Rapport Projet PAP

Courbes de Bézier et police de caractères

2021/2022

École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Table des matières

1 Introduction

La représentation de caractères dans les écrans de nos ordinateurs en utilisant les bitmaps possèdent de multiples usages, par exemple, le sous-titrage des films. Généralement, pour leur simplicité et leur facilité de lecture dans le cas où la résolution n'est pas très haute, les polices sans serif sont les plus utilisés.

En conséquence, l'objectif de ce projet est de représenter des polices de caractères sans serif en majuscule. On essayera de les créer de sorte que l'on puisse faire varier la taille des bitmaps. Pour réduire l'effet de la taille on utilisera des courbes de Bézier pour la création des polices de caractères. L'avantage principale de ces courbes est qu'on peut créer des bitmaps de différentes tailles en maintenant la structure de nos lettres grâce à l'utilisation de points de contrôle prédéfinis.

La programmation de la représentation de chaque bitmap se fera dans le langage C++ et à l'aide de la bibliothèque SDL. On considérera trois étapes différenciées : la création d'une lettre "vide", le remplissage de l'intérieur et le contour en rouge. La raison de l'inclusion de la dernière étape permet que les lettres restent bien visibles même si la couleur du support change.

2 Structure

Dans cette section on expliquera brièvement la structure de notre projet. Pour une meilleure compréhension de cette structure, on a ajouté le diagramme UML dans la Figure 1. On peut remarquer que notre projet est composé de 5 classes.

2.1 Diagramme UML

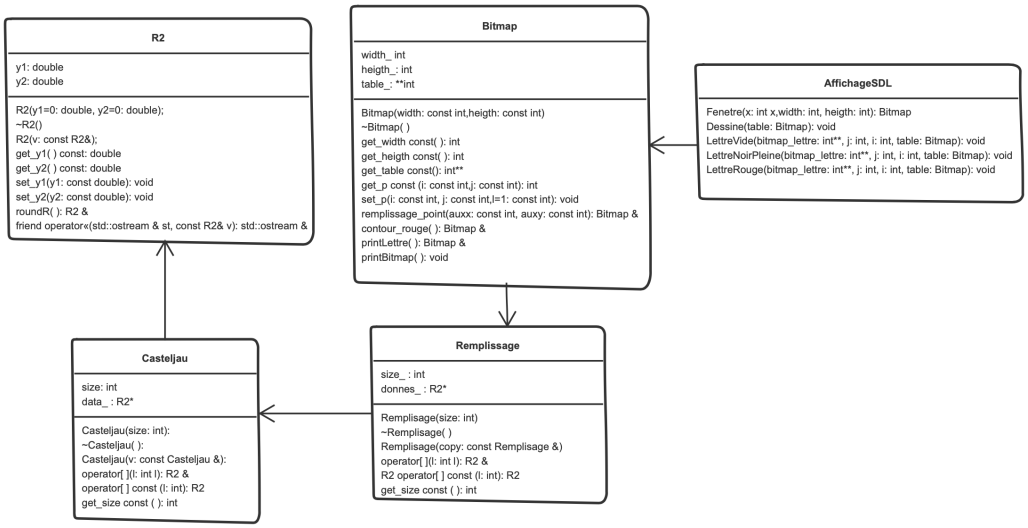

FIGURE 1 – Diagramme UML du projet

2.2 Description de la structure

Tout d'abord, on a créé une classe R2 qui répresente $\mathbb{R}^2$ pour l'appliquer à l'algorithme de Casteljau. Cette classe est celle que l'on utilisera comme base de notre projet, pour la majorité des autres classes, on créera des objets de R2. De plus, on ajoute des opérateurs nécessaires pour notre projet : sommation et soustraction de deux éléments de R2 et multiplication par double.

Ensuite, on a codé l'algorithme de Casteljau dans une classe héritant de R2 appelée Casteljau. Cette classe nous permettra de développer une fonction `pointcourbe(T: Casteljau &,t: const double)` pour calculer la valeur de la courbe de Bézier considérée en $t \in [0, 1]$.

Par ailleurs, pour faire en même temps tous les calculs de t qu'on considère, on a créé une classe appelée Remplissage. L'idée de cette classe est la création de la fonction `Calcul(E: Casteljau,T: Remplissage)` pour calculer toutes les valeurs considérées de notre courbe, c'est-à-dire, tous les t considérés.

Après, pour avoir une représentation sous forme de tableau, avant d'ajouter la bibliotéque SDL, et pour le cas où le terminal ne peut pas l'utiliser, on pourra imprimer ici une version du bitmap final composé de 0 de 1 et de 2, pour les trois couleurs blanc, noir et rouge respectivement.

Enfin, on a ajouté une classe `AffichageSDL` implémentant une unique fonction `Dessine` afin de simplifier l'usage que l'on fait de la bibliothèque SDL. Cette classe, permet à partir d'un bitmap d'une lettre, de créer la fenêtre SDL et d'afficher les trois polices demandées.

3 Les choix

Dans cette section, on résumera les différents choix que nous avons fait pour le développement de notre projet. De plus, on essayera d'argumenter certains choix que nous avons fait au lieu d'autres possibles. On n'analysera pas en détail toutes nos décisions mais on expliquera les plus importantes. Finalement, on supposera que le lecteur a connaissance des bases mathématiques et informatiques pour comprendre les bases de nos décisions. Par exemple, on n'expliquera pas l'algorithme de Casteljau.

