Fractale

Livrable final

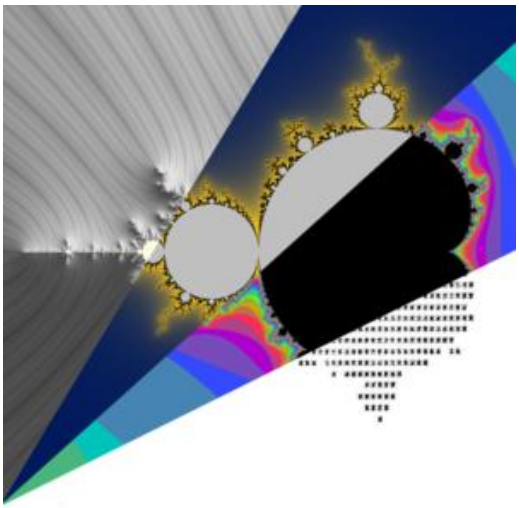
De Bouët du Portal Pauline, Chaulet Florian et Torcheux Fréderic

Inge 1 Groupe C
05/01/2012

Sommaire

Introduction

Nous voici à la fin du projet sur les fractales. Ceci est le dernier livrable vous pourrez donc y trouver un résumé de toutes nos recherches, compositions et créations. Nous mettrons en parallèle les parties théoriques correspondant à chaque type de fractale avec nos propositions d'implémentations. Nous vous ferons part de nos observations sur la totalité du projet, des difficultés rencontrés aux solutions mises en place pour résoudre les problèmes. Nous consacrerons aussi certaines de nos parties à la description de nos dernières optimisations. Nous sommes heureux de pouvoir vous présenter un programme fonctionnel que vous pourrez lancer en ligne de commande sous Linux et dont vous trouverez les instructions nécessaires au lancement dans le fichier README joint au projet. Bonne lecture et bon test.



Généralités sur les fractales

Découverte par Benoit Mandelbrot en 1975, la fractale est avant toute chose une figure géométrique : c'est une courbe ou une surface régulière et/ou morcelée qui se dessine à l'infini, et celle ci suit des règles qui correspondent à une répétition d'un schéma prédéterminé.

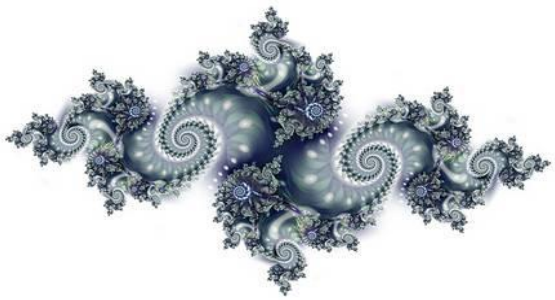
En d'autres termes, «une fractale désigne un objet dont la structure est invariante par changement d'échelle». Si l'on décide de zoomer sur l'image en question, nous retrouverons donc la même structure que celle de l'image initiale.

Il faut savoir que les fractales sont partout, dans l'art, dans les légumes (choux fleurs), dans un flocon de neige, dans le corps humain(les poumons ou plus précisément la structure des bronches puis des bronchioles, bronchiolites etc.)

Ensemble de Julia

Théorie

Les ensembles de Julia, correspondent à un objet fractal défini par un point particulier de l'ensemble, qui est son centre. Celui ci donne les paramètres de la fractale à dessiner. C'est à dire que la fractale n'est définie que par un seul point complexe. Une fonction spécifique $f: z \rightarrow z^2 + c$ est associée à ce point pour lui permettre de dessiner toutes les variations que vous pouvez visualiser dans la figure ci-dessous. (C'est d'ailleurs pour cela que l'ensemble de Mandelbrot est composé de plusieurs ensembles de Julia étant donné qu'il est sur le plan complexe).



Représentation de l'ensemble de Julia

Après cette courte explication sur l'ensemble de Julia, nous allons vous expliquer en détail l'aspect mathématique de ce type de fractale.

Formule précédente :

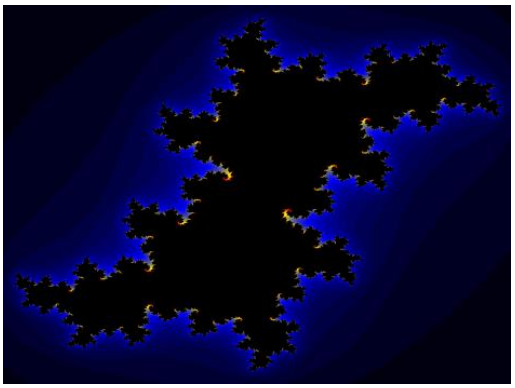
$$f: z \rightarrow z^2 + c$$

D'où une suite :

$$fn : z \rightarrow {z_n}^2 + c$$

La variable z est complexe, on lui attribut une valeur de départ et ensuite on calcule le polynôme correspondant sachant que c'est une constante complexe. Ensuite on réattribue la valeur trouvé à z. On refait ce calcul à l'infini (ou presque, dans notre cas : le nombre de calcul correspond aux nombres d'itérations des points). On remarque que pour certaines valeurs le résultat est à peu prés similaire au fil des itérations, mais pour d'autres, on remarque que le résultat diverge (le point qui est ainsi fabriqué s'échappe vers l'infini). D'où la structure spécial (point rapproché, point éloigné) de l'ensemble de Julia.

