GAUTIER Arthur

L1 Promotion 2018

Groupe A

02/04/2014

Rapport de Projet

Programmation en C

Fractales

Arthur G.

EFREI, ECOLE D'INGENIEUR EN INFORMATIQUE

Arthur GAUTIER
Groupe A
L1 Promo 2018
Rapport Projet C

Table des matières

Introduction

Dans ce troisième projet, réalisé dans le cadre de ma formation à l'algorithmique et à la programmation en C (et C++), il m'a été demandé de réaliser un programme permettant de générer des images fractales, spécialisé dans l'ensemble de Mandelbrot. Ce programme doit pouvoir traiter les nombres complexes, tester si un nombre complexe appartient ou non à l'ensemble de Mandelbrot et dessiner le dit-ensemble (en couleurs). En bonus, il était proposé (entre autres) d'afficher la fractale à l'aide d'une librairie graphique, de pouvoir zoomer sur une fenêtre donnée, de produire un GIF de ce zoom ou encore d'ouvrir l'horizon du programme pour permettre l'affichage d'autres fractales.

Pour ma part, j'ai choisi de concentrer mes efforts sur la possibilité pour l'utilisateur de choisir lui-même les couleurs de sa fractale ainsi que sur l'affichage des ensembles de Julia.

Ce rapport va se décomposer en trois parties. Une première partie sera dédiée à l'analyse générale du programme. Dans cette partie on expliquera les principales fonctionnalités du programme. Après cela, la seconde partie détaillera de façon précise le programme en précisant comme l'on obtient chaque fonctionnalité. Enfin, la troisième partie sera une annexe qui détaillera le fonctionnement de l'extension que j'utilise afin d'obtenir une image, le format pbm.

I. Analyse générale

Un des éléments qui a énormément facilité la programmation de ce projet est l'utilisation d'une structure appelée « complexe » pour permettre de gérer non seulement les nombres complexes mais aussi certaines variables qui se trouvaient être utilisées pour représenter l'abscisse et l'ordonnée de points, ou encore la hauteur et la longueur de la fenêtre.

On peut rassembler les fonctionnalités du programme en trois grandes parties : les affichages faits en console, la partie relative aux couleurs et la partie relative aux ensembles.

A. La console

Dans ce projet, j'ai choisi d'utiliser la console comme un intermédiaire entre l'utilisateur et le programme. En effet, les volontés de l'utilisateur (par rapport aux points sur lesquels il peut agir) seront toutes demandées à l'aide de la console.

La première question posée à l'utilisateur est celle de la résolution de l'image. Pourquoi avoir permis à l'utilisateur de choisir lui-même la résolution qu'il désire ? Pour que mon programme puisse fonctionner sur tous types de machines et que chacun puisse afficher une image à la hauteur des capacités de sa machine. En effet, si vous faites tourner le programme sur une machine très puissante, dotée d'une formidable puissance de calcul, libre à vous de demander au programme une fractale ayant une résolution de 53 333*30 000 ou plus encore (seule la hauteur sera demandé à l'utilisateur, la longueur sera calculée automatiquement en supposant que votre écran est calibré en format 16/9) !

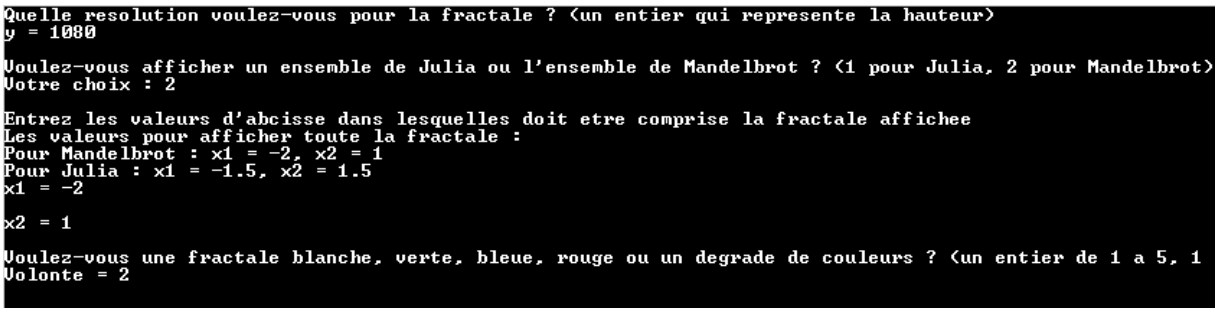
Après cela, le programme propose à l'utilisateur de choisir s'il désire afficher un des ensembles de Julia ou bien l'ensemble de Mandelbrot. Là, il s'agit d'un des choix cruciaux du programme. Libre à l'utilisateur de choisir l'un ou l'autre, par la suite certaines questions n'apparaîtront que si l'utilisateur choisit l'ensemble de Julia.

