Projet de Mathématiques

CHAPUS Louka

ESIREM DIJON

CHAPUS Louka

Groupe 3A IE

Table des matières

1. Introduction

Le but du projet est la construction d'un courbe reliant, de façon G^2, deux segments par une courbe définie par des points de contrôle. Il faut aussi prendre en compte le cas particulier de segments parallèles.

Pour ce faire, on va utiliser les courbes de Bézier, car elles permettent de générer des courbes à partir de quelques points (3 au minimum). On utilisera notamment les courbes de Bézier d'ordre 2 car elles permettent d'avoir des paraboles. De plus, une continuité G^2 est lorsque le centre de courbure est le même au point de jonction de deux segments, c'est aussi appelée continuité de courbure.

De plus, pour l'implémentation en C++ on aura besoin des bibliothèques suivantes :

```
#include <vector>
#include <array>
#include <assert.h>
#include <algorithm>                    //pour le std::sort
```

2. État de l'art

2.1 Définition générale des courbes de Bézier

On considère $n+1$ points du plan ($n \geq 1$, dans notre cas $n = 2$), $A_0, A_1, \ldots, A_n$ et on définit une courbe paramétrée $M(t), t \in [0,1]$, associée à ces points :

$$M(t) = B_0(t)A_0 + B_1(t)A_1 + \cdots + B_n(t)A_n$$

Il faut que les coefficients $B_i(t)$ soient ≥ 0, de somme 1 et qu'ils dépendent de t de manière la plus régulière possible. La solution adoptée par P. Bézier consiste à prendre les polynômes de Berstein.

Les polynômes de Berstein d'ordre n sont les polynômes :

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \qquad (1)$$

On vérifie aussitôt que les $B_{i,n}(t)$ sont ≥ 0 sur [0, 1] et que leur somme est bien égale à 1.

On peut donc réécrire l'équation de la courbe paramétrée $M(t), t \in [0,1]$, sous la forme :

$$M(t) = \sum_{i=0}^{n} B_{i,n}(t)\, A_n \qquad (2)$$

Avec (1) et (2), on obtient :

$$M(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} A_n \qquad (3)$$

Soient $A_0, A_1, \ldots, A_n$, $n+1$ points distincts du plan. La courbe de Bézier (d'ordre n) associée à ces points est la courbe paramétrée Γ définie par (3) pour $t \in [0,1]$, on a alors les propriétés suivantes :

0) La courbe Γ est une courbe C^∞.
1) On a $M(0) = A_0$ et $M(1) = A_n$.
2) La droite (A_0A_1) (resp. $A_{n-1}A_n$) est tangente à Γ en A_0 (resp. A_n).
3) La courbe Γ est dans l'enveloppe convexe des A_i.

2.2 Courbes de Bézier d'ordre 2

On s'intéresse plus particulièrement aux courbes de Bézier quadratique, car ce sont des courbes qui permettent de relier de façon G^2 deux segments de droite en définissant un point de contrôle au croisement des deux droites, le cas des droites parallèles est aussi à prendre en compte.

On considère trois points distincts A, B, C du plan. On pose $A(x_A, y_A)$, $B(x_B, y_B)$ et $C(x_C, y_C)$. On considère la courbe de Bézier Γ définie par :

$$M(t) = (1-t)^2 A + 2t(1-t)B + t^2 C \qquad (4)$$

Elle passe par A et C et elle est tangente respectivement à (AB) et (BC) en ces points. On suppose que A, B, C ne sont pas alignés, alors la courbe Γ est une parabole.

Démonstration. On a les formules suivantes :

$$x(t) = at^2 + bt + c \qquad et \qquad y(t) = a't^2 + b't + c$$

Avec $a = x_A - 2x_B + x_C, b = 2(x_B + x_A)$, $c = x_A$ et $a' = y_A - 2y_B + y_C, b' = 2(y_B + y_A)$, $c' = y_A$. On note que a et a' ne sont pas tous deux nuls (sinon B est le milieu de [AC]). On élimine les termes en t^2 entre ces équations :

$$a'x - ay = (a'b - ab')t + a'c - ac'$$

Si le coefficient $a'b - ab'$ est nul on obtient une droite, ce qui est absurde puisque les point A, B, C ne sont pas alignés. Sinon on a :

$$t = \frac{a'x - ay - a'c + ac}{a'b - ab'}$$

Si on désigne par X cette quantité (ce qui revient à faire un changement de coordonnées cartésiennes) l'équation de Γ s'écrit :

$$y = a'X^2 + b'X + c'$$

On a bien l'équation d'une parabole.

