TP14 - Algorithmique et programmation
1er mars 2006

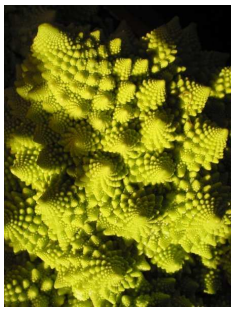
1 La minute culturelle

En 1974, le mathématicien Benoît Mandelbrot, en travaillant sur le hasard et la théorie de l'information, a ouvert un nouveau pan des mathématiques en introduisant des objets à propriétés étranges : les objets fractals. Ces objets géométriques ont au moins l'une de ces propriétés :

- ils ont des détails similaires à des échelles arbitrairement petites ou grandes ;
- ils sont trop irréguliers pour être décrits efficacement en termes géométriques traditionnels ;
- ils sont exactement ou statistiquement autosimilaires c'est-à-dire que le tout est semblable à une de ses parties ;
- leur dimension de Hausdorff est plus grande que leur dimension topologique.

(source : *Wikipedia*)

Si l'inspiration qui a conduit Mandelbrot à étudier ce genre d'objets vient de l'observation de phénomènes naturels (nuages, poumons, arbres, chou Romanesco,...), certains objets fractals sont purement mathématiques (triangle de Sierpinsky, courbe de Peano, flocon de Von Koch,...), et sont décrits par une procédure itérative. Leur construction se prête donc bien à un traitement algorithmique et donc informatique [1].



Le chou Romanesco est un exemple édifiant d'objet fractal naturel. La propriété d'auto-similarité y apparaît de manière spectaculaire [a]. (On peut aussi remarquer que le nombre de spirales orientées dans le sens des aiguilles d'une montre et le nombre de spirales orientées en sens inverse sont deux nombres de la suite de Fibonacci.)

(Image sous license Gnu Free Documentation Licence.)

a. En outre, c'est un légume excellent.

FIGURE 1 – Un exemple d'objet fractal naturel.

1. Vous voyez où je veux en venir ?

2 L'ensemble de Mandelbrot

L'objet fractal qui nous intéresse aujourd'hui est l'ensemble de Mandelbrot. Cet objet est un sous-ensemble du plan complexe, et est construit à l'aide d'une fonction :

$$f : \begin{array}{ccc} \mathbb{C} \times \mathbb{C} & \rightarrow & \mathbb{C} \\ (z, c) & \mapsto & z^2 + c \end{array}$$

L'ensemble de Mandelbrot est caractérisé par l'ensemble des points c du plan complexe tels que la suite :

$$\begin{cases} z_0^c = 0 \\ z_{n+1}^c = f(z_n^c, c) \text{ pour tout } n > 0 \end{cases}$$

converge.

3 Construction de l'ensemble

Vous aurez besoin de manipuler des nombres complexes. Les nombres complexes ne correspondent pas à un type primitif de C. Il vous faudra donc définir une structure correspondant à un nombre complexe :

```
typedef struct structureComplexe {
    double reel;
    double imag;
} complexe;
```

Vous aurez ensuite besoin des fonctions nécessaires à la construction de l'ensemble :

Exercice 1 *Écrire la fonction* `complexe f(complexe z, complexe c)`.

Exercice 2 *Écrire la fonction* `double module(complexe z)`, *dont le rôle est de calculer le module d'un nombre complexe.*

Exercice 3 *Écrire la fonction* `int converge(complexe c)`. *Cette fonction renvoie 1 si la suite z_n^c converge, 0 sinon.*

N.B. : L'ordinateur n'ayant qu'une idée un peu vague de l'infini, on dira qu'une suite complexe converge si son module ne dépasse pas une certaine valeur M au bout de n_0 itérations. Pour l'ensemble de Mandelbrot, les valeurs $M = 1000$ et $n_0 = 255$ devraient suffire.

