Travail pratique de programmation système avancée

Les ensembles de Julia

1 Objectif

- Rappel du Rust
- Afficher de jolies images de fractales
- Petit exercice de programmation concurrente en Rust
- Utilisation de `Arc` et `Mutex`

2 Ensembles de Julia

Les ensembles de Julia (du mathématicien français Gaston Julia, 1893-1978) sont des objets fractals du plan complexe (plus sur les complexes à la fin du document). Dans ce contexte, le mot fractal signifie que ces objets ont une dimension fractionnaire et présentent des invariances d'échelle. Pour définir les ensembles de Julia, on considère la famille de fonctions complexes

$$P_c : \mathbb{C} \to \mathbb{C}, \tag{1}$$

$$z \to z^2 + c, \tag{2}$$

où $c \in \mathbb{C}$.

On dénote K_c, l'ensemble des points $z \in \mathbb{C}$, tels que la suites des itérés $P_c^n(z)$ est bornée:

$$K_c = \{z \in \mathbb{C} \mid \sup |P_c^n(z)| < \infty\}, \tag{3}$$

où $P_c^n(z)$ est défini par

$$P_c^n(z) = P_c(P_c(...P_c(z))). \tag{4}$$

L'ensemble de Julia J_c est le bord de K_c. En d'autres termes, s'il existe $M \in \mathbb{R}$ tel que $|P_c^n(z)| \leq M$. Dans ce TP, nous considérons $M = 2$.

Les ensembles de Julia ont quelques propriétés importantes

1. J_c est inclu dans le cercle de centre 0 et de rayon $|c| + 1$.
2. J_c est symétrique par rapport à 0.
3. $J_{\bar{c}}$ est le symétrique de J_c par rapport à l'axe des réels. En particulier, si c est réel, J_c est symétrique par rapport à l'axe des réels.

Comme J_c est inclus dans le disque de rayon $|c| + 1$ centré en 0, on peut affirmer que $z \notin J_c$ si et seulement si la suite des itérés $P_c^n(z)$ sort de ce disque à une itération donnée.

3 Calcul d'un ensemble de Julia

Pour calculer l'ensemble de Julia, J_c, associé à une valeur, $c \in \mathbb{C}$, on peut se restreindre au domaine carré $[-2, 2] \times [-2i, 2i] \in \mathbb{C}$. Ce domaine est discrétisé en une grille dont la finesse du maillage déterminera la précision du calcul. La grille peut être stockée sous forme d'un tableau d'entiers. L'élément du tableau correspondant à un z donné, contient l'itération, k, à partir de laquelle $P_c^k(z)$ sort du disque de rayon 2. C'est cette valeur, k, que nous souhaitons calculer et afficher dans ce travail.

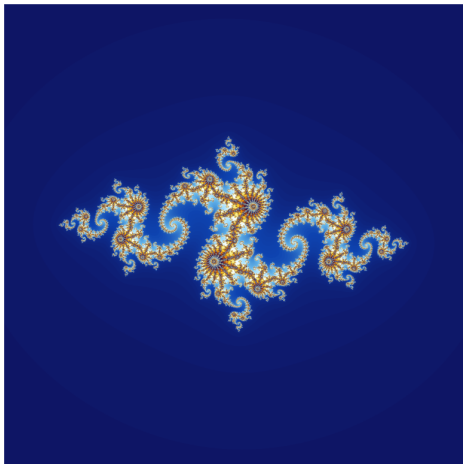

Figure 1: Exemple d'un ensemble de Julia pour $c = -0.8 + 0.156i$. Source wikipedia

4 Travail à réaliser

Le calcul de l'ensemble K_c associé à $c \in \mathbb{C}$, se fait sur le carré du plan complexe, dont le coin inférieur gauche est $-2 - 2i$ et supérieur droit $+2 + 2i$, carré qu'on discrétise en une grille. Pour chacune des valeurs complexes de la grille, appelons les z_0, on détermine si la suite des itérés

$$z_{n+1} = P_c(z_n) = z_n^2 + c, \tag{5}$$

est bornée ou non. Une bonne heuristique est obtenue en effectuant un certain nombre d'itérations, `nb_iter`, pour voir si la suite sort du disque de rayon 2. Ceci correspond à la condition

$$|z_n| \leq 2. \tag{6}$$

Évidemment, `nb_iter` représente une borne maximale, car on peut interrompre l'itération dès qu'on a quitté le disque de rayon 2. En d'autre termes, on enregistre le n, tel que $z_n > 2$. Si $n >$ nb_iter on interrompt la boucle et on enregistre ou $n =$ nb_iter.

Pour visualiser les résultats vous pouvez utiliser la crate `image` par exemple. Vous pouvez aller sur la page pour trouver des exemples de valeurs de c et les résultats attendus.

4.1 Parallélisme

Une fois le calcul de l'ensemble de Julia étant réalisé de façon séquentuielle, il vous reste la partie importante de ce travail pratique: paralléliser le code en utilisant des threads système. Vous pouvez essayer différentes stratégies de parallélisation. Par exemple, vous pouvez `spawn` un thread par pixel ou un thread par ligne ou colonne. Une autre façon de paralléliser votre travail est via la crate `rayon` que vous pouvez également tester. N'oubliez pas d'utiliser les `Arc` et les `Mutex` pour les accès concurrents à la matrice de pixels que vous allez créer.

Pour chaque exécution mesurez le temps de calcul de l'ensemble de Julia.

Pour avoir un gain visible pensez à faire des images assez grandes (1000 x 1000 p.ex.).

4.2 Nombres complexes

Comme vous l'aurez compris pour réaliser ce travail il faut réaliser une librairie de manipulation de nombres complexes. Chaque nombre complexe est constitué une paire de nombre réels: la partie **réelle** et la partie **imaginaire** du nombre. Le type complexe sera donc une structure

```
struct Complex {
    re: f64,
    im: f64,
}
```

Ensuite il faut implémenter les règles d'addition, de multiplication entre deux nombres complexes, ainsi que le module d'un nombre complexe.

Soit $z \in \mathbb{C}$ un nombre complexe. On note ce nombre

$$z = a + i \cdot b, \tag{7}$$

où a est la partie **réelle** de z, b la partie **imaginaire** de z, et i le nombre imaginaire qui a la super propriété $i^2 = -1$. Avec cette notation, on peut calculer presque comme avec des nombres réels. Ainsi l'addition de deux nombres complexes, $z_1 = a_1 + i \cdot b_1$, $z_2 = a_2 + i \cdot b_2$ s'écrit

$$z_1 + z_2 = (a_1 + a_2) + i \cdot (b_1 + b_2). \tag{8}$$

Le produit est un peu plus complexe mais s'écrit

$$z_1 \cdot z_2 = (a_1 \cdot a_2 - b_1 \cdot b_2) + i \cdot (a_1 b_2 + a_2 b_1). \tag{9}$$

Finalement le module de $z = a + i \cdot b$ s'écrit

$$|z| = \sqrt{a^2 + b^2}. \tag{10}$$

Comme vous pouvez le voir dans la définition de la structure des nombres complexes, à aucun moment nous n'avons besoin de représenter i dans notre structure de données. i est nécessaire pour les humains pour qu'ils puissent calculer plus facilement avec les nombres complexes.

Avec ce qui précède vous pouvez implémenter les trait `Add` et `Mul` pour le type `Complex`, ainsi que la fonction asssociée `module()` et avez en votre possession tout ce qu'il faut pour écrire le code qui vous permettra de faire des figures plus incroyables les unes que les autres.