3. Contribution

3.1 Point de contrôle

Pour utiliser l'équation donnée pour les courbes de Bézier d'ordre 2, nous avons besoin de 3 points, deux seront pris parmi les 4 points renseignés par l'utilisateur et le troisième sera le point d'intersection des deux droites formées par les points, le cas des droites parallèles sera traité plus tard.

Point d'intersection entre les deux droites, considérons 4 points $A(x_A, y_A)$, $B(x_B, y_B)$, $C(x_C, y_C)$ et) $D(x_D, y_D)$. Considérons ensuite les segments [AB] et [CD].

Pour le segment [AB], équation de la droite : $y_1 = a_1x + b_1$ avec $a_1 = \frac{y_A - y_B}{x_A - x_B}$ et $b_1 = y_1 - a_1x_A$

Idem pour le segment [CD], équation de la droite : $y_2 = a_2x + b_2$ avec $a_2 = \frac{y_C - y_D}{x_C - x_D}$ et

$b_2 = y_2 - a_2x_C$.

On note $M(x_M, y_M)$ le point d'intersection des deux droites. On a :

$$y_1 = y_2$$

$$x_M = \frac{b_1 - b_2}{a_1 - a_2} \qquad a_1 - a_2 \neq 0 \; car \; droites \; séquentes$$

Et

$$y_M = a_1x_M + b_1 \qquad (ou \; y_M = a_2x_M + b_2)$$

Implémentation en C++ :

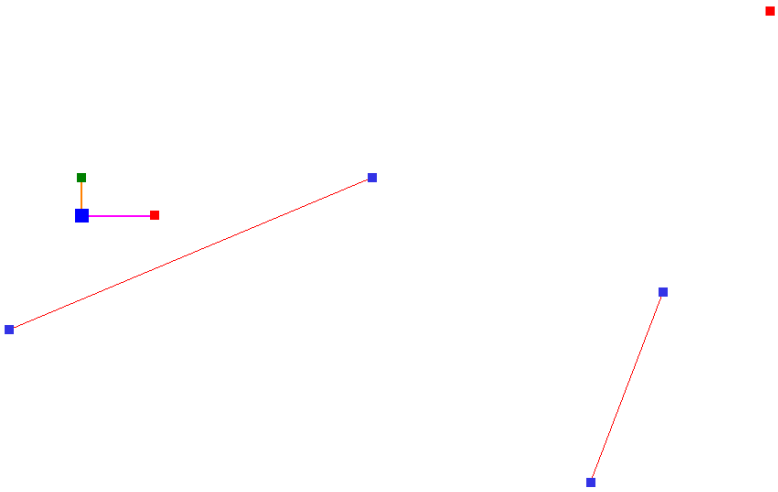
```
void point_controle(std::vector<float> x ,std::vector<float> y,float& x_M,float& y_M)
{
  assert (x.size()==y.size() && "Taille des listes différentes (point de contrôle)");

  float a1=0.0,a2=0.0,b1=0.0,b2=0.0;
  a1 = (y.at(0)-y.at(1))/(x.at(0)-x.at(1));   //coef directeur de la première droite
  a2 = (y.at(2)-y.at(3))/(x.at(2)-x.at(3));   //coef directeur de la deuxième droite
  b1 = y.at(0)-a1*x.at(0);                    //ordonné à l'origine de la première droite
  b2 = y.at(2)-a2*x.at(2);                    //ordonné à l'origine de la deuxième droite

  if((a1-a2)==0)                          //droites parallèles
  {
    droite_parallele(x,y,x_M,y_M);
  }
  else                                    //droites sécantes
  {
    x_M = (b2-b1)/(a1-a2);
    y_M = a1*x_M+b1;       //ou y_M = a2*x_M+b2  (même résultat)
  }
}
```

3.2 Distance entre deux points

On a besoin de la distance entre deux points pour pouvoir déterminer les deux autres points de contrôles et garder les points les plus éloignés du point de contrôle pour avoir une parabole qui maintient un rayon de courbure presque constant et donc avoir une jointure de façon G².



