TP6 : 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. Ces même courbes de Bézier peuvent entre autre définir des trajectoires pour les objets composant une scène virtuelle (caméra, modèle, lumière, ...).

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*.

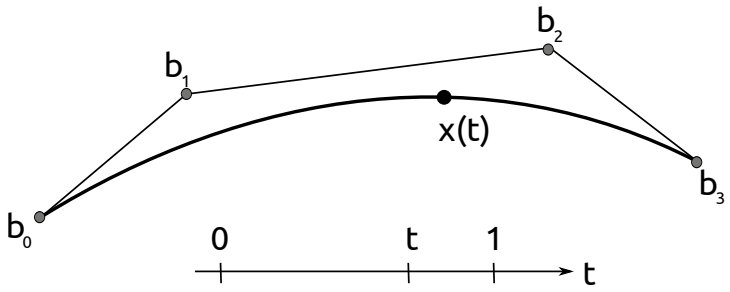

Figure 1: Exemple pour $n = 3$. Le polygone de contrôle donne une idée intuitive de la courbe.

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++.

Vous est fourni :

– Les fichiers `utils.*` contenant quelques fonctions utilitaires et les en-têtes des fonctions que vous devrez implémenter.

– Le fichier `data/simple.txt` contenant un exemple de points de contrôle.

1. Implémenter l'algorithme de De Casteljau dans les fichiers `utils.*`
2. Evaluer la courbe de bézier pour $t \in [0, 1]$ à l'initialisation et visualiser la courbe à l'aide du pipeline fixe d'OpenGL (voir ci-dessous).
3. Quelles sont les avantages de l'algorithme de De Casteljau en comparaison d'une évaluation naïve d'une courbe de Bézier ?
4. Faites se déplacer un objet le long d'une trajectoire définie par une courbe de Bézier en modifiant la matrice `model` à chaque dessin. La trajectoire 2D sera définie dans un plan que vous choisirez.
5. Changer les points de contrôles pour créer votre propre trajectoire. Vous pouvez l'appliquer à un autre composant que le modèle (caméra, lumière).

Schéma

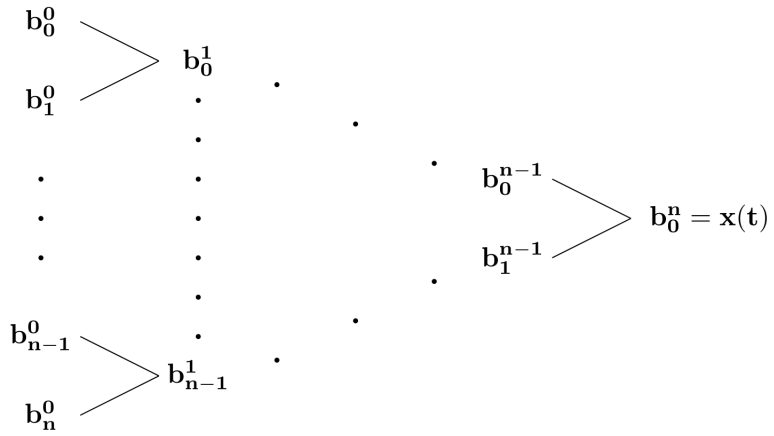

Figure 2: Illustration de l'algorithme de De Casteljau

Graphiquement

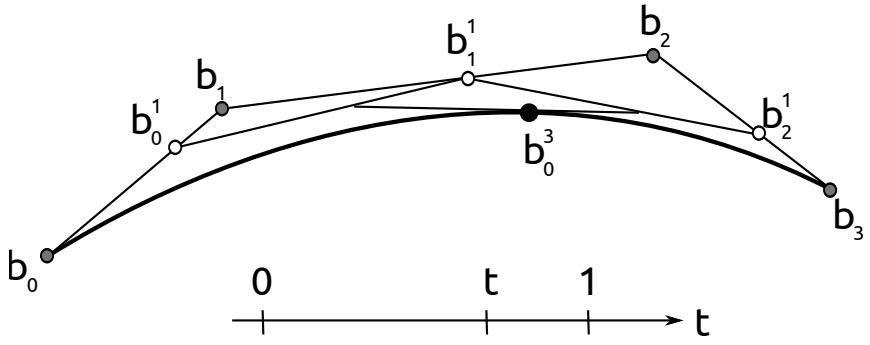

Figure 3: Illustration de l'algorithme de De Casteljau

OpenGL : Le pipeline fixe

Quand il s'agit d'afficher des primitives peu nombreuses et complètement statique, l'utilisation du pipeline fixe d'OpenGL s'avère plus simple que le nouveau pipeline programmable (shader). Voici un exemple simpliste d'utilisation :

```
//A appeler dans la boucle d'affichage
//apres glUseProgram(0)

void renderFix()
{
    // Projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadMatrixf( glm::value_ptr(camera.projection) );

    // Model view matrix
    glMatrixMode(GL_MODELVIEW);
    glm::mat4 model(1.0f);
    glm::mat4 modelView = camera.view*model;
    glLoadMatrixf( glm::value_ptr(modelView) );

    // On affiche un simple repere

    //Origine
    glPointSize(10.0);
    glBegin(GL_POINTS);
    glColor3f(0.0,0.0,0.0);
    glVertex3f(0.0,0.0,0.0);
    glEnd();

    //Axe
    glLineWidth(3.0);
    glBegin(GL_LINES);
    glColor3f(1.0,0.0,0.0);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(2.0f, 0.0f, 0.0f);
    glEnd();

    glBegin(GL_LINES);
    glColor3f(0.0,1.0,0.0);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 2.0f, 0.0f);
    glEnd();

    glBegin(GL_LINES);
    glColor3f(0.0f,0.0f,1.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 2.0f);
    glEnd();
}
```