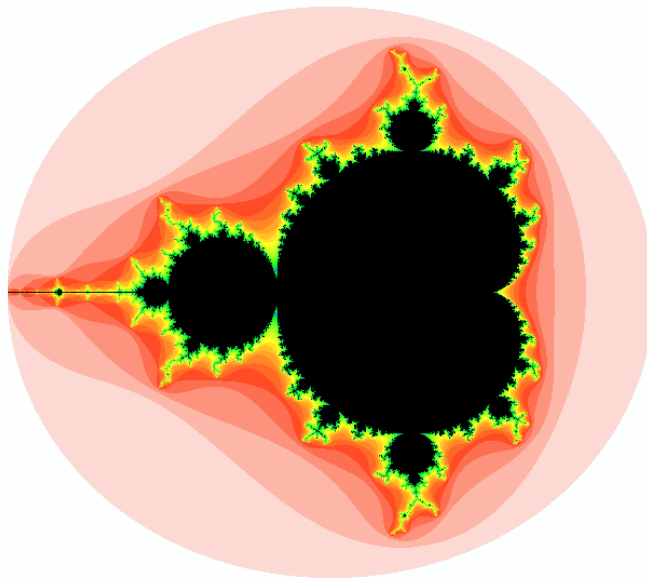


BUREAU D'ÉTUDES D'INFORMATIQUE

Fractales et FLTK

PMPB

Julia DUPUIS
Nils EXIBARD
Félix PIÉDALLU



31 mai 2014

Table des matières

1	Présentation du programme	1
2	Sources	3
2.1	Affichage	3
2.2	Structures de données	3
2.3	Fonctions	4
2.4	Dessin	5
2.5	Callback	5
3	Critique	5
4	Déroulement du projet	5
5	Conclusion	6

1 Présentation du programme

Le sujet sur lequel nous avons choisi de travailler est le projet « Fractales », grand classique de la programmation, utilisant le plan complexe.

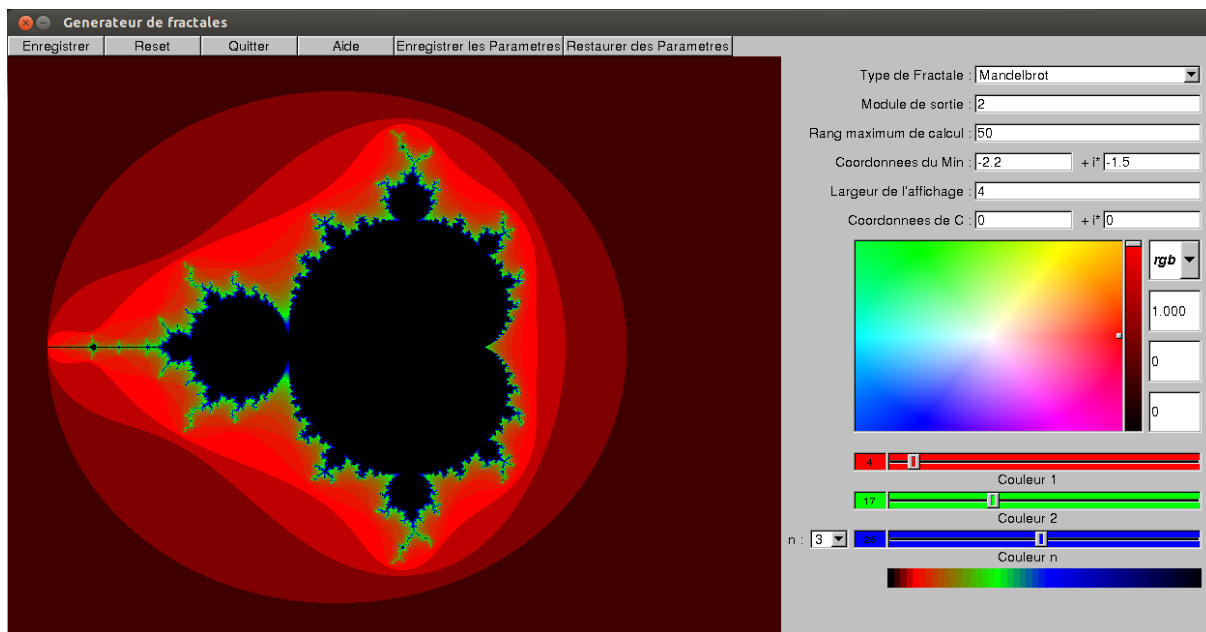


FIGURE 1 –

Les fonctionnalités que nous avons choisi d'intégrer sont les suivantes :

- Choix par l'utilisateur des différents paramètres de la fractale (type, module de sortie, rang maximal, coordonnées du point inférieur gauche, largeur de l'affichage, coordonnées de la constante).

