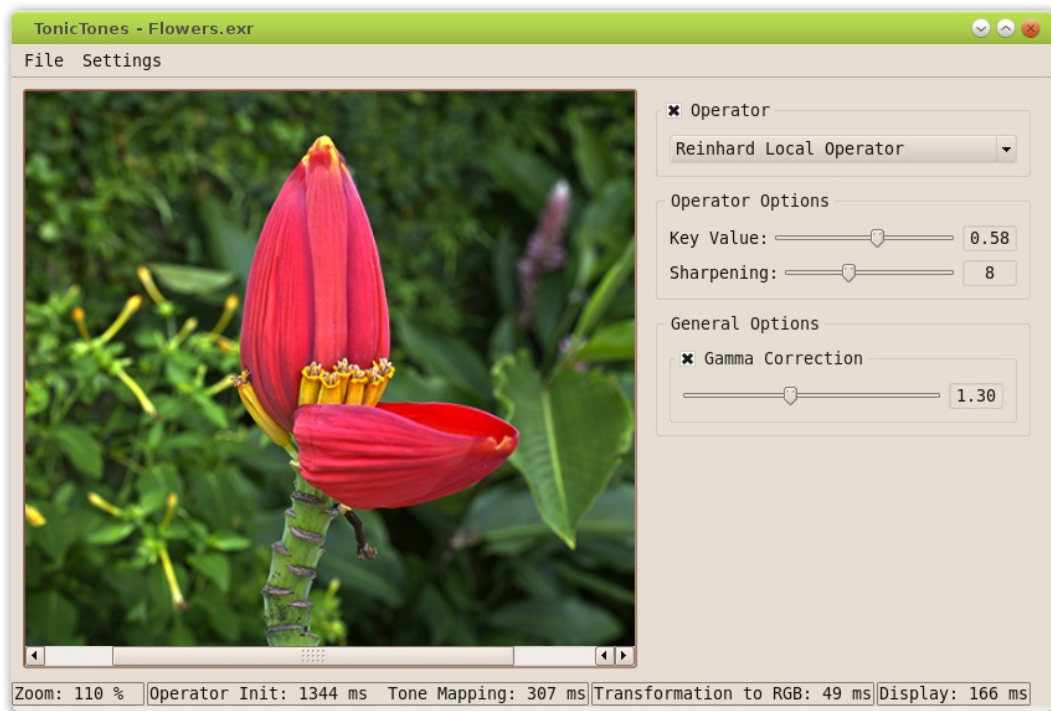


Développement : correction de ton sur des images HDR

JÉRÉMY LAUMON



Introduction

J’ai appelé le logiciel que j’ai développé “TonicTones”. Il est portable, libre et a été développé en C++ à l’aide de Qt. Dans l’état actuel, il est capable d’ouvrir des images HDR au format OpenExr ainsi que des images normales (jpeg, png etc.) et d’appliquer les opérateurs global et local décrits dans les travaux de Erik Reinhard à cette adresse : <http://www.cs.utah.edu/~reinhard/cdrom/>.

Une documentation du code est présente dans le dossier `Doc/Html`. La documentation a été générée depuis les commentaires du code grâce à Doxygen 1.7.2 et peut être régénérée en passant le fichier `Doxyfile` en paramètre à l’exécutable `doxygen`.

Le programme, son code, ses commentaires et sa documentation sont écrits en anglais. Toutefois, la totalité du texte affiché dans le programme a été passé à la fonction `QObject::tr()` et est donc traduisible facilement.

Pour compiler la totalité du projet il suffit de taper les commandes `qmake BuildProject.pro` suivi de `make` dans un shell.

Structure du code

Pour avoir un code le plus modulaire possible, TonicTones utilise des plugins qui sont chargés au lancement du programme. Deux sortes de plugins sont utilisés : les “image loaders” et les “tone mapping operators”. Ils servent respectivement à charger les images et à appliquer un traitement de type correction de ton. Ces plugins sont gérés par un “loader manager” et un “operator manager”.

