

**Universidade Estadual do Oeste do Paraná - UNIOESTE**  
**Centro de Ciências Exatas e Tecnológicas - CCET**  
**Curso de Ciência da Computação**

## **PROCESSAMENTO DE IMAGENS DIGITAIS**

**CASCABEL - PR  
2022**

## SUMÁRIO

<b>UNIDADE 1 – ARMAZENAMENTO DE IMAGENS .....</b>	<b>1</b>
1.1 MECANISMOS PARA ARMAZENAMENTO DE IMAGENS .....	1
1.2 ARMAZENANDO IMAGENS .....	2
1.3 PALETA DE CORES .....	3
1.3.1 Modelo RGB/CMYK .....	4
1.3.2 Modelo HSV .....	5
1.3.3 Look-up Tables ou Palettes .....	5
1.4 FORMATOS DE IMAGENS ( <a href="http://www.dcs.ed.ac.uk/home/mxr/gfx">www.dcs.ed.ac.uk/home/mxr/gfx</a> ) .....	6
1.4.1 Formato BMP .....	6
1.4.2 Formato FLIC .....	12
<b>UNIDADE 2 – PROCESSAMENTO DE IMAGENS.....</b>	<b>20</b>
2.1 FUNDAMENTOS DE IMAGENS DIGITAIS.....	20
2.1.1 Paradigmas de Abstração de Imagens .....	20
2.2 MODELO ESPACIAL DE IMAGEM .....	20
2.2.1 Imagem Contínua .....	21
2.2.2 Representação de uma Imagem .....	22
2.2.3 Imagem Digital .....	24
2.2.4 Topologia Digital e Representação Matricial .....	25
2.2.5 Geometria do Pixel .....	26
2.3 COMPONENTES DE UM SISTEMA DE PROCESSAMENTO DE IMAGENS .....	27
2.3.1 Aquisição .....	28
2.3.2 Armazenamento .....	28
2.3.3 Processamento .....	29
2.3.4 Transmissão .....	29
2.3.5 Exibição .....	29
2.4 ELEMENTOS DE SISTEMAS DE VISÃO ARTIFICIAL .....	31
2.4.1 Estrutura de um Sistema de Visão Artificial .....	31
2.4.2 Domínio do Problema e Resultado .....	32
2.4.3 Aquisição da Imagem .....	32
2.4.4 Pré-Processamento .....	33
2.4.5 Segmentação .....	33
2.4.6 Extração de Características .....	33
2.4.7 Reconhecimento e Interpretação .....	33
2.4.8 Base de Conhecimento .....	34
2.5 PROPRIEDADES ESTATÍSTICAS DE IMAGENS .....	34
2.5.1 Histograma .....	34
2.5.2 Média Amostral .....	38
2.5.3 Variância .....	38
2.5.4 Covariância .....	38
2.5.5 Correlação .....	38
2.6 DISCRETIZAÇÃO DE COR .....	39
2.6.1 Célula de Quantização .....	40
2.6.2 Percepção e Quantização .....	41
2.6.3 Método Geral de Quantização .....	43
2.6.4 Erro de Quantização .....	43
2.6.5 Quantização Uniforme e Adaptativa .....	44
2.6.6 Métodos Adaptativos de Quantização .....	45

2.7 TÉCNICAS DE MODIFICAÇÃO DE HISTOGRAMAS .....	51
2.7.1 Transformações de Intensidade .....	51
2.7.2 Uniformização das Médias e Variâncias de imagens.....	53
2.7.3 Equalização de Histograma.....	54
2.7.4 Especificação Direta de Histograma.....	57
2.7.5 Limiarização (Thresholding) .....	59
2.8 OPERAÇÕES LÓGICAS E ARITMÉTICAS COM IMAGENS.....	64
2.8.1 Operações Aritméticas Pixel a Pixel.....	64
2.8.2 Operações Lógicas Pixel a Pixel .....	67
2.8.3 Operações Orientadas a Vizinhança .....	68
2.9 OPERAÇÕES DE CONVOLUÇÃO COM MÁSCARAS.....	69
2.9.1 Detecção de Pontos Isolados e Detecção de Linhas.....	71
2.9.2 Detecção de Bordas .....	72
2.10 FILTRAGEM, REALCE E SUAVIZAÇÃO DE IMAGENS.....	75
2.10.1 Filtragem no Domínio Espacial.....	75
2.10.2 Filtragem no Domínio da Frequência.....	77
2.10.3 Suavização de Imagens no Domínio Espacial .....	77
2.10.4 Realce de Imagens no Domínio Espacial .....	88
<b>UNIDADE 3 – MORFOLOGIA MATEMÁTICA .....</b>	<b>95</b>
3.1 DILATAÇÃO E EROSÃO.....	95
3.1.1 Definições Básicas .....	95
3.1.2 Dilatação .....	96
3.1.3 Erosão.....	97
3.2 ABERTURA E FECHAMENTO .....	99
3.2.1 Interpretação geométrica da abertura e do fechamento .....	100
3.3 TRANSFORMAÇÃO HIT-OR-MISS.....	102
3.4 ALGORITMOS MORFOLÓGICOS BÁSICOS.....	103
3.4.1 Extração de contornos.....	103
3.4.2 Preenchimento de Regiões ( <i>Region Filling</i> ).....	103
3.4.3 Extração de Componentes Conectados .....	105
3.4.4 Casco Convexo ( <i>Convex Hull</i> ).....	106
3.4.5 Afinamento ( <i>Thinning</i> ) .....	106
3.4.6 Espessamento ( <i>Thickening</i> ) .....	108
3.4.7 Esqueletos .....	109
3.4.8 Poda ( <i>Pruning</i> ) .....	111
<b>UNIDADE 4 – COMBINAÇÃO ALGÉBRICA DE IMAGENS .....</b>	<b>113</b>
4.1 MISTURA DE IMAGENS .....	114
4.2 COMBINAÇÃO DE IMAGENS POR DECOMPOSIÇÃO .....	114
4.3 COMPOSIÇÃO DE IMAGENS.....	115
4.4 COMBINAÇÃO NO DOMÍNIO DISCRETO .....	116
4.5 A FUNÇÃO DE OPACIDADE .....	118
4.6 DISCRETIZAÇÃO DA FUNÇÃO DE OPACIDADE .....	119
4.6.1 Discretização com Canal Alfa.....	119
4.6.2 Discretização com Máscara de Bits.....	120
4.7 CÁLCULO DA FUNÇÃO DE OPACIDADE .....	121
4.7.1 Imagens Sintéticas Bidimensionais .....	121
4.7.2 Imagens Sintéticas Tridimensionais .....	122
4.7.3 Imagens Digitalizadas .....	122
4.8 COMPOSIÇÃO NO DOMÍNIO DISCRETO .....	123
4.8.1 Composição com Canal de Opacidade .....	123
4.8.2 Composição com Máscara de Bits .....	124

4.9 OPERAÇÕES DE COMPOSIÇÃO .....	126
4.9.1 Operador “OVER”.....	127
4.9.2 Operador “INSIDE”.....	128
4.9.3 Operador “OUTSIDE” .....	129
4.9.4 Operador “ATOP” .....	129
4.9.5 Operador “XOR” .....	130
4.9.6 Operador “CLEAR” .....	131
4.9.7 Operador “SET”.....	132
<b>UNIDADE 5 – WARPING E MORPHING.....</b>	<b>133</b>
5.1 FILTRO DE WARPING.....	133
5.2 EXPANSÃO E CONTRAÇÃO.....	134
5.3 WARPING NO DOMÍNIO DISCRETO .....	135
5.4 RECONSTRUÇÃO E REAMOSTRAGEM .....	137
5.5 WARPING POR TRANSFORMAÇÃO PROJETIVA.....	138
5.6 “ZOOM” DE UMA IMAGEM .....	140
5.6.1 “Zoom In” com o Filtro Box .....	141
5.6.2 “Zoom In” com o Filtro Triangular .....	142
5.6.3 “Zoom Out” de uma Imagem .....	142
5.7 MORPHING.....	143

## UNIDADE 1 – ARMAZENAMENTO DE IMAGENS

### 1.1 MECANISMOS PARA ARMAZENAMENTO DE IMAGENS

Quando armazenamos uma imagem, armazenamos uma matriz bidimensional de valores, onde cada valor representa um dado associado com um pixel na imagem. Para um bitmap monocromático, estes valores são dígitos binários. Para uma imagem colorida, o valor pode ser uma tupla de 3 números representando as intensidades das componentes vermelho, verde e azul da cor daquele pixel, ou então 3 números que são índices nas tabelas de intensidades do vermelho, verde e azul, ou ainda um índice para uma tabela de tuplas de cores, ou um índice para qualquer estrutura de dados que possa representar cores.

Somado a isso, cada pixel pode ter outras informações a ele associadas, como o valor z-buffer (profundidade do pixel), uma tupla de números indicando o vetor normal à superfície, etc. Assim, podemos considerar uma imagem como uma coleção de canais, onde cada um deles nos dá alguma informação sobre o pixel na imagem. Desta forma, podemos falar dos canais vermelho, verde e azul de uma imagem.

Antes de discutir algoritmos para armazenamento de imagens em vetores ou canais, descreveremos dois métodos importantes para armazenamento de imagens: o uso do ***metafile*** e o uso dos ***dados dependentes da aplicação***. Nenhum deles é, estritamente falando, um formato de imagem; são mecanismos para expressar a informação que é representada na imagem.

Se uma imagem é produzida por uma sequência de chamadas a alguma coleção de rotinas, o armazenamento metafile desta sequência de chamadas é preferível ao armazenamento da imagem que foi gerada. Esta sequência de chamadas pode ser muito mais compacta do que a própria imagem; por exemplo, a bandeira do Japão pode ser desenhada por uma chamada a uma rotina que desenhe um retângulo branco e uma chamada a uma rotina que desenhe um círculo vermelho. Se as rotinas são suficientemente simples ou estão implementadas em hardware, redesenhar uma imagem metafile pode ser mais rápido do que redesenhar um mapa de pixels. O termo ***metafile*** é usado também em descrições de estruturas de dados padronizadas independentes de dispositivos. Para armazenar uma imagem como um metafile, percorremos a estrutura de dados corrente e a armazenamos numa forma independente de dispositivo para posterior exibição. Esta descrição pode não ser uma sequência de chamadas a funções, mas uma transcrição textual da estrutura hierárquica da imagem.

O segundo esquema de armazenamento obriga que os dados sejam dependentes de uma aplicação. Se uma aplicação é utilizada para exibir uma classe particular de imagens, pode ser conveniente gravar as informações de como estas imagens foram criadas, ou mesmo diferenças para um conjunto padrão de dados. Por exemplo: se todas as imagens são visões diretas de faces humanas descritas por polígonos, pode ser mais simples armazenar uma lista dos polígonos cujas posições diferem da posição numa imagem facial padrão. Uma versão extrema deste tipo de condensação de informações foi utilizada no projeto “Cabeças Falantes” do MIT Media Lab, onde somente as posições dos olhos, lábios e outras características relevantes são armazenadas. Até este ponto, a descrição da imagem é muito mais uma descrição da cena no domínio da modelagem, do que no domínio do armazenamento de imagens.

## 1.2 ARMAZENANDO IMAGENS

Agora vamos considerar como armazenar as imagens que consiste em uma série de canais de dados. Se nosso dispositivo de exibição espera receber informações sobre uma imagem na forma de tuplas RGB, pode ser mais conveniente armazenar a imagem como tuplas RGB. Mas se o espaço para armazenamento é restrito, caso mais freqüente, então pode ser que valha a pena tentar comprimir os canais de alguma maneira. Métodos de compressão devem considerar o custo da descompressão. A técnica de compressão mais sofisticada provavelmente apresenta a descompressão mais cara. Embora todas as técnicas aplicam-se igualmente bem para qualquer canal de informação, nossos estudos considerarão apenas os canais de cores, uma vez que estes são os mais freqüentes em imagens.

Se uma imagem tem poucas cores e cada canal ocorre muitas vezes (como num jornal, onde há somente preto, cinza escuro, cinza claro e branco), pode ser que valha a pena criar uma tabela de ocorrência de cores, com apenas quatro entradas, e criar um canal simples que é um índice para esta tabela de cores.

Em nosso exemplo, este canal simples precisará apenas 2 bits de informação por pixel; isto é melhor que os 8 bits por cor por pixel. A imagem resultante foi comprimida por um fator 12:1. Em imagens com mais cores, a economia é menos substancial; no caso extremo, onde cada pixel da imagem é uma cor diferente, a paleta de cores será tão grande quanto à imagem armazenada através de tuplas RGB, e os índices nesta paleta de cores tomariam ainda mais espaço. A grosso modo, indexar a imagem numa tabela de cores é interessante se o número de cores é menor do que metade do número de pixels.

Este canal simples ainda requer ao menos uma pequena informação por pixel. Se a imagem possuir um grande número de repetições, pode ser possível comprimi-la mais pela codificação ***run-length*** do canal. A codificação ***run-length*** consiste num contador e num valor, onde o contador indica o número de vezes que o valor repete.

O projeto do Utah Raster Toolkit inclui algumas melhorias nesta idéia básica. Por exemplo, o contador  $n$  é um ***signed int*** de 8 bits (variando de -128 até 127): um contador negativo indica que o valor dos  $n$  pixels seguintes não estão comprimidos; um contador não negativo indica que o próximo valor aplica-se a  $n + 1$  pixels. Outras melhorias incluem a reserva de certos valores negativos para significados especiais: -128 indica que os próximos dois valores indicam uma scanline com a posição para onde será o salto.

Há outros formatos espertos para compressão de canais. Por exemplo, podemos armazenar o valor de cada pixel de um bitmap como um inteiro (com um 0 ou 1), mas muitos dos bits de um inteiro seriam desperdiçados. Ao invés disso, podemos armazenar o valor de um pixel em cada bit (isto é a origem do termo bitmap). Se a imagem representada contém regiões preenchidas com padrões cuja largura é um fator de 8, então podemos utilizar uma codificação similar à ***run-length***, na qual os primeiro byte representará um contador  $n$  e o próximo byte representará um padrão a ser repetido para os próximos  $8n$  pixels do bitmap.

A codificação ***run-length*** e outras aproximações teóricas padrão como a codificação Huffman tratam a imagem em canais, que podem ser imaginados como um vetor linear de valores (assim, múltiplos canais podem ser considerados como um único canal simples, de forma que possamos usar o ***run-length*** para codificar conjuntos de triplas RGB). Outros métodos tratam a imagem como um vetor 2D de valores, e então podem explorar qualquer coerência entre linhas. Uma destas técnicas está baseada no uso de ***quadtrees***.

O princípio fundamental da descrição de imagens baseada em ***quadtrees*** é que uma região de uma imagem pode ser constante e, portanto, todos os pixels desta região podem ser tratados como de mesmo valor. Determinar esta região aproximadamente constante é o coração deste algoritmo. Este algoritmo pode ser usado tanto em uma componente simples da imagem, como o array da componente vermelho, ou sobre a componente RGB associada com cada pixel; para simplificar, descreveremos o algoritmo para uma componente numérica simples. O algoritmo requer um mecanismo para determinar o valor médio de uma região da imagem, e a extensão dos desvios da média dentro da região.

A imagem é primeiramente considerada como um todo. Se o desvio da média dentro da imagem é suficientemente pequeno (menor ou igual a uma tolerância não negativa), então a imagem é relatada como tendo um valor igual à média, repetida sobre a imagem inteira. Se o desvio da média não é menor do que a tolerância, então a média da imagem é armazenada, dividida em quadrantes e o mesmo algoritmo é aplicado para cada quadrante. O algoritmo termina quando a subdivisão dos quadrantes conduzir a uma região com apenas um pixel; para uma região com apenas um pixel, o desvio da média deve ser igual a zero e portanto menor ou igual a qualquer tolerância.

Podemos melhorar o algoritmo gravando não apenas a média da imagem, mas as médias dos quatro quadrantes sempre que a imagem for subdividida, e a média da imagem quando não for subdividida. A vantagem é que quando a imagem for reexibida, percorrendo a quadtree primeiramente na largura, esta pode ser constantemente atualizada mostrando mais e mais detalhes da imagem. A primeira imagem é formada por quatro retângulos coloridos. Então, cada retângulo é subdividido e suas cores refinadas, e assim por diante. Num sistema projetado para visualizar um grande número de imagens, este método pode ser extremamente conveniente: depois de processar alguns bytes de informação, é possível ter uma visão aproximada da imagem; então o usuário pode optar por rejeitar a imagem atual e ir para a próxima. Esta rápida aproximação da imagem é especialmente útil se as imagens são transmitidas através de um canal de comunicação em banda-estreita.

### 1.3 PALETA DE CORES

Paleta, Palette, Look-Up Tables, são termos genericamente empregados para referir-se à mesma técnica de codificação de cores. O uso da paleta teve sua origem nas limitações de hardware impostas há alguns anos ou na simples conveniência de processamento menos oneroso.

Para melhor entender o uso das paletas é necessário primeiramente entender como nosso organismo percebe as cores ao seu redor e quais as limitações que enfrentamos ao tentar transpor esta realidade ao mundo digital dos computadores.

Imagens do mundo real apresentam tipicamente uma infinita variedade de cores. O olho humano é, no entanto, capaz de capturar, entender e discernir apenas alguns “poucos” milhões de cores. A “limitada” capacidade de entendimento de cores do olho e do cérebro humano é muito maior do que a maioria dos dispositivos eletrônicos de captura e apresentação de imagens são capazes de prover.

Na captura e apresentação de cores da natureza somos obrigados a traduzir uma realidade multidimensional e infinita em um modelo finito para manipulação e arquivamento. Existem vários modelos de transposição de cores do mundo real para o mundo digital aceitos na comunidade de processamento de

imagens; destacam-se entretanto os modelos RGB/CMYK e HSV.

### 1.3.1 Modelo RGB/CMYK

O modelo RGB codifica as cores utilizando a combinação de 3 cores básicas: Vermelho (R), Verde (G) e Azul (B) para equipamentos emissores de luz (telas de computador). É por isso conhecido como um modelo aditivo.

O modelo CMYK é um modelo complementar ao modelo RGB, empregado em dispositivos ou produtos não emissores de luz, atuando na subtração dos componentes de cor da luz incidente. A representação gráfica destes modelos pode ser melhor entendida pela visualização da Figura 1.1.

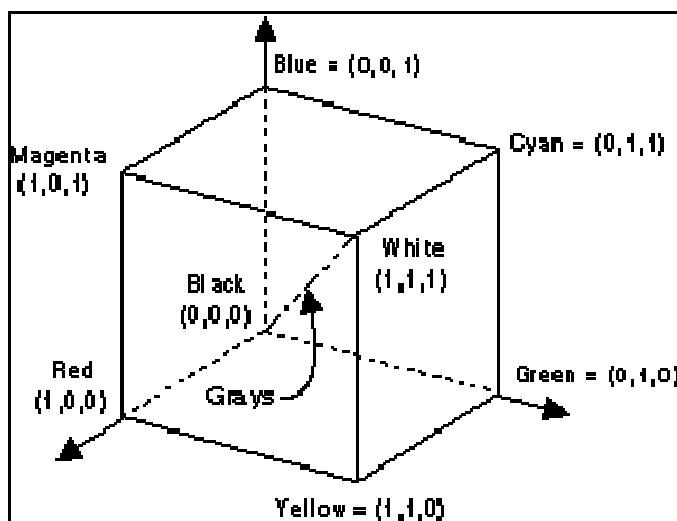


Figura 1.1 – Cubo RGB/CMYK

Embora complementares, os modelos RGB e CMYK não produzem sempre os mesmos resultados visuais, ou seja, não existe a transposição exata e precisa de cores de um modelo para outro. Existem cores de um modelo que simplesmente não podem ser expressas pelo modelo complementar enquanto algumas cores encontradas na natureza simplesmente não podem ser expressas por nenhum dos dois modelos (cores metálicas e cores fluorescentes). A Figura 1.2 ilustra a equivalência real de cores entre os modelos RGB e CMYK para valores de cores teoricamente equivalentes.

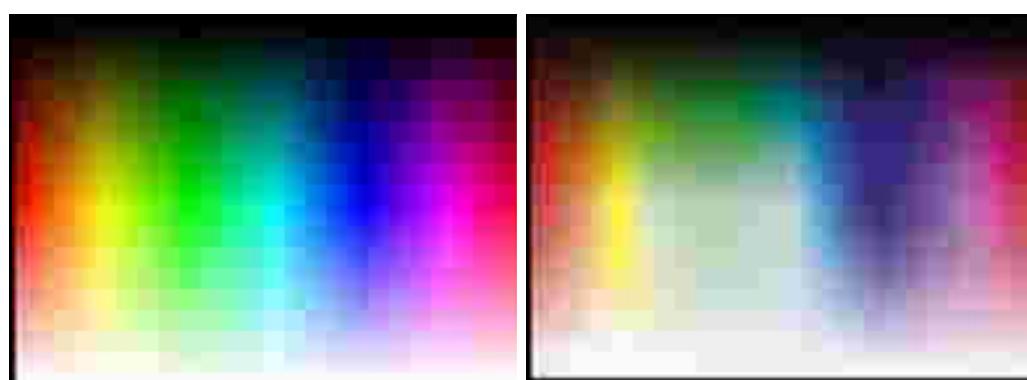


Figura 1.2 – Paleta RGB e CMYK

Apesar de suas limitações, o par RGB/CMYK é o modelo de cores

atualmente mais popular para processamento digital de imagens, principalmente em função da tecnologia dos tubos de imagens de computadores que se utilizam de canhões de 3 elétrons (RGB) para a composição de imagens coloridas no vídeo, e das impressoras a jato de tinta ou térmicas que utilizam tecnologia CMYK herdada da indústria fotográfica.

### 1.3.2 Modelo HSV

O modelo HSV foi criado a partir de uma concepção intuitiva da maneira de trabalhar de um artista ao misturar cores para obter o correto sombreamento e na obtenção de tons intermediários. A seleção e obtenção de cores no modelo HSV é muito mais intuitiva que nos modelos RGB e CMYK. Seu princípio baseia-se no controle dos valores de **Hue**, **Saturation** e **Value** (HSV).

**Hue** é a componente que seleciona a "tinta" em uso, sendo controlada pela posição angular de um ponteiro numa roda de cores definida de 0 a 359.

**Saturation** é a componente que determina a pureza da cor selecionada em **Hue**. Todos os tons de cinza possuem **Saturation** zero e todos os **Hues** puros possuem **Saturation** 1.

**Value** regula o brilho da cor determinada por **Hue** e **Saturation**. A cor preta possui brilho zero e qualquer valor de **Hue** ou **Saturation**. O valor 1 de **Value** determina uma intensidade pura de **Hue + Saturation**. A representação gráfica destes modelos pode ser melhor entendida pela visualização da Figura 1.3.

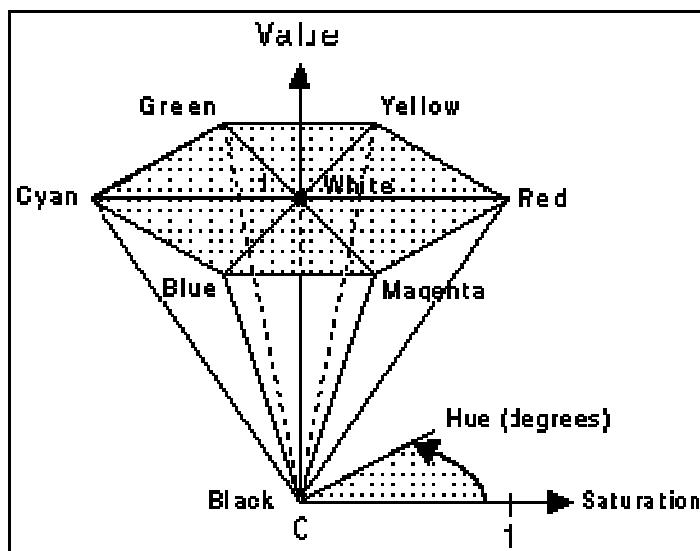


Figura 1.3 – O Modelo HSV

### 1.3.3 Look-up Tables ou Palettes

O principal uso de **Palettes** (**Look-up tables**) deve-se quase que exclusivamente à necessidade de processamento e armazenamento. Imagens que se utilizam de **palettes** são significativamente menores do que imagens que codificam a cor de seus pixels utilizando um dos modelos acima mencionados. A contrapartida encontra-se na qualidade da cores. **Palettes** nada mais são que mapas de cores limitados, escolhidas criteriosamente para serem utilizadas na apresentação ou captura de uma imagem. Como dissemos anteriormente, mesmo

quando utilizamos um modelo de conversão de cores para "digitalizar" uma imagem do mundo real, o número de cores presentes numa imagem digital é ainda muito alto situando-se muito comumente acima das 10.000 cores.

A representação de uma imagem de 800x600 pixels em uma codificação RGB de 24 bits (8 bits para cada cor) consome 1.440.000 bytes de memória. Se esta mesma imagem possuir algo como 10.000 cores podemos criar uma tabela de cores RGB com 10.000 posições com apenas 30.000 bytes. Ao invés de codificarmos cada pixel da imagem com uma trinca RGB de 3 bytes, usariamos apenas 2 bytes para designar uma cor na nossa tabela de cores (palette) com a qual se deve fazer o display do pixel. Este esquema poderia fazer com que a mesma imagem de 800x600 pudesse ser manipulada com apenas 990.000 bytes. No entanto em virtude das muitas incertezas relativas ao número de cores de uma imagem e ao esforço computacional necessário para calculá-lo, as **palettes** estão normalmente limitadas a 256 cores, necessitando então apenas 1 byte por pixel para endereçá-las, como no esquema da Figura 1.4.

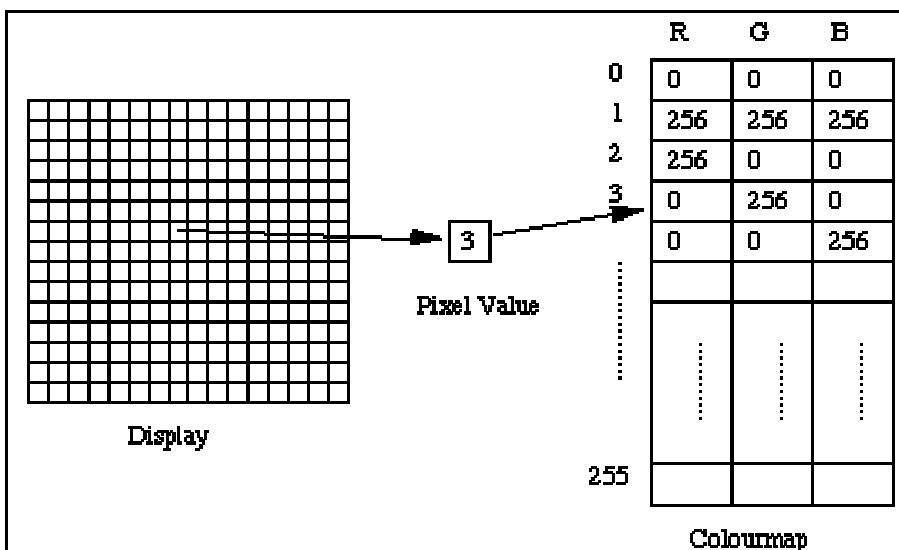


Figura 1.4 – Mapeamento de Paleta

## 1.4 FORMATOS DE IMAGENS ([www.dcs.ed.ac.uk/home/mxr/gfx](http://www.dcs.ed.ac.uk/home/mxr/gfx))

### 1.4.1 Formato BMP

O formato de codificação adotado pela Microsoft para arquivamento de imagens em seus sistemas operacionais MS Windows é o formato BMP. A estrutura destes arquivos é também utilizada para armazenamento de imagens em memória pelo próprio sistema operacional, assim sendo arquivos BMP podem ser lidos diretamente do disco para as áreas de arquivamento de imagens em memória do Windows, necessitando de muito pouco pré-processamento.

- Formatos suportados pelo padrão BMP:

Imagens manipuladas em memória para apresentação no MS Windows são internamente armazenadas em formato DIB (Device-Independent Bitmap) que permite ao Windows apresentar a imagem em qualquer tipo de dispositivo suportado

pelo sistema operacional (telas, impressoras, projetores, etc...). O termo "device independent" embute o conceito de especificação de cores da imagem de uma maneira independente das limitações do dispositivo padrão de saída.

- **Estrutura interna do formato BMP:**

Cada arquivo BMP é composto por um cabeçalho (**header**), por uma área de informação do cabeçalho (**bitmap-information header**), uma tabela de cores (**color table**) e uma sequência de bits (**bitmap bits**) que definem o conteúdo da imagem.

Assim sendo o arquivo possui a seguinte forma genérica:

```
BITMAPFILEHEADER    bmfh;
BITMAPINFOHEADER   bmih;
RGBQUAD            aColors [];
BYTE               aBitmapBits [];
```

O "**bitmap-file header**" (bmfh) contém informações referentes ao tipo, tamanho e formato do arquivo. O header é definido como uma estrutura BITMAPFILEHEADER.

A "**bitmap-information header**" (bmih) é definida pela estrutura BITMAPINFOHEADER e contém informações a respeito do tamanho, tipo de compressão e codificação de cores da imagem.

A "**color table**" (aColors[]) é definida como um vetor de estruturas do tipo RGBQUAD e contém quantos elementos forem necessários para acomodar a codificação da paleta de cores adotada pelo arquivo em questão. A paleta de cores, obviamente, não é utilizada para imagens de 24 bits pois o seu tamanho superaria em muito o tamanho da própria imagem. As cores na tabela são apresentadas por ordem decrescente de ocorrência. Esta prioridade possibilita ao driver do dispositivo otimizar o seu uso de maneira a apresentar os dados da imagem da melhor maneira possível no dispositivo de saída.

A sequência de bits descrita acima como "**bitmap bits**" (aBitmapBits[]), segue-se imediatamente à "**color table**" e consiste de uma sequência de valores do tipo BYTE, representantes das linhas (**scan lines**) que compõem a imagem. Cada "**scan line**" consiste de uma sequência de bytes que representam uma codificação de cor para cada pixel da linha, no sentido esquerda → direita.

O número de bytes utilizados para representar cada uma das linhas e seus pixels é dependente do formato de cor adotado pela imagem e da largura da imagem. Por definição, o comprimento de cada linha deve ser múltiplo de 4, ou seja base 32 bits, que traz melhor desempenho na manipulação em memória. Os pixels por ventura excedentes numa linha deverão ser preenchidos com o valor "**null**".

A ordem de apresentação das linhas é não usual: de baixo → para cima, assim sendo a origem da imagem é no canto inferior esquerdo da tela, enquanto o sistema de coordenadas do MS Windows tem sua origem do canto superior esquerdo. Esta característica técnica do Windows obriga a maioria dos códigos de carga de imagens não BMP a implementarem a carga "inversa" da imagem em memória pois a maioria dos formatos de imagem prevê a origem do sistema de coordenadas, acima e à esquerda da tela.

O valor "**biBitCount**", membro da estrutura BITMAPINFOHEADER, determina o número de bits utilizados para definir cada pixel da imagem e pode

assumir os valores apresentados na Tabela 1.1.

TABELA 1.1 - Decodificação de "***biBitCount***"

Valor	Significado
1	A imagem contida no arquivo é monocromática, assim sendo a " <b><i>color table</i></b> " contém apenas duas cores. Cada bit na " <b><i>bitmap bits</i></b> " representa um pixel. Se o bit for <b><i>null</i></b> sua cor de apresentação é definida pelo primeiro valor de " <b><i>color table</i></b> ", caso contrário, pelo segundo valor.
4	A imagem contida no arquivo possui um máximo de 16 cores, assim sendo a " <b><i>color table</i></b> " contém 16 cores. Cada byte na " <b><i>bitmap bits</i></b> " representa dois pixels. Assim sendo se o valor lido de um byte for 243 (11110011) significa que o primeiro pixel deve ser apresentado pela 16 <sup>a</sup> cor da " <b><i>color table</i></b> " e que o segundo pixel deve ser apresentado pela 3 <sup>a</sup> cor de " <b><i>color table</i></b> ".
8	A imagem contida no arquivo possui um máximo de 256 cores, assim sendo a " <b><i>color table</i></b> " contém 256 cores. Cada byte na " <b><i>bitmap bits</i></b> " representa um pixel. Assim sendo se o valor lido de um byte for 243 (11110011) significa que o pixel deve ser apresentado pela 243 <sup>a</sup> cor da " <b><i>color table</i></b> ".
24	A imagem contida no arquivo possui um máximo de 16777216 cores, assim sendo a " <b><i>color table</i></b> " contém 0 cores e seu valor é <b><i>null</i></b> . Cada pixel da imagem é representado por 3 bytes na " <b><i>bitmap bits</i></b> ", cada um referente a um componente de cor do pixel na ordem padrão RGB (vermelho, verde e azul).

O valor "***biClrUsed***" da estrutura BITMAPINFOHEADER contém o número de cores efetivamente utilizado pela imagem. Se "***biClrUsed***" for igual zero, então a imagem efetivamente utiliza todas as cores especificadas na "***color table***".

- **Compressão de dados:**

Os formatos BMP suportam compressão RLE de dados para codificações de imagens de 4 e 8 bits de cor.

Quando o valor "***biCompression***" da estrutura BITMAPINFOHEADER é setado em 1 (***BI\_RLE8***) então o arquivo utiliza compressão RLE de 8 bits. A compressão de 8 bits pode ocorrer de duas maneiras: "***encoded***" ou "***absolute mode***". Os dois modos de compressão podem ocorrer simultaneamente numa mesma imagem, em linhas ou coluna diferentes.

O modo "***encoded***" possui uma unidade de informação de 2 bytes. O primeiro byte especifica o número de repetições do valor do segundo byte. Quando o primeiro byte de um par é setado em zero, ele indica um fim de linha, fim de imagem ou um "***off set***". A interpretação deste par depende do valor do segundo byte que pode ser 0,1 ou 2 de acordo com a Tabela 1.2.

O modo "***absolute***" também possui uma unidade de informação de 2 bytes. Sua presença é sinalizada pelo primeiro byte do par contendo ***null***, e o segundo byte um valor entre 3 e 255. O segundo byte representa o número de bytes que se seguem sem compressão. Segue-se um exemplo de compressão RLE8 (Tabela 1.3).

TABELA 1.2 - Decodificação do segundo byte do modo "**encoded**"

Segundo byte	Significado
0	Fim de linha.
1	Fim de imagem.
2	Indica que os seguintes 2 BYTES contém o número de colunas e linhas, respectivamente, que o cursor de leitura deve se deslocar a partir da posição corrente.

TABELA 1.3 - Decodificação do segundo BYTE do modo "**absolute**"

Dado Comprimido	Dado Expandido
03 04	04 04 04
05 06	06 06 06 06 06
00 03 45 56 67	45 56 67
02 78	78 78
00 02 05 01	Cursor 5 à direita e 1 abaixo
02 78	78 78
00 00	Fim de linha
09 1E	1E 1E 1E 1E 1E 1E 1E 1E
00 01	Fim de imagem

Quando o valor "**biCompression**" da estrutura BITMAPINFOHEADER é setado em 2 (BI\_RLE4) então o arquivo utiliza compressão RLE de 4 bits. A compressão de 4 bits pode ocorrer de duas maneiras: "**encoded**" ou "**absolute mode**". Os dois modos de compressão podem ocorrer simultaneamente numa mesma imagem, em linhas ou colunas diferentes.

O modo "**encoded**" possui uma unidade de informação de 2 bytes. O primeiro byte especifica o número de repetições do valor do segundo byte. O segundo byte contém o índice de cor com 2 pixels consecutivos. O primeiro pixel deve utilizar o índice de cor dos bits de alta ordem do segundo byte e o segundo pixel deve usar o índice de cor dos bits de baixa ordem do segundo byte, sucessivamente até esgotar-se o número de repetições especificado pelo primeiro byte. Quando o primeiro byte de um par é setado em zero, ele indica um fim de linha, fim de imagem ou um "**off set**". A interpretação deste par depende do valor do segundo byte que pode ser 0, 1 ou 2 de acordo com a Tabela 1.2.

O modo "**absolute**" também possui uma unidade de informação de 2 bytes. Sua presença é sinalizada pelo primeiro byte do par contendo **null**, e o segundo byte um valor entre 3 e 255. O segundo byte representa o número de bytes que se seguem sem compressão, contendo cada um 2 pixels em seus bits de alta e baixa ordem. Segue-se um exemplo de compressão RLE4 (Tabela 1.4).

TABELA 1.4 - Decodificação do segundo BYTE do modo "**absolute**"

Dado Comprimido	Dado Expandido
03 04	0 4 0
05 06	0 6 0 6 0
00 06 45 56 67	4 5 5 6 6 7
04 78	7 8 7 8
00 02 05 01	Cursor 5 à direita e 1 abaixo
04 78	7 8 7 8

Dado Comprimido	Dado Expandido
00 00	Fim de linha
09 1E	1 E 1 E 1 E 1 E 1
00 01	Fim de imagem

- **Exemplo 1.1 – Arquivo BMP:**

O exemplo a seguir ilustra o conteúdo de cada variável das estruturas que compõem um arquivo exemplo BMP de 4 bits (16 cores):

```

BitmapFileHeader.Type = 19778
BitmapFileHeader.Size = 3118
BitmapFileHeader.Reserved = 0
BitmapFileHeader.Reserved2 = 0
BitmapFileHeader.OffsetBits = 118

BitmapInfoHeader.Size = 40
BitmapInfoHeader.Width = 80
BitmapInfoHeader.Height = 75
BitmapInfoHeader.Planes = 1
BitmapInfoHeader.BitCount = 4
BitmapInfoHeader.Compression = 0
BitmapInfoHeader.SizeImage = 3000
BitmapInfoHeader.XpelsPerMeter = 0
BitmapInfoHeader.YpelsPerMeter = 0
BitmapInfoHeader.ColorsUsed = 16
BitmapInfoHeader.ColorsImportant = 16

ColorTable
R   G   B  Unused
84  252 84  0
252 252 84  0
84  84   252 0
252 84   252 0
84  252 252 0
252 252 252 0
0   0   0   0
168 0   0   0
0   168 0   0
168 168 0   0
0   0   168 0
168 0   168 0
0   168 168 0
168 168 168 0
84  84   84  0
252 84   84  0
.
. Bitmap data
.
```

O algoritmo abaixo ilustra, simplificadamente, o esquema de descompressão RLE numa imagem de 8 bits e um plano de cor. Como o formato interno de armazenamento de memória do Windows aceita o arquivo de imagem comprimido, nenhum processamento é necessário, bastando que se copie o conteúdo do arquivo para a memória.

```

typedef struct tagBITMAPFILEHEADER {
    WORD Type;
    DWORD Size;
    WORD Reserved;
    WORD Reserved2;
    DWORD OffsetBits;
}

typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD bmiColors[1];
} BITMAPINFO;

typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    DWORD biWidth;
    DWORD biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    DWORD biXPelsPerMeter;
    DWORD biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;

/*****************************************/
// Lê uma imagem tipo BMP

int TImage::ReadBMP(char *FileName) {

    ifstream InputStream(FileName, ios::in | ios::binary);

    if(!InputStream) {
        return 0;
    }

    // Lê o header do arquivo imagem
    InputStream.read((unsigned char*)&BMPhdr, sizeof(BITMAPFILEHEADER));

    // Verifica se formato da imagem é aceitável (deve ser 'BM')
    if(BMPhdr.bfType != (('M' << 8) | 'B'))
        return NULL;

    // Determina o tamanho do DIB para leitura através do campo bfSize
    // menos o tamanho do BITMAPFILEHEADER

```

```

long bmpsize = BMPHdr.bfSize - sizeof(BITMAPFILEHEADER);

// Aloca espaço para o bitmap
PointerToDIB = GlobalAllocPtr(GHND, bmpsize);

// Se a memória falha retorna NULL
if(PointerToDIB == 0) return NULL;

// Aloca espaço para a imagem
unsigned char *rbuf = new unsigned char[MaxBlock];

if(rbuf == NULL) return NULL;

unsigned char huge
*DIBData = (unsigned char huge*) PointerToDIB;
unsigned int chunksize;
unsigned int i;

// Faz leitura por pedaços até acabar o arquivo
while(bmpsize > 0) {
    if(bmpsize > MaxBlock)
        chunksize = MaxBlock;
    else
        chunksize = (WORD)bmpsize;
    InputStream.read(rbuf, chunksize);

    // Copia para o DIB
    for(i = 0; i < chunksize; i++)
        DIBData[i] = rbuf[i];

    bmpsize -= chunksize;
    DIBData += chunksize;
}
delete rbuf;

// Computa o número de bytes por linha, arredondando para múltiplo de 4
LPBITMAPINFOHEADER pBMPIInfo = (LPBITMAPINFOHEADER)PointerToDIB;
BytePerLine = ((long)pBMPIInfo->biWidth * (long)pBMPIInfo->biBitCount +
               31L) / 32 * 4;
return TRUE;
}

```

#### 1.4.2 Formato FLIC

Em 1987 a AutoDesk Inc. lançou no mercado o seu primeiro software para criação de animações em duas dimensões, que viria se tornar um sucesso de mercado: o Animator 1.0. Este software introduziu o formato FLIC, desenvolvido por Jim Kent especialmente para a AutoDesk Inc.

Existem duas versões de arquivos FLIC, a .FLI originalmente desenvolvida para o Animator 1.0 e que suporta apenas resoluções de 320x200, e a versão .FLC, do Animator Pro, que suporta qualquer tamanho de tela. Embora não permitam a adição de som e apenas possam manipular imagens codificadas em 8 bits, a simplicidade, eficiência e principalmente a boa performance deste formato o tornaram um padrão de fato, amplamente suportado por dezenas de outros aplicativos.

- **Concepção:**

O formato FLIC, como em animações convencionais, abriga uma sequência de quadros que, quando apresentados em velocidade apropriada, criam a idéia de movimento. Sua idéia fundamental é baseada no fato de que na maioria das animações digitais o conteúdo de um quadro é sempre ligeiramente diferente de seu antecessor. Assim, o formato armazena apenas esta diferença entre os quadros, trazendo grande economia de meios de armazenamento e processamento, pois rejeita a informação redundante a respeito dos quadros.

Sua concepção levou em conta a possibilidade de melhoramentos, e assim criou-se uma estrutura baseada em blocos. Estes blocos possuem um cabeçalho que permite aos programas de versões anteriores ignorarem blocos desconhecidos, mantendo assim uma compatibilidade relativa entre sucessivas versões. Os blocos, por sua vez, dividem-se em pacotes e estes finalmente dividem-se em células. Assim, apresentar animações codificadas no formato FLIC consiste em continuamente decodificar blocos, pacotes e células, apresentando os resultados na tela.

- **Estrutura:**

Todo arquivo FLIC segue, de maneira geral, a seguinte estrutura de codificação:

1. Bloco de cabeçalho de animação;
2. Bloco de prefixo;
3. Bloco de primeiro quadro;
4. Bloco das diferenças para o segundo quadro;
  
- .....
- n. Bloco das diferenças para o primeiro quadro

Esta organização não é rígida e alguns blocos podem ser ignorados ou suprimidos, como é o caso do bloco de prefixo que abriga parâmetros exclusivos do Animator Pro. Dentro dos blocos os pacotes podem ocorrer em qualquer ordem, o que nos obrigará a desenvolver uma rotina de decodificação capaz de manipulá-los.

- **Bloco de cabeçalho:**

O bloco de cabeçalho é sempre o primeiro bloco de qualquer arquivo FLIC e abriga a descrição global da animação. Possui 128 bytes e contém os dados apresentados na Tabela 1.5.

TABELA 1.5 – Bloco de cabeçalho (128 bytes)

<b>Bytes</b>	<b>Nome</b>	<b>Descrição</b>
4	Size	Tamanho de todo o arquivo em bytes
2	Type	AF12 para .FLC AF11 para .FLI
2	Frames	Número total de frames
2	Width	Largura da animação
2	Height	Altura da animação

Bytes	Nome	Descrição
2	Depth	Bits por pixel (sempre 8)
2	Flags	Identificador (0x0003)
4	Speed	1/70 seg para .FLC 1/1000 seg para .FLI
2	Reserved	Setado para 0
4	Created	Data de criação no formato MS-DOS
4	Creator	Número de série do programa criador
4	Updated	Data de revisão no formato MS-DOS
4	Updater	Número de série do programa revisor
2	Aspectx	Relação de aspecto na horizontal
2	Aspecty	Relação de aspecto na vertical
38	Reserved	Setado para 0
4	Offset 1	Seek Off set para o primeiro frame
4	Offset 2	Seek Off set para o segundo frame
40	Reserved	Setados para 0

- **Bloco de prefixo:**

O bloco de prefixo é opcional para a grande maioria dos programas que suportam os formatos FLIC. Foi originalmente concebido para abrigar os parâmetros ambientais do Animator Pro. É seguro ignorar estes valores para a decodificação da animação. Um bloco de prefixo normalmente segue o bloco de cabeçalho e apresenta o cabeçalho apresentado na Tabela 1.6.

TABELA 1.6 – Conteúdo do cabeçalho de bloco fixo

Bytes	Nome	Descrição
4	Size	Tamanho total do bloco de prefixo
2	Type	Identificador de bloco de prefixo (0xF100)
2	Packets	Número de pacotes do bloco
8	Reserved	Setado para 0

- **Bloco de quadro:**

O bloco de quadro abriga a informação necessária para formar os quadros da animação. O bloco de primeiro quadro contém os pacotes necessários para a apresentação da imagem do primeiro quadro da sequência. Os blocos de quadro seguintes contém as informações sobre as diferenças entre o quadro anterior e o atual. O último bloco de quadro contém informações diferenciais com o primeiro quadro da sequência, permitindo assim que a animação seja executada indefinidamente. Um bloco de quadro sempre é iniciado com um cabeçalho que o identifica, como apresentado na Tabela 1.7.

TABELA 1.7 – Conteúdo do cabeçalho do bloco de quadro

Bytes	Nome	Descrição
4	Size	Tamanho total do bloco de quadro
2	Type	Identificador de bloco de quadro (0xF1FA)

Bytes	Nome	Descrição
2	Packets	Número de pacotes do bloco
8	Reserved	Setado para 0

- **Pacotes do bloco de quadro:**

Um bloco de quadro pode apresentar nenhum ou vários pacotes, em qualquer ordem. Quando um bloco não apresenta pacotes de dados isto significa que ele contém apenas a informação de que no intervalo de tempo destinado ao quadro não ocorreram mudanças na tela. Os pacotes, como os blocos, também possuem cabeçalho, como apresentado na Tabela 1.8.

TABELA 1.8 – Conteúdo do cabeçalho de pacotes do bloco de quadro

Bytes	Nome	Descrição
4	Size	Tamanho total do pacote
2	Type	Identificador do tipo de pacote

Conforme o tipo de pacote seguem-se complementos ao cabeçalho. Os pacotes de bloco de quadro podem ser de oito tipos, identificados pela Tabela 1.9.

TABELA 1.9 – Identificadores dos tipos de pacotes do bloco de quadro

Valor	Nome	Descrição
4	COLOR_256	Informação de Palette em 8 bits (.FLC)
7	DELTA_FLC	Delta imagem com compressão tipo .FLC
11	COLOR_64	Informação de Palette em 6 bits (.FLI)
12	DELTA_FLI	Delta imagem com compressão tipo .FLI
13	BLACK	Imagen é da cor de índice 0
15	BYTE_RUN	Imagen com compressão Run-Length
16	LITERAL	Imagen sem compressão
18	PSTAMP	Ícone do primeiro quadro

- Pacotes COLOR\_256 e COLOR\_64

Os pacotes COLOR\_256 e COLOR\_64 são muito semelhantes na sua estrutura. Eles contém informações sobre os valores da palette para o quadro corrente. Normalmente estes pacotes só estão presentes no primeiro quadro, mas não existe nenhum impedimento para que estejam presentes nos demais quadros. O pacote COLOR\_64 apresenta codificação de cores em 6 bits. Normalmente é encontrado nos arquivos .FLI. O pacote COLOR\_256 codifica as cores da palette em 8 bits.

Os pacotes COLOR apresentam um sub-cabeçalho de dois bytes que contém o número de células de cores do pacote. Cada célula inicia-se com um byte de offset do índice da palette e um byte para o número de cores que se seguem. As cores são codificadas em triplas RGB de três bytes cada. A seguinte rotina exemplifica a decodificação de um pacote COLOR\_256.

```

void Decode_Color_256() {
    WORD Cels,Colors,Index=0;
    BYTE OffSet,AuxColors,Cor[3];
    WORD i,t;

    _lread(FlicFile,&Cels,sizeof(Cels));

    for(i=0;i<Cels;i++) {
        _lread(FlicFile,&OffSet,sizeof(OffSet));
        Index+=OffSet;
        _lread(FlicFile,&AuxColors,1);

        if(AuxColors==0) Colors=256;
        else Colors=AuxColors;

        for(t=0;t<Colors;t++) {
            _lread(FlicFile,Cor,3);
            SetPalIndex(Index++, RGB(Cor[0],Cor[1],Cor[2]));
        }
    }
}

```

- **Pacote BLACK:**

O pacote do tipo BLACK, que pode surgir num quadro, nos informa que toda a tela deve ser preenchida com pixels da cor de índice 0 da palette. Normalmente esta cor é o preto, mas isto não é uma regra. Ao cabeçalho não segue nenhuma outra informação.

- **Pacote BYTE\_RUN:**

O pacote BYTE\_RUN abriga a informação necessária para compor uma tela. Ele é formado pela codificação em Run-Length da imagem completa do quadro. Este pacote é composto por células que representam as linhas. O número de células é igual ao número de linhas da imagem. Cada célula inicia-se com um byte cujo conteúdo deve ser desprezado. Seguem-se sub-células que deverão ser decodificadas para compor todos os pixels da linha. O primeiro byte destas sub-células indica o seu tipo. Caso o tipo da sub-célula seja menor que zero então o valor absoluto do tipo contém o número de bytes subsequentes que deverão ser copiados para a tela. Caso o tipo seja positivo, o byte seguinte deverá ser repetido este mesmo número de vezes. O código a seguir exemplifica esta operação.

```

void Decode_Byt_Run() {
    char Pix,Run;
    int i,x,y;

    for(y=0,y<Flic.Height;y++) {
        x=0;
        _lread(FlicFile,&Run,1);

        do {
            _lread(FlicFile,&Run,1);

            if(Run>0) {
                _lread(FlicFile,&Pix,1);

```

```

        for(i=0;i<Run;i++) Buf[y][x++]=Pix;
    }
    else {
        _lread(FlicFile,Buf[y]+x,-Run);
        x-=Run;
    }
} while(x<Flic.Width);

SetScreenBits(Buf[y]);
}
}
}

```

- **Pacote LITERAL:**

Um pacote literal contém, de forma seqüencial e não comprimida, todos os bytes que formam a imagem do quadro. Este pacote é pouco usual e somente é adotado quando a imagem comprimida por BYTE\_RUN ou DELTA\_FLI\* é maior que a imagem original. Seguem-se ao cabeçalho do pacote, com origem no topo à esquerda, (altura x largura) bytes que compõem a imagem.

- **Pacote DELTA\_FLI:**

O esquema ágil de compressão, desenvolvido inicialmente no formato .FLI, foi o grande responsável pelo imediato sucesso deste formato.

O pacote DELTA\_FLI é orientado a byte, ou seja armazena e codifica as informações em sub-células de um pixel (byte). Seguem-se ao cabeçalho quatro bytes de sub-cabeçalho.

O primeiro inteiro (dois bytes) contém o número da linha em que se inicia o pacote (de cima para baixo, iniciando em 0) e o segundo inteiro, o número de linhas (células) do pacote.

Cada célula (linha) inicia-se com um byte que indica o seu número de sub-células. Cada sub-célula inicia-se com dois bytes, o primeiro abriga um contador de colunas que posiciona a sub-célula de atualização. O segundo byte abriga o tipo da sub-célula.

Caso o tipo da sub-célula seja maior que zero então o valor do byte de tipo contém o número de bytes subsequentes que deverão ser copiados para a tela. Caso o tipo seja negativo, o valor absoluto do tipo indica o número de vezes que o byte seguinte deverá ser repetido. O código a seguir exemplifica esta operação.

```

void Decode_Delta_FLI() {
    int x,y,s,t,u;
    BYTE Pix,ColSkip,Cels,SubCels,TypeSize;

    _lread(FlicFile,&y,2);
    _lread(FlicFile,&Cels,2);

    for(s=0;s<Cels;s++) {
        _lread(FlicFile,&SubCels,1);
        x=0;

        for(t=0;t<SubCels;t++) {
            _lread(FlicFile,&ColSkip,1);
            x+=ColSkip;
            _lread(FlicFile,&TypeSize,1);

```

```

        if(TypeSize<=127) {
            _lread(FlicFile,Buf[y]+x,TypeSize);
            x+=TypeSize;
        }
        else {
            _lread(FlicFile,&Pix,1);

            for(u=0;u<256-(int)TypeSize;u++) Buf[y][x++]=Pix;
        }
    }
    SetScreenBits(Buf[y++]);
}
}

```

- **Pacote DELTA\_FLC:**

O pacote DELTA\_FLC surgiu com o aprimoramento das controladoras de vídeo que passaram a dispor de resoluções maiores para a codificação de 8 bits com palette. Embora mantendo a mesma filosofia do pacotes DELTA\_FLI, os pacotes DELTA\_FLC são mais complexos. São orientados a inteiros (dois bytes), isto é, os pixels da tela são armazenados em pares de bytes.

De maneira similar ao DELTA\_FLI o pacote traz um sub-cabeçalho de 2 bytes que contém o número de linhas do pacote. É importante notar que o número de linhas normalmente é diferente do número de células do pacote. Não existe informação a respeito do número de células do pacote que devem ser controladas dentro da lógica da codificação. Cada célula pode conter diversas sub-células de pixel, informações de controle sobre as linhas ou sobre o último pixel da linha quando a largura da imagem for ímpar. Isto é possível graças a uma codificação especial adotada para o tipo da sub-célula.

Cada sub-célula inicia-se com dois bytes que indicam, nos bits 14 e 15, o seu tipo, conforme a Tabela 1.10.

TABELA 1.10 – Identificadores do tipos de sub-células de pacote DELTA\_FLC

<b>Bit 15</b>	<b>Bit 14</b>	<b>Significado</b>
0	0	Sub-célula contém informação de linha
1	1	Sub-célula contém último pixel da linha
0	1	Sub-célula indica salto de linha

Se os bits 14 e 15 de uma sub-célula são 0, então segue-se a esta sub-célula uma sequência de bytes com informação Run-Length da linha. O byte seguinte abriga um contador de posicionamento de coluna e o segundo o tipo da codificação Run-Length. A codificação é similar à do DELTA\_FLI onde um tipo positivo indica um bloco de cópia e um tipo negativo indica um bloco de repetição (Run Length).

Se os bits 14 e 15 estiverem setados então a sub-célula abriga nos seus 8 bits de baixa ordem um byte que deve ser copiado para a posição do último pixel da linha.

No último caso, bit 14 setado e 15 zerado, o valor absoluto do inteiro de tipo indica um salto de linhas que deve ser aplicado na composição do quadro. O código que se segue ilustra um exemplo de descompactação de pacote

## DELT A\_FLC.

```

void Decode_Delta_FLC() {
    BYTE PixPair[2];
    WORD SkipCol,RLType;
    int i,t,u,x,y=0,SubCelType,TotLin,SubCel;

    _lread(FlicFile,&TotLin,2);

    for(i=0;i<TotLin;i++) {
        x=0;
        _lread(FlicFile,&SubCel,2);

        if(SubCel>0) { //bit 14 = 0 e bit 15 = 0

            for(t=0;t<SubCel;t++) {
                _lread(FlicFile,&SubCelType,2);
                SkipCol = LOBYTE(SubCelType);
                x+=SkipCol;
                RLType = HIBYTE(SubCelType);

                if(RLType<=127) {
                    _lread(FlicFile,Buf[y]+x,RLType*2);
                    x+=RLType*2;
                }
                else {
                    _lread(FlicFile,PixPair,2);

                    for(u=0;u<256-RLType;u++) {
                        memcpy(Buf[y]+x,PixPair,2);
                        x+=2;
                    }
                }
                SetScreenBits(Buf[y++]);
                y++;
            }
            else if((WORD)SubCelType & 0x4000) { //bit 14 = 1 e bit 15 = 0
                y-=SubCel;
                i--;
            }
            else if((WORD)SubCelType & 0xC000) { //bit 14 = 1 e bit 15 = 1
                Buf[y][Flic.Width-1] = HIBYTE(SubCelType);
                i--;
            }
        }
    }
}

```

## UNIDADE 2 – PROCESSAMENTO DE IMAGENS

### 2.1 FUNDAMENTOS DE IMAGENS DIGITAIS

A imagem digital é a materialização de grande parte dos processos da computação gráfica. Neste sentido, ela serve como elo de ligação entre o usuário e esses procedimentos, revelando os seus resultados. Podemos até mesmo afirmar que a imagem está presente em todas as áreas da computação gráfica, seja como produto final, no caso da visualização, ou como parte essencial do processo de interação, no caso da modelagem. Por esse motivo, é de fundamental importância a compreensão do significado da imagem nesses contextos. Isto requer uma formulação rigorosa das diversas noções associadas à imagem digital para instrumentalizar a análise das estruturas de dados utilizadas na sua representação e o estudo dos algoritmos empregados na sua criação e manipulação.

