João Luiz Lagôas de Almeida Bertolino

Análise Teórica e Prática:

Algoritmo de Colorização via Otimização

Rio de Janeiro, Brasil

Junho, 2014

João Luiz Lagôas de Almeida Bertolino

Análise Teórica e Prática:

Algoritmo de Colorização via Otimização

Monografia apresentada para obtenção do Grau
de Bacharel em Ciência da Computação pela
Universidade Federal do Rio de Janeiro.

Orientador:
Luziane Ferreira de Mendonça

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Rio de Janeiro, Brasil

Junho, 2014

Análise Teórica e Prática: Algoritmo de Colorização via Otimização

João Luiz Lagôas de Almeida Bertolino

Trabalho de Conclusão de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Autos do Trabalho:

_____

João Luiz Lagôas de Almeida Bertolino

Aprovado por:

_____

Profª. D.Sc. Luziane Ferreira de Mendonça
Universidade Federal do Rio de Janeiro
Orientadora

_____

Profª. D.Sc. Juliana Vianna Valério
Universidade Federal do Rio de Janeiro

_____

Prof. D.Sc. João Antônio Recio da Paixão
Universidade Federal do Rio de Janeiro

Resumo

Colorização é a arte de adicionar cores a uma imagem ou filme em preto e branco. Desde 1970 até os dias atuais, processos assistidos por computadores vêm sendo desenvolvidos com esse propósito.

No escopo da Computação Gráfica e Álgebra Linear, existem diversas abordagens com o intuito de colorir mídias em tons de cinza. Dentre elas, uma nos chamou atenção. Em agosto de 2004, um grupo de professores da Universidade Hebraica de Jerusalém apresentaram um algoritmo de colorização simples e intrigante: após rabiscar áreas específicas de uma imagem em preto e branco com traços coloridos, o método era capaz de propagar essas cores adequadamente para as regiões desejadas. Infelizmente, tal método apresenta pouco respaldo teórico disponível e, mesmo assim, ele é usado constantemente como referência em inúmeros artigos.

Este trabalho, portanto, tem por objetivo inicial expor um estudo teórico e prático do algoritmo de colorização via otimização. Por meio de toda a análise desenvolvida, mostramos métodos de resolução mais eficientes se comparados com a abordagem proposta pelos autores do mesmo. Indo além, propomos uma outra alternativa que se aplica ao algoritmo de forma a reduzir drasticamente o seu tempo de processamento.

Consolidamos então, esta monografia, não somente com uma ferramenta de edição gráfica prática que incorpora tal metodologia de colorização melhorada, mas que serve também como referência para futuros trabalhos.

Abstract

Colorization is the art of incorporating color to black-and-white image or vídeo. Since 1970, computer-assisted techniques have been developed to address this problem.

Within Computer Graphics and Linear Algebra, there are several methods with the objective of coloring media in tones of gray. Among them, a technique from the Hebrew University of Jerusalem drew our attention. The approach is simple and intriguing: after scribbling specific areas of a black-and-white image with color strokes, the method is capable of adequately propagating these colors throughout the image. Although the method does not have a strong theoretical backing, it is still is used as reference in a large number of works.

Therefore, this work has the initial objective of introducing a theoretical and practical study of the abovementioned colorization technique. By means of our analysis, we present methods of comparatively more efficient resolution. Furthermore, we propose an alternative that drastically reduces the running time of the algorithm.

We thus consolidate in this monography, not only a practical tool for image colorization that takes advantage of this improved colorization methodology, but also as theoretical reference for future works.

Agradecimentos

Gostaria de agradecer primeiramente a todos os membros da minha família que, de alguma forma, me incentivaram na constante busca pelo conhecimento. Em especial aos meus pais Cláudia Lagôas e Alexandre Bertolino por todo o esforço, incentivo e investimento na minha educação e formação pessoal. Aos meus irmãos André Luiz Lagôas e Ana Luiza Lagôas pelo apoio e paciência durante essa etapa da minha vida.

Ao colega de faculdade Henrique Souza por todo o seu companheirismo. As noites perdidas no Skype realizando tarefas, as horas de tensão antes das provas e, não menos importante, os momentos de descontração, fizeram dessa caminhada memorável.

Agradeço também a todos os professores que tive e fizeram diferença, em algum grau, no meu desenvolvimento educacional, desde a alfabetização, até o fim da graduação. À minha alfabetizadora tia Beth, à minha professora de terceira série tia Gilma, ao meu professor de terceiro ano Alex e, principalmente, à minha orientadora Luziane - seu papel como educadora é um exemplo a ser seguido.

Por fim, a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Sumário

1. Introdução

1.1 Contextualização e Motivação

O termo "colorização" foi introduzido em 1970 e posteriormente patenteado por Wilson Markle [1]. Seu significado define o processo de transferir informações de cores para algum material monocromático[1], comumente, imagens ou filmes em tons de cinza.

Algumas décadas atrás, o processo de colorização não era bem visto e se tornou impopular devido a opinião de que arruinava trabalhos originais [3]. No entanto, o inegável fato de que a presença de cores aumenta a impressão artística de obras em vários aspectos se manteve como principal motivação para que essa mudança fosse aceita [4].

Por meio de cores, podemos aumentar o apelo visual de figuras ou cenas em preto e branco. Fotos antigas ganham vida, filmes clássicos se tornam modernos e até mesmo resultados científicos ou médicos se beneficiam quando coloridos. Exemplificando, imagens obtidas por meio de Ressonância Magnética (RM), raios-X e tomografia computadorizada (TC) são naturalmente monocromáticas [5]. Contudo, por meio de um processo de colorização, podemos reforçar particularidades da gravura para fins de apresentação e demonstração.

Nos últimos anos, diversas técnicas avançadas e eficientes de colorização foram desenvolvidas. Essas técnicas podem ser baseadas em diferentes abordagens como: transferência de cores [6], analogia entre imagens [7], segmentação [8], previsão de cores [9], relação probabilística [10], entre muitas outras.

Ainda assim, apesar dos inúmeros métodos já propostos, a grande maioria deles requer considerável intervenção do usuário ou apresentam a necessidade de ter-se em mãos recursos muito específicos (imagens muito parecidas ou referências altamente detalhadas, por exemplo). Dentro desse contexto, o processo de colorização acaba tornando-se uma tarefa tediosa e de alto consumo de tempo.

[1] Imagens em escala de cinza são também chamadas de monocromáticas, denotando a presença de uma (mono) cor (croma) [2].

1.2 Trabalho e Objetivo

Em agosto de 2004, três professores e pesquisadores da Universidade Hebraica de Jerusalém, Anat Levin, Dani Lischinski e Yair Weiss, apresentaram na SIGGRAPH (*Special Interest Group on Graphics and Interactive Techniques*) - conferência anual sobre computação gráfica convocada pela organização ACM SIGGRAPH - um novo método de *colorização via otimização*. Tratava-se de um algoritmo matemático que automatizava o processo de adição de cores a imagens ou vídeos monocromáticos reduzindo drasticamente a quantidade de intervenção do usuário. O artigo referente ao método [11] é encontrado em [12].

O processo de colorização proposto vai além da detecção manual de regiões. Nessa abordagem, ao invés de delinear fronteiras com precisão, o usuário precisa apenas inserir na imagem alguns rabiscos coloridos. A partir daí, o algoritmo é capaz de propagar as cores indicadas nos traços e produzir uma imagem completamente colorida.

O método se baseia na simples premissa de que pixels vizinhos com intensidades semelhantes devem ter cores semelhantes. Essa premissa é formalizada usando uma função de custo quadrático e obtemos assim um problema de otimização não linear que pode ser resolvido eficientemente por técnicas padrões.

Fazendo uma pesquisa rápida, é possível perceber que várias novas abordagens de colorização utilizam tal método de alguma forma: seja como base, etapa de um processo particular ou referência. No entanto, pouco conteúdo é explicado e detalhado, tornando a análise teórica que envolve o método de colorização via otimização mais trabalhosa. De domínio público, só existem atividades fazendo uso do algoritmo, mas nenhuma delas se preocupa em estudar a fundo o funcionamento de tal algoritmo.

Diante disso, neste trabalho decidimos reproduzir o método de colorização via otimização apresentado por Anat Levin, Dani Lischinski e Yair Weiss, particularmente para imagens. Indo além do que é exposto no artigo [11], fazemos uma análise matemática de seus detalhes com o objetivo de especificar melhores caminhos para a resolução do problema. Realizamos também variações em algumas componentes da função objetivo obtida para fins de comparação e com o intuito de aferir novos resultados. Dessa forma, por meio de todo o respaldo teórico elaborado, determinamos uma maneira mais rápida e eficiente de executar o algoritmo.

Não obstante, o penúltimo capítulo deste trabalho vai além do escopo inicial do anteprojeto e incorpora todo o estudo que é feito nos capítulos antecedentes. Propomos a integração de uma ideia nossa capaz de tornar o algoritmo de colorização via otimização ainda mais eficiente. Nele, assuntos como filtros espaciais, de alcance, bilaterais e operações de redimensionamento são explicitados e auxiliam na construção de uma aplicação de colorização mais prática e atraente.

1.3 Estrutura da Monografia

Um processo de otimização pode ser dividido em duas etapas: a *modelagem* e a *resolução*. A primeira etapa consiste em construir um modelo matemático apropriado de acordo com as necessidades do problema, isto é, identificar uma função objetivo, variáveis e restrições. Já a segunda, uma vez que o modelo tenha sido formulado, determina e faz uso de um algoritmo de otimização adequado na resolução do problema [13].

Deste modo, decidimos organizar este trabalho em cinco capítulos. O primeiro, como já observado, introduz e estabelece o tema de colorização esclarecendo os objetivos a serem alcançados e apresentando de que forma esta monografia está disposta.

O segundo capítulo tem como escopo descrever a etapa de modelagem do problema de colorização via otimização. Inicialmente, apresentamos breves definições de imagens em tons de cinza e espaços de cores para melhor entendimento do assunto. Depois realizamos a formalização do problema seguida da descrição formal da nossa maneira de modelá-lo. Concluindo o capítulo, determinamos a formulação final e, assim, estaremos aptos a dar seguimento iniciando a etapa de resolução.

O terceiro capítulo se preocupa em indicar um algoritmo de otimização apropriado para resolver o problema de otimização obtido. Para isso, a princípio levantamos questões que envolvem um problema de minimização genérico bem como aspectos de funções quadráticas definidas em $\mathbb{R}^n$. Prontamente, classificamos a função objetivo do problema estudando suas características. Analisamos métodos de resolução apropriados e determinamos quais benefícios cada um deles pode nos proporcionar. Não somente, estudamos a forma de resolução adotada pelos autores em [11] fazendo um comparativo com a nossa. Uma implementação do algoritmo, deste modo, é sugerida e então estamos prontos para realizar testes e estudar resultados.

O quarto capítulo é a parte do trabalho que avalia os resultados. Nele, começamos explicitando questões técnicas e de implementação. Recordamos o leitor sobre todos os parâmetros que foram definidos ao longo da modelagem e resolução do problema e, assim, fazemos modificações nos mesmos visando estudar suas implicações nas saídas geradas - em termos de qualidade e tempo de processamento. Seguidamente, confrontaremos a nossa implementação do algoritmo com a proposta pelos autores em [12] verificando um ganho significativo de velocidade de execução da nossa abordagem. O Capítulo 4 é então encerrado expondo uma coleção de diversos tipos de imagens (gravuras de paisagens, artes digitais, fotografias antigas, entre outras) que foram coloridas pela nossa aplicação.

O quinto capítulo tem por objetivo aplicar ao método de colorização via otimização uma ideia nossa capaz de aumentar sua eficiência. Para isso, introduzimos a princípio assuntos de Computação Gráfica que nos levam a formalização da nossa sugestão: produzir uma solução aproximada para uma imagem de alta resolução por meio da mesma em baixa resolução. Dessa forma, mostramos que é possível alcançar ótimos resultados diminuindo consideravelmente o tempo de execução. Por fim, sugerimos um último extra a nível de implementação que torna a aplicação mais ostensiva.

O sexto e último capítulo conclui todo o estudo sucedido. Nele constam os objetivos de pesquisa e como eles foram alcançados, uma descrição metodológica das etapas de desenvolvimento, dificuldades encontradas e um conjunto de novas ideias que podem ser utilizadas e elaboradas em trabalhos futuros. As considerações finais, então, são dadas na forma de uma discussão sobre os benefícios pessoais que este trabalho nos proporcionou.

2. Formulação

Antes de partimos para a formalização do problema e iniciarmos discussões acerca da modelagem da função objetivo proposta no artigo [11], é necessário o clareamento de alguns conceitos relacionados. Os próximos dois itens desta seção explicam de forma sintética definições fundamentais como: o que é a intensidade numa imagem e a importância de um espaço de cor.

2.1 Imagens em Preto e Branco - Definindo Intensidade

Em fotografia e computação, uma imagem em tons de cinza é tal que o valor de cada pixel carrega uma única informação: a *intensidade* [14]. Imagens desse tipo, mais conhecidas como gravuras em preto e branco, contêm apenas tonalidades de preto, branco e cinza. Todas essas tonalidades podem ser obtidas simplesmente variando o nível de brilho, isto é, variando o nível de intensidade.

A intensidade de um pixel, portanto, define o tom de cinza e pode ser expressa por um valor dentro de uma faixa predeterminada. Em um modelo exemplificativo, esse valor poderia corresponder a qualquer fração entre 0 (ausência total de intensidade, preto) e 1 (presença total de intensidade, branco).

Figura 2.1: Níveis de intensidade variando entre seu mínimo e máximo.

Note que essa notação é apenas usada para simplificar a noção de intensidade em artigos acadêmicos. Elas não definem o que de fato as cores preto e branco representam em termos de colorimetria[2].

[2] Colorimetria é a ciência e tecnologia usada para quantificar e descrever fisicamente a percepção humana das cores [15].

2.2 Espaços de Cor

Espaço de cor é um espaço geométrico tridimensional com eixos apropriadamente definidos de modo que as representações para todas as possíveis percepções de cores dos seres humanos se encaixem dentro dele. Nesse espaço, cada cor é interpretada como um ponto [16].

Em um sentido mais descomplicado da definição acima, podemos dizer que espaços de cor definem uma gama de cores e tonalidades disponíveis segundo algum critério. Como seres humanos, podemos definir uma cor por seus atributos de luminosidade (claridade), matiz (tonalidade) e saturação (vivacidade). Um computador pode descrever uma cor usando quantidades de vermelho, verde e azul. Uma impressora pode produzir uma cor específica em termos da reflexão e absorção de tintas ciano, magenta, amarela e preta numa folha de papel [17].

O principal objetivo do espaço de cor é, então, nos auxiliar a descrever cores entre pessoas, máquinas ou programas. Por meio de modelos, as cores podem ser representadas por tuplas de números tipicamente compostas por três valores chamados de componentes. Essas componentes indicam a posição de uma determinada cor dentro de um espaço de cor.

Ao longo dos anos, diversos desses espaços foram criados com o propósito de atender determinadas necessidades ou limitações. Existem espaços de cor baseados no sistema visual humano (*Red, Green and Blue* - RGB, *Hue, Saturation and Value* - HSV, *Hue, Saturation and Lightness* - HSL), específicos de aplicações, dispositivos e sistemas (YUV, YIQ, CMY(K)), além dos espaços de cor determinados pela CIE (*International Commission on Illumination*) (CIE XYZ, Lab e Luv) [18].

Para o propósito deste trabalho, dois desses espaços nos interessam. São eles RGB e YUV. Os próximos dois itens desta seção descrevem um pouco seus aspectos assim como formas de se converter uma determinada cor de um espaço para outro.

2.2.1 Espaço de Cor RGB (Vermelho, Verde e Azul)

O espaço de cor RGB é amplamente utilizado no meio da computação gráfica. Vermelho, verde e azul são três cores aditivas primárias (elementos individuais que são misturados para formar uma cor desejada) e nesse espaço representam as três componentes [19]. Seus valores costumam variar dentro de uma faixa que vai de 0 até 255.

O espaço geométrico tridimensional pode ser obtido tratando os valores das componentes como coordenadas cartesianas comuns em um espaço euclidiano. Para o espaço de cor RGB, esse volume tridimensional pode ser representado por um cubo como mostrado na Figura 2.2. Note que o valor de intensidade mínima (preto) ocorre no vértice $(0,0,0)$, o valor de intensidade máxima (branco) ocorre no vértice $(255,255,255)$ e a diagonal que liga esses dois pontos representa os vários tons de cinza.

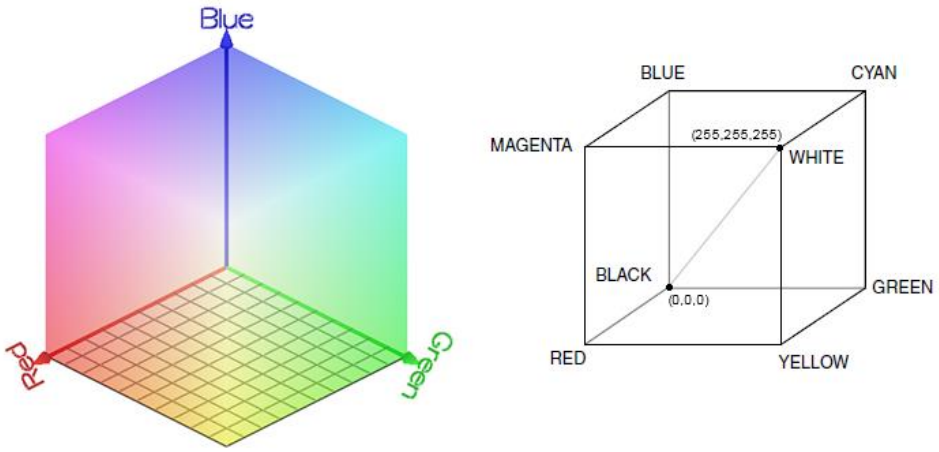


Figura 2.2: Espaço de cor RGB.

O espaço RGB é a escolha mais prevalecente no meio digital uma vez que grande parte dos monitores costumam produzir todas as cores necessárias por meio de vermelho, verde e azul [19]. Portanto, escolhê-lo simplifica a arquitetura e o projeto de um sistema ou aplicação.

Mesmo assim, esse espaço não costuma ser muito eficiente quando queremos trabalhar separadamente com tons de cinza e cores. Por exemplo, suponha que queremos modificar a intensidade e a cor de um determinado pixel separadamente. Seria necessário ler as três componentes, calcular a intensidade e a cor, fazer as modificações necessárias nesses valores e recalcular as novas componentes de vermelho, verde e azul. Se fosse possível acessar diretamente um valor que armazenasse a intensidade e outro que armazenasse a cor, esse processo seria mais rápido e eficiente.

Muitos padrões de vídeo apresentam essa necessidade, isto é, armazenar o valor da luminância[3] e crominância[4] de forma separada. Os espaços de cor mais comuns que atendem essa necessidade são YUV e YIQ.

[3] A luminância é a informação de brilho (intensidade) da luz presente em um sinal de vídeo [20].
[4] A crominância é a informação de cor presente em um sinal de vídeo [20].

2.2.2 Espaço de Cor YUV e YIQ (Luminância e Crominância)

Os espaços de cor YUV e YIQ são muito presentes no meio da transmissão de televisão. Enquanto que Y representa a componente de luminância - ao qual nós iremos nos referir simplesmente como intensidade - U e V, assim como I e Q, representam as componentes de crominância - responsáveis por codificar a cor.

A principal característica desses espaços é a capacidade de separar uma cor codificada em RGB em uma componente de intensidade (tons de cinza) e duas componentes de cor. No sistema de transmissão Europeu (PAL), os sinais RGB são convertidos para o espaço YUV pelas seguintes equações [21]:

$$Y = 0.299R + 0.587G + 0.114\text{B}, \tag{2.1}$$

$$U = -0.147R - 0.289G + 0.436B, \tag{2.2}$$

$$V = 0.615R - 0.515G - 0.100B. \tag{2.3}$$

Já a conversão de YUV para RGB é dada por:

$$R = Y + 1.140V, \tag{2.4}$$

$$G = Y - 0.395U - 0.581V, \tag{2.5}$$

$$B = Y + 2.032U. \tag{2.6}$$

Para valores de RGB definidos em um intervalo que varia de 0 a 255, temos Y variando de 0 a 255, U variando de -112 a 112 e V variando de -157 a 157. Com isso, o espaço de cor YUV pode ser visualizado na Figura 2.3. Note que para esses valores, o espaço formado não caracteriza um cubo como em RGB.

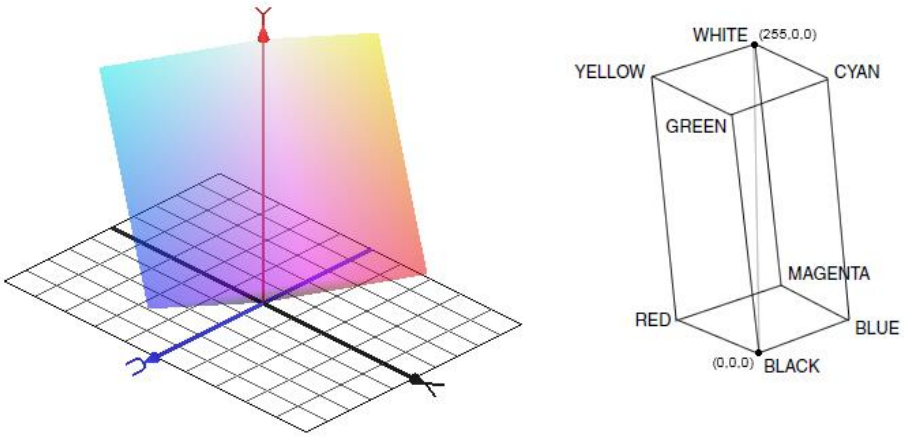


Figura 2.3: Espaço de cor YUV.

Ainda na Figura 2.3, podemos verificar que o valor de intensidade mínima (preto) ocorre no vértice $(0,0,0)$, o valor de intensidade máxima (branco) ocorre no vértice $(255,0,0)$ e a linha que liga esses dois pontos representa os vários tons de cinza, ou seja, o eixo Y.

Observe agora a Figura 2.4. Sua primeira imagem mostra a representação do espaço de cor YUV quando visto do seu lado mais escuro (quando $Y = 0$). Repare que o meio da região é completamente preto, onde U e V valem 0 e Y também. A medida que os valores de U e V movem-se em direção aos limites do volume, podemos começar a perceber o efeito das cores.

Já a segunda imagem da Figura 2.4 mostra a mesma região vista do seu lado mais claro (quando $Y = 255$). O raciocínio é análogo. O meio da região é completamente branco, onde U e V valem 0 e Y é igual a 255. A medida que os valores de U e V se afastam do eixo Y, as cores começam a se mostrar de forma mais evidente.

