TP5 – Splines cubiques

Le but de ce TP est de créer un programme permettant de dessiner des splines cubiques paramètrées fermées. Il ressemble un peu à ce que vous avez déjà fait avec les courbes de Bézier. Cependant les splines cubiques permettent de faire plus et seront donc un peu plus difficiles à mettre en place. Durant ce projet vous aurez besoin d'utiliser les polynômes et les matrices.

Un squelette de code est à télécharger sur ma page personnelle à l'adresse :

`http://www.lmpa.univ-littoral.fr/ fromentin/index.php`

Ce squelette comporte plusieurs fichiers.

- Les fichiers `interface.hpp` et `interface.cpp` sont les mêmes que ceux utilisés pour le TP sur les courbes de Bézier. Ils permettent un accès basique (mais suffisant) à la gestion des graphismes et des événements à l'aide de la librairie `SDL`. Vous n'aurez pas à modifier ces fichiers.
- Les fichiers `matrice.hpp` et `matrice.cpp` correspondent à la classe `Matrice`. Le code est la partie utile de ce que nous avons construit durant le TP 4. Vous n'aurez pas à modifier ces fichiers.
- Les fichiers `polynome.hpp` et `polynome.cpp` codent une classe `Polynome` plus simple que celle que vous avez créé durant le TP 2 car nous n'utiliserons que des polynômes de degré au plus 3. Vous serez amenés à modifier le fichier `polynome.cpp`
- Les fichiers `point.hpp` et `point.cpp` décrivent la classe `Point` permettant de représenter un point à l'écran. Vous serez amenés à modifier le fichier `point.cpp`
- Les fichiers `spline.hpp` et `spline.cpp` seront ceux codant la classe `Spline`. Le fonctionnement de cette classe sera décrit ultérieurement. Vous serez amenés à modifier le fichier `spline.cpp`
- Finalement il y a le fichier `main.cpp` qui décrit la boucle des événements de notre programme. Ce fichier devra être modifié dans ce projet.

1. Polynômes

La classe `Polynome` est constitué d'un tableau `coeffs` statique de taille 4. Le coefficient devant X^i est `coeffs[i]`. Par exemple pour `P` représentant $X^3 + 2X^2 - X + 4$ on aura `P.coeffs={1,2,-1,4}`.

Exercice 1. Dans le fichier `polynome.cpp`,

a. Coder le constructeur `Polynome::Polynome()` créant le polynome nul.

b. Coder le constructeur `Polynome::Polynome(const Polynome& P)` permettant de dupliquer le polynôme `P`.

Exercice 2. Dans le fichier `polynome.cpp`, coder la fonction

```
double Polynome::operator()(double t) const
```

permettant d'évaluer le polynôme courant en `t`.

2. Points

La classe `Point` est constituée de deux doubles `x` et `y`.

Exercice 3. Dans le fichier `point.cpp`, coder la fonction `void Point::dessine()` [1] dessinant le point de coordonnée (x, y) comme le carré de centre (x, y) et de longueur d'arête `2 taille +1` où `taille` est une constante fixée dans le fichier `point.hpp`.

Exercice 4. Dans le fichier `point.cpp`, coder la fonction `bool Point::est_proche(int cx,int cy)` permettant de déterminer si le point de coordonnées `(cx,cy)` est dans le carré représentant le point courant.

3. Spline

Une spline cubique est une courbe paramètrée passant par des points donnés. Entre deux point la spline est décrite par deux polynômes de degré 3. Plus précisément considérons la spline cubique passant par les points $P_0, ..., P_n$. Entre les points P_i et P_{i+1}, la spline est décrite par le couple de points $(q_i(t), r_i(t))$ où t parcours l'intervalle $[0, 1]$ et q_i, r_i sont des polynômes de degré 3. Le nombre de points maximums permettant de définir une spline cubique est donnné par `POINTS_MAX` et est défini dans le fichier `spline.hpp`.