Tracer des segments et du point de contrôle en rouge

Dans l'image précédente on remarque bien la nécessité de connaître la distance entre les points et le point d'intersection pour pouvoir choisir les bons points a utilisés dans l'équation (4).

Considérons deux points $A(x_A, y_A)$ et $B(x_B, y_B)$, notons d la distance entre ces deux points, on a :

$$d = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

Implémentation en C++ :

```
float distance(float x_A,float y_A,float x_B,float y_B)
{
  return (pow(pow(x_A-x_B ,2) + pow(y_A-y_B ,2) , 0.5));
}
```

Ensuite on a juste à tester toutes les distances pour connaître les deux autres points a utilisé dans l'équation (4).

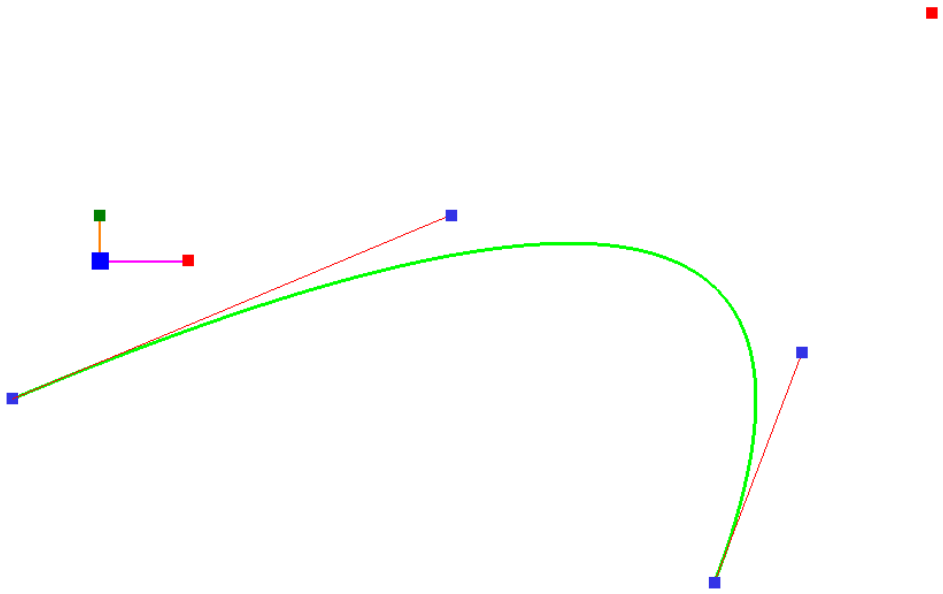
3.3 Tracer de la courbe

Maintenant que nous connaissons les trois points à utiliser, nous pouvons les utiliser dans l'équation (4) en faisant varier t entre 0 et 1 et à chaque fois en traçant un petit bout de droite en t et $t + dt$.

```
void trace_courbe(std::vector<float>& x ,std::vector<float>& y,float x_M,float y_M)
{
  assert (x.size()==y.size() && "Taille des listes différentes (tracer de la courbe)");

  std::array<float,3> x
  std::array<float,3> y
  if(distance(x.at(0),y
  {
    x0.at(0) = x.at(1);
    y0.at(0) = y.at(1);
  }
  else
  {
    x0.at(0) = x.at(0);
    y0.at(0) = y.at(0);
  }
  if(distance(x.at(2),y
  {
    x0.at(2) = x.at(3);
    y0.at(2) = y.at(3);
  }
  else
  {
    x0.at(2) = x.at(2);
    y0.at(2) = y.at(2);
  }

  double t = 0.0;
  float n = 1001;
  float x_tmp0=x0.at(0)
  for(int k=0;k<n;++k)
  {
    t=k/n;
    x_tmp1=(1-t)*(1-t)*
    y_tmp1=(1-t)*(1-t)*
    trace_segment(x_tmp
    x_tmp0=x_tmp1;
    y_tmp0=y_tmp1;
  }
}
```