Viens ensuite la question des coordonnées du plan complexe dans lequel apparaîtra la fractale. Afin de garder une image propre, le programme ne demande à l'utilisateur que les abscisses, il calcule lui-même les ordonnées à l'aide d'un calcul de proportionnalité. Ici, l'utilisateur a la possibilité de paramétrer un zoom en modifiant les valeurs des abscisses du plan. Toutefois, ce zoom n'est pas interactif puisqu'il est nécessaire de relancer le programme à chaque fois. De plus, zoomer de façon précise est particulièrement dur puisqu'il est nécessaire de connaître précisément les valeurs des abscisses dans le plan complexe pour que le zoom soit fait exactement là où l'utilisateur le veut.

Maintenant, le prochain affichage dépend de la réponse de l'utilisateur à la question de la fractale qu'il désire afficher. Si il a demandé l'affichage de l'ensemble de Mandelbrot, le programme lui demande directement les couleurs qu'il désire pour sa fractale. Cependant, s'il a demandé l'affichage d'un des ensembles de Julia, le programme, après avoir affiché certaines valeurs pour lesquels il existe des ensembles intéressants, va proposer à l'utilisateur de rentrer les coordonnées (dans le plan complexe) du point duquel il désire afficher l'ensemble de Julia qui lui correspond (tous les points n'ont cependant pas un ensemble visible qui leur correspond).

Pour finir, la dernière question posée à l'utilisateur est le choix des couleurs. Le programme laisse le choix à l'utilisateur de la couleur dans laquelle (ou lesquelles) il veut afficher la fractale. Si l'utilisateur choisit une fractale monochromatique (blanche, verte, bleue ou rouge), le programme passe directement au calcul et en une minute environ (pour une résolution de 1920*1080), il produit une image de la fractale demandée. Mais l'intérêt intervient dans le cas où l'utilisateur demande un dégradé de couleurs. Le programme va alors demander à l'utilisateur de rentrer, partie par partie, le code couleur des cinq couleurs qu'il tient à avoir sur sa fractale.

Voilà à quoi ressemble la console lorsqu'on demande l'affichage de l'ensemble de Mandelbrot dans ses dimensions originelles (-2 à 1 en abscisse et -1 à 1 en ordonnées), et cela en vert :



B. Les couleurs

Dans cette partie du programme, on trouve deux fonctions : la fonction qui sert à demander et à stocker les couleurs demandées à l'utilisateur et la fonction qui va associer à un point sa couleur. Ici, je ne vais parler que de la seconde fonction et uniquement en la survolant, l'analyse détaillée surviendra en seconde partie.

Ici, les choix faits par la fonction dépendent entièrement de la demande de l'utilisateur. Un seul élément reste constant : si les fonctions vérifiant l'appartenance du point à l'ensemble que le programme cherche à tracer prouvent que le point fait partie de l'ensemble, il sera tracé en noir.

Si l'utilisateur a demandé une fractale verte, bleue ou rouge, on affecte au point un dégradé de la couleur demandée. S'il a demandé du blanc, on affecte un dégradé du blanc. Enfin, s'il a demandé un dégradé de couleurs, on affecte au point une des couleurs en fonction du nombre de tours de boucles nécessaires à montrer que le point n'appartient pas à l'ensemble.

C. Les ensembles

Cette partie du programme sert à gérer l'appartenance d'un point à l'un des deux ensembles. En fonction de ce que l'utilisateur a demandé (l'ensemble de Mandelbrot ou un ensemble de Julia), le programme va exécuter la fonction correspondante et va vérifier si quels points appartiennent à l'ensemble qu'il veut tracer. La principale différence entre les deux fonctions revient à la différence entre l'ensemble de Mandelbrot et les ensembles de Julia (en général), l'ensemble de Mandelbrot est un ensemble de Julia particulier pour une valeur de c particulière. Ainsi, dans la fonction vérifiant l'appartenance à l'ensemble Mandelbrot, la valeur de c est fixée directement dans et par le programme. Dans la fonction vérifiant l'appartenance à un ensemble de Julia, c'est l'utilisateur qui va fixer lui-même cette valeur de c.