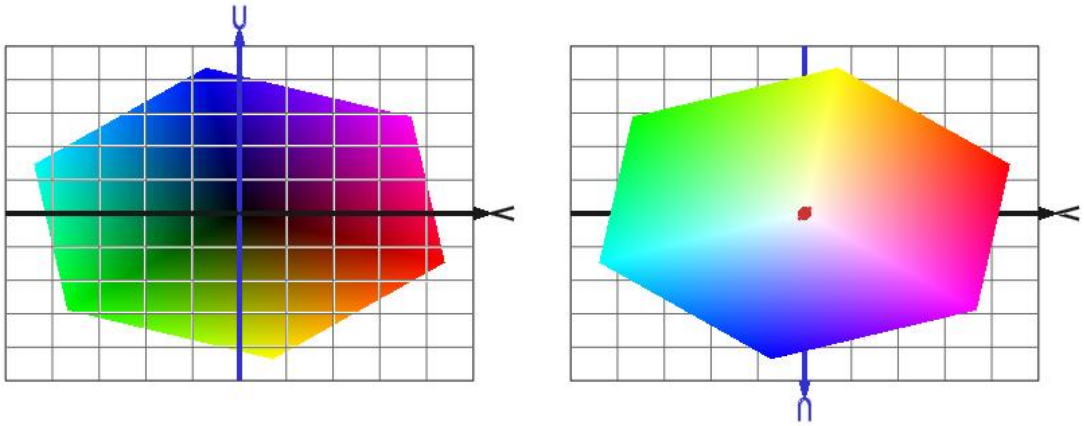


Figura 2.4: YUV visto do seu lado mais escuro e mais claro, respectivamente.

Da mesma forma, no sistema de transmissão americano (NTSC), os sinais RGB são convertidos no espaço YIQ de acordo com as seguintes equações [21]:

$$I = 0.596R - 0.275G - 0.321B = Vcos(33°) - Usen(33°), \quad (2.7)$$

$$Q = 0.212R - 0.523G + 0.311B = Vsen(33°) + Ucos(33°). \quad (2.8)$$

Enquanto que a componente Y é definida da mesma maneira, as componentes U e I, assim como V e Q, diferem por uma rotação de 33°. Por esse motivo, YUV e YIQ são considerados *espaços análogos*. Para mais informações detalhadas a respeito desses espaços de cor e suas correspondências, veja [22].

2.3 Formalização e Modelagem

Dada uma imagem em tons de cinza, o problema de colorização estudado consiste em gerar uma imagem colorida a partir de um conjunto de traços coloridos inseridos na imagem original, via técnicas de otimização.

Para tanto, as cores dos traços devem se propagar para as demais regiões de forma que a intensidade dos pixels coloridos e não coloridos seja semelhante. A Figura 2.4 ilustra esse objetivo.

Figura 2.4: Imagem rabiscada (esquerda), imagem produzida pelo método (meio) e imagem original (direita).

Repare que a própria descrição do problema sugere a conveniência de termos separado a intensidade e a cor de um pixel. O espaço de cor adequado para uma formalização matemática, portanto, seria YUV ou YIQ. Posto que esses espaços são análogos, a escolha de qualquer um deles é apropriada. Neste trabalho, assim como no artigo [11], utilizamos YUV. Vimos que, enquanto a componente Y representa a intensidade de um pixel - e por conta disso, é conhecida para todos os pixels em uma imagem em preto e branco -, as componentes U e V codificam sua cor. Dessa forma, estamos interessados em determinar as componentes U e V de todos os pixels.

Dito isso, o próximo passo é entender como a função objetivo foi estabelecida. Para tal, a partir daqui estaremos sempre considerando uma imagem de tamanho $w \times h$ composta por $n = w.h$ pixels, onde cada pixel é identificado por um inteiro entre 1 e n.

2.3.1 Função Objetivo

Como mencionado na introdução, nós queremos impor a condição de que dois pixels vizinhos tenham cores similares caso suas intensidades sejam parecidas. Portanto, nós estamos interessados em minimizar a diferença entre a cor (componentes U e V) de um dado pixel com

a média ponderada - medida em termos das intensidades (componente Y) - das cores dos pixels vizinhos.

Uma vez que essa função mede uma diferença para cada pixel da imagem, uma escolha adequada é o somatório das diferenças. Além disso, para evitar que essas subtrações resultem em valores negativos, é natural nesse tipo de situação fazer uso de soma de quadrados. Sendo assim, obtemos as seguintes funções quadráticas:

$$J(U) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \frac{w_{rs} U_s}{\sum_{s\prime \in N(r)} w_{rs\prime}} \right)^2, \tag{2.9}$$

$$J(V) = \sum_{r \in P} \left(V_r - \sum_{s \in N(r)} \frac{w_{rs} V_s}{\sum_{s\prime \in N(r)} w_{rs\prime}} \right)^2, \tag{2.10}$$

onde P é o conjunto de todos os n pixels da imagem, $N(r)$ é o conjunto de todos os pixels vizinhos (pixels adjacentes) de r, U_r e V_r são as componentes de cor do pixel r e, finalmente, w_{rs} é uma função de peso - grande quando as intensidades dos pixels r e s são parecidas e pequena quando diferentes.

As duas funções descritas acima são aprimoramentos das proposta em [11] incorporando a normalização dos pesos para as componentes U_s e V_s. Poderíamos denotar

$$\bar{w}_{rs} = \frac{w_{rs}}{\sum_{s\prime \in N(r)} w_{rs\prime}}$$

tal que

$$\sum_{s \in N(r)} \bar{w}_{rs} = 1,$$

e reescrever as Equações (2.9) e (2.10) como

$$J(U) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \bar{w}_{rs} U_s \right)^2,$$

$$J(V) = \sum_{r \in P} \left(V_r - \sum_{s \in N(r)} \bar{w}_{rs} V_s \right)^2.$$

Além disso, com o objetivo de facilitar a análise do comportamento dessas funções, decidimos reescrevê-las utilizando notação matricial conforme vemos a seguir:

$$J(U) = (U - WU)^T(U - WU), \tag{2.11}$$

$$J(V) = (V - WV)^T(V - WV). \tag{2.12}$$

Nas Equações acima temos que U e V denotam vetores de tamanho n e W denota uma matriz de tamanho $n \times n$ tal que cada elemento $W = \{a_{ij}\}_{i,j}^{n}$ é dado por

$$a_{ij} = \begin{cases} \bar{w}_{ij}, & se\ j \in N(i) \\ 0, & se\ j \notin N(i) \end{cases}. \tag{2.13}$$

É importante perceber que W será uma matriz esparsa. Repare que a quantidade de valores não nulos armazenados em cada linha i de W é igual ao número de vizinhos do pixel i. Se chamarmos de R o raio da vizinhança, então para $R = 1$ (um total de 8 pixels adjacentes) e uma imagem de dimensão 32×32 pixels, a estrutura dessa matriz é mostrada na Figura 2.5.

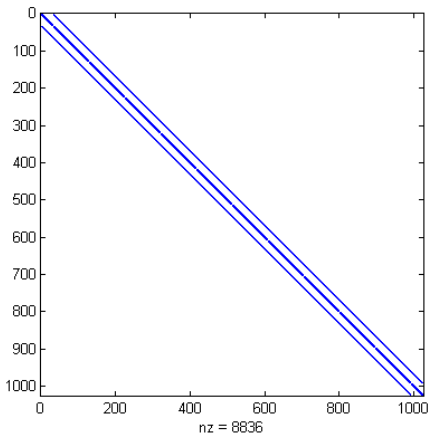


Figura 2.5: Estrutura da matriz esparsa W.

Temos com isso as definições das funções objetivo (2.9) e (2.10) em um formato muito mais prático. Assim, supostamente estamos preparados para atacar o problema de otimização, isto é:

$$\text{minimizar } J(U), \tag{2.14}$$

$$\text{minimizar } J(V). \tag{2.15}$$

No entanto, há dois pontos que merecem atenção. O primeiro trata da função de peso w_{rs}. Que tipo de comportamento essa função deve assegurar? O segundo diz respeito às variáveis do problema. O que de fato é variável e constante nessas funções?

Daqui em diante, todo o desenvolvimento matemático relacionado com as funções objetivo será narrado apenas em termos da Função (2.9) responsável pela componente U. O raciocínio que será desdobrado é idêntico para (2.10) e por isso será omitido. Vale também lembrar que o estudo que está sendo realizado foi por nós efetuado, uma vez que no artigo [11] os autores se limitam a fornecer apenas uma breve explicação das Equações (2.9) e (2.10).

2.3.2 Função de Peso

Uma função de peso w é uma função positiva tal que w: $A \rightarrow \mathbb{R}^+$ onde A é um conjunto discreto de elementos. Seu principal propósito é aumentar a influência (aumentar o "peso") de determinados elementos em um dado conjunto [23]. Por conta disso, esse tipo de função costuma ser usado em somas na produção de pesos relativos de médias ponderadas.

Na função objetivo apresentada em (2.9), podemos perceber que cada parcela do somatório é composta pela diferença entre a componente U de um pixel r e a média ponderada da componente U de cada pixel s vizinho de r. A função de peso w_{rs}, nesse contexto, é responsável por ponderar essa média impondo a condição de que dois pixels vizinhos r e s tenham cores parecidas se suas intensidades forem semelhantes. A Figura 2.5 mostra esse comportamento para o raio de vizinhança igual a 1. A espessura das linhas determina a magnitude do peso.

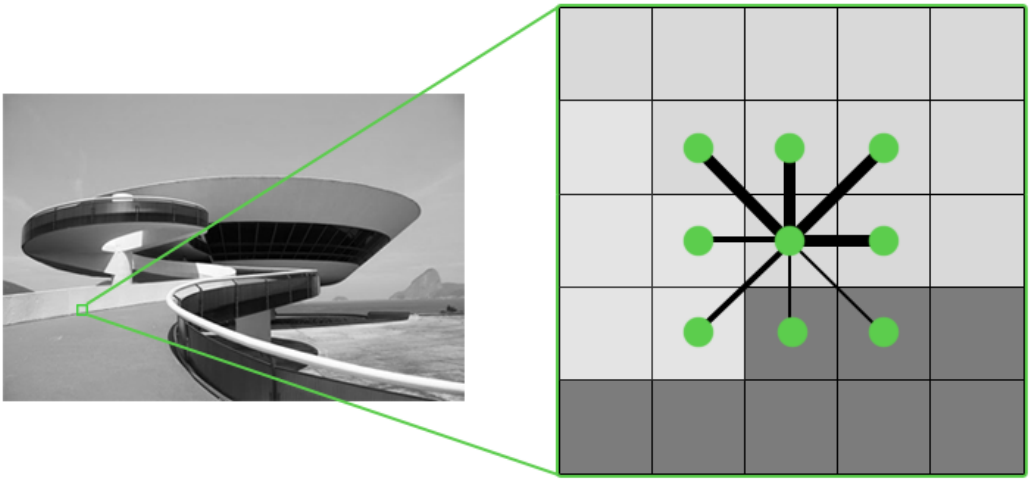


Figura 2.5: Peso relativo de cada pixel vizinho de acordo com suas intensidades para $R = 1$.

Quando falamos em uma função de peso responsável por medir o quão fortemente pares de pixels adjacentes estão juntos numa imagem, estamos, em outras palavras, tentando estabelecer um nível de afinidade entre esses pixels. Esse tipo de função é usado extensivamente em algoritmos de segmentação de imagens [24] e é usualmente referido como *função de afinidade*.

Em [11], duas funções de peso foram propostas. A mais simples é comumente utilizada em algoritmos de segmentação de imagens e é baseada no quadrado da diferença entre duas intensidades. Sua equação é dada por:

$$w_{rs} = e^{-(Y_r - Y_s)^2 / \sigma_r{}^2}. \tag{2.16}$$

Essa equação nada mais é do que uma função Gaussiana. A "curva de sino" característica de seu gráfico nos fornece o comportamento que a nossa função de afinidade w_{rs} deve apresentar: uma transição suave que leva em consideração a diferença de intensidades entre dois pixels vizinhos.

Já a segunda função é baseada na correlação normalizada entre duas intensidades:

$$w_{rs} = 1 + \frac{1}{\sigma_r^2}(Y_r - \mu_r)(Y_s - \mu_r). \tag{2.17}$$

Não entraremos no mérito de discuti-la uma vez que os próprios autores a apontam como sendo qualitativamente e eficientemente inferior.

Em ambos os casos, Y_r representa a componente de intensidade de um pixel r enquanto que μ_r e $\sigma_r{}^2$ representam a média e a variância das intensidades, respectivamente, na vizinhança desse pixel.

Além dessas funções, propomos uma terceira alternativa semelhante com a Equação (2.16):

$$w_{rs} = e^{-(Y_r - Y_s)^2 / t}. \tag{2.18}$$

O diferencial aqui é a substituição da variância por um fator de normalização constante t ao qual chamaremos de *valor de limiar*.

Propomos esta função por dois motivos: oferecer um grau de liberdade ao usuário - que poderá determinar qual valor t deve assumir - e evitar o cálculo de $\sigma_r{}^2$. A Figura 2.6 mostra o

comportamento da Função (2.18) para valores de limiar iguais a 10 (valor pequeno que ${\sigma_r}^2$ pode assumir), e 16256 (variância máxima aproximada das intensidades em uma vizinhança), respectivamente. O eixo vertical representa w_{rs} enquanto que o eixo horizontal a diferença $d = Y_r - Y_s$.

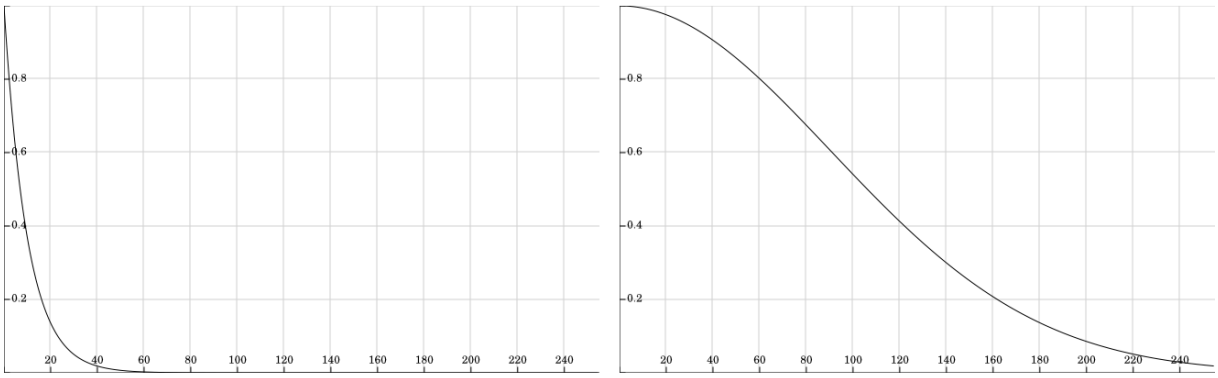


Figura 2.6: Gráficos para $t = 10$ (esquerda) e $t = 16256$ (direita).

Repare que quando $t = 16256$, menos sensível o peso se mostra frente à diferença de intensidades. Já para $t = 10$, a função de peso tende a ser mais rigorosa diante dessa diferença. Isso sugere que para gravuras com transições muito bem definidas, um valor de t pequeno se comportará de forma mais adequada enquanto que, para imagens mais desfocadas, valores de t maiores seriam mais apropriados. No Capítulo 4, onde discutimos os testes realizados, verificaremos como os resultados reagem a cada uma dessas funções de afinidade.

2.3.3 Variáveis

Para identificar as variáveis do problema, vamos atentar à descrição dada no início da Seção 2.3 e a função objetivo definida na Equação (2.11).

Estamos interessados em descobrir as cores de cada pixel não colorido, isto é, determinar a componente U de todos os pixels que não foram rabiscados. Repare que os pixels que receberam um rabisco não são variáveis. Esses pixels adquiriram uma cor (a componente U da cor do rabisco) e, portanto, são constantes do problema.

Na função objetivo, isso significa que o vetor U é composto de elementos desconhecidos (os pixels não rabiscados - variáveis) e elementos conhecidos (os pixels rabiscados - constantes). Exemplificando, para uma imagem com n pixels, onde os k primeiros pixels receberam um rabisco, teríamos que:

$$U = [\widetilde{U}_1 \quad \widetilde{U}_2 \quad \cdots \quad \widetilde{U}_k \quad U_{k+1} \quad \cdots \quad U_{n-1} \quad U_n]^T \tag{2.19}$$

onde $\widetilde{U}_x$ é a componente U da cor do pixel x rabiscado.

No entanto, é um inconveniente as variáveis e as constantes serem representadas da mesma maneira. Uma alternativa de resolver tal situação é adicionar uma restrição de igualdade no problema de minimização dado por (2.14). Feito isso, estamos interessados em resolver o seguinte problema:

$$\text{minimizar } J(U) \text{ sujeito à } U_k = \widetilde{U}_k \text{ para } k \in K \tag{2.20}$$

onde K é o conjunto de pixels que foram rabiscados.

Restrições de igualdade costumam ser tratadas por métodos de penalidade. Esses métodos tentam aproximar um problema de otimização restrito em um problema de otimização irrestrito. A aproximação é feita adicionando um novo termo à função objetivo que prescreve um alto custo caso a restrição seja violada [25]. A ideia, portanto, é substituir o problema (2.20) por uma aproximação irrestrita da forma:

$$\text{minimizar } J(U) + \lambda P(U) \tag{2.21}$$

onde λ é um escalar positivo conhecido como *parâmetro de penalidade* e P é uma função em $\mathbb{R}^n$ chamada de *função de penalidade* responsável por incorporar a restrição.

O próximo passo é, portanto, determinar uma função de penalidade adequada. Em casos em que a restrição é definida por um número de igualdades, uma função de penalidade muito utilizada é a soma das violações ao quadrado:

$$P(U) = \sum_{k \in K} (U_k - \widetilde{U}_k)^2. \tag{2.22}$$

Combinando o problema de minimização (2.21) com a equação (2.22) geramos uma função objetivo quadrática aumentada dada por:

$$\theta(U, \lambda) \equiv \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \overline{w}_{rs} U_s \right)^2 + \lambda \sum_{k \in K} \left(U_k - \widetilde{U}_k \right)^2. \tag{2.23}$$

Dessa forma, cada restrição não satisfeita influência $\theta(U, \lambda)$ acrescentando uma penalidade igual ao quadrado da infração. Essas penalidades são somadas e multiplicadas por λ. Obviamente, essa influência é contrabalanceada por $J(U)$ e pela ordem de grandeza de λ, no entanto, abordaremos essa questão também no Capítulo 4 onde falamos sobre os resultados.

Em notação matricial, a Equação (2.23) pode ser escrita de uma forma muito mais simples:

$$\theta(U,\lambda) \equiv (U - WU)^T(U - WU) + \lambda(U - \widetilde{U})^T C(U - \widetilde{U}), \tag{2.24}$$

onde $\widetilde{U}$ denota um vetor coluna de tamanho n armazenando a componente U dos pixels rabiscados, isto é, $\widetilde{U} = \{\widetilde{U}_k\}_k^n$ tal que

$$\widetilde{U}_k = \begin{cases} \widetilde{U}_k, & se\ k \in K \\ 0, & se\ k \notin K \end{cases}, \tag{2.25}$$

e C denota uma matriz diagonal de tamanho $n \times n$ da qual cada elemento $C = \{c_{ii}\}_{i,i}^n$ é dado por

$$c_{ii} = \begin{cases} 1, & se\ i \in K \\ 0, & se\ i \notin K \end{cases}. \tag{2.26}$$

Com isso, respondemos as duas questões que foram abertas - a caracterização da função de peso e a formalização das variáveis. Mantendo a simplicidade conceitual e a praticidade da função original, fomos capazes de determinar uma nova função, ainda quadrática, com n variáveis bem definidas.

2.3.4 Formulação Final

Na Subseção 2.3.1, descrevemos uma forma de determinar a função objetivo e levantamos questões acerca de detalhes não explicitados no artigo [11]. Já na Subseção 2.3.2, mostramos as funções de peso sugeridas pelos autores e propomos uma nova função a ser testada. Por fim, na Subseção 2.3.3, determinamos uma maneira amigável de reescrever a função objetivo discretizando as variáveis e as constantes do problema. Assim sendo, abaixo apresentamos a formulação final para o nosso problema de minimização:

$$\min \theta(U,\lambda) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \overline{w}_{rs} U_s \right)^2 + \lambda \sum_{k \in K} \left(U_k - \widetilde{U}_k \right)^2, \tag{2.27}$$

$$\min \theta(V,\lambda) = \sum_{r \in P} \left(V_r - \sum_{s \in N(r)} \overline{w}_{rs} V_s \right)^2 + \lambda \sum_{k \in K} \left(V_k - \widetilde{V}_k \right)^2. \tag{2.28}$$

Em notação matricial, os mesmos problemas podem ser escritos da forma:

$$\min \theta(U,\lambda) = (U - WU)^T(U - WU) + \lambda(U - \widetilde{U})^T C(U - \widetilde{U}), \tag{2.27}$$

$$\min \theta(V,\lambda) = (V - WV)^T(V - WV) + \lambda\left(V - \tilde{V}\right)^T C\left(V - \tilde{V}\right). \tag{2.28}$$

Assim, terminamos a fase de modelagem do processo de otimização e estamos aptos a dar seguimento iniciando a etapa de resolução.

3. Resolução

Uma vez que o modelo tenha sido formulado, um método de otimização deve ser especificado para encontrar sua solução. Não existe forma ou algoritmo de otimização universal, isto é, um modo ideal para resolver todo e qualquer problema. Em vez disso, existem inúmeras abordagens onde cada uma é mais apropriada para tipos particulares de problemas de otimização [13].

A importância do método escolhido está associada à velocidade que o problema será resolvido e ao tipo de solução que - possivelmente - será encontrada. No nosso caso, esses assuntos relacionam-se intrinsecamente ao caráter de um problema de minimização irrestrito assim como a natureza de uma função quadrática.

Posto isso, essa seção se encarrega de determinar o método pelo qual o problema de minimização final obtido na seção anterior será resolvido. Inicialmente, levantamos questões acerca de um problema de minimização irrestrito genérico, vemos os aspectos de uma função quadrática definida em $\mathbb{R}^n$ e fazemos uso dessas informações para caracterizar a função objetivo do problema. Desta forma, iremos naturalmente motivar e propor uma maneira de resolução adequada para o problema de minimização formalizado no fim do Capítulo 2.

3.1 Minimização irrestrita

Em um problema de minimização irrestrito, nós estamos interessados em minimizar uma função objetivo que depende de variáveis reais sem restrições em seus valores. A formulação matemática é dada por

$$\text{minimizar } f(x),$$

em que x é um vetor real com n componentes e f é uma função genérica tal que $f\colon \mathbb{R}^n \to \mathbb{R}$.

Esse tipo de problema costuma ser iniciado pela busca dos pontos críticos da função, ou seja, os pontos que anulam o vetor gradiente da mesma, os quais poderão ser minimizadores, maximizadores ou pontos de "sela". Mesmo fixando a análise em minimizadores, várias questões são levantadas:

(i) Quais as condições para que um mínimo de fato exista?

(ii) O mínimo é único?

(iii) Existe algum mínimo relativo?

Essa função f genérica pode ter um único mínimo, vários mínimos locais ou globais, ou até mesmo não apresentar um mínimo [26]. A Figura 3.1 retrata esses diferentes cenários por meio de gráficos de funções com apenas uma variável; o raciocínio pode ser estendido para funções definidas em $\mathbb{R}^n$ sem complicações.