Certaines classes sont regroupées dans une bibliothèque dynamique (`libTT_Api.so`) afin d’être accessibles aux plugins. Cette bibliothèque n’a pas besoin d’être placée dans un répertoire spécifique sous les systèmes Unix, elle est automatiquement recherchée dans le répertoire courant lors du lancement de l’application.

Cette structure à base de bibliothèques dynamiques et de plugins permet d’ajouter facilement des fonctionnalités au programme sans devoir le modifier et le recompiler, ou du moins, sans le recompiler entièrement.

Fonctionnement général

A l’ouverture du programme, le loader manager et l’operator manager vont parcourir les dossiers `Loaders/` et `Operators/` à la recherche de plugins à charger. Le chargement de ces plugins va permettre aux managers de construire chacun une liste de factories leur permettant d’instancier des loaders (ou des opérateurs) sur demande. L’operator manager instancie automatiquement un opérateur (le premier par ordre alphabétique) qui devient l’opérateur actif.

Lorsqu’une image est sélectionnée pour être ouverte, le loader manager va rechercher un loader capable d’ouvrir ce type de fichier et lancer une exception s’il n’en trouve pas (le type du fichier est déterminé par son extension). Si un loader est trouvé, il va charger le fichier ou à son tour lancer une exception s’il n’y arrive pas. Dans la version courante du programme, les loaders doivent renvoyer l’image dans l’espace de couleur Yxy. L’instance du loader est supprimée dès que l’image est chargée.

L'image est ensuite passée à l'opérateur actif qui va la traiter et envoyer un signal à l'interface graphique pour dire qu'une image est prête à être affichée. L'image modifiée est récupérée, passée en RGB puis affichée (avec accessoirement une correction gamma).

Note sur les espaces de couleurs

TonicTones a été conçu pour que les loaders puissent renvoyer des données dans n'importe quel espace de couleurs (à condition que celui ci contienne un champ luminance). La limite étant que la classe qui représente les images (**HdrImage**) sache faire la conversion vers RGB. Pour l'instant seule la transformation Yxy vers RGB a été implémentée. Pour travailler en Yuv par exemple, il suffirait d'ajouter la transformation qui convient à la fonction `toRGB()` de la classe **HdrImage**.

Performances

Les points limitant la vitesse du programme ont été optimisés autant que possible. Ces points comprennent notamment le changement d'espaces de couleurs, l'affichage de l'image, l'initialisation des opérateurs et leur application. Toutes les opérations qui le pouvaient ont été calculées une seule fois avant les boucles et certaines fonctions ont été déclarées **inline**.

Toutefois le programme n'utilise les possibilités de calcul parallèle ni du CPU ni du GPU. Les performances pourraient être grandement améliorées grâce à cela.

Voici quelques temps d'exécutions sur une image 1040x1040 pixels (Blobsies.exr) avec un processeur Intel Core 2 Duo T5550 (1.83GHz) :

Opérateur	Initialisation	Tone Mapping	Yxy → RGB	Affichage
Global	140 ms	55 ms	190 ms	190 ms
Global avec burn out	140 ms	75 ms	190 ms	190 ms
Opérateur local	3050 ms	1050 ms	190 ms	190 ms

La correction gamma ajoute environ 500 ms supplémentaires au temps d'affichage.

Résultats

Comme on peut le remarquer dans les figures 1 et 2, les images HDR affichées sans traitement sont souvent trop sombres ou trop lumineuse. L'opérateur global permet de ramener toutes les valeurs dans des limites affichables.

Grâce à l'ajout du "burn out" une petite partie des valeurs est maintenue au dessus du maximum affichable afin de permettre un effet d'éblouissement.

Enfin, l'opérateur local adapte sa correction à chaque pixel pour garder un maximum de détails dans les zones de fort contrastes. On remarque très bien les contours des arbres mis en évidence dans la figure 1 et les détails des nuages dans la figure 2.

Les images utilisées dans les résultats sont disponibles à cette adresse : <http://www.openexr.com/downloads.html>.



FIGURE 1 – Fog.exr - De gauche à droite et de haut en bas : image originale, operateur global, operateur global avec burn out, operateur local



FIGURE 2 – Kapaa.exr - De gauche à droite et de haut en bas : image originale, opérateur global, opérateur global avec burn out, opérateur local