Classes et généricité: template et polymorphisme en C++

durée: 4h

Ce TP propose une application des classes, des templates et du polymorphisme au travers du design de classes permettant de gérer des courbes de Bézier.

Contents

1- Bézier unidimensionnelle

On rappelle qu'une courbe de Bézier cubique peut s'exprimer sous la forme suivante:

$$p(s) = (1-s)^3 \, P_0 + 3(1-s)^2 s \, P_1 + 3(1-s)s^2 \, P_2 + s^3 \, P_3 \, ,$$

avec s étant un paramètre scalaire variant dans l'intervalle $[0,1]$, et (P_0, P_1, P_2, P_3) les points de contrôle de la courbe de Bézier définissant ainsi un polynôme appelé ***polygone de contrôle***.

Nous souhaitons définir une classe `bezier` en C++ qui permet de manipuler ce type de courbe.

- ▷ Prenez connaissance du programme 1. La fonction `main()` est prévue pour faire appel à une classe `bezier` qui n'existe pas encore, ce programme ne compile donc pas.
- ▷ Créez les fichiers `bezier.hpp` et `bezier.cpp`. Vous placerez l'en-tête de votre classe de Bézier dans le fichier .hpp, et l'implémentation dans le fichier .cpp.
- ▷ Ecrivez l'en-tête de la classe de Bézier dans le fichier `bezier.hpp`. Cette classe contiendra en tant que donnée privée un tableau de taille statique de 4 floats.
- ▷ D'après le code de la fonction, `main()` définissez les méthodes et fonctions nécessaires pour votre classe. Implémentez ces fonctions dans le fichier `bezier.cpp`.

Remarque:

- **Ne codez pas l'ensemble des fonctionnalités d'un coup**. Codez chaque fonctionnalité les unes après les autres (constructeur d'abord, puis méthode coeff, etc) en testant bien que votre programme compile à chaque ajout et qu'il donne le résultat attendu. Pour cela, **commentez les parties** de la fonction `main()` que vous n'utilisez pas afin de pouvoir avancer par étapes.
- Pour initialiser un tableau de taille 4 (`P[4]`) à l'aide d'une liste d'initialisation dans un constructeur à partir de 4 paramètres (P0,P1,P2,P3), il est possible d'utiliser en C++11 la syntaxe particulière suivante[1]:

```
Classe::Classe(Type P0,Type P1,Type P2,Type P3)
    :P{P0,P1,P2,P3}
{}
```

  Sinon il reste toujours possible d'initialiser P dans le corps de la fonction `P[0]=P0;` etc.

La fonction `export_matlab()` contenue dans le fichier du même nom est une fonction permettant d'exporter votre courbe polygone de contrôle ainsi qu'une version échantillonnée de votre courbe dans un fichier lisible par Matlab ou Octave. Le script `viewer.m` vient lire le fichier `data.m` exporté par cette fonction et affiche le résultat graphiquement.

- ▷ Vérifiez que l'exécution de ce code fonctionne correctement. On procèdera donc à la démarche suivante:

[1] L'initialisation d'un autre type de variable aurait nécessité l'utilisation de parenthèses

1. Ajout en fin de fonction `main()` de la ligne `export_matlab(``data.m'',b1);`
2. Vérification que l'exécution de ce code créé bien un fichier `data.m` dans le répertoire courant (au même niveau que `viewer.m`).
   Remarque, si vous utilisez le `CMakeLists.txt` ou `QtCreator`, votre fichier `data.m` sera exporté par défaut dans le répertoire de compilation. Vous pouvez paramétrer `QtCreator` pour qu'il exécute le programme dans le répertoire des fichiers sources, ou bien copier les fichiers de données ou le script `viewer.m` dans les dossiers appropriés.
3. Lancez Matlab (ou Octave) dans ce même répertoire et appelez `viewer`. Vous devriez visualiser votre polygone de contrôle ainsi que la courbe de Bézier associée comme illustré sur la figure 1.

