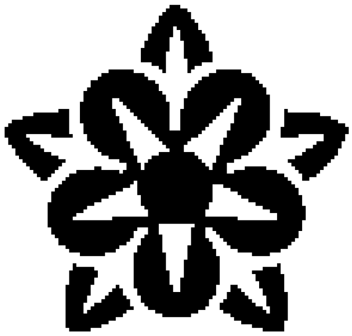


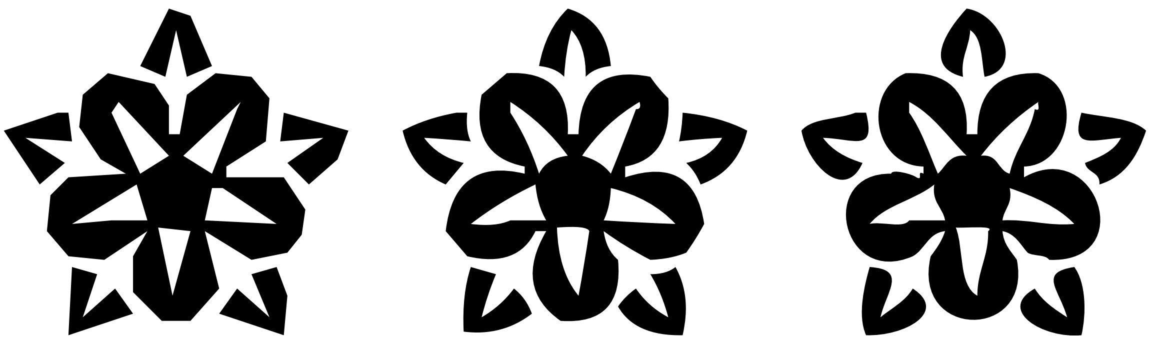
Licence 2 – Math–Info – Sciences&Design

MAP401 – Projet logiciel

Vectorisation et simplification d'image bitmap

DLST – Université Grenoble Alpes

2023–2024



Présentation

1 - Vectorisation d'image bitmap

Extraction de contour(s) (vectorisation) d'image bitmap Noir & Blanc

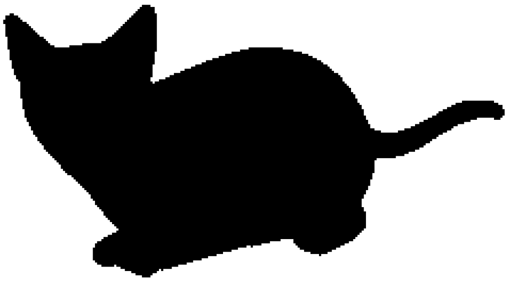
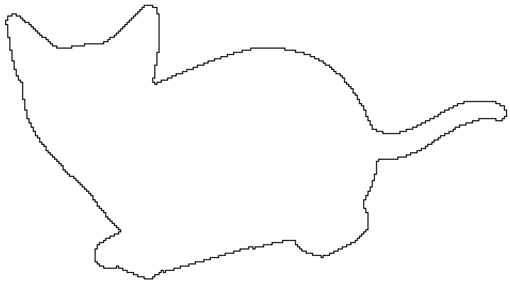
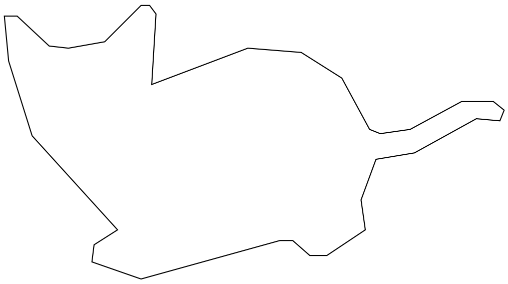


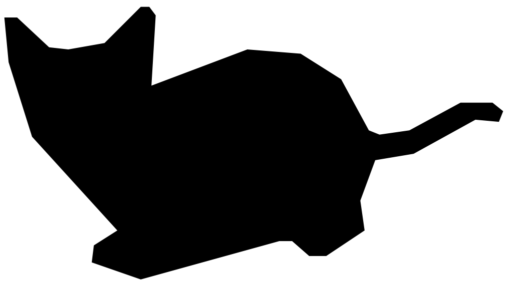
Image bitmap de dimensions 250×140
(soit 35000 pixels)



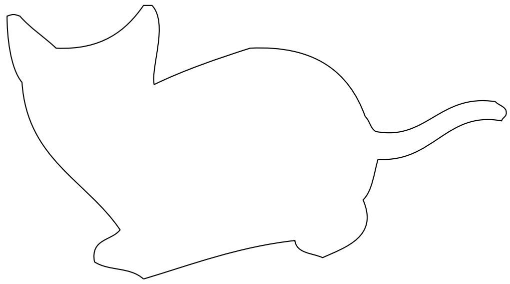
Contour vectorisé composé
de 881 points ou 880 segments

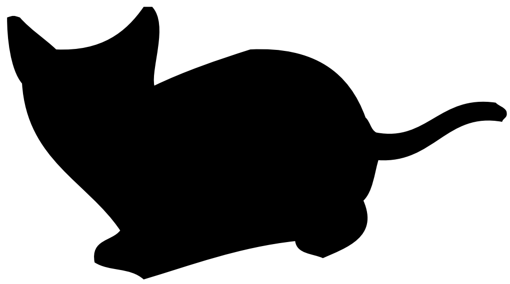
Simplification à l'aide de primitives géométriques





Contour simplifié
avec 34 segments





Contour simplifié
avec 20 courbes de Bézier cubiques

• Exemples d'applications :
– détection de contours (analyse d'image)
– compression de données géométriques, représentation de contours à différentes échelles (cartographie - systèmes d'information géographique)

Un contour simplifié peut suffire visuellement pour un affichage à une petite échelle :

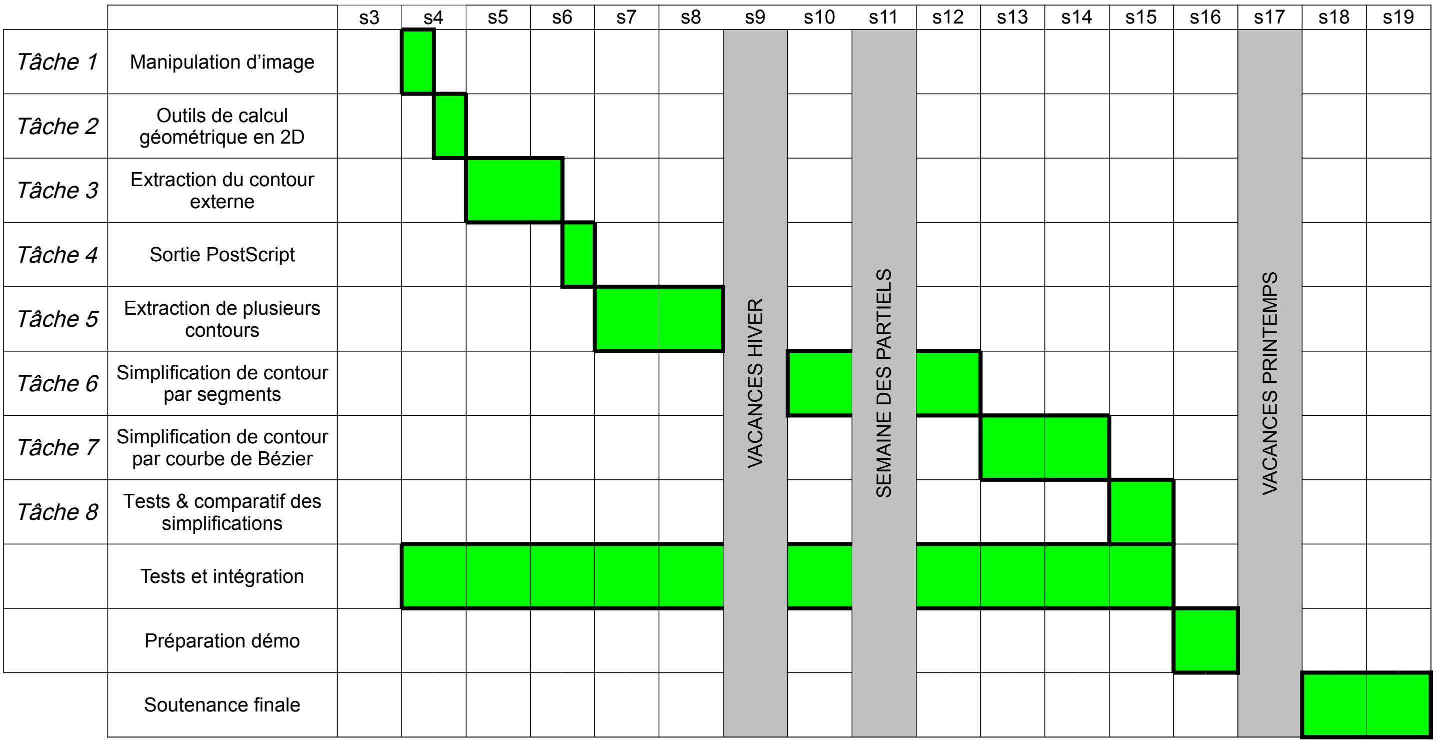
Image initiale	Image avec simplification par segments	Image avec simplification par courbes de Bézier cubiques
		

Affichage à l'échelle 1/4

2 - Déroulement

Programmation en langage C - Gestion de projet : cahier des charges avec calendrier prévisionnel

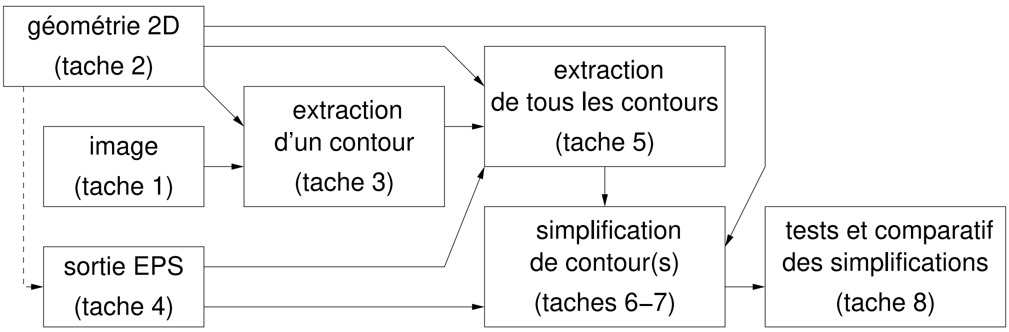
Diagramme de Gantt prévisionnel



→ découpage du projet en étapes (tâches) permettant une avancée progressive et une bonne gestion de l'écriture du code

→ phases de test, validation, intégration

Dépendances des différentes tâches entre elles



Documents à rendre :
– chaque semaine, le document *Suivi de projet* complété,
– à la fin de chaque tâche ou chaque partie de tâche, un compte rendu,
– un document de synthèse à la fin pour la soutenance orale finale.

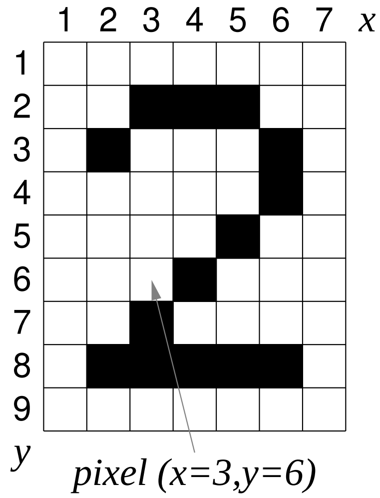
Modalités du Contrôle de Connaissances et des Compétences : **C**ontrôle **C**ontinu **I**ntégral
– **CC1** (coef 0,4) : partiel sur courbes de bézier + début du projet (écrit 2h)
– **CC2** (coef 0,5) : suivi de projet hebdomadaire + comptes rendus des tâches + présence/participation en séance machine.
– **CC3** (coef 0,1) : soutenance orale finale (15 min.).
Seconde chance uniquement sur CC1.

Tâche 1 - Image Bitmap

Module `image` permettant la création et la manipulation d'images *bitmap Noir & Blanc* à partir desquelles les contours seront extraits (tâches 3 et 5) puis simplifiées (tâches 6 et 7).

1.1 - Image bitmap Noir & Blanc

Une image *bitmap Noir & Blanc* est une grille rectangulaire, dont chaque élément de base (*pixel*) est un carré, soit noir, soit blanc.
L'exemple ci-contre image de dimensions 7×9, la première dimension est la *largeur* (notée L) et la deuxième la *hauteur* (notée H).
Dans l'image, un pixel est repéré par un couple (x, y) de coordonnées entières (on répère un pixel par *abscisse-ordonnée* et non pas *ligne-colonne*).
Dans l'image, l'indiçage des abscisses x va de 1 à L (de gauche à droite), et l'indiçage des ordonnées y va de 1 à H (de haut en bas).
L'extérieur de l'image est considéré comme étant formé de pixels blancs, par exemple, pour l'image ci-contre, les positions $(0, 3)$ ou $(4, 10)$ sont à l'extérieur de l'image, et correspondant à des pixels blancs.



1.2 - Format de fichier PBM texte

En général, les images utilisées dans les tâches suivantes seront créées à partir de fichier dont le format est décrit ci-dessous.

Fichier au format *PBM texte* : fichier au format texte formé de deux parties.

1. En-tête :
   — une première ligne obligatoire : `P1`↩
   — une ou plusieurs lignes facultatives de commentaire (chaque ligne commençant par le caractère *dièse* `#`)
2. Image : sur une seule ligne ou plusieurs lignes, la dimension L puis la dimension H puis une suite de caractères (`'0'` ou `'1'`) correspondant aux pixels de l'image, ligne après ligne de haut en bas, chaque ligne décrite de gauche à droite
   . le caractère `'0'` correspond à un pixel blanc,
   . le caractère `'1'` correspond à un pixel noir,
   (convention) tout caractère autre que `'0'` ou `'1'` n'est pas pris en compte.

Il faut au moins un séparateur entre L et H, et au moins un séparateur entre H et les pixels de l'image.

• **Exemple** : Pour l'image ci-dessus, voici trois possibilités différentes pour le fichier au format *PBM texte* :
(symboles des séparateurs : ↩ *fin de ligne* , ␣ *espace* , ↦ *tabulation*)

```
P1↩
7␣9↩
0000000↩
0011100↩
0100010↩
0000010↩
0000100↩
0001000↩
0010000↩
0111110↩
0000000↩
```

```
P1↩
# pixels séparés par des espaces ↩
␣7␣↦9␣↩
↦0␣0␣0␣0␣0␣0␣0↩
↦0␣0␣1␣1␣1␣0␣0↩
↦0␣1␣0␣0␣0␣1␣0↩
↦0␣0␣0␣0␣0␣1␣0↩
↦0␣0␣0␣0␣1␣0␣0↩
↦0␣0␣0␣1␣0␣0␣0↩
↦0␣0␣1␣0␣0␣0␣0↩
↦0␣1␣1␣1␣1␣1␣0↩
↦0␣0␣0␣0␣0␣0␣0↩
```

```
P1↩
# dimensions puis les pixels à la suite↩
7␣␣␣9␣␣␣000000000111000100010000001000001000001000001000001111100000000↩
```

1.3 - Module de manipulation d'image

1.3.1 - Type pixel

Avoir un type permettant la représentation des valeurs 0 et 1 .
Pour l'implémentation du type *pixel*, on propose la solution suivante (fichier *image.h*) :

```
/* type enuméré Pixel avec BLANC=0 et NOIR=1 */
typedef enum {BLANC=0,NOIR=1} Pixel;
```

1.3.2 - Type Image et fonctions associées

Avoir un type *abstrait*, et des fonctions permettant les opérations suivantes :

- création d'une image
  - — à partir de deux dimensions L et H données (image créée avec tous ses pixels blancs)
  - — à partir de la lecture d'un fichier *PBM texte*
- accès à l'image
  - — pour récupérer sa largeur L
  - — pour récupérer sa hauteur H
  - — pour récupérer la valeur d'un pixel à partir de ces coordonnées entières (x, y) ; si (x, y) correspond à une position à l'extérieur de l'image, on renvoie la valeur `BLANC`
- modification de l'image
  - — pour modifier la valeur d'un pixel à partir de ces coordonnées entières (x, y) ; si (x, y) correspond à une position à l'extérieur de l'image, on ne fait rien
- suppression de l'image

Cette liste n'est pas exhaustive, et on pourra rajouter d'autres opérations si nécessaire.

L'important n'est pas la manière dont est implémenté le type `Image` mais les différentes opérations de création, accés / modification et suppression d'une image.

L'implémentation du type *image* ainsi que certaines fonctions sont déjà fournies.

Dans les codes en langage C qui suivent, une partie cachée ou non importante est indiquée par $\boxed{\texttt{...}}$

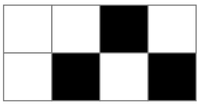
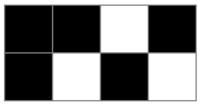
Dans le fichier `types_macros.h` :

```
...

typedef unsigned int UINT; /* UINT = entier non signé */

...
```

Dans le fichier `image.h` :

```
...

#include "types_macros.h"

...

/* Type Image */
typedef ... Image;

/* création d'une image PBM de dimensions L x H avec tous les pixels blancs */
Image creer_image(UINT L, UINT H);

/* suppression de l'image I = *p_I */
void supprimer_image(Image *p_I);
```

```
/* renvoie la valeur du pixel (x,y) de l'image I
   si (x,y) est hors de l'image la fonction renvoie BLANC */
Pixel get_pixel_image(Image I, int x, int y);

/* change la valeur du pixel (x,y) de l'image I avec la valeur v
   si (x,y) est hors de l'image la fonction ne fait rien */
void set_pixel_image(Image I, int x, int y, Pixel v);

/* renvoie la largeur de l'image I */
UINT largeur_image(Image I);

/* renvoie la hauteur de l'image I */
UINT hauteur_image(Image I);

/* lire l'image dans le fichier nommé nom_f
   s'il y a une erreur dans le fichier le programme s'arrete en affichant
   un message */
Image lire_fichier_image(char *nom_f);

/* écrire l'image I à l'écran */
void ecrire_image(Image I);            /* IMPLEMENTATION A COMPLETER */

/* calculer l'image "négatif" de l'image I */
Image negatif_image(Image I);          /* IMPLEMENTATION A COMPLETER */

...
```

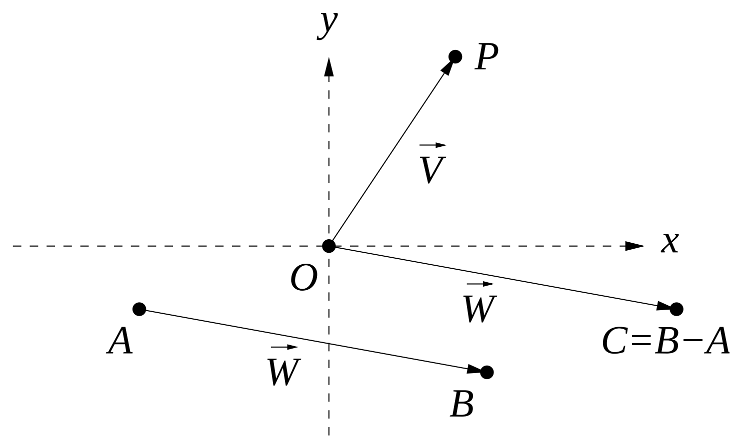
• **Exemple** : programme qui crée l'image `A`  puis calcule l'image `Aneg`  , négatif de l'image `A`.

```
#include"image_pbm.h"
...
Image A, Aneg;
A = creer_image(4,2);             /* créer l'image avec tous les pixels blancs */
set_pixel_image(A, 3,1 , NOIR);   /* noircir le pixel de coordonnées (3,1) */
set_pixel_image(A, 2,2 , NOIR);   /* noircir le pixel de coordonnées (2,2) */
set_pixel_image(A, 4,2 , NOIR);   /* noircir le pixel de coordonnées (4,2) */
ecrire_image(A);                  /* afficher l'image A à l'écran */
Aneg = negatif_image(A);          /* calculer Aneg, négatif de l'image A */
ecrire_image(Aneg);               /* afficher l'image Aneg à l'écran */
supprimer_image(&A);              /* supprimer l'image A */
supprimer_image(&Aneg);           /* supprimer l'image Aneg */
...
```

Tâche 2 - Géométrie 2D

Création d'un module pour manipuler des vecteurs, points, segments de droites, ...
— définitions de types,
— définitions d'opérations géométriques de base dans le plan en dimension 2

2.1 - Rappel de géométrie dans le plan $\mathbb{R}^2$



$$\text{Point } P = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2 \qquad \text{Vecteur } \overrightarrow{V} = \begin{pmatrix} x \\ y \end{pmatrix} = \overrightarrow{OP} \equiv \text{point } P = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\text{Vecteur bi-point } \overrightarrow{W} = \overrightarrow{AB} = \overrightarrow{OC} \equiv \text{point } C = B - A$$

$$\text{Somme de deux vecteurs } \overrightarrow{V_1} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \text{ et } \overrightarrow{V_2} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \quad : \quad \overrightarrow{V_1} + \overrightarrow{V_2} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}$$

$$\text{Somme de deux points } P_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \text{ et } P_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \quad : \quad P_1 + P_2 = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}$$

$$\text{Produit du réel } a \text{ par le vecteur } \overrightarrow{V} = \begin{pmatrix} x \\ y \end{pmatrix} \quad : \quad a\,\overrightarrow{V} = \begin{pmatrix} a\,x \\ a\,y \end{pmatrix}$$

$$\text{Produit du réel } a \text{ par le point } P = \begin{pmatrix} x \\ y \end{pmatrix} \quad : \quad a\,P = \begin{pmatrix} a\,x \\ a\,y \end{pmatrix}$$

$$\text{Produit scalaire entre deux vecteurs } \overrightarrow{V_1} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \text{ et } \overrightarrow{V_2} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} :$$

$$\langle \overrightarrow{V_1}, \overrightarrow{V_2} \rangle = \overrightarrow{V_1} \cdot \overrightarrow{V_2} = \langle \overrightarrow{V_2}, \overrightarrow{V_1} \rangle = \overrightarrow{V_2} \cdot \overrightarrow{V_1} = x_1\,x_2 + y_1\,y_2$$

$$\text{propriétés : } \langle \overrightarrow{U_1} + \overrightarrow{U_2}, \overrightarrow{V} \rangle = \langle \overrightarrow{U_1}, \overrightarrow{V} \rangle + \langle \overrightarrow{U_2}, \overrightarrow{V} \rangle \text{ et } \langle a\,\overrightarrow{U}, \overrightarrow{V} \rangle = a\,\langle \overrightarrow{U}, \overrightarrow{V} \rangle$$

$$\text{Norme euclidienne (longueur) du vecteur } \overrightarrow{V} = \begin{pmatrix} x \\ y \end{pmatrix} :$$

$$\|\overrightarrow{V}\| = \sqrt{\overrightarrow{V} \cdot \overrightarrow{V}} = \sqrt{x^2 + y^2}$$

$$\text{Distance entre deux points } A = \begin{pmatrix} x_A \\ y_A \end{pmatrix} \text{ et } B = \begin{pmatrix} x_B \\ y_B \end{pmatrix} :$$

$$d(A, B) = \|\overrightarrow{AB}\| = \|\overrightarrow{BA}\| = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

2.2 - Les types

• **Exemple** (déclaration des types `Point` et `Vecteur`) :

```
typedef struct Vecteur_ {
    double x,y; /* coordonnees */
} Vecteur;

typedef struct Point_ {
    double x,y; /* coordonnees */
} Point;
```

IMPORTANT : les coordonnées des points/vecteurs doivent être déclarées avec le type réel `double` (précision de 15 chiffres décimaux) afin d'avoir assez de précision pour les calculs lors des tâches de simplification.

