COMPTE RENDU
MATH

PROJET DE MATHÉMATIQUES

AU SEIN DE

L'ECOLE

Esirem Dijon

RÉALISÉ PAR : VINCENT **CANDAPPANE**

ENSEIGNANT : M. LIONEL **GARNIER**

11 janvier 2017

TABLE DES MATIÈRES

INTRODUCTION

Dans le cadre du module **ITC311** de Mathématique au sein de *l'*ESIREM *Ecole Supérieure d'Ingénieurs Matériaux/Infotronique* de Dijon, nous allons réaliser un projet qui portera sur le thème de la construction de courbe avec diverses notions.

Le but du projet est la construction d'une courbe reliant, de façon G^2, deux segments par une courbe. Le cas particulier de segments orthogonaux ou parallèles peut apporter une solution particulière.

L'objectif de ce projet est l'implémentation de code en langage C/C++ en utilisant les librairies **OpenGL**, en mode console de manière à pouvoir construire une courbe grâce à 2 segments. OpenGL est une librairie open source qui permet de faciliter la programmation lorsqu'il s'agit d'interagir avec un environnement graphique. C'est cette librairie qui enverra directement les instructions à la carte graphique de l'ordinateur.

Dans un premier temps, afin de mieux comprendre le projet, nous aborderons les différentes notions mathématiques théoriques nécessaires. Nous commencerons par introduire la notion de jointure ainsi que la signification de G^2. Par la suite, nous evoqueront la notion de rayons de courbure, essentielles à la compréhension d'une jointure façon G^2. Nous finirons par traiter des courbes de Bézier polynômiale afin de nous permettre de relier de façon G^2 deux segments. Enfin, pour mettre en place les solutions trouvées, nous décrirons les algorithmes nécessaires pour le fonctionnement du programme.

Afin d'implémenter le code demandé et les différentes fonctions, nous avons d'abord besoin de définir, de manière théorique, les notions mathématiques nécessaires.

I - NOTION THÉORIQUE

Dans cette partie, nous allons nous pencher sur l'aspect théorique du sujet. Nous donnerons les définitions des termes découvert lors de notre recherche. Cette étude nous permettra de faire un choix sur le type de courbe que nous utiliserons et modéliserons par la suite.

1 – JOINTURE DE TYPE G^2

Lorsque nous faisons la jonction entre deux courbes, deux critères peuvent être considérés : la continuité C^0, C^1 C^2 entre les chemins ; la continuité des propriétés géométriques (espace tangent, courbures). Dans le premier cas, nous parlons de jointure algébrique et dans le second cas de jointure géométrique.

A. CONTINUITÉ ALGÉBRIQUE C^0, C^1 ET C^2

Considérons deux segments de droites [AB] et [CD] reliés par une courbe P(t) (entre les points B et C) avec $t \in [0,1]$. Le raccordement de P(0) (début de courbe) avec B est dit de continuité :

- C^0 si et seulement si P(0)=B soit une égalité des points;
- C^1 si et seulement si cela respecte C^0 et que P'(0)=AB(égalité des tangentes);
- C^2 si et seulement si cela respecte C^1 et que l'on a une égalité des accélérations. (idem pour P(1))

B. CONTINUITÉ GÉOMÉTRIQUE G^0, G^1 ET G^2

Dans le cadre du projet, nous nous intéresserons uniquement aux jointures géométriques car l'objectif est d'étudier une jointure façon G^2 entre deux segments.

La continuité C^1 traduit l'égalité des tangentes en direction et en norme. Dans notre cas, il est possible de se limiter à une égalité des directions mais pas des normes, car nous travaillons ici sous un modèle informatique. On définit donc des raccordements avec des continuités géométriques :

- G^0 si et seulement si P(0)=B soit une égalité des points;
- G^1 si et seulement si cela respecte G^0 et que qu'il y a colinéarité des vecteurs tangentes : $P'(0)=k\tilde{AB}$ avec $k \in R$;
- G^2 si et seulement si cela respecte G^1 et que l'on a des centres de courbures identiques en B et P(0).

C. INTERPRETATION DE LA CONTINUITÉ GÉOMETRIQUE

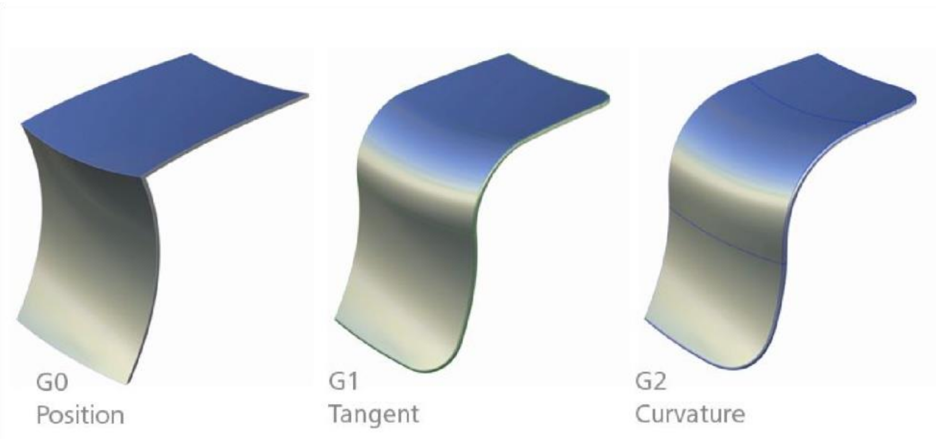
Nous avons plusieurs façons de tracer une courbe, il nous est demandé dans ce projet, de construire une courbe de façon G^2. Avant de définir G^2, nous allons expliquer ce que représente $\mathbf{G^0}$, $\mathbf{G^1}$ et enfin $\mathbf{G^2}$.

Position (G^0)
La position (continuité G^0) ne tient compte que de la position des objets. Si les extrémités de chaque courbe se trouvent au même endroit dans l'espace, les courbes présentent une continuité de position (G^0) au niveau de leur extrémité commune. En d'autres termes, les deux courbes en question se touchent au niveau de leurs extrémités.

Tangence (G^1)
La tangence (continuité G^1) mesure la position et la direction de la courbe aux extrémités. En d'autres termes, les deux courbes se touchent et se dirigent dans la même direction au point où elles se touchent. La direction est déterminée par le premier et le deuxième point de chaque courbe. Si ces deux points tombent sur une ligne, les deux courbes sont tangentes au niveau de leur extrémité commune. La première dérivée des deux courbes est égale au point où elles se touchent.

Courbure (G^2)
La continuité de courbure (continuité G^2) entre deux courbes tient compte de la position, de la direction et du rayon de courbure aux extrémités. Si le rayon de courbure est le même sur chaque courbe au niveau de leur extrémité commune, les courbes présentent une continuité de courbure (G^2). En d'autres termes, les courbes ont la même direction au point où elles se rencontrent mais elles ont également le même rayon en ce point. Cette condition n'est pas facile à déterminer en regardant simplement la position des points. La première et la deuxième dérivée des équations sont égales en ce point.

Nous pouvons observer sur la figure suivante, une **représentation graphique** effectuée sur des surfaces afin de mieux visualiser $\mathbf{G^0}$, $\mathbf{G^1}$ et $\mathbf{G^2}$.