Nesta seção vamos desenvolver uma conceituação da imagem digital, apresentar modelos abstratos de uma imagem, e diversas formas de representação e codificação de imagens no computador.

#### 2.1.1 Paradigmas de Abstração de Imagens

Para representar e manipular imagens no computador devemos definir modelos matemáticos adequados a esses objetivos.

A imagem é o resultado de estímulos luminosos produzidos por um suporte bidimensional. Essa é a percepção da imagem no universo físico no qual habitamos, seja ela o resultado de um processo intermediado, como por exemplo uma fotografia, ou ultimamente através da projeção do mundo tridimensional na retina do olho humano.

Devemos estabelecer um universo matemático no qual seja possível definir diversos modelos abstratos de uma imagem. Em seguida, criamos um universo de representação onde procuramos esquemas que permitam uma representação discreta desses modelos, com o objetivo de obter uma codificação da imagem no computador.

A imagem é um sinal bidimensional e, consequentemente, podemos usar os conceitos de Teoria de Sinais<sup>1</sup>. No que se segue, vamos particularizar aqueles conceitos para imagens. Temos, portanto, três níveis de abstração que correspondem a modelos contínuos, representações discretas, e codificação simbólica de imagens, conforme ilustramos na Figura 2.1.

É importante notar que esses níveis serão realizados concretamente de maneiras distintas em um sistema de processamento de imagens. Por esse motivo, para obter um esquema unificado para o processamento de imagens, temos que empregar as diversas transformações que permitem passar de um nível para outro e também manipular descrições em um mesmo nível.

### 2.2 MODELO ESPACIAL DE IMAGEM

Embora existam vários modelos matemáticos adequados para se descrever imagens, daremos ênfase ao chamado modelo espacial, que é o mais

---

<sup>1</sup> GOMES, J.; VELHO, L. **Computação Gráfica**: imagem. Rio de Janeiro : IMPA/SBM, 1994.

indicado para aplicações da computação gráfica.

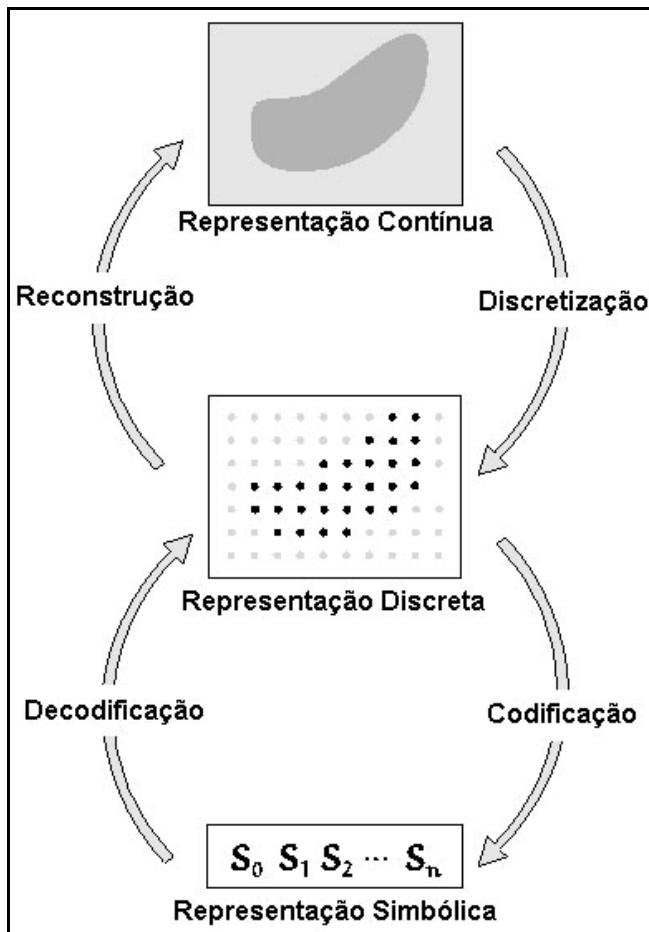


Figura 2.1 – Níveis de abstração na representação de uma imagem.

### 2.2.1 Imagem Contínua

Quando observamos uma fotografia, ou uma cena real, recebemos de cada ponto do espaço um impulso luminoso que associa uma informação de cor a esse ponto. Dessa forma, um modelo matemático natural para descrever uma imagem é o de uma função definida em uma superfície bidimensional e tomando valores em um espaço de cor.

Uma **imagem contínua** é uma aplicação  $i: U \rightarrow C$ , onde  $U \subset \mathbb{R}^3$  é uma superfície e  $C$  é um espaço vetorial. Na maioria das aplicações  $U$  é um subconjunto plano, e  $C$  é um espaço de cor. No entanto, é conveniente para o desenvolvimento da teoria, que  $C$  seja um espaço vetorial qualquer que, em geral, contém o espaço de cor como subespaço. A função  $i$  na definição é chamada de **função imagem**. O conjunto  $U$  é chamado de **suporte de imagem**, e o conjunto dos valores de  $i$ , que é um subconjunto de  $C$ , é chamado de conjunto dos **valores da imagem** ou **gamute de cores** da imagem.

Quando  $C$  é um espaço de cor de dimensão 1, dizemos que a imagem é **monocromática**. Nesse caso, a imagem pode ser vista geometricamente como o gráfico  $G(i)$  da função imagem  $i$ ,

$$G(i) = \{(x, y, z); (x, y) \in U \text{ e } z = i(x, y)\}$$

considerando os valores de intensidade como a altura  $z = i(x, y)$  em cada ponto  $(x, y)$  do domínio. A Figura 2.2 mostra uma imagem em meio-tom, juntamente com um esboço do gráfico da função imagem  $i(x, y)$  correspondente. Essa interpretação geométrica permite uma visão mais intuitiva de certos aspectos da imagem. No gráfico da Figura 2.2, por exemplo, é fácil de identificar as regiões de descontinuidade da função, que correspondem a variações bruscas da intensidade dos pontos da imagem. Com essa abordagem, que permite manipular uma imagem como um modelo geométrico e vice-versa, torna-se bastante clara a conexão entre o processamento de imagem, a modelagem geométrica e, numa segunda instância, a visão computacional.

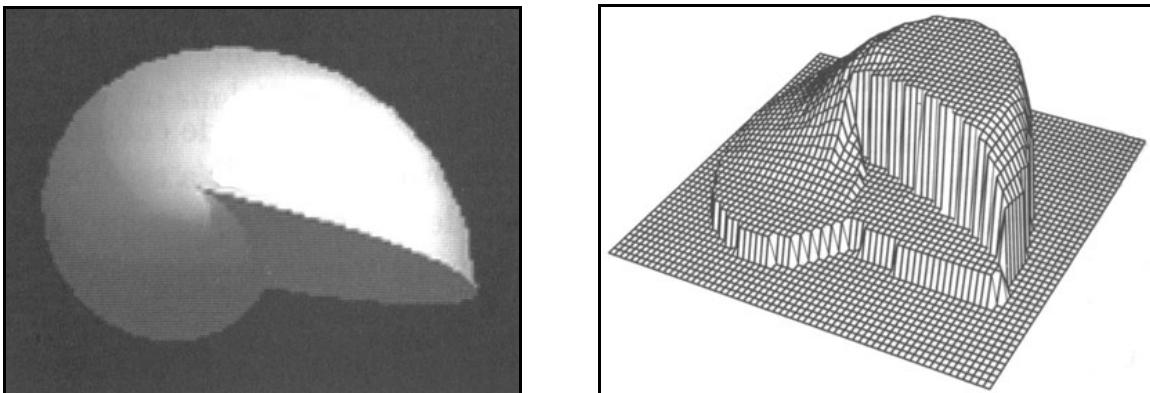


Figura 2.2 – Imagem meio-tom e gráfico de sua função

Em concordância com a literatura corrente sobre processamento de imagens, o conceito de imagem introduzido acima deveria, mais apropriadamente, caracterizar uma **imagem bidimensional**. Nas aplicações de computação gráfica na área de visualização científica, é muito comum se falar em **imagens volumétricas**. Podemos estender o conceito de imagem, dado acima, para incluir imagens volumétricas, simplesmente deixando de exigir a bidimensionalidade do conjunto suporte. Não fizemos isso por duas razões: o conceito de imagem aqui introduzido, se refere a imagens que podem ser realizadas em dispositivos gráficos de exibição. No entanto várias das técnicas descritas neste capítulo se generalizam para imagens volumétricas. Do ponto de vista da Computação Gráfica, os problemas associados a imagens volumétricas estão estreitamente relacionados com problemas da área de modelagem geométrica.

### 2.2.2 Representação de uma Imagem

A representação mais comum de uma imagem espacial nas aplicações da computação gráfica consiste em tomar um subconjunto discreto  $\mathbf{U}' \subset \mathbf{U}$  do domínio da imagem, um espaço de cor  $\mathbf{C}$  associado a um dispositivo gráfico, e a imagem é representada pela amostragem da função imagem  $i$  no conjunto  $\mathbf{U}'$ . Nesse caso a imagem  $i(x, y)$ , será espacialmente contínua ou discreta se as coordenadas  $(x, y)$  da cada ponto variam no conjunto  $\mathbf{U}$  ou  $\mathbf{U}'$  respectivamente. Cada ponto  $(x_i, y_i)$  do subconjunto discreto  $\mathbf{U}'$  é chamado de elemento da imagem ou pixel (do inglês “picture element”).

Salientamos mais uma vez que o conceito de imagem contínua e discreta se refere à discretização do domínio da função imagem, e não à continuidade topológica da função imagem. Para codificar a imagem no computador devemos

também trabalhar com modelos de imagem onde a função imagem  $i$  toma valores em um subconjunto discreto do espaço de cor  $\mathbf{C}$ . Esse processo de discretização do espaço de cor de uma imagem é chamado de **quantização**. Apesar de que a representação por ponto flutuante é uma discretização do conjunto dos números reais, em processamento de imagens podemos considerar como um “continuum” o espaço de cor cujas coordenadas são especificadas em ponto flutuante. Essa consideração é válida porque, apenas quando utilizamos um número muito reduzido de bits na representação das cores de uma imagem, o erro introduzido se traduz em problemas perceptíveis ou computacionais.

- Representação Matricial:

O caso mais utilizado de discretização espacial de uma imagem consiste em tomar o domínio como sendo um retângulo

$$U = [a, b] \times [c, d] = \{(x, y) \in \mathbb{R}^2; a \leq x \leq b \text{ e } c \leq y \leq d\}$$

e discretizar esse retângulo usando os pontos de um reticulado bidimensional  $\Delta = (\Delta x, \Delta y)$ ,

$$\Delta = \{(x_j, y_k) \in U; x_j = j \cdot \Delta x, y_k = k \cdot \Delta y \quad j, k \in \mathbb{Z}, \Delta x, \Delta y \in \mathbb{R}\}$$

conforme mostramos na Figura 2.3. Cada pixel  $(x_j, y_k)$  da imagem pode portanto ser representado por coordenadas inteiras  $(j, k)$ . Portanto, a imagem pode ser representada de forma conveniente no **formato matricial**. Nessa representação ela está associada a uma matriz  $A$  de ordem  $m \times n$ ,  $A = (a_{jk}) = (i(x_j, y_k))$ .

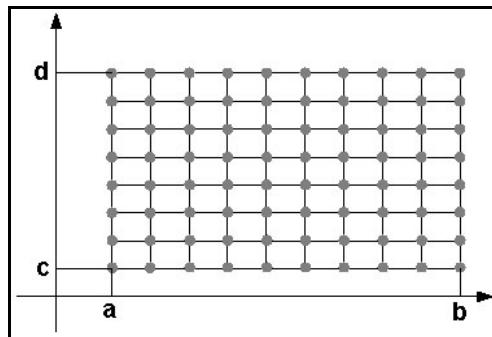


Figura 2.3 – Reticulado uniforme da representação matricial da imagem

Cada elemento  $a_{jk}$ ,  $j=1, \dots, m$  e  $k = 1, \dots, n$  da matriz representa um valor da função imagem  $i$  no ponto de coordenadas  $(x_j, y_k)$  do reticulado, sendo pois um vetor do espaço de cor, representando a cor do pixel de coordenada  $(j, k)$ . Se a imagem for monocromática,  $(a_{jk})$  é uma matriz real, onde cada elemento é um número real que representa o valor de luminância do pixel.

O número de colunas  $m$  da matriz  $A$  de pixels é chamado de **resolução horizontal** da imagem, e o número de linhas  $n$  é chamado de **resolução vertical**. Denominamos **resolução espacial** da representação matricial o produto  $m \times n$  da resolução vertical pela resolução horizontal. A resolução espacial estabelece a frequência de amostragem final da imagem. Dessa forma, quanto maior a resolução mais detalhe, isto é, altas frequências, da imagem podem ser captados na

representação matricial. A resolução espacial dada em termos absolutos não fornece muita informação sobre a resolução real da imagem quando realizada em um dispositivo físico. Isso ocorre porque ficamos na dependência do tamanho físico do pixel do dispositivo. Uma medida mais confiável de resolução é dada pela **densidade de resolução** da imagem que fornece o número de pixels por unidade linear de medida. Em geral se utiliza o número de pixels por polegada, **dpi** ("dots per inch").

Na Figura 2.4 mostramos uma imagem em quatro resoluções espaciais diferentes. Mostramos cada imagem com as mesmas dimensões, ampliando o tamanho do pixel, de forma a obter uma melhor ilustração da baixa resolução da imagem.



Figura 2.4 – Diferentes resoluções espaciais de uma imagem

Chamamos de **resolução de cor** ao número de bits utilizado para armazenar o vetor de cor  $a_{jk}$  de cada pixel da imagem. A resolução de cor será estudada com mais detalhe na seção sobre discretização de cor.

A representação de uma imagem por uma matriz  $m \times n$  ou um vetor de dimensão  $mn$ , possibilita o uso de técnicas de álgebra linear no processamento de imagens.

### 2.2.3 Imagem Digital

Embora seja necessário trabalhar no computador com representações dos modelos discretos de imagem, conceitualmente é importante considerar que podemos idealizar uma imagem em qualquer das possíveis combinações: **contínua-contínua**, **contínua-quantizada**, **discreta-contínua** e **discreta-quantizada**. Cada caso acima corresponde à natureza do domínio e do espaço de cor da imagem.

Na prática, de um lado a imagem contínua-contínua serve como conceito utilizado no desenvolvimento dos métodos matemáticos para o processamento de imagens; do outro lado, a imagem discreta-quantizada constitui a representação utilizada por vários dispositivos gráficos. A imagem discreta-contínua é um formato

conveniente para a maior parte das operações com imagens, pois a função imagem assume valores de ponto flutuante que, embora representados por um número finito de bits, aproximam valores reais. Uma imagem do tipo discreta-quantizada é chamada de ***imagem digital***.

- Elementos da Imagem Digital:

Os elementos da imagem consistem, essencialmente, das coordenadas dos pixels e da informação de cor de cada pixel. Esses dois elementos estão diretamente relacionados com a resolução espacial e a resolução de cor da imagem. O **número de componentes** do pixel é a dimensão do espaço de cor utilizado. Dessa forma, cada pixel em uma imagem monocromática tem uma única componente. O **gamute** de uma imagem digital é o conjunto das cores do espaço de cor quantizado da imagem. Uma imagem monocromática cujo gamute possui apenas duas cores é chamada de ***imagem de dois níveis***, ***imagem "bitmap"***, ou ***imagem binária***. Uma imagem monocromática cujo gamute possui mais de dois níveis é chamada de ***imagem com meio-tom*** ou ***imagem com escala de cinza*** (do inglês “halftone” ou “grayscale”).

Se o espaço de cor de uma imagem tem dimensão  $k$ , para grande parte dos processos podemos considerar cada componente de cor em separado. Nesse caso a imagem se reduz a  $k$  imagens com escala de cinza. Cada uma dessas imagens é chamada de **componente** da imagem original. É muito comum efetuar operações com imagens trabalhando separadamente com cada componente. O processamento por componentes simplifica bastante determinadas operações. No entanto, esse tipo de processamento não explora a correlação na informação de cor que existe entre as diversas componentes de cada pixel da imagem.

Além da informação de cor, as outras componentes de espaço vetorial  $C$ , do contra domínio de uma imagem, estão freqüentemente associados a informações auxiliares, tais como: **opacidade do pixel**, **profundidade do ponto da cena correspondente ao pixel**, etc. Essas informações são produzidas e utilizada por algoritmos gráficos para diversas finalidades, que são estudadas na área de síntese de imagens.

#### 2.2.4 Topologia Digital e Representação Matricial

A estrutura do formato matricial determina implicitamente uma relação de conectividade entre os elementos da imagem. Nesse contexto podemos definir uma topologia para o domínio da imagem de acordo com dois tipos de vizinhança discreta. Essas vizinhanças são denominadas de 4-conectada e 8-conectada. Dado um elemento  $a_{ij}$ , a **vizinhança 4-conectada** é o conjunto dos elementos  $a_{i-1, j}, a_{i+1, j}, a_{i, j-1}, a_{i, j+1}$ , conforme ilustrado na Figura 2.5(a). A **vizinhança 8-conectada** é o conjunto do elementos da vizinhança 4-conectada mais os elementos  $a_{i-1, j-1}, a_{i+1, j-1}, a_{i-1, j+1}, a_{i+1, j+1}$ , conforme mostrado na Figura 2.5(b).



Figura 2.5 – Tipos de vizinhança na representação matricial

Na Figura 2.6 mostramos uma curva que vai do pixel A até o pixel B. Essa curva, numa linguagem mais próxima da área de processamento de imagens, é 8-conectada, porém não é 4-conectada.

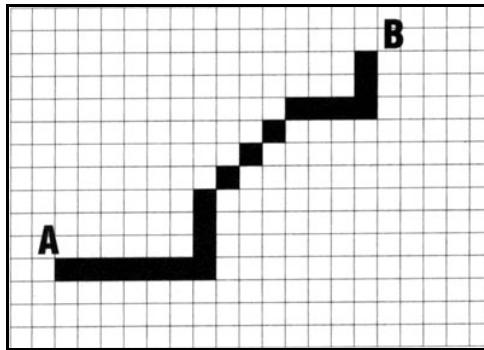


Figura 2.6 – Curva 8-conectada mas não 4-conectada

## 2.2.5 Geometria do Pixel

Considere a imagem digital e sua representação matricial mostrada na Figura 2.3. Considerada em um domínio discreto, o reticulado define na realidade uma malha poligonal uniforme de retângulos do plano. Nesse caso consideramos o **pixel geométrico** como sendo cada polígono da malha.

Associada a essa imagem contínua, definida por uma malha poligonal, temos duas imagens discretas. Uma delas é a imagem definida no reticulado de pixels original, a outra, é definida no reticulado de pixels dual. Nesse caso, cada pixel está situado no centro de cada retângulo da malha poligonal. esse fato é ilustrado na Figura 2.7(a), (b) e (c).

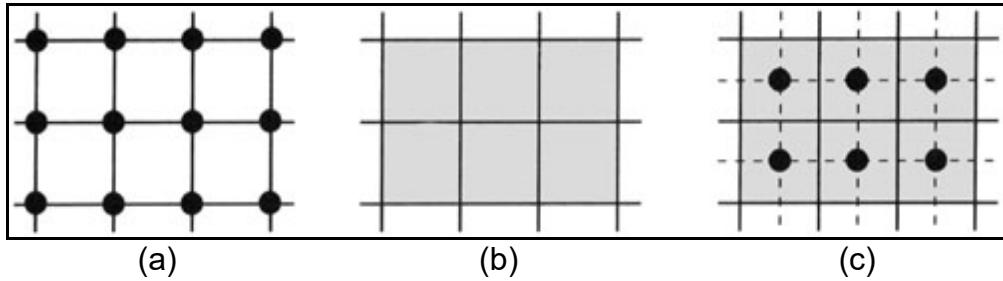


Figura 2.7 – Reticulado de pixels, malha poligonal e reticulado dual

Além da representação matricial, podemos utilizar outras representações para uma imagem discretizada espacialmente. Nesse caso, a geometria do pixel pode não ser retangular e é possível obter uma homogeneidade na definição da vizinhança do pixel. Isso acontece, por exemplo, ao utilizarmos uma discretização hexagonal do domínio, conforme indicado na Figura 2.8(a). A Figura 2.8(b), mostra que nessa discretização cada pixel apresenta apenas um tipo de conectividade com os pixels de sua vizinhança. Essa representação é muito utilizada quando estamos interessados em utilizar métodos topológicos para o processamento de imagens no domínio discreto.

Um problema que surge ao trabalhar com representações onde a topologia do pixel não é retangular é que os diversos dispositivos gráficos que manipulam imagens digitais se utilizam da representação matricial. Esse é um caso típico em que devemos reconstruir a imagem para reamostrá-la na representação

matricial do dispositivo de saída.

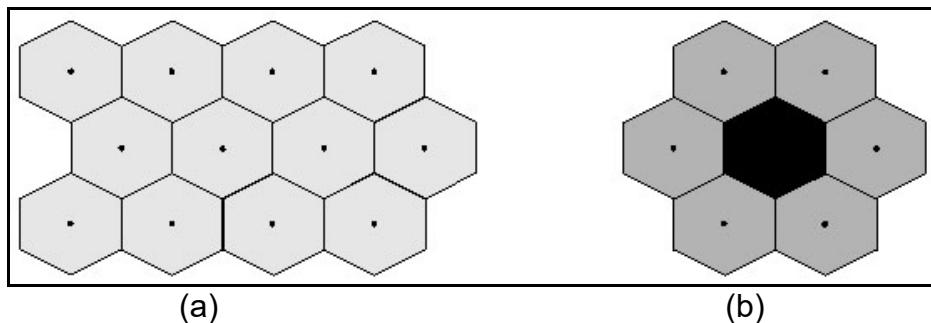


Figura 2.8 – (a) Representação hexagonal (b) Vizinhança na representação hexagonal

No caso da discretização hexagonal, mostrada na Figura 2.8, a malha de poligonalização dual, forma uma poligonalização do domínio da imagem por hexágonos.

Quando trabalhamos no domínio contínuo com o conceito de pixel geométrico, a informação de cor pode sofrer variação na região poligonal que define o pixel. Desse modo, para calcular a imagem digital no reticulado dual devemos levar em consideração essa variação de forma a minimizar os problemas de *aliasing*.

### 2.3 COMPONENTES DE UM SISTEMA DE PROCESSAMENTO DE IMAGENS

Os elementos de um sistema de processamento de imagens de uso genérico são mostrados na Figura 2.9. Este diagrama permite representar desde sistemas de baixo custo até sofisticadas estações de trabalho, utilizadas em aplicações que envolvem intenso uso de imagens. Ele abrange as principais operações que se podem efetuar sobre uma imagem, a saber: aquisição, armazenamento, processamento e exibição. Além disso, uma imagem pode ser transmitida à distância utilizando meios de comunicação disponíveis. Todas estas operações são descritas a seguir.

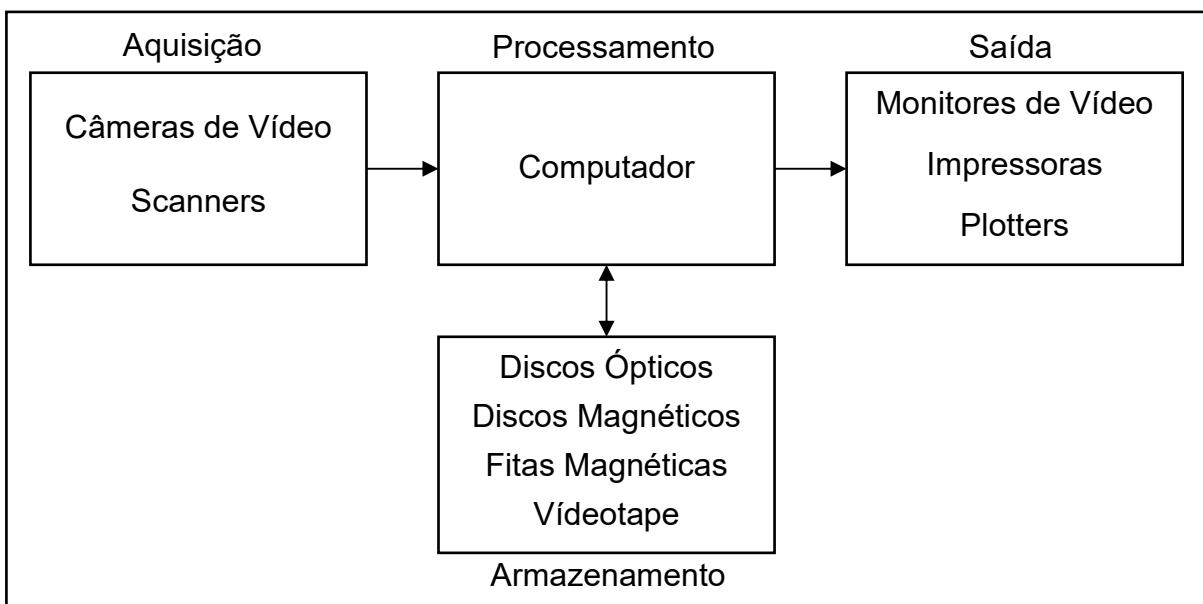


Figura 2.9 – Elementos de um sistema de processamento de imagens

### 2.3.1 Aquisição

A etapa de aquisição tem como função converter uma imagem em uma representação numérica adequada para o processamento digital subsequente. Este bloco compreende dois elementos principais. O primeiro é um dispositivo físico sensível a uma faixa de energia no espectro eletromagnético (como raio X, ultravioleta, espectro visível ou raios infravermelhos), que produz na saída um sinal elétrico proporcional ao nível de energia detectado. O segundo – o digitalizador propriamente dito – converte o sinal elétrico analógico em informação digital, isto é, que pode ser representada através de bits 0s e 1s. Um módulo de aquisição de imagens é normalmente conhecido pelo nome de frame grabber.

### 2.3.2 Armazenamento

O armazenamento de imagens digitais é um dos maiores desafios no projeto de sistemas de processamento de imagens, em razão da grande quantidade de bytes necessários para tanto. Este armazenamento pode ser dividido em três categorias: (1) armazenamento de curta duração de uma imagem, enquanto ela é utilizada nas várias etapas do processamento, (2) armazenamento de massa para operações de recuperação de imagens relativamente rápidas, e (3) arquivamento de imagens, para recuperação futura quando isto se fizer necessário. O espaço de armazenamento requerido é normalmente especificado em bytes e seus múltiplos.

Para o armazenamento de **curta duração**, a alternativa mais simples é utilizar parte da memória RAM do computador principal. Outra opção consiste no uso de placas especializadas, chamadas **frame buffers**, que armazenam uma ou mais imagens completas e podem ser acessadas em alta velocidade, tipicamente 30 imagens por segundo. O uso de frame buffers permite também que operações de *zoom*, *scroll* (rolagem na vertical) e *pan* (rolagem na horizontal) sejam executadas de forma praticamente instantânea. Placas frame buffers, disponíveis no mercado atualmente, apresentam capacidade de armazenamento na faixa de dezenas de MB de memória.

A segunda categoria de armazenamento normalmente requer o uso de discos magnéticos de no mínimo algumas dezenas de GB e recentemente passou a utilizar também discos magneto-ópticos, por vezes agrupados em *jukeboxes* contendo de 30 a 100 discos. Nesta categoria o fator "tempo de acesso" é tão ou mais importante que a capacidade do meio de armazenamento. Através de cálculos simples ( $nº$  de pixels na horizontal x  $nº$  de pixels na vertical x  $nº$  de bits necessários para a escala de cinza / 8), pode-se estimar a quantidade de bytes necessários para armazenar uma imagem monocromática em disco. Este cálculo entretanto considera a imagem representada como uma matriz, cujos elementos são os valores de tons de cinza dos respectivos pixels. Na prática, informações adicionais são necessárias; por exemplo o tamanho da imagem, o número de cores ou tons de cinza, etc. Estas informações costumam ser colocadas em um cabeçalho (header) no início do arquivo. Infelizmente, não existe um único cabeçalho ou formato de armazenamento de imagens padronizado. Alguns dos formatos mais conhecidos são o BMP, PCX, TIFF, JPEG e GIF.

Finalmente, o arquivamento de imagens é caracterizado por quantidades gigantescas de bytes contendo imagens cuja recuperação é esporádica. Nesta categoria, as fitas magnéticas estão dando lugar aos discos ópticos, com capacidade que pode chegar a mais de 10 GB por disco, e que também podem ser agrupados

em *jukeboxes*, com capacidade total de armazenamento superior a 1 TB.

### 2.3.3 Processamento

O processamento de imagens digitais envolve procedimentos normalmente expressos sob forma algorítmica. Em função disto, com exceção das etapas de aquisição e exibição, a maioria das funções de processamento de imagens pode ser implementado via software. O uso de hardware especializado para processamento de imagens somente será necessário em situações nas quais certas limitações do computador principal forem intoleráveis, por exemplo a velocidade de transferência dos dados através do barramento.

A tendência atual do mercado de hardware, para processamento de imagens, é a comercialização de placas genéricas compatíveis com os padrões de barramento consagrados pelas arquiteturas mais populares de microcomputadores e estações de trabalho. O software de controle dessas placas é que determinará sua aplicação específica a cada situação. As vantagens mais imediatas são: redução de custo, modularidade, reutilização de componentes de software em outra aplicação rodando sobre o mesmo hardware e independência de fornecedor. Convém notar, entretanto, que sistemas dedicados continuam sendo produzidos e comercializados para atender tarefas específicas, tais como processamento de imagens transmitidas por satélites.

### 2.3.4 Transmissão

Imagens digitalizadas podem ser transmitidas à distância utilizando redes de computadores e protocolos de comunicação já existentes. O grande desafio da transmissão de imagens à distância é a grande quantidade de bytes que se necessita transferir de uma localidade a outra, muitas vezes através de canais de comunicação de baixa velocidade e banda passante estreita. Este problema é ainda mais sério quando se deseja transmitir sequências de vídeo (imagens em movimento com áudio associado) em tempo real, onde outros fatores, como por exemplo sincronização, devem ser considerados. Nestes casos, o uso de técnicas de compressão e descompressão de imagens é obrigatório.

### 2.3.5 Exibição

O monitor de vídeo é um elemento fundamental de um sistema de processamento de imagens. Os monitores em uso atualmente são capazes de exibir imagens com resolução de pelo menos 640 x 480 pixels com 256 cores distintas. A tecnologia usual ainda é o CRT (Tubo de Raios Catódicos).

Um CRT para um sistema de processamento de imagens normalmente segue um padrão de vídeo. O padrão de vídeo mais comum para sistemas monocromáticos é o RS-170. Ele prevê 480 linhas horizontais entrelaçadas, isto é, a varredura de um quadro é feita em duas etapas, abrangendo primeiramente as linhas ímpares e posteriormente as linhas pares. Cada uma destas etapas é denominada campo. O tempo necessário para percorrer um campo é 1/60 s; consequentemente, o tempo total de um quadro é 1/30 s. As características de persistência visual do olho humano fazem com que, nesta velocidade, a varredura

individual de cada campo não seja perceptível, bem como dão a impressão de que a sequência de quadros explorados é perfeitamente contínua.

A resolução espacial dos monitores é normalmente especificada em pontos por polegada (**dots per inch – dpi**). Um valor típico de resolução é 72 dpi, suficiente para exibir uma imagem de 640 x 480 num monitor de 14 polegadas. A título de comparação, uma tela de TV tem resolução na faixa de 40 dpi.

Um CRT colorido difere radicalmente de seu antecessor monocromático, por apresentar três feixes eletrônicos, cada um correspondente a uma das três cores primárias (vermelho, verde e azul). A superfície interna da tela é constituída por três tipo de fósforo, disposto de forma triangular, cada qual sensível a uma das cores primárias e excitado pelo respectivo canhão eletrônico. Isto significa dizer que, do ponto de vista construtivo, cada pixel é na verdade uma combinação de três pequenos pixels, um para cada cor primária. A Figura 2.10 mostra uma representação esquemática deste tipo de monitor.

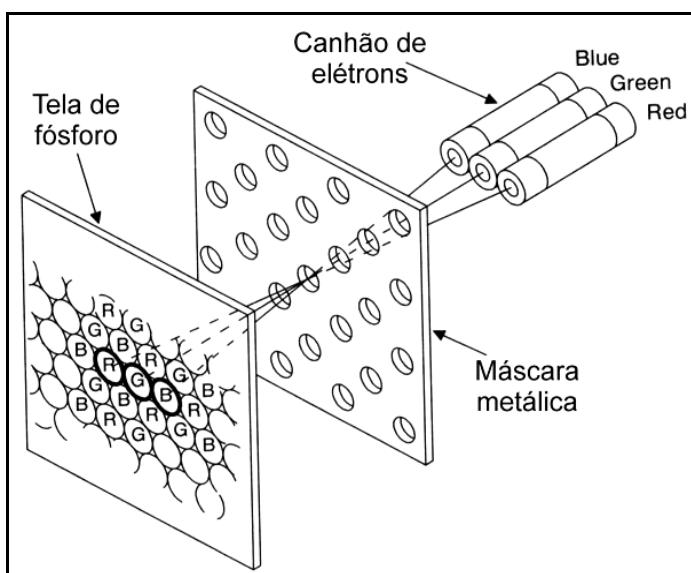


Figura 2.10 – Representação esquemática de um monitor CRT tricromático

A indústria de dispositivos de exibição vem apresentando sistematicamente novas tecnologias de fabricação de monitores de vídeo, dentre eles os monitores de cristal líquido (LCD), cada vez mais populares graças à disseminação dos computadores portáteis.

Existem diversas formas de reprodução de imagens em papel. A melhor, e mais cara, é a reprodução fotográfica, onde o número de gradações de cinza é função da densidade de grânulos de prata no papel. Outra possibilidade é o uso de papel sensível à temperatura, cuja composição química faz com que ele apresente coloração mais escura à medida que a temperatura aumenta. Este tipo de impressão ainda é o mais difundido em equipamentos de fax. Uma de suas desvantagens é o desvanecimento das imagens com o tempo. Nos últimos anos, aumentou consideravelmente a oferta de impressoras térmicas coloridas no mercado. Estas impressoras baseiam-se na deposição de cera colorida sobre um papel especial para produzir a impressão.

Dispositivos periféricos de saída, especializados na produção de cópias da imagem em forma de fotografias, slides ou transparências, também estão se tornando cada vez mais usuais.

Uma alternativa às técnicas fotográficas consiste no uso de técnicas de **halftoning**. É o método usado por jornais e por impressoras convencionais (laser,

matricial ou jato de tinta) para a impressão de imagens. Esta técnica consiste, basicamente, em imprimir pontos escuros de diferentes tamanhos, espaçados de tal maneira a reproduzir a ilusão de tons de cinza. À medida que a distância entre o observador e a imagem impressa aumentam, os detalhes finos vão desaparecendo e a imagem parece cada vez mais uma imagem contínua monocromática.

No jargão computacional, dá-se o nome de ***dithering*** ao processo de produção do efeito de ***halftoning***, bem como a todas as técnicas de conversão de uma imagem para adaptá-la a resoluções menores, tanto para efeito de exibição como para impressão. Existem vários algoritmos de ***dithering***, sendo o mais comum o de Floyd-Steinberg, que consiste de um processo adaptativo no qual o padrão de ***dither*** a ser atribuído a um pixel depende de seu tom de cinza e de seus vizinhos.

## 2.4 ELEMENTOS DE SISTEMAS DE VISÃO ARTIFICIAL

O objetivo da implementação de sistemas de visão artificial é dotar robôs com a capacidade de enxergar. Ao relacionarmos as dificuldades inerentes ao processo de dotar o computador de uma capacidade visual semelhante à dos seres humanos, deparamo-nos com três admiráveis características do processo de percepção visual humano, que são:

- uma base de dados muito rica;
- altíssima velocidade de processamento; e
- a capacidade de trabalhar sob condições muito variadas.

Os avanços na tecnologia de dispositivos de armazenamento de massa e o surgimento de novas CPUs e arquiteturas computacionais cada vez mais rápidas, com alto grau de paralelismo, nos induzem a crer que dispomos de condições cada vez melhores de modelar as duas primeiras características relacionadas anteriormente. O grande desafio permanece sendo o de fazer com que os sistemas de visão artificial trabalhem em diferentes condições de luminosidade, contraste, posicionamento relativo dos objetos em uma cena, sem perder a capacidade de interpretar a cena, de forma análoga à nossa capacidade de reconhecer um amigo ou parente com relativa facilidade, independentemente de ele estar usando óculos ou não, ter deixado crescer a barba ou estar no carro ao lado do nosso em uma esquina num final de tarde, onde não dispomos de outra imagem senão a vista de perfil e onde as condições de luminosidade são bastante inferiores às que obteríamos ao meio-dia.

### 2.4.1 Estrutura de um Sistema de Visão Artificial

Definiremos um Sistema de Visão Artificial (SVA) como um sistema computadorizado capaz de adquirir, processar e interpretar imagens correspondentes a cenas reais. A Figura 2.11 mostra esquematicamente um diagrama de blocos de um SVA. Suas principais etapas são explicadas a seguir, partindo da premissa de que um problema prático deve ser solucionado; por exemplo, a leitura do Código de Endereçamento Postal (CEP) de um lote de envelopes.

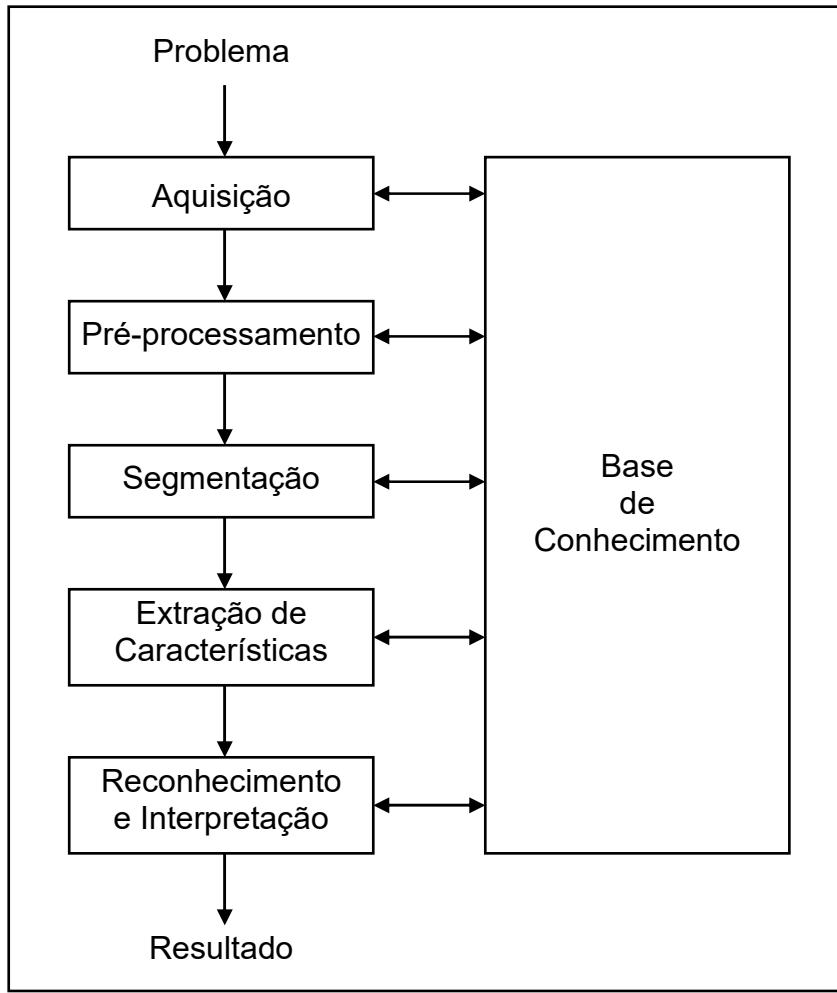


Figura 2.11 – Um Sistema de Visão Artificial (SVA) e suas principais etapas

#### 2.4.2 Domínio do Problema e Resultado

O domínio do problema, neste caso, consiste no lote de envelopes e o objetivo do SVA é ler o CEP presente em cada um deles. Logo, o resultado esperado é uma sequência de dígitos correspondentes ao CEP lido.

#### 2.4.3 Aquisição da Imagem

O primeiro passo no processo de aquisição de imagens dos envelopes. Para tanto são necessários um sensor e um digitalizador. O sensor converterá a informação óptica em sinal elétrico e o digitalizador transformará a imagem analógica em imagem digital.

Dentre os aspectos de projeto envolvidos nesta etapa, pode-se mencionar a escolha do tipo de sensor, o conjunto de lentes a utilizar, as condições de iluminação da cena, os requisitos de velocidade de aquisição, a resolução e o número de níveis de cinza da imagem digitalizada, dentre outros. Esta etapa produz à saída uma imagem digitalizada do envelope.

#### 2.4.4 Pré-Processamento

A imagem resultante do passo anterior pode apresentar diversas imperfeições, tais como: presença de pixels ruidosos, contraste e/ou brilho inadequado, caracteres interrompidos ou indevidamente conectados, etc. A função da etapa de pré-processamento é aprimorar a qualidade da imagem para as etapas subsequentes. As operações efetuadas nesta etapa são ditas de baixo nível porque trabalham diretamente com os valores de intensidades dos pixels, sem nenhum conhecimento sobre quais deles pertencem aos dígitos do CEP, a outras informações impressas no envelope ou ao fundo. A imagem resultante desta etapa é uma imagem digitalizada de melhor qualidade que a original.

#### 2.4.5 Segmentação

A tarefa básica da etapa da segmentação é a de dividir uma imagem em suas unidades significativas, ou seja, nos objetos de interesse que a compõem. Esta tarefa, apesar de simples de descrever, é das mais difíceis de implementar.

No caso específico do CEP, é possível que o problema seja dividido em duas etapas: em um primeiro momento, os algoritmos de segmentação tentarão localizar o CEP para, posteriormente, trabalhando sobre esta sub-imagem, segmentar cada dígito individualmente. Segundo esta linha de raciocínio, este bloco produzira à saída oito sub-imagens, cada qual correspondendo a um dígito do CEP.

#### 2.4.6 Extração de Características

Esta etapa procura extrair características das imagens resultantes da segmentação através de descritores que permitam caracterizar com precisão cada dígito e que apresentem bom poder de discriminação entre dígitos parecidos, como o “5” e o “6”. Estes descritores devem ser representados por uma estrutura de dados adequada ao algoritmo de reconhecimento. É importante observar que nesta etapa a entrada ainda é uma imagem, mas a saída é um conjunto de dados correspondentes àquela imagem.

Para maior clareza, suponhamos que os descritores utilizados para descrever um caractere sejam as coordenadas normalizadas  $x$  e  $y$  de seu centro de gravidade e a razão entre sua altura e largura. Neste caso, um vetor de três elementos é uma estrutura de dados adequada para armazenar estas informações sobre cada dígito processado por essa etapa.

#### 2.4.7 Reconhecimento e Interpretação

Nesta última etapa do sistema, denominamos reconhecimento o processo de atribuição de um rótulo a um objeto baseado em suas características, traduzidas por seus descritores. A tarefa de interpretação, por outro lado, consiste em atribuir significado a um conjunto de objetos reconhecidos. Neste exemplo, uma forma simples de interpretação seria a verificação do CEP em uma base de dados de CEPs válidos, para descobrir se o conjunto dos oito caracteres faz sentido ou não.

#### 2.4.8 Base de Conhecimento

Todas as tarefas das etapas, descritas antes, pressupõem a existência de um conhecimento sobre o problema a ser resolvido, armazenado em uma base de conhecimento, cujo tamanho e complexidade podem variar enormemente. Idealmente, esta base de conhecimento deveria, não somente guiar o funcionamento de cada etapa, mas também permitir a realimentação entre elas. Por exemplo, se a etapa de representação e descrição (extração de características) recebesse 7 caracteres ao invés de 8, ela deveria ser capaz de realimentar a etapa de segmentação para que esta procurasse segmentar novamente a sub-imagem “suspeita” (aquele de maior largura), buscando dividi-la em duas. Esta integração entre as várias etapas, através da base de conhecimento, ainda é um objetivo difícil de alcançar e não está presente na maioria dos SVAs existentes atualmente.

### 2.5 PROPRIEDADES ESTATÍSTICAS DE IMAGENS

Medidas estatísticas são capazes de sintetizar, numericamente, algumas características das imagens digitais. As propriedades abordadas nesta seção são a média, variância, covariância e correlação. Outro elemento da estatística capaz de sintetizar, graficamente, características das imagens é o histograma de frequências.

#### 2.5.1 Histograma

O histograma de uma imagem é simplesmente um conjunto de números indicando o percentual de pixels, naquela imagem, que apresentam um determinado nível de cinza. Estes valores são normalmente representados por um gráfico de barras que fornece para cada nível de cinza o número ou o percentual de pixels correspondentes na imagem. Através da visualização do histograma de uma imagem obtemos uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio, ou seja, se a imagem é predominantemente clara ou escura.

Cada elemento deste conjunto é calculado como:

$$p_r(r_k) = \frac{n_k}{n}$$

onde:

- $0 \leq r_k \leq 1$ ;
- $k = 0, 1, \dots, L - 1$ , onde  $L$  é o número de níveis de cinza da imagem digitalizada;
- $n$  = número total de pixels na imagem;
- $p_r(r_k)$  = probabilidade do  $k$ -ésimo nível de cinza;
- $n_k$  = número de pixels cujo nível de cinza corresponde a  $k$ .

- **Exemplo 2.1:**

Os dados da Tabela 2.1 correspondem a uma imagem de 128 x 128 pixels, com 8 níveis de cinza. O número de pixels correspondentes a um certo tom de cinza está indicado na coluna  $n_k$ , enquanto as respectivas probabilidades  $p_r(r_k)$  aparecem na coluna seguinte. A representação gráfica equivalente deste histograma é mostrada na Figura 2.12.

TABELA 2.1 – Exemplo de histograma

Nível de Cinza	$n_k$	$p_{r(r_k)}$
0	1120	0,068
1/7	3214	0,196
2/7	4850	0,296
3/7	3425	0,209
4/7	1995	0,122
5/7	784	0,048
6/7	541	0,033
1	455	0,028
Total	16384	1

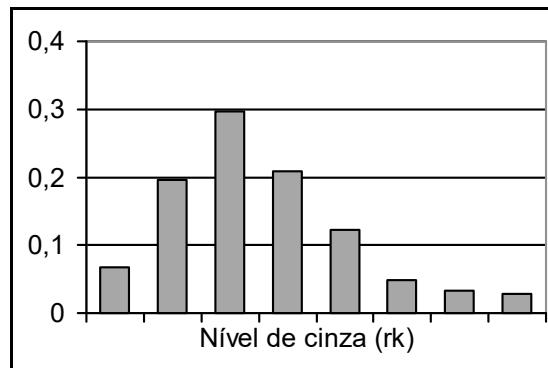
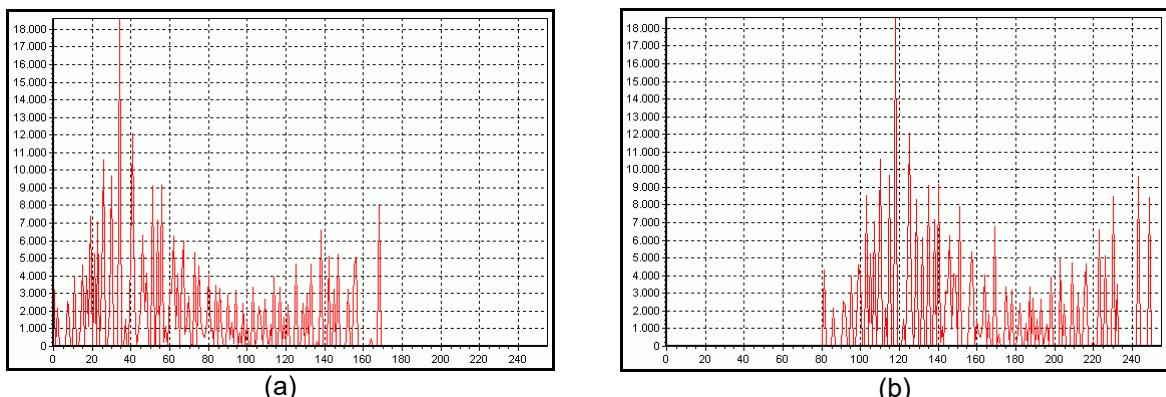


Figura 2.12 – Exemplo de histograma para imagem com oito níveis de cinza

A Figura 2.13 apresenta cinco exemplos de tipos de histogramas freqüentemente encontrados em imagens.

O histograma da Figura 2.13(a) apresenta grande concentração de pixels nos valores mais baixos de cinza, correspondendo a uma imagem predominantemente escura. Na Figura 2.13(b) os pixels estão concentrados em valores próximos ao limite superior da escala de cinza, caracterizando uma imagem clara. Na parte (c) da figura, os pixels estão agrupados em torno de valores intermediários de cinza, correspondendo a uma imagem de brilho médio. Nas figuras (a), (b) e (c) a maioria dos pixels estão concentrada em uma estreita faixa da escala de cinza, significando que as imagens correspondentes apresentam baixo contraste.

A Figura 2.13(d) corresponde a uma imagem com pixels distribuídos ao longo de toda a escala de cinza. É comum dizer que uma imagem com estas características apresenta um bom contraste. A Figura 2.13(e) mostra um histograma tipicamente bimodal, isto é, apresentando duas concentrações de pixels, uma delas em torno de valores escuros e outra na região clara da escala de cinza. Pode-se dizer que a imagem correspondente apresenta alto contraste entre as duas concentrações, uma vez que elas se encontram razoavelmente espaçadas.



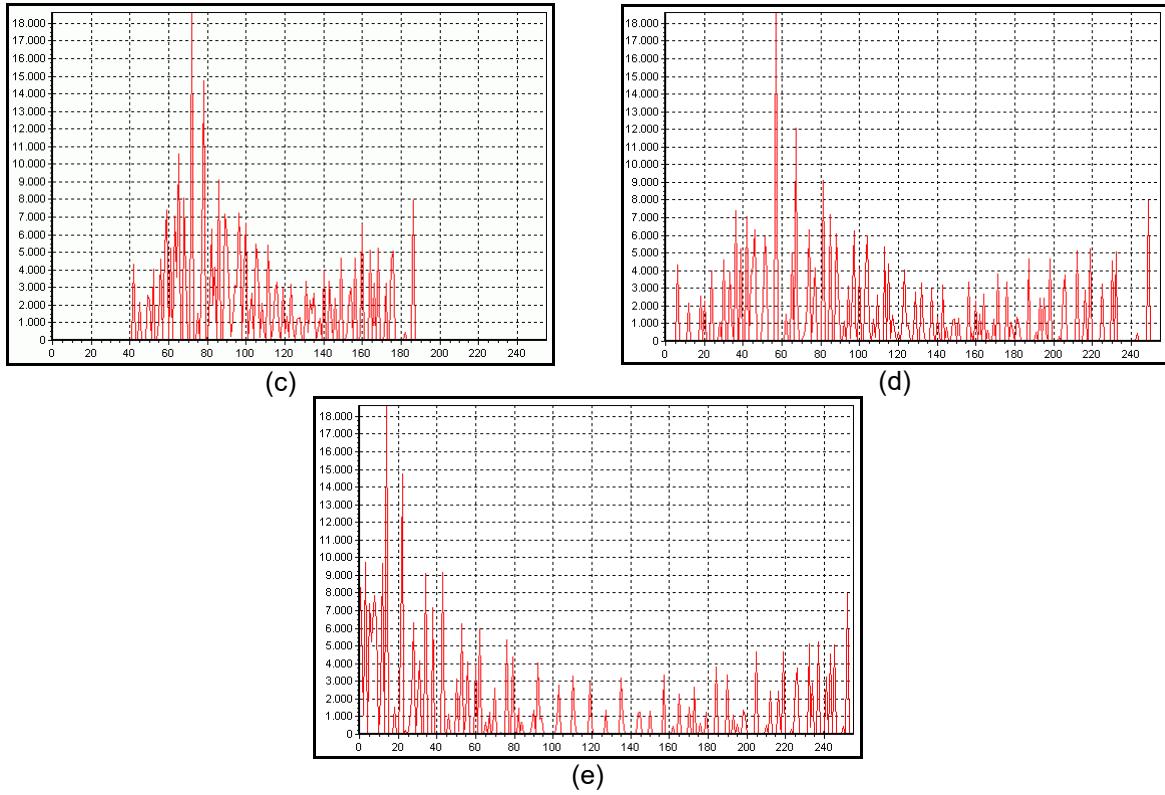
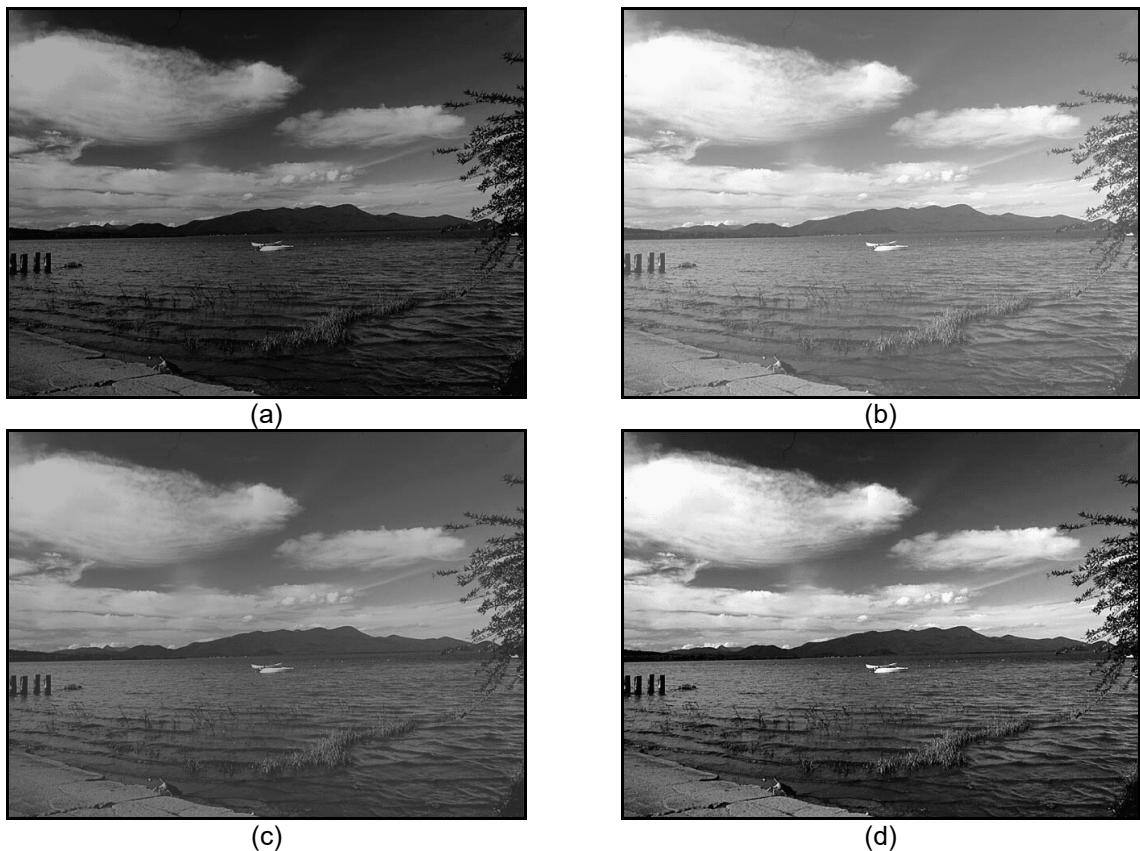


Figura 2.13 – Exemplos de histogramas

Para verificar a relação entre imagens e respectivos histogramas, a Figura 2.14 mostra cinco imagens monocromáticas cujos histogramas são aqueles da Figura 2.13.





