POLYTECH®
ORLÉANS
École d'Ingénieurs de l'Université d'Orléans

Projet d'informatique : Traceur de courbe

Tom ROQUAND

Sommaire :

I- Sujet

II- Plan d'action

III- Les commandes pour manipuler les pixels de la fenêtre

IV- La classe CPolynome

V- Les axes

VI- Les surcharges

VII- Les courbes

VIII- Les équations

IX- Ajustement du graphique

X- Détails importants

XI- Conclusion

I- Sujet

Il s'agit d'écrire un programme permettant de visualiser le tracé d'une courbe dans la console. Bien que la console soit par défaut destinée à afficher des caractères, on peut avec quatre instructions supplémentaires afficher des points (appelé pixel) dans la console. Vous écrirez une classe CPolynome, permettant de stocker un polynôme de degré quelconque, vous permettrez à l'utilisateur de définir plusieurs polynômes dans la fonction principale, puis vous afficherez la ou les courbes point par point dans la console.

Le programme devra être convivial (utilisation de pointeurs, surcharge d'opérateurs, etc.).

II- Plan d'action

L'idée avant même de commencer le codage est d'effectuer un plan d'action, c'est-à-dire de déterminer tout ce dont il est nécessaire pour obtenir un programme fonctionnel. J'ai donc écrit sous formes d'étapes comment un utilisateur peut entrer l'équation son polynôme et comment le programme va traiter ces informations pour retourner la courbe de celui-ci. Voici donc les étapes que j'ai rédigées :

Etape 1 : Demander à l'utilisateur le nombre de courbes à tracer (les étapes suivants sont pour un seul polynôme, il faudra donc les répéter autant de fois qu'il y a de polynômes).

Etape 2 : Demander à l'utilisateur le degré du polynôme et initialiser un tableau en conséquence.

Etape 3 : Demander à l'utilisateur d'entrer les coefficients en fonction du degré du polynôme puis les stocker dans le tableau que l'on vient d'initialiser.

Etape 4 : Afficher un repère avec deux boucles for (une pour les abscisses et une pour les ordonnées). On ne réalise cette étape qu'une seule fois, évidemment.

Etape 5 : A une variable que l'on nomme ici y, on va associer l'ordonnée d'un point en fonction d'un certain x posé en faisant la somme des coefficients respectivement multipliés par notre x et la puissance correspondante. On affiche alors sur la console le point de coordonnées (x ; y). Avec une boucle for, on effectue cette opération pour un certain nombre de x.

III- Les commandes pour manipuler les pixels de la fenêtre

Après quelques recherches sur internet, les commandes que j'ai trouvées afin de pouvoir manipuler les pixels sur la fenêtre de débogage sont :

- `HWND Hcon = GetConsoleWindow();`

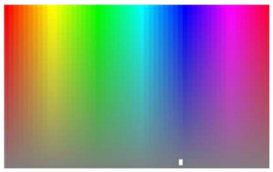
Cette fonction servant à récupérer un contrôleur de la fenêtre console

- `HDC Hdc = GetDC(Hcon);`

Cette fonction sert à récupérer le contrôleur sur le contexte de périphérique (propre à la fenêtre). HDC désignant la console.

- `COLORREF COLOR = *COULEUR*;`

Cette fonction sert à définir la couleur du pixel que l'on va modifier, il en existe des prédéfinies ou on peut également les définir avec des coordonnées qui vont déterminer une couleur dans une certaine plage en choisissant une valeur de rouge (Red), vert (Green) et bleu (Bleu), dans le spectre RGB, comme sur la photo ci-contre :



- `SetPixel(Hdc, abscisse, ordonnée, COULEUR);`

Enfin, cette fonction va modifier un pixel suivant les informations en arguments.

IV- La classe CPolynome

Toutes les informations propres aux polynômes seront dans une classe. Cette classe prendra donc en argument un nombre entier (le nombre de polynômes), un tableau d'entiers (répertoriant ne nombres de coefficients de chaque polynôme) et un tableau double de réels (les coefficients de chaque polynôme).

