PINSON Daniel
MOURELON Yann

Projet 2022

Système Distribué

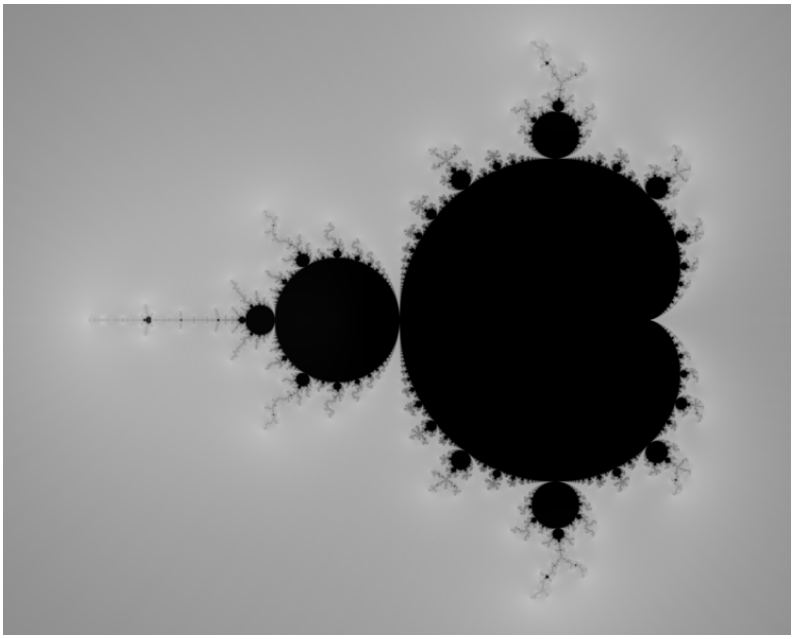
Master 1

Sujet 11 : Ensemble de Mandelbrot - 2 étudiants

Introduction :

Le **système distribué** est une architecture informatique visant à utiliser des ressources informatiques sur plusieurs nœuds de calcul distincts pour atteindre un objectif commun partagé. On va utiliser plusieurs périphériques différents (souvent des périphériques matériels physiques) pour exécuter une tâche lourde tout en partageant des ressources afin d'accélérer le temps de calcul.

Pour ce projet, nous allons travailler sur un objet bien connu du monde des mathématiques : les fractales. Ou plus spécifiquement l'ensemble de Mandelbrot qui représente une fractale définie sur l'ensemble des points $c \in \mathbb{C}$ du plan comme vu sur la figure ci-dessous.



Au vu de la nature mathématique du projet, il nous faut un cluster de calcul et nous avons choisi d'utiliser le langage **java** et plus particulièrement la librairie **java rmi.**

Le programme va donc lancer une fenêtre dans laquelle on va dessiner l'ensemble de Mandelbrot. Les points en noir sont les points qui appartiennent à l'ensemble de Mandelbrot et la couleur des autres points représente la vitesse de divergence de ces points. Le programme a une fonction de zoom, pour zoomer sur l'image, il suffit de faire un clic gauche sur l'image, rester enfoncé puis relâcher sur un autre point de l'image. Le rectangle délimité par ces deux points sera la nouvelle fenêtre.

Partie 1 : Analyse du sujet

Ensembles de Julia et l'ensemble de Mandelbrot

Avant de parler de l'ensemble de Mandelbrot, il faut comprendre ce que sont les **ensembles de Julia** et pour cela il faut comprendre comment ils ont été découverts.

Tout d'abord, il faut introduire la notion de système dynamique. Soit un système comme étant un ensemble d'objet, celui-ci est dynamique lorsqu'il évolue en fonction du

temps selon une certaine règle. Par la suite, on utilisera “système” pour parler de système dynamique.
On observe que dans un système, même lorsqu’il contient peu d’objets et qu’ils suivent une règle simple (telle une fonction par exemple), il y a toujours la possibilité que les objets évoluent de façon irrégulière/imprévisible. C’est ce qu’on appelle le chaos[1].
Il existe un sous-domaine à l’étude des systèmes dynamiques qui est la dynamique holomorphe. Il s’agit d’étudier les systèmes ayant pour fonction d’itération (ou règle d’évolution), une fonction à valeures complexes qui est définie et dérivable en tout point du plan de $\mathbb{C}$.

