1er Rendu - Projet Technologique Android

Sarah CHOUROUQ, Tom DEPORTE, Vincent DUFAU, Firas MESSAOUD

Février 2020

1 Liste des fonctions implémentées

Les différentes fontions et algorithmes implémentés dans l'application sont :

- Le choix de l'image ou la prise de photo
- Le passage en niveaux de gris
- La colorisation
- La conservation
- La modification de la luminosité
- L'extension du dynamique
- La diminution du contraste
- L'égalisation d'histogramme
- Le filtre moyenneur
- Le filtre Gaussien
- Le filtre de Sobel
- Le filtre Laplacien
- La visualisation de l'histogramme sous forme de graphique
- L'algorithme RGBToHSV recodé
- L'algorithme HSVToRGB recodé
- Le zoom et le scroll
- La réinitialisation de l'image
- La sauvegarde de l'image

L'ensemble de ces fonctions peuvent être utilisées aussi bien sur une image en niveaux de gris que sur une image en couleur (l'effet n'est cependant pas garanti, une colorisation d'une image en niveaux de gris ne transformera pas l'image).

Pour ce qui est des fonctions de modification de l'image, elles sont disponibles grâce à un menu en haut à droite de l'application.

Ce menu propose l'accés à deux sous-menus différents : l'un concerne les fonctions dans leur version Java et l'autre les fonctions dans leur version Renderscript.

Pour obtenir la représentation de l'histogramme de l'image, un bouton permet de changer d'activité et d'observer le graphique (afin de générer ces graphiques, nous avons utilisé la librairie GraphView disponible **ici)**.

Nous avons également utilisé une autre librairie pour le choix de la couleur. Celle ci est disponible **à cet endroit**.

Le smartphone utilisé est un Samsung A50 qui a comme numéro de modèle **SM-A505FN**. La version d'Android utilisée tout est la version **9**.

2 Structure du code

L'application possède actuellement 7 classes dont 3 activités ainsi que 2 énumérations en plus.

Les 3 activités sont les classes les plus importantes du code. Elles gérent les différentes "vues" auxquelles l'utilisateur a accés.

La première activité est la classe LoadActivity, cette classe permet à l'utilisateur de choisir comment il veut récupérer l'image qu'il modifiera ensuite dans l'application. Pour cela, deux choix s'offrent à lui au travers de deux boutons. Il pourra soit récupérer une image qui est directement sauvegardé sur son téléphone ou alors prendre une photo à l'aide de son appareil dans le cas où les autorisations nécessaires sont accéptées par l'utilisateur, autrement, il n'aura pas accés à cette fonctionnalité. Finalement, lorsque l'utilisateur à fait son choix ou a pris une nouvelle photo, la classe LoadActivity laisse place à une nouvelle activité, la MainActivity.

La MainActivity est la plus grosse classe de l'application, elle gére l'affichage de l'image et elle permet d'utiliser tout les traitements d'images disponibles dans l'application au travers d'un menu. Elle gére également la sauvegarde de l'image après modifications et elle permet de passer à la dernière activité de l'application à l'aide d'un bouton : l'HistogramActivity.

L'HistogramActivity permet simplement d'afficher l'histogramme de l'image.

Nous avons ensuite 4 autres classes :

- BitmapSingleton
- PixelTransformation
- ConvolutionMatrix
- ScaleListener

La classe BitmapSingleton permet d'avoir un singleton représentant la bitmap de l'image utilisée dans l'application. Cette classe est utile afin de pouvoir utiliser la bitmap dans différentes classes et notamment dans les différentes activités de l'application.

La classe PixelTransformation est une classe composé d'un constructeur privé (car on ne veut pas pouvoir l'instancier) et de méthodes toutes statiques. Elle regroupe les méthodes permettant de transformer un pixel en un autre grâce à un calcul où les transformations de modèles de couleur. On va donc pouvoir retrouver dans cette classe la méthode permettant de transformer en pixel en niveaux de gris et les méthodes permettant de passer un pixel de rgb à hsv ou de hsv à rgb.

La classe ConvolutionMatrix est la classe responsable des traitements de convolutions. Elle va effectuer les calculs de convolution afin de retourner les valeurs transformées afin de changer l'image selon le masque de convolution lié au traitement choisi par l'utilisateur.

Enfin, la classe ScaleListener va seulement permettre de gérer le zoom à l'aide d'un objet ScaleGestureDetector.

Les deux énumérations apportent seulement plus de propreté au code et n'ont pas de réel impact sur l'application.