Il existe plusieurs ensembles de Julia car la valeur de c peut être changée. (cf. ci-dessous)

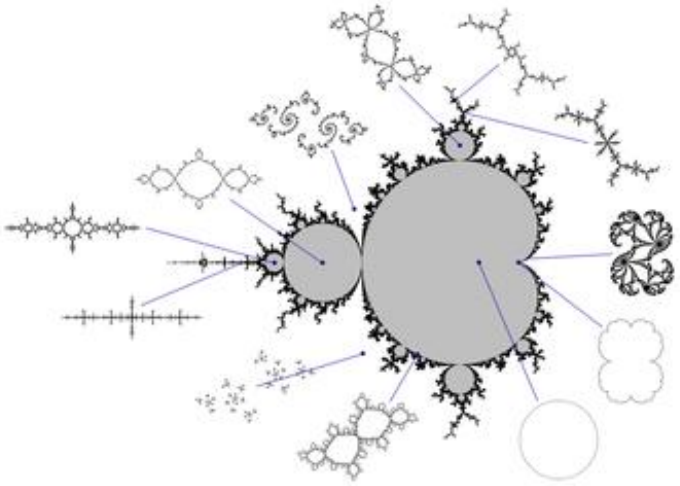


Exemple d'un autre ensemble de Julia

Ensemble de Mandelbrot :

Théorie

L'ensemble de Mandelbrot est un sous ensemble du plan, il est donc défini comme une collection de points, ces point appartiennent au plan complexe et forment des aires, des courbes lisses, des branches… Cet ensemble est composé des différents ensembles de Julia, chaque point de l'image est lié à un système dynamique (un ensemble dynamique est un ensemble qui varie selon un point spécifique (ici complexe)).



Ensemble de Mandelbrot

Maintenant regardons cette ensemble avec un œil scientifique.

Reprenons la formule mathématique de l'ensemble de Julia :

$$fn : z \rightarrow {z_n}^2 + c$$

Pour l'ensemble de Julia, z était une variable du plan complexe et c une constante. L'ensemble de Mandelbrot part sur la même formule précédente, mais la constante c devient variable. Pour chacune des valeurs de c, c correspondant au pixel de l'écran, on itère le polynôme en partant de la valeur initiale 0 pour z, puis on cherche l'ensemble des points pour lesquels le résultat ne diverge pas. Sachant que nous savons déjà que si le module de Zn est supérieur à 2 alors le résultat va diverger. Le point n'appartiendra donc pas à l'ensemble de Mandelbrot.

Le zoom des fractales de types Mandelbrot et Julia :

Pour Mandelbrot et Julia, on opère de la même manière, ces fractales sont dessinées en partant du coin supérieur gauche (contrairement aux flammes et IFS qui sont dessinées à partir du centre de l'interface graphique). En effet pour dessiner les fractales de Mandelbrot, l'algorithme réalise une double boucle qui calcule la valeur d'un nombre complexe pour tous les pixels qu'on lui donne. Ceci implique que notre image est toujours insérée dans un rectangle. On commence donc à la dessiner par le bord supérieur gauche. Par conséquent, on zoom vers le point en haut à gauche. Nous avons essayé diverses méthodes pour placer le zoom au centre mais la fractale ne correspondait plus à celle initiale, ou sinon nous avons essayé de ne plus dessiner l'image après l'incrémentation de la variable mais l'écart entre les pixels devenait trop important et l'image disparaissait au fur et à mesure que l'on zoomait. Nous sommes donc resté sur la première solution qui, combiné à notre fonction de déplacement (par les flèches), permet maintenant de visualiser notre fractale dans son ensemble et à différentes échelles de zoom. Du fait que nous calculons la valeur du nombre complexe uniquement pour les coordonnées que l'on voit à l'écran, nous accélérons la vitesse de notre algorithme.

Nota bene :

-Nous avons voulu créer une fonction zoom unique, à l'aide des pointeurs de fonctions, qui auraient permis de changer le nom de la fonction à zoomer à chaque appel. Mais après d'infructueux essais et de nombreuses recherches, nous avons préféré nous reporter sur un simple ajout dans les fonctions de construction des fractales.

Les L - Systems :

Théorie

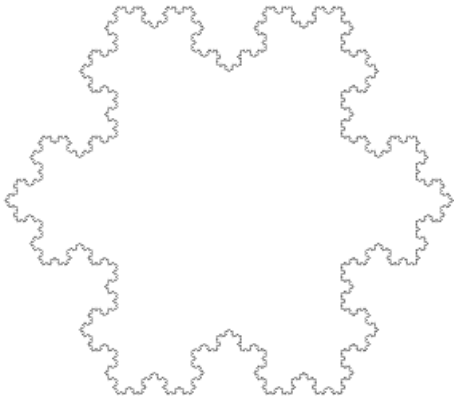
Un L-système est un type particulier de fractale. L'algorithme ne se base pas sur des calculs mathématiques afin de calculer la fractale. A la place, il utilise une grammaire qui est tout simplement un ensemble de mots. Mais Il faut savoir que l'on ne peut pas mettre n'importe quel mot dans cette grammaire. On doit respecter une certaine syntaxe, syntaxe qui est similaire pour tous les L-système.

Les L-système sont basés sur un algorithme récursif, ou de réécriture. Le principe consiste à prendre un morceau du code afin de le remplacer par d'autres éléments. Plus on fait d'itération, plus la fractale est grande ou bien détaillé, plus elle se complexifie, selon la grammaire choisie.

A la base, ce genre de fractale a été développé pour modéliser la croissance d'être vivant tel des plantes ou des champignons. Mais on peut très bien dessiner des fractales tel que le flocon de koch

w : F--F--F

F ---> F+F--F+F



Expliquons maintenant ce que signifie les signes à côté de la fractale.

Grammaire :

1) Un alphabet, c'est-à-dire un ensemble de variables.

2) Un ensemble de constantes. Celles-ci ont la particularité d'être commune à tous les L-system.

On peut par exemple citer les variables suivantes :

- 'F' : avancer sur une certaine longueur.
- '+' : tourner à gauche d'un certain angle.
- '-' : tourner à droite d'un certain angle.

- '[' : sauvegarder la position courante.

- ‘]‘ : restaurer la dernière position sauvegarder.

