

Tache7_A

Pascal Isak && Weber Loïc

Table Of Content

- [Explication ajouts et modification code pour tache7 et source code.](#)
- [Code source programme de tests unitaires.](#)
- [Tableau contenant les images EPS \(mode remplissage\).](#)

Explication ajouts et modification code pour tache7 et source code.

Pour cette tache nous avons choisi de refaire toute notre implémentation des séquences pour la rendre compatible avec notre façon de stocker les courbes de bézier.

Nous avons ainsi créé le paquetage `sequence_bezier.c` dans lequel nous avons inventé deux nouvelles structures (*tres similaires au Contour et Liste de Contour*) :

- **Motif_Bezier2**

Un **Motif** est une liste chaînée de courbe de bezier.

Chaque cellule d'un Motif est une `Cellule_Motif_Bezier2` qui a une valeur (le pointeur vers une courbe de bezier) et d'un suivant (pointeur vers la cellule suivante).

- **Dessin_Bezier2**

Un **Dessin** est une liste chaînée de **Motif**.

Chaque cellule d'un Dessin est une `Cellule_Dessin_Bezier2` qui a une valeur (pointeur vers un Motif) et d'un suivant (pointeur vers la cellule suivante).

`sequence_bezier2.c`

```
typedef struct Cellule_Motif_Bezier2_{
    Bezier2 val;
    struct Cellule_Motif_Bezier2_* suiv;
} Cellule_Motif_Bezier2;

typedef struct Motif_Bezier2_{
    unsigned int taille;
    Cellule_Motif_Bezier2 *first;
    Cellule_Motif_Bezier2 *last;
} Motif_Bezier2;

typedef struct Cellule_Dessin_Bezier2_{
    Motif_Bezier2 val;
    struct Cellule_Dessin_Bezier2_* suiv;
} Cellule_Dessin_Bezier2;

typedef struct Dessin_Bezier2_{
```

```

    unsigned int taille;
    Cellule_Dessin_Bezier2 *first;
    Cellule_Dessin_Bezier2 *last;
} Dessin_Bezier2;

```

Nous avons également fait toutes les fonctions en rapport avec la simplification par courbe de bezier dans le fichier bezier.c

bezier.h

```

////////////////////////////////////
//                               //
////////////////////////////////////

typedef struct {
    Point C0;
    Point C1;
    Point C2;
} Bezier2;

typedef struct {
    Point C0;
    Point C1;
    Point C2;
    Point C3;
} Bezier3;

// Permet de rapidement créer une structure Bezier avec les points.
Bezier2 init_bezier2(Point C0, Point C1, Point C2);
Bezier3 init_bezier3(Point C0, Point C1, Point C2, Point C3);

/*
Permet de transformer un segment en courbe de bezier 2,
puisque nous utilisons des liste chaînées,
on donne la cellule qui correspond au premier point du segment et au dernier,
ainsi que le nombre de points pour l'optimisation.
*/
Bezier2 approx_bezier2(Cellule_Point* depart, Cellule_Point* fin, int nombre_cellule);

// Calcul de la distance entre le point et la courbe
// (ti represente le numéro du point / nombre total de points du contour simplifiée)
double distance_point_courbe_bezier2(Point C, Bezier2 bez, double ti);

// Prend un bezier2 et renvoie un bezier3 grace a la transformation de l'elevation de degré.
Bezier3 elevation_de_deg_bezier(Bezier2 bez);

// Applique la fonction, avec x un point entre 0 et 1.
Point applique_bezier2_point(Bezier2 bez, double x);

// Applique la fonction, avec x un point entre 0 et 1.
Point applique_bezier3_point(Bezier3 bez, double x);

void print_Bezier2(Bezier2 b);

```

Et enfin dans EPS.C nous avons ajouter deux fonctions pour dessiner nos dessins simplifier par courbe de bézier

```

// Prend un fichier, un Motif et la hauteur de l'image,
// et ecris dans le fichier donnée le motif entier.
void dessiner_motif_Bezier2(FILE* f, Motif_Bezier2 *Mot_bez2, int hauteur_img);
// Prend un dessin, un nom de fichier,
// une largeur et hauteur et créer le fichier EPS correspondant au dessin.
void enregistrer_dessin_Bezier2_vers_EPS(
    Dessin_Bezier2 *des_bez2,
    char* nom_fichier,
    int largeur_img,
    int hauteur_img
);