II. Analyse détaillée

Ici, je vais reprendre les deux dernières sous-parties de l'analyse générale afin de détailler de façon précise le fonctionnement de chaque fonction : ce à quoi elle sert, ce dont elle a besoin, ce qu'elle modifie et ce qu'elle renvoie.

Je vais essayer d'illustrer le maximum de mes explications d'algorithmes reprenant le fonctionnement de ce que j'explique.

A. Les couleurs

Comme je l'ai dit précédemment, il y a deux fonctions dans la partie relative aux couleurs. La première est utilisée dans le cas où l'utilisateur demande un dégradé de couleurs pour afficher la fractale. La seconde est appelée pour associer à chaque point une couleur.

Commençons par étudier la première fonction. Cette fonction prend cinq paramètres en entrée, paramètres qui représentent l'adresse des cinq tableaux qui stockeront les cinq codes couleurs RGB entrés par l'utilisateur. Ces tableaux sont créés dans la fonction principale et on leur alloue dynamiquement la capacité de stocker 13 caractères (taille maximale d'un code RGB standard). Je vais maintenant détailler ce qu'il y a à l'intérieur de la fonction, et plus précisément une seule partie, étant donné que la structure se répète cinq fois (une fois par couleur à entrer).

Le programme commence par expliquer à l'utilisateur ce qu'il va rentrer et ce qu'il doit rentrer. Après cela, on entre dans une suite de trois boucles faire … tant que qui demande la saisie des trois valeurs de la première couleur. En voici l'algorithme :

```
Faire
{
        // Affichage des instructions
        saisir (valeur1);
}
Tant que ((valeur1 < 0) OU (valeur1 > 255));
couleur1 [0] <- valeur1;
// Saisie de la seconde valeur et stockage dans un tableau temporaire
concatener (couleur1, temporaire);
// Saisie de la troisième valeur et stockage dans le tableau temporaire
concatener (couleur1, temporaire) ;
// On a alors dans degrade1 (qui est un tableau de caractère) le code RGB de la
première couleur voulue par l'utilisateur
```

Cette structure de boucles et d'instructions est répétée pour chacune des cinq couleurs que doit rentrer l'utilisateur.

Ainsi, cette fonction modifie directement le contenu de chacun des tableaux fournis en entrée à la fonction en leur fournissant à chacun le code RGB de chacune des couleurs demandées par l'utilisateur. La fonction modifiant directement le contenu des tableaux, elle n'a pas besoin de sortie et ne renvoie donc aucune donnée.

Intéressons-nous désormais à la seconde fonction de cette partie : la fonction qui va permettre l'affectation des couleurs à chaque point.

Cette fonction prend huit paramètres en entrée. Il y a bien sûr les adresses des tableaux dont j'ai parlé dans la partie précédente et qui ont été saisis à l'aide la fonction précédente. Ces tableaux seront utiles en cas de demande de dégradé par l'utilisateur. Mais il y a aussi un paramètre de type FILE* ce qui correspond à un fichier, fichier dans lequel la fonction va écrire la couleur de chacun des points. Pour savoir de quelle(s) couleur(s) devra être la fractale, on fournit également en paramètre le choix fait par l'utilisateur pour la couleur. Enfin, le dernier paramètre dont a besoin la fonction est le nombre d'itérations réalisé par les boucles vérifiant si le point appartient ou non à l'ensemble que le programme cherche à tracer.

On doit séparer cette fonction en deux parties : si le point appartient à l'ensemble et s'il ne lui appartient pas. Si le point appartient à l'ensemble, c'est simple, le programme écrit dans le fichier que la couleur de ce point sera noir (0 0 0 en RGB). Cependant, si le point ne lui appartient pas, il y a trois possibilités selon la couleur demandée.

Si l'utilisateur a demandé du rouge, du bleu ou du vert, on définit une variable appelée couleur qui va, en fonction du nombre d'itérations, représentée une teinte plus ou moins sombre de la couleur demandée. Cette valeur va être écrite dans le fichier.

