Calcul de la fractale de Mandelbrot sur STM32

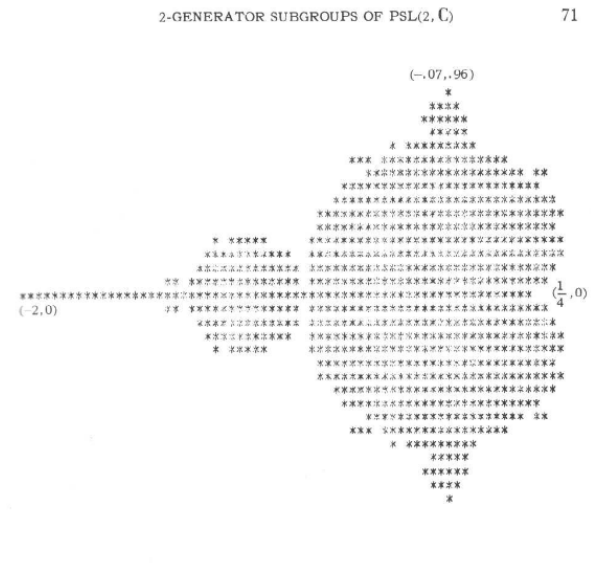
J.-M Friedt, 4 décembre 2020

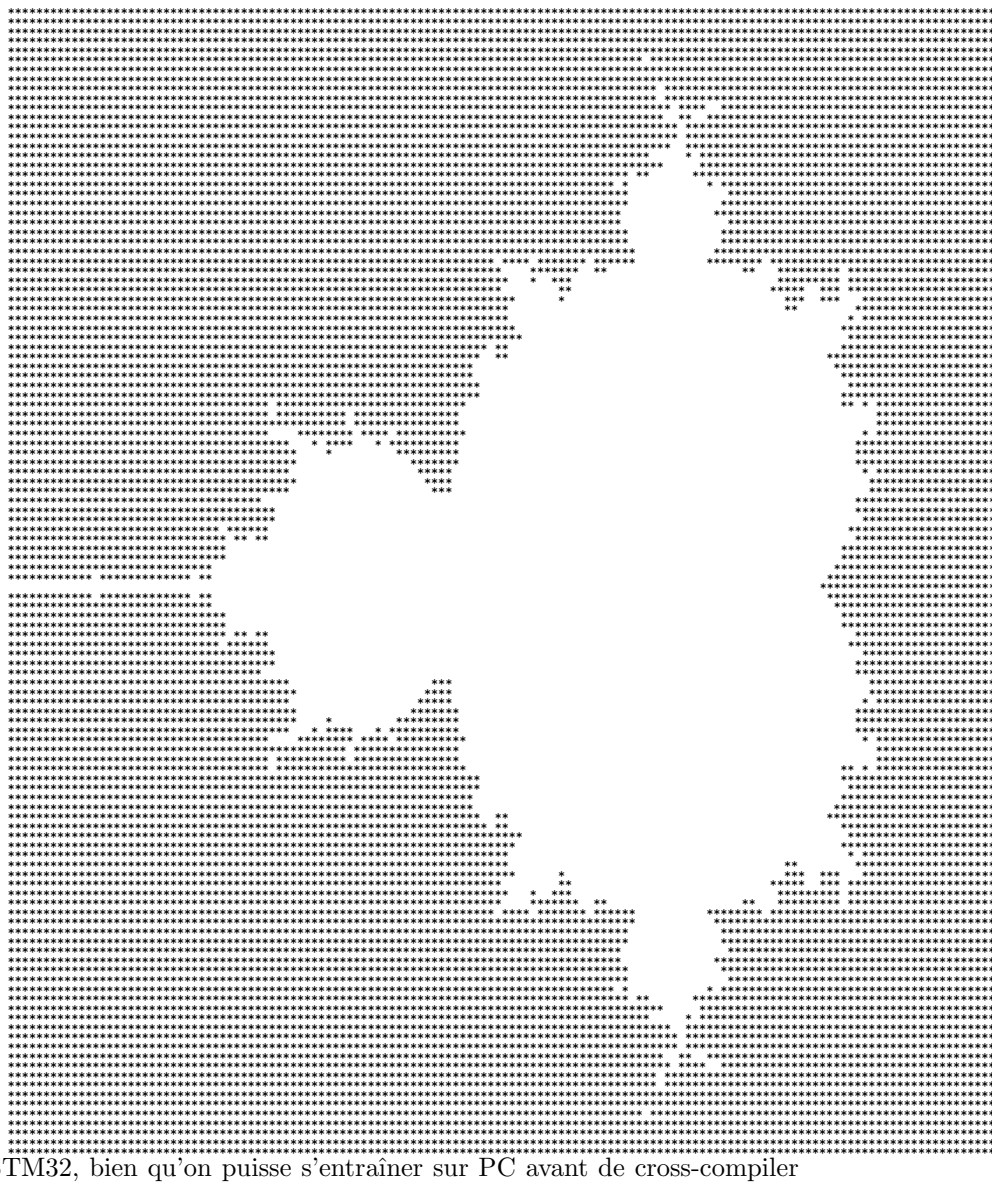
La fractale [1, 2] de Mandelbrot est une structure géométrique dans le plan complexe issue de l'analyse de la suite