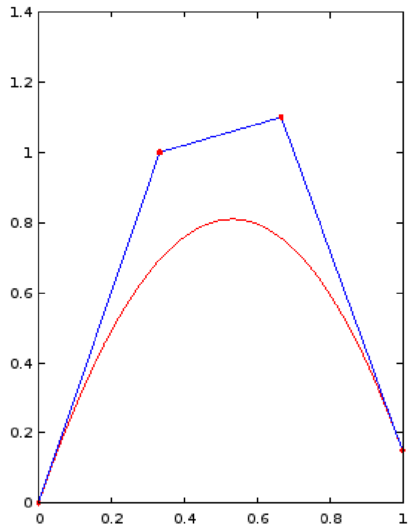

Figure 1: Exemple d'affichage de courbe de Bézier obtenur par le script d'export sous format Matlab.

2- Classe template

Dans l'exercice précédent, nous avions supposé que P_0, P_1, P_2 et P_3 étaient des scalaires (float). Or la formule d'une courbe de Bézier est générique et permet théoriquement de définir une courbe de Bézier dans le plan 2D si les points de contrôles P_i sont définis comme des vecteurs du plan (x,y), ou bien encore en 3D si ils sont définis comme des positions 3D (x, y, z). Il est ainsi possible d'étendre la notion de courbe de Bézier à toute dimension.

Nous proposons d'étendre la classe de courbe de Bézier à l'application en toute dimension et à tout type de variable (float, double, long double, etc) afin d'éviter de devoir en redéfinir une pour chaque type de point de contrôle. Pour cela, les points de contrôles P ne seront plus définis comme étant des float, mais comme étant une classe **template** de la classe `bezier`.

▷ Créez un autre répertoire pour cet exercice en repartant des mêmes fichiers que pour l'exercice précédent.

Notre but va être de redéfinir la classe `bezier` comme étant une classe template. Au final, le code de la fonction `main()` précédent devra toujours fonctionner après avoir modifiés les appels à `bezier` en `bezier<float>`. Notez qu'il faut également modifier le paramètre de la fonction `export_matlab()`.

▷ Supprimez le fichier `bezier.cpp` de cet exercice, car le template devra entièrement être implémenté dans l'en-tête. Adaptez le Makefile en conséquence.

▷ Implémentez la classe template dans votre fichier `bezier.hpp` en suivant les consignes données ci-après.

- N'attendez pas d'avoir codé entièrement votre classe avant d'essayer de compiler et de l'utiliser dans votre `main()`. Faites cela par étapes, et décommentez au fur et à mesure le code de la fonction `main()`.
- Notez que l'on ne connait pas les propriétés du type ou de la classe template qui sera utilisé. On supposera qu'il devra vérifier les propriétés suivantes pour que le code compile:
  - Multiplication par un scalaire.
  - Addition interne (par le même type template).
  - Envoi possible dans un flux de sortie `std::ostream&` (pour l'affichage par `std::cout`).
- La classe template sera passée de préférence en paramètre des fonctions en tant que référence constante plutôt que par copie, car il pourra s'agir de classes autres que des float ou des doubles.

3- Courbe de Bézier 2D

Nous allons désormais utiliser la classe `bezier` template afin que celle-ci puisse servir à tracer des courbes dans le plan. Nous allons donc considérer des `bezier` du type `bezier<vec2>`, avec `vec2` désignant un point du plan (x,y).

▷ Considérez désormais le programme 3, et placez votre fichier `bezier.hpp` contenant votre implémentation template de courbe de Bézier.

▷ Vérifiez que le programme compile et s'exécute. Observez la fonction `main()` utilisée cette fois. Notez l'utilisation d'une classe `vec2` similaire à celle que vous avez déjà rencontrée auparavant. Si vous obtenez des erreurs dans le main ou pour l'`export_matlab`, cela signifie que vous devez modifier votre classe afin de satisfaire aux fonctions demandées.

▷ Lancez à nouveau la visualisation du fichier `data.m` sous Matlab, observez que cette fois, la courbe correspond à une courbe quelconque du plan.

