Rapport de projet : Lyapunov

1 Fonctionnement générale

1.1 Gérer la SDL

Pour afficher la fractale, il a fallu travailler avec la SDL, qui est une librairie écrite en C, et donc l'adapter en C++ avec un wrapper, qui est la classe Window Manager. Le but de cette classe est ainsi de pouvoir travailler avec la SDL en évitant les pointeurs, et permettre un découplage entre la SDL et notre programme. Cette classe utilise une texture pour modifier les pixels que l'on veut afficher, et un render (qui est un élément de la SDL) pour afficher la texture à l'écran. On peut ainsi afficher la fractale facilement sans avoir à se préoccuper du fonctionnement au niveau de la SDL. Le programme fonctionne avec une boucle d'évènement, et pour chaque évènement qui nous intéresse, on peut le traiter avec une fonction évènementielle en dehors de la classes'occupant de la SDL.

1.2 Génération et affichage

Les fractales de Lyapunov sont une certaine représentation d'une suite chaotique calculée à partir des exposants de Lyapunov. L'algorithme de génération des fractales de Lyapunov en 2D dans le plan $[0, 4] \times [0, 4]$ est séparée en plusieurs étapes. Tout d'abord, on choisit une séquence de A et B. On construit ensuite une séquence Sn suffisament longue constitué d'une répétiton de la séquence de A et B. On choisit un point de coordonées x,y dans le plan de depart. On définit ensuite la fonction $R_n = a$ si $S_n = A$ et $R_n = b$ sinon. On construit ensuite une suite x_n notée $x_{n+1} = r_n x_n(1 - x_n)$ en prenant comme origine $x_0 = 0.5$. On calcule ensuite l'exposant de Lyapunov noté λ grâce à cette formule :

$$\lambda = \lim_{N \to +\infty} \frac{1}{N} \sum_{n=1}^{N} \log |r_n(1 - 2x_n)|$$

Ensuite, pour afficher les fractales à l'écran, on attribue une couleur à chaque exposant afin de pouvoir changer les couleurs à la volée, qui sont représentées en RGB. Nous avons essayé beaucoup de couleurs différentes, ainsi qu'une représentation arc-en-ciel en mouvement.

1.3 Optimisation et Multi-Threading

L'opération la plus coûteuse dans le calcul de l'exposant est le logarithme, mais puisqu'on réalise des sommes de logarithmes, on peut alors utiliser la propriété $\log a + \log b = \log a * b$. Lorsqu'on atteint une certaine limite (1^{100}), on prend le logarithme du produit que l'on ajoute au résultat final. Ainsi, on a moins de logarithme à calculer et on gagne en temps de calcul. Afin d'améliorer la vitesse de calcul et d'affichage des fractales de Lyapunov, on utilise aussi la parallélisation. On récupère le nombre de threads du processeur et on découpe ensuite la génération

à partir d'un bloc en plusieurs blocs correspondant à chacun une région de la fractale. Nous avons décidé de séparer les différentes régions de la fractale à partir du nombre de "lignes" dans notre tableau. Pour cela, on utilise la librairie <thread> native, qui est assez simple d'utilisation dans notre cas. En effet, en séparant les calculs des exposants, il est possible de gagner un temps assez conséquent lors du calcul de la fractale.

1.4 Les Couleurs

Les couleurs sont obtenues depuis le fichier configuration, ensuite, on stocke dans un tableau et répartis selon les composantes rouges, vertes, et bleues, les différences entre la couleur maximale et minimale suivant le signe de l'exposant, puis la couleur à qui cette différence a été imputée. Les valeurs maximale et minimale de l'exposant de la région calculée sont stockées, pour créer une échelle de couleurs suivant le signe. Pour chaque composante de couleur, la différence de couleur est multipliée par la division de l'exposant par la borne adaptée et enfin ajoutée à la couleur stockée.

1.5 Interactions avec l'utilisateur

Afin de pouvoir interagir avec la fractale, on a besoin de pouvoir convertir les coordonnées écrans en coordonnées de Lyapunov. Avec cela, on utilise une région, qui définit les coordonnées de départ et d'arrivée sur les deux axe. Pour convertir, on divise, pour chaque coordonnées, la différence entre le nouveau et l'ancien point par la largeur de la texture, puis on pultiplie par la largeur actuelle et on ajoute l'origine.