3) Un point de départ, c'est-à-dire un mot sur lequel on se basera pour commencer et auquel on appliquera des transformations en fonction des règles utilisées.

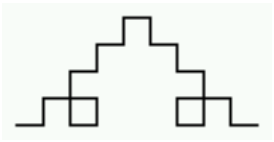
4) Ensemble de règles : défini à quels endroits et quelles modifications on doit apporter à notre axiome (déjà modifié ou non).

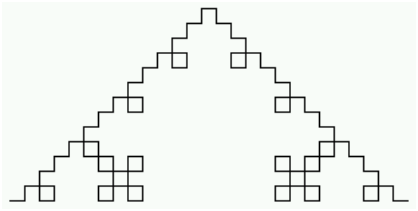
Au final, le mot que l’on crée grâce à tous ces éléments change en fonction du nombre d’itérations que l’on effectue. C’est à partir de lui que l’on peut tracer notre L-system. C’est lui qui définie la forme qu’il aura.

Exemple de L-system, Courbe de Koch :

Axiome, point de départ, nombre d’itération = 0 :

Après une itération : 

Après 2 itérations : 

Après 3 itérations : 

Implémentation :

Afin de créer cette fractale, nous implémentons donc les 4 points abordés dans la théorie. On définie un mot de départ que l’on modifiera en fonction des différentes règles.

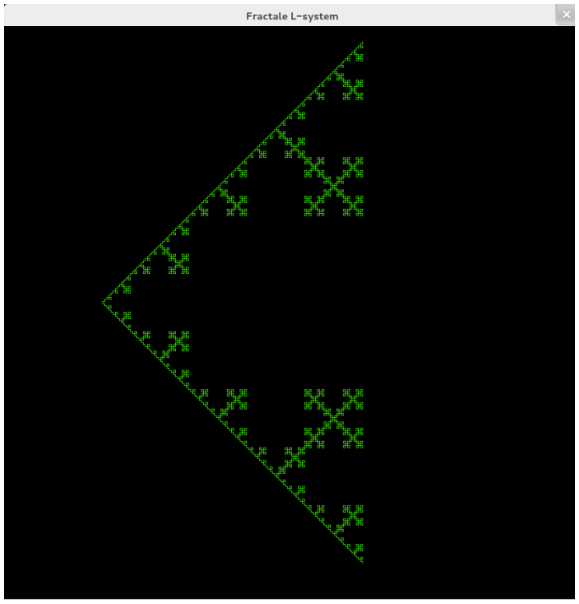
On peut diviser notre algorithme en 2. La première partie qui se charge uniquement de définir le mot, et qui donc, a force d’itération, arrivera à un mot plus ou moins complexe. Ensuite vient la partie consacrée au tracé de la fractale.

Dans notre algorithme nous avons utilisés la même syntaxe que la majorité des algorithmes des L-système. C'est-à-dire : F pour avancer, + pour tourner à gauche et – pour tourner à droite.

Dans la première moitié du programme, on réalise donc une boucle contenant différentes conditions qui vont regarder chaque caractère de notre mot, et en fonction de ce caractère,

vont réaliser une action. Par exemple la plus part des algorithmes disent de remplacer le F par un chaine de caractère plus complexe. Cette boucle contenant les conditions est elle-même placée au sein d'une autre boucle qui permet de définir le nombre d'itérations.

Pour la 2ème moitié, on va réaliser une boucle en fonction du nombre de caractères qu'il y a dans le mot qui définit notre fractale. Ceci va nous permettre de parcourir notre mot et donc de regarder à quoi correspond chaque caractère afin de réaliser l'action qu'il définit. Le 'F' va nous faire tracer une droite en fonction d'un certain angle. Le '+' et le '–' vont modifier la variable nommé angle. La variable est modifié en fonction d'un angle qui est propre à chaque fractale.



Notre courbe de Koch

Les IFS :

Théorie

Les Systèmes de fonctions itérés ou plus simplement, qui vient de l'anglais Iterated function System, ont étés crées par John Hutchinson en 1981. Elles sont définies par un ensemble de transformations géométriques élémentaires comme par exemple le calcul des transformations appliquées à chaque point par homothétie. Ces transformations sont des combinaisons de rotations, de translation et de diminution d'une forme et d'itération. Mais pour exprimer cela d'un point de vue mathématique, les IFS définissent un ensemble fini de n fonctions affines strictement contractantes dans un espace métrique M.

Une fonction contractante se définie de la manière suivante : elle est k-lipchitzienne avec 0<k<1. La définition est la suivante :

$$\forall\ (x, y) \in R^2, |f(x) - f(y)\ | \leq k|x - y|\ \ avec\ 0 < k < 1$$

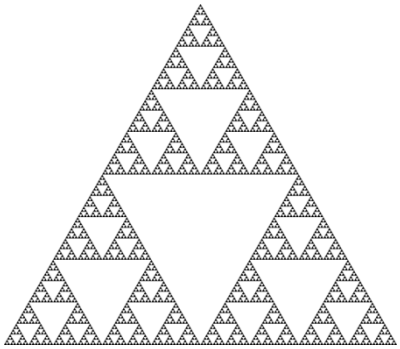
Les fonctions doivent être contractantes car elles doivent avoir un point fixe or le théorème du point fixe nous explique que si f est une fonction contractante alors f(x)=x. On nomme ce point de fuite un point attracteur. En résumé toutes les fonctions admettent un point vers lequel tendent toutes les valeurs de la fonction.