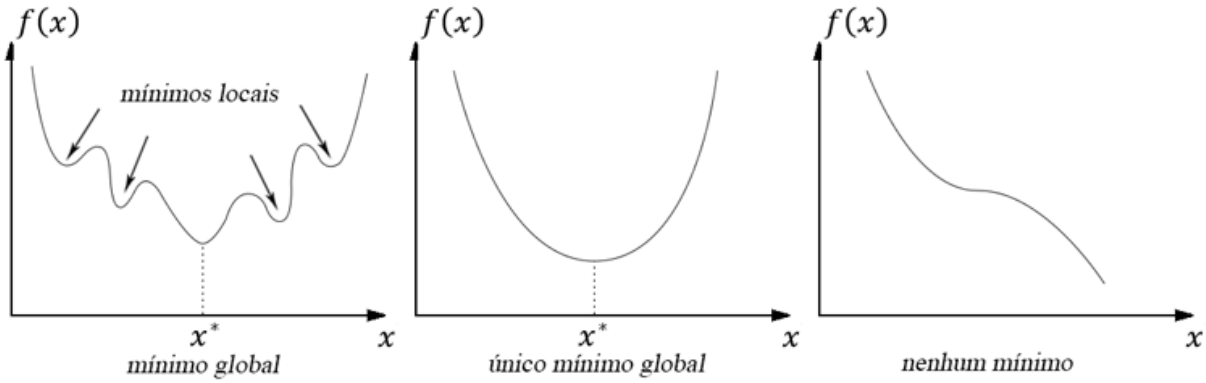


Figura 3.1: Exemplos de diferentes pontos de mínimo

Para que todas as perguntas acima possam ser respondidas analiticamente e de forma mais precisa, é necessário conhecer a natureza da função em questão. Na seção anterior, a formulação final do problema de minimização foi dada em termos de uma função objetivo quadrática. Assim sendo, a seção seguinte conduz uma breve discussão sobre essa classe de funções não lineares.

3.2 Função Quadrática em $\mathbb{R}^n$

Funções quadráticas representam a classe mais simples dentre todos os tipos de funções não lineares [27]. Isso se dá pelo fato de que essas funções apresentam expressões relativamente descomplicadas e, nelas, conceitos básicos de otimização apresentam interpretações claras e diretas.

Uma função quadrática com n variáveis pode ser dada por

$$q(x) = \frac{1}{2}x^T Ax + b^T x + c, \tag{3.1}$$

em que $c \in \mathbb{R}$, b é um vetor real de tamanho n e A é uma matriz real de tamanho $n \times n$. Essa matriz pode ser escolhida de maneira não única, mas mesmo assim, é muito comum querer que A seja simétrica. Quando isso acontece, o cálculo do gradiente e da hessiana são obtidos numa forma muito simples sem perda de generalidade. Segue o lema:

Lema 3.1: *Se $q(x) = \frac{1}{2}x^T Ax + b^T x + c$, com A simétrica, então seu gradiente é dado por uma função vetorial linear $\nabla q(x) = Ax + b$ e sua hessiana é dada por uma matriz constante $\nabla^2 q(x) = A$.*

Prova: *Derivando duas vezes a Equação (3.1) os resultados são obtidos de imediato.*

■

Para um problema de minimização, o Lema 3.1 representa um alto nível de simplificação. Note que igualando o vetor gradiente a zero, reduzimos o problema de encontrar pontos críticos da função (3.1) à resolução de um sistema linear

$$\nabla q(x) = 0 \leftrightarrow Ax = -b. \tag{3.2}$$

Assim sendo, dúvidas acerca da unicidade ou existência de pontos críticos numa função quadrática podem ser respondidas em termos das propriedades apresentadas por esse sistema correspondente. Lembremos portanto alguns conceitos de Álgebra Linear:

Definição 3.1: *Uma matriz $A \in \mathbb{R}^{n\times n}$ é dita não singular se A for inversível. Caso contrário, A é denominada singular* [28].

Lema 3.2: *O sistema linear (3.2) admite alguma solução se, e somente se, $b \in \mathcal{R}(A)$, onde $\mathcal{R}(A)$ é o espaço coluna de A.*

Lema 3.3: *O sistema linear (3.2) tem uma única solução se, e somente se, A é uma matriz não singular.*

Lema 3.4: *O sistema linear (3.2) tem nenhuma ou infinitas soluções se, e somente se, A é uma matriz singular.*

Prova: *A prova para os três lemas acima podem ser encontrados em* [28] *nas páginas 53, 54 e 116.*

■

Combinando esses lemas, podemos dizer se a equação dos pontos críticos (3.2) terá um, infinitos ou nenhum ponto crítico. Se A é não singular, então o sistema (3.2) tem uma única solução e esse será o único ponto crítico de (3.1). Contudo, ele pode ser tanto um minimizador, maximizador ou ponto de "sela". Agora, se A é singular e b está no seu espaço coluna, então o sistema (3.2) tem infinitas soluções e todas elas serão pontos críticos de (3.1). Veremos ainda que todos esses pontos são do mesmo tipo. Finalmente, se A é singular e b não pertence ao espaço coluna de A, então (3.2) não tem solução. Consequentemente, a quadrática (3.1) não apresenta nenhum ponto crítico [29].

Essas conclusões são interessantes mas não tão específicas. Ainda não somos capazes de dizer se um ponto crítico é um minimizador, um maximizador ou um ponto de "sela". Esse mérito costuma ser discutido quando classificamos a hessiana da função por meio das seguintes definições [30]:

Definição 3.1: *Uma matriz real A é chamada de definida positiva se* $x^T Ax > 0$ *para todo* $x \in \mathbb{R}^n$ *e* $x \neq 0$.

Definição 3.2: *Uma matriz real A é chamada de semidefinida positiva se* $x^T Ax \geq 0$ *para todo* $x \in \mathbb{R}^n$ *e* $x \neq 0$.

Definição 3.3: *Uma matriz real A é chamada de indefinida se* $x^T Ax > 0$ *para alguns* $x \in \mathbb{R}^n$ *e* $x^T Ax < 0$ *para outros*.

Esse tipo de análise costuma ser um recurso matemático muito utilizado para estudar tipos de pontos críticos em funções quadráticas quando se é possível obter a hessiana de forma simples. A partir de sua classificação quanto a ser definida positiva, semidefinida positiva ou indefinida, podemos entender o comportamento da função e determinar que tipos de pontos críticos ela apresenta. As proposições a seguir explicam essa questão:

Proposição 3.1: *Se* x^* *é um ponto crítico de (3.1) e A é definida positiva, então (3.1) tem um único ponto de mínimo dado por* x^*.

Prova: *Seja* x^* *um ponto crítico de (3.1) e* x *um ponto qualquer diferente de* x^*. *Então* $b = -Ax^*$, *logo*

$$q(x) = \frac{1}{2}x^T Ax + b^T x + c = \frac{1}{2}x^T Ax - x^{*T} Ax + c$$

$$= \frac{1}{2}(x - x^*)^T A(x - x^*) - \frac{1}{2}x^{*T} Ax^* + c.$$

Como A é semidefinida positiva, $(x - x^*)^T A(x - x^*) > 0$. *Portanto*

$$q(x) > -\frac{1}{2}x^{*T} Ax^* + c$$

$$= \frac{1}{2}x^{*T} Ax^* - x^{*T} Ax^* + c = \frac{1}{2}x^{*T} Ax^* - b^T Ax^* + c = q(x^*).$$

Logo $q(x) > q(x^*)$ *para todo* $x \neq x^*$, *isto é,* x^* *é minimizador global de (3.1) e único.*

■

Proposição 3.2: *Se* x^* *é um ponto crítico de (3.1) e A é semidefinida positiva, então (3.1) tem infinitos pontos de mínimo.*

Prova: *Seguindo o mesmo procedimento da última demonstração e considerando A como semidefinida positiva, chegamos a conclusão de que* $q(x) \geq q(x^*)$ *para todo* $x \neq x^*$. *Como toda matriz semidefinida positiva é também uma matriz singular, então temos que (3.1) apresenta infinitos pontos de mínimo; todos globais.*

■

Proposição 3.3: *Se* x^* *é um ponto crítico de (3.1) e A é indefinida, então* x^* *é ponto de "sela" e (3.1) não tem extremos locais.*

Prova: *Na mesma linha de raciocínio, se A é indefinida teríamos que* $q(x) > q(x^*)$ *para alguns x e* $q(x) < q(x^*)$ *para outros. Dessa forma, como* x^* *é ponto crítico e essas duas desigualdades se verificam, segue que* x^* *é um ponto de "sela" único e (3.1) não apresenta extremos locais.*

■

Para ilustrar esses tipos de funções quadráticas e suas relações com os pontos críticos - no nosso caso, pontos de mínimo -, observe a Figura 3.2. Ela conta com três gráficos simples de quadráticas definidas em $\mathbb{R}^3$. A primeira superfície é um paraboloide elíptico cujo formato lembra uma "cuia". Esse tipo de superfície é formado por funções quadráticas cuja hessiana é definida positiva. A segunda superfície mostra um cilindro parabólico cujo formato lembra uma

“calha”. Essa forma é obtida por uma quadrática com hessiana semidefinida positiva. Por fim, é mostrado um paraboloide hiperbólico cujo formato lembra o de uma “sela”. Tal superfície é desenhada por quadráticas onde a hessiana é indefinida.

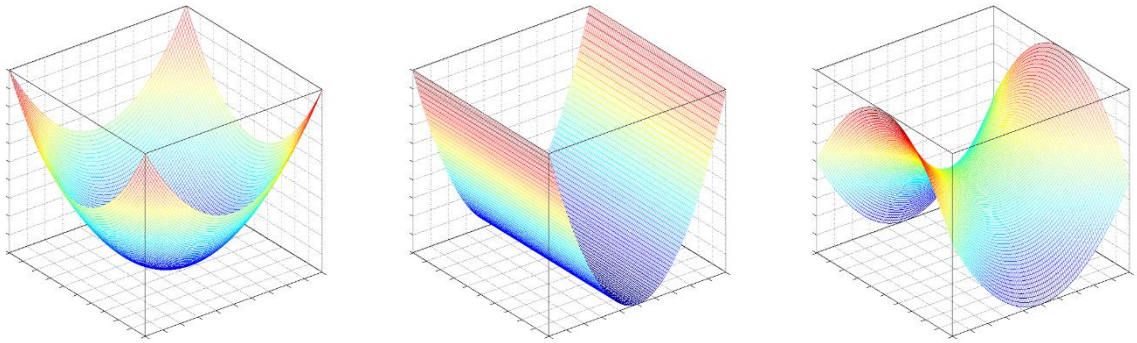


Figura 3.2: Funções $f(x,y) = x^2 + y^2$, $g(x,y) = x^2$ e $h(x,y) = x^2 - y^2$ respectivamente.

A nível de completude, vale dizer que resultados análogos existem para hessianas definidas negativas e semidefinidas negativas. Nesses dois casos, a função em questão apresentaria um único maximizador global, infinitos maximizadores globais ou nenhum ponto de máximo.

Enfim, uma vez discutido todos esses aspectos que explicam como uma função quadrática é tipificada e como seus pontos críticos podem ser dados, estamos preparados para caracterizar a nossa função objetivo.

3.3 Caracterização da Função

A função que estamos interessados em estudar foi dada pela Equação (2.23) no capítulo anterior. Copiamos novamente essa equação abaixo para facilitar o desenvolvimento:

$$\theta(U,\lambda) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \overline{w}_{rs} U_s \right)^2 + \lambda \sum_{k \in K} \left(U_k - \widetilde{U}_k \right)^2$$

Seguindo o nosso raciocínio, a meta agora é calcular a hessiana de θ afim de determinar que tipo de função quadrática estamos tratando - se apresenta um formato de “cuia”, “calha” ou “sela”. Mas antes de procedermos com essa formalização, vamos fazer uso da nossa intuição e tentar enxergar de antemão qual é o seu comportamento.

A formulação de θ acima foi dada em função de dois somatórios. O primeiro somatório é responsável por garantir as necessidades propostas na descrição do problema enquanto que, o

segundo, incorpora os valores dos pixels que foram rabiscados por meio de uma função de penalidade.

Imaginemos agora que nenhum pixel tenha sido rabiscado. Obviamente, esse segundo somatório não existiria e o problema de minimização seria simplificado por

$$\min \theta(U) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \overline{w}_{rs} U_s \right)^2 . \tag{3.3}$$

A Equação (3.3) é um somatório de diferenças de quadrados. Obviamente, o valor mínimo que θ pode assumir nesse caso é 0 e, isso acontece, quando todas essas parcelas do somatório também valem 0. Se cada parcela é responsável por calcular a diferença entre a componente U de um pixel r e a média ponderada da componente U dos pixels vizinhos de r, para que seu valor seja 0 teríamos que

$$U_r = \sum_{s \in N(r)} \overline{w}_{rs} U_s \ \ \forall \, r \in P \, . \tag{3.4}$$

Como $\sum_{s \in N(r)} \overline{w}_{rs} = 1$, podemos dizer de imediato que essa igualdade se verifica quando $U_r = U_s$ para todo pixel r.

Isso significa que quando todas as componentes U assumem o mesmo valor, a função θ assume o seu valor mínimo. Todos os escalares múltiplos do vetor constante são possíveis soluções para (3.3) e, portanto, temos infinitos pontos de mínimo. Diante disso, podemos afirmar antecipadamente que, para imagens não rabiscadas, a hessiana da função objetivo é semidefinida positiva.

Será que o mesmo acontece para imagens rabiscadas? Vamos supor agora que o usuário tenha dado um rabisco de apenas um pixel e o valor da componente U desse pixel rabiscado seja U_k. Nesse caso, o problema de minimização seria

$$\min \theta(U, \lambda) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \overline{w}_{rs} U_s \right)^2 + \lambda \left(U_k - \widetilde{U}_k \right)^2 . \tag{3.5}$$

Novamente, θ tem seu custo mínimo dado por 0. Como ambas as parcelas são quadrados e λ assume um valor estritamente positivo, basta que a Equação (3.4) seja satisfeita simultaneamente com a equação

$$U_k = \widetilde{U}_k. \tag{3.6}$$

Hora, se todo vetor constante anula (3.4) e queremos satisfazer (3.6) ao mesmo tempo, então uma solução para a Equação (3.5) é um vetor onde todas as componentes assumem o valor $\widetilde{U}_k$. Assim sendo, é razoável pensar que para uma imagem rabiscada, só existe uma forma de coloração - uma única solução - e, portanto, a hessiana de θ nesse caso seria definida positiva.

Toda essa discussão acerca do possível comportamento de θ permite apenas uma intuição dos resultados esperados. Evidentemente, esses resultados não são formais. Repare que (3.4) nem sempre terá seu conjunto solução gerado apenas por um vetor constante (isso pode acontecer, por exemplo, se para algum par de pixels r e s, $\overline{w}_{rs} = 0$). Caso isso ocorra, (3.5) possivelmente ainda teria infinitas soluções.

Sendo assim, partimos agora para a caracterização formal da função objetivo (2.23). Apesar dessa classificação poder ser feita de uma só vez, vamos seguir a linha de raciocínio adotada nesta seção. Primeiramente iremos classificar a hessiana da função para o caso em que uma imagem não recebeu rabiscos - sem a parcela de θ que incorpora a penalidade - e em seguida para uma imagem com rabiscos. Esperamos assim que, o primeiro resultado nos dê uma matriz semidefinida positiva enquanto que o segundo, uma matriz definida positiva.

3.3.1 Imagens não rabiscadas

Do Lema 3.1, sabemos que a hessiana de uma função quadrática nada mais é do que uma matriz constante. Para obtê-la, basta derivar a função em questão duas vezes. Com esse propósito, utilizaremos a equação em (3.3) que representa de forma simplificada a função objetivo para uma imagem sem rabiscos em notação matricial. Assim, seguem abaixo os cálculos do processo de derivação.

A Equação (3.3) pode ser escrita como

$$\theta(U) = (U - WU)^T(U - WU). \tag{3.7}$$

Aplicando a transposição e botando os vetores U^T e U em evidência, temos que

$$\theta(U) = (U^T - U^T W^T)(U - WU)$$

$$= U^T(I - W)^T(I - W)U,$$

em que I é a matriz identidade de tamanho $n \times n$. O vetor gradiente é obtido como

$$\nabla\theta(U) = 2(I - W)^T(I - W)U \tag{3.8}$$

e a segunda derivada é dada por

$$\nabla^2\theta(U) = 2(I - W)^T(I - W). \tag{3.9}$$

Nossa meta é classificar $\nabla^2\theta$ quanto a ser definida positiva, semidefinida positiva ou indefinida. Não é difícil provar que (3.9) é semidefinida positiva e que a minimizar (3.7) apresenta infinitas soluções. Assim, segue a proposição:

Proposição 3.4: *Para uma imagem sem rabiscos, a função objetivo (2.23) tem sua hessiana dada por uma matriz semidefinida positiva e o problema de minimizá-la apresenta infinitas soluções.*

Prova: *Seguindo as Definições 3.1, 3.2 e 3.3, multiplicaremos o lado direito da hessiana por um vetor arbitrário x e o lado esquerdo por x^T, onde $x \in \mathbb{R}^n$ e $x \neq 0$. Dessa forma, estamos interessados em analisar o comportamento do sinal de $x^T\nabla^2\theta(U)x$.*

Partindo da Equação (3.9), temos

$$\begin{aligned} x^T\nabla^2\theta(U)x &= x^T 2(I - W)^T(I - W)x \\ &= 2x^T(I - W)^T(I - W)x \\ &= 2[(I - W)x]^T(I - W)x. \end{aligned}$$

Vamos definir agora um vetor $s \in \mathbb{R}^n$ tal que

$$s = (I - W)x.$$

Combinando as duas equações acima, obtemos

$$x^T\nabla^2\theta(U)x = 2s^Ts$$

Se s_i representa uma componente i do vetor s, então

$$x^T\nabla^2\theta(U)x = 2s^Ts = 2\sum_{i=1}^{n} {s_i}^2 \geq 0. \tag{3.10}$$

Embora x seja não nulo, não podemos afirmar o mesmo para o vetor s. Dessa forma, demonstramos que a hessiana da função objetivo para uma imagem não rabiscada é de fato

semidefinida positiva. Resta saber se essa função apresenta nenhum ou infinitos pontos de mínimo. Para isso, seja o sistema linear obtido igualando (3.8) a zero

$$\nabla\theta(U) = 2(I-W)^T(I-W)U = 0$$

$$(I-W)^T(I-W)U = 0.$$

Por ser um sistema homogêneo, sabemos que este admite ao menos uma solução. Ou ainda, como o vetor nulo 0 *pertence a qualquer subespaço vetorial, em particular, o espaço coluna de* $(I-W)^T(I-W)$*, usando o Lema 3.2, temos que o sistema* $(I-W)^T(I-W)U = 0$ *admite alguma solução. Logo, (3.9) é semidefinida positiva e o problema de minimizar (3.7) admite infinitas soluções.*

■

3.3.2 Imagens rabiscadas

Estudar o comportamento dos pontos críticos da função objetivo de uma imagem que recebeu rabiscos não é tão trivial quanto a situação acima. No início da Seção 3.3, abrimos uma discussão e chegamos à conclusão de que é bem provável que nesse caso, a função objetivo tenha sua hessiana dada por uma matriz definida positiva. Para obtê-la, derivaremos a função (2.23) duas vezes a partir de sua notação matricial. Assim, seguem abaixo os cálculos do processo de derivação.

A equação (2.23) que representa a função objetivo para uma imagem rabiscada em notação matricial é reescrita a seguir:

$$\theta(U,\lambda) = (U-WU)^T(U-WU) + \lambda\left(U-\widetilde{U}\right)^T C\left(U-\widetilde{U}\right). \tag{3.11}$$

Organizando a primeira parcela da soma da mesma forma realizada na Subseção 3.3.2, temos

$$\theta(U,\lambda) = U^T(I-W)^T(I-W)U + \lambda\left(U-\widetilde{U}\right)^T C\left(U-\widetilde{U}\right).$$

O cálculo do gradiente pode ser obtido aplicando-se a regra da cadeia

$$\begin{aligned}\nabla\theta(U,\lambda) &= 2(I-W)^T(I-W)U + 2\lambda C\left(U-\widetilde{U}\right)\\ &= 2(I-W)^T(I-W)U + 2\lambda CU - 2\lambda C\widetilde{U},\end{aligned} \tag{3.12}$$

e, derivando mais uma vez a equação acima, obtemos a hessiana de θ

$$\begin{aligned}\nabla^2\theta(U) &= 2(I-W)^T(I-W)+2\lambda C \\ &= 2[(I-W)^T(I-W)+\lambda C].\end{aligned} \tag{3.13}$$

Para ter certeza da sua classificação, vamos seguir os mesmos passos adotados na seção anterior na demonstração da próxima proposição. Ela nos dará, por exemplo, condições necessárias para que uma imagem rabiscada tenha uma única forma de coloração.

Proposição 3.5: *Para uma imagem com rabiscos, a função objetivo (2.23) tem sua hessiana dada por uma matriz definida positiva e, portanto, apresenta uma única solução se* $w_{rs} > 0$ *para todo* $r \in P$.

Prova: *Para podermos classificar (3.13), iremos multiplicar o lado direito da hessiana por um vetor arbitrário* x *e o lado esquerdo por* x^T*, em que* $x \in \mathbb{R}^n e\ x \neq 0$*. Com isso, temos*

$$\begin{aligned}x^T\nabla^2\theta(U)x &= x^T 2[(I-W)^T(I-W)+\lambda C]x \\ &= 2x^T[(I-W)^T(I-W)+\lambda C]x \\ &= 2[x^T(I-W)^T(I-W)x + x^T\lambda C x].\end{aligned} \tag{3.14}$$

Analisando os dois termos da soma obtida, temos que $x^T(I-W)^T(I-W)x \geq 0$ *como verificado em (3.10) e* $x^T\lambda C x \geq 0$ *já que* λ *é um real estritamente positivo enquanto que* C *é uma matriz diagonal onde alguns elementos valem* 1 *e outros* 0 *(definição dada pela Equação (2.25)).*

Essas respostas seriam o suficiente para dizer que $x^T\nabla^2\theta(U)x \geq 0$*. No entanto, vamos atentar para a seguinte interrogação: quando que* $x^T\nabla^2\theta(U)x$ *assume o valor nulo? Evidentemente, isso acontece quando* $x^T(I-W)^T(I-W)x = 0$ *e* $x^T\lambda C x = 0$ *de forma simultânea. A seguir, avaliaremos para que valores de* x *essas equações são satisfeitas.*

Repare que resolver

$$x^T(I-W)^T(I-W)x = [(I-W)x]^T(I-W)x = 0 \tag{3.15}$$

é o mesmo que resolver o sistema homogêneo dado por

$$(I-W)x = 0.$$

Sabemos também que a soma dos elementos em cada linha da matriz $(I-W)$ *vale* 0*. Isso acontece porque definimos* W *de tal forma onde todos os elementos em uma linha somam* 1

enquanto que I é a matriz identidade (definição dada pela Equação (2.13)). Assim, se denotarmos

$$B = (I - W),$$

onde $B = \{b_{ij}\}_{i,j}^{n}$, *então*

$$\sum_{j=1}^{n} b_{ij} = 0, \forall i = 1, 2, \dots, n.$$

Se nós chamarmos

$$y = Bx,$$

então

$$y_i = \sum_{j=1}^{n} x_i b_{ij} = 0, \forall i = 1, 2, \dots, n.$$

Dessa forma, caso todas as componentes de x forem iguais, teríamos

$$x_i = t, \forall i = 1, 2, \dots, n \; e \; t \in \mathbb{R}^n.$$

Combinando as três equações acima, temos que

$$y_i = \sum_{j=1}^{n} x_i b_{ij} = \sum_{j=1}^{n} t b_{ij} = t \sum_{j=1}^{n} b_{ij} = t0 = 0, \forall i = 1, 2, \dots, n.$$

Isso mostra que qualquer vetor constante real está contido no espaço nulo de $(I - W)$. No entanto, não temos certeza se o sistema admite outras soluções não triviais. Para sanar tal dúvida, segue o lema:

Lema 3.5: *A equação (3.15) terá seu espaço solução gerado apenas por um vetor constante caso $w_{rs} > 0 \; \forall \; r \in P$.*

Prova: *Suponha por contradição que exista uma solução $\bar{x}$ para (3.15) que não seja um vetor constante. Então, o máximo dessa solução acontece em pelo menos uma componente $\bar{x}_r$ com no mínimo um valor estritamente menor em alguma componente $\bar{x}_s$.*

Em outras palavras, para satisfazer (3.15), temos que

$$x_r = \sum_{s \in N(r)} \bar{w}_{rs} x_s \,, \forall r = 1,2, \dots, n.$$

onde $\bar{w}_{rs} = \frac{w_{rs}}{\sum_{s\prime \in N(r)} w_{rs\prime}}$ *e* $\sum_{s \in N(r)} \bar{w}_{rs} = 1$, *ou seja, as componentes* $\bar{w}_{rs}$ *são parâmetros de ponderação.*

Se $w_{rs} \neq 0$ *e sabendo que* $w_{rs} \geq 0$ *da definição de função de peso, então* $0 < \bar{w}_{rs} < 1$. *Para algum par de valores* $\bar{x}_r$ *e* $\bar{x}_s$, *onde* $\bar{x}_r$ *assume o valor máximo da solução* $\bar{x}$ *e* $\bar{x}_s$ *representa um valor pertencente ao conjunto de vizinhos de* r *que não é máximo, então necessariamente para que a igualdade da soma ponderada seja satisfeita,* $\bar{x}_r = \bar{x}_s$. *Portanto, se existe um valor máximo* $\bar{x}_r$, *todos os valores* x_s *tal que* $s \in N(r)$, *devem ser iguais, ou, caso contrário,* $\bar{x}_r > \sum_{s \in N(r)} \bar{w}_{rs} x_s$. *O raciocínio se estende para todos os pixels* r.

