Table des matières

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.

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

— Calcul et tracé dans la zone de dessin de 4 types de fractales (Mandelbrot, Julia, sin, cos).
— 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).
— Choix des couleurs (jusqu'a 10 couleurs choisies par l'utilisateur avec écart modifiable).
— 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.
— 2 types de zoom : avec cadre(clic droit) ou grâce à la molette .
— Déplacement de l'image (clic gauche maintenu).

2 Sources

2.1 Structures de données

```
1  struct Pixel {
2      complex<double> z;       //CoordonnÃ©es du pixel dans le plan complexe
3      int n;                   //Rang de divergence du pixel
4  };
5  Cette strcure permettra de tracer la fractale Ã  partir du rang de ↩
       divergence et de la coordonnÃ©e de chaque pixel
```

```
1  struct Donnees {
2      enum fractype Fractale; // Type de fractales choisie (Type Ã©numÃ©rÃ©)
3      int     rangMax;        // Rang maximal de convergence
4      double  moduleMax;      // Module de convergence (determination de la ↩
           convergence on non de la fonction)
5      complex<double> C;      // Constante de calcul
6      complex<double> ig;     // Coordonnees du point inferieur gauche
7      double pasxy;           // Pas de la matrice (incrementation, en fait,↩
           et egale dans les 2 dimensions, car pixels carres)
8      struct Pixel Tab[L_ZONE][H_ZONE]; // Matrice des pixels de l'image.
9  int hauteur;              //Hauteur de l'image
10     unsigned char buffer[3*L_ZONE*H_ZONE];//contient l'image sous forme ↩
           RGB
11     unsigned char bufferDeg[3*325];//contient l'apperÃ§u du dÃ©gradÃ© sous↩
           forme RGB
12     int nbSlider;           //Nombre de slider actifs
13     unsigned long int slider[MAX_SLIDER+2][2];//contient le rang et la ↩
           couleur de chaque slider
14 };
```

Cette structure est la plus importante du programme.

```
1  struct Tests {
2      bool dessin;//faut-il refaire le dessin?
3      bool calcul;//faut il refaire le calcul
4      bool calccouleurs;//faut il refaire le calcul des couleurs ?
```

```
5       int slider; //contient le slider actif
6   };
```

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

```
1   enum fractype {
2       MANDELBROT,
3       JULIA,
4       COSC,
5       SINZO,
6       PERSONNA//Originellement prÃ©vue pour une fractale personnalisÃ©e, non↩
            utilisÃ©e faute de temps
7   };
```

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

2.2 Fonctions

```
1   void InitialiserDonnees() ;
```

Initialise les données du programmes

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

Effectue la correspondance entre les coordonnées complexe et les pixels

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

```
1   int mandelbrot(complex<double> position);
2   int julia     (complex<double> position);
3   int sinzo     (complex<double> position);
4   int cosc      (complex<double> position);
5   int personna  (complex<double> position);
```

Ce sont les algorithmes de calcul de convergence

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

Etudie la convergence ligne par ligne

```
1   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

```
1   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

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

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

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

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

```
1   void enregistrerParams(const char* fichier);
2   void restaurerParams(const char* fichier);
```

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

2.3 Affichage

```
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 rÃ©cursive d'origine.
```

Calcule SI nécéssaire 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'apperçu du dégradé

3 Critique

4 Déroulement du projet

```
#ifndef _Main_h
#define _Main_h
#include <FL/Fl_Widget.H>

class DrawingArea : public Fl_Widget{
public:
    DrawingArea(int X,int Y,int W,int H);
    void draw_callback( void (*Function) (Fl_Widget* w, void* data), void*↩
        Data);
    void mouse_callback( void (*Function) (Fl_Widget* w, void* data), void↩
       * Data);
    void keyboard_callback( void (*Function) (Fl_Widget* w, void* data), ↩
       void* Data);

private :
    void draw();
    int handle(int event);

    void (*_draw_callback_function) ( Fl_Widget* w, void* data);
    void* _draw_callback_data;

    void (*_mouse_callback_function) ( Fl_Widget* w, void* data);
    void* _mouse_callback_data;

    void (*_keyboard_callback_function) ( Fl_Widget* w, void* data);
    void* _keyboard_callback_data;
};
#endif
```