2.3 - Les opérations

Définition d'opérations pour les types `Vecteur` et `Point`.

• **Exemple** : une fonction créant un point, une fonction effectuant la somme de 2 points et une fonction créant un vecteur bi-point à partir de deux points.

→ dans le fichier `geom2d.h` :

```
/* cree le point de coordonnées (x,y) */
Point set_point(double x, double y);

/* somme P1+P2 */
Point add_point(Point P1, Point P2);

/* vecteur correspondant au bipoint AB */
Vecteur vect_bipoint(Point A, Point B);
```

→ dans le fichier `geom2d.c` :

```
Point set_point(double x, double y)
{
    Point P = {x,y};
    return P;
}

Point add_point(Point P1, Point P2)
{
    return set_point(P1.x+P2.x,P1.y+P2.y);
}

Vecteur vect_bipoint(Point A, Point B)
{
    Vecteur V = {B.x-A.x,B.y-A.y};
    return V;
}
```

→ exemple d'utilisation

```
#include"geom2d.h"
...
Point A, B, C; Vecteur U;
A = set_point(1.0,-3.0);
B = set_point(4.0, 1.0);
C = add_point(A,B);       /* --> C = ( 5.0,-2.0) */
U = vect_bipoint(C,A);   /* --> U = (-4.0,-1.0) */
```

• En TP, il faudra implémenter (au minimum) les opérations géométriques présentées en page 7.

Tâche 3 - Extraction d'un contour d'une image

3.1 - Présentation

Pour une image Noir & Blanc, extraire le contour polygonal entre la partie noire et la partie blanche de l'image (la zone hors de l'image est considérée comme blanche).

Exemple

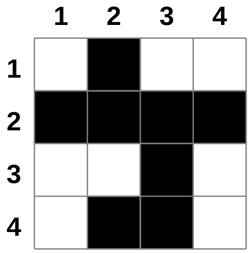


Image
coordonnées pixel

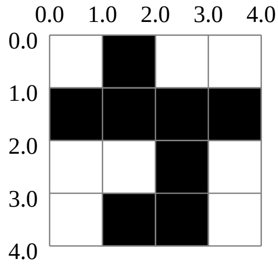
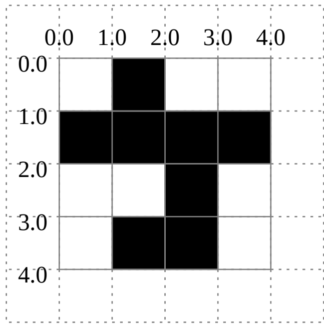
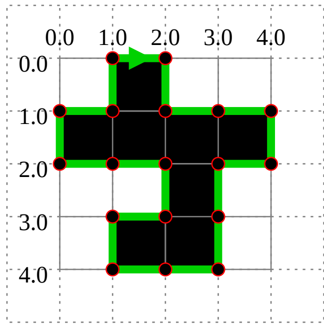


Image
coordonnées point



Partie "hors image"
blanche



Le contour

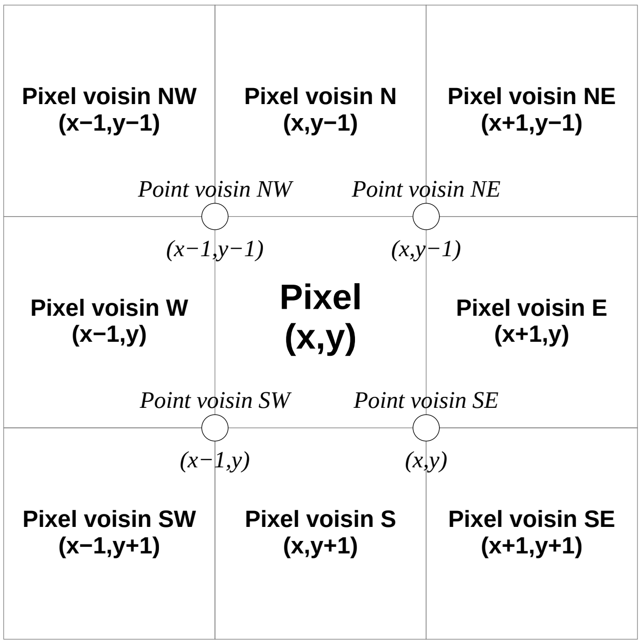
Le contour en partant du point $(1.0, 0.0)$ est le polygone suivant formé de 19 points (soit 18 segments) :

$$\{ \quad (1.0, 0.0), (2.0, 0.0), (2.0, 1.0), (3.0, 1.0), (4.0, 1.0), (4.0, 2.0), (3.0, 2.0), (3.0, 3.0), (3.0, 4.0), (2.0, 4.0), (1.0, 4.0), (1.0, 3.0), (2.0, 3.0), (2.0, 2.0), (1.0, 2.0), (0.0, 2.0), (0.0, 1.0), (1.0, 1.0), (1.0, 0.0) \quad \}$$

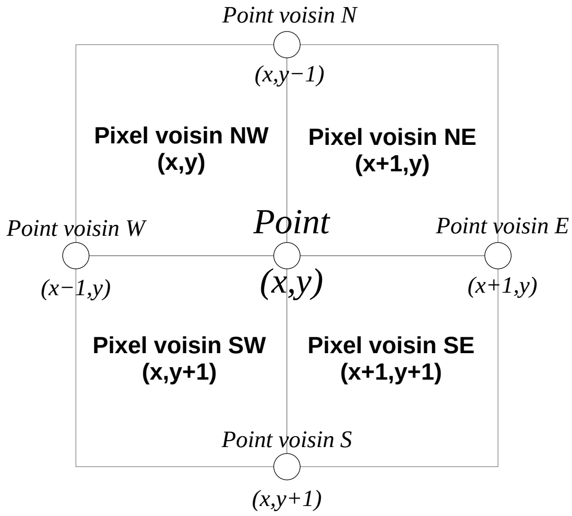
Le polygone est formé d'une suite de points, le premier et le dernier sont identiques (polygone fermé), deux points consécutifs sont à distance 1 l'un de l'autre, un segment relie deux points consécutifs.

Conventions utilisées :
- les lettres en gras $\mathbf{x}$,$\mathbf{y}$ sont utilisées pour les coordonnées entières des pixels dans l'image, et les lettres en italique x, y sont utilisées pour les coordonnées des points du plan,
- le pixel $(\mathbf{x}, \mathbf{y})$ de l'image correspond au carré $[x - 1, x]$ en abscisse et $[y - 1, y]$ en ordonnée.
- l'axe des ordonnées est orienté de haut en bas.
- l'orientation des voisins d'un point (x, y) ou d'un pixel $(\mathbf{x}, \mathbf{y})$ sera décrite en utilisant les points cardinaux

$$\begin{array}{rcl}
\mathsf{NW}(\text{Nord-Ouest}) & \mathsf{N}(\text{Nord}) & \mathsf{NE}(\text{Nord-Est}) \\
 & \nwarrow \ \uparrow \ \nearrow & \\
\mathsf{W}(\text{Ouest}) & \leftarrow \ \bullet \ \rightarrow & \mathsf{E}(\text{Est}) \\
 & \swarrow \ \downarrow \ \searrow & \\
\mathsf{SW}(\text{Sud-Ouest}) & \mathsf{S}(\text{Sud}) & \mathsf{SE}(\text{Sud-Est})
\end{array}$$



Voisins d'un pixel $(\mathbf{x}, \mathbf{y})$



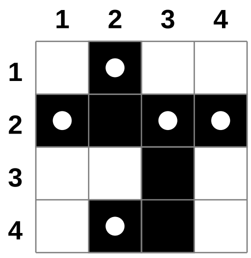
Voisins d'un point (x, y)

3.2 - Détermination du contour

La détermination du contour consiste à placer un "robot" sur le contour polygonal séparant les pixels noirs de l'image des pixels blancs, puis de faire avancer ce "robot" pour qu'il ait toujours un pixel blanc sur sa gauche et un pixel noir sur sa droite, jusqu'à revenir à sa position ET son orientation initiales.
Le robot peut avoir 4 orientations possibles : N (Nord), E (Est), S (Sud) et W (Ouest). On pourra définir un type enuméré en C :

```
typedef enum {Nord,Est,Sud,Ouest} Orientation;
```

A) Trouver le premier point du contour, i.e trouver la position initiale du robot avec une orientation initiale E : il faut trouver un pixel noir avec le pixel voisin N blanc.
Dans l'exemple, les pixels candidats sont (**2**,**1**), (**1**,**2**), (**3**,**2**), (**4**,**2**) et (**2**,**4**).



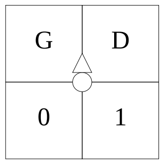
Un moyen simple de trouver un tel pixel est de parcourir les pixels de l'image dans l'ordre habituel (celui du fichier image) et s'arrêter au premier pixel noir dont le voisin N est blanc. Dans l'exemple ci-dessus, c'est le pixel (**2**,**1**) qui sera choisi.

B) Le point initial sera le point NW ($x_0 = \mathbf{x}$-1 , $y_0 = \mathbf{y}$-1) du pixel (**x**,**y**) sélectionné. Dans l'exemple ci-dessus, ce sera le point (*1*,*0*).
Ensuite, il s'agit d'une boucle où on mémorise la position, on avance d'une distance 1, on change (éventuellement) de direction jusqu'à revenir à la position initiale (x_0,y_0) et avec l'orientation E.

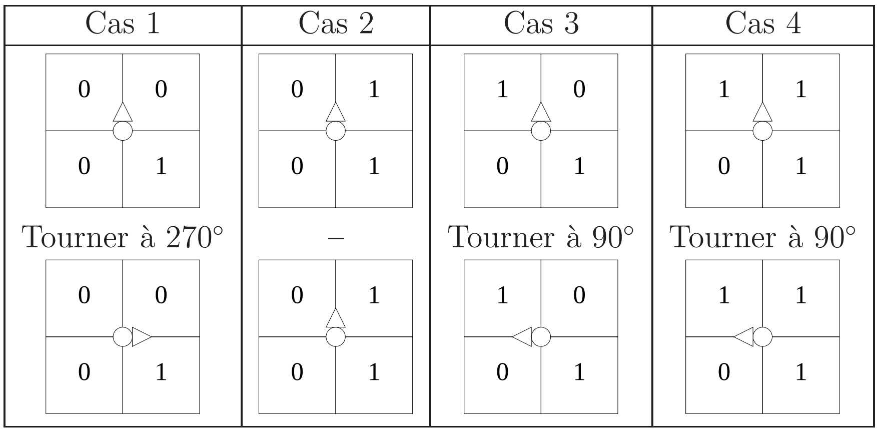
L'algorithme correspondant s'écrit alors :

> $(\mathbf{x}, \mathbf{y}) \leftarrow$ `trouver_pixel_depart(I)`
> $(x_0, y_0) \leftarrow (\mathbf{x} - 1, \mathbf{y} - 1)$ -- *position initiale*
> position $\leftarrow (x_0, y_0)$
> orientation $\leftarrow$ EST
> boucle $\leftarrow$ VRAI
> **tant_que** boucle **faire**
>   `memoriser_position` -- *mémoriser la position*
>   `avancer` -- *avancer de 1*
>   `nouvelle_orientation` -- *calculer la nouvelle orientation*
>
>   **si** position $= (x_0, y_0)$ **et** orientation $=$ EST **faire**
>   boucle $\leftarrow$ FAUX
>   **fin_si**
> **fin_tant_que**
> `memoriser_position` -- *mémoriser la position*

Après que le robot a avancé, pour le calcul de la nouvelle orientation du robot, il faut considérer le pixel voisin *devant à gauche* **G** et le pixel voisin *devant à droite* **D**.



Il y a quatre cas possibles :



La partie `nouvelle_orientation` s'écrit alors :

> $pG \leftarrow$ valeur_pixel_gauche
> $pD \leftarrow$ valeur_pixel_droite
> **si** $pG =$ NOIR **alors**
>   -- *tourner de 90° (un quart de tour dans le sens trigo)*
>   `tourner_90_degres`
> **sinon_si** $pD =$ BLANC **alors**
>   -- *tourner de 270° (un quart de tour dans le sens des aiguilles d'une montre)*
>   `tourner_270_degres`
> **fin_si**

Remarque : pour le cas 4, une autre stratégie (donc un autre algorithme) consisterait à tourner à 270°.

3.3 - Contour sous forme d'une séquence

Le contour calculé doit être stocké sous forme d'une séquence de `Point` mais on ne connait pas a priori la taille de la séquence.

Quelques solutions pour stocker un contour (séquence de points) :

A - Stockage sous forme de fichier

Stocker au fur et à mesure les différents points du contour dans un fichier, ...
L'inconvénient majeur est que l'accès fichier est beaucoup plus lent que l'accès mémoire.

B - Stockage en mémoire

B-1) utiliser une structure de taille fixe de type tableau

```
unsigned int DIM_MAX = ...;
typedef Point TabPoints[DIM_MAX]; /* TabPoints : tableau de DIM_MAX Point*/
typedef struct Contour_
{
    unsigned int np; /* nombre de points de la séquence */
    TabPoints tab  ; /* tableau pour stocker les points */
} Contour;
```

Inconvénient : taille fixée par la valeur de `DIM_MAX`.

– si `DIM_MAX` trop grand, impossible de calculer plusieurs contours, voire un seul contour (saturation de la mémoire),

– si `DIM_MAX` trop petit, impossible de stocker certains contours avec beaucoup de points.

B-2) utiliser une structure permettant d'adapter la taille de la mémoire en fonction du contour.

Le contour va être "construit" au fur et à mesure, à partir d'une séquence vide et ajoutant au fur et à mesure les différents points à la fin de la séquence.

→ avoir une structure de données permettant les opérations principales suivantes :

- initialiser avec une séquence vide,
- ajouter un élément supplémentaire à la fin d'une séquence,
- avoir une séquence ordonnée,
- effacer une séquence (la vider),

Les listes chainées sont des structures adaptées pour effectuer de telles opérations.

Les listes chaînées suffisent pour les tâches d'extraction de contour (tâches 3 et 5), par contre pour les tâches de simplification de contour (tâches 6 et 7), pour chaque contour à simplifier, il sera nécessaire d'avoir une structure de données de type tableau afin de pouvoir accéder à un élément d'un contour à l'aide d'un indice.

De plus pour les tâches de simplification de contour (tâches 6 et 7), les opérations de concaténation de deux listes chainées ainsi que suppression de l'élément en tête de liste pourront être utiles.

B-3) utiliser une autre structure de données (par exemple *tableau à taille variable*) permettant les opérations citées plus haut.

3.4 - Contour sous forme d'une liste chaînée de points

Ci-dessous est donné un exemple de création et manipulation de contours à l'aide de *liste chaînée* de points. L'intégralité du code source est fourni sur la plateforme pédagogique de l'UE.

```
#include<stdio.h>  /* utilisation des entrées-sorties standard de C */
#include<stdlib.h> /* utilisation des fonctions malloc et free */

/*---- le type Point et la fonction set_point ----*/
/* ou inclure le module de la tache 2             */
typedef ... Point;

Point set_point(double x, double y)
...

/*---- le type liste de point ----*/
...
typedef ... Liste_Point;
typedef Liste_Point Contour; /* type Contour = type Liste_Point */

/*---- le type tableau de point ----*/
typedef ... Tableau_Point;

...

/* créer une liste vide */
Liste_Point creer_liste_Point_vide()
...

/* ajouter l'élément e en fin de la liste L, renvoie la liste L modifiée */
Liste_Point ajouter_element_liste_Point(Liste_Point L, Point e)
...

/* suppression de tous les éléments de la liste, renvoie la liste L vide */
Liste_Point supprimer_liste_Point(Liste_Point L)
...

/* concatène L2 à la suite de L1, renvoie la liste L1 modifiée */
Liste_Point concatener_liste_Point(Liste_Point L1, Liste_Point L2)
...

/* si la liste est non vide, la fonction supprime le premier element de L
   si la liste est vide, la fonction ne fait rien
   la fonction renvoie la liste (eventuellement) modifiee */
Liste_Point supprimer_premier_element_liste_Point(Liste_Point L)
...

/* créer une séquence de points sous forme d'un tableau de points
   à partir de la liste de points L */
Tableau_Point sequence_points_liste_vers_tableau(Liste_Point L)
...

/* écrire le contour L à l'écran */
void ecrire_contour(Liste_Point L)
...
```

Exemple de programme principal :

```
int main()
{
    Contour C1,C2;

    /* initialiser C1 comme contour vide */
    C1 = creer_liste_Point_vide();
    printf("C1 : "); ecrire_contour(C1);
```

→ on obtient à l'écran

```
C1 : Contour avec 0 points
[]
```

```
    /* ajouter les points (5,3),(3,1),(7,2) et (1,6) dans C1 */
    C1 = ajouter_element_liste_Point(C1, set_point(5,3));
    C1 = ajouter_element_liste_Point(C1, set_point(3,1));
    C1 = ajouter_element_liste_Point(C1, set_point(7,2));
    C1 = ajouter_element_liste_Point(C1, set_point(1,6));
    printf("C1 : "); ecrire_contour(C1);
```

→ on obtient à l'écran

```
C1 : Contour avec 4 points
[(  5,  3)(  3,  1)(  7,  2)(  1,  6)]
```

```
    /* supprimer le premier point de C1, puis ajouter le point (4,1) dans C1 */
    C1 = supprimer_premier_element_liste_Point(C1);
    C1 = ajouter_element_liste_Point(C1, set_point(4,1));
    printf("C1 : "); ecrire_contour(C1);
```

→ on obtient à l'écran

```
C1 : Contour avec 4 points
[(  3,  1)(  7,  2)(  1,  6)(  4,  1)]
```

```
    /* créer le contour C2 avec les points (9,5) et (5,7) */
    C2 = creer_liste_Point_vide();
    C2 = ajouter_element_liste_Point(C2, set_point(9,5));
    C2 = ajouter_element_liste_Point(C2, set_point(5,7));
    printf("C2 : "); ecrire_contour(C2);
```

→ on obtient à l'écran

```
C2 : Contour avec 2 points
[(  9,  5)(  5,  7)]
```

```
    /* concaténer C2 à la suite de C1 */
    C1 = concatener_liste_Point(C1,C2);
    printf("C1 : "); ecrire_contour(C1);
```

→ on obtient à l'écran

```
C1 : Contour avec 6 points
[(  3,  1)(  7,  2)(  1,  6)(  4,  1)(  9,  5)(  5,  7)]
```

```
    /* supprimer le contour C1 */
    C1 = supprimer_liste_Point(C1);
    printf("C1 : "); ecrire_contour(C1);
} // fin du programme
```

→ on obtient à l'écran

```
C1 : Contour avec 0 points
[]
```

Tâche 4 - Sortie au format PostScript encapsulé

4.1 - Présentation

PostScript

Langage informatique spécialisé dans la description de pages, mis au point par la société Adobe.

Description des images au format *vectoriel*, c'est à dire, description d'image à l'aide de primitives géométriques (différent du format *bitmap* où une image est représentée par un tableau de pixels).

Autres fonctionnalités : affichage de texte, motif, couleur, instructions de programmation, variables, inclusion d'images au format bitmap, ...

Fichiers PostScript avec extension `.ps`

PostScript encapsulé (Encapsulated PostScript ou EPS)

Version de PostScript permettant la définition d'images (bitmap ou vectorielle) pouvant être ensuite incluses dans d'autres fichiers (PostScript, traitement de texte, ...)

Fichiers PostScript encapsulé avec extension `.eps`

Utilisation de fichiers EPS dans le projet

Utilisation comme fichier de sortie pour les tâches *vectorisation (extraction)* et *simplification* pour comparer visuellement les résultats obtenus.
→ écriture d'un module/paquetage pour la création de fichier EPS.
Visualisation des fichiers à l'aide du logiciel `GhostView` (`gv` sous Linux).

