TP3 - Courbes paramétriques

Léa Serrano M1 IMAGINE

27 Septembre 2022

Lien de mon git pour ce tp : https ://github.com/LeaSerrano/M1-IMAGINE-Modelisation3D-TP3.git

Le but de ce tp va être d'afficher plusieurs courbes paramétriques : la courbe cubique d'Hermique, la courbe de Béziers avec l'algorithme de Bernstein et la courbe de Béziers avec l'algorithme de Casteljau en utilisant OpenGL.

Table des matières

1 Exercice 1

Dans ce premier exercice, l'objectif va être de tracer une courbe cubique d'Hermite.

Nous avons donc, tout d'abord, écrit une fonction **DrawCurve** qui va nous permettre d'afficher nos courbes. On va, dans cette fonction, récupérer en entrée un liste de points et utiliser des fonctions d'OpenGL afin de tracer des segments entre chaque points côte à côte dans la liste.

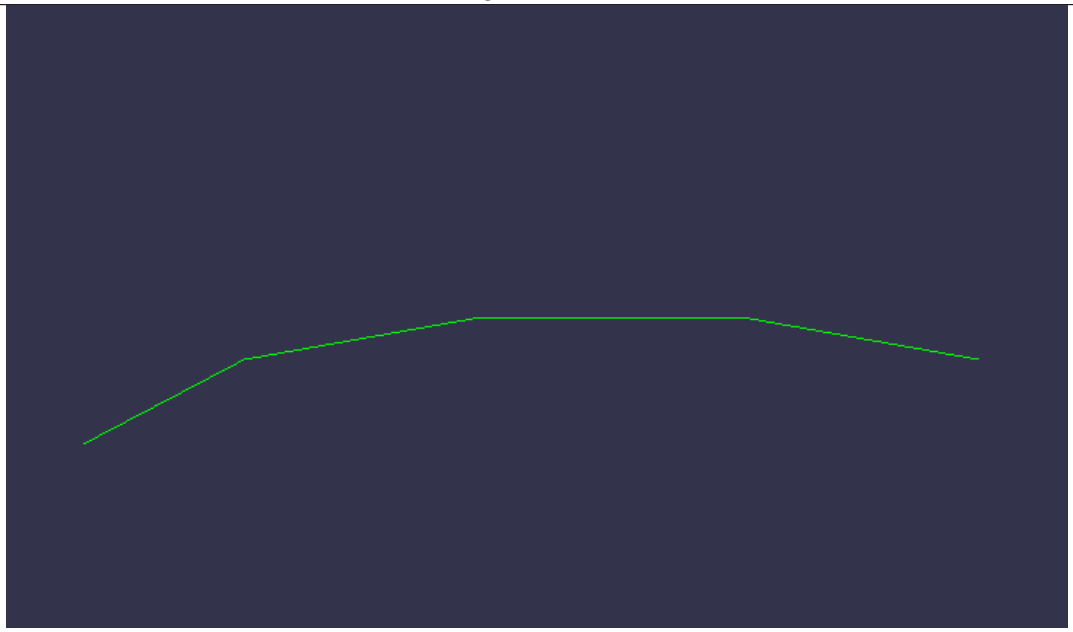
Ensuite nous avons écrit la fonction **HermiteCubicCurve** qui va permettre de calculer la courbe d'Hermite pour des points donnés. Elle va nous permettre de récupérer, en sortie, une liste de points à afficher pour pouvoir voir notre courbe d'Hermite.
Nous allons nous baser sur cette équation pour écrire notre fonction :