Pour faire varier t entre 0 et 1, on utilise un entier n assez grand ($n \gg 100$) et une variable k entière qui varie de 0 à n. Alors $t = \frac{k}{n}$ est un nombre réel en 0 et 1.Exemple de tracer d'une courbe de Bézier avec deux droites sécantes :

3.4 Cas des segments parallèles

On détecte que les segments sont parallèles lorsqu'on essaye de calculer le point de contrôle :

$$a_1 - a_2 = 0 \quad (même\ pente)$$

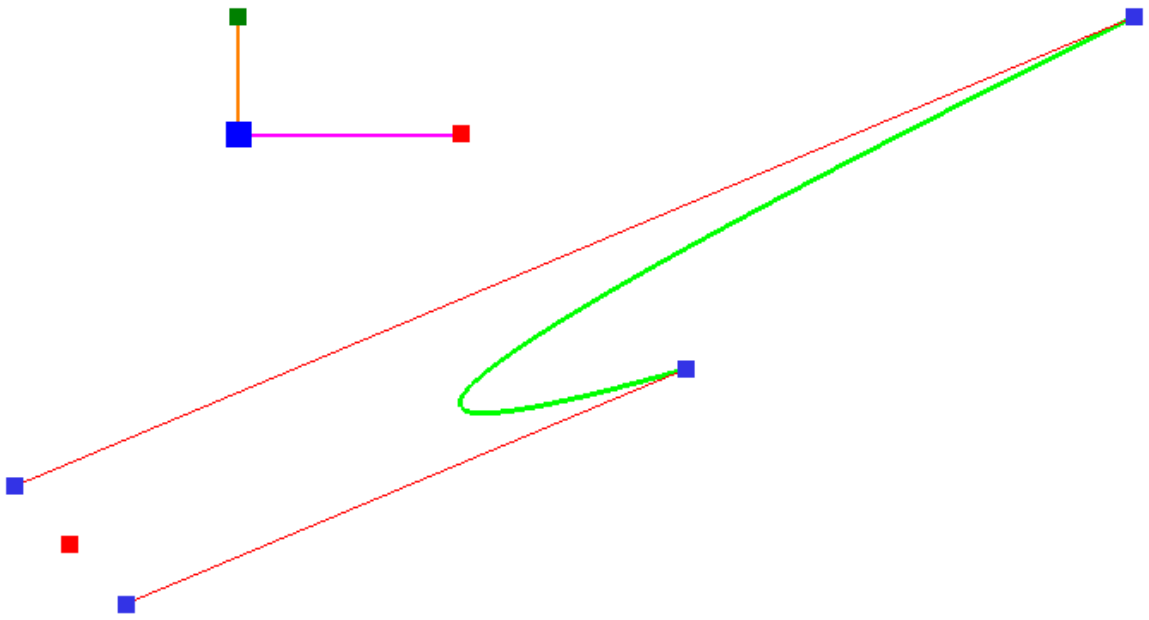
Lorsque les droites sont parallèles, il faut déterminer la distance la plus courte entre les quatre points pour pouvoir placer le point de contrôle entre les deux points les plus proches. Une fois qu'on a les deux points les plus proches, alors le point de contrôle sera simplement la moyenne des coordonnées des deux points. De plus, on considère que les deux premiers points forment un segment et que les deux derniers en forment un autre parallèle au premier (donc c'est inutile de calculer leurs distances).

