Groupe de séminaire D
ROSE Dominic
NGUYEN Minh Huy

À l'attention de Stefan Bornhofen

Rapport du projet différencié :

création d'une application permettant de générer des fractales

2011-2012

Les fractales

Une fractale ou figure fractale, est une courbe ou surface de forme irrégulière ou morcelée créée en suivant des règles déterministes ou stochastiques impliquant une homothétie interne. Le terme « fractale » est un néologisme créé par Benoît Mandelbrot en 19741 à partir de la racine latine *fractus*, qui signifie brisé, irrégulier. Dans la « théorie de la rugosité » développée par Mandelbrot, une fractale désigne des objets dont la structure est invariante par changement d'échelle.

Source : Wikipedia

Dans ce projet, nous nous sommes intéressés à quatre types de fractales.

1) Les fractales à temps d'échappement

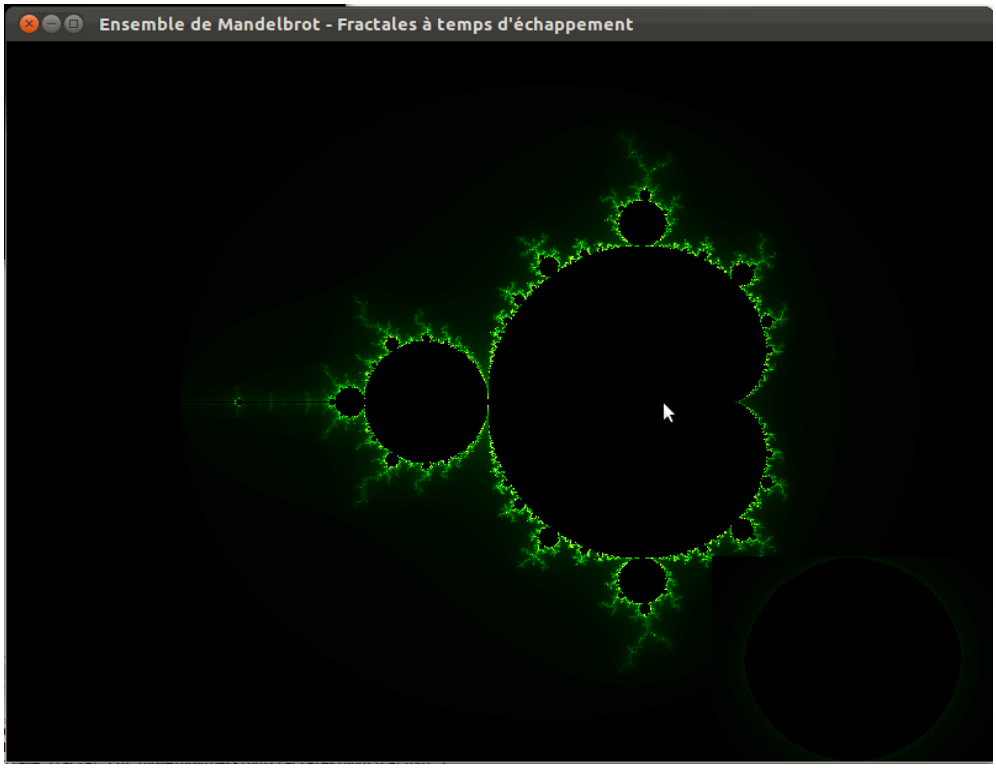


Illustration 1: Ensemble de Mandelbrot

Pour générer l'image de l'ensemble de Mandelbrot, on utilise un algorithme d'itération pixel par pixel. La formule utilisée est :

$$f(z) = z^2 + c$$

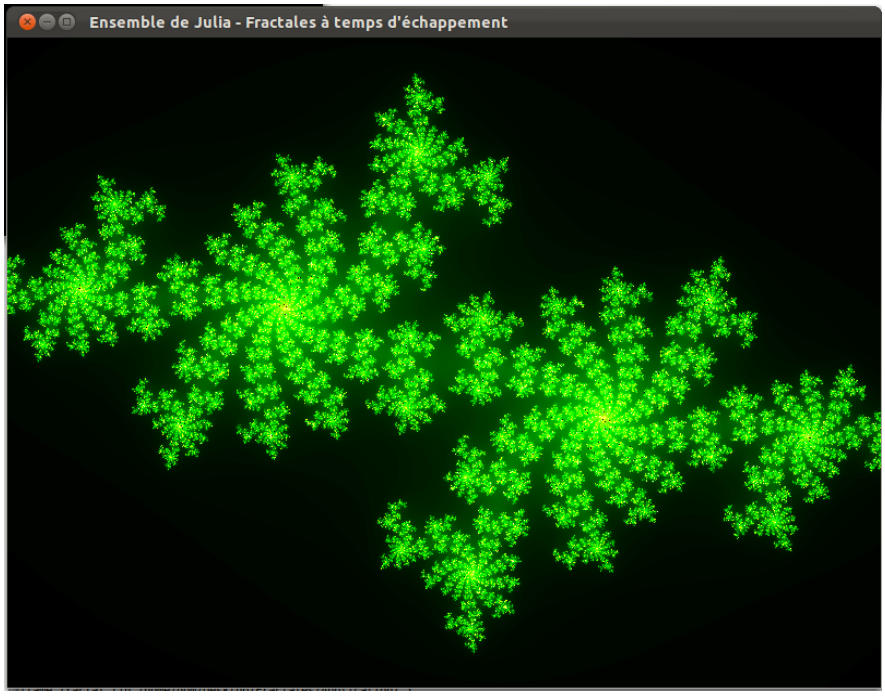
z est l'emplacement du plan complexe correspondant au pixel en cours d'analyse, du moins avant la première itération. Lors d'une itération on applique une transformation en utilisant des instructions qui sont équivalentes à :