$$P(u) = F_1(u)P_0 + F_2(u)P_1 + F_3(u)V_0 + F_4(u)V_1$$

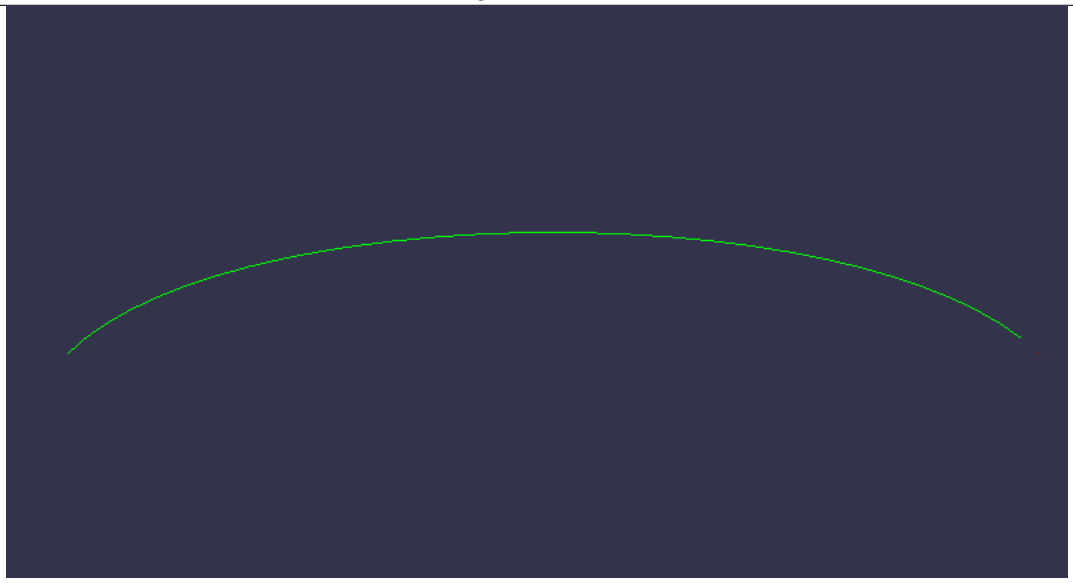
$$\text{avec} \begin{cases} F_1(u) = 2u^3 - 3u^2 + 1 \\ F_2(u) = -2u^3 + 3u^2 \\ F_3(u) = u^3 - 2u^2 + u \\ F_4(u) = u^3 - u^2 \end{cases}$$

On va vérifier que nos fonctions fonctionnent bien en affichant la courbe d'Hermite grâce à la fonction que nous avons écrite plus tôt avec les points P0(0, 0), P1(2, 0), V0(1,1) et V1(1,-1).
Voici ce qu'on obtient :

Affichage avec nbU = 5



Affichage avec nbU = 30



On voit qu'en fonction du nbU, on va avoir une courbe plus ou moins courbée.

2 Exercice 2

Nous allons maintenant tracer la courbe de Béziers par les polynômes de Bernstein.

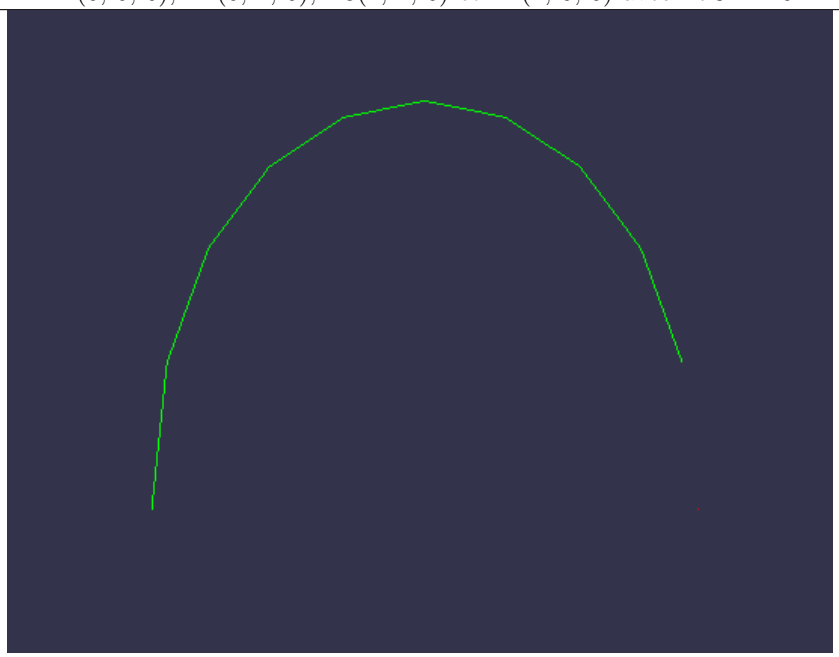
On va commencer par écrire la fonction **BezierCurveByBernstein** afin de tracer la courbe de Béziers. Nous allons nous implémenter cette équation :

$$B_i^n(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \quad , \quad i=0, \dots, n$$

