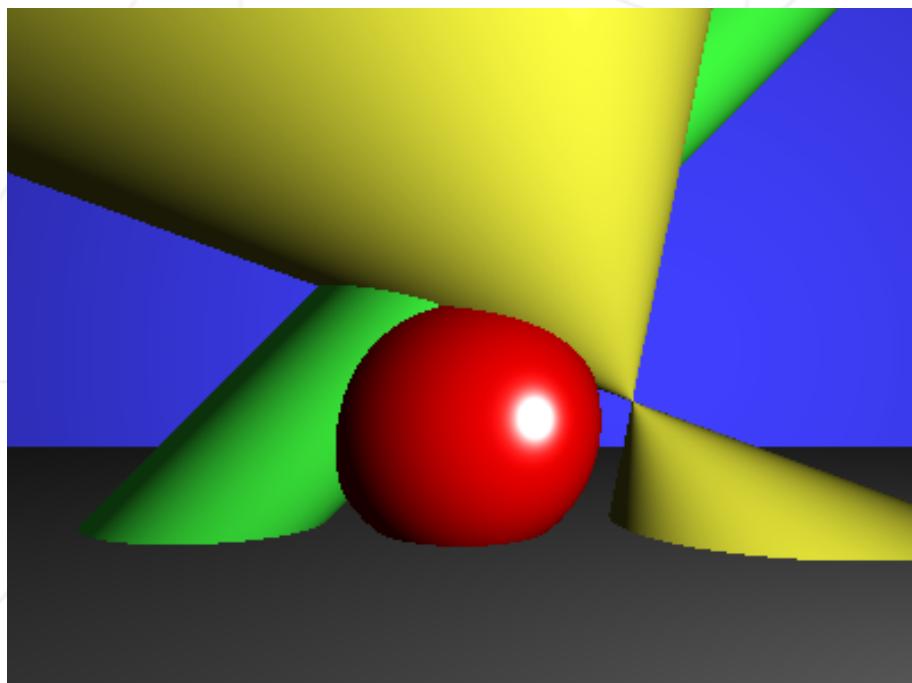


FIGURE V.4 – Un peu de tout, dont 2 plans

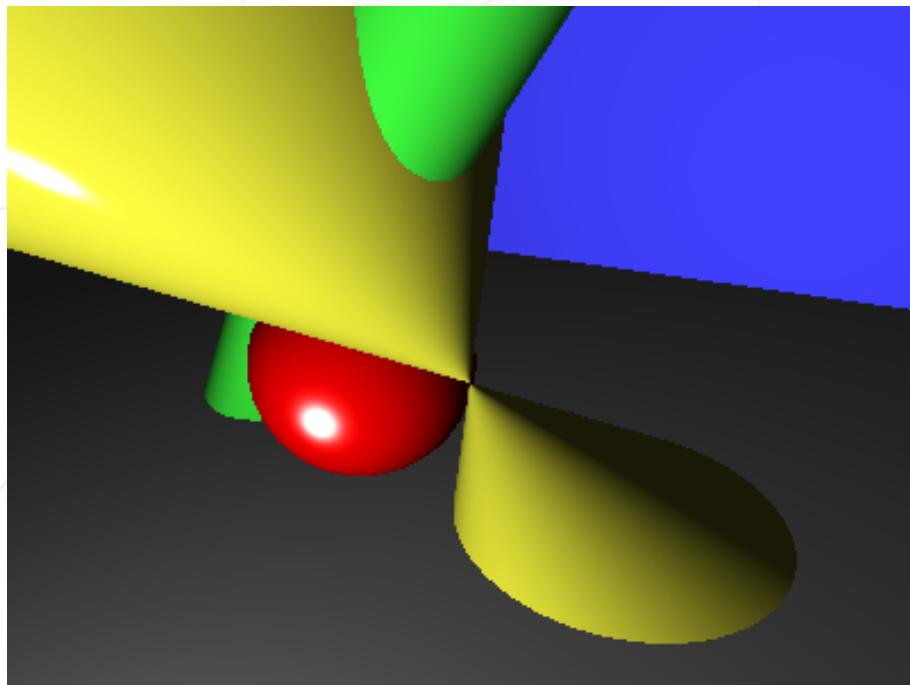


FIGURE V.5 – Même scène, différent point de vue

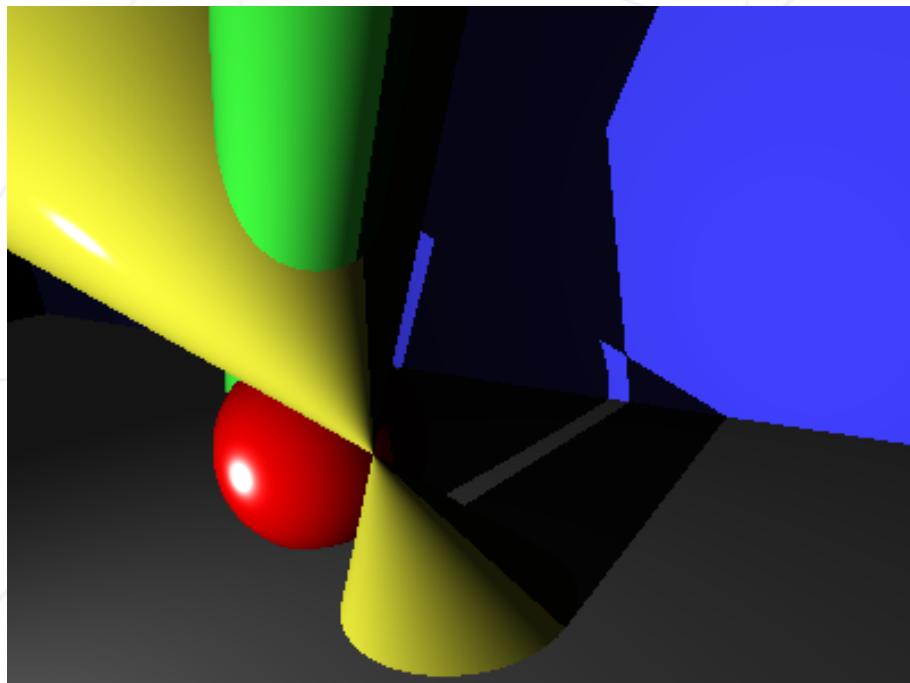


FIGURE V.6 – Cette fois-ci avec des ombres

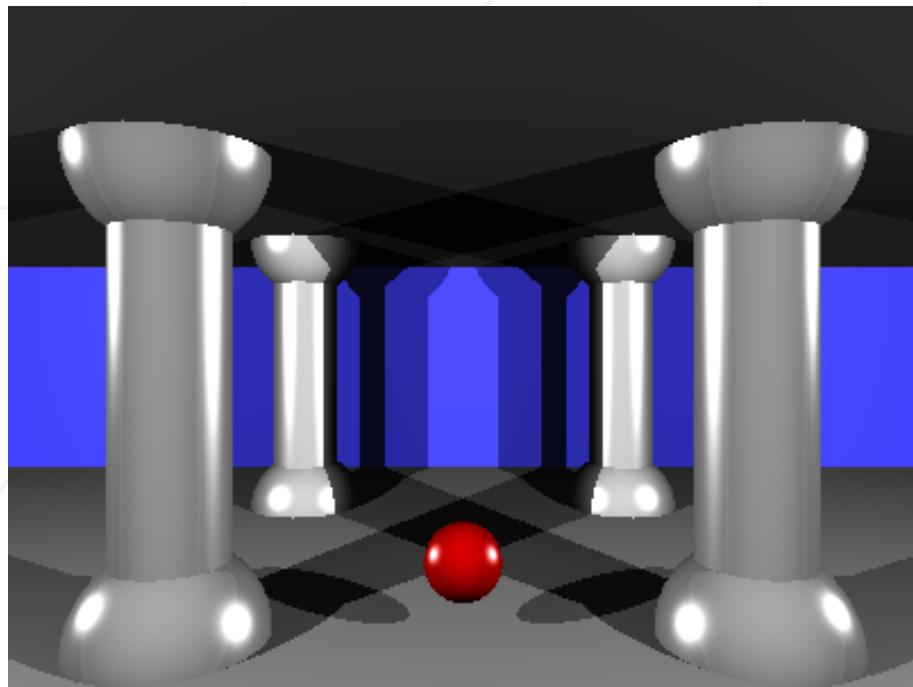


FIGURE V.7 – Et enfin avec plusieurs spots

Chapitre VI

Partie Bonus

Les bonus ne seront évalués que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc. Concrètement, cela signifie que si votre partie obligatoire n'obtient pas TOUS les points à la notation, vos bonus seront intégralement IGNORÉS.

Les bonus possibles :

- Multi-spots
- Brillance

Aucun autre bonus ne sera accepté, cela sera une des options du projet suivant : le RT.

Chapitre VII

Rendu et peer-évaluation

Rendez-votre travail sur votre dépôt GiT comme d'habitude. Seul le travail présent sur votre dépôt sera évalué.

Bon courage à tous et n'oubliez pas votre fichier auteur !