Il faut définir les fonctions essentielles pour le fonctionnement d'une classe, soit le constructeur par défaut (qui prendra 0 comme nombre de polynômes et qui associera les deux tableaux à la « valeur » NULL), le constructeur avec des arguments, le constructeur copie et le destructeur.

Il faudra également créer des fonctions amies, soit les surcharges ses opérateurs << et >> de telle sorte que dans le fichier source.cpp, il y ait juste besoin de créer un objet de type CPolynome, de faire cin et cout sur cet objet pour pouvoir tracer les courbes que l'on souhaite.

On créera également une fonction dans le fichier source afin d'afficher le repère sur lequel seront ensuite tracées les courbes.

V- Les axes

Afin de pouvoir afficher les axes, on dispose à la fin de la surcharge <<, des boucles for. On affiche deux droites perpendiculaires. J'ai fait le choix d'afficher les axes sur la droite de la fenêtre de débogage, étant donné que les écritures se placent par défaut sur la gauche. Les axes seront de couleur violette, donc un mélange de bleu et de rouge afin de bien différencier avec les courbes qui seront-elles en nuances de vert.

VI- Les surcharges

Dans le but d'avoir un programme final ludique comme demandé dans le sujet, les surcharges permettent de pouvoir entrer des polynômes et de les afficher simplement à l'aide des commandes cin et cout.

La surcharge >>

Cette surcharge de l'opérateur >> a pour but de pouvoir entrer des polynômes dans un objet de type CPolynome, en procédant de la manière suivante :

- Supprimer les éventuelles informations déjà stockées dans les tableaux
- Demander à l'utilisateur le nombre de polynômes
- Initialiser les deux tableaux en conséquence

Puis, pour chaque polynôme :

- Demander le nombre de coefficients (stockés dans le tableau prévu à cet effet)
- Initialiser la seconde dimension du tableau contenant les coefficients en conséquence
- Demander de spécifier chaque coefficient en fonction de son degré.

La surcharge <<

Cette surcharge de l'opérateur << a pour but d'afficher des polynômes stockés dans un objet de type CPolynome, en procédant de la manière suivante :

- Afficher les équations de tous les polynômes qui seront représentés
- Effectuer la stratégie énoncée dans la partie « plan d'action » pour chaque polynôme
- A chaque polynôme, on associe une nuance de vert (de plus en plus foncé) grâce à la commande RGB, on fixe le taux de vert à sa valeur maximale (255),

et on fait diminuer le taux de rouge et de bleu en fonction du numéro associé au polynôme par pas de 30 (ce qui fait donc un maximum de 7 polynômes affichés simultanément en plaçant la valeur de départ à 210, valeur bien entendue modifiable, mais pas par l'utilisateur), sachant que les courbes s'affichent également au-delà de 7, mais on ne peut plus les différencier aussi facilement.

VII- Les courbes

Afin de tracer les courbes, on a dit auparavant que l'on choisissait une plage de x sur laquelle représenter les points des courbes, le fait est que si on choisit une plage suffisamment grande pour bien distinguer les courbes, on ne verrait pas forcément la plage qui nous intéresse (la plupart du temps, on n'étudie pas les courbes pour des x variant entre + et -500). On multiplie alors les abscisses et les ordonnées par un coefficient commun (j'ai choisi ici 40). On a alors un rendu bien meilleur et des courbes tracées par défaut entre -10 et 10, le plus souvent suffisant pour étudier des polynômes. Il faut également, cette fois-ci à l'aide d'une boucle if, faire en sorte que les courbes ne puissent pas dépasser d'un certain cadre (imaginaire), pour une question esthétique.

VIII- Les équations

L'idée de donner les équations des polynômes rentrés m'est venue lorsque je me suis rendu compte que ma surcharge de l'opérateur << ne contenait aucun « os << », l'idée étant que si j'affichait les équations en utilisant cette commande os, je pourrais à la fin de ma surcharge mettre un return os. C'est dans le but d'afficher du mieux possible les équations que j'ai créé la fonction membre abs, qui renvoie la valeur absolue d'un réel (pour afficher 1 – x au lieu de 1 + -x). Avec des boucles for, j'affiche pour chaque polynôme les coefficients multipliés par x à la puissance correspondante. J'ai également fait en sorte que les x^0 ne s'affichent pas.

