Architetture e Programmazione dei Sistemi di Elaborazione

Progetto a.a. 2016/17

Algoritmo di *Corner Detection* in linguaggio assembly x86-32+SSE e x86-64+AVX

Fabrizio Angiulli

fabrizio.angiulli@unical.it

1 Decrizione del problema

Un'immagine digitale f in scala di grigi (grayscale) è una matrice $N \times M$ di valori reali appartenenti all'intervallo $[0, 1]$, i cui elementi rappresentano i singoli pixel dell'immagine.

L'elemento $f(x, y)$ ($0 \leq x \leq N-1$, $0 \leq y \leq M-1$) di f fornisce l'intensità luminosa del pixel di coordinate (x, y). Il valore 0 (1, rispettivamente) corrisponde al nero (bianco, rispettivamente).

In particolare, il vertice superiore sinistro (inferiore destro, rispettivamente) dell'immagine corrisponde al il pixel $(0, 0)$ (al pixel $(N-1, M-1)$, rispettivamente).

Si noti che un'immagine digitale si può riguardare anche come una funzione

$$f : \{0, 1, \ldots, N-1\} \times \{0, 1, \ldots, M-1\} \mapsto [0, 1].$$

Nel seguito i termini immagine (digitale), matrice e funzione sono utilizzati come sinonimi. Si noti inoltre che, benché i valori dei pixel di immagini da visualizzare sono compresi nell'intervallo $[0, 1]$, la loro elaborazione può portare ad immagini intermedie i cui pixel assumono valore all'esterno di questo intervallo.

Filtraggio spaziale. L'operatore $\star$ di *filtraggio* permette di effettuare svariati tipi di elaborazione su di un'immagine digitale. In particolare, l'operazione

$$g = f \star w$$

combina un'immagine f (di dimensione $N \times M$) con una seconda immagine w (di dimensione $n \times m$, con $n < N$ ed $m < M$), detta *filtro*, restituendo una nuova immagine g avente la stessa dimensione di f.

Si assume che le dimensioni n ed m di w siano dispari, ovvero tali che $n = 2n' + 1$ ed $m = 2m' + 1$. Nello specifico, ogni singolo pixel $g(x, y)$ dell'immagine risultato g è ottenuto come di seguito espresso:

$$g(x, y) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} w(i, j) \cdot f(x - n' + i, y - m' + j).$$

In altri termini, per ottenere $g(x, y)$ occorre idealmente sovrapporre il filtro w all'immagine f, in maniera tale che il pixel centrale $w(n', m')$ di w coincida con il pixel $f(x, y)$ di f, e quindi calcolare la somma dei prodotti delle intensità dei pixel di f e w che occupano la medesima posizione. Intuitivamente, $g(x, y)$ si può riguardare come una sorta di media pesata, attraverso i valori in w, dei pixel di f posti nell'intorno di $f(x, y)$.

Al fine di poter applicare il filtraggio anche ai pixel che si trovano nei pressi dei bordi dell'immagine f, si assume che i pixel posti all'esterno di f siano identici al pixel di f a loro più prossimo lungo la direzione orizzontale o verticale.

Derivate parziali. Le derivate parziali f_x ed f_y di un'immagine digitale f si determinano facendo uso di filtri di tipo derivativo $w^{\partial x}$ e $w^{\partial y}$:

$$w^{\partial x} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad w^{\partial y} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},$$

come di seguito specificato:

$$f_x = \frac{\partial f}{\partial x} = f \star w^{\partial x}, \quad f_y = \frac{\partial f}{\partial y} = f \star w^{\partial y}.$$

Smoothing. Lo *smoothing* di un'immagine digitale f consiste nella "levigatura" della superficie descritta dalla funzione f al fine di attenuare il rumore in essa presente e far emergere dettagli aventi specifiche risoluzioni. Lo smoothing si effettua filtrando l'immagine per mezzo di particolari filtri, detti appunto di smoothing.

