RELATÓRIO DE ANÁLISE DE COMPLEXIDADE

1 Introdução

Na implementação do código image8bit.c, foram criadas diversas funções para manipular imagens de diferentes formas. Para além de implementar, é necessário analisá-las para garantir um código mais eficiente. Para esse propósito, foram analisados os algoritmos das funções ImageLocateSubImage() e ImageBlur().

2 ImageLocate

A função ImageLocateSubImage() é usada para determinar se a imagem2 é uma parte da imagem1. Para esta função funcionar corretamente, esta chama a função ImageMatchSubImage() para comparar os pixeis de modo a garantir que a imagem2 é, de facto, igual a uma parte da imagem1.

2.1 Eficiência computacional da ImageLocateSubImage()

Para analisar a sua eficiência computacional, foi registado numa tabela Tabela 1 o número de comparações envolvendo uma sequência de testes com diferentes imagens.

Img1	**Img2**	**Iterações da Locate**	**Iterações da Match**	**Observações**
original.pgm (300x300)	crop.pgm (100x100)	3010	10000	
original.pgm (300x300)	rotate.pgm (300x300)	90000	90000	img2 doesn't fit if it starts at another pixel other than (0,0) in img1
ireland.pgm (1600x1200)	airfield (1600x1200)	1920000	1920000	img2 doesn't fit in img1, if it starts in a different ixel than (0,0)
airfield (1600x1200)	ireland (640x480)	1920000	1	first pixel of img2 is different than img1, so the loop is only executed one time
airfield (1600x1200)	airfieldCrop (100x100)	1	10000	
airfield (1600x1200)	airfieldCrop (10x10)	1	100	
airfield (1600x1200)	airfieldCrop (10x10) (no ponto 1549,1189)	1903950	1898830	

Tabela 1: Tabela com dados resultantes das iterações da função ImageLocateSubImage() e ImageMatchSubImage()

- **Best Case:** Imagens com pouca resolução e a imagem2 está localizada no pixel (0,0)
- **Average case:** Imagem tem pouca resolução e a imagem2 não se encontra localizada nela
- **Worst case:** Imagens com grande resolução e a imagem 2 não se encontra localizada nela ou a imagem2 é uma porção pequena do canto inferior direito da imagem2

2.2 Funcionamento da ImageLocateSubImage()

Este algoritmo segue uma lógica básica por detrás. Sâo enviadas duas imagens, img1 e img2, e é feita a comparação entre os pixeis da img1 e da img2, sendo o ínicio no pixel (0,0) até ao pixel (*width*,height), tal como consta na Figura 1.

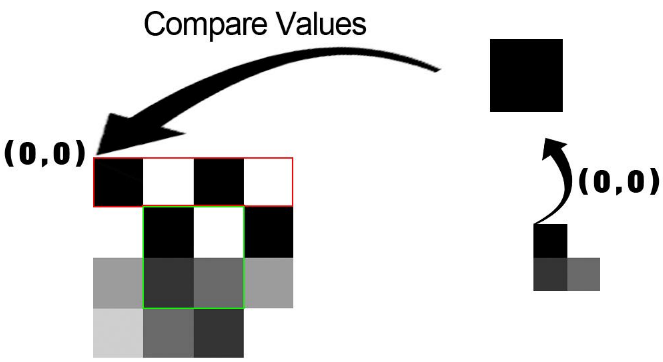

Figura 1: Figura 1: Figura representativa do funcionamento da ImageLocateSubImage()

Após detetar um valor igual ao primeiro píxel da img2 na img1, é ainda comparado os restantes píxeis para garantir que são iguais, podendo afirmar que img2 é, de facto, uma subimagem da img1. Para fazer essa comparação é usado o seguinte excerto de código, onde i e j são, respetivamente, as coordenadas x e y da img2, e são comparados os pixeis equivalentes. Para isso é adicionado ao i e j as coordenadas da img1, caso estejamos a localizar a img2 no ponto (x,y) que não seja (0,0) da img1.

```
for(int j =0;j < img2->height;j++) {
    for(int i = 0;i < img2->width;i++) {
      if (ImageGetPixel(img1,x + i ,y + j) != ImageGetPixel(img2, i, j)) {
        return 0}
    }
}
```

3 Image Blur

A seguinte função visa a implementar um filtro de imagem de *blur*, em que é aplicado um filtro da média de pixeis num dado retângulo de dimensão $(2dx + 1) \times (2dy + 1)$. Para tal, foram considerados dois algoritmos, detalhados nas subsecções seguintes.

