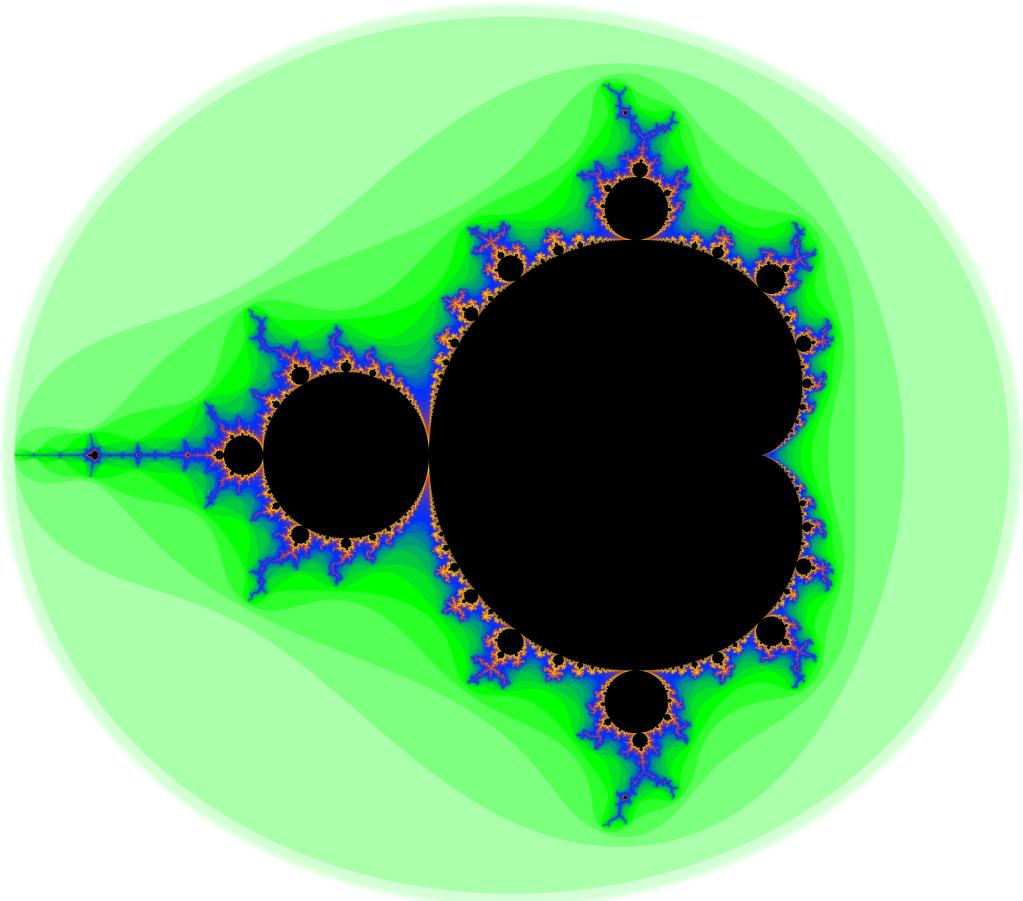
Grenoble INP phelma

Bureau d'Études d'Informatique

Fractales et FLTK

pmpB

Julia Dupuis
Nils Exibard
Félix Piédallu



1er juin 2014

Table des matières

1 Présentation du logiciel - Notice d'utilisation

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

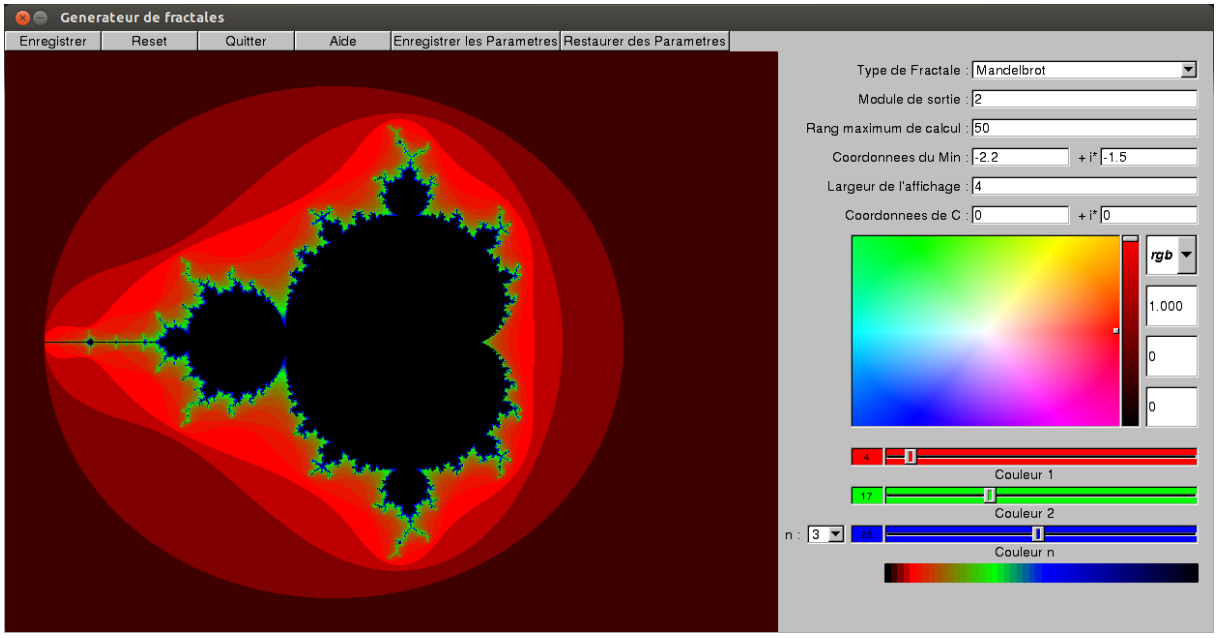


FIGURE 1 – Aperçu du logiciel

Ce logiciel permet d'afficher et d'explorer différents types de fractales, notamment les ensembles de Mandelbrot et de Julia. On peut se déplacer, zoomer, et choisir les couleurs du dégradé permettant un affichage intéressant de la fractale.

Le logiciel se veut assez intuitif pour qui connaît le vocabulaire des fractales, et ses principales fonctions sont décrite ici :