Type de Fractale :

Module de sortie :

Rang maximum de calcul :


Coordonnees du Min : + i*

Largeur de l'affichage :


Coordonnees de C : + i*


FIGURE 2 –


- Choix des couleurs (jusqu'à 10 couleurs choisies par l'utilisateur avec écart modifiable).



rgb

 Couleur 1

 Couleur 2

n :  Couleur n




FIGURE 3 –

- Enregistrement de l'image ainsi que des paramètres dans un fichier choisi par l'utilisateur avec une qualité choisie.
- Restauration de paramètres précédemment enregistrés.
- Remise à zéro des paramètres (bouton reset).
- Fonction « Quitter ».
- Un bouton aide expliquant les fonctionnalités des différents boutons de la souris.

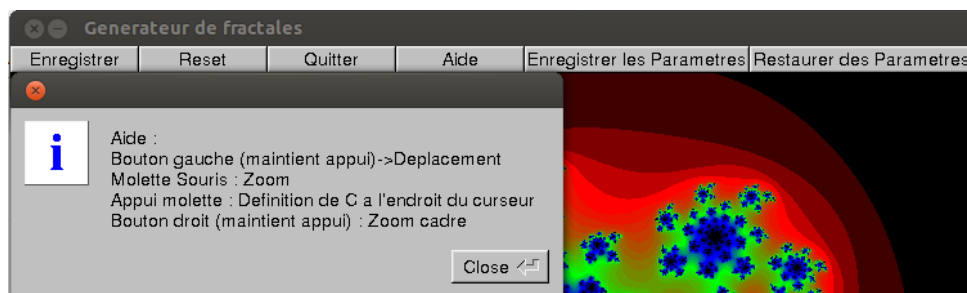


FIGURE 4 –

- 2 types de zoom : avec cadre (clic droit) ou grâce à la molette .

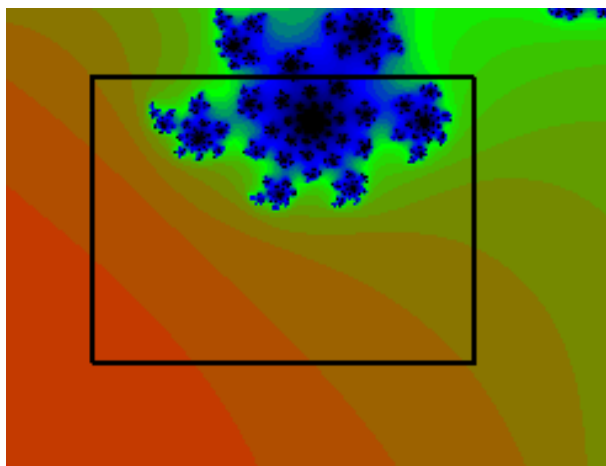


FIGURE 5 –

— Déplacement de l'image (clic gauche maintenu).

2 Sources

2.1 Affichage

2.2 Structures de données

```
struct Pixel {
    complex<double> z;           //Coordonnées du pixel dans le plan complexe
    int n;                      //Rang de divergence du pixel
};
```

Cette structure permettra de tracer la fractale à partir du rang de divergence et de la coordonnée de chaque pixel.

```
struct Donnees {
    enum fractype Fractale; // Type de fractales choisie (Type NumCr)
    int rangMax;           // Rang maximal de convergence
    double moduleMax;      // Module de convergence (determination de la
                           // convergence ou non de la fonction)
    complex<double> C;      // Constante de calcul
    complex<double> ig;     // Coordonnées du point inferieur gauche
    double pasxy;          // Pas de la matrice (incrementation, en fait,
                           // et egale dans les 2 dimensions, car pixels carres)
    struct Pixel Tab[L_ZONE][H_ZONE]; // Matrice des pixels de l'image.
    int hauteur;           //Hauteur de l'image
    unsigned char buffer[3*L_ZONE*H_ZONE]; //contient l'image sous forme
    // RGB
    unsigned char bufferDeg[3*325]; //contient l'aspect du d'grad sous
    // forme RGB
    int nbSlider;          //Nombre de slider actifs
    unsigned long int slider[MAX_SLIDER+2][2]; //contient le rang et la
    // couleur de chaque slider
};
```

Cette structure est la plus importante du programme.

```
struct Tests {
    bool dessin; //faut-il refaire le dessin?
    bool calcul; //faut il refaire le calcul
    bool calccouleurs; //faut il refaire le calcul des couleurs ?
};
```

```

    int slider; //contient le slider actif
};

```

Cette structure contient les variables nécessaires à la vérification des conditions, cela permet entre autre d'éviter les calculs inutiles.

```

enum fractype {
    MANDELBROT,
    JULIA,
    COSC,
    SINZO,
    PERSONNA //Originellement prévue pour une fractale personnalisée, non←
             utilisée faute de temps
};

```

Le type énuméré permet d'utiliser plus facilement les différents types de fractales.