(e)

Figura 2.14 – Imagens correspondentes aos histogramas da Figura 2.13

Histogramas não se restringem a imagens em tons de cinza e podem ser aplicados individualmente aos componentes RGB de imagens coloridas (Figura 2.15). A manipulação cuidadosa dos histogramas é ferramenta fundamental na melhora da visualização de imagens bem como na extração de informações pouco perceptíveis nas imagens originais.

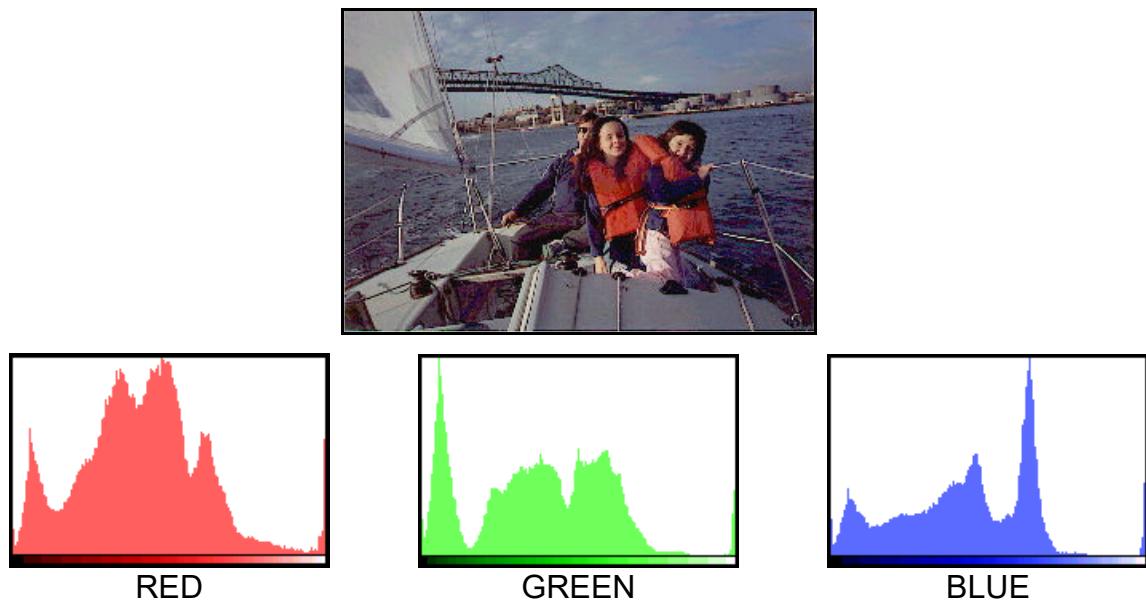


Figura 2.15 - Histograma de imagem colorida por banda

Para computar o histograma de uma imagem monocromática, inicializa-se com zero todos os elementos de um vetor de  $L$  elementos, onde  $L$  é o número de tons de cinza possíveis. Em seguida, percorre-se ao imagem, pixel a pixel, e incrementa-se a posição do vetor, cujo índice corresponde ao tom de cinza do pixel visitado. Após toda a imagem ter sido percorrida, cada elemento do vetor conterá o número de pixels, cujo tom de cinza equivale ao índice do elemento. Estes valores poderão ser normalizados, dividindo cada um deles pelo total de pixels na imagem.

O histograma de uma imagem fornece diversas informações qualitativas e quantitativas sobre ela como, por exemplo, o nível de cinza mínimo, médio e máximo, predominância de pixels claros ou escuros, etc. Entretanto, outras conclusões de caráter qualitativo, como por exemplo a qualidade subjetiva global da imagem, presença ou não de ruído, etc, somente podem ser extraídas dispondendo-se da imagem propriamente dita.

### 2.5.2 Média Amostral

Dada uma imagem digital monocromática  $\mathbf{g}$ , com  $m$  linhas e  $n$  colunas, podemos calcular a média amostral  $m_g$

$$m_g = \frac{1}{m \cdot n} \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} g(j, k)$$

A média de uma imagem representa a impressão geral de brilho. Uma imagem escura, com pouco brilho, apresenta média baixa enquanto que uma imagem com alto brilho apresenta média alta, maior que 128 por exemplo.

### 2.5.3 Variância

A variância de uma imagem é uma medida da variação dos níveis de cinza da imagem em relação à sua média. Esta medida dá uma idéia de contraste da imagem: uma imagem com grande  $V_g$  tem alto contraste.

$$V_g = \frac{1}{m \cdot n} \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} (g(j, k) - m_g)^2$$

### 2.5.4 Covariância

A covariância é uma grandeza que relaciona duas variâncias específicas e descreve o grau de correlação existente entre cada par de imagens (ou bandas). O valor da covariância entre as imagens  $\mathbf{g}_1$  e  $\mathbf{g}_2$  pode ser calculada da seguinte forma:

$$C_{g_1 g_2} = \frac{1}{m \cdot n} \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} (g_1(j, k) - m_{g_1})(g_2(j, k) - m_{g_2})$$

onde  $m_{g_1}$  e  $m_{g_2}$  são as médias das imagens  $\mathbf{g}_1$  e  $\mathbf{g}_2$ , respectivamente. A covariância dá uma medida do quanto de informações contidas em um par de imagens é comum a ambas. A covariância pode ser tanto positiva, quanto negativa. Neste último caso os dados são ditos negativamente correlacionados.

### 2.5.5 Correlação

A correlação possui a mesma interpretação que a covariância. A correlação é uma medida normalizada:  $-1 < \rho < 1$ .

$$\rho_{g_1 g_2} = \frac{C_{g_1 g_2}}{\sqrt{V_{g_1} \cdot V_{g_2}}}$$

## 2.6 DISCRETIZAÇÃO DE COR

Vimos anteriormente que o processo de discretização de cor é conhecido pelo nome de **quantização**. Esse processo permite a conversão de uma imagem com um conjunto contínuo de cores, em uma imagem com um conjunto de cores discreto. Ressaltamos mais uma vez que, muito embora a conversão de um número real para a representação de ponto flutuante também implique numa discretização, vamos considerar que essa representação é fiel para os propósitos da computação e não envolve um processo de quantização.

Nesta seção vamos estudar com detalhe os diversos problemas existentes na discretização de cor. Para isso, precisamos caracterizar de forma mais precisa o conceito de quantização. Chamamos de **quantização** a uma transformação sobrejetiva  $q: \mathbf{C} \rightarrow \mathbf{C}'$ , de um sólido de cor  $\mathbf{C}$  no qual as cores são representadas usando  $M$  bits, para um sólido de cor  $\mathbf{C}'$  que utiliza uma codificação do vetor de cor com  $N$  bits, sendo  $M > N$ . Denominamos  $q$  de **transformação de quantização**, ou simplesmente **quantização de  $N$  bits**. O espaço de  $N$  bits  $\mathbf{C}'$  é chamado de **espaço de quantização**.

A quantização de uma imagem digital consiste em quantizar o gamute de cores da imagem, o que acarreta na quantização da informação de cor de cada pixel da imagem. Mais precisamente, se  $i: U \rightarrow \mathbf{C}$ , é uma imagem discreta-contínua ou discreta-discreta, o resultado da quantização de  $i(x, y)$  é uma imagem discreta-discreta  $i': U \rightarrow \mathbf{C}'$ , tal que  $i'(x, y) = q(i(x, y))$ , onde  $q$  é a transformação de quantização. Desse modo, a quantização altera a resolução de cor da imagem.

Se os espaços de cor  $\mathbf{C}$  e  $\mathbf{C}'$  são unidimensionais, a quantização é chamada de **quantização escalar**, ou **quantização unidimensional**. Caso contrário, ela é chamada de **quantização vetorial** ou **quantização multidimensional**. Note que se os espaços de cor não são unidimensionais, podemos fazer uma quantização vetorial usando um método de quantização escalar em cada uma das componentes do espaço. Esse método de quantização vetorial, no entanto, não explora a correlação existente entre as diversas componentes de cor de uma imagem.

- **Exemplo 2.2 – Quantização de dois níveis:**

Consideremos o problema de quantizar um espaço monocromático de cores com 256 níveis de intensidade de cinza (gamute de 8 bits), para um espaço de cor com apenas dois níveis (gamute de 1 bit). Um possível método consiste em quantizar as cores com intensidade abaixo do valor médio de intensidade para 0, e o restante das cores para o valor máximo de intensidade 255. Esse processo de quantização faz um particionamento do espaço de cor em dois conjuntos. Os elementos de um dos conjuntos são quantizados para a intensidade 255. O trecho de pseudo-código abaixo mostra essa quantização:

```
if (Color_Intensity < 127) {
    Color_Intensity = 0;
} else {
    Color_Intensity = 255;
}
```

Por que quantizar uma imagem? Existem duas motivações básicas que justificam a importância do problema de quantizar uma imagem: exibição e

compressão.

- Exibição de imagens:

Para exibir uma imagem em algum dispositivo gráfico, o gamute de cor da imagem não pode ser maior do que as cores disponíveis no espaço físico de cor do equipamento. Nesse caso, o espaço de quantização  $\mathbf{C}'$  está diretamente ligado ao espaço de cor do dispositivo gráfico de exibição.

- Compressão de imagens:

A quantização de uma imagem, permite uma redução do número de bits utilizado para armazenar o seu gamute de cores. Reduzimos desse modo o espaço necessário para o armazenamento da imagem, e diminuímos o volume de dados no caso de transmissão da imagem através de algum canal de comunicação.

### 2.6.1 Célula de Quantização

Conforme vimos no exemplo anterior, uma quantização para um espaço de cor de 1 bit, gamute de 2 cores, determina uma partição do espaço inicial de cores em 2 conjuntos. Em cada um desses conjuntos a função de quantização assume um valor constante. Esse é um fato que ocorre em geral. Consideraremos uma transformação de quantização  $q: \mathbf{C} \rightarrow \mathbf{C}'$ . A cada cor quantizada  $c'_i \in \mathbf{C}'$ , corresponde um subconjunto de cores  $C_i \subset \mathbf{C}$ , determinado pelas cores do espaço  $\mathbf{C}$  que são quantizadas para a cor  $c_i$ , ou seja,

$$C_i = q^{-1}(c'_i) = \{c \in C; q(c) = c'_i\}$$

A família, finita, de conjunto  $C_i$ , constitui uma partição do espaço de cor  $\mathbf{C}$ . Cada um dos conjuntos  $C_i$  da partição é chamado de **célula de quantização**. Em cada uma dessas células a função de quantização assume um valor constante  $c'_i$ , que é chamado de **nível de quantização** ou **valor de quantização**.

No caso da quantização unidimensional, sejam  $q_i$ ,  $1 \leq i \leq L$ , os níveis de quantização assumidos pela transformação de quantização  $q$ . As células de quantização neste caso são intervalos

$$c_{1-i} < c \leq c_i, \quad 1 \leq i \leq L$$

Na Figura 2.16(a) mostramos três intervalos de quantização  $[c_{i-1}, c_i]$ , e em cada intervalo temos o nível de quantização associado  $q_i$ , que é constante em cada intervalo de quantização.

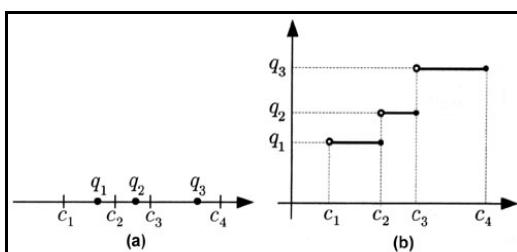


Figura 2.16 – Níveis de quantização e gráfico da função de quantização

Na quantização unidimensional a célula de quantização é sempre um intervalo e podemos variar apenas o seu comprimento. Na quantização multidimensional as células de quantização são regiões do espaço de cor que podem apresentar uma geometria bem mais complexa. A Figura 2.17 mostra um exemplo de quantização bidimensional com oito células e, portanto, oito níveis de quantização (3 bits).

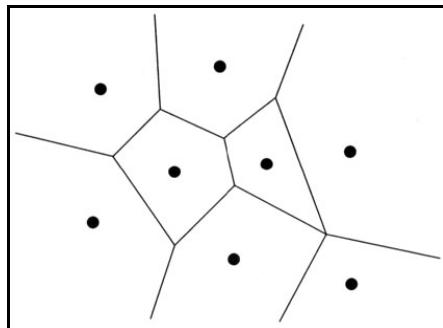


Figura 2.17 – Célula de quantização bidimensional

## 2.6.2 Percepção e Quantização

Considere uma função imagem  $f: \mathbf{U} \rightarrow \mathbf{C}$  monocromática cujo espaço de cor está quantizado em  $L$  níveis. Essa quantização determina uma partição do domínio  $\mathbf{U}$  da imagem em subconjuntos (células de quantização)  $\mathbf{C}_i$  definidos por

$$\mathbf{C}_i = f^{-1}(c_i) = \{(x, y) \in \mathbf{U}; f(x, y) = c_i\}$$

Ou seja, cada subconjunto  $\mathbf{C}_i$  da partição é constituído pelos pixels da imagem cuja intensidade assume o nível de quantização  $c_i$ . Se tivermos um número pequeno de níveis de quantização, teremos um número pequeno de regiões  $\mathbf{C}_i$  na partição. Além disso, se a função imagem for bem comportada a fronteira que separa essas regiões será definida por uma curva regular. Dependendo da diferença de valor entre os níveis de quantização de duas células vizinhas, a curva de fronteira será perceptível ao olho humano. Esse fenômeno é conhecido pelo nome de **contorno de quantização**.

As imagens (a), (b), (c) e (d) da Figura 2.18 ilustram o problema de contorno de quantização. Na imagem (a) temos uma imagem com 256 níveis de quantização; na figura (b) temos apenas 16 níveis; na figura (c) temos 8 níveis e na figura (d) temos 2 níveis de quantização. É claro, que à medida que diminuímos o número de níveis de quantização, o contorno de quantização fica mais perceptível. Do ponto de vista perceptual, o estudo de diferentes métodos de quantização está ligado à obtenção de uma quantização na qual o contorno de quantização não seja perceptível.

A percepção do contorno de quantização depende do número de níveis de quantização e do método de quantização utilizado. Em geral, para imagens monocromáticas de meio tom, 256 níveis de quantização (8 bits) são suficientes para evitar a percepção do contorno de quantização, independente do método de quantização utilizado. Esse fato é ilustrado pela Figura 2.18(a) onde a imagem mostrada está quantizada em 8 bits. Para imagens a cores, em geral é suficiente utilizar uma quantização de 24 bits, com 8 bits de quantização para cada componente de cor do espaço RGB.

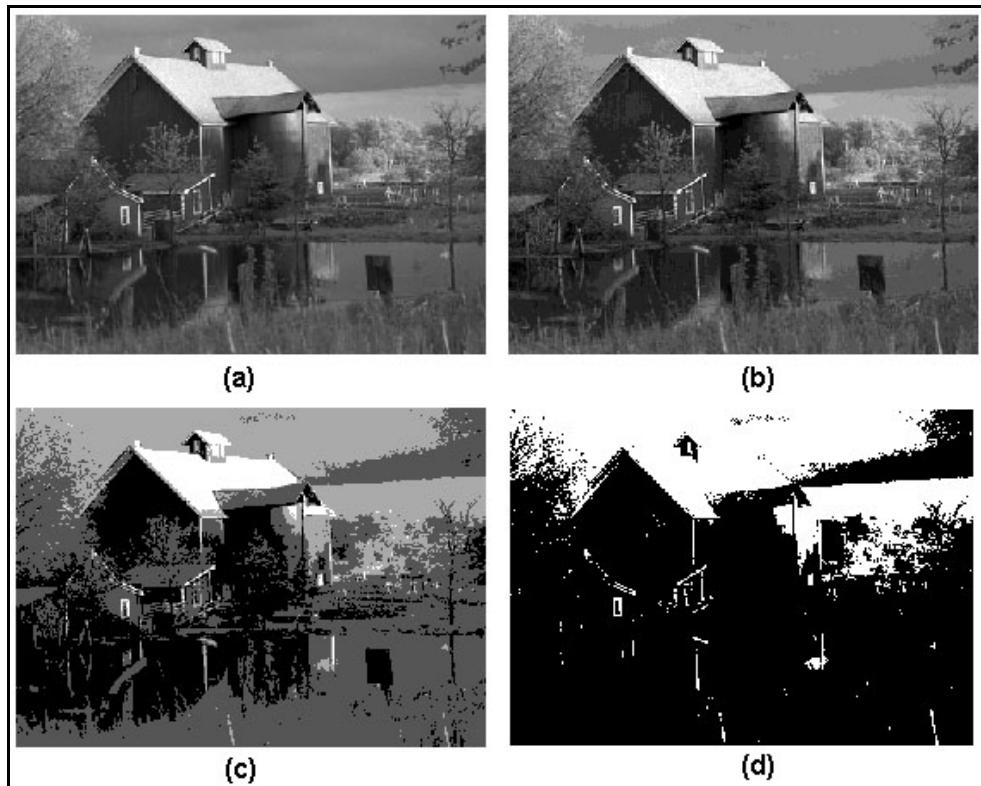


Figura 2.18 – Contorno de quantização para diferentes níveis de quantização

No entanto, dependendo da imagem e do método de quantização utilizado, podemos reduzir o número de níveis de quantização sem que seja possível perceber o contorno de quantização.

A percepção do contorno de quantização é agravada devido ao fenômeno perceptual conhecido por **bandas de mach** ("mach band"): o olho humano acentua transição de intensidade de cor, o que faz com que possamos perceber mais facilmente níveis de intensidade, mesmo próximos, se eles constituem regiões adjacentes da imagem. Esse fenômeno é ilustrado pelas duas imagens da Figura 2.19. A imagem (A) mostra 5 faixas verticais de intensidades diferentes. As faixas da esquerda são mostradas na imagem (B), porém com uma separação. Pode-se perceber claramente que a diferença de intensidade entre faixas adjacentes é bem mais acentuada na imagem (a), quando as faixas formam regiões contíguas da imagem.

O forte correlacionamento do erro de quantização entre os diversos pixels da imagem faz com que a fronteira entre duas regiões de quantização com níveis distintos seja uma curva conexa, sendo portanto mais fácil de ser percebida pelo olho humano devido ao fenômeno das bandas de Mach descrito anteriormente.

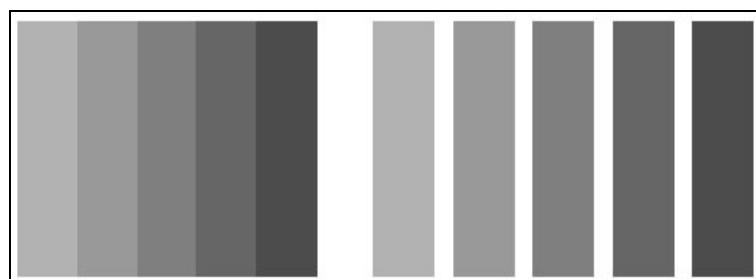


Figura 2.19 - Bandas de Mach

Na quantização de dois níveis o fenômeno de contorno aparece de forma bastante acentuada pois temos apenas duas intensidades na imagem final: preto e branco. Um método utilizado para se evitar a percepção do contorno de quantização, mesmo no caso mais crítico de quantização de imagens de meio-tom para dois níveis, é o "dithering".

O dithering consiste em um processo de filtragem cuja finalidade é descorrelacionar o erro de quantização, evitando desse modo que a fronteira entre dois níveis de quantização seja uma curva conexa. O processo de dithering é de grande importância no caso da quantização de dois níveis.

O fato de que a qualidade dos métodos de quantização está ligada a aspectos perceptuais de cor, tem sido bastante explorado, tecnologicamente, na busca de bons métodos de quantização. Um exemplo desse fato ocorre no sinal de televisão a cores NTSC, que é codificado a partir de coordenadas de cor no espaço YUV onde  $Y$  é a luminância e  $(U, V)$  são as coordenadas de crominância. Na quantização das componentes, podemos usar um número maior de bits para quantizar a informação de luminância, e um número menor para quantizar cada componente com a informação de crominância. Isso pode ser feito porque o olho humano é muito mais sensível à variação de luminância do que à variação de crominância.

### 2.6.3 Método Geral de Quantização

O processo de quantização de um espaço de cor consiste de duas etapas:

- Determinar as células de quantização;
- Determinar o nível de quantização em cada célula.

A função de quantização  $q$  é então definida de forma a associar às cores de uma célula, o nível de quantização correspondente. A função  $q$  é portanto constante em cada célula de quantização. Uma vez conhecida a transformação de quantização  $q$  do espaço de cor, o processo de quantizar uma imagem se torna uma tarefa simples: para cada cor  $c$  do pixel na imagem original, identifica-se a célula de quantização a que ela pertence, e substituímos a cor  $c$  pelo valor de quantização,  $q(c)$ , da célula.

Os diferentes métodos de quantização existentes refletem a descrição acima, e funcionam de três modos distintos:

- Determinamos inicialmente as células de quantização, e em seguida calculamos o nível de quantização de cada célula;
- Determinamos inicialmente os níveis de quantização, e em seguida determinamos as cores que devem ser quantizadas para cada nível;
- Determinamos de forma interdependente e simultânea as células e os níveis de quantização.

### 2.6.4 Erro de Quantização

A determinação ótima das células de quantização, e dos níveis de quantização em cada célula, depende do critério utilizado para medir o erro de

quantização, bem como da distribuição de cor na imagem. Vamos examinar esse problema com mais detalhe. Se  $q$  é a transformação de quantização e  $c$  uma cor a ser quantizada, então

$$c - q(c) = e_q$$

onde  $e_q$  é o erro introduzido no processo de quantização, também chamado de **ruído de quantização**. O quadrado  $e_q^2$  do ruído de quantização pode ser considerado como uma medida  $d(c, q(c))$  entre a cor  $c$  e seu valor quantizado  $q(c)$ . De modo mais preciso, devemos tomar uma métrica  $d$  no espaço de cor  $C$  e considerar o espaço de quantização  $C'$  como um subconjunto de  $C$ . A **medida de distorção** da quantização  $q(c)$  da cor  $c$  é dada então pelo erro médio quadrático

$$E((c, q(c))) = \int_{-\infty}^{+\infty} p(c) d(c, q(c)) dc$$

onde  $p$  é a função de distribuição de probabilidade da cor  $c$  no espaço de cor da imagem. O uso da equação acima para medida da distorção introduzida pela quantização é bastante intuitivo: na medida do erro devemos levar em consideração a probabilidade de ocorrência da cor  $c$  no espaço de cores a ser quantizado, o erro de quantização é então modulado pela probabilidade, resultando em uma média ponderada.

Diversas métricas  $d$  no espaço de cor podem ser escolhidas, com o objetivo de medir a qualidade do processo de quantização, a escolha dessas métricas deve levar em conta critérios perceptuais de cor. Além disso, devemos também levar em consideração problemas de eficiência computacional. Por essa razão, é comum a utilização de pseudo-métricas ou até mesmo outras funções positivas que, de algum modo, forneçam informações sobre "proximidade" no espaço de cor. Uma escolha possível é o quadrado da distância euclidiana, ou seja,  $d(c_1, c_2) = \langle c_2 - c_1, c_2 - c_1 \rangle$ , onde  $\langle \rangle$  é um produto interno no espaço de cor.

## 2.6.5 Quantização Uniforme e Adaptativa

Vimos anteriormente que um método de quantização pode facilmente ser definido a partir da escolha das células de quantização. Como determinar a célula de quantização? Uma escolha natural e simples à primeira vista, consiste em dividir o espaço de cor em células congruentes e em cada célula tomar o seu "centro" como sendo o nível de quantização associado. Esse método é chamado de **quantização uniforme**. No caso de quantização escalar com  $L$  níveis as células de quantização são intervalos  $(c_{i-1}, c_i]$  de igual comprimento, isto é,  $c_i - c_{i-1} = \text{constante}$ , e em cada célula o valor de quantização é dado pela média

$$q_i = \frac{c_i + c_{i-1}}{2}, \quad 1 \leq i \leq L$$

Se o espaço de cor é o cubo RGB, e utilizamos quantização uniforme em cada uma das componentes de cor, as células de quantização são cubos do espaço de cor, e em cada cubo o valor de quantização é dado pela cor no centro do cubo. A Figura 2.20(a) ilustra as células de quantização para o caso bidimensional. Na Figura 2.20(b) ilustramos uma outra geometria de células de quantização uniforme bidimensional.

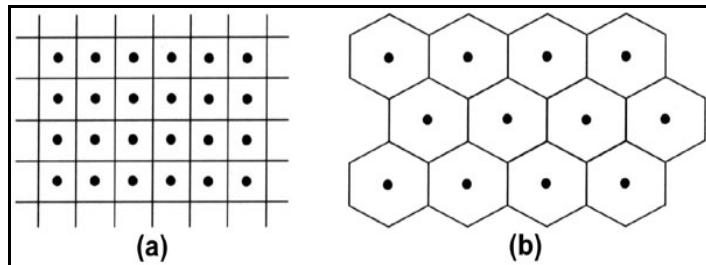


Figura 2.20 - Células de quantização uniforme bidimensional

Apesar de que a quantização uniforme é fácil de ser obtida, sua utilização pode não ser a mais recomendada. Basta observar que nesse método algumas células de quantização podem nem sequer possuir cores existentes no gamute da imagem.

Suponha que os valores de cor da imagem não estão uniformemente distribuídos e que, portanto, certas cores contidas em uma região do espaço de cor ocorrem na imagem com uma frequência maior do que as outras cores. Se subdividirmos esta região em um número maior de células de quantização, estaremos diminuindo o tamanho das células, e portanto haverá uma diminuição do erro de quantização dos elementos da imagem que possuem aquelas cores. Como essas cores ocorrem em grande número estamos, consequentemente, minimizando a diferença entre a imagem original e a imagem quantizada.

Um método de quantização que não utiliza um particionamento do espaço de cor em células congruentes é chamado de **quantização não-uniforme**. A quantização não uniforme é dita **adaptativa** quando a geometria das células é escolhida de acordo com características específicas da distribuição de cor na imagem.

## 2.6.6 Métodos Adaptativos de Quantização

Nesta seção vamos analisar vários métodos de quantização adaptativa de cor. Esses métodos podem ser descritos em duas etapas: na primeira etapa, é feita uma estimativa das propriedades estatísticas relevantes da imagem. Na segunda etapa, aplica-se o método geral de quantização, particionando-se o espaço de cor com base nos dados obtidos na fase anterior.

Dessa forma, inicialmente devemos construir um histograma de frequência da imagem que nos dá uma aproximação da função de distribuição de probabilidade das cores na imagem. Para reduzir o tamanho da memória ocupada pelo histograma, é comum fazer-se previamente uma quantização uniforme da imagem. Em geral utilizam-se 5 bits para cada uma das três componentes. Estamos supondo que uma imagem a cores é quantizada inicialmente, sem perda perceptual, utilizando 8 bits por componente de cor no espaço RGB.

Na Figura 2.21 mostramos uma imagem que utilizaremos para comparar os métodos de quantização a serem apresentados. A imagem da figura foi quantizada em 24 bits (8 bits por canal R, G e B), não apresentando, portanto, contornos de quantização perceptíveis.

Para efeitos de comparação posterior com outros algoritmos de quantização, mostramos na Figura 2.22(a) e (b) uma quantização uniforme da imagem na Figura 2.21 utilizando, respectivamente, 8 e 4 bits. Os contornos de quantização são perceptíveis nessas duas imagens.



Figura 2.21 - Reprodução de uma imagem digital a cores com 24 bits



Figura 2.22 - Quantização uniforme com 8 bits(a) e 4 bits(b)

- Quantização por seleção direta:

O método de seleção direta escolhe, inicialmente, com base em propriedades estatísticas da imagem, os níveis de quantização a serem utilizados e a partir daí determina a função de quantização de forma a minimizar o erro de quantização. Um exemplo desse método é o **algoritmo de populosidade**.

Esse método (algoritmo de populosidade) constrói inicialmente o histograma de frequência da imagem e, em seguida, escolhe para os  $K$  níveis de quantização as  $K$  cores que ocorrem com maior frequência no gamute da imagem (cores mais populosas). A função de quantização pode ser definida tomando para cada cor  $c$  do gamute da imagem,  $q(c)$  como sendo o nível de quantização mais próximo usando, por exemplo, o quadrado da métrica euclidiana. É claro que se houver mais de um nível de quantização atendendo a condição de minimalidade, devemos tomar uma decisão sobre o valor de  $q(c)$ . Uma possível solução é escolher aleatoriamente um dos possíveis níveis. Uma escolha mais prudente deve levar em conta os valores de quantização dos pixels vizinhos.

O problema do algoritmo de populosidade está em ignorar totalmente cores em regiões de baixa densidade do espaço de cor (um "highlight" em uma imagem pode desaparecer completamente nesse processo de quantização, uma vez que ele ocupa apenas um pequeno número de pixels da imagem). O algoritmo pode ser utilizado, com resultados satisfatórios, para imagens que apresentem uma

distribuição uniforme de cor.

A Figura 2.23(a) mostra uma reprodução das Figura 2.21 quantizada para 8 bits com o algoritmo de populosidade. Na Figura 2.23(b) mostramos a mesma imagem quantizada para 4 bits com o mesmo algoritmo. O leitor deve lembrar que a quantização uniforme da imagem acima, para 8 e 4 bits, pode ser vista nas Figuras 2.23(a) e (b), respectivamente.



Figura 2.23 - Algoritmo de populosidade  
(a) quantização com 8 bits (b) quantização com 4 bits

- Quantização por subdivisão recursiva:

Enquanto os métodos de seleção direta partem da escolha dos níveis de quantização para determinar a função de quantização, os métodos de subdivisão recursiva determinam inicialmente as células de quantização, para em seguida calcular a função de quantização em cada célula. O nome para o método deve-se ao fato de que ele utiliza um processo de subdivisão recursiva do espaço de cores com a finalidade de determinar as células de quantização.

Iniciamos com uma região do espaço de cor contendo o gamute de cores da imagem. Em cada iteração, essa região é subdividida em duas ou mais subregiões do espaço. Essa subdivisão é baseada em características estatísticas sobre a distribuição de cores nos diversos pixels da imagem. O processo de recursão continua até que não existam mais cores da imagem original contidas em algum conjunto da divisão, ou até que se obtenha o número desejado de células de quantização, que corresponde ao número de níveis. Uma vez determinadas as células de quantização, a função de quantização é obtida através da escolha de um nível de quantização em cada célula.

Um método simples e eficaz de quantização por subdivisão recursiva, consiste em quantizar uma imagem escolhendo os níveis de quantização de forma que cada nível de quantização seja utilizado pelo mesmo número de pixels. Usando o histograma da imagem, essa transformação de quantização efetua uma equalização do histograma da imagem, ou seja, substitui o histograma original por um histograma correspondente a uma distribuição uniforme das intensidades dos pixels, conforme ilustrado na Figura 2.24.

A medida estatística que permite obter uma transformação de quantização, com a propriedade de equalizar o histograma, é a **mediana**.

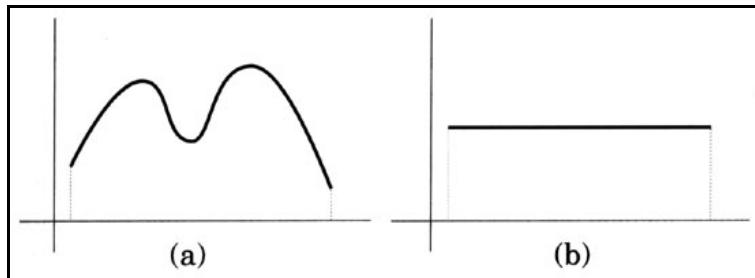


Figura 2.24 – (a) Histograma original; (b) Histograma equalizado

A **mediana**  $mc$  de um conjunto finito e ordenado de pontos do espaço  $C = \{c_1 \leq c_2 \leq \dots \leq c_{n-1} \leq c_n\}$  pode ser definida como o elemento do meio  $c_{(n+1)/2}$  se  $n$  é ímpar, e pela média dos dois elementos intermediários, se  $n$  é par. A mediana é uma medida de localização estatística que divide o conjunto de dados  $C$  em duas partes com igual número de elementos. Observe que, ao contrário da média, a mediana não é influenciada pela magnitude dos elementos do conjunto. Um fato importante a ser mencionado é que no cálculo da mediana devemos levar em consideração a frequência de cada elemento  $c_i$  do conjunto  $C$ . Desse modo, a construção do histograma de frequência associado ao conjunto de dados é uma etapa importante do cálculo da mediana.

O processo de quantização por equalização de histograma para imagens monocromáticas, consiste em se fazer subdivisões sucessivas do intervalo de intensidades da imagem utilizando a mediana do conjunto de intensidades em cada processo de subdivisão.

A extensão do algoritmo de quantização por equalização de histograma descrito acima, para imagens a cores, é conhecido na literatura pelo nome de **algoritmo do corte mediano**. Esse é um dos algoritmos de quantização mais populares dentro da comunidade de computação gráfica. Isso se dá devido à facilidade de implementação, aliada à sua eficiência computacional, juntamente com os bons resultados perceptuais conseguidos na quantização de imagens de 24 para 8 bits. De forma simples, o método consiste em utilizar o algoritmo de quantização por equalização de histograma em cada uma das componentes de cor do gamute da imagem.

- Algoritmo do Corte Mediano:

Indicamos por  $K$  o número de níveis de quantização desejado. Tomamos um paralelepípedo

$$V = ([r_0, r_1] \times [g_0, g_1] \times [b_0, b_1]),$$

de volume mínimo que contém todas as cores do gamute da imagem a ser quantizada. Em seguida, tomamos a componente do espaço de cor em cuja direção o paralelepípedo  $V$  possui a aresta de maior comprimento. Vamos supor que essa é a componente verde  $g$ . Fazemos então uma ordenação das cores no gamute da imagem pela componente  $g$ , e calculamos a mediana  $m_g$  do conjunto de cores com base nessa ordenação. Dividimos assim a região  $V$  em duas sub-regiões

$$\begin{aligned} V_1 &= \{(r, g, b) \in C; g \leq m_g\} \\ V_2 &= \{(r, g, b) \in C; g \geq m_g\} \end{aligned}$$

Aplicamos então o mesmo método de subdivisão a cada uma das regiões  $V_1$  e  $V_2$ . Continuamos o processo de subdivisão, recursivamente, até que uma das

duas condições seguintes sejam satisfeitas: as duas sub-regiões  $V_1$  e  $V_2$  obtidas não contêm cores do gamute da imagem, ou o número desejado,  $K$ , de células de quantização já foi obtido.

Após a subdivisão do espaço de cor no número de células desejado, determinamos o nível de quantização de cada célula. Para se obter o valor de quantização de um pixel da imagem, devemos localizar a célula que contém a cor desse pixel, e fazer a quantização para o nível correspondente a essa célula. A implementação do algoritmo pode ser feita de modo eficiente utilizando-se uma estrutura de dados espaciais adequada no processo de subdivisão recursiva do espaço de cor.

Vejamos um exemplo para o caso de um espaço de cor bidimensional.

- **Exemplo 2.3:**

Considere um conjunto bidimensional de 9 cores distintas, mostrado na Figura 2.25(a), cujas frequências são dadas pela Tabela 2.2 à esquerda.

TABELA 2.2 – Cores e suas frequências obtidas na análise da Figura 2.25(a)

Cor	Frequência	Cor	Frequência
$c_1$	2	$c_1$	2
$c_2$	3	$c_9$	2
$c_3$	2	$c_8$	1
$c_4$	1	$c_2$	3
$c_5$	2	$c_3$	2
$c_6$	1	$c_4$	1
$c_7$	1	$c_7$	1
$c_8$	1	$c_5$	2
$c_9$	2	$c_6$	1

A direção mais longa do retângulo de cores é a vertical. Ordenando as cores pela ordenada  $y$ , obtemos a Tabela 2.2 à direita.

É imediato verificar que a mediana do conjunto está na posição 7,5 e, portanto, é dada pela média das cores nas posições 7 e 8. Como essas posições são ocupadas pela cor  $c_2$ , concluímos que a mediana é  $c_2$ . A Figura 2.25(b) mostra a subdivisão do conjunto em sua mediana  $c_2$ .

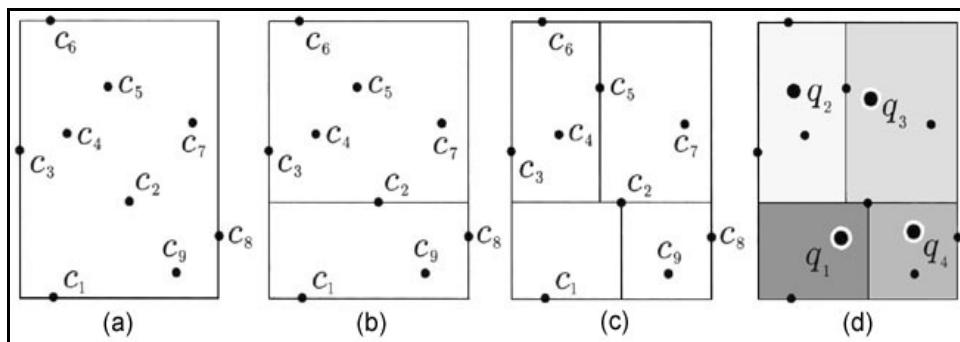


Figura 2.25 – Subdivisão recursiva pelo corte mediano (a), (b), (c); níveis de quantização (d)

Temos portanto dois retângulos de cores resultantes da subdivisão. A aresta mais longa de cada um desses retângulos é a horizontal. Ordenando as cores pela componente horizontal  $x$ , obtemos a Tabela 2.3.

TABELA 2.3 – Cores e frequências da Figura 2.25 (b) ordenadas pela horizontal x

Cor	Frequência
$c_1$	2
$c_2$	3
$c_9$	2
$c_8$	1
$c_3$	2
$c_6$	1
$c_4$	1
$c_5$	2
$c_2$	3
$c_7$	1

No primeiro caso, a mediana é a cor  $c_2$ , e no segundo caso, a mediana é a cor  $c_5$ . As subdivisões correspondentes no retângulo de cor aparecem na Figura 2.25(c).

Após determinadas as quatro células de quantização, o nível de quantização em cada célula é calculado tomando a média das cores do gamute em cada célula. As cores que estão no bordo da célula devem ser quantizadas para o nível de quantização mais próximo. A Figura 2.25(d) mostra os quatro níveis de quantização associados a cada célula de quantização. A função de quantização  $q$  é dada pela Tabela 2.4.

TABELA 2.4 – Função de quantização  $q$ 

c (cor)	q(c)
$c_1, c_2$	$q_1$
$c_3, c_4, c_6$	$q_2$
$c_5, c_7$	$q_3$
$c_8, c_9$	$q_4$

Podemos mudar no algoritmo do corte mediano o cálculo do nível de quantização e o cálculo do ponto de subdivisão utilizado. O uso da mediana no processo de subdivisão das regiões de cor corresponde a se fazer uma equalização do histograma da imagem. Portanto cada célula está associada, aproximadamente, ao mesmo número de pixels da imagem.

A Figura 2.26(a) mostra uma reprodução da imagem de Figura 2.21 quantizada com 8 bits pelo algoritmo do corte mediano. A Figura 2.26(b) é uma reprodução de uma quantização em 4 bits pelo algoritmo do corte mediano.

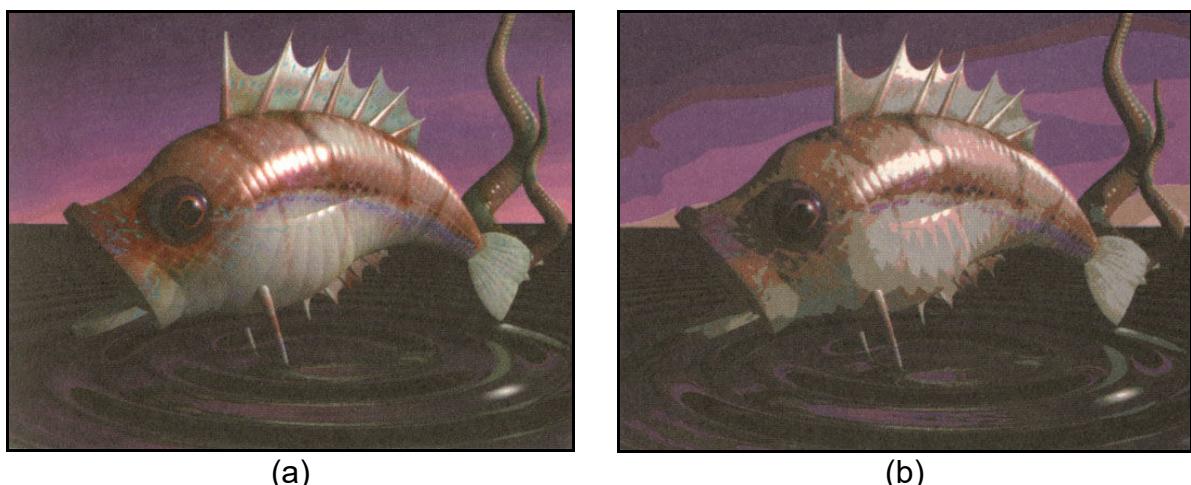


Figura 2.26 – Quantização pelo algoritmo do corte mediano: (a) 8 bits; (b) 4 bits

A quantização dessa imagem com o número correspondente de bits, utilizando, respectivamente, quantização uniforme e o algoritmo de populosidade, pode ser vista nas Figura 2.22(a), (b), e 2.23(a) e (b), respectivamente.

Diversas mudanças podem ser efetuadas no algoritmo de corte mediano, dando origem a diversas variações do algoritmo. Uma mudança simples consiste em se escolher um particionamento de modo a minimizar a variância entre as cores de cada sub-região e o nível de quantização associado. Em geral, para a grande maioria das imagens, as diferentes mudanças que podem ser feitas no algoritmo são, em geral, imperceptíveis quando fazemos quantização de 24 para 8 bits. Essas mudanças devem ser analisadas, em grande parte, do ponto de vista da eficiência computacional.

## 2.7 TÉCNICAS DE MODIFICAÇÃO DE HISTOGRAMAS

As técnicas de modificação de histograma são conhecidas como técnicas ponto-a-ponto, uma vez que o valor do tom de cinza de um certo pixel, após o processamento, depende apenas de seu valor original. Em contraste, nas técnicas de processamento orientadas a vizinhança, o valor resultante depende também, de alguma forma, dos pixels que circundam o elemento na imagem original.

### 2.7.1 Transformações de Intensidade

Diversas técnicas de modificação da distribuição dos pixels na escala de cinza podem ser implementadas a partir do conceito de transformações de intensidade, ou seja,

$$0 \leq f \leq 1$$

onde  $f = 0$  representa um pixel preto e  $f = 1$  indica pixel branco.

Para qualquer  $f$  no intervalo  $[0, 1]$ , denominaremos transformações de intensidade as funções do tipo:

$$g = T(f)$$

que mapearão cada pixel de tom de cinza  $f$  da imagem original em um novo tom de cinza  $g$ , na imagem destino. Estas funções devem satisfazer duas condições:

- i) devem retornar um único valor para cada valor distinto de  $f$  e devem crescer monotonicamente no intervalo  $0 \leq f \leq 1$ ;
- ii)  $0 \leq T(f) \leq 1$  para  $0 \leq f \leq 1$ .

Um exemplo de função que satisfaz estes critérios é dado na Figura 2.27. O efeito desta transformação não-linear de intensidade sobre a imagem é um aumento de seu contraste.

As transformações de intensidade podem ser lineares ou não-lineares. As transformações lineares podem ser genericamente descritas pela equação:

$$g = c \cdot f + b$$

onde o parâmetro  $c$  controla o contraste da imagem resultante, enquanto  $b$  ajusta seu brilho. A Figura 2.28 apresenta diversos exemplos de transformações lineares e seus respectivos valores de  $c$  e  $b$ .

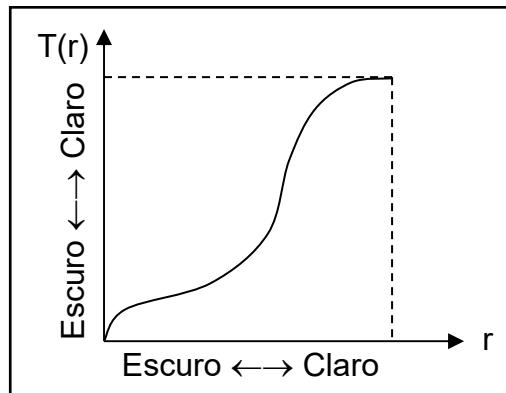


Figura 2.27 – Exemplo de transformação de intensidade

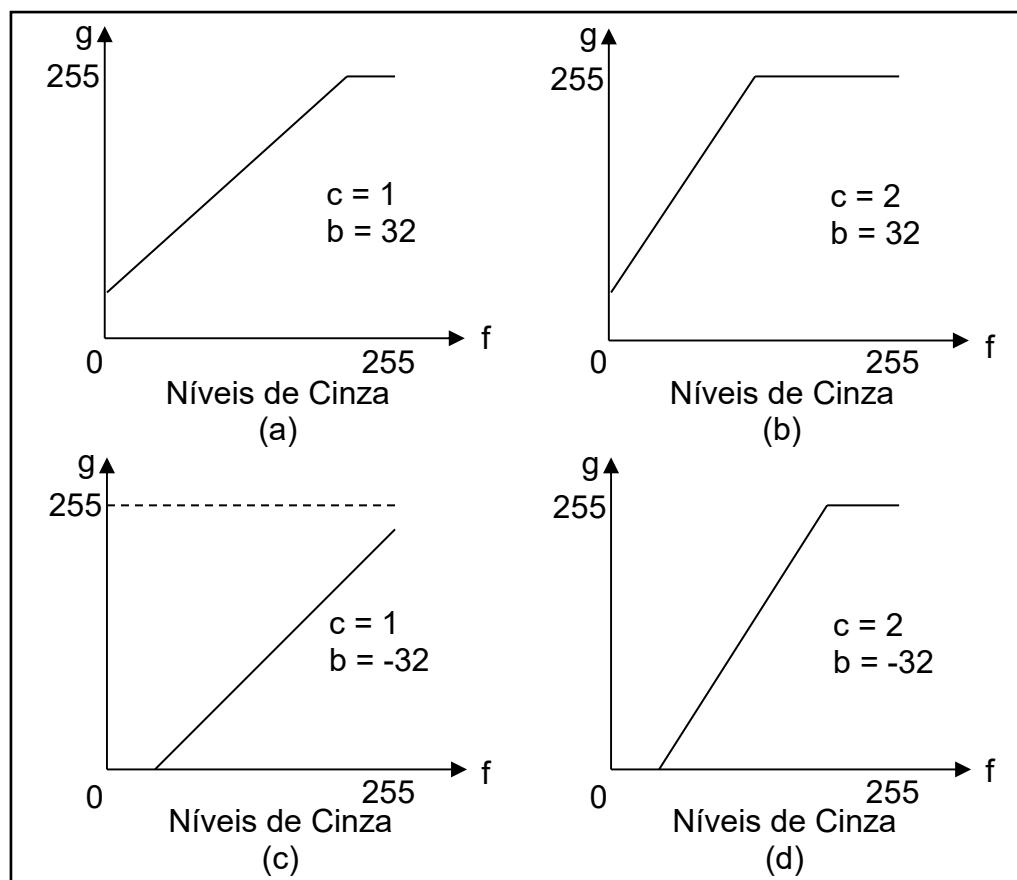


Figura 2.28 – Exemplos de transformações de intensidade lineares

As transformações não-lineares podem ser descritas por equações tais como:

$$g = 31,875 \cdot \log_2(f + 1)$$

produzindo o resultado mostrado na Figura 2.29. Nos aplicativos para processamento de imagens disponíveis atualmente, freqüentemente estas transformações são especificadas de forma interativa pelo usuário, utilizando o mouse ou dispositivo equivalente e “desenhando” a curva desejada.

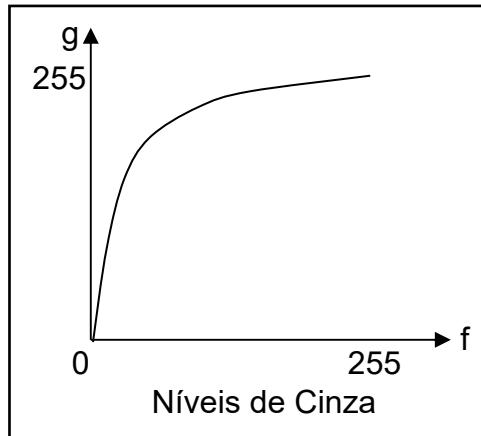


Figura 2.29 – Exemplo de transformação de intensidade não-linear

O conceito de transformação de intensidade linear pode ser utilizado para implementar uma função que automaticamente expande a escala de tons de cinza de uma imagem para que ela ocupe todo o intervalo possível. Esta função recebe o nome de auto-escala. Para um sistema que opera com imagens com 256 níveis de cinza, uma função de auto-escala pode ser implementada calculando, para cada pixel com tom de cinza  $f$ , o nível de cinza resultante  $g$ , pela equação:

$$g = \frac{255}{f_{\max} - f_{\min}}(f - f_{\min})$$

onde  $f_{\max}$  e  $f_{\min}$  são, respectivamente, os níveis máximo e mínimo de cinza presentes na imagem original.

### 2.7.2 Uniformização das Médias e Variâncias de imagens

Para se realizar transformações sobre pares de imagens, capturadas em tempos ou sob condições ambientais diferenciadas, torna-se necessário o ajuste do brilho e contraste destas imagens. Uma das formas de se fazer este ajuste é o processo de uniformização das médias e variâncias de imagens.

Considere duas imagens, a imagem de referência ( $g_r$ ) e a imagem de ajuste ( $g_a$ ). Calcula-se, então, suas médias ( $m_r$  e  $m_a$ ) e variâncias ( $V_r$  e  $V_a$ ).

Dois fatores devem ser calculados no processo: o fator de ganho e o fator de deslocamento (*offset*).

$$\begin{aligned} ganho &= \sqrt{\frac{V_r}{V_a}} \\ offset &= m_r - \sqrt{\frac{V_r}{V_a}} \cdot m_a \end{aligned}$$

A correção da imagem de ajuste se dá através da transformação linear aplicada a todos seus pixels.

$$g = ganho \cdot f_a + offset$$

O fator de ganho corresponde ao coeficiente angular da reta e modifica a variância da imagem (contraste) enquanto que o *offset* corresponde a um deslocamento do histograma no eixo x, modificando a média da imagem (brilho).

### 2.7.3 Equalização de Histograma

A equalização de histograma é uma técnica a partir da qual se procura redistribuir os valores de tons de cinza dos pixels em uma imagem, de modo a obter um histograma uniforme, no qual o percentual de pixels de qualquer nível de cinza é praticamente o mesmo. Para tanto, utiliza-se uma função auxiliar, denominada função de distribuição acumulada (**cdf** – *cumulative distribution function*), que pode ser expressa por:

$$s_k = T(r_k) = \sum_{j=0}^k \frac{n_j}{n} = \sum_{j=0}^k p_r(r_j)$$

onde  $0 \leq r_k \leq 1$  e  $k = 0, 1, \dots, L - 1$ ;

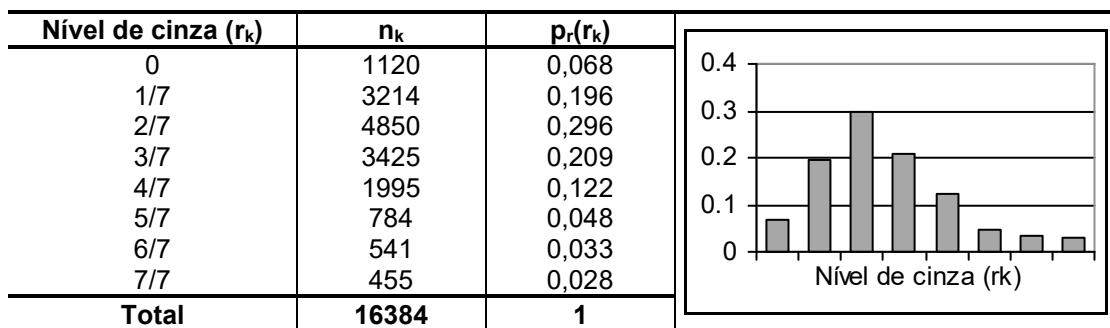
A inversa desta função é dada por:

$$r_k = T^{-1}(s_k) \quad \text{para } 0 \leq s_k \leq 1$$

e embora ela não seja necessária no processo de equalização de histograma, será importante no método descrito na seção seguinte. Convém dizer que outras funções de transformação, que não a **cdf**, podem ser especificadas.

- **Exemplo 2.4:**

Seja o histograma da Tabela 2.1, ilustrado graficamente na Figura 2.12, ambas reproduzidas a seguir para maior facilidade. Equalizá-lo utilizando a função de distribuição acumulada e plotar o histograma resultante.



Utilizando a **cdf** como função de transformação, calculamos:

$$\begin{aligned} s_0 &= T(r_0) = \sum_{j=0}^0 p_r(r_j) \\ &= p_r(r_0) = 0,068 \end{aligned}$$

De forma similar,

$$\begin{aligned}s_1 &= T(r_1) = \sum_{j=0}^1 p_r(r_j) \\&= p_r(r_0) + p_r(r_1) \\&= 0,068 + 0,196 = 0,264\end{aligned}$$

e

$$\begin{array}{lll}s_2 = 0,560 & s_3 = 0,769 & s_4 = 0,891 \\s_5 = 0,939 & s_6 = 0,972 & s_7 = 1\end{array}$$

Esta função está plotada na Figura 2.30.

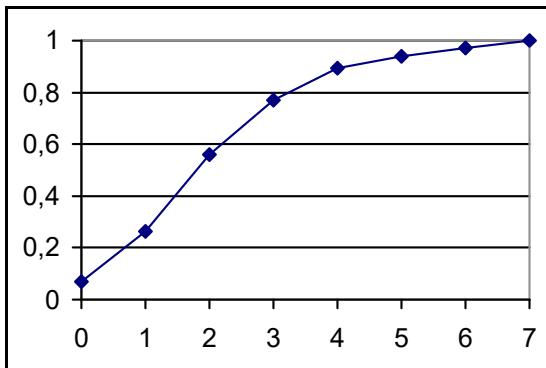


Figura 2.30 – Função de transformação utilizada para a equalização

Como a imagem foi quantizada com apenas 8 níveis de cinza, cada valor deverá ser arredondado para o valor válido (múltiplo de 1/7) mais próximo. Dessa forma:

$$\begin{array}{llll}s_0 = 0 & s_1 = \frac{2}{7} & s_2 = \frac{4}{7} & s_3 = \frac{5}{7} \\s_4 = \frac{6}{7} & s_5 = 1 & s_6 = 1 & s_7 = 1\end{array}$$

Concluindo o mapeamento, verificamos que o nível original  $r_0 = 0$  foi mapeado para  $s_0 = 0$  e, portanto, a raia correspondente não sofreu alteração. Já os 3214 pixels que apresentavam tom de cinza 1/7 foram remapeados para  $s_1 = 2/7$ . Similarmente, os pixels com tom de cinza 2/7 foram modificados para 4/7, aqueles com  $r = 3/7$  passaram a 5/7 e os de 4/7 foram mapeados para 6/7. Convém observar, entretanto, que as três raias correspondentes aos pixels com tons de cinza 5/7, 6/7 e 1 foram somadas em uma só raia, com tom de cinza máximo, isto é, 1.

Agrupando os resultados na Tabela 2.5, teremos o histograma após a equalização, mostrado graficamente na Figura 2.31.

Pode-se notar que o histograma equalizado, apesar de estar longe de ser perfeitamente plano, apresenta melhor distribuição de pixels ao longo da escala de cinza em relação ao original.

A Figura 2.32 apresenta um exemplo de aplicação da técnica de equalização de histograma para aumentar o contraste em uma imagem com 256 tons de cinza. A parte (a) apresenta a imagem original, cujo histograma é plotado na Figura 2.32 (b). A parte (d) mostra o histograma equalizado, correspondente à imagem da Figura 2.32(c).

