UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DEPARTAMENTO DE COMPUTAÇÃO

"MÉTODO PARA COMPRESSÃO DE IMAGENS DIGITAIS FUNDAMENTADO EM PROCEDIMENTOS DE HUFFMAN E WAVELETS"

Luciana Aparecida de Oliveira Betetto

SÃO CARLOS – SP

2005

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

B565mc

Betetto, Luciana Aparecida de Oliveira.
Método para compressão de imagens digitais fundamentado em procedimentos de Huffman e Wavelets / Luciana Aparecida de Oliveira Betetto. -- São Carlos : UFSCar, 2006.
163 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2006.

1. Processamento de imagens. 2. Compressão de imagens. 3. Descompressão de imagens. 4. Wavelets (Matemática). I. Título.

CDD: 006.42 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Método para Compressão de Imagens Digitais Fundamentado em Procedimentos de Huffman e Wavelets"

LUCIANA APARECIDA DE OLIVEIRA BETETTO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Paulo Estevão Cruvinel
(Orientador – CNPDIA/EMBRAPA)

Prof. Dr. José Hiroki Saito
(DC/UFSCar)

Prof. Dr. Joaquim Teixeira de Assis
(UERJ)

São Carlos
Agosto/2005

Mas a vida é curta e a informação infinita...
Abreviações são um mal necessário e a tarefa do abreviador
é fazer o melhor trabalho que, embora intrinsecamente ruim,
seja ainda melhor que nada.

Aldous Huxley

Ao meu marido Ronaldo e
ao meu filho André Luiz,
que ainda tão pequeno e
frágil deu-me força para a
conclusão deste trabalho.

À minha mãe Elvira e
minha irmã Lúcia pela
eterna dedicação.

AGRADECIMENTOS

Agradeço a Deus, que sempre abre um caminho onde não existe caminho.

Ao Prof. Dr. Paulo Estevão Cruvinel pelo trabalho de orientação realizado, além de mostrar-se um grande amigo nas horas mais difíceis desta caminhada, nunca medindo esforços para auxiliar-me. Sua compreensão e dedicação foram os alicerces que me sustentaram.

Aos meus amigos do curso de Pós-Graduação com os quais pude compartilhar dificuldades e comemorar vitórias.

Ao amigo Maurício Fernando de Lima Pereira pelo auxílio e incentivo no término deste trabalho.

Aos professores do Programa de Pós-Graduação do Departamento de Computação, com os quais tanto tive a oportunidade de aprender.

Aos funcionários do Departamento de Computação pela presteza e simpatia.

Enfim, a todos que auxiliaram no desenvolvimento deste trabalho.

SUMÁRIO

DEDICATÓRIA
AGRADECIMENTOS
SUMÁRIO
LISTA DE FIGURAS
LISTA DE TABELAS
RESUMO
ABSTRACT
PREFÁCIO
CAPÍTULO 1: PROCESSAMENTO DE IMAGENS DIGITAIS E SUA COMPRESSÃO **..... 1**
1.1 *PROCESSAMENTO DE IMAGENS DIGITAIS* 1
1.2 *FUNDAMENTOS DO PROCESSAMENTO DE IMAGENS DIGITAIS* 2
1.3 *CONSIDERAÇÕES INICIAIS* 5
1.4 *FUNDAMENTOS DA COMPRESSÃO DE IMAGEM DIGITAL* 7
1.4.1 *Redundância de Codificação* 15
1.4.2 *Redundância Interpixel* 17
1.4.3 *Redundância Psicovisual* 18
1.4.4 *Critérios de Fidelidade* 18
1.5 *MODELOS DE COMPRESSÃO DE IMAGENS* 19
1.5.1 *Codificadores e Decodificadores Tipo Fonte* 20
1.5.2 *Codificadores e Decodificadores Tipo Canal* 21
1.6 *COMPRESSÃO SEM PERDAS* 22
1.6.1 *Codificação por Tamanho Variável* 24
1.6.2 *Codificação por Planos de Bits* 31
1.6.3 *Codificação Previsora sem Perdas* 32
1.7 *COMPRESSÃO COM PERDAS* 32
1.7.1 *Codificação Previsora com Perdas* 33
1.7.2 *Codificação por Transformada* 33
1.8 *CLASSIFICAÇÃO DOS ALGORITMOS DE COMPRESSÃO* 34
1.8.1 *Compressão por Transformação do Modelo* 34
1.8.2 *Compressão por Discretização* 36
1.8.3 *Compressão por Codificação* 36
1.9 *OUTROS PADRÕES DE COMPRESSÃO* 38

CAPÍTULO 2: TRANSFORMADAS WAVELETS E ESPECTRO DE WIENER NO PROCESSAMENTO DE IMAGEM DIGITAL 42

2.1 *FUNDAMENTOS* 42

2.2 *TRANSFORMADA DISCRETA DE WAVELETS* 49

2.3 *A WAVELET DE HAAR* 49

2.4 *ANÁLISE MULTIRESOLUÇÃO* 50

2.5 *DECOMPOSIÇÃO UNIDIMENSIONAL USANDO TRANSFORMADA WAVELET HAAR* *56*

2.6 *COMPRESSÃO DE IMAGENS DIGITAIS UTILIZANDO A TRANSFORMADA WAVELET* 58

2.7 *O ESPECTRO DE WIENER* 65

CAPÍTULO 3: MÉTODO PARA COMPRESSÃO DE IMAGENS DIGITAIS FUNDAMENTADO EM PROCEDIMENTOS DE HUFFMAN E WAVELETS 68

3.1 *CARACTERÍSTICAS DO SISTEMA* *68*

3.2 *COMPRESSÃO E DESCOMPRESSÃO UTILIZANDO HUFFMAN* *71*

3.3 *DECOMPOSIÇÃO DA IMAGEM UTILIZANDO A TRANSFORMADA WAVELET HAAR* *73*

CAPÍTULO 4: RESULTADOS, DISCUSSÕES E CONCLUSÕES 77

4.1 *RESULTADOS E DISCUSSÕES* *77*

4.2 *SÍNTESE SOBRE OS RESULTADOS OBTIDOS NOS ESTUDOS DE CASOS CONSIDERADOS* *127*

4.3 *CONCLUSÕES* *130*

4.4 *SUGESTÕES DE TRABALHOS FUTUROS* *131*

APÊNDICE A 132

APÊNDICE B 133

APÊNDICE C 134

REFERÊNCIAS BIBLIOGRÁFICAS 159

LISTA DE FIGURAS

FIGURA 1.1 - (a) Imagem contínua, (b) Uma linha escaneada de A para B na imagem contínua, (c) Amostragem e Quantização, (d) Linha digital escaneada 4

FIGURA 1.2 - Diagrama de um sistema de comunicação geral. 8

FIGURA 1.3 - Diagrama de uma codificação fonte. 9

FIGURA 1.4 - Técnicas de compressão de imagem digital agrupadas em duas categorias: métodos sem perda e com perda de informações 11

FIGURA 1.5 - Diagrama para operações de compressão e descompressão de imagem. 12

FIGURA 1.6 - (a)Caminho da varredura da maior fileira enrolado em espiral, (b)Caminho da varredura diagonal enrolado em espiral, (c)Caminho da varredura em espiral, (d)Caminho da varredura em desenho. 13

FIGURA 1.7 - Modelo de sistema de compressão genérico 20

FIGURA 1.8 - Um sistema típico de compressão 22

FIGURA 1.9 - Construção da árvore do Código de Huffman 30

FIGURA 1.10 - Árvore do Código de Huffman arranjada 30

FIGURA 1.11 - Compressão por mudança de representação de uma imagem 35

FIGURA 2.1 - *Wavelet* mãe de Morlet 47

FIGURA 2.2 - Uma *Wave* (sinusóide) e uma *Wavelet* 47

FIGURA 2.3 - A *Wavelet* de Haar, $\psi(x)$ 50

FIGURA 2.4 - Codificação baseado na decomposição usando o modelo *Wavelet*. As taxas de compressão são: (a) 57:1, (b) 59:1 e (c) 49:1 59

FIGURA 2.5 - Codificação baseado na decomposição usando o modelo *Wavelet*. As taxas de compressão são: (a) 118:1, (b) 87:1 e (c) 84:1 59

FIGURA 2.6 - Decomposição Padrão de uma imagem utilizando *Wavelets* 63

FIGURA 2.7 - Decomposição Não-Padrão de uma imagem utilizando *Wavelets* 64

FIGURA 3.1 - Diagrama geral do Sistema desenvolvido 69

FIGURA 3.2 - Pseudocódigo para busca, compressão, descompressão e análise de qualidade de imagens digitais 70

FIGURA 3.3 - Diagrama do Módulo de Compressão e Descompressão utilizando Huffman. 71

FIGURA 3.4 - Pseudocódigo para a implementação do Módulo de Compressão e Descompressão utilizando Huffman..... 72