$$z = f(z)$$

La constante c est et reste égale à l'emplacement du pixel dans le plan complexe. L'étape suivante lors d'une itération consiste à vérifier que le module de z, $|z|$, est inférieur à 2. Le nombre d'itérations requises pour échapper à cette condition est ensuite utilisé pour donner une couleur au pixel.

On parle bien de fractales à temps d'échappement, mais pour certains points, un nombre infini d'itérations ne suffirait pas à le faire échapper de ce cercle de rayon 2 centré sur l'origine. On définit donc un nombre maximal d'itérations, et on fixe une couleur particulière (le noir dans notre cas) aux points qui « survivent » à toutes les itérations.

Les algorithmes permettant de générer l'ensemble de Mandelbrot ou ceux de Julia utilisent la même équation, mais il y a une différence importante entre les deux. Pour un ensemble de Julia, la constante peut être choisie arbitrairement.



Notre façon de présenter les fractales à temps d'échappement est d'avoir un aperçu d'un ensemble de Julia qui vient se calquer sur notre image de l'ensemble de Mandelbrot dans un coin. Cet aperçu correspond à l'ensemble de

Julia où la constante c est égale à l'emplacement de la souris sur l'ensemble de Mandelbrot.

Lorsque l'on clique sur un pixel de l'ensemble de Mandelbrot, la fractale de Julia correspondant est affichée en grand. Un nouveau clique revient à Mandelbrot.

La représentation des fractales est souvent faite par ordinateur pour avoir une meilleure précision de l'infiniment petit, que l'on ne peut évidemment qu'approcher, car la fréquence des processeurs et la gestion des nombres flottants sont limitées.

2) Les IFS

Les IFS ont servi à développer une méthode de compression des images efficace mais qui n'a pas obtenu une grande diffusion et seuls des programmes spécialisés savent afficher ces images, contrairement au format JPEG bien connu.

C'est un type de fractales assez original qui a été introduit par Michael Barnsley. Leur structure est décrite par un ensemble de fonctions affines qui calculent les transformations appliquées à chaque point par homothétie, par translation et par rotation. Chaque transformation utilise 2 fonctions réelles (où une fonction complexe) pour calculer des nouvelles valeurs x1 et y1 à partir des coordonnées x0 et y0 de chaque point. Dans notre cas on a :

$$x1 = ax_0 + by_0 + e$$
$$y1 = cx_0 + dy_0 + f$$

a, c, c, d, e, et f sont des coefficients qui sont constants pour une même fractale de type IFS.

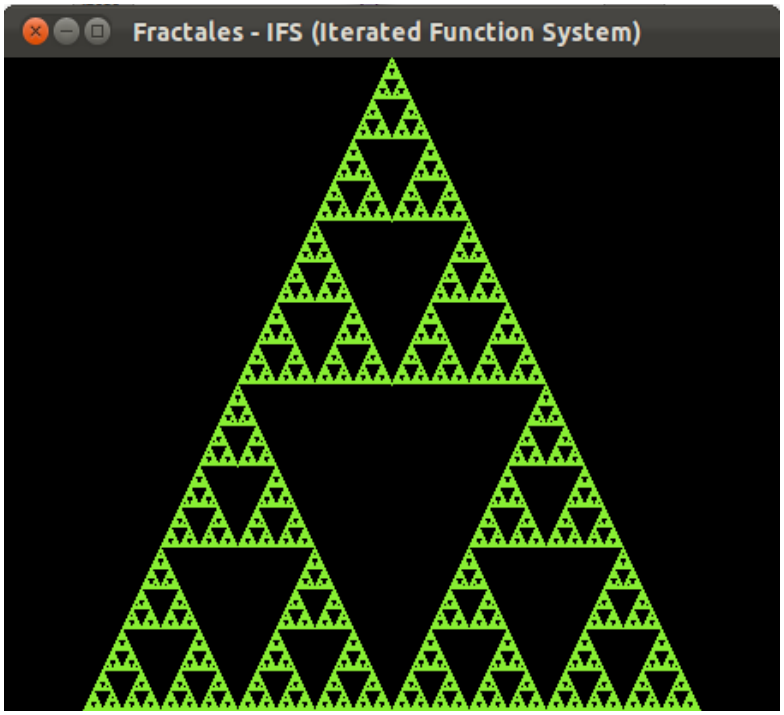
Enfin, une probabilité comprise entre 0 et 1 est associée à chaque transformation (le total de toutes les probabilités devant être 1). Ces images sont donc construites par un processus aléatoire : on voit apparaître sur l'écran des point de plus en plus nombreux qui dessinent une forme floue, puis de plus en plus précise. En général le programme permet de fixer le critère d'arrêt du calcul. Plus le temps choisi est long, plus les images sont précises.

