
 Universidade do Minho	Módulo 1 Introdução ao caso de estudo	
--	--	---

Motivação

Este módulo apresenta a operação de convolução entre dois sinais discretos e analisa o código *assembly* gerado por um compilador para uma versão não otimizada do operador `convolve2D()` escrito em C.

A convolução será utilizada como caso de estudo ao longo desta Unidade Curricular (UC) para ilustrar vários dos conceitos apresentados durante as aulas.

Convolução

A convolução é um operador matemático que, a partir de 2 funções $f()$ e $g()$, produz uma terceira, $h()$. A função resultante, $h()$, é vista como uma versão modificada da função original, $g()$, mas ponderada (ou filtrada) pela função $f()$. $h()$ é então a média pesada de $g()$, sendo os pesos dados por $f()$. Formalmente, e para funções definidas sobre domínios contínuos, a convolução é escrita como um integral sobre o domínio de $f()$. A convolução é uma operação comumente utilizada em áreas como o processamento de sinal, visão por computador, engenharia electrotécnica e electrónica, entre outras. No contexto desta Unidade Curricular (UC) iremos aplicá-la a um problema de Visão por Computador.

No contexto desta UC estamos interessados em utilizar a convolução como um filtro aplicado a imagens digitais monocromáticas. Uma imagem é um sinal bidimensional (largura e altura : WxH) discreto (definido para um número finito e não contínuo de pontos que normalmente designamos por pixels). Uma imagem, $I[y][x]$, pode ser convolvida com um filtro $f[j][i]$ 2D com largura U e altura V. Neste caso a expressão da convolução 2D é dada por

$$h[y][x] = (f * I)[y][x] = \frac{\sum_{i=-\lfloor U/2 \rfloor}^{\lfloor U/2 \rfloor} \sum_{j=-\lfloor V/2 \rfloor}^{\lfloor V/2 \rfloor} f[j + \lfloor V/2 \rfloor][i + \lfloor U/2 \rfloor] I[y + j][x + i]}{\sum_{i=0}^U \sum_{j=0}^V f[j][i]}$$

$I[y][x]$ identifica o pixel da linha y e coluna x da imagem I. O operador matemático $\lfloor z \rfloor$ representa a operação *floor()*, isto é, devolve o maior inteiro menor do que z (ex. $\lfloor \frac{3}{2} \rfloor = 1$). Note que para calcular $h[y][x]$ é utilizada uma janela bidimensional centrada em $I[y][x]$ com dimensões UxV, isto é, as dimensões do filtro $f[j][i]$ – ver figura abaixo.

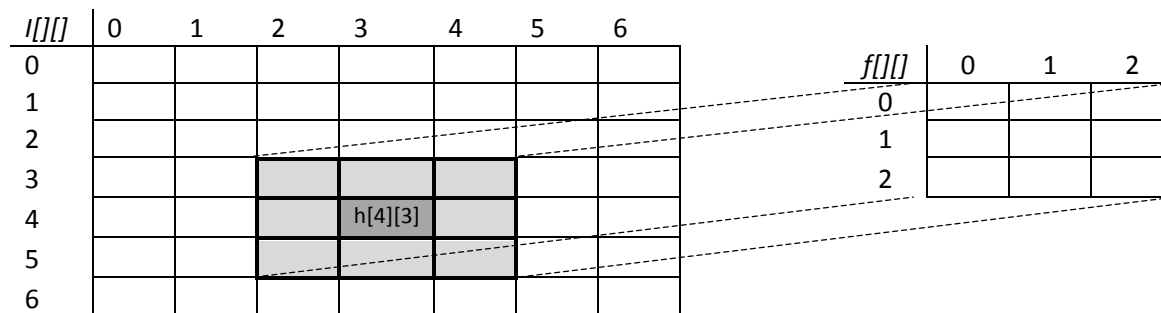


Ilustração 3 - janela de $I[y][x]$ usada para calcular $h[4][3]$: é usada uma janela com a dimensão 3x3, isto é, a mesma dimensão do filtro $f[y][x]$

Resumindo, o valor de $h[y][x]$ é dado por uma média pesada da vizinhança de $I[y][x]$, sendo que os pesos são os valores de $f[y][x]$ e a dimensão da vizinhança é igual à dimensão de $f[y][x]$.