FIGURA 3.5 - Diagrama do Módulo de Decomposição utilizando *Wavelet* Haar 73

FIGURA 3.6 - Pseudocódigo para a implementação do Módulo de Decomposição utilizando a *Wavelet* Haar 74

FIGURA 3.7 - Imagem Lena..... 75

FIGURA 3.8 - Imagem Girl 75

FIGURA 3.9 - Corte transversal de uma amostra de solo reconstruído pelo método de retroprojeção filtrada 76

FIGURA 3.10 - Phantom homogêneo de nylon reconstruído por retroprojeção filtrada..... 76

FIGURA 4.1 - Lena128x128.bmp..... 77

FIGURA 4.2 – Lena128x128: Espectro de Wiener resultante da imagem original..... 79

FIGURA 4.3 – Lena128x128: Resultado da imagem descompressa após a codificação Huffman..... 79

FIGURA 4.4 – Lena128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 80

FIGURA 4.5 – Lena128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 80

FIGURA 4.6 – Lena128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 81

FIGURA 4.7 – Lena256x256.bmp 82

FIGURA 4.8 – Lena256x256: Espectro de Wiener resultante da imagem original..... 84

FIGURA 4.9 – Lena256x256: Resultado da imagem descompressa após a codificação Huffman..... 84

FIGURA 4.10 – Lena256x256: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 85

FIGURA 4.11 – Lena256x256: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 85

FIGURA 4.12 – Lena256x256: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 86

FIGURA 4.13 – Lena512x512.bmp 87

FIGURA 4.14 – Lena512x512: Espectro de Wiener resultante da imagem original 89

FIGURA 4.15 – Lena512x512: Resultado da imagem descompressa após a codificação Huffman 89

FIGURA 4.16 – Lena512x512: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 90

FIGURA 4.17 – Lena512x512: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 90

FIGURA 4.18 – Lena512x512: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 91

FIGURA 4.19 – Lena1024x1024.bmp 92

FIGURA 4.20 – Lena1024x1024: Espectro de Wiener resultante da imagem original 94

FIGURA 4.21 – Lena1024x1024: Resultado da imagem descompressa após a codificação Huffman 94

FIGURA 4.22 – Lena1024x1024: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 95

FIGURA 4.23 – Lena1024x1024: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 95

FIGURA 4.24 – Lena1024x1024: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 96

FIGURA 4.25 - Girl128x128.bmp 97

FIGURA 4.26 – Girl128x128: Espectro de Wiener resultante da imagem original 99

FIGURA 4.27 – Girl128x128: Resultado da imagem descompressa após a codificação Huffman 99

FIGURA 4.28 – Girl128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 100

FIGURA 4.29 – Girl128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 100

FIGURA 4.30 – Girl128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 101

FIGURA 4.31 – Girl256x256.bmp 102

FIGURA 4.32 – Girl256x256: Espectro de Wiener resultante da imagem original 104

FIGURA 4.33 – Girl256x256: Resultado da imagem descompressa após a codificação Huffman 104

FIGURA 4.34 – Girl256x256: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 105

FIGURA 4.35 – Girl256x256: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 105

FIGURA 4.36 – Girl256x256: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 106

FIGURA 4.37 – Girl512x512.bmp 107

FIGURA 4.38 – Girl512x512: Espectro de Wiener resultante da imagem original 109

FIGURA 4.39 – Girl512x512: Resultado da imagem descompressa após a codificação Huffman 109

FIGURA 4.40 – Girl512x512: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 110

FIGURA 4.41 – Girl512x512: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 110

FIGURA 4.42 – Girl512x512: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 111

FIGURA 4.43 – Girl1024x1024.bmp 112

FIGURA 4.44 – Girl1024x1024: Espectro de Wiener resultante da imagem original..... 114

FIGURA 4.45 – Girl1024x1024: Resultado da imagem descompressa após a codificação Huffman..... 114

FIGURA 4.46 – Girl1024x1024: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 115

FIGURA 4.47 – Girl1024x1024: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 115

FIGURA 4.48 – Girl1024x1024: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 116

FIGURA 4.49 - Solo128x128.bmp 117

FIGURA 4.50 – Solo128x128: Espectro de Wiener resultante da imagem original..... 119

FIGURA 4.51 – Solo128x128: Resultado da imagem descompressa após a codificação Huffman..... 119

FIGURA 4.52 – Solo128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman..... 120

FIGURA 4.53 – Solo128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 120

FIGURA 4.54 – Solo128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 121

FIGURA 4.55 - Phantom128x128.bmp..... 122

FIGURA 4.56 – Phantom128x128: Espectro de Wiener resultante da imagem original..... 124

FIGURA 4.57 – Phantom128x128: Resultado da imagem descompressa após a codificação Huffman..... 124

FIGURA 4.58 – Phantom128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman 125

FIGURA 4.59 – Phantom128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 125

FIGURA 4.60 – Phantom128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman 126

FIGURA 4.61 - Resultados das taxas de compressão. 128

FIGURA 4.62 - Resultados dos coeficientes de Wiener. 129

LISTA DE TABELAS

TABELA 1.1 - Escala de notas da *"Television Allocations Study Organization"* 19

TABELA 2.1 - Processo de decomposição por *Wavelets* em várias resoluções 57

TABELA 4.1 - Lena128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 78

TABELA 4.2 - Lena128x128: Resultados obtidos quanto à análise da qualidade das imagens. 81

TABELA 4.3 - Lena256x256: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 83

TABELA 4.4 - Lena256x256: Resultados obtidos quanto à análise da qualidade das imagens. 86

TABELA 4.5 - Lena512x512: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 88

TABELA 4.6 - Lena512x512: Resultados obtidos quanto à análise da qualidade das imagens. 91

TABELA 4.7 – Lena1024x1024: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 93

TABELA 4.8 – Lena1024x1024: Resultados obtidos quanto à análise da qualidade das imagens. 96

TABELA 4.9 - Girl128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 98

TABELA 4.10 - Girl128x128: Resultados obtidos quanto à análise da qualidade das imagens. 101

TABELA 4.11 - Girl256x256: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 103

TABELA 4.12 - Girl256x256: Resultados obtidos quanto à análise da qualidade das imagens. 106

TABELA 4.13 - Girl512x512: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 108

TABELA 4.14 - Girl512x512: Resultados obtidos quanto à análise da qualidade das imagens. 111

TABELA 4.15 – Girl1024x1024: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 113

TABELA 4.16 – Girl1024x1024: Resultados obtidos quanto à análise da qualidade das imagens. 116

TABELA 4.17 - Solo128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 118

TABELA 4.18 - Solo128x128: Resultados obtidos quanto à análise da qualidade das imagens. 121

TABELA 4.19 - Phantom128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman. 123

TABELA 4.20 - Phantom128x128: Resultados obtidos quanto à análise da qualidade das imagens. 126

TABELA 4.21 - Resultados das taxas de compressão dos métodos utilizados. 127

TABELA 4.22 - Resultados dos coeficientes de Wiener dos métodos utilizados. 128

TABELA 4.23 - Resultados dos tempos utilizados para a compressão.. 130

RESUMO

A quantidade de informações armazenadas, transmitidas e manuseadas por computadores tem crescido exponencialmente nas últimas décadas. Dois fatores têm contribuído para este efeito, um deles é o surgimento dos sistemas de multimídia juntamente com suas numerosas aplicações. O outro é quanto ao fato de que atualmente computadores não operam apenas textos e números e sim sons, imagens, filmes e realidade virtual. Junto a este cenário de novas realidades também se encontra o desenvolvimento e a crescente disponibilidade da internet (*World Wide Web)*, um sistema interativo e de multimídia baseado na informação, a qual fez com que informações estivessem à disposição de uma quantidade enorme de usuários, com interatividade acentuada. Este trabalho apresenta o desenvolvimento de um sistema para Compressão de Imagens Digitais com base nos procedimentos de Huffman e *Wavelets* para armazenamento e transmissão de dados em ambientes que utilizam processamento digital de imagens. Foi desenvolvido na linguagem C++ sob o ambiente Windows®. Para validação dos resultados foram utilizados padrões de imagens clássicas do Processamento Digital de Imagens, como a imagem *Lena* e a imagem *Girl*, ambas nas resoluções 128x128, 256x256 e 512x512 pixels, bem como imagens tomográficas de amostras de solo e madeira em resolução 128x128 pixels, obtidas a partir do uso do minitomógrafo dedicado à agricultura do Centro de Instrumentação da Empresa Brasileira de Pesquisa Agropecuária.