```

Code source programme de tests unitaires.

```

// Fonction pour savoir si un float est égal a un autre a 0.0001 pres.
// On prie pour que ça marche, car souvent les opération avec des flottant casse tout.
int environ_egal(double val_calcule, double val_reele){
    return (val_calcule > val_reele - 0.0001 && val_calcule < val_reele + 0.0001);
}

void test_distance_point_bezier2(){
    // On va verifier que la fonction donne les meme valeurs que dans l'exemple page 51.
    printf("Fonction distance_point_bezier2()\n");

    printf("Test 1/2\n");

    printf("On test sur l'exemple page 51\n");
    Bezier2 bez = init_bezier2(nouveau_point(0,0),nouveau_point(1.547619,2.452381),nouveau_point

    // Les points sur lesquels on va tester la fonction distance_point_courbe_bezier2
    Point tab[] = {nouveau_point(0,0),nouveau_point(1,0),nouveau_point(1,1),nouveau_point(1,2),
        nouveau_point(2,2),nouveau_point(3,2),nouveau_point(3,3),nouveau_point(4,3),
        nouveau_point(5,3)};

    // Les distances que l'on doit obtenir selon l'exemple page 51.
    double tab_res[] =
    {0.0,0.824958,0.151523,0.606091,0.033672,0.454569,0.555584,0.235702,0.0};

    for (int i = 0; i < 9; i++){
        double distance = distance_point_courbe_bezier2(tab[i], bez, i/8.0);
        printf("Distance : %lf ",distance);
        afficher_resultat_test(envIRON_egal(distance,tab_res[i]));
    }
}

void test_approx_bezier2(){
    // On va verifier que la fonction donne les meme valeurs que dans l'exemple page 50.
    printf("Fonction approx_bezier2()\n");
    printf("Test 2/2\n");

    Liste_Point LP = creer_liste_Point();
    ajouter_element_liste_Point(&LP, nouveau_point(0,0));
    ajouter_element_liste_Point(&LP, nouveau_point(1,0));

```

```

ajouter_element_liste_Point(&LP, nouveau_point(1,0));
ajouter_element_liste_Point(&LP, nouveau_point(1,1));
ajouter_element_liste_Point(&LP, nouveau_point(1,2));
ajouter_element_liste_Point(&LP, nouveau_point(2,2));
ajouter_element_liste_Point(&LP, nouveau_point(3,2));
ajouter_element_liste_Point(&LP, nouveau_point(3,3));
ajouter_element_liste_Point(&LP, nouveau_point(4,3));
ajouter_element_liste_Point(&LP, nouveau_point(5,3));

Bezier2 bez = approx_bezier2(LP.first, LP.last, LP.taille-1);











print_Bezier2(bez);

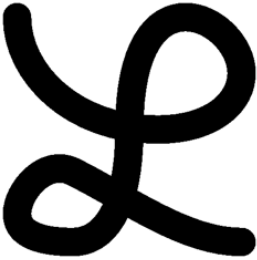
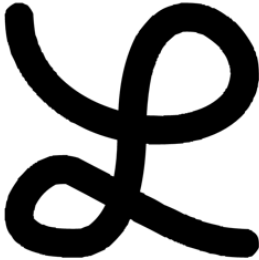
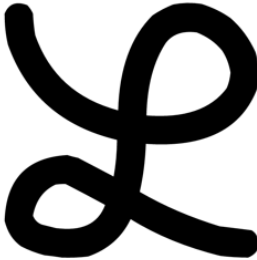
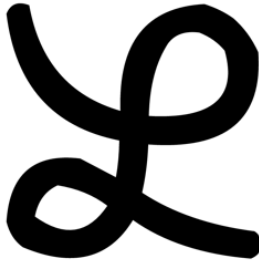

printf("\nC0 : (%lf.%lf)\n",bez.C0.x, bez.C0.y);
afficher_resultat_test(environ_egal(bez.C0.x,0.0) && (environ_egal(bez.C0.y, 0.0)));
printf("C1 : (%lf.%lf)\n",bez.C1.x, bez.C1.y);
afficher_resultat_test(environ_egal(bez.C1.x,1.547619) && (environ_egal(bez.C1.y, 2.452381)));
printf("C2 : (%lf.%lf)\n",bez.C2.x, bez.C2.y);
afficher_resultat_test(environ_egal(bez.C2.x,5.0) && (environ_egal(bez.C2.y, 3.0)));
printf("\n");
}

int main(int argc, char** argv){
    test_distance_point_bezier2();
    test_approx_bezier2();
    return 0;
}

```

Tableau contenant les images EPS (mode remplissage).

				
Image initiale (32 contours) 12958 segments	Simplification pour d = 1 969 Bézier	Simplification pour d = 3 296 Bézier	Simplification pour d = 10 159 Bézier	Simplification pour d = 30 72 Bézier
				
Image initiale (106 contours) 21870 segments	Simplification pour d = 1 1579 Bézier	Simplification pour d = 3 579 Bézier	Simplification pour d = 10 289 Bézier	Simplification pour d = 30 153 Bézier

				
Image initiale (3 contours) 4231 segments	Simplification pour d = 1 259 Bézier	Simplification pour d = 3 41 Bézier	Simplification pour d = 10 24 Bézier	Simplification pour d = 30 16 Bézier