O resultado desta operação de filtragem depende obviamente do tipo de filtro usado. Um filtro do tipo passa-baixo elimina altas frequências, isto é, tende a suavizar as imagens diluindo os contornos dos objectos. Um exemplo de filtro passa-baixo é a média (ou *box filter*) que soma todos os elementos da vizinhança e depois divide pelo número desses elementos. Já um filtro passa-alto elimina as baixas frequências; o seu efeito é realçar os contornos dos objectos presentes numa imagem.

Para mais detalhes ver o documento disponibilizado sobre Convolução.

Sessão Laboratorial

Arranque a sua máquina usando a imagem do Fedora Core 13, inicie sessão usando as credenciais fornecidas pelo docente e inicie o gestor de janelas (`startx`).

Descarregue o código associado a este módulo e construa o executável (`make`).

O executável gerado permite aplicar a uma imagem no formato PPM (i) uma convolução com um filtro média com dimensão 3x1 ou (ii) uma convolução com um filtro Gaussiano de largura e altura U .

Corra o executável para o caso (i) escrevendo:

```
./convolve AC_images/brian_kernighan.ppm result.ppm 1
```

O último parâmetro especifica que se pretende o filtro 3x1. No ecrã foram apresentadas várias estatísticas relativas ao desempenho da máquina que serão utilizadas em sessões futuras. Visualize a imagem gerada e armazenada no ficheiro `result.ppm`.

Algoritmo

O excerto de código abaixo implementa a convolução 3x1 sem qualquer optimização. Algumas notas sobre este código:

- recebe como parâmetros os apontadores para a zona de memória (*buffer*) onde deve ser guardado o resultado (h) e o *buffer* onde se encontra a imagem original (I). Recebe ainda as dimensões da imagem (W e H);

- apesar de os *buffers* *h*, *I* e *inp* conterem dados que sabemos representarem quantidades bidimensionais o acesso é feito considerando cada *buffer* como uma estrutura unidimensional. Assim *buffer[y][x]* é acedido como *buffer[y*W+x]*, sendo *y* a linha a aceder, *x* a coluna e *W* a largura (número de colunas) da estrutura bidimensional.

```

1 static void kernel (int res[], int inp[], int ndx) {
2     res[ndx] = (inp[ndx-1] + inp[ndx] + inp[ndx+1]) / 3;
3 }
4
5 void convolve3x1 (int h[], int I[], int W, int H) {
6     register int x, y;
7
8     for (x=1 ; x<(W-1) ; x++) { // for each column of I
9         for (y=0 ; y<H ; y++) { // for each row of I
10
11             // compute h[y][x]
11             kernel (h, I, y*W+x);
12         } // y loop
13     } // x loop
14 }

```

Assembly

Gere o código *assembly* referente às duas rotinas apresentadas acima, escrevendo:

```
gcc -O0 -S -g convolve3x1.cpp
```

Abra o ficheiro *convolve3x1.s*, que contém o *assembly* gerado. Para facilitar a análise deste código tenha presente que o segundo argumento da directiva `.loc` indica o número da linha do código em C que corresponde às instruções *assembly* seguintes.