Voilà l'algorithme :

```
couleur <- 55 + iteration % 170;
// Admettons que l'utilisateur veuille du bleu
ecrirefichier (fichier, « 0 0  » couleur) ;
```

Avec cela, on perçoit un dégradé dans la couleur, celle-ci se faisant plus forte et plus clair à proximité de la fractale et s'estompant et s'assombrissant lorsque l'on s'en éloigne.

Si l'utilisateur a demandé du blanc, on définit la variable couleur différemment, et on écrit dans le fichier trois fois la même valeur, donnant ainsi un dégradé de blanc tendant vers le gris à proximité de la fractale.

Enfin, si l'utilisateur a demandé un dégradé de couleur, on applique un principe de dégradé linéaire : entre tant et tant d'itérations, on applique la couleur 1, puis la couleur 2 entre tant et tant etc … jusqu'à revenir sur la première couleur puis atteindre la valeur maximale d'itérations. Pour un nombre d'itérations maximale de 60 et 3 couleurs, voilà ce à quoi pourrait ressembler l'algorithme :

```
Si ((iteration >= 1) ET (iteration < 20)
{
        ecrirefichier (fichier, couleur1) ;
}
Sinon si ((iteration <= 20) ET (iteration < 40))
{
        ecrirefichier (fichier, couleur2) ;
}
Sinon
{
        ecrirefichier (fichier, couleur3) ;
}
```

Enfin, cette fonction écrivant directement dans le fichier, elle n'a pas non plus besoin de sortie.

B. Les ensembles

Cette partie du programme est sûrement la plus intéressante puisque c'est celle qui va vérifier pour chacun des points si oui ou non, il appartient à l'ensemble que l'utilisateur à demander au programme de tracer.

Je vais présenter une seule des deux fonctions, celle pour l'ensemble de Mandelbrot, la seconde ne différant que très peu.

Cette fonction a huit paramètres. Quatre d'entre eux représente les valeurs minimales et maximales en abscisses et ordonnées du plan complexe dans lequel on va tracer la fractale. Deux de ces paramètres représentent la résolution de l'image en hauteur et en longueur. Enfin, les deux derniers représentent le point de l'écran que l'on est en train de tester.

On définit deux variables de type complexe dans cette fonction : l'une servant à la transcription du point de l'écran vers un point du plan complexe, la seconde représentant le point en question. Dans cette fonction, c'est le point du plan complexe que l'on tester qui va varier, dans la fonction pour les ensembles de Julia, on fixe ce point et on fait varier la seconde variable.

La vérification de l'appartenance ou non du point à l'ensemble se fait à l'aide d'une boucle faire … tant que dont voici l'algorithme :

```
Faire
{
        tmp <- complexe.reelle ;
        complexe.reelle <- (complexe.reelle)^2 – (complexe.imagin)^2 + c.x ;
        complexe.imagin <- 2 * tmp * complexe.imagin + c.y ;
        iteration <- iteration + 1

}
Tant que ((module(complexe) < 2) && (iteration < iteration_max)) ;
```

La meilleure façon d'avoir une image la plus précise possible est bien entendu d'augmenter le nombre maximum d'itérations. Cependant, plus ce nombre est grand, plus le temps de calcul est long. Ainsi, c'est pourquoi j'ai fixé ce nombre à 500 de façon arbitraire.

