

Programmazione Sistemi Embedded

Multi-Shot Photography

Biasetton Matteo 1141416
Dal Degan Daniele 1152537
Ziggiotto Andrea 1140706

3 Luglio 2018

1 Introduzione

Nell'ambito della computational photography si stanno diffondendo innumerevoli algoritmi di multi-shot photography. Tali algoritmi permettono di ottenere, attraverso l'unione di una serie di scatti eseguiti con camere di bassa qualità, una foto finale di buona qualità in condizioni di scarsa luminosità.

Questi algoritmi solitamente ricevono in ingresso una serie di foto (*burst*) dello stesso soggetto, eseguite al variare del tempo di esposizione, ovvero il tempo durante il quale l'otturatore della macchina fotografica rimane aperto per permettere alla luce di raggiungere la pellicola o il sensore, con l'obiettivo di ottenere alcune foto molto chiare, dovute alla sovraesposizione e altre foto più scure, dovute alla sotto esposizione.

I due principali problemi legati a questa famiglia di algoritmi sono la difficoltà insita nel tenere ferma la camera mentre si eseguono gli scatti e la difficoltà di fermare la scena che si sta ritraendo, evitando così fenomeni di ghosting.

Per evitare il primo problema descritto, solitamente questi algoritmi vengono corredati con un algoritmo di allineamento, con l'obiettivo di evitare la presenza burst non allineati.

In questo progetto si è deciso di implementare l'algoritmo Exposure Fusion proposto nell'articolo [1], perché secondo gli autori tale sistema permette di ottenere buoni risultati con tempi di computazione ragionevoli.

2 Sviluppo dell'applicazione

Nello sviluppo dell'applicazione si è deciso di dividere in due rami il progetto in modo da permettere ai membri del gruppo di lavorare contemporaneamente. Il primo ramo comprendeva lo sviluppo della parte di backend, ossia il processing delle immagini, mentre il secondo ramo era costituito dallo sviluppo dell'applicazione stessa, vale a dire la parte frontend.

Dopo aver sviluppato le due parti separatamente, la parte di computazione in codice nativo è stata integrata all'interno dell'applicazione.

2.1 Processing immagine

In questa sezione verrà descritto a grandi linee lo sviluppo della parte relativa alla computazione dell'immagine finale. Il codice di questa parte di applicazione è stato scritto in linguaggio *C/C++* avvalendosi dell'utilizzo della libreria *OpenCV*.

Per avere un maggior controllo nello sviluppo di tale parte si è deciso di implementare e di testare il codice sul computer e solo una volta verificato il funzionamento, lo si è aggiunto all'interno all'applicazione *Android*. Il codice sviluppato dopo aver caricato la serie di foto in input, calcola per ognuna di queste una matrice dei pesi legati ai pixel che le compongono. Questi pesi sono calcolati attraverso la combinazione di tre valori, uno legato al contrasto dell'immagine, che si ottiene calcolando la risposta dell'immagine ad un filtro di Laplace e calcolando il valore assoluto del risultato ottenuto. Il secondo parametro è legato alla saturazione e viene calcolato attraverso la moltiplicazione delle deviazioni standard dei tre valori di colori primari (*Red*, *Green*,

Blue), infine l'ultimo parametro è legato all'esposizione e si ottiene pesando il valore dell'intensità di quel pixel con una gaussiana di media 0.5 e di varianza 0.2.

Dopo aver costruito tutte le matrici dei pesi, per avere valori consistenti tra tutte le matrice calcolate si esegue una normalizzazione dividendo ogni valore della matrice per il corrispondente valore della matrice ottenuta attraverso la somma di tutte le matrici dei pesi.

In seguito alla normalizzazione di tutte le matrici dei pesi per ogni foto e per la relativa matrice dei pesi si costruiscono rispettivamente una piramide Laplaciana ed una Gaussiana. Queste piramidi sono costituite da una serie di livelli a partire dall'ultimo che è composto dal burst fino ad arrivare al primo livello che contiene una sua rappresentazione scalata e filtrata.

Nella costruzione della piramide Gaussiana, ogni livello viene costruito attraverso il dimezzamento dell'immagine che costituisce il livello precedente ed il suo smoothing con l'obiettivo di rimuovere i dettagli. Diversamente nella costruzione della piramide Laplaciana, ogni livello viene calcolato attraverso la sottrazione del livello stesso ottenuto come nel caso della piramide Gaussiana e la versione scalata del livello precedente.