- 1 Desenhe o *activation record* da função `kernel()`. Para esse efeito localize a respectiva invocação na função `convolve3x1()`, observe a ordem com que os parâmetros são colocados na pilha (*stack*) e tenha presente que além dos parâmetros o *activation record* inclui ainda o endereço de retorno, o *frame pointer* anterior (`%ebp`), valores de registos que o compilador entenda salvar e variáveis locais.
- 2 Analise a forma como são calculados os endereços e lidos de memória os valores de `inp[ndx-1]`, `inp[ndx]` e `inp[ndx+1]`. Lembre-se que:
 - a. o cálculo do endereço de uma posição de um vector implica conhecer o endereço base (*B*), o índice do elemento a aceder (*I*) e o factor de escala (*s*). O endereço (*E*) é dado por $E = B + I*s$
 - b. O IA-32 dispõe de um modo de endereçamento constituído por 4 campos: Offset (*O*), Base (*B*), Índice (*I*) e factor de escala (*s*). O *O* e *s* são dados como valores imediatos (constantes) e *B* e *I* são dados como registos. Numa instrução estes aparecem no seguinte formato: `O(B, I, s)`.
O endereço é calculado como $E = B + I*s + O$.
 - c. A instrução *LEA* (*Load Effective Address*) permite calcular o endereço expresso no formato acima descrito sem aceder à memória. Esta instrução limita-se a efectuar os cálculos necessários para gerar um endereço e em alguns processadores usa uma unidade funcional específica designada por *AGU* (*Address Generation Unit*).
- 3 Desenhe o *activation record* da função `convolve3x1()`.
- 4 Identifique as instruções responsáveis pelos testes dos 2 ciclos presentes nesta função.
- 5 Familiarize-se com todo o código *assembly* associado a estas 2 funções.

Convolve 2D

O excerto de código abaixo implementa a convolução bidimensional sem qualquer optimização. Algumas notas sobre este código:

- recebe como parâmetros os apontadores para a zona de memória (*buffer*) onde deve ser guardado o resultado (*h*), o *buffer* onde se encontra a imagem original (*I*) e o *buffer* onde se encontra o filtro (*f*). Recebe ainda as dimensões da imagem (*W* e *H*) bem com a largura do filtro (*U*) que deve ser quadrado;
- Nas fronteiras da imagem (por exemplo, $y=0$) a vizinhança necessária para calcular a convolução não existe na totalidade. Nestes casos é utilizada a subregião da janela de vizinhança que existe para cada ponto. Por exemplo para $h[0][0]$ e um filtro de largura 3 são usados apenas os pontos $g[0][0]$, $g[0][1]$, $g[1][0]$ e $g[1][1]$.

```
1 void convolve2D (int *h, int *I, int W, int H, int *f, int U) {
2     int x, y, i, j;
3     int halfU;
4     int sumW;
5
6     halfU = U/2;
7     for (x=0 ; x<W ; x++) { // for each column of I
8         for (y=0 ; y<H ; y++) { // for each row of I
9
10            // compute h[y][x]
11            sumW = h[y*W+x] = 0;
12            for (i=-halfU ; i<=halfU ; i++) {
13                // verify I horizontal bounds
14                if (x+i<0 || x+i>=W) continue;
15
16                for (j=-halfU ; j<=halfU ; j++) {
17                    // verify I vertical bounds
18                    if (y+j<0 || y+j>=H) continue;
19
20                    h[y*W+x] += (f[(j+halfU)*U+i+halfU] * I[(y+j)*W + (x+i)]);
21                    sumW += f[(j+halfU)*U+i+halfU];
22                }
23            }
24            h[y*W+x] /= (sumW ? sumW : 1);
25        } // y loop
26    } // x loop
27 }
```

Gere o código *assembly* referente à rotina apresentada acima, escrevendo:

```
gcc -O0 -S -g convolve.cpp
```

Abra o ficheiro `convolve.s`, que contém o *assembly* gerado.

- 1 Desenhe o *activation record* da função `convolve2D()`.
- 2 Analise a forma como são calculados os endereços e lidos de memória os valores de `f[]` e `I[]`.
- 3 Identifique as instruções responsáveis pelos testes dos 4 ciclos presentes nesta função.
- 4 Familiarize-se com todo o código *assembly* associado a esta função.