Traitement d’image sous Android
Cahier des charges partiel

Boris Mansencal, Anne Vialard

1 Introduction

Ce document contient une description du travail à rendre dans le cadre du projet de traitement d’image sous Android en 3ème année de licence. Il est volontairement incomplet, une partie du travail consistant à le compléter.

Le projet est à faire en binôme, vous devrez donc découper le projet en tâches afin de vous les partager.

2 Description

Le but de ce projet est de développer une application de traitement d’image sur smartphone avec système Android. Les images peuvent aussi bien être obtenues depuis la galerie du téléphone que directement depuis la caméra.

Le projet est découpé en deux parties, et deux releases du logiciel seront donc à rendre. La première release est commune à tous les groupes. Elle est composée des fonctionnalités de base afin de gérer, afficher et sauvegarder les images. De plus seront intégrés quelques traitements d’image basés sur une transformation d’histogramme ou une convolution. Avec cette première release, chaque groupe rendra un cahier des charges complété, avec un ensemble de fonctionnalités additionnelles choisies parmi une liste de choix.

3 Besoins fonctionnels

Les besoins fonctionnels sont donc découpés en deux parties : une liste de besoin fonctionnels obligatoires qui forment le contenu de la première release, et une liste de besoins fonctionnels au choix, qui compléteront la première release afin de former la seconde release : le rendu final.

3.1 Obligatoires

1-Charger une image Le logiciel doit permettre d’obtenir une image de plusieurs manières :

(a)**depuis la galerie** Le logiciel doit permettre de sélectionner une image parmi celles présentes dans la galerie du téléphone :

On crée un Intent « gallery » paramétré sur ACTION_GET_CONTENT afin de récuperer les images de la gallerie. On appelle ensuite startActivityForResult qui récupère le résultat (l'image) via onActivityResult.

(b) **depuis la caméra** Le logiciel doit permettre de capturer une image depuis la ou les caméra(s) du téléphone :

On crée un Intent « gallery » paramétré sur ACTION_IMAGE_CAPTURE afin de prendre une photo depuis la caméra du téléphone. On appelle ensuite startActivityForResult qui récupère le résultat (l'image) via onActivityResult.

2-Afficher une image Une fois une image chargée, le logiciel doit l'afficher sur l'écran du terminal :

A l'ouverture de l'application, une image de base est affichée. Une fois une nouvelle image chargée (depuis la gallerie ou la caméra), on change l'ImageView grâce au setter, et on réinitialise la bitmap de l'image.

3-Zoomer Lorsqu'une image est affichée dans le logiciel, il doit être possible de zoomer et dézoomer, en utilisant l'interaction avec deux doigts :

On distingue plusieurs cas :

- lorsqu'un doigt touche l'écran : l'utilisateur peut déplacer l'image jusqu'à ce que le doigt soit retiré.

- lorsque deux doigts touchent l'écran : l'utilisateur peut zoomer ou dézoomer sur l'image suivant que les doigts s'éloignent ou se rapprochent. Le zoom se centre à mi-distance des deux doigts.

4-Scroller Lorsqu'une image est affichée et déborde de l'écran, il doit être possible de déplacer la zone affichée à l'aide d'une interaction avec un doigt :

Voir le premier cas du zoom.

5-Appliquer des filtres Le logiciel doit permettre d'appliquer quelques filtrages usuels sur les images chargées.

(a) Régler la luminosité Lorsqu'une image est affichée il doit être possible d'en régler la luminosité :

On affiche une SeekBar qui permet d'augmenter ou de diminuer la valeur de la luminosité. En effet, on récupère les valeurs RGB de chaque pixel, puis on ajoute (ou retire si la valeur est négative) la valeur retournée par la SeekBar. Si la valeur des couleurs dépasse les bornes [0, 255], on les fixe aux limites des bornes.