On note alors deux comportement différents selon les points de départ à partir duquel on itère:

- L’un des comportements a la caractéristique suivante, un faible changement du point de départ entraîne un faible changement sur la suite d’itération. On parle de stabilité.
- L’autre fait qu’un faible changement du point de départ entraîne un changement radical par la suite. On parle de chaos.

On définit le *bassin d’attraction de l’infini* comme étant l’ensemble des points qui sont stable par itération après perturbation et *l’ensemble de Julia rempli,* l’ensemble des points qui changent radicalement par itération[2]. Et l’ensemble de Julia est défini comme étant la frontière entre l’ensemble de Julia remplie et le bassin d’attraction de l’infini[3].
Ainsi chaque système dynamique suivant les critères précédents admettent 2 ensembles représentant l’évolution de chaque point.

Il existe autant d’ensemble de Julia qu’il n’y a de point sur le plan $\mathbb{C}$. La définition mathématique d’un ensemble de Julia est définie de la façon suivante : Ainsi J_c est un ensemble de Julia tel que les points du plan $z_0 \in \mathbb{C}$ pour lesquelles la suite ci-après est bornée.

$$(1) \begin{cases} z_0 = x + iy \\ z_{n+1} = z_n^2 + c \end{cases}$$

avec $i^2 = -1$ et $n \in \mathbb{N}$
L’ensemble de Mandelbrot est similaire aux ensembles de Julia J_c. La définition est la même à ceci-près que $z_0 = 0$ et que les points du plan sont les $c = x + iy$.
L’ensemble de Mandelbrot représente l’ensemble des $c \in \mathbb{C}$pour lesquels J_c est connexe. Ainsi il n’existe qu’un seul ensemble de Mandelbrot.
L’ensemble de Mandelbrot est défini comme l’ensemble des points $c \in \mathbb{C}$ pour lequel la suite $(z_n)_n$ suivante est bornée

$$(2) \begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

---

[1] Chéritat, Arnaud. 2010. “Images des mathématiques.” Images des mathématiques. https://images.math.cnrs.fr/L-ensemble-de-Mandelbrot.html
[2] Dynamique holomorphe — Wikipédia https://fr.wikipedia.org/wiki/Dynamique_holomorphe

[3] Ensemble de Julia — Wikipédia https://fr.wikipedia.org/wiki/Ensemble_de_Julia

Pour savoir si la suite n'est pas bornée, il faut trouver z_n tel que $|z_n| > 2$ [4]
D'après cette définition, on peut introduire un nouveau terme, la divergence.

Divergence: Le premier n de la suite $(z_n)_n$ tel que $|z_n| > 2$.

Structures de données et algorithmes

Au vu du fait que l'on doit générer une image, les données entrées sont des points. On utilise donc une liste qui contient tous les points de l'image. La taille de cette liste dépend donc de la résolution de cette image, plus la résolution est grande, plus la liste des points à traiter est grande.
L'idée est donc de traiter chaque point en le convertissant en complexe selon la délimitation de plan complexe.
Ensuite il faut calculer les termes de la suite $(z_n)_n$ et voir à chaque itération si le module est supérieur à 2. Si il l'est, il faut noter la divergence et colorer le point en conséquence.
Il n'y a plus qu'à afficher les points sur l'image pour voir la fractale colorée.

Étant donné que nos données en entrées sont des points avec des coordonnées et que l'on représente le plan complexe, on a d'abord pensé à calculer les points de l'ensemble de Mandelbrot avec les pixel de l'image avec Point(x,y) comme étant Z(x,y). Les Points représentent les pixels des fenêtres. Ce qui signifie que l'on calculerait si le Point(200,200) appartient à l'ensemble de Mandelbrot. Or plus les points avec un réel ou un imaginaire sont élevés, moins ils ont de chance d'appartenir à l'ensemble car le module va rapidement être supérieur à 2. Puisque le module de z_n doit être inférieur ou égal à 2, on peut en déduire que les points de l'ensemble de Mandelbrot sont compris entre -2;2 sur l'axe des Réels et des Imaginaires.

Il a donc fallu partir d'un cadre initial. Nous avons choisi de commencer le programme avec une image contenant les réels entre -2 et 1, et les imaginaires entre -1 et 1 comme représenté sur l'image ci-dessous tirée de wikipédia.