D. RAYON DE COURBURE

Afin d'obtenir une jointure G^2 entre deux segments, il nous faut une courbure nulle aux points où nous souhaitons relier notre courbe et nos extrémités de segments. Mathématiquement, la courbure $\rho(t_0)$ à la courbe γ en un point est définie par :

$$\rho(t_0) = \frac{|det(\overrightarrow{\gamma'}(t_0); \overrightarrow{\gamma''}(t_0))|}{||\overrightarrow{\gamma'}(t_0)||^3}$$

Sachant que notre courbure est nulle, le rayon de courbure définie par :

$$R(t_0) = \frac{1}{\rho(t_0)}$$

Le rayon de courbure est infini. Nous avons donc $\rho(t_0)$ qui tend vers 0 car le déterminant tend vers 0. Pour obtenir une jointure G^2 entre deux segments nous utiliserons les courbes de Béziers car elles permettent de répondre aux conditions que nous imposent la jointure G^2 pour les espaces tangents et les rayons de courbure.

2 - POLYNÔME DE BERNSTEIN ET COURBES DE BÉZIERS

Afin de mener à bien et d'avancer dans le projet (continuité géométrique G^2) concernant les jointures entre la courbes et les segments [AB] et [CD] nous allons nous intéresser aux polynômes de Bernstein et aux courbes de Bézier.

A. POLYNÔMES DE BERNSTEIN

Afin de définir les courbes de Bézier, nous devons définir les polynômes de Bernstein qui sont eux-même définies à partir des combinaisons et des factorielles. Nous allons considérer comme acquis les notions citées au-dessus. Ainsi, nous définirons seulement les polynômes de Bernstein :
Soit n appartenant à $\mathbb{N} - \{0;1\}$. Pour $i \in [[0;n]]$, le i-ème polynôme de Bernstein de degré n est

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}$$

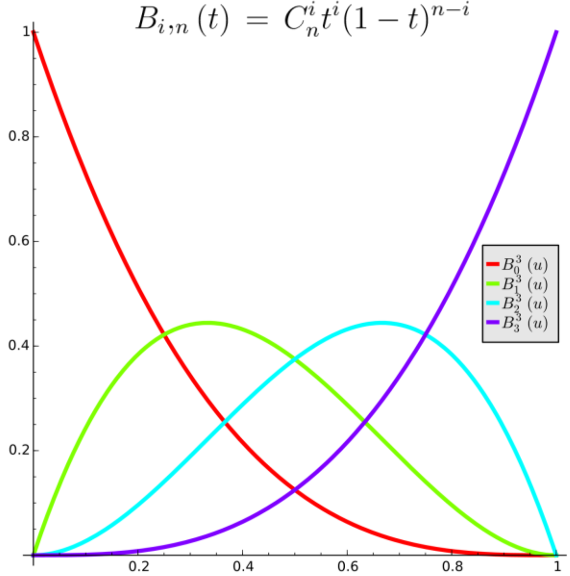


Figure : Tracé du poynôme de Bernstein de degré 3

B. COURBE DE BÉZIER DE DEGRÉ N :

Ainsi, nous pouvons maintenant définir une courbe de Bézier de degré n : Soit n appartenant à N-{0;1}. Soit $(M_i)_{i\in[[0;n]]}$ et O, $n+2$ points de ε. La courbe de Bézier de degré n de points de contrôle $(M_i)_{i\in[[0;n]]}$ est l'ensemble des points $M(t), t \in [0;1]$, vérifiant la définition et formule suivante :

Définition :

La définition des courbes de Bézier utilise les polynômes de Bernstein. La courbe de Bézier associée à $n+1$ points $P_0,\ldots,P_n$ de $\mathbb{R}^2$ est la courbe paramétrée $P : [0,1] \to \mathbb{R}^2$ donnée pour tout $t \in [0,1]$ par :

$$P(t) = \sum_{i=0}^{n} B_i^n(t) P_i$$

où B_i^n est le polynôme de Bernstein $B_i^n : t \to C_i^n t^i (1-t)^{n-i}$.

Propriétés :

On établit les propriétés suivantes pour une courbe de Bézier :

— Elle a pour extrémités les points P_0 et P_n ($P(0) = P_0$ et $P(1) = P_n$).

— Le vecteur P_0P_1 est tangent à la courbe de Bézier au point de paramètre t=0.

— Le vecteur $P_{n-1}P_n$ est tangent à la courbe de Bézier au point de paramètre t=1.

— La courbe est "attirée" par les points $P_1,\ldots,P_{n-1}$.

— La courbe de Bézier ne dépend pas du repère choisi. Si on effectue une rotation des points de contrôles, la forme de la courbe de Bézier reste la même.

— Les points $P_0,\ldots,P_n$ sont appelés points de contrôles de la courbe de Bézier, ce qui est assez naturel. En effet, la courbe dépend de ces points. Quand on les bouge, on modifie la courbe qui est "attirée" par ces points. Plus précisément, chaque point $P(t_0)$ de la courbe de Bézier est combinaison linéaire des points de contrôles $P_0,\ldots,P_n$:

Les courbes de Bézier sont couramment utilisées en informatique car elles permettent de construire des courbes régulières satisfaisant des contraintes géométriques simples. Voici une comparaison :

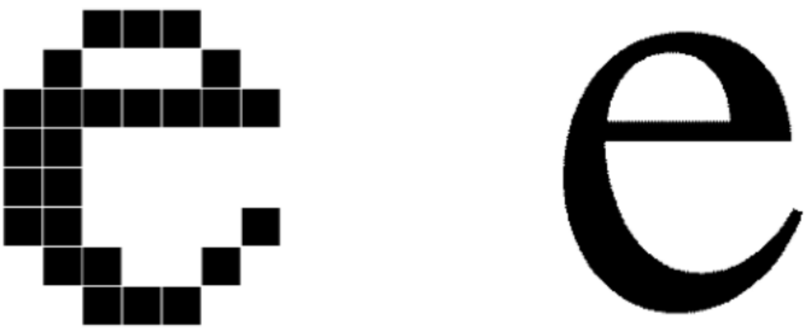


Figure : A gauche la lettre e grossie avec un ordinateur des années 1980, à droite avec un ordinateur actuel

Dans la suite de ce projet, ce qui nous intéresse est d'intégrer les différentes notions vues précédemment à un programme informatique. Avant d'arriver à cette étape, nous avons besoin de trouver les points de contrôle de la courbe de Bézier que nous voudrions tracer pour relier nos deux segments de façon G^2.

II – POINTS DE CONTÔLE : ÉQUATIONS

1 – CALCULS DE POINTS DE CONTRÔLE & COURBES DE BÉZIER

Nous savons qu'une courbe paramétrée est définie par : $M(t) = \begin{cases} x(t) \\ y(t) \end{cases}$

Nous prenons alors deux segments de droites $[AB]$ et $[CD]$ définis par les points :

- $A(x_a, y_a)$,
- $B(x_b, y_b)$,
- $C(x_c, y_c)$,
- $D(x_d, y_d)$.

Soit les vecteurs directeurs de ces segments de droites :

$$\overrightarrow{AB} = \begin{pmatrix} x_b - x_a \\ y_b - y_a \end{pmatrix} \qquad \overrightarrow{CD} = \begin{pmatrix} x_d - x_c \\ y_d - y_c \end{pmatrix}$$

Une courbe de Bézier est définie pour $t \in [0;1]$. Nous étudierons donc les cas pour $t = 0$ et $t = 1$.

