TP n°10



Prolégomènes

Ce sujet de TP permet de s'entraîner sur la manipulation de types structurés et l'écriture dans un fichier en C.

Le code ainsi produit est à envoyer par mail à civ.info.mp2i@gmail.com avant le **vendredi 30 octobre à 17h00**.



En 1980, **Benoît Mandelbrot** utilise un programme informatique pour visualiser pour la première fois un ensemble qui le rendra célèbre.

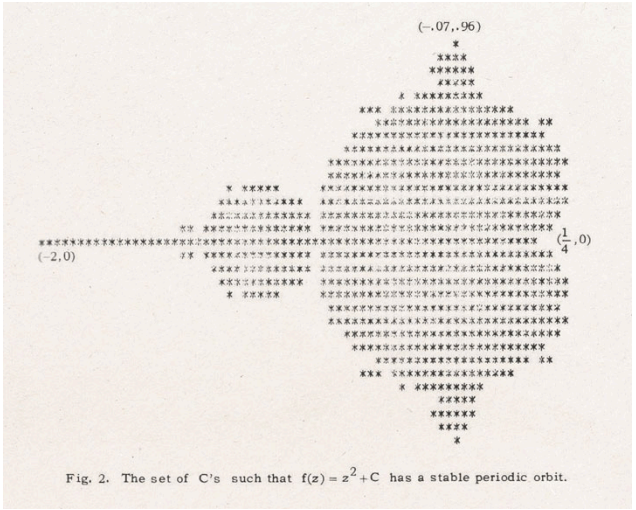


Fig. 2. The set of C's such that $f(z) = z^2 + C$ has a stable periodic orbit.

I. Ensemble de Mandelbrot

Suite d'itération de Mandelbrot

Pour un nombre complexe c, on définit la suite de nombres complexes $\left(U_n^{(c)}\right)_{n\in\mathbb{N}}$ par :

$$U_0^{(c)} = 0 \qquad \text{et} \qquad \forall n \in \mathbb{N},\quad U_{n+1}^{(c)} = \left(U_n^{(c)}\right)^2 + c$$

On remarque que pour certaines valeurs de c, la suite reste bornée alors que pour d'autres elle s'écarte arbitrairement loin de l'origine.

1 Proposer, en justifiant, une autre valeur de c telle que $(U_n^c)_{n\in\mathbb{N}}$ reste bornée, et une autre telle que la suite diverge.

Ensemble de Mandelbrot

L'ensemble de Mandelbrot est l'ensemble des nombres complexes c tels que $U_n^{(c)}$ reste bornée.

II. Une implémentation des nombres complexes

2 Proposer et définir un type structuré pour modéliser un nombre complexe, avec un champ pour sa partie réelle et un champ pour sa partie imaginaire. Utiliser `typedef` pour renommer votre type structuré en `complexe`.

3 Implémenter une fonction de prototype `complexe zero()` qui renvoie le complexe nul.

4 Implémenter une fonction de prototype `float module_carré(complexe z)`, qui prend en argument un nombre complexe et qui renvoie son module au carré.

5 Implémenter une fonction de prototype `complexe somme(complexe z1, complexe z2)` qui renvoie la somme de deux nombres complexes.

6 Implémenter une fonction de prototype `complexe produit(complexe z1, complexe z2)` qui renvoie le produit de deux nombres complexes.

7 Implémenter une fonction de prototype `complexe carre(complexe z)` qui renvoie le carré d'un nombre complexe.

III. Appartenance à l'ensemble de Mandelbrot

L'objectif de cette partie est d'aboutir à une fonction qui détermine si un nombre complexe c appartient à l'ensemble de Mandelbrot ou non. Il n'est cependant pas possible de calculer tous les termes de la suite $(U_n^c)_{n\in\mathbb{N}}$ pour déterminer son comportement, puisqu'il y en a une infinité. Nous allons ici adopter un critère de divergence simplifié.

8 Montrer que si l'on atteint un terme U_n^c tel que $|U_n^c| > 2$, alors la suite $(U_n^c)_{n\in\mathbb{N}}$ est divergente.

9 Implémenter une fonction de prototype `int echappement(complexe c)` qui calcule les 255 premiers termes de la suite $(U_n^c)_{n\in N}$ et qui renvoie l'indice k du premier terme U_k^c tel que $|U_k^c| > 2$. Si aucun des 255 premiers termes ne vérifie ce critère, la fonction renvoie alors 255 et on supposera que c est dans l'ensemble de Mandelbrot.

10 Implémenter une fonction de test illustrant le résultat de la fonction `echappement` sur quelques exemples dont ceux que vous avez proposés. Obtenez-vous les résultats attendus ?