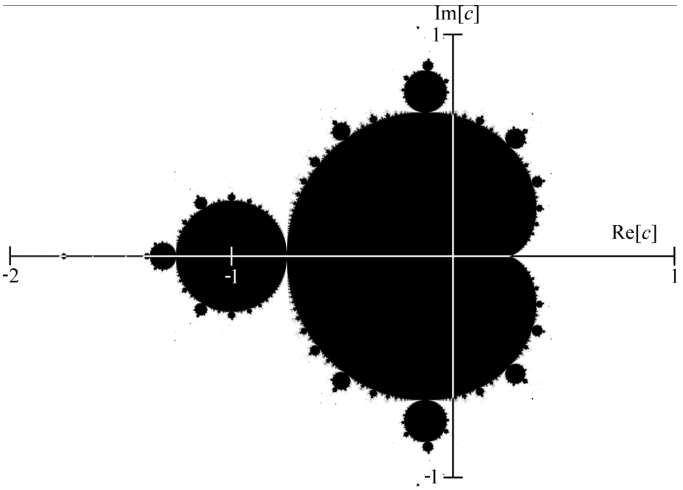


Figure 1

Il a donc fallu faire une mise à l'échelle pour convertir le Point de coordonnées (400,200) en (0,0) sur le plan complexe par exemple. Pour cela on utilise cette formule.

[4] Ensemble de Mandelbrot — Wikipédia

$$(3)(minReel + \frac{x_{Pointcomplex}}{width_{fenetre}}; minIm + \frac{y_{Pointcomplex}}{height_{fenetre}})$$

Les variables x_{Point} et y_{Point} sont les coordonnées du pixel (par exemple (300;200)
Les variables $width_{fenetre}$ et $height_{fenetre}$ sont la résolution de la fenêtre (600x400) par exemple)
Les variables $minReel$ et $minIml$ sont les valeurs réelles et imaginaires minimales de notre plan complexe. -2 et -1 sur la figure 1
Les variable $width_{complex}$ et $width_{complex}$ représentent la longueur de l'intervalle en x et y sur le plan complexe. Donc 1-(-2)=3 et 1-(-1)=2 sur la figure 1

Les coordonnés du point complexe du Point (400;200) sont:

$$(-2 + \frac{400 \times 3}{600}; -1 + \frac{200 \times 2}{400}) = (0; 0)$$

Le serveur contient une fonction qui permet d'écrire un fichier csv comptant le nombre de points ayant une certaine divergence. Cette fonction peut servir pour les calculs de complexité.

Paradigme

Comme précisé en introduction, le paradigme retenu est le RMI. Le modèle Map-Reduce est utile lorsqu'il faut contenir des volumes de données très importants, ce qui n'est pas notre cas d'utilisation. Il est possible d'utiliser un algorithme d'élection car tous les nœuds vont effectuer le même algorithme mais il n'y a pas de décision à prendre ce qui rend l'élection du leader peu importante.
Nous allons donc utiliser la même architecture que le *bag of task* car il est nécessaire de donner un paramètre afin d'effectuer la tâche et nous avons déjà eu l'occasion de manipuler ce type de technologie. Pour la partie graphique, nous allons utiliser Java Swing.

Partie 2 : Programme

Les classes avec leurs attributs et méthodes

Point: Contient deux *Doubles* pour représenter les coordonnées et une couleur de type Couleur. La classe implémente *Serializable* car appelée dans le constructeur de la *Task*
Complexe: Contient deux *Doubles* qui sont les parties réelles et imaginaires.
Méthodes permettant les additions, multiplications et calcul du module.
Fenêtre: Contient tout ce qui est nécessaire pour faire la fenêtre.
Ajout d'un MouseListener dans le constructeur pour détecter les clics et demande au serveur de ré-afficher la fenêtre si on relâche un clic.
Possède une méthode *Complexe convert(Point)* qui permet de convertir un *Point* en complexe.

Panneau: Contient une liste de Point de tous les pixels de l'image.

void paintComponent() qui permet d'afficher tous les points dans sa liste avec leurs couleurs

ImpTask: A pour attributs un Point, la divergence de ce point ainsi qu'un booléen qui vaut *true* si ce point appartient à l'ensemble de Mandelbrot.

void run() qui exécute une **tâche**, change le booléen et la divergence selon le point.

void convert(*Point*) qui permet de convertir un *Point* en complexe selon l'algorithme décrit précédemment (3).

ImpMandelbrot: Contient 2 listes, une contenant tous les *Point* à traiter et l'autre contenant les *Task* complétées. À aussi un entier qui permet de dire le nombre de *Task* déjà effectuées. Cette classe représente le Bag Of Task.

void addTask(Point) qui permet d'ajouter une tâche.