3 Fonctionnalités

Etant donné que depuis la création des groupes, nous nous sommes surtout penchés sur ce que personne n'avait encore fait lors du premier semestre, c'est-à-dire le chargement d'une image, la sauvegarde d'une image, le scroll, le zoom, nous avons décidé de ne pas faire de tests de performances comme pour le dernier rendu individuel sur ce rapport. La base du projet en groupe étant construite sur le code de Vincent DUFAU, il est possible de retrouver ces tests sur son dernier rapport individuel. Les fonctionnalités citées juste avant ne sont pas liés à des algorithmes et il est donc moins pertinent de faire des tests de performances sur celles-ci (la sauvegarde étant la seule fonctionnalité qui peut prendre un certain temps pour de grosses images). Nous ne ferons donc que la présentation et les résultats obtenus des fonctionnalités développées.

3.1 Le passage en niveaux de gris

3.1.1 Présentation

Le passage en niveaux de gris est un algorithme simple visant à transformer l'image dans une version entièrement en niveaux de gris. Pour ce traitement, 3 versions différentes sont disponibles en Java et une en Renderscript.

3.1.2 Résultats obtenus

La première version n'est pas utilisable. Les autres versions ont des temps d'exécution qui sont satisfaisants. L'utilisation de colorMatrix dans ce traite-

ment semble être la meilleure solution avec une durée d'exécution quasi nulle. Pour ce traitement, l'utilisation du Renderscript n'est pas nécessaire, même la seconde version semble avoir de meilleurs résultats.

3.2 La colorisation

3.2.1 Présentation

La colorisation consiste à transformer la teinte de chaque pixel de l'image pour correspondre à la teinte choisie par l'utilisateur grâce à un ColorPickerView disponible **ici**.

Pour ce traitement, une version est disponible en Java et une en Renderscript. Au niveau du code, il est également possible de réaliser ce traitement avec les fonctions implémentées par Android (RGBToHSV et HSVToColor) en décommentant les lignes correspondantes. Autrement, cet algorithme utilisera les fonctions recodées.

Cette fonctionnalité produira un effet seulement sur les images qui sont déjà en couleurs.

3.2.2 Résultats obtenus

Les fonctions implémentées dans Android Studio ne sont vraiment pas utilisables car bien trop longues. Ainsi, les fonctions recodées offrent des performances légèrement supérieures même si elles ne sont pas optimales. Le Renderscript, quant à lui, permet d'avoir des temps d'exécution toujours inférieurs à 1 seconde. De plus, plus l'image est grande, plus le temps d'exécution est long pour les versions Java du traitement contrairement à la version Renderscript.

3.3 La conservation

3.3.1 Présentation

La conservation est un traitement qui va permettre de mettre tout les pixels de l'image en niveaux de gris qui ne font pas partie d'un certain intervalle choisi par l'utilisateur.

L'utilisateur doit choisir une couleur, elle fera office de centre d'intervalle, puis il pourra rentrer une valeur correspondant à l'intervalle total à conserver.

Comme pour la colorisation, une version est disponible en Java et en Renderscript et il est possible de réaliser ce traitement avec les fonctions implémentées par Android.

Cette fonctionnalité produira également un effet seulement sur les images qui sont déjà en couleurs.

3.3.2 Résultats obtenus

De la même manière que pour la colorisation, les fonctions implémentées par Android Studio offrent les moins bonnes performances et l'utilisation du Renderscript semble être la réponse idéale à ce traitement d'image.

3.4 La modification de la luminosité

3.4.1 Présentation

Cette fonctionnalité permet de transformer l'image en changeant sa luminosité selon un facteur paramétrable par l'utilisateur.

Une seule version en Java est disponible pour ce traitement.

3.4.2 Résultats obtenus

La modification de la luminosité utilise une colorMatrix comme la troisième version du passage en niveaux de gris. Et comme ce dernier, elle est extrèmement performante.

3.4.3 Problèmes ou bug

Ce traitement semble s'effectuer sur l'image de base choisi par l'utilisateur, si entre temps, il a fait d'autres modifications et qu'ensuite il veut changer la luminosité, ces modifications sont ignorées et le traitement se fait sur l'image de base.

3.5 L'extension de dynamique

3.5.1 Présentation

L'extension de dynamique va transformer la valeur des pixels dans le cas où l'intervalle de valeurs concernant le V (value) du HSV n'est pas maximal. Cela aura pour effet d'améliorer le contraste de l'image. Ce traitement peut être efficace en particulier sur les images en niveaux de gris. Dans le cas des images en couleur, l'intervalle est en général maximal.

Une version est disponible en Java et une en Renderscript.

3.5.2 Résultats obtenus

Le Renderscript est toujours aussi performant avec des temps d'exécution aux alentours des 0.5 secondes.