(b) Régler le contraste Lorsqu'une image est affichée il doit être possible d'en régler le contraste :

On affiche une SeekBar qui permet d'augmenter ou de diminuer la valeur du contraste. On applique un facteur à chaque composante RGB puis on les fixe aux limites des bornes si elles dépassent de [0,255].

(c) Égalisation d'histogramme Le logiciel doit permettre d'égaliser l'histogramme de l'image affichée :

On calcule l'histogramme de chaque couleur de l'image (rouge, vert et bleu). On calcule ensuite l'histogramme cumulé de chaque composante grâce à l'histogramme précédemment calculé. On modifie alors la valeur de chaque pixel en fonction de l'histogramme cumulé.

(d) Filtrage couleur Le logiciel doit permettre de mettre en œuvre les traitements couleur vus en TP, à savoir modification de la teinte (niveaux de gris, sépia, ...) ainsi que la sélection d'une teinte à conserver lors du passage en niveaux de gris :

- Niveaux de gris : On calcule un niveau de gris en fonction des composantes RGB puis on modifie la valeur de chaque pixel en remplaçant chaque composante par le niveau de gris calculé.

- Sépia : On calcule un nouveau rouge, un nouveau vert et un nouveau bleu pour chaque pixel à partir des composantes RGB initiales. Puis, on fixe aux bornes ces nouvelles composantes si nécessaire avant de remplacer les composantes RGB de chaque pixel par les nouvelles calculées.

- Coloriser une image : On récupère la teinte de chaque pixel (via tableau HSV) puis on la remplace par la teinte passée en paramètre.

- Sélection d'une teinte : On récupère la teinte de chaque pixel (via tableau HSV). On calcule un intervalle de teinte grâce à la teinte passée en paramètre. Si la teinte du pixel n'est pas comprise dans l'intervalle calculé, alors on le passe en niveau de gris. Sinon, on garde le pixel tel quel.

(e) Convolution Le logiciel doit permettre d'appliquer différents filtres basés sur une convolution sur l'image affichée. Les filtres moyenneur, Gaussien, Sobel et Laplacien seront implémentés.

Dans le cas des 4 filtres de convolution (moyenneur, gaussien, sobel et laplacien), une fonction apliquant un masque carré impair quelconque sur une liste de pixel est utilisé afin de factoriser le code.

Les bords de l'image sont traités par « extension de l'image d'origine ». Une bitmap plus grande est créée. On remplit le centre de cette bitmap par notre btmap courante et on remplit les bords en dupliquant le pixel le plus proche sur chaque ligne contenue dans les bords. On peut ensuite appliquer notre convolution seulement sur le centre de cette bitmap et récupérer dans notre bitmap courrante la transformation des pixels.

- Filtre moyenneur : Le masque utilisé est une matrice dont les coefficients sont tous égaux à $1 / n^2$ où n le nombre de lignes (ou de colonnes) du masque. Le

rendu final donne une image floutée dont chaque pixel est la moyenne des valeurs de ses pixels voisins de l'image initiale.

- Filtre Gaussien : Le masque utilisé est une matrice dont les coefficients représentent les valeurs d'une fonction gaussienne. L'écart type de la fonction dépend de la taille du masque. Le rendu donne une image floutée de meilleure qualité que le flou moyenneur.

- Filtre Sobel : Le filtre de sobel permet de faire ressortir les contours de l'image. Pour cela, on utilise 2 masques (un pour les X et l'autre pour les Y), résultant chacun d'un produit entre une différentielle première et un moyenneur.

- Filtre Laplacien : Le filtre laplacien permet de faire ressortir de manière plus détaillée les contours de l'image que sobel. Pour cela, on utilise un masque représentant une différentielle seconde. Ici aussi, on fixe les composantes de niveaux de gris aux limites des bornes si elles dépassent de [0,255].

> **6-Réinitialiser** Le logiciel doit permettre de réinitialiser l'image c'est-à-dire d'annuler les effets appliqués depuis le chargement de l'image.