Task getTask() qui incrémente le nombre de tâches faites et donne la prochaine *Task* à faire.

void addResult(Task) qui ajoute la *Task* finie dans la liste des *Task complétés*.

Task et *Mandelbrot*: qui sont les interfaces contenant les prototypes des classes *ImpMandelbrot* et *ImpTask*.

Server: Le serveur qui remplit la Bag Of Task et dessine l'image une fois toutes les tâches terminées.

void draw() qui bloque le serveur et permet de dessiner dans la fenêtre.

Client: Le client qui va récupérer le Bag Of Task et les *Task.* Il va les exécuter après les avoir récupéré et va remettre les résultats dans le bag.

Constantes: Contient des attributs servant de variables globales tel que la résolution de la fenêtre.

void calculCoordPlan() permettant de calculer les attributs tels que les intervalles sur les axes des réels et des imaginaires, ainsi que la valeur minimale sur chaque axe.

Pour se servir de RMI, on a dû faire en sorte que les classes *ImpMandelbrot* et *ImpTask* héritent de la classe *UnicastRemoteObjet* et les interfaces *Mandelbrot* et *Task* doivent hériter de *Remote*.

Fonctionnement

Ainsi le fonctionnement de l'algorithme est le suivant:

- Lancement du serveur. Celui-ci va créer la fenêtre, dans laquelle on va afficher Mandelbrot. Il initialise le Bag of Task, le rempli de toutes les données à traiter, donc de tous les points de la fenêtre, et va le mettre dans l'annuaire.
  Il appelle ensuite la méthode *draw()* qui va bloquer le serveur jusqu'à ce que toutes les tâches soient effectuées (il vérifie toutes les secondes si c'est le cas), pour pouvoir remplir la liste des points à dessiner dans Panneau. Il lui ordonne ensuite de dessiner.
  Cette méthode peut être appelée de nouveau si l'utilisateur fait un zoom.
- Lancement des clients. Chacun va essayer de récupérer le Bag Of Task dans l'annuaire. Une fois fait, il entre dans une boucle infinie.
  Dans cette boucle, il récupère une tâche et l'exécute puis ajoute le résultat dans le sac. Si il n'y a pas de tâches à récupérer, il s'endort et vérifie toutes les secondes s'il n'y a pas de nouvelles tâches à faire.

- Pour faire une tâche, on regarde pour chaque terme de la suite z_n si le module est supérieur à 2. On fixe une limite arbitraire et si z_{limite} à un module inférieur ou égal à 2, on considère que cette suite est bornée et le point appartient à l'ensemble de Mandelbrot. Dans le cas où le module est supérieur à 2, on note le n comme étant la divergence et on colore le point selon la divergence.

Architecture logiciel

Notre architecture est très basique. Elle comporte 1 serveur, 1 registre et 1 ou plusieurs clients.

Le serveur va générer les données à traiter puis il va les envoyer au registre qui sera accessible par le serveur et les clients puis va se mettre en attente jusqu'à obtenir tous les résultats. Les clients vont récupérer les données qui sont dans le registre, les traiter, les modifier et recommencer jusqu'à ce qu'il n'y ait plus de données. Il reste cependant à l'écoute de futures données à traiter. Le serveur récupère ensuite ces résultats pour faire l'affichage graphique et l'affiche. Lors d'un zoom ou dézoom, le processus recommence.

Lancement du programme

Le programme a besoin de plusieurs terminaux pour se lancer.
Soit terminal 1 qui permet de lancer le serveur et terminal 2 qui permet de lancer le client.
Il y a un Makefile dans chaque dossier et faire make dans chacun d'entre eux permet de lancer le programme avec des valeurs par défaut, soit:
Une résolution de 600x400
Une limite pour calculer les termes de la suite de 100.
Des valeurs limite du plan complexe : (-2;1) pour l'axe des réels et (-1;1) pour l'axe des imaginaires.
Le rmiregistry est lancé par un fichier shell et relancé à chaque fois que l'on utilise le Makefile
Il est possible de dessiner Mandelbrot avec des arguments pour pouvoir choisir ce que l'on veut dessiner. Pour cela il faut le préciser au serveur.
Cela ne marche pas de lancer "make arg1 arg2…" dans le terminal du serveur. Il faut lancer le serveur avec "java Server arg1 arg2 arg3…"
Il est possible de lancer le serveur avec 3 ou 7 arguments. Si le serveur est lancé avec 3 arguments, ils correspondent à :

