université de BORDEAUX

RAPPORT

PROJET TECHNOLOGIQUE

ANDROID « TRAITEMENT D'IMAGE »

Résponsable : **Anne Vialard & Fabien Baldacci & Boris Mansencal**

Membre du groupes :

Noms	Prénoms
Diallo	**Ibrahima**
Diallo	**Mamadou Mouctar**
Fall	**El hadji Pathé**
Houchat	**Kamel**

Présentation

Description et justification de l'organisation du code

Nous avons regroupé notre projet sous la model MVC (Model View Controller). Qui est le standard adopté pour mieux structurer son code.

- Le modèle :

Constitue la partie qui regroupe l'ensemble des données de notre projet (généralement la base de donnée). Dans notre cas notre modèle constitue les différentes classes de fonctions que nous avions implémenté au niveau du premier semestre, qu'on utilisera au niveau de notre contrôleur, telle que :

- AuxiliaryFunction : qui contient les fonction auxiliaires
- Conversion
- Convolution
- DynamicExtension
- Equalization
- Filter

- Le contrôleur :

Constitue la partie qui fait le lien entre le Model et le View; il interprète les données du modèle demandées dans le View selon l'appel qu'on en fait dans le contrôleur. Donc c'est la partie qui définir le comportement
de notre modèle. Ainsi, nous avons :

- MainActivity : qui affiche notre premier layout de l'application avec le choix de prendre directement une image depuis la galerie ou de le prendre directement avec l'appareil photo du téléphone.
- StudioActivity : qui constitue la page ou on fait les manipulations sur l'image grâce au données de notre modèle.

- Le package fragment qui contient : Studio_fragment et Plus_fragment, dont le premier est le fragment qui gère StudioActivity et la deuxième pour les options supplémentaires et paramètres ajoutés.

- Le View :

Généralement du code code xml comme dans java (ou javafx) et android. Il permet de définir des vue pour l'utilisateur de façon à communiquer avec le modèle selon les différentes requêtes émis par l'utilisateur. Dans notre cas vue que dans android studio y'a le package layout contenue dans le dossier ressources qui regroupe l'ensemble des fichiers xml dédié à notre vue, nous le considérons directement sans faire de nouveau package. De ce fait tous les activités liées à la vue sont créés à partir de ce package.

Liste et Rôles des Layouts:

fragment_studio: est le layout qui nous permet de gérer l'interface d'implémentation de nos différents fonctions. Il contient ainsi les différents filtres appliqués à l'image.

fragment_plus: est le layout qui nous permet de gérer l'interface d'implémentation de nos différents paramètres appliqués à l'image et les informations portants sur l'image. Ainsi ;

- l'image en petite icône
- la largeur de l'image
- la hauteur de l'image
- Histog: pour afficher l'histogramme de l'image
- RS bouton d'option: pour appliquer les filtres en java ou en renderScript
- Br bouton d'option: pour appliquer les la luminosité en HSV ou RGB
- GITHUB bouton: qui donne le lien sur le dépôt github du projet.

add_text_dialog_fragment: contient le texte qui nous permet d'ajouter du texte sur notre image.

editing_tool_row: contient les boutons des différents fonctions et filtres qu'on applique sur notre image depuis l'interface de notre fragment studio.

Description des fonctionnalités disponibles:

Notre application implémente toutes les fonctionnalités de base décrite dans le cahier de charge à savoir :

1. Charger une image : Cette partie consiste à récupérer une image soit à partir de la galerie ou de la caméra
    a. depuis la galerie l'application : Vu que pour accéder dans la galerie d'un appareil on a besoin que l'utilisateur nous donne le droit d'accès alors lors du lancement de l'application on demande l'autorisation à l'utilisateur pour pouvoir manipuler c'est image,
    b. depuis la caméra l'application : Comme pour le chargement des images depuis la galerie, nous demandons l'accès aussi pour la caméra.
    Le principe de chargement d'image que nous avons utilisé a partir de la caméra est :
        - Prendre un photo sur la caméra ensuite l'enregistrer au niveau de la galerie du téléphone.
        - Récupérer l'image que nous venons d'enregistrer dans la galerie pour ne pas perdre aucun pixel
2. Afficher une image Une fois une image chargée
    Comme le titre l'indique a partir de l'image chargée (galerie ou caméra) nous affectons à une Photoview(Explication au niveau de Scroller)
3. Zoomer
    Permet la possibilité de zoomer et dézoomer une image déjà chargée
4. Scroller
    Pour scroller l'image nous avons utilisé un code déjà implémenté qui devait juste être intégré dans notre projet pour cela au lieu d'utiliser une ImageView nous l'avons remplacé par PhotoView décrit dans l'utilisation du code que nous avons intégré.
5. Appliquer des filtres
    A ce niveau nous utilisons les fonctions déjà implémentées au premier semestre pour appliquer les différents filtres l'image que nous voulons modifier.
    a. Régler le contraste
    b. Égalisation d'histogramme
    Nous utilisons pour l'égalisation d'histogramme les fonctions se trouvant dans le model Equalization
    c. Régler la luminosité
    d. Filtrage couleur
    e. Convolution