4.2 - Structure d'un document EPS

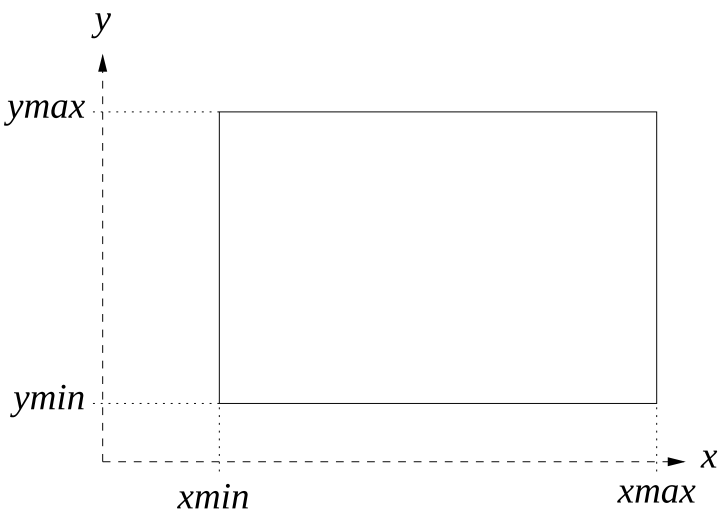
Document format texte avec extension `.eps`

Fichier formé :

- d'un en-tête
  `%!PS-Adobe-3.0 EPSF-3.0`
- suivi de la boîte englobante
  `%%BoundingBox:` *xmin ymin xmax ymax*
  (avec *xmin ymin xmax ymax* réels)
- une suite d'instructions en langage Postscript
  utilisation de la notation postfixée (opérandes suivis de l'opérateur)
- la commande `showpage`

La zone définie par la *BoundingBox* correspond au rectangle qui est affiché / imprimé.
Normalement tous les ordres graphiques doivent être contenus dans cette boite englobante :
xmin correspond au bord de gauche et *xmax* au bord de droite.
ymin correspond au bord du bas et *ymax* au bord du haut.

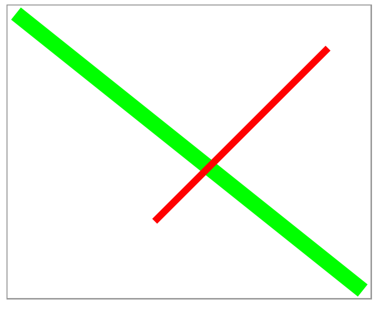
ATTENTION : en PostScript, axe des ordonnées orienté de bas en haut.



Un commentaire dans un fichier source est la partie d'une ligne suivant les deux caractères `%` et *espace*.

Exemple :

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 100 80

% segment du point (0,80) au point (100,0)
0 80 moveto 100 0 lineto
% de couleur vert, et épaisseur 4.5
0 1 0 setrgbcolor 4.5 setlinewidth
stroke

% segment du point (40,20) au point (90,70)
40 20 moveto 90 70 lineto
% de couleur rouge, et épaisseur 2.0
1 0 0 setrgbcolor 2.0 setlinewidth
stroke

showpage
```



4.3 - Les instructions essentielles

PostScript permet de définir des formes géométriques complexes à partir de primitives géométriques notamment segments de droite, lignes polygonales et courbes de Bézier cubiques, puis de les tracer ou de les remplir.

Modes de tracé Le tracé effectif se fait avec l'une des deux instructions suivantes :

$\boxed{\textbf{stroke}}$: tracé de formes géométriques

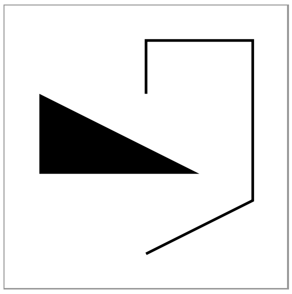
$\boxed{\textbf{fill}}$: remplissage de formes géométriques

Segment / Ligne polygonale

Un tracé d'un segment de droite se fait en utilisant les instructions `moveto` (pour placer le point initial) puis `lineto` (pour placer le point final).

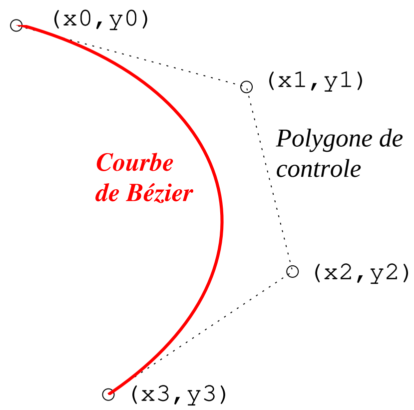
Les instructions `lineto` peuvent être enchaînées afin de tracer une ligne polygonale (sans à avoir recours à l'instruction `moveto`) car le point final d'un segment défini par `lineto` devient le point courant donc le point initial de la primitive géométrique suivante.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 100 100

% définition d'un polygone ouvert
% formé de 5 points soit 4 segments
50 10 moveto 90 30 lineto 90 90 lineto
50 90 lineto 50 70 lineto
stroke  % tracé

% définition d'un triangle :
% polygone fermé formé de 4 points soit de 3 segments
10 70 moveto 70 40 lineto 10 40 lineto 10 70 lineto
fill % remplissage

showpage
```



Remarque : Les coordonnées des points peuvent être des entiers ou des réels.

Courbes de Bézier cubiques (utilisation dans la tâche 7)



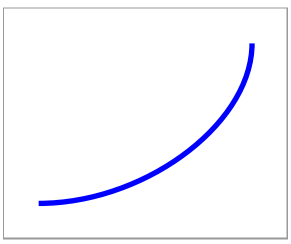
L'instruction `curveto` permet de tracer des courbes de Bézier de degré 3 dont les 4 points de contrôle sont :

P_0=`(x0,y0)`, P_1=`(x1,y1)`, P_2=`(x2,y2)`, P_3=`(x3,y3)`

Courbe de Bézier cubique $\mathcal{C} = \left\{ C(t)\,,\, t \in [0,1] \right\}$

avec $C(t) = (1-t)^3\, P_0 + 3\,t\,(1-t)^2\, P_1 + 3\,t^2\,(1-t)\, P_2 + t^3\, P_3$

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 100 80

% une courbe de Bézier en bleu
10 10 moveto 50 10 90 40 90 70 curveto
0 0 1 setrgbcolor 2.0 setlinewidth
stroke
showpage
```



Point courant et tracé composite

La définition de segments, lignes polygonales et courbes de Bézier se fait en positionnant puis déplaçant un point courant.

Le positionnement du point courant en $(x0, y0)$ se fait avec l'instruction **moveto** :

$\boxed{x0\ y0\ \textbf{moveto}}$

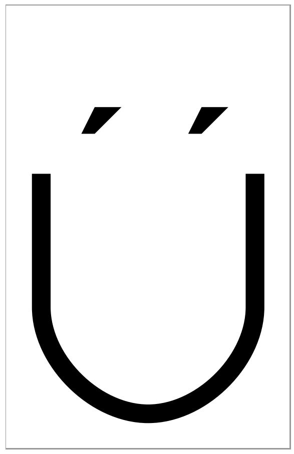
Le point courant est déplacé après l'instruction **lineto** ou **curveto** :

$\boxed{x1\ y1\ \textbf{lineto}}$: le point courant est alors $(x1, y1)$

$\boxed{x1\ y1\ x2\ y2\ x3\ y3\ \textbf{curveto}}$: le point courant est alors $(x3, y3)$

On peut alors créer des formes complexes formées de segments, lignes polygonales et courbes de Bézier.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 100 160

% lettre U tréma modélisée à l'aide de 2 polygones remplis
25 115 moveto 30 115 lineto
40 125 lineto 30 125 lineto 25 115 lineto % polygone fermé
65 115 moveto 70 115 lineto
80 125 lineto 70 125 lineto fill % polygone non fermé

% et une courbe composite formée de 1 segment
% puis 2 courbes de Bézier, puis 1 segment
10 100 moveto
10 50 lineto
10 30 30 10 50 10 curveto
70 10 90 30 90 50 curveto
90 100 lineto
7 setlinewidth stroke % épaisseur du tracé 7

showpage
```



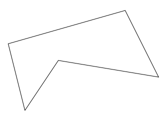
L'instruction `stroke` permet de tracer de segments et/ou courbes.
L'instruction `fill` permet de remplir des contours définis par suite de segments et/ou courbes.
En mode remplissage (`fill`), si un polygone n'est pas fermé (i.e. premier point $\neq$ dernier point), il est refermé automatiquement.

Contours polygonaux (tâches 4-5-6)

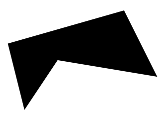
Sauvegarde au format EPS d'un ou plusieurs contours, chaque contour étant une suite continue de segments formant un polygone fermé.

Exemple : polygone $\{(10, 0), (20, 15), (50, 10), (40, 30), (5, 20), (10, 0)\}$ provenant d'une image de dimensions $L = 55$ et $H = 35$

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 55 35

% polygone : moveto pour le premier point
% et lineto pour les suivants
10  0 moveto  20 15 lineto  50 10 lineto
40 30 lineto   5 20 lineto  10  0 lineto

%<instruction tracé>%

showpage
```



%<instruction tracé>% `stroke`



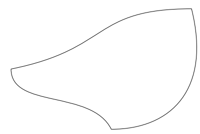
%<instruction tracé>% `fill`

Contours formés de courbes de Bézier (tâches 7)

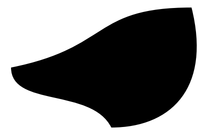
Sauvegarde au format EPS d'un ou plusieurs contours, chaque contour étant une suite de Bézier de degré 3 formant un contour fermé.

Exemple : contour formé des 3 courbes de Bézier
$\mathcal{B}_1 = \mathcal{B}[(0,15),(0,5),(20,10),(25,0)]$,
$\mathcal{B}_2 = \mathcal{B}[(25,0),(40,0),(50,10),(45,30)]$,
$\mathcal{B}_3 = \mathcal{B}[(45,30),(20,30),(25,20),(0,15)]$,
et provenant d'une image de dimensions $L = 55$ et $H = 35$

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 55 35

% suite de Bézier 3 :
% moveto avec le premier pt de c.
% de la premiere courbe de Bezier
% puis chaque courbe de Bézier
% curveto avec les 3 derniers pt. de c.
 0 15 moveto
 0  5   20 10   25  0 curveto
40  0   50 10   45 30 curveto
20 30   25 20    0 15 curveto

%<instruction tracé>%

showpage
```



%<instruction tracé>%
`stroke`

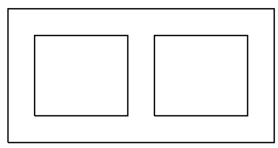


%<instruction tracé>%
`fill`

Contours multiples

Description de tous les contours, puis instruction de tracé.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 200 100

% polygone externe décrit dans le sens +
0 0 moveto 200 0 lineto 200 100 lineto
0 100 lineto 0 0 lineto

% polygone interne décrit dans le sens -
20 20 moveto 20 80 lineto 90 80 lineto
90 20 lineto 20 20 lineto

% polygone interne décrit dans le sens -
110 20 moveto 110 80 lineto 180 80 lineto
180 20 lineto 110 20 lineto

%<instruction de tracé>%

showpage
```



%<instruction tracé>%
`stroke`

%<instruction tracé>%
`fill`

Remarque : dans la tâche 5, pour les images avec contours imbriqués, l'algorithme utilisé assurera la bonne description (orientation) de chaque contour, et le remplissage se fera correctement.

4.4 - Macro-instructions

Possibilité de définir des macros-instructions ou *clés* afin de réduire la taille du fichier.

```
...
% redefinition de moveto, lineto, stroke
/l {lineto} def     % l équivalent à lineto
/m {moveto} def     % m équivalent à moveto
/s {stroke} def     % s équivalent à stroke

% polygone externe décrit dans le sens +
0 0 m 200 0 l 200 100 l 0 100 l 0 0 l

% polygone interne décrit dans le sens −
20 20 m 20 80 l 90 80 l 90 20 l 20 20 l

% polygone interne décrit dans le sens −
110 20 m 110 80 l 180 80 l 180 20 l 110 20 l