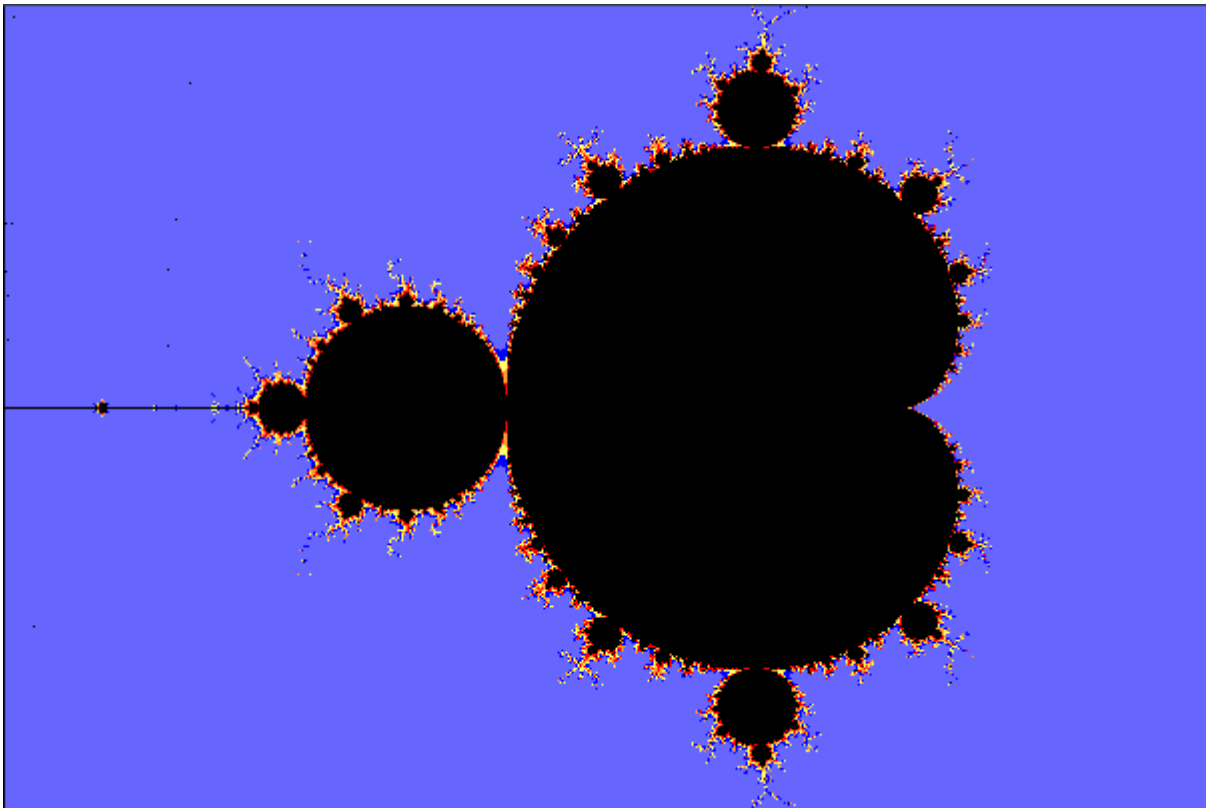
1) La largeur de la fenêtre
2) La hauteur de la fenêtre
3) La limite du terme à partir duquel on déclare que la suite est limitée.

Si le serveur est lancé avec 7 argument, alors ils correspondent au éléments suivants:

4) La valeur minimale possible pour les Réels
5) La valeur maximale possible pour les Réels
6) La valeur minimale possible pour les Imaginaire
7) La valeur maximale possible pour les Imaginaire

Côté client, il est possible de choisir le nombre de clients que l'on lance. Pour cela, les options sont: "make" pour lancer 1 client et "make x" avec x qui vaut soit 2,5 ou 10 pour lancer 2,5 ou 10 clients

Exemple de lancement :



Ensemble de Mandelbrot, 600x400 limite=100 (-2;1) (-1;1)

Il y a 6 différentes couleurs qui sont définies selon leurs divergences. Les couleurs les plus chaudes montrent une divergence forte, tandis que les couleurs froides montrent une divergence faible. Autrement dit, les points de couleurs rouge on comme le premier terme de la suite ayant un module supérieur à 2 avec n>80.
Comme il y a 6 couleurs différentes, elles correspondent aux intervalles de la limite/6.