Pour la convolution juste le filtre moyenneur a été implémenté pour l'instant

6. Réinitialiser
   Avec cette partie lorsque nous appliquons des effets que nous plaise plus nous pouvons réinitialiser l'image c'est-à-dire d'annuler les effets appliqués depuis le chargement de l'image.
7. Sauvegarder une image :
   La sauvegarde nous permettra d'enregistrer une image après toutes modification.
8. Recadrer l'image
   Permet de d'utiliser l'option Zoom jusqu'à obtenir un cadre de l'image, sélectionné qui nous convient et appliquer le recadrage afin de garder sur l'image que la partie sélectionnée.
9. Rotation
   Permet de pivoter l'image selon votre convenance.

10. Luminosité
    Permet d'appliquer sur l'image une diminution ou une augmentation de luminosité.
11. Saturation
    Permet d'appliquer sur l'image une diminution ou une augmentation de la saturation.
12. Filtre
    Permet d'appliquer les différents filtres des fonctions demandées en premier semestre.

Description des fonctionnalités disponibles

Controller :

HistogramActivity -> la classe nous permet de visualiser l'histogramme des niveaux de gris des trois composants (Color.red, Color.green, Color.blue) des images. Elle contient les trois fonctions suivantes:

- ★ **onCreate()** -> qui permet de récupérer l'image et de l'instancier. Et de définir les différentes composants du graphe.
- ★ **histoOfThreeColors()** -> qui crée l'histogramme de l'image instancié par onCreate().

- ★ **setBarEntry()** -> renseigne les données de l'histogramme sur les trois courbes du graphe définit.

MainActivity -> C'est la classe de l'activité de base, elle gère:

- ★ Les animations de lancement de l'application.
- ★ Octroi les autorisations d'accès.
- ★ Le chargement de l'image depuis la galerie ou la caméra.

Elle contient les méthodes suivantes:

- ★ initView() -> La méthode qui initialise toutes les vues.
- ★ **onActivityResult()** ->qui comporte deux deux conditions: l'une qui récupère l'image une fois les permissions admises à partir du caméra et l'autre depuis la galerie.
- ★ **loadImage()** -> Méthode utilisant la bibliothèque matisse pour charger l'image depuis la galerie.
- ★ **takeImage()** -> Méthode qui ouvre la caméra et prend une nouvelle photo.
- ★ **isGranted ()** -> Méthode booléenne qui vérifie si les autorisations sont accordées.
- ★ **checkPermission()** -> Méthode qui demande à l'utilisateur d'accorder des autorisations d'écriture et de lecture sur le stockage du téléphone s'il n'ont pas été données.
- ★ **onRequestPermissionsResult()** -> qui affiche la bannière d'information pour avertir l'utilisateur que les permissions ont été acceptées ou non et lui demande à nouveau si ces dernières n'ont pas été accordées.

StudioActivity -> C'est la classe qui gère :

- ★ Le chargement de l'image envoyée par l'activité principale.
- ★ Le chargement de la barre de navigation.
- ★ La gestion de la barre de navigation avec ViewPager.
- ★ Masquer les barres des outils de façon automatique.

Elle contient:

- ★ **MyPagerAdapter** -> c'est la classe adaptateur du pager de vue.
- ★ **getContextOfApplication()** -> Méthode statique qui renvoie le contexte de l'activité.
- ★ **hideSystemBars()** -> Méthode qui masque les barres de notre application.

- **adapters** :

EditingRecyclerViewAdapter -> la classe est un adaptateur qui nous permet de générer tous les outils nécessaires à la gestion des images en spécifiant l'icône et le nom de l'outil, il permet de les classer côte à côte et de faire appel à eux à chaque fois qu'on veut modifier une image.