Sendo assim, chegamos a uma contradição, isto é, $x_r = x_s \; \forall s \in N(r)$, *e* $\forall r \in P$.

■

Partindo para o segundo termo da soma apresentado na Equação (3.14), queremos investigar para que valores

$$x^T \lambda C x = 0. \tag{3.16}$$

Pela definição de C *descrita em (2.25), temos que*

$$x^T \lambda C x = \lambda x^T C x = \lambda \sum_{i \in K} {x_i}^2. \tag{3.17}$$

Como $\lambda > 0$, *a igualdade* $x^T \lambda C x = 0$ *se verifica apenas quando* $x_i = 0 \; \forall i \in K$ *ou quando* $x = 0$. *Essa segunda opção é descartada já que da Definição 3.2,* x *não pode ser o vetor nulo.*

Agora que sabemos quando que $x^T (I - W)^T (I - W) x = 0$ *e* $x^T \lambda C x = 0$, *vamos averiguar em que momento essas equações são satisfeitas simultaneamente. Hora, supondo que* $w_{rs} \neq 0 \; \forall r \in P$, *pelo Lema 3.5, a igualdade* $x^T (I - W)^T (I - W) x = 0$ *é satisfeita apenas quando* x *for um vetor constante. Se isso acontece, segue de (3.17) que* $x^T \lambda C x$ *assume necessariamente um valor positivo e, assim, (3.14) é maior que* 0 *sempre.*

Com isso, mostramos que, se $w_{rs} > 0 \; \forall r \in P$*, então a função objetivo (3.11) terá uma hessiana definida positiva e o problema de minimizar essa função terá um único ponto de mínimo.*

■

Assim sendo, finalizamos esta subseção com resultados importantes. Analisamos os principais aspectos da função objetivo do problema de minimização e conseguimos caracterizá-la estudando seu comportamento. Usando todo o conhecimento desenvolvido, seremos capazes de indicar um método mais eficiente de solução do que foi usado pelos autores do problema de colorização em [11]. O próximo e último passo é, portanto, apontar de que forma essa função será minimizada.

3.4 Método utilizado

Da Seção 3.2, vimos que a minimização de uma função quadrática pode ser tratada como a resolução de um sistema linear onde a matriz dos coeficientes é dada pela hessiana da função (Equação 3.2). Em seguida, na Seção 3.3, fomos capazes de determinar uma condição simples que torna essa matriz sempre definida positiva e, com isso, a solução de tal sistema linear é única (Proposição 3.5).

Seguindo essa linha de raciocínio, a Equação (3.11) - função objetivo (2.23) expressa em notação matricial -, pode ser minimizada igualando seu vetor gradiente (3.12) a zero:

$$\nabla\theta(U,\lambda) = \; 2(I-W)^T(I-W)U + 2\lambda C\left(U - \; \widetilde{U}\right) = 0.$$

Botando 2 em evidência e distribuindo a matriz λC, temos

$$2[(I-W)^T(I-W)U + \lambda CU - \lambda C\widetilde{U}] = 0$$

Separando a variável do problema U e arrumando as parcelas, podemos determinar o seguinte sistema linear:

$$[(I-W)^T(I-W) + \lambda C]U = \lambda C\widetilde{U}. \tag{3.18}$$

Com isso, minimizar (3.11) é o mesmo que resolver o sistema linear (3.18).

$$\nabla\theta(U,\lambda) = 0 \leftrightarrow [(I-W)^T(I-W) + \lambda C]U = \lambda C\widetilde{U}.$$

Uma forma direta pela qual sistemas lineares costumam ser resolvidos é chamada de *fatoração*. Esse tipo de algoritmo é desenvolvido usando matrizes de permutação e tem como

objetivo decompor a matriz dos coeficientes do sistema num produto de matrizes de forma a tornar a resolução do sistema muito mais simples.

Dentre os vários métodos de fatoração conhecidos, escolhemos para este trabalho a *fatoração de Cholesky*. A principal motivação desta escolha foi o fato de que essa fatoração foi criado especialmente para casos onde a matriz dos coeficientes é simétrica e definida positiva - o que casa perfeitamente com a nossa abordagem. Note que $[(I-W)^T(I-W)+\lambda C]$ é simétrica e, de acordo com a Proposição 3.5, podemos torná-la definida positiva escolhendo uma função de peso adequada.

Por meio desse método, é possível obter uma solução fiel para o problema consumindo um número menor de operações frente outros tipos de fatoração. Além disso, outro ponto que foi levado em consideração nessa decisão foi a disponibilidade de excelentes rotinas (particularmente, CHOLMOD [31]) que implementam de forma eficiente seu algoritmo.

No site do projeto que resultou o artigo [11], os autores disponibilizam uma implementação do algoritmo de colorização via otimização em MATLAB utilizando a *fatoração LU*. Sendo assim, as próximas duas subseções descrevem brevemente o funcionamento tanto desta fatoração assim como a de Cholesky. Nelas, faremos um comparativo simples mas importante que diferencia significativamente tais algoritmos.

3.4.1 Fatoração LU

A fatoração LU de uma matriz $A \in \mathbb{R}^{n\times n}$, originária da famosa eliminação Gauss, é definida como

$$PA = LU,$$

onde P é uma matrix de permutação de tamanho $n \times n$ (matriz identidade rearranjada responsável por reordenar as linhas de A de modo a evitar problemas procedurais), L é uma matriz triangular inferior onde todos os elementos da diagonal são iguais a 1 e U é uma matriz triangular superior. Para ilustrar, uma matriz $n \times n$ seria decomposta da seguinte forma:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix}.$$

Esta fatoração é usada para resolver sistemas lineares da forma $Ax = b$ eficientemente por meio de um processo de poucos passos. Primeiramente é necessário obter

$$PAx = Pb \equiv \tilde{b}$$

permutando os elementos de b. Substituindo LU por PA nós ficamos com

$$LUx = \tilde{b}.$$

Por fim, basta apenas resolver dois problemas de substituição para trás dados por

$$Lz = \tilde{b}$$

e

$$Ux = z,$$

obtendo assim o vetor solução x.

A fatoração LU apresentada acima é comumente realizada usando eliminação de Gauss com pivoteamento parcial de linhas. Quando isso acontece, seu algoritmo requer aproximadamente $2n^3/3$ operações de ponto flutuante quando A é uma matriz densa.

Existem diversos programas padrões que implementam esse algoritmo (notavelmente, UMFPACK [39]) disponíveis na WEB. A seguir mostramos um algoritmo base para tal método:

Algoritmo 1 (Fatoração LU com Pivoteamento Parcial de Linhas – Eliminação de Gauss)

Dado $A \in \mathbb{R}^{n\times n}$;

Seja $P \leftarrow I, L \leftarrow 0$;

1. **Para** $i = 1, 2, \dots, n$
   // Etapa de permutação
2. Encontrar índice $j \in \{i, i+1, \dots, n\}$ tal que $|A_{ji}| = \max_{k=i,i+1,\dots,n} |A_{ki}|$;
3. **Se** $A_{ij} = 0$
4. Pare; // matriz A é singular
5. **Se** $i \neq j$
6. Trocar linhas i e j da matriz A e L;
   // Etapa de eliminação
7. $L_{ii} \leftarrow 1$;
8. **Para** $k = i+1, i+2, \dots, n$
9. $L_{ki} \leftarrow A_{ki}/A_{ii}$;
10. **Para** $l = i+1, i+2, \dots, n$
11. $A_{kl} \leftarrow A_{kl} - L_{ki}A_{il}$;
12. **Fim** (**Para**)
13. **Fim** (**Para**)
14. **Fim** (**Para**)
15. $U \leftarrow$ parte triangular superior de A.

Variantes deste algoritmo base permitem rearranjar as colunas assim como as linhas durante a fatoração, no entanto isso não adiciona grande ganho à estabilidade prática do processo. Mesmo assim, vale mencionar que o pivoteamento de coluna pode aumentar o desempenho da eliminação Gaussiana quando A é uma matriz esparsa, garantindo que os fatores L e U também sejam relativamente esparsos [13].

3.4.2 Fatoração de Cholesky

Quando $A \in \mathbb{R}^{n\times n}$ é simétrica e definida positiva, é possível computar uma fatoração similar à fatoração LU por aproximadamente metade do custo - em torno de $n^3/3$ operações. Essa fatoração, conhecida como fatoração de Cholesky, produz uma matriz L tal que

$$A = LL^T,$$

em que L é uma matriz triangular inferior onde todos os elementos da diagonal são reais e estritamente positivos. Para ilustrar, uma matriz qualquer $n \times n$ seria decomposta da seguinte forma:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{nn} \end{pmatrix}.$$

Para obter $A = LL^T$, nós simplesmente equacionamos os coeficientes nos dois lados da equação. Resolvendo para as variáveis (elementos l_{ij} não nulos), para $i = 1, \dots, n$ e $j = i - 1, \dots, n$, obtemos:

$$l_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} {l_{ik}}^2 \right)},$$

$$l_{ji} = \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik} \right) / l_{ii}.$$

Como A é simétrica e definida positiva, a expressão dentro da raiz quadrada é sempre positiva e todos os valores l_{ij} são reais.

Assim como a fatoração LU, a fatoração de Cholesky pode ser utilizada para computar a solução de um sistema $Ax = b$. Apesar das etapas de decomposição serem diferentes, o

processo de resolução do sistema é bem semelhante. Basta resolvermos apenas dois problemas de substituição para trás obtidos por

$$Ly = b$$

e

$$L^T x = y,$$

obtendo o vetor solução x.

O algoritmo base dessa decomposição é descrito a seguir:

Algoritmo 2 (Fatoração de Cholesky)

Dado $A \in \mathbb{R}^{n\times n}$ simétrica e definida positiva;

1. **Para** $i = 1, 2, \dots, n$
2. $\quad L_{ii} \leftarrow \sqrt{A_{ii}}$;
3. $\quad$ **Para** $j = i+1, i+2, \dots, n$
4. $\quad\quad L_{ji} \leftarrow A_{ji}/L_{ii}$;
5. $\quad\quad$ **Para** $k = i+1, i+2, \dots, j$
6. $\quad\quad\quad A_{jk} \leftarrow A_{jk} - L_{ji}L_{ki}$;
7. $\quad\quad$ **Fim** (**Para**)
8. $\quad$ **Fim** (**Para**)
9. **Fim** (**Para**)

Note que esse procedimento faz referência apenas aos elementos presentes no triângulo inferior de A; de fato, só é necessário armazenar estes valores já que, por simetria, eles são duplicados nas posições superiores da matriz.

Diferentemente do caso na eliminação Gaussiana, o algoritmo de Cholesky pode produzir uma fatoração válida para uma matriz simétrica definida positiva sem trocar nenhuma linha ou coluna, isto é, sem precisar da matriz P. No entanto, alguns tipos de reordenação podem ser usados para aperfeiçoar o nível de esparsidade do fator L. Neste caso, o algoritmo produziria uma permutação da forma

$$P^T AP = LL^T$$

para alguma matriz de permutação P.

3.4.3 O caso esparso

A análise teórica feita nas duas subseções anteriores é válida independentemente da estrutura da matriz A. No entanto, nos Algoritmos 1 e 2, usamos, implicitamente, a suposição de que

todas as entradas dos fatores são armazenadas. Com isso, esses algoritmos usariam mais de n^2 posições de memória.

Por exemplo, na fatoração de Cholesky, quando A é esparsa algumas entradas inicialmente zero podem se tornar valores não nulos de L, os quais são denominados de *preenchimentos*. Para reduzir tempo e necessidades de armazenamento, apenas as posições não nulas de L são armazenadas e sofrem operações durante o processo de fatoração. Portanto, uma etapa de ordenação é tipicamente aplicada primeiro. Como mencionado em 3.4.2, uma permutação conveniente de linhas e colunas em A pode trazer tal benefício. Assim, é possível reduzir os eventuais elementos de preenchimento que possam vir a surgir no fator L [33].

Dessa forma, a fatoração de Cholesky de matrizes esparsas procede da mesma maneira que o Algoritmo 2, mas toma o cuidado de armazenar apenas os elementos não nulos de A e L além de evitar fazer operações com zeros. Com isso, não apenas a memória, mas também o tempo computacional pode diminuir muito e a economia é bastante significativa quando n é grande [29].

É importante mencionar essa situação uma vez que a matriz dos coeficientes de nosso problema é esparsa e, para imagens de tamanhos frequentes, a ordem de grandeza de n costuma ser bem alta. Para se ter uma ideia, a Figura 3.3 ilustra o padrão de esparsidade de uma hessiana gerada para uma pequena imagem de dimensões 64×64 pixels, isto é, um total de 4096 pixels, para o raio de vizinhança igual a 1 e 2, respectivamente.

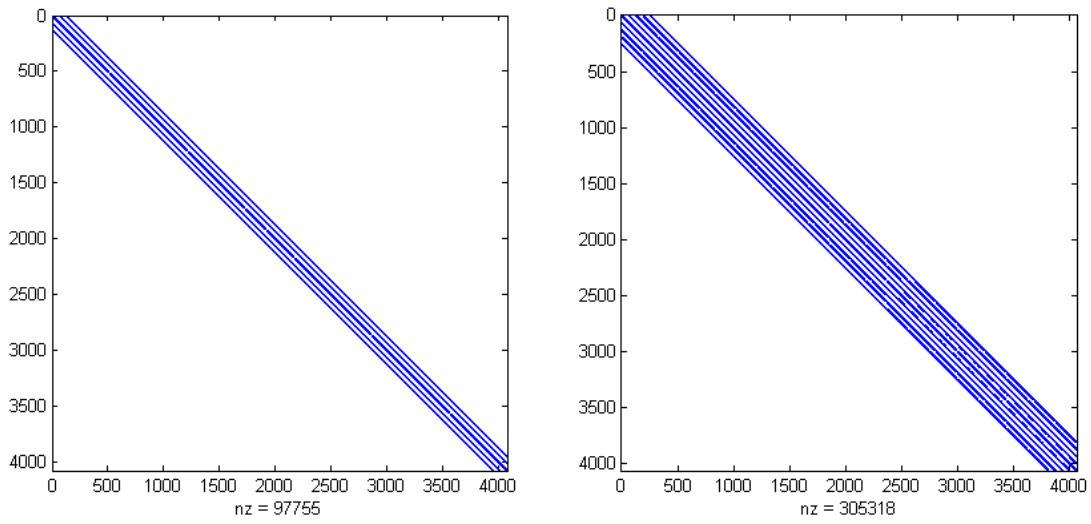


Figura 3.3: Padrão de esparsidade da matriz $[(I - W)^T(I - W) + \lambda C]$ para $R = 1$ e $R = 2$.

Nesses casos, a dimensão da matriz é 4096×4096 com um total de 16777216 elementos. Desse total, temos 97755 elementos não nulos para $R = 1$ e 305318 para $R = 2$; o que equivale a aproximadamente $0{,}5\%$ e 2% da matriz, respectivamente. Diante disso, a escolha de uma rotina especializada é mais adequada para tal resolução e de suma importância a nível de desempenho.

Diversos programas e bibliotecas foram desenvolvidas com esse propósito específico. Nomes como *CHOLDMOD*, *CSparse*, *LDL* e *UMFPACK* podem ser mencionados como bons exemplos; todos desenvolvidos em C/C++. Neste trabalho, todas essas bibliotecas - além de outras menos relevantes - foram testadas. A que se portou melhor e apresentou melhores resultados foi *CHOLMOD* e, por conta disso, se encontra na implementação final. Seus resultados são apresentados no Capítulo 4.

3.4.4 Método de resolução dos autores

No arquivo de apresentação encontrado em [12], Anat Levin, Dani Lischinski e Yair Weiss afirmam que a função objetivo do problema pode ser dada por

$$J'(U) = \sum_{r \in P} \left[\sum_{s \in N(r)} \overline{w}_{rs} (U_r - U_s)^2 \right], \tag{3.19}$$

ao invés da apresentada no artigo [11], ou seja, a função que viemos lidando desde o início deste trabalho

$$J(U) = \sum_{r \in P} \left(U_r - \sum_{s \in N(r)} \overline{w}_{rs} U_s \right)^2. \tag{3.20}$$

Em notação matricial, as Funções (3.19) e (3.20) são dadas por

$$J'(U) = U^T (I - W) U, \tag{3.21}$$

$$J(U) = U^T (I - W)^T (I - W) U, \tag{3.22}$$

respectivamente.

Segundo eles, (3.19) é exatamente a mesma função de custo (3.20), o que não é verídico. Se reescrevermos (3.20), é possível obter uma função muito semelhante

$$J(U) = \sum_{r \in P} \left[\sum_{s \in N(r)} \overline{w}_{rs} (U_r - U_s) \right]^2,$$

em que a única diferença é o fato dos pesos $\overline{w}_{rs}$ estarem sendo elevados ao quadrado.

O que de fato acontece é que minimizar (3.19) e (3.20) se resume ao mesmo problema. Por meio da Proposição 3.6, explicaremos essa situação.

Proposição 3.6: *$\breve{U}$ é minimizador de (3.19) se e somente se $\breve{U}$ é minimizador de (3.20).*

Prova: *Sabendo que os pontos críticos são dados igualando o vetor gradiente da função a zero, temos que para (3.19)*

$$\nabla J'(U) = 0 \leftrightarrow (I - W)U = 0,$$

e para (3.20)

$$\nabla J(U) = 0 \leftrightarrow (I - W)^T (I - W)U = 0.$$

Se $\breve{U}$ é ponto crítico de (3.19), então

$$(I - W)\breve{U} = 0.$$

Multiplicando o lado esquerdo da equação acima por $(I - W)^T$, temos que

$$(I - W)^T (I - W)\breve{U} = 0.$$

Portanto, $\breve{U}$ é também ponto crítico de (3.20).

Provando a volta, sabemos que o mínimo que (3.20) pode assumir é 0 *uma vez que a equação se trata de uma soma de quadrados. Sendo assim, $\breve{U}$ é ponto crítico de (3.20) tal que*

$$\breve{U}^T (I - W)^T (I - W)\breve{U} = 0$$

$$\left\| (I - W)\breve{U} \right\| = 0$$

$$(I - W)\breve{U} = 0.$$

Portanto, segue da última igualdade que $\breve{U}$ é ponto crítico de (3.19).

Como $(I - W)$ *e* $(I - W)^T(I - W)$ *são semidefinidas positivas (segue da condição exposta pelo Lema 3.5 e da Proposição 3.4, respectivamente) e sabendo que os sistemas correspondentes* $(I - W)\breve{U} = 0$ *e* $(I - W)^T(I - W)\breve{U} = 0$ *admitem pelo menos a solução trivial, então as funções (3.19) e (3.20) apresentam infinitos pontos de mínimo e, em particular,* $\breve{U}$ *é minimizador de ambas.*

■

Esse resultado é interessante uma vez que apresenta uma equação mais simples de ser tratada: observe que de fato (3.21) é uma função mais simplificada que (3.22). No entanto, em momento algum iremos minimizar (3.19) ou (3.20) sem antes ter dado um rabisco, isto é, sem antes ter adicionado a restrição de igualdade. Quando incorporamos a restrição, seja em (3.19) ou (3.20), geramos um sistema linear dado por uma matriz definida positiva em ambos os casos mas que não são equivalentes; o conjunto solução não é mais o mesmo.

Sendo assim, escrevemos a seguir o sistema linear que os autores propõe resolver em [11] seguido do nosso:

$$[(I - W) + \lambda C]U = \lambda C\tilde{U}, \tag{3.23}$$

$$[(I - W)^T(I - W) + \lambda C]U = \lambda C\tilde{U}. \tag{3.24}$$

O primeiro sistema de fato parece mais atraente. A ausência da multiplicação de matrizes sugere menos tempo de processamento. Contudo, apesar da matriz dos coeficientes de (3.23) ser definida positiva, ela não é simétrica e, por conta disso, a fatoração que os autores utilizam para resolvê-lo é LU. Já no nosso caso, aproveitaremos o fato da nossa matriz dos coeficientes (3.24) ser simétrica definida positiva e fazer uso da fatoração de Cholesky.

O próximo capítulo se encarrega de fazer todos esses testes. O mérito de discutir qual dos dois sistemas é mais eficiente a nível de qualidade e tempo de processamento é mostrado em seu final. Veremos que, apesar do nosso sistema ser um pouco mais complexo, o conveniente de resolvê-lo por meio da fatoração de Cholesky torna o algoritmo bem mais rápido mantendo a qualidade da colorização.

4. Resultados

Para podermos verificar resultados e realizar comparativos, utilizamos um editor gráfico de código aberto: *EasyPaint 0.1.0* [41] como base. Após estudar o código, implementamos filtros capazes de executar o algoritmo de colorização via otimização provendo uma interface agradável e a facilidade ao usuário de manipular determinados parâmetros do problema.

Assim sendo, a primeira seção deste capítulo especifica questões técnicas de implementação. Em seguida, fazemos testes modificando parâmetros que foram estabelecidos ao longo do nosso desenvolvimento. São eles: parâmetro de penalidade, função de afinidade e raio de vizinhança. Não obstante, confrontaremos os resultados gerados entre a nossa abordagem de resolução e a proposta utilizada pelos autores do problema. Encerramos então este capítulo mostrando um conjunto de imagens diversificadas que foram coloridas pelo nosso algoritmo.

4.1 Detalhes Técnicos de Implementação

A implementação das rotinas foi feita na linguagem de programação C++ juntamente com uma framework denominada Qt. As bibliotecas e sub-rotinas relacionadas com os procedimentos de Álgebra Linear necessários utilizadas foram: BLAS, LAPACK, CHOLMOD, UMFPACK e OpenBLAS. Descrevemos a seguir cada uma dessas ferramentas assim como a motivação de uso:

- Qt é uma framework para desenvolvimento de aplicações multiplataforma. A API é rica, útil, torna a programação em C++ mais amigável, facilita na criação de uma interface robusta e, principalmente, oferece rotinas especializadas para o tratamento de imagens. A versão utilizada foi a 4.8.4, já que apresenta maior estabilidade em relação às demais [38].
- BLAS (*Basic Linear Algebra Subprograms*) é um conjunto específico de sub-rotinas de baixo nível que realizam operações comuns de Álgebra Linear como: dimensionamento de vetores, produto escalar de vetores, combinações lineares e multiplicação de matrizes. Essas sub-rotinas podem ser vistas como uma API padrão para que bibliotecas de Álgebra Linear mais específicas sejam

desenvolvidas (em particular, LAPACK, como veremos adiante) [34]. O seu uso está diretamente relacionado com a velocidade de processamento.

