# Dessin de courbes

## Introduction

Le but de ce présent document et d’expliquer sommairement la manière dont le projet a été réalisé, les choix qui ont été fait et les tests qui ont permis de s’assurer du bon fonctionnement général du programme.

## Travailler en groupe

### Méthodes de travail

Nous avons séparé le travail en fonction des éléments qui composent le projet. Cela comprend les différentes parties du programme mais aussi la rédaction du présent document. Nous avons aussi défini dès le départ une nomenclature pour nos variables, fonctions, noms de fichier, etc.

### Logiciels

Pour pouvoir travailler de manière parallèle et dans de bonne condition, nous avons commencé par définir l’environnement de développement que nous utiliserons pour la réalisation du projet. Cela n’est pas indispensable dans le sens où un code source C peut être compilé sur n’importe quel environnement de n’importe quel système d’exploitation, mais en définissant un environnement unique, cela évite de potentiels problèmes de compatibilité qui pourrait subvenir. Nous avons donc choisis d’utiliser « Visual Studio 2013, professionnel » sous Windows®. Ce choix a été motivé du fait que nous avions déjà les deux beaucoup utilisé cet environnement par le passé.

Pour pouvoir travailler simultanément sur le projet, nous avons choisi de créer un dépôt « Git » afin d’y sauvegarder les sources. Cela permet de créer des branches qui peuvent correspondre aux différents éléments à implémenter au projet (lecture du fichier de configuration, génération de l’image, etc). Une fois cette partie de programme terminée et fonctionnel, il suffit de l’intégrer au programme principal qui est la branche « maître ». Pour héberger le code source nous avons choisis d’utiliser la populaire plateforme de GitHub et SourceTree comme programme de gestion de dépôt.

## Programme

### Architecture

Le programme a été décomposé en multiples fonctions afin d’améliorer la lisibilité et la maintenabilité du code. Elles ont été pensées pour être le plus générique et simple possible, ainsi que pour offrir une certaines abstraction lors de l’écriture de données bitmap.

Chaque fonction a été nommée avec un préfixe en fonction de son utilité.

Les fonctions draw, par exemple, sont toutes les fonctions permettant de dessiner et modifier les données bitmap bruts.

Nous avons donc scindé les fonctions en plusieurs catégories :

* Les fonctions de dessin
* Les fonctions bitmaps
* Les fonctions de lecture/écriture de fichier
* Les fonctions d’insertion de texte
* Les fonctions de vérification d’erreurs

A l’origine, chaque fonction avait son propre header et son propre fichier source. Par soucis de portabilité, et de faciliter de compilation sur les autres environnements, nous avons quasiment tout regroupé dans un seul fichier source.

### Problèmes rencontrés

Les dépassements de tableau, qui sont arrivés à plusieurs reprises lors de nos tests, ne faisaient que rarement planter le logiciel. En effet, la majeure partie du temps, cela n’avait d’incidence que sur les données bitmaps. Nous nous sommes retrouvés à plusieurs reprises avec des images complètement erronées, voir artistiquement intéressantes.

L’allocation dynamique de mémoire a été l’un des points les plus critiques à mettre en place. En effet, il fallait allouer dynamiquement un tableau en 2D. Afin de palier au problème, nous avons généré un tableau de pointeur référençant d’autres tableaux alloués dynamiquement.

Le plus gros problème rencontré a été l’utilisation des « *fonts* ». En effet, aucune bibliothèque standard ne permet de convertir un caractère en une image bitmap. Nous avions déjà prévu les noms de fonctions, mais absolument pas comment nous y prendre pour générer des caractères imprimables. En recherchant sur le très populaire site Web « StackOverflow », un utilisateur suggérait de créer les caractères dans un fichier xbm, contenant une structure composée d’octet. Cette structure, mise sous forme de tableau, représentait parfaitement le caractère demandé. Après pas mal de recherche, nous n’avons malheureusement trouvé aucun header xbm utilisable pour notre programme, surtout vu la taille de fonte demandée. Nous avons donc choisi de le générer. A l’aide d’un peu de script bash, du logiciel ImageMagick, ainsi que d’un peu de script Perl, nous avons pu directement générer nos caractères, ainsi que le « switch/case » de la fonction « drawChar » en un minimum de temps. Certains caractères étaient complètement illisibles et ont dû être modifiés à la main. Une fois le header xbm créé, il suffisait de lire chaque bit de chaque ligne du caractère demandé, et d’écrire le caractère en conséquence.

### Tests

Les tests effectués ont majoritairement été de simples insertions de valeurs susceptibles de poser problème. Des bornes négatives, des marges supérieurs à la surface de l’image, ou encore des valeurs négatives ou impossibles.

Afin de palier à un maximum de problème, nous avons mis en place une fonction de vérification des valeurs entrées.