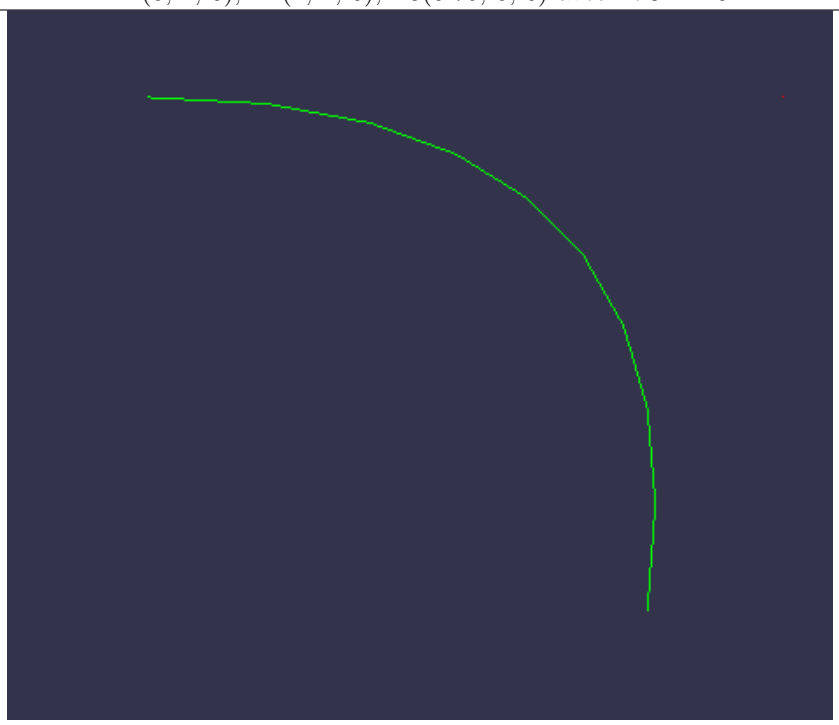
Nous allons ensuite récupérer, en sortie, une liste de points qui vont permettre d'afficher notre courbe de Béziers.

Plusieurs points ont été testés afin de vérifier que notre fonction est correcte :

P1(0, 0, 0), P2(0, 1, 0), P3(1, 1, 0) et P4(1, 0, 0) avec nbU = 10



P1(0, 1, 0), P2(1, 1, 0), P3(0.75, 0, 0) avec nbU = 10



Notre fonction est correcte et on voit qu'elle va se tracer en prenant en compte les points qu'on lui a donné en entrée.

3 Exercice 3

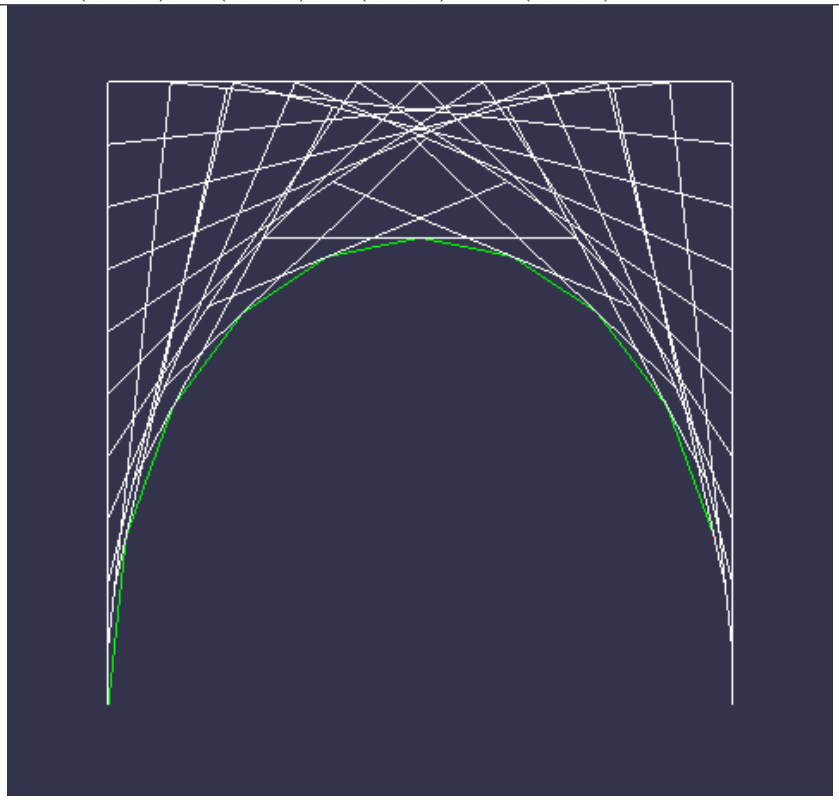
Pour ce dernier exercice, l'objectif va être d'afficher la courbe de Béziers comme à la question précédente, mais cette fois-ci nous utiliserons l'algorithme de Casteljau.