— Une section de l'interface permet de choisir les différents paramètres de calcul et d'affichage de la fractale (type, module de sortie, rang maximal, coordonnées du point inférieur gauche, largeur de l'affichage, coordonnées de la constante C utilisée par Julia)

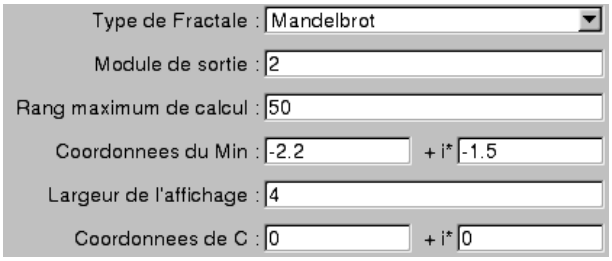


FIGURE 2 – Choix des paramètres généraux de la fractale

— Une autre permet la gestion des couleurs (jusqu'à 10 couleurs, avec écart modifiable selon le rang de divergence).

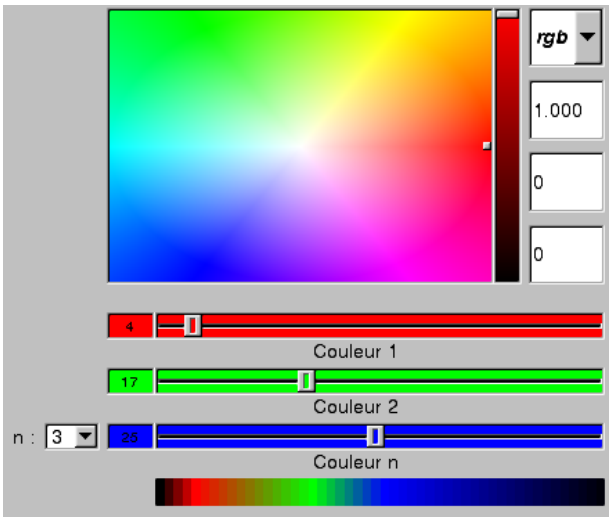


FIGURE 3 – Choix de la couleur et des points de contrôle du dégradé

— On peut enregistrer l'image a résolution souhaitée, ou bien n'en enregistrer que les paramètres.
— Restaurez les paramètres précédemment enregistrés.
— Réinitialisez les paramètres (Bouton Reset).
— Fonction « Quitter ».
— Un bouton "Aide" donne les interactions possible avec la souris.