❖ Pour une **jointure G^0** entre deux segments, AB et CD, aux points B et C, nous avons comme conditions :

$$M(0) = B(x_b, y_b) \quad et \quad M(1) = C(x_c, y_c)$$

❖ Ensuite, pour répondre à la définition d'une **jointure G^1**, nous avons besoin que :

$$\begin{pmatrix} x'(0) \\ y'(0) \end{pmatrix} = k_1 \overrightarrow{AB} \qquad et \qquad \begin{pmatrix} x'(1) \\ y'(1) \end{pmatrix} = k_2 \overrightarrow{CD}$$

❖ Enfin, pour avoir une **jointure G^2**, le centre de courbure d'un point d'un segment est nul :

$$x'(0)y''(0) - x''(0)y'(0) = 0 \qquad et \qquad x'(1)y''(1) - x''(1)y'(1) = 0$$

A. 4 POINTS DE CONTRÔLE (COURBE DE BÉZIER D'ORDRE 3) :

Nous utiliserons toutes les conditions posées précédemment afin de pouvoir poser le système d'équation pour 4 points. Il faudra pour cela calculer ce système de façon à le rendre libre.

Pour avoir une jointure G^2, nous devons prendre un polynôme de degré 3 minimum. Nous obtenons donc :

$x(t) = (1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3$

$x'(t) = 3[(1-t)^2(x_1 - x_0) + 2t(1-t)(x_2 - x_1) + t^2(x_3 - x_2)]$

$x''(t) = 6[(1-t)(x_0 - 2x_1 + x_2) + t(x_1 - 2x_2 + x_3)]$

Grâce au développement précédent, nous arrivons à obtenir un système de 8 équations libres. Pour $x_a \neq x_b$ *et* $x_c \neq x_d$, nous avons :

$$\begin{cases} x(0) = x_b \\ y(0) = y_b \\ x(1) = x_c \\ y(1) = y_c \\ y'(0) = \frac{y_b - y_a}{x_b - x_a} x'(0) \\ y'(1) = \frac{y_d - y_c}{x_d - x_c} x'(1) \\ x'(0)y''(0) - x''(0)y'(0) = 0 \\ x'(1)y''(1) - x''(1)y'(1) = 0 \end{cases}$$

Afin d'avoir les 4 points de contrôles de la Courbe de Bézier de degré 3 recherchée, il suffit de procéder à la résolution des équations suivantes :

$$\begin{cases} x(0) = x_0 \\ x(1) = x_3 \end{cases} \quad et \quad \begin{cases} y(0) = y_0 \\ y(1) = y_3 \end{cases}$$

$$\begin{cases} x'(0) = 3(x_1 - x_0) \\ y'(0) = 3(y_1 - y_0) \end{cases} \quad et \quad \begin{cases} x'(1) = 3(x_3 - x_2) \\ y'(1) = 3(y_3 - y_2) \end{cases}$$

$$\begin{cases} x''(0) = 6(x_0 - 2x_1 + x_2) \\ y''(0) = 6(y_0 - 2y_1 + y_2) \end{cases} \quad et \quad \begin{cases} x''(1) = 6(x_1 - 2x_2 + x_3) \\ y''(1) = 6(y_1 - 2y_2 + y_3) \end{cases}$$

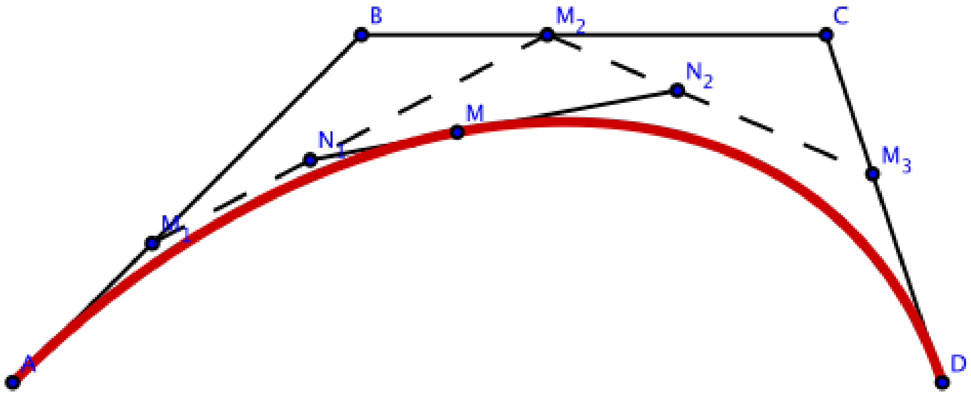
En posant :

$$\alpha = \frac{y_b - y_a}{x_b - x_a} \quad et \quad \beta = \frac{y_d - y_c}{x_d - x_c}$$

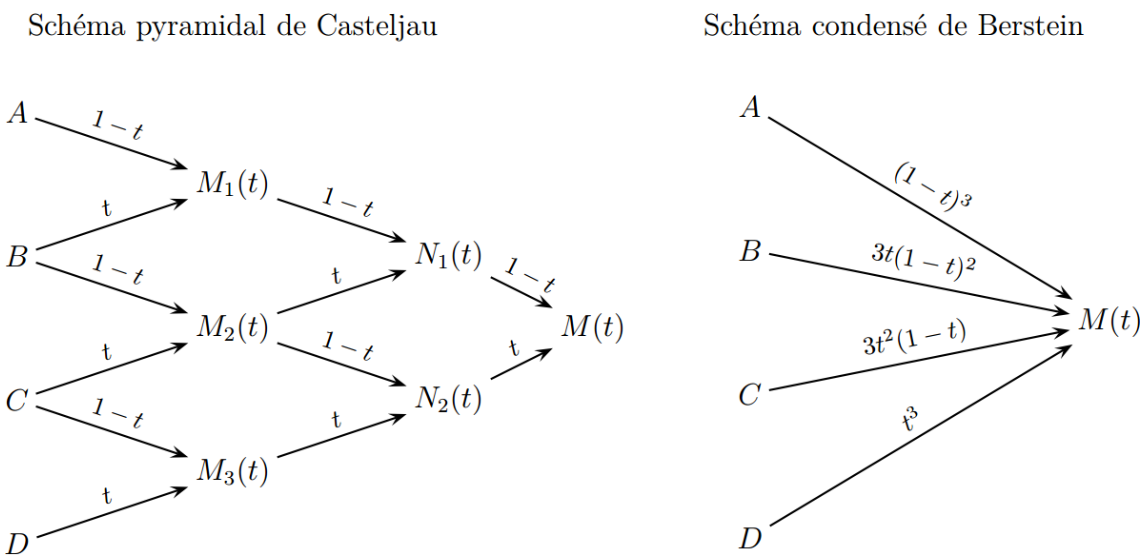
Nous obtenons donc nos 4 points de contrôles :

- $P_0[x_0 = x_b \,;\, y_0 = y_b]$
- $P_1\left[x_1 = \frac{-y_0 + \alpha x_0 + y_3 - \beta x_3}{\alpha - \beta} \,;\, y_1 = y_0 + \alpha(x_1 - x_0)\right]$
- $P_2\left[x_2 = \frac{y_0 - \alpha x_0 - y_3 + \beta x_3}{\beta - \alpha} \,;\, y_2 = y_3 - \beta(x_3 - x_2)\right]$
- $P_3[x_3 = x_c \,;\, y_3 = y_c]$

En faisant varier t dans l'intervalle [0 ; 1], le point M(t) décrit la courbe ci-dessous :



Remarque : Il y a dans ce projet plusieurs manière de calculer les points de contrôles, comme par exemple utiliser les algorthme de Casteljau ou d'utiliser directement les polynomes de Bernstein. Dans notre cas, nous n'avons pas utilisé les algorithmes de Casteljau et la notion de barycentre, qui aurait pu être une solution. Voici une illustration :



Dans notre cas nous avions utilisé le schéma de Bernstein.