4- Interaction avec Qt

- ▷ Considérez désormais le programme 4, et placez votre fichier `bezier.hpp` contenant votre implémentation template de courbe de Bézier.
- ▷ Vérifiez que le programme compile et s'exécute.
- ▷ Notez que vous pouvez cette fois interagir directement avec votre courbe par le biais d'une interface développée en Qt. Celle-ci suit le principe que vous connaissez avec des appels d'affichage dans la classe `render_area`.

5- Polymorphisme

Considérons désormais une scène 2D où sont placés des objets géométriques de natures différentes. Dans notre cas, on supposera qu'une scène pourra être constituée de cercles et de courbes de Bézier.

Lorsque l'utilisateur désigne un endroit de la scène, nous souhaitons connaitre le point de l'objet le plus proche, et dessiner le segment reliant la sélection de l'utilisateur à ce point de l'objet (voir exemple en figure 2).

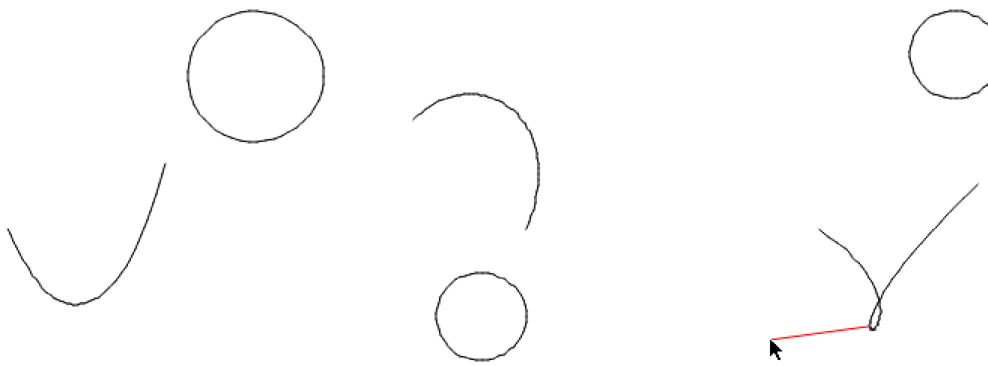

Figure 2: Exemple de scène contenant des arcs de courbes de Bézier et des cercles. Le segment rouge indique le chemin reliant la souris au point le plus proche par rapport à tous les objets.

L'algorithme de recherche du point le plus proche parmi l'ensemble des objets est le suivant:

```
p0 : point sélectionné par l'utilisateur
dist_min=infini
Pour tous les objets i de la scène
  pi : point de l'objet i le plus proche de p
  dist_i : distance entre pi et p
  Si di<dist_min
    dist_min=di
    p_plus_proche=pi
return p_plus_proche
```

Cet algorithme nécessite que l'on puisse connaitre pour un point p quelconque du plan, le point p_i le plus proche de p d'une forme géométrique de type cercle ou courbe de Bézier.

▷ Soit un cercle de centre c et de rayon R. Soit p un point quelconque du plan. Quelle est l'expression du point p_i le plus proche de p appartenant à ce cercle?

On définira dans la suite, une classe de cercle implémentant cette évaluation de point le plus proche.

Dans le cas de la courbe de Bézier, on utilisera une approche discrète approximée. Pour cela, on calculera N échantillons de la courbe, et on considère que le point p_i le plus proche de la courbe de Bézier est donné par l'échantillon le plus proche du point p.

Afin d'avoir une scène générique, nous souhaitons placer tous les objets géométriques dans un même conteneur, ce cette manière, il sera possible d'étendre aisément la scène à d'autres types de figures géométriques. On peut cependant noter que l'implémentation de la fonction de calcul du point le plus proche est différente si l'on considère un cercle, ou si l'on considère une courbe de Bézier. Pour n'avoir à traiter qu'un seul appel générique, nous allons utiliser une approche par **polymorphisme**. La classe cercle et la classe Bézier vont donc hériter d'une même classe parente permettant l'évaluation générique du point le plus proche. On nommera cette classe parente `geometrical_object`.

