TP : Macros C et tableaux

Projet L3 Vision

Année scolaire 2021 – 2022

Présentation

Le but de ce TP est de se familiariser avec quelques points de base du langage C, à travers le tracé de figures utilisant des algorithmes simples.

Les objectifs de ce sujet, du point de vue du langage C, sont les suivants :

— Utilisation de macros et du préprocesseur C
— Manipulation des tableaux
— Manipulation des opérateurs binaires «, », |, &
— Utilisation des types C99

Tout au long de ce sujet, il vous est demandé de vérifier la correction de l'appel au programme, i.e. que le nombre et le type des arguments de la ligne de commandes sont corrects. Dans le cas contraire, votre programme devra afficher un message d'aide avec la forme d'utilisation correcte. Cette aide pourra aussi être consultée en appelant votre programme avec l'option -h.

Nous vous demandons par ailleurs de n'utiliser que des types C99 pour vos variables : `uint32_t`, `int16_t`, etc.

Nous vous conseillons enfin de garder une copie des sources de votre programme pour chacune de questions, car certaines questions impliquent de supprimer des parties de code.

1 Exercice préliminaire : Manipulation du préprocesseur C (cpp)

1.1 Affichage conditionnel

À l'aide de la directive `#define`, réaliser une macro `debug_trace` prenant en paramètre une chaine à afficher au format de printf ainsi qu'un niveau de trace, et n'affichant la trace que si le niveau passé en paramètre est inférieur à une constante globale définie pour tout le fichier (on pourra se renseigner sur la macro `__VA_ARGS__`).

Ainsi, dans le fichier suivant ne sera affiché que le deuxième appel à la macro.

```
#define LEVEL_TRACE 3
/* Définition de la macro debug_trace */
...
ma_variable = 5;
debug_trace(4, "ma_variable : %d\n",ma_variable);
...
debug_trace(0, "Fin du main\n");
```

Cela permet de donner une importance différente à chaque trace. En positionnant `LEVEL_TRACE` à 0, on désactive toutes les traces ; en le mettant à 1, on active que les traces les plus importantes, etc.

Pourquoi n'est-il pas possible de faire cela avec une fonction standard ?

1.2 Affichage d'informations supplémentaires

Écrire une macro `debug_info`, prenant en paramètre un message et qui affiche ce message, mais précédé du nom de fichier, numéro de ligne du message et nom de la fonction dans lequel se trouve l'appel, le tout sous le format suivant :

```
[fichier.c: ma_fonction, l.57] message
```

Il existe en C99 des macros `__FILE__`, `__func__` et `__LINE__` pour vous aider à faire cela. Votre macro ne devra faire qu'un seul appel à `printf`.

Que se passe-t-il si on appelle cette macro avec une simple chaine, comme "Bonjour" ? Pourquoi ? Quelle est l'extension Gnu pour contourner ce problème ?

1.3 Combinaison des deux

En utilisant la macro de la question précédente, écrire une macro `debug_info_trace` qui prend en paramètre un message et un niveau de trace, et qui affiche le message avec les mêmes informations qu'à la question précédente, mais seulement si le niveau de trace passé en paramètre est inférieur à la constante globale définie. Vérifier le bon fonctionnement de cette macro.

2 Tracé de cercle

La première partie du sujet consiste à utiliser l'algorithme d'Andres pour tracer un cercle sur des points discrets (pixels). L'algorithme d'Andres pour un octant est fourni figure 1 (les autres peuvent être obtenus par symétrie).

```
1  // r est le rayon du cercle
2  x := 0
3  y := r
4  d := r - 1
5  Tant que y >= x
6          TracerPixel(x, y)
7          Si d >= 2x alors
8                  d := d - 2x - 1
9                  x := x + 1
10         Sinon si d <= 2(r - y) alors
11                 d := d + 2y - 1
12                 y := y - 1
13         Sinon
14                 d := d + 2(y - x - 1)
15                 y := y - 1
16                 x := x + 1
17 Fin tant que
```

FIGURE 1 – Algorithme d'Andres pour un octant

Image produite par l'algorithme

La principale difficulté de cette partie consiste en l'écriture de la sortie.

De manière à pouvoir visualiser le résultat, on se propose d'utiliser le format d'image ppm (Portable Pixel Map). Ce format très simple consiste en un tête spécifiant le type d'image (ex : couleur ou noir et

blanc) et ses dimensions, puis en la suite des pixels de l'image ligne par ligne, chaque pixel étant codé sur 3 octets : un pour le rouge (R), un pour le vert (V) et un pour le bleu (B).