B. 5 POINTS DE CONTRÔLE (COURBE DE BÉZIER D'ORDRE 4) :

De le même façon, dans le cas où l'on souhaite 5 points de contrôle, il nous faut poser k=1 (respectivement k' = 1), k étant le coefficient de colinéarité entre $\overrightarrow{AB}$ et P'(0) (respectivement k' le coefficient de colinéarité entre $\overrightarrow{CD}$ et P'(1)). Nous obtenons donc le développement de M(t), M'(t) et M''(t) au degré 4 :

$x(t) = (1-t)^4 x_0 + 4t(1-t)^3 x_1 + 6t^2(1-t)^2 x_2 + 4t^3(1-t)x_3 + t^4 x_4$
$x'(t) = 4(1-t)^3(x_1 - x_0) + 12t(1-t)^2(x_2 - x_1) + 12t^2(1-t)(x_3 - x_2) + 4t^3(x_4 - x_3)$
$x''(t) = 12(1-t)^2(x_2 - 2x_1 + x_0) + 24t(1-t)(x_3 - 2x_2 + x_1) + 12t^2(x_4 - 2x_3 + x_2)$

Dans notre projet, il a été judicieux de choisir le nombre de point de contrôle favorable à la construction de la courbe de **façon G^2**. Ce nombre de point de contrôle permet à la courbe d'être recalculé lors de chaque agrandissement, ce qui évite les phénomènes de **pixellisation'**.

En effet, plus le nombre de point de contrôle utilisés est important, mieux la courbe est tracée et plus la courbe est régulière (satisfaisant des contraintes géométriques). De plus cela permettra de résoudre les cas particuliers.

C. 6 POINTS DE CONTRÔLE (COURBE DE BÉZIER D'ORDRE 5) :

Il semble ne pas y avoir de cas particulier avec des courbes de Bézier de degré 5. Il est donc intéressant d'étudier ce cas fastidieusement, nous allons utiliser dans ce projet 6 points de contrôle.

De la même manière, dans le cas où l'on souhaite 6 points de contrôle, il nous faut poser k=1, k étant le coefficient de colinéarité entre $\overrightarrow{AB}$ et P'(0) ainsi que l'egalité suivante :

$$x''(0) = x''(1) = y''(0) = y''(1) = 0.$$

Nous obtenons donc le développement de M(t), M'(t) et M''(t) au degré 5 ci-dessous :

$x(t) = (1-t)^5 x_0 + 5t(1-t)^4 x_1 + 10t^2(1-t)^3 x_2 + 10t^3(1-t)^2 x_3 + 5t^4(1-t)x_4 + t^5 x_5$
$x'(t) = 5(1-t)^4(x_1 - x_0) + 20t(1-t)^3(x_2 - x_1) + 30t^2(1-t)^2(x_3 - x_2) + 20t^3(1-t)(x_4 - x_3) + 5t^4(x_5 - x_4)$
$x''(t) = 20(1-t)^3(x_2 - 2x_1 + x_0) + 60t(1-t)^2(x_3 - 2x_2 + x_1) + 60t^2(1-t)(x_4 - 2x_3 + x_2) + 20t^3(x_5 - 2x_4 + x_3)$

Ce développement permet de trouver après résolution du système, les points de contrôle suivants :

$$\begin{cases} x_0 = x_B \\ y_0 = y_B \\ x_5 = x_C \\ y_5 = y_C \\ x_1 = \frac{6x_B - x_A}{5} \\ y_1 = \frac{6y_B - y_A}{5} \\ x_4 = \frac{6x_C - x_D}{5} \\ y_4 = \frac{6y_C - y_D}{5} \\ x_2 = \frac{7x_B - 2x_A}{5} \\ y_2 = \frac{7y_B - 2y_A}{5} \\ x_3 = \frac{7x_C - 2x_D}{5} \\ y_3 = \frac{7y_C - 2y_D}{5} \end{cases}$$

III – CODE OPENGL :

1 – INSTALLATION DE LIBRAIRIE POUR OPEN_GL ET C++:

Nous tenons à préciser que l'installation et la simulation vont être faites sur une distribution **GNU Linux Mint 18**.

Afin de pouvoir compilé le squelette du code ainsi que le code modifé, il nous a fallu installer certaine librairie et fixer certains bugs. Avant l'installation de l'outil **Open GL** (*librairie permettant d'interagir avec un environnement graphique*), nous avons tout d'abord installés les paquets suivants :

- build-essential ;
- autoconf ;
- Automak ;
- libxmu-dev ;
- g++ :

1. Voici les commandes du Shell pour l'installation complète :

```
su -        //Pour devenir root
yum install build-essential autoconf automake libxmu-dev g++
```

```
//Installation de freeglut3-dev
sudo apt-get install freeglut3-dev
```

```
//Installation de libjpeg-dev
sudo apt-get install libjpeg-dev
```

2. Librairie d'en-tête Glut et autres à ajouter dans le code :

```
#include <stdio.h>
#include <stdlib.h>
#include <jpeglib.h>
#include <GL/glut.h>  //Librairie glut : OpenGL
#include <math.h>     //Librairie pour calcul Mathématique
#include <string.h>
#include <fstream>
#include <iostream>
#include <jerror.h>
```

3. Nous lançerons ensuite notre code grâce aux deux commandes ci-dessous :

```
//Compilation du code & création d'un makefile
make
```

```
//Lancement du programme Open GL
./OpenGL
```

2– CODE :

Les constructions de courbe se déclinent de plusieurs manières. Le code fourni permet déjà de tracer (*fonction trace_segment*) et de simuler un plan, de tracer des points et des segments.

1 - Tous d'abord il faut récupérer les coordonnées des points A, B, C et D. Deux méthodes s'y présentent : Soit nous entrons en brut les valeurs de ces points dans le code, soit nous demandons à l'utilisateurs de rentrer les valeurs adéquates. Nous avons opté pour la 2eme option :

```
double xA, yA, xB, yB, xC, yC, xD,
yD
cout"Entrer les coordonnees des
points 'A,B,C,D' des deux segments
AB et CD"<<endl
  cout<<"Entrer A(x,y):"<<endl
  cin>>xA
  cin>>yA
  cout<<"Entrer B(x,y):"<<endl
  cin>>xB
  cin>>yB
  cout<<"Entrer C(x,y):"<<endl
  cin>>xC
  cin>>yC
  cout<<"Entrer D(x,y):"<<endl
  cin>>xD
  cin>>yD
```

2 - Nous avons ensuite calculé, les rayons de courbures pour chaque coordonnée la liaison de courbe des extrémités de segments. Mathématiquement, la courbure ρ(t0) à la courbe γ en un point est définie par :

$$\rho(t_0) = \frac{|det(\overrightarrow{\gamma'}(t_0);\overrightarrow{\gamma''}(t_0))|}{||\overrightarrow{\gamma'}(t_0)||^3}$$

Nous l'avons implémenté de la façon suivante, le système prenant en compte les coordonnées des 6 point de contrôle trouvé précédemment(via le système) avec la formule de la courbure :

```
  double xM1=xB
  double yM1=yB
  double xM2=(6*xB-xA)/5
  double yM2=(6*yB-yA)/5
  double xM3=(7*xB-2*xA)/5
  double yM3=(7*yB-2*yA)/5
  double xM4=(7*xC-2*xD)/5
  double yM4=(7*yC-2*yD)/5
  double xM5=(6*xC-xD)/5
  double yM5=(6*yC-yD)/5
  double xM6=xC
  double yM6=yC
```

On dérive ensuite deux fois la courbe de bézier et on évalue en 0 et en 1.