Soit d la distance, a et b deux points et f une fonction contractante. Dans notre sujet f correspondrait à la fractale.

$$d\ (a, b) > d\ (f(a), f(b)\ )$$

Exemple : Le triangle de Sierpinski

Le triangle de Sierpinski est l'exemple le plus simple à donner car son explication ne nécessite pas de connaissances poussées en mathématiques. On part d'un triangle équilatéral peint en noir, on trace à l'intérieur trois segments. Le premier segment lit deux cotés des triangles, il part d'un milieu d'un des deux segments et rejoins le milieu de l'autre segment. Les deux autres segments sont tracés de la même manière. Ainsi sont formés 4 nouveaux triangles. On supprime le triangle central, par conséquent il ne reste plus que 3 triangles. Puis on applique ce procédé sur chacun des triangles restants. Si l'on veut revenir à la définition mathématique, il y a bien un point de fuite au centre, la fonction qui est itéré n fois est donc contractante.

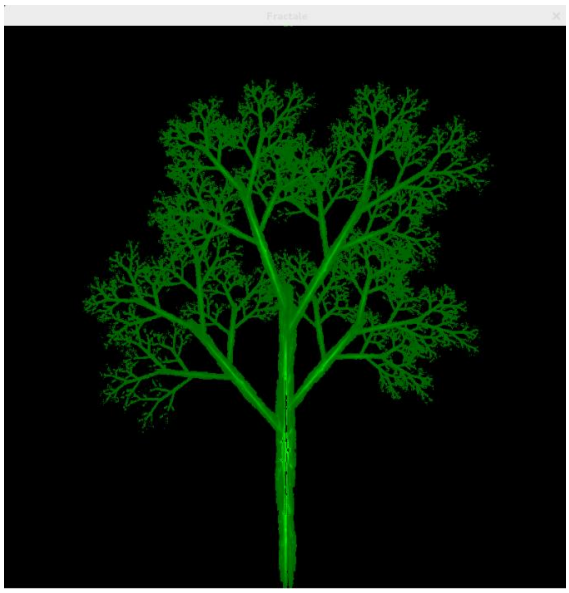


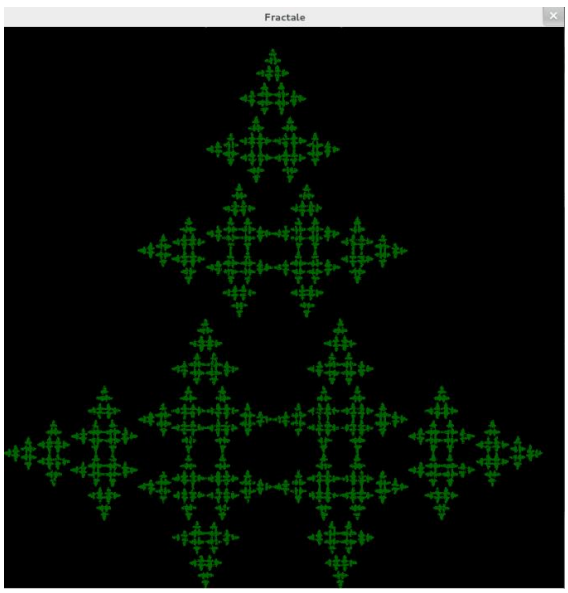
Le triangle de Sierpinski

Implémentation de notre algorithme

Le principe des IFS est simple. On définit plusieurs fonctions linéaires ainsi qu'un point de départ. On utilise aussi une variable aléatoire qui va nous permettre, selon sa valeur, de choisir quelle fonction on utilise afin de réaliser le prochain mouvement de notre point. A chaque fonction, on associe un poids. Plus ce poids est élevé et plus la fonction aura de chance d'être choisie.

Pour dessiner un grand nombre de points, on réalise une boucle 1.000.000 de fois. Ainsi la fractale ne prend pas trop de temps pour se dessiner et le dessin est identique à la même fractale que si l'on avait bouclé 10 ou 100 fois plus. Car en bouclant un grand nombre de fois, les points dessinés finissent par tomber à des endroits où le point est déjà passé. Du coup cela nécessite du temps de calcul sans modifier l'aspect de la fractale.



