Responsable pédagogique	AF	**AM**	OP	PM
Période	Sem1	Sem2	**Sem3**	Sem4
Volume horaire	Cours/TD		TP	
			6	

[CPP]
JULIA & MANDELBROT

Indicateur temporel (hors rédaction du compte-rendu) :

questions	1h	2h	3h	4h	5h	6h
1 .. 2						
3 .. 5						
6 .. 13						
14 .. 19						
20 .. 24						
25						

Documents à rendre : Rapport de TP (modèle IRIS)

Ressources : Archive `tp_Fractales-v0.6-src.tar.gz`
Bibliothèque XamGraph version SDL-0.27 ou supérieure

Les ensembles de Julia et celui de Mandelbrot sont des images fractales particulières, qui n'existent pas dans la nature. Elles sont appelées « déterministes » car elles sont issues d'une opération déterministe : on décide si un point appartient ou non à ces ensembles, selon qu'une certaine suite, qui est fonction des coordonnées de ce point, converge ou non.

Les ensembles de Julia

Ces ensembles appartiennent au plan. Un point M du plan peut être défini par 2 coordonnées réelles (x,y), ou par un nombre complexe z. On définit une suite de points du plan complexe, où chaque point M_n a pour affixe z_n :

$z_0 = z$ où c est un nombre complexe fixé : $c = a + ib$

$z_{n+1} = z_n^2 + c$ pour $n \geq 0$

Si cette suite de points diverge, c'est-à-dire si les points « partent à l'infini », c'est-à-dire encore, si la distance de M_n à l'origine tend vers $+\infty$, le point M n'appartient pas à l'ensemble de Julia associé au nombre complexe c.

L'ensemble de Julia associé à c est donc constitué, par une logique implacable, du reste des points ! Cette remarque anodine a pour but de souligner le fait que pour un point M appartenant à Julia, la suite M_n a deux comportements possibles ; elle peut bien sûr converger vers un point du plan, mais également rester à distance finie de l'origine, sans toutefois converger.

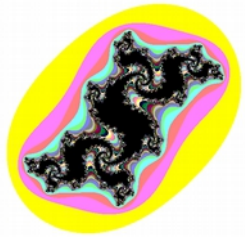
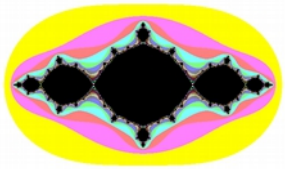
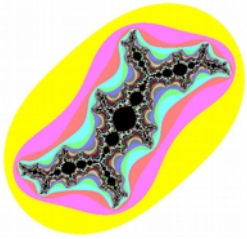
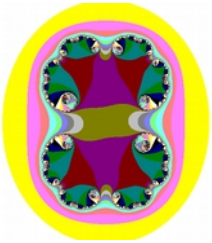
Est-il possible, dans le cas général, de prévoir si la suite associée à un point M va diverger ou non ?

On montre que si $|c| < 2$, ce que l'on peut décider, alors, si pour un certain n, le point M_n est à une distance de O supérieure à 2 $(|zn|>2)$, alors la suite diverge. Cela fait un critère de décision dans un sens. Par contre, on ne connaît pas de critère général permettant d'affirmer qu'une suite ne va pas diverger...

Les programmes d'ordinateur qui tracent les ensembles de Julia procèdent de la manière suivante : pour chaque point du plan affiché à l'écran, ils calculent les coordonnées z_n de la suite jusqu'à un certain rang qui dépend de la précision que l'on veut avoir (entre 20 et 500 généralement).

Si un de ces points est tel que $x_n^2+y_n^2 > 4$, le point n'appartient pas à l'ensemble et on lui attribue une couleur non-noire qui dépend de la vitesse n avec laquelle la suite a divergé. Si on arrive au $N^{ième}$ point de la suite sans que le test ait été positif, le programme colorie le point en noir, ce qui signifie qu'il appartient à l'ensemble !