4 Tracé de l'ensemble

Nous souhaitons tracer dans un fichier image l'ensemble de Mandelbrot. Dans un premier temps, les points appartenant à l'ensemble seront tracés en noir (couleur RVB $(0, 0, 0)$), et les points n'y appartenant pas en blanc (couleur RVB $(255, 255, 255)$). Vous réutiliserez donc une partie du code que vous avez produit lors du TD13 (tracé de drapeaux), pour écrire dans un fichier `.ppm`.

Votre image de taille (w, h) en nombre de pixels représentera une portion du plan complexe située entre z_{min} et z_{max} (les paramètres w, h, z_{min} et z_{max} sont fixes). Vous aurez donc besoin de convertir les coordonnées d'un pixel dans l'image en coordonnées complexes cartésiennes. Comme je suis sympa, je vous donne la fonction de conversion. À un pixel de coordonnées (k, l) dans le fichier image correspond le complexe :

$$z = \left(\Re(z_{min}) + \frac{k}{w \times (\Re(z_{max}) - \Re(z_{min}))}\right) + i\left(\Im(z_{min}) + \frac{h - l}{h \times (\Im(z_{max}) - \Im(z_{min}))}\right) \quad (1)$$

Exercice 4 *Écrivez une fonction* `main` *dont le but est de tracer l'ensemble de Mandelbrot dans un fichier* `mandel.ppm`*. Aidez-vous de la fonction de conversion des coordonnées donnée par l'équation 1. Pour exemple, vous pourrez prendre* $z_{min} = -2 - 1.125i$*,* $z_{max} = 1 + 1.125i$*,* $w = 640$ *et* $h = 480$*.*

Remarques :

- Suivant la machine que vous utilisez, le calcul peut prendre quelques dizaines de secondes. Soyez patients.
- Comme pour les drapeaux, attention aux boucles infinies qui écrivent dans un fichier. Vous allez rapidement saturer l'espace disque, ce qui risque de vous poser quelques problèmes.
- N'oubliez pas de fermer votre fichier image avec l'appel à `fclose`.

5 Pour les plus courageux...

Pour améliorer cette image bicolore déprimante, on peut penser à attribuer une couleur à un pixel, non plus selon son appartenance ou non à l'ensemble de Mandelbrot, mais selon sa vitesse de divergence.

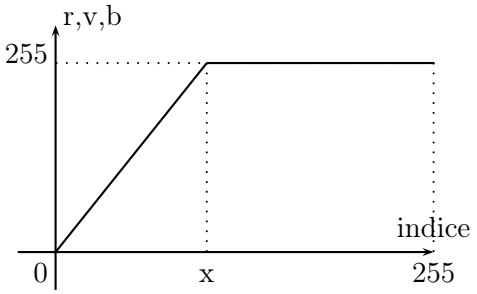
Exercice 5 *Modifier légèrement la fonction* `converge` *pour qu'elle renvoie, non plus 0 ou 1, mais un entier entre 0 et 255 indicant l'entier n au-delà duquel $z(c)_n$ est supérieur à un certain seuil (1000 par exemple).*

La couleur d'un pixel sera ensuite attribué selon le principe des couleurs indexées : à chaque entier entre 0 et 255 (le n de la divergence dans notre cas) correspond une couleur (donc trois composantes : rouge, vert, bleu). Les couleurs seront donc attribuées grâce à une fonction

$$g : \begin{array}{rcl} [0,255] & \to & [0,255]^3 \\ x & \mapsto & (r(x), v(x), b(x)) \end{array}$$

Exercice 6 *Faire une fonction* `int[] g(int indice)` *qui prend en paramètre un entier entre 0 et 255 correspondant à l'indice de la couleur, et qui renvoie un tableau de 3 entiers entre 0 et 255 comportant les composantes de la couleur correspondant au paramètre.*

Remarque : Je vous conseille d'utiliser pour chaque composante (rouge, vert, bleu) une fonction linéaire par morceaux de ce genre :



Exercice 7 *Modifiez la fonction* `main` *de la section précédente pour qu'elle attribue à chaque pixel représentant le complexe z la couleur $g(v_{convergence}(z))$. (La fonction $v_{convergence}$ correspond à la fonction* `converge` *que vous avez modifiée au début de cette section).*