La classe contient des méthode comme :

- ❖ **getItemCount** : qui permet de retourner la liste des différentes méthodes contenues dans la liste de EditingRecyclerViewAdapter.
- ❖ **ViewHolder**: qui permet d'extraire les éléments de la mise en page exemplaire.

FilterRecyclerViewAdapter -> la classe est un adaptateur du FilterRecyclerView,il nous permet de générer tous les filtres qui seront appliqués aux images en précisant le nom et le type du filtre. Elle nous permet aussi de les classer côte à côte et de créer plus de copie à partir d'une copie de filtre.
Son constructeur, nous ajoutons tous les filtres qui seront appliqués aux images et que nous voulons voir dans la listes de Filtre du R FilterRecyclerViewAdapter.
Elle a les mêmes constructeurs comme **getItemCount** et **ViewHolder**, en plus d'une méthode sur laquelle **chargeImage** : qui désigne une nouvelle image sur laquelle le filtre a été appliqué.

- **dialogFragment** :

TextDialogFragment -> la classe nous permet d'ajouter un nouveau fragment dont nous appliquons le texte que nous voulons ajouter sur l'image.

- **Fragment :**

Plus_fragment -> Plus fragment est une classe qui s'étend de Fragment, elle est utilisée pour la gestion et l'affichage de la mise en page. Elle implémente la classe OnClickListener et contient, pour faire appel au fonction appliquer à l'image quand on appuie sur leur bouton correspondant.

- ★ **buttonHistog.setOnClickListener()** -> pour afficher l'histogramme de l'image correspondant.
- ★ **iconSwitchRenderscriptJava.setCheckedChangeListener()** -> pour choisir l'option d'application des filtres sur l'image ; soit en **java** ou **renderScript**, avec un toggle bouton.
- ★ **iconSwitchBrightnessRGB_HSV.setonCheckedChangeListener()** -> pour choisir l'application de la luminosité sur l'image; soit en **HSV** soit **RGB**, avec un toggle bouton.
- ★ **onClick()** -> associé au bouton **GITHUB** qui donne le lien du compte github du projet.

Studio_fragment -> Studio fragment est une classe qui s'étend de Fragment, elle est utilisée pour la gestion et l'affichage de la disposition de l'image. Elle gère les fonctionnalités suivantes :

- ★ Initialisation et affichage de toutes les vues.
- ★ Initialisation de la vue de recyclage du filtre.
- ★ Initialisation de la vue de recyclage des outils.

- ★ Auditeurs d'outils comme le filtre, le recadrement, la rotation, la luminosité, la saturation, le texte, le pinceau, la gomme, les emoji, les autocollants et la reconnaissance faciale.

Et des fonction d'animation qui gère la sélection, l'affichage et les boutons de retour d'enregistrement et réinitialisation de l'image.

Model:

- **animation** :

ViewAnimation{} -> c'est une classe qui contient des méthodes qui vous permettent d'appliquer des animations aux vues pour différentes cibles.
Elle contient:

viewAnimatedChange() -> Méthode statique qui remplace une vue par une autre à l'aide d'une animation d'entrée et d'une animation de sortie. La méthode nous permet de définir la durée avant le lancement et la durée entre deux animations.
En même temps elle contient les trois méthodes:

- ★ **onAnimationStart()** -> qui définisse le commencement de l'animation.
- ★ **onAnimationRepeat()** -> qui définisse le temps de répétition de l'animation.
- ★ **onAnimationEnd()** -> qui définisse la fin de l'animation.

imageViewAnimatedChange() -> Méthode statique qui vous permet de remplacer l'image source d'une ImageView à l'aide des deux animations: fondue et sortant.

viewAnimatedHideOrShow() -> Méthode statique qui vous permet de masquer ou d'afficher une vue à l'aide d'une animation. Elle définit la durée avant le lancement.

- **editingImage** :

❑ **additionalFilters**

AdditionalFilters{} -> c'est une classe qui contient des filtres supplémentaires qui seront appliqués aux images. Elle contient les méthodes:

- ★ **snowAndBlackEffect()** -> c'est une méthode qui permet d'appliquer une neige et un effet noir à l'image.
- ★ **noiseEffect()** -> c'est une méthode qui permet d'appliquer un effet de bruit à l'image.
- ★ **colorizeRGB()** -> c'est une méthode qui booste l'un des trois canaux de pixels de l'image.
- ★ **invertEffect()** -> c'est une méthode qui permet d'appliquer un effet inverse à l'image.
- ★ **shadingFilter()** -> c'est une méthode qui permet d'appliquer un effet d'ombrage à l'image.
- ★ **equalizationYuvY()** -> c'est une méthode permettant d'appliquer une égalisation d'histogramme à la valeur Y d'un pixel.