On parle des ensembles de Julia, car il en existe un pour chaque complexe c (de module inférieur à 2). Ils ont des formes variées :

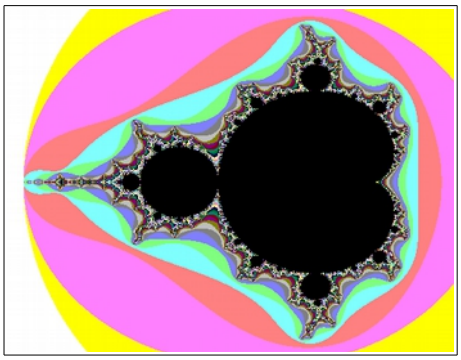
			
C = -0,0519 + i.0,688	*C = -1*	*C = -0,1134 + i.0.8606*	*C = 0,32 + i.0,0043*

L'ensemble de Mandelbrot

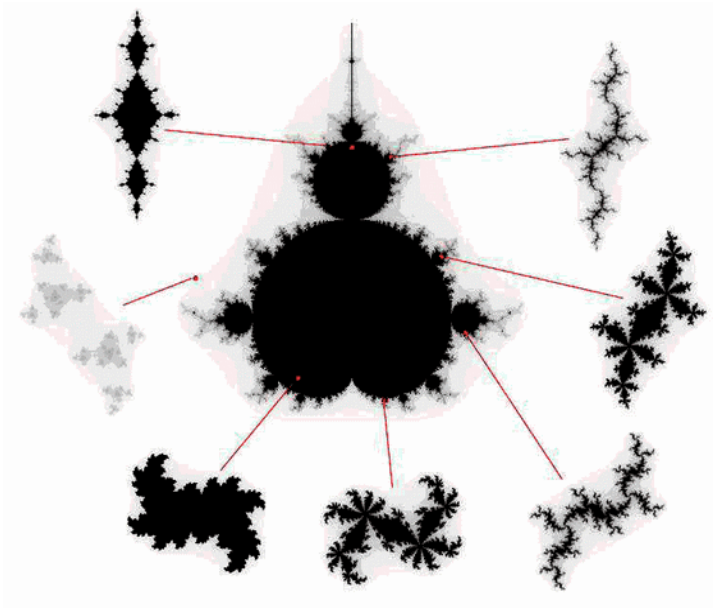
La construction de l'ensemble de Mandelbrot est très similaire à celle des ensembles de Julia ; pour un point M du plan, d'affixe z ; on considère la suite :

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + z \quad \text{pour } n \geq 0$$

Si cette suite ne diverge pas, le point M appartient, par définition, à l'ensemble de Mandelbrot. Le critère de divergence et l'algorithme informatique sont les mêmes que pour les ensembles de Julia.



Cet ensemble particulier -nommé aussi « pou » de Mandelbrot- est le préféré des amateurs de fractales, car il présente une très grande complexité qui lui donne une certaine beauté et un caractère fascinant : on y trouve des hippocampes, des éléphants (avec un peu d'imagination), des répliques de lui-même…



L'itération des polynômes complexes a été étudiée de façon indépendante par Julia (1918) et Fatou (1919-1920). Pour faire bref, on peut dire que les ensembles de Julia sont les frontières des domaines de Fatou…

C'est en 1975 qu'apparaît le mot « fractale » (de l'adjectif latin *fractus*, qui signifie « irrégulier ou brisé »), avec la parution du livre « Les objets fractals » de Benoît Mandelbrot, chercheur à IBM.

Lorsqu'on lit la liste des publications de Mandelbrot de 1951 à 1975, date de la publication de son livre en français, on est stupéfait par la diversité des domaines étudiés : bruit sur les lignes téléphoniques, théorie des jeux, linguistique, économie, cosmologie, turbulence… La multiplicité de ces domaines d'intérêt a certainement joué un rôle capital dans la genèse de sa découverte.

Les ensembles étudiés ici ne représente qu'une partie du vaste domaine de la géométrie fractale…

Question subsidiaire (pour les curieux et/ou les matheux…) : quel rapport existe-t-il entre l'ensemble de Mandelbrot et les ensembles de Julia ? L'image ci-dessus peut vous aider à répondre…

Cahier des Charges

L'objectif du TP est la réalisation d'un programme simple permettant de visualiser les ensembles décrits précédemment. Le programme doit permettre de tracer un ensemble de Julia (après sélection des parties réelle et imaginaire de c), ou l'ensemble de Mandelbrot.

Afin de garder le temps de calcul dans des limites raisonnables (avec un nombre d'itérations de l'ordre de 200), la fenêtre graphique utilisée sera de dimensions maximales 800 × 600 pixels.

L'utilisateur doit avoir la possibilité de zoomer une partie choisie du dessin, ceci autant de fois qu'il le désire…

La sélection d'une zone à agrandir doit impérativement se faire grâce à la souris :

- un clic gauche pour marquer un premier point d'une zone rectangulaire,
- le rectangle en cours de définition suit les mouvements de la souris,
- un nouveau clic gauche pour le deuxième point.

