Rapport du projet Flamefract

Valérian Rousset Guy-Laurent Subri

2 juin 2013

Améliorations

Dans ce document, nous allons présenter quelques améliorations que nous avons faîtes. Pour les grandes améliorations, nous allons expliquer brièvement l'implémentation en Java, tandis que pour les plus petites, nous allons juste mentionner pourquoi nous voulions implémenter cette fonctionnalité et ses avantages (et inconvéniants s'il y en a).

Apparition "incrémentale" de la fractale

Pourquoi ? L'apparition "incrémentale" de la fractale permet à l'utilisateur de voir la fractale se déssiner petit à petit devant ses yeux émerveillé par la beauté de l'image qu'il a créée (ou presque) ! Avant cette feature, nous devions attendre, sans savoir quoi que ce soit sur l'état du calcul, avant de voir apparître brusquement une nouvelle fractale (en particulier si on met le GUI en plein écran ...). Cette feature est plus "user-friendly" qu'autre chose, mais elle est assez élégante. De plus, l'interface se bloquait le temps de calculer la fractal, ce qui est en général très dérangeant pour l'utilisateur lambda. Pour suivre la progression du calcul (et savoir si des points sont encore calculés, une barre de chargement vient complèter l'interface, elle est décrite plus loin). Ce besoin d'une fractale incrémentale devient plus forte proportiellement à la taille de la fenêtre, car le nombre de points augmente drastiquement, et est pratique pour avoir une premier apreçu des modifications.

Comment ? L'idée est d'avoir un Timer (de swing) pour gérer le lancement périodique du rafraichissement, avec une valeur décidée par l'utilisateur (voir amélioration "Fichier de configuration") ou calculée dynamiquement avec une algorithme adaptif. Cet algorithme est simple, on calcul un premier set de point, plus petit, on regarde combien de temps cela nous a pris, et nous augmentons le nombre de points à calculer en fonction du temps possible d'utilisateur ; a l'inverse, si nous prennons trop de temps, on reduit le nombre de point à calculer. Si, même avec un très faible nombre de points, nous n'arrivons pas à mettre à jour assez vite, nous augmentons la fréquence du rafraichissement.

Maintenant, la partie incrémentale : l'idée est de relancer la fonction de rafraichissement tant que le nombre total de point n'est pas dessiné, via un Timer. Il suffit après de gérer les différents cas : si le Timer ne tourne plus, nous le créons et le lançons, de plus nous créons un nouveau FlameAccumulator.Builder que nous gardons dans la class ; s'il nous reste des points à calculer, nous en calculons un autre set et l'affichons ; si le nombre total de point est calculé, on arrête le Timer. Dernier cas, si la taille de l'accumulateur n'est pas la même que celle demandée par swing (ce qui signifie que la fenêtre est redimensionnée, en donc que l'état actuel du calcul n'est plus intéressant), on arrête le calcul de la fractal, et le processus va de nouveau appeler la fonction.

Barre de chargement

Pourquoi ? Nous avons ajouté une barre de chargement sous l'image de la fractale. Cette barre s'affiche uniquement lorsque la fractale est calculée. Elle est discrète et utile pour l'utilisateur, d'autant plus que celui-ci voit sa fractale apparaître petit à petit.

Comment ? L'implémentation est facile à mettre en place après l'ajout de l'apparition incrémentale de la fractale ; comme nous calculons une serie de nouveau points à chaque rafraichissement de l'interface, nous avons le nombre total de points calculé jusqu'à lors et le nombre total de point à calculer, nous avons donc un ratio. Il suffit après de dessiner un rectangle en bas de la fractale après chaque rafraichissement, et de ne plus en faire quand le fractale est finie.

Fichier de configuration

Pourquoi ? Tous les deux, venant du monde du logiciel libre et des power-users, nous aimons avoir le contrôle sur notre système et sur les programmes que nous utilisons, et nous apprécions particulièrement le fait de pouvoir configurer les programmes à notre guise. C'est pourquoi nous avons mis en place un fichier de configuration ! Dans ce fichier, nous pouvons changer la matrice de base, les poids accordés aux différentes transformations, la "density" de l'image, . . .Le fichier de configuration supporte les commentaires. Nous avons la possibilité d'avoir des valeurs "aléatoires" (pas pour tous les champs configurables, comme par exemple le refresh rate, car il évident qu'avoir un refresh rate aléatoire est insensé). Le fichier de preférence est ainsi un simple fichier text, lisible par n'importe quel processus ; l'ajout de commentaires détaillée aidant à comprendre chaque variable ; utilisant le dièse comme début de commentaire, le fichier est très proche de la plus par des fichiers de configuration POSIX, où le choix de l'utilisateur est prioritaire.