La classe `Spline` est constituée :

- d'un tableau statique `P` de `Point` et de taille `POINTS_MAX+1`;
- d'un entier `n` correspondant au nombre de points définissant la spline;
- de deux tableaux statiques `q` et `r` de type `Polynome` et de taille `POINTS_MAX+1`;
- d'une variable `AInverse` de type `Matrice` contenant l'inverse d'une matrice `A` que nous allons introduire;
- d'un entier `point_selectionne` dont l'utilité sera donnée à la section 4.

Afin de gérer la dynamique de la construction de la spline au mieux, la classe `Spline` dispose d'une variable `statut` de type `StatutSpline` qui peut valoir `EnConstruction`, `Statique`, `Dynamique`. La fonction membre `StatutSpline Spline::lire_statut` permet de retourner le statut de la spline courante. Lors de sa phase de construction, la spline aura le statut `EnConstruction`. Lorsqu'elle aura été entièrement déterminée, elle aura le statut `Statique`. Le statut `Dynamique` sera l'objet de la section 4.

Exercice 5. Le but de cet exercice est de permettre l'ajout de points à la spline courante.

a. Dans le fichier `spline.cpp` coder la fonction `void Spline::ajoute_point(int x,int y)` qui ajoute le point de coordonnées (x, y) à la spline courante, si possible, c'est-à-dire si le nombre `n` de points ne vaut pas `POINTS_MAX`, et affiche un message d'erreur sinon.

b. Modifier le fichier `main.cpp` pour faire en sorte qu'un point soit ajouté à la spline courante si son statut est `EnConstruction`.

c. Dans le fichier `spline.cpp` coder la fonction `void Spline::dessine_points()` affichant tous les points de contrôle de la spline.

d. Dans le fichier `spline.cpp` modifier la fonction `void Spline::dessine()` pour permettre l'affichage des points de contrôle de la spline courante lors de son exécution (de la fonction `dessine`).

[1] Oui je sais, ce découpage en classe ne respecte pas le motif MVC (Modèle-Vue-Conrtolleur) utilisé pour les interfaces graphiques. Mais ce programme n'est pas une interface graphique. Pour la classe `Spline` ce sera pire `:p`.

Supposons que n points $P_0(x_0, y_0), ..., P_{n-1}(x_{n-1}, y_{n-1})$ soient donnés et sont tels qu'on ait $P_{n-1} = P_0$. Nous construisons alors la matrice de taille $n \times n$:

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots & 0 & -1 \\ 1 & 4 & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 4 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 & 4 & 1 \\ 2 & 1 & 0 & \dots & 0 & 1 & 2 \end{bmatrix}$$

ainsi que les matrices de taille $n \times 1$:

$$Bx = \begin{bmatrix} 0 \\ 6(x_2 - 2x_1 + x_0) \\ \vdots \\ 6(x_{n-1} - 2x_{n-2} + x_{n-3}) \\ 6(x_1 - x_0 + x_{n-2} - x_{n-1}) \end{bmatrix} \quad \text{et} \quad By = \begin{bmatrix} 0 \\ 6(y_2 - 2y_1 + y_0) \\ \vdots \\ 6(y_{n-1} - 2y_{n-2} + y_{n-3}) \\ 6(y_1 - y_0 + y_{n-2} - y_{n-1}) \end{bmatrix}.$$

On calcul alors les matrices s et t de taille $n \times 1$ par

$$s = \begin{bmatrix} s_0 \\ \vdots \\ s_{n-1} \end{bmatrix} = A^{-1} \times Bx \quad \text{et} \quad t = \begin{bmatrix} t_0 \\ \vdots \\ t_{n-1} \end{bmatrix} = A^{-1} \times By.$$

Pour $i = 0, ..., n-1$, le coefficient de X^3 dans `r[i]` est $\frac{s_{i+1} - s_i}{2}$, celui de X^2 est $\frac{s_i}{2}$, celui de X^0 est x_i et celui de X^1 est

$$x_{i+1} - x_i - \frac{s_{i+1} - s_i}{6} - \frac{s_i}{2}.$$

De même pour $i = 0, ..., n-1$, le coefficient de X^3 dans `q[i]` est $\frac{t_{i+1} - t_i}{2}$, celui de X^2 est $\frac{t_i}{2}$, celui de X^0 est y_i et celui de X^1 est

$$y_{i+1} - y_i - \frac{t_{i+1} - t_i}{6} - \frac{t_i}{2}.$$

Exercice 6. Modifier la fonction `void spline::calcule_Ainverse()` de `spline.cpp` pour

- **a.** Changer le statut de la spline en `Statique`.
- **b.** Affecter A^{-1} à la matrice `Ainverse` membre de la spline. La matrice A est celle définie précédemment.