3.1 Algoritmo Melhorado

Este algoritmo utiliza uma *Summed-area table*, definido por um *array* bidimensional, com tamanho $width \times height$. Este pode ser dividido em duas partes: cálculo da soma das áreas em cada pixel e cálculo do *blur* numa determinada área.

3.1.1 Summed-area table

O seguinte método implementa uma forma de, em um dado ponto (x, y), obter a soma de todos os valores dos pixeis desde $(0, 0)$ a esse mesmo ponto. Da forma em que foi implementado, apenas é feita a média dos valores na atribuição na própria imagem, que resulta em o *array* ser do tipo inteiro.

A forma geral para a implementação deste algoritmo é:

```
integral[x][y] = ImageGetPixel(img, x, y) +
integral[x - 1][y] +
integral[x][y - 1] -
integral[x - 1][y - 1];
```

Com o seguinte bloco de código, está a ser atribuído às coordenadas (x, y) do *array* criado anteriormente a soma total dos valores dos pixeis.

Esta fórmula é visualmente representada na figura abaixo.

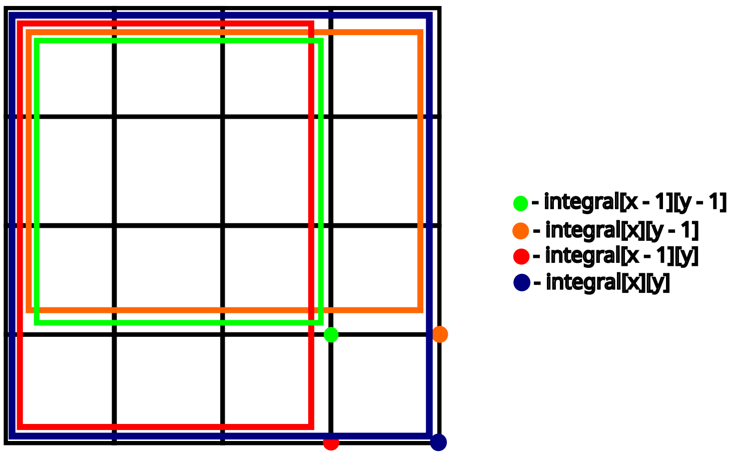

Figura 2: Figura ilustrativa do algoritmo *Summed-area table*

Ainda considerando o algoritmo anterior, Verifica-se um problema de *assertion*, causado quando x ou y são 0. Para evitar esse problema, é atribuído inicialmente ao $(0,0)$ (já que a soma total nesse ponto é ele próprio) o seu próprio valor.

De seguida, são preenchidas as linhas $x = 0$ e $y = 0$. Após isso, é possível preencher o resto do *array* com a fórmula geral anteriormente apresentada.

3.1.2 Média dos pixeis

O seguinte algoritmo é responsável por calcular a média dos pixeis numa determinada área, definida por um retângulo de dimensão $(2dx + 1) \times (2dy + 1)$. São usados os valores de soma obtidos na parte anterior.

Inicialmente, são definidos o canto superior esquerdo e canto inferior direito, e verificar se o retângulo está dentro da imagem. Caso não esteja, é lhe atribuído o valor representante do extremo que está a ultrapassar.