3.6 La diminution du contraste

3.6.1 Présentation

Contrairement à l'extension de dynamique, le but de ce traitement est de diminué le contraste de l'image. Il faut donc réduire l'intervalle des V (value) de l'image.

L'utilisateur a le choix de la valeur de la diminution.

Une version est disponible en Java et une en Renderscript.

3.6.2 Résultats obtenus

La version Java a un temps d'exécution de plus de 2 minutes (avec le profiler, c'est plus rapide sans) pour une grande image. Ce n'est clairement pas satisfaisant pour une application de traitement d'image. Le Rendersctipt permet d'avoir encore une fois de bien meilleures performances.

3.7 L'égalisation d'histogramme

3.7.1 Présentation

L'égalisation d'histogramme permet tout comme l'extension de dynamique d'améliorer le contraste de l'image. Les résultats de ce traitement sont généralement bien plus visibles notamment sur des images en couleurs.

Une version est disponible en Java et une en Renderscript.

3.7.2 Résultats obtenus

Constat similaire à la diminution de contraste. Des temps d'exécutions très longs pour de grandes images comparé à une grande efficacité du Renderscript.

3.8 Le floutage

Deux traitements permettent de flouter l'image : le filtre moyenneur et le filtre Gaussien.

Ces traitements sont surtout efficaces pour des images de petites tailles, autrement les modifications sont quasiment invisibles.

3.8.1 Le filtre moyenneur

Le filtre moyenneur est un traitement de convolution. Chaque pixel va se transformer selon la moyenne des pixels l'entourant et selon la taille du noyau utilisé. Plus la taille du noyau est grande, plus l'effet sera observable mais le temps d'exécution sera plus long. L'utilisateur a le choix de la taille de ce noyau et il est contraint de rentrer une valeur impaire.

La seule version disponible est en Java.

Pour ce traitement, il n'y a pas de quoi faire de comparaison cependant au vu des résultats précédents, il est facile de penser que le Renderscript fournira de meilleurs temps d'exécution que les foncions codées en Java. On peut néanmoins observer que le temps d'exécution dépend à la fois de la taille de l'image et de la taille du noyau.

3.8.2 Le filtre Gaussien

Deux noyaux de Gauss sont disponibles : 3x3 et 5x5, plus le noyau est gros, plus l'effet est visible.

La seule version disponible est en Java.

La méthode utilisée est la même que pour le filtre moyenneur donc les observations sont les mêmes.

3.9 La détection de contours

Deux traitements sont disponibles pour la détection de contours : le filtre de Sobel et le filtre Laplacien.

Avant d'effectuer ces traitements, on transforme dans un premier temps l'image en niveaux de gris. La couleur ici n'est pas utile car ces algorithmes permettent de mettre en évidence en blanc les contours de l'image et le reste sera noir.

3.9.1 Le filtre de Sobel

Ici, on applique deux filtres de convolution permettant d'obtenir les gradients (dérivées horizontale et verticale de chaque point) des pixels et de faire apparaître les contours présents dans l'image.

La seule version disponible est en Java.

Comme pour le filtre moyenneur, l'utilisation du Renderscript nous fournira surement de meilleurs résultats.

3.9.2 Le filtre Laplacien

Contrairement au filtre de Sobel, un seul noyau est utilisé pour le filtre Laplacien.

La seule version disponible est en Java.

Toujours le même constat, en attente du Renderscript.

3.10 La visualisation de l'histogramme sous forme de graphique

En cliquant sur le bouton "Obtenir l'histogramme", l'application change de page pour arriver sur un graphique mettant en évidence les courbes représentant l'histogramme de l'image. Dans le cas d'une image en couleurs, 3 courbes seront visibles (une par canaux rouge, vert et bleu). Dans le cas d'une image en niveaux de gris, ces 3 courbes se superposent et seule une courbe bleue est visible.

4 Pour le prochain rendu

Pour le rendu final, nous voulons développer les différents algorithmes de convolution présents dans l'application en renderscript. Cela nous permettra de faire des comparaisons avec le code Java et d'avoir (surement) des meilleures performances pour l'application.

Une fonctionnalité non décrite dans ce rapport mais qui a été commencée est l'incrustation d'objet.

Autrement, nous ne savons pas encore vraiment quelles fonctionnalités nous allons ajouter à notre application, il y a des idées dans le cahier des charges mais

peut-être que d'autres idées nous viendront. Nous préférons d'abord terminer au mieux ce qui a été commencé et pas complété (comme le renderscript pour la convolution) avant de partir sur de nouvelles idées.