- LAPACK (*Linear Algebra Package*) é uma biblioteca focada em Álgebra Linear numérica. Ela provê rotinas para resolver sistemas de equações lineares, códigos especializados na realização de processos como diagonalização, inversão, fatoração e é capaz de tratar matrizes densas e esparsas. Suas rotinas são escritas de modo a realizar o máximo dos cálculos necessários por meio de chamadas ao BLAS [32]. Seu uso também está associado à otimização do tempo de processamento.
- CHOLMOD é um conjunto de rotinas capazes de fatorar uma matriz esparsa simétrica definida positiva da forma A ou AA^T na forma LL^T, atualizar uma fatoração de Cholesky, resolver sistemas lineares e promover facilidades de manuseio para matrizes esparsas. Para que tais funcionalidades executem de forma adequada, CHOLMOD requer que LAPACK e BLAS estejam disponíveis e associados à biblioteca [31]. Seu uso se mostra importante para resolver o principal problema de otimização deste trabalho.
- UMFPACK é um conjunto de rotinas capazes de resolver sistemas lineares não simétricos e esparsos do tipo $Ax = b$ por meio de uma fatoração LU. Essa biblioteca também depende do conjunto de sub-rotinas BLAS para que o desempenho de execução seja razoável [39]. Nós a utilizamos para reproduzir a implementação dos autores.
- OpenBLAS é uma biblioteca portátil e otimizada baseada em BLAS. Muitas implementações de bibliotecas que utilizam BLAS foram desenvolvidas para arquiteturas de computadores específicas visando diminuir ainda mais o tempo de suas operações básicas. No nosso caso, para a máquina onde o trabalho foi desenvolvido, a biblioteca otimizada apropriada é OpenBLAS, uma vez que adiciona implementações otimizadas para a arquitetura de processadores Intel Sandy Bridge[5].

[5] Sandy Bridge é o nome dado a microarquitetura desenvolvida pela Intel presente nos processadores da família Core [37].

Resumidamente, CHOLMOD e UMFPACK são as bibliotecas responsáveis por prover a fatoração de Cholesky e LU, respectivamente. OpenBLAS e BLAS disponibilizam um conjunto de rotinas de Álgebra Linear mais triviais enquanto que LAPACK as utiliza para fornecer rotinas mais especializadas. A Figura 4.1 ilustra tais ferramentas e seus níveis de relacionamento de uma forma simples e informal.

Figura 4.1: Esquema de relacionamento entre as bibliotecas, sub-rotinas e framework.

Todos os testes que serão mostrados nas subseções sequentes foram realizados em uma máquina com processador Intel® Core™ i5-2500, com clock de 3.30 GHz, 8GB de memória RAM e rodando o sistema operacional Ubuntu 13.10 64 bits.

4.2 Parâmetros do Problema

Durante o desenvolvimento deste trabalho, citamos em alguns momentos especificações do problema de colorização via otimização que podem variar. Essas especificações podem ser vistas como parâmetros do mesmo e, durante o estudo de testes e resultados, devem ser analisados discretamente.

No início do Capítulo 2, enquanto descrevíamos o processo de modelagem, mencionamos as implicações que o tamanho do raio da vizinhança acarretam na matriz de afinidade W. Vimos que quanto maior for o raio, mais densa essa matriz se torna.

Em seguida, apontamos três funções de peso (funções de afinidade) das quais apenas duas são relevantes: a primeira é a padrão e foi proposta pelos autores em [11] enquanto que a segunda foi uma sugestão nossa.

Ainda no Capítulo 2, deixamos em aberto a questão da ordem de grandeza do parâmetro de penalidade. Lembrando, esse valor é estritamente positivo e responsável por garantir o cumprimento da restrição de igualdade conseguinte dos pixels que foram coloridos.

Assim sendo, os parâmetros que serão tratados são: parâmetro de penalidade, tamanho do raio da vizinhança e função de afinidade. As próximas subseções se encarregam da tarefa de analisar resultados provenientes de variações nesses números. A ordem de abordagem foi escolhida para melhor fluidez do raciocínio.

4.2.1 Parâmetro de Penalidade

Para definir o parâmetro de penalidade (λ) de forma adequada, realizamos testes com diversas imagens de tamanhos diferente; variamos o seu valor e observamos a qualidade dos resultados. A métrica utilizada para comparar a qualidade das imagens obtidas foi dada pelo cálculo do valor de PSNR.

Esclarecendo seu funcionamento, PSNR (*Peak signal-to-noise ratio*) é uma forma comumente usada para medir qualidade de gravuras reconstruídas, comprimidas ou tratadas. Seu valor é dado em dB e, quanto maior, mais semelhantes as imagens comparadas são. Para isso, um cálculo é realizado levando em consideração a gravura modificada e original. Sua fórmula é mostrada a seguir:

$$PSNR = 10log\left(\frac{{MAX_I}^2}{MSE}\right)$$

onde MAX_I é o valor máximo que um valor de pixel pode assumir e MSE (*mean squared error*) é a média da diferença ao quadrado entre todos os valores que cada pixel pode assumir. No nosso caso, comparamos duas imagens no espaço de cor RGB e, portanto, $MAX_I = 255$. Logo, MSE é dado por

$$MSE = \frac{1}{3mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left[\left(R_{ij} - R'_{ij}\right)^2 + \left(G_{ij} - G'_{ij}\right)^2 + \left(B_{ij} - B'_{ij}\right)^2\right]$$

em que m e n são as dimensões da imagem, R, G e B são as componentes de cor da imagem original e R', G' e B' são as componentes de cor da imagem modificada [40].

Posto isso, dentre o conjunto de gravuras as quais aplicamos tal teste, escolhemos para exemplificar uma de tamanho 750×483 pixels ao qual chamaremos de *Química*. Sua escolha foi motivada por possuir um número de cores bem definido e por apresentar transições também bem delineadas. A Figura 4.2 mostra essa imagem colorida (original) e em tons de cinza.

Figura 4.2: Imagem de teste: *Química*. Original e preto e branco.

Utilizando a imagem em tons de cinza, introduzimos os rabiscos. Logo após, executamos o algoritmo de colorização para valores do parâmetro de penalidade (λ) assumindo 10^{-5}, 1 e 10^{5}. O raio de vizinhança escolhido foi 1 e a função de afinidade usada foi a padrão (Equação 2.16). A Figura 4.3 mostra a imagem original com os rabiscos (entrada do algoritmo) ao passo que a Figura 4.4 mostra os resultados da execução.

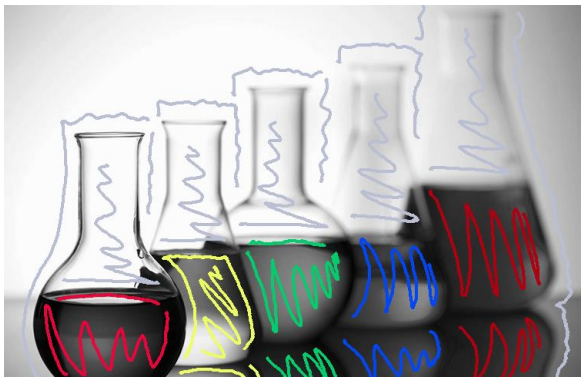


Figura 4.3: Imagem de teste rabiscada.

Figura 4.4: Resultados da colorização para λ igual a 10^{-5}, 1 e 10^{5}, respectivamente.

Visualmente falando, é difícil notar alguma diferença entre os três resultados. O mesmo acontece para imagens de dimensões bem diferentes. Portanto, concluímos que, independentemente da ordem de grandeza de λ e das dimensões da imagem, o resultado para o problema é o mesmo. A Tabela 4.1 mostra os valores de PSNR para as três saídas.

λ	10^{-5}	1	10^{5}
PSNR (dB)	20.0358	21.5417	21.5378

Tabela 4.1: Valor de PSNR para as três imagens de testes.

Observe que todos esses valores são muito semelhantes, indicando que qualquer uma das imagens geradas apresentam colorizações muito parecidas frente à gravura original.

Comparando os valores de PSNR entre as três imagens obtidas, podemos também verificar o quão semelhante elas são. A Tabela 4.2 apresenta essa relação.

λ	10^{-5}	1	10^{5}
10^{-5}	99	36.1702	36.1021
1	36.1702	99	66.8532
10^{5}	36.1021	66.8532	99

Tabela 4.2: Valor de PSNR entre as três imagens de teste.

Ela nos mostra que as imagens com os maiores parâmetros de penalidade são altamente semelhantes. Já para $\lambda = 10^{-5}$, esses mesmos valores não se verificam. Isso significa que quando o parâmetro de penalidade tende a zero, a imagem produzida se torna ligeiramente diferente do resultado que seria gerado para valores de λ maiores.

A explicação para esse acontecimento pode ser derivada de todo o nosso processo de resolução. Vimos no Capítulo 3 que a matriz dos coeficientes do sistema linear do problema se torna definida positiva pelo fato do parâmetro de penalidade ser estritamente positivo. Caso ele fosse zero, essa matriz seria semidefinida positiva. Assim sendo, acreditamos que a ordem de grandeza de λ é responsável por transformar o formato de "calha" da função em um formato de "cuia". Quanto maior for seu valor, mais fechado seria o paraboloide elíptico (gráfico da função) enquanto que, à medida que esse valor diminuir, tal superfície degeneraria gradualmente para um cilindro elíptico.

4.2.2 Raio da Vizinhança

O tamanho do raio da vizinhança (R) determina o número de pixels que serão influenciados por cada pixel em particular. No Capítulo 2, discutimos sua importância. Em síntese, quanto maior for seu valor, mais cálculos da função de afinidade devem ser realizados e mais denso o sistema linear do problema se torna. Assim sendo, é natural esperarmos que os resultados gerados para valores de R maiores sejam melhores em troca de tempo de processamento.

Dentre as imagens utilizadas nesta fase de teste, selecionamos duas para exemplificar esta subseção. A primeira delas tem tamanho de 800×600 pixels e chamaremos de *Barco*. A segunda delas tem tamanho 768×768 pixels e denominaremos de *Beagle*. As Figura 4.5 e 4.6 mostram essas duas imagens em seus formatos originais e em tons de cinza, respectivamente.

Figura 4.5: Imagens de teste originais: *Barco* e *Beagle*.

Figura 4.6: Imagens de teste em preto e branco: *Barco* e *Beagle*.

A seguir, as gravuras de teste rabiscadas são apresentadas pela Figura 4.7 ao passo que os resultados para valores de tamanho de vizinhança iguais a 1, 2 e 3, são mostrados nas Figuras 4.8 e 4.9.

Figura 4.7: Imagens de teste rabiscadas: *Barco* e *Beagle*.

Figura 4.8: Resultados para *Barco* para *R* igual a 1, 2 e 3, respectivamente.

Figura 4.9: Resultados para *Beagle* para *R* igual a 1, 2 e 3, respectivamente.

Como podemos observar, os resultados gerados foram muito parecidos - contrariando a nossa intuição. As diferenças entre cada conjunto de três imagens são praticamente inexistentes. Afim de obter uma análise mais minuciosa, atentaremos mais uma vez para os valores de PSNR e também ao tempo de processamento. As Tabelas 4.3 e 4.4 expõem esses números, nesta ordem.

Imagem\R	1	2	3
Beagle	27.9608	27.9880	27.9919
Barco	28.6391	28.6431	28.7014

Tabela 4.3: Valores de PSNR (dB) para os seis resultados.

Imagem\R	1	2	3
Beagle	5.636	24.809	229.014
Barco	4.086	21.342	210.530

Tabela 4.4: Tempo de processamento (s) para os seis resultados.

Repare que os valores de PSNR são semelhantes. Apesar de ligeiramente maiores à medida que o tamanho de raio aumenta, esses valores não representam nenhuma melhora visual aos resultados. Não somente, o tempo de processamento relacionado aumenta de forma exorbitante, tornando a escolha de R superior a 1 desnecessária e até mesmo inviável.

A explicação para essa diferença nos tempos consumidos é óbvia. É necessário calcular mais pesos relativos, fatorar uma matriz com um número muito maior de entradas e resolver um sistema linear mais denso. A Tabela 4.5 exibe o tempo gasto em cada uma das principais etapas do algoritmo para a imagem *Beagle*. A nível de esclarecimento, preencher tupla e convertê-la em matriz são mecanismos utilizados por inúmeras bibliotecas de Álgebra Linear para tratar casos esparsos.

Etapas\R	1	2	3	1	2	3
Preencher tupla	0.399	1.136	2.192	7%	4%	0%
Converter tupla em matriz	0.114	0.267	0.523	2%	1%	0%
Calcular $A^T A$	0.252	1.450	4.185	4%	5%	1%
Calcular $A^T A + \lambda C$	0.095	0.290	0.596	1%	1%	0%
Executar fatoração	4.108	19.457	155.732	72%	78%	68%
Resolver sistema	0.651	2.190	65.769	11%	8%	28%

Tabela 4.5: Tempo de processamento (s) e porcentagem para as seis principais etapas do algoritmos. Imagem usada: *Beagle*.

É interessante perceber que as etapas mais dispendiosas são: a fatoração da matriz, isto é, o momento em que o algoritmo calcula LL^T, e a resolução de fato do sistema linear. Podemos verificar também que, quando diminuímos o grau de esparsidade da matriz dos coeficientes (aumentando o raio), o tempo gasto nessas duas etapas aumenta consideravelmente.

Por fim, podemos afirmar que a escolha adequada para o tamanho de raio da vizinhança de um pixel deve ser fixada em 1. Apesar de nosso programa permitir a liberdade ao usuário de escolher outros valores, mostramos que tal tentativa não irá trazer benefícios condizentes com o tempo de processamento que será consumido.

4.2.3 Função de Afinidade

As duas funções de afinidade discutidas no capítulo de modelagem são reescritas abaixo:

$$w_{rs} = e^{-(Y_r - Y_s)^2 / \sigma_r^2}, \tag{4.1}$$

$$w_{rs} = e^{-(Y_r - Y_s)^2 / t}. \tag{4.2}$$

A primeira foi exposta pelos autores e a segunda foi uma sugestão nossa. Nela, definimos t como um parâmetro de limiar constante responsável por ponderar o expoente: quanto maior for seu valor, mais inflexível será o peso; quanto menor for o seu valor, mais sensível o peso será perante à diferença de intensidades.

Dito isso, faremos testes utilizando as Equações (4.1) e (4.2) para $t = 10$ (valor suficientemente pequeno que a variância σ_r^2 pode assumir) e $t = 16256.$ (valor

aproximadamente máximo que a variância σ_r^2 pode assumir). Chamaremos essas funções de *#1*, *#2* e *#3*, respectivamente.

Já para avaliar os diferentes resultados gerados, utilizamos 8 imagens distintas onde cada uma delas possui características interessantes. Entre esse conjunto de gravuras, mostraremos a seguir *Chocolate Quente* (512 × 512 pixels) que tem poucas mudanças de cores entre pixels adjacentes, *Salada de Frutas* (512 × 512 pixels) que apresenta várias cores mas com transições de contorno bem definidas e *Papagaio* (512 x 512 pixels) - imagem clássica no âmbito da Computação Gráfica. As fotos originais e em tons de cinza são mostradas a seguir pela Figura 4.10 e pela Figura 4.11, respectivamente.

Figura 4.10: Imagens originais utilizadas nos experimentos: *Chocolate Quente* (esquerda), *Salada de Frutas* (meio), *Papagaio* (direita).

Figura 4.11: Imagens em tons de cinza utilizadas nos experimentos: *Chocolate Quente* (esquerda), *Salada de Frutas* (meio), *Papagaio* (direita).

A Figura 4.12 exibe essas imagens com os rabiscos adicionados enquanto que os resultados gerados da execução do algoritmo para as três funções de afinidade (*#1*, *#2* e *#3*) são expostos pelas Figuras 4.13, 4.14 e 4.15, nessa ordem.

Figura 4.12: Imagens de teste em tons de cinza rabiscadas: *Chocolate Quente* (esquerda), *Salada de Frutas* (meio), *Papagaio* (direita).

Figura 4.13: Saída para função de afinidade *#1*.

Figura 4.14: Saída para função de afinidade *#2*.

Figura 4.15: Saída para função de afinidade *#3*.

Para medir a qualidade dos resultados, recorremos novamente ao cálculo do valor de PSNR que são apresentados pela Tabela 4.6. Note que esses números são ligeiramente diferentes mas, mesmo assim, informações interessantes podem ser extraídas.

Imagem\Função	Função de Afinidade *#1*	Função de Afinidade *#2*	Função de Afinidade *#3*
Chocolate	26.5444	26.4705	26.4667
Salada de Frutas	20.7356	21.0378	20.4342
Papagaio	23.7186	23.5946	22.0340
Peixe	19.0163	19.0253	18.8423
Babuíno	28.3870	28.4699	28.2008
Maçã	24.6926	25.7546	23.6096
Barco	28.6391	28.7245	27.7280
Beagle	27.9608	27.8862	27.7950
Média	24.9618	25.1204	24.3888

Tabela 4.6: Valores de PSNR para diversas imagens e funções de afinidade.

Lembrando o que foi discutido na Subseção 2.3.2, acreditamos que a função *#2* funcionaria melhor para imagens onde a transição das cores é mais bem delimitada. Enquanto que, a função de afinidade *#3*, intuitivamente se comportaria melhor para gravuras formadas por mais gradientes de cores.

De fato, os valores de PSNR mostram essa tendência para o primeiro caso. Observe os números para *Salada de Frutas* e *Maçã* (imagens com transições de cores bem determinadas). A função de afinidade #2 se mostra superior.

Para ficar mais claro essas questões, visualizaremos com mais detalhe a imagem que demonstrou maior discrepância no valor de PSNR: *Maçã*. A Figura 4.16 coloca seu formato original, em tons de cinza e rabiscada. Em seguida, a Figura 4.17 mostra a saída para as três funções de afinidade.

Figura 4.16: Imagem de teste: *Maçã*. Original , preto e branco e rabiscada.

Figura 4.17: Saída para as funções de afinidade #1, #2 e #3, respectivamente.

Preste bem atenção aos contornos da maçã. As saídas para as funções #1 e #3 são mais borradas em suas extremidades ao passo que a saída da função #2, tem seus limites bem definidos - o vermelho da maçã não é propagado para o fundo da gravura assim como o cinza do fundo não se estende à maçã . A Figura 4.18 destaca essas minúcias.

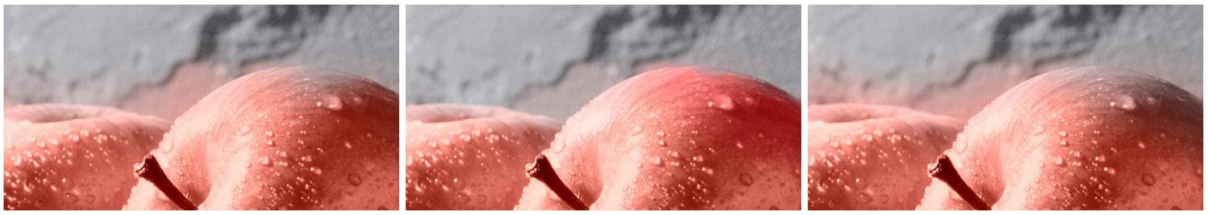


Figura 4.18: Saída para as funções de afinidade #1, #2 e #3 no detalhe, respectivamente.

Deste modo, encerramos discussões acerca da qualidade de saída gerada pelas funções de peso. Verificamos que não há uma equação ideal para qualquer tipo de imagem e que as três funções de afinidade abordadas apresentam comportamentos relativamente parecidos. Mesmo assim, deixamos claro seus pontos de contraste apontando como a manipulação do valor de limiar influência nos resultados.

Assim, nos resta analisar os tempos de processamento na geração desses resultados. Uma vez que o cálculo da variância pode ser incorporado à implementação do algoritmo sem adição de complexidade, todas as três funções apresentam durações de execução aproximadas. A diferença no consumo de tempo só é observada quando variamos as dimensões da imagem, assunto que será tratado na próxima seção.

4.3 Tempo de Execução

O tempo de execução depende das dimensões da gravura independentemente da ilustração. Isso acontece pois a dimensão da matriz dos coeficientes é dada pelo número de pixels que a imagem contém. Sendo assim, a etapa de fatoração e resolução do sistema se torna mais demorada.

Para fazer esse teste, colorimos uma imagem muito conhecido no meio da edição digital: *Lena*. Executamos o algoritmo para essa fotografia dimensionada em 6 tamanhos diferentes visando analisar o tempo gasto. A Figura 4.19 expõe a sua versão original, em tons de cinza, rabiscada e gerada pelo algoritmo. Logo após, a Figura 4.20 apresenta o crescimento do tempo em segundos para imagens de tamanhos diferentes.

Figura 4.19: Imagem de teste *Lena*. Original, escalada de cinza, rabiscada e resultado.

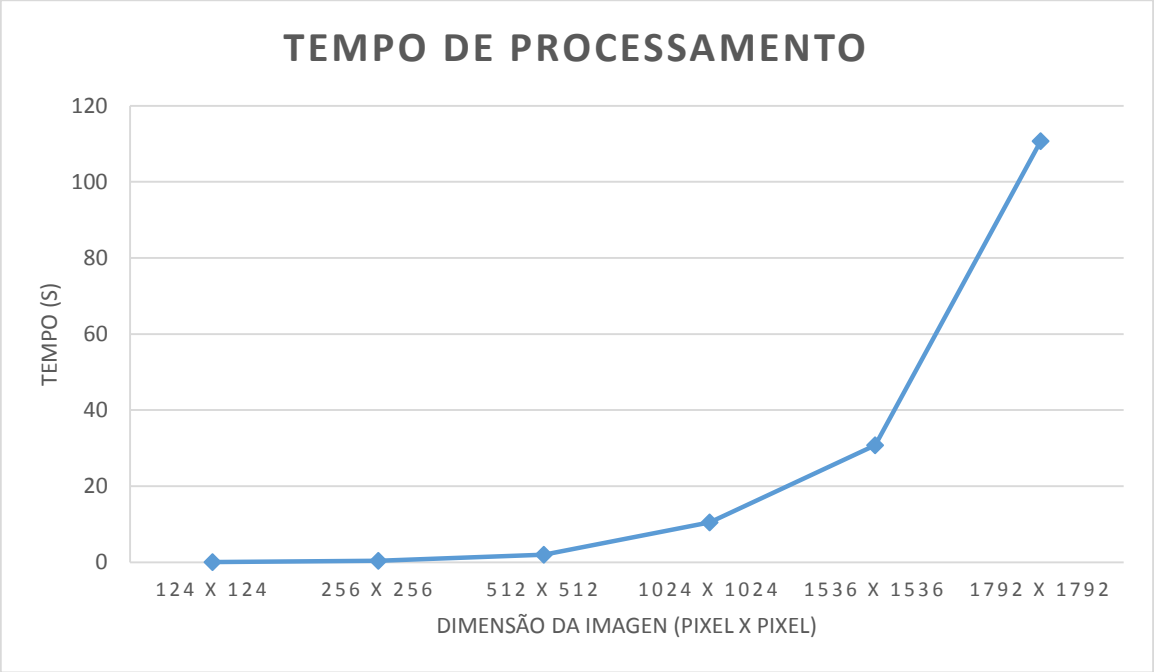


Figura 4.20: Tempo de execução para imagens de dimensões distintas.