% tracé en mode stroke
s
...
```

4.5 - Résumé des instructions

Instruction	**Code PostScript**	**Point courant après exéc. de l'instruction**
En-tête avec boite englobante $[x0, x1] \times [y0, y1]$	`%!PS-Adobe-3.0 EPSF-3.0` `%%BoundingBox:` $x0$ $y0$ $x1$ $y1$	– *indéfini* –
Définition du point courant $P0 = (x0, y0)$	$x0$ $y0$ `moveto`	$P0 = (x0, y0)$
Segment de $P0$ à $P1 = (x1, y1)$	$x1$ $y1$ `lineto`	$P1 = (x1, y1)$
Bezier cubique de points de contrôle $P0$ (point courant), $P1 = (x1, y1)$, $P2 = (x1, y1)$, $P3 = (x1, y1)$	$x1$ $y1$ $x2$ $y2$ $x3$ $y3$ `curveto`	$P3 = (x3, y3)$
Tracé	`stroke`	– *inchangé* –
Remplissage	`fill`	– *inchangé* –
Epaisseur du tracé e (*par défaut* $e = 1$) Conseil : utiliser e=0	e `setlinewidth`	– *inchangé* –
Couleur de dessin (r, g, b) (*par défaut* $(r, g, b) = (0, 0, 0)$ = *Noir*)	r g b `setrgbcolor`	– *inchangé* –
Commentaire	`%` *commentaire* → *fin de ligne*	– *inchangé* –
Définition d'une macro-instruction	/*cle* {*code PostScript*} `def`	– *inchangé* –
Fin de fichier	`showpage`	

$x0$ $y0$ $x1$ $y1$ $x2$ $y2$ $x3$ $y3$	entiers ou réels flottants
e	entier ou réel flottant positif ou nul
r g b	réels flottants entre 0 et 1
cle	identificateur formé de caractères alphanumériques premier caractère : lettre

Visualisation d'un fichier EPS : par exemple en utilisant gv (sous Linux), Evince (sous Windows), Preview (sous MacOS).

Tâche 5 - Extraction des contours d'une image

5.1 - Présentation

Dans la tâche 3, un seul contour dans l'image est calculé.
Des images complexes peuvent présenter plusieurs contours (disjoints ou imbriqués).

Exemples

Exemple 1

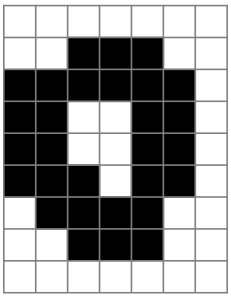


Image 7×9
(2 contours)

Exemple 2

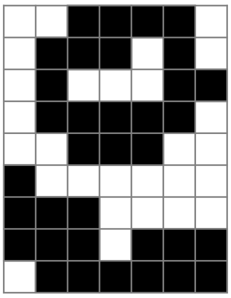


Image 7×9
(3 contours)

Exemple 3

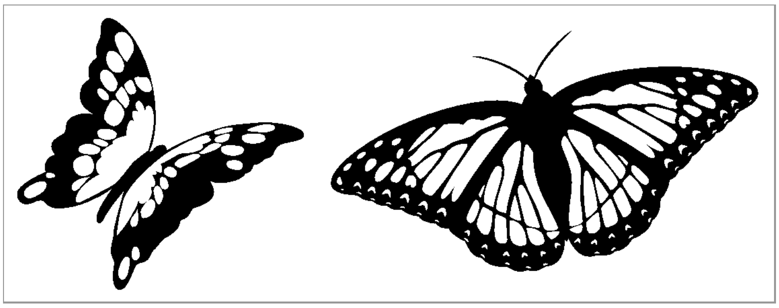


Image 1122×417
(140 contours)

Dans cette tâche, on souhaite obtenir pour une image, l'ensemble de tous les contours.
Par exemple, pour l'image de l'exemple 2 :

Contour 1 :

x	2	3	4	5	6	6	6	7	7	6	6	5	5	4	3	2	2	1	1	1	1	2	2
y	0	0	0	0	0	1	2	2	3	3	4	4	5	5	5	5	4	4	3	2	1	1	0

23 points (soit 22 segments)

Contour 2 :

x	2	3	4	5	5	5	4	4	3	2	2
y	3	3	3	3	2	1	1	2	2	2	3

11 points (soit 10 segments)

Contour 3 :

x	0	1	1	2	3	3	3	4	4	5	6	7	7	7	6	5	4	3	2	1	1	0	0	0	0
y	5	5	6	6	6	7	8	8	7	7	7	7	8	9	9	9	9	9	9	9	8	8	7	6	5

25 points (soit 24 segments)

Au total, 3 contours pour un total de 56 segments (59 points).

5.2 - Méthode de détection de tous les contours

Chaque contour étant fermé, il passe nécessairement par un point (x, y) dont le pixel voisin NE est blanc et le pixel voisin SE est noir.

Réciproquement tout pixel $(\mathbf{x}, \mathbf{y})$ noir avec le pixel voisin N blanc est adjacent à un contour passant par le point voisin NW $(x-1, y-1)$ et le point voisin NE $(x, y-1)$.

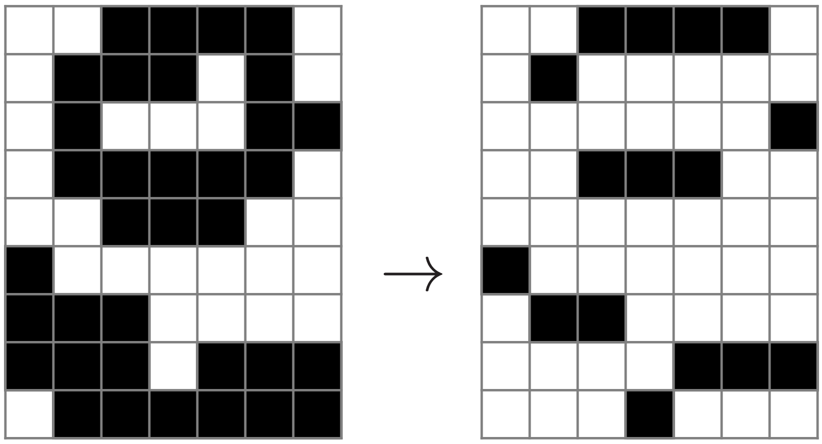
L'idée de la méthode pour détecter tous les contours entre composante(s) *blanc* et composante(s) *noir* est la suivante :

Etape 1 détecter dans l'image I initiale, tous les pixels noirs dont les voisins N (au-dessus) sont blancs : créer une image auxiliaire M appelée *image-masque* de mêmes dimensions que l'image I, chaque pixel $(\mathbf{x}, \mathbf{y})$ de M est noir si et seulement si le pixel $(\mathbf{x}, \mathbf{y})$ de I est noir avec son voisin N blanc.

Etape 2 tant qu'il existe un pixel $(\mathbf{x}, \mathbf{y})$ noir dans M alors rechercher un nouveau contour à partir du point voisin NW $(x-1, y-1)$ (et en commençant à se déplacer dans la direction E).
Dans le parcours d'un contour, lorsque le "robot" se déplace dans la direction E à partir du point (x, y), modifier dans M le pixel voisin SE $(\mathbf{x}, \mathbf{y})$ en le mettant à blanc.

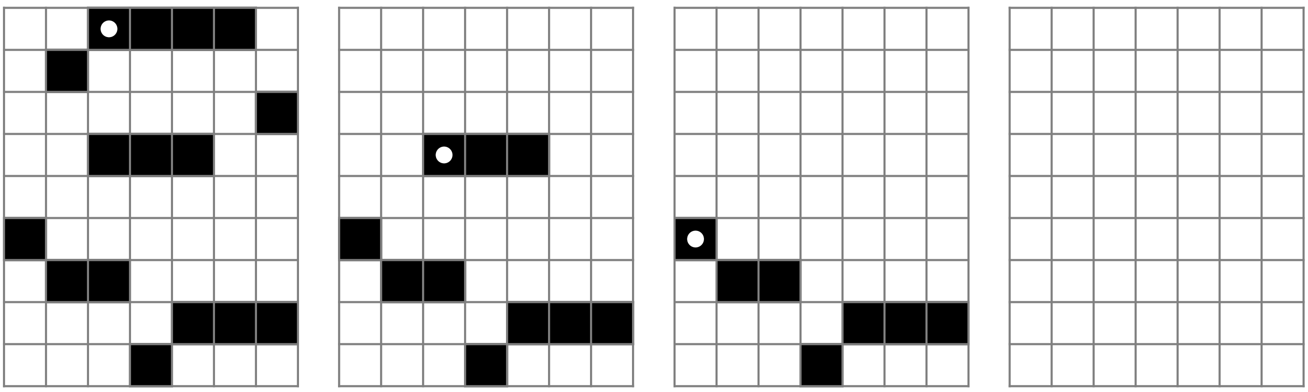
Exemple : image de l'exemple 2

1. Initialisation :



L'image I L'*image-masque* M

2. Image-masque M : choix du pixel candidat, et modification après le calcul de chaque contour :



Pixel (**3**,**1**) trouvé → contour 1 à partir du point $(2.0, 0.0)$	Pixel (**3**,**4**) trouvé → contour 2 à partir du point $(2.0, 3.0)$	Pixel (**1**,**6**) trouvé → contour 3 à partir du point $(0.0, 5.0)$	plus de pixel noir **FIN**

Tâche 6 - Simplification de contour par segment

6.1 - Présentation

La vectorisation d'image donne un ou plusieurs contours, chaque contour étant formé d'une séquence de points. On peut ensuite essayer de simplifier chaque contour obtenu afin de réduire la taille de chacun.

Différentes méthodes peuvent être envisagées pour la simplification d'un contour, par exemple :

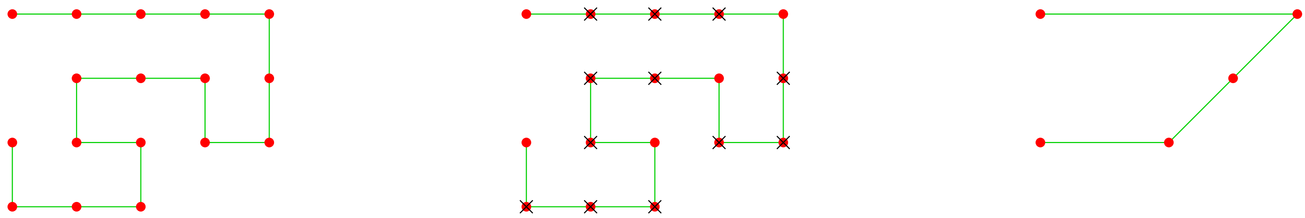
1. exemple de méthode 1 : trouver des sous-séquences de points alignés qui peuvent être simplifiées sans modifier la forme du contour :

Séquence initiale   Séquence simplifiée

2. exemple de méthode 2 : trouver des sous-séquences de points proches d'un segment qui peuvent être simplifiées en modifiant très peu la forme du contour :

Séquence initiale   Séquence simplifiée

3. exemple de méthode 3 : réduire le nombre de points en ne conservant qu'un point sur n.

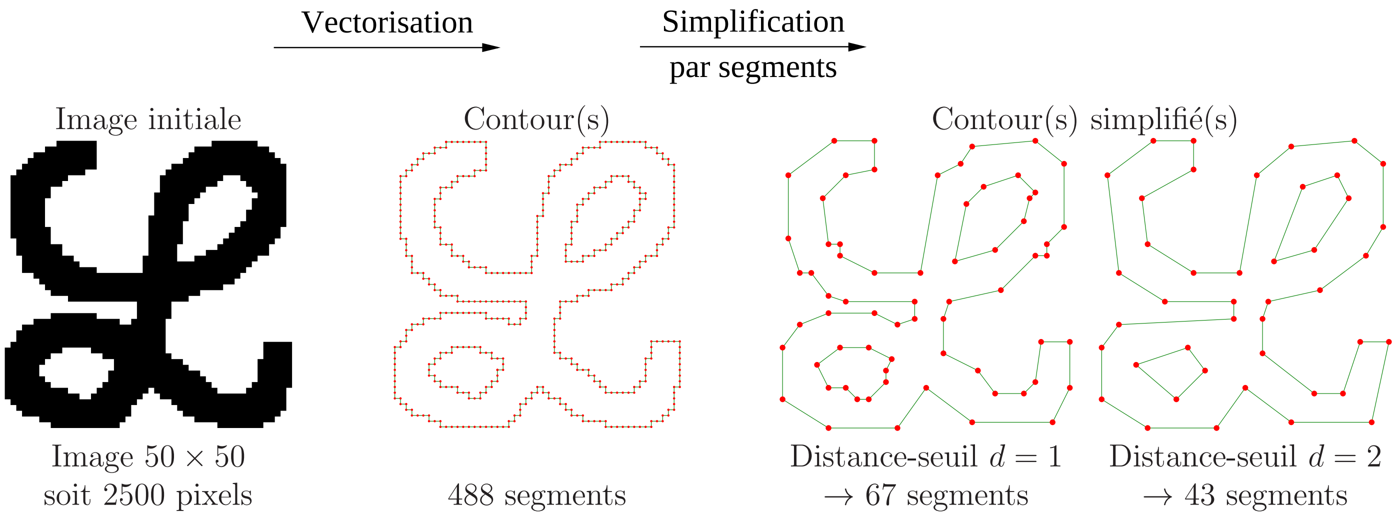


Séquence initiale   Points supprimés (n=4)   Séquence simplifiée

Cependant ces trois méthodes ont des inconvénients, les deux premières nécessitant une recherche préalable dans le contour initial pour détecter des configurations particulières, la troisième est plus simple mais mal adaptée à la conservation de formes ou détails pertinents.

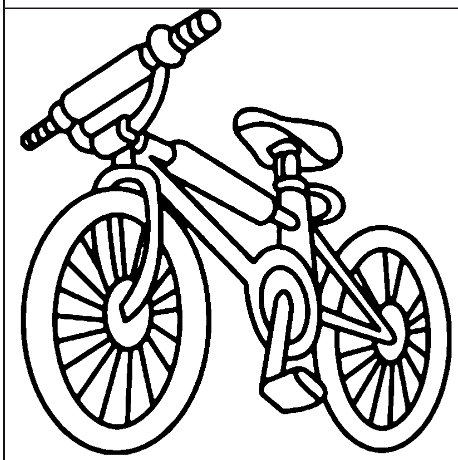
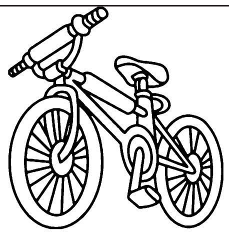
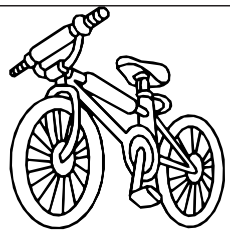
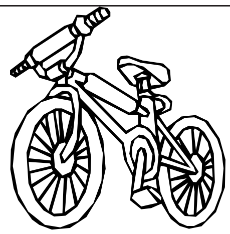
La méthode utilisée dans les tâches 6 et 7 a pour objet de simplifier des contours suivant un critère géométrique simple et utilisant une *distance-seuil* (écart permis entre la séquence initiale et la séquence simplifiée).

Exemple 1 :



Souvent le choix de la distance-seuil est fait en fonction de l'échelle d'affichage d'une image.

Exemple 2 : Image bitmap de dimensions 774×800

Echelle 1		
		
Echelle 0,5		
		
Echelle 0,2		

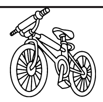
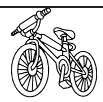
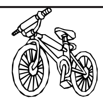
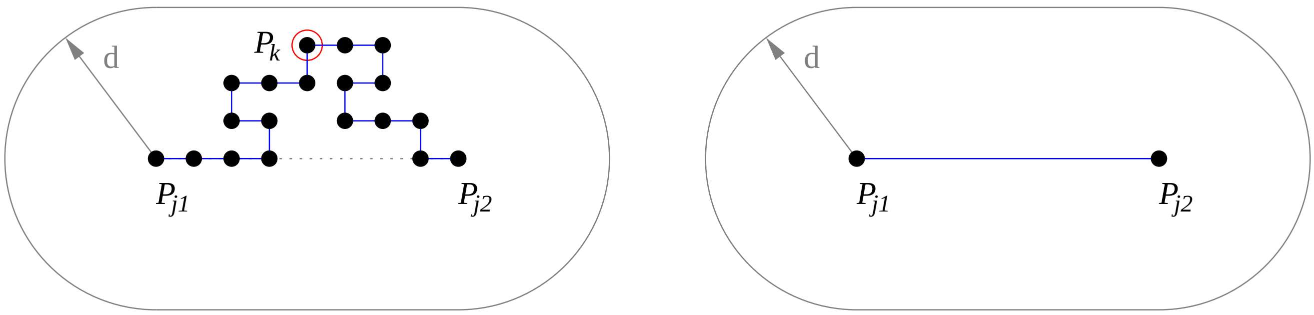

		

Image initiale 86 contours 32098 segments	Simplification distance-seuil $d = 2$ 998 segments	Simplification distance-seuil $d = 5$ 584 segments

Principe de simplification suivant une distance-seuil

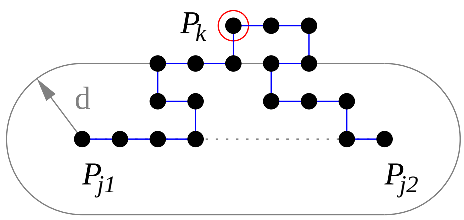
Pour un contour polygonal $C = \{P_{j1}, \ldots, P_{j2}\}$ avec $j1 < j2$, considérer la zone à distance inférieure ou égale à la distance-seuil d du segment $S = [P_{j1}, P_{j2}]$, puis rechercher le point P_k le plus éloigné du segment S.

Exemple 1 :

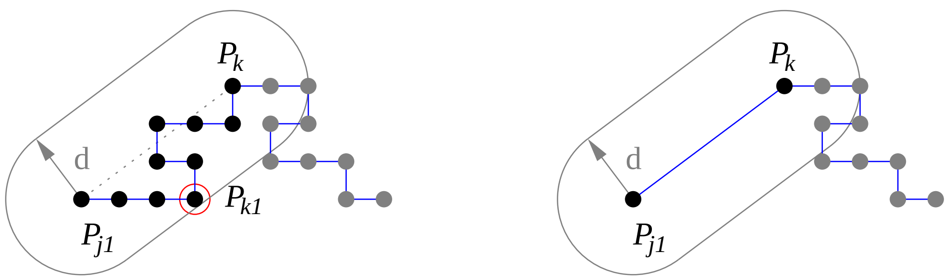


Le point P_k est à distance inférieure ou égale à d du segment S
$\rightarrow$ on simplifie le contour polygonal C par le segment S

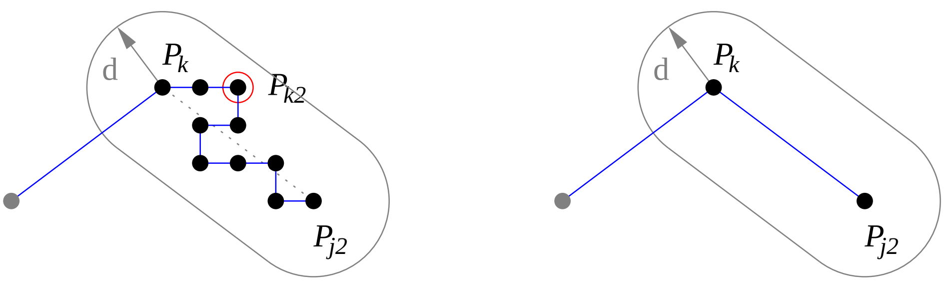
Exemple 2 :



Le point P_k est à distance strictement supérieure à d du segment S
$\rightarrow$ on "divise" le problème en deux sous-problèmes :



Simplification de la première partie de $C = \{P_{j1}, \ldots, P_k\}$



Simplification de la seconde partie de $C = \{P_k, \ldots, P_{j2}\}$

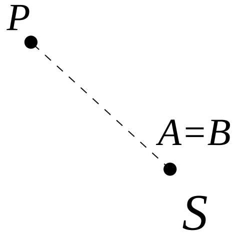
- Cette méthode est basée sur le calcul de la distance entre un point et un segment.

6.2 - Calcul de la distance point-segment

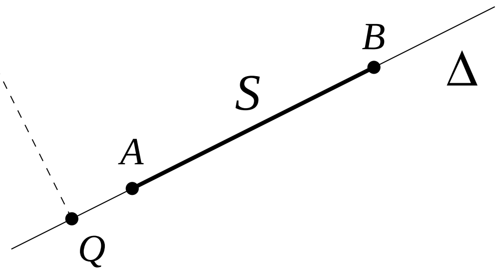
Soit P un point du plan et $S = [A, B]$ un segment $\rightarrow$ calculer $d_P = \text{distance\_point\_segment}(P, S)$, distance entre P et le point du segment S le plus proche de P.

• cas $A = B$:

$d_P = \|\overrightarrow{AP}\| = \|\overrightarrow{BP}\|$



• cas $A \neq B$:



Soit Δ la droite support du segment S :

$$\Delta = \{M, \overrightarrow{OM} = \overrightarrow{OA} + t\,\overrightarrow{AB}, t \in \mathbb{R}\}$$

et Q la projection orthogonale de P sur Δ.

Le point Q se calcule à partir des trois points P, A et B ainsi :

$$\left\{ \begin{array}{c} Q \in \Delta \\ \overrightarrow{PQ} \text{ orthogonal à } \overrightarrow{AB} \end{array} \right\} \iff \left\{ \begin{array}{c} \overrightarrow{OQ} = \overrightarrow{OA} + \lambda\,\overrightarrow{AB} \iff Q = A + \lambda\,(B - A) \\ \langle \overrightarrow{PQ}, \overrightarrow{AB} \rangle = 0 \end{array} \right\}$$

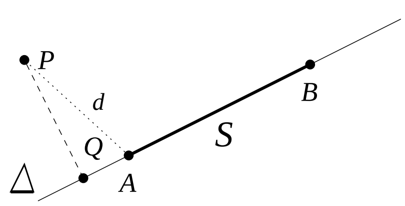
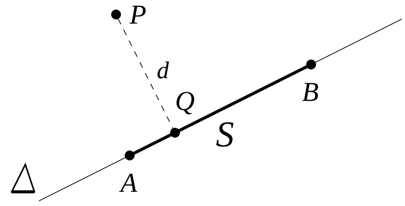
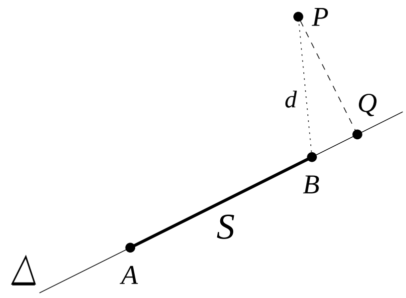
On en déduit alors l'expression de λ permettant ensuite de calculer Q :

$$\langle \overrightarrow{PQ}, \overrightarrow{AB} \rangle = \langle \overrightarrow{OQ} - \overrightarrow{OP}, \overrightarrow{AB} \rangle = \langle \overrightarrow{OQ}, \overrightarrow{AB} \rangle - \langle \overrightarrow{OP}, \overrightarrow{AB} \rangle = \langle \overrightarrow{OA} + \lambda\,\overrightarrow{AB}, \overrightarrow{AB} \rangle - \langle \overrightarrow{OP}, \overrightarrow{AB} \rangle$$

$$= \langle \overrightarrow{OA}, \overrightarrow{AB} \rangle + \lambda \langle \overrightarrow{AB}, \overrightarrow{AB} \rangle - \langle \overrightarrow{OP}, \overrightarrow{AB} \rangle = \lambda \langle \overrightarrow{AB}, \overrightarrow{AB} \rangle - \langle \overrightarrow{OP} - \overrightarrow{OA}, \overrightarrow{AB} \rangle = 0$$

$$\iff \lambda = \frac{\langle \overrightarrow{OP} - \overrightarrow{OA}, \overrightarrow{AB} \rangle}{\langle \overrightarrow{AB}, \overrightarrow{AB} \rangle} = \frac{\langle \overrightarrow{AP}, \overrightarrow{AB} \rangle}{\langle \overrightarrow{AB}, \overrightarrow{AB} \rangle}$$

Trois cas sont à distinguer suivant la position du point Q par rapport au segment $S = [A, B]$.

$Q \notin S$, Q du coté de A $\lambda < 0$	$Q \in S$ $0 \leq \lambda \leq 1$	$Q \notin S$, Q du coté de B $\lambda > 1$
		
$d_P = \|\overrightarrow{AP}\|$	$d_P = \|\overrightarrow{QP}\|$ avec $Q = A + \lambda\,(B - A)$	$d_P = \|\overrightarrow{BP}\|$

6.3 - L'algorithme de Douglas-Peucker

Le principe

• Données : $C = \{P_0, \ldots, P_p\}$ séquence ordonnée des points d'un contour polygonal
$j1$ et $j2$ deux indices avec $0 \leq j1 < j2 \leq p$
d `distance-seuil` (avec d réel positif ou nul)

• Résultat : $L = \{S_1, S_2, \ldots, S_q\}$ séquence de segments avec $S_i = [A_i, B_i]$, $A_{i+1} = B_i$ pour $1 \leq i \leq q-1$

On considère le segment $S = [A, B] = [P_{j1}, P_{j2}]$
rechercher le point P_k ($j1 \leq k \leq j2$) le plus éloigné du segment S
si $distance(P_k, S) \leq d$ alors
on simplifie $\{P_{j1}, \ldots, P_{j2}\}$ par le seul segment S : $L = \{S\}$
sinon
on scinde le contour polygonal C en deux suivant le point P_k :
$C_1 = \{P_{j1}, P_{j1+1}, \ldots, P_{k-1}, P_k\}$ et $C_2 = \{P_k, P_{k+1}, \ldots, P_{j2-1}, P_{j2}\}$
on simplifie C_1 avec la distance-seuil $d \rightarrow$ séquence de segments L_1
on simplifie C_2 avec la distance-seuil $d \rightarrow$ séquence de segments L_2
on fusionne (concatène) les séquences L_1 et $L_2 \rightarrow$ L
fin_si

On voit que dans le cas $distance(P_k, S) > d$ la simplification de C consiste à faire deux simplifications du même type
$\rightarrow$ algorithme récursif de type "diviser pour régner" ("divide and conquer")

L'algorithme de simplification par segment

• Données : `CONT` $= \{P_0, \ldots, P_p\}$ séquence ordonnée des points d'un contour polygonal
$j1$ et $j2$ (indices avec $0 \leq j1 < j2 \leq p$)
d `distance-seuil` (avec d réel positif ou nul)

• Résultat : `L` $= \{S_1, S_2, \ldots, S_q\}$ séquence ordonnée de segments, le premier point de S_1 est P_{j1}, le second point de S_j est le premier point de S_{j+1}, le second point de S_q est P_{j2}.

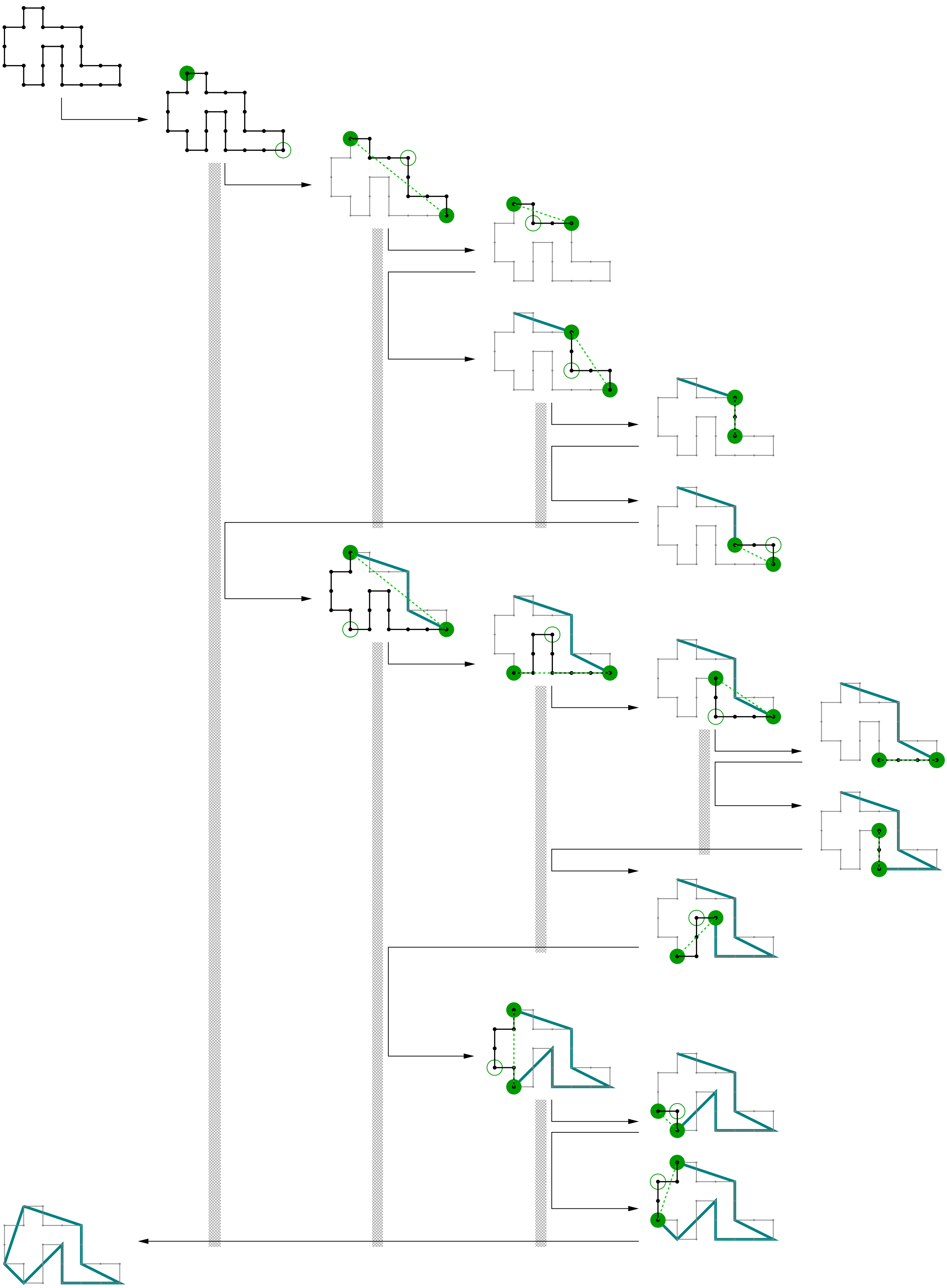
```
-- simplifier la partie du contour CONT compris entre les indices j1 et j2 avec la distance-seuil d
-- la fonction renvoie la séquence de segments L
-- procédure récursive de type "diviser pour régner" ("divide and conquer")
fonction simplification_douglas_peucker (CONT,j1,j2,d) → L
  -- (0) approcher la séquence de n + 1 points CONT(j1..j2) par le segment S = [Pj1, Pj2]
  S ← segment(Pj1,Pj2)

  -- (1) rechercher le point Pk le plus éloigné du segment S
  -- ainsi que la distance dmax correspondante
  dmax ← 0
  k ← j1
  pour j de j1 + 1 à j2 faire
    dj ← distance_point_segment(Pj , S)
    si dmax < dj alors
      dmax ← dj
      k ← j
    fin_si
  fin_pour

  si dmax ≤ d alors
    -- (2) dmax ≤ d : simplification suivant le segment S
    L ← {S}
  sinon
    -- (3) dmax > d : "diviser pour régner"

    -- (3.1) décomposer le problème en deux
    -- simplifier la partie du contour CONT compris entre les indices j1 et k avec la distance-seuil d
    L1 ← simplification_douglas_peucker (CONT,j1,k,d)

    -- simplifier la partie du contour CONT compris entre les indices k et j2 avec la distance-seuil d
    L2 ← simplification_douglas_peucker (CONT,k,j2,d)

