TP4 C++ - Conception

B3129 : Pierre-Louis LEFEBVRE et Nicolas SIX

Vendredi 5 février 2016

Table des matières

1 Diagramme UML

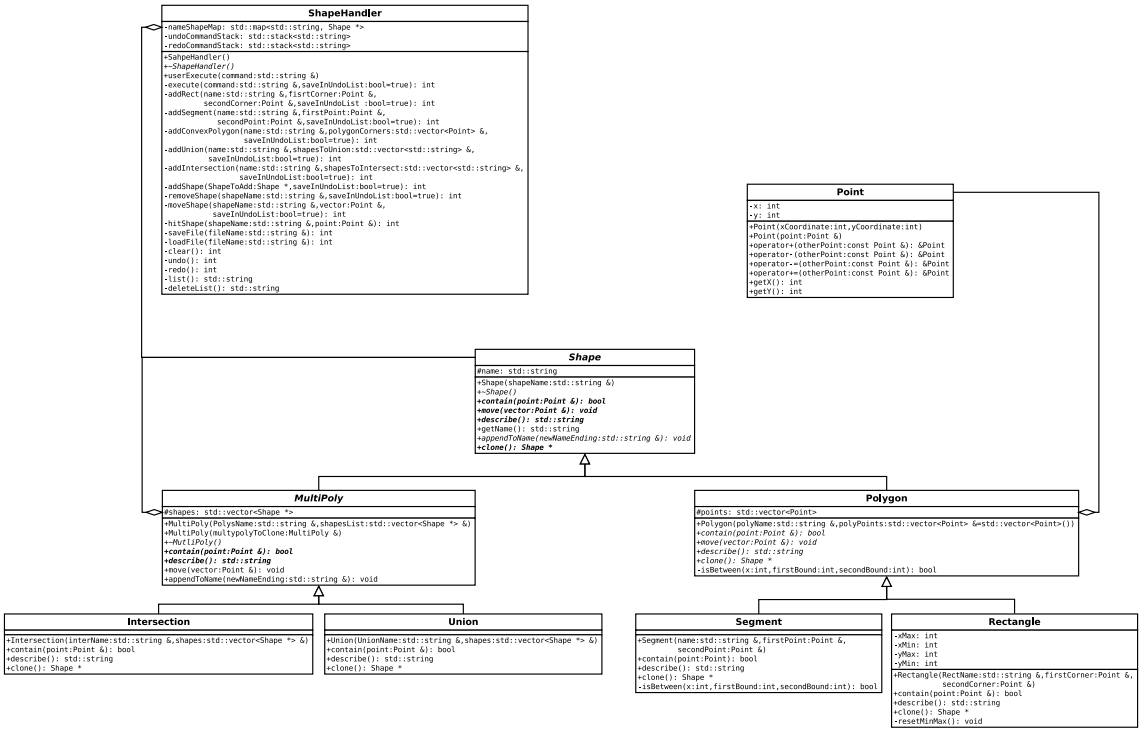

FIGURE 1 – Diagramme de classe UML

2 Détection des dappartenance d'un point à une forme

2.1 Appliquée à un ensemble

Chaque ensemble, intersections et réunions, contient chacun des éléments qui le compose. Pour effectuer la détection un ensemble se contente donc de vérifier si le point donné appartient ou non aux éléments qui le compose. Cette opération peut sembler coûteuse mais permet de ne pas avoir les problèmes dimprécision due aux arrondis qui seraient apparus lors du calcul d'une forme qui résumerait cet ensemble. De plus il est facile de ne pas étudier chaque sous-forme dans les nombreux cas que représentent les réponses négatives.

Il apparaît donc qu'une détection efficace des hit sur les formes élémentaire est primordiale.

coût : $O(n)$ avec n le nombre de figure contenu dans l'ensemble.

2.2 Appliquée à un segment

La détection des hit sur un segment se base sur léquation de la droite correspondant au segment avec tout d'abord vérification de lappartenance au rectangle dont la diagonale est le segment considéré. Cette méthode est relativement complexe, car elle nécessite en plus de quelques additions une multiplication et surtout une division, toute fois ces calculs restent rapides et ne posent pas de problèmes de performance.

coût : $O(1)$

2.3 Appliquée à un rectangle

La détection des hit sur un rectangle et ici triviale vu qu'elle se compose de quelques comparaisons par rapport aux valeurs max et min de la figure suivant les deux axes une opération très rapide.

coût : $O(1)$

2.4 Appliquée à un Polygone convexe

La détection des hit sur un polygone convexe détermine si un point appartient à la figure en cherchant un segment se situant au-dessus du point et un en dessous. La grande majorité des cas sont traités en quelques conditions. Les cas les plus compliqués quant à eux utilisent le même système que pour les segments en utilisant l'équation du côté considéré. Cette méthode traite donc les côtés un par un juste à en trouver un au-dessus et un en dessous ou bien jusquà trouver un point appartenant à un côté. Bien que relativement lourds sur les polygones ayant de nombreux cotés, cette technique reste capable de traiter tous les côtés sauf dans le pire des cas deux d'entre eux juste en faisant quelques comparaisons de valeur ce qui lui permet de rester très efficace.

coût : $O(n)$ avec n le nombre de points du polygone.

3 Détection de la convexité d'un polygone

Afin de vérifier qu'un polygone est bien convexe, nous vérifions que le sinus de l'angle entre deux points formant un côté du polygone et un autre point du polygone reste toujours de même signe opération que nous répétons pour toutes les combinaisons possibles dans le polygone. Cette opération est assez lourde en calcul mais nous assure que le polygone et bien convexe et nous permet notamment de détecter les polygones croisés et repliées sur eux-mêmes.

coût : $O(n^2)$ avec n le nombre de points du polygone.

4 Structures de données

4.1 L'ensemble des formes

Les formes géométriques gérées par l'éditeur sont toutes contenues dans un dictionnaire attribut de la classe ShapeHandler, avec comme clé le nom donné par l'utilisateur à la forme. Comme on vérifie à chaque création de forme que le nom donné n'existe pas déjà, la structure est appropriée car la clé est alors unique. De plus la recherche d'une forme dans le dictionnaire, notamment pour déterminer l'appartenance d'un point, pour faire une suppression, une union, une intersection, ou pour bouger la forme, la performance est constamment en logarithme à base 2 du nombre de formes dans le modèle. Cette performance est meilleure que la plupart des autres structures pour la recherche, sauf pour la table de hachage, et encore pour un nombre limité de formes.

4.2 Les gestionnaires des UNDO/REDO

A chaque commande pouvant être annulée (refaite), la commande inverse, ou liste de commandes inverses, est stockée dans une pile sous forme de chaînes de caractère de UNDO (REDO). Cette pile permet à l'appel de UNDO (REDO) de récupérer avec une performance en $O(1)$ la commande à faire pour annuler (refaire) la dernière commande effectuée non annulée (refaite). Par ailleurs, à chaque exécution d'une commande modifiant le modèle, la pile des REDO est vidée car elle n'a plus de sens.