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

pour chaque point $c \in \mathbb{C}$ du plan complexe en initialisant $z_1 = c$. Chaque point c est représenté par deux symboles, selon que la suite converge ou diverge. Dans la pratique, la convergence ou la divergence est établie après quelques itérations et seuillage du module de z_n.

Dans leur publication [3] les auteurs tracent l'ensemble de Mandelbrot dans l'intervalle réel $[-2; 0,25]$ et l'intervalle imaginaire $[-0,96; 0,96]$ pour rechercher des orbites périodiques. Ici nous allons nous intéresser à un ensemble plus large avec une condition de divergence en établissant que si au bout de 16 itérations le carré du module de $|z_n|$ n'a pas dépassé 10, alors la suite converge probablement et nous afficherons le premier symbole. Dans le cas contraire, si le module dépasse ce seuil avant 16 itérations, nous afficherons le second symbole. L'objectif est de cartographier les valeurs de c pour lesquelles la suite converge et diverge.





Gauche : graphique publié dans [3] en 1978. Droite : la solution recherchée avec ce programme.

Tous ces calculs se feront sur STM32, bien qu'on puisse s'entraîner sur PC avant de cross-compiler sur la cible pour afficher le résultat par communication RS232 via le port série virtuel sur USB depuis le STM32 vers le PC. En séparant correctement la partie algorithmique du code de la partie communication, le passage du PC au STM32 doit se faire sans problème.

1. le problème a été exprimé sous forme de nombres à virgules (0,96, 0,25 ...) : sachant que nous imposons de ne travailler que sur des entiers, et que nous voudrons tracer une centaine de points tout au plus en abscisse et en ordonnée, comment seront représentées les parties réelle et imaginaire de chaque complexe ? Comment exprimer dans ce contexte l'intervalle $[-0,95; 1,20]$ par pas de 0,02 ?

   Représentation à virgule fixe, par exemple sous forme de 1000 fois la valeur réelle considérée pour ne travailler que sur des entiers, bien que la multiplication par 100 suffise si une centaine de pas est recherchée. Ainsi en multipliant par 1000, l'intervalle s'écrira (notation Matlab) : $[-950 : 20 : 1200]$.

2. Compte tenue de la réponse précédente, proposer une fonction qui prenne en argument les parties réelles et imaginaires de deux complexes représentés de façon adéquate, et renvoie la somme des deux complexes. On pourra investir un peu de temps à créer une structure adéquate pour représenter un complexe qui facilitera la suite de la discussion, et provisoirement tester la fonction mise en œuvre sur PC avant de la transposer vers le microcontrôleur pour s'assurer des résultats obtenus.

   La somme en virgule fixe ne pose pas de problème puisque les retenues se propagent vers la partie entière de la somme et aucune décimale n'est ajoutée. Nous restons donc sur un notation avec 3 décimales après la virgule implicite :

```
1 #include <stdio.h>
2 struct cpl {int re;int im;};
3
4 struct cpl addcomp(struct cpl in1,struct cpl in2)
5 {struct cpl tmp;
6  tmp.re=in1.re+in2.re;
7  tmp.im=in1.im+in2.im;
8  return(tmp);
9 }
```

3. Compte tenue de la première réponse, proposer une fonction qui prenne en argument les parties réelles et imaginaires de deux complexes et renvoie le produit des deux complexes. On prendra soin à conserver la résolution avec laquelle les nombres sont représentés.