    -- (3.2) fusionner les deux séquences L1 et L2
    L ← concatenation(L1,L2)
  fin_si
  retourner L   -- retourner la séquence L
```

Appel principal :
`L ← simplification_douglas_peucker(CONT, premier_indice(CONT), dernier_indice(CONT),d)`

IMPORTANT : dans cet algorithme, la simplification concerne la partie du contour `CONT` entre les indices `j1` et `j2`, d'où l'intéret d'avoir une structure de données de type tableau pour `CONT`.

Exemple

Données : Contour avec 24 segments - distance seuil $d = 0,9$



Résultat de la simplification : séquence avec 8 segments

Courbes de Bézier

1 - Courbes paramétrées polynomiales

• **Definition 1** Soients $I = [a, b]$ un intervalle borné de $\mathbb{R}$, deux fonctions $x : t \mapsto x(t)$ et $y : t \mapsto y(t)$ définies sur I. Dans le plan $\mathbb{R}^2$, l'ensemble des points

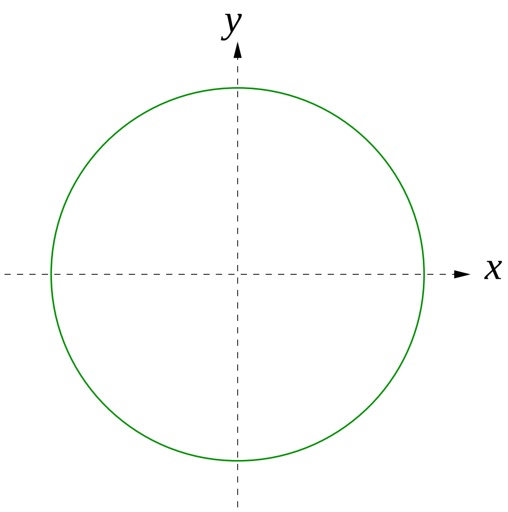
$$\mathcal{C} = \left\{ C(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \ , \ t \in [a, b] \right\}$$

est une ***courbe paramétrée***.
t est appelé le ***paramètre***, et $C(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$ est le point de la courbe correspondant au paramètre t.

• **Exemple 1**

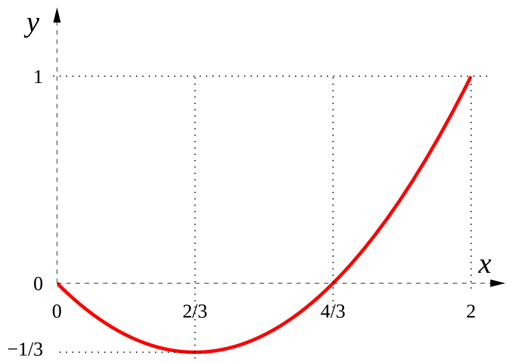
Exemple 1-1 : le cercle unité

$$\mathcal{C} = \left\{ C(t) = \begin{pmatrix} \cos(t) \\ \sin(t) \end{pmatrix} \ , \ t \in [0, 2\pi] \right\}$$



Exemple 1-2 : une courbe polynomiale

$$\mathcal{C} = \left\{ C(t) = \begin{pmatrix} 2\,t \\ 3\,t^2 - 2\,t \end{pmatrix} \ , \ t \in [0, 1] \right\}$$



• **Definition 2** Une courbe est dite ***de classe*** C^k si les deux fonctions $x(t)$ et $y(t)$ sont k-fois continument dérivables sur I.
Une courbe est dite ***de classe*** C^∞ si les deux fonctions $x(t)$ et $y(t)$ sont infiniment continument dérivables sur I.

• **Definition 3** Une courbe paramétrée $\mathcal{C} = \left\{ C(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \ , \ t \in [a, b] \right\}$ est ***polynomiale*** si les deux fonctions $x(t)$ et $y(t)$ sont des polynomes de la variable t.
Une courbe paramétrée polynomiale est de classe C^∞.
Le degré d'une courbe paramétrée polynomiale est le degré maximal des deux fonctions $x(t)$ et $y(t)$.
Par exemple, dans l'exemple 1-2 ci-dessus, la courbe est de degré 2 car $x(t)$ est de degré 1 et $y(t)$ est de degré 2.

• Dans ce chapitre, on utilisera uniquement des courbes polynomiales de degré d supérieur ou égal à 1.

• **Propriété 1** Soit $\mathcal{C} = \{C(t), t \in I\}$ une courbe polynomiale de degré $d \geq 1$ et $t \in I$.
La droite tangente Δ_t à la courbe $\mathcal{C}$ au point $C(t)$ a pour équation

$$\Delta_t = \Big\{ M_t(u) = C(t) + u\,V(t) \ ,\ u \in \mathbb{R} \Big\}$$

avec $V(t)$ correspondant à la première dérivée $C^{(k)}(t)$ non nulle (k entier strictement positif).
Cette droite tangente Δ_t a donc pour équation paramétrée :

$$\Delta_t = \left\{ M_t(u) = C(t) + u\,C^{(k)}(t) = \begin{pmatrix} \bar{x}(u) = x(t) + u\,x^{(k)}(t) \\ \bar{y}(u) = y(t) + u\,y^{(k)}(t) \end{pmatrix} \ ,\ u \in \mathbb{R} \right\}$$

Preuve : cela provient du développement limité de $x(t)$ et $y(t)$: supposons qu'on ait

$$C'(t) = C''(t) = \cdots = C^{(k-1)}(t) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \text{et} \qquad C^{(k)}(t) \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\left\{ \begin{array}{lllllllllllll} x(t+h) & = & x(t) & + & h\,x'(t) & + & \dfrac{h^2}{2}\,x''(t) & + & \dfrac{h^3}{3!}\,x^{(3)}(t) & + \ \ldots \ + & \dfrac{h^k}{k!}\,x^{(k)}(t) & + & o(h^k) \\ y(t+h) & = & y(t) & + & h\,y'(t) & + & \dfrac{h^2}{2}\,y''(t) & + & \dfrac{h^3}{3!}\,y^{(3)}(t) & + \ \ldots \ + & \dfrac{h^k}{k!}\,y^{(k)}(t) & + & o(h^k) \end{array} \right.$$

$$\Rightarrow \left\{ \begin{array}{lllllll} x(t+h) & = & x(t) & + & \dfrac{h^k}{k!}\,x^{(k)}(t) & + & o(h^k) \\ y(t+h) & = & y(t) & + & \dfrac{h^k}{k!}\,y^{(k)}(t) & + & o(h^k) \end{array} \right\} \iff C(t+h) = C(t) + \frac{h^k}{k!}\,C^{(k)}(t) + o(h^k)$$

Au voisinage de $C(t)$, h reste petit, et en négligeant le terme $o(h^k)$, alors $C(t+h) \simeq C(t) + \dfrac{h^k}{k!}\,C^{(k)}(t)$ donc proche de la droite passant par $C(t)$ et de vecteur directeur égal à $C^{(k)}(t)$. $\square$

Pour une courbe paramétrée polynomiale de degré $d \geq 1$, pour tout $t \in \mathbb{R}$, il existe toujours un entier $k \geq 1$ tel que $C^{(k)}(t)$ est non nulle, car la dérivée d-ième $C^{(d)}(t)$ est non nulle (et constante pour tout $t \in \mathbb{R}$).

• **Exemple 2** : soit la courbe $C(t) = \begin{pmatrix} t^5 \\ t^3 \end{pmatrix}$ pour $t \in \mathbb{R}$.

$$C'(t) = \begin{pmatrix} 5\,t^4 \\ 3\,t^2 \end{pmatrix},\ C''(t) = \begin{pmatrix} 20\,t^3 \\ 6\,t \end{pmatrix},\ C^{(3)}(t) = \begin{pmatrix} 60\,t^2 \\ 6 \end{pmatrix},\ C^{(4)}(t) = \begin{pmatrix} 120\,t \\ 0 \end{pmatrix},\ C^{(5)}(t) = \begin{pmatrix} 120 \\ 0 \end{pmatrix}$$

En $t = 0$, on a

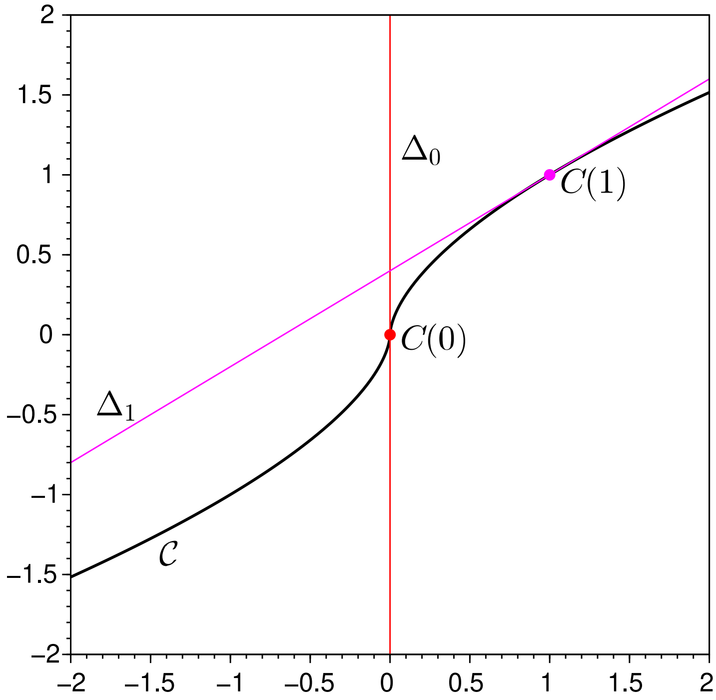
$$C'(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}\ ,\ C''(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}\ ,\ C^{(3)}(0) = \begin{pmatrix} 0 \\ 6 \end{pmatrix}$$

donc la direction de la droite tangente en $t = 0$ est donnée par le vecteur $C^{(3)}(0)$ qui est non nul et vertical :

$$\Delta_0 = \left\{ \begin{pmatrix} \bar{x}(u) & = & x(0) + u\,x^{(3)}(0) & = & 0 \\ \bar{y}(u) & = & y(0) + u\,y^{(3)}(0) & = & 6\,u \end{pmatrix} \ ,\ u \in \mathbb{R} \right\}$$

En $t = 1$, on a $C'(1) = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$ qui est donc la direction de la droite tangente en $t = 1$:

$$\Delta_1 = \left\{ \begin{pmatrix} \bar{x}(v) = x(1) + v\,x'(1) = 5\,v + 1 \\ \bar{y}(v) = y(1) + v\,y'(1) = 3\,v + 1 \end{pmatrix} \ ,\ v \in \mathbb{R} \right\}$$



Courbe paramétrée $C(t) = \begin{pmatrix} t^5 \\ t^3 \end{pmatrix}$ et droites tangentes Δ_0 et Δ_1

Exercice 1

a) pour la courbe $\mathcal{C} = \left\{ C(t) = \begin{pmatrix} x(t) = 2\,t \\ y(t) = 3\,t^2 - 2\,t \end{pmatrix} \;,\; t \in [0,1] \right\}$
calculer les équations paramétrées des droites Δ_0 et Δ_1 puis le point $I = \Delta_0 \cap \Delta_1$, intersection de ces deux droites. Tracer les points $C(0)$, $C(1)$ et I, les deux droites Δ_0, Δ_1, puis l'allure de la courbe $\mathcal{C}$.

b) Même exercice avec $\mathcal{C} = \left\{ C(t) = \begin{pmatrix} x(t) = 2\,t^2 \\ y(t) = 2\,t^2 - 2\,t + 1 \end{pmatrix} \;,\; t \in [0,1] \right\}$

• Un polynome p de degré d s'exprime à l'aide de $d+1$ coefficients q_k réels.
Par exemple dans la base des monômes $\{1, t, t^2, \ldots, t^d\}$, on a :

$$p(t) = \sum_{k=0}^{d} q_k\, t^k$$

Une courbe $\mathcal{C}$ dans $\mathbb{R}^2$, polynomiale de degré d s'exprime à l'aide de $d+1$ coefficients $Q_k \in \mathbb{R}^2$:

$$\mathcal{C} = \left\{ C(t) = \sum_{k=0}^{d} Q_k\, t^k \right\}$$

l'expression ci-dessus utilisant la ***base des monômes*** $\{1, t, t^2, \ldots, t^d\}$ avec $Q_k = \dfrac{1}{k!} C^{(k)}(0)$.

• **Exemple 3** La courbe polynomiale

$$\mathcal{C} = \left\{ \begin{pmatrix} x(t) & = & t^3 & = & t^3 + 0\,t^2 + 0\,t + 0 \\ y(t) & = & (1-t)^3 & = & -t^3 + 3\,t^2 - 3\,t + 1 \end{pmatrix} \;,\; t \in [0,1] \right\}$$

de degré 3 s'exprime avec 4 coefficients de $\mathbb{R}^2$:

$$C(t) = \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}}_{Q_3} t^3 + \underbrace{\begin{pmatrix} 0 \\ 3 \end{pmatrix}}_{Q_2} t^2 + \underbrace{\begin{pmatrix} 0 \\ -3 \end{pmatrix}}_{Q_1} t + \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{Q_0}$$

2 - Courbes de Bézier

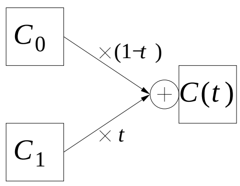
Dans la base des monômes, l'expression d'une courbe paramétrée polynomiale s'écrit à l'aide de coefficients Q_k de nature (géométrique) différente : $Q_0 = C(0)$ est le point de la courbe correspondant à $t = 0$, $Q_1 = C'(0)$ est le "vecteur-vitesse" de la courbe correspondant à $t = 0$, $Q_2 = \frac{1}{2}C''(0)$ est le "vecteur-accélération" de la courbe correspondant à $t = 0$, ...

Les ***courbes de Bézier*** de degré d sont des courbes polynomiales dont l'expression analytique utilisent une autre base que la base des monômes $\{1, t, t^2, \ldots, t^d\}$, avec des "points-coefficients" C_k de même nature (géométrique).

Principe de l'expression d'une courbe de Bézier

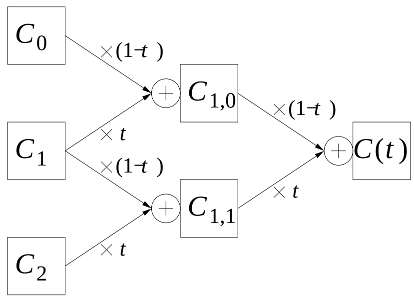
Les schémas ci-dessous montrent le principe de calcul du point d'une courbe de Bézier correspondant au paramètre $t \in I = [0, 1]$.

— cas $d = 1$: courbe $\mathcal{C}$ définie par deux points C_0 et C_1 :



$$\Rightarrow C(t) = C_0 \times (1 - t) + C_1 \times t$$

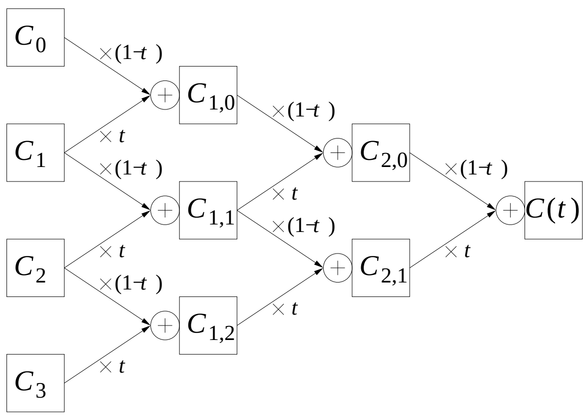
— cas $d = 2$: courbe $\mathcal{C}$ définie par trois points C_0, C_1 et C_2 :



$$\left\{\begin{array}{lllllllll} C_{1,0} & = & C_0 & \times & (1-t) & + & C_1 & \times & t \\ C_{1,1} & = & C_1 & \times & (1-t) & + & C_2 & \times & t \end{array}\right.$$

$$\Rightarrow C(t) = C_0 \times \ (1-t)^2 \ + \ C_1 \times 2\,t\,(1-t) \ + \ C_2 \times t^2$$

— cas $d = 3$: courbe $\mathcal{C}$ définie par quatre points C_0, C_1, C_2 et C_3 :



$$\left\{\begin{array}{lllllllll} C_{2,0} & = & C_0 & \times & (1-t)^2 & + & C_1 & \times & 2\,t\,(1-t) & + & C_2 & \times & t^2 \\ C_{2,1} & = & C_1 & \times & (1-t)^2 & + & C_2 & \times & 2\,t\,(1-t) & + & C_3 & \times & t^2 \end{array}\right.$$

$$\Rightarrow C(t) = C_0 \times \ (1-t)^3 \ + \ C_1 \times 3\,t\,(1-t)^2 \ + \ C_2 \times 3\,t^2\,(1-t) \ + \ C_3\,t^3$$

• **Definition 4** Les ***courbes de Bézier de degré*** d sont des courbes paramétrées polynomiales de degré d exprimées non pas avec la base des monômes habituelle $\{1, t, t^2, \ldots, t^d\}$ mais avec une autre base de fonctions polynomiales appelée ***base de Bernstein***.

L'expression paramétrée d'une courbe de Bézier $\mathcal{B}$ de degré d est :

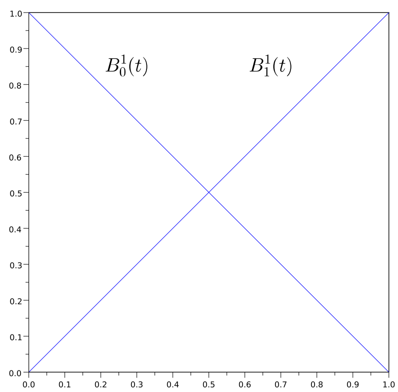
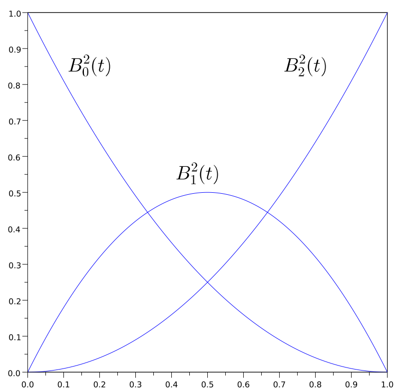
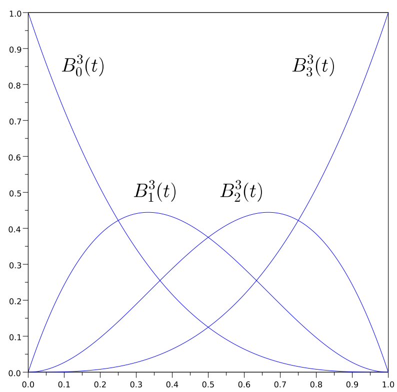
$$\mathcal{B} = \left\{ C(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \displaystyle\sum_{i=0}^{d} x_i B_i^d(t) \\ \displaystyle\sum_{i=0}^{d} y_i B_i^d(t) \end{pmatrix} = \sum_{i=0}^{d} C_i B_i^d(t) \quad , \quad t \in [0,1] \right\}$$

avec $C_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ et $B_i^d(t) = \begin{pmatrix} d \\ i \end{pmatrix} t^i (1-t)^{d-i} = \dfrac{d!}{i!(d-i)!}\, t^i (1-t)^{d-i}$, $0 \leq i \leq d$

L'ensemble des $d+1$ fonctions $\{B_i^d(t)\}_{0 \leq i \leq d}$ est la ***base de Bernstein*** de degré d.

L'ensemble des $d+1$ points $\{C_0, C_1, \ldots, C_d\}$ est appelé ***polygone de contrôle*** noté $[C_0, C_1, \ldots, C_d]$, chaque point C_i est appelé ***point de contrôle***.

Une courbe de Bézier dont les points de contrôle sont les points $\{C_0, C_1, \ldots, C_d\}$ est notée $\mathcal{B}[C_0, C_1, \ldots, C_d]$.

Base de Bernstein ($d=1$)	Base de Bernstein ($d=2$)	Base de Bernstein ($d=3$)
$\begin{aligned} B_0^1(t) &= 1 \times (1-t)^1 \times t^0 \\ &= 1-t \\ B_1^1(t) &= 1 \times (1-t)^0 \times t^1 \\ &= t \end{aligned}$	$\begin{aligned} B_0^2(t) &= 1 \times (1-t)^2 \times t^0 \\ &= (1-t)^2 \\ B_1^2(t) &= 2 \times (1-t)^1 \times t^1 \\ &= 2\,t\,(1-t) \\ B_2^2(t) &= 1 \times (1-t)^0 \times t^2 \\ &= t^2 \end{aligned}$	$\begin{aligned} B_0^3(t) &= 1 \times (1-t)^3 \times t^0 \\ &= (1-t)^3 \\ B_1^3(t) &= 3 \times (1-t)^2 \times t^1 \\ &= 3\,t\,(1-t)^2 \\ B_2^3(t) &= 3 \times (1-t)^1 \times t^2 \\ &= 3\,t^2\,(1-t) \\ B_3^3(t) &= 1 \times (1-t)^0 \times t^3 \\ &= t^3 \end{aligned}$
		

Cette écriture des courbes polynomiales est due à deux ingénieurs français Pierre Bézier (Renault) et Paul de Casteljau (Citroen) permettant d'avoir des propriétés géométriques sur les points (coefficients) C_i.

Les deux propriétés suivantes sont les propriétés géométrqies principales des courbes de Bézier.

• **Propriété 2** Pour tout i entre 0 et d et pour tout $t \in [0,1]$, on a $B_i^d(t) \geq 0$ et $\displaystyle\sum_{i=0}^{d} B_i^d(t) = 1$:

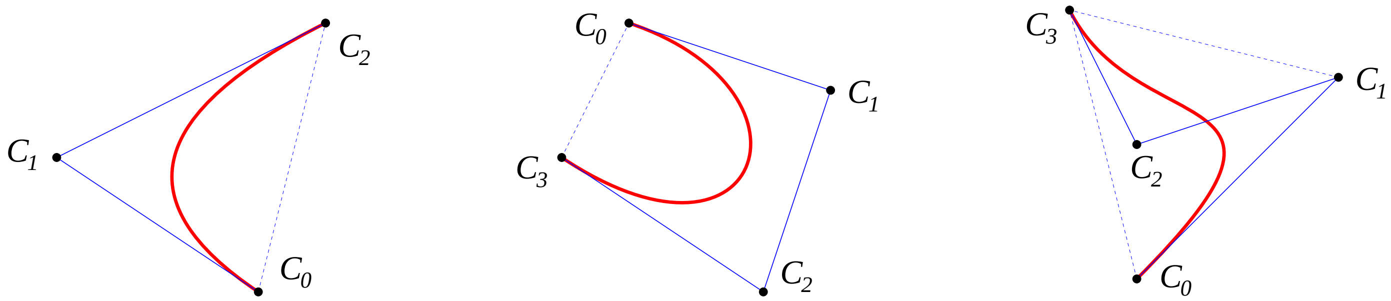
$$t \in [0,1] \Rightarrow t \geq 0 \text{ et } 1-t \geq 0 \Rightarrow B_i^d(t) = \binom{d}{i}\, t^i (1-t)^{d-i} \geq 0$$

$$1 = 1^d = \big(t + (1-t)\big)^d = \sum_{i=0}^{d} \binom{d}{i}\, t^i (1-t)^{d-i} = \sum_{i=0}^{d} B_i^d(t)$$

La conséquence de ceci est que l'expression

$$C(t) = \sum_{i=0}^{d} C_i\, B_i^d(t)$$

est une ***combinaison convexe*** des points C_i, et donc tout point de la courbe (et la courbe en totalité) est compris dans l'enveloppe convexe de ses points de contrôle.



Exercice 2

Dans cet exercice, on suppose les points de contrôle C_i distincts entre eux.

a) Soit $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2] = \big\{ C(t) = (1-t)^2\, C_0 + 2\,t\,(1-t)\, C_1 + t^2\, C_2\ ,\ \ t \in [0,1] \big\}$

— calculer $C(0)$ et $C(1)$,

— déterminer l'expression de la dérivée $C'(t)$ et en déduire l'expression de $C'(0)$ et $C'(1)$.

b) Même exercice avec

$\mathcal{C} = \mathcal{B}[C_0, C_1, C_2, C_3] = \big\{ C(t) = (1-t)^3\, C_0 + 3\,t\,(1-t)^2\, C_1 + 3\,t^2\,(1-t)\, C_2 + t^3\, C_3\ ,\ \ t \in [0,1] \big\}$

• **Propriété 3** (corollaire de la propriété 1 - tangente en un point d'une courbe)

Une courbe de Bézier $\mathcal{B}[C_0, C_1, \ldots, C_d]$ de degré d vérifie les propriétés géométriques suivantes :

— le point initial est le premier point de contrôle C_0,
la direction de la tangente en ce point est donnée par le vecteur $\overrightarrow{C_0\, C_k}$ formé des deux premiers points de contrôle différents,

— le point final est le dernier point de contrôle C_d,
la direction de la tangente en ce point est donnée par le vecteur $\overrightarrow{C_{d-k}\, C_d}$ formé des deux derniers points de contrôle différents.

• En pratique, on se limite à d petit ($d \leq 5$), le cas $d = 3$ étant le cas le plus utilisé.
Dans ce cours, on se limitera à $d \leq 3$, le cas $d = 3$ correspondant au cas cubique géré par le format PostScript.

Bézier de degré 1 $\mathcal{B}[C_0, C_1]$

$$\forall t \in [0,1]\ ,\ C(t) = (1-t)\, C_0 + t\, C_1$$

Une courbe de Bézier de degré 1 est un segment de droite (segment reliant les points C_0 et C_1).

Bézier de degré 2 (Bézier quadratique) $\mathcal{B}[C_0, C_1, C_2]$

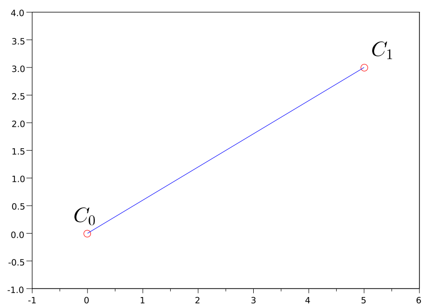