Uno dei più diffusi filtri di smoothing è il filtro gaussiano G_σ, con σ che rappresenta la deviazione standard della funzione gaussiana. Il filtro G_σ ha dimensione $n \times n$, dove $n = 2n' + 1$ è uguale al più piccolo intero dispari maggiore o uguale a 6σ:

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[-\frac{(x - n')^2 + (y - n')^2}{2\sigma^2}\right], \quad \forall (x, y) \in \{0, 1, \ldots, n - 1\}^2.$$

I coefficienti così ottenuti sono poi normalizzati sulla loro somma in modo che la somma degli elementi di G_σ corrisponda ad 1.

Harris Corner Detector. Gli algoritmi di *corner detection* hanno come obiettivo la determinazione dei punti in cui convergono due bordi, detti anche *corner*. La strategia generale utilizzata da questi algoritmi è isolare i punti in cui si verifica un cambiamento significativo di intensità in tutte le direzioni, ovvero i punti aventi bassa autosimilarità.

Per brevità non illustriamo qui la teoria alla base dell'algoritmo, rimandando alla letteratura per ulteriori dettagli.

Dal punto di vista delle computazioni effettuate, l'algoritmo di Harris calcola le derivate parziali f_x ed f_y dell'immagine, quindi le combina per ottenere le matrici f_{xx}, f_{yy} ed f_{xy}, che codificano l'informazione riguardante la somiglianza di ogni pixel con il suo intorno, ed infine determina la matrice di risposta R, rappresentante il grado di dissimilarità di ogni pixel rispetto al suo intorno. I punti (x, y) di massimo locale della matrice R forniscono le coordinate dei corner.

L'algoritmo riceve in input l'immagine digitale f in scala di grigi da elaborare ed i seguenti parametri:

– σ, deviazione standard del filtro di smoothing gaussiano da impiegare (default 1.5);

– $\theta \in [0, 1]$, numero reale impiegato durante la fase di sogliatura (default 0.2; si veda di seguito);

– $t \geq 1$, numero intero impiegato durante la fase di soppressione dei non-massimi (default $t = 1$, si veda di seguito).

I passi eseguiti dall'algoritmo di corner detection di Harris sono di seguito dettagliati:

1. Effettua lo smoothing dell'immagine f:

$$\hat{f} = f \star G_\sigma$$

2. Calcola le derivate parziali di $\hat{f}$:

$$\text{(a) } f_x = \hat{f} \star w^{\partial x}, \quad \text{(b) } f_y = \hat{f} \star w^{\partial y}$$

3. Calcola le immagini f_{xx}, f_{xy}, f_{xy}:

$$\text{(a) } f_{xx} = f_x * f_x, \quad \text{(b) } f_{yy} = f_y * f_y, \quad \text{(c) } f_{xy} = f_x * f_x,$$

dove $f * g$ indica la matrice h tale che $h(x, y) = f(x, y) \cdot g(x, y)$.

4. Effettua un ulteriore smoothing delle immagini f_{xx}, f_{yy} ed f_{xy} usando un filtro gaussiano con $\sigma' = 2\sigma$:

$$\text{(a) } S_{xx} = f_{xx} \star G_{2\sigma}, \quad \text{(b) } S_{yy} = f_{yy} \star G_{2\sigma}, \quad \text{(c) } S_{xy} = f_{xy} \star G_{2\sigma}$$

5. Per ogni pixel (x, y) di f, definisci la sua matrice di Harris $H(x, y)$ come segue:

$$H(x, y) = \begin{bmatrix} S_{xx}(x, y) & S_{xy}(x, y) \\ S_{xy}(x, y) & S_{yy}(x, y) \end{bmatrix}$$

Si nota che ogni matrice $H(x, y)$ ha dimensione 2×2.

6. Calcola la matrice R di risposta come:

$$R(x, y) = \mathbf{det}\Big[H(x, y)\Big] - 0.05 \cdot \mathbf{tr}\Big[H(x, y)\Big]^2,$$

dove **det** denota il determinante e **tr** denota la traccia (ovvero la somma degli elementi posti sulla diagonale principale) della matrice $H(x, y)$.