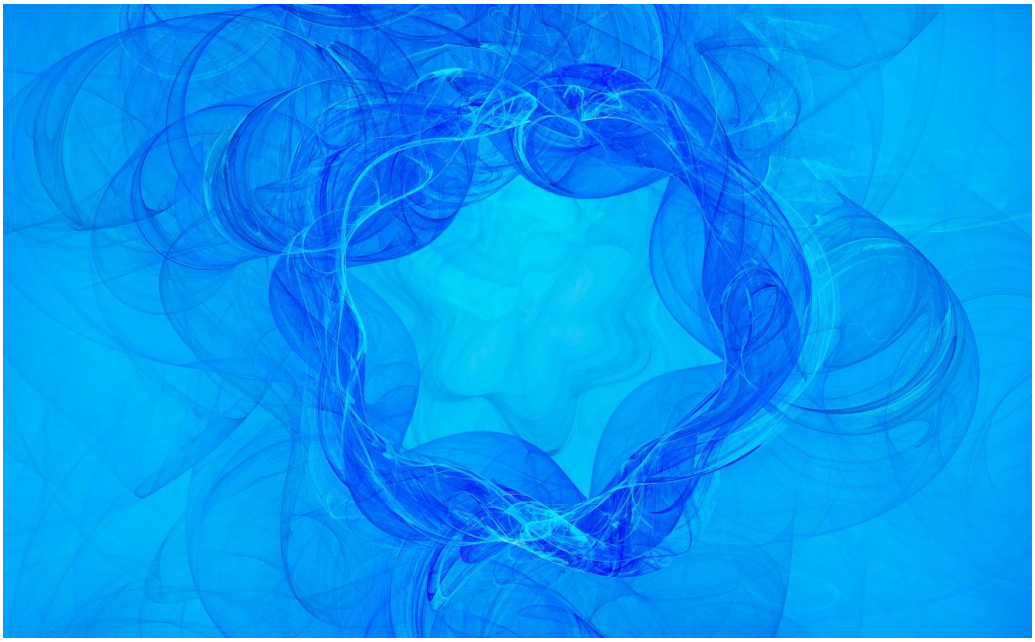


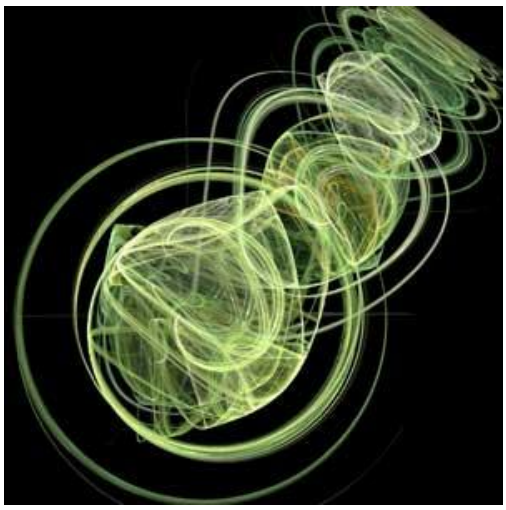
Les flammes

Nous nous sommes confrontés à l'implémentation de ces fractales mais dans un premier temps rappelons quelques notions théoriques sur ces dernières.

Théorie

Tout d'abord nous pouvons dire que l'inventeur de ce type de Fractale n'est autre que Scott Draves, ayant également travaillé sur les fractales de type IFS. Si les IFS sont mentionnés ici ce n'est pas par hasard, en effet on peut considérer que les flammes sont des types d'IFS un peu particuliers: Voici quelques représentations.





Il n'est nul besoin de détailler le « pourquoi » du nom flamme que l'on donne à ce type de fractale. Comme leurs ainés les IFS il s'agit d'une suite de points dont on calcule les coordonnées à partir de différentes fonctions et ce à un instant t.

Les fonctions itérés : linéarité, non linéarité

Les itérations successives de ces fonctions dispersent les points dans la fenêtre d'affichage et génère un rendu tel que celui que vous avez pu voir sur ses images. Les premières fonctions qui « transforment » les coordonnées des points à tracer sont affines (ce qui rappelle la structure des IFS, il s'agit de la transformation initiale.
Nous entrons ensuite dans une boucle de calcul des coordonnées des points à afficher, avec des fonctions non linéaires qui doivent être « contractantes » (↔ k-lipchitzienne avec $0<k<1$) (cf. définitions et explications dans la partie IFS).

Les différentes fonctions qui sont implémentées sont « choisies » avec un poids différent, c'est-à-dire que l'on peut en favoriser certaines (pour plus de précisions consulter la partie descriptive de l'algorithme implémenté).
Il est à noter que les calculs sont effectués mais que durant les 20 premières itérations on ne dessine rien sur la fenêtre. Ensuite, chaque coordonnée correspond à un pixel à dessiner et ce jusqu'à la fin de la boucle. Les différentes théories peuvent aussi inclure ou non une transformation finale (qui ne se situe pas dans les itérations) et qui serait de type linéaire.
Pour résumé, on peut donc dire que les flammes sont des IFS avec des transformations non linéaires qui interviennent en plus.

L'affichage à densité logarithmique

Comme nous l'avons montré l'affichage se fait dans un premier temps en dessinant point par point la figure grâce aux transformations. Cependant selon les fonctions utilisées les points peuvent être très proches les uns des autres voire même superposés. C'est pour cette raison que l'affichage doit être mené d'une manière différente en faisant apparaître ces différences. Lorsque les points sont trop proches sur la fenêtre le rendu de l'image est bruité, les contours sont flous. Des procédés peuvent occultés ces nuisances, c'est le cas de l'affichage à densité logarithmique qui consiste à améliorer la netteté de l'image. Lorsqu'un pixel est dessiné « sur » un autre, on change son opacité de façon à l'assombrir et à marquer un peu plus sa présence dans la fenêtre. On peut dire d'une certaine façon qu'on améliore le contour de la figure.

Implémentation de notre algorithme

En se basant sur la théorie développée ci-dessus ainsi que des remarques formulées par nos camarades de deuxième année nous avons développé notre algorithme pour créer des flammes fractales. Nous décrirons ci-dessous l'algorithme utilisé pour la flamme 'Heart' mais les autres flammes fonctionnent sur le même principe avec un nombre de points calculés et des fonctions différentes. Nous avons modifié notre algorithme initiale pour y ajouter des optimisations au niveau de l'affichage et réaliser un effet d'anticrénelage (ou anti aliasing).

Nous avons décidé de choisir un système dont la hiérarchie est la suivante :

- 1 pixel (celui qui est dessiné (on dit aussi « hit » en anglais), composé de
- 2 coordonnées regroupées dans la variable position (on fait appel à position.x et position.y)

Le calcul des coordonnées (première boucle de calcul) s'appuient donc sur les variables de l'alphabet. Le poids est géré par un nombre aléatoire qui détermine une probabilité plus ou moins importante dans le choix d'une fonction à chaque itération. Exemple de fonctions utilisées :

Soit $r = \sqrt{(x^2 + y^2)}$ et $\theta = Arctan(x/y)$

Heart: $f(x,y) = r(\sin(\theta r), -\cos(\theta r))$

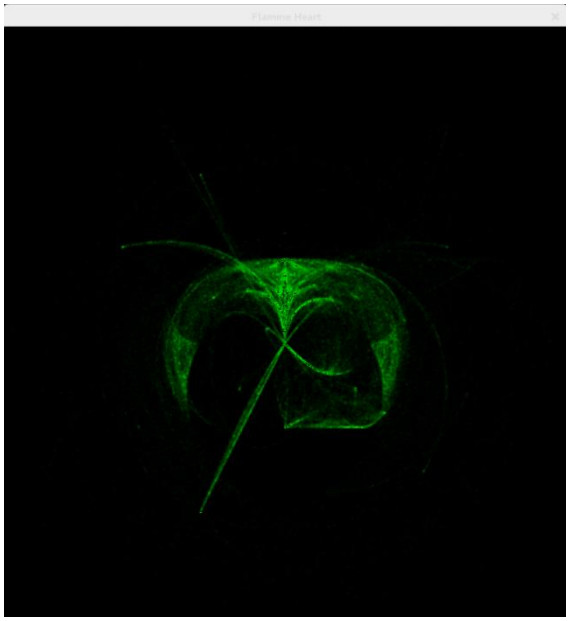
Handkerchief: $f(x,y) = r.sin(\theta + r), \cos(\theta - r)$