$$\forall t \in [0,1]\ ,\ C(t) = (1-t)^2\, C_0 + 2\, t(1-t)\, C_1 + t^2\, C_2$$

Une (courbe de) Bézier quadratique est un arc de parabole (éventuellement dégénérée en un segment de droite).

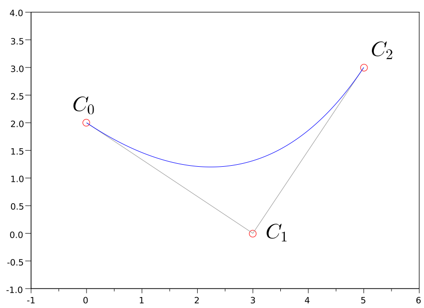
Bézier de degré 3 (Bézier cubique) $\mathcal{B}[C_0, C_1, C_2, C_3]$

$$\forall t \in [0,1]\ ,\ C(t) = (1-t)^3\, C_0 + 3\, t\, (1-t)^2\, C_1 + 3\, t^2\, (1-t)\, C_2 + t^3\, C_3$$

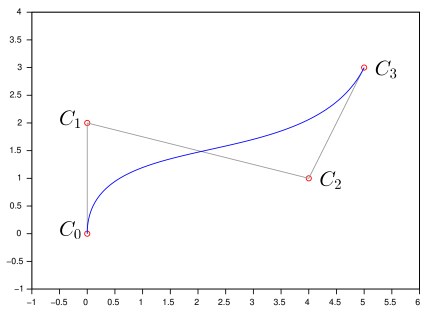
Une courbe de Bézier cubique permet d'obtenir un arc de courbe avec un point d'inflexion.



Bézier de degré 1



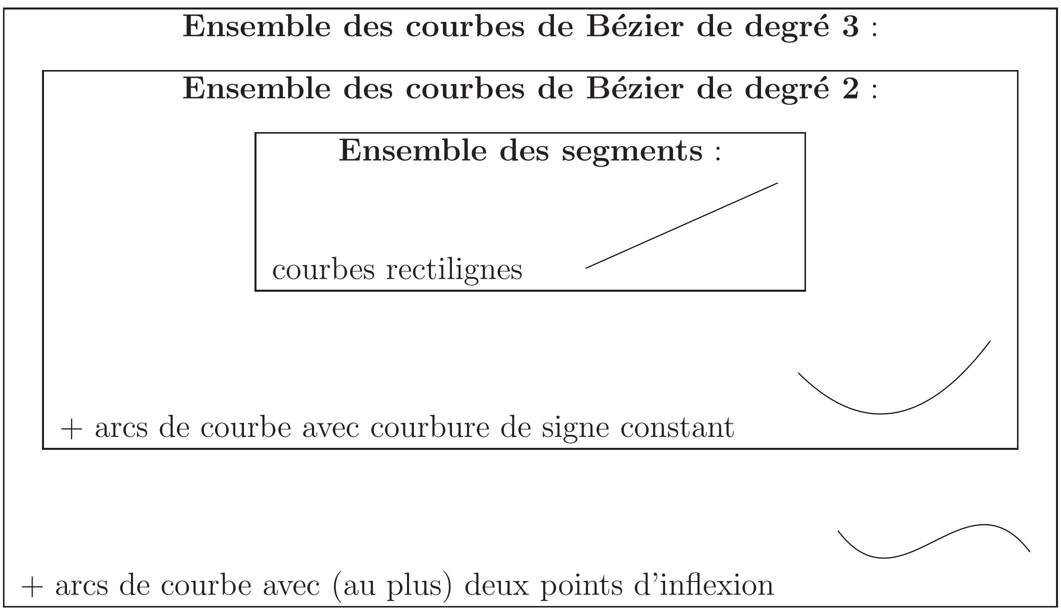
Bézier de degré 2



Bézier de degré 3

• **Propriété 4**

Un polynôme de degré d peut s'exprimer sous forme d'un polynôme de degré $d+1$. Donc l'ensemble des polynômes de degré d est inclus dans l'ensemble des polynômes de degré $d+1$.



Elévation de degré
On peut donc exprimer une Bézier $\mathcal{B}[C_0, \ldots, C_d]$ de degré d sous forme d'une Bézier $\mathcal{B}[\bar{C}_0, \ldots, \bar{C}_{d+1}]$ de degré $d+1$. Par exemple, pour exprimer une Bézier $\mathcal{B}[C_0, C_1]$ de degré 1 sous forme d'une Bézier $\mathcal{B}[\bar{C}_0, \bar{C}_1, \bar{C}_2]$ de degré 2, on procède ainsi :

$$\mathcal{B}[C_0, C_1] = \Big\{ C(t) = C_0\,(1-t) + C_1\,t\ ,\ t \in [0;1] \Big\}$$

$$C_0\,(1-t) + C_1\,t = C_0\,(1-t)\,\big[(1-t)+t\big] + C_1\,t\,\big[(1-t)+t\big]$$

$$= C_0\,(1-t)^2 + C_0\,(1-t)\,t + C_1\,t\,(1-t) + C_1\,t^2 = C_0\,(1-t)^2 + \frac{C_0+C_1}{2}\,2\,t\,(1-t) + C_1\,t^2$$

$$\Rightarrow \qquad \boxed{\bar{C}_0 = C_0\ ,\ \bar{C}_1 = (C_0+C_1)/2\ ,\ \bar{C}_2 = C_1}$$

L'opération qui permet de passer de $[C_0, C_1]$ à $[\bar{C}_0, \bar{C}_1, \bar{C}_2]$ est appelé *élévation de degré* (ici du degré 1 au degré 2). De même, on peut exprimer une Bézier $\mathcal{B}[C_0, C_1, C_2]$ de degré 2 sous forme d'une Bézier $\mathcal{B}[\bar{C}_0, \bar{C}_1, \bar{C}_2, \bar{C}_3]$ de degré 3 (*élévation de degré* du degré 2 au degré 3).

Exercice 3

Soit $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2]$ une courbe de Bézier de degré 2. Déterminer, en fonction de C_0, C_1 et C_2, l'expression des quatre points $\bar{C}_0, \bar{C}_1, \bar{C}_2, \bar{C}_3$ tels que $\mathcal{C} = \mathcal{B}[\bar{C}_0, \bar{C}_1, \bar{C}_2, \bar{C}_3]$.

L'*élévation de degré* (du degré 2 au degré 3) sera utilisée pour exporter des courbes de Bézier de degré 2 en PostScript.

Export de Bézier au format PostScript
Le format PostScript permet la définition de courbes de Bézier de degré 3 à l'aide des instructions `moveto` et `curveto`. Pour des courbes de Bézier de degré 2, il faut donc d'abord les exprimer sous forme de courbes de Bézier de degré 3 avant de les exporter en PostScript.

• **Exemple 4** exporter en PostScript la courbe composite formée des deux courbes de Bézier
$\mathcal{B}_1 = \mathcal{B}\left[\begin{pmatrix}6\\7\end{pmatrix}, \begin{pmatrix}5\\9\end{pmatrix}, \begin{pmatrix}1\\8\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}\right]$ et $\mathcal{B}_2 = \mathcal{B}\left[\begin{pmatrix}0\\2\end{pmatrix}, \begin{pmatrix}9\\5\end{pmatrix}, \begin{pmatrix}6\\8\end{pmatrix}\right]$:

1. la courbe $\mathcal{B}_1$ étant de degré 3, l'export en PostScript se fait directement (instruction `moveto` pour le premier point de contrôle, puis instruction `curveto` pour les trois points de contrôle suivants) :
   $\boxed{6\ 7\ \textbf{moveto}\quad 5\ 9\ \ 1\ 8\ \ 0\ 2\ \textbf{curveto}}$
2. la courbe $\mathcal{B}_2$ étant de degré 2, il faut d'abord l'exprimer sous forme la forme d'une courbe de Bézier de degré 3 par élévation de degré : $\mathcal{B}_2 = \mathcal{B}\left[\begin{pmatrix}0\\2\end{pmatrix}, \begin{pmatrix}6\\4\end{pmatrix}, \begin{pmatrix}8\\6\end{pmatrix}, \begin{pmatrix}6\\8\end{pmatrix}\right]$
   L'export PostScript est donc : $\boxed{0\ 2\ \textbf{moveto}\quad 6\ 4\ \ 8\ 6\ \ 6\ 8\ \textbf{curveto}}$
3. Le premier point de contrôle de $\mathcal{B}_2$ étant le dernier point de contrôle de $\mathcal{B}_1$, l'instruction `moveto` pour $\mathcal{B}_2$ peut être omise (elle doit l'être en mode `fill` pour l'export EPS des contours simplifiés par courbes de Bézier). L'export PostScript pour $\mathcal{B}_1$ et $\mathcal{B}_2$ est donc :
   $\boxed{6\ 7\ \textbf{moveto}\quad 5\ 9\ \ 1\ 8\ \ 0\ 2\ \textbf{curveto}\quad 6\ 4\ \ 8\ 6\ \ 6\ 8\ \textbf{curveto}}$

Ainsi pour la tâche de simplification par courbes de Bézier de degré 2, lors de l'export au format PostScript, il faudra convertir chaque courbe de Bézier de degré 2 en une courbe de Bézier de degré 3.

Exercice 4

Exporter en PostScript la courbe composite formée des 3 coubes de Bézier suivantes :
$\mathcal{B}_1 = \mathcal{B}\left[\begin{pmatrix}1\\5\end{pmatrix}, \begin{pmatrix}7\\2\end{pmatrix}, \begin{pmatrix}13\\8\end{pmatrix}\right]$, $\mathcal{B}_2 = \mathcal{B}\left[\begin{pmatrix}13\\8\end{pmatrix}, \begin{pmatrix}7\\8\end{pmatrix}, \begin{pmatrix}7\\5\end{pmatrix}\right]$, $\mathcal{B}_3 = \mathcal{B}\left[\begin{pmatrix}7\\5\end{pmatrix}, \begin{pmatrix}4\\8\end{pmatrix}, \begin{pmatrix}1\\5\end{pmatrix}\right]$

Exercices supplémentaires

Exercice 5 *(courbe polynomiale et courbe de Bézier correspondante)*

Soit la courbe paramétrée polynomiale de degré 2 suivante :

$$\mathcal{C} = \left\{ C(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} -8\,t^2 + 8\,t \\ 2\,t^2 + 2 \end{pmatrix}, t \in [0,1] \right\}$$

a) Calculer $C(0)$ puis une équation paramétrée de la droite Δ_0, tangente à la courbe $\mathcal{C}$ au point $C(0)$.

b) Calculer $C(1)$ puis une équation paramétrée de la droite Δ_1, tangente à la courbe $\mathcal{C}$ au point $C(1)$.

c) Calculer le point I intersection des deux droites Δ_0 et Δ_1.

d) Tracer sur un même graphe, les points $C(0)$ et $C(1)$, les deux droites Δ_0 et Δ_1 et l'allure de la courbe $\mathcal{C}$.

e) Quels sont les trois points de contrôle C_0, C_1 et C_2 tel que $\mathcal{B}[C_0, C_1, C_2] = \mathcal{C}$?

Exercice 6 *(courbe de Bézier, expression dans la base des monômes et droites tangentes aux extrémités)*

Soit les 3 points $C_0 = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$, $C_1 = \begin{pmatrix} 8 \\ 8 \end{pmatrix}$ et $C_2 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$, et soit la courbe de Bézier

$$\mathcal{C} = \mathcal{B}[C_0, C_1, C_2] = \left\{ C(t) = C_0\,(1-t)^2 + C_1\,2\,t\,(1-t) + C_2\,t^2,\ t \in [0,1] \right\}$$

a) Calculer le point $C(1/2)$.

b) Tracer sur un même graphe, le polygone de contrôle $[C_0, C_1, C_2]$, le point $C(1/2)$ et l'allure de la courbe $\mathcal{C}$.

c) Ecrire l'expression de $C(t)$ dans la base des monomes.

d) Calculer une équation paramétrée de la droite Δ_0, tangente à la courbe $\mathcal{C}$ au point correspondant à $t = 0$.

e) Calculer une équation paramétrée de la droite Δ_1, tangente à la courbe $\mathcal{C}$ au point correspondant à $t = 1$.

f) Calculer le point I intersection des deux droites Δ_0 et Δ_1.

Exercice 7 *(points de contrôle d'une courbe de Bézier et propriétés géométriques)*

Soit la courbe de Bézier $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2]$ de degré 2 représentée ci-dessous vérifie les propriétés suivantes :

— le point initial de la courbe est le point $A = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$, la tangente à la courbe $\mathcal{C}$ en ce point A est horizontale,

— le point final de la courbe est le point $B = \begin{pmatrix} 7 \\ 1 \end{pmatrix}$, la tangente à la courbe $\mathcal{C}$ en ce point A est verticale.

Quelles sont les coordonnées des 3 points de contrôle C_0, C_1 et C_2 ?

Exercice 8 *(calcul d'une courbe de Bézier passant par des points)*

Soient les 3 points $P_0 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$, $P_1 = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ et $P_2 = \begin{pmatrix} 8 \\ 6 \end{pmatrix}$.
Déterminer les trois points de contrôle de la courbe de Bézier de dégré 2

$$\mathcal{C} = \mathcal{B}[C_0, C_1, C_2] = \left\{ C(t) = C_0\,(1-t)^2 + C_1\,2\,t\,(1-t) + C_2\,t^2 \ ,\ t \in [0,1] \right\}$$

tel que $C(0) = P_0$, $C(1/2) = P_1$, $C(1) = P_2$.
Tracer sur un même graphe les points P_0, P_1, P_2, le polygone de contrôle $[C_0, C_1, C_2]$ et l'allure de la courbe $\mathcal{C}$.

Exercice 9 *(courbe composite formée de courbes de Bézier)*

La figure 1 ci-dessous représente une courbe $\mathcal{C}$.
Cette courbe $\mathcal{C}$ est formée de quatre courbes de Bézier $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ et $\mathcal{C}_4$, chaque courbe de Bézier $\mathcal{C}_k$ est de degré 2, commence au point P_{k-1} et se termine au point P_k (cf. figure 2).

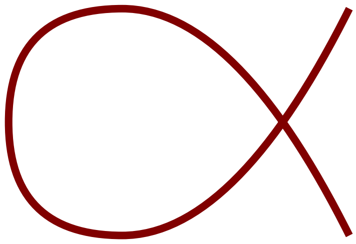


Figure 1

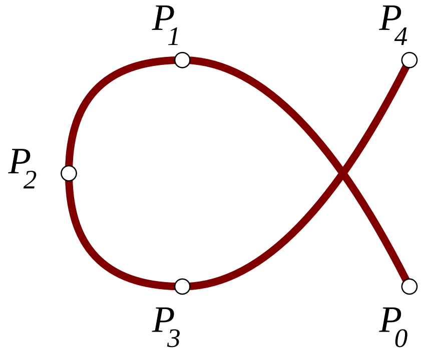


Figure 2

Le tableau ci-dessous donne pour chaque point P_k, ses coordonnées, ainsi qu'une équation paramétrée de la droite Δ_k tangente à la courbe $\mathcal{C}$ au point P_k.

$P_0 = \begin{pmatrix} 9 \\ 0 \end{pmatrix}$	$\Delta_0 = \left\{ M_0(t_0) = \begin{pmatrix} 9 - t_0 \\ 2\,t_0 \end{pmatrix}, t_0 \in \mathbb{R} \right\}$
$P_1 = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$	$\Delta_1 = \left\{ M_1(t_1) = \begin{pmatrix} t_1 \\ 6 \end{pmatrix}, t_1 \in \mathbb{R} \right\}$
$P_2 = \begin{pmatrix} 0 \\ 3 \end{pmatrix}$	$\Delta_2 = \left\{ M_2(t_2) = \begin{pmatrix} 0 \\ t_2 \end{pmatrix}, t_2 \in \mathbb{R} \right\}$
$P_3 = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$	$\Delta_3 = \left\{ M_3(t_3) = \begin{pmatrix} t_3 \\ 0 \end{pmatrix}, t_3 \in \mathbb{R} \right\}$
$P_4 = \begin{pmatrix} 9 \\ 6 \end{pmatrix}$	$\Delta_4 = \left\{ M_4(t_4) = \begin{pmatrix} t_4 + 6 \\ 2\,t_4 \end{pmatrix}, t_4 \in \mathbb{R} \right\}$

a) Déterminer les points de contrôle de chaque courbe de Bézier $\mathcal{C}_k$.

b) Ecrire les instructions PostScript permettant de décrire la courbe $\mathcal{C}$.

Exercice 10 *(élévation de degré)*

a) Soit la courbe de Bézier $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2]$ de degré 2 avec
$C_0 = \begin{pmatrix} 0 \\ 5 \end{pmatrix}$, $C_1 = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$ et $C_2 = \begin{pmatrix} 9 \\ 8 \end{pmatrix}$.
Déterminer les quatre points de controle D_0, D_1, D_2 et D_3 tels que $\mathcal{B}[D_0, D_1, D_2, D_3] = \mathcal{C}$.

b) Soit la courbe de Bézier $\mathcal{E} = \mathcal{B}[E_0, E_1]$ de degré 1 avec $E_0 = \begin{pmatrix} -5 \\ 0 \end{pmatrix}$ et $E_1 = \begin{pmatrix} 7 \\ 6 \end{pmatrix}$.
Déterminer les trois points de controle F_0, F_1 et F_2 tels que $\mathcal{B}[F_0, F_1, F_2] = \mathcal{E}$ puis les quatre points de controle G_0, G_1, G_2 et G_3 tels que $\mathcal{B}[G_0, G_1, G_2, G_3] = \mathcal{E}$.

Exercice 11 *(courbe de Bézier et droite tangente en $t = 1/2$)*

Soient les 3 points $C_0 = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$, $C_1 = \begin{pmatrix} 8 \\ 2 \end{pmatrix}$ et $C_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ et la courbe de Bézier

$$\mathcal{C} = \mathcal{B}[C_0, C_1, C_2] = \left\{ C(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = C_0\,(1-t)^2 + C_1\,2\,t\,(1-t) + C_2\,t^2\ ,\ t \in [0,1] \right\}$$

a) Déterminer les expressions de $x(t)$ et $y(t)$ dans la base des monômes.

b) Déterminer la droite tangente Δ à la courbe $\mathcal{C}$ au point de paramètre $t = 1/2$ ainsi qu'une équation paramétrée pour cette droite.

c) Tracer le polygone de contrôle $[C_0, C_1, C_2]$, le point $C(1/2)$, la droite Δ et l'allure de la courbe $\mathcal{C}$.

Exercice 12 *(passage de la base des monômes à la base de Bernstein)*

a) Soit la courbe de l'exercice 1-a : $\mathcal{C} = \left\{ C(t) = \begin{pmatrix} 2\,t \\ 3\,t^2 - 2\,t \end{pmatrix}\ ,\ t \in [0,1] \right\}$
En utilisant les formules de changement de base

$$\left\{ \begin{array}{ccccccccccc} 1 & = & 1 & \times & B_0^2(t) & + & 1 & \times & B_1^2(t) & + & 1 & \times & B_2^2(t) \\ t & = & 0 & \times & B_0^2(t) & + & \frac{1}{2} & \times & B_1^2(t) & + & 1 & \times & B_2^2(t) \\ t^2 & = & 0 & \times & B_0^2(t) & + & 0 & \times & B_1^2(t) & + & 1 & \times & B_2^2(t) \end{array} \right.$$

écrire $C(t)$ sous la forme $C_0\,B_0^2(t) + C_1\,B_1^2(t) + C_2\,B_2^2(t)$ et en déduire les trois points de contrôle C_0, C_1 et C_2.

b) Même exercice avec la courbe de l'exercice 1-b : $\mathcal{C} = \left\{ C(t) = \begin{pmatrix} 2\,t^2 \\ 2\,t^2 - 2\,t + 1 \end{pmatrix}\ ,\ t \in [0,1] \right\}$

Exercice 13 *(passage de la base de Bernstein à la base des monômes)*

Déterminer dans la base des monômes, les expressions paramétrées des courbes de Bézier suivantes :

$$\mathcal{C}_1 = \mathcal{B}\left[\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 5 \end{pmatrix} \right] \qquad \mathcal{C}_2 = \mathcal{B}\left[\begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 6 \\ 0 \end{pmatrix} \right]$$

Exercice 14 *(principe de subdivision pour les courbes de Bézier)*

• Soient $C_0 = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$, $C_1 = \begin{pmatrix} 8 \\ 0 \end{pmatrix}$ et $C_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ trois points du plan et la courbe de Bézier $\mathcal{C}$:

$$\mathcal{C} = \mathcal{B}[C_0, C_1, C_2] = \{C(t) = C_0\,(1-t)^2 + C_1\,2\,t\,(1-t) + C_2\,t^2\ ,\ t \in [0,1]\}$$

a) Calculer l'expression de $C(t)$ dans la base des monômes.

b) Calculer le point D_0 milieu du segment $[C_0, C_1]$, le point D_1 milieu du segment $[C_1, C_2]$, et le point E_0 milieu du segment $[D_0, D_1]$.

• On considère les deux courbes de Bézier $\mathcal{C}_1 = \mathcal{B}[C_0, D_0, E_0]$ et $\mathcal{C}_2 = \mathcal{B}[E_0, D_1, C_2]$:

$$\mathcal{C}_1 = \{C_1(u) = C_0\ B_0^2(u) + D_0\ B_1^2(u) + E_0\ B_2^2(u)\ ,\ u \in [0,1]\}$$

$$\mathcal{C}_2 = \{C_2(v) = E_0\ B_0^2(v) + D_1\ B_1^2(v) + C_2\ B_2^2(v)\ ,\ v \in [0,1]\}$$

c) Calculer l'expression de $C_1(u)$ dans la base des monômes,
puis faire le changement de variable $u = 2\,t \quad (u \in [0,1] \iff t = u/2 \in [0, 1/2])$.

d) Calculer l'expression de $C_2(v)$ dans la base des monômes,
puis faire le changement de variable $v = 2\,t - 1 \quad (v \in [0,1] \iff t = (v+1)/2 \in [1/2, 1])$.

e) Que peut-on conclure des résultats de c) et d) ?