Le tracé dynamique du rectangle représentant la zone à zoomer n'altère pas le dessin initial. Dès que le deuxième point est défini, le contenu de la fenêtre est remplacé par la partie de dessin agrandie.

Le clic droit peut éventuellement être utilisé pour annuler une sélection en cours.

Préparation de la vue graphique

1. Travaillez sous Linux. Récupérez l'archive **`tp_Fractales-v0.6-src.tar.gz`** et la décompresser dans un répertoire de travail.

Cette archive contient une classe dérivée **`CVue`** qui va assurer la mise en place de notre fenêtre graphique ; et un embryon de classe **`CJulia`** destinée à modéliser les ensembles du même nom.

Un programme de test et un **`Makefile`** adéquat sont également gracieusement fournis. L'application utilise la désormais traditionnelle bibliothèque **`XamGraph`**, il convient donc de renseigner le **`Makefile`** quant à sa localisation (variable XAMRELDIR).

2. Testez le programme. Il doit normalement afficher la fenêtre graphique avec un titre… ainsi que la position courante du pointeur souris dans le plan.

La vue représente un plan complexe ! L'abscisse (axe X) est la partie réelle, et l'ordonnée (axe Y) la partie imaginaire. La classe **`CVue`** contient les méthodes nécessaires pour la conversion des pixels en coordonnées complexes.

Classe CJulia

La classe **`CJulia`** maintient ses complexes Zn et C (de classe standard C++ **`complex`**) ainsi que les bornes d'un plan de représentation. Son constructeur reçoit un nombre maximum d'itérations et les parties réelle et imaginaire de l'ensemble de Julia à représenter.

3. Complétez le constructeur de **`CJulia`** de manière à fixer le plan complexe avec $-2{,}0 \leq x \leq +2.0$ et $-1{,}5 \leq y \leq +1{,}5$. Ajoutez l'initialisation du membre C avec les valeurs reçues en arguments.

En analysant les suites des ensembles de Julia et de l'ensemble de Mandelbrot, on constate que les expressions se ressemblent… mis à part l'initialisation de Zn et de C. Plaçons donc ces opérations dans une méthode à part (*y'a de l'héritage en perspective*).

4. C'est le rôle de la méthode virtuelle protégée **`init()`** qui reçoit un complexe en argument. Dans le cas de **`CJulia`**, initialisez simplement Zn avec ce complexe.

Reste à implémenter la méthode **`dot()`** qui calcule la divergence d'un point du plan complexe pour un ensemble de Julia donné. La méthode doit renvoyer un rang entier compris entre 0 et le nombre maximum d'itérations, ou -1 si le point appartient à l'ensemble.

5. Implémentez le corps de la méthode **`dot()`** en respectant la théorie exposée en tête de ce document ; à savoir :
   - invoquez **`init()`** avec les parties réelle et imaginaire reçues en arguments pour positionner la valeur initiale de Zn ;
   - pour n allant de 0 à la valeur max. (attribut privé) ;
     - calculez Zn+1 (variable locale) en fonction de Zn et de C (attributs protégés) ;
     - si le module de Zn+1 est supérieur ou égal à 2 (autrement dit, si la norme de Zn+1 est supérieure ou égale à 4), alors retournez n ;
     - copiez Zn+1 dans Zn.
   - finpour ;
   - retournez -1.

La classe **`CJulia`** est maintenant complète ! Mais la vue doit avoir accès à un objet **`CJulia`** pour pouvoir le représenter…

6. **`CVue`** possède une agrégation **`CJulia*`** ; modifiez son constructeur de manière à ce qu'il accepte un argument lui permettant d'initialiser cet agrégat.

7. Toujours dans le même constructeur, initialisez le plan complexe géré par la vue à partir des bornes de l'objet **`CJulia`**.

8. Ajoutez au programme de test la déclaration : **`CJulia* fractale = NULL ;`**

9. Placez judicieusement, toujours dans **`main()`**, la création dynamique de l'instance **`CJulia`** avec les données maintenues localement (variables **`nMax`**, **`re`** et **`im`**). Ne pas oublier de terminer le programme par la destruction de l'objet **`fractale`**.

10. Modifiez la création de l'objet **`CVue`** en conséquence.

Toujours rien sur la fenêtre graphique ?
C'est normal, il faut encore renseigner la méthode **`CVue::drawFractal()`** !

11. Celle-ci doit dessiner la fractale point par point sur le plan complexe. Mettez en place pour cela une double boucle de balayage des pixels du plan, dans laquelle vous devez invoquer la méthode **`dot()`** de la fractale (Attention, cette dernière attend des coordonnées logiques et non des pixels… employez les ressources privées **`xLogical()`** et **`yLogical()`** de **`CVue`** !).