3.1 L'algorithme de Casteljau

Pour la construction de chaque lettre on utilisera des courbes de Bézier linéaires et quadratiques. L'idée générale est d'utiliser des points de contrôle sur un tableau 1×1 et après de le multiplier pour la largeur et la hauteur du bitmap.

Cet algorithme a été choisi comme indiqué dans les instructions. L'avantage de cet algorithme est observable dansle cas où la largueur et la hauteur du bitmap n'ont pas le même ratio d'affichage. Ceci nous permettra de créer des polices de caractères de différentes tailles. De même, le choix de décrire des points de contrôle sur un tableau 1×1 au lieu d'autres tailles, aide à simplifier le code. Par exemple, si le tableau était 64×64 on devrait multiplier la première coordonnée par $\frac{\text{width}}{64}$ pour obtenir la coordonnée dans le bitmap.

3.2 Critère d'arrêt

Pour le critère d'arrêt on a choisi de créer un nombre de points égal au maximum du nombre de pixels en hauteur et en largueur. L'avantage d'utiliser ce critère est d'éviter de faire des comparaisons chaque fois qu'on ajoute des nouveaux éléments, ce qui ralentirait la création du bitmap.

3.3 La représentation des points

Pour la représentation des points, on doit transformer des valeurs dans $\mathbb{R}^2$ à $\mathbb{N}^2$ car on travaille sur des pixels. Alors, on a choisi comme forme la transformation de C++ en changeant le type de `float` à `int`.

On a choisi cette méthode pour deux raisons. En premier lieu, il est plus facile de programmer cette méthode parce qu'elle nous permet de ne pas créer une autre méthode pour le faire. En deuxième lieu, on voit qu'il est plus rapide, car il est implementé directement en C++. Donc, il n'y a pas besoin de le chercher dans un fichier créé par nous mêmes.

3.4 Le stockage des lettres

Afin de stocker des valeurs pour la création de chaque lettre on a d'abord créé une fonction par lettre, puis on a construit une fonction qui, avec l'ASCII de la lettre, appelle la fonction associée à cette lettre.

Le raison de ce choix vient de la possibilité d'exécuter une lettre de façon indépendante, par exemple, dans le cas où l'on souhaite afficher une lettre en particulier. De plus, le stockage de toutes les valeurs dans la même fonction impliquerait qu'on ait une fonction avec plus de 50 lignes, ce qui est le maximum demandé dans les intructions du projet.

4 Difficultés techniques

Dans cette section, on va décrire les principales difficultés techniques trouvées et notre façon de les résoudre.

4.1 Le contour

On n'était pas capable de trouver une façon simple de montrer le contour avant de connaître le remplissage des lettres dans les cas où on utilise des courbes de Bézier quadratiques.

La solution qu'on a donc choisi est de créer l'ensemble du bitmap et de le remplir. On aurait pu créer la police avec le contour noir uniquement dans la classe rélative à la SDL. On sait que cette solution n'est pas optimale.

4.2 Utilisation de la SDL

Dans un premier temps, nous n'avions pas réussi à installer la bibliothéque SDL, cela a été résolu pour l'un des deux membres du binôme. En effet, on a d'abord essayé d'installer la SDL sur macos, sans succès, avant d'essayer sous WSL (Windows Subsystem for Linux). Postérieurement, il aurait été plus simple d'installer directement une distribution linux sans machine virtuelle. En effet, l'affichage d'une fenêtre externe, ici la fenêtre SDL, depuis la machine virtuelle, pose des problèmes qui sont résolvables en installant un petit logiciel qui permet justement de gérer ces fenêtres.

En parallèle, on a décidé de créer une classe auxiliaire appelée Bitmap pour créer avec des 0, 1 et 2, la lettre demandée. Ensuite, lorsque l'on a résolu le problème d'installation de la SDL on a décidé de maintenir la classe créée pour les utilisateurs qui voudraient voir le résultat sans utiliser la SDL.

5 Réalisation

5.1 Description des fournitures

D'une part, le répertoire **src** contient l'ensemble des fichiers .cpp et .h utilisés pour exécuter le code avec la commande

```
g++ -g -Wall -Wextra -o projet *.cpp ‘pkg-config --cflags --libs sdl2‘
```

comme il est démandé.

D'autre part, le répertoire **rapport** contient le rapport en .pdf et .tex ainsi que l'ensemble des fichiers nécessaires à l'exécution du fichier .tex.

5.2 Synthèse des fonctionnalités remplies

Les principales fonctionnalités remplies dans le projet sont :

1. Création d'un bitmap de taille séléctionnable par l'utilisateur.
2. Sélection de lettres majuscules de l'alphabet.
3. Remplissage du bitmap de trois façons distinctes : contour noir, remplissage intérieur ou remplissage intérieur avec contour rouge.

6 Conclusions

Pour conclure, nous avons réussi à réaliser des bitmaps pour les lettres de l'alphabet en majuscule.

FIGURE 2 – Lettre B réalisée avec notre code.

On peut voir les trois polices différentes dans la Figure 2. Il y a le contour en noir de la lettre, le remplissage interne et le remplissage interne avec le contour en rouge pour pouvoir la lire plus facilement sur des supports changeants.

6.1 Améliorations

Pour améliorer le projet, nous avons pensé à deux possibilités.

En premier lieu, on pourrait étendre nos possibilités avec la création de lettres minuscules et des signes de ponctuation.

En deuxième lieu, on pourrait considérer l'option de pouvoir écrire des mots complets directement sur un bitmap. Cette option permettrait de ne pas initialiser le programme lettre par lettre.