Una volta calcolate tutte le piramidi, per ogni immagine e per ogni livello della piramide che la rappresenta si esegue la moltiplicazione tra il valore dei pixel dell'immagine di un determinato livello ed il relativo valore nel livello corrispondente della matrice dei pesi. Quando si hanno tutti questi contributi è possibile costruire la matrice risultante sommando per ogni livello tutti i contributi relativi a tutti le immagini del burst riferiti ad un particolare livello. Una volta terminata questa fase è possibile percorrere la piramide dall'alto verso il basso per ricostruire l'immagine finale, questa fase avviene sommando ogni livello al precedente opportunamente scalato, fino ad arrivare alla base della piramide, che conterrà la foto risultante.

Al termine dello sviluppo del codice si sono confrontati i risultati ottenuti attraverso il software da noi sviluppato e quello fornito insieme all'articolo scritto in linguaggio Matlab. Il confronto per mancanza di metriche oggettive, è stato fatto in modo soggettivo confrontando visivamente le due immagini, in termini di nitidezza ed in termini di varietà di colori. Una volta ottenuti dei risultati con qualità comparabile a quelli dell'articolo, il codice è stato aggiunto alla parte di calcolo Native all'interno dell'applicazione Android.

Il codice sviluppato si compone di due metodi presenti nella lista *processing.cpp* che vengono chiamati in successione dall'applicazione quando si deve eseguire la computazione, questi metodi caricano le immagini del burst e le passano ad un secondo metodo che inizializza la computazione, chiamando prima il metodo *computeWeight* della classe *ComputeWeight*, con l'obiettivo di calcolare le matrici dei pesi. Una volta terminata la chiamata a questo metodo viene invocato il metodo *computePyramid* della classe *Pyramid* che costruisce le due piramidi, calcola l'immagine risultante e la ritorna al metodo native principale.

2.2 Sviluppo applicazione

Contemporaneamente è stata progettata e sviluppata l'infrastruttura dell'applicazione Android che conterrà il codice descritto nella sezione precedente. Nella fase di progettazione è stato deciso di implementare 4 Activity separate che verranno ora descritte in dettaglio.

2.2.1 CameraActivity

Questa è l'Activity principale ed è quella che viene mostrata all'utente all'apertura dell'applicazione. Essa permette di: scattare una foto, aprire la galleria per visualizzare delle foto scattate in precedenza oppure di accedere alla pagina delle impostazioni che permette di settare alcuni parametri di configurazione per l'algoritmo e l'applicazione.

Questa Activity è stata costruita utilizzando l'applicazione di esempio per le API Camera2 fornita da Google [2], tale Activity non è altro che un contenitore per un unico Fragment che realizza tutte le funzioni necessarie. Il Fragment in questione è chiamato *CameraFragment.java*, esso contiene al suo interno tutti i metodi necessari per gestire ed elaborare il flusso completo di acquisizione di un'immagine.

A differenza della versione precedente delle API della camera di Android, le API Camera2 offrono molta più flessibilità e molto più controllo sull'hardware. Un esempio di queste funzionalità sono: l'acquisizione di immagini in formato RAW, acquisizione di foto multiple in sequenza utilizzando lo stesso flusso di acquisizione e la modifica manuale dei parametri della camera (shutter time, ISO, white balancing...).

Per questo progetto queste funzionalità hanno permesso di scattare foto multiple in sequenza variando i livelli

di esposizione della camera per poter catturare le immagini necessarie all'esecuzione dell'algoritmo. Vista la complessità raggiunta dalla classe *CameraFragment.java* ci si limiterà a descrivere solamente i metodi aggiunti o modificati per raggiungere l'obiettivo. In particolare i metodi di maggiore interesse ai fini del progetto sono:

- **onSensorChanged()**: questo metodo si occupa di catturare gli eventi del sensore accelerometro e permette, una volta sovrascritto, di calcolare l'orientazione del dispositivo. Tale metodo è stato implementato per gestire le rotazioni del dispositivo e per modificare di conseguenza l'interfaccia utente e l'orientazione delle foto scattate senza ricorrere alla procedura standard di Android, la quale prevede la distruzione dell'Activity e la successiva reinizializzazione. Questa scelta progettuale è stata fatta per garantire una situazione di continuità nel flusso di anteprima della fotocamera, con il metodo standard invece si percepiva un'interruzione dell'interfaccia utente non trascurabile, cosa assolutamente non comune nelle applicazioni che interagiscono con la fotocamera.
- **setUpCameraOutputs()**: questo metodo è stato modificato per poter gestire il form factor delle foto scattate, secondo le preferenze dell'utente specificate nelle impostazioni.
- **captureBurstPicture()**: questo è il metodo principale utilizzato dalla camera, infatti è il metodo che viene invocato al momento della pressione del tasto per scattare la foto. Questo ha lo scopo di settare tutti i parametri necessari e avviare la procedura di cattura del burst di foto desiderato. Per avviare la procedura, il metodo a sua volta invoca il metodo *captureRequest()* che riceve in input una lista di richieste (una per ogni foto della sequenza) che vengono definite nel metodo *createBurstRequest()*.
- **createBurstRequest()**: questo metodo genera una lista di richieste corrispondenti alle foto del burst. Ogni richiesta può avere caratteristiche differenti. Per la realizzazione di questo algoritmo, all'inizio della procedura di cattura, vengono salvati il tempo di shutter e la sensibilità. Successivamente viene fissata la sensibilità per tutte le richieste mentre viene variato il tempo di shutter per variare il livello di esposizione di ogni foto.
- **executeAlgorithm()**: questo metodo viene invocato alla fine della procedura di cattura del burst quando tutte le foto catturate sono state salvate nella memoria interna. Esso invoca l'Activity *VisualImageActivity* tramite un Intent a cui viene passato come parametro l'indirizzo in memoria della prima foto della sequenza, in modo che l'algoritmo scritto in linguaggio nativo possa recuperare tutte le foto del burst corrispondente.



Figura 1: CameraActivity

2.2.2 SettingsActivity

Questa Activity permette di modificare alcuni parametri relativi all'applicazione come: la possibilità di salvare i burst oppure no, permette di scegliere il numero di foto che compongono un burst, il form factor desiderato per la foto(16:9 oppure 4:3) e le preferenze relative alla scelta del passo di variazione dell'esposizione. Le impostazioni selezionate dall'utente vengono salvate tramite SharedPreferences e vengono caricate ogni volta che viene avviata l'Activity, tali impostazioni vengono recuperate dalle altre Activity con lo stesso processo.

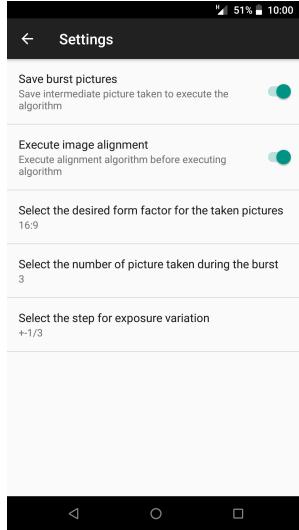


Figura 2: SettingsActivity

2.2.3 GalleryActivity

Questa Activity realizza una galleria interna all'applicazione che permette all'utente di visualizzare le foto scattate in precedenza.

Quando l'utente seleziona una delle foto nella galleria, viene lanciato un *Dialog* che consente di visualizzare le singole foto che compongono il burst (Figura 3). Tramite l'interfaccia proposta dal *Dialog* è possibile eliminare oppure selezionare il burst per eseguire nuovamente l'algoritmo Exposure Fusion.

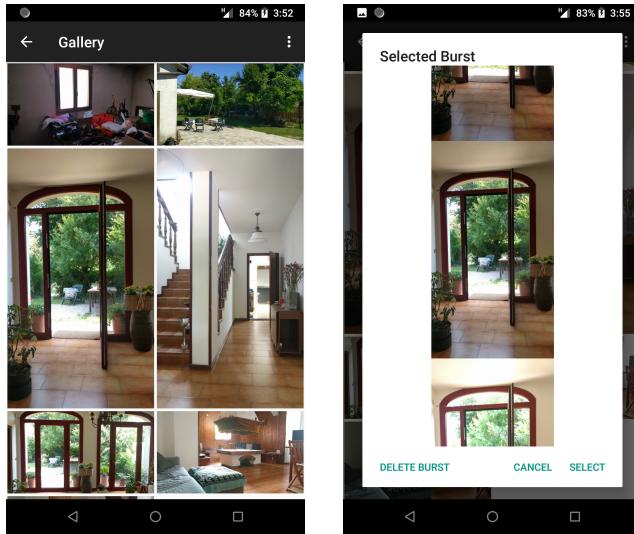


Figura 3: Gallery Activity

2.2.4 VisualImageActivity

Questa Activity permette all’utente di visualizzare l’output finale dell’algoritmo implementato. Essa viene lanciata in automatico in seguito allo scatto di una foto oppure alla selezione di una foto già scattata presente nella galleria.

