

««< HEAD

»»> aa9ffccc95334f970d51ec98232438720c64bf9d

Rapport de projet

Retouche d'images sur Android

Manuel Ricardo Guevara Garban
Loïc Lachiver
Romain Pigret-Cadou
Sofiane Menadjlia

L3 Informatique
Projet technologique
2020

Université
de **BORDEAUX**

Table des matières

1 Introduction :

Ce rapport synthétise notre travail de développement d'une application de retouche d'image exécutable sur la plateforme mobile Android.

Réalisé dans le cadre de l'UE *Projet technologique* lors du 2ème semestre de Licence 3 informatique à l'Université de Bordeaux, les objectifs de ce rendu de groupe étaient les suivant :

- Réaliser une application graphique Android
- Permettre de charger, éditer et sauvegarder facilement une image
- Proposer des effets par simple modification de pixels
- Proposer des effets manipulant des histogrammes
- Proposer des effets de convolution
- Utiliser la technologie d'accélération RenderScript
- Gérer le développement d'un projet de groupe

➔Lien GitHub du projet : <https://github.com/picachoc/pimp-android>

2 L'application :

2.1 Lancement :

TODO

2.2 Utilisation :

TODO

3 Effets :

Interface :

L'utilisateur choisit un effet parmi la liste d'effets affichée en bas de l'écran, ce qui affiche les réglages disponibles (s'il y en a). L'utilisateur peut ensuite choisir de confirmer ou d'annuler la modification apportée par l'effet choisi.

Structure du code :

Les effets sont rassemblés dans le package filters (fr.ubordeaux.pimp.filters). La classe *Retouching* contient les réglages de luminosité, contraste, saturation et teinte. La classe *Convolution* contient tous les effets liés à la convolution (flou, détection de contour etc.). Toutes ces méthodes sont appelées lors de l'appui de boutons ou de glissement de seekbars. Les seekbars ont généralement une étendue allant de 0 à 255 (sauf pour les réglages de teinte), et certains effets nécessitent des valeurs pouvant être négatives. Ainsi la valeur de seekbar est modifiée dans la méthode appelante de ces mêmes effets.

3.1 Luminosité (Brightness) :



Méthode appelante : *Retouching.setBrightness()*

Script : *brightness.rs*

Ce réglage se contente d'ajouter une valeur (positive ou négative) aux trois canaux RGB de l'image. Cette valeur est fixée par la seekbar. Les valeurs sont tronquées entre 0 et 255, par conséquent on perd de l'information dans les valeurs extrêmes de luminosité.

Cet effet n'utilise pas la luminosité existante de l'image, ainsi on peut obtenir des résultats qui sont parfois discutables, par exemple le noir qui s'éclaircit et inversement pour le blanc. Pour pallier à ce problème, on pourrait introduire une multiplication afin de modifier la luminosité proportionnellement à celle existante. Cependant, cette solution modifie aussi le contraste, nous avons donc choisi de laisser l'algorithme tel quel.

3.2 Contraste (Contrast) :

Méthode appelante : *Retouching.dynamicExtensionRGB()*

Script : dynamicExtension.rs

Ce réglage effectue une extension linéaire de dynamique. Les nouveaux extremum de l'histogramme sont définis à partir de la position de la seekbar. La dynamique est ainsi étendue autour d'une valeur se situant au milieu des deux anciens extremum de l'histogramme*. On a donc une image uniforme lorsque l'on règle le contraste au minimum.

Le principal problème de l'extension dynamique est qu'elle ne permet pas d'augmenter le contraste à des grandes valeurs. En effet, les extremum sont vite atteints. La solution serait de faire une transformation linéaire par morceaux.

*Il serait peut-être plus juste de prendre la médiane de l'histogramme cumulé afin d'avoir une valeur qui représente mieux la "valeur moyenne" de l'image.