f) Tracer sur un même graphe, les polygones de contrôle $[C_0, C_1, C_2]$, $[C_0, D_0, E_0]$, $[E_0, D_1, C_2]$ et l'allure de la courbe $\mathcal{C}$.

Exercice 15 *(calcul d'une courbe de Bézier passant par des points)*

Soient les quatre points du plan

$$P_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 5 \\ 2 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 8 \\ 2 \end{pmatrix} \qquad P_3 = \begin{pmatrix} 9 \\ 0 \end{pmatrix}$$

et la courbe de Bézier $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2, C_3]$

$$\mathcal{C} = \left\{ C(t) = \sum_{i=0}^{3} C_i\, B_i^3(t)\ ,\ t \in [0,1] \right\}$$

tel que $C(0) = P_0$, $C(1/3) = P_1$, $C(2/3) = P_2$ et $C(1) = P_3$.

a) Sans faire aucun calcul, quels sont les points de contrôle C_0 et C_3 ?

b) Montrer que

$$\left\{ \begin{array}{rcl} 12\,C_1 + 6\,C_2 & = & 27\,P_1 - 8\,P_0 - P_3 \\ 6\,C_1 + 12\,C_2 & = & 27\,P_2 - P_0 - 8\,P_3 \end{array} \right\}$$

et en déduire les coordonnées des points C_1 et C_2.

c) Montrer que $\mathcal{C}$ est en fait une courbe de Bézier de degré 2.

Exercice 16 *(une autre propriété des courbes de Bézier de degré 2)*

a) Soit $C_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $C_1 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$ et $C_2 = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$.
Calculer une équation paramétrée de Δ, droite tangente à $\mathcal{C}$ au point correspondant à $t = 1/2$.
Montrer que le vecteur $\overrightarrow{C_0C_2}$ est un vecteur directeur de Δ.

b) Cas général : montrer que si on a une courbe de Bézier de degré 2 $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2]$ avec $C_0 \neq C_2$ alors le vecteur $\overrightarrow{C_0C_2}$ est un vecteur directeur de Δ, droite tangente à $\mathcal{C}$ au point correspondant à $t = 1/2$.

Exercice 17 *(approximation d'un quart de cercle par une courbe de Bézier de degré 3)*

Dans cet exercice, a désigne un réel strictement positif.
Soit la courbe de Bézier de degré 3

$$\mathcal{C} = \mathcal{B}[C_0, C_1, C_2, C_3] = \left\{ C(t) = (1-t)^3\, C_0 + 3\,t\,(1-t)^2\, C_1 + 3\,t^2\,(1-t)\, C_2 + t^3\, C_3 \ ,\ t \in [0,1] \right\}$$

$$\text{avec } C_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \ ,\ C_1 = \begin{pmatrix} 1 \\ a \end{pmatrix} \ ,\ C_2 = \begin{pmatrix} a \\ 1 \end{pmatrix} \ ,\ C_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

a) Pour quelle valeur de a, la courbe $\mathcal{C}$ est-elle aussi une courbe de Bézier de degré 2 ?
b) Soit le point $M = \begin{pmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{pmatrix}$ appartenant au cercle unité.
Pour quelle valeur de a, a-t-on $C(1/2) = M$?

Exercice 18 *(changement de base* monômes $\leftrightarrow$ Bernstein *et écriture matricielle)*

• Pour les polynomes de degré 2, on note $V_1(t) = \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix}$ le vecteur de la base des monômes, et $V_2(t) = \begin{pmatrix} (1-t)^2 \\ 2\,t\,(1-t) \\ t^2 \end{pmatrix}$ le vecteur de la base de Bernstein.

a) Déterminer la matrice carrée M de dimension 3 telle que $V_2(t) = M\, V_1(t)$.

b) Déterminer la matrice carrée M^{-1} de dimension 3, inverse de la matrice M , c'est à dire telle que $V_1(t) = M^{-1}\, V_2(t)$.

• Soit $\mathcal{C}$ la courbe polynomiale de degré 2 suivante :

$$\mathcal{C} = \left\{ C(t) = \begin{pmatrix} 1 - 3\,t + 2\,t^2 \\ 1 - 4\,t \end{pmatrix} \ ,\ t \in [0,1] \right\}$$

c) Déterminer la matrice A_1 de dimensions 2×3 telle que $C(t) = A_1\, V_1(t)$.

d) Déterminer la matrice A_2 de dimensions 2×3 telle que $C(t) = A_2\, V_2(t)$.

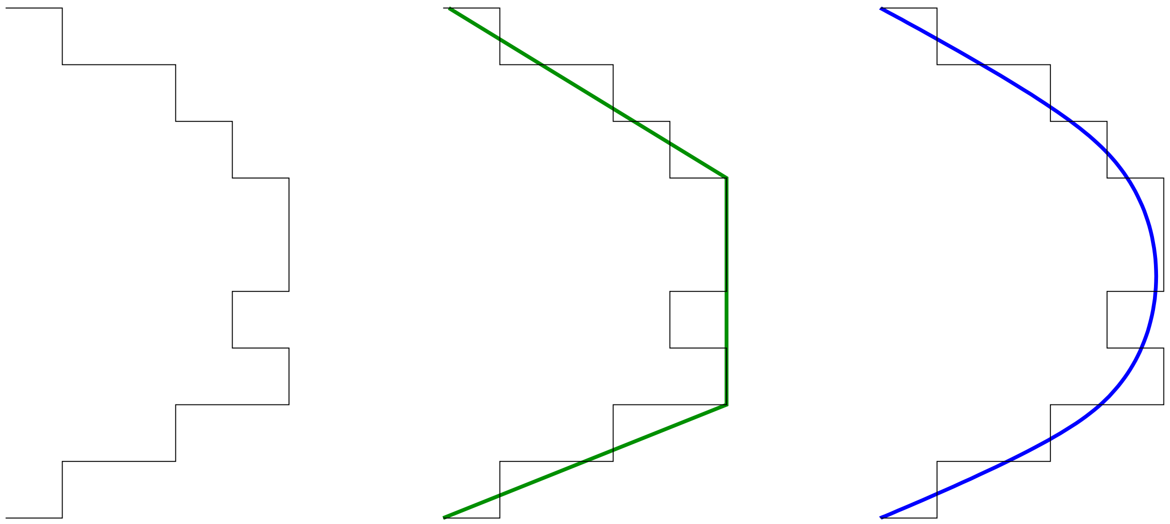
e) En déduire les trois points de controle C_0, C_1 et C_2 tels que $\mathcal{C} = \mathcal{B}[C_0, C_1, C_2]$.

Tâche 7 - Simplification de contour par courbe de Bézier

7.1 - Présentation

Les exemples suivants montrent que la simplification par segment n'est pas très efficace (nombre de segments / rendu visuel), et qu'il est parfois préférable d'utiliser des courbes lisses.

Simplification d'une ligne polygonale



Contour initial — Approximation par segment(s) — Approximation par courbe(s) lisse(s)

Simplification des contours d'une image avec la distance-seuil $d = 3$

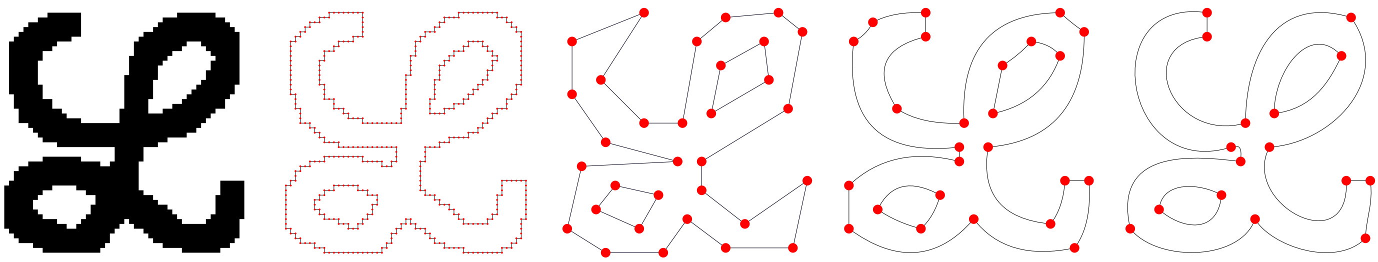


Image	Image initiale	Simplification par segment	Simplification par Bézier de degré 2	Simplification par Bézier de degré 3
50×50	3 contours	32 segments	25 courbes	16 courbes
2500 pixels	488 segments			

En général les courbes paramétrées de classe C^1 (dérivables et à dérivée continues) sont lisses (l'aspect visuel lisse est lié à la continuité de la tangente). Les courbes C^1 les plus simples sont les courbes polynomiales, de plus si le degré est supérieur ou égal à deux, on peut obtenir des courbes autres que des segments de droite. Donc on utilisera les courbes de Bézier (de degré 2 ou 3) permettant une manipulation géométrique des courbes polynomiales à l'aide des points de contrôle.

7.2 - Simplification de contour par courbe(s) de Bézier quadratiques (de degré 2)

Le procédé de simplification par courbes de Bézier reprend le principe de l'algorithme de Douglas-Peucker utilisé pour la simplification par segment mais en remplaçant la séquence de segments (courbes de Bézier de degré 1) par une séquence de courbes de Bézier de degré 2 $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_p\}$ avec :

— pour tout j, $1 \leq j \leq p$, la courbe $\mathcal{B}_j = \mathcal{B}[C_{j,0}, C_{j,1}, C_{j,2}]$,

— pour tout j, $1 \leq j \leq p-1$, il y a raccord continu entre la fin de la courbe $\mathcal{B}_j$ et le début de la courbe suivante $\mathcal{B}_{j+1}$, c'est à dire $C_{j+1,0} = C_{j,2}$,

L'algorithme de Douglas-Peucker pour la simplification par courbe de Bézier (de degré 2) est présenté en page 50.

Les différences par rapport à l'algorithme de Douglas-Peucker initial (page 28) sont :

— les séquences de segments par des séquences de (courbes de) Bézier,

— pour un contour initial `CONT` $= \{P_{j1}, \ldots, P_{j2}\}$, on l'approxime non pas par le segment $S = [P_{j1}, P_{j2}]$ mais une courbe de Bézier $\mathcal{B} = \mathcal{B}[C_0, C_1, C_2]$ avec le point initial $C_0 = P_{j1}$ et le point final $C_2 = P_{j2}$
$\rightarrow$ fonction `approx_bezier2`

— on remplace le calcul de la distance entre un point et un segment par une fonction calculant la distance entre un point et une courbe de Bézier de degré 2
$\rightarrow$ fonction `distance_point_bezier2`

Approximation d'une ligne polygonale par une courbe de Bézier de degré 2 (fonction `approx_bezier2`)

Dans la méthode de simplification par segment d'une séquence de points $\{P_{j1}, \ldots, P_{j2}\}$, le segment S est unique ($S = [P_{j1}, P_{j2}]$) puis on recherche le point P_k le plus éloigné de S pour savoir si le segment convient ou bien s'il faut diviser le problème en deux.

Pour simplifier une séquence de points $\{P_{j1}, \ldots, P_{j2}\}$ (avec $n = j2 - j1 \geq 1$) par une courbe de Bézier quadratique (de degré 2) $\mathcal{B} = \mathcal{B}[C_0, C_1, C_2]$, il faut que $C_0 = P_{j1}$ et $C_2 = P_{j2}$, par contre le point de contrôle C_1 peut être choisi d'une infinité de manière.
Cependant il faut le choisir convenablement en fonction des différents points du contour à simplifier, et le calcul de C_1 se fait ainsi :

A) Cas $n = 1$ (contour $\{P_{j1}, \ldots, P_{j2}\} = \{P_{j1}, P_{j1+1}\}$ réduit à deux points, soit un seul segment) :

$$C_1 = \frac{1}{2}(P_{j1} + P_{j2})$$

B) Cas $n \geq 2$ (contour $\{P_{j1}, \ldots, P_{j2}\}$ avec au moins 3 points) :

$$C_1 = \alpha \left(\sum_{i=1}^{n-1} P_{i+j1} \right) + \beta(P_{j1} + P_{j2}) \quad \text{avec} \quad \alpha = \frac{3\,n}{n^2 - 1} \quad \text{et} \quad \beta = \frac{1 - 2\,n}{2\,(n+1)}$$

IMPORTANT : dans les différentes formules, n est un entier, alors que α et β sont des réels
$\rightarrow$ il faut convertir n en réel avant de calculer α et β.

• Ce qui suit explique le choix pour le calcul de C_1.

A) Cas $n = 1$: il suffit de choisir $C_1 \in S = [P_{j1}, P_{j2}]$, et le choix naturel est $C_1 = (P_{j1} + P_{j2})/2$ correspondant à l'écriture du segment $S = [P_{j1}, P_{j2}]$ sous forme d'une courbe de Bézier de degré 2.

B) Cas $n \geq 2$: on cherche $\mathcal{B}[C_0, C_1, C_2]$ courbe de Bézier de degré 2 avec $C_0 = P_{j1}$, $C_2 = P_{j2}$, et proche des différents points du contour.

$$\mathcal{B}[C_0, C_1, C_2] = \{C(t) = (1-t)^2\, C_0 + 2\,t\,(1-t)\,C_1 + t^2\, C_2,\ t \in [0; 1]\}$$

$P_{j1} = C(0)$ i.e. $P_{j1} = P_{j1+0}$ correspond au paramètre $t = 0 = 0/n$ pour la courbe $\mathcal{B}$.
$P_{j2} = C(1)$ i.e. $P_{j2} = P_{j1+n}$ correspond au paramètre $t = 1 = n/n$ pour la courbe $\mathcal{B}$.
$\rightarrow$ dans l'idéal, pour tout $1 \leq i \leq n-1$, le point P_{j1+i} devrait être le point de la courbe $\mathcal{B}$ correspondant au paramètre $t_i = i/n$ c'est à dire $P_{j1+i} = C(i/n)$, et alors la courbe passerait par tous les points du contour (la courbe serait une courbe *interpolante*), ce qui n'est pas le cas en général pour $n > 2$.
Donc pour choisir le point C_1, on procède ainsi :

1. soit i un indice entre 1 et $n-1$, on détermine la courbe de Bézier $\mathcal{B}_i = \mathcal{B}([C_{i,0}, C_{i,1}, C_{i,2}])$ telle que :

$$C_{i,0} = C_i(0) = P_{j1} = P_{j1+0}\ ,\ C_i(t_i) = C_i(i/n) = P_{j1+i}\ ,\ C_{i,2} = C_i(1) = P_{j2} = P_{j1+n}$$

$$\Rightarrow C_i(t_i) = (1-t_i)^2\, P_{j1} + 2\,t_i\,(1-t_i)\,C_{i,1} + t_i^2\, P_{j2} = P_{j1+i}$$

$$\iff C_{i,1} = \frac{P_{j1+i} - (1-t_i)^2\, P_0 - t_i^2\, P_{j2}}{2\,t_i\,(1-t_i)} = \frac{P_{j1+i} - (1-t_i)^2\, P_0 - t_i^2\, P_{j2}}{\omega_i} \quad \text{avec } \omega_i = 2\,t_i\,(1-t_i)$$

(remarque : comme $0 < t_i < 1$, $\omega_i = 2\,t_i\,(1-t_i) > 0$, et donc $C_{i,1}$ est bien définie)

2. la courbe de Bézier finale $\mathcal{C} = \mathcal{B}([C_0, C_1, C_2])$ approchant les points $\{P_{j1}, \ldots, P_{j2}\}$ est déterminée ainsi :

$$C_0 = P_{j1}\ ,\ C_1 = \frac{\displaystyle\sum_{i=1}^{n-1} \omega_i C_{i,1}}{\displaystyle\sum_{i=1}^{n-1} \omega_i}\ ,\ C_2 = P_{j2} \quad \text{avec } \omega_i = 2\,t_i\,(1-t_i) = \frac{2\,i\,(n-i)}{n^2}$$

Pourquoi choisir $C_1 = \dfrac{\sum_{i=1}^{n-1} \omega_i\, C_{i,1}}{\sum_{i=1}^{n-1} \omega_i}$ et non pas $C_1 = \dfrac{1}{n-1}\displaystyle\sum_{i=1}^{n-1} C_{i,1}$ (barycentre des $n-1$ points $C_{i,1}$) ?

1) supposons qu'on remplace le point P_{j1+i} par $\overline{P}_{j1+i} = P_{j1+i} + \Delta P$ (par exemple ΔP peut représenter une erreur d'arrondi), alors

$$\bar{C}_{i,1} = \frac{\overline{P}_{j1+i} - (1-t_i)\, P_{j1} - t_i^2\, P_{j2}}{\omega_i} = C_{i,1} + \frac{\Delta P}{\omega_i}$$

   • Si on choisit $C_1 = \dfrac{1}{n-1}\displaystyle\sum_{i=1}^{n-1} C_{i,1} \Rightarrow \bar{C}_1 = C_1 + \dfrac{\Delta P}{(n-1)\omega_i}$ donc une modification de ΔP sur le point P_{j1+i} induit une modification de $\Delta C = \bar{C}_1 - C_1 = \dfrac{\Delta P}{(n-1)\omega_i}$ dépendante de l'indice i.

   • Si on choisit $C_1 = \dfrac{\sum_{i=1}^{n-1} \omega_i C_{i,1}}{\sum_{i=1}^{n-1} \omega_i} \Rightarrow \bar{C}_1 = C_1 + \dfrac{\Delta P}{\sum_{i=1}^{n-1} \omega_i} = C_1 + \alpha\, \Delta P$ donc une modification de ΔP sur le point P_i induit une modification de $\Delta C = \bar{C}_1 - C_1 = \alpha\, \Delta P$ indépendante de l'indice i.

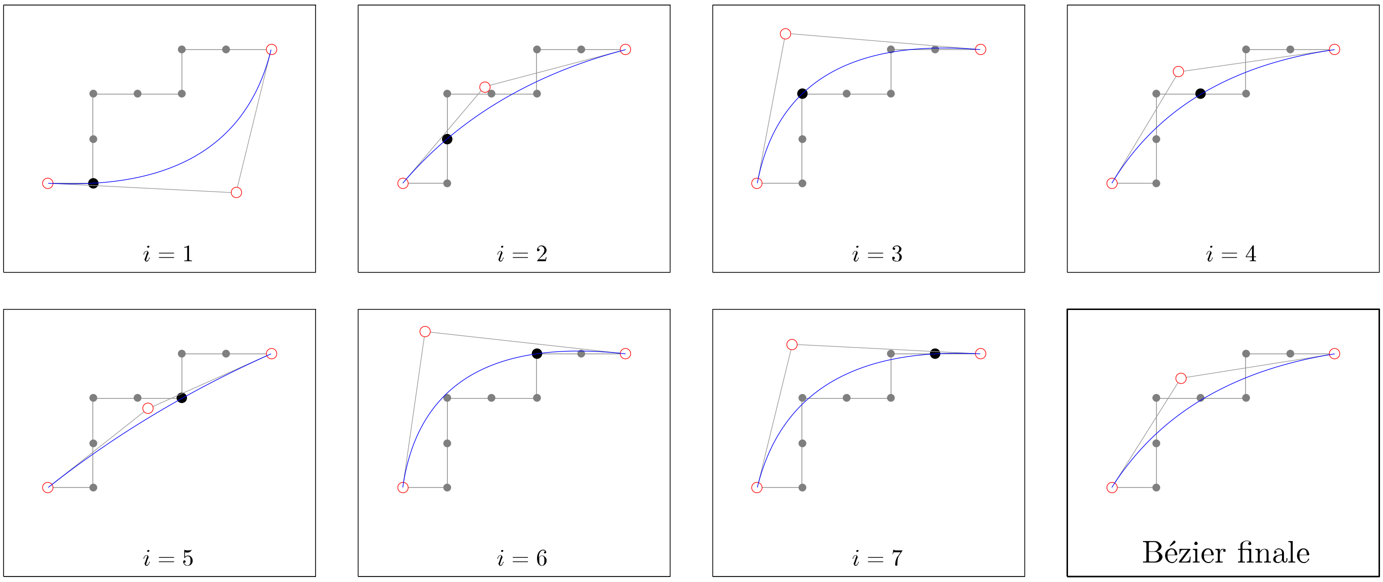
2) le choix de la formule $C_1 = \dfrac{\sum_{i=1}^{n-1} \omega_i\, C_{i,1}}{\sum_{i=1}^{n-1} \omega_i}$ fournit une formule directe pour C_1 sans nécessité de calculer explicitement les points $C_{i,1}$.

Un exemple de calcul de courbe de Bézier de degré 2 approchant une séquence de points

Données :

i	0	1	2	3	4	5	6	7	8
P_i	$(0,0)$	$(1,0)$	$(1,1)$	$(1,2)$	$(2,2)$	$(3,2)$	$(3,3)$	$(4,3)$	$(5,3)$

Tracé des différentes Bézier $\mathcal{B}_i$, $1 \leq i \leq 7 = n-1$, et de la Bézier finale $\mathcal{B}$



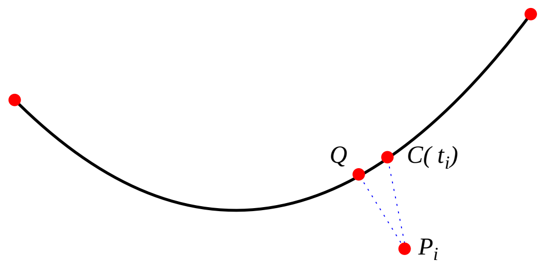
i	$C_{i,1}$			ω_i		
1	$(7080,-360)/1680$	$\simeq$	$(4.21,-0.21)$	$14/64$	$=$	0.21875
2	$(3080,3639)/1680$	$\simeq$	$(1.83,2.17)$	$24/64$	$=$	0.375
3	$(1064,5656)/1680$	$\simeq$	$(0.63,3.37)$	$30/64$	$=$	0.46875
4	$(2520,4200)/1680$	$=$	$(1.5,2.5)$	$32/64$	$=$	0.5
5	$(3752,2968)/1680$	$\simeq$	$(2.23,1.77)$	$30/64$	$=$	0.46875
6	$(840,5880)/1680$	$=$	$(0.5,3.5)$	$24/64$	$=$	0.375
7	$(1320,5400)/1680$	$\simeq$	$(0.79,3.21)$	$14/64$	$=$	0.21875

$$\Rightarrow C_1 = (2600,4120)/1680 \simeq (1.547619, 2.452381)$$

Le calcul de la distance entre un point et une courbe de Bézier de degré 2

Une fois la courbe $\mathcal{B}$ déterminée, il faut trouver le point $P_k = P_{j1+i}$ le plus éloigné de $\mathcal{B}$.
Pour un point $P = (x, y)$, la distance de P à la courbe $\mathcal{B}$ est donnée par :

$$d(P, \mathcal{B}) = \underset{t\in[0,1]}{\text{Min}} \|\overrightarrow{C(t)P}\| = \|\overrightarrow{QP}\|$$



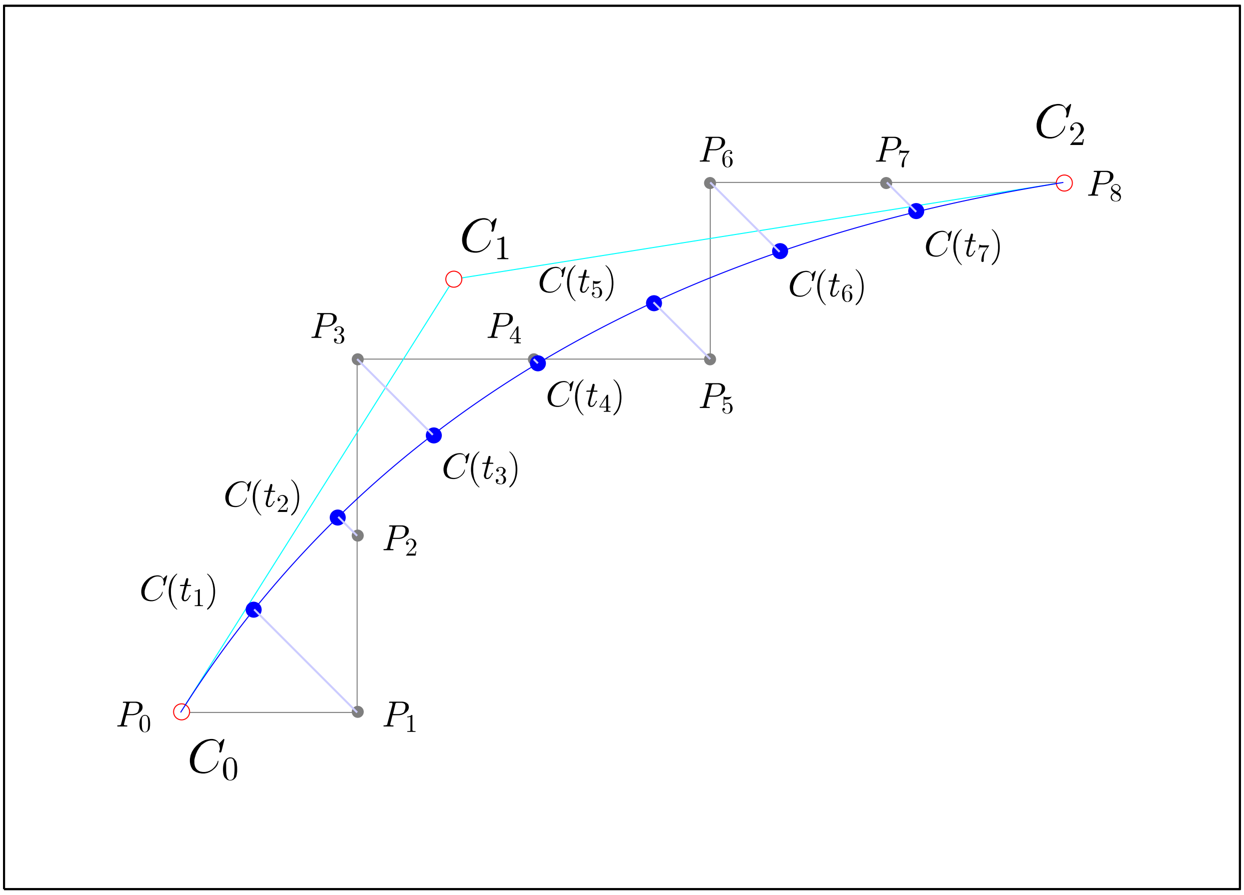
(avec Q point de la courbe le plus proche de P) ce qui peut être compliqué à calculer car il faut trouver $t \in [0, 1]$ tel que $\|\overrightarrow{P\,C(t)}\|$ soit minimal ($\rightarrow$ problème d'optimisation).
Au lieu de calculer la vraie distance $d(P_{j1+i}, \mathbf{B})$ entre P_{j1+i} et $\mathcal{B}$, on va la remplacer par $d(P_{j1+i}, C(t_i))$, distance entre le point P_{j1+i} et le point $C(t_i) = C(i/n)$, point de la courbe de Bézier $\mathcal{B}$ correspondant à $t_i = i/n$.

Exemple

Données :

i	0	1	2	3	4	5	6	7	8
P_i	$(0, 0)$	$(1, 0)$	$(1, 1)$	$(1, 2)$	$(2, 2)$	$(3, 2)$	$(3, 3)$	$(4, 3)$	$(5, 3)$

$$\Rightarrow C_1 = (2600, 4120)/1680 \simeq (1.547619, 2.452381)$$



Indice i	1	2	3	4	5	6	7
$d(P_i, C(t_i))$	0.825	0.151	0.606	0.033	0.455	0.556	0.236

Dans ce cas, le point P_1 est le plus éloigné de la courbe de Bézier, et $d_{max} \simeq 0,825$.

L'algorithme de simplification par Bézier de degré 2

• Données : CONT $= \{P_0, \ldots, P_p\}$ séquence ordonnée des points d'un contour polygonal
$j1$ et $j2$ (indices avec $0 \leq j1 < j2 \leq p$)
d distance-seuil (avec d réel positif ou nul)

• Résultat : L $= \{B_1, B_2, \ldots, B_q\}$ séquence ordonnée de Bézier quadratiques, le dernier point de contrôle de B_j est le premier point de contrôle de B_{j+1}