Abstract

The amount of transmitted, stored and accessed information by computers have been growing exponentially on the last decades. Two factors have contributed to this effect, one is the appearing of the multimedia systems with its variety of applications. The other is all that fact that nowadays computers not only operate texts and numbers but sounds, images, movies and virtual reality. Inside this scenario of new reality, we can also find, the development and the growth of the internet availability (*World Wide Web)*, an interactive and multimedia system based on information, making this information available for a great quantity of users, with stressed interactivity. This work shows the development of a system for digital images Compression with base on Huffman and *Wavelets* procedures of data storage and transmission in environments using digital images processing. It was developed on the C++ language under Windows® environment. For results validation there were used standard images patterns of digital image Processing, as the *Lena* and *Girl* image, both on 128x128, 256x256 and 512x512 pixels resolutions, as well as tomographics images of soil and wood sample from tomograph of the Brazilian Agricultural Research Corporation.

Prefácio

O processamento de imagens digitais teve início em 1920, quando figuras foram transmitidas através de um cabo submarino entre Londres e Nova Iorque, reduzindo o tempo de transporte de mais de uma semana para menos de três horas (Gonzales & Woods, 2002). Desde então, as técnicas de processamento de imagens digitais estão sendo aprimoradas e ganharam nos últimos anos um grande impulso com o desenvolvimento de técnicas de processamento de sinais. A utilização de imagens digitais nas mais diversas áreas tem sido crescente, tornando imperativo o desenvolvimento de técnicas de compressão que viabilizem o armazenamento e a sua transmissão em redes de computadores. Neste contexto, encontra-se a necessidade de uma avaliação cada vez mais rigorosa do quanto a imagem compressa representa a imagem a partir da qual ela foi gerada, ou seja, a imagem original. Atualmente, uma série de estudos com o objetivo de avaliar a qualidade de imagens obtidas a partir dos mais diferentes equipamentos têm sido realizados, sendo esta uma das motivações para a realização deste trabalho.

O texto é composto de quatro capítulos, sendo que no Capítulo 1 detalha-se os fundamentos e técnicas do Processamento de Imagens Digitais, envolvendo desde os princípios da digitalização até o armazenamento destes dados. São apresentados conceitos de Compressão e Descompressão de Imagens Digitais, abordando as duas técnicas existentes, ou seja, compressão sem perda de informação e compressão com perda de informação.

As Transformadas *Wavelets* e Espectro de Wiener no Processamento de Imagem Digital são discutidas no Capítulo 2. Há destaque para a *Wavelet* de Haar pelo fato da mesma ser utilizada neste trabalho. Além disso, este capítulo contempla conceitos da Análise Multiresolução e relata alguns estudos referentes à compressão de imagens digitais utilizando a transformada *Wavelet*. Aborda também a Análise da Qualidade da Imagem através do Espectro de Wiener.

O Capítulo 3 introduz o método para Compressão de Imagens Digitais fundamentado em Procedimentos de Huffman e *Wavelets,* detalha-se as características do sistema desenvolvido e no Capítulo 4 contemplam-se os Resultados, Discussões e Conclusões detalhados obtidos com este estudo.

Capítulo 1: Processamento de Imagens Digitais e sua Compressão

1.1. Processamento de Imagens Digitais

O Processamento de Imagens é uma área em constante crescimento. Dentre os diversos temas científicos abordados, pode-se citar: a compressão de imagens, a análise em multi-resolução e em multi-frequência, a análise estatística, a codificação e a transmissão de imagens.

A disciplina Processamento de Imagens deriva do Processamento de Sinais. Os sinais, como as imagens, são um suporte físico que carrega no seu interior uma determinada informação. Esta informação pode estar associada a uma medida (neste caso, um sinal em associação a um fenômeno físico), ou pode estar associado a um nível cognitivo (neste caso, o conhecimento). Processar uma imagem consiste em transformá-la sucessivamente com o objetivo de extrair as informações que se encontram presente para um fim específico. Processar uma imagem como feito pelo sistema visual humano (SVH) é tarefa complexa. Realizar as mesmas tarefas que o sistema visual humano com a ajuda de máquinas exige uma compreensão dos conhecimentos humanos. Esta característica faz com que o processamento de imagem seja, atualmente, uma disciplina com extrema dependência do sistema no qual ele está associado, não existindo uma solução única e abrangente para todos os problemas. Daí a não existência, até o momento, de sistemas de análise de imagens complexos e que funcionem para todos os casos (Albuquerque & Albuquerque, 1998).

Do ponto de vista da ótica, uma imagem é um conjunto de pontos que convergem para formar um todo, ou de maneira mais ampla, é o suporte para efetuar-se troca de informações. O termo imagem estava inicialmente associado ao domínio da luz visível. Atualmente é muito freqüente ouvir-se falar de imagem quando uma grande quantidade de dados está representada sob a forma bidimensional (por exemplo: as imagens acústicas, sísmicas, de satélites, infravermelhas, magnéticas, etc..). Os métodos recentes de exploração automática

desta informação permitiram o desenvolvimento de técnicas complexas, que podem ser globalmente classificadas em duas grandes linhas. A primeira está associada a uma Análise da Informação e a segunda representa as técnicas que permitem obter uma melhoria (*Enhancement*) significativa da imagem.

Um sistema de processamento de imagens pode ser constituído dos seguintes módulos:

(1) Aquisição da Imagem;

(2) Processamento;

(3) Representação;

(4) Armazenamento.

Na etapa de aquisição das imagens ocorre a captura e a discretização das mesmas. O processo de captura das imagens ocorre de várias maneiras, dentre elas pode-se citar imagens geradas por tomógrafos, as quais também foram utilizadas neste trabalho. Outra maneira de capturar imagens ocorre através da câmera de vídeo.

Na etapa de processamento é realizado o controle de exibição e/ou armazenamento das imagens, além da aplicação de técnicas do processamento digital de imagens.

A representação da imagem é realizada através de algum dispositivo de saída como os monitores de vídeo, impressoras entre outros.

O armazenamento das imagens é a etapa onde os dados das imagens processadas são convertidos em arquivos que possam ser armazenados em algum dispositivo de entrada e saída.

1.2. Fundamentos do Processamento de Imagens Digitais

Um sinal é uma descrição de como um parâmetro está relacionado a um outro parâmetro. Desde que ambos os parâmetros podem assumir um intervalo contínuo de valores, chamamos esse de sinal contínuo. Em comparação, passando este sinal por um conversor analógico/digital força cada um dos dois parâmetros a serem quantizados. Sinais formados por parâmetros que são quantizados desta

maneira são ditos sinais discretos ou sinais digitalizados. Na maioria das vezes sinais contínuos existem na natureza, enquanto sinais discretos existem dentro dos computadores. É também possível ter sinais onde um parâmetro é contínuo e o outro discreto.

Os sinais elétricos de interface com o meio ambiente em sistemas de telecomunicações ou provenientes de microfones e antenas que captam e/ou que agem sobre os sistemas físicos são geralmente sinais analógicos. O processamento de sinal no domínio analógico é uma tarefa que requer estruturas e circuitos analógicos. Utiliza-se para o processamento do sinal analógico no domínio digital processadores digitais de sinal (designados por DSP's – *Digital Signal Processors*).

Quando transportado para o domínio digital, o sinal pode ser processado de forma tão exata quanto o desejado - dependendo da resolução escolhida - sendo esta uma das principais vantagens do processamento digital de sinal. O desempenho de um sistema deste tipo é geralmente limitado pela velocidade e resolução dos blocos de conversão e pela qualidade dos blocos de filtragem envolvidos na aquisição e reconstrução do sinal.

Os blocos de conversão de sinal analógico-digital (conversor A/D) desempenham o papel de converter o sinal analógico de entrada num sinal digital, o que permite o seu processamento no domínio digital como um vetor de bits. A tarefa complementar de construção ou reconstrução de um sinal analógico de saída, a partir de um conjunto de bits, é feita por um bloco de conversão digital-analógico (conversor D/A) que transforma um número digital num nível de tensão correspondente. O objetivo do processo de aquisição de imagens é a geração de imagens digitais em razão dos dados. Para criar uma imagem digital, precisa-se converter os dados no sentido contínuo para a forma digital. Isto envolve dois processos: amostragem e quantização.

Um *pixel*, também conhecido por *"pel"* ou *"picture element"* é o elemento básico de uma imagem. O *pixel* é um elemento de dimensões finitas na representação de uma imagem digital, sendo sua forma mais comum a retangular ou quadrada. Segundo o pesquisador Gonzales, considerando-se uma imagem $f(x,y)$ a qual deseja-se converter para uma imagem digital, esta pode ser contínua considerando-se as coordenadas x e y e também a amplitude. Para convertê-la

para a forma digital deve-se considerar a função em ambas as coordenadas e na amplitude. A FIGURA 1.1(a) ilustra uma imagem contínua, $f(x,y)$, a qual será convertida para a forma digital (Gonzales & Woods, 2002).