- ★ **mixEqualizationRgbYuv()** -> c’est une méthode qui applique l'égalisation RVB et l'égalisation YUV et les mélange tous en un seul.

Elle contient aussi ces mêmes fonctions, tous en renderScript.

AuxiliaryFunction{} -> Classe contenant les méthodes statiques auxiliaires utilisées par la classe AdditionalFilters. Elle contient la méthode:

- ★ **histogramYuv()** -> c’est une méthode qui calcule l'histogramme de la valeur v d'un pixel après conversion.

Conversion -> c’est une classe qui contient des méthodes statiques de conversion entre différents espaces colorimétriques utilisés par la classe qui contient des filtres supplémentaires. Elle contient les méthodes:

- ★ **rgbToYuv()** -> c’est une méthode statique qui convertit l'espace colorimétrique RVB en YUV.
- ★ **yuvToRgb()** -> c’est une méthode statique qui convertit l'espace YUV en RVB.

- ❑ **basicFilters**

AuxiliaryFunction -> c’est une classe qui contient les fonctions auxiliaires:

- ★ **RGBtoR_G_B()** -> Fonction qui sépare une couleur RGB en trois composants (R, G et B).
- ★ **Is_like()** -> fonction qui renvoie True si la teinte que nous passons dans le paramètre est proche de la teinte localisée, sinon False.
- ★ **histogram()** -> Fonction qui calcule l'histogramme à partir d'un tableau contenant les pixels d'une image grise.
- ★ **histogramHSV()** -> Fonction qui calcule l'histogramme à partir d'un tableau contenant les pixels de l'image en couleur, en mode HSV (elle prends la plus grande valeur entre h, s et v).
- ★ **min_max_histo()** -> Fonction qui calcule les min et max d'un histogramme d'une image (en couleur ou en noir et blanc).
- ★ **minMax()** -> Fonction qui calcule les min et max à partir d'un histogramme.
- ★ **LUT_Init()** -> Fonction qui initialise la table LUT en utilisant la formule de vue actuelle et la formule inverse pour décrémenter.

Conversion -> c’est une classe contenant des méthodes statiques de conversion entre différents espaces colorimétriques. Elle contient les fonctions:

- ★ **RGBToHSV_new()** -> Fonction qui convertit de RVB en HSV.
- ★ **HSVToColor_new()** -> Fonction qui convertit de HSV en RGB.

Convolution ->

DynamicExtension ->

Equalization ->

Filters ->

- ❑ **filters**

FilterModel -> FilterModel est un type de filtre qui contient: le nom du filtre et le type de filtre. Elle contient deux guetteurs: l'un qui retour le nom et l'autre le type.

FilterType -> c'est une énumération qui contient les filtres qui sont utilisés pour l'initialisation d'un FilterModel.

OnItemFilterSelected -> c'est une 'interface qui sera implémentée par la classe là où le FilterRecyclerViewAdapter sera instancié.

- ❑ **imageAdaptation**

BitmapImageAdaptation -> c'est une classe qui contient toutes les méthodes statiques qui permettent l'adaptation et le réglage des problèmes des images avant le chargement. elle contient des méthodes:

- ★ **getReducedBitmap** -> c'est une méthode qui permet de calculer une nouvelle hauteur et une nouvelle largeur pour réduire la taille de l'image passée en paramètre, et retourne un tableau d'entier contenant la nouvelle hauteur et la nouvelle largeur (réduit).
- ★ **fixAutoRotate** -> c'est une méthode qui permet de vérifier si la rotation de l'image est correcte et de la corriger si ce n'est pas le cas.

- ❑ **tools**

OnItemToolSelected -> c'est l'interface qui sera implémentée par la classe où le EditingToolRecyclerViewAdapter sera instancié. Elle contient une méthode: **onToolSelected** -> qui, dans laquelle l'auditeur sera placé selon le ToolType.

ToolModel -> c'est une classe de type d'outil qui contient:

- ➔ Le nom de l'outil.
- ➔ L'icône d'outil.
- ➔ Le type d'outil.

Son constructeur prend ces trois attributs et génère un guetteur pour chacun d'eux.

ToolType -> c'est une énumération qui contient les outils de modification d'image qui
sont utilisés pour l'initialisation d'un ToolModel.