En voici deux exemples :

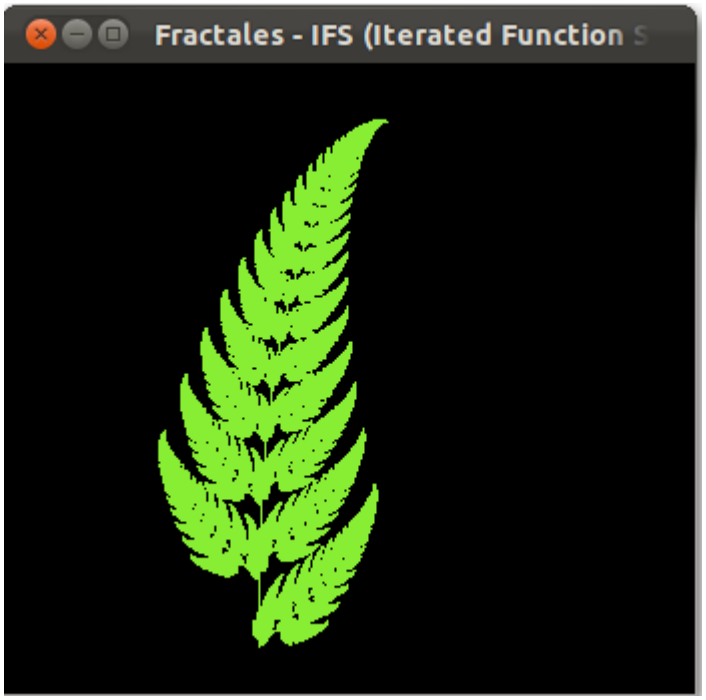
Le triangle (ou tamis) de Sierpinski.

Sierpinski

```
    Constantes :        a   b c    d e   f          p
transformation 1 : 0.5   0   0  0.5   0   0          0.333
transformation 2 : 0.5   0   0  0.5   1   0          0.333
transformation 3 : 0.5   0   0  0.5   0.5 0.8660254  0.334
```



Voici la célèbre fougère de Barnsley :



3) Les L-Systèmes

Les Lindenmayer-systèmes ont été inventées pour modéliser la croissance et les interactions cellulaires. Une L-système est une grammaire formelle dont les règles définissent la procédure pour transformer par itération une chaîne de caractères de départ en une autre chaîne. Les transformations sont basées sur des règles qui spécifient comment des caractères ou parties de la chaîne de caractères sont remplacés par d'autres. Si on applique cet ensemble de règles de façon récursive à une chaîne de caractère, une nouvelle chaîne ayant une structure fractale peut être créée.

Fondamentalement chaque caractère peut être un symbole ayant n'importe quelle signification conventionnelle, mais on a utilisé très tôt les L-systèmes pour dessiner des figures complexes, chaque caractère étant interprété comme une commande graphique.

Les conventions peuvent être choisies librement par chaque programme utilisant les L-systèmes. L'exemple suivant est extrait du manuel du célèbre programme de fractales : *Fractint*.

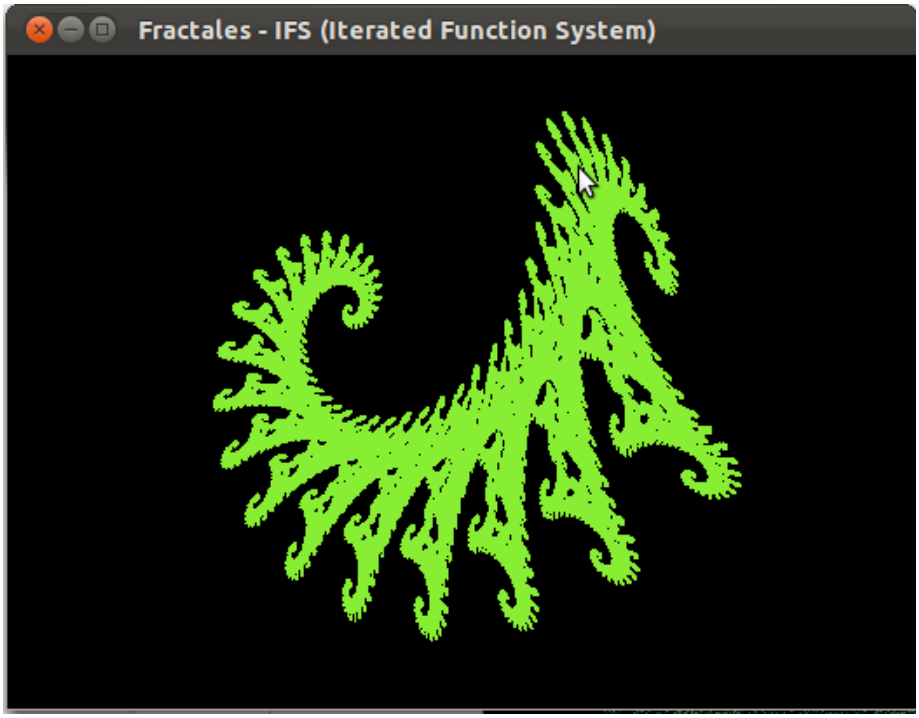


Illustration 2: IFS Dragon

La courbe du dragon ci-dessus a été obtenue en indiquant au programme d'appliquer cette procédure 8 000 000 de fois. Il s'agit du nombre d'itérations utilisées par notre programme d'IFS.