▷ Implémentez la méthode `closest_point` de la classe `bezier`. Cette méthode prendra en argument une position et renverra la position du point le plus proche. Cette méthode sera qualifiée de `const` au niveau de la classe, car elle ne modifie pas les attributs de celle-ci.

▷ Faites en sorte que votre classe `bezier` dérive d'une classe générique `geometrical_object`. Faite en sorte que la classe `geometrical_object` permette de rendre la méthode `closest_point` polymorphe, ainsi que l'évaluation d'un point de la courbe de Bézier en fonction de son paramètre (par le biais de la surcharge de l'opérateur()).

▷ Implémentez la classe `circle` qui dérivera également de `geometrical_object`. Un cercle sera défini par un centre c et un rayon R. Votre cercle devra posséder au moins une méthode permettant de calculer le point le plus proche, ainsi que d'évaluer un point du cercle suivant un paramètre s variant entre 0 et 1. On pourra supposer pour cela que votre cercle est paramétré par

$$c + R(\cos(2\pi\, s), \sin(2\pi\, s)) \; .$$

▷ Vérifiez sur quelques exemples simples le comportement polymorphe de vos classes.

6- Polymorphisme et interface Qt

Considérez les fichiers de l'exercice 6. Il s'agit cette fois d'un ensemble de fichiers réalisant une interface Qt qui présente une scène formée d'un ensemble de cercles et de courbes de Bézier. Lors d'un clic souris, le plus proche est affiché. Pour que le programme compile, vous devez ajouter vos fichiers: `bezier.hpp`, `circle.hpp`, `circle.cpp`, et `geometrical_object.hpp`

▷ Vérifiez le bon comportement de ce programme.

7- Ordre générique de la courbe

- ▷ Rappelez la relation entre C_n^k, C_{n-1}^k, et C_{n-1}^{k-1}.
- ▷ Créez une fonction permettant de calculer les valeurs des C_n^k au moment de la compilation. On pourra utiliser les `constexpr`.
- ▷ Modifiez votre classe de Bézier afin que celle-ci ait un ordre donné (un ordre n correspond à un polygone de contrôle de $n+1$ points) au moment de la compilation. La classe prendra donc désormais 2 paramètres templates: un type, et un entier donnant le degré du polynôme.
- ▷ Adaptez la fonction `export_matlab` afin que celle-ci puisse afficher une courbe de Bézier dont la taille du polygone de contrôle est caractérisée par un paramètre template. Faites en sorte que l'évaluation des N points de la courbe de Bézier à afficher soit réalisée en parallèle (les valeurs de la courbe seront temporairement stockées dans un vecteur avant d'être écrits dans le fichier dans l'ordre). Vérifiez visuellement que votre courbe correspond bien à une Bézier du degré fixé (voir exemple en figure 3).

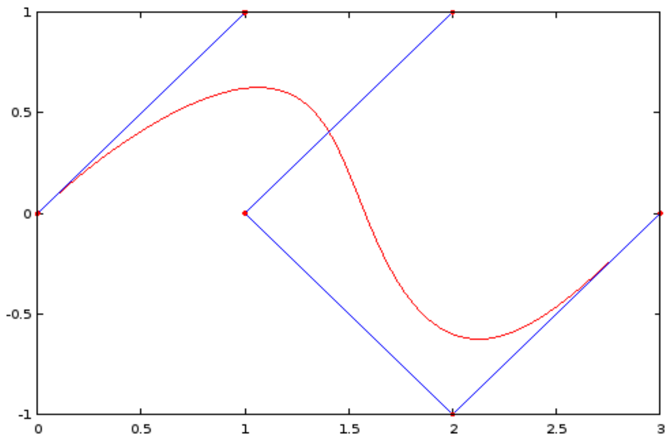

Figure 3: Exemple de courbe de Bézier de degré 5 et son polygone de contrôle.