MOOC : C++

Thomas Kowalski

Octobre - Décembre 2018

ensiie

Motivations

- Apprendre un langage
  - Moderne / Orienté objet (comme Java)
  - Rigoureux (comme OCaml)
  - Utilisé dans la vraie vie

Pourquoi le C++ ?

- Témoignage de professionnels
- Curiosité (très peu d’expérience)
- Tentation du compilé

Comment ?

- Trois cours théoriques (Débutant, Intermédiaire, Avancé) sur edX, proposés par Microsoft
- Des lectures de blogs de professionnels utilisant le C++ dans leurs projets
- Deux projets personnels :
  - Rendu 3D par lancer de rayons
  - Adaptation d'un algorithme personnel pour la résolution de problèmes d'ordonnancement *open-shop*

Contenu du cours : débutant

- Types de données (façon C)
- Syntaxe (contrôle)
- Déclaration de classes et de membres
  - Encapsulation et conventions
  - Objets constants

Contenu du cours : intermédiaire

- Pointeurs (non *smart*)
- Implémentation des classes
- Concepts de programmation orientée objet
  - Héritage
  - Classes abstraites
  - Niveau d’accessibilité
  - ...
- Flux
- Fichiers

Contenu du cours : avancé

- Exceptions et gestion
- Templates basiques
- *Iterators* (car plus compliqué qu’en Java)
- Patrons de conception
  - Callbacks
  - Interfaces
  - Singleton
- Namespaces
- Smart pointers

Conclusion sur les cours

- Niveaux et temps nécessaires très différents :
  - Débutant faisable très rapidement car très proche du C
  - Intermédiaire faisable rapidement car proche de l'ILO
  - Avancé demandant beaucoup plus de temps car caractéristique du C++
- Evaluations :
  - Débutant : QCM + Peer review : très bien, le défaut du peer reviewing étant de devoir attendre l'avis des gens
  - Intermédiaire : QCM et "lab" : bien pour s'assurer de la bonne compréhension, mais les lab sont inutiles
  - Avancé : QCM : bien pour faire un résumé de chaque module, mais insuffisant
- Contenu du cours : clair / correct / fourni / intéressant
- MAIS Aucun projet proposé

Lecture de blogs : en bref

- Objectif : se cultiver sur les bonnes pratiques et sur des cas d’utilisation concrets des fonctionnalités de la STL
- Avoir des “mini-projets” à faire fonctionner de mon côté
- Mettre en pratique les acquis des cours

Lecture de blogs : contenu

- *auto-to-stick*
- std::optional
- *Template Meta-Programming*
- Lambda-expressions
- Foncteurs
- Actions sur les *containers*
- Utilisation d'algorithmes de la STL avec des *containers* personnalisés
- Dépréciation de *std::iterator*
- Smart pointers (types, utilisations, exemples)
- *Strong / Named Types*
- ...

Lecture de blogs : conclusion

- Un réel plus par rapport au cours
- Une façon de continuer mon enseignement régulièrement et sans overdose
- Une préparation pour l’avenir

Ray tracing : présentation du projet

- Inspiration d'un projet de l'UE PAP
- Reprise d'un algorithme existant
- Adaptation dans une version plus orientée objet et C++ moderne
- Objectif : comparer les performances

Question : quelles sont les pertes sur les performances lorsque l'on créé un code plus lisible / plus orienté objet et C++ moderne

Pourquoi : C++ apprécié pour ses performances et son côté orienté objet, mais dans quelle mesure peut-on concilier les deux ? Ne devrait-on pas préférer Java (OO) ou C (performances) ?

Ray tracing : déroulement du projet

- Partie majeure du temps dédié au MOOC
- En parallèle avec les MOOC 2 et 3
- Découpage en parties :
  - (Pré-analyse)
  - Fonctionnalités de base
    - *libpng*
    - Modélisation
  - Rendu simple
  - Rendu plus complexe (réflexion)

Ray tracing : algorithme sans réflexion

Pour chaque point de l'image de sortie
    Lancer un rayon depuis l'origine et passant par le point de la matrice
    Trouver la première intersection : Point *P* de la forme *S*
    Pour chaque forme *S'*
        Si *S'* émet de la lumière
            Soit *R'* partant de *P* et allant vers le centre de *S'*
            Si *R'* intersecte une forme (!= *S'*) alors le point est à l'ombre pour *S'*
            Sinon ajouter la couleur émise par S' à la couleur du point
            // On travaille en synthèse additive pour les couleurs

