ensiie
école nationale supérieure d'informatique
pour l'industrie et l'entreprise

Rapport Projet PAP 2A 2018-2019

Courbes de Bézier et polices de caractères

XU Kevin
LI Ziheng

13 janvier 2019

Sommaire

Préambule

L'objectif de ce projet est de réaliser des polices de caractères en utilisant des courbes de Bézier. Le programme est implémenté en C++, sans l'aide d'une quelconque bibliothèque extérieure au C++. Trois polices de caractères pour les lettres de l'alphabet en majuscule et sans sérif ont été créées :

1. La première police correspond aux glyphes décrivant le contour du caractère en noir dans un bitmap (rectangle de pixels blancs et noirs, les pixels noirs étant donc le contour du caractère).
2. La deuxième police est la première police mais avec l'intérieur des caractères rempli en noir, toujours dans un bitmap.
3. La troisième police est la deuxième police mais avec un contour rouge de deux pixels à l'extérieur du caractère. Ce style de police est très utile pour les sous-titrages des films, pour pouvoir toujours lire le sous-titre quelque soit la scène du film.

Pour fournir les images png de chaque caractère des différentes polices, nous avons utilisé la bibliothèque libpng.

Nous allons présenter dans ce rapport les différentes classes du programme. Nous évoquerons les objectifs, les problèmes rencontrés et les solutions trouvés pour chacune des classes.