7. Fase di sogliatura. Sia $R_{\max}$ il valore di intensità più grande presente nella matrice R. Al fine di ridurre la probabilità di falsi positivi, i pixel $R(x, y)$ il cui valore di intensità è inferiore a $\theta \cdot R_{\max}$ vengono posti a 0.

8. Fase di soppressione dei non-massimi. La matrice R viene trasformata in una matrice binaria R'. In particolare, $R'(x, y) = 1$ se l'elemento $R(x, y)$ è strettamente positivo (> 0) e maggiore o uguale ($\geq$) di tutti gli altri elementi nel suo intorno di raggio t, definito come l'insieme degli elementi $R(x', y')$ tali che $|x' - x| \leq t$ e $|y' - y| \leq t$. Diversamente, $R'(x, y) = 0$.

9. I corner dell'immagine f sono infine ottenuti come l'insieme delle coordiante (x, y) di R' per cui $R'(x, y)$ vale 1. Si assuma in questo caso che i pixel esterni all'immagine valgano 0.

Proprietà del filtraggio. Di seguito si ricordano due proprietà notevoli del filtraggio.

Un filtro w, avente dimensione $n \times m$, si dice *separabile* se può essere espresso come il prodotto di due filtri w_x, di dimensione $n \times 1$, e w_y, di dimensione $1 \times m$, ovvero se $w = w_x \cdot w_y$. In questo caso vale anche che $f \star w = (f \star w_x) \star w_y$. Si nota che i filtri $w^{\partial x}$, $w^{\partial y}$ e G_σ sono tutti separabili.

L'operazione di filtraggio è *associativa*, ovvero $(f \star w_1) \star w_2 = f \star (w_1 \star w_2)$.

2 Descrizione dell'attività progettuale

Obiettivo del progetto è mettere a punto un'implementazione dell'algoritmo di *Corner Detection di Harris* e dell'algoritmo di *Filtraggio Spaziale* in linguaggio C e di migliorarne le prestazioni utilizzando le tecniche di ottimizzazione basate sull'organizzazione dell'hardware.

In particolare, l'algoritmo riceve l'immagine f e deve poter funzionare in modalità:

1. *corner detection*, nel qual caso sarà opzionalmente possibile specificare i parametri σ, θ e t, e verranno restituite in uscita la matrice di risposta R e l'elenco dei corner di f; oppure

2. *filtering*, nel qual caso dovrà ricevere in ingresso una seconda immagine w rappresentante il filtro e verrà restituita in uscita l'immagine $g = f \star w$.

L'ambiente sw/hw di riferimento è costituito dal linguaggio di programmazione C (`gcc`), dal linguaggio assembly x86-32+SSE e dalla sua estensione x86-32+AVX (`nasm`) e dal sistema operativo Linux (`ubuntu`).

Di seguito si riportano le linee guida per il corretto svolgimento del progetto:

- Si consiglia di affrontare il progetto nel seguente modo:

  1. Inannzitutto, codificare l'algoritmo interamente in linguaggio C, possibilmente in maniera modulare (ovvero come sequenza di chiamate a funzioni);
  2. Sostituire una o più (idealmente tutte le) funzioni scritte in linguaggio ad alto livello con corrispondenti funzioni scritte in linguaggo assembly.

  Ciò consentirà (1) di verificare che l'algoritmo che si intende ottimizzare è corretto, (2) di gestire meglio la complessità del progetto e (3) di avere la possibilità di presentare il progetto (funzionante) anche se non si è riusciti a riscrivere in linguaggio assembly tutte le funzioni introdotte.

- Le dimensioni $N \times M$ dell'immagine f, nonché i parametri σ, θ e t, oppure le dimensioni $n \times m$ del filtro w, devono poter variare da esecuzione ad esecuzione. I test di prestazione verranno effettuati con immagini ad elevata risoluzione (ad esempio, UHD in scala di grigi con $N = 2160$ ed $M = 3840$, oppure superiori), con valori di σ e t dell'ordine delle svariate decine e con risoluzioni n ed m dei filtri dell'ordine delle decine/centinaia, compatibilmente con i tempi di risposta dell'implementazione.