Observando o gráfico, é possível notar que o crescimento do tempo de processamento se aproxima de uma curva exponencial. Para dimensões muito altas, o número de pixels se torna extremamente elevado e, consequentemente, a resolução do sistema já não é mais viável. Se formos fazer os cálculos para a imagem de tamanho 1792×1792 pixels, teríamos uma matriz de dimensão 3211264×3211264 com aproximadamente 28901376 entradas não nulas (menos de 1% do total).

O tempo gasto para colorir essa gravura foi de quase 2 minutos - o que não é muito aceitável frente à proposta do algoritmo. Deste modo, não recomendamos o uso do método para fotografias cujas dimensões ultrapassem um número de pixels de aproximadamente 3×10^6. Mesmo assim, podemos adiantar que essa limitação será discutida e contornada no Capítulo 5.

4.4 Implementação dos autores

No final do Capítulo 3, levantamos uma discussão que envolvia a nossa maneira de resolver o problema de colorização e a maneira proposta pelos autores. Vimos que o sistema linear que eles propuseram, apesar de se assemelhar com o nosso, é mais esparso, requer menos cálculos mas conta com uma matriz dos coeficientes que não é simétrica. Por esse motivo, a implementação que eles aventaram faz uso da fatoração LU.

Para comparar da melhor maneira possível ambas as abordagens, implementamos um algoritmo muito análogo ao disponível no site dos autores [12]. Tomamos o cuidado de utilizar a mesma biblioteca de resolução empregada e tentar ser o mais fiel possível ao código original. Assim sendo, adicionamos ao nosso programa um filtro extra com a opção de executar tal procedimento.

Seguindo a linha de raciocínio adotada durante as demonstrações dos resultados, iremos aferir a qualidade das saídas geradas pelos dois algoritmos além de estudar o tempo de processamento. Vale lembrar que todos os testes estão sendo executados para $\lambda = 1$, $R = 1$ e utilizando a função de afinidade *#1*.

Assim sendo, chamamos a nossa implementação de *#A* e a implementação dos autores de *#B.* A Figura 4.21 mostra as imagens *Sapo* (1000×750 pixels) e *Grumpy Cat* (800×991 pixels) originais, em tons de cinza e rabiscadas, respectivamente. Prontamente, as Figuras 4.22 e 4.23 exibem as saídas geradas pelo nosso algoritmo e o dos autores.

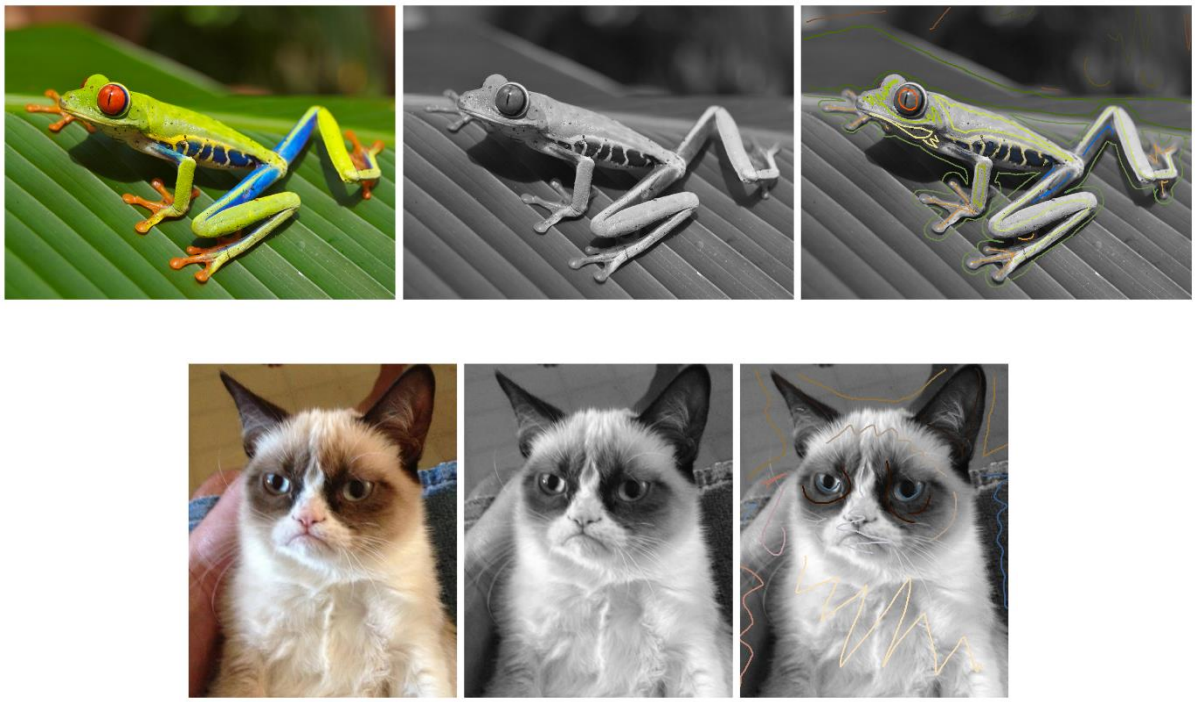


Figura 4.21: Imagens de teste *Sapo* e *Grumpy Cat*. Originais, em tons de cinza e rabiscadas.

Figura 4.22: *Sapo*: Saída para a nossa implementação (esquerda) e dos autores (direita).

Figura 4.23: *Grumpy Cat*: saída para a nossa implementação (esquerda) e dos autores (direita).

Observando os resultados, fica difícil notar alguma diferença evidente. Ambas as saídas são suficientemente adequadas. Isso não é surpreendente. Vimos na Subseção 3.4.4 que as funções minimizadas nos dois casos são muito similares. Sendo assim, segue a tabela com os valores de PSNR para essas duas gravuras afirmando essa semelhança.

Imagem\Implementação	*#A*	*#B*
Sapo	23.5612	23.3502
Grumpy Cat	24.8907	24.9748

Tabela 4.7: Valores de PSNR (dB) para saídas de implementações diferentes.

Uma vez que estudamos a qualidade dos resultados, está na hora de observarmos o tempo de execução desses algoritmos. Lembremos que, enquanto o nosso necessita realizar uma multiplicação de matrizes esparsas, a fatoração que ele utiliza é Cholesky já que a matriz dos coeficientes é simétrica definida positiva. Em contrapartida, o algoritmo de resolução dos autores não precisa desse passo de multiplicação de matrizes mas se apoia na fatoração LU em razão de sua matriz não ser simétrica.

O mesmo teste de colorização usado para calcular a variação do tempo em função das dimensões da imagem foi feito. Aqui, usamos quatro exemplares da gravura *Lena* em tamanhos diferentes. A Figura 4.24 mostra o tempo total de execução para os dois casos.

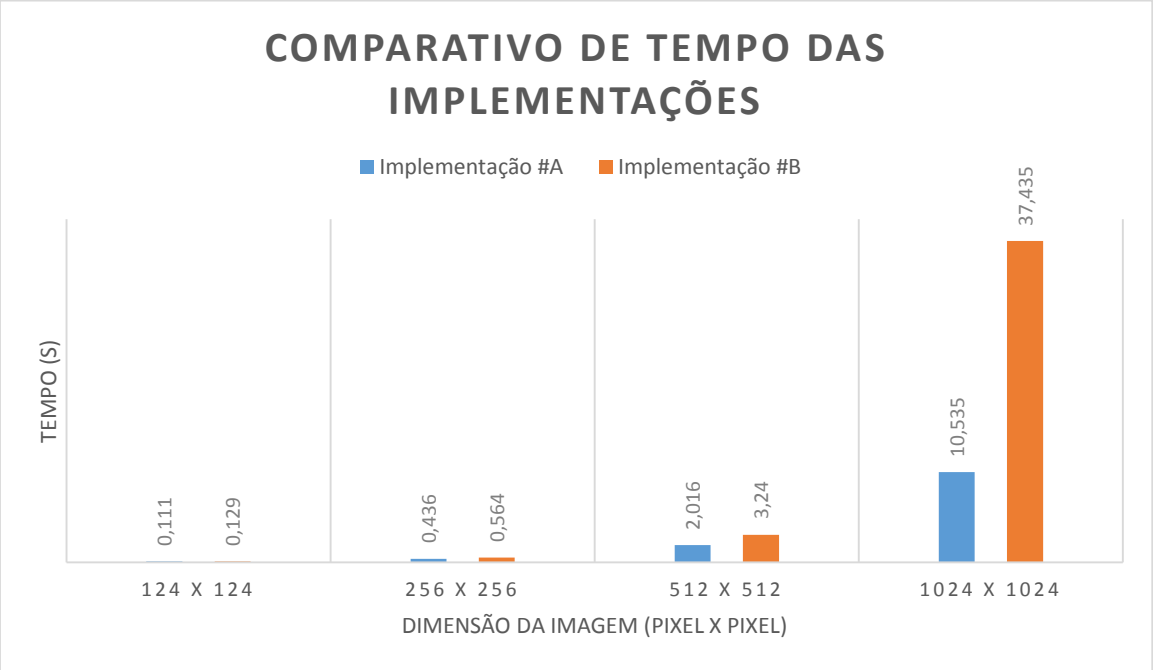


Figura 4.24: Comparação do tempo de execução entre as duas implementações.

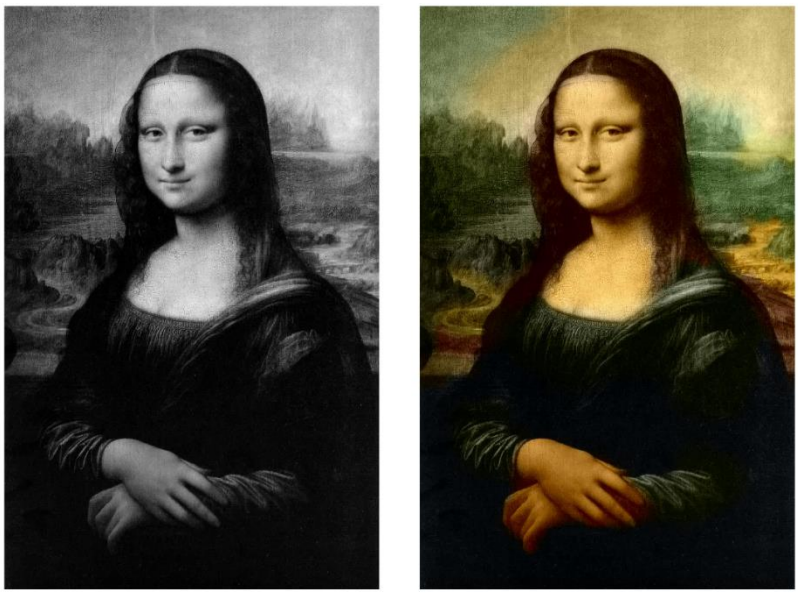
Quando observamos os tempos de execução, percebemos que a nossa forma de resolução se mostra superior mesmo para imagens com dimensões não muito grandes. A partir de aproximadamente 512×512 pixels, ganhos de consumo de tempo podem ser notados e tendem a aumentar em função das dimensões da imagem. Isso significa que o tempo gasto na multiplicação de matrizes esparsas é justificável pelo uso de um método de fatoração mais refinado gerando benefícios.

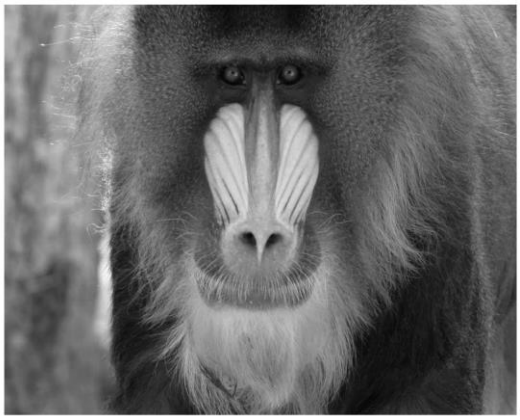
Finalmente, diante desses resultados mostrados, provamos que de fato a nossa implementação é mais eficiente. Justificamos por meio de testes que a nossa forma de resolução baseada nas restrições que foram estabelecidas ao longo do desenvolvimento deste trabalho, garante uma versão do algoritmo de colorização via otimização oportuna e melhorada.

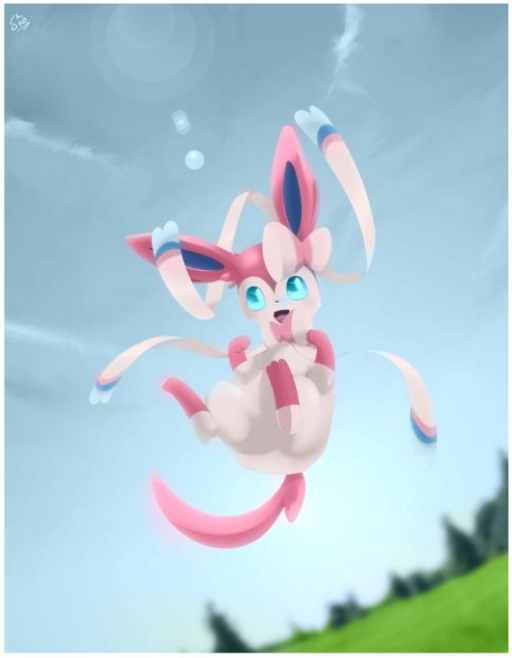
4.5 Outros exemplos e considerações

A seguir mostramos um total de 6 gravuras pertencentes a classes de fotografias distintas que foram coloridas pela nossa aplicação. Dentre elas selecionamos fotos tradicionais, artes digitais, fotografias antigas e imagens pessoais.

Dessa forma, poderemos ter a certeza de que bons resultados podem ser gerados para qualquer tipo de imagem. Os parâmetros utilizados na geração das saídas que seguem foram: $\lambda = 1$, $R = 1$ e a função de afinidade escolhida foi #1.







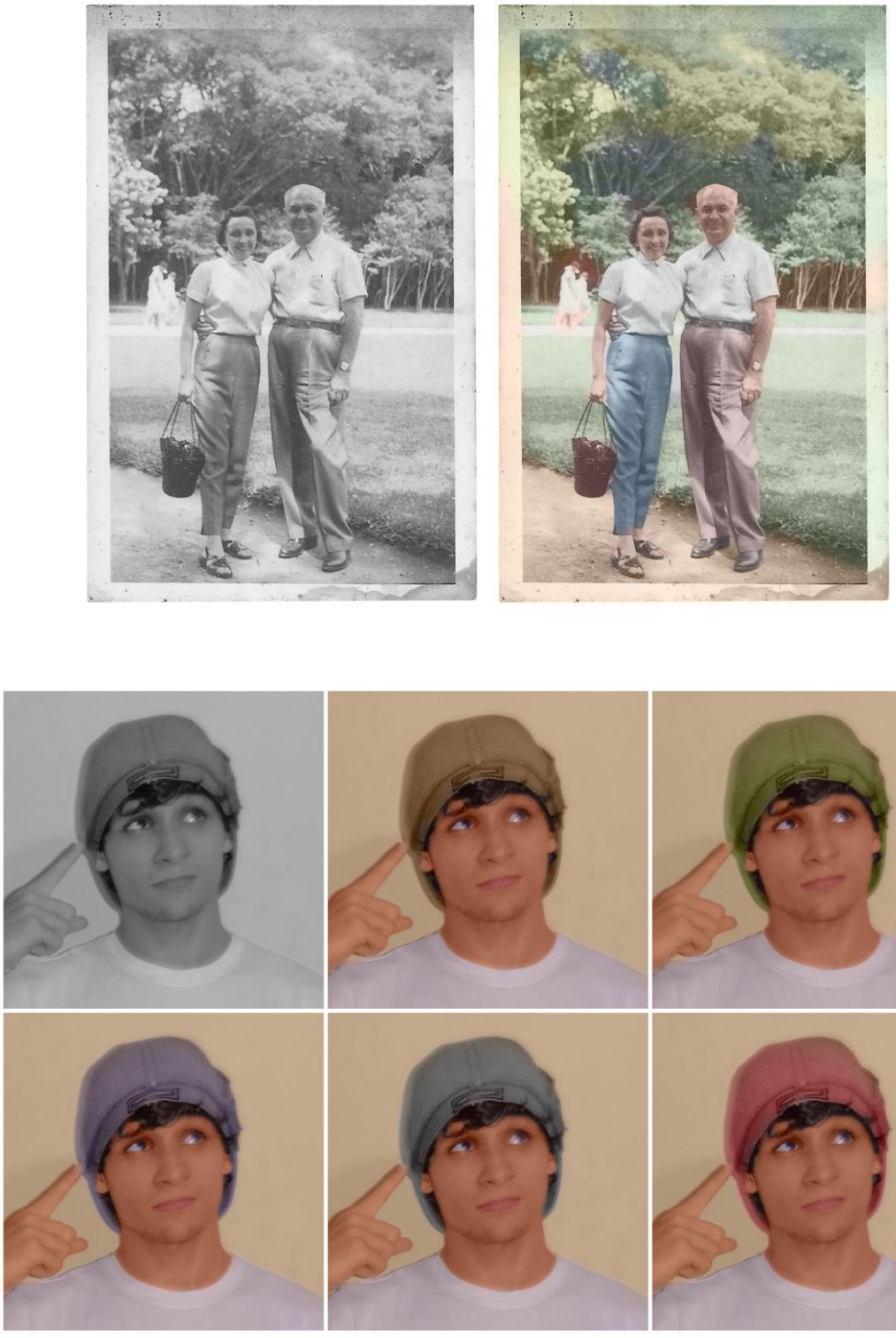


Figura 4.25: Exemplos variados de imagens coloridas pela nosso aplicativo.

Encerraremos assim as discussões sobre os resultados e damos seguimento ao último assunto deste trabalho. O próximo capítulo introduz uma ideia capaz de reduzir drasticamente o tempo de processamento do algoritmo de colorização.

5. Redimensionamento de Solução e Colorização Iterativa

Os resultados apresentados no Capítulo 4 foram promissores. De fato, a nossa forma de resolver o sistema linear decorrente do problema de colorização via otimização é mais eficiente. Ficamos satisfeitos com os números e o melhoramento proposto para o algoritmo mas, mesmo assim, sabemos que imagens de resoluções muito altas ainda tenderão a consumir um grande tempo de processamento.

Já não é de hoje que imagens digitais não param de crescer: fotografias geradas pelas câmeras mais simplórias disponíveis no mercado costumam oferecer pelo menos 10 megapixels, o incontestável progresso dos smartphones tem aumentado ainda mais a resolução das fotografias tiradas por celulares e, não somente, gravuras atingindo a casa dos gigapixels já são uma realidade (produto de câmeras especializadas ou decorrente da criação de panoramas, por exemplo).

Diante disso, sentimos a necessidade de melhorar ainda mais o algoritmo de colorização. A medida que avaliávamos os resultados do capítulo anterior, tivemos uma ideia capaz de fazer com que tal método execute de forma extremamente rápida; para muitas imagens, em tempo real.

Sendo assim, este penúltimo capítulo inicialmente formaliza essa ideia explicando sua motivação. Fazemos depois uma breve discussão sobre um outro ferramental matemático empregado para que esse novo método possa ser implementada e, por fim, mostraremos os resultados gerados.

5.1 Formalização da Ideia

Avaliando os resultados do capítulo anterior, em especial, os obtidos na Subseção 4.3 e exibidos pela Figura 4.20, chegamos à conclusão de que o tempo de processamento cresce exponencialmente à medida que as dimensões da imagem aumenta. Isso não foi nenhuma surpresa, afinal, resolver um sistema linear com 124×124 variáveis definitivamente é menos custoso do que resolver um com 1792×1792 variáveis.

Nosso algoritmo gasta por volta de 100 milissegundos para colorir uma imagem de tamanho 128×128 pixels (dentro das especificações da máquina mostradas no início do Capítulo 4) e aproximadamente 110 segundos para colorir a mesma imagem mas com dimensões iguais a 1792×1792 pixels onde, a maior fatia de tempo em ambos os casos se concentra na etapa de fatoração e resolução dos sistemas lineares correspondentes.

Ao observar mais atentamente essa análise uma pergunta é levantada: será que a colorização de uma mesma imagem com dimensões tão diferentes é também muito diferente? A Figura 5.1 responde esta pergunta na forma de um resultado. Ela mostra uma imagem colorida de dimensões 128×128 pixels e 1792×1792 pixels para conjuntos de rabiscos equivalentes.

Figura 5.1: Resultado do processo de colorização para imagens de tamanhos 128×128 pixels (esquerda) e 1792×1792 pixels (direita).

Obviamente, a qualidade delas é diferente uma vez que as resoluções estão em ordens de grandeza discrepantes mas, em termos de colorização, ambos os resultados são bem semelhantes. Em outras palavras, as cores foram propagadas para as mesmas regiões.

Hora, se resolver esse sistema linear pequeno é tão rápido e gera uma solução parecida (visualmente falando) a do sistema linear grande, será que não existe uma forma de interpolar essa solução e gerar uma nova que seja adequada para a imagem de alta resolução? A resposta é sim. Uma classe de filtros presentes no meio da Computação Gráfica conhecidos como *upsampling* apresentam exatamente esse propósito.

Diante dessa pequena discussão, a ideia por trás do nosso método de colorização via otimização em tempo real já deve ter ficado clara. A partir de uma imagem de alta resolução,

iremos primeiramente redimensioná-la tornando-a suficientemente pequena - por exemplo, dimensionaremos a fotografia de tamanho 1792×1792 pixels para 128×128 pixels. Na imagem de baixa resolução, efetuamos a colorização e obtemos uma solução: um vetor U e um vetor V. Por meio de um filtro de *upsampling* - ao qual entraremos no mérito logo em seguida - iremos interpolar a solução de baixa resolução gerando uma solução de alta resolução e, finalmente, iremos colorir a imagem original.

Essa nossa ideia sugere que estamos trocando o tempo gasto na resolução de um sistema linear grande pela execução desse filtro. A princípio, isso é verdade. Contudo, veremos ao longo deste capítulo que, ao contrário de precisar resolver um sistema enorme (para imagens de resolução alta) toda vez que um novo conjunto de rabiscos é inserido na foto, por meio da nossa forma de implementação a etapa dispendiosa do filtro de *upsampling* (que ainda sim é mais barata que a resolução do sistema linear maior) só será executada uma única vez. Assim, o nosso algoritmo se resumiria basicamente a resolver o sistema linear de baixa resolução; o que é feito de forma muito rápida. Todos esses detalhes ficarão claros ao longo deste capítulo.

5.2 Operação de *Upsampling* e Filtro Bilateral

Para darmos seguimento a formalização das técnicas envolvidas no nosso propósito, dois conceitos devem ser explicitados. A seguir entenderemos o que de fato é uma operação de *upsampling* e qual é o significado e importância de um filtro bilateral.

Upsampling é uma operação fundamental em processamento de imagens vastamente utilizada em aplicações como compressão, transmissão progressiva e exibição de imagens. Seu principal propósito é aumentar a resolução enquanto mantém a representação 2D de uma gravura. Esse método é tipicamente utilizado para ampliação em pequenas regiões de algumas imagens e para eliminação de efeitos de pixelização que são gerados quando imagens de baixa resolução são exibidas em frames relativamente maiores [44]. De modo mais simples, upsampling é uma forma de reamostrar uma imagem de baixa resolução em uma grade de alta resolução.