Finalmente, é calculada a média dos pixeis, com o seguinte bloco de código:

```
(double)((integral[x2 - 1][y2 - 1] -
integral[x1][y2 - 1] -
integral[x2 - 1][y1] +
integral[x1][y1])) /
npixels + 0.5;
```

O valor de 0.5 é adicionado para que o valor seja arredondado corretamente, já que o valor é do tipo inteiro. Também é subtraído o valor de 1 a $x2$ e $y2$, de forma a que o valor seja o correto, já que o *array* começa em $(0,0)$.

3.1.3 Análise da Complexidade

Na seguite tabela, é possível observar a complexidade de cada função, em termos de iterações de ciclos e de operações aritméticas das duas partes do algoritmo.

Imagem	**Fator de *blur***	**Iterações de 3.1.1.**	**Iterações de 3.1.2**	**Complexidade**
original.pgm (300x300)	$7,7$	90000	90000	$O((width \times height) \times 2)$
original.pgm (300x300)	$40,40$	90000	90000	$O((width \times height) \times 2)$
original.pgm (300x300)	$200,200$	90000	90000	$O((width \times height) \times 2)$
airfield.pgm (1600x1200)	$40,40$	1920000	1920000	$O((width \times height) \times 2)$
airfield.pgm (1600x1200)	$200,200$	1920000	1920000	$O((width \times height) \times 2)$
ireland.pgm (640x480)	$40,40$	307200	307200	$O((width \times height) \times 2)$

Tabela 2: Tabela com a complexidade das funções

RELATÓRIO DE ANÁLISE DE COMPLEXIDADE

Como é possível observar na tabela acima, a complexidade das funções é $O((width \times height) \times 2)$, já que o número de iterações é igual ao número de pixeis da imagem. A complexidade das operações aritméticas é linear, já que o número de operações é igual ao número de pixeis da imagem. Isto resulta em:

- **Best case** - A imagem ser pequena;
- **Average case** - Não existe um caso médio, já que a complexidade é sempre a mesma;
- **Worst case** - A imagem ser grande.

3.2 Algoritmo Básico

Como o nome indica, este algoritmo é mais básico, que acaba por ser menos eficiente que o anterior. Este algoritmo baseia-se em ir de pixel a pixel da imagem, e de pixel a pixel da área definida pelo retângulo de dimensão $(2dx + 1) \times (2dy + 1)$, verificar se esse pixel se encontra dentro da imagem e calcular a média dos pixeis nessa área dependendo do resultado.

3.2.1 Análise da complexidade

Abaixo é apresentado uma tabela com a complexidade geral do algoritmo, em termos de iterações de ciclos.

Imagem	**Fator de *blur***	**Iterações de 3.2.1.**	**Complexidade**
original.pgm (300x300)	$7, 7$	20250000	$O((2dx + 1) \times (2dy + 1) \times w \times h)$
original.pgm (300x300)	$40, 40$	590490000	$O((2dx + 1) \times (2dy + 1) \times w \times h)$
original.pgm (300x300)	$200, 200$	14472090000	$O((2dx + 1) \times (2dy + 1) \times w \times h)$
airfield.pgm (1600x1200)	$40, 40$	12597120000	$O((2dx + 1) \times (2dy + 1) \times w \times h)$
airfield.pgm (1600x1200)	$200, 200$	308737920000	$O((2dx + 1) \times (2dy + 1) \times w \times h)$
ireland.pgm (640x480)	$40, 40$	2015539200	$O((2dx + 1) \times (2dy + 1) \times w \times h)$

Tabela 3: Tabela com a complexidade das funções

Como é possível observar na tabela acima, a complexidade das funções é $O((2dx + 1) \times (2dy + 1) \times w \times h)$, já que o número de iterações está dependente desta vez não só pelo tamanho da imagem, como também pelo tamanho do fator do blur. Isto resulta em:

- **Best case** - A imagem ser pequena e o fator de blur ser pequeno;
- **Average case** - Um *balance* entre o tamanho da imagem e o fator de blur;
- **Worst case** - A imagem ser grande e o fator de blur ser grande.

3.3 Conclusão