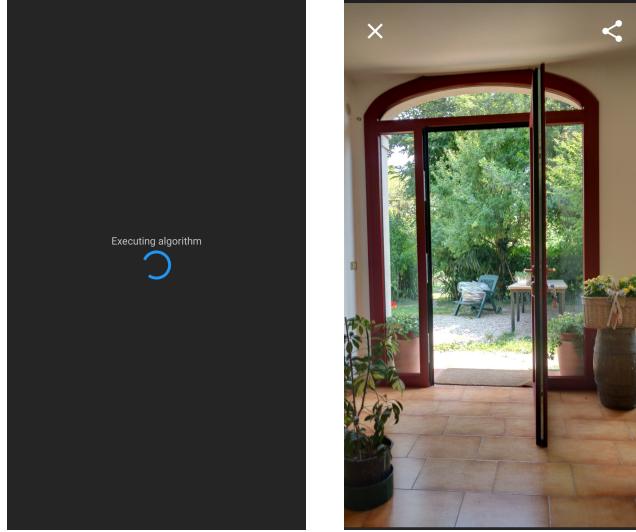


Figura 4: VisualImageActivity

3 Step e Problematiche

3.1 Configurazione della parte Native

Inizialmente è stato necessario includere la parte di codice native nell’applicazione, processo essenziale per lo scopo della stessa. È stata un’operazione che ha richiesto molteplici configurazioni all’interno dell’applicazione e che ha presentato pochi problemi.

Si è dovuto includere la libreria di *OpenCV* in formato *Lite*, ossia in una versione più leggera e adatta ai dispositivi embedded.

3.2 Galleria

Per la realizzazione della galleria inizialmente era stato utilizzato un layout di tipo *GridView* contenente degli elementi di tipo *ImageView*. Utilizzando questo sistema abbiamo notato che all’aumentare delle immagini presenti nella galleria le prestazioni dell’applicazione peggioravano drasticamente. Questo accadeva perché le immagini dovevano essere caricate tutte in memoria prima di essere mostrate a video. Per cercare di risolvere questo problema abbiamo provato a ridurre la dimensione delle immagini da visualizzare come anteprima; questo ha mitigato il problema ma non lo ha risolto completamente perché il tempo di risposta della UI comunque tendeva ad aumentare linearmente con il numero di foto.

Infine abbiamo deciso di utilizzare la libreria Glide[3] la quale utilizza meccanismi di caching e caricamento asincrono delle immagini in modo da garantire una risposta immediata da parte dell’applicazione. In questo modo l’Activity viene caricata immediatamente e le immagini vengono mostrate all’utente non appena vengono caricate, in modo indipendente l’una dalle altre.

3.3 Implementazione codice Native

Nell’implementazione del codice Native non abbiamo riscontrato grosse difficoltà, tranne in alcuni passaggi come nel capire la giusta codifica da assegnare al Mat per eseguire la computazione. Il paper utilizzato spiegava con precisione tutte le funzionalità da implementare per ottenere il risultato proposto.

3.4 Coordinamento Java-Native

Abbiamo riscontrato alcuni problemi nel trasferire le foto appartenenti ad un burst tra la parte Java e la parte native. Il primo metodo sviluppato prevedeva il passaggio di un vettore di Mat (vettore di immagini nel formato utilizzato da OpenCV) ma questo rallentava molto la parte di processing delle immagini oltre a richiedere una notevole quantità di memoria RAM che causava inevitabilmente un crash dell'applicazione, soprattutto nei dispositivi aventi poca memoria libera.

Si è pensato, quindi, di effettuare il trasferimento delle immagini salvandole in una cartella apposita e successivamente eliminarle qualora l'utente non desiderasse mantenerle nella memoria del proprio telefono. La scelta di utilizzare questo metodo ci ha permesso di ridurre notevolmente il tempo di transizione tra le Activity, fornendo un migliore funzionamento all'utente finale.

Alcuni problemi sono stati trovati quando si è aggiunta la parte native di computazione della foto nell'applicazione Android, poiché OpenCV elabora le immagini nello spazio di colore BGR invece Android le elabora utilizzando lo spazio RGB, per questo ogni volta che si passa dalla parte Android alla parte Native-OpenCv e viceversa è stato necessario eseguire delle conversioni.

3.5 Calcolo esposizione delle foto

Dovendo scattare più foto con tempi di esposizione diversi, il problema principale è stato quello di capire quale valore di esposizione fosse più appropriato per questo scopo.

