

Développement mobile - TP4 & TP5

October 8, 2018

Victor COLEAU
victor.coleau@etu.univ-tours.fr
Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr



Contents

1	Questions	2
1.1	Installation de l'APK	2
1.2	Premier lancement	2
1.2.1	Cycle de vie	2
1.2.2	Gestion camera	2
1.3	Gradient JAVA	2
1.4	Sobel JAVA	2
1.5	JNI	3
1.6	Résultats C++	3
1.7	Résumé des FPS	3

Chapter 1

Questions

1.1 Installation de l'APK

Nous pouvons grâce à `adb devices` obtenir la liste des périphériques connectés. Puis l'APK est installée avec `adb install <path>`.

1.2 Premier lancement

L'application permet d'obtenir la vidéo de la caméra a 30 fps.

1.2.1 Cycle de vie

L'application passe par les états suivants:

1. Création d'une instance de l'Activity
2. Appel de `onCreate` permettant d'initialiser nos différentes vues
3. Appel de `onResume` démarrant l'accès à la caméra

Lors du passage de l'application en tâche de fond, `onPause` est appelé ce qui a pour effet d'arrêter les mises à jours depuis la caméra. De manière similaire lors du passe au premier plan de l'application, `onResume` est appelée et relance la caméra.

Lors de la destruction de l'application, `onDestroy` est appelé et détruit nos différents objets.

Lors de l'appel à `onResume`, on initialise `openCV`. Deux cas peuvent se présenter:

- `OpenCV` est fourni dans le package de l'application (notre cas), et on a juste à appeler le callback pour démarrer la vue
- `OpenCV` n'est pas dans le package de l'application, dans ce cas on demande à `OpenCV` de l'initialiser à partir d'`OpenCV Manager` (qui se chargera d'appeler le callback pour démarrer la vue).

1.2.2 Gestion camera

`onCameraViewStarted` est appelée lorsque la connexion à la caméra a été effectuée. Cela nous permet notamment de récupérer la taille de celle-ci pour initialiser nos différentes variables.

`onCameraFrame` est appelée lorsque une nouvelle image de la caméra est prête à être affichée. A ce moment nous pouvons donc la modifier si nécessaire.

1.3 Gradient JAVA

Avec l'opération du gradient, nous devrions observer les coutons de l'image.

Avec cette opération supplémentaire, nous arrivons aux alentours de 2-3fps.

1.4 Sobel JAVA

Le filtre de sobel étant encore plus gourmand en ressources que le filtre gradient, nous arrivons avec un résultat d'environ 0.02fps avec un masque de convolution 3×3 .

1.5 JNI

Le nom de la fonction dans le code CPP est: `Java_com_example_polytech_app_MainActivity_process`.

La classe `MainActivity` possède cette méthode.

Le mot clef `native` permet de dire à Java que l'implémentation de la fonction se trouve dans une librairie externe en C.

Le `System.loadLibrary()` permet de charger cette librairie afin d'obtenir l'implémentation de notre fonction.

1.6 Résultats C++

De manière générale avec le même code transposé en C++, nous arrivons à observer que le traitement se fait plus rapidement.

Dans le cas des tablettes Samsung la différence reste peu perceptible. En revanche les tablettes HP bénéficient d'un gros boost grâce à ce passage.

1.7 Résumé des FPS

Méthode	JAVA (Samsung)	Java (HP)	C++ (Samsung)	C++ (HP)
Aucun	30	28	-	-
Gradient	2.83	0.02	3.99	2.8
Sobel	0.9	0.01	1.7	1.9