

SINF1252 - Fractales

2018

1 Introduction

Les fractales sont des objets mathématiques fascinants: elles exhibent une forme que l'on retrouve à toutes les échelles. Autrement dit, il est possible de zoomer à l'infini sur une fractale et de toujours retrouver le même modèle qui se répète. Ces formes se retrouvent dans la nature, par exemple dans les rivières, les arbres, etc. Les fractales peuvent être générées de plusieurs manières différentes, et il existe plusieurs ensembles de fractales. Pour ce projet, nous allons nous intéresser à l'ensemble de Julia¹.

Les fractales de l'ensemble de Julia se construisent par une relation de récurrence définie comme suit:

$$z_{n+1} = z_n^2 + c$$

où z et c sont des nombres complexes. Il existe une fractale de Julia pour chaque point $c = a + bi$ du plan complexe où $a, b \in [-1; 1]$. Concrètement, pour générer une fractale donnée sur une image de w pixels de large et h pixels de haut, il faut procéder comme suit. Pour chaque coordonnée (x, y) de l'image, on normalise les coordonnées pour les faire rentrer dans l'intervalle $[-1.5; 1.5]$, qui donne une bonne vue d'ensemble de la fractale sur l'image générée. Ensuite, pour chaque coordonnée normalisée, on calcule la relation de récurrence jusqu'à ce que les coordonnées normalisées s'éloignent de plus de deux unités de l'origine. Auquel cas, le nombre d'itérations effectuées est retourné. Il se peut également que l'on reste à l'infini dans ce cercle de rayon 2, auquel cas on définit une limite maximale d'itérations. Si cette limite est atteinte, on retourne la valeur zéro. Au final, on obtient un nombre d'itérations pour chaque pixel de l'image. Il reste à transformer cette valeur en couleur RGB et le tour est joué.

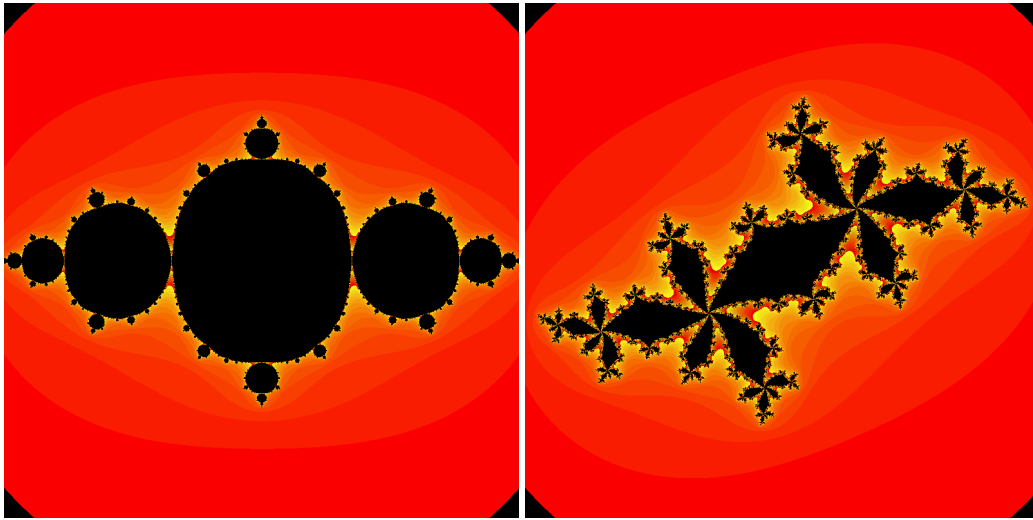


Figure 1: Exemple de fractales. La première est générée pour $c = -0.8 + 0.0i$ et la deuxième pour $c = -0.52 + 0.57i$. La fractale de droite contient les couleurs générées "naturellement" par la fonction de transformation.

¹https://fr.wikipedia.org/wiki/Ensemble_de_Julia

2 Énoncé

📌 **Important:** le choix du binôme est conditionné par la participation aux tâches Ingenious. Les étudiants ayant 0 à ces tâches **doivent** soit s'associer avec d'autres étudiants ayant également 0 à ces tâches, soit réaliser le projet seuls.

2.1 Description du projet

En binôme, vous devez implémenter, en langage C, un programme prenant en entrée (dans un ou plusieurs fichiers ou sur l'entrée standard) une liste de fractales (sous forme de coordonnées) et générant en sortie un fichier BMP représentant la fractale dont la moyenne des valeurs de chaque point est la plus élevée. Votre programme doit donc pouvoir effectuer les opérations suivantes:

- Charger les descriptions des fractales depuis une source, que ça soit l'entrée standard ou un fichier
- Calculer les fractales
- Déterminer la fractale dont la moyenne des valeurs de chaque point est la plus élevée

Vous devez baser l'architecture de votre programme sur un modèle **multi-thread**.