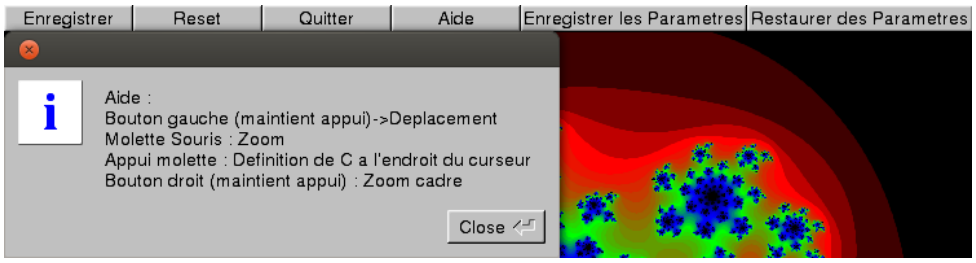


FIGURE 4 – Petite aide

— Déplacer l'image avec le clic gauche maintenu.

— Zoom cadre (bouton droit) ou zoom centré sur la souris (molette).

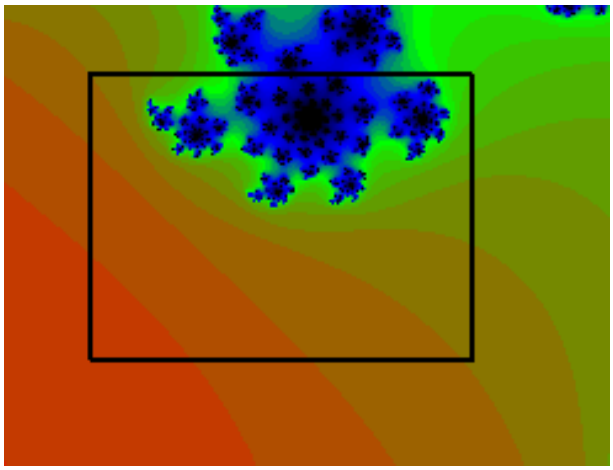


FIGURE 5 – Zoom cadre, en maintenant le bouton droit de la souris

2 Sources - Notice de maintenance

2.1 Interface

L'interface est constituée de deux zones de dessin (la zone d'affichage principale et la zone d'affichage du dégradé, et d'un certain nombre de widgets. Chaque widget est relié à son callback widgetCB. Pour une organisation plus simple de l'interface utilisateur, des variables de positions sont utilisées pour faciliter l'insertion d'un widget ou le positionnement de chaque widget.

2.2 Structures de données

```
struct Pixel {
    complex<double> z;      //CoordonnÃ©es 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 Ã©numÃ©rÃ©)
    int     rangMax;        // Rang maximal de convergence
    double  moduleMax;      // Module de convergence (determination de la ↩
       convergence on non de la fonction)
    complex<double> C;      // Constante de calcul
    complex<double> ig;     // Coordonnees 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'apperÃ§u du dÃ©gradÃ© 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 prÃ©vue pour une fractale personnalisÃ©e, non↩
        utilisÃ©e 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();
```

Retourne la fonction de calcul de la fractale, pour son utilisation dans le projet.

