Rapport De Groupe Intermdiaire

Projet Tech Android

Théo Dupont, Sebastian Pagès, Hugo Bergon, Elliot Renel

28 Février 2020

Table des matières

1 Structuration du Code

La structure globale de notre projet.

Notre application MainActivity.java manipule essentiellement une classe nommée

BitmapPlus, qui a la particularité de posséder deux bitmaps en son sein, ainsi que la PhotoView, seul place ou sera affichée la bitmap. Ainsi on manipule directement la classe BitmapPlus qui s'occupe ensuite de modifier et d'afficher la bitmap désirée en les fonctions de la classe BasicFilter appellant des fonctions getPixels, setPixels.... adaptées à la structure de BitmapPlus (et non plus une bitmap simple). Ces fonctions se trouvent dans le package com.example.editionimage.imagehandling

Nous avons également utilisé une nouvelle structure pour l'affichage de notre image. Nous avons donc remplacé l'ImageView par une PhotoView, qui possède comme caractéristiques les mêmes que ImageView, mais auquelles on peut rajouter la faculté de scroll et de zoom de cette image.
Lien : https ://github.com/chrisbanes/PhotoView

Kernel implémente une matrice de taille arbitraire qui sera utilisé pour n'importe quelle convolution. Lors d'un appel d'une fonction convolution, on peut alors créer la matrice représentant notre effet avec un Kernel.

MainActivity s'occupe également de la visibilité de certains boutons comme pour le choix de la couleur à garder, ou du degré de luminosité à modifier.

Cette classe manipule donc les objets produits par le fichier xml activity_main.xml. Celui-ci est composé d'un layout qui prend la taille de l'écran, les boutons d'accès au choix de la photo, la PhotoView, et enfin, un scrollView.
Ce scrollView contient toutes les fonctions de modification d'image, ainsi que d'autres scrolls, définis pour des modifications avec un choix, comme KeepColor, Colorize, et setLightlevel. Ces seconds scrollView sont composés d'une seekBar pour le choix de la valeur a appliquer, ainsi que d'un bouton Apply pour confirmer son choix.
Toutes ces fonctions, s'il n'y a pas d'image de chargée, réalisent une erreur, d'ou notre classe ToasterNoImage, et sa fonction isToastShowed, qui teste si l'image est accessible.

2 Choix techniques

Premièrement, nous allons parler du choix de regrouper deux bitmaps en une seule dans BitmapPlus.
Ceci permet de ne pas se tromper quand on appelle une fonction appellant la bitmap, et permet une bien meilleur lisibilité dans les différentes classes.

La classe Kernel permet de manipuler plus simplement et plus visuellement les matrices que l'on veut appliquer. Ainsi on peut facilement appliquer une matrice dans une fonction plusieurs fois si nécessaire.

Nous avons également utilisé des fonctions de récupération/dépot de tableaux en hsv directement, pour permettre une plus grande fluidité.
Les fonctions de modification de contraste et de luminosité manipulent les canaux RGB car plus facile a optimiser et manipuler, que le hsv.

3 Besoins fonctionnels

3.1 Chargement et sauvegarde d'image

L'application charge correctement une image depuis la galerie, et permet également la prise de photo pour les importer dans l'application, avec les boutons situés en haut de l'application.
L'application peut également sauvegarder, à condition d'autoriser manuellement l'application à enregistrer des photos.

3.2 Image et manipulation

L'image chargée est donc maintenant visible à l'écran, prête à être modifiée, et on peut la zoomer avec l'action des deux doigts, et scroller si elle est trop grosse par rapport à la taille définie dans l'application.

3.3 Filtres

Les fonctions ci-dessous sont retrouvées dans la fonction BasicFilter, et testables via les boutons dans le bas de l'application.

— toGray() :
Transforme l'image appellante en niveau de gris.
L'algorithme prend juste les niveaux de rouge, vert et bleu de l'image, en fait une moyenne pondérée avec les facteurs donnés en cours (0.3/0.59/0.11)

et met la moyenne comme niveau de gris dans le pixel.
La fonction présentée dans l'application est la version renderscript, qui reprend le même principe.