Pour vous aider dans cette tâche, nous vous fournissons un squelette de code, une API à respecter, ainsi qu'un exemple d'entrée et le résultat attendu.

Une fractale est représentée par une structure `struct fractal`. Seul le nom de la structure est imposé, son contenu devra être rempli par vos soins. Les fonctions à implémenter sur cette structure sont les suivantes:

- `fractal_new()`: alloue une nouvelle `struct fractal`
- `fractal_free()`: libère la mémoire correspondant à une `struct fractal`
- `fractal_get_name()`: retourne le nom de la fractale
- `fractal_get_value()`: retourne la valeur correspondant à un pixel de l'image
- `fractal_set_value()`: définit la valeur correspondant à un pixel de l'image
- `fractal_get_width()`: retourne la largeur de l'image, en pixels
- `fractal_get_height()`: retourne la hauteur de l'image, en pixels
- `fractal_get_a()`: retourne la partie réelle des coordonnées de la fractale ($c = a + bi$)
- `fractal_get_b()`: retourne la partie imaginaire des coordonnées de la fractale ($c = a + bi$)

Nous vous fournissons deux fonctions auxiliaires déjà implémentées afin de vous faciliter la tâche:

- `fractal_compute_value()`: calcule la valeur correspondant à un pixel de l'image, en appliquant la relation de récurrence
- `write_bitmap_sdl()`: transforme une `struct fractal` en un fichier BMP

L'ensemble des fonctions ci-dessus forment la librairie `libfractal` et les détails des paramètres sont indiqués dans le fichier `fractal.h`. Cette librairie doit pouvoir se compiler de manière indépendante, et votre programme principal doit venir se greffer dessus.

2.2 Format d'entrée

Comme indiqué dans la section précédente, votre programme doit lire une description de fractales sur l'entrée standard, et/ou dans des fichiers. Le format décrit une fractale par ligne, et chaque ligne contient un nom, une taille d'image ainsi que les coordonnées de la fractale:

```
fract1 800 800 -0.8 0.4

fract2 1920 1080 1.0 -1.0
fract3 2 2 0 0
# ceci est un commentaire
test 654 987 -0.56123 0.12345678
```

La première colonne représente le nom de la fractale dont la longueur ne dépasse pas 64 caractères. La deuxième colonne représente la largeur de l'image à générer et est encodée sous la forme d'un entier non signé de 32 bits. La troisième colonne représente la hauteur de l'image à générer et est encodée sous la forme d'un entier non signé de 32 bits. La quatrième colonne représente la partie réelle des coordonnées c de la fractale et est encodée sous la forme d'un double. La cinquième et dernière colonne représente la partie imaginaire des coordonnées c de la fractale et est encodée sous la forme d'un double. Les lignes vides ainsi que les lignes commençant par le caractère # doivent être ignorées. Il ne peut pas y avoir plusieurs fractales ayant le même nom. Si c'est le cas, les duplicata doivent être ignorés et un message d'erreur doit être affiché sur `stderr`.

2.3 Suite de tests

Vous devez également fournir avec vos sources une suite de tests qui vous permet de vérifier le bon fonctionnement des composants de votre programme. Ces tests **doivent** être implémentés avec CUnit.

2.4 Architecture des sources

Le répertoire contenant vos sources doit être organisé de la manière suivante, et contenir au **minimum** les fichiers suivants (excepté pour le répertoire `tests`):

```
-- fractal_GROUPNum_NOM1_NOM2
|-- gitlog.stat
|-- libfractal
|   |-- fractal.c
|   |-- fractal.h
|   |-- Makefile
|   |-- tools.c
|-- main.c
|-- Makefile
|-- rapport.pdf
|-- tests
|   |-- test1.c
|   |-- test2.c
```

Le répertoire `fractal_GROUPNum_NOM1_NOM2` est le répertoire racine de votre code, en remplaçant `GROUPNum` par le numéro de votre groupe et `NOM1` et `NOM2` par les deux noms de votre groupe (ou un seul si vous ne travaillez pas en binôme). La description des fichiers est la suivante:

- `Makefile`: principal `Makefile` du projet. L'exécution de la commande `make` sans arguments doit générer un exécutable `main` permettant de lancer votre programme. L'exécution de la commande `make lib` doit pouvoir compiler la `libfractal`, et uniquement elle. L'exécution de la commande `make clean` doit pouvoir supprimer de vos sources les exécutables et les fichiers objets qui ont été générés. L'exécution de la commande `make tests` doit compiler et exécuter vos tests.
- `main.c`: Ce fichier doit contenir la fonction `main()` de votre programme.
- `libfractal`: Ce répertoire contient la librairie `libfractal`. Une commande `make` exécutée **dans** ce répertoire doit pouvoir compiler la librairie.
- `libfractal/fractal.c`: Ce fichier contient l'implémentation des fonctions de la librairie.
- `libfractal/fractal.h`: Ce fichier contient le prototype des fonctions de la librairie ainsi que la description de la structure `struct fractal`.
- `libfractal/Makefile`: `Makefile` correspondant à la librairie.
- `tests`: Ce répertoire contient vos tests. Son contenu est libre.
- `rapport.pdf`: Un rapport de maximum 2 pages. Il décrira et justifiera succinctement les choix d'implémentation **importants** réalisés si nécessaire (avez-vous fait un choix que vous désirez mettre en avant? Avez-vous été plus loin/moins loin que ce qui a été demandé? Avez-vous réalisé des tests, avec quels résultats?). De plus, il contiendra une **évaluation** qualitative et/ou quantitative de votre projet.
- `gitlog.stat`: Ce fichier est généré par la commande `git log --stat > gitlog.stat`. Le projet **doit** donc être versionné avec `git`.

2.5 Utilisation du programme

Votre programme doit supporter l'utilisation suivante: `./main [-d] [--maxthreads n] fichier1 fichier2 fichierN fichierOut`.

Chaque `fichierX` correspond à un fichier d'entrée à partir duquel il faut lire la description des fractales. Le fichier `fichierOut`, qui est **toujours** le dernier argument, est le nom du fichier de sortie **final**. Si l'option `-d` est présente, alors votre programme doit générer les fichiers BMP pour **chaque** fractale calculée. Le nom du fichier doit être le nom de la fractale suivie de l'extension `.bmp`. Ces fichiers doivent être générés dans le répertoire courant.

Lorsqu'un fichier d'entrée vaut `-`, cela signifie qu'il faut également lire les données sur l'entrée standard. Exemple: `./main fichier1 fichier2 - fichierOut`. Un seul `-` peut être spécifié.

Notez que votre programme peut être appelé depuis n'importe où: l'invocation peut donc se faire selon un chemin absolu, par exemple `/home/user/projet/fractal-GROUPENUM.NOM1.NOM2/main fichier1 ... fichierOut` ou encore selon un chemin relatif différent, par exemple `.././main -d fichier1 ... fichierOut`

L'argument `--maxthreads` indique le nombre maximum de threads de **calcul** que votre programme peut utiliser. La valeur minimum vaut 1.

2.6 Librairie externe

La fonction `write_bitmap_sdl()`, permettant de transformer une `struct fractal` en un fichier BMP, utilise la librairie SDL. Pour ce faire, vous devez linker votre programme final avec cette librairie en utilisant le paramètre `-lSDL`. Sous Ubuntu/Debian, les packages à installer sont `libsdl1.2debian` et `libsdl1.2-dev`.

3 Délivrables

Le projet se déroule en trois phases: architecture, implémentation et reviews.

3.1 Architecture

Durant cette phase, vous devez spécifier à haut niveau l'architecture de votre programme. Il faut au moins définir:

- Structures de données (représentation des fractales,...)
- Types de threads entrant en jeu (quelle(s) tâche(s) ils réalisent)
- Méthode(s) de communication entre les threads
- Informations communiquées entre les threads

Afin de valider votre architecture, vous réaliserez une interview de 10 minutes avec les assistants. Cela vous permettra de ne pas faire fausse route. Les interviews se feront les **jeudi 29 mars** et **vendredi 30 mars**. Lors de votre interview, vous devez amener un **mini-rapport de une à deux pages** expliquant les quatre points d'architecture décrits ci-dessus.

3.2 Implémentation

Dès que votre architecture sera validée, vous pourrez commencer à l'implémenter. La remise de l'implémentation complète est fixée à la S12. Les détails pratiques ainsi que le template de code vous seront communiqués en temps utile.

3.3 Reviews

Immédiatement après la remise de l'implémentation, vous recevrez chacun (individuellement) 5 projets d'autres groupes. Parmi ces projets, vous devrez en sélectionner deux qu'il vous faudra reviewer. La deadline des reviews est fixée à la S13. Les détails suivront.

4 Dates importantes

- 13/03: Lancement de la phase d'architecture
- S7: Inscription des groupes sur Moodle
- S8: Interviews et lancement de la phase de programmation
- S9, S10, S11, S12: Permanences les mercredis 12h45-14h en Intel

- Vendredi 11 mai à 18h: Remise du projet sur Inginious
- S13: Remise des reviews