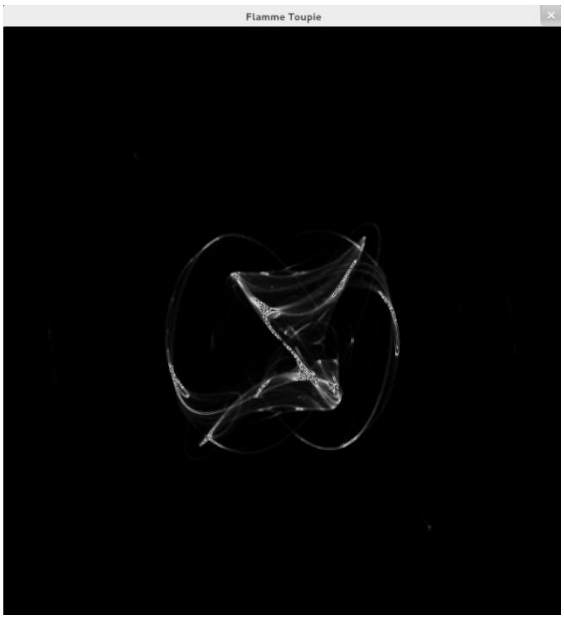
Certaines autres fonctions sont utilisées et proviennent du PDF sur les flammes (fournis avec l'énoncé du sujet dur les fractales).

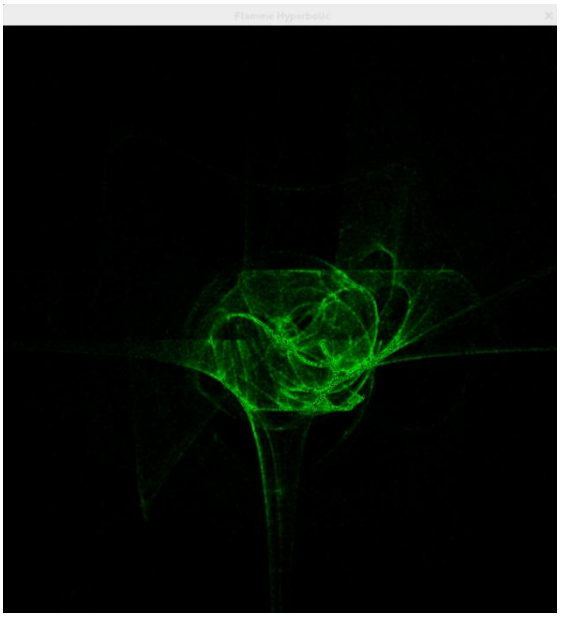
Une fois la boucle de calcul effectuée, on se sert d'un tableau à double dimension (longueur et largeur de la taille de la fenêtre SDL). Dans ce tableau on considère que les indices des cases sont les pixels de l'écran et que la valeur à l'endroit donné correspond au nombre de fois qu'un pixel est posé. Exemple: aux coordonnées [100,100] si le pixel calculé est positionné 10 fois nous aurons tab [100,100] = 10.

Dans une deuxième double boucle, on parcourt le tableau ainsi créé et en fonction du nombre de fois qu'un pixel est censé se trouver au même endroit on modifie sa couleur.

On dessine le pixel sur la fenêtre excepté durant les 20 premières itérations. En jouant sur les fonctions utilisées et les poids associés à celles-ci on obtient des résultats comme les suivants. On reconnait quelques tracés comme le 'Heart' (figure1) et le 'hyperbolic' (figure 3).