12. Ajoutez les instructions nécessaires au dessin proprement dit du point coloré (voir pour cela la ressource **`putPixel()`** héritée de **`XamGraph`**). La couleur est à choisir en fonction de la valeur de retour de **`dot()`** :

    -1 → couleur noire
    0 → couleur blanche
    n → proportions RGB variables en fonction de n

    note : essayez par exemple color(r,g,b) avec r = g = b = (int)(255 * (n % 32 + 1) / 33.0)
    ou utilisez directement la méthode privée scaledColor() de CVue…

13. Testez le programme en faisant varier le nombre complexe défini dans la fonction principale. Le concours des plus belles fractales est ouvert ! (les plus belles peuvent être jointes au rapport, légendées avec leur valeur de C.)

Classe dérivée CMandelbrot

Toujours partisans du moindre effort, nous allons maintenant tout simplement spécialiser la classe **`CJulia`** de manière à obtenir le célèbre pou de Mandelbrot…

14. Créez une nouvelle classe **`CMandelbrot`**, dérivée de **`CJulia`**, dont le constructeur reçoit en argument un nombre maximum d'itération ; à transmettre à sa classe de base.

15. Pensez à mettre à jour le **`Makefile`**…

16. Complétez le constructeur par l'initialisation du plan complexe : $-2{,}4 \leq x \leq +1.2$ et $-1{,}5 \leq y \leq +1{,}5$.

17. Surchargez la méthode **`init()`** de la classe de base afin d'assurer une initialisation correcte de Zn et de C pour l'ensemble de Mandelbrot.

18. Modifiez **`main()`** en positionnant le drapeau **`julia`** par défaut à **`false`** et en ajoutant la création d'un objet **`CMandelbrot`** à l'endroit idoine (**`fractale = new`** …).

19. Testez le programme. Vous devez obtenir la figure attendue !

Gestion du zoom

Les fractales obtenues sont jolies, mais il est temps de sortir la loupe pour s'assurer qu'il s'agit effectivement de fractales…

Pour réaliser cela, il faut gérer les événements souris liés à notre fenêtre graphique. Le programme de test contient déjà une boucle d'événements et les intercepteurs idoines (cf. doc. XamGraph) ; ces derniers pourraient être enrichie par les traitements suivants :

```
si ( un clic gauche est détecté ) alors
      initialiser le rectangle de capture avec les coord. de la souris
finsi

si ( un clic droit est détecté ) alors
      si ( une capture est en cours ) alors
            abandonner la capture
      finsi
finsi

si ( un mouvement de la souris est détecté ) alors
      si ( une capture est en cours ) alors
            changer le rectangle de capture avec les coord. de la souris
      finsi
finsi
```

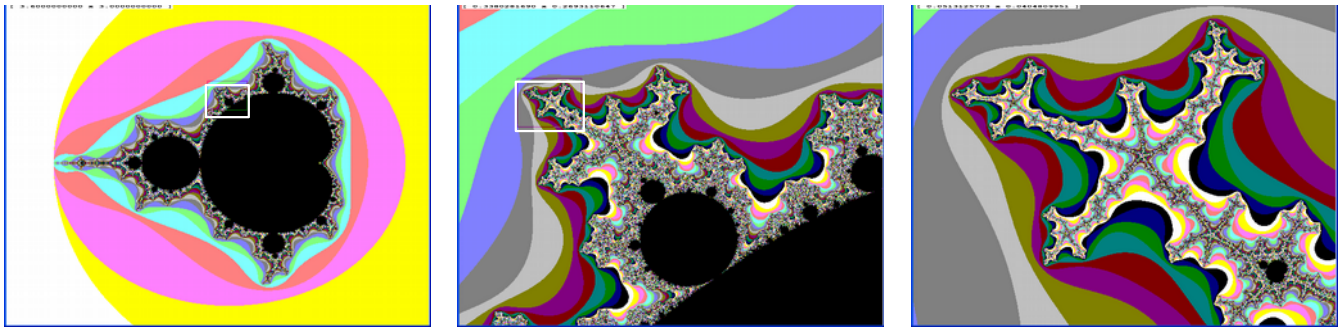
Les plus attentifs d'entre vous auront remarqué que les items soulignés correspondent comme par hasard à des méthodes publiques de la classe **`CVue`**.

L'initialisation du rectangle de capture concerne aussi bien le point de départ que le point d'arrivée, comme le montre le code de la méthode correspondante.