   Le produit ajoute des décimale : si nous avons multiplié par 10^N, alors le produit a créé $2 \times N$ décimales et il faut en éliminer N pour revenir sur la notation initiale, donc diviser le résultat par 10^N :

```
1 struct cpl mulcomp(struct cpl in1,struct cpl in2)
2
3 {struct cpl tmp;
4  tmp.re=in1.re*in2.re-in1.im*in2.im;
5  tmp.im=in1.re*in2.im+in1.im*in2.re;
6  tmp.re/=1000;
7  tmp.im/=1000;
8  return(tmp);
9 }
```

4. Finalement, proposer une troisième fonction qui prenne en argument un complexe et renvoie son carré du module dans une représentation cohérente avec les autres opérations.

   Le module consiste à mettre au carré chaque terme du complexe et d'en faire la somme, donc les argumentaires des deux questions précédentes sont valables : la somme n'introduit pas de décimale et le produit induit une division du résultat par 10^N :

```
1 int modcomp(struct cpl in1)
2 {int tmp;
3  tmp=in1.re*in1.re+in1.im*in1.im;
4  return(tmp/1000);
5 }
```

5. Fort de ces trois fonctions, proposer un programme qui boucle sur l'axe réel de -1,2 à +1,2 par pas de 0,02 et selon l'axe imaginaire de -2,0 à 0,85 par pas de 0,02 (120 points en abscisse par 142 points en ordonnée). Pour chaque point, itérer la suite $z_{n+1} = z_n^2 + c$ avec c la coordonnée du point analysé, et arrêter l'itération soit si le module de z_n dépasse 10, soit si le nombre d'itérations atteint 16. On affichera un espace dans le premier car, un symbole "*" dans le second.

   Il ne reste plus qu'à utiliser les fonctions citées ci-dessus dans une fonction principale qui boucle dans le plan complexe selon l'axe imaginaire en boucle extérieure et selon l'axe réel en boucle intérieure, et pour chaque point mémoriser la coordonnée du complexe, itérer au plus 16 fois la suite ou sortir de l'itération si le module a dépassé le seuil, et afficher le caractère résultant à l'écran selon la condition de sortie :

```
1  int main()
2  {struct cpl tmp;
3   struct cpl current;
4   int n,zr,zi;
5   for (zi=-1200;zi<1200;zi+=20)
6     {for (zr=-2000;zr<850;zr+=20)
7       {n=0;
8        current.re=zr;
9        current.im=zi;
10       tmp.re=zr;
11       tmp.im=zi;
12       do {tmp=addcomp(current,mulcomp(tmp,tmp));
13           n++;
14          }
15       while ((modcomp(tmp)<10000)&&(n<16));
16       if (n<16) printf("*"); else printf("␣");
17      }
18     printf("\n");
19    }
20 }
```

   Le résultat est la figure de droite dans l'énoncé.

6. Même si cela est strictement interdit par les précepts de programmation sur système embarqué, proposer une implémentation utilisant une représentation des nombres à virgule flottante (`float`) et comparer le temps d'exécution avec l'implémentation sur des entiers. Quelle étape de traitement faut-il absolument éliminer pour rendre la mesure pertinente ?

   Il faut éliminer la communication sur port série qui prend un temps considérablement plus long que le temps de calcul. Une fois cette précaution prise, les résultats du temps de calcul (en seconde) sont, pour 30 itérations du calcul complet de la fractale :