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

Voici alors les fonctions fractales utilisées; elles retournent un rang de convergence en un point du plan complexe.

```
void convergenceLigne(int j, pointeurFct fonction);
```

Etudie la convergence ligne par ligne

```
void degradeRGB(unsigned long A, unsigned long 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]);
```

Remplit tab d'une suite de dégradé grâce à degradeRGB à partir des informations contenues dans tabSlider

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

Calcule et stocke dans gDonnees.bufferDeg à 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 grâce 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

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

Gère l'ensemble des événements commandables à l'aide de la souris, à savoir le zoom molette, le zoom cadre(clic droit), le déplacement de l'affichage(clic gauche) et choix de la constante (clic molette)

```
void ChampModuleDeSortieCB(Fl_Widget* w, void* data);
```

Permet de régler le module de sortie de la fractale (module à partir duquel on considère qu'il y a convergence)

```
void ChampProfondeurCB  (Fl_Widget* w, void* data);
```

Choix du rang maximal d'étude

```
void MenuFractaleCB     (Fl_Widget* w, void* data);
```

Met en relation la fractale choisie par l'utilisateur avec sa fonction de calcul.

Voici les différents boutons de la barre de menu :

```
void BoutonQuitterCB    (Fl_Widget* w, void* data);
```

```
void BoutonEnregistrerCB(Fl_Widget* w, void* data);

void BoutonResetCB      (Fl_Widget* w, void* data);

void BoutonAideCB       (Fl_Widget* w, void* data);

void BoutonSaveParamsCB (Fl_Widget* w, void* data);
```

Permet de sauvegarder les paramètres actuels.

```
void BoutonBackParamsCB (Fl_Widget* w, void* data);
```

Permet de revenir aux paramètres précédemment sauvegardés.

```
void ChampXMinCB        (Fl_Widget* w, void* data);
void ChampYMinCB        (Fl_Widget* w, void* data);
void ChampLargeurCB     (Fl_Widget* w, void* data);
```

Définit le point inférieur gauche et la largeur de l'affichage (taille en complexe)

```
void ChampCXCB          (Fl_Widget* w, void* data);
void ChampCYCB          (Fl_Widget* w, void* data);
```

Permet de rentrer une valeur de constante (utilisée dans certaines fractales)

```
void CarreChoixCouleurCB(Fl_Widget* w, void* data);
void Slider1CB          (Fl_Widget* w, void* data);
void Slider2CB          (Fl_Widget* w, void* data);
void ChoixSliderCB    (Fl_Widget* w, void* data);
void Slider3CB          (Fl_Widget* w, void* data);
```

Ces trois callbacks permettent la gestion des couleurs

```
void setColorChooserColor(unsigned long int couleur);
```

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). Au cours du développement il aurait donc été judicieux de tout travailler de manière à ce que les paramètres principaux soient facilement modifiables,une solution aurait été d'utiliser des fonctions intermédiaires entre les fonctions de calcul et les structures de données.
- 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 étés réécris 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 avions à l'origine effectué une répartition des tâches (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.

Une communication constante et le respect des dénominations ont été particulièrement importantes pour développer de manière cohérente le projet et prévenir les dysfonctionnement lors de la complexification du projet.

De plus, nous avons décidé dès le début du projet d'utiliser Git, un gestionnaire de version, très utile pour la gestion d'un projet d'informatique à plusieurs. Cette plateforme nous a permis une communication du code très facilitée, et donc une organisation beaucoup plus claire.
Notre projet se trouve sur Github : http://github.com/Salamandar/Fractalis

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.
L'un des problèmes les plus difficiles à appréhender a été l'affichage, car il s'agit d'avoir une bonne définition et un visuel attractif tout en limitant le temps de création de l'image. Nous avons réussi notre pari grâce à une fonction d'affichage assez complexe, tout en mettant en commun les connaissances de chacun.

Notre organisation a permis une rapide évolution du logiciel une fois les briques de base posées.
En effet, comme nous l'avons précisé plus haut, la répartition des tâches, très souple, nous a permis d'utiliser les compétences de chaque membre dans chaque partie du logiciel. Nous sommes donc assez peu restés bloqués grâce à ce travail de coopération.

Ce travail 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 fonctionnalités (par exemple, il a fallu recoder beaucoup de choses pour passer à plus de trois points de contrôle du dégradé, ce qui aurait pu être évité en choisissant une autre structure de donnée dès le départ).

En résumé, ce projet nous a permis de découvrir de façon très intéressante la programmation modulaire et en groupe, et les bases qu'il nous a apportées nous permettront d'appréhender plus sereinement de futurs projets.