Réponses aux questions

J. Odersky C. Vázquez

22 mars 2011

Question P1.1

Un vecteur est représenté par la classe 'Vector3D'. Cette classe comprend trois champs privés du type double : x, y et z correspondant aux composantes du vecteur. Ces champs peuvent être accédés respectivement par les méthodes publiques getX(), getY() et getZ(). Les méthodes "opérateurs" sur les vecteurs (par exemple l'addition, la norme, etc...) sont toutes publiques.

Ces méthodes s'appuient entre-elles, par exemple la méthode 'opposée' retourne le vecteur multiplié par moins un et la méthode 'soustraction' additionne l'opposé. Ces appels consécutifs diminuent la performance d'une façon minimale mais évitent un duplicage de code considérable.

Un vecteur est complètement invariable. C'est-à-dire qu'une fois un vecteur initialisé, on ne peut plus changer ses composantes (pas de méthodes 'set'). De même, tous les opérations internes du sens mathématique (qui renvoyent un vecteur), renvoyent une nouvelle instance d'un vecteur. En aucun cas l'instance d'un vecteur n'est modifiée ! Ceci facilite énormément le raisonement sur toute variable de type vecteur. De plus, l'invariance d'un vecteur paraît naturelle, comme celle d'un nombre réel.

Quelques vecteurs remarquables sont définis comme variables statiques constantes. Parmi ceux-ci notamment le vecteur nulle (Null) et les vecteurs unitaires i, j, k selon respectivement les axes x, y et z.

Question P3.1

Nous n'avons pas rajouté un constructeur de copie. Comme la classe 'Vector3D' est invariable et ne contient pas de pointeurs ou références sur d'autres objets mutables, elle n'a pas d'état et donc l'utilisation du constructeur de copie par défaut suffit.

Question P3.2

Lorsque l'on décide d'implémenter le constructeur par défaut (qui crée un vecteur nul) et le constructeur par coordonnés cartésiennes dans une seule et même méthode, il se peut que lorsque l'on crée une nouvelle instance de Vector3D sans l'initialiser (en laissant donc le soin au constructeur par défaut de le faire), on se retrouve avec un vecteur nul à un endroit où il vaudrait mieux ne pas en avoir. Par conséquent, en séparant ces deux méthodes, le programmeur est amené à penser dès l'instanciation à quelle fin il crée son objet.

Question P3.3

a

En ajoutant un constructeur par coordonnées sphériques, les attributs de la classe ne devraient pas forcémant être changées. Il serait toute à fait envisageable, de garder les coordonnées carthésiennes comme attributs et de convertir les coordonnées sphériques avec le constructeur.

b

La surcharge serait une difficulté majeur pour créer un constructeur par coordonnées sphériques. Etant donné qu'un tel constructeur prendrait comme paramètres deux angles et une longueur, représentés par trois doubles,

il serait en conflit avec le constructeur de coordonnées carthésiennes. Il serait alors impossible d'avoir les deux constructeurs dans une classe.

Néanmoins, une solution alternative serait d'implémenter une méthode statique ("factory method") qui prendrait comme paramètres des coordonnées sphériques et qui renverait un vecteur ayant des coordonnées carthésiennes équivalentes (par exemple `Vector3D Vector3D::fromSpherical(double phi, double theta, double r)`).

Question P3.4

La méthode 'affiche()' d'un vecteur a été implémenté sous forme de l'opérateur '<<' de 'std : :ostream'. La méthode 'compare' est équivalent à l'opérateur '==='.

Question P5.1

Réponse incohérente ! La question conçernant l'énergie me perturbe...

Les membres d'une particule représentant le facteur gammaγ et l'énergie$E$ peuvent être implémentés soit sous forme d'attributs soit sous forme de méthodes. Il y a des avantages et inconvénients pour chaque forme.

L'avantage d'un attribut est que son accès est très rapide et ne prend (presque) pas de temps de calcul. Par contre, si la valeur d'un attribut est relié logiquement à la valeur d'un autre attribut et que ce dernier est modifié, il faudra manuellement changer le premier. Par exemple, le facteur gamma étant défini par :

$$\gamma = \frac{1}{\sqrt{1 - \left(\frac{v}{c}\right)^2}}$$

si il est défini comme variable, il faudrait le mettre à jour à chaque fois que la vitesse change.

Contrairement à un attribut, une méthode est évaluée à chaque fois qu'on l'appelle. Ceci a l'avantage que si le résultat d'une méthode dépend d'une variable, la variable pourra être modifiée sans considération de la méthode. Ainsi, si le facteur gamma est une méthode, une mise à jour de la vitesse pourra être effectuée directement sans explicitement changer γ.

Dans le cas de notre projet, nous avons décidés d'implémenter l'énergie sous forme d'attribut et le facteur gamma sous forme de méthode. Ceci pour plusieurs raisons :

1. Au cours de la simulation, la vitesse sera changée très fréquemment, beaucoup plus que les appels à gamma. Une implémentation de gamma sous forme de méthode améliore donc la performance.
2. Le premier argument pourrait s'appliquer également à l'énergie, or il faut remarquer que l'énergie est une grandeur spécifiée durant la création d'une particule et que en fait la vitesse dépend de l'énergie [1]. On pourrait alors penser à définir la vitesse comme méthode mais d'après l'argument 1, ce serait complètement absurde au niveau de la performance.

Question P6.1

Pour représenter et oraniser les éléments, nous avons choisi dans ce projet de coder les éléments (magnétiques et électriques, droits et courbes) sous formes de classes. Concrètement, la classe `Element` est la classe grand-mère, les quatre classes filles sont les classes `Droit` et `Courbe`, `Magnétique` et `Electrique`. La classe `Magnétique` hérite à la fois de `Droit` et de `Courbe`, tandis que `Electrique` n'hérite "que" de `Droit`.

Question P6.2

Les champs magnétiques et électriques sont représentés à l'aide :

- d'un `Vector3D` indiquant la direction du champ, pour l'instant ne dépendant pas de la position de la particule dans l'`Element`,

1. Complément mathématique : "Aux énergies atteintes dans un accélérateur, les particules sont tellement proches de la vitesse de la lumière qu'il faudrait garder au moins 7 décimales pour que la vitesse permette de connaître l'énergie de façon assez précise. On caractérise donc plutôt une particule en termes d'énergie totale que de vitesse."

- d'un `double` indiquant la valeur constante de l'intensité de chaque champ.

En effet, en physique on représente les lignes de champ par des lignes parrallèles, continues, et nous disons que le champ a une certaine intensité, qui reste la même tout le long des lignes. Or en informatique, cela ne nous intéresse pas de représenter ces lignes. Puisque les seules interactions de ces champs avec les particules invoquent des vecteurs représentant l'intensité et la direction du champ à l'endroit où se trouve la particule par rapport à l'`Element`, on n'a besoin que d'un `Vector3D`, et d'un `double` fournissant son intensité. Nous avons cependant choisi d'opérer un distinction entre norme et direction, en vue d'une éventuelle amélioration du code, où nous n'aurions besoin de l'intensité qu'une seule fois pour toutes, et nous aurions le loisir de modifier la direction comme bon nous semble.

Question P6.3

Dans la classe `Courbe`, nous avons implémenté le centre de courbure sous forme d'un `Vector3D`, puisque tous les points sont naturellement représentés par des vecteurs dans $\mathbb{R}^{\not{4}}$.

Question P6.4

Afin qu'une particule soit dans un et un seul à la fois, nous avons décidé de rajouter un pointeur à chaque particule, qui soit responsable de lui indiquer dans quel `Element` elle doit être. Cela pour garantir un effet de traçababilité.