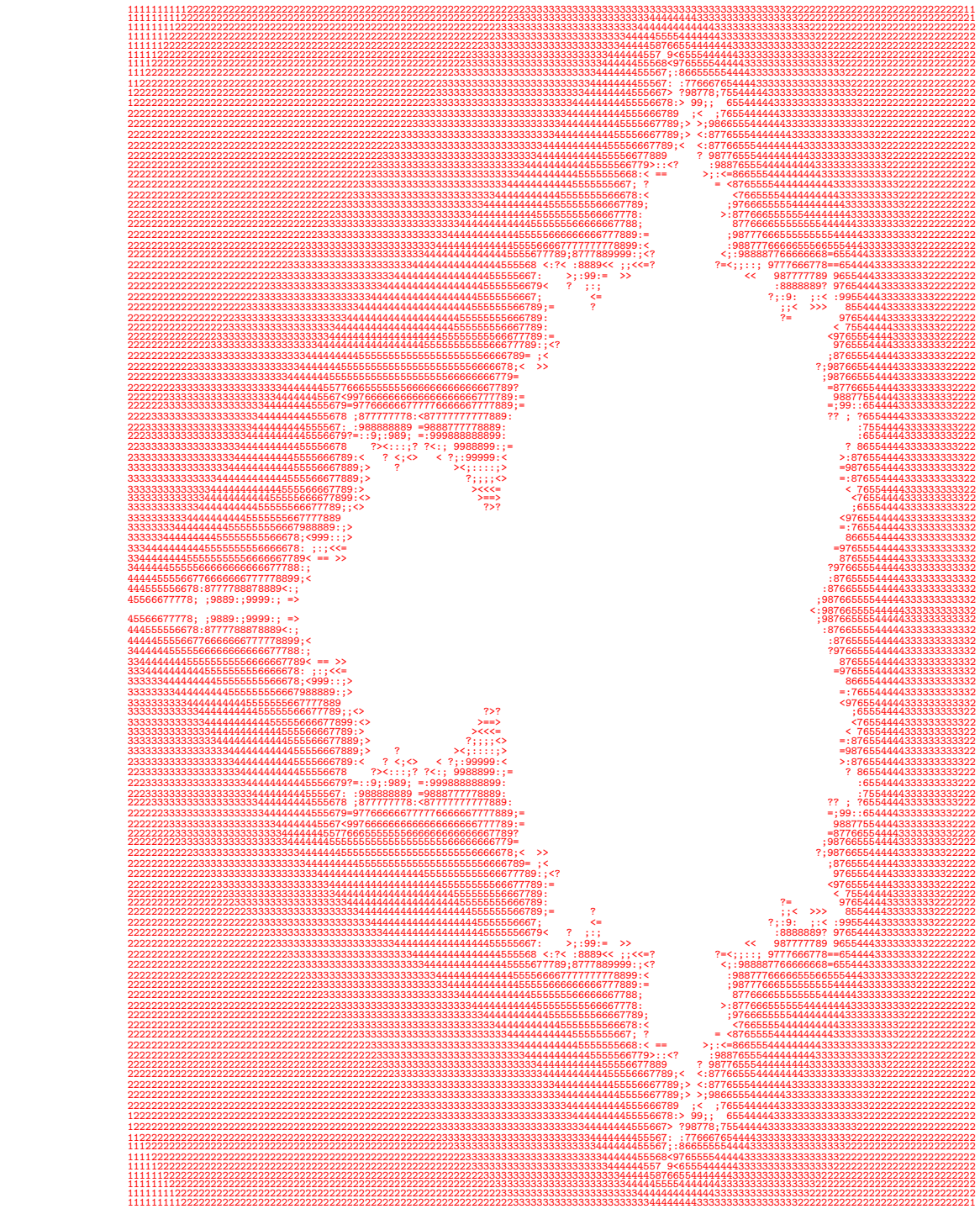
Calcul	STM32F3	STM32F4
flottants	55	5
int, /1000	12	5
int, $\gg$10	11	5

7. Au lieu d'afficher un symbole quelconque si la suite converge et un autre symbole lorsqu'elle diverge, afficher plutôt dans le second cas le nombre d'itérations qu'il a fallu pour atteindre le module pour lequel la suite est supposée diverger. L'image résultante sera bien plus élégante.

   Il suffit de changer une unique ligne dans le code après la condition `while` avec

```
1    if (n<16) printf("%c",'0'+n); else printf("␣");
```

   et en effet le résultat est bien plus beau :



On pourra passer un temps infini à étudier la fractale de Mandelbrot et tous ses petits ensembles de Julia au moyen de `xaos`.

Références

[1] H.-O. Peitgen, H. Jürgens, D. Saupe, *Chaos and Fractals : New Frontiers of Science*, Springer (1992)

[2] B. Mandelbrot, *The Fractal Geometry of Nature*, Times Books (1982)

[3] R. Brooks & P. Matelski, *The dynamics of 2-generator subgroups of PSL(2,C)*, in Irwin Kra (ed.). *Riemann Surfaces and Related Topics : Proceedings of the 1978 Stony Brook Conference*