TABELA 2.5 – Histograma equalizado

Nível de cinza ( $s_k$ )	$n_k$	$p_s(s_k)$
0	1120	0,068
1/7	0	0,000
2/7	3214	0,196
3/7	0	0,000
4/7	4850	0,296
5/7	3425	0,209
6/7	1995	0,122
7/7	1780	0,109
<b>Total</b>	<b>16384</b>	<b>1</b>

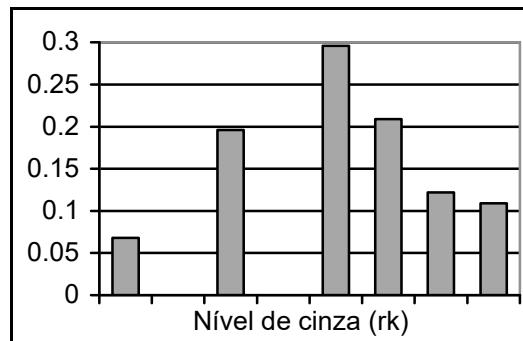
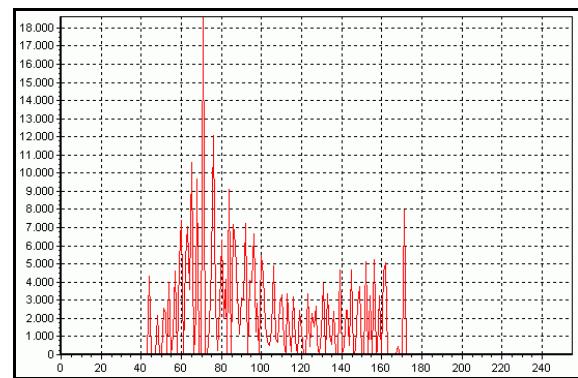


Figura 2.31 – Histograma equalizado



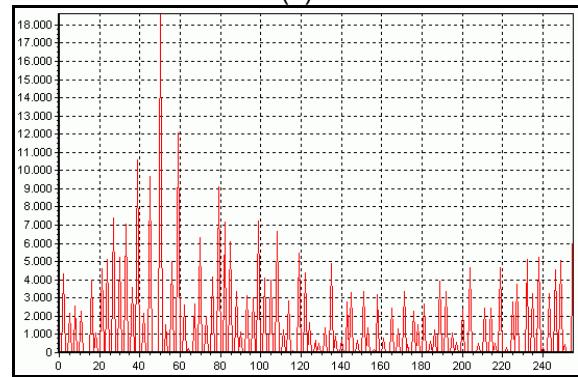
(a)



(b)



(c)



(d)

Figura 2.32 – Aplicação da equalização de histograma a imagens com baixo contraste

As técnicas de obtenção e equalização de histogramas também podem ser aplicadas a trechos de imagens, por exemplo, janelas  $n \times m$ . Estas técnicas locais servem principalmente para realçar detalhes sutis de pequenas porções da imagem.

#### 2.7.4 Especificação Direta de Histograma

Apesar de sua grande utilização em situações de aprimoramento de contraste de imagens, a equalização de histograma apresenta como principal limitação o fato de não permitir a especificação de nenhum parâmetro, a não ser a função de transformação, que, como vimos na seção anterior, costuma ser a **cdf** da distribuição de probabilidade original. Existem situações, entretanto, em que seria desejável poder especificar que tipo de mudança se deseja sobre o histograma. Nestes casos, uma das possíveis técnicas é a especificação direta do histograma.

Dada uma imagem e, consequentemente seu histograma, e um novo histograma desejado, o procedimento de especificação direta de histograma consiste em:

- 1) equalizar os níveis da imagem original usando a **cdf** discreta:

$$s_k = T(r_k) = \sum_{j=0}^k \frac{n_j}{n} = \sum_{j=0}^k p_r(r_j)$$

- 2) obter a função de distribuição acumulada (**cdf**) do histograma desejado:

$$v_k = G(z_k) = \sum_{j=0}^k p_z(z_j)$$

- 3) aplicar a função de transformação inversa  $z = G^{-1}(s)$  aos níveis obtidos no passo 1.

- **Exemplo 2.5:**

Seja novamente histograma da Tabela 2.4. Deseja-se modificar este histograma de modo que a distribuição de pixels resultante seja aquela da Tabela 2.6.

TABELA 2.6 – Histograma desejado

Nível de cinza ( $z_k$ )	$n_k$	$p_z(z_k)$
0	0	0,000
1/7	0	0,000
2/7	0	0,000
3/7	1638	0,100
4/7	3277	0,200
5/7	6554	0,400
6/7	3277	0,200
7/7	1638	0,100
<b>Total</b>	<b>16384</b>	<b>1</b>

O histograma após equalização já foi calculado no exemplo anterior e seus resultados estão na Tabela 2.5.

O próximo passo consiste em obter a **cdf** da distribuição de probabilidade desejada. Seguindo o mesmo raciocínio utilizado para o cálculo da **cdf** do histograma original, encontramos:

$$\begin{array}{llll} v_0 = 0 & v_1 = 0 & v_2 = 0 & v_3 = 0,1 \\ v_4 = 0,3 & v_5 = 0,7 & v_6 = 0,9 & v_7 = 1 \end{array}$$

O último passo – e o mais difícil de entender quando se estuda este assunto pela primeira vez – é a obtenção da inversa. Como estamos lidando com níveis discretos, a obtenção da função inversa consistirá basicamente em procurar, para cada valor de  $s_k$ , o valor de  $v_k$  que mais se aproxima dele. Por exemplo, o valor de  $v_k$  que mais se aproxima de  $s_1 = 2/7 \approx 0,286$  é  $G(z_4) = 0,3$ , ou seja  $G^{-1}(0,3) = z_4$ . Portanto, os pixels que, após a equalização do histograma original, foram repositionados no tom de cinza  $s_1$ , serão mapeados para o tom de cinza  $z_4$ . Em outras palavras, os 3214 pixels que apresentavam originalmente tom de cinza  $1/7$  e que foram remapeados para  $s_1 = 2/7$  devido à equalização, serão transladados novamente para  $z_4 = 4/7$  por força da especificação direta de histograma. Procedendo de forma similar para os demais valores de  $s_k$ , teremos:

$$\begin{aligned} s_0 = 0 &\rightarrow z_2 & s_1 = \frac{2}{7} \approx 0,286 &\rightarrow z_4 \\ s_2 = \frac{4}{7} \approx 0,571 &\rightarrow z_5 & s_3 = \frac{5}{7} \approx 0,714 &\rightarrow z_5 \\ s_4 = \frac{6}{7} \approx 0,857 &\rightarrow z_6 & s_5 = 1 &\rightarrow z_7 \\ s_6 = 1 &\rightarrow z_7 & & s_7 = 1 \rightarrow z_7 \end{aligned}$$

Neste caso, assumimos que o algoritmo de cálculo da inversa, para um valor de  $s_k$ , percorreria os diversos valores de  $v_k$ , armazenando o índice do último valor que tenha resultado na menor diferença encontrada. Se o algoritmo possuir outra forma de solucionar “empates”, o nível de  $s_0$  poderá mapear em  $z_0$  ou  $z_1$ . A Tabela 2.7 resume os histogramas original e desejado, suas respectivas **cdfs** e o processo de mapeamento descrito anteriormente.

TABELA 2.7 – Resumo da especificação direta de histograma

<b>k</b>	<b>p<sub>r</sub>(r<sub>k</sub>)</b>	<b>s<sub>k</sub></b>	<b>v<sub>k</sub></b>	<b>p<sub>z</sub>(z<sub>k</sub>)</b>
0	0,068	0	0,00	0,000
1	0,196	2/7	0,00	0,000
2	0,296	4/7	0,00	0,000
3	0,209	5/7	0,10	0,100
4	0,122	6/7	0,30	0,200
5	0,048	1	0,70	0,400
6	0,033	1	0,90	0,200
7	0,028	1	1,00	0,100
<b>Total</b>	<b>16384</b>			<b>1</b>

A Tabela 2.8 apresenta os valores obtidos para o histograma resultante. Para uma comparação visual entre o histograma desejado e o obtido, plotamos cada um deles nas Figuras 2.33 e 2.34, respectivamente.

TABELA 2.8 – Histograma obtido

<b>Nível de cinza (z<sub>k</sub>)</b>	<b>p<sub>z</sub>(z<sub>k</sub>)</b>
0	0,000
1/7	0,000
2/7	0,068
3/7	0,000
4/7	0,196
5/7	0,505
6/7	0,122
1	0,109
<b>Total</b>	<b>1</b>

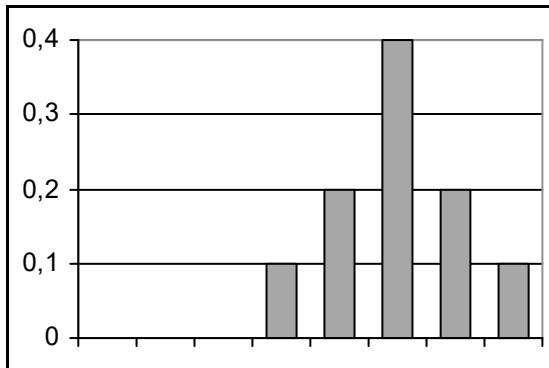


Figura 2.33 – Histograma desejado

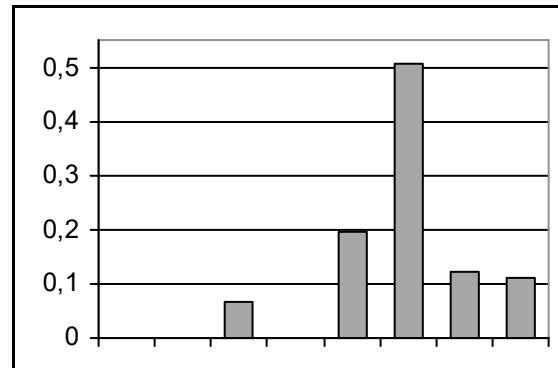


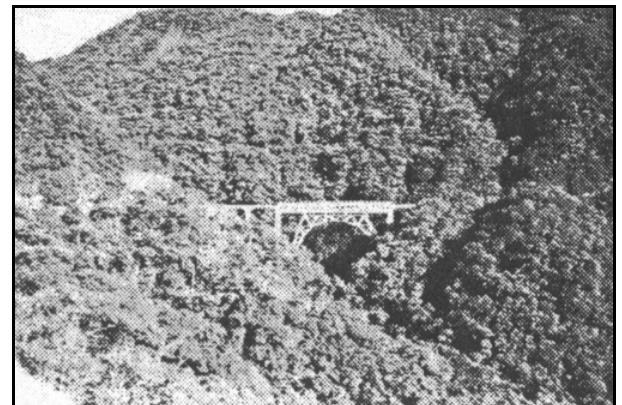
Figura 2.34 – Histograma obtido

Pode-se notar que o histograma obtido aproxima-se, dentro do possível, do histograma desejado.

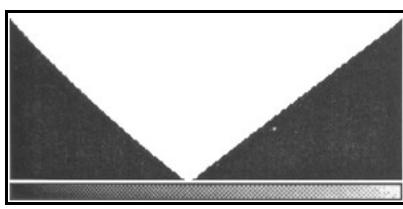
A Figura 2.35 apresenta um exemplo de aplicação da técnica de especificação direta de histograma aplicada a uma imagem com 256 tons de cinza. A parte (a) apresenta a imagem original, cujo histograma é plotado na Figura 2.35(c). A parte (d) mostra o histograma desejado, enquanto a Figura 2.35(e) mostra o histograma obtido, que corresponde à imagem da Figura 2.35(b).



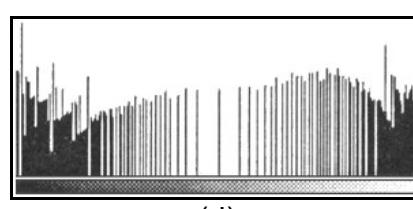
(a)



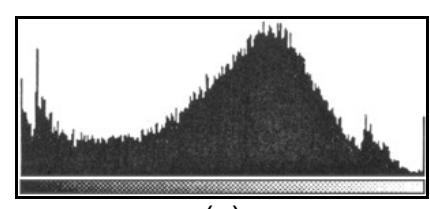
(b)



(c)



(d)



(e)

Figura 2.35 – Exemplo de aplicação da especificação direta de histograma

## 2.7.5 Limiarização (Thresholding)

O princípio da limiarização consiste em separar as regiões de uma imagem quando esta apresenta duas classes, o fundo e o objeto. Devido ao fato da limiarização produzir uma imagem binária à saída, o processo também é denominado binarização.

A forma mais simples de limiarização consiste na bipartição do histograma, convertendo os pixels cujo tom de cinza é maior ou igual a um certo valor de limiar ( $T$ ) em brancos e os demais em pretos, como ilustra a Figura 2.36.

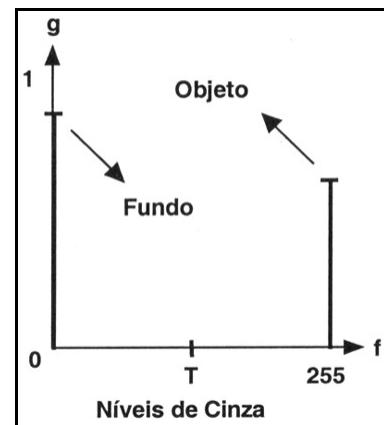
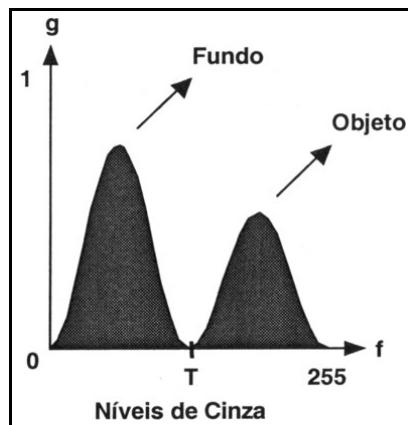
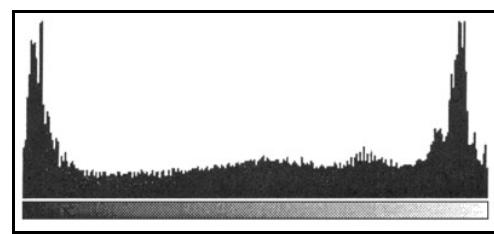


Figura 2.36 – Limiarização de uma imagem monocromática utilizando limiar  $T$ : (a) histograma original, (b) histograma da imagem binarizada

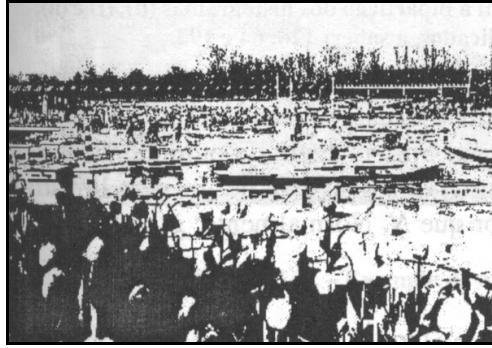
No caso de níveis de cinza divididos basicamente em duas classes, onde o histograma apresenta dois picos e um vale, a limiarização é trivial. Ainda assim, os efeitos decorrentes da escolha de um valor específico de limiar dentre os diversos pontos situados na região de vale podem ser analisados na Figura 2.37.



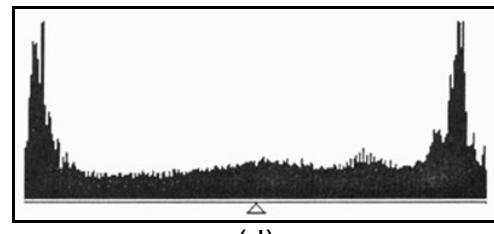
(a)



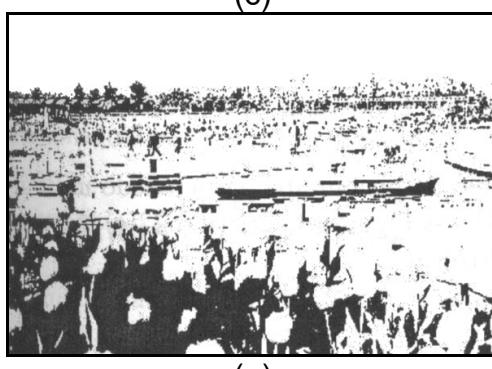
(b)



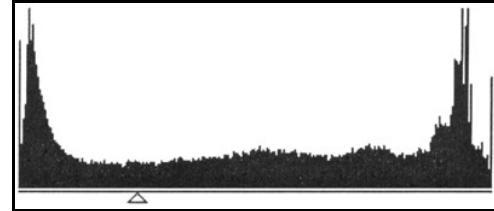
(c)



(d)



(e)



(f)

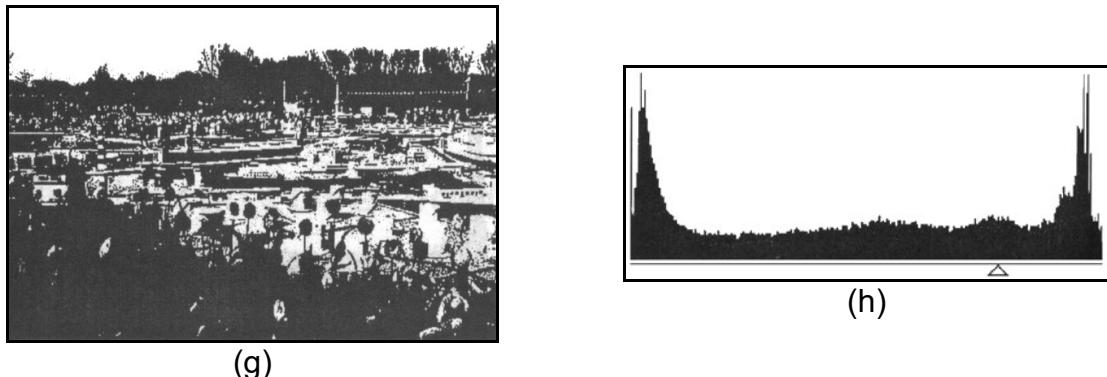


Figura 2.37 – Efeitos da escolha do limiar na binarização de uma imagem grayscale. As imagens (c), (e) e (g) correspondem à bipartição dos histogramas (d), (f) e (h), respectivamente, nos limiares indicados, a saber: 128, 64 e 192.

Matematicamente, a operação de limiarização pode ser descrita como uma técnica de processamento de imagens na qual uma imagem de entrada  $f(x, y)$  de  $N$  níveis de cinza produz à saída uma imagem  $g(x, y)$ , chamada de imagem limiarizada, cujo número de níveis de cinza é menor que  $N$ . Normalmente,  $g(x, y)$  apresenta 2 níveis de cinza, sendo:

$$\begin{aligned} g(x, y) &= 1 \text{ se } f(x, y) \geq T \\ &= 0 \text{ se } f(x, y) < T \end{aligned}$$

onde os pixels rotulados com **1** correspondem aos objetos e os pixels etiquetados com **0** correspondem ao fundo (background) e  $T$  é um valor de tom de cinza predefinido, ao qual denominamos limiar.

A Figura 2.38(a) mostra um exemplo de histograma particionado utilizando dois valores de limiar:  $T_1 = 37$  e  $T_2 = 233$ . As Figuras 2.38(b) e 2.38(c) mostram a imagem original e a imagem após a limiarização.

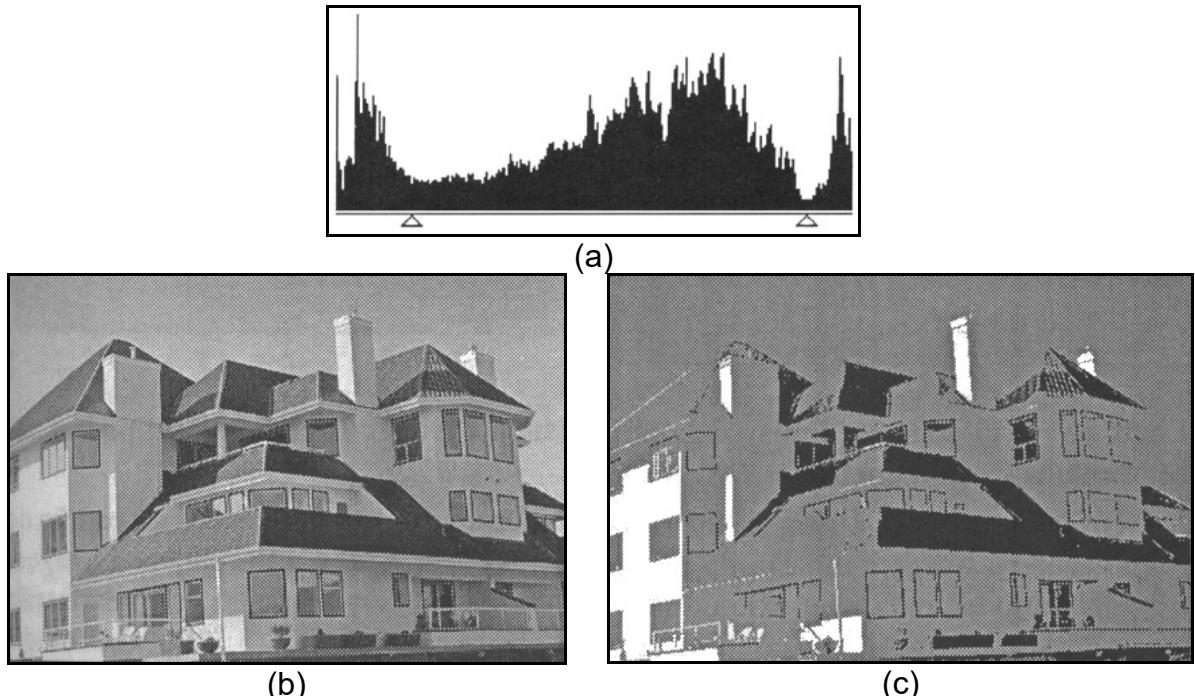


Figura 2.38 – Exemplo de utilização de múltiplos limiares

A limiarização pode ser vista como uma operação que envolve um teste com relação a uma função  $T$  do tipo  $T = T[x, y, p(x, y), f(x, y)]$ , onde  $f(x, y)$  é o tom de cinza original no ponto  $(x, y)$  e  $p(x, y)$  indica alguma propriedade local deste ponto, por exemplo a média de seus vizinhos. Quando  $T$  depende apenas de  $f(x, y)$ , o limiar é chamado global; quando  $T$  depende de  $f(x, y)$  e de  $p(x, y)$ , o limiar é chamado local. Se, além disso,  $T$  depende das coordenadas espaciais de  $(x, y)$ , o limiar é chamado dinâmico ou adaptativo.

- Influência da Iluminação

A iluminação desempenha um papel significativo no processo de limiarização, uma vez que provoca alterações no histograma original da imagem, eventualmente eliminando uma região de vale entre dois picos, naturalmente propícia para a definição de um limiar global.

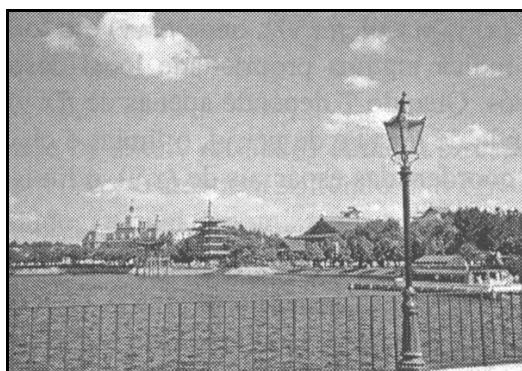
Pode-se provar que, sendo  $f(x, y) = i(x, y) \cdot r(x, y)$  e sendo  $z(x, y) = \ln f(x, y) = \ln i(x, y) + \ln r(x, y) = i'(x, y) + r'(x, y)$ , onde  $i'(x, y)$  e  $r'(x, y)$  são variáveis aleatórias independentes, o histograma de  $z(x, y)$  é dado pela convolução do histograma de  $i'(x, y)$  com o de  $r'(x, y)$ .

Uma técnica comum utilizada para compensar a não uniformidade da iluminação consiste em projetar o padrão de iluminação em uma superfície refletora branca. Isto nos dá uma imagem  $g(x, y) = K \cdot i(x, y)$ , onde  $K$  depende da superfície utilizada. Deste modo, para qualquer imagem  $f(x, y) = i(x, y) \cdot r(x, y)$  obtida com a mesma função iluminação, simplesmente divide-se  $f(x, y)$  por  $g(x, y)$ , obtendo-se uma função normalizada:

$$h(x, y) = \frac{f(x, y)}{g(x, y)} = \frac{r(x, y)}{K}$$

Logo, se  $r(x, y)$  pode ser limiarizada utilizando o limiar  $T$ , então  $h(x, y)$  poderá ser segmentada usando um limiar  $T/K$ .

A Figura 2.39 ilustra as alterações causadas por modificações no padrão de iluminação na imagem binarizada resultante. Na parte (a) é apresentada a imagem original, cujo histograma é exibido na Figura 2.41(e). O resultado da limiarização desta imagem com limiar  $T = 128$  é mostrada na Figura 2.41(c). Na coluna da direita são mostradas a imagem com padrão de iluminação alterado (b), seu histograma (f) e o resultado da limiarização com o mesmo limiar utilizado anteriormente (d).



(a)



(b)

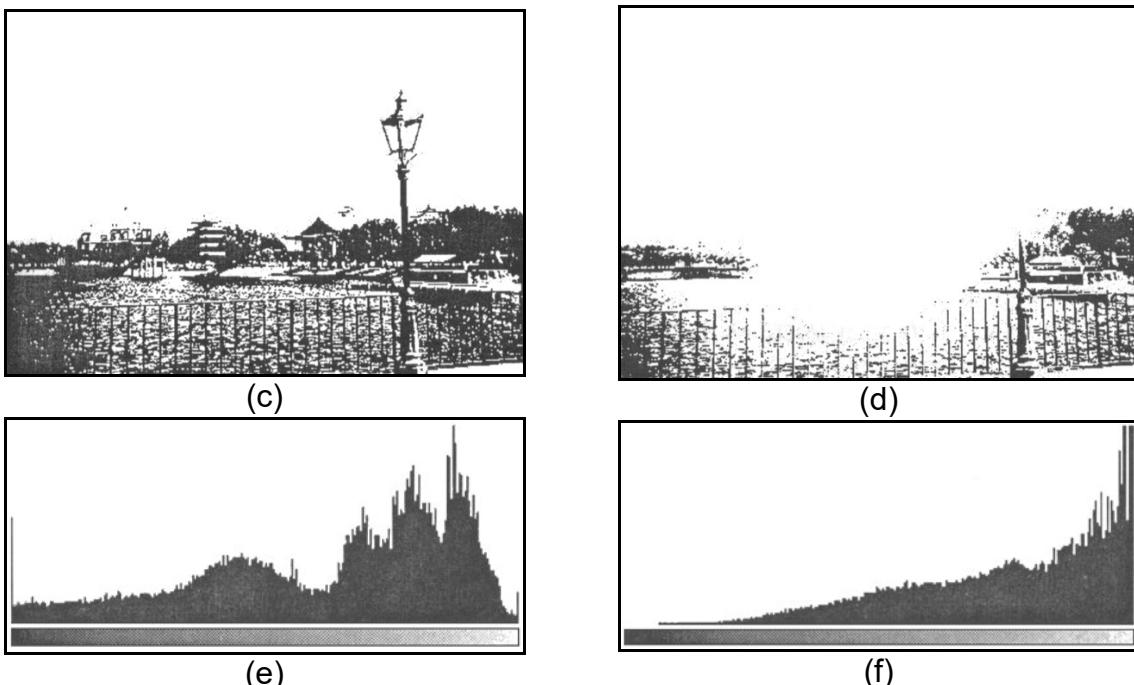


Figura 2.39 – Influência da iluminação no processo de limiarização

- Limiarização pelas Propriedades Estatísticas da Imagem

Pelo exposto até aqui, assumiu-se que a escolha do valor de limiar é arbitrária e subjetiva. Sabendo que o histograma é uma representação gráfica da distribuição de probabilidade de ocorrência dos níveis de cinza em uma imagem, é lícito imaginar a possibilidade de uso de técnicas de cálculo do valor ótimo de limiar com base nas propriedades estatísticas da imagem.

Uma destas técnicas, denominada limiarização ótima, parte de uma imagem da qual se conhecem as principais propriedades estatísticas, a saber:

$\mu_1$  = média dos tons de cinza da região de interesse;

$\mu_2$  = média dos tons de cinza da região de fundo (background);

$\sigma_1, \sigma_2$  = desvios padrão;

$P_1, P_2$  = probabilidade de ocorrência dos pixels pertencentes a esta ou aquela região;

Pode-se mostrar que existe um valor ótimo de limiar  $T$ , dado por uma das raízes da equação

$$AT^2 + BT + C = 0$$

onde:

$$A = \sigma_1^2 - \sigma_2^2$$

$$B = 2(\mu_1\sigma_2^2 - \mu_2\sigma_1^2)$$

$$C = \mu_2^2\sigma_1^2 - \mu_1^2\sigma_2^2 + 2\sigma_1^2 - \sigma_2^2 \ln\left(\frac{\sigma_2 P_1}{\sigma_1 P_2}\right)$$

Duas raízes reais e positivas indicam que a imagem pode requerer dois

valores de limiar para obter uma solução ótima.

Se as variâncias forem iguais ( $\sigma^2 = \sigma_1^2 = \sigma_2^2$ ), um único valor  $T$  é necessário:

$$T = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2}{\mu_1 - \mu_2} \ln\left(\frac{P_2}{P_1}\right)$$

Se, além disso, as duas classes forem equiprováveis:

$$T = \frac{\mu_1 + \mu_2}{2}$$

o que está em acordo com o conceito intuitivo de que o valor ótimo de limiar, quando as classes apresentam a mesma distribuição de probabilidade, é o ponto médio entre as médias das classes.

## 2.8 OPERAÇÕES LÓGICAS E ARITMÉTICAS COM IMAGENS

Sabemos que após uma imagem ter sido adquirida e digitalizada, ela pode ser vista como uma matriz de inteiros e portanto pode ser manipulada numericamente, utilizando operações lógicas e/ou aritméticas. Estas operações podem ser efetuadas pixel a pixel ou orientadas à vizinhança. No primeiro caso, elas podem ser descritas pela seguinte notação:

$$X \text{ opn } Y = Z$$

onde  $X$  e  $Y$  podem ser imagens (matrizes) ou escalares,  $Z$  é obrigatoriamente uma matriz e  $\text{opn}$  é um operador aritmético (+, -, \*, /) ou lógico (and, or, xor) binário.

Sejam duas imagens  $X$  e  $Y$  de igual tamanho. Estas imagens podem ser processadas pixel a pixel, utilizando um operador aritmético ou lógico, produzindo uma terceira imagem  $Z$ , cujos pixels correspondem ao resultado de  $X \text{ opn } Y$  para cada elemento de  $X$  e  $Y$ , conforme ilustra esquematicamente a Figura 2.40.

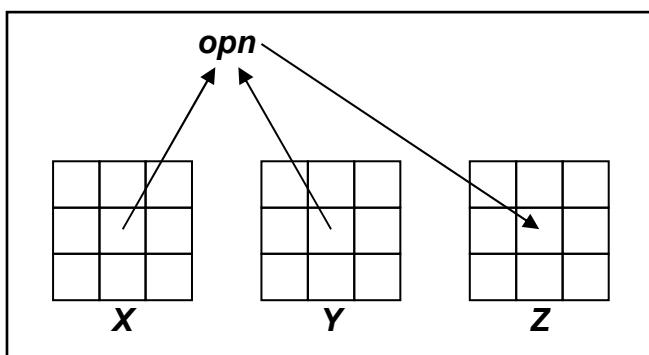


Figura 2.40 – Operações lógicas/aritméticas pixel a pixel.

### 2.8.1 Operações Aritméticas Pixel a Pixel

Ao executarmos operações aritméticas sobre imagens, devemos tomar especial cuidado com os problemas de *underflow* ou *overflow* do resultado. A adição

de duas imagens de 256 tons de cinza, por exemplo, pode resultar em um número maior que 255 para alguns pixels, ao mesmo tempo que a subtração de duas imagens pode resultar em valores negativos para alguns elementos. Para contornar estes problemas, existem basicamente duas alternativas: (1) manter os resultados intermediários em uma matriz, na qual o espaço em memória alocado para cada pixel permita a representação de números negativos e/ou maiores que 255 e, em seguidas, proceder a uma normalização destes valores intermediários; (2) truncar os valores maiores que o máximo valor permitido, bem como os valores negativos, igualando-os a 255 e 0, respectivamente. A decisão depende do objetivo que se tem em mente ao executar determinada operação. Efetivamente, a segunda alternativa é mais simples que a primeira.

- **Exemplo 2.6:**

Dadas as matrizes  $X$  e  $Y$  a seguir, correspondentes a trechos  $3 \times 3$  de imagens de 256 tons de cinza, adicioná-las e informar:

- o resultado intermediário (sem considerações de *underflow* e *overflow*);
- o resultado final utilizando normalização;
- o resultado final utilizando truncamento.

$$X = \begin{bmatrix} 200 & 100 & 100 \\ 0 & 10 & 50 \\ 50 & 250 & 120 \end{bmatrix} \quad Y = \begin{bmatrix} 100 & 220 & 230 \\ 45 & 95 & 120 \\ 205 & 100 & 0 \end{bmatrix}$$

$$\text{a)} X + Y = \begin{bmatrix} 300 & 320 & 330 \\ 45 & 105 & 170 \\ 255 & 350 & 120 \end{bmatrix}$$

b) Fazendo com que a escala  $[45, 350]$  seja adequada ao intervalo  $[0, 255]$ , utilizando-se a relação

$$g = \frac{255}{f_{\max} - f_{\min}}(f - f_{\min})$$

obtém-se:

$$(X + Y)_N = \begin{bmatrix} 213 & 230 & 238 \\ 0 & 50 & 105 \\ 175 & 255 & 63 \end{bmatrix}$$

d) Truncando os valores maiores que 255, obtém-se:

$$(X + Y)_T = \begin{bmatrix} 255 & 255 & 255 \\ 45 & 105 & 170 \\ 255 & 255 & 120 \end{bmatrix}$$

As principais aplicações das operações aritméticas sobre imagens são a adição, a subtração, a multiplicação e a divisão. Assim como  $\mathbf{Y}$  foi, implicitamente, considerado até aqui como sendo uma matriz, ela também pode ser um escalar. As Figuras 2.41 e 2.42 mostram exemplos de cada operação aritmética.

Na adição a imagem resultante  $\mathbf{Z}$ , é o resultado da soma dos valores de intensidade de  $\mathbf{X}$  e  $\mathbf{Y}$ . Se  $\mathbf{Y}$  for um escalar positivo,  $\mathbf{Z}$  será uma versão mais clara de  $\mathbf{X}$ ; o acréscimo de intensidade será o próprio valor de  $\mathbf{Y}$ . A adição pode ser utilizada na normalização do brilho de imagens e na remoção de ruídos.

Na subtração  $\mathbf{Z}$  é o resultado da diferença dos valores de intensidade de  $\mathbf{X}$  e  $\mathbf{Y}$ . Se  $\mathbf{Y}$  for um escalar positivo,  $\mathbf{Z}$  será uma versão mais escura de  $\mathbf{X}$ ; o decréscimo de intensidade será o próprio valor de  $\mathbf{Y}$ . A subtração pode ser utilizada para detectar eventuais diferenças entre duas imagens (eventualmente adquiridas de forma sucessiva) da mesma cena.

Na multiplicação  $\mathbf{Z}$  é o produto dos valores de intensidades de  $\mathbf{X}$  e  $\mathbf{Y}$ . Se  $\mathbf{Y}$  for um escalar positivo, os valores de intensidade de  $\mathbf{Z}$  serão diretamente proporcionais a  $\mathbf{X}$  por um fator  $\mathbf{Y}$ . A multiplicação pode ser utilizada na calibração de brilho. A calibração é um processo semelhante à normalização de brilho, mas que pode estar relacionado à adequação a diferentes valores de luminância sobre uma mesma cena, por exemplo.

Na divisão  $\mathbf{Z}$  é a razão dos valores de intensidade de  $\mathbf{X}$  pelos valores correspondentes em  $\mathbf{Y}$ . Se  $\mathbf{Y}$  for um escalar positivo, os valores de intensidade de  $\mathbf{Z}$  serão inversamente proporcionais a  $\mathbf{X}$  por um fator  $\mathbf{Y}$ . Também é utilizada na normalização de brilho.

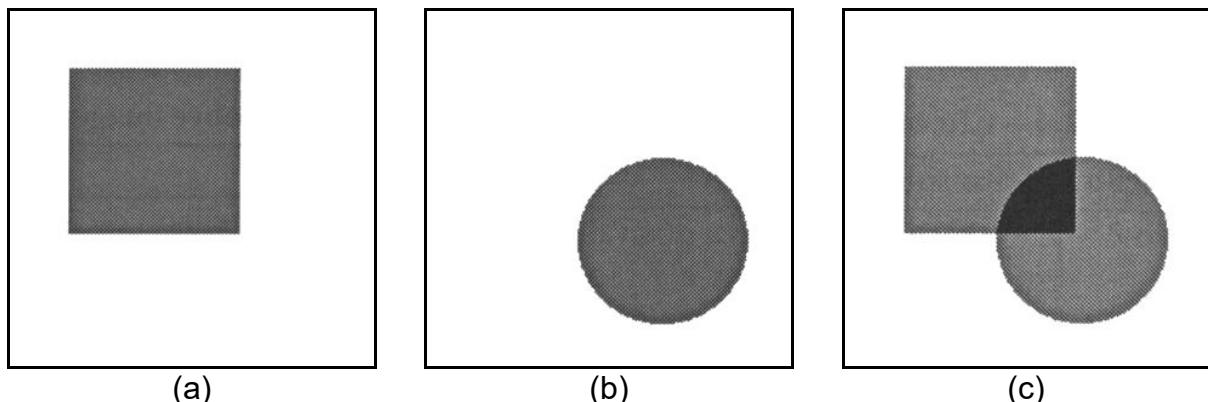


Figura 2.41 – Exemplo de adição de imagens monocromáticas  
(a)  $\mathbf{X}$ , (b)  $\mathbf{Y}$ , (c)  $(\mathbf{X} + \mathbf{Y})_N$

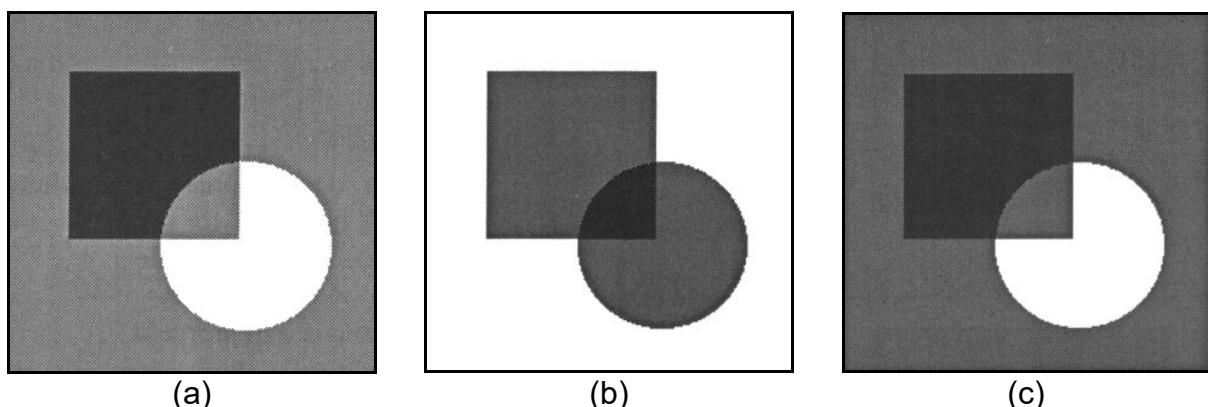


Figura 2.42 – Exemplos de subtração, multiplicação e divisão das imagens das figuras 2.41(a) e 2.41(b). (a)  $(\mathbf{X} - \mathbf{Y})_N$ ; (b)  $(\mathbf{X} * \mathbf{Y})_N$ ; (c)  $(\mathbf{X}/\mathbf{Y})_N$

### 2.8.2 Operações Lógicas Pixel a Pixel

Todas as operações lógicas (ou booleanas) conhecidas podem ser aplicadas entre imagens, inclusive a operação de complemento (NOT), que é uma operação unária. Operações lógicas podem ser efetuadas em imagens com qualquer número de níveis de cinza mas são melhor compreendidas quando vistas em imagens binárias, como ilustra a Figura 2.43. As Figuras 2.44 a 2.47 ilustram as operações AND, OR, XOR e NOT aplicadas a imagens com múltiplos tons de cinza.

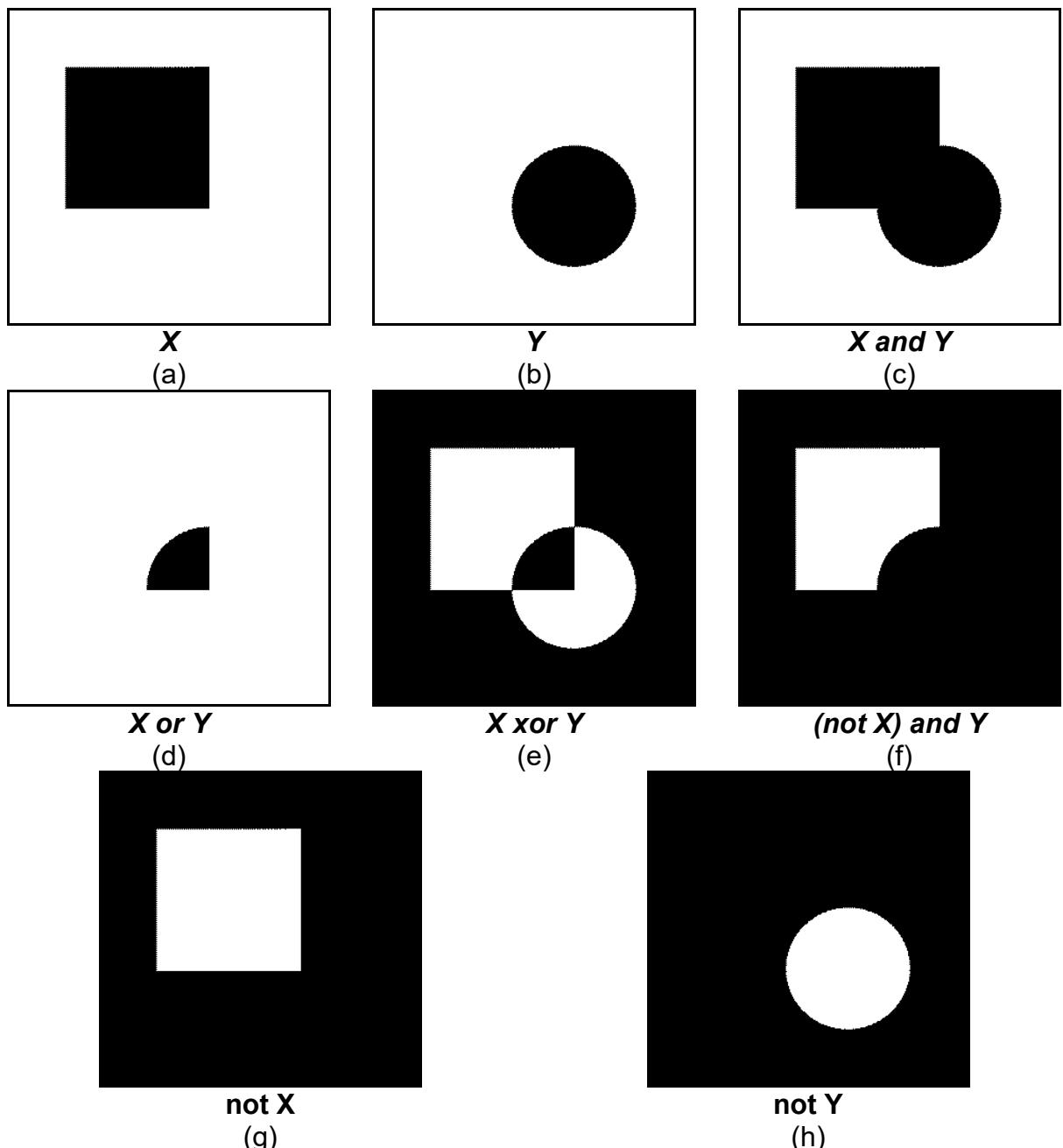


Figura 2.43 – Exemplos de operações lógicas em imagens binárias

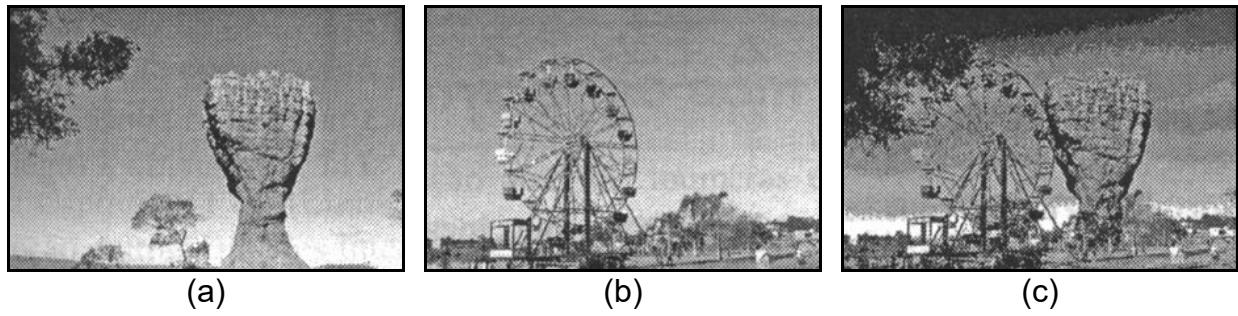


Figura 2.44 – Exemplo de operação AND entre imagens monocromáticas:  
 (a)  $X$ , (b)  $Y$ , (c)  $X$  and  $Y$

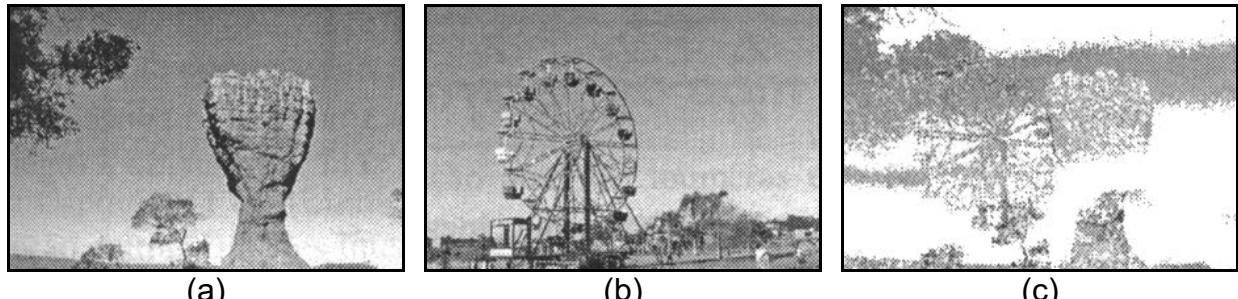


Figura 2.45 – Exemplo de operação OR entre imagens monocromáticas:  
 (a)  $X$ , (b)  $Y$ , (c)  $X$  or  $Y$

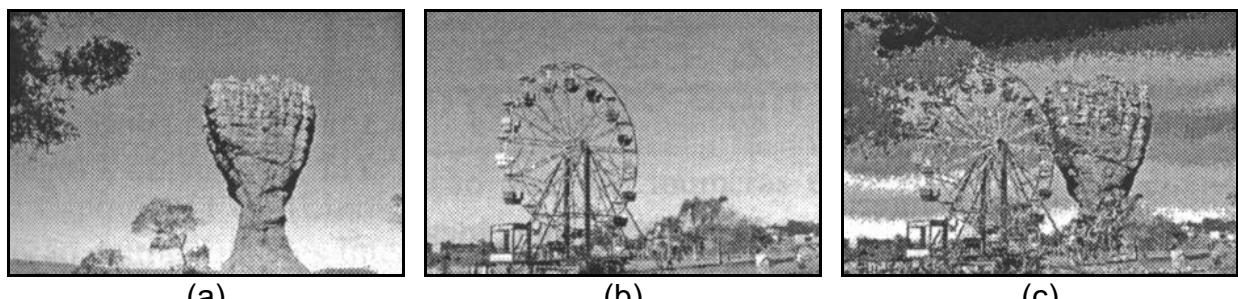


Figura 2.46 – Exemplo de operação XOR entre imagens monocromáticas:  
 (a)  $X$ , (b)  $Y$ , (c)  $X$  xor  $Y$

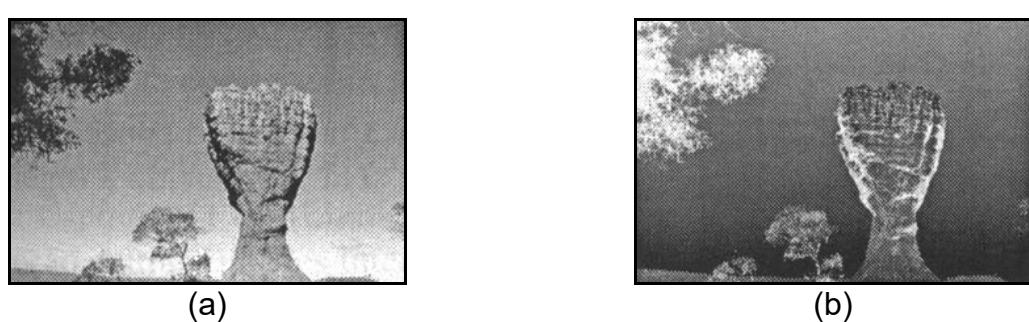


Figura 2.47 – Exemplo de operação NOT sobre imagens monocromáticas:  
 (a)  $X$ , (b)  $\text{not } X$

### 2.8.3 Operações Orientadas a Vizinhança

As operações lógicas e aritméticas orientadas a vizinhança utilizam o conceito da convolução com máscaras, que será introduzido a seguir e detalhado na próxima seção.

Seja uma sub-área de uma imagem, onde  $Z_1, \dots, Z_9$  são os valores de tons de cinza de cada pixel, e uma máscara  $3 \times 3$  de coeficientes genéricos  $W_1, \dots, W_9$ .

$Z_1$	$Z_2$	$Z_3$	$W_1$	$W_2$	$W_3$
$Z_4$	$Z_5$	$Z_6$	$W_4$	$W_5$	$W_6$
$Z_7$	$Z_8$	$Z_9$	$W_7$	$W_8$	$W_9$

A máscara anterior percorrerá a imagem, desde o seu canto superior esquerdo até seu canto inferior direito. A cada posição relativa da máscara sobre a imagem, o pixel central da sub-imagem em questão será substituído, em uma matriz denominada “imagem-destino”, por um valor:

$$Z = \sum_{i=1}^9 W_i \cdot Z_i$$

As operações de convolução com máscaras são amplamente utilizadas no processamento de imagens. Uma seleção apropriada dos coeficientes  $W_1, \dots, W_9$  torna possível uma grande variedade de operações úteis, tais como redução de ruído, afinamento e detecção de características da imagem. Deve-se observar, entretanto, que a operação de convolução com máscaras exige grande esforço computacional. Por exemplo, a aplicação de uma máscara  $3 \times 3$  sobre uma imagem  $512 \times 512$  requer 9 multiplicações e oito adições para cada localização de pixel, num total de 2.359.296 multiplicações e 2.097.152 adições. Por esta razão, aliada à relativa simplicidade de implementação de multiplicadores, somadores e registradores de deslocamento (shift registers), a literatura registra diversas implementações de convolução com máscaras em hardware.

## 2.9 OPERAÇÕES DE CONVOLUÇÃO COM MÁSCARAS

Inúmeras operações úteis em processamento de imagens são efetuadas a partir de um mesmo conceito básico, o de convolução com máscaras.

A operação de convolução unidimensional entre dois vetores  $\mathbf{A}$  e  $\mathbf{B}$ , denotada  $\mathbf{A} * \mathbf{B}$ , pode ser entendida como um conjunto de somas de produtos entre os valores de  $\mathbf{A}$  e  $\mathbf{B}$ , sendo que inicialmente o vetor  $\mathbf{B}$  é espelhado e, após cada soma de produtos, é deslocado espacialmente de uma posição. Para ilustrar este conceito, mostraremos a seguir, passo a passo, a convolução do vetor  $\mathbf{A} = \{0, 1, 2, 3, 2, 1, 0\}$  com o vetor  $\mathbf{B} = \{1, 3, -1\}$ .

1) Inicialmente, o vetor  $\mathbf{B}$  é espelhado e alinhado com o primeiro valor de  $\mathbf{A}$ . O resultado da convolução é  $(0 \times (-1)) + (0 \times 3) + (1 \times 1) = 1$  e é colocado em  $\mathbf{A} * \mathbf{B}$  na posição correspondente ao centro do conjunto  $\mathbf{B}$ .

$\mathbf{A}$		0	1	2	3	2	1	0	
$\mathbf{B}$	-1	3	1						
$\mathbf{A} * \mathbf{B}$		1							

2) O conjunto  $\mathbf{B}$  é deslocado de uma posição. O resultado da convolução  $\mathbf{A} * \mathbf{B}$  é  $(0 \times (-1)) + (1 \times 3) + (2 \times 1) = 5$ .

$\mathbf{A}$		0	1	2	3	2	1	0	
$\mathbf{B}$		-1	3	1					
$\mathbf{A} * \mathbf{B}$		1	5						

3) E assim sucessivamente até obtermos a última convolução possível, dada por  $(1 \times (-1)) + (0 \times 3) + (0 \times 1) = -1$

<b>A</b>		0	1	2	3	2	1	0	
<b>B</b>							-1	3	1
<b>A * B</b>		1	5	8	9	4	1	-1	

O conjunto  $\{1, 5, 8, 9, 4, 1, -1\}$  é o resultado final da operação de convolução.

Este raciocínio pode ser expandido para o caso bidimensional, onde a imagem a ser processada é uma matriz bidimensional relativamente grande e correspondente ao conjunto **A** de nosso exemplo anterior, enquanto que a matriz de pequenas dimensões (máscara) corresponde ao conjunto **B**. A máscara, após ter sido espelhada tanto na horizontal quanto na vertical, percorrerá todos os pontos da imagem deslocando-se ao longo de cada linha e entre as várias colunas, da direita para a esquerda, de cima para baixo, até ter processado o último elemento da matriz imagem. O resultado será armazenado em uma matriz de mesmas dimensões que a imagem original.

Seja a matriz **A** (imagem) dada por:

$$A = \begin{bmatrix} 5 & 8 & 3 & 4 & 6 & 2 & 3 & 7 \\ 3 & 2 & 1 & 1 & 9 & 5 & 1 & 0 \\ 0 & 9 & 5 & 3 & 0 & 4 & 8 & 3 \\ 4 & 2 & 7 & 2 & 1 & 9 & 0 & 6 \\ 9 & 7 & 9 & 8 & 0 & 4 & 2 & 4 \\ 5 & 2 & 1 & 8 & 4 & 1 & 0 & 9 \\ 1 & 8 & 5 & 4 & 9 & 2 & 3 & 8 \\ 3 & 7 & 1 & 2 & 3 & 4 & 4 & 6 \end{bmatrix}$$

e seja a matriz **B** (máscara) a seguir:

$$B = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

A operação de convolução bidimensional produzirá como resultado a matriz:

$$A * B = \begin{bmatrix} 20 & 10 & 2 & 26 & 23 & 6 & 9 & 4 \\ 18 & 1 & -8 & 2 & 7 & 3 & 3 & -11 \\ 14 & 22 & 5 & -1 & 9 & -2 & 8 & -1 \\ 29 & 21 & 9 & -9 & 10 & 12 & -9 & -9 \\ 21 & 1 & 16 & -1 & -3 & -4 & 2 & 5 \\ 15 & -9 & -3 & 7 & -6 & 1 & 17 & 9 \\ 21 & 9 & 1 & 6 & -2 & -1 & 23 & 2 \\ 9 & -5 & -25 & -10 & -12 & -15 & -1 & -12 \end{bmatrix}$$

A Figura 2.48 ilustra em detalhes o cálculo do resultado correspondente ao pixel no canto superior esquerdo da imagem. Observar que a máscara **B** foi espelhada em relação a **x** e **y** antes do cálculo das somas de produtos.