L'algorithme contient néanmoins un défaut, la limite ne peut jamais être assez élevée. En effet, il faut bien fixer une limite à partir de laquelle on suppose que la suite est bornée, sinon quoi, les points appartenant à l'ensemble de Mandelbrot font une boucle infinie. La contrepartie de cette logique est la suivante avec une limite fixée à 100.
Imaginons un point dont la suite n'est pas bornée car le terme z_{200} à un module supérieur à 2. Alors, ce point appartiendra à l'ensemble de Mandelbrot selon le programme.
Cela a pour effet d'avoir certains point n'appartenant pas à l'ensemble qui sont coloriés en noir, comme s'ils y appartenaient.
D'où les points noir, isolés, observables à certains endroits de l'image.
Pour enlever les aspérités, il faudrait donc une très grande limite (par exemple 10000), mais cela alourdit considérablement les calculs pour les points appartement à l'ensemble de Mandelbrot, comme expliqué dans la partie optimisation.

Partie 3 : Conclusion

Optimisation

Terme initial: Il est possible d'optimiser l'algorithme pour savoir si la suite est bornée. On peut observer l'émergence de différents patterns dans la figure dessinée. Un cardioïde principal ainsi que le bourgeon principal. Ces patterns sont importants car on remarque qu'ils constituent des sous-ensembles fermés dans lesquels tous les points appartenant à ce sous-ensemble appartiennent aussi à l'ensemble de Mandelbrot.

Ces points sont les plus coûteux car il faut calculer tous les termes de la suite $(z_n)_n$ jusqu'à la limite.
Il suffirait donc de savoir si les points appartiennent à ces ensembles pour éviter de faire des calculs inutiles. Or ces deux sous-ensembles sont des figures géométriques, faisant que l'on connaît leurs équations paramétriques.
Cela revient donc à vérifier que c appartient au disque de centre (-1;0) ou à la cardioïde.
Pour savoir si un point appartient au disque de centre (-1;0) revient à vérifier l'égalité suivante pour $c = x + iy$:

$$(x+1)^2 + y^2 < \frac{1}{16}$$

Et pour vérifier que qu'un point appartient à la cardioïde,

$$x < p - 2p^2 + \frac{1}{4} \text{ avec } p = \sqrt{(x - \frac{1}{4})^2 + y^2}$$

Il y a d'autres bourgeons dans lesquels on pourrait lancer les calculs mais qui ne sont pas parfaitement circulaires. Il est néanmoins possible de trouver un cercle dans un de ceci, ce qui permettrait d'adapter la condition initiale et vérifier que ce point appartient au cercle. Nous avons optimisé notre algorithme de cette manière pour alléger les calculs.

Périodicité: Pour optimiser le calcul il est aussi possible de vérifier la périodicité. Par celà on entend vérifier si un terme de la suite que l'on calcule a déjà été calculé. Cela permettrait de directement savoir si la suite est bornée. Cependant, même si cela permet d'améliorer le temps de calcul, cela vient au prix de la mémoire car il faut garder en mémoire toutes les itérations de chaque terme de la suite.

Complexité

Soit 3 variables importante :

- N le nombre de pixel en longueur de l'image
- M le nombre de pixel en hauteur de l'image
- L le terme à partir duquel on admet que la suite est borné si son module est inférieur ou égal à 2, soit le terme limite.

Un calcul est le calcul d'un terme de la suite avec la vérification.
Pour calculer la complexité dans le pire des cas, cela revient à calculer tous les termes de la suite pour tous les points de l'image. On a donc

$$O(NMn) = N \times M \times L$$

Ce qui donne pour une image de résolution 600x400 avec une limite de 100, soit un nombre de calcul de 24 000 000.
Ce cas n'est cependant pas très réaliste car cela revient à dire que tous les points du plan appartiennent à l'ensemble de Mandelbrot. C'est néanmoins possible si l'utilisateur décide de zoomer dans un sous-ensemble fermé.
La périodicité permettrait d'économiser plus de calcul dans le cas général, sauf dans l'éventualité où le zoom de l'utilisateur est dans l'un des 2 sous-ensembles remarquables, auquel cas l'optimisation du terme initial est la plus efficace.