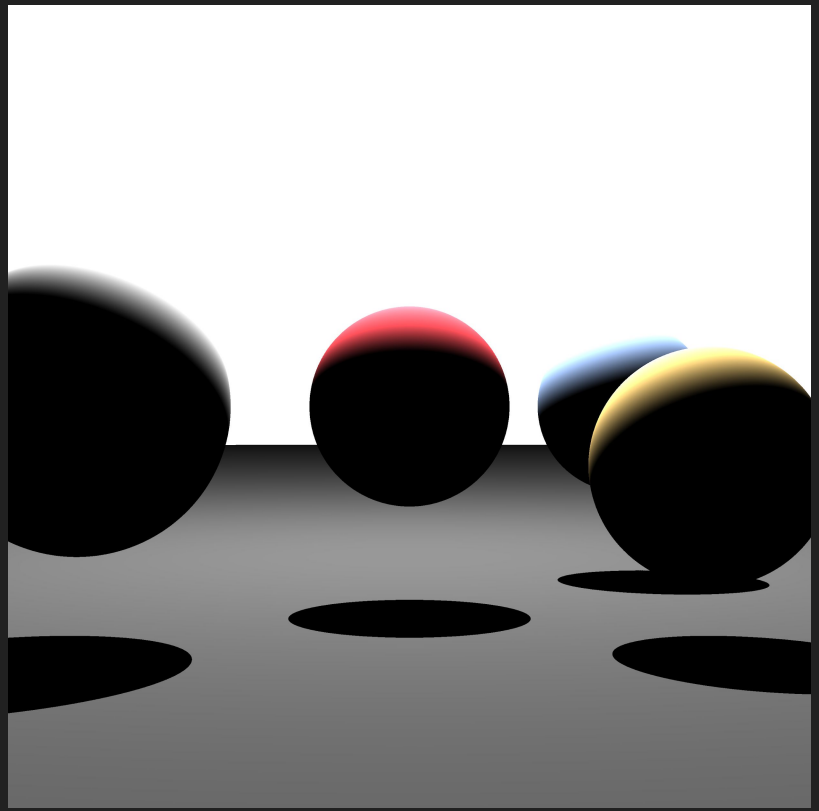
Ray tracing : algorithme (partie réflexion)

Pour chaque point de l'image de sortie

- Lancer un rayon depuis l'origine et passant par le point de la matrice
- Trouver la première intersection : Point *P* de la forme *S*
- **Si *S* est non réfléchissante alors appliquer l'algo précédent et renvoyer**
- Soit *C* la couleur de *S*
- Calculer le rayon réfléchi *R'* partant de *P* (par lois de Descartes)
- Soit *C'* la couleur reçue en *P* et venant de *R'*
- Renvoyer un produit entre *C* et *C'* avec des constantes définies à l'avance

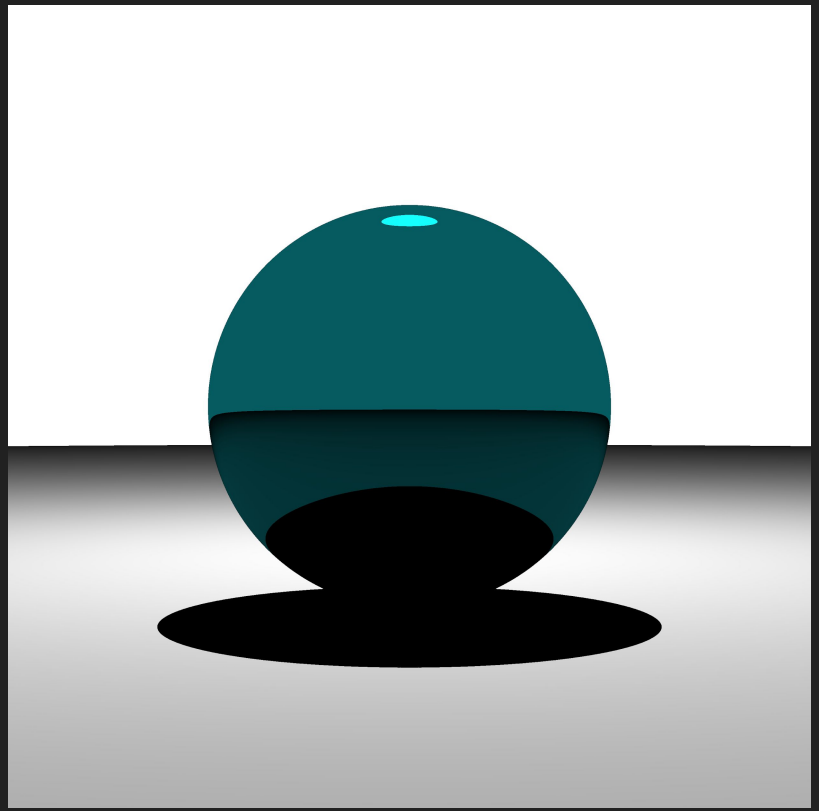
En pratique, *on applique évidemment une limite à la récursion.*