Nous avons donc écrit la fonction **BezierCurveByCasteljau** qui va nous permettre de définir la courbe à afficher.
Nous allons implémenter cette équation pour réaliser notre fonction :

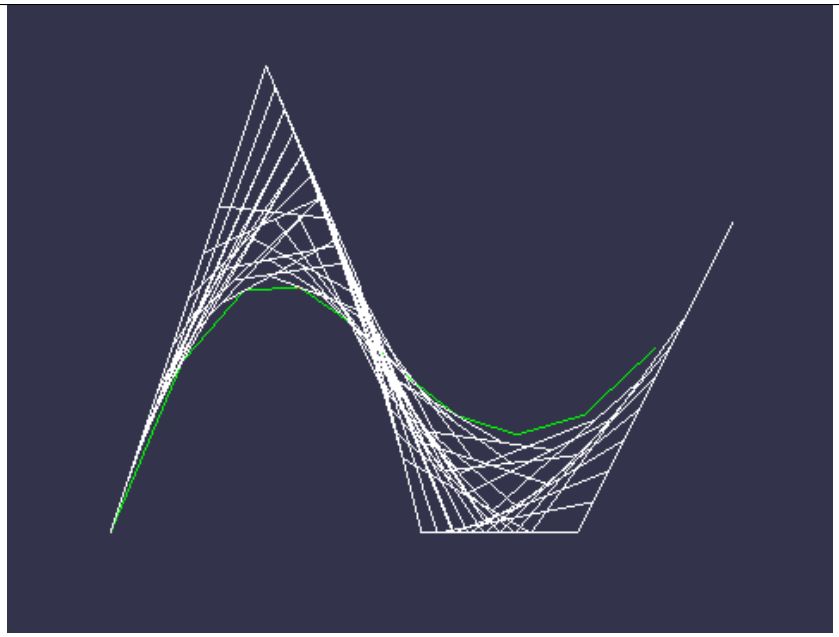
$$P_i^{k+1} = (1-u)P_i^k + uP_{i+1}^k$$

Nous avons aussi affiché les étapes intermédiaires de la construction afin de mieux voir comment l'algorithme fonctionne.

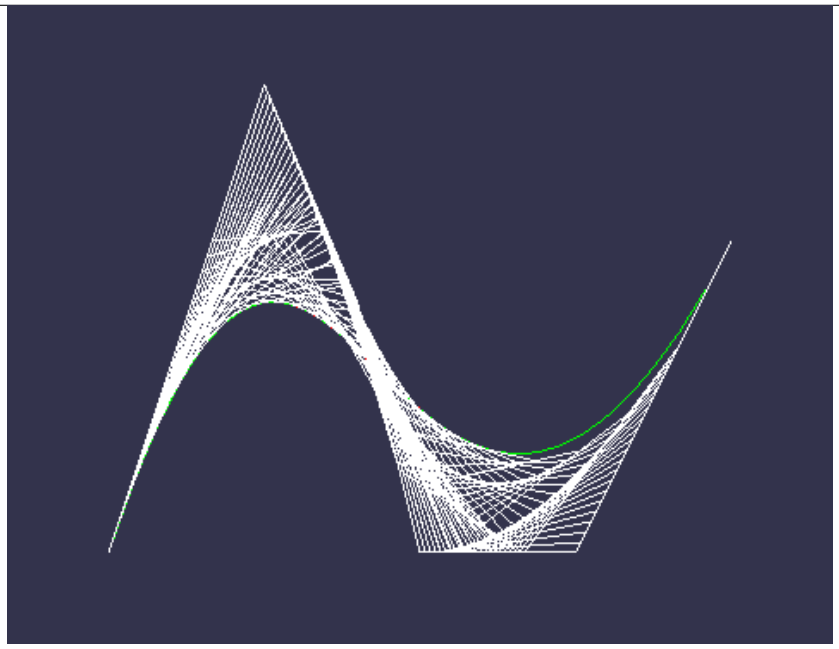
P1(0, 0, 0), P2(0, 1, 0), P3(1, 1, 0) et P4(1, 0, 0) avec nbU = 10



P1(0, 0, 0), P2(0.25, 0.75, 0), P3(0.4, 0.4, 0), P4(0.5, 0, 0), P5(0.75, 0, 0), P6(1, 0.5, 0) et P7(1, 0, 0) avec nbU = 10



P1(0, 0, 0), P2(0.25, 0.75, 0), P3(0.4, 0.4, 0), P4(0.5, 0, 0), P5(0.75, 0, 0), P6(1, 0.5, 0) et P7(1, 0, 0) avec nbU = 30



On peut voir en blanc, toutes les étapes intermédiaires. On voit aussi que plus nbU est grand, plus on a d'étapes intermédiaires de construction. Cela est dû au fait que plus le nbU est grand, plus la courbe est courbée et moins on voit les segments, on aura donc besoin de plus d'étapes de construction.