SINF1252 - Fractales

2016

1 Introduction

Les fractales sont des objets mathématiques fascinants: elles exhibent une forme que l'on retrouve à toute 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 Julia [1].

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 défini 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.

2 Énoncé

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

[1] `https://fr.wikipedia.org/wiki/Ensemble_de_Julia`

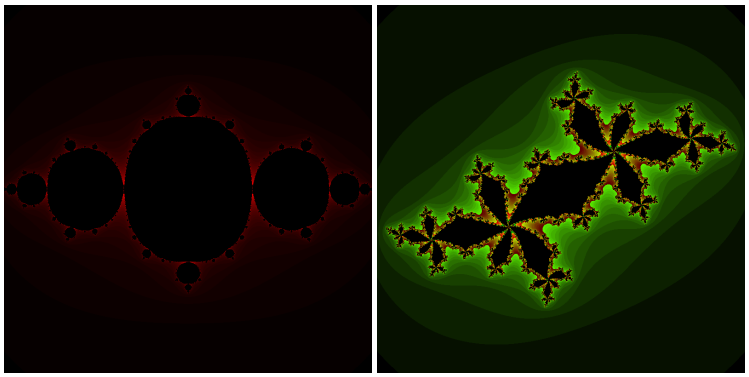

Figure 1: Exemple de fractales. La première est générée pour $c = -0.8 + 0.0i$ et la deuxième pour $c = -0.52 + 0.57i$. La fractale de droite contient les couleurs générées "naturellement" par la fonction de transformation. Un filtre rouge a été appliqué sur la fractale de gauche.

- 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_value()`: retourne la valeur correspondant à un pixel de l'image
- `fractal_set_value()`: défini 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 encodé sous la forme d'un entier non signé de 32 bits. La troisième colonne représente la hauteur de l'image à générer et est encodé sous la forme d'un entier non signé de 32 bits. La quatrième colonne représente la partie réelle des coordonnéees c de la fractale et et encodé sous la forme d'un `double`. La cinquième et dernière colonne représente la partie imaginaire des coordonnées c de la fractale et est encodé 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 `test`):

```
-- fractal_NOM1_NOM2
   |-- libfractal
   |   |-- fractal.c
   |   |-- fractal.h
   |   |-- Makefile
   |   |-- tools.c
```

```
|-- main.c
|-- Makefile
|-- test
    |-- test1.c
    |-- test2.c
```

Le répertoire `fractal_NOM1_NOM2` est le répertoire racine de votre code, en remplaçant `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 test` 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.
- `test`: Ce répertoire contient vos tests. Son contenu est libre.

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_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`.

2.7 Phases du projet

Le projet se déroulera en deux phases principales. La première phase est la phase d'**architecture**, où vous devrez réfléchir à comment vous aller concevoir votre programme afin qu'il puisse réaliser son objectif de la manière la plus efficace qu'il soit. Vous ne devez rien programmer durant cette phase. Celle-ci se terminera par une interview où vous nous expliquerez en dix à quinze minutes l'architecture que vous avez conçu.

La deuxième phase consistera à la programmation même du projet.

- 16/03: Lancement de la phase d'architecture
- S8: Interviews et lancement de la phase de programmation
- S11: Permanences
- S12: Remise du projet