```
-- simplifier la partie du contour CONT comprise entre les indices j1 et j2 avec la distance-seuil d
-- la fonction renvoie la séquence de Bezier2 L
-- procédure récursive de type "diviser pour régner" ("divide and conquer")
fonction simplification_douglas_peucker_bezier2 (CONT,j1,j2,d) → L
    n ← j2 − j1   -- nombre de segments de CONT entre les indices j1 et j2

    -- (0) approcher la séquence de n + 1 points CONT(j1..j2) par une Bézier B de degré 2
    B ← approx_bezier2(CONT,j1,j2)

    -- (1) rechercher le point Pk le plus éloigné de la Bézier B ainsi que la distance dmax correspondante
    dmax ← 0
    k ← j1
    pour j de j1 + 1 à j2 faire
        i ← j − j1
        ti ← reel(i)/reel(n)
        dj ← distance_point_bezier2(Pj ,B ,ti)
        si dmax < dj alors
            dmax ← dj
            k ← j
        fin_si
    fin_pour

    si dmax ≤ d alors
        -- (2) dmax ≤ d : simplification suivant la Bézier B
        L ← {B}
    sinon
        -- (3) dmax > d : "diviser pour régner"

        -- (3.1) décomposer le problème en deux
        -- simplifier la partie du contour CONT compris entre les indices j1 et k avec la distance-seuil d
        L1 ← simplification_douglas_peucker_bezier2 (CONT,j1,k,d)

        -- simplifier la partie du contour CONT compris entre les indices k et j2 avec la distance-seuil d
        L2 ← simplification_douglas_peucker_bezier2 (CONT,k,j2,d)

        -- (3.2) fusionner les deux séquences L1 et L2
        L ← concatenation(L1,L2)
    fin_si
    retourner L   -- retourner la séquence L
```

Appel principal :

```
L ← simplification_douglas_peucker_bezier2
                    (CONT, premier_indice(CONT),dernier_indice(CONT), d)
```

7.3 - Simplification de contour par courbe(s) de Bézier cubiques (de degré 3)

Le procédé de simplification par courbes de Bézier cubiques (de degré 3) est identique à l'algorithme de simplification par courbes de Bézier quadratiques (degré 2) mais en remplaçant la séquence de Bézier quadratiques par une séquence de courbes de Bézier cubiques $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_p\}$ avec :

- pour tout j, $1 \leq j \leq p$, la courbe $\mathcal{B}_j = \mathcal{B}[C_{j,0}, C_{j,1}, C_{j,2}, C_{j,3}]$,
- pour tout j, $1 \leq j \leq p-1$, il y a raccord continu entre la fin de la courbe $\mathcal{B}_j$ et le début de la courbe suivante $\mathcal{B}_{j+1}$, c'est à dire $C_{j+1,0} = C_{j,3}$,

Les différences par rapport à l'algorithme de simplification par courbes de Bézier quadratiques (page 50) sont :

- les séquences de (courbes de) Bézier quadratiques sont remplacées par des séquences de (courbes de) Bézier cubiques,
- pour un contour initial `CONT` $= \{P_{j1}, \ldots, P_{j2}\}$, on le simplifie par une courbe de Bézier de degré 3 $\mathcal{B} = \mathcal{B}[C_0, C_1, C_2, C_3]$ avec le point initial $C_0 = P_{j1}$ et le point final $C_3 = P_{j2}$
  $\rightarrow$ fonction `approx_bezier3`
- on remplace le calcul de la distance entre un point et une Bézier quadratique par une fonction calculant la distance entre un point et une courbe de Bézier cubique
  $\rightarrow$ fonction `distance_point_bezier3`

Approximation d'une ligne polygonale par une courbe de Bézier de degré 2
(fonction `approx_bezier3`)

Pour simplifier une séquence de points $\{P_{j1}, \ldots, P_{j2}\}$ (avec $n = j2 - j1 \geq 1$) par une courbe de Bézier cubique (de degré 3) $\mathcal{B} = \mathcal{B}[C_0, C_1, C_2, C_3]$, il faut que $C_0 = P_{j1}$ et $C_2 = P_{j2}$, par contre les deux points de contrôle C_1 et C_2 peuvent être choisis d'une infinité de manière.
Cependant il faut les choisir convenablement en fonction des différents points du contour à simplifier, et le calcul de C_1 et C_2 se fait ainsi :

A) Cas $n < 3$ ($n = 1$ **ou** $n = 2$)
on se ramène au cas d'une Bézier de degré 2 que l'on convertit ensuite en Bézier de degré 3.
B) Cas $n \geq 3$

$$\left\{\begin{array}{rcl} C_1 & = & \alpha\, P_{j1} + \lambda \left(\displaystyle\sum_{i=1}^{n-1} \gamma(i)\; P_{j1+i}\right) + \beta\, P_{j2} \\ C_2 & = & \beta\, P_{j1} + \lambda \left(\displaystyle\sum_{i=1}^{n-1} \gamma(n-i)\; P_{j1+i}\right) + \alpha\, P_{j2} \end{array}\right.$$

avec

$$\alpha = \frac{-15\,n^3 + 5\,n^2 + 2\,n + 4}{3(n+2)(3\,n^2+1)} \qquad \beta = \frac{10\,n^3 - 15\,n^2 + n + 2}{3(n+2)(3\,n^2+1)} \qquad \lambda = \frac{70\,n}{3(n^2-1)(n^2-4)(3\,n^2+1)}$$

$$\gamma(k) = 6\,k^4 - 8\,n\,k^3 + 6\,k^2 - 4\,n\,k + n^4 - n^2$$

IMPORTANT : dans les différentes formules, n et k sont des entiers, alors que α, β, λ et $\gamma(k)$ sont des réels
$\rightarrow$ il faut convertir n et k en réels avant de calculer α, β, λ et $\gamma(k)$.

• Pour $n \geq 3$, la méthode utilisée pour calculer les points de contrôle C_1 et C_2 reprend le principe de la méthode utilisée pour les courbes de Bézier de degré 2.
On procède de la manière suivante :

$P_{j1} = C(0)$ i.e. $P_{j1} = P_{j1+0}$ correspond au paramètre $t = 0 = 0/n$ pour la courbe $\mathcal{B}$

$P_{j2} = C(1)$ i.e. $P_{j2} = P_{j1+n}$ correspond au paramètre $t = 1 = n/n$ pour la courbe $\mathcal{B}$

$\rightarrow$ dans l'idéal, pour tout $1 \leq i \leq n-1$, le point P_{j1+i} devrait correspondre au paramètre $t_i = i/n$ ($P_{j1+i} = C(i/n)$) et la courbe passerait ainsi par tous les points (la courbe serait une courbe *interpolante*) ce qui n'est pas le cas en général pour $n > 4$.

Pour calculer la courbe de Bézier cubique $\mathcal{C}$ approchant la séquence de points $\{P_{j1}, \ldots, P_{j2}\}$, on va procéder ainsi (méthode similaire à celle utilisée pour la simplification par courbes de Bézier de degré 2) :

1. Soient i et j deux indices tels que $1 \leq i < j \leq n-1$, on calcule la courbe de Bézier $\mathcal{B}_{i,j} = \mathcal{B}([P_{j1}, C_{i,j,1}, C_{i,j,2}, P_{j2}])$ telle que :

$$\left\{ \begin{array}{ccccc} C_{i,j}(i/n) & = & C_{i,j}(t_i) & = & P_{j1+i} \\ C_{i,j}(j/n) & = & C_{i,j}(t_j) & = & P_{j1+j} \end{array} \right\}$$

$$\Rightarrow \left\{ \begin{array}{ccccc} B_1^3(t_i)\,C_{i,j,1} & + & B_2^3(t_i)\,C_{i,j,2} & = & P_{j1+i} - B_0^3(t_i)\,P_{j1} - B_3^3(t_i)\,P_{j2} \\ B_1^3(t_j)\,C_{i,j,1} & + & B_2^3(t_j)\,C_{i,j,2} & = & P_{j1+j} - B_0^3(t_j)\,P_{j1} - B_3^3(t_j)\,P_{j2} \end{array} \right\}$$

   En résolvant ce système, on peut alors déterminer les expressions de $C_{i,j,1}$ et $C_{i,j,2}$.

2. La courbe de Bézier finale $\mathcal{B}_i = \mathcal{B}([C_0, C_1, C_2, C_3])$ approchant les points $\{P_{j1}, \ldots, P_{j2}\}$ est déterminée ainsi :

$$C_0 = P_{j1}\ ,\ C_1 = \frac{\displaystyle\sum_{1=i<j=n-1} \omega_{i,j} C_{i,j,1}}{\displaystyle\sum_{1=i<j=n-1} \omega_{i,j}}\ ,\ C_2 = \frac{\displaystyle\sum_{1=i<j=n-1} \omega_{i,j} C_{i,j,2}}{\displaystyle\sum_{1=i<j=n-1} \omega_{i,j}}\ ,\ C_3 = P_{j2}$$

   avec $\omega_{i,j} = B_1^3(t_i) B_2^3(t_j) - B_2^3(t_i) B_1^3(t_j) = \dfrac{9(n-i)i(j-i)j(n-j)}{n^5}$

On obtient pour C_1 et C_2 les formules de la page 51 uniquement dépendant de $n = j2 - j1 > 0$ et des points P_{j1+i} $(0 \leq i \leq n)$ du contour à simplifier sans avoir à calculer explicitement les points $C_{i,j,1}$ et $C_{i,j,2}$.

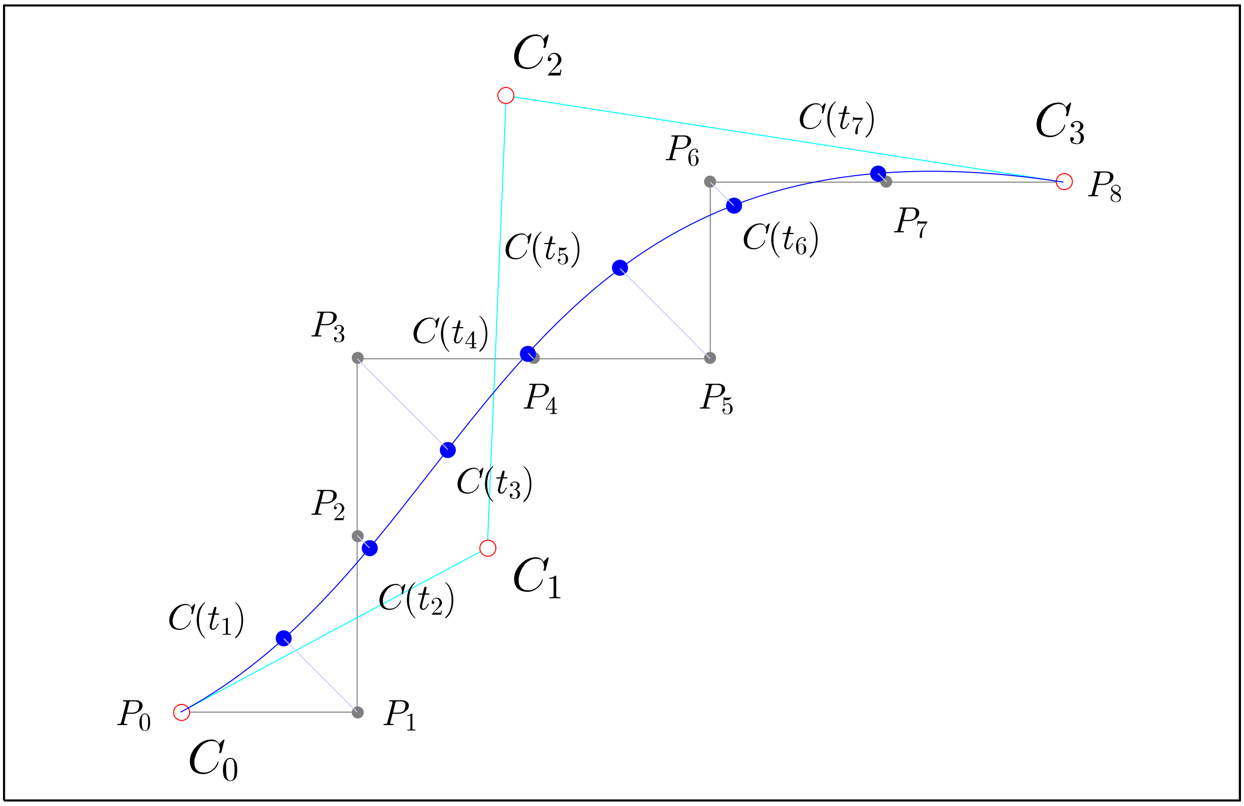
Le calcul de la distance entre un point et une courbe de Bézier de degré 3

Une fois la courbe $\mathcal{B}$ déterminée, il faut trouver le point P_i le plus éloigné de $\mathcal{B}$.
Comme pour le cas des Bézier de degré 2, au lieu de calculer la vraie distance entre $P_j = P_{j1+i}$ et $\mathcal{B}$, on va la remplacer par $d(P_{j1+i}, C(t_i))$ avec $t_i = i/n$.

Un exemple de calcul de courbe de Bézier de degré 3 approchant une séquence de points

Données :

i	0	1	2	3	4	5	6	7	8
P_i	$(0,0)$	$(1,0)$	$(1,1)$	$(1,2)$	$(2,2)$	$(3,2)$	$(3,3)$	$(4,3)$	$(5,3)$



$$C_0 = P_0 = (0,0) \qquad C_1 = \frac{1}{5211}(9053, 4843) \simeq (1.737287, 0.929380)$$

$$C_2 = \frac{1}{5211}(9610, 18182) \simeq (1.844176, 3.489158) \qquad C_3 = P_n = (5,3)$$

Indice i	1	2	3	4	5	6	7
$d(P_i, C(t_i))$	0.588	0.100	0.726	0.045	0.722	0.186	0.070

Le point P_3 est le point le plus éloigné de la courbe de Bézier, et $d_{max} \simeq 0,726$.