Enfin les L-systèmes ont été largement utilisées pour modéliser la structure des végétaux.

4) Les flammes

Ces fractales sont plus connues sous leur nom anglais (flame fractals). Il s'agit d'une combinaison d'attracteurs étranges produits par la méthode IFS, auxquels sont appliqués un ensemble de transformations très complexes. Comme toutes les fractales de type IFS elles se construisent progressivement par accumulation de points apparaissant de façon semi-aléatoire. En utilisant un nombre d'itérations élevé et une palette de couleurs très progressive on peut obtenir des images très délicates.

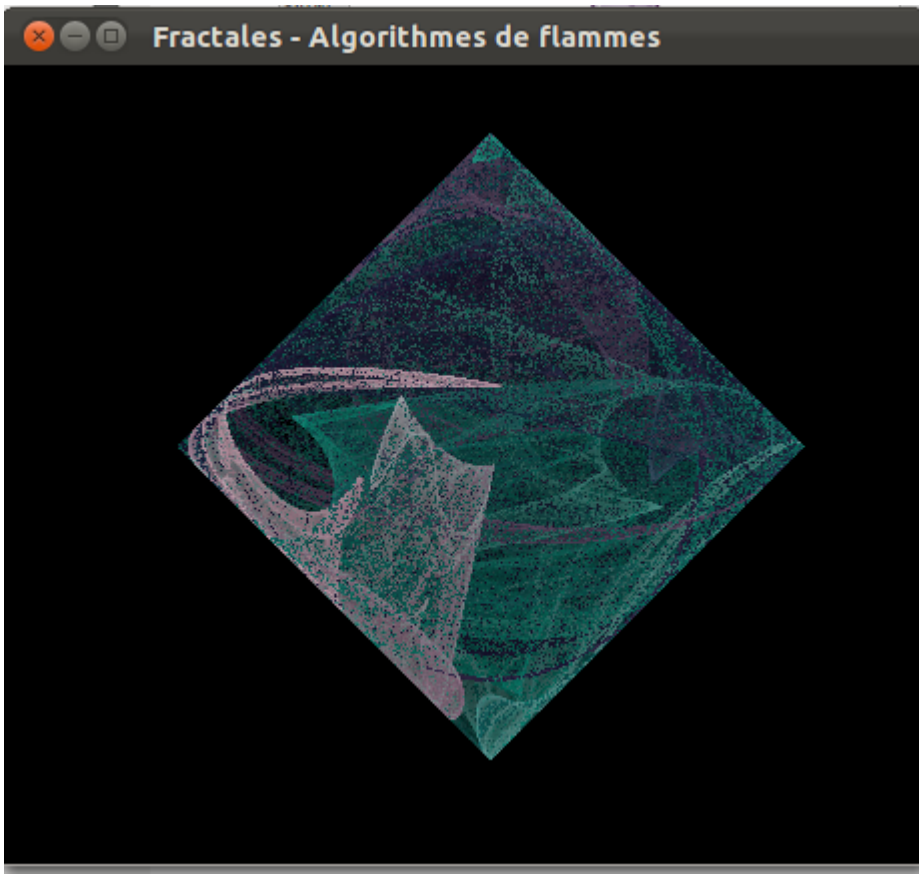


Illustration 3: Flamme (Variation Diamand)

La variation « Diamond » utilise un algorithme appliquant la transformation :

V(x,y) = (sin(theta)*cos(r), cos(theta)*sin(r))

On a défini des macros qui définissent le calcul des variables r et theta utilisées dans beaucoup de variations :

```
#define FLAME_R double r2 = ((*x)*(*x) + (*y)*(*y));\
                double r = sqrt(r2)
#define FLAME_THETA double t = atan((*x)/(*y))

void flame_diamond(double* x, double* y)
{
    FLAME_THETA; FLAME_R;
    FLAME_SYMY(sin(t)*cos(r), cos(t)*sin(r));
}
```