Comment ? En utilisant un patron de type Builder, nous avons une classe chargeant tout le contenu du fichier de preférence, le parse et en crée un instance de Preferences. La mise en place du parsing est plutôt simple : nous lisons chaque ligne du fichier "flamefract.conf", chaque ligne est soit un commentaire, soit une variable avec une valeur avec ou sans commentaire postfixant. Enlever les commentaires et les espaces non-necessaires sont deux regex chainée, un test pour voir si la chaine resultant est vide autrement, un gros switch. Dans ce switch, si nous avons une valeur connue, nous assignons la bonne variable avec la valeur que nous parsons, autrement, nous affichons un message d'erreur et quittons. Si le magic word "random" apparait dans le champ de valeur, chacune des fonctions le supportant génère une valeur aléatoire pour ce qu'il doit retourner. Par design, il est facile d'ajouter de nouveaux parametres, car la plupart des fonctions telles que parseDouble, parseRectangle, parseColor sont implementées et retournent respectivement un double, un rectangle et une couleur. L'utilisation du fichier est décrite simplement dans le fichier par défaut, créé s'il n'existe pas déjà, en particulier que chaque nouvelle matrix dans le fichier s'ajoute au constructeur au lieu d'écraser l'ancienne valeur comme pour le reste des variables. Bien que l'utilisation d'un fichier soit peu connu des utilisateurs lambdas des systèmes non-POSIX, il est très fréquent de les utiliser et ainsi de laisser le choix à l'utilisateur de contrôler les paramètres de ses programmes.

Menus et raccourcis

Pourquoi ? Nous avons décidé d'implémenter des menus et des raccourcis clavier (car nous aimons pouvoir gérer le maximum de choses sans avoir besoin de la souris). Nous avons plu-

sieurs menus, qui permettent d'afficher d'autres fonctionnalités (terminées ou non), comme, par exemple, la sauvegarde de l'image, la sauvegarde de la configuration actuelle du programme (dans le fichier de configuration sus-mentionnée) ou le plein-écran.

Comment ? Les menus principaux ("Fichier" et "Affichage"), ainsi que leurs contenu sont des enums, facilitant l'ajout et la supression, un switch se faisant pour decider quelle action on fait pour chaque sous-menu.

Sauvegarde de l'image

Pourquoi ? Sans cette fonctionnalité, nous trouvons que le programme n'a pas vraiment d'intérêt. Avant de créer le GUI, nous pouvions enregistrer nos images (c'était d'ailleurs le seul moyen de les voir). Nous avons donc voulu ajouter ce que nous avions perdu en implémentant l'interface graphique. Une conséquence assez sympathique de cette feature est le fait que nous pouvons ensuite mettre notre fractale en fond d'écran.

Comment ? Pour sauvegarder, nous laissons le choix à l'utilisateur de l'endroit et du nom pour l'image, pour nous calculons dans le fond la fractal (pour la taille de l'écran, l'idée étant que c'est probablement la plus grande résolution que l'utilisateur ait besoin), et l'enregistrons dans le fichier demandé. Par l'utilisation d'un builder et d'un enrigistreur incrémentale, nous pouvons afficher une barre de chargement durant le calcul. Pour sauvegarder, nous laissons le choix à l'utilisateur de l'endroit et du nom de l'image, puis nous calculons la fractal (avec la taille de l'écran, l'idée étant que c'est probablement la plus grande résolution que l'utilisateur ait besoin), et l'enregistrons dans le fichier demandé. Par l'utilisation d'un builder et d'un exportateur de fichier incrémentale, nous pouvons afficher une barre de chargement durant le calcul.

Sauvegarde de la configuraiton de l'image

Pourquoi ? L'idée de sauver la configuration (c'est-à-dire le poids des variations, le nombre de transformations, etc...) est venue du fait que si nous enregistrions une image, nous ne pouvions pas reprendre l'image et modifier quelques paramètres. Nous nous sommes donc dit qu'il nous *fallait* pouvoir reprendre la configuration d'une image.

Comment ? Très simplement, nous créeons une instance de Preferences avec l'état actuel de la fractal, puis nous enregistrons ces valeurs dans le fichier par defaut de configuration.

Plein écran

Pourquoi ? Nous voulions avoir la possibilité de mettre notre image en plein écran, pour plusieurs raisons :

- C'est pratique, pour voir les détails de la fractale
- On a une meilleure idée du résultat de la sauvegarde de l'image
- Ça le fait...

Comment ? Nous créons un nouveau JFrame et y attachons une nouvelle instance (pour ne pas avoir à la recalculer à la sorti du plein écran) de la fractale, que nous mettons simplement en plein-écran.

Réinitilisation

Pourquoi ? Nous nous sommes dit qu'il était pratique de pouvoir revenir à l'image initiale sans avoir à fermer et relancer le programme (surtout après que nous ayons modifier le fichier de configuration sans retrouver les paramètres originelles). Nous avons donc mis en place un menu et un raccourci clavier (Ctrl-N) qui réinitialisait les paramètres du programme.

Comment ? Pour eviter des problèmes avec les relations d'observateurs, nous simulons un clic sur le boutons supprimer jusqu'à qu'il nous reste qu'une seule fractale, après, nous ajoutons toutes les transformations que nous avons dans les preférence par défaut puis supprimons la fractale que nous avions encore de la dernière fois.