Exercice 7. Le but de cet exercice est de déterminer les polynômes caractérisant la spline.

- **a.** Modifier la fonction `void Spline::calcule()` afin de calculer la matrice `Ainverse` seulement dans le cas où la spline à le statut `EnConstrution`.
- **b.** Modifier la fonction `void Spline::calcule()` du fichier `spline.cpp` pour calculer la matrice `t` introduite précédemment. Le code pour la matrice `s` est déjà donné en commentaire.
- **c.** Modifier la fonction `void Spline::calcule()` du fichier `spline.cpp` pour que `r[i]` ait les bons coefficients pour `i` dans {`0,...,n-1`}. Le code pour les `q[i]` est déjà donné en commentaire.

La fonction `calcule` est appelée lors de l'appui de la touche `S`.

Exercice 8.

a. Modifier la fonction `void Spline::dessine_spline()` du fichier `spline.cpp` pour afficher la spline. Pour tout `i` de {`0,...,n-1`}, on affcherai les points de coordonnées `(q[i](t),r[i](t))` où `t` va de 0 à 1 par pas de $0,001$.

b. Modifier la fonction `void Spline::dessine()` du fichier `spline.cpp` pour afficher la spline en plus des points de contrôles lorsque la spline a un statut différent de `EnConstruction`.

4. Interactions

Après l'affichage d'une spline nous souhaitons pouvoir recommencer à en tracer une nouvelle.

Exercice 9. Modifier la fonction `void Spline::reset()` remettant le statut de la spline à `EnConstruction` et permettant d'ignorer les points de contrôle déjà enregistrés. Cette fonction sera appelée lors de l'appui de la touche `R`.

Maintenant nous souhaitons pouvoir déplacer les points de contrôle d'une spline dessinée à l'écran (donc avec le statut `Statique`). L'indice du point de contrôle ainsi déplacé sera stocké dans la donnée `point_selectionne` de la classe `Spline`. Si aucun point de contrôle n'est sélectionné, `point_selectionne` vaudra `-1`.

Exercice 10.

a. Modifier la fonction `void Spline::selectionne_point(int cx,int cy)` du fichier `spline.cpp` tel que celle-ci affecte `Spline::point_selectionne` à l'indice du point de contrôle proche de la position `(cx,cy)` (position du curseur de la souris) ou `-1` si aucun point n'est proche. Si un point de contrôle est ainsi sélectionné, le statut de la spline devra passer à `Dynamique`. De plus on empêchera la sélection des points d'indice 0 et $n-1$.

b. Modifier le fichier `main.cpp` pour que la fonction `sectionne_point` soit appelée lorsque le bouton gauche de la souris est enfoncé et que la spline a le statut `Statique`.

c. Modifier la fonction `void Spline::deplace_point(int cx,int cy)` du fichier `spline.cpp` qui déplace le point précédemment sélectionné (s'il existe) à la position `(cx,cy)`, puis appelle la fonction `void Spline::calcule()`.

d. Modifier le fichier `main.cpp` pour que la fonction `deplace_point` soit appelée lors du déplacement de la souris et que la spline ait pour statut `Dynamique`.

e. Modifier la fonction `void Spline::fin_deplacement()` du fichier `spline.cpp` permettant de stopper le déplacement d'un point de contrôle. Le statut de la spline redeviendra alors `Statique`.

f. Modifier le fichier `main.cpp` pour que la fonction `fin_deplacement` soit appelée lorsque le bouton de la souris est relâché et que la spline ait pour statut `Dynamique`.