20. Complétez l'Implémentation des intercepteurs d'événements du programme principal de manière à invoquer judicieusement les méthodes **`initCapture()`**, **`isCapturing()`**, **`changeCapture()`** et **`stopCapture()`** de l'objet **`vue`**.

Le tracé dynamique du rectangle représentant la zone à zoomer n'altère pas le dessin initial. Dès que le deuxième point est défini, le contenu de la fenêtre est remplacé par la partie de dessin agrandie.

Exemples de zooms successifs :



Le code de la méthode **`drawCapture()`** est donné. Il assure le dessin en mode XOR d'un rectangle dont les dimensions sont maintenues par les attributs privés (**`xBegin`**,**`yBegin`**) et (**`xEnd`**,**`yEnd`**).

Le mode XOR permet un affichage/effacement graphique sans perturbation de l'image de fond. Un premier appel à **`drawCapture()`** montre le rectangle, le suivant l'efface, et ainsi de suite…

21. Complétez la méthode **`initCapture()`**. Celle-ci est appelée en réponse aux clics sur le bouton gauche de la souris ; elle sert donc aussi bien à démarrer une capture qu'à la terminer. Le drapeau **`capture`** signale si une capture est en cours ou non. La méthode doit mettre à jour ce drapeau et les membres qui maintiennent les dimensions du rectangle.
    Le code assurant le zoom proprement dit est déjà en place ; il doit être précédé d'une mise à jour des coordonnées du plan complexe à partir de celles du rectangle de capture.

22. Implémentez maintenant la méthode **`changeCapture()`**. Elle est utilisée pour répondre aux événements traduisant un déplacement de la souris ; elle doit effacer, mettre à jour la taille puis ré-afficher le rectangle de capture.

23. Pour en finir avec le chapitre « zoom », remplissez la méthode **`stopCapture()`** afin de permettre l'abandon d'une capture en cours ; sans oublier d'effacer le rectangle courant de capture.

24. Testez le programme et plongez dans les entrailles des fractales…

Amélioration de l'IHM

Pour terminer en beauté cette petite application d'illustration des fractales, nous allons améliorer l'interface utilisateur en permettant à celui-ci de passer sur la ligne de commande les informations de paramétrage de la fractale :

- sélection ensemble de Julia ou ensemble de Mandelbrot (défaut) ;
- pour Julia, indication des parties réelle et imaginaire du complexe ;
- choix du nombre maximum d'itérations ;
- … …

Par un heureux hasard, les informations citées sont toutes maintenues par le programme principal sous forme de variables locales (**`nMax`**, **`julia`**, **`re`** et **`im`**).

Il nous suffit donc de définir une syntaxe d'utilisation du programme, puis d'assurer la récupération des arguments de la ligne de commande pour affecter ces variables.

Proposition de syntaxe : **`fractale [-n<iter>] [-j  [-r<re>] [-i<im>] ]`**

Exemples d'usage :

`fractale -n 100`	100 itérations, Mandelbrot
`fractale -j -r -0.25`	200 itérations, Julia C=(-0.25,0.8606)
`fractale -j -r -1 -i0`	200 itérations, Julia C=(-1,0)

25. Consultez la documentation (*man*) de la routine standard d'analyse des arguments de la ligne de commande : **`getopt()`**. Enrichir le programme principal de manière à satisfaire l'IHM décrite ci-dessus.

Annexes

Fausses couleurs

Les exemples de fractales présentées dans le présent document sont en fausses couleurs.

Celles-ci sont obtenues en utilisant le modulo 16 de n (voir étape 12) comme indice dans un tableau de couleurs définies sous forme de chaînes de caractères (le tableau ne contient bien évidemment ici ni noir ni blanc) →

La ressource **`stringToColor()`** héritée de **`XamGraph`** permet d'affecter la couleur d'un pixel directement à partir de sa définition sous la forme "#RRGGBB".

```
string colors[16] = {
                "#E6E6FA",    // lavande
                "#FFFF00",    // jaune
                "#FF00FF",    // magenta
                "#FF0000",    // rouge
                "#00FFFF",    // cyan
                "#00FF00",    // vert
                "#0000FF",    // bleu
                "#800080"     // violet
                "#C0C0C0",    // gris clair
                "#800000",    // marron
                "#C000C0",    // magenta
                "#800000",    // rouge foncé
                "#008080",    // cyan foncé
                "#008000",    // vert foncé
                "#000080",    // bleu foncé
                "#FFA500"     // orange
} ;
```

Diagramme de classes de l'application