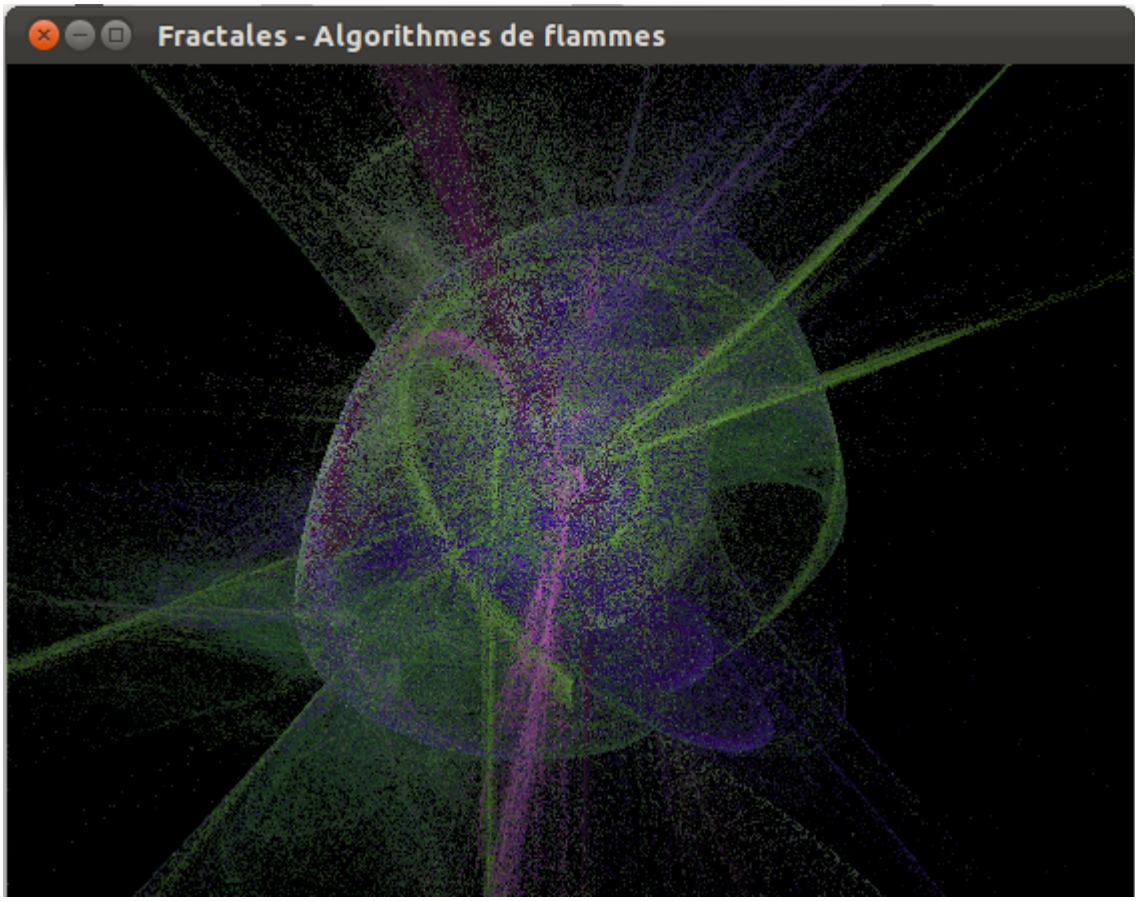
On utilise aussi un macro pour que l'image générée aie une forme de losange.

```
#define FLAME_SYMY(f1, f2) if(*y < 0)\
{\
    *x = (f1);\
    *y = (f2);\
}\
else\
{\
    *x = -(f1);\
    *y = -(f2);\
}
```

Le document PDF fourni en début de projet nous a permis d'implémenter 48 variations différentes.

Voici l'image qu'on a pu obtenir en utilisant plusieurs algorithmes des variations :

Polar (5); Tangent (42); Disc (8); PDJ (24); Bent (14); Linear (0); Disc (8)



Implémentation (pour GNU/Linux)

1) Description de notre application

Le programme en C

Ce programme est exécutable avec comme paramètre l'emplacement d'un fichier de configuration. Le fichier sera chargé par le programme pour afficher une première fractale, en pré-chargeant éventuellement une liste de fractales dans laquelle naviguer. Notre programme se présente par une fenêtre avec un titre et une image. Cette image est initialement une représentation de la première fractale chargée, en utilisant les paramètres de son fichier de configuration. On pourra toutefois faire varier certains de ces paramètres avec le clavier et la souris, ainsi que passer d'une fractale à une autre.

Les fichiers de configuration

Le fichier de configuration décrit comment représenter une fractale, en indiquant plusieurs paramètres. Il peut sinon être constitué d'un ensemble de liens vers d'autres fichiers de configuration, ce qui permet au programme de charger une liste de fractales.

Le générateur de fichier de configuration

Il s'agit d'une page HTML utilisant Javascript pour pouvoir générer un fichier de configuration pour une fractale ou une liste de fractales. Le contenu du fichier à charger avec le programme s'affiche à mesure qu'on avance dans des étapes de la configuration.

Générateur de documentation technique

Pour générer la documentation technique, on a utilisé le logiciel Doxygen qui génère un ensemble de pages HTML et Latex d'après un fichier de configuration et le code source de notre programme.