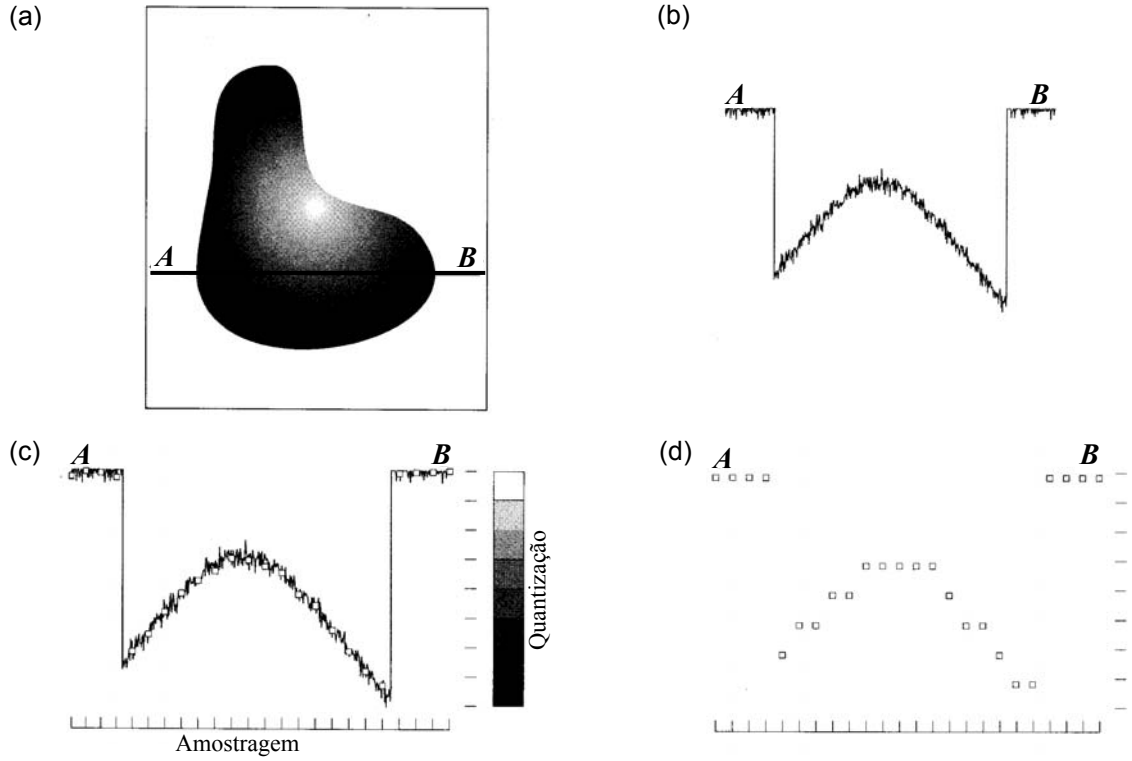


FIGURA 1.1 - (a) Imagem contínua, (b) Uma linha escaneada de A para B na imagem contínua, (c) Amostragem e Quantização, (d) Linha digital escaneada. (Gonzales & Woods, 2002, p.53)

A digitalização dos valores da coordenada é chamada amostragem. A digitalização dos valores da amplitude é chamada quantização. A função unidimensional mostrada na FIGURA 1.1(b) é uma plotagem dos valores da amplitude (nível de cinza) da imagem contínua ao longo da linha do segmento AB. As amostras são marcadas como pequenos quadrados brancos e o nível de cinza presente em cada ponto selecionado segue a escala de quantização presente na posição vertical, conforme a FIGURA 1.1(c). A amostra digital resultante da amostragem e quantização está presente na FIGURA 1.1(d) (Gonzales & Woods, 2002).

1.3. Considerações Iniciais

A utilização de imagens digitais destaca-se como importante fonte de informação no moderno mundo dos sistemas de comunicação.

Desde o início do processamento digital de imagens, a codificação da imagem tem sido reconhecida como um problema importante. A razão para isso é a grande quantidade de dados que precisa ser manuseado para transmissão ou armazenamento de imagens digitais, sendo necessário a utilização de métodos para reduzir esta quantidade para usar os canais de transmissão e para armazenar de maneira eficiente (Kunt *et al.*, 1987).

O termo compressão de dados refere-se ao processo de redução da quantidade de dados necessária para representar uma certa quantidade de informação. Deve-se ter em mente a diferença entre dados, informação e conhecimento, pois eles não são sinônimos. Os dados são os meios pelos quais a informação é conduzida. Muitas quantidades de dados podem ser usadas para representar a mesma quantidade de informação, ou seja, pode ocorrer redundância de dados. A redundância de dados é um tema central em compressão de imagens digitais. Através do conjunto de informações obtém-se o conhecimento. O conhecimento obtido através das informações disponíveis numa imagem é essencial na área de processamento de imagens e sinais quando se busca tomar uma decisão através do uso desta imagem.

A compressão de imagens trata o problema de reduzir a quantidade de dados necessária para representar uma imagem digital, ou seja, dada uma imagem em uma determinada especificação, os métodos de compressão têm por finalidade obter uma outra especificação dessa imagem que ocupe o menor espaço possível no seu armazenamento (Gonzales & Woods, 2002).

A base do processo de compressão é a remoção de dados redundantes. Segundo o pesquisador Baxes, esquemas de compressão de imagem são baseados no fato de que qualquer conjunto de dados pode, e geralmente ocorre, conter redundâncias (Baxes, 1994).

A área de compressão de imagens trata deste problema, ou seja, dada uma imagem em uma determinada especificação, os métodos de compressão têm

por finalidade obter outra especificação dessa imagem que ocupe o menor espaço possível no seu armazenamento.

A compressão de dados é composta de duas etapas: modelagem e codificação. A etapa de modelagem dos dados de entrada é responsável pela decisão de qual código representará a informação, ou seja, é simplesmente um conjunto de regras usadas para processar os dados de entrada e determinar qual código os representará. A codificação é um conjunto de símbolos que representa a informação.

A compressão de imagens coloridas é usualmente feita por compressão de cada componente colorido na imagem individualmente. No caso de uma imagem *RGB* [1], simplesmente aplicado o esquema de compressão para as três cores componentes da imagem. Do mesmo modo, a operação de descompressão é feita para as três componentes compressas da imagem.

A operação de compressão converte dados da imagem original em uma forma de dados da imagem compressa. A operação de descompressão converte os dados da imagem compressa anteriormente para sua forma descompressa original. As operações de compressão e descompressão de imagens são conhecidas como operações de código da imagem.

Ao longo dos anos, a compressão de imagens digitais vem crescendo continuamente em várias aplicações, tais como: crescimento da computação multimídia (uso de computadores digitais para impressão, publicação, produção de vídeo e disseminação); trabalho sobre resoluções espaciais de sensores de imageamento; a evolução das padronizações e transmissão de sinais de televisão; videoconferências, sensoriamento remoto; imageamento médico e de documentos; transmissão de fac-símiles; controle de veículos pilotados remotamente em aplicações militares e espaciais; controle de resíduos perigosos e outros.

---

[1] *RGB (Red, Green e Blue):* Em termos do processamento digital de imagens, o modelo mais utilizado é o *RGB*. No modelo *RGB*, cada cor aparece em seus componentes espectrais primário de vermelho, verde e azul. Este modelo está baseado no sistema de coordenada cartesiana. Imagens representadas no modelo de cor *RGB* consistem de três componentes de imagens, uma para cada cor primária.

1.4. Fundamentos da Compressão de Imagem Digital

Pesquisas sobre compressão de imagens digitais surgiram quando houve o desenvolvimento de métodos analógicos para a redução da largura de banda de transmissão de sinais de vídeo, um processo chamado compressão de largura de banda. A largura de banda consiste na capacidade física (velocidade admissível) do equipamento de transmissão de dados na rede. De uma forma simples representa a quantidade de informação transmitida na unidade de tempo, desta forma, uma maior largura de banda permite a transmissão de uma maior quantidade de informação no mesmo espaço de tempo.

Com o surgimento do computador digital e o desenvolvimento dos circuitos integrados avançados passou-se grande parte das abordagens analógicas para as digitais. Do ponto de vista da engenharia, sinais são funções ou seqüências que servem para transportar informação de uma fonte de mensagens a um destinatário. As características específicas dos sinais dependem do canal de comunicação utilizado para este transporte, que é definido segundo o tipo de distorção que introduz nos sinais. Os sinais são processados no lado transmissor com a finalidade de produzi-los e configurá-los e no lado receptor para extrair a informação neles contida, se possível com a máxima eficiência.

A etapa de quantização da maioria dos algoritmos de compressão de sinais e imagens permite a maior redução no tamanho de dados. A quantização causa perda de dados e introduz erros entre os sinais e a imagem compressa e seus originais (Seales *et al.*, 1997).