Chaque image est une variable de la classe FilteredImage, où sont stockées deux bitmaps : l'une où seront faites les modifictaions, et l'autre correspondant à l'image non traitée. Lors de la réinitialisation, on copie tout simplement la bitmap non traitée dans la bitmap courante.

> **7-Sauvegarder une image** Le logiciel doit permettre de sauvegarder une image modifiée

On crée un fichier avec l'adresse de la carte SD du téléphone et un nom d'image. On crée un flux pour écrire dans ce fichier. On compresse ensuite l'image dans ce flux et on l'ajoute à la gallerie du téléphone via la fonction «insertImage». Pour récupérer la permission de stockage accordée ou non par l'utilisateur, on affiche une pop-up.

3.2 Au choix

> **8-Simuler un effet cartoon** Le logiciel devra restreindre les couleurs d'une image et renforcer les contours afin de lui donner un effet de bande dessinée

Afin d'appliquer l'effet cartoon, nous avons dans un premier temps du restreindre les couleurs : pour cela nous avons créé une nouvelle classe *ColorCube*. Cette dernière permet une représentation en 3 dimensions de l'espace des couleurs, en fonction des axes Rouge, Vert et Bleu. Dans cette classe, qui prend en paramètre notamment une liste de pixels, nous avons ensuite ajouté l'algorithme de K-mean, ou de k-cluster.

Le principe est de créer k points aléatoire dans le cube minimal des couleurs de l'image (ici, nous les appellerons centre des cluster), puis comparer les distances de chaque pixels avec ces centres. Les pixels seront ensuite stockés dans le cluster correspondant au centre le plus proche. Les centres sont alors

modifiés, pour valoir la moyenne des couleurs de chaques pixels dans son cluster correspondant. On réitaire alors l'opération un nombre n de fois fixé.

A la fin, on applique à chaque pixels la couleur du centre de son cluster, et on obtient une image avec seulement k couleurs.

Pour obtenir les contours, nous les avons récupéré grâce au filtre sobel appliqué sur l'image après un flou gaussien.

> **9-Comparer avec l'image de base** Le logiciel devra permettre à l'utilisateur de comparer une image traitée avec l'image initiale sans avoir à la réinitialiser.

Tant que l'utilisateur reste appuyer sur le bouton « compare », l'image d'origne remplace l'image courante.

> **10-Revenir en arrière** Le logiciel devra permettre à l'utilisateur d'annuler le dernier filtre appliqué. Cette action peut être réitéré jusqu'à revenir à l'image originale.

Pour cela, nous avons rajouter dans la classe *FilteredImage* une liste de Bitmap, qui permet de stocker les bitmap modifiées par les différents filtres choisis par l'utilisateur.

> **11-Inversion des couleurs**

L'inversion des couleurs se fait en soustrayant à 255 la valeur du pixel pour chaque canal rouge, vert et bleu.

En plus de ces fonctionnalités optionnelles, nous avons déjà pu personnaliser notre application, via la modification de l'icone, le nom de l'application et les couleurs du thème.

4 Besoins non fonctionnels

4.1 Obligatoire

> **1.Gestion de version** Le développement devra utiliser l'outil de gestion de version git.

4.2 Au choix

> **2.Optimiser le code** Le logiciel devra être fluide. On pourra optimiser l'implémentation des algorithmes. On pourra aussi chercher à utiliser plusieurs threads et/ou utiliser un langage plus efficace tel renderscript. Il faudra mesurer les gains obtenus par ces changements.

Concernant l'optimisation du code, nous avons optimisé les algorithmes de convolution (filtre moyenneur, Gaussien, Laplacien et Sobel) en écrivant une fontion « Convolution ». Celle-ci permet d'appliquer un filtre carré quelconque sur un pixel donné.

Dans la gestion des bords, nous avons créé une fonction qui retourne une bitmap plus grande remplit comme expliqué précédemment (le centre par l'image que l'on traite, et les bords par duplication du pixel le plus proche).