Pour le meilleur des cas, pour tous les points de l'image, le premier terme de la suite a un module strictement supérieur à 2. Ainsi, la limite n'est plus un facteur de la complexité et on a:

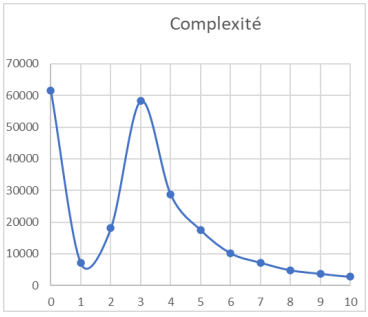
$$o(NM) = N \times M$$

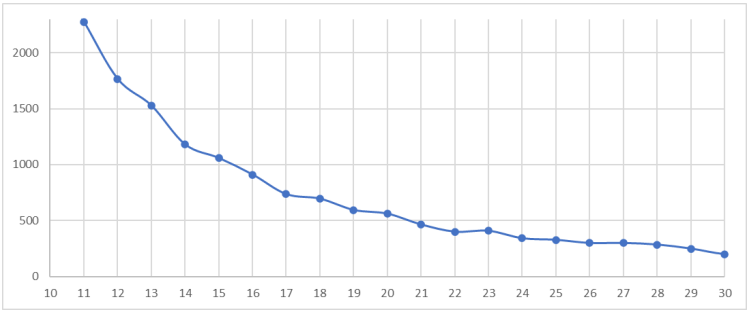
Ce qui donne pour une image de résolution de 600x400, un nombre de calcul de 240 000. Ainsi dans le meilleur des cas l'algorithme est 100 fois plus efficace.
Cependant, ce cas est non seulement irréaliste mais aussi peu intéressant. Un tel cas revient à avoir tous les points du plan n'appartiennent pas à Mandelbrot, soit, l'utilisateur a fait un zoom sur une zone blanche de l'image.
Si l'optimisation du terme initial a été effectué, alors on a aussi cette complexité si l'utilisateur effectue un zoom où tous les points sont inclus dans l'un des 2 sous-ensembles remarquables
L'optimisation de la périodicité ajoute un calcul supplémentaire car on vérifie la périodicité avant de vérifier la valeur du module.
Une complexité plus "réaliste" consiste à dire que la moitié des points appartiennent à l'ensemble de Mandelbrot et l'autre moitié n'en fait pas partie. Cela donne comme complexité:

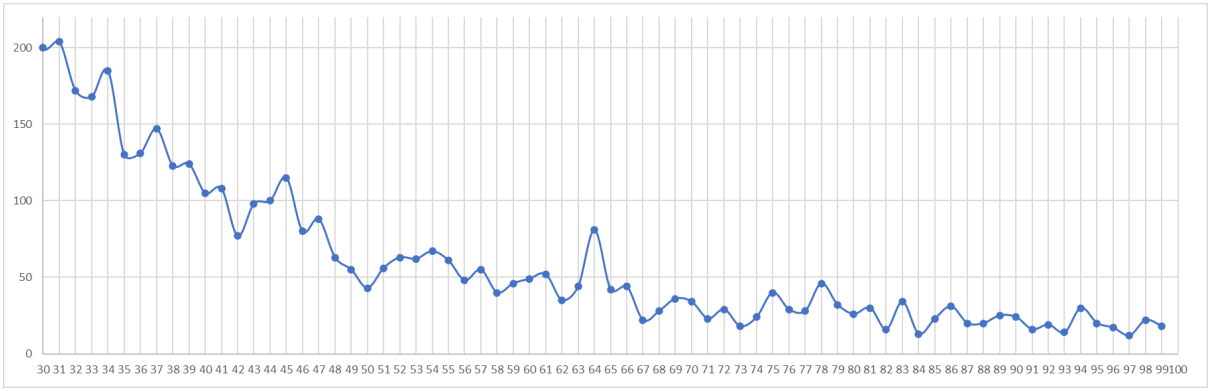
$$\theta(NML) = \frac{NML}{4} + \frac{NM}{4}$$

Ce qui donne pour une image de résolution 600x400 et une limite de 100, un nombre de calcul qui est de 6 060 000.
Cette approche, bien qu'elle est plus réaliste que les autres, a pour défaut de ne pas prendre en compte la divergence. Autrement dit, on part du principe que pour tous les points qui n'appartiennent pas à Mandelbrot, $|z_1| > 2$. Or ce n'est pas le cas et pour certains point, le terme ayant un module supérieur à 2 est $z_{limite-1}$.