3 - Avec Open GL, il est nécessaire d'initialiser et d'instancier chaque objet devant s'afficher sur l'écran, sinon rien ne s'affiche. Voici par exemple l'initalisation des listes (ici, numérotés).

4 - Voici la création des points O, I, J via la liste 1 :

```
  //création des points O,I et J
  glNewList(1,GL_COMPILE_AND_EXECUTE); //liste numero 1
    openGL(xI,yI,1.,0.,0.,10.); //I
    openGL(xJ,yJ,0.,0.5,0.,10.); //J
    openGL(xO,yO,0.,0.,1.,15.);//O
  glEndList();
```

5 - Puis la création segments [OI] et [OJ] :

```
//création des segments [OI] et [OJ]
  glNewList(2,GL_COMPILE_AND_EXECUTE);  //liste numero 2
    trace_segment(xO,yO,xI,yI,1.0,0.0,1.0,2.0); // on trace [OI]
    trace_segment(xO,yO,xJ,yJ,1.0,0.50,0.0,2.0);// on trace [OJ]
  glEndList();
```

6 – Une 3eme liste qui va permettre la visualisation des points A, B, C et D avec des couleurs différentes, via les fonctions `openGL` :

```
                                       //liste numero 3
glNewList(3,GL_COMPILE_AND_EXECUTE);
    openGL(xA,yA,0.,0.,1.,5.)          //A
    openGL(xB,yB,1.,0.,0.,5.)          //B en rouge
    openGL(xC,yC,1.,0.,0.,5.)          //C en rouge
    openGL(xD,yD,0.,0.5,0.,5.)         //D
  glEndList()
```

7 - Puis la création des segments [AB] et [CD], à partir desquels la courbe sera tracée :

```
//création des deux segments [AB] et [CD]
  glNewList(4,GL_COMPILE_AND_EXECUTE);  //liste numero 4
    trace_segment(xA,yA,xB,yB,0.0,0.0,1.0,2.0); //on trace [AB] en bleu
    trace_segment(xC,yC,xD,yD,0.0,0.5,0.0,2.0); //on trace [CD] en vert
  glEndList();
```

8 - Nous avons ensuite calculé grâces aux coordonnées des points, les 6 points de contrôles via une boucle `for` et grâce à la fonction `glVertex2f`. Nous utiliserons dans notre boucle le tableau à 2 dimensions(abscisses/ordonnée) initié au début : `tab_point_control[6][2],`

```
//création des points de controle
  glNewList(5,GL_COMPILE_AND_EXECUTE);  //liste numero 5
    glPointSize(5.0);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_POINTS);
      for (i = 0; i < 6; i++)
         glVertex2f(tab_point_control[i][0], tab_point_control[i][1]);
    glEnd();
  glEndList();
```

9 – Finalement, nous implémentons le code de manière à créer la courbe de façon G^2, via une boucle `for`. Cette boucle permet de tracer la courbe utilisant les 6 points de contrôle ainsi que l'equation calculée théoriquement via la courbe de bézier de degré 5 :

```
//création de la courbe de façon G2
  glNewList(6,GL_COMPILE_AND_EXECUTE); //liste numero 6
      //trace_courbe(1.0,0.0,0.0,2.0);
      for (t=0;t<=1;t+=0.0001){
        xt=pow(1-t,5)*xM1+5*t*pow(1-t,4)*xM2+10*t*t*pow(1-
t,3)*xM3+10*pow(t,3)*pow(1-t,2)*xM4+5*pow(t,4)*(1-t)*xM5+pow(t,5)*xM6;
        yt=pow(1-t,5)*yM1+5*t*pow(1-t,4)*yM2+10*t*t*pow(1-
t,3)*yM3+10*pow(t,3)*pow(1-t,2)*yM4+5*pow(t,4)*(1-t)*yM5+pow(t,5)*yM6;
        openGL(xt,yt,1.,0.,0.,2.);
      }
  glEndList();
```

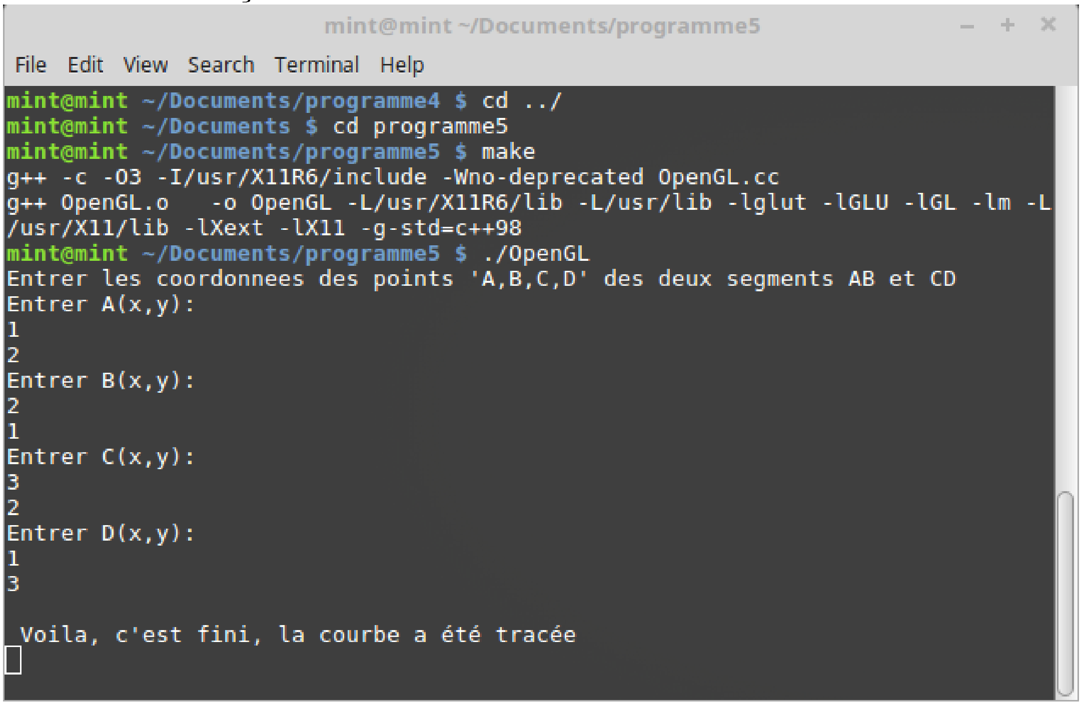
10 - Il suffit ensuite de réaliser un simple appel à chacunes des listes crées via la fonction `glCallList(n)` de la librairie openGL, n étant le numéro de liste :

```
glRotatef(-anglex+angley,0.0,0.0,1.0);
  glScalef(Scal,Scal,Scal); // diminution de la vue de la scene
  glRotatef(180,0.0,1.0,0.0);
  glRotatef(180,1.0,0.0,0.0);
  glTranslatef(-trX,trY,0.);
      glCallList(1); // appel de la liste numero 1
      glCallList(2); // appel de la liste numero 2
      glCallList(3); // appel de la liste numero 3
      glCallList(4); // appel de la liste numero 4
      glCallList(5); // appel de la liste numero 3
      glCallList(6); // appel de la liste numero 4
 glFlush();
  // On echange les buffers
  glutSwapBuffers();
```