Inizialmente si erano fissati dei parametri standard, in modo da avere per ogni dispositivo un set di valori prefissati. Questa soluzione è risultata insoddisfacente perché il tempo di esposizione non può essere stabilito a priori a causa della sua dipendenza dal tipo di dispositivo e dalle condizioni di luminosità, per esempio in condizioni di scarsa luminosità i tempi di esposizione devono essere aumentati. Per ovviare a questo problema si è reso necessario abilitare inizialmente l'auto-esposizione della fotocamera per prelevare tale valore e successivamente impostare manualmente il parametro opportunamente scalato con l'obiettivo di scattare il burst di foto.

3.6 Compatibilità dispositivi

Per lo sviluppo di questa applicazione sono stati testati diversi dispositivi per motivi di compatibilità con le API Camera2 le quali richiedono un dispositivo con versione Android ≥ 21 .

In particolare sono stati testati diversi dispositivi: Huawei p8, Nexus 7, Nexus 5x, OnePlus 6. Di questi solamente gli ultimi due si sono rivelati compatibile al 100% con tutte le funzionalità richieste, come ad esempio l'impostazione manuale dei tempi di esposizione.

Per evitare errori inaspettati è stato inserito un controllo sulla compatibilità al momento dello scatto di una foto, il quale avvisa l'utente in caso di incompatibilità.

Un altro problema che è stato riscontrato è che negli scatti multipli con 5 foto, a volte viene generato un errore durante la cattura della sequenza il quale impedisce il completamento della procedura. In questo caso l'errore viene catturato e segnalato all'utente, tuttavia l'algoritmo viene comunque eseguito con le foto che sono state scattate con successo.

Non siamo stati in grado di risolvere questo problema, in quanto il codice di errore restituito dà un'informazione generica e non permette di risalire alla causa dell'errore. Questo problema è stato riscontrato solamente nel dispositivo Nexus 5x che è stato utilizzato per lo sviluppo e i test, tuttavia non si è presentato con il OnePlus 6.

4 Risultati e Conclusione

Utilizzando l'algoritmo *Exposure Fusion* sono stati ottenuti risultati soddisfacenti sia in termini di risultato visivo finale che di tempo di processing.

I tempi per il processing delle immagini si sono dimostrati abbastanza veloci sia per i dispositivi nuovi che per quelli datati (sull'ordine di qualche secondo), l'immagine risultante calcolata utilizzando più foto ha dimostrato un netto miglioramento rispetto a quelle singole.

L'applicazione sviluppata offre un notevole miglioramento nel caso di burst scattati in ambienti poco luminosi, diversamente se le foto vengono scattate in buone condizioni luminose la foto risultante migliora leggermente

rispetto alla migliore foto del burst.

Di seguito è possibile confrontare visivamente i risultati ottenuti con il codice Matlab allegato con il paper [4] e quelli ottenuti con l'applicazione Android da noi sviluppata. La prima foto era presente all'interno del codice Matlab e quindi è stata presa come riferimento, si può notare nel confronto che le due foto risultati sono molto simili.

Diversamente da quanto descritto nella proposta di progetto, non è stato implementato nessun algoritmo di allineamento per mancanza di tempo, ma per fornire comunque un prodotto abbastanza completo abbiamo provato due algoritmi di allineamento, il primo che abbiamo utilizzato si basava sulla feature detection, però è stato scartato perché i risultati ottenuti non erano soddisfacenti. Il secondo algoritmo che abbiamo provato è l'algoritmo MTB basato sull'articolo [5] presente all'interno della libreria OpenCV, tale algoritmo ha fornito risultati interessanti e quindi si è deciso di aggiungerlo all'applicazione sviluppata.



Figura 5: Burst



Figura 6: Risultato applicazione sviluppata



Figura 7: Risultato Matlab

Riferimenti bibliografici

- [1] T. Mertens, J. Kautz, and F. V. Reeth. Exposure fusion. In Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on, pages 382–390, Oct 2007.
- [2] Camera 2 API Sample fornito da Google: applicazione android-camera2basic. <https://github.com/googlesamples/android-Camera2Basic>. Accessed: 3 luglio 2018.
- [3] Glide library. <https://github.com/bumptech/glide>. Accessed: 3 luglio 2018.
- [4] Exposure Fusion implementazione algoritmo in codice matlab. <https://github.com/Mericam/exposure-fusion>. Accessed: 3 luglio 2018.
- [5] Greg Ward. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. Journal of Graphics Tools, 8(2):17–30, 2003.