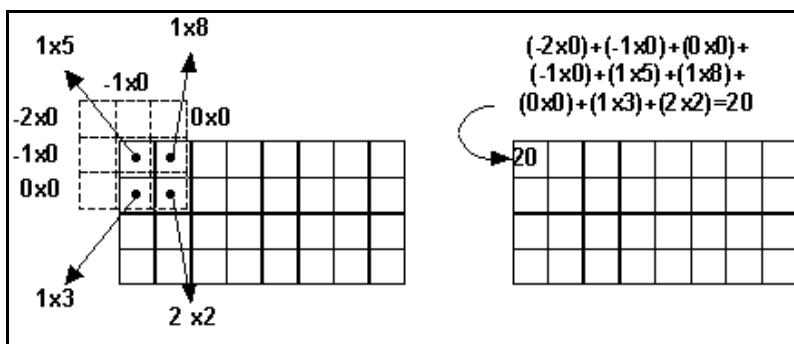


Figura 2.48 – Cálculo do primeiro valor de convolução de A por B

Para calcular os valores resultantes dos pixels próximos às bordas da imagem, podem ser adotadas diversas estratégias, dentre elas:

- preencher com zeros os contornos da imagem, de maneira condizente com o tamanho da máscara utilizado, como ilustra a Figura 2.48;
- preencher o contorno da imagem com os mesmos valores da(s) primeira(s) e última(s) linha(s) e coluna(s);
- prevenir a eventual introdução de erros nas regiões de bordas da imagem causados por qualquer um dos métodos acima, considerando na imagem resultante apenas os valores para os quais a máscara de convolução ficou inteiramente contida na imagem original.

A seguir, ilustraremos o uso do conceito de convolução com máscaras, aplicado à detecção de características de imagens, particularmente pontos isolados, linhas e bordas.

### 2.9.1 Detecção de Pontos Isolados e Detecção de Linhas

A máscara a seguir é um exemplo de operador de convolução que, quando aplicado a uma imagem, destacará pixels brilhantes circundados por pixels mais escuros.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

As máscaras a seguir podem ser utilizadas para a detecção de linhas horizontais e verticais (acima) e diagonais (abaixo).

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

### 2.9.2 Detecção de Bordas

O tema “detecção de bordas” (*edge detection*) vem desafiando os pesquisadores da área de Processamento de Imagens há muitos anos e sobre ele continuam sendo experimentadas novas técnicas, cujos resultados são publicados ainda hoje nos mais conceituados periódicos científicos mundiais. Trata-se, portanto, de um tema em aberto, a detecção de bordas em cenas consideradas difíceis.

Apenas a título de ilustração, da operação de convolução com máscaras, apresentamos a seguir alguns exemplos de máscaras que podem ser utilizadas para a tarefa de detecção de bordas.

Define-se borda (*edge*) como a fronteira entre duas regiões, cujos níveis de cinza predominantes são razoavelmente diferentes. PRATT (1991) define uma borda de luminosidade como uma descontinuidade na luminosidade de uma imagem. Analogamente, pode-se definir borda de textura ou borda de cor, em imagens onde as informações de textura ou cor, respectivamente, são as mais importantes. Aqui trataremos somente as bordas de luminosidade, às quais denominaremos simplesmente bordas.

Para a detecção e realce de bordas, aplicam-se habitualmente filtros espaciais lineares de dois tipos: (a) baseados no gradiente da função de luminosidade,  $I(x, y)$ , da imagem, e (b) baseados no laplaciano de  $I(x, y)$ .

Tanto o gradiente quanto o laplaciano costuma ser aproximados por máscaras de convolução ou operadores 3 x 3. Exemplos destas máscaras são os operadores de Roberts, Sobel, Prewitt e Frei-Chen, mostrados na Tabela 2.9.

TABELA 2.9 – Operadores 3x3 utilizados para estimar a amplitude do gradiente através de uma borda

Operador	Vertical	Horizontal
Roberts	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Sobel	$\frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{4} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Prewitt	$\frac{1}{3} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{3} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Frei-Chen	$\frac{1}{2+\sqrt{2}} \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{2+\sqrt{2}} \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

A Figura 2.49 mostra os resultados da aplicação dos operadores de Prewitt e Sobel a uma imagem monocromática. Os resultados obtidos com a aplicação dos operadores verticais e horizontais foram combinados por meio de uma operação lógica OR.

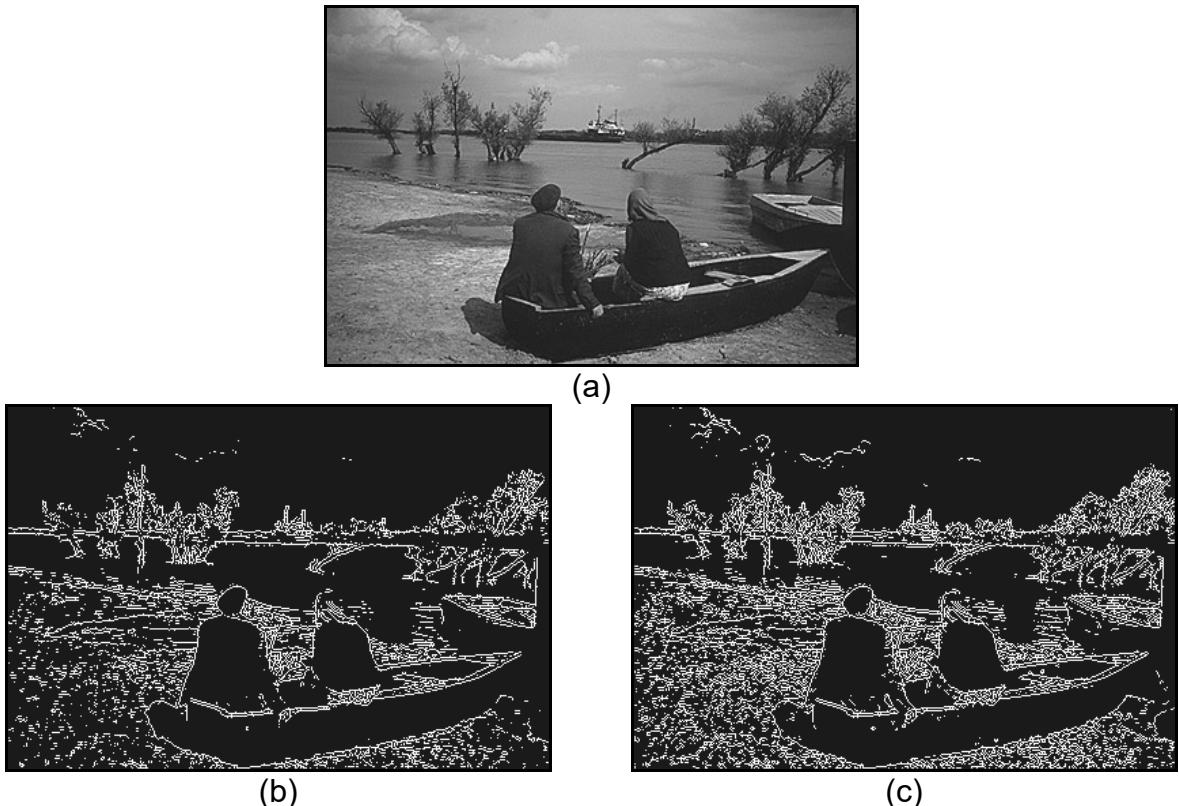


Figura 2.49 – Exemplos de realce e detecção de bordas, (a) imagem original, (b) realce de bordas utilizando os operadores de Prewitt horizontal e vertical, (c) realce de bordas utilizando os operadores de Sobel horizontal e vertical

O laplaciano é um operador definido como:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

e que pode ser aproximado pelas máscaras  $(3 \times 3)$ ,  $(5 \times 5)$  e  $(9 \times 9)$ :

A Figura 2.50 mostra o resultado obtido com a aplicação da máscara  $3 \times 3$  sobre uma imagem monocromática.

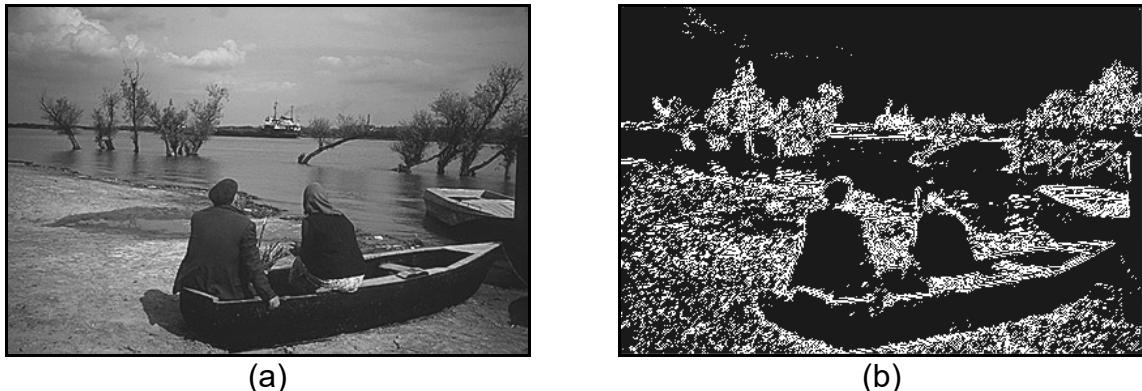


Figura 2.50 – Resultado da aplicação da máscara do laplaciano: (a) figura original, (b) laplaciano com máscara  $3 \times 3$

Embora o laplaciano seja sensível à rotação, e portanto capaz de realçar ou detectar bordas em qualquer direção, seu uso é restrito devido a sua grande suscetibilidade a ruído.

Existem outros operadores direcionais, que nada mais são que conjuntos de máscaras que representam aproximações discretas de bordas ideais em várias direções. Estes operadores incluem as máscaras direcionais introduzidas por Prewitt, Kirsch, e as máscaras simples de 3 e 5 níveis de Robinson. A Tabela 2.10 mostra estas máscaras com suas respectivas direções cardinais.

TABELA 2.10 – Máscaras de Prewitt, Kirsch e Robinson

Direção da Borda	Direção Gradiente	Prewitt	Kirsch	Robinson 3 níveis	Robinson 5 níveis
0	N	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
1	NO	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$
2	O	$\begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$
3	SO	$\begin{bmatrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$
4	S	$\begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Direção da Borda	Direção Gradiente	Prewitt	Kirsch	Robinson 3 níveis	Robinson 5 níveis
5	SE	$\begin{bmatrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$
6	E	$\begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
7	NE	$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$
Fator de escala		$\frac{1}{5}$	$\frac{1}{15}$	$\frac{1}{3}$	$\frac{1}{4}$

## 2.10 FILTRAGEM, REALCE E SUAVIZAÇÃO DE IMAGENS

O principal objetivo das técnicas de realce de imagens é processar uma certa imagem de modo que a imagem resultante seja mais adequada, que a imagem original, para uma aplicação específica. Desta afirmativa decorrem duas importantes conclusões:

- 1) A interpretação de que o resultado é mais adequado, ou não, normalmente é subjetiva e depende de conhecimento prévio do observador a respeito das imagens analisadas;
- 2) As técnicas de realce de imagens a serem estudas nesta seção são, por natureza, orientadas a um problema que se deseja resolver. Logo, não existem técnicas capazes de resolver 100% dos problemas que uma imagem digital possa apresentar, como também nem sempre uma técnica que produz bons resultados para imagens biomédicas, adquiridas através de um tomógrafo computadorizado, apresentará desempenho satisfatório, se aplicada a uma imagem contendo uma impressão digital, por exemplo.

Os métodos de filtragem de imagens discutidos nesta seção são normalmente classificados em duas categorias: as técnicas de **filtragem espacial** e as técnicas de **filtragem no domínio da frequência**. Os métodos que trabalham no domínio espacial operam diretamente sobre a matriz de pixels que é a imagem digitalizada, normalmente utilizando operações de convolução com máscaras. Os métodos que atuam no domínio da frequência se baseiam na modificação da transformada de Fourier da imagem. Existem técnicas de filtragem que combinam ambas as abordagens.

### 2.10.1 Filtragem no Domínio Espacial

As técnicas de filtragem no domínio espacial são aquelas que atuam diretamente sobre a matriz de pixels que é a imagem digitalizada. Logo, as funções

de processamento de imagens no domínio espacial podem ser expressas como:

$$g(x, y) = T[f(x, y)]$$

onde:  $g(x, y)$  é a imagem processada,  $f(x, y)$  é a imagem original e  $T$  é um operador em  $f$ , definido em uma certa vizinhança de  $(x, y)$ . Além disso, o operador  $T$  pode também operar sobre um conjunto de imagens de entrada.

A vizinhança normalmente definida ao redor de  $(x, y)$  é a 8-vizinhança do pixel de referência, o que equivale a uma região  $3 \times 3$ , na qual o pixel central é o de referência, como indica a Figura 2.51. O centro dessa região ou sub-imagem é movido pixel a pixel, iniciando no canto superior esquerdo da figura e aplicando a cada localidade o operador  $T$  para calcular o valor de  $g$  naquele ponto.

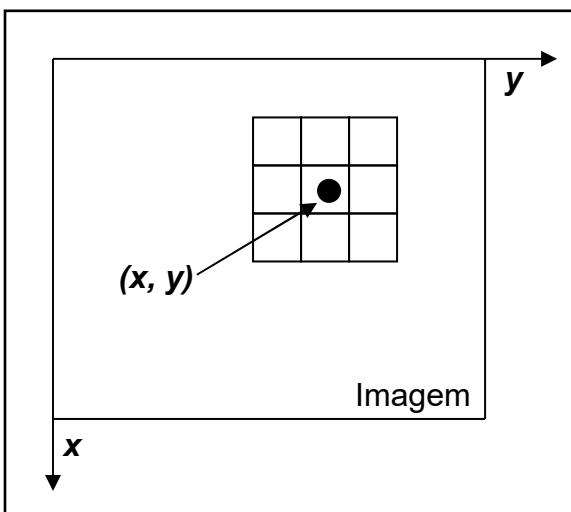


Figura 2.51 – Uma vizinhança  $3 \times 3$  ao redor de um ponto de coordenadas  $(x, y)$  em uma imagem

Nos casos em que a vizinhança é  $1 \times 1$ , o operador  $T$  torna-se uma função de transformação (ou de mapeamento), do tipo:

$$s = T(r)$$

onde:  $r$  é o nível de cinza de  $f(x, y)$  e  $s$  é o nível de cinza de  $g(x, y)$  em um certo ponto.

Um exemplo deste tipo de operação é a conversão de uma imagem colorida para tons de cinza em que, dada uma imagem  $f(x, y)$ , podemos obter uma imagem monocromática  $g$ , calculando para cada pixel o valor da luminância. Ou seja,  $g(x, y) = L(f(x, y))$ , onde  $L$  é o operador de luminância. Mais precisamente, se  $f(x, y) = (R(x, y), G(x, y), B(x, y))$ , é dada por suas componentes no sistema RGB, podemos calcular  $g$  usando o operador de luminância NTSC,  $g(x, y) = 0,299R(x, y) + 0,587G(x, y) + 0,114B(x, y)$ . Esse é o método de “descolorizar” uma imagem, obtendo uma imagem “grayscale” a partir de uma imagem colorida. Este é um exemplo de operador unário e pontual.

As técnicas de processamento de imagem pertencentes a este caso são freqüentemente denominadas técnicas ponto-a-ponto e outros exemplos foram abordados na seção Transformações de intensidade.

## 2.10.2 Filtragem no Domínio da Frequênciа

A base matemática das técnicas de filtragem no domínio da frequência é o teorema da convolução. Seja  $\mathbf{g}(\mathbf{x}, \mathbf{y})$  a imagem formada pela convolução (denotada pelo símbolo  $*$ ) da imagem  $\mathbf{f}(\mathbf{x}, \mathbf{y})$  com um operador linear  $\mathbf{h}(\mathbf{x}, \mathbf{y})$ , ou seja,

$$\mathbf{g}(\mathbf{x}, \mathbf{y}) = \mathbf{f}(\mathbf{x}, \mathbf{y}) * \mathbf{h}(\mathbf{x}, \mathbf{y})$$

Então, pelo teorema da convolução, a seguinte relação no domínio da frequência também é válida:

$$G(u, v) = F(u, v)H(u, v)$$

onde  $\mathbf{G}$ ,  $\mathbf{F}$  e  $\mathbf{H}$  são as transformadas de Fourier de  $\mathbf{g}$ ,  $\mathbf{f}$  e  $\mathbf{h}$ , respectivamente. Na terminologia de sistemas lineares, a transformada  $\mathbf{H}(u, v)$  é denominada função de transferência do filtro.

Inúmeros problemas de processamento de imagens podem ser expressos na forma da equação anterior. Em uma aplicação de suavização de imagens, por exemplo, dada  $\mathbf{f}(\mathbf{x}, \mathbf{y})$ , o objetivo, após calcular  $\mathbf{F}(u, v)$ , é selecionar  $\mathbf{H}(u, v)$  de tal maneira que a imagem desejada

$$\mathbf{g}(\mathbf{x}, \mathbf{y}) = \mathcal{I}^{-1}[F(u, v)H(u, v)]$$

remova componentes de alta frequência (possivelmente ruidosos) de  $\mathbf{f}(\mathbf{x}, \mathbf{y})$ . Isto poderia ser obtido usando um filtro Butterworth passa-baixa, por exemplo.

A equação  $\mathbf{g}(\mathbf{x}, \mathbf{y}) = \mathbf{f}(\mathbf{x}, \mathbf{y}) * \mathbf{h}(\mathbf{x}, \mathbf{y})$  descreve um processo espacial análogo ao explicado anteriormente e, por esta razão,  $\mathbf{h}(\mathbf{x}, \mathbf{y})$  é freqüentemente denominada máscara de convolução espacial.

## 2.10.3 Suavização de Imagens no Domínio Espacial

O uso de máscaras espaciais no processamento de imagens é normalmente denominado filtragem espacial e as máscaras são conhecidas como filtros espaciais. Nesta seção consideraremos filtros lineares e não-lineares aplicados ao processamento de imagens.

Os filtros lineares se baseiam no conceito de que a função de transferência de um sistema linear  $\mathbf{H}(u, v)$  e sua função de resposta a impulso unitário  $\mathbf{h}(\mathbf{x}, \mathbf{y})$  estão relacionadas entre si através da transformada de Fourier, como ilustra a Figura 2.52.

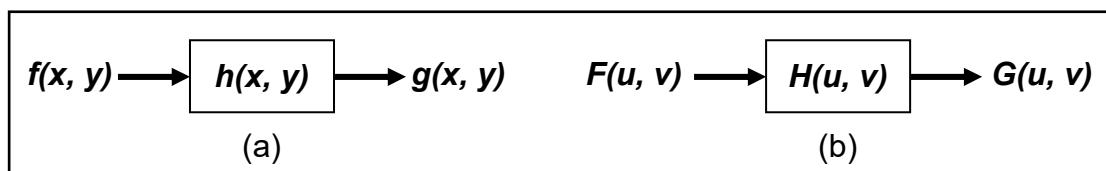


Figura 2.52 – Fundamentos de sistemas lineares. Na parte (a) (domínio espacial), a saída do sistema é obtida através da convolução de sua entrada com sua função de resposta a impulso unitário  $\mathbf{h}(\mathbf{x}, \mathbf{y})$ . Em (b) (domínio da frequência), a saída do sistema é o produto de sua função de transferência  $\mathbf{H}(u, v)$  pela entrada

Os filtros são denominados “passa-baixa” quando atenuam ou eliminam as componentes de alta frequência no domínio das transformadas de Fourier. Como as componentes de alta frequência correspondem a regiões de bordas e/ou detalhes finos na imagem, o efeito da filtragem passa-baixa é a suavização da imagem, provocando um leve borramento na mesma. Já os filtros passa-alta atenuam ou eliminam as componentes de baixa frequência e, em função disto, realçam as bordas e regiões de alto contraste da imagem. Os filtros passa-faixa, capazes de remover ou atenuar componentes acima de sua frequência de corte superior e abaixo de sua frequência de corte inferior, embora existam, são de pouca utilidade prática, com exceção de algumas tarefas específicas de restauração de imagens.

A Figura 2.53 mostra as respostas em frequência dos três principais tipo de filtros existentes e os respectivos filtros espaciais correspondentes.

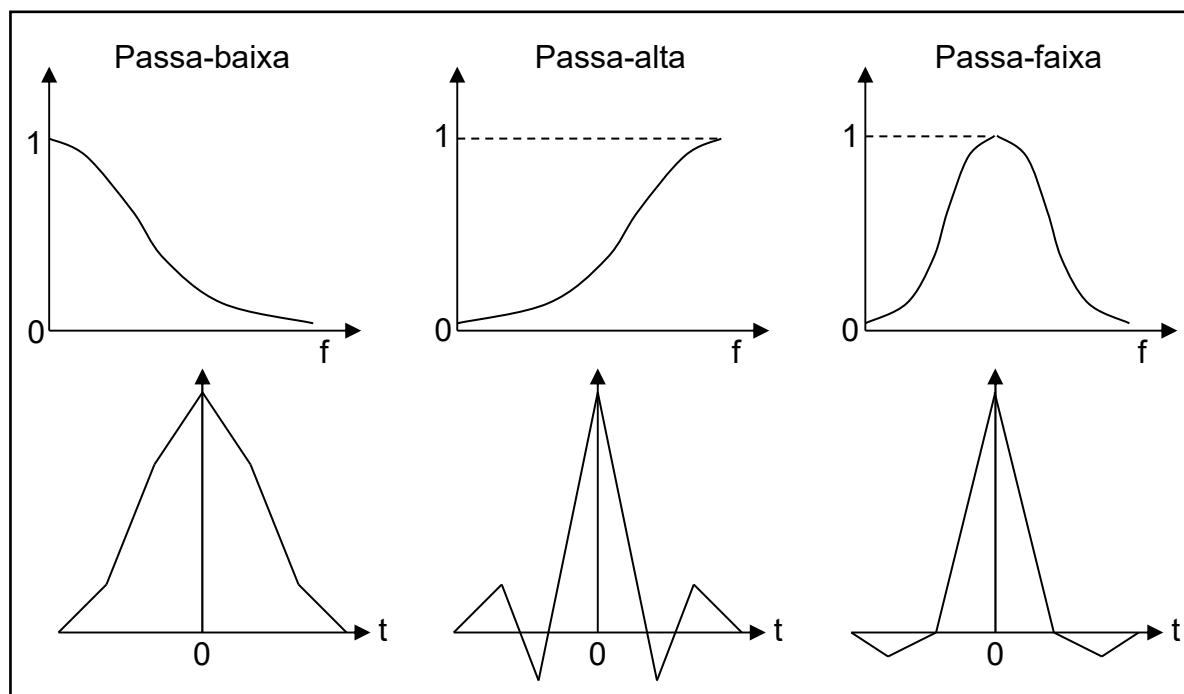


Figura 2.53 – (Acima) Resposta em frequência dos principais tipos de filtros.  
(Abaixo) Filtros correspondentes no domínio espacial

A suavização de imagens no domínio espacial baseia-se no uso de máscaras de convolução adequadas para o objetivo em questão, normalmente o borramento da imagem ou a remoção de ruídos nela presentes. Dentre as técnicas mais conhecidas de suavização estão a filtragem pela média e o filtro da mediana, detalhadas a seguir.

- Filtro da Média

Como se pode perceber na Figura 2.53, a resposta ao impulso de um filtro passa-baixa indica que ele deve apresentar todos os seus coeficiente positivos. A forma mais simples de implementar um filtro com tais características é construir uma máscara  $3 \times 3$  com todos os coeficientes iguais a 1, dividindo o resultado da convolução por um fator de normalização, neste caso igual a 9. Um filtro com esta característica é denominado filtro da média. A seguir apresentamos as máscaras  $3 \times 3$ ,  $5 \times 5$  e  $7 \times 7$ .

$$\begin{array}{c}
 \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{49} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}
 \end{array}$$

Na escolha do tamanho da máscara, deve-se ter em mente que, quanto maior a máscara, maior o grau de borramento da imagem resultante. A Figura 2.54 mostra exemplos de máscaras de filtragem pela média de diferentes dimensões aplicadas a uma mesma imagem.

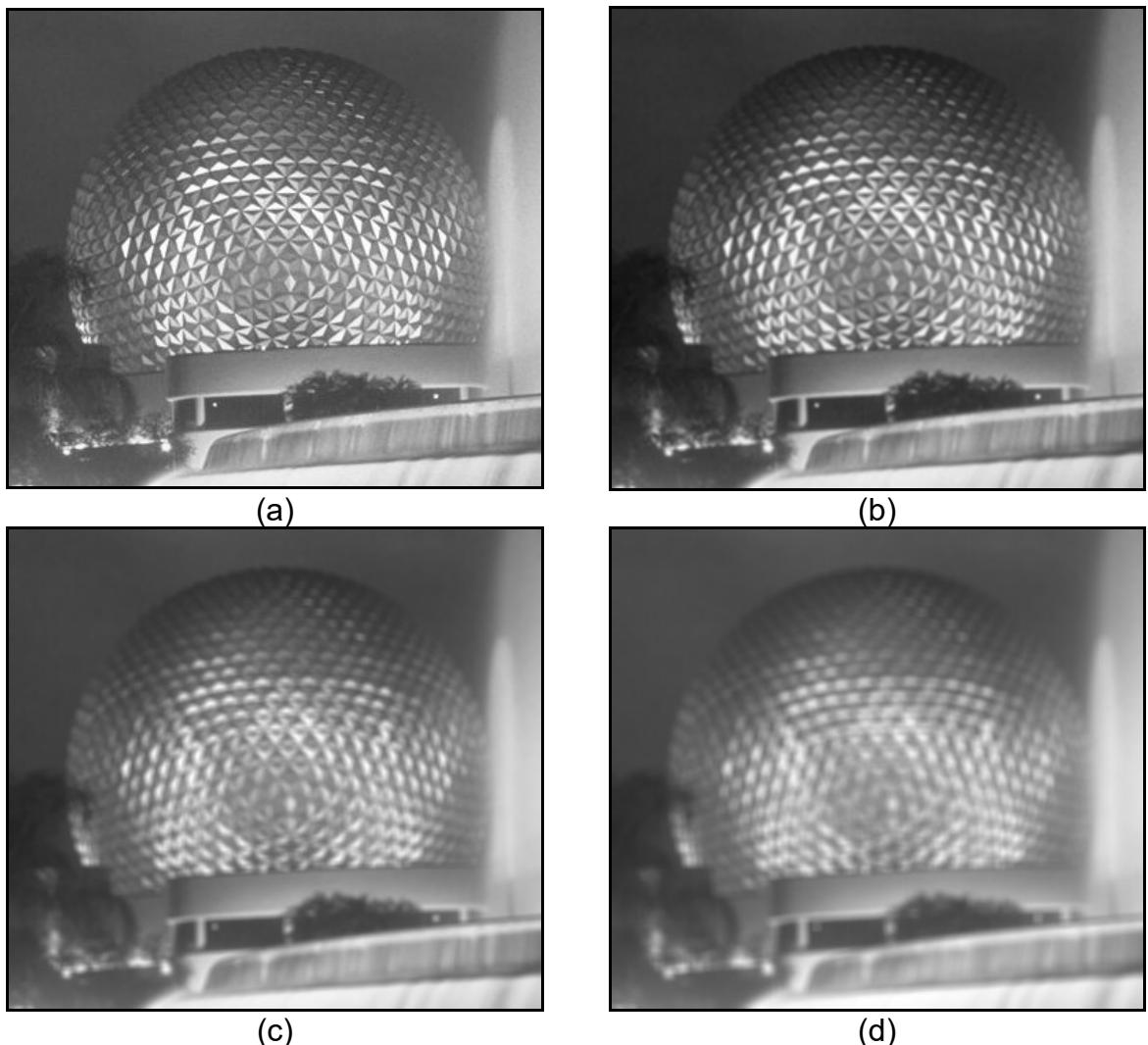


Figura 2.54 – (a) Imagem original. (b)-(d) resultados da aplicação do filtro da média com máscara de dimensões  $n \times n$ ,  $n = 3, 5, 7$ .

As Figura 2.55 e 2.56 mostram exemplos de aplicações do filtro da média para remoção de ruídos em imagens monocromáticas.

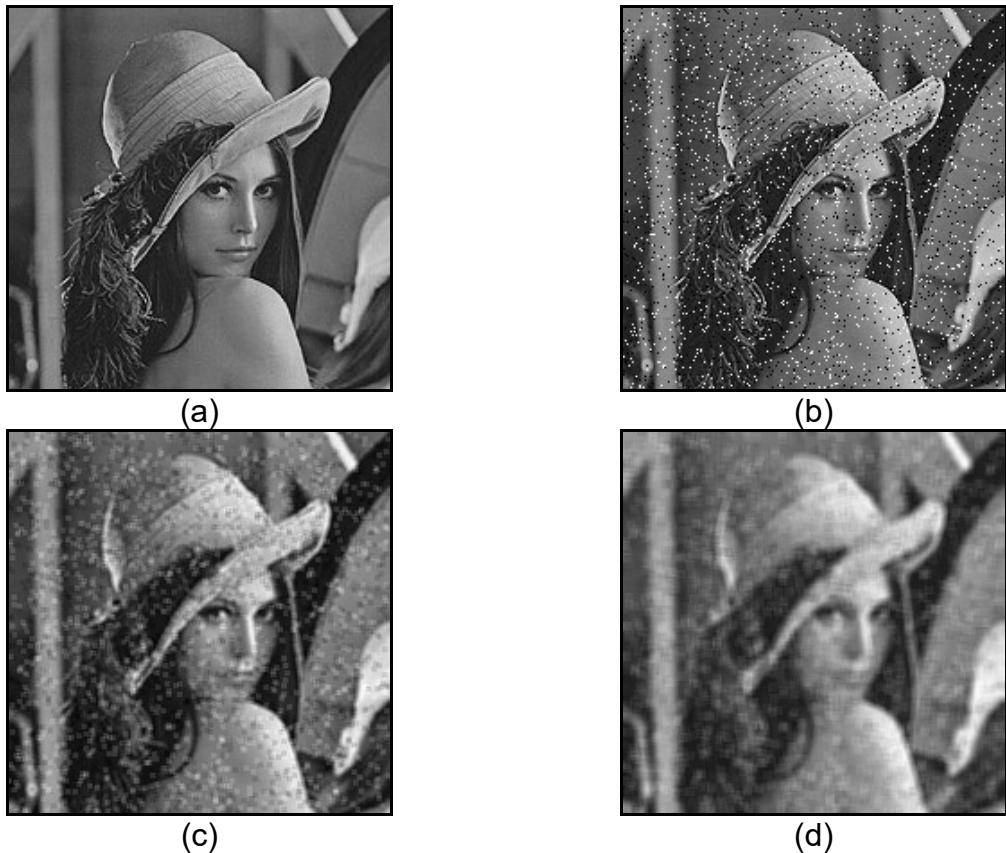


Figura 2.55 – (a) Imagem original; (b) imagem contaminada por ruído impulsivo (sal e pimenta); (c)-(d) resultado da filtragem com filtro da média com máscara  $3 \times 3$  e  $5 \times 5$

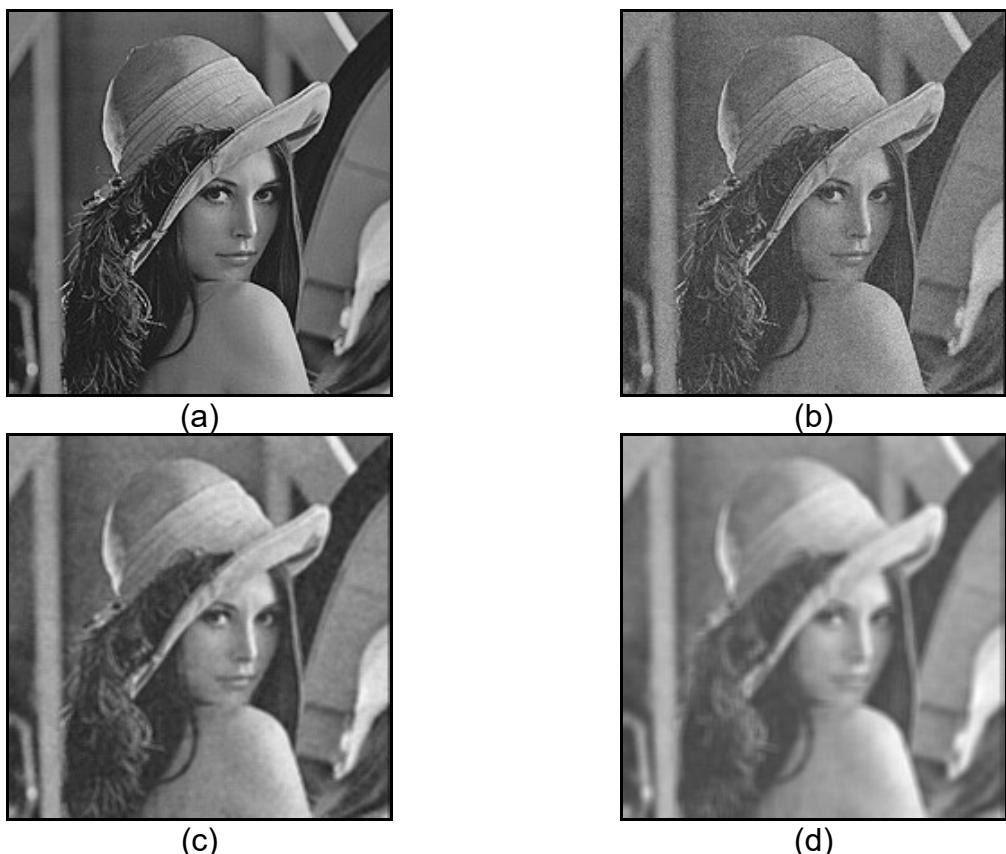


Figura 2.56 – (a) Imagem original; (b) imagem contaminada por ruído gaussiano; (c)-(d) resultado da filtragem com filtro da média com máscara  $3 \times 3$  e  $5 \times 5$

O algoritmo básico de filtragem pela média pode ser alterado no sentido de minimizar a perda de definição na imagem resultante. Uma possível modificação consiste em incluir uma comparação do valor calculado com um limiar ( $T$ ), antes de alterar seu tom de cinza. Se o valor absoluto da diferença entre o nível de cinza original do pixel ( $f(x, y)$ ) e o valor calculado pela aplicação do filtro da média for menor que  $T$ , substitui-se o tom de cinza do pixel pelo valor calculado; caso contrário, mantém-se o valor de cinza original. O objetivo principal desta modificação é diminuir o efeito de suavização indesejável das bordas dos objetos presentes na imagem.

- Filtro da Mediana

Uma das principais limitações do filtro da média, em situações onde o objetivo é remoção de ruídos em imagens, está na sua incapacidade de preservar bordas e detalhes finos da imagem. Para contorná-la, uma técnica alternativa é o filtro da mediana. Nesta técnica, o nível de cinza do pixel central da janela é substituído pela mediana dos pixels situados em sua vizinhança.

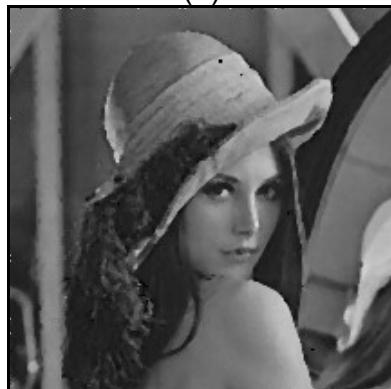
Este método não-linear apresenta desempenho particularmente bom em situações nas quais a imagem é contaminada por ruído impulsivo (sal e pimenta), como ilustra a Figura 2.57. Já para situações em que o ruído é do tipo gaussiano (Figura 2.58), seu desempenho é apenas satisfatório, comparável ao filtro pela média.



(a)



(b)



(c)



(d)

Figura 2.57 – (a) Imagem original; (b) imagem contaminada por ruído impulsivo (sal e pimenta); (c)-(d) resultado da filtragem com filtro da mediana e média com máscara 3x3



Figura 2.58 – (a) Imagem original; (b) imagem contaminada por ruído gaussiano; (c)-(d) resultado da filtragem com filtro da mediana e média com máscara 3x3

A mediana  $m$  de um conjunto de  $n$  elementos é o valor tal que metade dos  $n$  elementos do conjunto situem-se abaixo de  $m$  e a outra metade acima de  $m$ . Quando  $n$  é ímpar, a mediana é o próprio elemento central do conjunto ordenado. Nos casos em que  $n$  é par, a mediana é calculada pela média aritmética dos dois elementos mais próximos do centro. A ordenação constitui uma etapa de tempo de processamento relativamente alto, apesar de inúmeros métodos eficientes existentes na literatura. Para reduzir o custo computacional do filtro da mediana foi proposto um método alternativo, denominado filtro da pseudomediana, o qual estabelece que a pseudomediana de um conjunto de  $L$  elementos ( $S_L$ ) pode ser computada como:

$$PMED\{S_L\} = \frac{MAXMIN\{S_L\} + MINIMAX\{S_L\}}{2}$$

onde:

$$MAXMIN\{S_L\} = MAX\{[MIN(S_1, \dots, S_M)], [MIN(S_2, \dots, S_{M+1})], \dots, [MIN(S_{L-M+1}, \dots, S_L)]\}$$

e

$$MINIMAX\{S_L\} = MIN\{[MAX(S_1, \dots, S_M)], [MAX(S_2, \dots, S_{M+1})], \dots, [MAX(S_{L-M+1}, \dots, S_L)]\}$$

$$\text{para } M = \frac{L+1}{2}.$$

- Média de Múltiplas Imagens

Seja uma imagem ruidosa  $\mathbf{g}(x, y) = \mathbf{f}(x, y) + \boldsymbol{\eta}(x, y)$  onde  $\mathbf{f}(x, y)$  é a imagem original e  $\boldsymbol{\eta}(x, y)$  é um padrão de ruído aditivo de média zero e descorrelacionado, que se sobrepõe à imagem. Supondo também a existência de  $M$  imagens ruidosas, cada qual adquirida em um instante diferente, pode-se calcular uma imagem média:

$$\bar{\mathbf{g}}(x, y) = \frac{1}{M} \sum_{i=1}^M \mathbf{g}_i(x, y)$$

na qual a influência do ruído terá sido minimizada. Pode-se mostrar que:

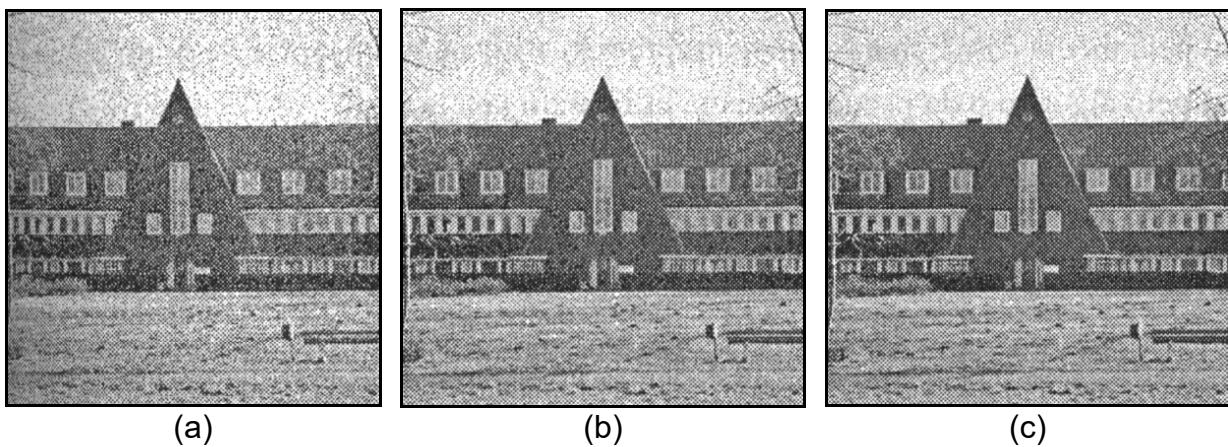
$$\begin{aligned} E\{\bar{\mathbf{g}}(x, y)\} &= \mathbf{f}(x, y) \\ \sigma_{\bar{\mathbf{g}}(x, y)} &= \frac{1}{\sqrt{M}} \sigma_{\boldsymbol{\eta}(x, y)} \\ \sigma_{\bar{\mathbf{g}}(x, y)}^2 &= \frac{1}{M} \sigma_{\boldsymbol{\eta}(x, y)}^2 \end{aligned}$$

onde  $E\{\bar{\mathbf{g}}(x, y)\}$  é o valor esperado de  $\bar{\mathbf{g}}(x, y)$ ,  $\sigma_{\bar{\mathbf{g}}(x, y)}^2$  e  $\sigma_{\boldsymbol{\eta}(x, y)}^2$  são, respectivamente, as variâncias da imagem filtrada e do ruído aditivo, enquanto  $\sigma_{\bar{\mathbf{g}}(x, y)}$  e  $\sigma_{\boldsymbol{\eta}(x, y)}$  são seus respectivos desvios-padrão.

Estas equações nos permitem concluir que, quanto maior for o valor de  $M$ , menor a variância (e portanto o desvio-padrão) dos pixels de  $\bar{\mathbf{g}}(x, y)$  e mais a imagem  $\bar{\mathbf{g}}(x, y)$  irá se aproximar de  $\mathbf{f}(x, y)$ .

Esta técnica opera de forma igualmente satisfatória para ruído gaussiano ou aleatório, quando o número de imagens utilizadas no cálculo da imagem média é significativo, devido ao Teorema do Limite Central, que estabelece que a soma de um grande número de termos, representando ruídos aleatórios, tende a produzir um ruído resultante do tipo gaussiano e independente dos tipos dos ruídos incluídos naquela soma.

A Figura 2.59 apresenta um exemplo de uso da técnica da média de múltiplas imagens para redução de ruído.



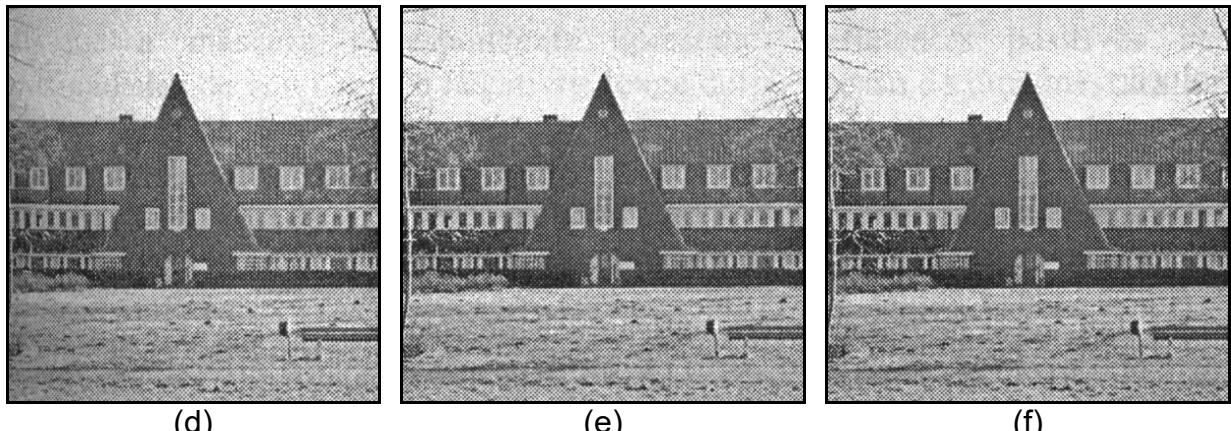


Figura 2.59 – Exemplo de redução usando média de múltiplas imagens: (a) imagem ruidosa; (b)-(f) resultados para  $M = 2, 4, 8, 16$  e  $32$  imagens

- Média dos  $k$  Vizinhos mais Próximos

Esta técnica, consiste em uma variação do método da filtragem pela média, na qual o pixel central da janela é substituído pela média dos  $k$  vizinhos, cujas amplitudes mais se aproximam da amplitude do pixel central. Seu objetivo é deliberadamente evitar incluir no cálculo da média valores que possam estar sob a janela em decorrência de bordas ou regiões de alto contraste. Quanto maior o valor de  $k$ , mais o desempenho deste filtro se aproximarará do filtro da média.

- Filtro de Bartlett

Este filtro também é conhecido pelo nome de ***filtro triangular***. No caso unidimensional o seu núcleo é definido por

$$h_1(x) = \begin{cases} 1 - |x| & \text{se } |x| \leq 1 \\ 0 & \text{se } |x| \geq 1 \end{cases}$$

Podemos obter uma máscara de ordem 3 do filtro de Bartlett unidimensional, tomando uma discretização do suporte  $[-1, 1]$  do núcleo  $h_1(x)$  nos pontos  $x = -1/2, x = 0, x = 1/2$ . Obtemos então a máscara:

$$\frac{1}{2} \cdot \left[ \frac{1}{2} \quad 1 \quad \frac{1}{2} \right]$$

ou, de forma equivalente,

$$\frac{1}{4} \cdot [1 \quad 2 \quad 1]$$

Podemos obter uma máscara de ordem 5, fazendo a discretização uniforme do suporte do núcleo nos pontos do conjunto  $\{-2/3, -1/3, 0, 1/3, 2/3\}$ . Obtemos então

$$\frac{1}{3} \cdot \begin{bmatrix} 1 & 2 & 1 & 2 & 1 \end{bmatrix} \text{ ou } \frac{1}{9} \cdot [1 \ 2 \ 3 \ 2 \ 1]$$

Procedendo de modo análogo podemos obter uma discretização do núcleo do filtro triangular de qualquer ordem. O leitor pode verificar que o núcleo de ordem 7 é dado por:

$$\frac{1}{16} \cdot [1 \ 2 \ 3 \ 4 \ 3 \ 2 \ 1]$$

Para obter uma máscara bidimensional do filtro de Bartlett, basta lembrar que o núcleo deste filtro é separável, ou seja,

$$h_2(x, y) = h_1(x)h_1(y)$$

Esse processo é ilustrado abaixo, para obter a máscara de ordem 5 do filtro de Bartlett.

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

Não se pode esquecer que a máscara acima deve ser normalizada de forma a ter média 1. O fator de normalização nesse caso é 1/81. Portanto a máscara de ordem 5 para o filtro de Bartlett bidimensional é dada por:

$$\frac{1}{81} \cdot \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

Na Figura 2.60 mostramos um exemplo de uma imagem filtrada utilizando o filtro de Bartlett de ordem 5. A atenuação das altas frequências com o filtro de Bartlett é bem mais acentuada que a atenuação efetuada pelo filtro da média.

O uso do filtro de Bartlett é conhecido na literatura pelo nome de **interpolação bilinear**. A razão desse nome provém do fato de que, quando utilizado para reconstrução de imagens, seu valor em um pixel é obtido fazendo duas interpolações lineares, uma na linha e outra na coluna da imagem que contém o pixel. O leitor pode verificar esse fato no cálculo dos pixels **a**, **b**, **c**, **d** e **e** da imagem

$f(i, j)$	<b>a</b>	$f(i, j + 1)$
<b>b</b>	<b>c</b>	<b>d</b>
$f(i + 1, j)$	<b>e</b>	$f(i + 1, j + 1)$

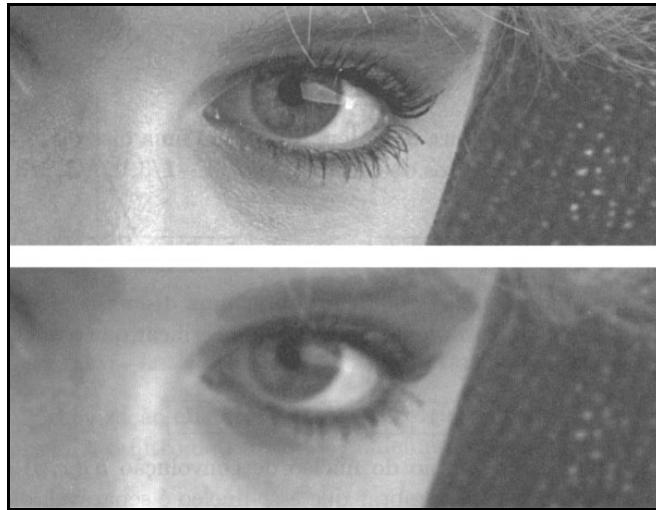


Figura 2.60 – A imagem inferior mostra a filtragem da imagem superior por um filtro de Bartlett de ordem 5

obtemos o resultado:

$$\begin{aligned}
 a &= \frac{f(i, j) + f(i, j+1)}{2}; \\
 b &= \frac{f(i, j) + f(i+1, j)}{2}; \\
 c &= \frac{f(i, j) + f(i, j+1) + f(i+1, j) + f(i+1, j+1)}{4}; \\
 d &= \frac{f(i, j+1) + f(i+1, j+1)}{2}; \text{ e} \\
 e &= \frac{f(i+1, j) + f(i+1, j+1)}{2}.
 \end{aligned}$$

Ao efetuar os cálculos acima, os valores de **a**, **b**, **c**, **d** e **e** são considerados nulos inicialmente.

- Filtro Gaussiano

No caso unidimensional, no domínio contínuo, o filtro gaussiano tem um núcleo  $\mathbf{G}_\sigma(\mathbf{x})$  dado pela função de distribuição gaussiana

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$$

com média 0 e variância  $\sigma^2$ . No caso bidimensional o núcleo é definido por

$$G_\sigma(x, y) = \frac{1}{2\sigma^2\pi} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Uma análise rápida no domínio contínuo, mostra que o filtro gaussiano é

um filtro de passa-baixa.

Para obter uma discretização unidimensional do filtro gaussiano, podemos proceder como no caso da discretização do filtro de Bartlett vista acima. No entanto, devido à expressão que define o núcleo do filtro gaussiano, esse processo nos levaria a uma máscara discreta com valores contendo valores aproximados. Isso acarreta problemas diversos, principalmente de ordem computacional. Um método mais elegante, e eficiente, consiste em obter por aproximações sucessivas máscaras para o filtro gaussiano. Com efeito, partimos de um filtro da média com máscara

$$\frac{1}{2} \cdot [1 \ 1]$$

obtendo a máscara

$$\frac{1}{4} \cdot [1 \ 2 \ 1]$$

que, obviamente, coincide com a máscara de ordem 3 calculada anteriormente para o filtro de Bartlett unidimensional. Repetindo a operação mais uma vez, obtemos a máscara

$$\frac{1}{8} \cdot [1 \ 3 \ 3 \ 1]$$

Por indução é fácil ver que a máscara unidimensional de ordem  $k + 1$  é dada por:

$$\frac{1}{2^k} \cdot \left[ 1 \ k \ \dots \ \binom{k}{i-1} \ \binom{k}{1} \ \binom{k}{i+1} \ \dots \ k \ 1 \right]$$

Esta máscara é chamada de **máscara binomial**. O filtro binomial é uma aproximação do filtro gaussiano unidimensional, e pode portanto ser utilizado como uma versão discreta do mesmo.

Usando o processo acima, juntamente com a separabilidade do filtro gaussiano, podemos facilmente obter máscaras para o filtro gaussiano bidimensional, de forma análoga ao que fizemos anteriormente para o filtro de Bartlett. Abaixo são apresentadas as máscaras de ordem 2, 3 e 4.

$$\frac{1}{4} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \frac{1}{64} \cdot \begin{bmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{bmatrix}$$

É claro que a máscara de ordem 3 coincide com a máscara de mesma ordem do filtro de Bartlett. No caso de ordem 5, temos a máscara unidimensional

$$\frac{1}{4} \cdot [1 \ 4 \ 6 \ 4 \ 1]$$

que permite calcular a máscara bidimensional, dada pela matriz

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

A imagem inferior da Figura 2.61 foi obtida da imagem superior por uma filtragem com um filtro gaussiano de ordem 5. Observe que a perda de altas frequências nessa imagem é bem mais acentuada do que na filtragem com o filtro de Bartlett, mostrado na Figura 2.60.

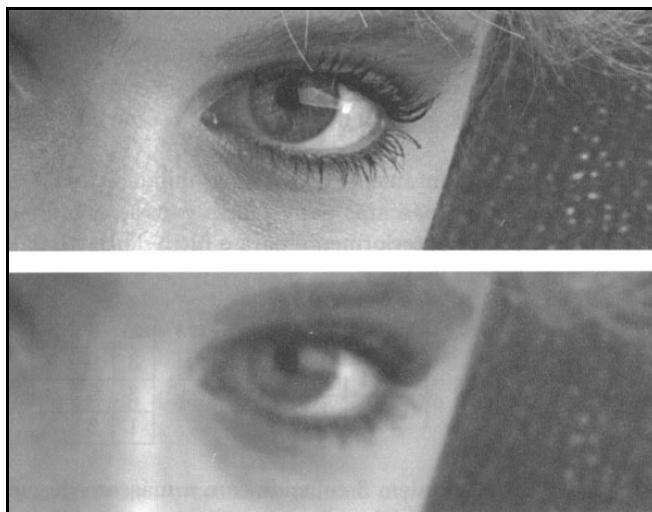


Figura 2.61 – Filtragem com um filtro Gaussiano de ordem 5

#### 2.10.4 Realce de Imagens no Domínio Espacial

O principal objetivo das técnicas de realce é o de destacar detalhes finos na imagem. São três os métodos de realce de imagens no domínio espacial, a saber: filtro passa-altas básico, realce por diferenciação e ênfase em alta frequência.

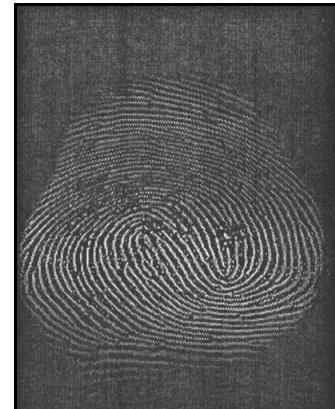
- Filtro Passa-alta Básico

O formato de resposta ao impulso de um filtro passa-alta (Figura 2.53) deve ser tal que a máscara correspondente apresente coeficientes positivos nas proximidades de seu centro e negativos longe dele. No caso de uma máscara  $3 \times 3$ , isso significa projetar uma máscara com pixel central positivo e todos os oito vizinhos, negativos. Um exemplo de máscara com estas características é apresentada a seguir. Pode-se notar que a soma algébrica dos coeficientes desta máscara é zero, significando que, quando aplicada a regiões homogêneas de uma imagem, o resultado será zero ou um valor muito baixo, o que é consistente como princípio da filtragem passa-alta. A Figura 2.62 mostra um exemplo de resultado de aplicação da máscara seguinte a uma imagem monocromática.

$$\frac{1}{9} \cdot \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



(a)



(b)

Figura 2.62 – (a) Imagem original; (b) imagem resultante após filtragem passa-alta

- Realce por Diferenciação

Sabendo-se que o cálculo da média dos pixels, em um trecho de imagem, produz como efeito a remoção de seus componentes de alta frequência e que o conceito de média é análogo à operação de integração, é razoável esperar que a diferenciação produza o efeito oposto e, portanto, enfatize os componentes de alta frequência presentes em uma imagem. O método mais usual de diferenciação, em aplicações de processamento de imagens é o gradiente. Em termos contínuos, o gradiente de  $f(x, y)$  em um certo ponto  $(x, y)$  é definido como o vetor:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

A magnitude deste vetor é dada por:

$$\nabla f = mag(\nabla f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

e é utilizada por várias técnicas de realce de imagens por diferenciação.

Para uma imagem digital, o gradiente pode ser aproximado por:

$$G[f(x, y)] \approx \sqrt{[f(x, y) - f(x+1, y)]^2 + [f(x, y) - f(x, y+1)]^2}$$

ou por

$$G[f(x, y)] \approx |f(x, y) - f(x+1, y)| + |f(x, y) - f(x, y+1)|$$

Outra aproximação, conhecida como gradiente de Roberts, utiliza as diferenças cruzadas, isto é, na diagonal:

$$G[f(x, y)] \approx \sqrt{[f(x, y) - f(x+1, y+1)]^2 + [f(x+1, y) - f(x, y+1)]^2}$$

ou

$$G[f(x, y)] \approx |f(x, y) - f(x+1, y+1)| + |f(x+1, y) - f(x, y+1)|$$

As equações acima podem ser implementadas usando máscaras de tamanho 2 x 2, como as mostradas a seguir, ou de dimensões 3 x 3, como por exemplo os operadores de Prewitt e Sobel, apresentados na seção “Operações de Convolução com Máscaras”.

$$\begin{array}{cc} \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} (a) \\ (b) \end{array}$$

Estas máscaras representam discretamente o gradiente no formato 2 x 2. (a) é a forma discreta do gradiente convencional e (b) é a forma discreta do gradiente de Roberts.