1 Les classes - Diagramme UML

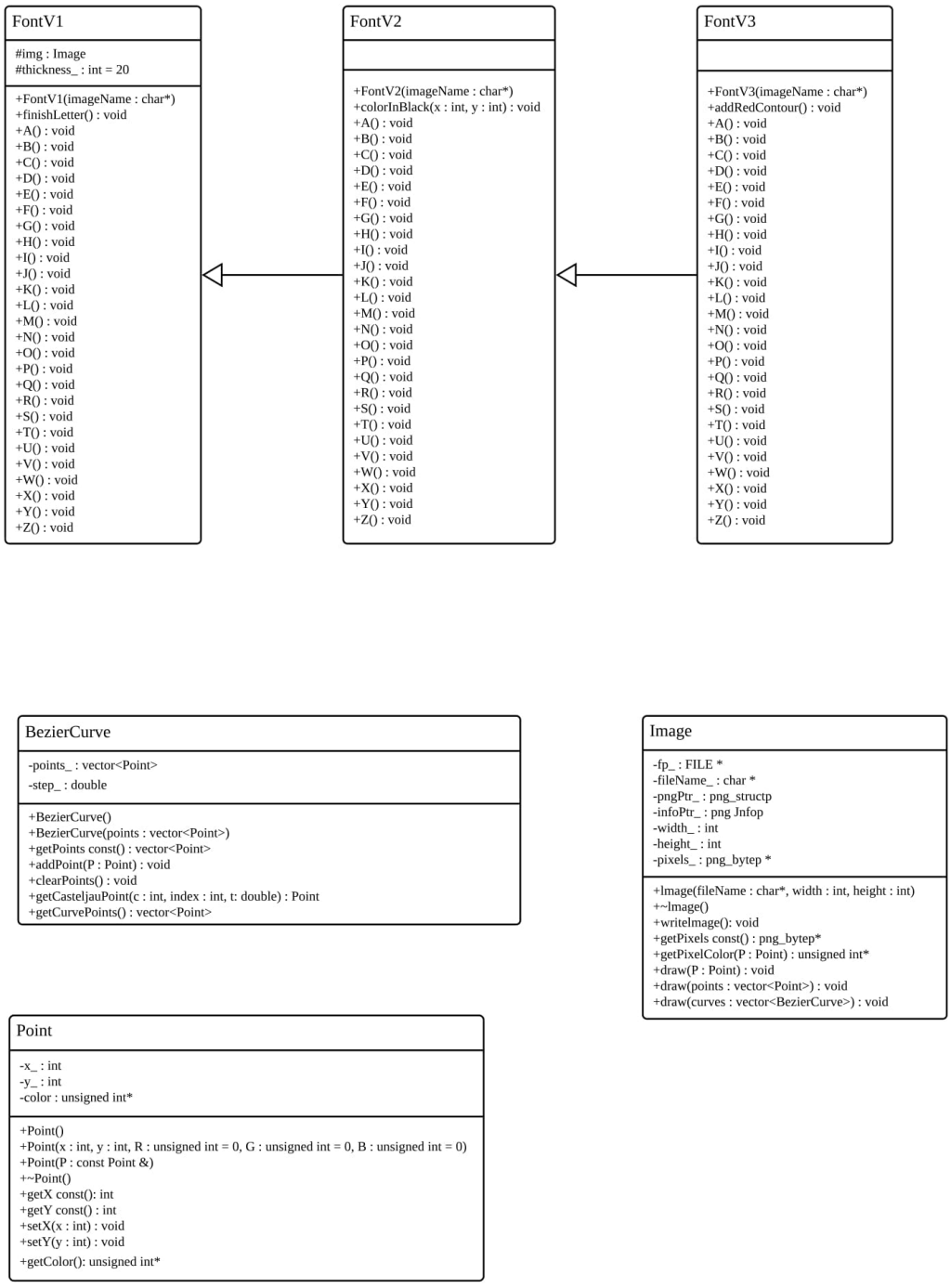

FIGURE 1 – Diagramme UML du programme

2 Point - Classe `Point`

2.1 Objectifs

Tout d'abord, sachant qu'une courbe de Bézier est formée par plusieurs points, il faut avoir une classe définissant un point sur un système de coordonnées dans un plan. Le but de cette classe est alors de définir les point des courbes de Bezier que l'on va tracer. En outre, il faut aussi pouvoir avoir la possibilité de dessiner des points de différentes couleurs.

2.2 Solution

Une instance de la classe `Point` a 3 attributs :
— La coordonnée $x\_$
— La coordonnée $y\_$
— Un tableau *color_* contenant les niveaux de Rouge, Vert et de Bleu (allant de 0 à 255).
Cette classe contient aussi les getters et setters pour ces attributs.

3 Les image - Classe `Image`

3.1 Objectifs

Afin de pouvoir fournir les images PNG de chaque caractère, nous avons décidé de créer une classe `Image`. Cette classe s'occupe de toute la gestion des images comme la création et la modification des pixels sur l'image.

3.2 Solution et Réalisation

Pour réaliser cette classe, on a utiliser la bibliothèque **libpng**. Cette dernière nous permet de créer des images et de modifier les pixels avec la couleur que l'on souhaite. Elle contient notamment les méthodes permettant de dessiner des `Point` et des `BezierCurve`.

Dans le constructeur de la classe, on a décidé que l'image créée aura un fond blanc par défaut.

4 Courbes de Bézier - Classe `BezierCurve`

4.1 Objectifs

Cette classe est l'une des classes les plus importantes du programme. Afin de tracer les différentes courbes d'un caractère, nous devons implémenter une classe permettant de calculer les points à tracer afin d'obtenir des courbes de Bézier. En effet, chaque caractère est composé de courbes de Bézier linéaire et/ou quadratique.

4.2 Solution et Réalisation

La classe contient deux attributs :
— Un vecteur de `Point` *points_* qui contient les points de contrôles de la courbe de Bézier à tracer.
— Un réel *step_* qui correspond à la précision de l'algorithme de Casteljau.

Nous disposons de deux méthodes primordiales pour créer les courbes de Bézier :

— `void addPoints(Point)` : Cette méthode permet d'ajouter un point de contrôle à la courbe de Bézier.

— `void clearPoints()` : Cette méthode permet de supprimer tous les points de contrôle de la courbe de Bézier.

4.2.1 Algorithme de Casteljau

Pour tracer une courbe de Bézier, nous n'avons pas utilisé la formulation mathématique comme présentée dans le sujet. En effet, l'algorithme de Casteljau développé par Paul de Casteljau permet de réaliser cela en utilisant une approche récursif pour approximer les courbes de Bézier.