Implémentation C++ :

```
void droite_parallele(std::vector<float> x ,std::vector<float> y,float& x_M,float& y_M)
{
  float d1 = distance(x.at(0),y.at(0),x.at(3),y.at(3));       //[AD]
  float d2 = distance(x.at(0),y.at(0),x.at(2),y.at(2));       //[AC]
  float d3 = distance(x.at(1),y.at(1),x.at(2),y.at(2));       //[BC]
  float d4 = distance(x.at(1),y.at(1),x.at(3),y.at(3));       //[BD]
  std::array<float,4> x0 = {d1,d2,d3,d4};
  std::sort(x0.begin(), x0.end());                            //trie dans l'ordre croissant les distances et ne conserve que la disatnce la plus petite
  if(x0.at(0)==d1)
  {
    x_M=(x.at(0)+x.at(3))/2;
    y_M=(y.at(0)+y.at(3))/2;
  }
  if(x0.at(0)==d2)
  {
    x_M=(x.at(0)+x.at(2))/2;
    y_M=(y.at(0)+y.at(2))/2;
  }
  if(x0.at(0)==d3)
  {
    x_M=(x.at(1)+x.at(2))/2;
    y_M=(y.at(1)+y.at(2))/2;
  }
  if(x0.at(0)==d4)
  {
    x_M=(x.at(1)+x.at(3))/2;
    y_M=(y.at(1)+y.at(3))/2;
  }
}
```

La fonction std::sort ordonne les valeurs contenues dans la liste de la plus petite à la plus grande, c'est ce que l'on veut pour trouver la plus petite distance.

Exemple de jointure avec des droites parallèles :



En rouge est le point de contrôle utilisé, les autres points étant renseigner par l'utilisateur.

3.5 Fonctions supplémentaires

On a besoin de quelque fonctions supplémentaires pour notamment tracer les points et les segments choisi par l'utilisateur. On va donc introduire la fonction suivante qui permet de faire ces tracer.

Implémentation C++

```
void trace_init(std::vector<float> x ,std::vector<float> y)
{
  assert (x.size()==y.size() && "Taille des listes différentes (tracer initiale)");
  openGL(x.at(0),y.at(0),0.2,0.2,0.90,10.);                          //permier point
  openGL(x.at(1),y.at(1),0.2,0.2,0.90,10.);                          //deuxième point
  trace_segment(x.at(0),y.at(0),x.at(1),y.at(1),1.0,0.,0.,0.5);
  int fin=x.size();
  openGL(x.at(fin-1),y.at(fin-1),0.2,0.2,0.90,10.);                  //avant dernier point (troisième dans notre cas)
  openGL(x.at(fin-2),y.at(fin-2),0.2,0.2,0.90,10.);                  //dernier point (quatrième dans notre cas)
  trace_segment(x.at(fin-1),y.at(fin-1),x.at(fin-2),y.at(fin-2),1.0,0.,0.,0.5);
}
```

Enfin, on introduit une dernière fonction qui va s'occuper de tout faire, c'est-à-dire tracer les tous les points (points de contrôle compris), tracer les segments qui relient les points et tracer la courbe grâce à l'équation (4).

On appelle cette fonction *trace_tout*.

Implémentation C++ :

```
void trace_tout(std::vector<float>& x ,std::vector<float>& y)
{
  trace_init(x,y);
  float x_M=0.0,y_M=0.0;              //point de contrôle
  point_controle(x,y,x_M,y_M);
  trace_courbe(x,y,x_M,y_M);
  openGL(x_M,y_M,1.,0.,0.,10.);       //tracer du point de contrôle
}
```

4. Conclusion

Donc, pour conclure, nous avons vu que les courbes de Bézier d'ordre 2 permettent de faire une jointure, de façon G^2, entre deux segments quelconque (parallèles ou sécants). De même, le point de contrôle est déterminé en fonction de la géométrie des segments et donc des droites. Une fois le point de contrôle déterminé, on peut savoir quels sont les deux autres points à utiliser en prenant à chaque fois le plus loin du point de contrôle. Ensuite, il suffit d'appliquer l'équation (4) pour pouvoir tracer notre courbe.

Dans le programme C++, il suffit de définir deux listes, une pour les x et une pour les y, puis d'appeler la fonction précédente avec comme paramètre les deux listes et on obtient la courbe, les segments et les points. Cela permet de pouvoir rapidement changer les points sans rien avoir à changer d'autres.

Bibliographie

-Cours GPA445 par Antoine Brière-Côté, https://slideplayer.fr/slide/1141576/

-Wikipédia, Courbe de Bézier, https://fr.wikipedia.org/wiki/Courbe_de_Bézier

-Cours sur les courbes de Bézier de Daniel Perrin