IV. Le format d'image PGM

Il existe de nombreux formats d'images, les plus connus étant **PNG** et **JPEG**. Dans ce TP, nous utiliserons un autre format nommé **PGM** (pour Portable GrayMap file format) qui stocke une image sous forme d'un texte lisible pour un humain. Il est donc plus simple de générer ce type d'images, au prix d'un fichier beaucoup plus lourd à la fin. Ce qui suit n'est pas à connaître par coeur, mais sera utilisé dans ce TP pour générer des images en suivant précisément les conventions données. Les noms de fichiers devront se terminer par l'extension **.pgm**.

- La première ligne d'un fichier PGM contient uniquement le code **P2**, permettant de préciser le format;
- la deuxième ligne contient un entier indiquant la largeur de l'image en pixels, puis un espace, puis un second entier indiquant la hauteur de l'image en pixels;
- la troisième ligne contient un entier indiquant le nombre de couleurs utilisées, ici ce sera 255;
- les lignes suivantes contiennent les valeurs des pixels. On donne un entier (entre 0 et 255) pour chaque pixel, pour indiquer sa couleur. On représente les lignes de l'image de haut en bas. On peut séparer ces entiers par des espaces et des sauts de lignes, mais ici on fera un retour à la ligne après chaque entier (il y a donc une ligne par pixel différent).

Exemple d'image PGM

```
P2
11 7
255
0   0   0   0   0   0   0   0   0   0   0
0 200   0 100 100 100   0  20 130 120   0
0 200   0 100   0 100   0  30   0 110   0
0 200   0 100   0 100   0  40 100 100   0
0 200   0 100   0 100   0  50   0  90   0
0 200   0 100 100 100   0  60  70  80   0
0   0   0   0   0   0   0   0   0   0   0
```

→

Image correspondant à ce fichier **.pgm**



V. Génération d'images et de vidéos

L'objectif de cette partie de générer une image de l'ensemble de Mandelbrot, en appelant la fonction `echappement` sur chaque pixel de l'image pour voir si le complexe associé est dans l'ensemble ou non. Commençons par définir deux constantes globales indiquant les dimensions de l'images qui sera générée :

```c
const int TAILLE_X = 1920;
const int TAILLE_Y = 1280;
```

11 Dans quel zone mémoire ces variables sont-elles stockées ?

12 Implémenter une fonction de prototype `void genere_image(char* nom_de_fichier, complexe centre, float zoom)` qui ouvre un fichier et y écrit les trois premières lignes d'un fichier PGM valide à l'aide d'un seul appel à `fprintf`. Cette fonction sera complétée par la suite.

13 Implémenter une fonction de prototype `complexe decalage(int x, int y, float zoom)` qui renvoie le complexe ayant pour partie réelle `(TAILLE_X / 2 - x) / zoom` et pour partie imaginaire `(TAILLE_Y / 2 - y) / zoom`. Il s'agit du décalage d'un pixel par rapport au centre de l'image.

14 Compléter votre fonction `genere_image` en ajoutant deux boucles imbriquées parcourant tous les pixels de l'image. Pour chaque pixel, il faut calculer le décalage associé, l'ajouter au complexe `centre`, puis calculer l'échappement du complexe obtenu et écrire cette valeur dans le fichier (suivi d'un retour à la ligne).

15 Lancer votre fonction sur le centre 0.5 + 0i avec un zoom de 500. Contempler le résultat produit en ouvrant le fichier ainsi généré depuis l'explorateur de fichiers. Ne pas l'ouvrir sur VSCodium.

16 Voici un petit compendium de jolis coins à aller visiter. Pour chacun, générer une image et l'ouvrir :
- $-0.7436 + 0.1318i$ avec un zoom de 10^5, puis 10^6, puis 10^7;
- $0.2800 + 0.0080i$ avec un zoom de 10^5, puis 10^6;
- $-0.7453 + 0.1865i$ avec un zoom de 10^5, puis 10^6;

17 Proposer une nouvelle fonction permettant de générer n images centrées sur le même complexe, chaque image étant obtenu en multipliant le zoom de la précédente par 1.1. On utilisera `snprintf(buf, sizeof(buf), "animation/%d.pgm", k);` pour écrire dans `buf` (un tableau de 100 caractères) le nom de la $k$ème image. Une fois les n images générées, utiliser la commande `ffmpeg -i %d.pgm animation.gif` dans le bon dossier pour produire une vidéo.

VI. Concours "prix de la plus belle image"

18 Envoyer par mail votre soumission pour le concours de la plus belle image d'une fractale. À vous d'explorer pour trouver le plus bel endroit ! Il est possible de jouer sur l'affichage (regarder le format PPM pour de la couleur). Il est possible d'utiliser une autre suite d'itération. Il est possible d'aller regarder la définition des ensembles de Julia sur Internet. Il est interdit d'éditer/retoucher l'image, seul le code C doit être envoyé et je me chargerai de le lancer moi-même pour récupérer l'image.