Cas 1



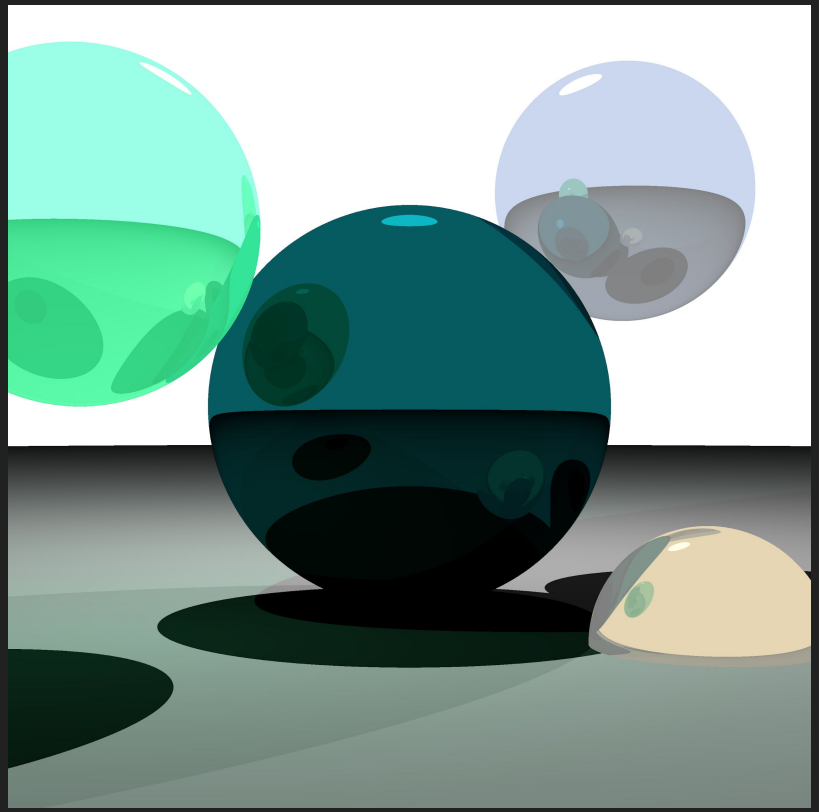
- 5 sphères
- Pas de réflexion

Cas 2



- 2 sphères
- Réflexion

Cas 3



- 5 sphères
- Réflexion

Ray tracing : présentation des résultats

	C / Normal	C / O3	C++ / Normal	C++ / O3
Cas 1	14,777	2,594	17,772 (+20%)	2,650
Cas 2	4,79	1,268	5,587 (+15%)	1,359
Cas 3	6,796	1,483	7,947 (+15%)	1,52

Ray tracing : analyse des résultats

- Résultats temporels attendus :
  - Calculs en C plus rapides (entre 15 et 20% de plus en C++)
  - L’optimisation G++ améliore énormément les résultats
  - MAIS la différence persiste

Alors que choisir dans la “vraie” vie ?

(on en reparle plus tard)

Ray tracing : conclusion

- Satisfaisant sur les résultats
  - Beau rendu
  - Projet intéressant sur l'idée
- Insatisfaisant sur le temps passé
  - Beaucoup de temps passé à déboguer à tâtons
  - Temps que je ne pouvais pas passer à véritablement me cultiver sur le C++
  - Débogage sur l'aspect mathématique et non sur l'aspect programmation
  - Pas de réel travail en C++ (j'aurais débogué de la même façon en un autre langage)

Ordonnancement : projet personnel

Objectif : Ordonnancer les passages d'oraux pour *n* élèves avec *p* jurys (chaque élève passe une fois avec chaque jury)

Réécriture d'un algorithme mis au point en CPGE (TIPE)

Conclusion :

- Les solutions sont comparables en matière de *makespan* ;
- Le temps d'exécution est bien supérieur en C++, légèrement inférieur avec -O3

C++ et ses utilisations

Alors qu'utiliser dans la "vraie" vie ?

- Développement applicatif : C++
- Développement intégré : C ou C++ (fonction du cas)
- Développement système : C
- Programmation scientifique : C ou C++ (fonction du cas et de l'utilisateur final)

Ne pas oublier que la tendance générale est au gain de temps développeur et non gain de temps de calcul et que les deux solutions restent performantes.

MOOC : Conclusion

Ses avantages :

- Langage riche et moderne, toujours en évolution aujourd'hui
- Propose une syntaxe agréable à écrire mais aussi (et surtout) à lire
- Tout en gardant la performance et la compilation

Ses défauts :

- Un entre deux entre la performance absolue plus bas-niveau (langage C) et une lisibilité et praticité optimale (Python, Java…)
- Preuve formelle *a priori* plus difficile à mettre en oeuvre car plus d'abstraction (bien que pas explorée dans ce MOOC)