Concernant l'optimisation du temps d'execution des algorithmes, nous avons créé trois fichiers RenderScript pour les fonctions « toGray », « sepia » et « invert ». Afin de vérifier que ces fonctions en RenderScript sont bel et bien plus rapides, nous avons utilisé Android Profiler. Nous avons comparé le temps d'execution (**en microsecondes**) de la fonction « onClick » avec et sans les fonctions RenderScript et nous avons obtenu les résulats suivants :

	toGray	Sepia	Invert
Without RenderScript	5872	17006	4235
With RenderScript	**2797**	**3513**	**2752**

Nous observons que le RenderScript nous permet de gagner du temps dans les trois cas.

Nous avons testé notre application sur différentes versions d'Android. Nous avons travaillé pendant le semestre sur la version 7.0 de nos téléphones. Puis nous avons testé sur deux autres téléphones :

- Le RenderScript ne fonctionne pas sur la version 4.2.2

- L'application tourne correctement sur la version 5.1.1

5 Architecture

Vous devrez réfléchir à une architecture logicielle permettant de maintenir efficacement votre application, et d'ajouter des fonctionnalités supplémentaires sans duplication de code.

Vous devrez à minima coder votre propre classe image, votre propre viewer d'image ainsi qu'une interface pour les filtres.

Nous avons crée plusieurs classes. La première et la plus importante, est la classe *FilteredImage*, qui comprend une ImageView, deux Bitmap, et les dimensions de ces dernières, ainsi qu'une LikedList de Bitmap. Cette classe permet d'appliquer les différents filtres à l'image choisie, de la réinitialiser et d'annuler le dernier filtre.

La classe suivante est la classe *PopUp*, dont 2 autres classes : *PopUpColor* et *PopUpConvolution* héritent. Ces deux dernière permettent d'afficher une pop up pour choisir des paramètres : soit la couleur à appliquer,

soit la taille du masque de convolution. Ce sont des activités liées à leur layout respectif.

Comme expliqué précedemment, la classe *ColorCube* permet une représentation en 3 dimensions de l'espace des couleurs, en fonction des axes Rouge, Vert et Bleu. Cette classe est uniquement utilisée pour l'algorithme cartoon .

La dernière classe est *MyTask*. Elle hérite de *AsyncTask*, qui permet de lancer des tâches en arrière plan. Ici, lors de l'execution des différents filtres, *MyTask* affiche une progress bar.

L'interface *AsyncResponse* nous permet de récupérer les informations de *MyTask* quand l'application des filtres se termine .

6 Calendrier

Le projet démarre le 22 Janvier, date de remise du sujet, et dure 3 mois. Lors de ce projet, vous devez faire deux rendus obligatoires.

Pour ces rendus, vous devez envoyer le lien de votre projet git aux chargés de TD par e-mail. Les documents en format pdf (cahier des charges, rapport) peuvent être mis sur le git et/ou envoyés par e-mail s'ils ne sont pas trop volumineux.

L'heure maximale de rendu est 23h59.

6.1 1er rendu - 7 Mars

Le premier rendu comporte une première release du logiciel ainsi que le cahier des charges complété.

La première release doit comporter tous les besoins fonctionnels de la section 3.1, mais cela ne doit pas vous empêcher de commencer le développement des fonctionnalités supplémentaires.

En vous basant sur l'expérience acquise lors du développement de cette première release, vous devrez compléter (et rendre) le cahier des charges. Cette nouvelle version contiendra la description des besoins complétés, l'architecture logicielle retenue détaillée et expliquée, ainsi que vos choix de fonctionnalités supplémentaires qui vous semblent réalisables dans le temps restant.

6.2 Rendu final - 23 Avril

Le rendu final contiendra votre logiciel terminé avec toutes les fonctionnalités que vous aviez choisies. Vous devez aussi fournir une série de tests garantissant le bon fonctionnement de votre logiciel, ainsi qu'une évaluation de ses performances en temps et en mémoire.