Le script de test (Bash)

Notre programme n'utilise pas d'interface graphique pour configurer les fractales, ou bien pour les charger. On a donc créé un script qui présente un menu permettant de choisir entre plusieurs options. On peut choisir d'afficher une série de fractales pré-configurée, ou bien ouvrir le générateur de fichier de configuration. On peut aussi ouvrir la version HTML de la documentation technique du programme, ou encore afficher des informations sur comment contrôler le programme. Les fractales pré-configurées sont celles que l'on a configurées pour une démonstration du programme. Ces fichiers sont dans le dossier conf du projet.

Le Makefile

Il contient une fichier doc permettant de générer la documentation technique ; une cible config pour installer le générateur de fichier de configuration dans le dossier *bin* (même si ce n'est pas un fichier binaire, il est exécutable avec un navigateur internet) ; les cibles clean et very-clean demandées pour le nettoyage ; une cible fractal pour la compilation du programme principal.

Finalement, il contient une cible all appelle les cibles fractal, doc, et test .

Cette dernière appelle la cible config et lance le script de test, qui utilisera toutes les fonctionnalités de notre application.

Le fichier README

Ce fichier affiche des informations sur comment compiler le programme, le tester, le configurer, l'exécuter et le contrôler.

Comment compiler :

Il faut la librairie SDL, après il suffit de se placé dans le répertoire 'src' et utiliser la commande `make all`.

`make all` permet de compiler le programme et générer la documentation technique, avant de proposer un test du programme.

`make very-clean` permet de nettoyer les fichiers et dossiers créés par make.

Tester le programme :

`make test` (aussi effectué par `make all`) permet entre autre de lancer le programme avec les fichiers du dossier de configuration conf déjà existants. Ceci exécute un script bash qui ouvre le menu suivant :

1 - Voir comment commander le programme (clavier, souris)
2 - Afficher l'ensemble de Mandelbrot et des ensembles de Julia
3 - Afficher des fractales de type IFS, L-systèmes
4 - Afficher les mêmes fractales en utilisant l'algorithme de flammes
5 - Afficher des fractales de type flamme (48 variations)

6 - Afficher des fractales de type flamme complexes (1 ou plusieurs algorithmes choisis aléatoirement)
7 - Créer un fichier de configuration de fractale(s)
8 - Afficher la documentation technique

---------- COMMANDE DU PROGRAMME ----------

FLÈCHE GAUCHE/DROITE : fractale précédente/suivante
ESPACE, ENTRÉE, F5 : génère de nouvelles valeurs pour les fractales aléatoires (flammes)
ECHAP : quitter

Pour les ensembles de Mandelbrot et de Julia, les cliques gauche et droit permettent de passer de l'ensemble de Mandelbrot à l'ensemble de Julia et inversement.

CLIQUE DROIT/GAUCHE : fractale précédente/suivante (sauf Mandelbrot/Julia)
CLIQUE DU MILIEU : génère de nouvelles valeurs pour les fractales aléatoires (flammes)
ROULETTE : zoom avant/arrière

---------- CONFIGURATION ----------

Charger un fichier de configuration :
./fractal fichier_de_conf

Vous pouvez créer ce fichier avec un éditeur de texte, avec l'aide de la page Configuration.html du répertoire bin. Ce script tentera d'ouvrir cette page avec Firefox (dans un nouvel onglet).

Exécuter le programme :

```
Le programme nécessite d'être exécuté avec comme paramètre
l'emplacement d'un fichier de configuration compatible.

$ ./fractal
Syntaxe : ./fractal fichier_de_config
$ ./fractal ../conf/nom_fichier
```

2) L'ensemble des choix qu'on a fait

Fractales à temps d'échappement

Nous avons choisi d'en représenter deux versions : l'ensemble de Mandelbrot, et les ensembles de Julia. Ces deux types sont les plus connus et se ressemblent beaucoup. La représentation des couleurs se fait par un algorithme improvisé, plutôt qu'en utilisant des gradients.

IFS et L-systèmes

On a remarqué que l'algorithme pour les IFS pouvait aussi être utilisé pour représenter des L-systèmes. On a donc implémenté les deux types en une fois.