Le principe de l'algorithme est de construire récursivement un ensemble de barycentres se rapprochant au fur et à mesure de la courbe de Bézier. L'algorithme s'arrête alors lorsque toutes les distances entre deux points consécutifs sont assez petites. Nous avons imposé une valeur du paramètre $step\_ = 0.000001$ pour les calculs de barycentre.

Algorithm 1 `getCasteljauPoint`

Require: $c \in \mathbb{N}$, $index \in \mathbb{N}, t \in \mathbb{R}^*$, *points* A vector which contains the control points of the Bezier Curve

Ensure: The Point computed with the De Casteljau's algorithm

1: **function** GETCASTELJAUPOINT(c, $index$, t, *points*)
2: **if** c = 0 **then**
3: **return** points[index]
4: **end if**
5: Set a Point in $P1$ to `getCasteljauPoint(`$c-1$, $index$, t, *points*`)`
6: Set a Point in $P2$ to `getCasteljauPoint(`$c-1$, $index+1$, t, *points*`)`
7: Set a Point in P with $x = (1-t) \times (x\ of\ P1) + t \times (x\ of\ P2)$ and $y = (1-t) \times (y\ of\ P1) + t \times (y\ of\ P2)$
8: **return** P
9: **end function**

Algorithm 2 `getCurvePoints`

Require: *points* The control points of the Bezier Curve, *step* The parameter of the barycenter

Ensure: A vector *Res* of Points which picture the Bezier Curve

1: **function** GETCURVEPOINTS(*points*, *step*)
2: Set an empty vector *Res* of Points
3: Set *size* to the size of the vector *points*
4: **for** $t = 0$ to 1 with a step of *step* **do**
5: Add to the vector *Res* the Point : `getCasteljauPoint(`$size-1$, 0, t, *points*`)`
6: **end for**
7: **return** *Res*
8: **end function**

Cette algorithme nous permet de tracer des courbes de Bézier linéaires et quadratiques. Mais on peut aussi tracer des courbes avec plus de trois points de contrôle comme des courbes de Bézier cubiques ou plus. Cependant, dans le cadre de notre projet, nous avons seulement utiliser l'algorithme pour trois points de contrôle au plus seulement.

4.2.2 Résultats et Problèmes rencontrés

Pour certaines courbes de Bézier, on a rencontré quelques problèmes. En raison des erreurs d'approximations, il peut arriver que certaines courbes de Bézier présentes des pixels non reliés entre elles. Voici un exemple :

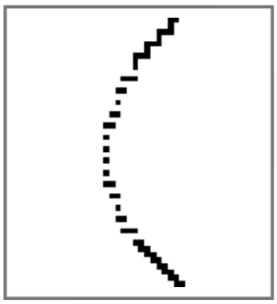

FIGURE 2 – Courbe non reliée

En effet, cela est dû dans certains cas à un *step_* pas assez petit. Mais plus la valeur de cette variable est petite, plus le temps d'exécution sera élevé. C'est pour cela qu'on a décidé de définir $step\_ = 0.000001$ après avoir tester différentes valeurs.

5 Première Police - Classe `FontV1`

5.1 Objectifs

Pour réaliser la première police, on doit créer une classe qui contient les méthodes pour dessiner toutes les lettres de l'alphabet dans des fichiers PNG. En outre, le but de la deuxième question est de remplir l'intérieur des lettres de la première question, donc il faut produire un résultat utilisable par des classes filles.

5.2 Solution et Réalisation

La classe `FontV1` contient alors les attributs suivants :

— `img_` Une instance de la classe `Image` sur laquelle la lettre va être dessiner de taille 500x500 pixels

— Un entier `thickness_`= 20 qui correspond à l'épaisseur des lettres que l'on souhaite dessiner.

Tout d'abord, on a pensé à implémenter des méthodes qui génèrent les images des lettres lorsqu'on saisit une lettre que l'on veut l'afficher. Cependant, on s'en rendu compte que cela était impossible pour notre niveau de connaissance.

De fait, pour générer les caractères, on a implémenté 26 méthodes différentes qui tracent chacune les courbes de Bézier permettant de dessiner une lettre de l'alphabet sur l'image `img_`. Pour certaines lettres, on a beaucoup de courbes de Bézier à tracer, donc la taille des méthodes est parfois supérieur à 50 lignes.