Alguns trabalhos teóricos sobre compressão de imagens digitais começaram a surgir na década de 40, quando pesquisadores formularam a visão probabilística da informação e sua representação, transmissão e compressão. Em 1948, Shannon abordou os principais aspectos da comunicação, estendendo o assunto para incluir novos fatores: o efeito do ruído no canal e a possível economia devido à estrutura estatística da mensagem original e devido à natureza do destino final da informação. Segundo Shannon, o problema fundamental da comunicação é aquele da reprodução exatamente em um ou outro ponto ou aproximadamente uma mensagem selecionada em qualquer ponto. As mensagens freqüentemente possuem sentido e o aspecto significante é que a mensagem real é uma selecionada de um conjunto de possíveis mensagens. Como o número de mensagens num

conjunto é finito, este número pode ser considerado como uma medida de informação produzida quando uma mensagem é escolhida de um conjunto sendo que as escolhas são igualmente prováveis (Shannon, 1948).

Um sistema de comunicação, conforme ilustra a FIGURA 1.2, consiste essencialmente de cinco partes:

1. Uma codificação tipo fonte (fonte de informação) que produz uma mensagem ou uma seqüência de mensagens para ser comunicada para um terminal receptor;
2. Uma codificação tipo canal (transmissor) que opera na mensagem de algum modo para produzir um sinal apropriado para transmissão sobre o canal;
3. O canal é simplesmente o meio utilizado para transmitir o sinal do transmissor para o receptor. Pode ser um par de fios, um cabo coaxial, uma banda de freqüência de rádio, um feixe de luz, etc;
4. O decodificador tipo canal (receptor) executa a operação inversa feita pelo transmissor, reconstruindo a mensagem através do sinal;
5. O decodificador tipo fonte (destinatário) é a pessoa ou coisa para a qual a mensagem é direcionada (Kunt *et al.*, 1987).

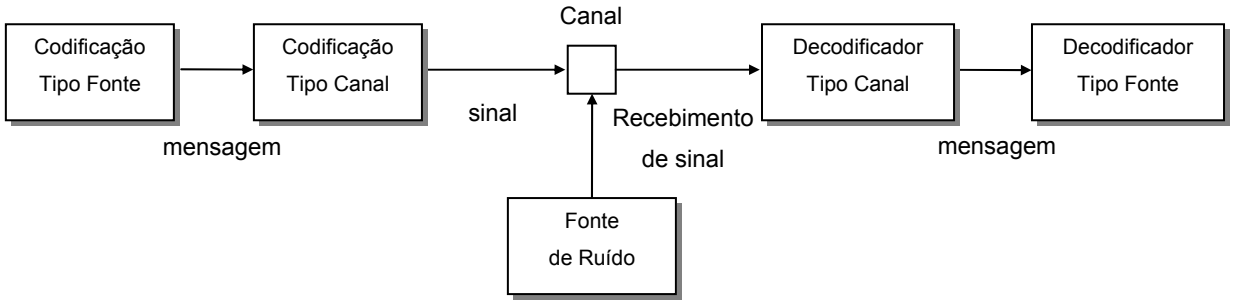


FIGURA 1.2 - Diagrama de um sistema de comunicação geral. (Kunt *et al.,* 1987, p.1307)

O bloco codificação tipo fonte é dividido em dois sub blocos: extrator de mensagem e atribuição da palavra código para mensagens, conforme ilustrado na FIGURA 1.3 abaixo (Kunt *et al.*, 1987).

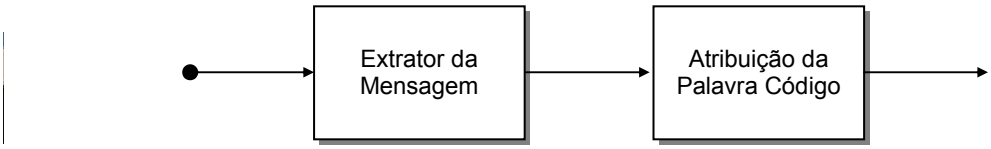


FIGURA 1.3 - Diagrama de uma codificação fonte. (Kunt *et al.,* 1987, p.1307)

Quando o objetivo é minimizar o tamanho de uma imagem para que a mesma possa ser eficientemente armazenada ou transportada, a solução é o processo de compressão. Posteriormente, a imagem compressa pode ser descompressa com uma operação inversa que restaura a imagem na sua forma original.

Operações de compressão e descompressão de imagens reduzem o conteúdo de dados necessários para descrever uma imagem. A compressão da imagem é possível porque a maioria das imagens, inerentemente possui grande quantidade de informação redundante. A eliminação dessas redundâncias com a conseqüente redução de espaços de memória para armazenagem e custo é o objetivo de todas as operações de compressão de imagens. Baseado nesse fato, os

métodos de compressão visam produzir, através da eliminação da redundância, o código mais compacto que preserve as informações essenciais contidas na imagem.

Todo sistema de aquisição de imagem digital produz quadros[2] na sua forma canônica. Isto significa que a cena análoga é modelada no espaço e quantizada no brilho. Se o tamanho do passo da modelagem é bastante pequeno, a habilidade da integração do sistema visual humano irá dar a ilusão de quadro contínuo para o observador humano. Neste contexto, uma imagem digital é uma matriz *N1* x *N2* de números inteiros. Entretanto, esta forma canônica requer uma grande quantidade de número de bits para sua representação. A compressão de imagens permite minimizar o número de bits requisitado para representar uma imagem (Egger *et al.*, 1999).

A técnica de compressão torna-se necessária onde uma grande quantidade de dados é produzida quando uma função intensidade de luz bidimensional é amostrada e quantizada para gerar uma imagem digital. Quando a quantidade de dados gerada é demasiadamente grande pode-se inviabilizar o armazenamento, o processamento e a comunicação. Segundo Brown e Shepherd, a compressão reduz a quantidade de memória necessária para armazenar um conjunto de dados. Conseqüentemente, isto reduz a quantidade de tempo necessária para transmitir um conjunto de dados após uma conexão de comunicação numa dada taxa (Brown & Shepherd, 1954).

Na literatura da área tem-se conhecimento sobre duas formas de compressão de imagens. Dependendo da área de aplicação, as informações que se desejam preservar podem ser de natureza objetiva ou subjetiva. No primeiro caso, o método de compressão deve permitir a recuperação exata dos dados da imagem original. Neste caso o processo é reversível, ou seja, tem-se uma codificação sem perda na imagem. O outro caso é chamado irreversível, ou seja, a técnica de compressão permite perda na imagem, que leva a representações que não são exatamente dados da imagem original, mas mantém um nível particular de qualidade da imagem subjetiva.

---

[2] Quadro: Um quadro define uma imagem completa que é estabelecida em um tempo t de varredura.

A FIGURA 1.4 ilustra as duas categorias de compressão de imagem digital.

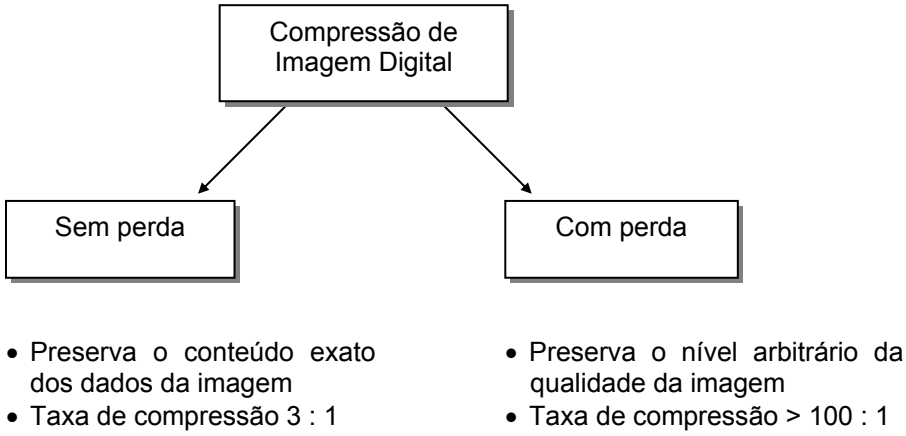


FIGURA 1.4 - Técnicas de compressão de imagem digital agrupadas em duas categorias: métodos sem perda e com perda de informações. (Baxes, 1994, p.180)

Todo esquema de compressão de imagem digital é considerado de mão dupla, ou seja, envolve ambas as operações de compressão e uma operação inversa denominada descompressão. A operação de compressão converte os dados da imagem original em um formato de dados de imagem compressa, conforme ilustrado na FIGURA 1.5. A operação de descompressão converte os dados da imagem compressa anteriormente para seu formato descompresso original. Operações de compressão e descompressão de imagens são chamadas operações de codificação da imagem porque os processos utilizam métodos de codificação de dados para representar uma imagem numa nova forma concisa.

