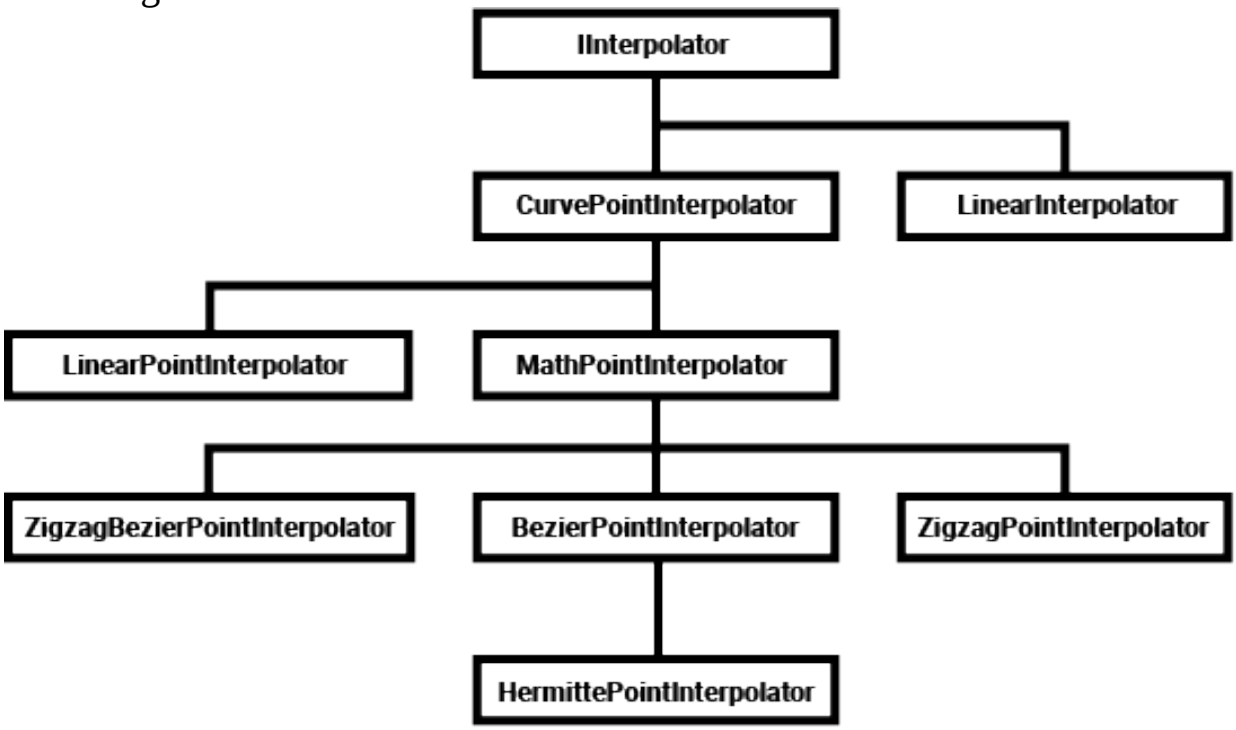
Compte rendu TP3
Dorian Delannoy

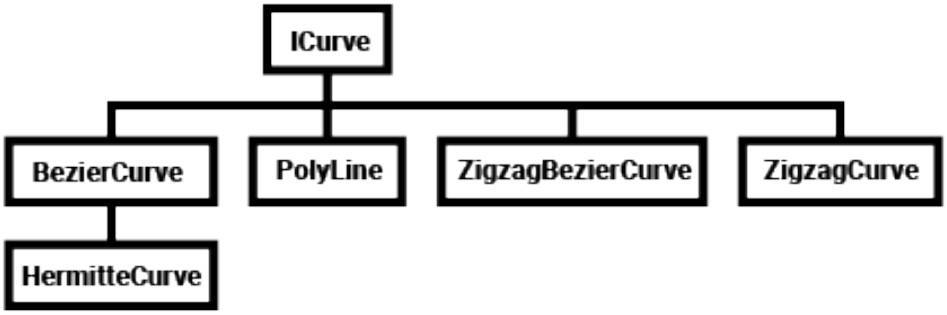
Objectif du TP :

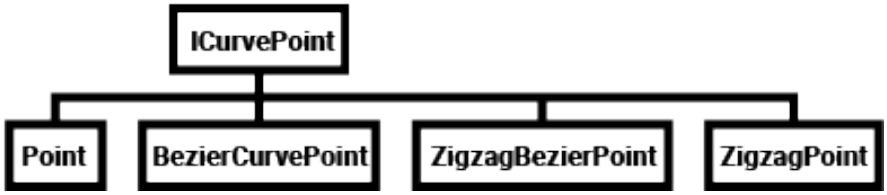
Le but du TP était de créer une librairie permettant de calculer un point à un certain pourcentage d'une courbe quel que soit l'algorithme de calcul de la courbe.