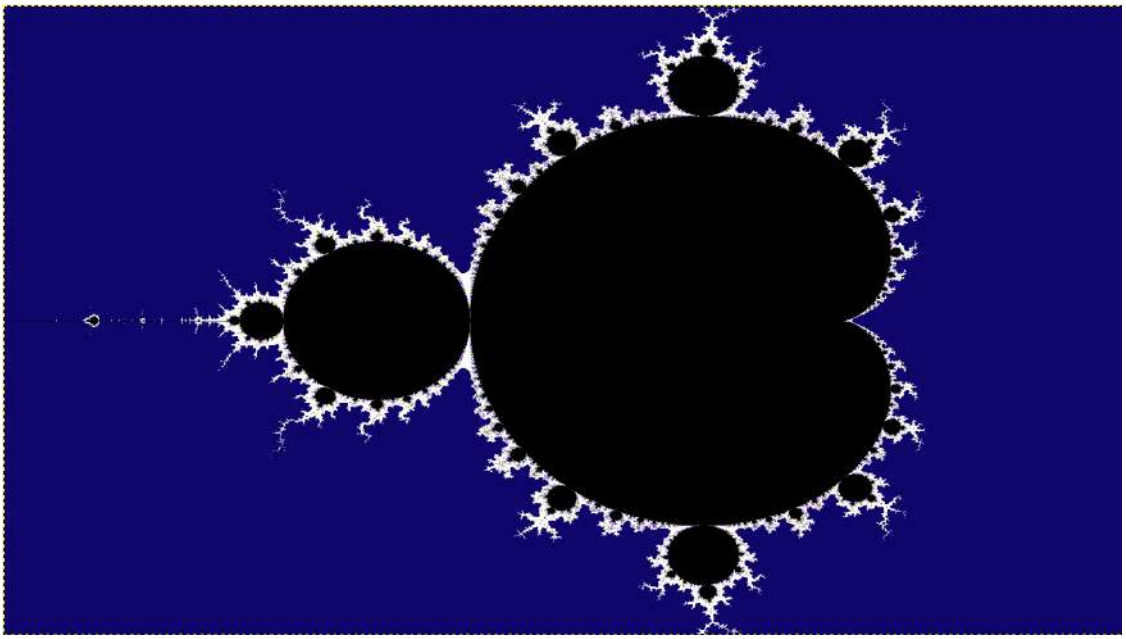
Cette fonction a une sortie, elle retourne -1 si le nombre d'itérations maximum a été atteint et elle retourne le nombre d'itérations sinon.

III. Conclusion

Dans ce projet, on m'a demandé de programmer un outil pouvant générer des images fractales. La première difficulté que j'ai rencontrée a été de réussir à transcrire les pixels de l'écran en points du plan complexe cela n'étant pas évident au premier abord. Mais, grâce à Mr. Flasque et son amphithéâtre sur le projet, j'ai réussi à approcher ce problème sous un nouvel angle et à trouver la solution. Le second problème que j'ai eu a été de dompter l'extension que j'emploie afin de stocker mes informations, celle-ci m'étant inconnue avant ce projet, il m'a été nécessaire de réfléchir à comment faire en sorte que je m'en sorte au mieux. Enfin, le plus gros problème qui s'est posé à moi a été de réussir à permettre la saisie des couleurs par l'utilisateur. Cependant, grâce à un tutoriel sur les chaînes de caractères trouvables sur OpenClassRoom, j'ai découvert la nouvelle fonction qu'est sprintf ce qui m'a permis de surmonter cette difficulté.

De plus, même si mon programme n'est pas optimisé au possible, personnellement, je le trouve satisfaisant et représentatif de mes capacités de programmeur.

Pour conclure, ce projet m'aura permis de découvrir ce qu'était réellement une fractale mais également de quelle manière elles étaient dessinées ainsi que de nouvelles possibilités de programmation (avec notamment l'utilisation du pbm).



Annexe

Le PNM

Le PNM qu'est-ce que c'est ?

Le PNM (ou portable anymap) est un ensemble de trois formats de fichiers graphiques. Ils ont été définis par le projet NetPBM et ont été développés par Jef Poskanzer dans les années 1980.

Pour ma part, dans ce projet, j'ai utilisé le format PPM (pour Portable PixMap) (extension : .ppm). Ce format est utilisé pour des images couleur. Il en existe deux variantes : la variante binaire ou la variante ASCII (que j'ai utilisé).

Un fichier PPM ASCII se compose de la façon suivante :

- Un nombre dit « magique » (ici, P3) suivi d'un passage à la ligne : ce nombre est demandé par le format et ne peut donc pas être modifié.
- Le nombre de colonnes dans l'image.
- Suivi du nombre de ligne, lui-même suivi par passage à la ligne.
- La valeur maximale utilisée pour coder les couleurs (ici, 255 qui signifie en plus le codage en RGB), suivie d'un passage à la ligne.
- Le codage de chaque pixel, codé par trois valeurs (RGB).

Pourquoi avoir choisir le format PPM ?

J'ai choisi de travailler avec ce format puisqu'il était conseillé dans le sujet du projet, et que même si je ne le connaissais pas, il me semblait relativement simple à maitriser.