Il est possible d'aller voir à l'adresse *http ://netpbm.sourceforge.net/doc/ppm.html* pour plus de détails sur ce format. Néanmoins, dans le cadre du travail demandé, le fichier devra commencer par l'en-tête suivant :

```
P6
LARGEUR HAUTEUR
255
```

où LARGEUR et HAUTEUR sont la largeur et la hauteur de l'image au format texte.

Un tel format d'images est visualisable par beaucoup de visionneurs, comme par exemple `eog` (eye of gnome) sous linux/Unix. Vous pouvez aussi convertir vos fichiers ppm en images jpg ou png avec les commandes `pnmtojpg` ou `pnmtopng`.

Travail demandé

Pour cette question, vous devrez partir du fichier `andres.c` fourni. Le programme devra créer un fichier `cercle.ppm` de taille 400*400 avec un cercle de rayon 100 pixels, obtenu par l'algorithme d'Andres. Vous devrez veiller en particuliers aux points suivants :

- Vous utiliserez une variable de type `uint32_t *` pour la représentation interne de l'image dans le programme, soit un tableau à une dimension alloué dynamiquement. Chaque élément de ce tableau est composé de 4 octets, dont 3 serviront à stocker la valeur d'une composante (R, V ou B) d'un pixel. Le dernier octet (octet de poids fort) sera inutilisé.
- Vous utiliserez des macros pour accéder au tableau de l'image en lecture et en écriture, en passant en paramètre les dimensions du tableau et les indices i et j accédés. Il devra y avoir au moins deux versions des macros : une avec vérification des bornes et une autre sans.
- Vous utiliserez des macros pour vos affichages, telles que celles vues dans le premier exercice.
- Vous utiliserez des macros ou des fonctions `static inline` pour récupérer les valeurs de chacune des composantes couleur d'un pixel (`get_r`, `get_v`, `get_b`)
- Vous vérifierez le bon fonctionnement de votre programme avec la commande `valgrind` qui doit ne produire aucune erreur. Vous réaliserez le débug de votre code à l'aide de macros, de valgrind et de gdb.

3 Tracé d'une fractale de Mandelbrot

On se propose désormais de tracer une image représentant une partie de l'ensemble de Mandelbrot. L'ensemble de Mandelbrot est l'ensemble des points c du plan complexe tels que la suite : $z_{n+1} = z_n^2 + c$ et $z_0 = 0$ converge vers un point de $\mathbb{C}$.

Dans cette partie, on demande de produire une image en noir et blanc de l'ensemble : un point est colorié en noir s'il appartient à l'ensemble, et en blanc sinon. Les intervalles à utiliser sont $[-2.2; 0.8]$ pour les abscisses, et $[-1.5; 1.5]$ pour les ordonnées. L'image fournie devra être aux dimensions 400*400 pixels.

Pour déterminer si un point diverge ou converge, il est possible de s'arrêter après 1000 itérations. Enfin, on pourra remarquer que si $\Re(z)^2 + \Im(z)^2 \geq 4$, alors la suite diverge.

4 Tracé d'autres fractales de Mandelbrot

On souhaite désormais pouvoir spécifier d'autres valeurs pour le sous-ensemble à visualiser, ainsi que pouvoir fournir des tailles d'images différentes. Comme on souhaite ne pas avoir de déformation dans l'image, les paramètres sont les suivants :

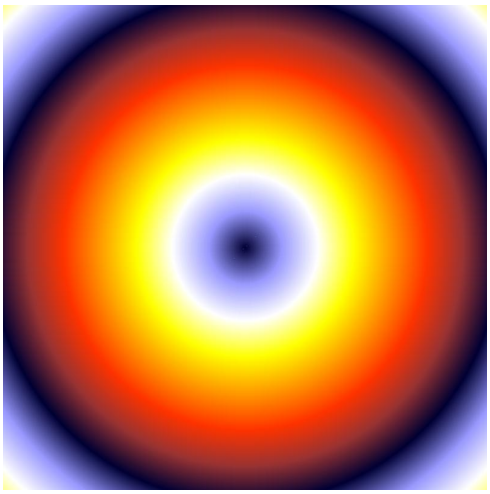


FIGURE 2 – Exemple de figure que l'on peut obtenir avec le tracé de cercles concentriques par la méthode d'Andres

— Résolution horizontale et résolution verticale de l'image
— Coordonnées X et Y du centre du sous-ensemble (centre = $X + iY$)
— Intervalle horizontal du sous-ensemble
— Nombre d'itérations maximal

Le nombre d'itérations maximal ne peut pas être connu à l'avance, mais est dépendant soit du zoom sur le sous-ensemble, soit de l'intervalle horizontal représenté. Quelques exemples d'images sont donnés dessous. Si vous voulez en faire d'autres, à vous de trouver une valeur suffisamment grande, mais pas trop pour ne pas alourdir inutilement le calcul.