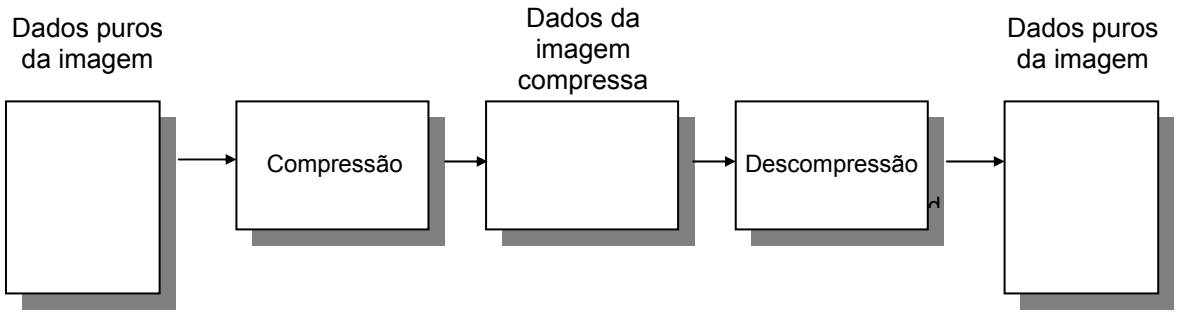


FIGURA 1.5 - Diagrama para operações de compressão e descompressão de imagem.

Operações de compressão e descompressão de imagens não são sempre operações simétricas. Isto significa que, para um esquema particular de compressão, uma operação pode demorar mais ou requerer mais esforço computacional que outra. Quando as duas operações requerem o mesmo esforço, elas são chamadas simétricas. Quando uma operação demora mais que outra, elas são chamadas assimétricas. As três formas de compressão de imagens simétricas e assimétricas são: código simétrico (a compressão e descompressão possuem tempo e esforço computacional similares), compressão de código assimétrico (a compressão demanda mais tempo ou esforço computacional, ou ambos, que a descompressão) e descompressão de código assimétrico (a descompressão demanda mais tempo ou esforço computacional, ou ambos, que a compressão).

Quando esquemas de codificação como o *Huffman*, Aritmético e Liv-Zempel são usados para comprimir uma imagem bidimensional, a imagem deve primeiramente ser convertida em uma seqüência unidimensional. Esta conversão é chamada linearização (Sahni et al., 1997).

As imagens naturais possuem redundância local e global. Redundância local causa uma perda na vizinhança na imagem para exibir coerência ou correlação. Estes esquemas são mais efetivos na preservação da redundância local da imagem e possuem expectativa de alcançar melhor compressão quando ligado com um esquema de codificação que pode tirar vantagem da redundância local. Alguns esquemas de linearização são mostrados na FIGURA 1.6. Cada *pixel* da

imagem é escaneado em alguma ordem para produzir a seqüência uni-dimensional (Sahni *et al.*, 1997).

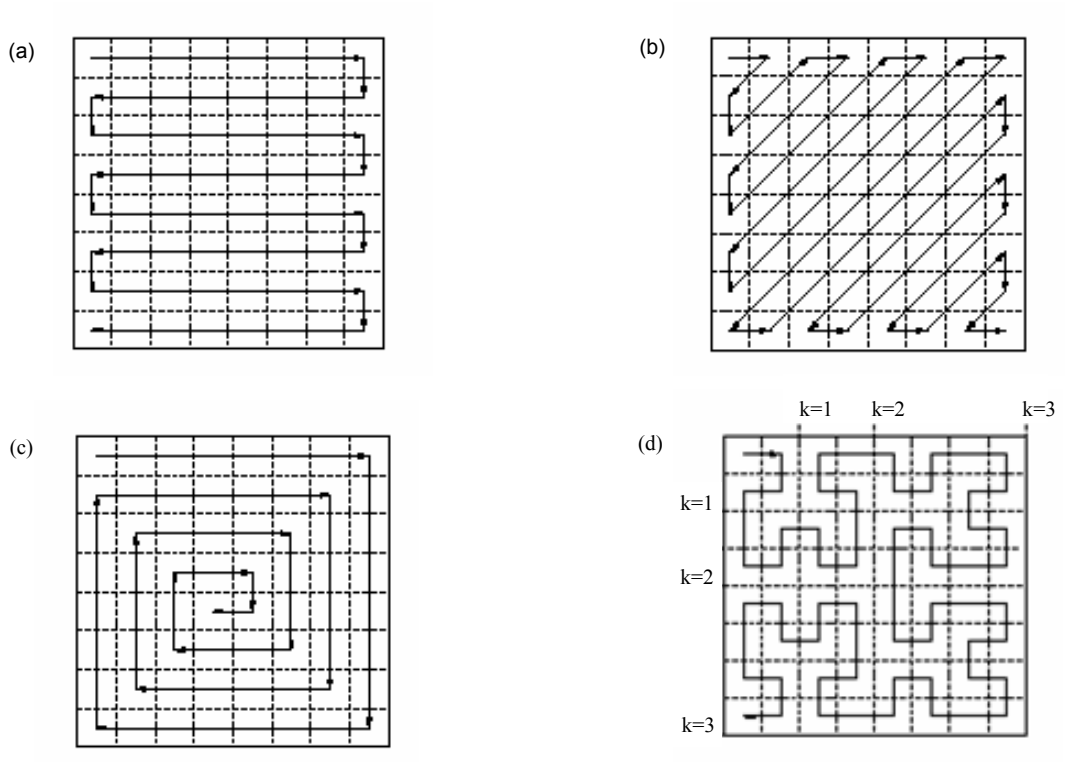


FIGURA 1.6 - (a)Caminho da varredura da maior fileira enrolado em espiral, (b)Caminho da varredura diagonal enrolado em espiral, (c)Caminho da varredura em espiral, (d)Caminho da varredura em desenho. (Sahni *et al.*, 1997, p.9)

O esquema da varredura da maior fileira enrolado em espiral é ilustrado na FIGURA 1.6(a). Neste caso, a imagem é escaneada fileira por fileira do topo à base, e as fileiras são alternadamente escaneadas da esquerda para a direita e da direita para a esquerda.

Na FIGURA 1.6(b) mostra-se o esquema da varredura diagonal, onde a imagem é escaneada ao longo das anti-diagonais (linhas com fileiras constante acrescido do valor da coluna) começando com o maior topo antidiagonal. Cada anti-diagonal é escaneada do canto da base esquerda para o canto do topo direito.

No esquema da varredura em espiral nota-se que a imagem é escaneada do exterior para o interior, traçando de fora uma espiral curva começando

do topo do canto esquerdo da imagem e procedendo no sentido horário, conforme ilustrado na FIGURA 1.6(c).

A FIGURA 1.6(d) ilustra o esquema da varredura em desenho (Hilbert). Este método é o melhor descrito recursivamente e requer uma imagem 2^k x 2^k. Quando k é ímpar, o caminho da varredura começa no *pixel* mais à esquerda da primeira fileira e termina no *pixel* mais à esquerda da fileira da base. Quando K é par, o caminho começa no *pixel* mais à esquerda da primeira fileira e termina no *pixel* mais à direita desta coluna. Neste método a imagem é escaneada quadrante por quadrante.

A compressão de imagem é possível porque as imagens, em geral, apresentam um alto grau de coerência, que se traduz em uma redundância de informação quando codificada. A base do processo de redução é a remoção de dados redundantes. Matematicamente, isto significa transformar uma matriz de *pixels* de duas dimensões num conjunto de dados estatisticamente descorrelacionado. Esta transformação é aplicada antes do armazenamento ou transmissão da imagem. Ao final do processo, a imagem compressa é descompressa para reconstruir a imagem original ou uma aproximação dela.

Sendo n_1 e n_2 o número de unidades de transporte de informação em dois conjuntos de dados que representam a mesma informação, a redundância de dados relativa R_D do primeiro conjunto de dados pode ser definida como:

$$R_D = 1 - \frac{1}{C_R} \tag{1.1}$$

onde C_R, geralmente denominado taxa de compressão, é dado por:

$$C_R = \frac{n_1}{n_2} \tag{1.2}$$

Se n_2 = n_1, C_R = 1 e R_D = 0, tem-se que, relativo ao segundo conjunto de dados, a primeira representação da informação não contém dados redundantes. Quando n_2 << n_1, $C_R \rightarrow \infty$ e $R_D \rightarrow$ 1, tem-se compressão significativa e dados altamente redundantes. No último caso, onde n_2 >> n_1, $C_R \rightarrow 0$ e $R_D \rightarrow$ - ∞, tem-se que o segundo conjunto de dados contém muito mais dados do que a representação

original. Este é o caso normalmente indesejável de expansão de dados. Em geral, C_R e R_D situam-se nos intervalos abertos (0, ∞) e (-∞, 1), respectivamente.

No estudo da compressão de imagens digitais, três redundâncias básicas de dados podem ser identificadas e exploradas: redundância de codificação (a forma como a imagem é representada – codificada - introduz redundância), redundância interpixels (a imagem apresenta repetições de padrões de *pixels*) e redundância psicovisual (a imagem inclui informação que visualmente não é relevante). A compressão de dados tem êxito quando uma ou mais dessas redundâncias são reduzidas ou eliminadas.