[1] *le code C++ complet est visualisable dans la partie annexes.*

3 – SIMULATIONS DES CAS PARTICULIERS :

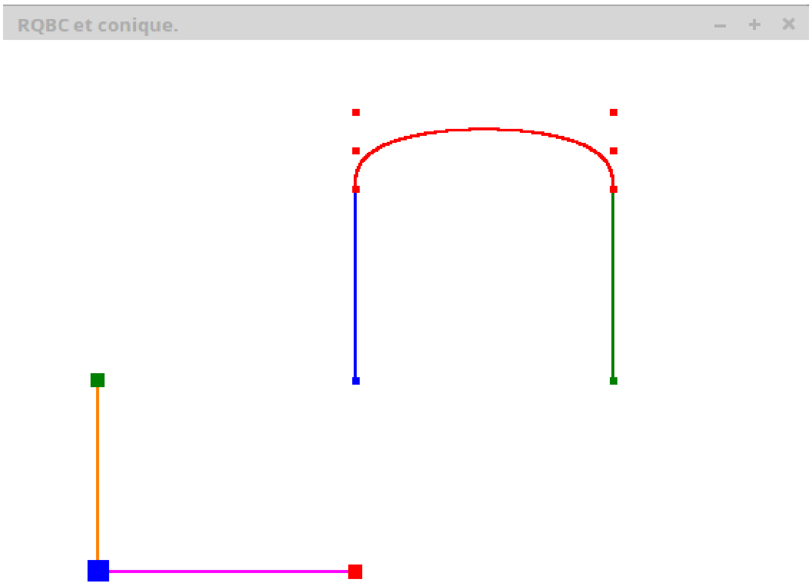
Voici comment se déroule l'initialisation des coordonnées lors du lancement du terminal, avant le traçage de la courbe de façon G^2 :



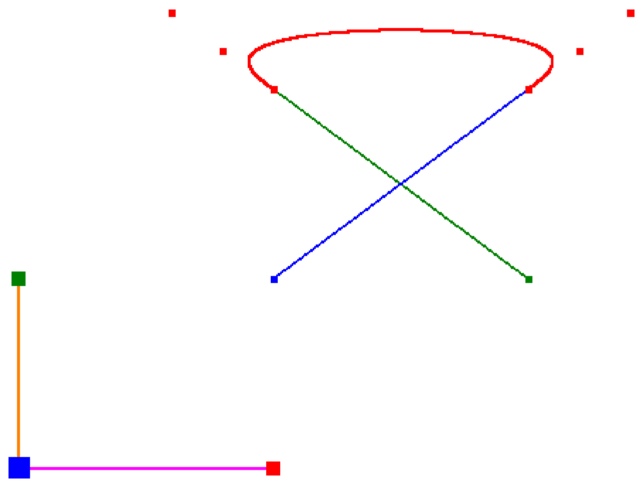
```
mint@mint ~/Documents/programme5
File Edit View Search Terminal Help
mint@mint ~/Documents/programme4 $ cd ../
mint@mint ~/Documents $ cd programme5
mint@mint ~/Documents/programme5 $ make
g++ -c -O3 -I/usr/X11R6/include -Wno-deprecated OpenGL.cc
g++ OpenGL.o   -o OpenGL -L/usr/X11R6/lib -L/usr/lib -lglut -lGLU -lGL -lm -L
/usr/X11/lib -lXext -lX11 -g-std=c++98
mint@mint ~/Documents/programme5 $ ./OpenGL
Entrer les coordonnees des points 'A,B,C,D' des deux segments AB et CD
Entrer A(x,y):
1
2
Entrer B(x,y):
2
1
Entrer C(x,y):
3
2
Entrer D(x,y):
1
3

 Voila, c'est fini, la courbe a été tracée
```

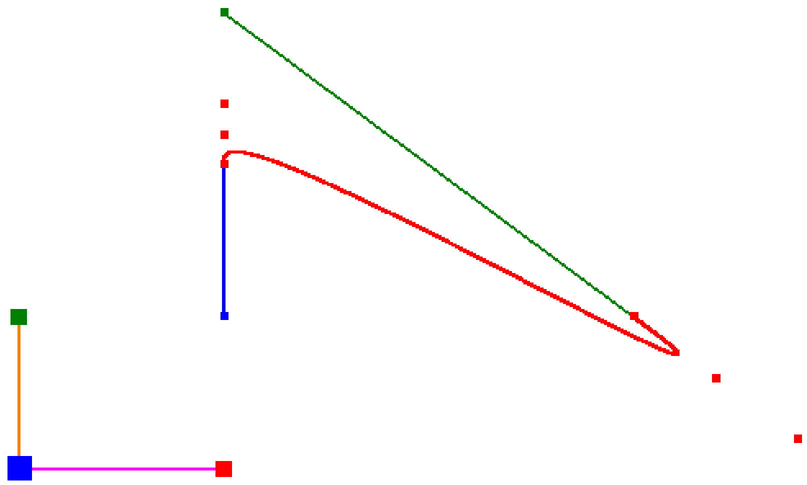
1 – Simulation de construction de la courbe lorsque les segments sont **parallèles** :



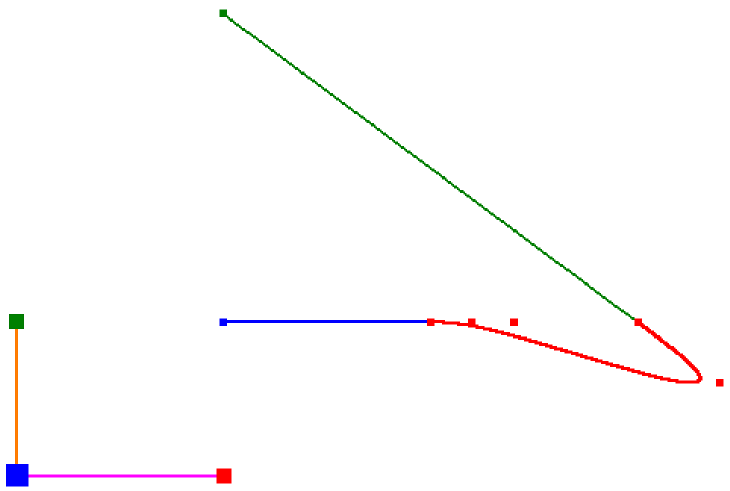
2 - Simulation de construction de courbe lorsque les segments sont **perpendiculaires** :