- Filtragem *high-boost*

A filtragem passa-alta também pode ser obtida subtraindo de uma imagem original uma versão filtrada por um filtro passa-baixa, ou seja:

$$\text{Passa-alta} = \text{Original} - \text{Passa-baixa}$$

O filtro ***high-boost*** ou técnica de ênfase em alta frequência, nada mais é que uma extensão da idéia original formulada acima, na qual a imagem original é multiplicada por um fator de amplificação ***A***:

$$\begin{aligned} \text{High-boost} &= (\text{A})(\text{Original}) - \text{Passa-baixa} \\ &= (\text{A} - 1)(\text{Original}) + \text{Original} - \text{Passa-baixa} \\ &= (\text{A} - 1)(\text{Original}) + \text{Passa-alta} \end{aligned}$$

Quando  $\text{A} = 1$ , o filtro se comporta de forma idêntica a um passa-alta. Nos casos em que  $\text{A} > 1$ , parte da imagem original é adicionada ao resultado, restaurando parcialmente os componentes de baixa frequência. O resultado é uma imagem que se parece com a original, com um grau relativo de realce das bordas, dependente do valor  $\text{A}$ . O processo genérico de subtração de uma imagem borrada

da imagem original é conhecido na literatura como ***unsharp masking***.

A ênfase em alta frequência pode ser implementada utilizando a máscara a seguir:

$$\frac{1}{9} \begin{bmatrix} -1 & -1 & -1 \\ -1 & w & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

onde  $w = 9A - 1$ . Com  $A \geq 1$ . A Figura 2.63 mostra o efeito da variação de  $A$  no resultado final da filtragem.

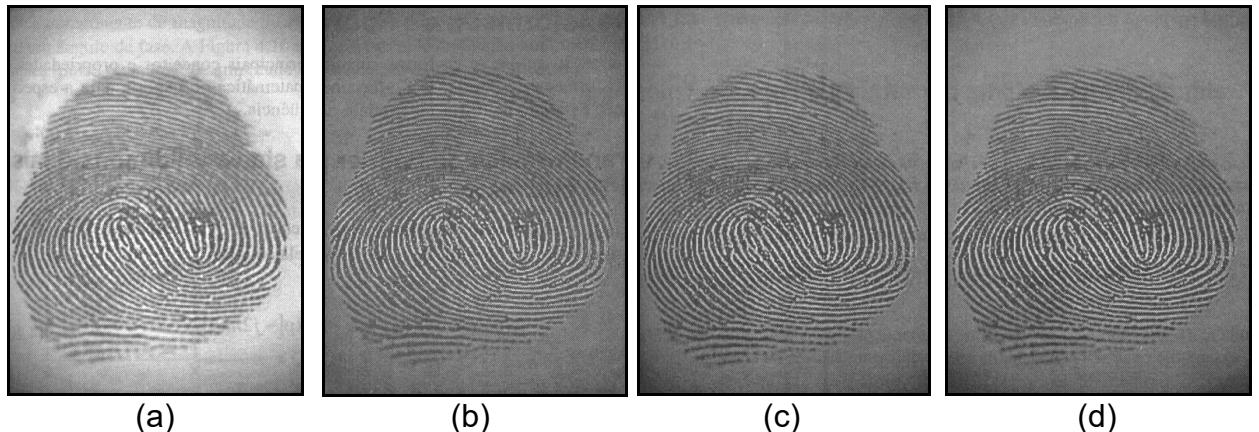


Figura 2.63 – (a) Imagem original;  
resultados da filtragem *high-boost* (b)  $A = 1,1$  (c)  $A = 1,15$  e (d)  $A = 1,2$

- Filtro Laplaciano

O ***laplaciano***  $\nabla$  de uma função duas vezes diferenciável é definido por

$$\nabla f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} .$$

O filtro laplaciano no domínio contínuo é um filtro de passa-alta. A forma discreta do laplaciano é dada por:

$$\begin{aligned} \nabla f(i, j) &= \Delta_x^2 f(i, j) + \Delta_y^2 f(i, j) \\ &= [f(i+1, j) + f(i-1, j) + f(i, j+1) + f(i, j-1)] - 4f(i, j) \end{aligned}$$

Uma máscara  $3 \times 3$  do filtro definido por esse operador é dada por

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Note que a menos do fator de proporcionalidade  $1/5$ , o lado direito da equação acima pode ser escrito na forma

$$f(i, j) - \frac{1}{5} [f(i+1, j) + f(i-1, j) + f(i, j) + f(i, j+1) + f(i, j-1)]$$

que é exatamente a diferença entre a imagem original e uma versão filtrada da imagem utilizando o filtro da média. Portanto o filtro laplaciano pode ser obtido, a menos de um fator constante, subtraindo da imagem original uma outra imagem obtida da imagem original atenuando as altas frequências. Esse resultado é importante tanto do ponto de vista de aplicações práticas como do ponto de vista conceitual. Segue-se daí, por exemplo, que o filtro laplaciano atenua as baixas frequências da imagem, e acentua as altas frequências, sendo portanto um filtro de passa-alta. O efeito desse filtro em uma imagem é mostrado na Figura 2.64.



Figura 2.64 – Filtragem Laplaciana de uma imagem

Nessa figura, a imagem inferior é obtida da superior por uma filtragem com um filtro laplaciano de ordem 3. Podemos observar que todos os detalhes de baixas frequências da imagem original (variações suaves) são perdidos na filtragem.

O resultado anterior sobre o filtro laplaciano nos leva a obter outros filtros de passa-alta com resultados melhores. Um desses filtros, que é bastante utilizado em visão computacional, consiste em utilizar inicialmente um filtro gaussiano de forma a atenuar as altas frequências da imagem, e em seguida aplicar o operador laplaciano. Uma máscara discreta que implementa esse filtro é dada por

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & -2 & -2 & 1 \\ 1 & -2 & -2 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Em termos gerais, a filtragem passa-alta realça detalhes, produzindo uma “agudização” (“sharpening”) da imagem, isto é, as transições entre regiões diferentes tornam-se mais nítidas. Estes filtros podem ser usados para realçar certas características presentes na imagem, tais como bordas, linhas curvas ou manchas, mas enfatizam o ruído existente na imagem. Alguns exemplos podem ser dados por:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

As três máscaras a seguir diferem quanto à intensidade de altos valores de níveis de cinza presentes na imagem resultante. A máscara alta deixa passar menos os baixos níveis de cinza, isto é, a imagem fica mais clara. A máscara baixa produz uma imagem mais escura que a anterior. A máscara média apresenta resultados intermediários.

Alta	Média	Baixa
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 3 & -2 \\ 1 & -2 & 1 \end{bmatrix}$

- Filtro de Roberts

Apresenta a desvantagem de certas bordas serem mais realçadas do que outras dependendo da direção, mesmo com magnitude igual. Como resultado de sua aplicação, obtém-se uma imagem com altos valores de nível de cinza, em regiões de limites bem definidos e valores baixos em regiões de limites suaves, sendo 0 para regiões de nível de cinza constante.

O operador consiste na função

$$a' = \sqrt{(a-d)^2 + (c-b)^2}$$

onde  $a'$  é o nível de cinza correspondente à localização  $a$ , a ser substituído;  $a, b, c, d$  são as localizações cujos valores serão computados para a operação.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$



Figura 2.65 – Efeito da aplicação do filtro de Roberts

- Filtro de Sobel

O filtro de Sobel realça linhas verticais e horizontais mais escuras que o

fundo, sem realçar pontos isolados. Consiste na aplicação de duas máscaras, descritas a seguir, que compõem um resultado único.

$$\mathbf{a} = \frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \mathbf{b} = \frac{1}{4} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

A máscara **a** detecta as variações no sentido horizontal e a máscara **b**, no sentido vertical. O resultado desta aplicação, em cada pixel, é dado por

$$a' = \sqrt{a^2 + b^2}$$

onde **a'** é o valor de nível de cinza correspondente à localização do elemento central da máscara.

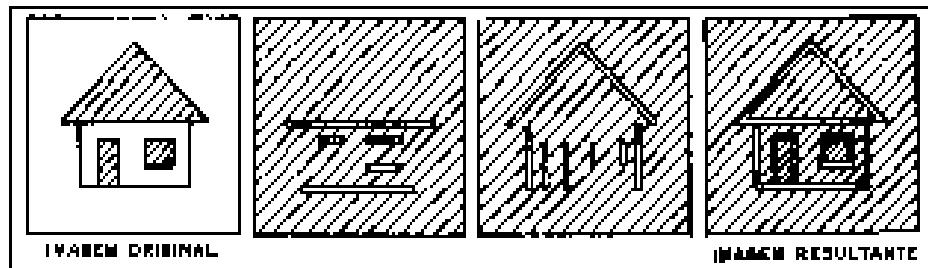


Figura 2.66 – Efeito da aplicação do filtro de Sobel

## UNIDADE 3 – MORFOLOGIA MATEMÁTICA

Assim como na biologia, onde a expressão morfologia se refere ao estudo da estrutura dos animais e plantas, a morfologia matemática, elaborada inicialmente por Georges Matheron e Jean Serra, concentra seus esforços no estudo da estrutura geométrica das entidades presentes em uma imagem. A morfologia matemática pode ser aplicada em várias áreas de processamento e análise de imagens, com objetivos tão distintos como realce, filtragem, segmentação, detecção de bordas, esqueletização, afinamento, dentre outras.

O princípio básico da morfologia matemática consiste em extrair as informações relativas à geometria e à topologia de um conjunto desconhecido (uma imagem), pela transformação através de outro conjunto completamente definido, chamado elemento estruturante. Portanto, a base da morfologia matemática é a teoria de conjuntos. Por exemplo, o conjunto de todos os pixels pretos em uma imagem binária descreve completamente a imagem. Em imagens binárias, os conjuntos em questão são membros do espaço inteiro bidimensional  $\mathbb{Z}^2$ , onde cada elemento do conjunto é um vetor 2D cujas coordenadas são as coordenadas  $(x, y)$  do pixel preto (por convenção) na imagem. Imagens com mais níveis de cinza podem ser representadas por conjuntos cujos elementos estão no espaço  $\mathbb{Z}^3$ . Neste caso, os vetores têm três elementos, sendo os dois primeiros as coordenadas do pixel e o terceiro seu nível de cinza.

### 3.1 DILATAÇÃO E EROSÃO

Iniciaremos nossa discussão de operações morfológicas pelas duas operações básicas: dilatação e erosão. Para bem compreender-las, inicialmente apresentaremos algumas definições úteis da teoria de conjuntos.

#### 3.1.1 Definições Básicas

Sejam  $A$  e  $B$  conjuntos em  $\mathbb{Z}^2$ , cujos componentes são  $a = (a_1, a_2)$  e  $b = (b_1, b_2)$ , respectivamente. A translação de  $A$  por  $x = (x_1, x_2)$ , denotada por  $(A)_x$ , é definida como:

$$(A)_x = \{c \mid c = a + x, \text{ para } a \in A\}$$

A reflexão de  $B$ , denotada por  $\hat{B}$ , é definida como:

$$\hat{B} = \{x \mid x = -b, \text{ para } b \in B\}$$

O complemento do conjunto  $A$  é:

$$A^c = \{x \mid x \notin A\}$$

Finalmente, a diferença entre dois conjuntos  $A$  e  $B$ , denotada por  $A - B$ , é definida como:

$$A - B = \{x \mid x \in A, x \notin B\} = A \cap B^c$$

A Figura 3.1 ilustra geometricamente as definições apresentadas, onde os pontos pretos identificam a origem do par de coordenadas. A Figura 3.1(a) mostra o conjunto **A**. A parte (b) mostra a translação de **A** por  $x = (x_1, x_2)$ . O conjunto **B** é exibido na parte (c), enquanto (d) mostra sua reflexão em relação à origem. Finalmente, a parte (e) apresenta o conjunto **A** e seu complemento, enquanto a Figura 3.1(f) mostra a diferença entre este conjunto **A** e o conjunto **B**.

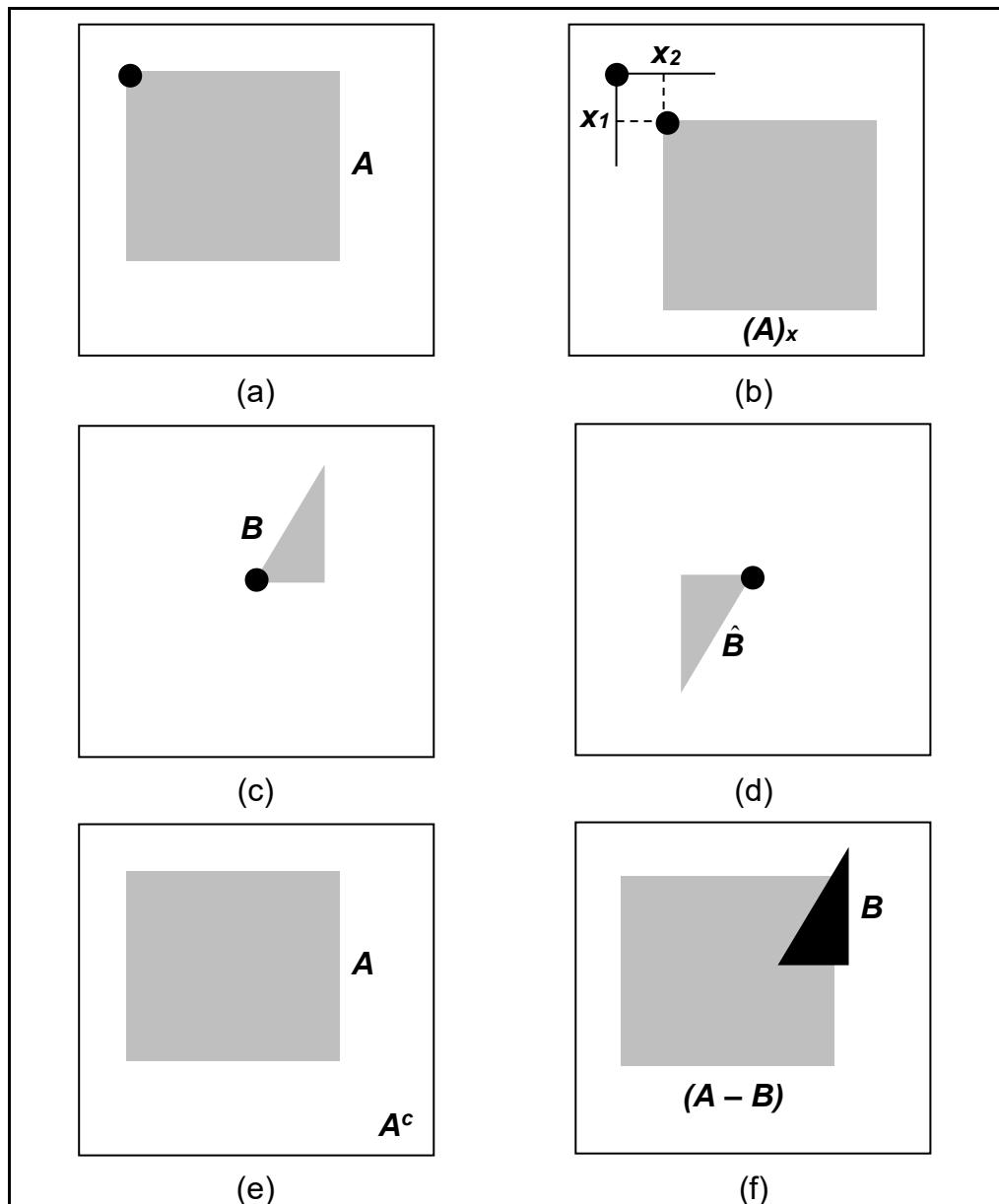


Figura 3.1 – Exemplos de operações básicas sobre conjuntos

### 3.1.2 Dilatação

Sejam **A** e **B** conjuntos no espaço  $\mathbf{Z}^2$  e seja  $\phi$  o conjunto vazio. A dilatação de **A** por **B**, denotada por  $\mathbf{A} \oplus \mathbf{B}$ , é definida como:

$$\mathbf{A} \oplus \mathbf{B} = \{x \mid (\hat{\mathbf{B}})_x \cap \mathbf{A} \neq \phi\}$$

Portanto, o processo de dilatação consiste em obter a reflexão de **B** sobre

sua origem e depois deslocar esta reflexão  $\mathbf{x}$ . A dilatação de  $\mathbf{A}$  por  $\mathbf{B}$  é, então, o conjunto de todos os  $\mathbf{x}$  deslocamentos para os quais a interseção de  $(\hat{\mathbf{B}})_x \cap \mathbf{A}$  inclui pelo menos um elemento diferente de zero. Com base nesta interpretação, a equação anterior pode ser escrita como:

$$\mathbf{A} \oplus \mathbf{B} = \{x \mid [(\hat{\mathbf{B}})_x \cap \mathbf{A}] \subseteq \mathbf{A}\}$$

O conjunto  $\mathbf{B}$  é normalmente denominado elemento estruturante. A Figura 3.2 mostra os efeitos da dilatação de um conjunto  $\mathbf{A}$  usando três elementos estruturantes ( $\mathbf{B}$ ) distintos. Observar que as operações morfológicas são sempre referenciadas a um elemento do conjunto estruturante (neste caso, o elemento central).

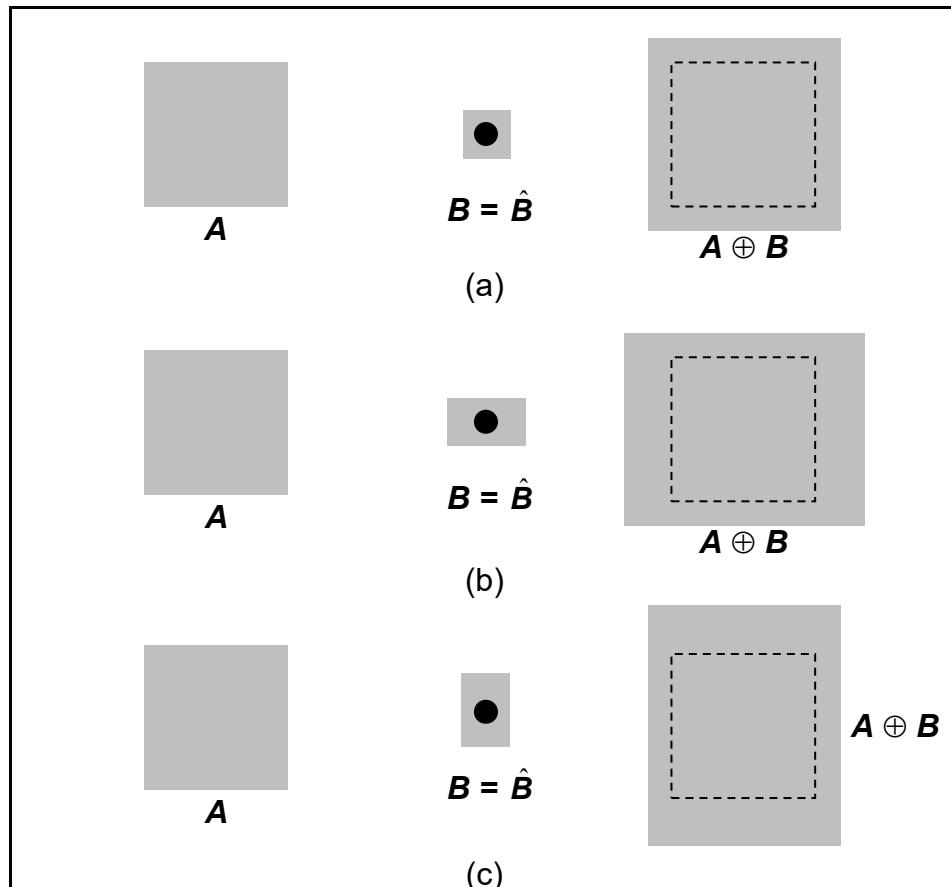


Figura 3.2 – Dilatação

### 3.1.3 Erosão

Sejam  $\mathbf{A}$  e  $\mathbf{B}$  conjuntos no espaço  $\mathbf{Z}^2$ . A erosão de  $\mathbf{A}$  por  $\mathbf{B}$ , denotada por  $\mathbf{A} \ominus \mathbf{B}$ , é definida como:

$$\mathbf{A} \ominus \mathbf{B} = \{x \mid (\mathbf{B})_x \subseteq \mathbf{A}\}$$

o que, em outras palavras, significa dizer que a erosão de  $\mathbf{A}$  por  $\mathbf{B}$  resulta no conjunto de pontos  $\mathbf{x}$  tais que  $\mathbf{B}$ , transladado de  $\mathbf{x}$ , está contido em  $\mathbf{A}$ .

A Figura 3.3 mostra os efeitos da erosão de um conjunto  $A$  usando três elementos estruturantes ( $B$ ) distintos.

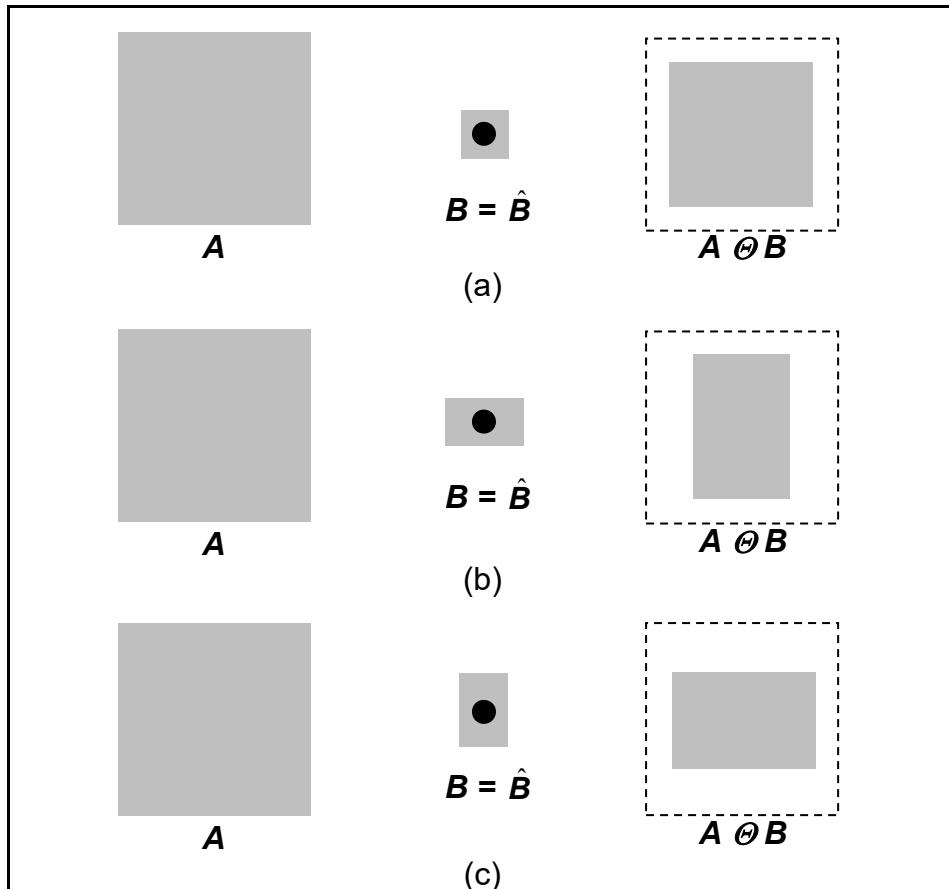


Figura 3.3 – Erosão

A dilatação e a erosão são operações duais entre si com respeito a complementação e reflexão. Ou seja,

$$(A \ominus B)^c = A^c \oplus \hat{B}$$

A prova desta dualidade está demonstrada a seguir. Partindo da definição de erosão, temos:

$$(A \ominus B)^c = \{x \mid (B)_x \subseteq A\}^c$$

Se o conjunto  $(B)_x$  está contido no conjunto  $A$ , então  $(B)_x \cap A^c = \emptyset$ . Portanto, a equação anterior torna-se:

$$(A \ominus B)^c = \{x \mid (B)_x \cap A^c = \emptyset\}^c$$

Porém, o complemento do conjunto dos  $x$ 's que satisfazem  $(B)_x \cap A^c = \emptyset$  é o conjunto dos  $x$ 's tais que  $(B)_x \cap A^c \neq \emptyset$ . Logo,

$$(A \ominus B)^c = \{x \mid (B)_x \cap A^c \neq \emptyset\}$$

$$(A \ominus B)^c = A^c \oplus \hat{B}$$

como queríamos demonstrar.

### 3.2 ABERTURA E FECHAMENTO

Como vimos nas figuras da seção anterior, a dilatação expande uma imagem, enquanto a erosão a encolhe. Nesta seção discutiremos duas outras importantes operações morfológicas: a abertura e o fechamento.

A abertura, em geral, suaviza o contorno de uma imagem, quebra istmos estreitos e elimina proeminências delgadas. O fechamento, por sua vez, funde pequenas quebras e alonga os golfos finos, elimina pequenos orifícios e preenche gaps no contorno.

A abertura de um conjunto  $\mathbf{A}$  por um elemento estruturante  $\mathbf{B}$ , denotada por  $\mathbf{A} \circ \mathbf{B}$ , é definido como:

$$\mathbf{A} \circ \mathbf{B} = (\mathbf{A} \ominus \mathbf{B}) \oplus \mathbf{B}$$

o que equivale a dizer que a abertura de  $\mathbf{A}$  por  $\mathbf{B}$  é simplesmente a erosão de  $\mathbf{A}$  por  $\mathbf{B}$  seguida de uma dilatação do resultado por  $\mathbf{B}$ .

O fechamento do conjunto  $\mathbf{A}$  pelo elemento estruturante  $\mathbf{B}$ , denotado por  $\mathbf{A} \bullet \mathbf{B}$ , é definido como:

$$\mathbf{A} \bullet \mathbf{B} = (\mathbf{A} \oplus \mathbf{B}) \ominus \mathbf{B}$$

o que nada mais é que a dilatação de  $\mathbf{A}$  por  $\mathbf{B}$  seguida da erosão do resultado pelo mesmo elemento estruturante  $\mathbf{B}$ .

A Figura 3.4 mostra um exemplo de operação de abertura, enquanto a Figura 3.5 mostra um exemplo de operação de fechamento. Ambas utilizam um elemento estruturante circular. A Figura 3.4 mostra a operação de abertura, indicando no alto o conjunto original  $\mathbf{A}$ , na linha intermediária, a etapa de erosão e na linha inferior, o resultado da operação de dilatação aplicada ao conjunto resultante da erosão. Na Figura 3.5 são detalhadas as operações de dilatação do conjunto original  $\mathbf{A}$  e subsequente erosão do resultado.

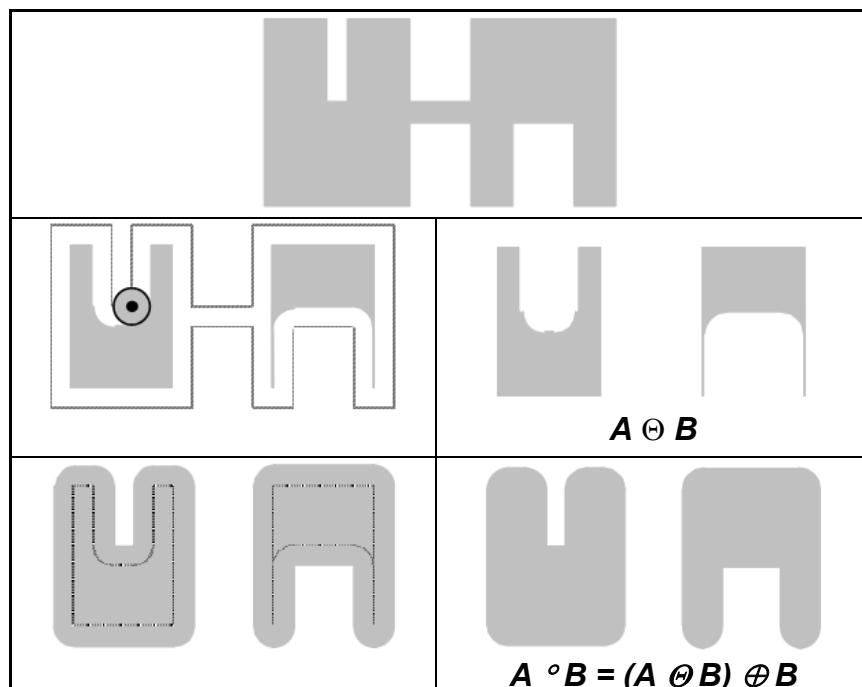


Figura 3.4 – Exemplo de abertura

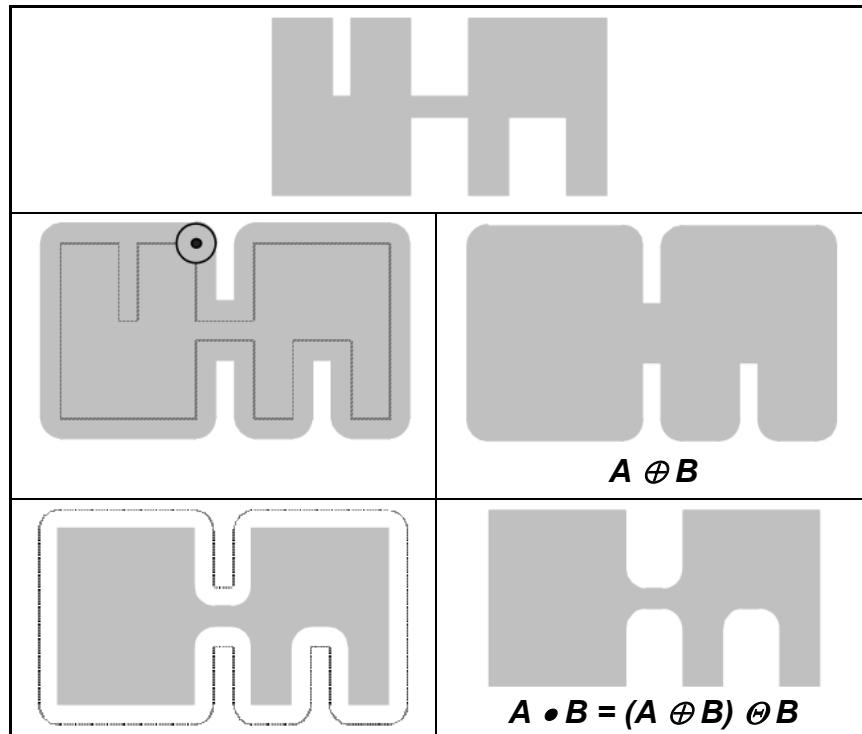


Figura 3.5 – Exemplo de fechamento

### 3.2.1 Interpretação geométrica da abertura e do fechamento

A abertura e o fechamento podem ser interpretados geometricamente de maneira simples. Suponha-se, por exemplo, o elemento estruturante circular  $\mathbf{B}$  da Figura 3.5 como um disco plano. A fronteira de  $\mathbf{A} \circ \mathbf{B}$  é composta pelos pontos da fronteira de  $\mathbf{B}$  que se distanciam mais para dentro da fronteira de  $\mathbf{A}$ , à medida que  $\mathbf{B}$  é girado em torno da parte interna desta fronteira. Esta propriedade geométrica de “encaixe” da operação de abertura pode ser expressa em termos da teoria de conjuntos como:

$$\mathbf{A} \circ \mathbf{B} = \cup \{(\mathbf{B})_x \mid (\mathbf{B})_x \subset \mathbf{A}\}$$

A Figura 3.6 mostra este conceito com um elemento estruturante de outro formato.

De maneira similar, a operação de fechamento pode ser interpretada geometricamente, supondo que o disco desliza pela parte externa da fronteira de  $\mathbf{A}$ . Geometricamente, um ponto  $\mathbf{z}$  é um elemento de  $\mathbf{A} \bullet \mathbf{B}$  se e somente se  $(\mathbf{B})_x \cap \mathbf{A} \neq \emptyset$  para qualquer translação de  $(\mathbf{B})$  que contenha  $\mathbf{z}$ . A Figura 3.7 mostra esta propriedade.

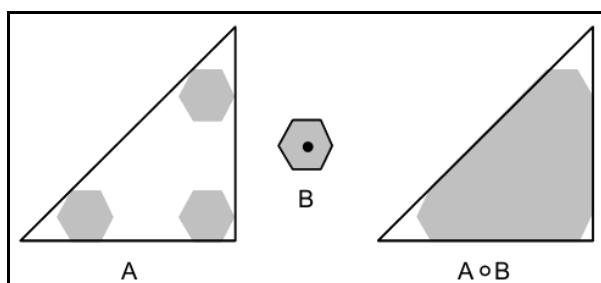


Figura 3.6 – Propriedade de ‘encaixe’ da abertura

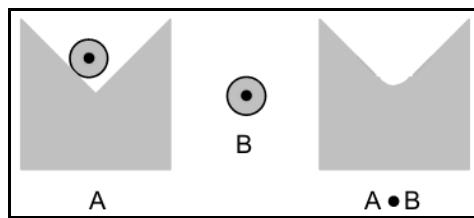


Figura 3.7 – Interpretação geométrica do fechamento

Assim como no caso da dilatação e erosão, a abertura e o fechamento são duais, ou seja:

$$(A \bullet B)^c = (A^c \circ \hat{B})$$

- Propriedades da abertura

- $A \circ B$  é um subconjunto (sub-imagem) de  $A$ ;
- Se  $C$  é um subconjunto de  $D$ , então  $C \circ B$  é um subconjunto de  $D \circ B$ ;
- $(A \circ B) \circ B = A \circ B$ .

- Propriedades do fechamento

- $A$  é um subconjunto  $A \bullet B$ ;
- Se  $C$  é um subconjunto de  $D$ , então  $C \bullet B$  é um subconjunto de  $D \bullet B$ ;
- $(A \bullet B) \bullet B = A \bullet B$ .

Estas propriedades auxiliam na interpretação dos resultados obtidos, quando as operações de abertura e fechamento são utilizadas para construir filtros morfológicos. Para um exemplo de filtro morfológico, na Figura 3.8 apresentamos uma imagem de um objeto retangular com ruído à qual se aplica o filtro  $(A \circ B) \bullet B$ .

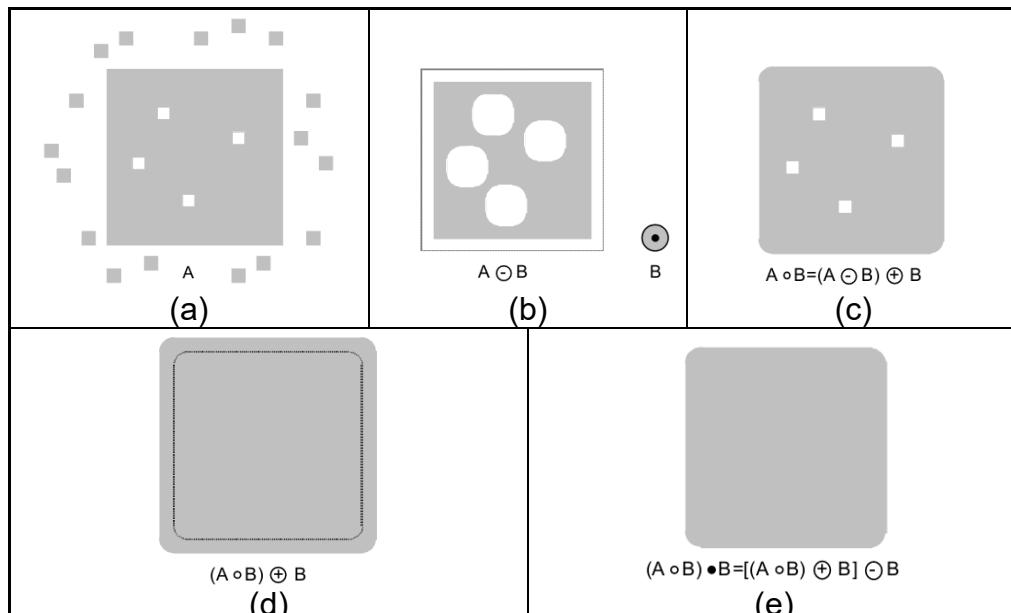


Figura 3.8 – Filtro morfológico: (a) imagem original, ruidosa; (b) resultado da erosão; (c) abertura de  $A$ ; (d) resultado de uma operação de dilatação aplicada à imagem (c); (e) resultado final

Após a operação de abertura, os pontos ruidosos externos ao objeto já foram removidos. A etapa de fechamento remove os pixels ruidosos do interior do objeto. Convém observar que o sucesso desta técnica depende do elemento estruturante ser maior que o maior aglomerado de pixels ruidosos conectados presente na imagem original.

### 3.3 TRANSFORMAÇÃO HIT-OR-MISS

A transformação morfológica *hit-or-miss* é uma ferramenta básica para o reconhecimento de padrões. Na Figura 3.9 se vê um conjunto  $\mathbf{A}$  que consiste de três padrões (subconjuntos),  $\mathbf{X}$ ,  $\mathbf{Y}$  e  $\mathbf{Z}$ . O sombreado das partes (a)-(c) indica os conjuntos originais, enquanto que as áreas sombreadas das partes (d) e (e) da figura indicam os resultados das operações morfológicas. Seja o objetivo: buscar a localização de um dos objetos de  $\mathbf{A}$ , por exemplo,  $\mathbf{Y}$ .

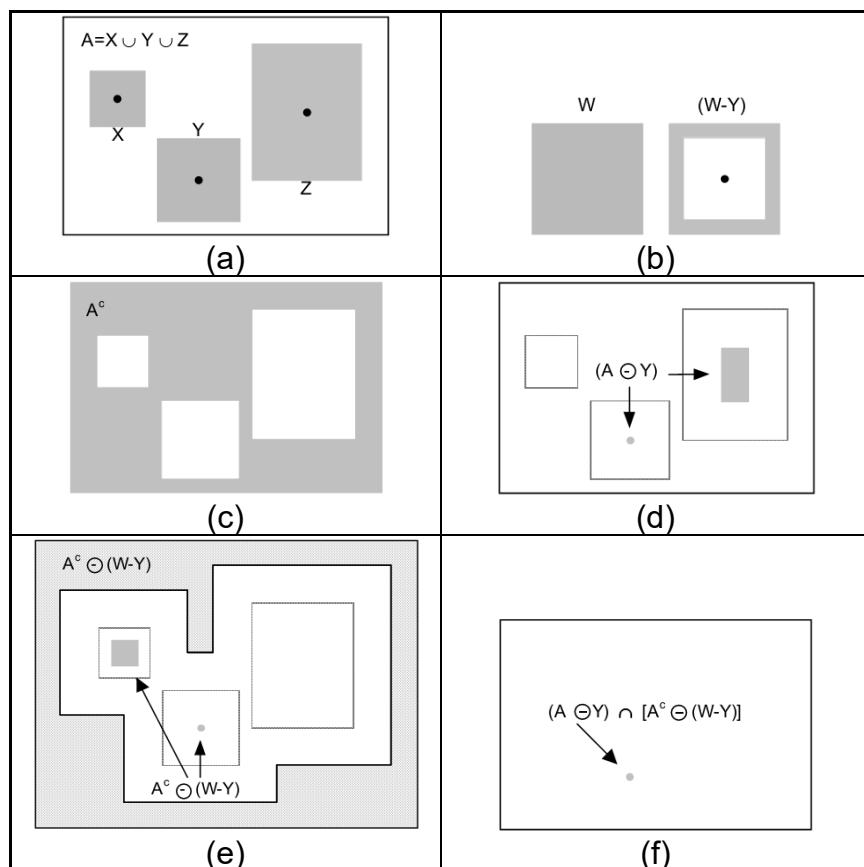


Figura 3.9 – Transformação *hit-or-miss*

Seja a origem de cada forma localizada em seu centro de gravidade. Se circundarmos  $\mathbf{Y}$  com uma pequena janela  $\mathbf{W}$ , o “fundo local” de  $\mathbf{Y}$  com respeito a  $\mathbf{W}$  será o conjunto diferença  $(\mathbf{W} - \mathbf{Y})$  mostrado na parte (b). A parte (c) da figura mostra o complemento de  $\mathbf{A}$  por  $\mathbf{Y}$ . A parte (e) mostra a erosão do complemento de  $\mathbf{A}$  pelo conjunto fundo-local  $(\mathbf{W} - \mathbf{Y})$ ; a região sombreada externa é parte da erosão. Notar que o conjunto dos lugares, para os quais  $\mathbf{Y}$  cabe exatamente dentro de  $\mathbf{A}$ , é a interseção da erosão de  $\mathbf{A}$  por  $\mathbf{Y}$  e a erosão de  $\mathbf{A}^c$  por  $(\mathbf{W} - \mathbf{Y})$ , como mostra a parte (f). Esta interseção é precisamente o lugar que se está buscando. Em outras palavras, se  $\mathbf{B}$  denota o conjunto composto por  $\mathbf{X}$  e seu fundo, o encaixe de  $\mathbf{B}$  em  $\mathbf{A}$ , denotado  $\mathbf{A}$  hom  $\mathbf{B}$ , é:

$$A \text{ hom } B = (A \ominus Y) \cap [A^c \ominus (W - Y)]$$

Generalizando, pode-se fazer  $B = (B_1, B_2)$ , onde  $B_1$  é o conjunto dos elementos de  $B$  associados com um objeto e  $B_2$  o conjunto dos elementos de  $B$  associados com o fundo correspondente. No caso anterior,  $B_1 = Y$  e  $B_2 = (W - Y)$ . Com esta notação, pode-se escrever:

$$A \text{ hom } B = (A \ominus B_1) \cap [A^c \ominus B_2]$$

Usando a definição de diferenças de conjuntos e a relação dual entre erosão e dilatação podemos escrever:

$$A \text{ hom } B = (A \ominus B_1) - (A \oplus \hat{B}_2)$$

Logo, o conjunto  $A \text{ hom } B$  contém todos os pontos para os quais, simultaneamente,  $B_1$  encontrou uma correspondência (ou um *hit*) em  $A$  e  $B_2$  encontrou uma correspondência em  $A^c$ .

### 3.4 ALGORITMOS MORFOLÓGICOS BÁSICOS

Começaremos agora a tratar dos usos práticos da morfologia matemática em processamento de imagens. Quando se está trabalhando com imagens binarizadas, a principal aplicação da morfologia é extrair componentes da imagem que sejam úteis na representação e descrição de formatos. A seguir, serão apresentados algoritmos de extração de contornos, extração de componentes conectados, delimitação do casco convexo de um objeto e esqueletização de uma região. Também são apresentados algoritmos úteis para as etapas de pré ou pós-processamento, tais como os de afinamento (*thining*), preenchimento de regiões (*region filling*), espessamento (*thickening*) e poda (*pruning*).

#### 3.4.1 Extração de contornos

É possível extrair o contorno de um conjunto  $A$ , denotado por  $\beta(A)$ , executando a erosão de  $A$  por  $B$  e então calculando a diferença entre  $A$  e sua erosão. Isto é,

$$\beta(A) = A - (A \ominus B)$$

onde  $B$  é um elemento estruturante adequado.

A Figura 3.10 mostra a mecânica da extração de contornos. Na parte (a) tem-se o conjunto original, na parte (b) o elemento estruturante, em (c) o resultado da erosão e finalmente em (d) a diferença, que corresponde ao contorno de  $A$ .

#### 3.4.2 Preenchimento de Regiões (*Region Filling*)

Seja um contorno fechado  $A$ , que pode ser expresso como um conjunto contendo um subconjunto cujos elementos são pontos do contorno 8-conectados. Partindo de um ponto  $p$ , situado dentro do contorno, o que se deseja é preencher o interior desta região com 1's.

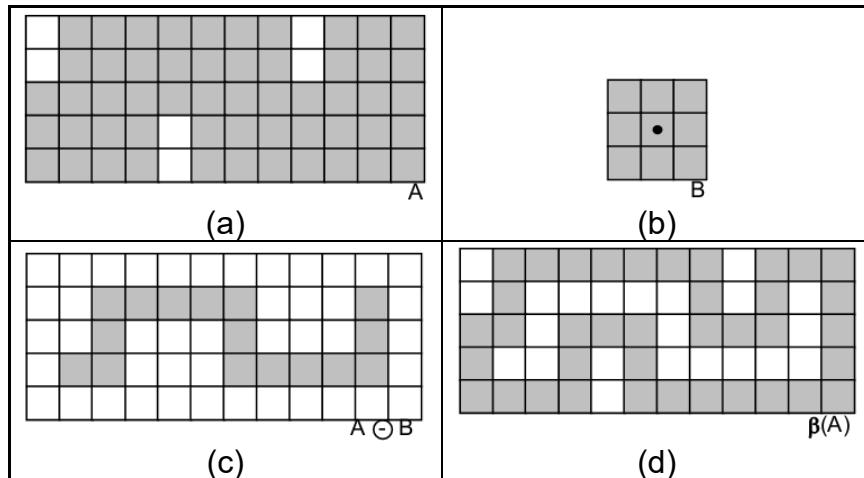
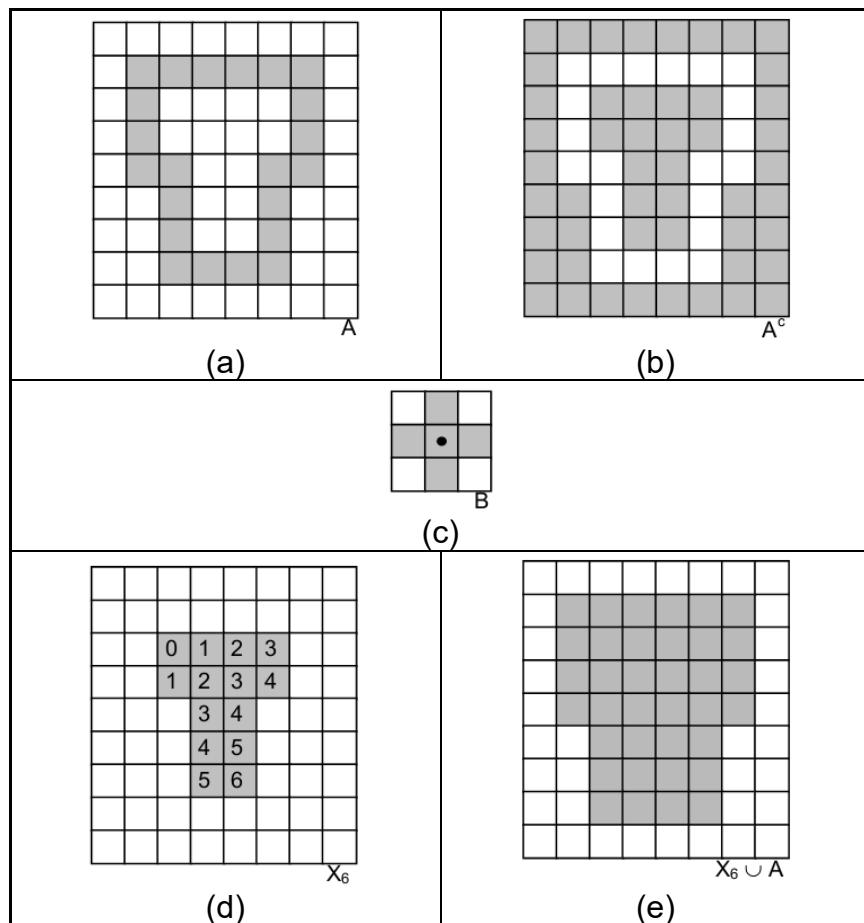


Figura 3.10 – Extração de contornos

Assumindo que todos os pontos, que não estão sobre a fronteira, estão rotulados como 0, atribuímos o valor 1 a  $p$  para iniciar o procedimento. O processo a seguir preenche a região com 1's:

$$X_k = (X_{k-1} \oplus B) \cap A^c \quad k = 1, 2, 3, \dots$$

onde  $X_0 = p$  e  $B$  é o elemento estruturante simétrico mostrado na parte (c) da Figura 3.11. O algoritmo termina na  $k$ -ésima iteração se  $X_k = X_{k-1}$ . O conjunto união de  $X_k$  e  $A$  contém a fronteira e os pontos internos a ela.

Figura 3.11 – Preenchimento de regiões (*Region filling*)

Este procedimento é ilustrado na Figura 3.11. Na parte (a) tem-se o conjunto original **A**, cujo complemento é mostrado em (b). A Figura 3.11(c) mostra o elemento estruturante utilizado. A parte (d) indica o resultado obtido após a sexta iteração (a última que ainda produziu alguma diferença em relação à iteração anterior), em que os números indicam que iteração contribuiu para o surgimento de quais pixels no resultado parcial. Finalmente, o resultado da união do conjunto da Figura 3.11(d) com o conjunto original é mostrado na parte (e).

### 3.4.3 Extração de Componentes Conectados

Seja **Y** um componente conectado contido em um conjunto **A** e suponha-se que um ponto **p** de **Y** é conhecido. Então, a expressão iterativa a seguir provê todos os elementos de **Y**:

$$\mathbf{X}_k = (\mathbf{X}_{k-1} \oplus \mathbf{B}) \cap \mathbf{A} \quad k = 1, 2, 3, \dots$$

onde  $\mathbf{X}_0 = \mathbf{p}$  e **B** é o elemento estruturante simétrico mostrado na parte (b) da Figura 3.12, que ilustra a mecânica da equação anterior. O algoritmo converge quando  $\mathbf{X}_k = \mathbf{X}_{k-1}$ . O valor final de  $\mathbf{X}_k$  será atribuído a **Y**.

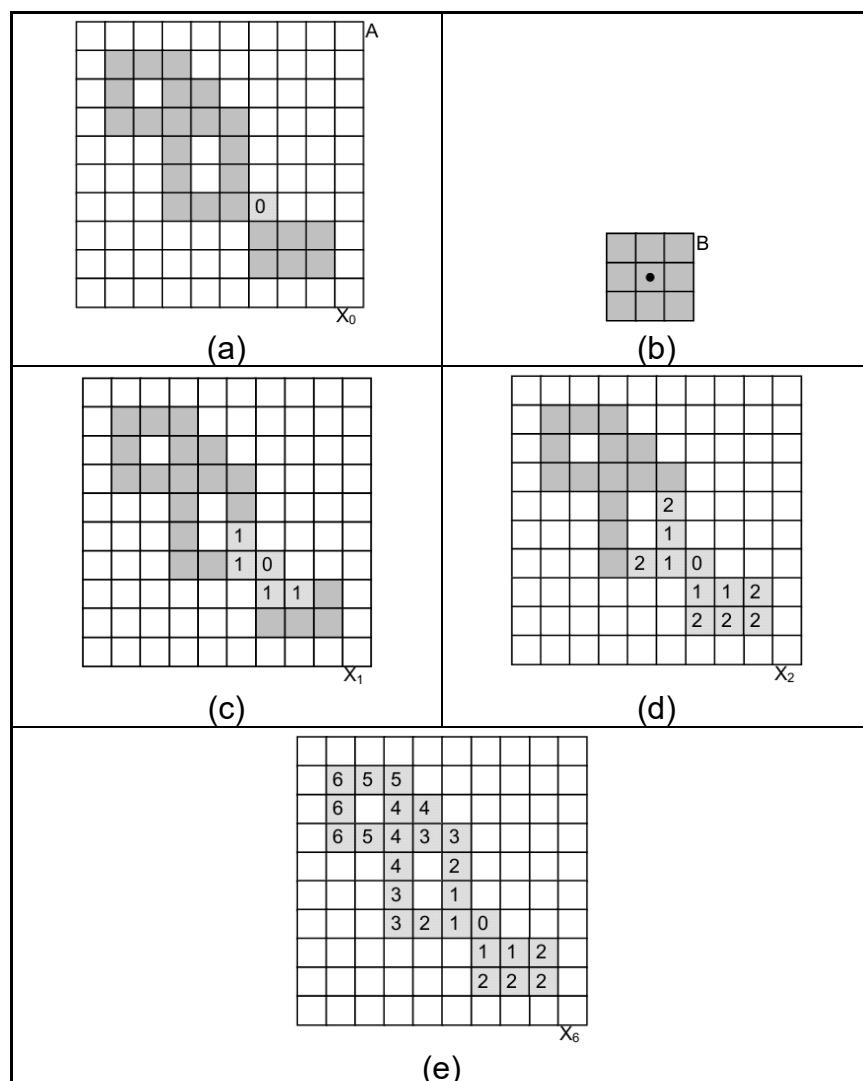


Figura 3.12 – Extração de componentes conectados

A Figura 3.12 mostra um exemplo de extração de componentes conectados. Em sua parte (a) são mostrados o conjunto original  $\mathbf{A}$  e o pixel de partida, indicado pelo número 0. O elemento estruturante utilizado está na Figura 3.12(b). As partes (c) e (d) mostram, respectivamente, os resultados após a primeira e a segunda iterações. O resultado final (após 6 iterações) é mostrado na Figura 3.12(e).

### 3.4.4 Casco Convexo (*Convex Hull*)

Define-se casco convexo  $\mathbf{H}$  de um conjunto arbitrário  $\mathbf{S}$  como o menor conjunto convexo que ainda contém  $\mathbf{S}$ . Apresentaremos, a seguir, um algoritmo baseado em morfologia matemática para a obtenção do casco convexo  $\mathbf{C}(\mathbf{A})$  de um conjunto  $\mathbf{A}$ . Seja  $\mathbf{B}^i$ ,  $i = 1, 2, 3$  e  $4$ , representando quatro elementos estruturantes. Notar que estes elementos possuem pontos indicados com  $X$  que significam uma condição “*don't care*”, quer dizer, o pixel naquela posição pode ter valor 0 ou 1. O procedimento consiste em implementar a equação:

$$\mathbf{X}'_k = (\mathbf{X} \text{ hom } \mathbf{B}^i) \cup \mathbf{A} \quad i = 1, 2, 3 \text{ e } 4 \quad \text{e} \quad k = 1, 2, 3, \dots$$

Com  $\mathbf{X}'_k = \mathbf{A}$ . Agora, seja  $\mathbf{D}^i = \mathbf{X}'_{k \text{ conv}}$ , onde o subscrito “conv” indica convergência no sentido que  $\mathbf{X}'_k = \mathbf{X}'_{k-1}$ . Então, o casco convexo de  $\mathbf{A}$  é:

$$\mathbf{C}(\mathbf{A}) = \bigcup_{i=1}^4 \mathbf{D}^i$$

Em outras palavras, o procedimento consiste em se aplicar iterativamente a transformada *hit-or-miss* sobre  $\mathbf{A}$  com  $\mathbf{B}^1$ ; quando não houverem mais mudanças, executa-se a união de  $\mathbf{A}$  e dá-se ao resultado o nome  $\mathbf{D}^1$ . O procedimento é repetido com  $\mathbf{B}^2$  até que não existam outras mudanças e assim sucessivamente. A união dos quatro  $\mathbf{D}$ 's resultantes constitui o casco convexo de  $\mathbf{A}$ .

A Figura 3.13 mostra as etapas deste procedimento, iniciando pela exibição dos quatro elementos estruturantes utilizados, na parte (a). A parte (b) mostra o conjunto original  $\mathbf{A}$ . As partes (c), (d), (e) e (f) mostram o resultado afinal do processamento para cada elemento estruturante, indicando numericamente a contribuição de cada iteração no resultado final para aquele elemento. O resultado final aparece na parte (g) e é detalhado na parte (h), que ilustra a contribuição de cada elemento estruturante para o resultado final. Nesta figura, a notação  $\mathbf{X}'_k$  indica a  $k$ -ésima iteração (aquele em que não houve mudança em relação a  $\mathbf{X}'_{k-1}$ ).