Flamme Heart


Flamme Toupie

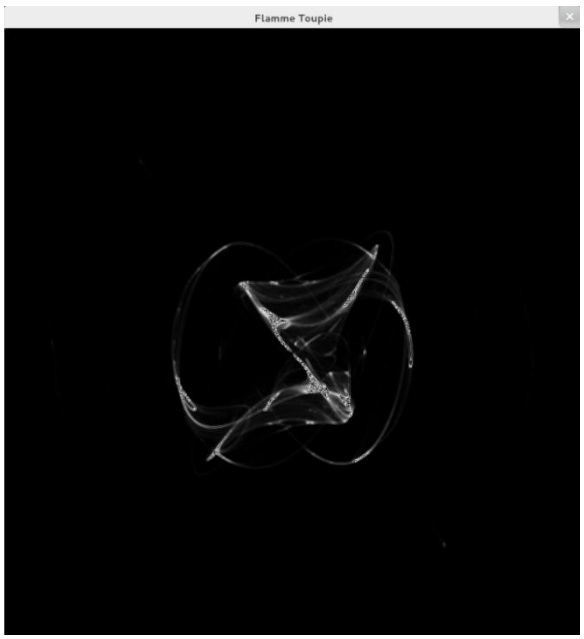

Flamme Hyperbolic

Le zoom des fractales du type flamme, L-system et Ifs

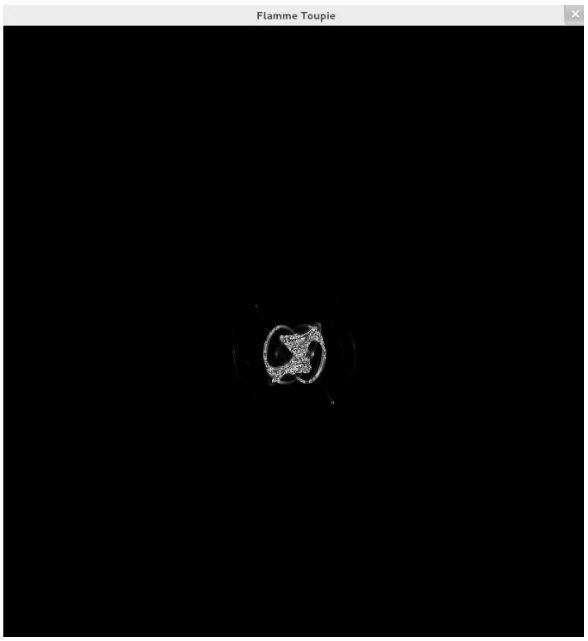
Pour ces types de fractales, on utilise la variable zoom qui est déjà présente. On la place en tant que paramètre dans la fonction, ensuite grâce à la bibliothèque SDL on introduit l'événement: SDL_Event event qui signifie que l'on créé un événement qui permet d'interagir avec la fenêtre SDL. Ensuite à l'aide d'une condition on vérifie l'événement molette de la souris, c'est-à-dire actionne-t-on la molette ou non ? Si celle-ci « monte » (scrolling-up) on ajoute 100 à la variable zoom puis on rappelle la fonction. Ce qui la redessine avec la nouvelle valeur du zoom. On utilise ainsi le principe de récursivité. Pour éviter tous problème, de type segmentation fault, nous avons fait en sorte qu'au moment de l'appel récursif, la fonction appelante soit « arrêtée » afin qu'une seule fonction dessinant notre fractale soit active au même instant.

Exemple : Flamme Toupie

Avant :



Après Zoom Arrière :



Après zoom avant :

Après déplacement :

La gestion du clavier

Pour la gestion du clavier, il nous a fallu créer deux variables. Abs et ord qui correspondent pour l'une à un déplacement sur l'axe des abscisses et pour l'autre à un déplacement sur l'axe des ordonnées. Ces variables n'existant pas dans les créations de nos fractales, nous les avons introduits dans les fonctions, au moment du positionnement des pixels. On ajoute aux variables de position : abs ou ord selon la spécification de la variable position (abscisse ou ordonnée). Au début, c'est-à-dire lors du lancement du menu dans l'invite de commande : les variables x et y sont initialisées à zéro. Puis on incrémente de 100 selon l'événement clavier. Et bien entendu ces variables, sont aussi passées en paramètre dans les fonctions. On peut donc déplacer notre fractale comme bon nous semble dans l'enceinte de la fenêtre.

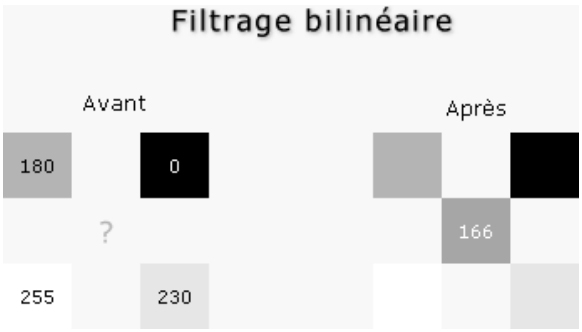
Cette partie n'a pas posé de problème pour les fractales de type IFS, Flamme et L-system car l'algorithme calcule tous les points de l'espace, et non pas uniquement un petit périmètre comme peut le faire l'algorithme des fractales de Mandelbrot. Au début de notre développement, la fonction déplaçait notre image mais continuait de calculer les mêmes points. Par conséquent, on se retrouvait à voir des pixels qui n'avaient pas de couleur, étant donnés que l'on n'avait pas appliqué l'algorithme en ces points. Nous avons donc réfléchi à une manière de corriger cela. Après réflexion, nous avons ajouté le déplacement, c'est-à-dire la modification des variables abs et ord a notre double boucle, ainsi il nous est possible de « suivre » notre vue de la fractale, ou plutôt de calculer tous les points, et uniquement (ainsi on ne ralentit pas l'algorithme) les points que l'on peut voir. De cette manière, la gestion du déplacement est parfaitement gérée, et ce même avec Mandelbrot.

Fichier de configuration

Nous avons laissé un certain pouvoir à l'utilisateur de notre programme. En effet, se contenter de lui afficher une vingtaine de fractals fixes aurait été d'un intérêt relatif. Nous vous avons déjà parlé du fait que nous avons ajouté des interactions à l'aide du clavier et de la souris, mais ceci ne suffisait pas. Nous voulions ajouter d'autres paramètres modifiables. Par conséquent, le choix d'un fichier de configuration s'est imposé de lui-même. Ce fameux fichier de configuration consiste en un fichier texte que l'utilisateur peut ouvrir à sa guise, pour modifier les paramètres qui se trouvent à l'intérieur. Après réflexions, nous avons trouvé 4 paramètres important à y ajouter. La taille de la fenêtre sur l'axe x, la taille sur l'axe y. Mais aussi quel zoom initial applique-ton, au lancement de la fenêtre à notre fractale (sachant que l'on peut toujours modifier le zoom avec la molette de la souris). Enfin un dernier paramètre qui est la qualité de la fractale. Cette qualité fait référence au nombre d'itérations de nos algorithme, pour le cas des fractales à temps d'échappements, des IFS et des flammes. Ce nombre a un lien avec le nombre de points qui apparaitront à l'image et donc ce nombre influe sur la qualité du rendu de la fractale. Pour les fractales L-system, cette dernière valeur correspond au nombre de génération de notre fractale. Plus ce nombre est grand et plus la fractale est découpé en une multitude de traits (pour les l-system on ne parle pas de point car ce sont des traits que l'on dessine).