2.3 Fonctions

```

void InitialiserDonnees();

```

Initialise les données du programmes

```

void realFromTab(double *bi, double *bj);

```

Effectue la correspondance entre les coordonnées complexe et les pixels

```

pointeurFct retourne_fonction(); // Pointe vers les fonctions suivantes←
    en fonction de la fractale choisie

```

```

int mandelbrot(complex<double> position);
int julia      (complex<double> position);
int sinzo      (complex<double> position);
int cosc       (complex<double> position);
int persona    (complex<double> position);

```

Ce sont les algorithmes de calcul de convergence

```

void convergenceLigne(int j, pointeurFct fonction);

```

Etudie la convergence ligne par ligne

```

void degradeRGB(unsigned long int A, unsigned long int B, int N, int tab←
    [[3]]);

```

Effectue le calcul d'un dégradé de taille N entre une couleur A et une couleur B, et stocke les trois composantes RGB dans tab

```

void couleursRGB(unsigned long tabSlider [[2], int tab [[3]]) ;

```

Remplis tab d'une suite de dégradé grace à degradeRGB à partir des informations contenues dans tabSlider

```

void calcBuffer(int tabdeg [[3]]);

```

calcule et stocke dans gDonnees.bufferDeg a partir d'un tableau de couleurs RGB

```

int enregistrerPPM(int Largeur, char Fichier [32]);

```

Enregistre une image en ppm de Largeur pixel de large et de ratio constant dans Fichier.

```

void enregistrerParams(const char* fichier);
void restaurerParams(const char* fichier);

```

Permet d'enregistrer (et de lire, mais non finalisé) les paramètres permettant de redessiner la fractale

2.4 Dessin

```
void ZoneDessinInitialisation(Fl_Widget* widget, void* data);
```

```
void afficheFractaleLigne();
```

Commande le calcul puis l'affichage d'une ligne

```
void afficheLigneRGB(int j, int tableauCouleurs[][3]);
```

Affiche la ligne j avec pour couleurs un tableau RGB

```
void gestionAffichage_iter(void* data); //iter car en remplacement de la ↔  
fonction recursive d'origine.
```

Calcule SI nécessaire la fractale puis l'affiche grace aux diverses autres fonctions.

```
void tracerCadre (int x1, int y1 , int x2, int y2);
```

Trace le cadre du zoom cadre a partir de 2 points (conserve le ratio d'écran)

```
void zoneDegrade(Fl_Widget* widget, void* data);
```

Gère l'affichage de la zone d'aperçu du dégradé

2.5 Callback

3 Critique

- La fonction d'affichage est chaotique, elle appelle d'abord initialiser affichage qui appelle elle même gestionAffichage_iter.
- La structure de données initiale pour les couleurs, qui limitaient le programme à 3 sliders. Il a fallu réécrire beaucoup de choses pour pouvoir augmenter le nombre de slider (on peut maintenant potentiellement mettre autant de point de couleurs que l'on veut),
- Le tableau contenant le dégradé n'est pas alloué dynamiquement->soit on utilise un très grand tableau en variable globale, soit on le recalcule dès qu'on en a besoin (option choisie, cela entraîne quelques calculs supplémentaire mais de temps faible devant le calcul de la fractale)
- Le programme ne fonctionne pas uniquement en RGB, mais on utilise un peu Fl_Color, car nous nous sommes rendus compte après avoir tout implémenté avec Fl_Color qu'il existait des accesseurs permettant d'accéder directement aux composantes RGB. Certains algorithmes ont été réécrits en RGB, qui est bien plus pratique à se représenter, mais aussi pour écrire dans des buffer (nous avons abandonnés l'affichage point par point avec Fl_point pour utiliser Fl_draw_image, bien plus rapide).
- La gestion du multithreading, commencée mais abandonnée faute de temps (fonctionnait pour le calcul mais produisait des erreurs pour l'affichage).
- La gestion des animations (couleurs variable) a été commencée, mais ayant a l'époque un problème d'affichage (passage par un écran noir a chaque nouveau dessin à cause de l'appel de la fonction "initialiser__affichage", le résultats était peu convainquant. Après résolution du problème, nous n'avons pas eu le temps de la ré-implémenter.

4 Déroulement du projet

Nous avons à l'origine effectué une répartition des taches (Julia pour l'interface et les Callbacks, Nils pour la gestion des couleurs, et Félix pour l'algorithmique des fractales et l'affichage).

Cette répartition a été dans l'ensemble tenue, pour la base du programme, mais ensuite chaque nouvelle fonctionnalité a été codée intégralement par celui qui voulait l'implémenter, sans refaire de répartition des tâches.

5 Conclusion

Coder un programme entier s'est révélé très intéressant même si très coûteux en temps, surtout pour arriver à une interface assez intuitive (si l'on connaît un minimum le vocabulaire des fractales) et efficace. Cela nous a permis de comprendre l'architecture d'un programme basé sur plusieurs fichiers et l'importance des structures de données. En effet, avec le recul, nos choix initiaux ne paraissent pas toujours optimaux, notamment si l'on veut modifier ou ajouter certaines fonctions (par exemple, il a fallu recoder beaucoup de chose pour passer à plus de trois point de couleurs, ce qui aurait pu être évité en choisissant une autre structure de donnée dès le départ).