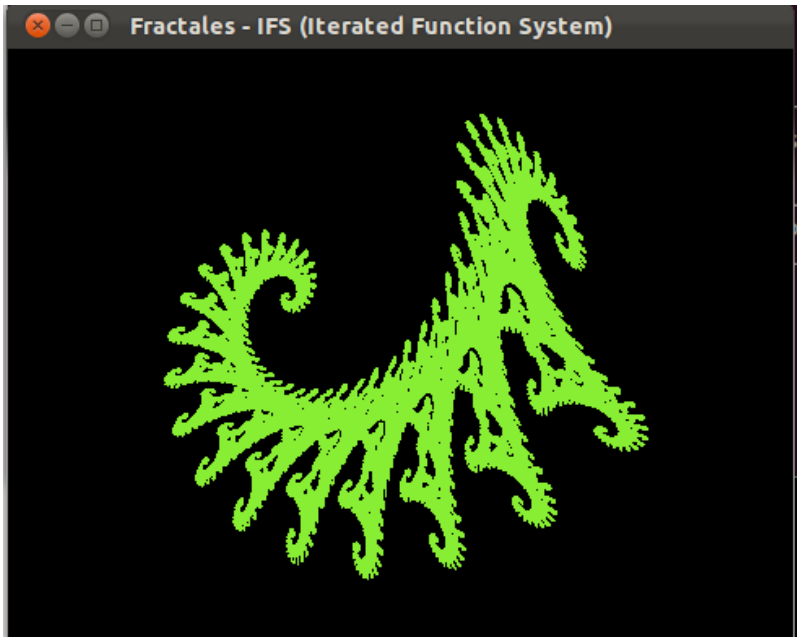


Illustration 4: IFS Dragon

Flammes

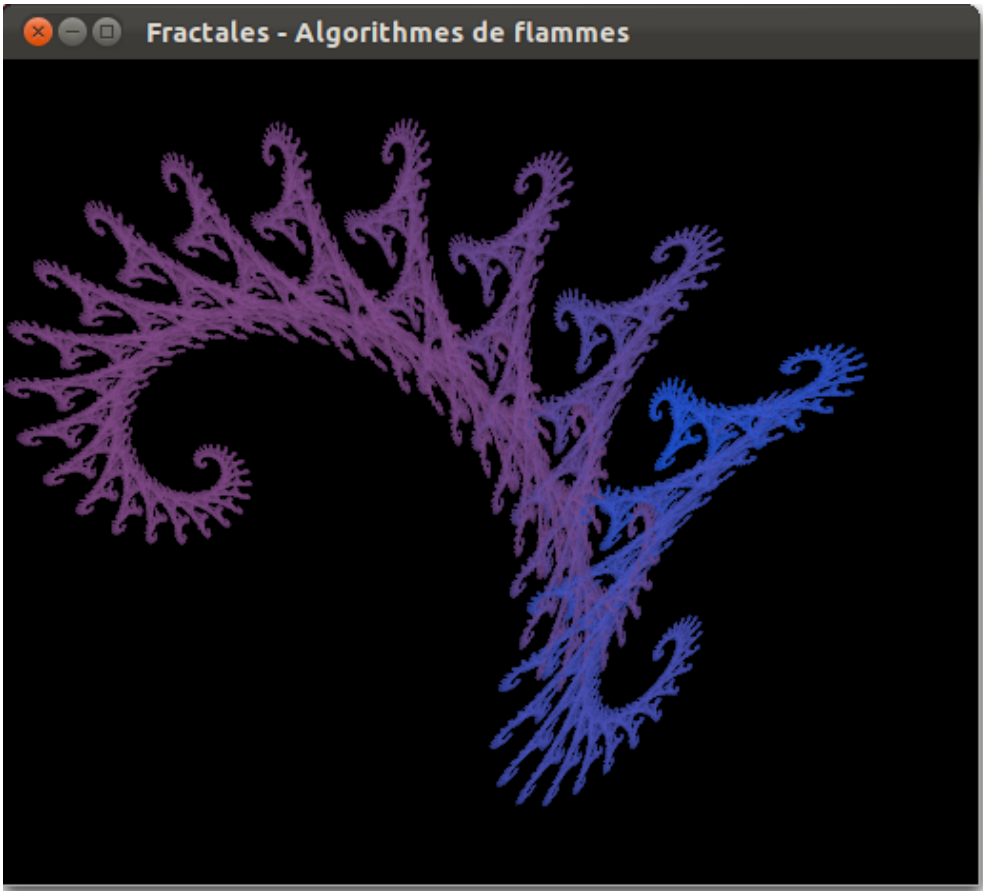


Illustration 5: IFS Dragon (algorithme des flammes : axe Y inversé)

Même si une partie de l'algorithme de flammes est identique à celui des IFS, on a implémenté ce type séparément. Ceci rend presque obsolète l'algorithme des IFS car notre algorithme de flammes utilise des couleurs, la correction gamma, parce qu'une fractale de type flamme peut-être configurée pour ressembler à une IFS ou une L-système. Il suffit pour cela d'utiliser les bons coefficients et la variation linéaire (fonction identité) des flammes.