1.4.1. Redundância de Codificação

Na redundância de código, o processo de codificação atribui código com tamanho variável, número de bits, de acordo com a probabilidade de ocorrência de determinado tom de cinza ou cor do *pixel* na cena; ou seja, o nível de cinza ou cor com maior ocorrência serão representados por um código com comprimento menor; ao contrário, se um nível de cinza ou cor tem pouca presença na cena é representado por um código maior.

Para o pesquisador Pinho, a maioria das técnicas de compressão de imagens confia na hipótese de que as imagens são uniformes. Esta hipótese é verificada através da maioria das imagens que representam conteúdo natural e são utilizadas para testar a performance de técnicas de compressão. Entretanto, a supremacia do uso de imagens naturais tem sido diminuída, dando lugar a imagens que contém texto incluso, gráfico e materiais gerados por computador, além do usual conteúdo natural. A diversidade de conteúdos apresentados nas imagens gera um desafio para as técnicas de propósito geral que vem sendo utilizadas para comprimir imagens naturais, gerando degradação na medida de compressão, afetando assim as técnicas de compressão com e sem perda. A importância das imagens compressas eficientemente que são caracterizadas por terem histogramas esparsos está aumentando. O trabalho desenvolvido pelo pesquisador Pinho descreve e compara a eficiência de várias técnicas utilizadas com a finalidade de aumentar a compressão de imagens sem perda de conteúdo com a utilização de histogramas esparsos (Pinho, 2002).

Ao estudar-se o histograma de uma imagem, nota-se que uma grande quantidade de informação sobre a aparência de uma imagem pode ser obtida através de um histograma de seus níveis de cinza (Gonzales & Woods, 2002). Pode-se utilizar técnicas de realce de imagens por modificação de histograma, assumindo-se que os níveis de cinza de uma imagem são quantidades aleatórias. O histograma de níveis de cinza de uma imagem pode também fornecer uma grande quantidade de *"insight"* na construção de códigos (um código é um sistema de símbolos – letras, números, bits – usados para representar um corpo de informações ou um conjunto de eventos. Para cada peça de informação ou evento é atribuída uma seqüência de símbolos de codificação, denominados de palavra código – "*code word*". O número de símbolos em cada palavra de código é seu comprimento), para reduzir a quantidade de dados usada para representá-la.

Assume-se que uma variável aleatória discreta r_k no intervalo [0, 1] representa os níveis de cinza de uma imagem e que cada r_k ocorre com probabilidade $p_r(r_k)$.

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, ..., L - 1 \tag{1.3}$$

em que L é o número de níveis de cinza, n_k é o número de vezes que o k-ésimo nível de cinza aparece na imagem, e n é o número total de *pixels* da imagem. Se o número de bits utilizado para representar cada valor de r_k for $l(r_k)$, o número médio de bits necessários para representar cada *pixel* é

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) \tag{1.4}$$

Desta maneira, tem-se que o comprimento médio das palavras de código atribuídas aos vários valores de níveis de cinza é o somatório do produto do número de bits utilizados para representar cada nível de cinza e a probabilidade em que o nível de cinza ocorre. Portanto, o número total de bits necessários para codificar uma imagem *M* x *N* é MNL_{avg}.

A atribuição de menos bits aos níveis de cinza mais prováveis do que aos menos prováveis permite a compressão de dados. Esta técnica denomina-se codificação de comprimento variável. Se os níveis de cinza de uma imagem são codificados de modo que utilizam mais símbolos de codificação do que o absolutamente necessário para representar cada nível de cinza, diz-se que ocorre redundância de codificação. A redundância de codificação ocorre quando os códigos atribuídos a um conjunto de eventos (como exemplo citamos os níveis de cinza) não foram escolhidos de forma a tirar toda vantagem das probabilidades dos eventos. Ocorre na maioria das imagens que certos níveis de cinza são mais prováveis do que outros (os histogramas da maioria das imagens não são uniformes). Uma codificação binária natural de seus níveis de cinza atribui o mesmo número de bits tanto para o valor mais provável quanto para o menos provável, não minimizando a equação (1.4) e resultando em redundância de codificação.

1.4.2. Redundância Interpixel

A redundância interpixel permite prever o valor de um *pixel* pelo valor de seus *pixels* vizinhos; esta correlação espacial está ligada ao relacionamento geométrico entre os objetos na imagem (Gonzales & Woods, 2002).

Em algumas imagens existem padrões de *pixels* que se repetem, implicando que um *pixel* introduz pouca informação, relativamente aos seus vizinhos, porque o valor do *pixel* pode ser previsto a partir do valor dos vizinhos. Muito da contribuição visual de um único *pixel* para uma imagem é redundante, já que ela poderia ser prevista com base nos valores de seus vizinhos. Essa redundância é conhecida por redundância interpixel.

A redundância interpixel é normalmente removida através da transformação para um formato mais eficiente (geralmente “não visual”), por exemplo, utilizando as diferenças entre *pixels* adjacentes para representar uma imagem. As transformações que removem redundância interpixel são chamadas mapeamentos, e são consideradas reversíveis se os elementos da imagem original puderem ser reconstruídos a partir do conjunto de dados transformado.

1.4.3. Redundância Psicovisual

A redundância psicovisual está relacionada à informação visual real ou quantificada em uma cena. Portanto, a redução ou a eliminação da redundância psicovisual, leva necessariamente a um processamento com perdas.

Algumas informações em imagens têm menos importância relativa do que outras no processamento visual normal, pois o olho humano não responde com uma mesma sensibilidade a todas as informações visuais. Estas informações são ditas psicovisualmente redundantes e podem ser eliminadas sem prejudicar a qualidade de percepção da imagem. Esta redundância é fundamentalmente diferente das redundâncias anteriores, pois está associada com a informação visual quantificável ou real. Sua eliminação é possível porque a informação em si não é essencial para o processamento visual normal. A eliminação dos dados psicovisualmente redundantes resulta numa perda de informação quantitativa denominada quantização. Esta operação é irreversível (a informação visual é perdida), a quantização resulta em compressão de dados com perda.

1.4.4. Critérios de Fidelidade

A remoção de dados psicovisualmente redundantes resulta numa perda na informação visual quantitativa ou real. Muitas vezes esta informação perdida pode ser de grande interesse, por isso um meio de quantização da natureza e extensão da perda de informação que possa ser repetido ou reproduzido é altamente desejável. Utilizam-se duas classes de critérios para essa avaliação: critérios de fidelidade objetivos e critérios de fidelidade subjetivos (Gonzales & Woods, 2002).

Quando o nível de perda da informação puder ser expresso como uma função da imagem de entrada e a imagem de saída for compressa e subseqüentemente descompressa, diz-se que ela baseia-se num critério de fidelidade objetivo. Os critérios de fidelidade objetivos oferecem um mecanismo simples para a avaliação da perda de informação, entretanto essas imagens são geralmente visualizadas por seres humanos. Conseqüentemente, a medida da qualidade da imagem através de avaliações subjetivas de um observador humano é mais apropriada. Isso ocorre mostrando-se uma imagem descompressa para uma

porção apropriada de espectadores, tirando-se a média das suas avaliações, que podem ser feitas usando-se uma escala de notas absolutas ou através de comparações lado a lado de $f(x,y)$ e $f'(x,y)$. A tabela 1.1 representa as avaliações subjetivas que podem ser feitas, e nesse caso se diz que as avaliações baseiam-se em critérios de fidelidade subjetivos.

Valor	***Conceito***	***Descrição***
1	Excelente	Imagem de qualidade extremamente alta, tão boa quanto se possa desejar.
2	Boa	Imagem de alta qualidade, permitindo visualização agradável. A interferência não é objetável.
3	Regular	Imagem de qualidade aceitável. A interferência não é objetável.
4	Limite	Imagem de qualidade ruim; há necessidade de melhorá-la. A interferência é um tanto objetável.
5	Inferior	Imagem muito ruim, mas pode-se apreciar a mesma. Interferência objetável faz-se definitivamente presente.
6	Inútil	Imagem tão ruim que não pode ser apreciada.

TABELA 1.1 - Escala de notas da *"Television Allocations Study Organization".* (Gonzales & Woods, 2002, p.420)

1.5. Modelos de Compressão de Imagens

As três técnicas gerais para reduzir ou comprimir dados são geralmente combinadas para formar sistemas práticos de compressão de imagens.

A FIGURA 1.7 mostra um sistema de compressão consistindo de dois blocos estruturais distintos: um codificador e um decodificador. Uma imagem de entrada $f(x,y)$ é alimentada no codificador, que cria um conjunto de símbolos a partir dos dados de entrada. Depois da transmissão ao longo do canal, a representação codificada é alimentada no decodificador, onde uma imagem de saída reconstruída $f'(x,y)$ é gerada. A imagem de saída pode ou não ser uma réplica

exata da imagem de entrada. Se for, o sistema é livre de erro (sem perdas); se não, algum nível de distorção se faz presente na imagem reconstruída.