Teste :

TP 1

Togray && Tograys:

Taille de l'image: 428 x 356

Dans la première version de la fonction, on a utiliser les deux méthodes getpixel() et setpixel() qui étaient beaucoup moins éfficace que getpixels() et setpixels(), et après l'implémentation d'un noyau en utilisant RenderScript on a constaté une amélioration en comparons le temps qu'elles prennent et l'utilisation de la mémoire.

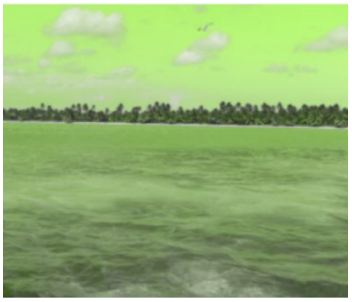
Je tiens à préciser que tous les testes sont effectué sur un téléphone SAMSUNG GALAXY A8 | Version android : 8

Pour le temps voici les résultats :

ToGray	0.721
ToGrays	0.033
ToGray (RenderScript)	0.016

TP 2

Colorize :



Taille de l'image: 428 x 356

La Première version étant implémenter avec JAVA est clairement plus lente que la version faite avec un noyau RenderScript,le hue qui est généré aléatoirement.

Pour le temps voici les résultats :

Colorize	0.788
Colorize (RenderScript)	0.021

Keepcolor :

Taille de l'image: 1000 x 560

La Première version étant implémenter avec JAVA est clairement plus lente que la version faite avec un noyau RenderScript,le hue qu'on veut conserver est passé en paramètre à la fonction qui génère le noyau, pour la version java on lui passe une couleur en RGB.

Pour le temps voici les résultats :

Keepcolor	0.988
Keepcolor (RenderScript)	0.023

TP 3

Contrast plus gray:

Taille de l'image: 615 x 410

Dans la première version de la fonction, on n'a pas utilisé la table LUT , elle était beaucoup moins efficace que la deuxième (utilisation de la table LUT) , et après l'implémentation d'un noyau en utilisant RenderScript on a constaté une amélioration en comparons le temps qu'elles prennent et l'utilisation de la mémoire.

Pour le temps voici les résultats :

Sans lut	0.110

Avec lut	0.044
RenderScript	0.036

Contrast moins gray:

Taille de l'image: 615 x 410

La première version est implémenté en java (avec lut) et après l'implémentation d'un noyau en utilisant RenderScript on a constaté une amélioration en comparons le temps qu'elles prennent et l'utilisation de la mémoire.

Pour le temps voici les résultats :

c-moins java	0.050
c-moin RenderScript	0.034

Contrast plus RGB:

Taille de l’image: 770 x 513

La première version est implémenté en java et après l’implémentation d’un noyau en utilisant RenderScript on a constaté une amélioration en comparons le temps qu’elles prennent et l’utilisation de la mémoire.

Pour le temps voici les résultats :

contraste plus rgb (java)	0.060
contraste plus rgb (RS)	0.034

Contrast plus HSV:

Taille de l’image: 770 x 513

La première version est implémenté en java et après l’implémentation d’un noyau en utilisant RenderScript on a constaté une amélioration en comparons le temps qu’elles prennent et l’utilisation de la mémoire.

l’histogramme est fait avec la plus grande valeur entre le h,v et s

Pour le temps voici les résultats :

contraste plus HSV (java)	0.065
contraste plus HSV (java)	0.033

Egalisation gray:

Taille de l'image: 1000 x 560

Le calcule de l'histogramme cumulé est l'opération la plus gourmande dans la réalisation de l'égalisation ! ,l'implémentation d'un noyau en utilisant RenderScript a amélioré la performance en comparons avec la version java.

Pour le temps voici les résultats :

Egalisation java	0.280
Egalisation RenderScript	0.033

Egalisation RGB:

Taille de l'image: 2000 x 1332

Le calcule de l'histogramme cumulé est l'opération la plus gourmande dans la réalisation de l'égalisation ! ,l'implémentation d'un noyau en utilisant RenderScript a amélioré la performance en comparons avec la version java.

l'histogramme est fait sur le gris et appliqué aux trois canaux

Pour le temps voici les résultats :

java	0.300
RenderScript	0.034

TP 4

Filtre moyenneur:

Taille de l'image: 256 x 256

C'est une convolution 3X3 et les bordures de l'image sont mis à zéro.
la version RS est en cours d'implémentation

Pour le temps voici les résultats :

conc 3X3	0.156