Les lettres comportant que des courbes de Bézier linéaires ont été plus faciles à tracer que celle comportant beaucoup de courbes de Bézier quadratiques. Voici quelques lettres tracées avec cette classe :

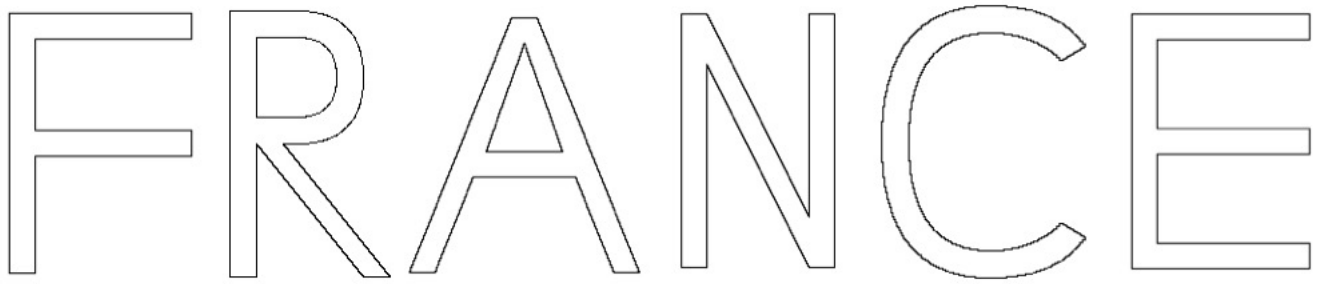

Figure 3 – Le mot FRANCE avec la police 1

6 Deuxième Police - Classe `FontV2`

6.1 Objectifs

Pour générer les caractères de la deuxième police, on a créé une classe `FontV2`. Cette police est la police précédente mais avec l'intérieur des contours colorié en noir.

6.2 Solution et Réalisation

Au début, on a voulu remplir l'intérieur des lettres en traçant plusieurs courbes de Bézier en suivant les contours des lettres. Mais le résultat n'était pas très précis car il y avait parfois des trous blancs à l'intérieur des contours des lettres. De plus, cette méthode devait calculer plusieurs courbes de Bézier en plus de celles du contours de la lettre. Et le temps d'exécution pour tracer une seule lettre était assez long pour chacune des lettres. Par exemple, voici à quoi ressemblait notre première version de la police 2 pour la lettre O :

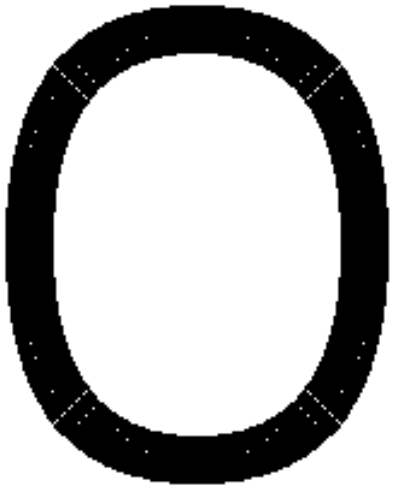

FIGURE 4 – Première version de la police 2

Pour résoudre ce problème, nous avons finalement utilisé l'**algorithme de remplissage par diffusion (Flood fill ou seed fill)**. Voici son pseudo-code :

Algorithm 3 `colorInBlack`

Require: $x \in \mathbb{N}$, $y \in \mathbb{N}$
Ensure: Color a white area in black

```
1: function colorInBlack
2:     if The coordinates (x, y) are out of bounds then
3:         return
4:     end if
5:     if The pixel of coordinates (x, y) is white then
6:         Color the pixel of coordinates (x, y) in black on the Image img
7:         colorInBlack(x+1, y)
8:         colorInBlack(x-1, y)
9:         colorInBlack(x, y+1)
10:        colorInBlack(x, y-1)
11:    end if
12: end function
```