— colorize(int color) :
Transforme l'image appellante pour que chaque pixel aie une teinte donné en paramètre. La méthode modifie simplement la teinte en passant les pixels en Hsv, puis en modifiant le paramètre h par celui donné en paramètre, pour repasser les pixels en rgb.

— keepColor(int color, int range) :
Transforme l'image appellante pour ne garder la couleur que de certains pixels.
La methode calcule grâce aux paramètres "color" (donné dans MainActivity par une seekBar) et "range" (ici arbitraire, avec une valeur de 30 dans BitmapPlus) un intervalle de couleurs en degrés (couleur HSV) qui sont à garder. Puis dans le parcours des pixels de l'image il met la saturation du pixel à 0 si sa couleur n'est pas dans l'intervalle.
A noter que si l'intervalle est discontinu (couleur proche de 0 ou 360), la variable booléen "inRange" est calculée différement.

— contrastLinear() : Transformation de contraste de manière linéaire de la bitmap appellante.
Après avoir récupéré l'histogramme de l'image (histogramme de V*100 de HSV), on trouve les indices minimal et maximal non nuls dans l'histogramme, afin d'appliquer la formule Range*(Valeur-Min)/(Max-Min) où Valeur est la valeur du pixel en cours (V*100 pour Color) et Range est la taille des valeurs possible -1. Pour l'histogramme la valeur V utilisée pour l'indice est multiplié par 100 pour pouvoir avoir des indices de tableau déscents, puis redivisé par 100 une fois récupéré.

— contrastEqual() : Transformation de contraste par égalisation d'histogramme du bitmapPlus appellant.
Après avoir récupéré l'histogramme de l'image (de la même manière que le contraste linéaire) et calculé l'histogramme cumulé C[], on calcule pour chaque pixel la valeur C[Value]*Range/Size où Value et Range ont le même

rôle que dans le contraste linéaire et Size est le nombre de pixels de l'image. La valeur de V pour l'histogramme est aussi multiplié par 100 pour les mêmes raison que ci-dessus, puis de-même redivisé par 100.
En RS : La fonction d'égalisation d'histogramme renderscript convertit l'image au format YUV qui sépare le luma des couleurs (chrominance) pour ne traiter qu'un seul canal.

— modifLight(double alpha) : Modification de la luminosité de l'image appellante. La fonction ajoute/réduit jusqu'a une valeur de 80 a chaque canal rouge, vert et bleu simultanément pour faire varier la luminosité de chaque pixel.

— modifLight(double alpha) : Modification du contraste de l'image appellante. Cette fonction utilise la valeur donnée pour augmenter ou réduire les canaux du pixel en conséquence, par rapport à ses canaux de base. Ainsi les canaux élevés seront augmentés alors que ceux qui sont faibles seront plus proches de 0.

3.4 Convolutions

Toutes les fonctions de convolution présentées ici sont appliquées de la même manière. On crée un Kernel avec les données que l'on veut rentrer (la matrice du flou gaussien par exemple.) On donne ensuite ce noyau a applyMask qui donnera les nouvelles valeurs après passage de la matrice sur l'image. Puis on actualise notre bitmap courante.

— GaussianBlur() : Gaussian est une application de flou gaussien avec une matrice de Gauss de dimension 5*5 comme ci-dessous.

$$\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix} \tag{1}$$

— simpleEdgeDetection()(Laplace) : simpleEdgeDetection est une application de la détection de bordure de Laplace avec une distribution gaussienne,

Laplace étant utilisé pour la détection de bords.

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix} \tag{2}$$

— Sobel() : Sobel est un autre moyen de réaliser la détection des bords. Il utilise ces deux matrices 3x3 ci-dessous, avec la première qui réalisera la détection de bord verticaux et la deuxième ceux horizontaux (cela simule l'effet d'une grosse matrice d'une manière plus efficace) :

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{4}$$

4 Problèmes connus

— Les images sont sauvegardées sur le téléphone dans un dossier dédié dans le dossier de base des photos, mais ce dossier n'est pas visible depuis la gallerie.

— Les fonctions de conversion hsv renderscript ne fonctionnent pas (l'image n'est pas celle attendue). Tous les fichiers renderscript sont dans le dossier ./projet/src/rs/ mais seuls gray.rs et histEq.rs sont appellés.