- Al fine di migliorare la valutazione dell'attività progettuale è possibile presentare nella relazione un confronto tra le prestazioni delle versioni intermedie, ognuna delle quali introduce una particolare ottimizzazione, e finale del codice. Obiettivo del confronto è sostanziare la bontà delle ottimizzazioni effettuate.

- Occorre lavorare in autonomia e non collaborare con gli altri gruppi. Soluzioni troppo simili riceveranno una valutazione negativa. I progetti verranno messi in competizione.

- Sono richieste due soluzioni software, la prima per l'architettura x86-32+SSE e la seconda per l'architettura x86-64+AVX. Per la codifica dei numeri in virgola mobile utilizzare la precisione singola (32 bit per numero).

  Per i dettagli riguardanti la redazione del codice fare riferimento ai files `corner32c.c`, `corner32.nasm`, `runcorner32` (versione a 32 bit) e `corner64c.c`, `corner64.nasm`, `runcorner64` (versione a 64 bit) disponibili sulla piattaforma `didattica.dimes.unical.it`.

  Per l'interfacciamento tra programmi in linguaggio C e programmi in linguaggio assembly fare riferimento al documento allegato alla descrizione del progetto.

- Le soluzioni non devono far uso delle istruzioni `OpenMP`. Opzionalmente, è possibile consegnare delle soluzioni aggiuntive che facciano uso anche di istruzioni `OpenMP`.

- Il software dovrà essere corredato da una relazione. Per la presentazione del progetto è possibile avvalersi di slide.

- Prima della data di consegna del progetto verranno pubblicate le convenzioni da rispettare riguardanti i nomi e la collocazione dei file/directory al fine della compilazione e l'esecuzione dei codici di programma mediante script appositamente predisposti. **Dato l'elevato numero di progetti, sarà cura del candidato accertarsi di aver rispettato pienamente le convenzioni di consegna, pena l'impossibilità di eseguire il progetto e la conseguente valutazione negativa dell'elaborato.**

Buon lavoro!

Riferimenti bibliografici

[1] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing. *Pearson Prentice Hall.* Third Edition, 2010.

Interfacciamento tra linguaggio C e linguaggio Assembly

Fabrizio Angiulli

Abstract

Il presente documento fornisce una breve descrizione delle convenzioni di chiamata delle funzioni in linguaggio C (*C Calling Conventions*) utilizzate dal compilatore GNU, sia con riferimento all'architettura x86-32 che all'architettura x86-64. La conoscenza di tali convenzioni consente di interfacciare programmi scritti in linguaggio C con programmi scritti in linguaggio Assembly, mediante l'invocazione di funzioni Assembly da funzioni C e viceversa.

1 C Calling Conventions a 32 bit

I programmi in linguaggio C su architettura x86-32 utilizzano le seguenti convenzioni per il passaggio dei parametri tra funzioni:

- I parametri passati alle funzioni vengono inseriti (*push*) nello stack in ordine inverso. Quindi, data la chiamata di funzione `func(a, b, c)`, il valore del parametro `c` viene posto nello stack per primo, quello di `b` per secondo e infine quello di `a` per terzo;
- La funzione chiamante assume che il contenuto dei registri ESP, EBP, EBX, ESI e EDI non venga alterato dalla funzione chiamata. Questo non significa che questi registri non possono essere utilizzati dalla funzione chiamata, ma piuttosto che il loro valore al termine dell'esecuzione della funzione chiamata deve coincidere con il valore all'inzio dell'esecuzione. Tutti gli altri registri possono essere modificati senza nessun accorgimento dalla funzione chiamata;
- La funzione restituisce il valore di ritorno nel registro EAX se la dimensione del tipo di ritorno è minore o uguale a 32 bit. I puntatori (a qualsiasi tipo) sono da assimilarsi ad interi a 32 bit. Nel caso il valore di ritorno sia a 64 bit, viene restituito nella coppia di registri EAX (i 32 bit meno significativi) e EDX (i 32 bit più significativi). I valori in virgola mobile (ovvero `float` oppure `double`) vengono posti in cima allo stack dei registri x87 (che non abbiamo trattato). Le strutture ed ogni altro tipo avente dimensione maggiore di 32 bit devono essere restituite per riferimento, ovvero la funzione ne restituirà l'indirizzo di partenza nel registro EAX;
- La funzione chiamata non rimuove i parametri dallo stack. La rimozione dei parametri dallo stack è a carico della funzione chiamante e può essere effettuata sommando allo stack pointer ESP la dimensione dei parametri passati.

