Réponses

Vock Raphaël, Vezin Lomàn

March 14, 2019

1 Vector3D

1.1 Première modélisation

Nous avons fait le choix d'une représentation cartésienne des vecteurs de $\mathbf{R}^3$, ces derniers sont donc modélisés par une classe `Vector3D` dont les trois attributs x, y, z sont des `double`. L'accès aux attributs est privé, et se fait par le bisais d'un getter qui les retourne dans un `array` de dimension 3. Nous utilisons une méthode `is_zero` qui renvoie `true` si la norme carrée du vecteur en instance est inférieure à une marge `DEFAULT_EPSILON` donnée. Les attributs peuvent toutefois être initialisés par le biais s'un setter. La classe dispose de plus d'un constructeur faisant également office de constructeur par défaut, initialisant en l'absence d'argument chacune des coordonnées à 0.

1.2 Constructeur de copie

Nous avons décidé de ne pas ajouter de constructeur de copie, la classe `Vector3D` ne contient pas, entre autres, de pointeur dans ses attributs, ses instances ne nécessitent donc pas de copie profonde. la copie de surface offerte par le constructeur de copie par défaut est amplement suffisante.

1.3 Coordonnées sphériques

Nous n'avons pas implémenté de constructeur en coordonnées sphériques. L'ajout de ce dernier poserait problème au niveau du constructeur dont le prototype demanderait tout comme le constructeur cartésien 3 `double` en entrée ce qui ne constitue pas une surcharge. Bien que l'on ai la possibilité de contourner ce problème les coordonnées sphériques restent, dans le contexte d'un accelerateur de particules, moins pertinentes que les coordonnées cylindriques.

1.4 Surcharge des opérateurs

Nous avons défini par surcharge les opérateurs d'auto-affectation nécessaires aux calculs vectoriels `+=`, `-=`, `*=`. A partir de ces derniers nous avons ensuite défini les opérateurs `+`, `-`, `*`. Nous avons de plus introduit une seconde surcharge de l'opérateur `*` pour la multiplication d'un vecteur par un scalaire.

Nous avons également défini une surcharge pour les opérateurs binaires produit scalaire `|` et produit vectoriel `^`, quant au produit mixte, il s'agit d'une fonction externe prenant trois vecteurs en paramètre et retournant un `double`.

Finalement nous avons surchargé les opérateurs `==` et `!=` permettant de tester l'égalité de deux vecteurs, dans le cas où la norme de la différence est inférieure à la marge mentionnée plus haut.