3.3 Saturation (Saturation) :

3.4 Convolution (Blur, Sharpen, Neon) :

Dans cette sous-section on retrouve des effets pour flouter une image en passant par deux types différents de kernel : Gaussien et moyenneur (bouton "Blur"), des effets pour améliorer la netteté d'une image avec la fonction "Laplacian of gaussian" (bouton Sharpen) et des effets de détection de contours (bouton "Neon") en réalisant une convolution avec des opérateurs comme Sobel, Prewitt, Kirsch ou simplement avec un kernel de Laplace.

Pour cet effet nous avons implémenté 3 méthodes de convolution.

- Convolution classique.
- Convolution séparable.
- Produit de convolution.

Convolution classique

Cette méthode de convolution suit le même algorithme vu en cours, on peut l'appliquer avec des noyaux asymétriques de dimensions impaires uniquement, cette opération a été parallélisée grâce à l'outil "renderscript" qui parallélise le calcul par CPU multithreading/GPU. En plus dans cette méthode (et dans toutes les autres) nous avons rajouté des optimisations pour parcourir seulement les index nécessaires de l'image lors de la convolution. En calculant les index des centres des deux dimensions du kernel on peut anticiper et éviter les appels de fonction inutiles sur les bords de l'image par exemple.

$X_{début}$ et X_{fin} étant le premier et dernier index à parcourir dans l'axe des abscisses respectivement et $Y_{début}$ et Y_{fin} étant le premier et dernier index à parcourir dans l'axe des ordonnées respectivement.

$$Kernel_{CentreX} = \frac{Largeur_{kernel}}{2}$$

$$Kernel_{CentreY} = \frac{Hauteur_{kernel}}{2}$$

$$X_{début} = Kernel_{CentreX}$$

$$X_{fin} = Largeur_{image} - Kernel_{CentreX}$$

$$Y_{début} = Kernel_{CentreY}$$

$$Y_{fin} = Hauteur_{image} - Kernel_{CentreY}$$

En définissant les index du début et de fin de parcours pour les deux dimensions de l'image sur le script de lancement renderscript comme ci-dessus on peut s'en passer de quelques appels de fonctions sur les bords de l'image et ainsi gagner du temps de calcul.

Convolution séparable

Pour la convolution classique avec un kernel de taille $N * M$ on doit faire $N * M$ multiplications pour chaque échantillon, cependant si le kernel est séparable on peut passer à $N + M$ opérations.

Or afin d'optimiser le calcul de la convolution sur des filtres séparables comme le filtre gaussien et moyenneur nous avons implémenté une méthode de convolution séparable.

$$\begin{bmatrix} A \cdot a & A \cdot b & A \cdot c \\ B \cdot a & B \cdot b & B \cdot c \\ C \cdot a & C \cdot b & C \cdot c \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot [a \quad b \quad c]$$

Un kernel est séparable quand sa matrice des poids peut être représentée par le produit de deux vecteurs.

Ce calcul est réalisé en faisant deux convolutions unidimensionnelles successives (en X et en Y) sur l'image d'origine en stockant le résultat dans une image intermédiaire. La propriété d'associativité de la convolution rend ce calcul possible.

$$x * (N * N) \iff (x * N) * N$$

La limite de cette méthode c'est que l'on doit passer par une image partielle pour la réalisation du calcul en utilisant plus de mémoire que pour une convolution classique.

Produit de convolution

Cette méthode prend en paramètre deux kernels de même taille et réalise deux convolutions avec chaque kernel et ensuite calcule le produit des deux. Vu que ces deux convolutions sont indépendantes l'une de l'autre on peut se permettre de les réaliser au même temps dans le script et de faire l'addition des valeurs absolues résultantes des deux accumulateurs. Cette méthode est utilisé notamment pour faire des convolutions avec les opérateurs de Sobel, Kirsch, Prewitt.

3.4.1 Flou gaussien et moyenieur (Blur) :



FIGURE 2 – Image originale et image avec flou gaussien Méthode appelante

4 Structure du projet :

4.1 Structure graphique Android et navigation :

Pour afficher certains éléments d'interface comme par exemple les informations de l'image (bouton **1**) nous utilisons des **Fragment**. En effet une activité supplémentaire n'est pas nécessaire car cette petite partie de l'application ne correspond pas à un point d'entrée de l'application. Par ailleurs changer de fragment (plutôt que de changer directement de layout) pourrait faciliter l'implémentation d'une interface différente, pour tablette par exemple.