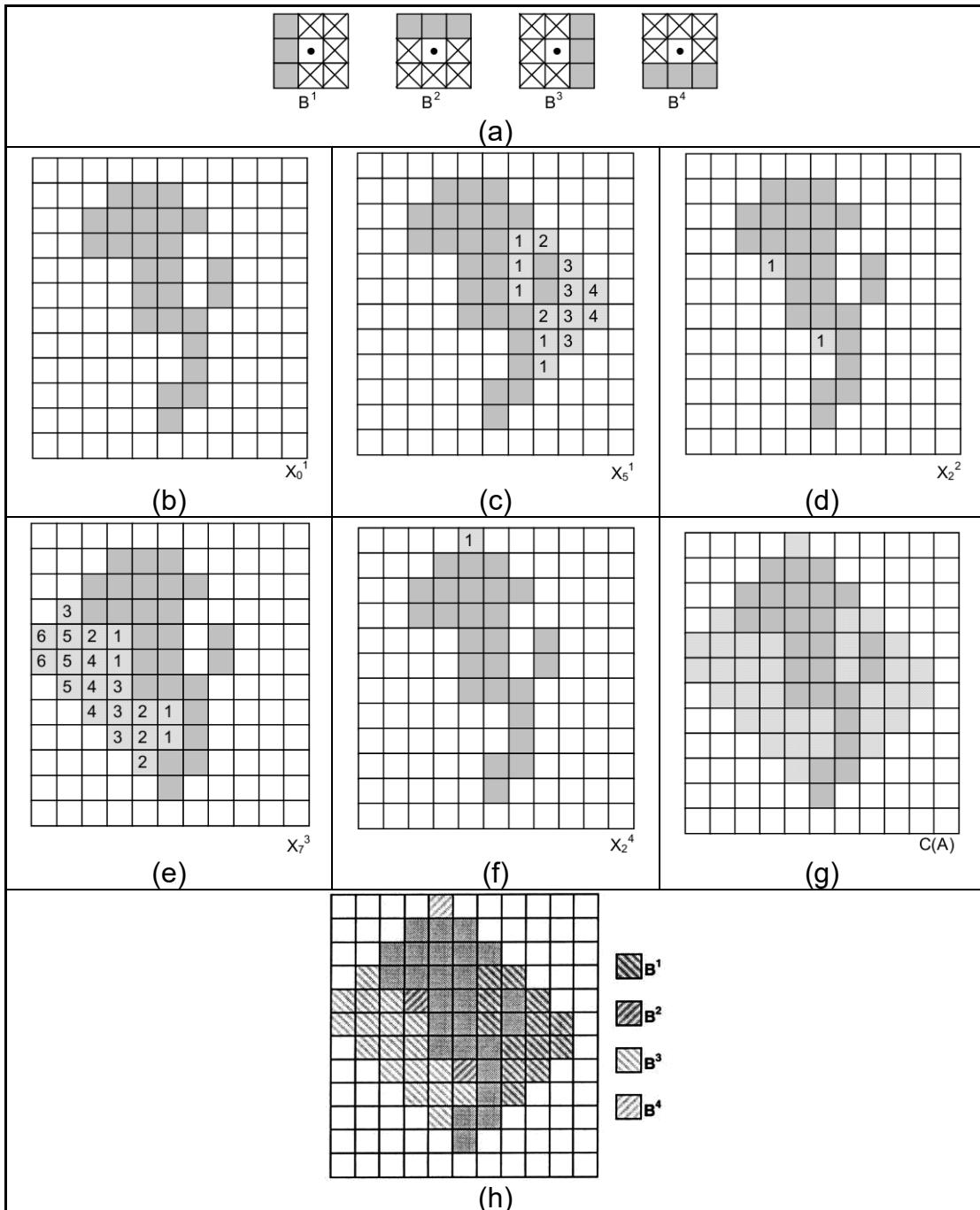
### 3.4.5 Afinamento (*Thinning*)

O afinamento de um conjunto  $\mathbf{A}$  por um elemento estruturante  $\mathbf{B}$ , denotado  $\mathbf{A} \otimes \mathbf{B}$ , pode ser definido com a ajuda da transformada *hit-or-miss*:

$$\begin{aligned} \mathbf{A} \otimes \mathbf{B} &= \mathbf{A} - (\mathbf{A} \text{ hom } \mathbf{B}) \\ \mathbf{A} \otimes \mathbf{B} &= \mathbf{A} \cap (\mathbf{A} \text{ hom } \mathbf{B})^c \end{aligned}$$

Outra expressão para o afinamento de  $\mathbf{A}$  é baseada em uma sequência de elementos estruturantes:

$$\{\mathbf{B}\} = \{\mathbf{B}^1, \mathbf{B}^2, \mathbf{B}^3, \dots, \mathbf{B}^n\}$$

Figura 3.13 – Casco convexo (*convex hull*)

onde  $\mathbf{B}^1$  é uma versão rotacionada de  $\mathbf{B}^{i-1}$ . Usando este conceito, define-se o afinamento por uma sequência de elementos estruturantes como:

$$\mathbf{A} \otimes \{\mathbf{B}\} = ((\dots((\mathbf{A} \otimes \mathbf{B}^1) \otimes \mathbf{B}^2)\dots) \otimes \mathbf{B}^n)$$

Em outras palavras, o processo consiste em afinar  $\mathbf{A}$  por um passo com  $\mathbf{B}^1$ , então afinar o resultado com um passo de  $\mathbf{B}^2$  e, assim sucessivamente, até que  $\mathbf{A}$  seja afinado com um passo de  $\mathbf{B}^n$ . O processo todo é repetido até que não ocorram mudanças.

A Figura 3.14 mostra todas as etapas de afinamento de uma imagem, usado oito elementos estruturantes, indicados no alto da figura. A partir do conjunto original  $\mathbf{A}$ , são ilustrados os resultados parciais mais relevantes, até a situação em

que nenhum elemento estruturante consiga remover nenhum outro pixel da imagem. Como este resultado ainda possui conexões diagonais redundantes entre pixels pretos, estas são removidas, utilizando o conceito de  $m$ -conectividade. Note que nesta figura, todos os pixels, não representados explicitamente, podem ser considerados brancos e que os números, dentro de cada pixel, indicam o elemento estruturante utilizado e não a iteração.

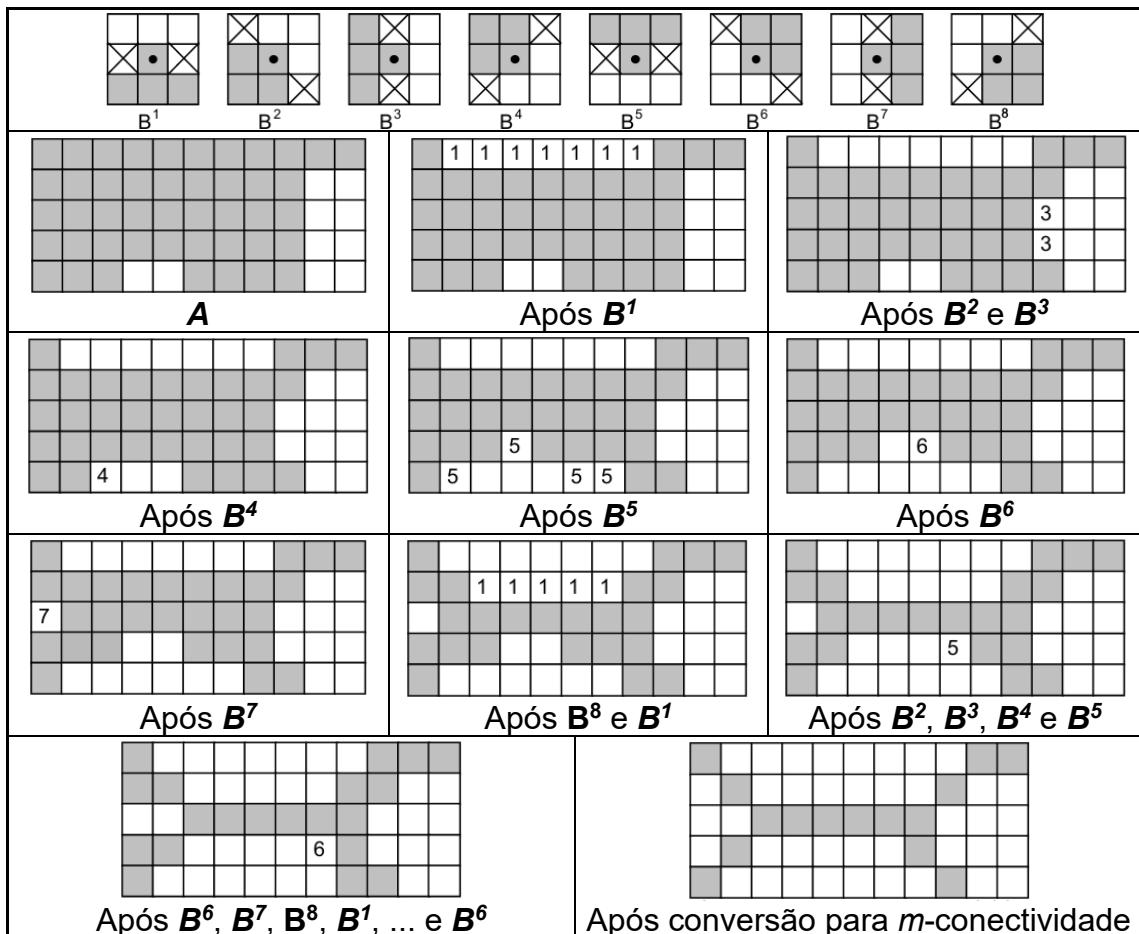


Figura 3.14 – Afinamento (*Thinning*)

### 3.4.6 Espessamento (*Thickening*)

O espessamento é o dual morfológico do afinamento e pode ser definido como:

$$A \text{ thi } B = A \cup (A \text{ hom } B)$$

onde  $B$  é um elemento estruturante adequado. O espessamento também pode ser definido como uma operação seqüencial:

$$A \text{ thi } \{B\} = ((\dots((A \text{ thi } B^1) \text{ thi } B^2)\dots) \text{ thi } B^n)$$

Os elementos estruturantes usados para o espessamento têm o mesmo formato dos usados para afinamento, porém com os 1's e 0's intercambiados. Entretanto, um algoritmo separado para espessamento, raramente é usado. O procedimento usual é afinar o fundo do conjunto e complementar o resultado. Ou

seja, para espessar o conjunto  $A$ , faz-se  $C = A^c$ , afina-se  $C$ , e então obtém-se  $C^c$ . A Figura 3.15 mostra o processo de espessamento, obtido pelo afinamento do complemento de  $A$ , onde a parte (a) representa a imagem original, a parte (b) seu complemento e a parte (c) a versão afinada do complemento de  $A$ . Na parte (d) da figura observa-se que este procedimento resulta em alguns pixels isolados, que são removidos em uma etapa complementar de processamento, como mostra a parte (e).

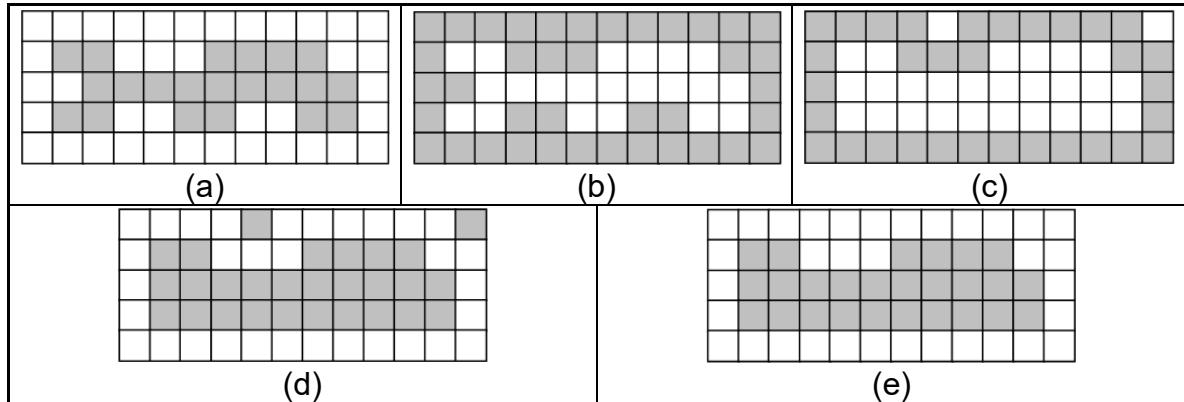


Figura 3.15 – Espessamento (*Thickening*) através do afinamento do fundo

### 3.4.7 Esqueletos

Uma abordagem importante para representar a forma estrutural de uma região plana é reduzi-la a um grafo. Esta redução pode ser implementada obtendo o esqueleto da região através de um algoritmo de afinamento (também denominado esqueletização).

O esqueleto de uma região pode ser definido usando a Transformação do Eixo Médio (MAT – *Medial Axis Trasformation*). A MAT é assim definida:

“A MAT de uma região  $R$  com fronteira  $B$  é obtida da seguinte forma: para cada ponto  $p$  em  $R$ , encontra-se seu vizinho mais próximo em  $B$ . Se  $p$  tem mais de um vizinho à mesma distância mínima, diz-se que  $p$  pertence ao eixo médio (esqueleto) de  $R$ ”. A Figura 3.16 mostra exemplos de MAT usando a distância euclidiana.

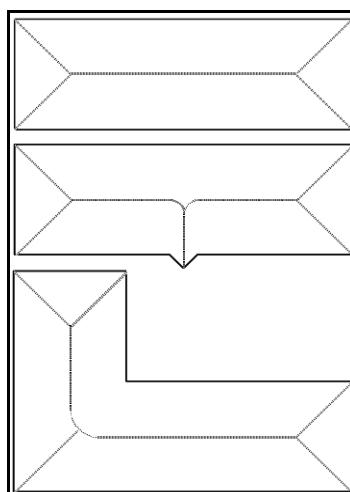


Figura 3.16 – Exemplos de esqueletos obtidos através do conceito da MAT para três regiões simples

A obtenção do esqueleto de um objeto plano também é possível através de técnicas morfológicas. O esqueleto de um conjunto (região)  $\mathbf{A}$  pode ser expresso em termos de erosões e aberturas. Isto é, sendo  $\mathbf{S}(\mathbf{A})$  o esqueleto de  $\mathbf{A}$ , pode-se provar que:

$$\mathbf{S}(\mathbf{A}) = \bigcup_{k=0}^K \mathbf{S}_k(\mathbf{A})$$

com

$$\mathbf{S}_k(\mathbf{A}) = \bigcup_{k=0}^K \{(A \ominus kB) - [(A \ominus kB) \circ B]\}$$

onde  $\mathbf{B}$  é um elemento estruturante,  $(\mathbf{A} \ominus k\mathbf{B})$  indica  $k$  erosões sucessivas de  $\mathbf{A}$ ; ou seja,

$$(\mathbf{A} \ominus k\mathbf{B}) = ((\dots(\mathbf{A} \ominus \mathbf{B}) \ominus \mathbf{B})\dots) \ominus \mathbf{B}$$

$k$  vezes, e  $K$  é o último passo iterativo antes de  $\mathbf{A}$  resultar, por erosão, em um conjunto vazio. Em outras palavras,

$$K = \text{Max}\{k \mid (\mathbf{A} \ominus k\mathbf{B}) \neq \emptyset\}$$

Estas equações indicam que  $\mathbf{S}(\mathbf{A})$ , o esqueleto de  $\mathbf{A}$ , pode ser obtido pela união dos subconjuntos esqueletos  $\mathbf{S}_k(\mathbf{A})$ . Pode-se mostrar, além disso, que  $\mathbf{A}$  pode ser reconstruído a partir destes subconjuntos utilizando a equação

$$\mathbf{A} = \bigcup_{k=0}^K (\mathbf{S}_k(\mathbf{A}) \oplus k\mathbf{B})$$

onde  $(\mathbf{S}_k(\mathbf{A}) \oplus k\mathbf{B})$  denota  $k$  dilatações sucessivas de  $\mathbf{S}_k(\mathbf{A})$ ; ou seja,

$$(\mathbf{S}_k(\mathbf{A}) \oplus k\mathbf{B}) = ((\dots(\mathbf{S}_k(\mathbf{A}) \oplus \mathbf{B}) \oplus \mathbf{B})\dots) \oplus \mathbf{B}$$

$k$  vezes, sendo o limite  $K$  de uniões o mesmo utilizado anteriormente.

A Figura 3.17 mostra estes conceitos. Nesta figura, a imagem original está na primeira posição da primeira coluna e sua versão afinada, na última posição da quarta coluna. Notar que o resultado não satisfaz os requisitos de um algoritmo de afinamento, que são:

- Não remover pontos terminais de um segmento;
- Não violar a conectividade da imagem original;
- Não causar erosão excessiva da região.

A imagem resultante tem dois graves problemas: é mais espessa do que deveria ser e, pior ainda, remove a conectividade dos elementos da imagem original. Por estes motivos, apesar da elegante formulação matemática dos algoritmos morfológicos, os algoritmos heurísticos muitas vezes produzem melhores resultados. A interseção da última linha com a última coluna contém o resultado da reconstrução do conjunto  $\mathbf{A}$ .

$\diagdown \backslash k$	$A \otimes kB$	$(A \otimes kB) \circ B$	$S_k(A)$	$\bigcup_{k=0}^K S_k(A)$	$S_k(A) \oplus kB$	$\bigcup_{k=0}^K S_k(A) \oplus kB$	
0							
1							
2							

Figura 3.17 – Esqueletos

### 3.4.8 Poda (*Pruning*)

Os métodos de poda são complementos essenciais dos algoritmos de afinamento, uma vez que estes, em geral, deixam componentes parasitas que devem ser removidos em uma etapa de pós-processamento. O conceito de poda será apresentado através de um exemplo. Seja um caractere, cujo esqueleto contém pontos espúrios, causados por diferentes espessuras nos traços do caractere original (antes do afinamento), mostrado na Figura 3.18(b). Assumindo que os componentes parasitas têm comprimento menor ou igual a três pixels, neste caso podem ser definidos dois elementos estruturantes (cada qual com suas versões rotacionadas de 90°) projetados para detectar pontos terminais, mostrados na parte (a). Notar que quatro destes elementos têm duas quadrículas indicadas com  $X$  que significam uma condição “*don't-care*”, quer dizer, o pixel naquela posição pode ter valor 0 ou 1. A primeira etapa será afinar o conjunto  $A$  com os elementos estruturantes  $B$ , obtendo o conjunto  $X_1$ , isto é:

$$X_1 = A \otimes \{B\}$$

onde  $\{B\}$  denota a sequência dos oito elementos estruturantes.

Aplicando a equação acima, por exemplo, três vezes (pois os ramos parasitas têm comprimento menor ou igual a 3 pixels), produz-se o resultado  $X_1$ , mostrado na Figura 3.18(c). O passo seguinte é “restaurar” o caractere à sua forma original, porém com os ramos parasitas removidos. Para fazê-lo, inicialmente se constrói o conjunto  $X_2$ , contendo todos os pontos terminais de  $X_1$  (Figura 3.18(d)):

$$X_2 = \bigcup_{k=1}^8 (X_1, hom B^k)$$

onde os  $B_k$ , são os mesmos detectores de pontos terminais usados anteriormente. O passo seguinte é a dilatação dos pontos terminais três vezes, usando o conjunto  $A$  como delimitador:

$$X_3 = (X_2 \oplus H) \cap A$$

onde  $H$  é um elemento estruturante de  $3 \times 3$  composto por 1's. Este tipo de dilatação previne a criação de pontos de valor 1 fora da região de interesse, como se pode comprovar na parte (e) da Figura 3.18. Finalmente, a união de  $X_3$  e  $X_1$  dá o resultado final (Figura 3.18(f)):

$$X_4 = X_1 \cup X_3$$

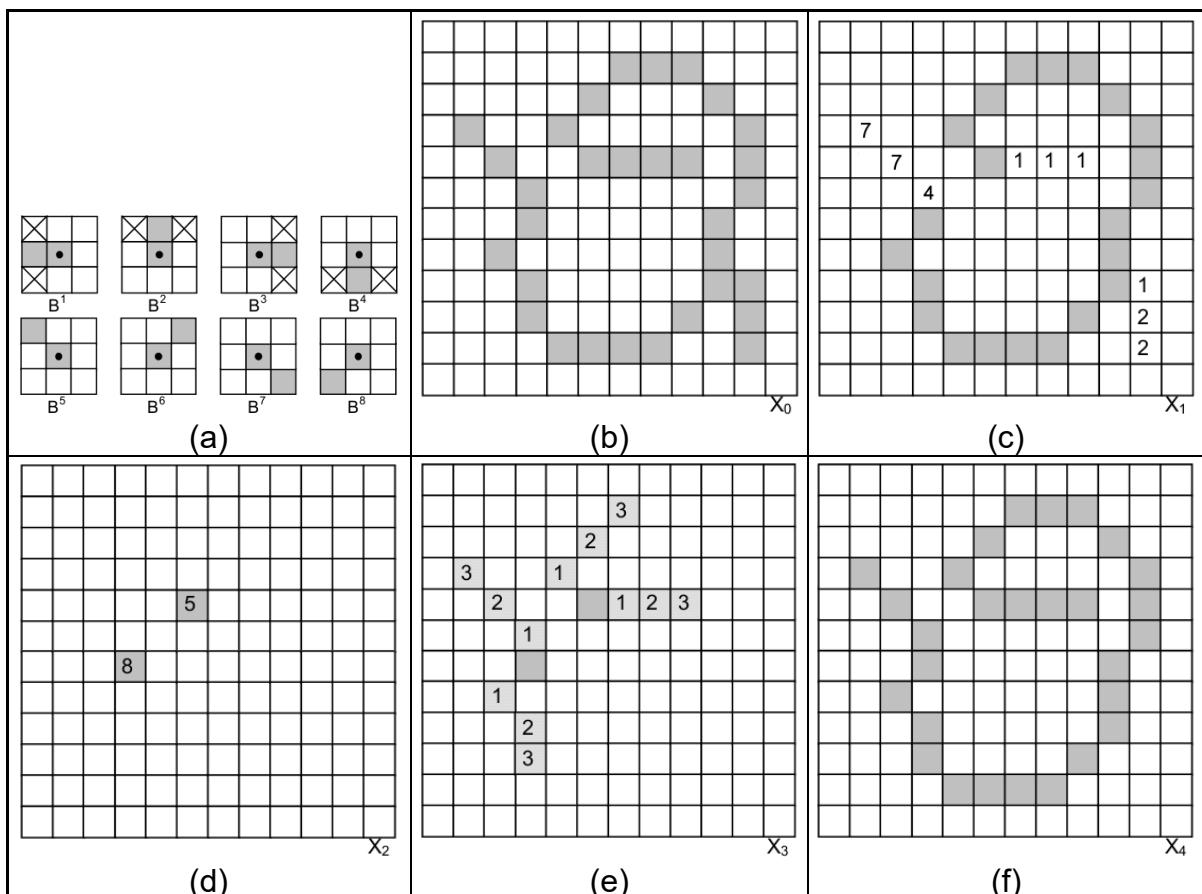


Figura 3.18 – Poda (*pruning*)

## UNIDADE 4 – COMBINAÇÃO ALGÉBRICA DE IMAGENS

A combinação de imagens digitais tem sido utilizada largamente para se obter efeitos especiais no cinema e na televisão. A indústria do cinema, em particular, se utiliza da combinação de imagens por processos analógicos desde a década de 20.

A operação de combinação tem como objetivo juntar, em uma única imagem, objetos apresentados separadamente em imagens distintas. No caso de imagens sintéticas podemos gerar separadamente os diferentes elementos, e depois combinar esses elementos em uma única imagem. Por outro lado, a combinação é o único método possível para juntar elementos de imagens geradas por processos distintos, por exemplo, cenas reais com cenas geradas no computador.

Com o avanço das técnicas de computação gráfica, de processamento e de análise de imagens, juntamente com a grande capacidade de processamento dos computadores atuais, cada vez mais vêm sendo utilizadas técnicas de combinação digital de imagens, tanto pela indústria cinematográfica como pela indústria de vídeo e televisão.

Para combinar imagens devemos definir uma operação, ***comp***, entre imagens que associa às imagens ***f*** e ***g***, uma imagem ***h = comp (f, g)***. Em algumas aplicações a operação ***comp*** deve ser estendida para ***n*** imagens, ***comp (f<sub>1</sub>, ..., f<sub>n</sub>)***. Porém, em geral, a combinação de ***n*** imagens se reduz a combinações sucessivas de duas imagens.

A operação ***comp (f, g)*** só está bem definida se as imagens ***f*** e ***g*** pertencem a um mesmo espaço de imagens, desse modo elas devem estar definidas no mesmo conjunto suporte e devem assumir valores no mesmo espaço de cor.

Uma imagem ***f*** pode ser obtida usando a combinação linear:

$$f = c_1 f_1 + c_2 f_2 + \dots + c_n f_n$$

A operação acima pode ser generalizada se tomarmos ao invés de ***n*** números reais, uma família de ***n*** funções reais ***α<sub>i</sub>***, definidas no mesmo conjunto suporte do espaço de imagens. A equação acima se escreve então na forma:

$$f(x, y) = \sum_{i=1}^n \alpha_i(x, y) f_i(x, y) = \alpha_1(x, y) f_1(x, y) + \dots + \alpha_n(x, y) f_n(x, y)$$

Para que a imagem ***f*** na equação acima esteja bem definida em geral, devemos supor que ***α<sub>i</sub>(x, y) ≥ 0***. Além disso, a equação acima deve calcular uma intensidade média da imagem em cada ponto, de forma a não resultar valores de intensidade de cor não compatíveis com a intensidade de cor em cada imagem. Para isso devemos exigir que

$$\alpha_1(x, y) + \dots + \alpha_n(x, y) = 1, \quad \forall (x, y) \in U$$

Dessa forma calculamos, em cada ponto ***(x, y)*** do domínio ***U***, uma média ponderada dos valores de cada imagem ***f<sub>i</sub>*** em ***(x, y)***.

As duas condições impostas às funções ***α<sub>i</sub>*** da família definem o que chamamos de partição da unidade. Ou sejam uma ***partição da unidade*** finita de um subconjunto ***U ⊂ ℝ<sup>m</sup>*** é uma família ***α<sub>i</sub>***, ***i = 1, 2, ..., n*** de funções ***α<sub>i</sub>: U → ℝ*** tais que,

para todo  $(x, y) \in U$ , tem-se:

- $\alpha_i(x, y) \geq 0$ ;
- $\alpha_1(x, y) + \alpha_2(x, y) + \dots + \alpha_n(x, y) = 1$ .

#### 4.1 MISTURA DE IMAGENS

Um caso particular importante de operação de combinação algébrica de imagens ocorre quando as funções  $\alpha_i$  da partição são constantes no domínio  $U$  do espaço de imagens, ou seja,  $\alpha_i(x, y) = c_i$ , para todo  $(x, y) \in U$ . Nesse caso a operação se reduz à combinação linear de imagens, e a operação é chamada de **mistura** de imagens: dadas duas imagens  $f$  e  $g$ , e um número real  $0 \leq t \leq 1$ , definimos o **dissolve**  $h_t$  de  $f$  e  $g$  pondo

$$h_t = \text{dissolve}_t(f, g) = (1 - t)f + tg$$

Se  $t = 0$  obtemos  $\text{dissolve}_0(f, g) = f$ , e para  $t = 1$  temos  $\text{dissolve}_1(f, g) = g$ . No caso geral, isto é  $0 < t < 1$ , a imagem  $h_t$  possui em cada ponto uma média ponderada das cores dos pontos correspondentes nas imagens  $f$  e  $g$ , além disso, os pesos dessa média são os mesmos em todos os pontos do domínio da imagem, uma vez que as funções da partição da unidade são constantes, e definidas por  $\alpha_1(x, y) = 1 - t$  e  $\alpha_2(x, y) = t$ . Temos portanto uma mistura de imagens conforme definimos acima.

A imagem (c) da Figura 4.1 mostra um dissolve das imagens (a) e (b), onde o valor do parâmetro  $t$  é 0,4.

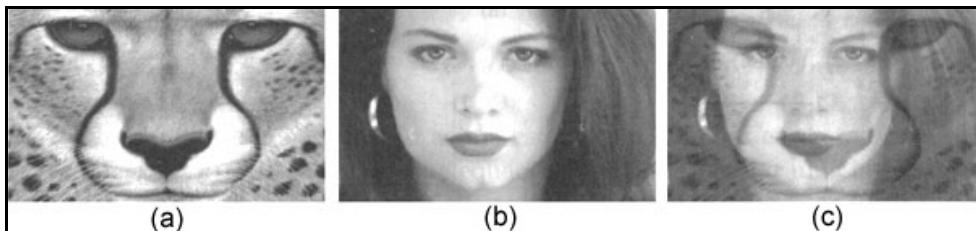


Figura 4.1 – Dissolve de duas imagens

#### 4.2 COMBINAÇÃO DE IMAGENS POR DECOMPOSIÇÃO

Na seção anterior vimos um método algébrico para obtermos uma determinada imagem a partir da combinação de um número finito de imagens. É muito comum em computação gráfica obtermos uma imagem fazendo combinação de elementos específicos de diferentes imagens, todas pertencentes a um mesmo espaço de imagens. Esse fato é ilustrado na Figura 4.2, onde fazemos uma superposição de um elemento da imagem (a), sobre a imagem (b), de modo a obter a imagem em (c).

O estudo desse método de construir uma imagem a partir de combinação de elementos de diversas outras imagens, se reduz ao estudo de decomposição do domínio  $U$  das imagens dadas. Com efeito, usando a decomposição do domínio  $U$  em dois conjunto  $\mathbf{A}$  e  $\mathbf{B}$  conforme mostramos na Figura 4.3, a imagem  $g$  da Figura 4.2(c), é obtida a partir da imagem  $f_1$  da Figura 4.2(a) e da imagem  $f_2$  da Figura

4.2(b), pondo

$$g(x, y) = \begin{cases} f_1(x, y) & \text{se } (x, y) \in A; \\ f_2(x, y) & \text{se } (x, y) \in B. \end{cases}$$



Figura 4.2 – Combinação com superposição de elementos

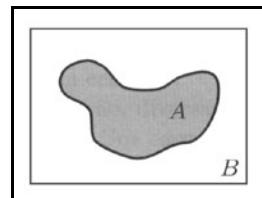


Figura 4.3 – Decomposição do domínio  $U$

Nas Figuras 4.4 (a) e (b), mostramos duas imagens  $f_1$  e  $f_2$ . Na Figura (c) mostramos uma decomposição do domínio  $U$  em três conjuntos  $A$ ,  $B$  e  $C$ . Na Figura (d) mostramos uma combinação  $f$  das imagens  $f_1$  e  $f_2$ . A combinação  $f$  é definida por

$$f(x, y) = \begin{cases} f_1(x, y) & \text{se } (x, y) \in A; \\ f_2(x, y) & \text{se } (x, y) \in B; \\ \text{dissolve}_{0,5}(f_1, f_2) & \text{se } (x, y) \in C. \end{cases}$$

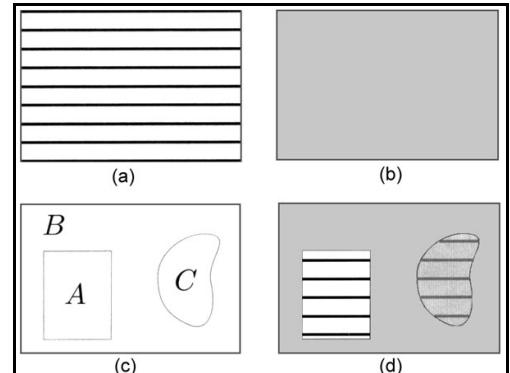


Figura 4.4 – Combinação de imagens

### 4.3 COMPOSIÇÃO DE IMAGENS

Perceptualmente, quando observamos uma imagem, ou mesmo uma cena real, identificamos dois tipos de regiões, que chamamos de **figura** e **fundo** (“foreground” e “background”). A **figura** corresponde a elementos distintos da imagem, em geral associados a objetos frontais da cena. O **fundo** corresponde ao suporte visual dos objetos em cena.

A identificação de regiões de figura e fundo em uma imagem define naturalmente uma decomposição do domínio  $U$  da imagem em dois conjuntos  $U_f$  e  $U_b$  que contêm respectivamente os elementos da figura e do fundo.

Existe um caso particular do método de combinar imagens por decomposição que tem grande interesse nas aplicações. Esse caso ocorre quando cada imagem é combinada através de uma decomposição figura-fundo. A essa

operação de combinação damos o nome de **composição de imagens**. Esse é um dos operadores de combinação mais utilizados em Computação Gráfica.

Um caso particular da operação de composição é definido pelo operador de **superposição de imagens**. Nesse caso, dadas duas imagens  $f_1$  e  $f_2$ , e uma decomposição figura-fundo  $\{U_f, U_b\}$  definida do domínio dessas duas imagens, a operação de superposição resulta em uma imagem  $f = \text{over}(f_1, f_2)$  definida por

$$f(x, y) = \begin{cases} f_1 & \text{se } (x, y) \in U_f; \\ f_2 & \text{se } (x, y) \in U_b. \end{cases}$$

Na Figura 4.5 mostramos a superposição de uma imagem sintética, isto é, gerada no computador, sobre uma imagem real de um ambiente que foi fotografado e digitalizado.

É claro que a operação de superposição de imagens não é comutativa em geral, uma vez que a ordem das imagens determina os elementos que vão compor os elementos da figura e do fundo da cena.

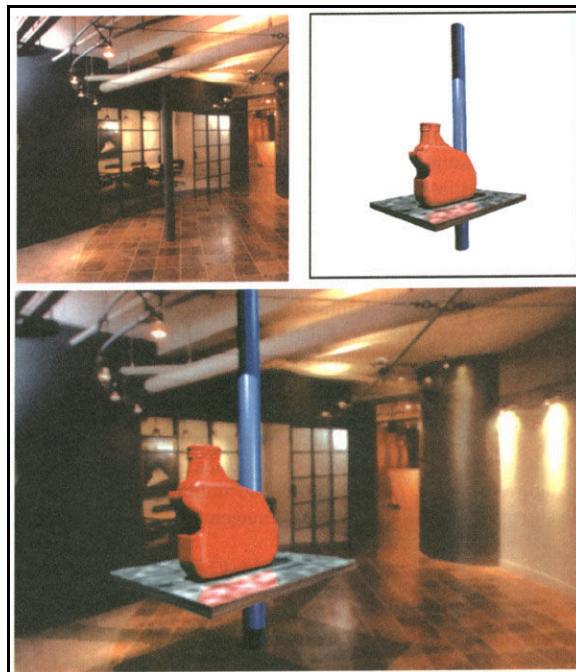


Figura 4.5 – Superposição de uma imagem sintética sobre uma imagem real

#### 4.4 COMBINAÇÃO NO DOMÍNIO DISCRETO

Quando fazemos uma mistura de imagens, as funções da partição são constantes em todo o domínio. Portanto, supondo que as imagens estão amostradas corretamente, não temos qualquer problema em trabalhar no domínio discreto. A operação se reduz no domínio discreto a uma operação pixel a pixel, usando o valor amostrado das imagens em cada pixel.

No caso da operação de composição, trabalhamos com a restrição das imagens aos conjuntos de uma decomposição do domínio. Portanto, ao discretizar o domínio da imagem devemos levar em consideração a geometria de cada conjunto da decomposição no interior do pixel.

Considere o operador de superposição  $h = \text{over}(f, g)$ , baseado em uma

decomposição figura-fundo do domínio. Trabalhando no domínio contínuo temos duas opções para o valor  $h(p) = \overline{\{f, g\}(p)}$  da imagem final:

- $h(p) = f(p)$ , ou seja, a cor do pixel da imagem final será a cor do pixel da figura (“foreground”);
- $h(p) = g(p)$ , ou seja, a cor do pixel da imagem final será a cor do pixel da imagem de fundo.

Quando passamos para o domínio discreto devemos tomar um cuidado especial com os pixels da fronteira dos conjuntos de decomposição. Esses pixels estão na fronteira que separa os elementos da figura dos elementos do fundo. Eles possuem informações tanto da imagem da frente, como da imagem do fundo. Nesse caso, temos mais uma opção de cor para o pixel da imagem superposta:

- $h(p)$  é uma combinação das cores de  $f$  e  $g$ .

As três condições acima para os valores de intensidade dos pixels na imagem superposta  $h$ , são ilustradas na Figura 4.6.

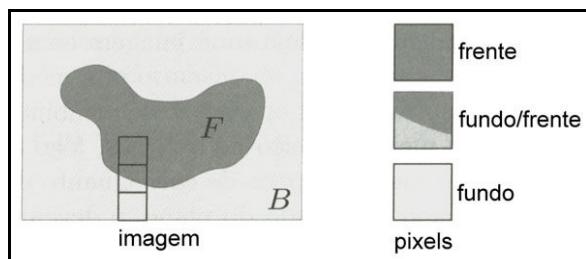


Figura 4.6 – Tipos de pixel na decomposição de uma imagem

Existem dois métodos de combinar imagens. Um método algébrico, baseado em uma partição da unidade, e um método geométrico, baseado em uma decomposição do domínio das imagens. Além disso, esses dois métodos estão estreitamente relacionados.

Na combinação algébrica cada função  $\alpha_i$  da partição carrega informações sobre a subdivisão do domínio das imagens a serem compostas, e é sob o ponto de vista da geometria dessa subdivisão que devemos analisar a operação de combinação de imagens no domínio discreto. Esse é um ponto muito importante, e deve ser enfatizado. Uma curva de fronteira de subdivisão divide o pixel em duas regiões conforme ilustrado na Figura 4.7(a). Esse caso ocorre, por exemplo, em uma decomposição figura-fundo, uma das regiões faz parte da figura, e a outra do fundo. Quando fazemos composições genéricas com imagens, precisamos obter interseções das regiões de decomposição. Na Figura 4.7(b) mostramos a interseção de duas regiões, o que resulta em uma decomposição do interior do pixel com quatro regiões.

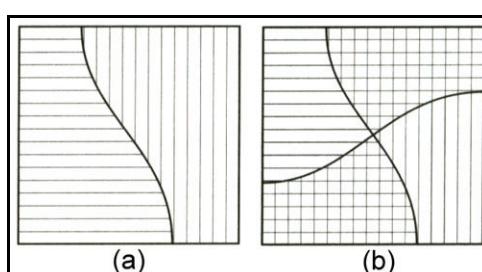


Figura 4.7 – Geometria da decomposição no interior do pixel

No processo de discretização é importante manter a informação da geometria da decomposição no interior do pixel. No que se segue chamaremos a geometria da subdivisão do pixel de **geometria do pixel**.

#### 4.5 A FUNÇÃO DE OPACIDADE

É útil pensar na função  $\alpha_f$  da partição da unidade definida por uma decomposição figura-fundo, como sendo uma medida do grau de opacidade do pixel da imagem da frente: quando o pixel é opaco,  $\alpha_1 = 1$ , o pixel do fundo não aparece; quando o pixel é transparente,  $\alpha_1 = 0$ , o pixel da imagem final assume a cor do pixel de fundo; quando o pixel não é totalmente transparente,  $0 < \alpha_1 < 1$ , podemos ver parcialmente a cor do pixel da imagem do fundo. Desse modo, a cor do pixel final é obtida a partir de uma mistura da cor do pixel na imagem da frente, com a cor do pixel correspondente na imagem de fundo. A função  $\alpha_f$  que mede a opacidade do pixel é de grande importância na definição de diversos operadores de combinação de imagens.

A função de opacidade é obtida multiplicando-se a função característica da figura pelo grau de opacidade, que é um número entre 0 e 1.

$$h(p) = \alpha_1(p)f(p) + (1 - \alpha_1)(p)g(p)$$

A equação acima define a combinação de duas imagens usando-se partição da unidade. Devemos multiplicar a imagem de frente pelo seu canal de opacidade, e fazer a soma que define a operação. A operação pode ser agilizada armazenando cada canal de cor da imagem pré-multiplicado pelo canal  $\alpha$ . Ou seja, ao invés de armazenar para cada pixel o valor  $(r, g, b, \alpha)$ , armazenamos  $(\alpha_r, \alpha_g, \alpha_b, \alpha)$ .

O operador de superposição pode ser definido através de uma partição da unidade definida por duas funções  $\alpha$  e  $1 - \alpha$ , onde  $\alpha$  mede o grau de opacidade do pixel da imagem da frente. A função  $\alpha$  é chamada de **função de opacidade do pixel**. Nos pixels que assumem informações da frente e do fundo, a função de opacidade assume um valor entre 0 e 1.

O valor de  $\alpha$  representa a percentagem de área do pixel coberta pela região da imagem da frente. Quando o pixel está contido na região da imagem da frente, Figura 4.8(a), sua área é totalmente coberta por essa região. Nesse caso a sua opacidade é 100%, e pombos  $\alpha = 1$ . Quando ocorre a situação da Figura 4.8(b), a área do pixel não é totalmente coberta pela região da frente. Nesse caso o pixel não tem uma opacidade de 100%, e atribuímos à função de opacidade um valor  $0 < \alpha < 1$ , que representa a percentagem de área do pixel coberta pela região da imagem da frente. Quando o pixel é disjunto da região da frente, Figura 4.8(c), então ele é totalmente transparente, e sua função de opacidade assume valor  $\alpha = 0$ .

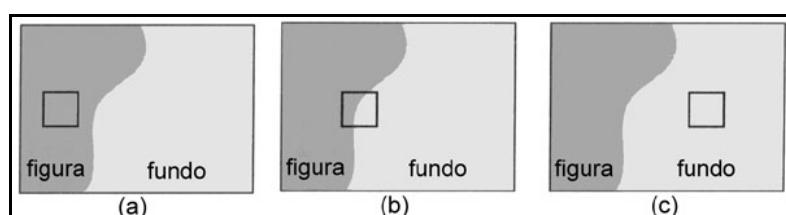


Figura 4.8 – Canal alfa e geometria do pixel

## 4.6 DISCRETIZAÇÃO DA FUNÇÃO DE OPACIDADE

Quando temos uma partição da unidade  $\alpha_i$ ,  $i = 1, 2, \dots, n$  do domínio de uma imagem, cada função  $\alpha_i$  pode ser vista como uma imagem monocromática. Temos então dois métodos para discretizar cada função  $\alpha_i$ , de forma a minimizar problemas de *aliasing*:

- amostragem por área;
- superamostragem.

- Amostragem por área: Esse método de amostragem é equivalente a utilizar um filtro de suavização (passa-baixa) antes de proceder à amostragem da função  $\alpha_i$  da partição da unidade.

- Superamostragem: Esse método consiste em subdividir o pixel em subpixels e calcular o valor da partição da unidade em cada sub-pixel.

Os dois métodos de amostragem acima descritos, dão origem a duas maneiras distintas de se fazer composição de imagens no domínio discreto.

### 4.6.1 Discretização com Canal Alfa

No método de amostragem por área, procuramos manter a informação da geometria do pixel calculando a área ocupada por uma das regiões. No caso de uma decomposição figura-fundo, guardamos a informação da área do pixel ocupado pela figura.

Considere uma decomposição figura-fundo e seja  $\alpha$  e  $1 - \alpha$  a partição da unidade associada. Discretizando a função de opacidade  $\alpha$  no mesmo reticulado da imagem, usando amostragem por área, e fazendo uma quantização do valor de  $\alpha$  em cada pixel, obtemos uma imagem digital monocromática. Em geral essa imagem digital resultante é armazenada juntamente com os canais de cor da imagem, constituindo assim mais um canal, que é chamado de **canal  $\alpha$** .

Não devemos esquecer a interpretação geométrica do valor do pixel associado ao canal  $\alpha$  da imagem: o valor de  $\alpha$  representa a percentagem de área do pixel coberta pela figura. Quando o pixel está contido na região da figura, sua área é totalmente coberta por essa região. Nesse caso o valor de  $\alpha$  é 1. Quando o pixel está contido na região de fundo, o valor do canal  $\alpha$  será zero. Quando o pixel está na fronteira que divide a figura do fundo, o valor do canal  $\alpha$  da imagem é um número entre 0 e 1.

Na Figura 4.9 mostramos, à esquerda, uma imagem sintética, e à direita, o seu canal  $\alpha$ . Nessa figura percebemos claramente que o canal  $\alpha$  é o análogo digital das máscaras utilizadas tradicionalmente no processo de composição analógica na indústria do cinema e da televisão.

A função  $1 - \alpha$ , forma, juntamente com a função de opacidade  $\alpha$ , a partição da unidade da decomposição figura-fundo de uma imagem. Quando discretizamos a função  $1 - \alpha$  e a representamos como uma imagem, obtemos a contra-máscara da máscara definida pela função opacidade. Essa contra-máscara é mostrada na Figura 4.10.

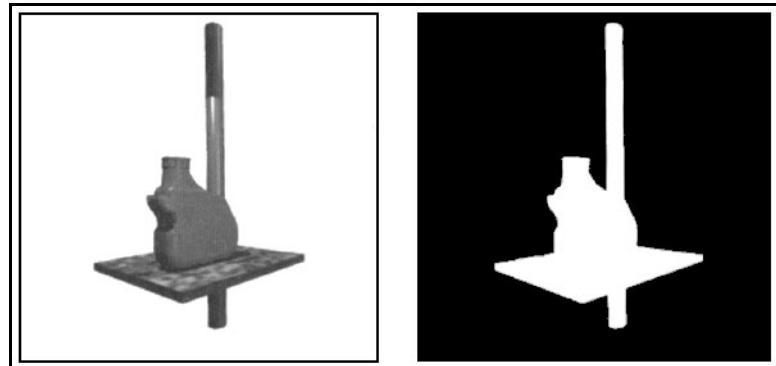


Figura 4.9 – Imagem digital e seu canal alfa

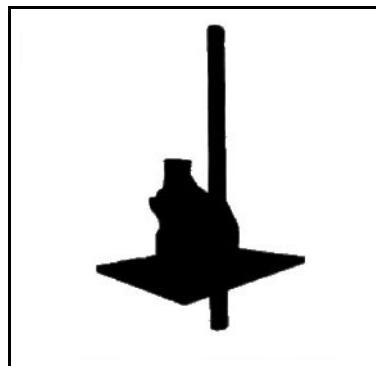


Figura 4.10 – Contra-máscara da máscara de opacidade

Na Figura 4.11(a) mostramos um detalhe do canal  $\alpha$  de uma imagem sintética mostrada na Figura 4.9 (o detalhe é da extremidade direita da mesa de mármore). O leitor deve observar as gradações de cinza na região de fronteira da máscara. Na Figura 4.11(b) mostramos o mesmo detalhe fazendo apenas amostragem pontual, isto é, sem levar em consideração o percentual da área ocupada pela figura e pelo fundo em cada pixel.

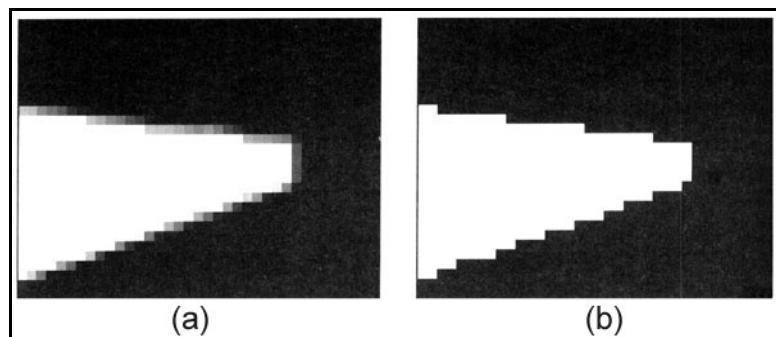


Figura 4.11 – Detalhe do canal alfa de uma imagem

#### 4.6.2 Discretização com Máscara de Bits

O canal  $\alpha$  procura manter a informação da geometria do pixel fazendo apenas uma amostragem de área. As regiões de fronteira entre os diversos conjuntos da decomposição do domínio da imagem possuem, em geral, altas variações de frequência. Isso ocorre porque é nessas regiões onde temos descontinuidade das informações da imagem da frente e do fundo. O método do canal  $\alpha$  minimiza os problemas de *aliasing* nessas regiões no processo de

composição. No entanto, o canal  $\alpha$  não contém informações detalhadas sobre a geometria do pixel.

Para melhor preservar a informação da geometria do pixel no processo de discretização podemos utilizar o processo de superamostragem. Nesse caso, fazemos uma subdivisão do pixel em sub-pixels, em cada sub-pixel avaliamos a função da partição, e utilizamos 1 bit para informar se o valor dessa função no sub-pixel é 1 ou 0, o que corresponde ao sub-pixel pertencer à figura ou ao fundo.

Desse modo, ao invés de obter uma imagem monocromática definindo a geometria da subdivisão de cada pixel, obtemos uma máscara de bits com informações sobre essa geometria. Na Figura 4.12 ilustramos um pixel com duas regiões correspondendo a uma decomposição figura-fundo, mostramos a divisão do pixel em sub-pixels, e uma máscara de bits correspondente.

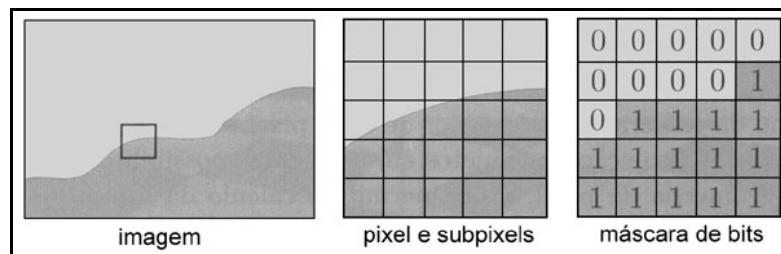


Figura 4.12 – Máscara de bits numa decomposição figura-fundo de um pixel

## 4.7 CÁLCULO DA FUNÇÃO DE OPACIDADE

O cálculo da função de opacidade está diretamente relacionado com o método utilizado para gerar a imagem. Podemos destacar três casos principais, baseados no tipo das imagens utilizadas em cada aplicação:

- Imagens sintéticas bidimensionais, criadas com o auxílio de um sistema de pintura, ou de um programa de ilustração;
- Imagens sintéticas tridimensionais geradas por um sistema de síntese de imagens. Essas imagens estão associadas a cenas tridimensionais criadas no computador;
- Imagens de cenas reais capturadas por um dispositivo de digitalização (câmera de vídeo, scanner, etc.).

### 4.7.1 Imagens Sintéticas Bidimensionais

Essas imagens sintéticas podem ser geradas por um sistema de pintura, ou por um programa de ilustração, que são sistemas de síntese de imagens bidimensionais. Nas imagens sintéticas geradas em um sistema de pintura, em geral, o canal de opacidade é produzido pelo próprio programa de pintura, de forma integrada aos canais de cor. O usuário determina os atributos do pincel, tais como geometria, cor e transparência. A cada nova pincelada, as informações de cor e opacidade são combinadas com as já existentes na tela, de acordo com a operação de pintura selecionada. Os programas de ilustração criam a imagem a partir de uma definição precisa da geometria da cena, sendo portanto possível a determinação precisa da geometria de cada pixel, o que possibilita o cálculo da função de opacidade de cada pixel.

#### 4.7.2 Imagens Sintéticas Tridimensionais

Nas imagens sintéticas tridimensionais, o programa de visualização dispõe de todos os dados geométricos da cena, e pode portanto gerar com precisão o canal de opacidade. Os objetos tridimensionais são projetados no plano da câmera virtual e a cor de cada pixel é determinada pelo modelo de iluminação. A projeção dos objetos em cada pixel possibilita o conhecimento preciso da geometria do pixel, o que permite o cálculo da função de opacidade do pixel. A representação discreta dessa geometria depende do método de visualização utilizado, e pode ser através da máscara de bits ou do canal alfa.

#### 4.7.3 Imagens Digitalizadas

Quando capturamos uma imagem utilizando um dispositivo de digitalização, dispomos somente dos canais de cor da imagem. A função de opacidade precisa ser estimada a partir da informação de cor. Como todos os problemas relativos a combinação de imagens, esse problema se reduz a se fazer uma decomposição do domínio da imagem. Ou seja, queremos obter uma decomposição figura-fundo da imagem de forma a determinar a função de opacidade associada. Essa decomposição do domínio é, em geral, um dos problemas difíceis na área de análise de imagens. Nas aplicações práticas porém a geometria da decomposição é simples e é possível prover métodos bastante robustos para calcular a decomposição figura-fundo. Uma vez obtida a decomposição figura-fundo a função de opacidade pode ser definida em cada pixel utilizando algoritmos de “área fill”. É claro que neste caso a geração de uma máscara de bits é, em geral, impossível, uma vez que não temos uma definição precisa da geometria de cada pixel.

- **Exemplo 4.1 – Blue Screen:**

Um caso especial, que torna bastante fácil o cálculo da função de opacidade é quando a imagem tem um fundo de cor única (por exemplo: azul ou verde). Esse artifício, conhecido como “blue screen” ou “chroma key” é largamente utilizado, há várias décadas, na composição analógica de imagens em filme e vídeo. Na Figura 4.13 mostramos a imagem de um toro com fundo azul, e o canal de opacidade da imagem, obtido a partir do fundo azul. Mostramos também a composição, por superposição, da imagem do toro com a imagem de um piso xadrez.

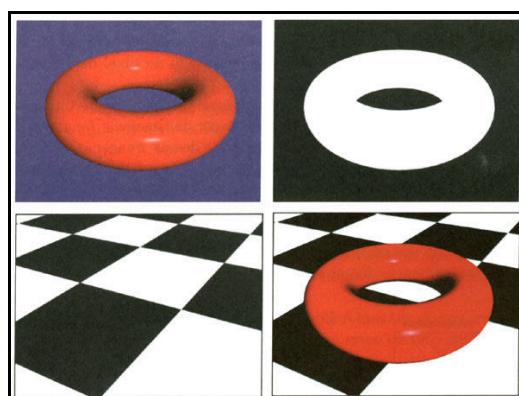


Figura 4.13 – Blue Screen e canal alfa

## 4.8 COMPOSIÇÃO NO DOMÍNIO DISCRETO

Os dois métodos de discretização da função de opacidade estudados anteriormente dão origem a dois modos distintos de se obter composição de imagens no domínio discreto: composição com canal  $\alpha$ , e composição com máscara de bits.

### 4.8.1 Composição com Canal de Opacidade

O canal  $\alpha$  é obtido através de uma discretização da função de opacidade. Essa função é quantizada espacialmente, usando o mesmo reticulado da discretização da imagem. Além disso a informação de opacidade é quantizada usando o mesmo número de bits da quantização dos canais de cor da imagem. A imagem resultante é acrescentada às componentes de cor da imagem, dando origem ao canal  $\alpha$  da imagem. Esse novo canal passa a fazer parte integrante da imagem para a qual ela foi calculada. Obtemos desse modo uma representação bastante homogênea para a imagem juntamente com a sua função de opacidade.

Estudamos anteriormente o operador de superposição de imagens que se utiliza da função de opacidade do pixel para sobrepor uma imagem sobre outra. Veremos agora como podemos utilizar a função de opacidade do pixel de forma a definir outros operadores de combinação de imagens.

Quando fazemos a combinação de duas imagens  $f$  e  $g$ , obtendo uma imagem  $h$ , devemos combinar não apenas as informações de cor das imagens  $f$  e  $g$ , mas também a informação dos canais de opacidade  $\alpha_f$  e  $\alpha_g$  dessas imagens. A combinação desses canais, resulta no canal  $\alpha$  da imagem final, o que é de grande importância para futuras combinações dessa imagem com outras imagens.

O canal  $\alpha$  mede apenas a percentagem de área do pixel ocupada pela cor de uma determinada imagem, não dando qualquer informação sobre a geometria exata da região de cor no interior do pixel. Desse modo, quando combinamos o canal  $\alpha$  de duas imagens temos de fazer determinadas suposições.

A ilustrações (a), (b) e (c) da Figura 4.14 mostram as três possíveis configurações para a geometria do pixel das imagens  $f$  e  $g$ . Temos três opções: as regiões de cor das imagens  $f$  e  $g$  não se superpõem, (a), ou essas regiões se superpõem de modo completo ou parcial, (b) ou (c). Essa última configuração, que é a mais provável genericamente, tem o nome de **posição genérica**.

Vamos sempre admitir que ocorre a posição genérica da Figura 4.14(c). Nesse caso, os polígonos de cor de cada imagem  $f$  e  $g$  determinam uma partição do pixel em quatro subconjuntos:  $\bar{f} \cap \bar{g}$ ,  $f \cap \bar{g}$ ,  $\bar{f} \cap g$  e  $f \cap g$ , onde a barra indica o conjunto complementar em relação à região do pixel. Na Figura 4.15 mostramos o pixel juntamente com os quatro conjuntos dessa partição.

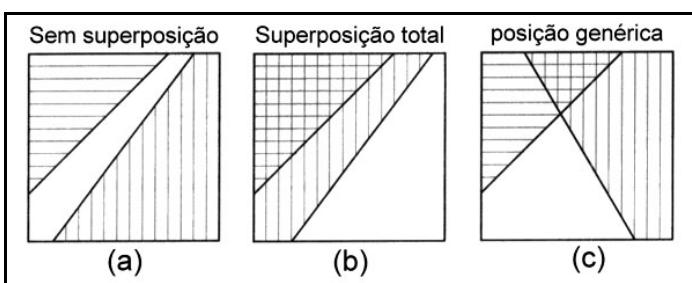


Figura 4.14 – Configurações da geometria do pixel

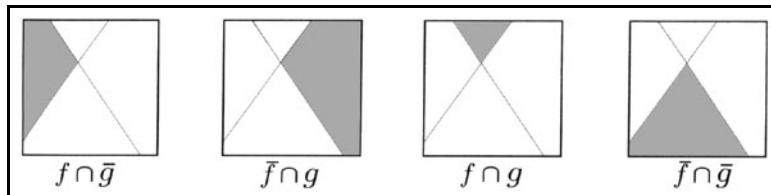


Figura 4.15 – Geometria do sub-pixel

Diferentes configurações da geometria do pixel podem ser obtidas fazendo operações de conjuntos com as regiões das partições. Por exemplo, a região da imagem  $f$  é dada por  $[f \cap \bar{g}] \cup [f \cap g]$ .