Il faut donc prendre en compte la divergence de chaque point. Pour cela on a fait une fonction pour afficher le nombre de points qui ont chaque divergence. Ainsi pour le zoom initial on a fait une fonction qui permet d'écrire dans un fichier en .csv le nombre de points pour chaque divergence. On a ensuite représenté le nombre de points qui ont chaque divergence. En sachant que les points ayant une divergence de 0 sont les points le l'ensemble de Mandelbrot.







On voit donc que la majorité des points ont une divergence faible.
Le nombre total de calcul effectué pour calculer la première image est de 7 236 036, en sachant que l'affichage coûte $N \times M$, on a un coût d'affichage total pour la première image de programme qui est de 7 476 036.
Ce qui est un peu supérieur à la $\theta(NML)$.
A noter que ce chiffre varie de plusieurs dizaines de milliers d'unités. Cela peut venir du problème de concurrence mentionné dans la section Problèmes rencontrés.
Ce calcul a été fait sans optimisations quelconque.

Problèmes rencontrés :

Nous avons commencé à réaliser notre programme sur une architecture simple pour voir si celui-ci marchait bien et pouvoir corriger les problèmes liés au calcul de l'ensemble de Mandelbrot même si nous pensions que ça allait prendre beaucoup de temps. Or nous n'avons pas eu de soucis à le compiler ainsi qu'à le faire fonctionner. Nous nous sommes donc mis à utiliser java rmi pour répartir les calculs de points sur plusieurs clients et un serveur qui s'occupe d'initialiser la fenêtre et l'affichage une fois les points calculés, et, à partir de ce moment-là, nous avons rencontré quelques problèmes.

L'exécution qui devait aller plus vite au vus des multiples clients qui lancent notre programme s'est vu ralentir à cause de notre code qui à la base n'était pas optimisé pour un système distribué. Nous avons donc utilisé la même architecture que pour le *bag of task* vus en TP ce qui nous a permis d'arranger quelques problèmes que nous avions et qui nous en a montré de nouveau.

Le problème principal de notre programme est la gestion de concurrence qui ne fonctionne pas. Nous avons plusieurs clients qui veulent accéder à la liste des données à

traiter, probablement, en même temps. Or il se peut donc que 2 clients (ou plus) récupèrent la même donnée, la traitent en parallèle et renvoient la même valeur dans la liste des tâches exécutées. Or à cause de cela, certains points ne seront pas traités, ou/et traités plusieurs fois.

Nous ne pouvons pas fermer les clients sans les forcer à se fermer dans la console avec un *ctrl+c* ou en fermant la fenêtre de notre serveur.

Nous avons testé notre programme en utilisant un outil de débug ou bien en observant les résultats obtenus nous-même, mais nous n'avons pas fait de test automatisé.

Nous avons pus tester notre programme sur nos machine, en revanche, n'ayant plus accès aux machines de mirande lors de la finition de notre programme, nous n'avons pas pus tester sur plusieurs machines différents (1 serveur et 5 clients sur différentes machines)

Options :

Ensemble de Julia

Les ensembles de Julia nécessitent d'adapter notre programme un peu.
Tout d'abord, il faudrait fixer le point complexe c pour définir le J_c que l'on est en train de calculer. Ensuite il faut faire en sorte que la méthode *run()* de *Task* calcule la même suite mais cette fois-ci le point à traiter est le point z_n tandis que la constance est le point c.

Ensemble de Mandelbulb

Les ensembles de Mandelbulb sont "l'équivalent" de l'ensemble de Mandelbrot dans un espace à 3 dimensions. Cela nécessite donc d'adapter la classe complexe pour qu'elle puisse contenir des nombres "supercomplexe" pour avoir une définition semblable à $w = x + iy + jz$. Le fait de dessiner dans un espace à 3 dimensions pose aussi tous les problèmes de scène avec les lumières, positionnement de caméra

Généralisation

L'ensemble de Mandelbrot pourrait être généralisé. Pour celà, il suffit de changer de puissance dans la suite. Comme l'ensemble de Mandelbrot est né de l'étude des ensembles des paramètres des polynômes quadratiques, il est naturel que la formule de la suite comprend un carré. Cependant, il est possible de généraliser pour former l'ensemble parfois baptisé "multibrot".

Bibliographie

[1] Chéritat, Arnaud. 2010. "Images des mathématiques." Images des mathématiques.

https://images.math.cnrs.fr/L-ensemble-de-Mandelbrot.html

[2] Dynamique holomorphe — Wikipédia

[3] Ensemble de Julia — Wikipédia

[4] Ensemble de Mandelbrot — Wikipédia