On notera que dans la structure actuelle de l'application, l'image actuellement éditée est contenue et manipulée depuis l'activité principale. Les fragments n'apportent à l'application que des éléments d'interface.

4.2 Classe Image :

Cette classe a été conçue comme une alternative à l'utilisation directe de la classe **Bitmap** fournie par Android.

Le cœur de la classe est évidemment une instance de Bitmap, qu'il est possible de récupérer à tout moment. Par ailleurs la classe offre des fonctionnalités supplémentaires, parmi celle-ci notamment la possibilité de restaurer l'image à son état au moment de sa création ou de son chargement, via la méthode **reset()**.

On notera la nécessité pour Image d'avoir la référence d'une Activité de l'application, en effet elle est requise à plusieurs moments par les librairies Android pour charger la Bitmap en mémoire.

4.2.1 Classe ImageInfo :

La classe Image ?? génère et garde une instance de la classe **ImageInfo**, cette classe contient un grand nombre de valeurs à propos de l'Image (dimensions, coordonnées GPS, date de prise de vue, ...).

L'idée de cette classe est d'empaqueter toutes ces informations afin de faciliter le passage de ces informations à l'avenir, notamment à travers des Fragments ou des Activités. On notera que tous les accesseurs appliquent des opérations de formatage sur ces données, certaines opérations pourraient être déplacées dans les constructeurs s'ils venaient à être utilisés régulièrement.

4.3 Package util :

Ce package contient de nombreuses classes contenant des méthodes statiques permettant une meilleure factorisation du code.

La classe **Utils** offre par exemple des méthodes pour récupérer la taille de l'écran ou pour calculer un ratio de redimensionnement.

La classe **BitmapIO** permet d'effectuer le chargement d'une Bitmap de plusieurs manières, depuis les resources ou un autre emplacement du téléphone, et avec la taille voulue.

5 Performances :

5.1 Temps d'exécution :

TODO

5.2 Mémoire :

TODO montrer l'ancienne instance d'image se faire garbage collecter ? voir remarques

TODO

6 Remarques et améliorations :

6.1 Remarques sur le code

TODO Remarques poids mémoire de la copie orginale des Images.

Lors du chargement d'une nouvelle image, nous ré-instancions un objet de la classe Image. Ce qui veut techniquement dire que jusqu'au prochain passage du ramasse miette Android, deux images sont en mémoires, donc deux Bitmap et deux tableaux de pixels (la copie originale des Images, voir ??). C'est un élément discutable cependant notre application limite la taille des Images chargées. Ce qui évite largement les dépassements mémoire.

6.2 Remarques sur les librairies Android

TODO Remarques sur l'utilisation obligatoire d'une activité contexte pour charger une Bitmap.

6.3 Améliorations à court terme :

TODO

7 Gestion du projet :

7.1 Organisation générale :

TODO

7.2 Avis personnels :

TODO

8 Conclusion :

TODO

9 Annexes :

Cahier des charges :

<https://dept-info.labri.fr/~vialard/ANDROID/cahierDesCharges.pdf>

Représentation de la couleur :

Système de couleur RGB :

https://fr.wikipedia.org/wiki/Rouge_vert_bleu

Système de couleur HSV ou HSB :

https://fr.wikipedia.org/wiki/Tinte_Saturation_Valeur

Documentation Android :

Activity :

<https://developer.android.com/reference/android/app/Activity>

Bitmap :

<https://developer.android.com/reference/android/graphics/Bitmap> <https://developer.android.com/topic/performance/graphics>

Gestion des dimensions des Bitmap :

<https://developer.android.com/topic/performance/graphics/load-bitmap>

RenderScript :

<https://developer.android.com/guide/topics/renderscript/compute>

Utilisations des API :

<https://developer.android.com/about/dashboards>