TP1 : Courbes de Bézier - Algorithme de De Casteljau

L'objectif de ce TP est d'implémenter l'algorithme de De Casteljau permettant l'évaluation de courbes de Bézier.

Courbe de Bézier

Soit la *base de Bernstein*

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \quad \text{où} \quad \binom{n}{i} = \frac{n!}{(n-i)!i!}$$

La courbe paramétrique

$$\mathbf{x}(t) = \sum_{i=0}^{n} \mathbf{b_i} B_i^n(t) \quad t \in [0,1]$$

est appelée *courbe de Bézier*. Les points $\mathbf{b_i}, i = 0, \ldots, n$ s'appellent *points de Bézier* (ou *points de contrôle*) et forment le *polygone de contrôle.*

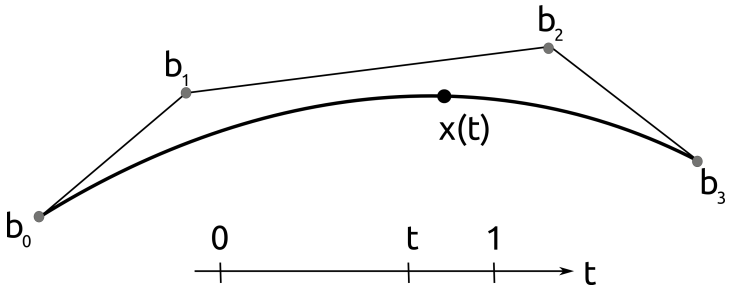

Figure 1: Exemple pour $n = 3$. Le polygone de contrôle donne une idée intuitive de la courbe.

Note : Pour un intervalle quelconque $[a, b]$, il faut simplement faire un changement de variable $u(t) = \frac{t-a}{b-a}$ pour revenir à la définition précédente.

Algorithme de De Casteljau (1959)

L'algorithme de De Casteljau sert à évaluer $\mathbf{x(t)}$ en un paramètre t avec les points de contrôles de la manière suivante :

$$\begin{cases} \mathbf{b_i^0} = \mathbf{b_i} & i = 0, \ldots, n \\ \mathbf{b_i^k} = (1-t)\mathbf{b_i^{k-1}} + t\mathbf{b_{i+1}^{k-1}} & k = 1 \ldots n \quad i = 0, \ldots n-k \\ \mathbf{x(t)} = \mathbf{b_0^n} \end{cases}$$

L'algorithme de De Casteljau est un algorithme triangulaire à n niveaux.

Travail demandé

Le TP est à faire en binôme. Le code et le rapport contenant les images et réponses aux questions sera rendu sous forme d'archive *nom1_nom2.zip*. Les algorithmes seront implémentés en C++.

Un code de départ vous est fourni. Il contient un système de compilation utilisant *CMake*, des utilitaires permettant de lire/écrire un fichier de points 2D (fichiers *utils.**) ainsi que deux fonctions que vous pourrez trouver utiles (fonctionne sous Linux mais aucune garantie sous Windows). Ces fonctions sont basées sur la librarie d'algèbre linéaire *Eigen* `http://eigen.tuxfamily.org` que vous pouvez utiliser pour ce projet.

Pour compiler un projet CMake, placer vous dans le dossier *build* et rentrer les commandes :

```
cmake ..
make
```

La première commande génère un makefile pour le projet et il suffit de compiler normalement ensuite.

Deux fichiers contenant des exemples de points de contrôle sont fournis dans le dossier *data* mais vous pouvez bien sûr créer vos propres exemples.

La visualisation des courbes de Bézier et des polygones de contrôle se fera à partir des sorties du programme C++ à l'aide de l'outil de votre choix (Gnuplot, R, ...). Une notice sommaire de Gnuplot et un code R de départ est mis à disposition.

1. Implémenter une fonction calculant un courbe de bézier en t pour un ensemble de points de contrôle $\mathbf{b_i}$.
2. Evaluer la courbe de bézier pour $t \in [0, 1]$ et visualiser la courbe.
3. Implémenter l'algorithme de De Casteljau.
4. Visualiser la même courbe que précédemment mais avec l'algorithme de De Casteljau.
5. Visualiser les polygones de contrôle à chaque étape de l'algorithme.
6. Quelles sont les avantages de l'algorithme de De Casteljau en comparaison d'une évaluation naïve d'une courbe de Bézier ?

Schéma

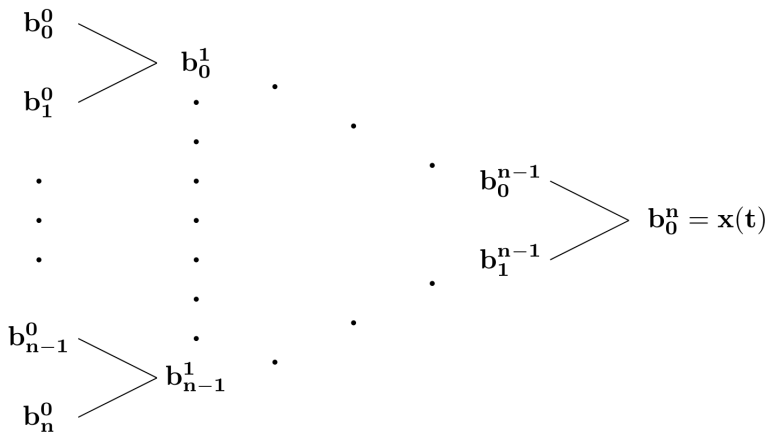

Figure 2: Illustration de l'algorithme de De Casteljau

Graphiquement

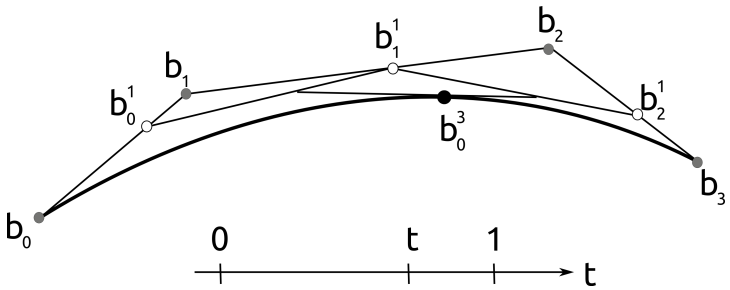

Figure 3: Illustration de l'algorithme de De Casteljau