Já um filtro bilateral é uma técnica não linear capaz de suavizar uma determinada imagem ao mesmo tempo que preserva suas arestas. Para isso, combina-se filtros de espaço (distância entre pixels) e de alcance (valores do espaço de cor em questão). O valor de intensidade de cada pixel na imagem é substituído por uma média ponderada das intensidades de pixels próximos. O interessante é que esses pesos não dependem apenas da distância

euclidiana dos pixels (filtro de espaço) mas também da variação dos valores de intensidade (filtro de alcance) [42]. Isso significa que para o filtro bilateral dois pixels são próximos um do outro (apresentam pesos maiores) não somente se ocupam posições espacialmente próximas, mas também se eles tem alguma similaridade na escala fotométrica; no nosso caso, escala de cinza.

Para exemplificar, um filtro bilateral é tipicamente dado pela seguinte equação:

$$J_p = \frac{1}{k_p} \sum_{q \in \Omega} I_q f(\|p - q\|) f\left(\left\|I_p - I_q\right\|\right), \tag{5.1}$$

onde J_p é a imagem filtrada, I é a imagem de entrada a ser filtrada (e, por exemplo, seus valores representam as intensidades), p é a coordenada do pixel atual a ser filtrado, Ω é uma janela centrada em p, f é um filtro espacial responsável por suavizar as diferenças de coordenas e g é um filtro de alcance responsável por suavizar a diferença de intensidades; ambos esses filtros costumam ser funções Gaussianas. k_p é um fator de normalização tipicamente dado pela soma dos pesos de $f \cdot g$ [43].

Posteriormente, foi-se introduzido um novo tipo de filtro bilateral chamado de *Joint Bilateral Filter* (filtro bilateral conjunto). A diferença é que ao invés de aplicar o filtro de alcance nas intensidades da imagem de entrada I, ele usa uma segunda imagem $\tilde{I}$ como forma de orientação. Isso por exemplo é útil quando tentamos combinar as altas frequências de uma gravura com as baixas frequências de outra. A equação desse filtro é dada por

$$J_p = \frac{1}{k_p} \sum_{q \in \Omega} I_q f(\|p - q\|) f\left(\left\|\tilde{I}_p - \tilde{I}_q\right\|\right), \tag{5.2}$$

em que a única diferença de (5.1) é que g usa as intensidades de $\tilde{I}$ ao invés de I.

Uma vez explicado ambos esses mecanismos, estamos preparados para entender o ferramental que é utilizado nessa última etapa do projeto. A subseção seguinte explica o funcionamento do filtro JBU (*Joint Bilateral Upsampling Filter*).

5.3 Filtro JBU

Em julho de 2007, dois pesquisadores da *Microsoft*, Matt Uyttendaele e Michel F. Cohen em conjunto com dois professores da Universidade de *Konstanz* e da Universidade Hebraica, Johannes Kopf e Dani Lischinski, respectivamente, propuseram na SIGGRAPH do ano um

novo tipo de filtro bilateral que combinava a ideia do *upsampling* com a ideia do filtro bilateral conjunto [43].

Eles demonstraram que a disponibilidade de uma imagem em alta resolução pode ser utilizada no contexto desse tipo de filtro na produção de uma solução adequada gerada a partir de uma solução de baixa qualidade. Isto é, a proposta deles é fazer uso de uma solução produzida para uma imagem de baixa resolução e interpolá-la afim de produzir uma solução para a gravura de alta resolução - exatamente o que precisamos.

Esse novo filtro, ao qual foi chamado de *Joint Bilateral Upsampling Filter* e que denotaremos daqui para frente como filtro JBU, apresenta grande importância para este trabalho. Nós o adaptamos afim de atender as necessidades do nosso problema de colorização e o incorporamos no algoritmo com o objetivo de obter resultados de forma ainda mais rápida.

A sua ideia consiste em utilizar um filtro espacial na solução de resolução baixa, enquanto que um filtro de alcance similar é aplicado ao mesmo tempo na imagem de resolução alta. Assim sendo, dado uma imagem de resolução alta $\ddot{I}$ de tamanho $n \times n$, e uma solução S gerada a partir da imagem de resolução baixa de tamanho $\bar{n} \times \bar{n}$, o filtro JBU computa a solução $\ddot{S}$ para a imagem original. Para isso, aplica-se um filtro espacial f à solução S, ao passo que um filtro de alcance g é aplicado à versão maior da imagem $\ddot{I}$. Assim, se p e q denotam coordenadas de pixels em $\ddot{I}$ e $p_{\downarrow}$ e $q_{\downarrow}$ coordenadas correspondentes na solução de resolução baixa, então a solução da imagem original (solução "redimensionada") de S é dada por

$$\ddot{S}_p = \frac{1}{k_p} \sum_{q_{\downarrow} \in \Omega} S_{q_{\downarrow}} f(\|p_{\downarrow} - q_{\downarrow}\|) g(\|\ddot{I}_p - I_{q_{\downarrow}}\|), \tag{5.3}$$

onde k_p é um fator de normalização (soma dos pesos de $f \cdot g$) e Ω é o suporte do filtro espacial f centrado em $p_{\downarrow}$ [43], isto é, uma vizinhança de pixels ao redor de $p_{\downarrow}$. Portanto, a complexidade desse filtro está diretamente relacionada com o número de pixels da imagem de alta resolução $\ddot{I}$.

Repare que a Equação (5.3) é praticamente idêntica a Equação (5.2), exceto que estamos construindo uma solução de resolução alta ao invés de uma imagem propriamente e operando em duas dimensões diferentes simultaneamente. A Equação (5.3) é a fornecida no artigo do filtro JBU em [43]. A seguir a incorporaremos ao nosso problema fazendo alterações de forma a simplificá-la.

5.4 Incorporação do Filtro JBU no Algoritmo de Colorização

Uma vez entendido como funciona o filtro JBU, temos que aplicá-lo ao nosso problema. Dentro do nosso contexto, as soluções a serem "redimensionadas" são os vetores U e V gerados a partir da resolução dos sistemas lineares

$$[(I-W)^T(I-W)+\lambda C]U=\lambda C\tilde{U}, \tag{5.4}$$

$$[(I-W)^T(I-W)+\lambda C]V=\lambda C\tilde{V}, \tag{5.5}$$

correspondentes a uma imagem de resolução baixa. Já os valores que $\ddot{I}$ assume são exatamente as intensidades Y da imagem de resolução alta. Dessa forma, estamos interessados em calcular

$$\ddot{U}_p=\frac{1}{k_p}\sum_{q_\downarrow\in\Omega}U_{q_\downarrow}f(\|p_\downarrow-q_\downarrow\|)g(\|Y_p-Y_{q_\downarrow}\|),$$

$$\ddot{V}_p=\frac{1}{k_p}\sum_{q_\downarrow\in\Omega}V_{q_\downarrow}f(\|p_\downarrow-q_\downarrow\|)g(\|Y_p-Y_{q_\downarrow}\|).$$

Os filtros espacial e de alcance utilizados foram Gaussianas muito semelhantes com as funções de afinidade empregadas nos capítulos anteriores. Assim, definimos f e g como

$$f(\|p_\downarrow-q_\downarrow\|)=e^{-\|p_\downarrow-q_\downarrow\|^2},$$

$$g(\|Y_p-Y_q\|)=e^{-\|Y_p-Y_{q_\downarrow}\|^2}.$$

Podemos reescrever então as Equações (5.4) e (5.5) como

$$\ddot{U}_p=\sum_{q_\downarrow\in\Omega}U_{q_\downarrow}\frac{e^{-\left(\|p_\downarrow-q_\downarrow\|^2+\|Y_p-Y_{q_\downarrow}\|^2\right)}}{k_p},$$

$$\ddot{V}_p=\sum_{q_\downarrow\in\Omega}V_{q_\downarrow}\frac{e^{-\left(\|p_\downarrow-q_\downarrow\|^2+\|Y_p-Y_{q_\downarrow}\|^2\right)}}{k_p}.$$

Chamando a fração da exponencial sobre o fator de normalização de $g_{pq_\downarrow}$, conseguimos sintetizar as fórmulas. Segue então que

$$\ddot{U}_p = \sum_{q_\downarrow \in \Omega} U_{q_\downarrow} g_{pq_\downarrow},$$

$$\ddot{V}_p = \sum_{q_\downarrow \in \Omega} V_{q_\downarrow} g_{pq_\downarrow}.$$

Essa notação significa mais um nível de simplificação uma vez que deixa claro determinadas constantes envolvidas na fórmula. Repare que $g_{pq_\downarrow}$ é uma função da posição e da intensidade do pixel - informações essas que temos armazenadas para qualquer imagem em tons de cinza. O que realmente muda na hora de aplicar o filtro JBU são as soluções U e V geradas a partir da resolução do sistema de baixa resolução (Equações 5.4 e 5.5).

Dito isso, podemos construir uma matriz G de tamanho $n \times n$ tal que cada elemento $G = \{g_{ij}\}_{i,j}^{n}$ é dado por

$$g_{ij} = \begin{cases} \dfrac{e^{-\left(\left\|x_{i_\downarrow} - x_j\right\|^2 + \|Y_i - Y_j\|^2\right)}}{k_p}, & se\ j \in \Omega \\ 0, & se\ j \notin \Omega, \end{cases}$$

em que $i_\downarrow$ é o pixel na imagem de resolução baixa correspondente ao pixel i da imagem de resolução alta e x_j significa a coordenada do pixel j na imagem de resolução baixa. Note que essa matriz também será altamente esparsa.

Além disso, se concatenarmos um vetor nulo 0 de tamanho $n - \bar{n}$ à solução U e V, geramos dois vetores $\bar{\bar{U}}$ e $\bar{\bar{V}}$ aumentados de tamanho n. Prontamente, podemos simplificar ainda mais a forma de obtenção dos vetores $\ddot{U}$ e $\ddot{V}$ efetuando uma simples multiplicação de estruturas esparsas

$$\ddot{U} = \bar{\bar{U}}^T G,$$

$$\ddot{V} = \bar{\bar{V}}^T G.$$

Esse último resultado se mostra importante uma vez que sugere uma maneira clara de resolução e traz benefícios em termos de implementação e computação no momento de calcular as soluções para o nosso problema. Reduzimos a incorporação do filtro a uma multiplicação de estruturas esparsas.

Entendido isso, o próximo passo é finalmente verificar os resultados e determinar o que de fato é menos custoso: aplicar o filtro JBU na solução proveniente do sistema associado à imagem de resolução baixa e produzir assim uma solução de resolução alta ou de fato resolver um sistema linear maior.

5.5 Filtro JBU x Resolução de um Sistema Linear Grande

Para dar início a etapa de testes, é importante lembrar dos principais passos do nosso algoritmo. Primeiramente vamos redimensionar a imagem rabiscada tornando-a suficientemente pequena, resolveremos os sistemas lineares correspondentes e então aplicaremos o filtro JBU nas soluções desses sistemas para gerar a solução de alta resolução (uma aproximação).

Dito isso, o primeiro teste foi realizado em uma imagem de tamanho 1024×1024 pixels a qual chamaremos de *Pepper*. Obtemos sua versão em escala de cinza, inserimos os rabiscos e geramos duas saídas: uma resolvendo o sistema linear da imagem diretamente (como fizemos durante todo o Capítulo 4) e outra utilizando essa nossa nova abordagem. Vale ressaltar que o tamanho utilizado na imagem de resolução baixa foi de 124×124 pixels. A Figura 5.2 mostra a imagem original, em escala de cinza e rabiscada. Já a Figura 5.3, mostra a saída para ambos os método de colorização.

Figura 5.2: Imagem de teste *Pepper*. Original, escalada de cinza e rabiscada.

Figura 5.3: *Pimentão*: saída para a implementação padrão (esquerda) e utilizando o filtro JBU (direita).

As diferenças entre essas duas imagens são poucas. A gravura gerada a partir da aplicação do filtro JBU aparenta ter as cores um pouco mais vibrantes, mas não há nenhuma característica notável. Os valores de PSNR (métrica que utilizamos para aferir qualidade) para essas duas imagens foram 21.4343 dB e 21.1199 dB que, por serem muito parecidas, comprovam esse fato.

Já quando falamos em termos de tempo de processamento, os resultados são bem díspares. Enquanto o algoritmo padrão de colorização gastou 10669 milissegundos, o algoritmo que incorpora o filtro JBU levou 1402 milissegundos.

O motivo para essa discrepância é óbvio e reflete toda nossa motivação de aprimorar e implementar tal mecanismo. Enquanto que no primeiro caso temos que resolver dois sistemas lineares (Equações 5.4 e 5.5) em que a matriz dos coeficientes tem tamanho 1024 × 1024, no segundo caso resolvemos os mesmos sistemas mas com matrizes de tamanho 128 × 128 seguido da aplicação do filtro JBU nas soluções geradas.

Como a complexidade envolvida na execução desse filtro está relacionada ao número total de pixels na gravura de alta resolução, resolvemos realizar testes com essa mesma imagem variando as suas dimensões e verificando de que forma o tempo gasto é dado. A Figura 5.4 mostra um comparativo do tempo total de execução para o nosso algoritmo padrão e o algoritmo que incorpora o filtro JBU para imagens de dimensões variadas.

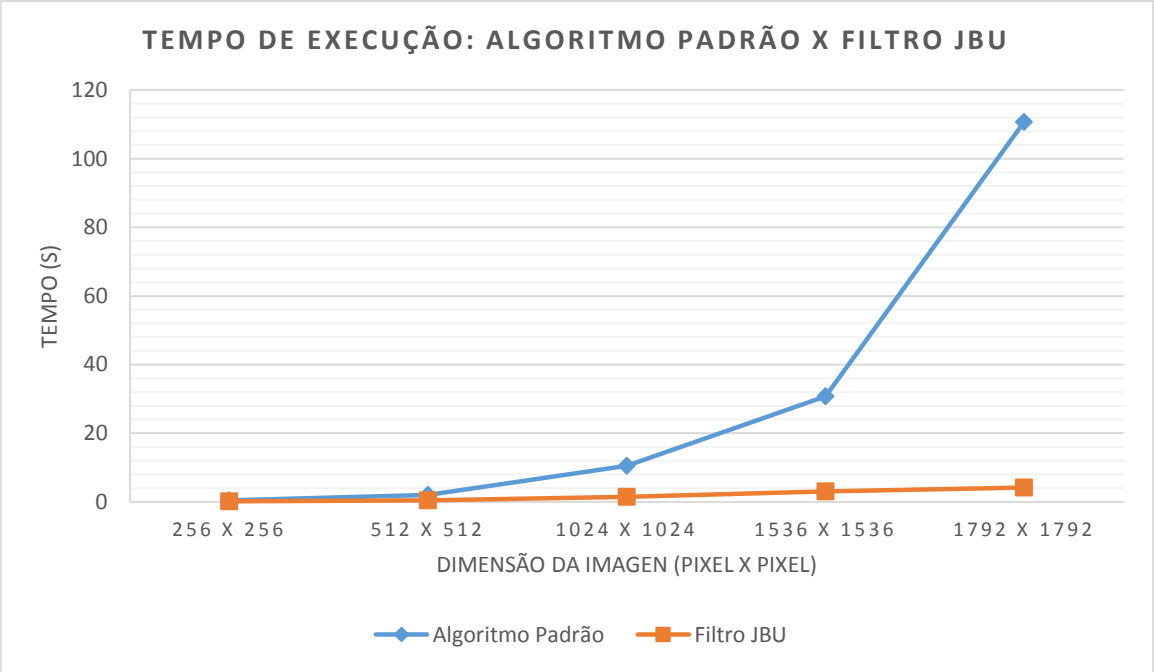


Figura 5.4: Tempo de execução para imagens de dimensões distintas.

Os ganhos de tempo são indiscutíveis. De fato, aplicar o filtro JBU é muito mais barato do que resolver os sistemas lineares correspondentes. Não somente, as saídas geradas foram qualitativamente muito semelhantes, o que afirma a sua eficiência.

A segunda imagem de teste é denominada *Nemo* e apresenta tamanho de 1024×768 pixels. Resolvemos redimensioná-la ainda mais produzindo a gravura de baixa resolução com tamanho de 64×48 pixels. A Figura 5.5 expõe sua versão original, em preto e branco e rabiscada, ao passo que a Figura 5.6 mostra as saídas para execução do algoritmo padrão e da aplicação do filtro JBU.

Figura 5.5: Imagem de teste *Nemo*. Original, escalada de cinza e rabiscada.

Figura 5.6: *Nemo*: saída para a implementação padrão (esquerda) e utilizando o filtro JBU (direita).

Dessa vez começamos discutindo o tempo de processamento. Enquanto que a solução gerada pelo algoritmo padrão consumiu 7504 milissegundos, a produzida pelo JBU executou em apenas 975 milissegundos.

No entanto, as saídas geradas nesse caso já não são tão similares qualitativamente. É possível reparar certas diferenças entre as duas imagens. Note que o resultado do filtro JBU apresenta um fundo mais borrado e que a colorização do peixe azul não foi tão fiel ao resultado gerado pelo algoritmo padrão. Os valores de PSNR dessas duas imagens foram 22.9745 dB e 19.1411 dB, respectivamente. De fato, essa diferença mostra numericamente que a primeira solução é melhor.

Isso sugere que reduzir de mais a imagem de alta resolução pode nos levar a resultados não tão satisfatórios. O que estamos fazendo de fato é gerar uma aproximação para a solução da imagem original por meio da solução produzida a partir dos sistemas lineares associados a uma imagem de resolução baixa. Portanto, quanto menos informação essa solução apresentar em relação a solução original, menos fiel ela será.

Diante disso, o terceiro teste faz essa análise. Escolhemos uma imagem de tamanho 1600×1200 pixels a qual chamaremos de *Praia*. Geramos soluções executando o filtro JBU em cima de imagens de baixa resolução correspondentes com tamanhos diferentes: 32×24, 64×48, 124×93, 256×192, 512×384 e 1024×768 pixels. Em seguida, produzimos a solução da imagem de resolução alta por meio do algoritmo de colorização padrão. Assim, a Figura 5.7 expõe essa gravura original, em tons de cinza , rabiscada e produzida pelo algoritmo

padrão ao passo que a Figura 5.8 mostra todas as seis saídas para as dimensões citadas naquela ordem.

Figura 5.7: Imagem de teste *Praia*. Original, escalada de cinza, rabiscada e produzida pelo algoritmo padrão.

Figura 5.8: *Praia*: saída gerada pelo filtro JBU para imagens de resoluções: 32 × 24, 64 × 48, 124 × 93, 256 × 192, 512 × 384 e 1024 × 768 pixels, indo da direita para a esquerda respectivamente.

Observando essas saídas, podemos notar que ganhos de qualidade são obtidos à medida que aumentamos as dimensões da imagem de resolução baixa. Mesmo assim, todos os resultados, exceto o primeiro (onde a redução da imagem original foi exagerada), são satisfatórios. Isso é melhor evidenciado pelos valores de PSNR. A Tabela 5.1 os apresenta.

Dimensão	32 × 24	64 × 48	124 × 93	256 × 192	512 × 384	1024 × 768	Original
PSNR (dB)	16.7976	23.9660	24.4702	25.7329	26.6527	26.7977	26.9317

Tabela 5.1: Valores de PSNR para saídas de implementações diferentes.

Falando sobre o tempo de processamento, esperamos também que ele aumente em função do tamanho das gravuras (de baixa resolução). Como já verificamos no Capítulo 4 e reiteramos na Figura 5.4, o crescimento do tempo que envolve fatorar a matriz dos coeficientes e resolver o sistema linear é exponencial. Assim, esperamos que, independentemente da aplicação do filtro JBU, o tempo de processamento para imagens maiores seja dominado por essas etapas. Assim sendo, a Tabela 5.2 mostra de que forma esse tempo é distribuído entre as principais fases da execução.

Etapas\Dimensões	64 × 48		124 × 93		256 × 192		512 × 384		1024 × 768	
Preencher tupla	0.05	2%	0.06	2%	0.09	4%	0.20	5%	0.63	6%
Converter tupla em matriz	0.00	0%	0.00	0%	0.00	0%	0.03	1%	0.15	1%
Calcular $A^T A$	0.00	0%	0.00	0%	0.02	1%	0.08	2%	0.35	3%
Calcular $A^T A + \lambda C$	0.00	0%	0.00	0%	0.00	0%	0.03	1%	0.13	1%
Executar fatoração	0.00	0%	0.05	2%	0.19	7%	1.09	27%	6.07	56%
Resolver sistema	0.00	0%	0.01	0%	0.05	2%	0.22	5%	1.12	10%
Calcular G	2.24	95%	2.24	93%	2.25	83%	2.23	57%	2.24	22%
Calcular $\ddot{U} = \bar{\bar{U}}^T G$ e $\ddot{V} = \bar{\bar{V}}^T G$	0.08	3%	0.08	3%	0.08	3%	0.08	2%	0.09	1%
Total (s)	2.357		2.453		2.693		4.078		10.988	

Tabela 5.2: Valores de PSNR para saídas de implementações diferentes.

Como vimos, o filtro JBU depende apenas do tamanho da imagem de alta resolução; nesse caso, 1600×1200 pixels. Por conta disso, o cálculo de G e a multiplicação que gera as soluções $\ddot{U}$ e $\ddot{V}$ apresentam tempo constante. O que realmente muda é o tempo gasto nas operações necessárias na construção e resolução dos sistemas lineares. Repare que quanto maior for o tamanho da imagem de resolução baixa, mais tempo será gasto nessas operações. Em contrapartida, quanto menor for o seu tamanho, o tempo total se concentra praticamente na aplicação do filtro JBU (as duas últimas etapas) - o que de fato é o que desejamos.

Evidentemente, aplicar o filtro JBU sobre uma imagem de baixa resolução que ainda é grande vai contra o propósito da ideia. Esse teste foi feito para mostrar como a fatia de tempo é distribuída pelas etapas do processo e aferir a qualidade dos resultados. Não apenas na imagem *Praia*, outros experimentos foram realizados e, constatamos que a gravura de resolução baixa com tamanho próximo a 128×128 pixels é uma opção adequada. Os sistemas lineares ficam suficientemente baratos (para as configurações da máquina), o tempo de processamento se

concentra na aplicação do filtro JBU (como observado na Tabela 5.2) e os resultados gerados são satisfatórios.

Não obstante, caso o usuário deseje melhores resultados, todo o processo de colorização pode ser feito aplicando-se o filtro e então ser finalizado executando o algoritmo de colorização padrão. Dessa forma, gastando muito menos tempo o usuário teria a certeza de que os rabiscos estão propriamente posicionados antes de executar a rotina mais dispendiosa. Nosso programa oferece essa facilidade assim como o tamanho do redimensionamento a ser utilizado na escolha da imagem de resolução baixa.

Esse último mecanismo sugere que o processo de colorização tenha sido feito de forma iterativa, ou seja, o usuário rabisca a imagem, verificava o resultado e então faz correções apagando ou inserindo traços. Discutimos a seguir como agregar esse conceito de forma prática e eficiente.

5.6 Colorização Iterativa e em Tempo Real

Colorir não é uma tarefa que se faz de uma vez só. Pensando no algoritmo de colorização, após inserir os rabiscos e executá-lo para obter uma saída, é bem provável que o usuário ainda não esteja satisfeito. Parte da imagem pode ter ficado borrada de uma cor errada ou outro tipo de resultado indesejado pode acontecer. Assim, é bem razoável esperar que o usuário retorne a etapa de rabiscos e posso ajeitar esses detalhes.

O nosso programa já fazia isso, ou seja, caso o usuário não ficasse satisfeito com o resultado gerado ele poderia retornar à imagem rabiscada e adicionar ou remover traços coloridos. Obviamente, isso significa resolver um novo sistema linear afim de produzir uma nova imagem colorida.

