



Rémy Malgouyres,
Algorithmes pour la synthèse d'images
et l'animation 3D,
DUNOD, 2002

TP n° 5 : Courbes Hermitiennes et de Bézier

Objectifs :

Le but de ce TP est de compléter la hiérarchie de classes vue au TP1. On implémentera ici des classes représentant les courbes Hermitiennes, courbes de Bézier cubiques, et courbes de Bézier d'ordre n .

1 La classe Hermitienne

On trouvera dans les fichiers fournis *Matrice.h* et *Matrice.cpp* la classe `Matrice`, la classe `Vecteur4D` (vecteurs verticaux), et la classe `Quadruplet` qui seront utilisées pour le calcul de points de courbes cubiques.

On implémentera la classe `Hermitienne` correspondant aux courbes hermitiennes. Les paramètres du constructeur sont les deux points de contrôle et les deux vecteurs dérivés aux points de contrôle définissant une courbe Hermitienne.

```
class Hermitienne : public Courbe2D{
protected:
    Pixel P1, P4, R1, R4;
    Vecteur4D coefs_x, coefs_y;
public:
    Hermitienne(Pixel p1, Pixel p4, Pixel r1, Pixel r4);
    void getpoint(double t, double & xpos, double & ypos);
};
```

Les données `coefs_x` et `coefs_y`, qui doivent être initialisées dans le constructeur de la classe, correspondent au précalcul de la matrice $M_H.G_H$, produit de la matrice Hermitienne et de la matrice géométrique de la courbe. On utilisera ensuite ces données dans la fonction `getpoint`.

2 La classe BézierCubique

On implémentera ensuite une classe `BézierCubique`, comprenant comme données 4 points de contrôle. Le calcul des points de la courbe se fera de manière analogue au cas des courbes Hermitiennes, par un calcul matriciel.

```

class BezierCubique{
protected:
    Pixel P1, P2, P3, P4;
    Vecteur4D coefs_x, coefs_y;
public:
    BezierCubique(Pixel p1, Pixel p2, Pixel p3, Pixel p4);

    void getpoint(double t, double & xpos, double & ypos);
};

```

3 La classe BezierOrdreN

On implémentera enfin la classe `BezierOrdreN`, correspondant aux courbes de Bézier d'ordre n , pouvant prendre en compte un nombre quelconque de points de contrôle, lesquels seront stockés dans le tableau `ctrlpoints`.

```

class BezierOrdreN : public Courbe2D{
protected:
    Pixel *ctrlpoints;
    int nbctrlpoints;
public:
    BezierOrdreN();
    BezierOrdreN(Pixel *controlpoints, int nbcontrolpoints);
    virtual ~BezierOrdreN();

    void getpoint(double t, double & xpos, double & ypos);
};

```

On utilisera pour cela l'algorithme de Casteljau généralisé.

4 Facultatif : Obtention des courbes de Bézier par les polynômes de Bernstein

On définira et on implémentera une classe `BezierBernstein`, où le calcul de points $Q(t)$ se fera à partir du calcul des polynômes de Bernstein :

$$Q(t) = \sum_{i=0}^{n-1} P_i B_{i,n-1}(t)$$

et les polynômes de Bernstein seront calculés par récurrence en utilisant la formule :

$$B_{i,n}(t) = (1-t)B_{i,n-1} + tB_{i-1,n-1}$$