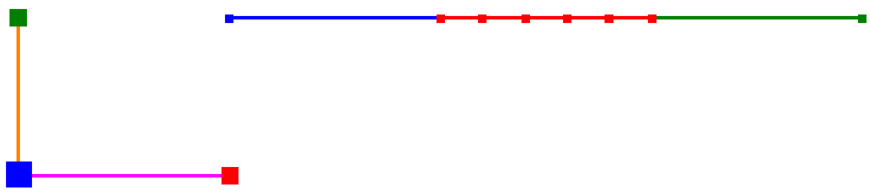
3 - Simulation de construction de courbe lorsqu'un segment (ici AB) est **vertical** :



4 - Simulation de construction de courbe lorsqu'un segment (ici AB) est **horizontal** :



5 - Simulation de construction de courbe lorsque les 2 segments (AB et CD) sont **horizontaux** :



6 - Simulation de construction de courbe lorsque 2 points sont **confondus** :

CONCLUSION GÉNÉRALE

Les polynômes de Bernstein, les courbes de Bézier, les systèmes de calculs, les espaces-vectoriel, le langage C++ ainsi que l'outil Open GL, GLUT sont particulièrement adaptés à l'étude de traçage de courbe dans l'espace de façon G^2.

A travers ce projet, nous avons pu cerner la problématique de continuité géométrique, de notion droite paramétrée, la puissance des polynômes de Bernstein ainsi que les différents cas particuliers résolus ici avec un degré 5. Cette étude pourra cependant être affiner afin de pouvoir modéliser des objets de la vie réelle.

DOCUMENTATION :

Livre de Lionel GARNIER : *Mathématiques pour la modélisation géométrique, la représentation 3D et la synthèse d'images. Ellipses, 2007*

Youtube : *Cours sur les courbes parametrées*

Site :

http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf
https://www.math.u-psud.fr/~perrin/CAPES/geometrie/BezierDP.pdf
http://lyceeenligne.free.fr/IMG/pdf/CourbesParametrees-Cours.pdf
http://docs.mcneel.com/rhino/5/help/fr-fr/popup_moreinformation/continuity_descriptions.htm
http://www.aliasworkbench.com/theoryBuilders/TB3_continuity1.htm
http://members.gamedev.net/skyork/pdfs/Bspline_Construction_Summary2005.pdf
https://www.irif.fr/~carton/Enseignement/InterfacesGraphiques/MasterInfo/Cours/Swing/splines.html
https://en.wikipedia.org/wiki/B-spline#Cubic_B-Spline http://cagd.cs.byu.edu/~557/text/ch2.pdf
https://fr.wikipedia.org/wiki/Courbe_de_B%C3%A9zier

RÉALISÉ PAR
VINCENT **CANDAPPANE**

ANNEXES :