1.5.1 Le zoom

Le zoom est représenté par un carré blanc réglable avec la molette de la souris. En cliquant, on récupère la région en convertissant les coordonnées écrans, et on calcule la zone à afficher avec cette région. Pour dézoomer, on conserve chaque région avant de zoomer que l'on place dans une pile. À chaque clic droit, on récupère cette région et on recalcule pour ré-afficher l'ancienne fractale

1.5.2 Le déplacement

Pour se déplacer, on utilise les flêches directionnelles qui permettent un déplacement d'un demi-écran. Si on essaye de se déplacer en dehors de la bordure, la région reste collée à la bordure

1.5.3 Le Menu

La touche Echap fait réapparaitre le menu pour pouvoir changer les options des fractales. On peut ainsi changer les couleurs, la précision et la séquence. Si seule la couleur est inchangée le programme ne recalcule pas les valeurs et met à jour seulement les couleurs

1.5.4 Capture

Pour pouvoir exporter la représentation, on appuie sur la touche Entrée, qui génère une image BMP, stockée dans le dossier screenshot.

1.6 Le Menu

Le menu a été réalisé avec la libraire "GTKmm", la version en C++ de GTK+, la librairie graphique de GIMP notamment. Elle dispose de nombreuses classes afin de permettre à l'utilisateur de créer des interfaces graphiques et des menus de navigation très facilement. Nous avons donc opté pour cette libraire afin d'implémenter le menu car elle dispose de nombreux widgets comme des selecteurs de couleurs, ou bien encore des zones de saisie de texte afin de permettre à l'utilisateur de pouvoir choisir la séquence et les couleurs entre autres . Le menu fait principalement appel à plusieurs petites fonctions qui lui permettent de compléter plus facilement le fichier de configuration. Nous avons décidé d'écrire les différents choix de l'utilisateur dans un ficher de configuration afin de pouvoir récuperer facilement et rapidement les valeurs entre les différentes classes sans avoir à passer par un Getter par exemple. Le menu est capable d'écrire les différentes couleurs sous la forme (255 255 255), la séquence saisie et aussi la précision souhaitée par l'utilisateur dans le fichier de configuration.

2 Les choix

2.1 Le langage

Nous avons décidé d'utiliser le langage C++ car c'est un langage nouveau pour nous, et intéressant de le maîtriser.

2.2 La librairie graphique

Nous avons utilisé la SDL car c'est une librairie que nous connaissons, et bien qu'elle fut créée pour le C, il est possible de l'utiliser pour le C++. Nous avons étudié d'autres libraries mais elles semblaient moins adapté pour de l'affichage pixel par pixel.

3 Les difficultés rencontrées

3.1 Le menu

Le menu a été une source de quelques difficultés. Nous avons décidé de nous orienter vers GTKmm, la version de GTK+ pour le C++. GTKmm dispose de nombreux widgets, comme des boutons ou des saisies de texte afin de permettre la création de menu plus facilement qu'en SDL, où l'on doit tout créer. La première et principale difficulté que nous avons rencontrée lors de la création du menu était d'apprendre le fonctionnement des librairies de GTK. Il a fallu apprendre l'utilisation et les normes de GTKmm qui sont très différentes de la SDL que nous connaissions déjà. La création du menu n'a requis que les bases de GTKmm réduisant ainsi la durée d'apprentissage. Le menu a également posé des problèmes de mise en forme. En effet, la première version du menu utilisait des box afin de contenir les différents widgets de GTK comme les boutons ou les labels. Mais, il n'etait pas possible de placer précisement les éléments à l'intérieur du conteneur. On a alors décidé de s'orienter vers une "grid", un conteneur qui permet lui de séparer la fenêtre du menu en plusieurs compartiments. Il était alors beaucoup plus facile de pouvoir positionner les différents éléments qui composent le menu. Mais, un autre problème était source de difficultés : l'affichage des textes à côté des boutons. GTKmm dispose de "labels" qui sont censés permettre à l'utilisateur d'écrire du texte. Mais, il suffisait juste de remplacer les "labels" par une de ses sous classes "AccelLabel".

3.2 Le Multi Threading

La difficulté principale du multi-threading / parallélisation était de comprendre les différents concepts du multi-threading. Une fois, les notions comprises, l'implémentation a été réalisée dans la foulée sans grande difficulté supplémentaire en "découpant" la génération en plusieurs parties.

3.3 Effets d'image

Originellement, nous avions prévu d'octroyer à l'utilisateur la possibilité de tourner la texture de 90° ou de la renverser à volonté. Cependant, ce n'était pas compatible avec la fonction de zoom, ces effets ont donc été supprimées du programme final.

4 Ressentis personnels