Além disso, se  $\alpha_f$  e  $\alpha_g$  representam os valores das funções de opacidade de  $f$  e  $g$  em um pixel, vamos assumir que a percentagem de área da interseção das duas regiões é dada pelo produto  $\alpha_f \alpha_g$ . Essa suposição é bastante plausível de um ponto de vista genérico, uma vez que o produto  $\alpha_f \alpha_g$  indica a probabilidade de um ponto pertencer à interseção das duas regiões. Vale ressaltar que essa suposição implica que a percentagem de área da região da imagem  $g$  que intersecta a região da imagem  $f$  é dada por  $\alpha_g$ .

É claro que com todas as suposições acima poderemos ter graves erros quando fazemos composições sucessivas de imagens. Esses erros de fato ocorrem, porém são irrelevantes na maioria dos casos.

Como a percentagem de área coberta pela região  $f \cap g$  é  $\alpha_f \cdot \alpha_g$ , podemos calcular facilmente a percentagem de área do pixel correspondente às outras regiões da partição. A área do pixel coberta apenas pela imagem  $f$  é dada por  $f - (f \cap g)$ , portanto, a percentagem da área coberta apenas pela imagem  $f$  é  $\alpha_f - (\alpha_f \cdot \alpha_g)$ . Analogamente, a percentagem coberta apenas por informação da imagem  $g$  é  $\alpha_g - \alpha_f \cdot \alpha_g$ . A percentagem de área do pixel que não é coberta por elementos nem de  $f$  nem da imagem  $g$ , é dada por

$$(1 - \alpha_f)(1 - \alpha_g)$$

#### 4.8.2 Composição com Máscara de Bits

Vimos anteriormente que um dos métodos de discretizar a função de opacidade de um pixel consiste em dividir o pixel em sub-pixels e atribuir a cada sub-pixel um bit de opacidade. Se o bit é 1, o pixel é opaco naquela região, caso contrário, ele é transparente. Ilustramos esse processo de discretização na Figura 4.16.

A discretização do domínio da função de opacidade utilizando uma máscara de bits traz vantagens na combinação de imagens em relação ao método de combinação da geometria do pixel.

Utilizando o processo de máscara de bits, a operação de composição se dá em três etapas:

- composição das máscaras de bits;
- cálculo do canal alfa;
- combinação dos canais de cor.

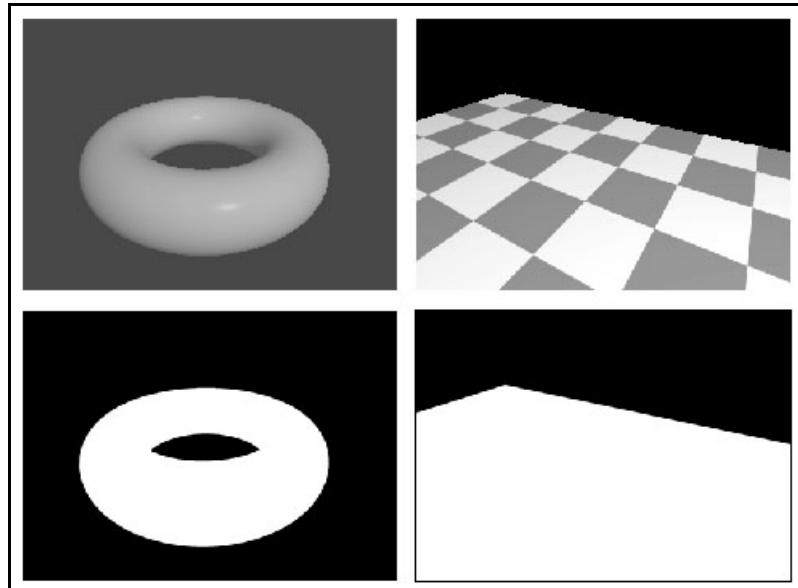


Figura 4.16 – Imagens usadas para ilustrar as operações de composição

- **Composição das máscaras de bits:** Na primeira etapa, as máscaras de bits da figura e do fundo são combinadas por meio de operações lógicas bit a bit, de forma a determinar a decomposição do pixel em sub-regiões. As operações lógicas utilizadas são definidos pelos operadores **and** e **not**. Mais especificamente, indicando por  $M_f$  e  $M_g$  a máscara de bits de um pixel genérico nas imagens  $f$  e  $g$ , respectivamente, podemos escrever:

$$\begin{aligned}\bar{f} \cap \bar{g} &= (\text{not } M_f) \text{ and } (\text{not } M_g) \\ f \cap \bar{g} &= M_f \text{ and } (\text{not } M_g) \\ \bar{f} \cap g &= (\text{not } M_f) \text{ and } M_g \\ f \cap g &= M_f \text{ and } M_g\end{aligned}$$

- **Cálculo do canal alfa:** De posse da informação da máscara de bits de cada pixel da imagem composta, utilizamos essa máscara para calcular a informação de opacidade do pixel. Temos que calcular as percentagens da área do pixel correspondentes à figura e ao fundo,  $A_f$  e  $A_g$ . As percentagens são calculadas por:

$$A = \frac{\text{número de bits "on"}}{\text{número total de bits}}$$

- **Combinação dos canais de cor:** Nesta etapa, utilizamos a informação de opacidade, calculada na etapa anterior, para obter os valores de cor da imagem composta. Isso é feito de modo inteiramente análogo ao processo utilizado no método do canal de opacidade  $\alpha$ :

$$A_f f + A_g g$$

Conforme vimos acima, a composição utilizando máscara de bits é

análoga à composição pelo método do canal de opacidade. Devemos determinar a geometria da decomposição do pixel em quatro subconjuntos:  $\bar{f} \cap \bar{g}$ ,  $f \cap \bar{g}$ ,  $\bar{f} \cap g$  e  $f \cap g$ , para então proceder ao cálculo da opacidade e da informação de cor da imagem composta. A diferença com o método do canal  $\alpha$  é que neste caso a geometria do pixel é conhecida, portanto nenhuma suposição precisa ser feita. Desse modo, temos o resultado com a precisão da discretização dada pela máscara de bits construída no processo de superamostragem.

#### 4.9 OPERAÇÕES DE COMPOSIÇÃO

As diferentes operações de composição de imagens são obtidas a partir de decomposições do domínio das imagens a serem compostas. A operação de composição consiste de duas etapas:

- determinar os diversos conjuntos que definem a composição;
- fazer a atribuição de cor a cada um dos pixels nos diversos conjuntos da decomposição.

O atributo de cor provém das cores nos pixels das imagens  $f$  e  $g$  a serem compostas. Devemos salientar que além das cores das imagens  $f$  e  $g$ , podemos também atribuir a cor 0, para o caso em que desejamos eliminar qualquer informação de cor em uma das regiões da decomposição. A tabela abaixo indica as possíveis escolhas de cor em cada uma das regiões.

Região	Possíveis Cores
$\bar{f} \cap \bar{g}$	0
$f \cap \bar{g}$	0 ou $f$
$\bar{f} \cap g$	0 ou $g$
$f \cap g$	0, $f$ ou $g$

Com as diversas possibilidades de variação de cor em cada região, teremos um total de doze variações possíveis, o que dá origem a 12 operadores de composição de imagens. Vamos estudar brevemente 7 desses operadores. Em cada caso, indicamos por  $A_f$  e  $A_g$  a percentagem de área do pixel ocupada com uma cor proveniente da imagem  $f$  e  $g$  respectivamente. Desse modo, a cor resultante do pixel na imagem composta é dada por:

$$A_f f + A_g g$$

De modo análogo, o valor do canal  $\alpha$  de opacidade do pixel é dado por

$$A_f \alpha_f + A_g \alpha_g$$

Cada operador será também analisado utilizando o método da máscara de bits. Nesse caso, devemos calcular a máscara de bits  $M_r$  da imagem composta, bem como as percentagens  $A_f$  e  $A_g$ , para cada uma das operações de composição.

Para ilustrar as diversas operações de composição, vamos utilizar as

duas imagens  $f$  e  $g$  mostradas na Figura 4.16. A imagem  $f$  é um toro, e a imagem  $g$  também é um piso xadrez. Ambas as imagens foram geradas sinteticamente. Mostramos também na figura o canal  $\alpha$  de cada uma das imagens.

Na Figura 4.17 mostramos em diferentes tons de cinza as partições  $\bar{f} \cap \bar{g}$ ,  $f \cap \bar{g}$ ,  $\bar{f} \cap g$  e  $f \cap g$ , determinadas pelos elementos da imagem  $f$  e  $g$  no conjunto suporte. A cor preta indica o fundo da imagem, onde não temos elementos nem da imagem  $f$  nem da imagem  $g$ .

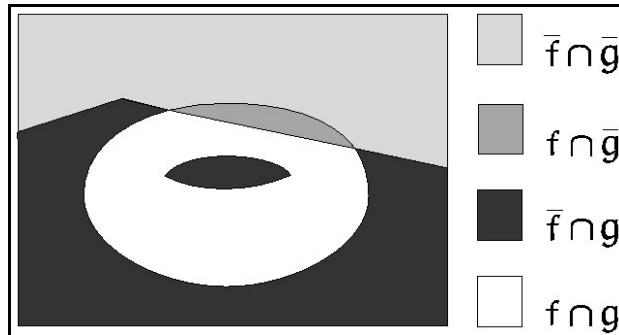


Figura 4.17 – Partição do conjunto suporte das imagens  $f$  e  $g$

#### 4.9.1 Operador “OVER”

Esse é o operador de superposição que estudamos anteriormente. A cor dos elementos da imagem da frente  $f$  no pixel tem sempre predominância sobre a cor proveniente da imagem do fundo  $g$ . Na Figura 4.18 ilustramos o operador juntamente com a tabela de atributos de cor em cada região da partição do pixel. Neste caso a percentagem de área do pixel ocupado pela imagem  $f$  é  $A_f = 1$ , e a percentagem ocupada pela imagem  $g$  é  $A_g = 1 - \alpha_f$ . É claro que o operador não é comutativo.

Região	Cor			
$f \cap \bar{g}$	$f$			
$\bar{f} \cap g$	$g$			
$f \cap g$	$f$			
$\bar{f} \cap \bar{g}$	0			

$f$ 
 $g$ 
 $over(f, g)$

Figura 4.18 – Geometria do pixel do operador “OVER”

Utilizando o método de mascada de bits, obtemos:

$$M_r = M_f;$$

$$A_f = 1;$$

$$A_g = \frac{\text{número de bits } M_g}{\text{número de bits}[(not } M_f) \text{ and } M_g]}.$$

Na Figura 4.19, mostramos o efeito do operador “over” nas imagens  $f$  e  $g$ , bem como o canal  $\alpha$  da imagem resultante, obtido através da combinação do canal  $\alpha$  de cada imagem.

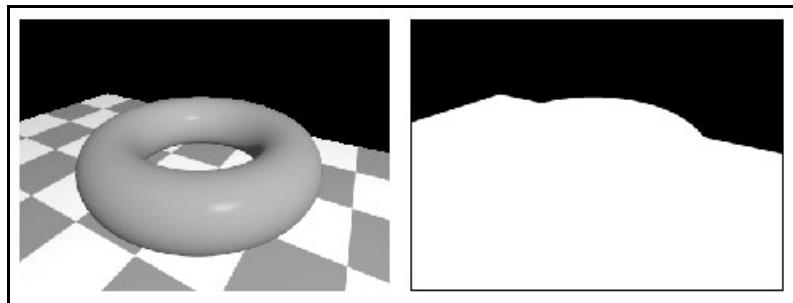


Figura 4.19 – Operador “OVER”

#### 4.9.2 Operador “INSIDE”

Esse operador aplicado às imagens  $f$  e  $g$ , considera apenas a informação da imagem  $f$  que está contida na imagem  $g$ . A ilustração da Figura 4.20 esclarece a definição, e mostra os valores de cor de cada região da partição. A percentagem de área do pixel com informação de cor da imagem  $f$  é  $A_f = \alpha_g$ . A percentagem de área ocupada pela cor da imagem  $g$  é  $A_g = 0$ . Também aqui o operador não é comutativo.

Região	Cor			
$f \cap \bar{g}$	0			
$\bar{f} \cap g$	0			
$f \cap g$	$f$			$f$
$\bar{f} \cap \bar{g}$	0			$g$
				$\text{inside}(f,g)$

Figura 4.20 – Geometria do pixel do operador “INSIDE”

Usando a composição com o método da máscara de bits, obtemos:

$$M_r = M_f \text{ and } M_g;$$

$$A_f = \frac{\text{número de bits } M_r}{\text{número de bits } M_f};$$

$$A_g = 0.$$

Na Figura 4.21 mostramos o resultado do operador *inside* nas imagens  $f$  e  $g$ , bem como a máscara da imagem resultante definida pela combinação do canal  $\alpha$  de cada imagem.

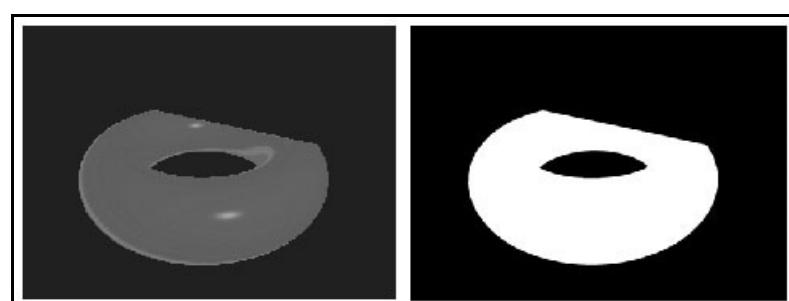


Figura 4.21 – Operador “INSIDE”

#### 4.9.3 Operador “OUTSIDE”

O resultado da operação  $\text{outside}(f, g)$  consiste em considerar a região da imagem  $f$  fora da região delimitada pelos elementos da imagem  $g$ . A Figura 4.22 ilustra a operação nas regiões da partição do pixel, e fornece uma tabela de cores de cada região. A percentagem de área do pixel com atributos de cor da imagem  $f$  é  $A_f = 1 - \alpha_g$ ; a percentagem correspondente à imagem  $g$  é 0. É claro que o operador não é comutativo.

Pelo método da máscara de bits, obtemos:

$$M_r = M_f \text{ and } (\text{not } M_g);$$

$$A_f = \frac{\text{número de bits } M_r}{\text{número de bits } M_f};$$

$$A_g = 0.$$

Região	Cor			
$f \cap \bar{g}$	$f$			
$\bar{f} \cap g$	0			
$f \cap g$	0			
$\bar{f} \cap \bar{g}$	0			

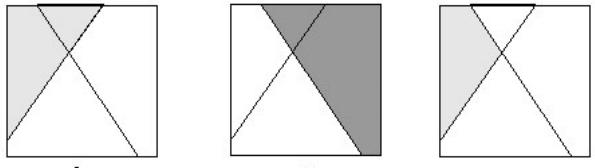


Figura 4.22 – Geometria do pixel do operador “OUTSIDE”

A operação de composição  $\text{outside}$  é ilustrada na Figura 4.23, onde mostramos também o canal  $\alpha$  da imagem resultante.

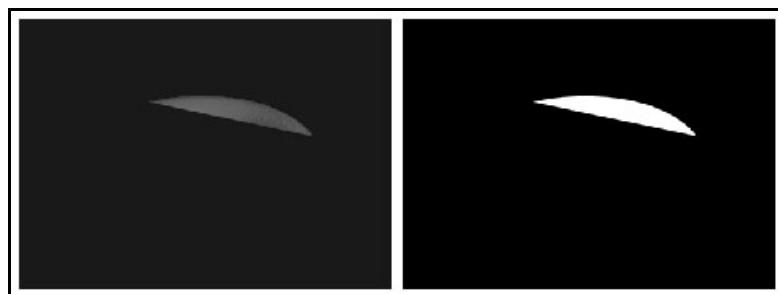


Figura 4.23 – Operador “OUTSIDE”

#### 4.9.4 Operador “ATOP”

O resultado do operador  $\text{atop}(f, g)$  consiste em superpor a cor da imagem  $f$  nas regiões onde existem elementos de cor da imagem  $g$ . Esse operador é ilustrado na Figura 4.24. Nessa figura mostramos também uma tabela das cores nas diversas sub-divisões do pixel. A percentagem de área do pixel ocupada com informação de cor da imagem  $f$  é  $A_f = \alpha_g$ ; a percentagem para a imagem  $g$  é dada por  $A_g = 1 - \alpha_f$ .

Região	Cor			
$f \cap \bar{g}$	$0$			$f$
$\bar{f} \cap g$	$g$			$g$
$f \cap g$	$f$			
$\bar{f} \cap \bar{g}$	$0$			$\text{atop}(f, g)$

Figura 4.24 – Geometria do pixel do operador “ATOP”

Usando a discretização do canal de opacidade com a máscara de bits, obtemos:

$$M_r = M_g;$$

$$A_f = \frac{\text{número de bits}(M_f \text{ and } M_g)}{\text{número de bits } M_f};$$

$$A_g = \frac{\text{número de bits}[M_f \text{ and } (\text{not } M_g)]}{\text{número de bits } M_g}.$$

O resultado de aplicar o operador atop, nas imagens de teste  $f$  e  $g$ , é mostrado na Figura 4.25. Nessa figura mostramos também o canal  $\alpha$  da imagem resultante. É claro que o canal  $\alpha$  coincide com o canal  $\alpha$  da imagem  $g$ .

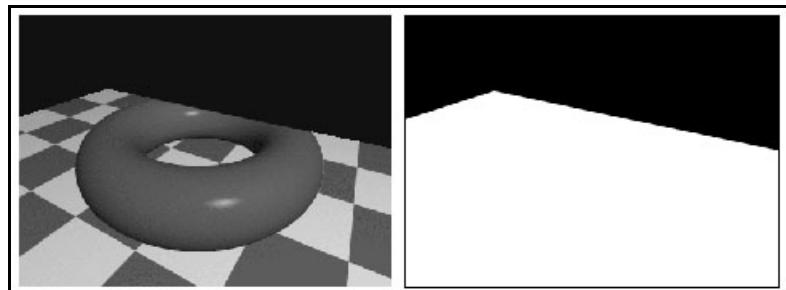


Figura 4.25 – Operador “ATOP”

#### 4.9.5 Operador “XOR”

O operador  $\text{xor}(f, g)$  executa a operação de diferença simétrica entre os conjuntos de pixels das imagens  $f$  e  $g$ . Ou seja,  $\text{xor}(f, g) = (f - g) \cup (g - f)$ . A Figura 4.26 ilustra como o operador atua nas regiões da partição do pixel, e fornece o valor de cor de cada região. A percentagem de área do pixel com a cor da imagem  $f$  é  $A_f = 1 - \alpha_g$ , e a percentagem de área do pixel com a cor da imagem  $g$  é  $A_g = 1 - \alpha_f$ .

Região	Cor			
$f \cap \bar{g}$	$f$			$f$
$\bar{f} \cap g$	$g$			$g$
$f \cap g$	$0$			
$\bar{f} \cap \bar{g}$	$0$			$\text{xor}(f, g)$

Figura 4.26 – Geometria do pixel do operador “XOR”

No processo com máscara de bits, obtemos os seguintes valores dos parâmetros:

$$M_r = \lfloor M_f \text{ and } (\text{not } M_g) \rfloor \text{ or } \lfloor (\text{not } M_f) \text{ and } M_g \rfloor;$$

$$A_f = \frac{\text{número de bits } (M_f \text{ and } (\text{not } M_g))}{\text{número de bits } M_f};$$

$$A_g = \frac{\text{número de bits } ((\text{not } M_f) \text{ and } M_g)}{\text{número de bits } M_g}.$$

Na Figura 4.27 mostramos o resultado da operação  $\text{xor}(f, g)$ . Nessa figura mostramos também o canal  $\alpha$  da imagem resultante.

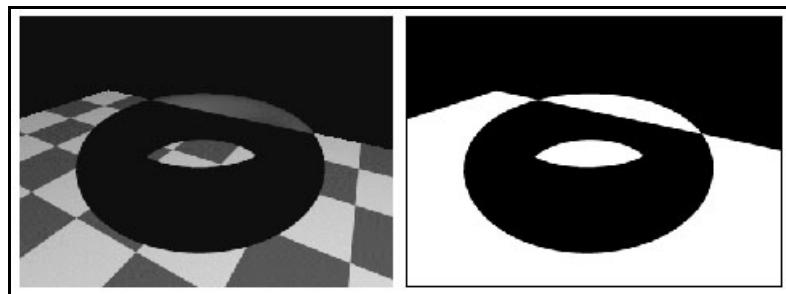


Figura 4.27 – Operador “XOR”

Os operadores de composição podem ser utilizados conjuntamente para se obter diversas combinações entre imagens. Nesse contexto é conveniente definir, utilizando o mesmo procedimento acima, dois operadores unários de imagens: o operador ***clear*** e o operador ***set***.

#### 4.9.6 Operador “CLEAR”

Faz atribuição da cor nula no pixel (cor de fundo), independente da informação de geometria do pixel, e torna o pixel totalmente transparente. Temos, portanto,  $A_f = A_g = 0$ . Devemos atentar aqui para a diferença entre um pixel  $(0, 0, 0, \alpha)$ , com  $\alpha > 0$ , e um pixel  $(0, 0, 0, 0)$ . No primeiro caso temos um pixel de cor nula e não transparente, e no segundo caso temos um pixel de cor nula, e transparente. O operador “*clear*” transforma o pixel em um pixel do tipo  $(0, 0, 0, 0)$ . Na Figura 4.28 ilustramos essa operação.

Região	Cor			
$f \cap \bar{g}$	0			
$\bar{f} \cap g$	0			
$f \cap g$	0			
$\bar{f} \cap \bar{g}$	0			

*f*

*g*

*clear(f, g)*

Figura 4.28 – Geometria do pixel do operador “CLEAR”

Utilizando máscara de bits, temos obviamente  $A_f = 0$ ,  $A_g = 0$  e  $M_r = 0$ .

#### 4.9.7 Operador “SET”

O operador  $\text{set}(f, g)$ , considera apenas a informação da cor proveniente da imagem  $f$ , desaparecendo portanto a cor da imagem  $g$  do pixel. Desse modo, temos  $A_f = 1$  e  $A_g = 0$ . Na Figura 4.29 mostramos os valores de cor em cada região da partição do pixel, e ilustramos a operação a nível da geometria do pixel. É claro que o operador não é comutativo.

Utilizando máscara de bits, temos  $M_r = M_f$ ,  $A_f = 1$  e  $A_g = 0$ .

Região	Cor			
$f \cap \bar{g}$	$f$			
$\bar{f} \cap g$	0			
$f \cap g$	$f$			
$\bar{f} \cap \bar{g}$	0			

Figura 4.29 – Geometria do pixel do operador “SET”

As operações de composição estudadas nesta seção, juntamente com as duas operações unárias introduzidas acima podem ser utilizadas para se obter diversos efeitos de combinação de imagens.

## UNIDADE 5 – WARPING E MORPHING

Nesta seção vamos estudar os filtros topológicos cujo objetivo é deformar objetos em uma imagem. Essas operações de deformação de imagens são comumente conhecidas na literatura de processamento de imagem pelo nome de **warping**. Combinados com filtros de amplitude que permitem fazer alteração na cor de uma imagem, podemos obter efeitos de transição entre os diversos objetos presentes na imagem. Esses efeitos são conhecidos na literatura pelo nome de **morphing**. O uso conjunto dos filtros de *warping* e de *morphing* são importantes em diversas aplicações desde a correção de distorções em imagens, à obtenção de efeitos especiais de transição na indústria de entretenimento.

### 5.1 FILTRO DE WARPING

*Warping* é o nome dado ao processo de alteração de uma imagem de tal modo que a relação espacial entre seus objetos e características é alterada conforme outra imagem ou gabarito (*template*).

Dada uma imagem  $f: U \subset \mathbb{R}^2 \rightarrow C$ , um **filtro de warping** é definido por uma aplicação  $h: U \subset \mathbb{R}^2 \rightarrow V \subset \mathbb{R}^2$ . O efeito dessa transformação se manifesta perceptualmente por uma deformação dos objetos na imagem. Na Figura 5.1, mostramos um exemplo do uso de um filtro de *warping* em uma imagem.

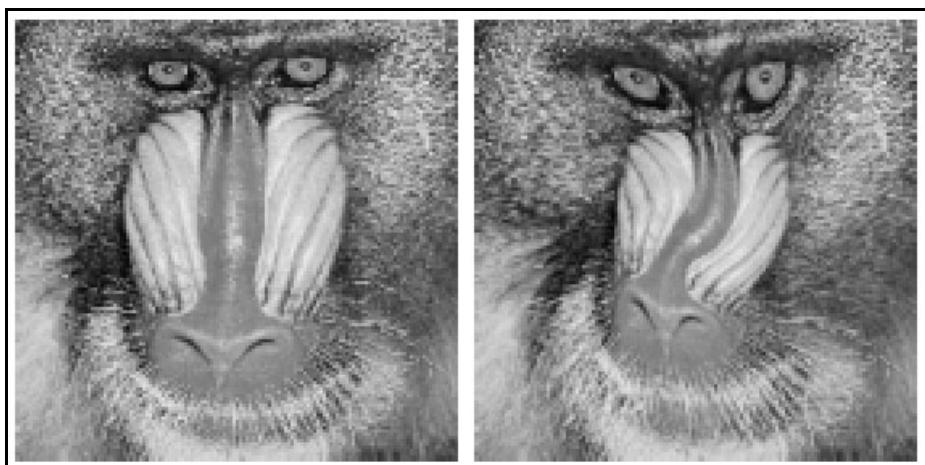


Figura 5.1 – *Warping* de uma imagem

Em geral fazemos algumas restrições à função  $h$  que define o *warping*. Uma restrição natural consiste em exigir que  $h$  seja uma aplicação injetiva e contínua. A injetividade garante que não há superposição de pontos na deformação, a continuidade garante que não há rupturas da imagem no processo de deformação.

Resumindo, a operação de *warping* faz uma mudança das coordenadas  $(x, y)$  dos pixels da imagem, para outro sistema de coordenadas  $(u, v)$ .

Um exemplo simples de *warping* é a reflexão de uma imagem digital  $n \times n$  em torno de sua diagonal. A transformação geométrica é definida por  $(x, y) \rightarrow (y, x)$ , ou seja,  $u(x, y) = y$  e  $v(x, y) = y$ . Portanto,  $g(u, v) = f(u(x, y), v(x, y)) = f(y, x)$ .

Outro exemplo simples de um filtro de *warping* é dado pela rotação da imagem por um ângulo de  $90^\circ$  em torno do pixel localizado na origem. Nesse caso, a transformação é dada por:

$$h(x, y) = \begin{bmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = (-y, x)$$

A Figura 5.2 mostra o efeito dessa transformação em uma imagem.



Figura 5.2 – Rotação de uma imagem

Os dois exemplos acima, reflexão na diagonal e rotação de 90º, têm uma particularidade que torna o filtro de *warping* extremamente simples de ser implementado. A reflexão e a rotação são movimentos rígidos, portanto não há deslocamento relativo entre os pixels após a transformação de *warping*. Além disso, nessas duas transformações os pixels do reticulado, após a transformação, ficam completamente alinhados com os pixels do reticulado original. Essa situação é ideal. Em geral, quando aplicamos um filtro de *warp* ocorre exatamente o oposto:

- A posição relativa dos pixels da imagem muda após a transformação;
- O reticulado de pixels, após transformado, não se alinha com o reticulado original.

Esses dois fatos estão na origem da maioria dos problemas quando trabalhamos com filtros e *warping* no domínio discreto.

## 5.2 EXPANSÃO E CONTRAÇÃO

Uma transformação  $T: U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^2$  é uma **expansão**, se

$$|T(X) - T(Y)| \geq \lambda |X - Y|, \quad \forall X, Y \in U,$$

onde  $\lambda > 1$ . Isso significa que a distância entre dois pontos  $X$  e  $Y$  do domínio, após transformados, é sempre maior do que a distância entre os pontos originais.

De modo análogo,  $T$  é uma **contração**, se

$$|T(X) - T(Y)| \leq \lambda |X - Y|, \quad \forall X, Y \in U,$$

com  $\lambda < 1$ . Ou seja, a distância entre os pontos  $X$  e  $Y$ , após transformados, é menor do que a distância entre os pontos originais.

- **Exemplo 5.1 – Homotetia:**

Um exemplo simples de contração e expansão pode ser obtido usando a transformação linear de homotetia, definida por  $T(X) = \lambda X$ ,  $\lambda \in \mathbb{R}$ . Para  $\lambda > 1$  temos uma transformação expansiva, e para  $\lambda < 1$  temos uma contração. O número  $\lambda$  é chamado de **fator de escala**, ou **razão de homotetia**.

Se uma transformação não é uma expansão, nem uma contração, então ocorre a igualdade

$$|T(X) - T(Y)| = |X - Y|, \forall X, Y \in U,$$

Nesse caso a transformação preserva a distância entre pontos e é, portanto, chamada de **isometria**.

Em geral, uma transformação de **warping** não realiza nem uma expansão nem uma contração globalmente, como no exemplo da homotetia acima. O caso mais comum é a transformação ser expansiva em uma determinada região e fazer uma contração em outra região do domínio.

Esse fato pode ser comprovado com o exemplo dado na Figura 5.3. A imagem (b) mostra o resultado da aplicação de um filtro de warping na imagem (a). Observando com atenção, o leitor pode perceber que, em algumas regiões temos expansão, em outras ocorre uma contração, e há algumas regiões onde não há qualquer deformação da imagem inicial.

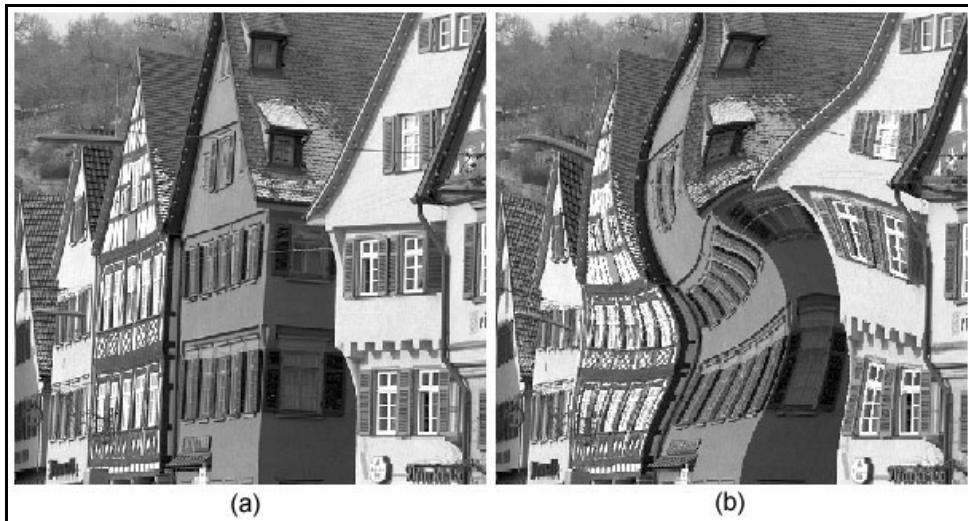


Figura 5.3 – Exemplo de warping de uma imagem

### 5.3 WARPING NO DOMÍNIO DISCRETO

A dificuldade básica ao utilizar um filtro de warping é introduzida pelo fato de trabalharmos com imagens no domínio discreto. Se pudéssemos manipular concretamente as representações contínuas de imagens, as transformações de warping não apresentariam grandes problemas.

Na prática, trabalhamos com o modelo discreto-contínuo: imagens espacialmente discretas, com os valores de cor codificados com números em ponto flutuante. Assim, como o mapeamento leva a pontos de um reticulado em pontos arbitrários do plano, se simplesmente transformássemos cada elemento da imagem

copiando seu valor para os elementos mais próximos da outra imagem, o resultado muito provavelmente conteria tantos elementos com valores indeterminados, quanto com valores múltiplos.

Podemos ilustrar os fatos acima mencionados com um exemplo unidimensional. Podemos entender esse exemplo como representando a transformação de uma linha de uma imagem. Na Figura 5.4 mostramos o gráfico da transformação de *warping*. De forma aproximada, o gráfico nos mostra que a transformação efetua uma expansão entre os pixels 4 e 5, uma contração entre os pixels 5 e 6. Ela é aproximadamente neutra, isto é não realiza expansão nem contração, no intervalo que compreende os pixels de 1 a 4. Observando a figura, podemos destacar dois fatos:

- As coordenadas inteiras dos pixels não são transformadas em coordenadas inteiras;
- Apesar de  $h$  ser uma função bijetiva, não temos uma transformação bijetiva no domínio discreto.

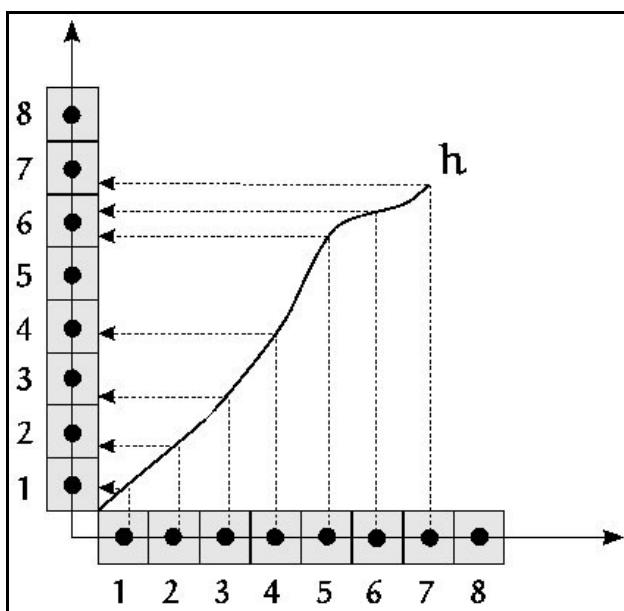


Figura 5.4 – Efeito da expansão e contração no domínio discreto

Na região onde a transformação é expansiva (entre os pixels 4 e 6) alguns pixels da imagem podem ser omitidos (pixel 5 por exemplo). Esses pixels ficam sem atribuição de cor, criando um efeito de **buracos** na imagem deformada. Nas regiões onde a transformação faz uma contração (entre os pixels 5 e 7 da figura) temos um efeito contrário: vários pixels podem ser mapeados em um mesmo pixel (pixels 5 e 6 da figura), criando um efeito de superposição na imagem final. Esse efeito de superposição resulta das altas frequências introduzidas ao fazermos a contração de uma imagem, e pode dar origem a problemas graves de *aliasing*.

Portanto, trabalhando com imagens discretas, ao aplicar uma transformação de expansão teremos vários pixels na imagem resultante para os quais teremos de calcular seus valores. Esse fato está ilustrado na Figura 5.5(a). Na contração a situação é a oposta, o aumento de frequências se traduz, no domínio discreto, no fato de que vários pixels são colapsados em um único pixel, conforme mostramos na Figura 5.5(b).

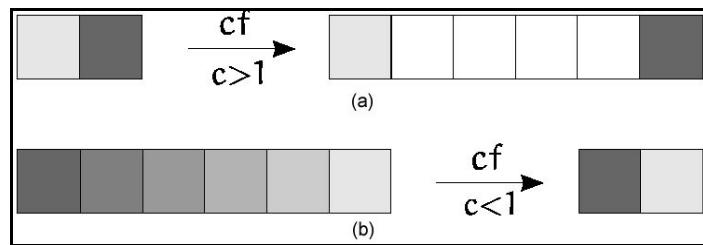


Figura 5.5 – Expansão (a) e contração (b), no domínio discreto

Podemos resumir o problema acima afirmando que na expansão o problema é de reconstruir a imagem nos novos pixels introduzidos pela transformação; na região de contração devemos usar um filtro de forma a obter a amostragem correta da imagem em um determinado pixel. Os problemas de buracos e superposição de pixels na imagem final podem ser resolvidos de forma unificada usando um processo de reconstrução e reamostragem da imagem, conforme explicado em seguida.

#### 5.4 RECONSTRUÇÃO E REAMOSTRAGEM

Os problemas de superposição e existência de buracos na imagem após ser processada por um filtro de *warping* podem ser resolvidos trabalhando com a imagem no domínio contínuo: reconstruimos a imagem original, aplicamos o filtro de *warping* na imagem contínua, e em seguida fazemos uma amostragem da imagem transformada. Esse processo, que é conhecido na literatura pelo nome de **reamostragem**, é ilustrado na Figura 5.6.

A imagem original, Figura 5.6(a), é reconstruída de modo a obter uma imagem contínua, Figura 5.6(b), utilizando um filtro de reconstrução. Aplicamos a transformação de *warping* à imagem no domínio contínuo obtendo uma outra imagem no domínio contínuo, Figura 5.6(c). A imagem resultante é então amostrada para se obter a imagem final na resolução desejada, Figura 5.6(d).

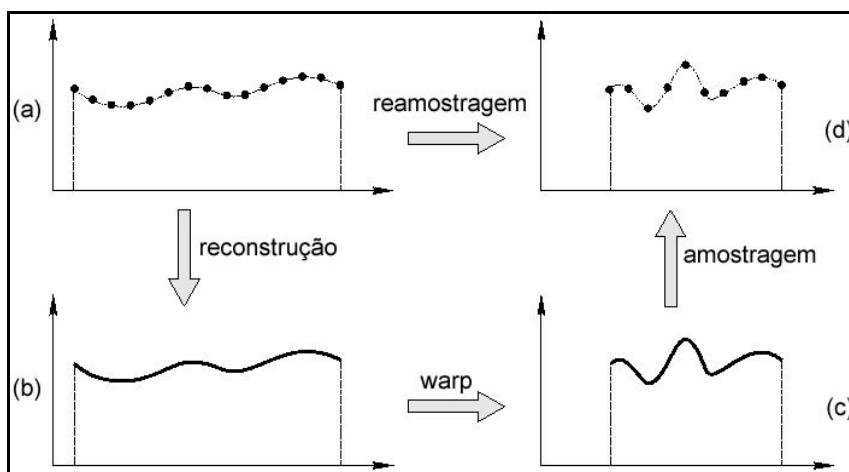


Figura 5.6 – *Warping* com reamostragem de uma imagem

Na expansão, reconstruimos a imagem e fazemos a amostragem nos pixels do reticulado. Como as frequências da imagem transformada diminuem, os erros provenientes do processo de reconstrução e amostragem são minimizados. Ou seja, no processo de expansão a transformação trabalha favorecendo o processo de reconstrução/amostragem.

No caso da contração, há um aumento das altas frequências do sinal transformado. Portanto, como em geral a imagem após ser filtrada é amostrada na mesma resolução da imagem original, podemos enfrentar graves problemas de *aliasing* no processo de reamostragem. Devemos, portanto, usar um filtro de suavização antes de reamostrar a imagem. O processo de reamostragem, ilustrado anteriormente na Figura 5.6, fica então como indicado na Figura 5.7.

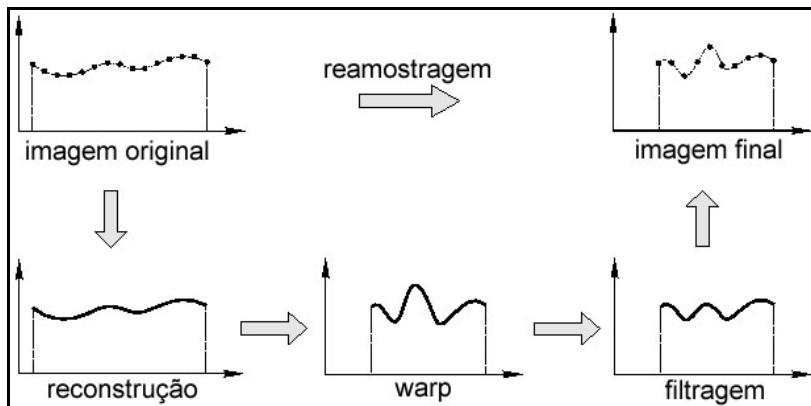


Figura 5.7 – *Warping* com reamostragem e filtragem

O método acima resolve por completo os dois problemas mencionados anteriormente: a reconstrução inicial evita buracos na imagem resultante, enquanto que a filtragem minimiza os problemas de *aliasing* decorrentes do fato de termos vários pixels na imagem original, mapeados no mesmo pixel.

Devido à grande variação da transformação em cada região do domínio da imagem, não é aconselhável o uso de filtros espacialmente invariantes. Devemos utilizar um tipo de filtragem que seja adequado às características da imagem numa vizinhança de cada pixel. Esse processo de filtragem é chamado de **filtragem adaptativa**.

## 5.5 WARPING POR TRANSFORMAÇÃO PROJETIVA

A transformação matemática mais comum, utilizada em *warping*, é baseada na transformação projetiva, dada pelas equações:

$$\begin{aligned} X' &= \frac{aX + bY + c}{iX + jY + 1} \\ Y' &= \frac{dX + eY + f}{iX + jY + 1} \end{aligned}$$

onde **X** e **Y** são as coordenadas antigas e **X'** e **Y'** as novas. Os coeficientes **a**, **b**, **c**, **d**, **e**, **f**, **i** e **j** são determinados a partir de um conjunto de pontos de controle que correspondem à congruência desejada entre as duas imagens ou entre a imagem original e o *template* selecionado. A Figura 5.8 mostra um exemplo do processo de *warping* aplicado a uma imagem binária simples.

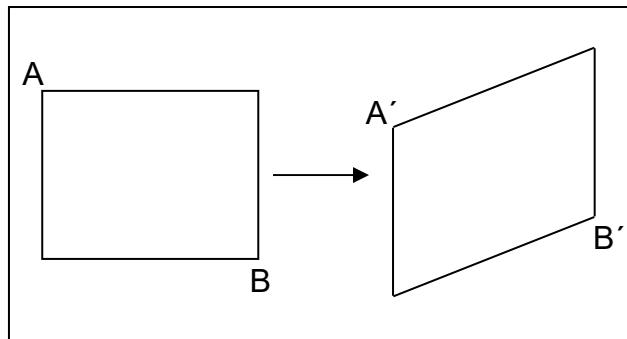


Figura 5.8 – Exemplo de *warping*

Neste exemplo, o ponto **A** é movido para a posição **A'** e **B** é movido para **B'**. A projeção perspectiva exige que sejam selecionados quatro pares de pontos de controle, para resultar um sistema de oito equações a oito incógnitas. Os pares **A-A'** e **B-B'** são duas escolhas óbvias. Os outros dois pontos escolhidos, neste exemplo, são os dois cantos restantes do retângulo. Se tivéssemos selecionado mais de quatro pontos de controle, um ajuste por mínimos quadrados seria necessário para determinar os melhores valores para a transformação.

Na prática, um programa para a solução simultânea de um sistema de equações é utilizado para calcular os valores dos coeficientes. Então, entrando com as coordenadas **X** e **Y** da imagem destino, calcula-se os valores correspondentes de **X** e **Y** na imagem original. O nível de cinza do ponto de coordenadas (**X**, **Y**) é então atribuído à posição (**X'**, **Y'**) na imagem destino. Este processo de mapeamento pode ser facilmente executado em paralelo, pois cada ponto, na imagem resultante, depende de apenas um ponto da imagem original.

A Figura 5.9 mostra um exemplo de *warping* utilizando imagem monocromática.

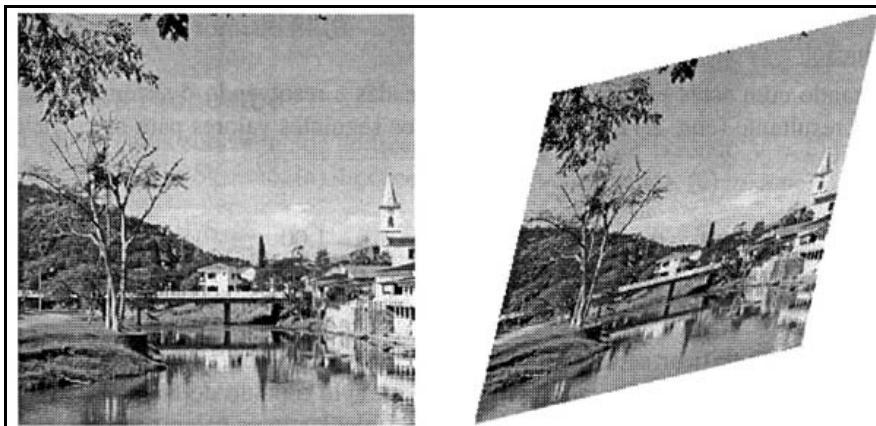


Figura 5.9 – Exemplo de *warping* de uma imagem monocromática utilizando padrão (template) em forma de losango.

- **Exemplo 5.2:**

Baseando-se na Figura 5.8, dadas as coordenadas originais dos vértices do quadrado e as coordenadas desejadas para o quadrado após o *warping*, indicadas na tabela a seguir e ilustradas na Figura 5.10, calcular os valores dos coeficientes **a**, **b**, **c**, **d**, **e**, **f**, **i** e **j** correspondentes à transformação desejada.

Ponto de controle	Coordenadas originais (X, Y)	Coordenadas após warping (X', Y')
1	(0, 0)	(1, 1)
2	(4, 4)	(3, 3)
3	(4, 0)	(4, 0)
4	(0, 4)	(0, 4)

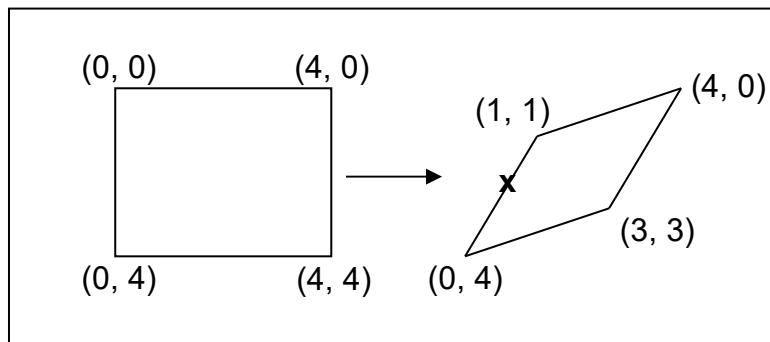


Figura 5.10 – Coordenadas dos pontos de controle antes e depois do warping

Entrando com estes valores para as coordenadas na equação da transformação projetiva, obtemos um sistema a 8 equações; resolvendo este sistema obtemos os seguintes valores para os coeficientes:

$$a = 0,75 \quad b = -0,25 \quad c = 1,00 \quad d = -0,25 \quad e = 0,75 \quad f = 1,00 \quad i = 0,00 \quad j = 0,00$$

Para verificar se os coeficientes calculados estão corretos, podemos escolher um ponto na imagem modificada, por exemplo o ponto de coordenadas (0,5, 2,5), indicando com um x na Figura 5.10. Calculando os valores de X e Y correspondentes a este ponto na imagem original, obteremos o par (0, 2), como esperado.

## 5.6 “ZOOM” DE UMA IMAGEM

Chamamos de *zoom* de uma imagem ao processo de fazer uma ampliação ou uma redução do domínio da imagem usando uma transformação de homotetia (Exemplo 5.1). No caso de expansão, temos o efeito de “*zoom in*”, e no caso da contração, o warping é conhecido pelo nome de “*zoom out*”.

A transformação de *zoom* de uma imagem tem duas peculiaridades que facilitam sua implementação:

- apesar de haver movimento relativo dos pixels, se a razão de escala for um número inteiro, haverá um alinhamento do reticulado da imagem original com o reticulado da imagem final;
- a razão de expansão é constante em todos os pixels.

O primeiro fato acima permite que possamos trabalhar diretamente com a imagem digital sem recorrer a um processo de reamostragem, desde que a razão de escalamento seja um número inteiro. O segundo fato permite o uso de filtros espacialmente invariantes no processamento da imagem.

### 5.6.1 “Zoom In” com o Filtro Box

O problema de se fazer um *zoom* de expansão de uma imagem requer um filtro de interpolação. Vamos discutir o processo utilizando um filtro *box* para reconstrução, quando a razão de escala é 2. Neste caso, a resolução horizontal e vertical da imagem é duplicada. Aplicando o método sucessivamente obtemos ampliações com razão 2,  $2^2$ ,  $2^4$ , etc.

Fazemos inicialmente uma extensão da imagem, acrescentando entre cada linha (e cada coluna) uma linha (coluna) de zeros. Em seguida, reconstruímos a nova imagem nos pixels acrescentados, fazendo a convolução com o filtro *box* de ordem 2, com máscara

1	1
1	1

Note que neste caso a máscara que define o filtro não tem média unitária. Esse fato garante que a média de intensidades na imagem ampliada é igual à média na imagem original.

É fácil verificar que esse processo de convolução é equivalente a reconstruir a imagem em cada pixel repetindo, para cada linha, o valor do pixel pelo valor do pixel anterior, e em seguida fazendo o mesmo procedimento para cada coluna. Por essa razão esse algoritmo de *zoom* com filtro *box* é conhecido na literatura pelo nome de ***zoom por replicação de pixels***.

Considere a imagem.

...	...	...
$f(i, j)$	$f(i, j + 1)$	...
$f(i + 1, j)$	$f(i + 1, j + 1)$	...
...	...	...

Acrescentando linhas e colunas de zeros, obtemos

...	...	...
$f(i, j)$	0	$f(i, j + 1)$
0	0	0
$f(i + 1, j)$	0	$f(i + 1, j + 1)$
...	...	...

Após o processo de filtragem temos a imagem reconstruída em tamanho duplicado.

...	...	...
$f(i, j)$	$f(i, j)$	$f(i, j + 1)$
$f(i, j)$	$f(i, j)$	$f(i, j + 1)$
$f(i + 1, j)$	$f(i + 1, j)$	$f(i + 1, j + 1)$
...	...	...

A Figura 5.11 mostra uma imagem e duas ampliações sucessivas utilizando o filtro *box*. Como aumentamos os detalhes da imagem a cada transformação de *zoom*, os defeitos de reconstrução inerentes ao filtro *box* ficam mais perceptíveis em cada imagem gerada. No entanto, esse algoritmo de *zoom* é

bastante popular devido à sua facilidade de implementação e eficiência computacional. Esse algoritmo é muito implementado em hardware para se obter uma ampliação rápida de uma imagem.



Figura 5.11 – Zoom de ordem 2 e 4 com o filtro box

### 5.6.2 “Zoom In” com o Filtro Triangular

A introdução de altas frequências no processo de reconstrução com o filtro de Bartlett é bem menos acentuada. A interpolação com esse filtro é conhecida na literatura de Computação Gráfica pelo nome de **zoom por interpolação bilinear**. Como no caso anterior, onde utilizamos o filtro box, estendemos a imagem acrescentando linhas e colunas de pixels com intensidade nula. Em seguida, a imagem é reconstruída nesses pixels fazendo interpolação linear sucessivamente entre os dois elementos vizinhos nas linhas e em seguida nas colunas.

Na Figura 5.12, mostramos duas ampliações sucessivas de uma imagem, usando o filtro triangular para reconstruir a imagem ampliada. O leitor deve comparar o resultado dessa figura com o resultado obtido usando o filtro box, na Figura 5.11.

Para grande parte das aplicações, principalmente na indústria de televisão, onde a resolução espacial e de cor das imagens é baixa, a ampliação com interpolação bilinear dá resultados bastante satisfatórios.



Figura 5.12 – Zoom de ordem 2 e 4 com interpolação bilinear

### 5.6.3 “Zoom Out” de uma Imagem

A transformação de *zoom* com contração provoca um aumento das altas frequências da imagem. Como sabemos, há o efeito de superposição no qual diversos pixels são colapsados em um único pixel. Devemos utilizar um filtro de suavização para minimizar os problemas de *aliasing* na imagem final. Como o fator de escala é constante, podemos utilizar um filtro espacialmente invariante. No caso do *zoom* de razão 2, cada bloco de ordem 2 da imagem.

$f(i, j)$	$f(i, j + 1)$
$f(i + 1, j)$	$f(i + 1, j + 1)$

é mapeado em um pixel da imagem final.

Usando o filtro *box* com máscara

$\frac{1}{4}$	1	1
	1	1

o valor desse pixel é dado por

$$\frac{1}{4} \sum_{i=1}^4 [f(i, j) + f(i+1, j) + f(i, j+1), f(i+1, j+1)]$$

Na Figura 5.13 mostramos o efeito do método acima aplicado duas vezes consecutivas em uma imagem.

Filtros de suavização mais eficientes podem ser utilizados para se obter melhores resultados na reconstrução da imagem final. É claro que esses filtros minimizam a introdução de altas frequências no processo de reconstrução, em troca de uma perda de definição da imagem reconstruída.



Figura 5.13 – “Zoom Out” consecutivo de uma imagem com filtro *box* de ordem 2

## 5.7 MORPHING

Uma categoria importante de filtros são os filtros de amplitude, que alteram apenas o espaço de cor da imagem. Nesse caso, tomamos uma transformação  $T: \mathbf{C} \rightarrow \mathbf{C}$  do espaço de cor da imagem  $f$ , e obtemos uma nova imagem  $g$ , fazendo a composição com  $T$ . Ou seja,  $g(x, y) = T(f(x, y))$ .

Os filtros de amplitude são bastante utilizados para se obter efeitos de transição entre duas imagens. Um exemplo clássico desse fato é a operação de

“cross-dissolve”, que transforma o gamute de uma imagem  $f$ , no gamute de uma imagem  $g$ , dada através de uma sequência de interpolações lineares do espaço de cor. Mais precisamente, temos um conjunto de transformações dependendo da variável  $t$ ,

$$h_t(x, y) = (1 - t) f(x, y) + t g(x, y)$$

onde para  $t = 0$ , temos a imagem original  $h_0(x, y) = f(x, y)$ , e para  $t = 1$ , temos a imagem final  $h_1(x, y) = g(x, y)$ .

Uma transformação de uma imagem, que combina simultaneamente transformação de domínio (*warping*) com transformação de amplitude (cor), é chamada de *morphing*. Como as transformações de *morphing* combinam mudanças de forma (filtros de *warping*) com mudanças na intensidade de cor (filtros de amplitude), podemos obter efeitos de transição entre imagens com qualidade extremamente superior ao efeito de *cross-dissolve*, uma vez que paralelamente à transformação de cor, utilizamos uma transformação de *warp* para obter um alinhamento de geometria.

Na Figura 5.14(c) mostramos um dissolve das imagens (a) e (b), onde a imagem em (a) tem um peso de 40%. Na Figura 5.14(d), mostramos uma outra combinação das imagens (a) e (b). Essa última combinação tem duas características distintas:

- Aplicamos um filtro de *warp* no domínio, de modo a obter um melhor alinhamento da geometria do rosto da mulher com a geometria da cara do tigre (note o alinhamento do contorno do rosto, da boca e do nariz);
- A transformação de dissolve é feita de forma adaptativa, ou seja, as funções de combinação não são constantes. Isso significa que o percentual de intensidade de cada imagem nos pixels, varia de pixel a pixel. Isso pode ser notado comparando com atenção a região do cabelo da mulher, no lado direito da imagem (d) com essas regiões na imagem (c), onde temos um dissolve tradicional.

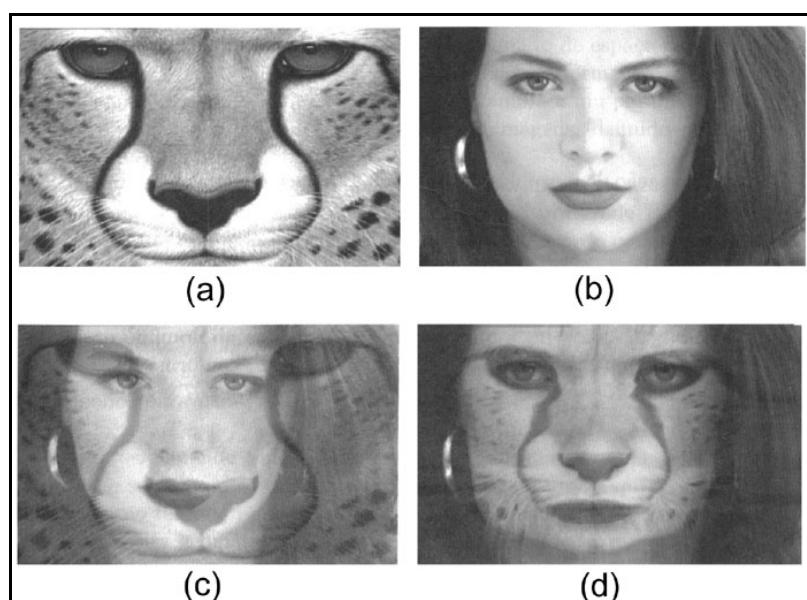


Figura 5.14 – *Morphing* entre duas imagens