Il nous faudra donc une class abstraite représentant une courbe et ayant au moins une fonction du type « obtenir le point a t % de la courbe »
Il nous faudra aussi implémenter des class dérivant de la courbe abstraite représentant : une poly ligne, une courbe de Bézier de degrés 3 et une courbe d'Hermite.
Pour démontrer la flexibilité de mon système j'ai ajouté des courbes « Zigzag » qui ajoutent des ondulations.

Organisation des classes :







Classes de base :

L'algorithme d'échantillonnage de la courbe se trouve dans `ICurve::pointAtPercent(double)` et il fait appelle a l'interpolateur obtenu en appelant la fonction abstraite `ICurve ::interpolator()` qui retourne un objet de type `CurvePointInterpolator.`

Le `CurvePointInterpolator` sert de class de base pour tous les interpolateurs ayant pour rôle d'interpoler un point le long d'une courbe. Il définit aussi une fonction `distanceBetween(ICurvePoint*, ICurvePoint*)` Utilisée par ICurve dans l'algorithme échantillonnage.

Chaque courbe ayant besoin de données différentes pour être calculée (position, tangente, autres…) `CurvePointInterpolator::interpolate(ICurvePoint*, ICurvePoint*, double)` attend un objet de type `ICurvePoint` qui est la class de base pour tous les points faisant partie d'une courbe. Cette classe n'a pas de membre.

Ainsi pour définir une nouvelle courbe il nous faut :

– (optionnel) Définir une class dérivant de `ICurvePoint` recevant les données nécessaires dans son constructeur.

– Définir une class dérivant de `CurvePointInterpolator` et sa fonction `interpolate` qui fait le calcul

– Définir une class dérivant de `ICurve` et retourner l'interpolateur voulut dans sa fonction `interpolator()`. Si on ne dérive pas de `MathPointInterpolator`, définir la fonction `distanceBetween`.

Le reste est déjà implémenté dans `ICurve`, Il est bien sur toujours possible de surcharger les méthodes de `ICurve` si nécessaire.

L'avantage de faire de cette manière est que chaque élément peut être réutilisé car il n'est pas lié aux autres. En plus, créer chaque élément est plus facile, par exemple en écrivant un interpolateur on a juste à se soucier de l'interpolation entre deux points. Le reste est pris en charge par les autres éléments du système.

Implementations :

`BezierPointInterpolator`, `LinearPointInterpolator`, `HermittePointInterpolator`, `ZigzagPointInterpolator`, `ZigzagBezierPointInterpolator` sont les interpolateurs non abstraits utilisés par les différentes `ICurves`, leur nom défini leur fonction. `LinearPointInterpolator` est utilisé par le poly ligne et les interpolateurs ayant « Zigzag » dans leur nom utilisent une fonction cosinus pour faire des traits ondulés.

Autres détails :

`MathPointInterpolator` est une class abstraite qui s'occupe d'implémenter la fonction `distanceBetween` en échantillonnant la courbe, elle sert de class de base a toutes les courbes mathématiquement complexes.
`HermittePointInterpolator` dérive de `BezierPointInterpolator`, car les deux class nécessitent un point de type `BezierCurvePoint`. Il s'agit plus de fainéantise que d'optimisation, un meilleur choix aurait été de créer une class « BezierBaseInterpolator » servant de class de base pour tous les interpolateurs utilisant des `BezierCurvePoint`.

Pour réduire le temps de calcul j'ai ajouté un cache pour les distances entre deux points, cela évite a la courbe de recalculer la distance entre chaque point et la taille totale de courbe à chaque fois que `getPointAtPercent` est appelé. Cela a considérablement amélioré les performances.
Si besoin il serait possible d'améliorer encore plus les performances en ajoutant le même genre de cache dans `MathPointInterpolator`, nous donnant une courbe pré-échantillonnée et ont ferait une interpolation entre les échantillons si nécessaire. Ainsi la courbe ne serait calculée qu'une fois, au coup d'une perte de précision et d'un peu de mémoire.

`CurveRenderer` sert à créer le fichier post-script et afficher les objets de type `ICurve`.

L'algorithme d'échantillonnage de ICurve :

On commence par déterminer a quelle distance sur la courbe est le point au pourcentage voulut. Ensuite on parcourt les points de la courbe jusqu'à atteindre le premier point à une distance plus grande que celle voulut. On sait donc que le point voulut est entre le point précédent et le point actuel. On détermine ensuite à quel pourcentage entre les deux points est le pourcentage voulut et c'est cette valeur qu'on passe a l'interpolateur.

Problème :

Pour que `ICurvePoint` soit considérée polymorphique par le compilateur j'ai dû mettre une méthode virtuelle qui ne sert a rien. Sur internet on me conseillait de mettre un destructeur virtuel. Malheureusement si je fais cela faire un `dynamic_cast` vers une class fille retourne toujours `nullptr`, je ne sais pas vraiment pourquoi.

Pour tester l'algorithme en situation réelle j'ai vectorisé puis exporté une photo de vous avec Adobe Illustrator.