Implémentation du code C++ :

```
/*********************************************************/
/*                    openGL.cc                          */
/*********************************************************/
/*                                                       */
/*********************************************************/

/* inclusion des fichiers Glut-OpenGL, de Mathématiques et de C++ */
#include <GL/glut.h>
#include <math.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <jpeglib.h>
#include <jerror.h>
#define Pi 3.141592654
using namespace std;

double Scal=36;

double trX=0.0,trY=0.0,dist=0.;//,trZ=0.0
char presse;
int anglex,angley,x,y,xold,yold;

/* Prototype des fonctions */
void affichage();//  procedure a modifier en fonction de la scene
void clavier(unsigned char touche,int x,int y);
void reshape(int x,int y);
void idle();
void mouse(int bouton,int etat,int x,int y);
void mousemotion(int x,int y);
void openGL(double x, double y, double r0,double g0, double b0, double size);
void trace_segment(double x0, double y0,double x1, double y1, double red, double
green,    double blue, double size);
void trace_courbe(double red, double green, double blue, double size);
//-************************************************************
//
//  Procedure avec mise en file des sommets des primitives
//
//-************************************************************
void init();

int main(int argc,char **argv)
{
  /* initialisation de glut et creation de la fenetre */
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(1000,1000);
  glutCreateWindow("RQBC et conique.");
  /* Initialisation d'OpenGL */
  glClearColor(1.0,1.0,1.0,0.0);
  glColor3f(0.0,0.0,0.0);
  glPointSize(2.0);
```

```
glEnable(GL_DEPTH_TEST);

   glColor3f(0.0,0.0,0.0);
   //glEdgeFlag(GL_FALSE);
   glEdgeFlag(GL_TRUE);
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
 // glEnable(GL_LIGHTING);
  glDisable(GL_LIGHTING);

  /* enregistrement des fonctions de rappel */
  init();
  glutDisplayFunc(affichage);
  glutKeyboardFunc(clavier);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutMotionFunc(mousemotion);
  /* Entree dans la boucle principale glut */
  glutMainLoop();
  return 0;
}

void clavier(unsigned char touche,int x,int y)
{
  switch (touche)
    {
    case 'q' : /*la touche 'q' permet de quitter le programme */
        exit(0);
    case '+' :
      dist+=0.5;
      Scal=Scal+0.5;
      glutPostRedisplay();
      break;
    case '-' :
      dist-=0.5;
      Scal=Scal-0.5;
      glutPostRedisplay();
      break;
    case '6' : trX-=0.25;  glutPostRedisplay();  break;
    case '4' : trX+=0.25;  glutPostRedisplay();  break;
    case '8' : trY+=0.25;  glutPostRedisplay();  break;
    case '2' : trY-=0.25;  glutPostRedisplay();  break;
    }
}

void reshape(int x,int y)
{
  glViewport(0, 0, (GLsizei) x, (GLsizei) y);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  //taille de la scene
  double  Ortho=-150;
  glOrtho(-Ortho,Ortho,-Ortho,Ortho,-Ortho,Ortho);// fenetre
  glMatrixMode(GL_MODELVIEW);
  glViewport(0,0,x,y);
}

void mouse(int button, int state,int x,int y)
{
  /* si on appuie sur le bouton gauche */
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
  {
```

```
      presse = 1; /* le booleen presse passe a 1 (vrai) */
      xold = x; /* on sauvegarde la position de la souris */
      yold=y;
    }
    /* si on relache le bouton gauche */
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
      presse=0; /* le booleen presse passe a 0 (faux) */
  }

  void mousemotion(int x,int y)
    {
      if (presse) /* si le bouton gauche est presse */
      {
        /* on modifie les angles de rotation de l'objet
     en fonction de la position actuelle de la souris et de la derniere
     position sauvegardee */
        anglex=anglex+(x-xold);
        angley=angley+(y-yold);
        glutPostRedisplay(); /* on demande un rafraichissement de l'affichage */
      }

      xold=x; /* sauvegarde des valeurs courante de le position de la souris */
      yold=y;
    }

  /*****************************************************************
   **                                                             **
   **                    Affichage de la scene                    **
   **                                                             **
   *****************************************************************/

  void openGL(double x, double y, double r0,double g0, double b0, double size)
  {
    glColor3f(r0,g0,b0);  //initialisation de la couleur
    glPointSize(size);  // initialisation de la taille
    glBegin(GL_POINTS); // on trace un point
    glVertex2f(x,y);  // coordonnees du point
    glEnd();  // fin de glBegin
  }

  void trace_segment(double x0, double y0,double x1, double y1, double red, double
green, double blue, double size)
  {
    glColor3f(red,green,blue);//initialisation de la couleur
    glLineWidth(size); // initialisation de la taille
    glBegin(GL_LINES); // on trace un segment
    glVertex2f(x0,y0); // coordonnees du premier point
    glVertex2f(x1,y1); // coordonnees du dernier point
    glEnd(); // fin de glBegin
  }

  //fonction ou les objets sont a definir
  void init()
  {
    int i;
         //CAS PARTICULIER :
//double xA=1, yA=1, xB=1, yB=2, xC=2, yC=2, xD=2, yD=1;  --> PARALLELLE INVERSER
//double xA=1, yA=1, xB=1, yB=2, xC=2, yC=1, xD=2, yD=2;  --> PARALLELLE inv
//double xA=1, yA=1, xB=2, yB=2, xC=1, yC=2, xD=2, yD=1;  --> PERPENDICULAIRE 1
//double xA=1, yA=1, xB=2, yB=2, xC=2, yC=1, xD=1, yD=2;  --> PERPENDICULAIRE 2
//double xA=1, yA=1, xB=1, yB=2, xC=3, yC=1, xD=1, yD=3;  --> AB vertical
//double xA=1, yA=1, xB=2, yB=1, xC=3, yC=1, xD=1, yD=3;  --> AB horizontal
```

```
//double xA=1, yA=1, xB=2, yB=1, xC=3, yC=1, xD=4, yD=1;  --> AB & CD horizontaux
//double xA=1, yA=1, xB=2, yB=3, xC=2, yC=3, xD=6, yD=6;  --> AB & CD confonfus


         double xA, yA, xB, yB, xC, yC, xD, yD;
    cout<<"Entrer les coordonnees des points 'A,B,C,D' des deux segments AB et CD"<<endl;
    cout<<"Entrer A(x,y):"<<endl;
    cin>>xA;
    cin>>yA;
    cout<<"Entrer B(x,y):"<<endl;
    cin>>xB;
    cin>>yB;
    cout<<"Entrer C(x,y):"<<endl;
    cin>>xC;
    cin>>yC;
    cout<<"Entrer D(x,y):"<<endl;
    cin>>xD;
    cin>>yD;

    double xM1=xB;
    double yM1=yB;
    double xM2=(6*xB-xA)/5;
    double yM2=(6*yB-yA)/5;
    double xM3=(7*xB-2*xA)/5;
    double yM3=(7*yB-2*yA)/5;
    double xM4=(7*xC-2*xD)/5;
    double yM4=(7*yC-2*yD)/5;
    double xM5=(6*xC-xD)/5;
    double yM5=(6*yC-yD)/5;
    double xM6=xC;
    double yM6=yC;

    double t,xt,yt;
    for (t=0;t<=1;t+=0.01){
      xt=pow(1-t,5)*xM1+5*t*pow(1-t,4)*xM2+10*t*t*pow(1-t,3)*xM3+10*pow(t,3)*pow(1-t,2)*xM4+5*pow(t,4)*(1-t)*xM5+pow(t,5)*xM6;
      yt=pow(1-t,5)*yM1+5*t*pow(1-t,4)*yM2+10*t*t*pow(1-t,3)*yM3+10*pow(t,3)*pow(1-t,2)*yM4+5*pow(t,4)*(1-t)*yM5+pow(t,5)*yM6; }

    GLdouble tab_point_control[6][2] = {
    {xM1, yM1}, {xM2, yM2},
    {xM3, yM3}, {xM4, yM4},
    {xM5, yM5}, {xM6, yM6}};

    double xO=0.,yO=0.,xI=1.,yI=0.,xJ=0.,yJ=1.;
    //création des points O,I et J
    glNewList(1,GL_COMPILE_AND_EXECUTE); //liste numero 1
      openGL(xI,yI,1.,0.,0.,10.); //I
      openGL(xJ,yJ,0.,0.5,0.,10.); //J
      openGL(xO,yO,0.,0.,1.,15.);//O
    glEndList();
    //création des segments [OI] et [OJ]
    glNewList(2,GL_COMPILE_AND_EXECUTE);  //liste numero 2
      trace_segment(xO,yO,xI,yI,1.0,0.0,1.0,2.0); // on trace [OI]
      trace_segment(xO,yO,xJ,yJ,1.0,0.50,0.0,2.0);// on trace [OJ]
    glEndList();
    glNewList(3,GL_COMPILE_AND_EXECUTE); //liste numero 3
      openGL(xA,yA,0.,0.,1.,5.); //A
      openGL(xB,yB,1.,0.,0.,5.); //B en rouge
      openGL(xC,yC,1.,0.,0.,5.);//C en rouge
      openGL(xD,yD,0.,0.5,0.,5.);//D
```

```
  glEndList();
  //création des deux segments [AB] et [CD]
  glNewList(4,GL_COMPILE_AND_EXECUTE);  //liste numero 4
    trace_segment(xA,yA,xB,yB,0.0,0.0,1.0,2.0); //on trace [AB] en bleu
    trace_segment(xC,yC,xD,yD,0.0,0.5,0.0,2.0); //on trace [CD] en vert
  glEndList();
  //création des points de controle
  glNewList(5,GL_COMPILE_AND_EXECUTE);  //liste numero 5
    glPointSize(5.0);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_POINTS);
      for (i = 0; i < 6; i++)
         glVertex2f(tab_point_control[i][0], tab_point_control[i][1]);
    glEnd();
  glEndList();
  //création de la courbe de façon G2
  glNewList(6,GL_COMPILE_AND_EXECUTE); //liste numero 6
      //trace_courbe(1.0,0.0,0.0,2.0);
      for (t=0;t<=1;t+=0.0001){
        xt=pow(1-t,5)*xM1+5*t*pow(1-t,4)*xM2+10*t*t*pow(1-t,3)*xM3+10*pow(t,3)*pow(1-t,2)*xM4+5*pow(t,4)*(1-t)*xM5+pow(t,5)*xM6;
        yt=pow(1-t,5)*yM1+5*t*pow(1-t,4)*yM2+10*t*t*pow(1-t,3)*yM3+10*pow(t,3)*pow(1-t,2)*yM4+5*pow(t,4)*(1-t)*yM5+pow(t,5)*yM6;
        openGL(xt,yt,1.,0.,0.,2.);
      }
  glEndList();

  cout<<"\n Voila, c'est fini, la courbe a été tracée"<<endl;
}

// fonction permettant d'afficher les objets en utilisant des listes
void affichage()
{
  /* effacement de l'image avec la couleur de fond */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

  glTranslatef(0.0,0.0,dist);
 // En 2D
  glRotatef(-anglex+angley,0.0,0.0,1.0);
  glScalef(Scal,Scal,Scal); // diminution de la vue de la scene
  glRotatef(180,0.0,1.0,0.0);
  glRotatef(180,1.0,0.0,0.0);
  glTranslatef(-trX,trY,0.);
      glCallList(1); // appel de la liste numero 1
      glCallList(2); // appel de la liste numero 2
      glCallList(3); // appel de la liste numero 3
      glCallList(4); // appel de la liste numero 4
      glCallList(5); // appel de la liste numero 3
      glCallList(6); // appel de la liste numero 4
 glFlush();
  // On echange les buffers
  glutSwapBuffers();
}
```