Ad esempio, si consideri il seguente frammento di codice C, in cui il `main` chiama una funzione esterna di nome `asmfunc` scritta in linguaggio assembly x86-32:

```
extern int asmfunc(float* A, int m, int n);

int main() {
    int m = 10, n = 2;
    float* A = calloc(m*n,sizeof(float));
    int y = asmfunc(A,m,n);
    ...
}
```

Di seguito si riporta la struttura generale della funzione `asmfunc`:

```
global asmfunc       ; rende la funzione visibile all'esterno

; Posizione dei parametri nel Recordi di Attivazione della funzione
; (i primi 8 bytes sono occupati dall'indirizzo di ritorno e da EBP)
;
A          equ       8                    ; puntatore a float, occupa 32 bit (4 bytes)
m          equ       12                   ; intero a 32 bit
n          equ       16                   ; intero a 32 bit

asmfunc:
           ;
           ; sequenza di ingresso nella funzione
           ;
           push      ebp                  ; salva il Base Pointer
           mov       ebp, esp             ; il Base Pointer punta al Record di Attivazione corrente
           push      ebx                  ; salva i registri da preservare
           push      esi
           push      edi
           ;
           ; lettura dei parametri dal Recordi di Attivazione
           ;
           mov       ..., [ebp+A]         ; legge A
           mov       ..., [ebp+m]         ; legge m
           mov       ..., [ebp+n]         ; legge n
           ;
           ; corpo della funzione
           ;
           ...
           mov       eax, <valore-di-ritorno>; asmfunc restituisce un numero intero
           ;
           ; sequenza di uscita dalla funzione
           ;
           pop       edi                  ; ripristina i registri da preservare
           pop       esi
           pop       ebx
           mov       esp, ebp             ; ripristina lo Stack Pointer
           pop       ebp                  ; ripristina il Base Pointer
           ret                            ; ritorna alla funzione chiamante
```

Per completezza, di seguito si mostra come il compilatore `gcc` traduce in assembly la chiamata della funzione `asmfunc`:

```
main:
         ...
         push     <valore-di-n>       ; posiziona i parametri nello stack
         push     <valore-di-m>
         push     <indirizzo-di-A>
         call     asmfunc             ; chiama la funzione asmfunc
         add      esp, 12             ; rimuove i parametri dallo stack
         ...
```

2 C Calling Conventions a 64 bit

I programmi in linguaggio C su architettura x86-64 utilizzano le seguenti convenzioni per il passaggio dei parametri tra funzioni:

- I primi sei parametri interi (scorrendo l'elenco dei parametri da sinistra verso destra) vengono passati, rispettivamente, nei registri RDI, RSI, RDX, RCX, R8 ed R9. Ulteriori parametri interi vengono passati sullo stack. I registri di cui sopra, insieme ai registri RAX, R10 ed R11 possono essere modificati dalla funzione chiamata senza la necessità di ripristinarli ai loro valori originari;
- I valori di ritorno interi vengono restituiti nei registri RAX e RDX;
- Gli argomenti floating-point vengono passati nei registri da XMM0 a XMM7. I valori di ritorno floating point vengono restituiti nei registri XMM0 ed XMM1;
- Tutti i registri SSE ed x87 possono essere alterati dalla funzione chiamata.

Ad esempio, nel caso della chiamata della funzione

`void func(long a, double b, int c)`,

il parametro `a` viene passato in RDI, il parametro `b` viene passato in XMM0, mentre il parametro `c` viene passato in RSI.