Fonction d'anti aliasing

Pour améliorer la qualité de l'image, nous avons implémenté l'anti aliasing où plutôt le filtrage bilinéaire. Nous avons réalisé ceci pour les fractales IFS et Flamme. Cette méthode consiste à lisser une image ou plutôt éviter les phénomènes de crénelages. En réalisant la moyenne des couleurs entourant un point, on peut appliquer le résultat à ce point pour éviter d'avoir un trop fort contraste si les couleurs entourant ce point sont trop différentes.



Normalement la fonction bilinéaire est plus difficile à réaliser que simplement calculer la moyenne des points entourant le point dont on veut savoir la couleur. Seulement ici le point dont on cherche la couleur est toujours à mi-distance des points dont on se sert pour

calculer la couleur, du coup ça ne pose pas de problème. Au final grâce à cette méthode, les points isolés n'apparaissent pas et l'image est bien plus nette que sans cette fonction.

Commentaires Personnelles

Commentaire de Pauline de Bouët du Portal

Lorsque j'ai vu que notre projet concernait les fractales, cela m'a rappelé mon enfance quand ma mère me parlait du choux romanesco, cela m'a donc fais plaisir dés le départ de travailler sur ce projet. Au début j'étais tout de même un peu paniquée car mon niveau en langage C était assez faible, il m'a donc fallu revoir le langage C. Durant le projet, ce qui m'a le plus dérangé c'est la compréhension des algorithmes des fractales, les algorithmes étaient trop abstrait et devoir les transcrire en ligne de code fut assez compliqué. Mais au fur et à mesure j'ai réussi à réaliser des fractales. Sinon, j'estime avoir bien progressée en langage C, face à plusieurs problèmes rencontrés j'ai su me remettre en cause et évolué. Je me souviens lors de la réalisation du zoom avoir de nombreuses fois dis '' c'est de la faute du pc'' mais au final je me rendais souvent compte que j'oubliais parfois l'essentiel, ce projet m'a donc aussi permis de revenir à l'essentiel (par exemple : le problème ne vient-il pas de la création de variable … des choses basiques mais qui parfois sont la cause d'une multitude d'ennuies).

Au sujet du groupe : la répartition fut facile à réaliser car nous étions amis et nous connaissions nos méthodes de travail. Tout le monde a donc travaillé et lorsque quelqu'un rencontrait un problème, les autres membres venaient le soutenir. Nous avons donc su travailler en groupe tant d'un point de vue travail que d'un point de vue communication, ce qui me parait important lors d'un projet. Il ne faut jamais oublier l'aspect relationnel (communication) car souvent si le projet se passe mal, c'est qu'il n'y a pas de communication entre les interlocuteurs. Je suis donc très contente sur tout les points de vues (travail et "équipe") d'avoir réalisé ce projet.

Commentaire de Frédéric Torcheux

Au début, ce projet m'a fait un peu peur car je n'avais pas la moindre idée de comment un fractale pouvait être représenté mathématiquement. Lors des première recherche, j'ai découvert qu'il fallait utiliser des nombres complexes, et ceci avec un langage de programmation. Je me suis demandé dans quoi on s'embarquait. Mais après plus de recherche, j'ai commencé à comprendre comment les différents algorithmes marchaient, que l'on pouvait facilement trouver un moyen de travailler avec des complexes et toutes mes craintes on disparut pour laisser place à une grande envie de développer toutes ces fractales. La partie gestion de projet avec différents livrables a été très intéressantes car nous avons dû réfléchir à comment nous allions découper les différentes tâches, de plus

grâce aux livrables, nous n'avons pas travaillé le projet à la dernière seconde, mais bien de manière continue.

Grâce à ce projet, j'ai retrouvé la joie de voir des segmentation fault, et donc la joie de retrouver les chers pointeurs. Mais finalement, j'ai réappris à m'en servir, et à la fin du projet, il n'y avait plus aucun problème. Sinon, après 4 mois de travail sur ce projet, j'ai pu revoir tout ce que l'on avait appris en CPI1 et continuer mon apprentissage du c, ainsi que de la librairie SDL. Du coup ce projet a été très intéressant et enrichissant et je suis bien content de l'avoir suivi, en espérant l'avoir bien réalisé.

Commentaires de Florian Chaulet

Ce projet fut au début une surprise pour moi, en effet je n'avais jamais entendu parler des fractales. Lors de nos premières recherches j'ai trouvé ce sujet pour le moins intrigant et intéressant. La vue d'un reportage sur les applications possibles de cette géométrie a fini d'attiser ma curiosité. Je me suis donc lancer dans le projet avec l'envie de bien faire. Connaissant déjà mes camarades (ex CPI) et leur façon de travailler il n'a pas été trop dur de se diviser les taches. Chacun a essayé de mettre en avant ses qualités personnelles afin d'apporter sa pierre à l'édifice. Bien que notre prévision des dates de rendus intermédiaires n'ait pas été très réussie cela nous a prouvé l'importance de la bonne organisation d'un projet et la difficulté qu'elle représente. Je pense avoir progressé tant sur le plan technique (consolidation des connaissances en C) que sur le plan de la gestion de projet et ce projet m'as permis de murir et de prendre toujours un peu plus d'expérience. J'espère pouvoir continuer à travailler de la sorte (méthode des projets) au cours de mes années d'études à l'EISTI pour progresser et toujours m'améliorer.

Conclusion

Nous vous avons donc présenté les fractales, tels que nous les avons comprises (cf. parties théoriques), avec nos propositions d'implémentations, nos optimisations et ajouts. Bien entendu notre programme n'est qu'un échantillon de toutes les possibilités qu'offrent la géométrie fractale mais nous espérons qu'il est assez complet et approfondi pour satisfaire vos attentes. Nous avons pris plaisir à son élaboration et à sa réalisation. Le travail d'équipe nous permet de progresser et la gestion de projet fait partie intégrante de nos futurs métiers, ce travail était donc un excellent défi en ce sens et nous espérons l'avoir relevé avec succès.