La classe `FontV2` est une classe fille de la classe `FontV1`. En effet, on doit dans un premier temps faire appel à une des méthodes de la classe `FontV1` pour dessiner sur l'image les contours d'une lettre. C'est alors que par la suite on peut faire appel à l'algorithme **seed fill** en choisissant un point de départ à l'intérieur du contour de la lettre pour remplir l'intérieur de noir. De plus, il faut que les lettres dessinées par la police 1 aient des contours fermés.
On obtient alors le résultat suivant pour la lettre O :

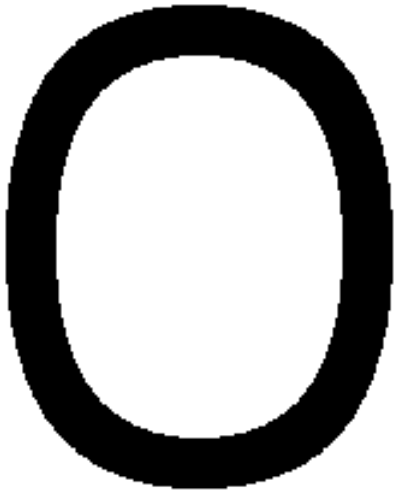

FIGURE 5 – Lettre O avec la police 2

FIGURE 6 – Le mot FRANCE avec la police 2

7 Troisième Police - Classe `FontV3`

7.1 Objectifs

La troisième police consiste à ajouter un contour de 2 pixels autour des contours des lettres de la deuxième police.

7.2 Solution et Réalisation

La classe `FontV3` hérite de la classe `FontV2`. En utilisant cette héritage, il suffit de tracer les lettres grâce aux méthodes de la classe `FontV2`. Puis l'algorithme suivant est appliqué :

Algorithm 4 `addRedContour`

Require: An Image img with a letter drawn by the class FontV2
Ensure: Add a contour of two pixels around the letter

```
1: function addRedContour(img)
2:     for x ∈ ⟦1; 499⟧ do
3:         for y ∈ ⟦1; 499⟧ do
4:             if The pixel of coordinates (x, y) is black then
5:                 if The pixel of coordinates (x − 1, y) is white or red then
6:                     Color the pixel of coordinates (x − 1, y) in red on the Image img
7:                     Color the pixel of coordinates (x − 2, y) in red on the Image img
8:                 end if
9:                 if The pixel of coordinates (x + 1, y) is white or red then
10:                    Color the pixel of coordinates (x + 1, y) in red on the Image img
11:                    Color the pixel of coordinates (x + 2, y) in red on the Image img
12:                end if
13:                if The pixel of coordinates (x, y − 1) is white or red then
14:                    Color the pixel of coordinates (x, y − 1) in red on the Image img
15:                    Color the pixel of coordinates (x, y − 2) in red on the Image img
16:                end if
17:                if The pixel of coordinates (x, y + 1) is white or red then
18:                    Color the pixel of coordinates (x, y + 1) in red on the Image img
19:                    Color the pixel of coordinates (x, y + 2) in red on the Image img
20:                end if
21:            end if
22:        end for
23:    end for
24: end function
```

Comme les images générées ont tous une taille de 500x500 pixels, l'algorithme parcourt tous les pixels de l'image pour ensuite ajouter le contour de pixels rouges selon les différents cas.

Figure 7 – Le mot FRANCE avec la police 3

Conclusion

En conclusion, on a réussi à dessiner tous les contours des lettres, les remplir en noir et ajouter un contour rouge de deux pixels. Cependant, il y a encore beaucoup de choses qui peuvent être améliorées. Par exemple, les lettres compliquées sont un peu moins esthétiques et ce n'est pas très facile de dessiner les lettres qui contiennent beaucoup de courbes de Bezier. De plus, on a imposé une taille d'image de 500x500 pixels, donc il y a parfois des problèmes d'anti-aliasing.

Au niveau de la programmation, l'algorithme de Casteljau peut présenter quelques erreurs d'approximation dans certains cas. En outre, il existe aussi des façons plus optimisés de coder l'algorithme seed fill, mais celle qu'on a utilisé est une version basique.

Enfin, pour générer l'ensemble des 26 lettres pour chacune des trois polices, il faut compter un temps d'exécution d'environ 20 minutes.