Combinando essa forma iterativa de colorir com a maneira pela qual adequamos o filtro JBU (criação da matriz G, em particular), nós implementamos um modo de colorização em tempo real para imagens ainda bem grandes. O funcionamento é simples. Vimos que o tamanho adequado para a imagem de resolução baixa é de aproximadamente 128×128 pixels. Da Tabela 5.2 verificamos que o tempo gasto nas operações associadas à resolução dos sistemas lineares consomem aproximadamente 100 milissegundos enquanto que, as etapas do filtro JBU (cálculo de G, $\ddot{U}$ e $\ddot{V}$) tomam por volta de 2 segundos do tempo. O ponto é que esses 2 segundos são dedicados essencialmente à construção da matriz G (onde os elementos dependem apenas

das intensidades e posições dos pixels). Sendo assim, G só precisa ser computada uma única vez.

Ainda na Tabela 5.2, verificamos que a geração de $\ddot{U}$ e de $\ddot{V}$ (etapa de multiplicação) é efetuada em aproximadamente 80 milissegundos. Portanto, se a matriz G for calculada na primeira tentativa de colorização, as próximas vezes vão levar pouco menos de 200 milissegundos. Isso para uma imagem de tamanho 1600×1200 pixels que é o caso da *Praia*. Resumindo, depois da primeira tentativa de colorização, todas as próximas tentativas serão executadas em poucas frações de segundo.

Evidentemente, quanto maior for a imagem original, mais tempo essa etapa de multiplicação vai tomar. Mas mesmo assim, pela forma como rearranjamos a fórmula do filtro JBU e fazendo uso de rotinas de multiplicação de estruturas esparsas da biblioteca CHOLMOD, conseguimos reduzir substancialmente o tempo gasto nessa fase. A tabela que segue mostra o tempo que as etapas de geração da matriz G e geração dos vetores $\ddot{U}$ e $\ddot{V}$ (multiplicação) consomem.

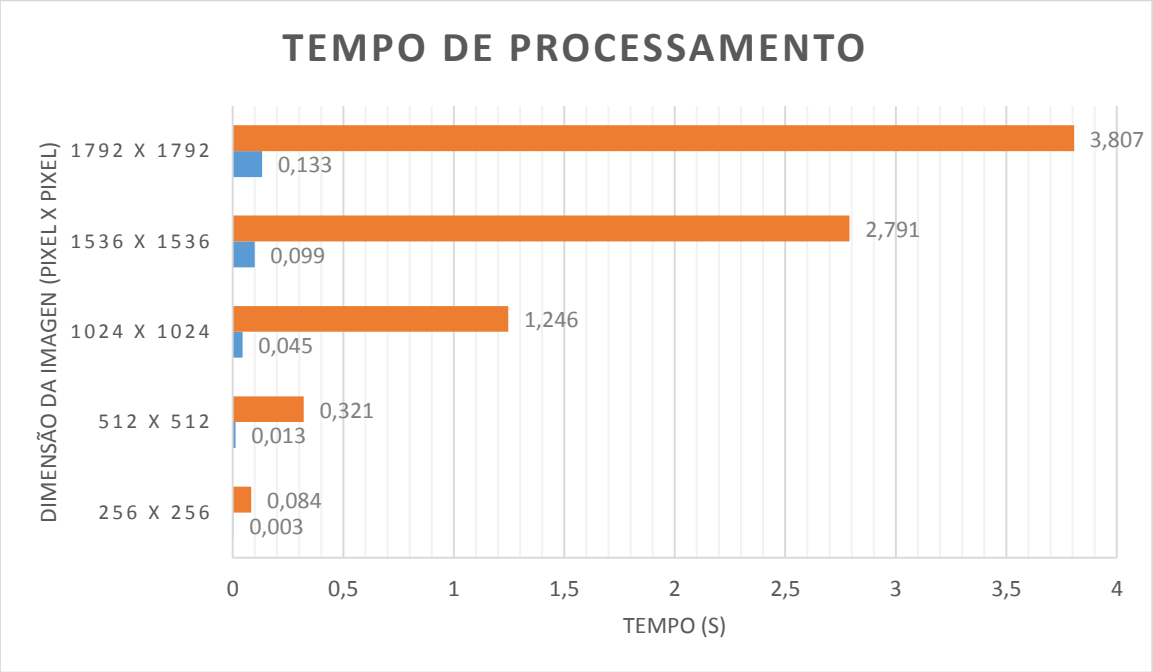


Tabela 5.3: Comparação entre as etapas de produção de G e os vetores $\ddot{U}$ e $\ddot{V}$.

Note que mesmo para imagens de tamanho 1792×1792 pixels, o tempo gasto na etapa de multiplicação é muito pequeno. Isso mostra que a nossa forma de utilizar o filtro, associada à construção de uma nova matriz e ao algoritmo de colorização via otimização torna a tarefa muito mais viável para imagens grandes. Assim, superamos o problema de ter que resolver

sistemas lineares exageradamente maiores sem perdas consideráveis de qualidade. Em outas palavras, tudo isso significa que o método de colorização se resume à resolução de um sistema de poucas dimensões e à multiplicação de estruturas esparsas - tarefas que são executadas de forma extremamente rápida.

Outro ponto que podemos levantar é o momento da geração da matriz G. Até aqui falamos que ela seria calculada na primeira tentativa de colorização. Obviamente, esse processo pode ser realizado no momento em que a imagem é carregada. Se esse for o caso, o tempo de colorização fica fixado em poucos milissegundos desde a primeira tentativa.

5.7 Colorização Interativa

Explicamos aqui um último adicional de implementação que faz uso de todo esse ferramental discutido. Usamos os números obtidos nos resultados anteriores para proporcionar um efeito de colorização interativa, isto é, um efeito capaz de colorir a imagem ao mesmo tempo que o usuário adiciona os rabiscos. Para exemplificar a explicação, vamos levar em conta que a imagem de resolução baixa tem tamanho próximo de 128×93 pixels e a imagem original tem tamanho de 1600×1200 pixels (como no teste da *Praia*).

A ideia é manter duas thread trabalhando: uma responsável pela UI da aplicação (capturando os rabiscos que são adicionados na imagem) e outra encarregada de resolver os sistemas lineares e aplicar o filtro JBU simultaneamente. Supondo que a matriz G já tenha sido calculada (no momento em que a imagem foi carregada, por exemplo), sabemos que - na máquina em questão - o tempo de processamento despendido para colorir a imagem será de aproximadamente 140 milissegundos (tempo de resolução dos sistemas somado ao tempo de multiplicação das estruturas esparsas). Isso nos dá por volta de sete colorizações por segundo; o suficiente para criar o efeito desejado.

Dessa forma, adicionamos à implementação final do programa uma opção de colorização interativa. Nela, à medida que o usuário adiciona rabiscos na imagem, a colorização vai sendo executada concorrentemente mostrando quase que de imediato o resultado. Vale mencionar que o efeito se mostra mais evidente quando o número de vezes que a máquina for capaz de executar o algoritmo de colorização por segundo for maior. Esse valor está associado às suas configurações assim como ao tamanho da imagem.

5.8 Considerações

Uma vez que o filtro JBU não foi alvo principal de estudo neste trabalho, melhorias ainda poderiam ser empregadas. Estudar mais a fundo os parâmetros empregados na Equação (5.3) é uma outra forma de análise mais aplicada. Mudanças nas funções de espaço e de alcance são sugestões válidas assim como nos próprios parâmetros da Gaussiana.

Mesmo assim, ficamos muito satisfeitos com todos os resultados. A incorporação do filtro JBU diante de nossas modificações aplicado ao algoritmo de colorização via otimização também aprimorado, rendeu excelentes efeitos: práticos e teóricos. Portanto, encerramos esta última etapa do trabalho e daremos seguimento com o capítulo de conclusão.

6. Conclusão

6.1 Objetivos da Pesquisa

O objetivo principal e inicial da pesquisa colocada no anteprojeto desta monografia foi fazer um estudo teórico e prático do algoritmo de colorização via otimização proposto por Anat Levin, Dani Lischinski e Yair Weiss em [11]. Nossa principal motivação foi a falta de respaldo teórico do método por eles apresentado combinado com a constante presença do mesmo em diversos artigos acadêmicos na forma de base ou referência. Não somente, um estudo mais aprofundado da técnica seria uma maneira sadia de alavancar novas alternativas de resolução e ideias associadas ao assunto - o que de fato ocorreu e contribui para enriquecer este trabalho.

6.2 Etapas de Desenvolvimento

O Capítulo 2 tratou de explicar como os autores aventaram o algoritmo impulsionando toda uma discussão sobre diversos assuntos de Computação Gráfica e otimização. Definimos conceitos como a intensidade de uma imagem, discutimos a importância e o significado dos espaços de cor no âmbito do processamento de imagens e finalmente promovemos um estudo a respeito do algoritmo de colorização. Durante seu desenvolvimento, apontamos detalhes que foram deixados de lado no artigo de origem [11] os quais mereciam sua devida atenção. À medida que fomos interpretando e modelando da nossa maneira o problema, fomos capazes de propor, assim, uma formulação final para o mesmo já diferente da apresentada pelos autores em [12].

No Capítulo 3 tomamos o cuidado de recapitular conceitos de otimização irrestrita e funções quadráticas em $\mathbb{R}^n$ com a intenção de tornar o trabalho mais completo e de fácil compreensão. Prontamente, iniciamos uma discussão intuitiva sobre o caráter da função objetivo do problema e prepararmos dessa forma as seções subsequentes. Usamos todo o ferramental matemático concedido no início do capítulo e desenvolvemos nossos próprios lemas e proposições. Por meio deles, determinamos restrições que foram capazes de tornar o sistema linear derivado do problema de otimização possível e determinado, onde a matriz dos coeficientes era simétrica e definida positiva. Dentro desse contexto, revisamos condições de otimalidade assim como fatoração matricial e então apontamos um método adequado de resolução para a nossa formulação. Ao fim, explicitamos a forma pela qual os autores do

algoritmo de colorização resolvem o problema e levantamos paralelos com a nossa abordagem já esperando que o nosso algoritmo de colorização modificado fosse mais eficiente.

O Capítulo 4 foi destinado a exposição dos resultados e fazer testes com todos os parâmetros que foram levantados durante o desenvolvimento. Começamos definindo os detalhes técnicos e de implementação do projeto. Apresentamos todas as bibliotecas, sub-rotinas e framework envolvidas na aplicação tomando o cuidado de expor os benefícios de cada uma. Em seguida variamos o parâmetro de penalidade, o tamanho do raio de vizinhança e a função de afinidade associada ao problema. Cada caso foi estudado cuidadosamente e exposto na forma de figuras, gráficos e tabelas. Para tal, aferimos a qualidade das imagens mediante o cálculo do valor de PSNR e verificamos o comportamento do tempo de processamento para todas essas situações. O capítulo é então encerrado confrontando a nossa forma de implementação com a utilizada pelos autores em [12] chegando à conclusão de que a nossa abordagem é de fato mais eficiente.

O Capítulo 5 vai além da proposta inicial do trabalho. A medida que os testes foram sendo realizadas, novas ideias surgiam. Este capítulo consolida uma dessas ideias mas sem se preocupar em fazer uma análise tão minuciosa assim como foi feito durante a construção dos três capítulos anteriores. Dito isso, a princípio explicamos toda a motivação por trás da nova sugestão. Explicamos de forma conveniente conceitos como filtros bilaterais e operações de redimensionamento. Incorporamos tais conceitos no nosso algoritmo já desenvolvido e os adaptamos de forma a tornar a resolução do problema de minimização muito menos custosa. Testes são realizados e experimentos são expostos onde resultados gratificantes foram observados. Concluímos o capítulo explicando mais um adicional que torna nossa aplicação mais interessante: uma forma simples de colorização interativa.

6.3 Trabalhos Futuros

Muitas ideias foram surgindo à medida que o trabalho foi sendo desenvolvido. Infelizmente, a maioria delas foram despertadas ao fim do capítulo de resultados, quando o tempo hábil necessário para desenvolvê-las já não era suficiente. Mesmo assim, conseguimos adicionar uma delas neste trabalho por meio do Capítulo 5. A seguir levantamos algumas ideias que tivemos que podem ser utilizadas em trabalhos futuros.

A sugestão inicial surge do processo de colorização iterativa. Assim como qualquer processo artístico, colorir é uma tarefa que se faz progressivamente. Dessa maneira, do Capítulo

2 verificamos que a geração de uma imagem colorida está diretamente relacionada a uma atualização na matriz dos coeficientes do sistema linear associado. Essa atualização se dá por meio da soma de uma matriz simétrica do tipo $A^T A$ por uma matriz extremamente simples que chamamos de λC. A sugestão então é tentar estudar uma forma de atualizar a fatoração de Cholesky (etapa mais dispendiosa do algoritmo) uma vez que o processo envolvido entre uma resolução do sistema e outra, é a adição ou subtração dessa segunda matriz simplória. Alguns testes foram realizados utilizando-se as rotinas da biblioteca CHOLMOD afim de alcançar tal objetivo, mas os resultados não foram satisfatórios; efetuar a fatoração uma segunda vez ainda era mais barato do que fazer uma atualização. Mesmo assim, tal sugestão pode ser melhor preparada.

A segunda proposta é a utilização de processamento paralelo de modo a diminuir o tempo total de execução do programa. Muitas bibliotecas dedicadas à resolução de sistemas lineares esparsos com suporte de GPU estão disponíveis livremente na WEB. Isso é algo importante em face da dificuldade de se lidar com imagens de tamanhos exageradamente grandes.

Uma terceira possibilidade é a utilização de um espaço de cor alternativo. Ao invés de YUV, deixamos como sugestão o espaço de cor HSL (*Hue, Saturation and Lightness*). Seria interessante verificar como a matiz (*Hue*) e a saturação (*Saturation*) se propagariam mediante uma função objetivo semelhante à utilizada neste trabalho.

A quarta alternativa é uma ideia simples mas promissora. A sugestão é, ao invés de desenvolver um algoritmo de colorização via otimização, desenvolver um algoritmo de recolorização via otimização. Neste trabalho utilizamos as intensidades como forma de ponderar os pixels adjacentes e assim propagar as cores. Se o interesse fosse recolorir uma imagem, poderíamos adaptar as mesmas funções de afinidade de modo a incorporar as informações de crominância no cálculo dos pesos. Isto é, ao recolorir uma maçã de verde, a propagação seria mais eficaz uma vez que os pesos além das intensidades, conhecem também as componentes de cor desse objeto (as tonalidades de vermelho da maçã original).

Por fim, gostaríamos de mencionar que melhorias no nosso algoritmo de colorização interativa podem ser efetuadas. Como já dito, a falta de tempo hábil nos impediu de melhorá-lo. Acreditamos que uma análise mais profunda do filtro JBU além da realização de testes com outras funções de espaço e de alcance podem render benefícios.

6.4 Dificuldades

A principal dificuldade deste trabalho foi a falta de apoio teórico do mesmo. O artigo se limitava ao escopo de fornecer a função objetivo juntamente com poucas explicações da mesma. Além disso, tentamos também fazer contato com os autores afim de tirar dúvidas mas sem sucesso.

Outra dificuldade se deu a nível de implementação. Diversas bibliotecas de resolução de sistemas lineares esparsos foram testadas na busca de obter melhores resultados. Essa tarefa se torna desgastante uma vez que é necessário compilar e instalar todas as sub-rotinas necessárias, fazer testes de funcionamento e então aprender a utilizá-las estudando a documentação que, infelizmente, nem sempre é tão amigável. Finalmente, depois de inúmeras tentativas, chegamos à conclusão de que CHOLMOD é uma excelente ferramenta computacional nesse sentido.

6.5 Considerações Finais

O projeto incorpora assuntos abordados em diversas disciplinas do curso como forma de fundamentação em seu desenvolvimento. Elementos oriundos principalmente de Álgebra Linear, Computação Gráfica e Programação Não Linear foram revisados, compreendidos e explicitados de forma simples e didática na medida do necessário. Dessa forma, tentamos neste trabalho mostrar o devido valor de tais conteúdos; os quais merecem tanta atenção quanto os detalhes de implementação.

Assim, a maior satisfação que encontramos durante o desenvolvimento foi a forte contribuição que ele nos deu no sentido do amadurecimento no âmbito da pesquisa e na consolidação de vários conteúdos visto durante todos os anos do curso. Espero que este trabalho sirva como referência teórica para alunos que se interessem pelo algoritmo de colorização (ou assuntos do gênero) assim como um objeto capaz de despertar entusiasmo para a criação de novas abordagens.

Bibliografia

[1] MARKLE, W. The development and application of colorization, SMPTE

Journal (1984) 632–635.

[2] Dictionary.com. Palavra "Monochromatic". Unabridged. Random House, Inc. Último acesso Novembro, 2013.

[3] COOPER, R. Colorization and moral rights, Journalism Quarterly (Urbana,Illinois) 68 (1990) 465–473.

[4] LEIBOWITZ, F. Movie colorization and the expression of mood, Journal of Aesthetics and Art Criticism (Cleveland, Ohio) 49 (4) (1991) 363–364.

[5] KUMAR; SINGH. S.; DEEPAK; "Colorization of Gray Scale Images in YCbCr Color Space Using Texture Extraction and Luminance Mapping", IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 2278-0661 Volume 4, Issue 5 (Sep.-Oct. 2012), PP 27-32.

[6] JACOB, G.; GUPTA, S. "Colorization of Grayscale and Videos using a Semiautomatic Approach", IEEE International Conference on Image Processing (ICIP) Novembro, 2009.

[7] HERTZMANN, A.; JACOBS, C.; OLIVER, N.; CURLESS, B.; SALESIN, D. "Image Analogies", SIGGRATH 2001, Conference Proceedings.

[8] GUAN, J.; GUOPING, Q. "Interactive Image Segmentation using Optimization with Statistical Priors", International Workshop on The Representation and Use of Prior Knowledge in Vision (In conjunction with ECCV 2006).

[9] CHARPIAT, G.; HOFMANN, M.; SCHOLKOPF, B. "Automatic Image Colorization via Multimodal Predictions", European Conference on Computer Vision ECCV, 2008.

[10] HORIUCHI, T. "Colorization algorithm for gray-level image by probabilistic relxation", Proceedings of IEEE International Conference on Pattern Recognition, pp. 867-870, Agosto, 2003.

[11] LEVIN, A.; LISCHINSKI, D.; WEISS, Y. "Colorization using Optimization". ACM Trans. Graph. 23, 3 (Agosto), 2004.

[12] LEVIN, A.; LISCHINSKI, D.; WEISS, Y. Colorization using Optimization. http://www.cs.huji.ac.il/~yweiss/Colorization/. Agosto, 2004.

[13] NOCEDAL, J.; WRIGHT, S. J. Numerical optimization. 2. ed. New York: Springer New York, 2006.

[14] JOHNSON, S. Stephen Johnson on Digital Photography. O'Reilly. ISBN 0-596-52370-X, 2006.

[15] YOSHI, O. “CIE Fundamentals for Color Measurements”. IS&T NIP16 Intl. Conf. on Digital Printing Technologies, pp. 540-45. 18 de Julho, 2009.

[16] KUEHNI, R. Color Space and Its Divisions: Color Order from Antiquity to the Present ISBN 0-471-32670-4, 2003.

[17] FORD, A; ROBERTS, A. Colour space conversions. Technical report, Westminster University, London,August, 1998.

[18] TKALCIC, M; TASIC, J. Colour spaces: perceptual, historical and applicational background. EUROCON 2003. Ljubljana, Slovenia, pp. 304–308. 2003.

[19] JACK, K. “Video Demystified: A Handbook for the Digital Engineer”, 5th Edition, Newnes, 2007. Capítulo 3, página 15.

[20] DOUGLAS, A. Kerr Issue 5. Chromaticity and Chrominance in Color Definition, 3 Outubro, 2010.

[21] PALUS, H. Colour spaces, Chapmann and Hall, 1998. Página 72.

[22] FORD, A; ROBERTS, A. Colour space conversions (Revised), Westminster University, London, 1998.

[23] GROSSMAN, J.; GROSSMAN, M.; KATZ, R. The First Systems of Weghted Diffrential and Integral Calculus, ISBN 0-9771170-1-4, 1980.

[24] SHI, J.; MALIK, J. Normalized cuts and image segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 731–737. 1997.

[25] JENSEN, A.; BARD, F. Operations Research Models and Methods. https://www.me.utexas.edu/~jensen/ORMM/supplements/units/nlp_methods/const_opt.pdf. 2003.

[26] SNYMAN, A. Practical Mathematical Optimization. Volume 97. Springer Science and Business Media, Inc., 2005.

[27] SIMON; BLUME. Mathematics For Economists, 1994.

[28] MEYER, D. Matrix Analysis and Applied Linear Algebra. Siam, 19 de Abril, 2004.

[29] MARTINEZ, J.; SANTOS, S. Métodos Computacionais de Otimização. Dezembro, 1995.

[30] LIMA, E. L. Álgebra Linear. Rio de Janeiro: IMPA, 2006.

[31] TIMOTHY, A.; HAGER, W. CHOLMOD User’sGuide, version 2.1.2, 15 de Abril, 2013.

[32] DONGARRA, J.; Basic Linear Algebra Subprogramas Technical Forum Standard, International Journal of High Performance Applications and Supercomputing, 16 de Janeiro, 2002, pp. 1-111, and International Journal of High Performance Applications and Supercomputing, 16 de Fevereiro, 2002, pp. 115-199.

[33] GEORGE, A.; LIU, J. Computer Solution of Large Sparse Positive Denite Systems, Prentice Hall, Englewood Clis, NJ, 1981.

[34] ANDERSON, E.; BAI, Z.; BISCHOF, C.; BLACKFORD, S.; DEMMEL, J.; DONGARRA, J.; DU CROZ, J.; GREENBAUM, A.; HAMMARLING, S.; MACKENEEY, A.; SORENSEN. D. LAPACK User's Guide, 3th ed. Society for Industrial and Applied Mathematic, Philadelphia, PA, 1999.

[35] MARKOFF, J. "Writing the fastest code, by hand, for fun". New York Times, 28 de Novembro, 2005.

[36] GotoBLAS 2. https://www.tacc.utexas.edu/tacc-projects/gotoblas2. Último acesso Março, 2014.

[37] The Man Behind Sandy Bridge. http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/84. Último acesso Março, 2014.

[38] Qt Digia. http://doc.qt.digia.com/. Último acesso Março, 2014.

[39] TIMOTHY, A. UMFPACK User's Guide, version 5.6.2, 25 de Abril, 2013.

[40] HUYNH-THU, Q.; GHANBARI, M. "Scope of validity of PSNR in image/video quality assessment". Electronics Letters 44 (13): 800. doi:10.1049/el: 20080522, 2008.

[41] Qt-apps. http://qt-apps.org/content/show.php/EasyPaint?content=140877. Último acesso Março, 2014.

[42] DEVARAJAN, H.; NYIKAL, H. Image Scaling and Bilateral Filtering. http://scien.stanford.edu/pages/labsite/2006/psych221/projects/06/imagescaling/bilati.html. 2006.

[43] KOPF, J.; COHEN, M.; LISCHINSKI, D.; UYTTENDAELE, M.; "Joint Bilateral Upsampling", Proc. Of the SIGGRAPH conf. ACM Trans. On Graphics, 26, 3 (Julho), 2007.

[44] YOUSSEF, A.; "Image Downsampling and Upsampling Methods", International Conference on Imaging, Science, Systems, and Technology CISST '99, Las Vegas, pp. 132-138, Junho, 1999.