Modifier votre code précédent pour prendre en compte ces paramètres. On pourra dans un premier temps rendre ces paramètres génériques par l'intermédiaire de macros. On demande par la suite de prendre en compte ces paramètres sur la ligne de commande. Il faudra penser à spécifier le format de la ligne de commande. Un format possible est :
`./mandelbrot <RESOL_X> <RESOL_Y> <CENTRE_X> <CENTRE_Y> <SPAN_X> <NB_ITER_MAX>`
Il faudra entre autres utiliser les fonctions `atoi` et `atof`.

Quelques valeurs de paramètres d'images sont données dans la table 1.

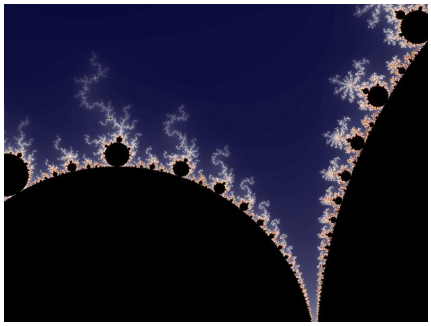
5 Utilisation de couleurs pour le tracé de la fractale

De manière à rendre les dessins plus jolis, on souhaite dans cette section utiliser des couleurs pour le tracé de la fractale. L'approche est la suivante : on définit un petit nombre P de couleurs (par exemple jaune (#FFFF00), vert (#00FF00), bleu (#0000FF) et rouge (#FF0000)), appelées paliers, et une valeur N correspondant au nombre de couleurs entre deux paliers, de manière à obtenir une transition entre ces deux paliers. À partir de ces informations, il faut générer toutes les couleurs correspondantes (au nombre de $(P-1) \times N$). Entre autres, il faut bien faire attention à diviser les paliers selon les 3 composantes R, V et B.

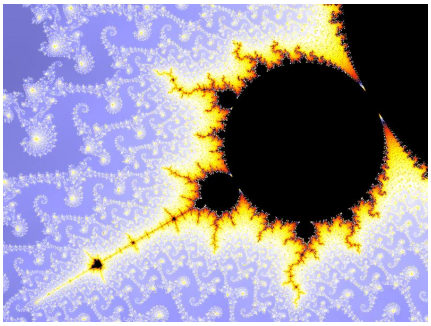
Une fois toutes les couleurs obtenues, il suffit de colorier un point divergeant avec la couleur ayant pour indice le numéro de l'itération ayant fait diverger la suite (modulo le nombre de couleurs total).

Implémenter cette fonctionnalité, et intégrer la spécification des couleurs dans le fichier de configuration.

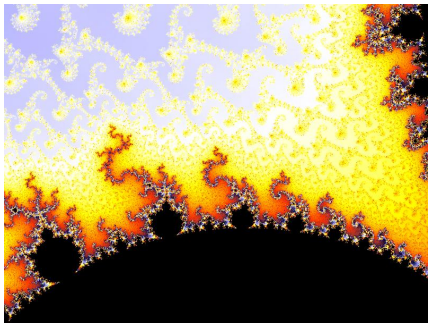
TABLE 1 – Exemples de paramètres pour la fractale de Mandelbrot



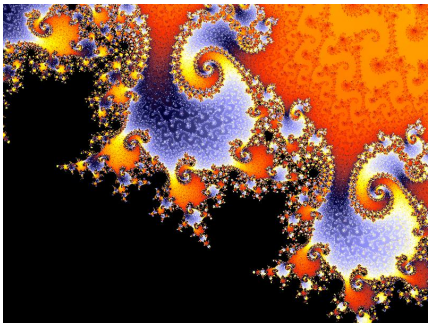
Centre x : -0.87591
Centre y : 0.25464
Longeur de l'intervalle : 0.53184
Nombre d'itérations max. : 1000



Centre x : -0.7436447860
Centre y : 0.1318252536
Longeur de l'intervalle : 2.9336×10^{-6}
Nombre d'itérations max. : 3000



Centre x : -0.74364421961
Centre y : 0.13182604688
Longeur de l'intervalle : 6.6208×10^{-7}
Nombre d'itérations max. : 5000



Centre x : -0.743643905055
Centre y : 0.131825885901
Longeur de l'intervalle : 4.9304×10^{-8}
Nombre d'itérations max. : 8000