IX- Ajustement du graphique

Cette partie là de mon programme ne devait à la base même pas exister. Lorsque l'on affiche un certains nombre de courbes, je me suis rendu compte que la fenêtre de débogage commence à défiler. Donc le graphique, avec tout ce qu'il contient, s'affichait mal voire pas du tout car je l'avais alors défini pour un placement fixe.

En me demandant comment faire le lien entre le nombre de lignes utilisées dans la fenêtre et le placement, je me suis d'abord dit que je devais utiliser le nombre total de coefficients comme référence pour abaisser les ordonnées. Cette méthode étant approximative, j'ai ensuite décidé d'étudier plus en détail la fenêtre de débogage. Les éléments que j'ai étudiés sont :

- Les lignes d'écriture qui sont permanentes (9)
- Le nombre de lignes sur une seule fenêtre de la fenêtre de débogage (sans scroller du tout) (50).
- Le nombre de lignes par défaut utilisées par un polynôme (4)
- Le nombre de lignes utilisées par un coefficient (2)

J'ai alors défini une variable entière « ajord » (pour ajustement ordonnée) qui prend comme valeur par défaut -41 (soit le nombre de lignes sur une fenêtre moins le nombre de lignes permanentes) avec comme idée que si ce nombre était négatif ou nul, les ordonnées resteraient à leur placement par défaut.

A ce nombre on additionne alors le nombre de polynômes multiplié par 4 et le nombre de coefficients total multiplié par 2 (donc on multiplie par le nombre de lignes utilisées).

Il y a donc finalement trois éventualités finalement pour ce nombre :

- Il est négatif, auquel cas avec une boucle if on lui associe la valeur 0, car les ordonnées n'ont pas besoin d'être modifiées.
- Il est nul, et donc il n'agit pas sur les ordonnées.
- Il est positif. Dans ce cas précis, cela signifie que les ordonnées ont besoin d'être modifiées car la fenêtre de débogage va être amenée à défiler. Je me suis rendu compte que si on multiplie cette valeur par 5, alors les ordonnées s'ajustent plutôt bien en fonction du nombre de lignes utilisées.

X- Détails importants

Tous ce que j'ai mis dans la catégorie des détails importants sont des éléments importants à comprendre dans mon programme qui ne correspondaient à aucune des autres parties de mon compte-rendu, ou bien à des éléments que j'ai rajoutés par la suite afin d'avoir un programme plus optimisé.

- On demande à l'utilisateur de passer la fenêtre de débogage en mode plein écran car le programme fonctionne tel que la courbe sera bien visible et complète si la fenêtre est en plein écran.

- Le débogueur affiche par défaut un message de fin, si l'on ne fait rien, ce message peut s'afficher sur la courbe. Afin d'éviter ceci, on insère une ligne à la fin de la surcharge << qui va passer un certain nombre de lignes afin que ce message ne dérange pas.

- Les ordonnées sont par défaut vers le bas avec les fonctions utilisées, il faut donc multiplier chaque élément en ordonnée par -1 si l'on souhaite avoir un rendu « classique ».

XI- Conclusion

Le programme fonctionne plutôt bien. Ce projet m'a beaucoup plu et j'y ai consacré un certain temps, pas forcément par obligation mais car j'ai apprécié le travail que j'ai fourni. Depuis la lecture des sujets jusqu'à la rédaction de cette conclusion, ce travail m'a apporté de l'autonomie en ce qui concerne les recherches à propos de mon travail. J'ai également su trouver le moyen d'améliorer mon programme dans le temps imparti autant que possible. Je trouve ça réellement intéressant de travailler sous la frome de projets, que ce soit dans le cadre de l'informatique ou bien même des autres matières, car c'est plus concret. Je suis finalement satisfait de mon programme même si je pense bien qu'il n'est pas parfait.