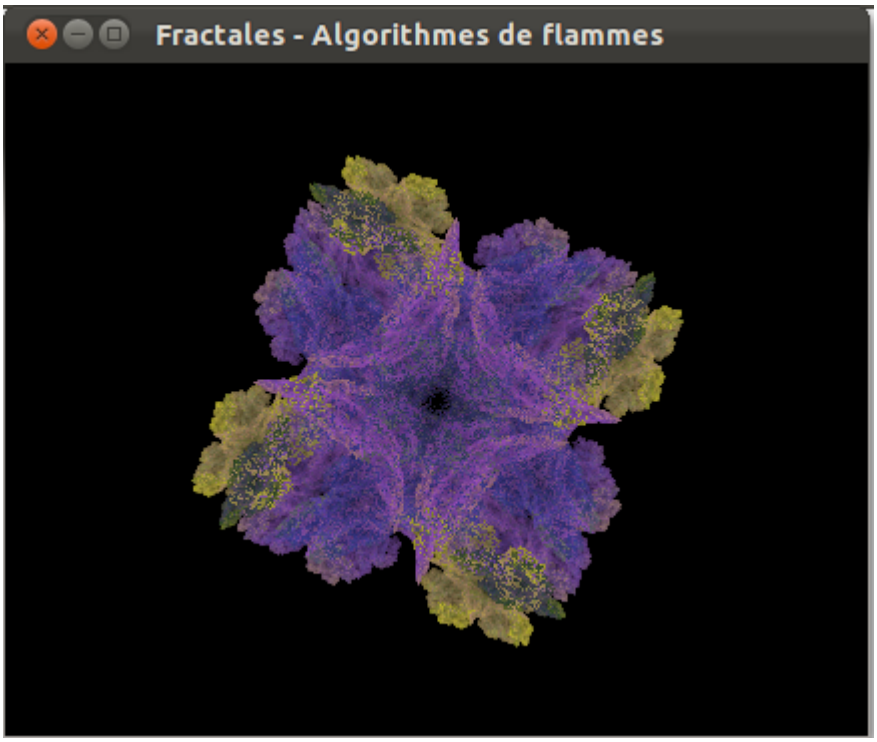


Illustration 6: Flamme : variation Julia

Multi-threading

On a choisi d'utiliser les threads de la bibliothèque SDL pour pouvoir gérer les évènements en même temps que le dessin des fractales. Il y a donc au maximum deux threads qui fonctionnent en même temps (en dehors de ceux gérés par la bibliotèque SDL même), et ils ont des tâches différentes. Ceci nous a permis d'éviter l'attente lors d'un événement comme un clique de souris par exemple. L'évènement du clique peut être reçu et peut demander l'arrêt du thread de dessin des fractales pour le relancer. Ceci permet aussi d'éviter de trop limiter le nombre d'itérations. Cependant un nombre d'itérations trop élevé peut quand même engendrer un problème d'allocation de mémoire dans le cas des flammes, où les points calculés sont mémorisés.

Échelle et déplacement dans une fractale

La roulette de la souris permet de zoomer vers une position, ou de s'en éloigner, ce qui peut aussi permettre de se déplacer. La fractale est générée à chaque fois que l'on veut zoomer, en utilisant les mêmes paramètres. La seule exception est dans les flammes, où les points déjà calculés sont gardés en mémoire.

Optimisations

On a cherché à optimiser le code autant que possible sans perdre trop de temps. Une chose qui peut être un problème est la gestion de la mémoire lors du chargement d'une liste de plusieurs flammes dont le nombre d'itérations est élevé. En effet, avec 8 000 000 d'itérations, une flamme seule peut occuper environ 60 Mo de mémoire vive. On aurait pu choisir d'ordonner au programme « d'oublier » les fractales générées lorsqu'on passe à la suivante, ou du moins leurs couleurs, mais la mémoire de notre PC a supporté la mémorisation de 8 millions de points pour 48 flammes (avec la configuration des 48 variations).

Flammes et variations

La documentation de départ fournie pour les flammes décrit 48 algorithmes différents qu'on a implémenté pour obtenir 48 variations différentes. Certaines sont paramétriques et on a inséré des paramètres en « dur », dans le code source, pour des raisons esthétiques mais principalement pour éviter de trop compliquer la configuration des fractales.

3) Les problèmes qu'on a rencontrés

Une des solutions pas évidentes à implémenter est la gestion de l'échelle, de la position, de la taille de la fenêtre, et de bonne adaptation de tout cela avec les différents types de fractales. Il a fallu utiliser un papier et un crayon pour bien comprendre tout cela jusqu'à-ce que tout fonctionne correctement.

Les difficultés principales dans ce projet étaient la gestion du temps, garder une bonne notion de l'avancement du projet, mais aussi l'implémentation des quatre types dans un même programme. Je pense qu'on aurait pu faire des choses beaucoup plus intéressantes et plus optimisées si on ne s'était occupé que des flammes par exemple.

La gestion des threads était complexe, surtout pour corriger les erreurs. Mais en réécrivant tout, en utilisant des stuctures et des pointeurs de fonction, tout est devenu plus clair. La gestion de l'affichage de l'aperçu de Julia a été dure à synchroniser car on a obtenu beaucoup d'erreurs de segmentation dures à corriger.

4) Le bilan de notre organisation par rapport au planning

Notre notion du temps et celle du niveau d'ambition qu'a ce projet était très mauvaise, surtout au début. Notre planning était donc très mauvais et nous ne

l'avons pas respecté. On a finalement géré le temps de façon intuitive, même si le planning nous a aidé à structurer nos idées par rapport à la liste et l'ordre des choses à faire. En programmation, surtout lors de l'apprentissage, il y a beaucoup d'imprévus qui peuvent faire perdre beaucoup de temps, notamment lorsqu'on commence à s'intéresser réellement au projet et aux possibilités de ce qu'on peut faire. Une bonne stratégie aurait été de faire un planning et de le changer à plusieurs reprises, mais nous avions utilisé Microsoft Project Manager, qui fonctionne seulement sur Windows.