O codificador é composto de um codificador tipo fonte (que remove redundâncias da entrada) e de um codificador tipo canal (que aumenta a imunidade ao ruído da saída do codificador tipo fonte). O decodificador é composto por um decodificador tipo canal seguido de um decodificador tipo fonte. Se o canal entre o codificador e o decodificador for livre de ruídos (não propenso a erro), o codificador tipo canal e decodificador são omitidos e o codificador geral e decodificador torna-se o codificador tipo fonte e decodificador, respectivamente.

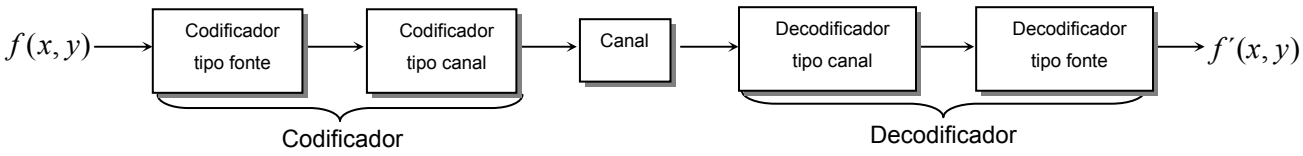


FIGURA 1.7 - Modelo de sistema de compressão genérico. (Gonzales & Woods, 2002, p.421)

1.5.1. Codificadores e Decodificadores Tipo Fonte

O codificador tipo fonte é responsável pela redução ou eliminação de qualquer redundância psicovisual ou interpixel ou de codificação na imagem de entrada. Normalmente, o codificador tipo fonte é composto por três operadores independentes: o mapeador (que transforma os dados em entrada num formato projetado para reduzir as redundâncias interpixels nas imagens de entrada; essa operação é reversível); o bloco quantizador (que reduz as redundâncias psicovisuais da imagem de entrada; essa operação é irreversível, portanto, ela deve ser omitida quando se deseja compressão livre de erros); o codificador de símbolos (que cria um código de comprimento fixo ou variável para representar a saída do quantizador e mapeia a saída de acordo com o código; essa operação é reversível). Depois de

completar o passo de codificador de símbolos, uma imagem de entrada é processada para remover uma das três redundâncias: de codificação, interpixel ou psicovisual (Gonzales & Woods, 2002).

Nem sempre estas três operações são incluídas em todos os sistemas de compressão, pois se desejarmos uma compressão livre de erros, o quantizador deverá ser omitido do processo.

O decodificador tipo fonte é composto por dois componentes: um decodificador de símbolos e um mapeador inverso. Esses blocos desempenham, em ordem invertida, as operações inversas dos blocos codificador de símbolos do codificador tipo fonte e mapeador. Um bloco quantizador inverso não é incluído no modelo de decodificador tipo fonte genérico porque a quantização resulta em perda irreversível.

1.5.2. Codificadores e Decodificadores Tipo Canal

Os codificadores e decodificadores tipo canal são importantes quando o canal ilustrado no modelo de sistema de compressão genérico acima for ruidoso ou passível de erro. A função de ambos é reduzir o impacto do ruído do canal através da inserção de uma forma controlada de redundância nos dados fonte codificados (Gonzales & Woods, 2002).

Uma das técnicas de codificação tipo canal baseia-se na justaposição de bits suficientes aos dados sendo codificados para garantir que o número mínimo de bits tenha que mudar entre as palavras de código válidas. Esta técnica foi concebida por R. W. Hamming em 1950 e garante a detecção e/ou correção de erros de múltiplos bits.

Operações de compressão de imagem não podem ser confundidas com formatos de troca de arquivos de imagem. Muitas vezes os dois termos são usados permutavelmente. Geralmente, esquemas de compressão definem um algoritmo para compressão e descompressão dos dados da imagem, eles raramente definem formatos de troca de arquivos. Formatos de troca de arquivos definem as estruturas dos dados para organização de um arquivo imagem.

1.6. Compressão Sem Perdas

As técnicas de compressão e descompressão sem perdas da imagem são usadas quando os dados na imagem devem ser exatamente preservados. Nesta área incluem-se as imagens médicas e outras com finalidades científicas. Sempre que a imagem possuir características que possam conter informações importantes, é preferível utilizar a técnica de compressão sem perda.

Este tipo de técnica é utilizado quando a imagem poderá ser mais tarde processada com o objetivo de acentuação, técnicas de restaurações ou extração de atributos.

A FIGURA 1.8 descreve um típico sistema que permite a compressão sem perda de informação:

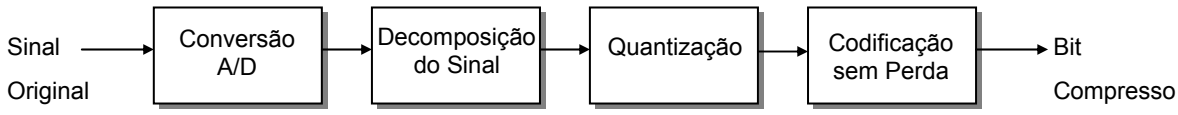


FIGURA 1.8- Um sistema típico de compressão. (Girod, 1998, p.40)

Para o pesquisador Girod e seus colaboradores, devido a vasta quantidade de dados associado com imagem e vídeo, a compressão é uma tecnologia chave para a transmissão digital e armazenamento. A disponibilidade e demanda para imagens e vídeo continua crescendo na capacidade da rede. A conversão de análogo para digital converte e quantiza uma imagem, produzindo uma representação digital. Uma decomposição do sinal usa transformação linear ou filtros para quebrar esta representação digital com canais paralelos de imagens separadas. A maior parte da compressão ocorre no estágio de quantização, com operações no

domínio da transformada em *pixels* individuais (quantização escalar) ou em um grupo de *pixels* (quantização vetorial). A compressão sem perda (codificação de entropia) tipicamente envolve codificação *Run-Length* combinado com *Huffman* ou códigos aritméticos para guardar bits num modelo transformável. Ocasionalmente, componentes específicas podem ser combinadas ou omitidas, mas a maioria das codificações de imagem padrões possui todos os componentes de alguma forma (Girod *et al.*, 1998).

A performance de um algoritmo de compressão de imagens sem perda pode ser especificada em termos de eficiência de compressão e complexidade. A eficiência da compressão é medida pela proporção da compressão ou pela taxa de bits. A proporção da compressão é a proporção do tamanho da imagem original pelo tamanho da imagem compressa; e a taxa de bits é o número de bits por *pixel* requerido pela imagem compressa.

$$\mathrm{Pr}\,oporçãoDaCompressão = \frac{NúmeroDeBitsPorPixelDa\,\mathrm{Im}\,agemComprimida}{TaxaDeBit} \qquad (1.5)$$

A eficiência do método de compressão pode ser comparada à entropia da imagem fonte. A entropia fonte é definida como a quantidade de informação contida na fonte e sua unidade são definidas através de bits por *pixel*. A complexidade de um algoritmo de compressão de imagens é medida pelo número de operações aritméticas requeridas para performance entre a codificação e processos de decodificação. Este é um importante fator para aplicações envolvendo compressão e descompressão de imagens *on line* onde a velocidade é crucial (Sahni *et al.*, 1997).

Três estratégias são utilizadas para obter-se a compressão livre de erros: codificação por tamanho variável, codificação por planos de bits e codificação previsora sem perdas. Todas as três técnicas são aplicáveis tanto a imagens

binárias como níveis de cinza. As técnicas de compressão livre de erros são compostas de duas operações:

1. definição de uma representação alternativa da imagem em que as redundâncias interpixels sejam reduzidas (mapeamento);
2. codificação da representação para eliminação das redundâncias de codificação.

1.6.1. Codificação por Tamanho Variável

Reduzir-se apenas a redundância de codificação é a maneira mais simples de obter-se a compressão de imagens livres de erros.

Esta técnica torna-se possível através da construção de um código de tamanho variável que atribua as menores palavras possíveis aos níveis de cinza mais prováveis. Algumas técnicas, como a Codificação de Huffman e a Codificação Aritmética são utilizadas para a construção de tal código.

Codificação de Huffman

O método de codificação não uniforme mais comum para a redução de redundância é a codificação de Huffman. Esse método, escrito em 1952 por David Huffman, utiliza a distribuição de probabilidade dos níveis de quantização da imagem (obtida através do histograma de freqüência), de forma a obter uma codificação adaptativa que é simples de ser implementada e resulta em uma taxa média de bits bastante próxima da entropia da imagem.

Na codificação Huffman cada símbolo do dado não compresso é substituído por um código. Os códigos de símbolos são de tamanho variável e o símbolo que ocorrer mais freqüentemente nos dados não compressos tem código de menor tamanho que símbolos que ocorrem menos freqüentemente. Os códigos de símbolos satisfazem uma propriedade prefixa: nenhum código é um prefixo próprio de outro código. Pela não codificação freqüentemente ocorre símbolos pelo menor código e não freqüentemente ocorrem símbolos por código maior, uma redução significativa no espaço necessário para os dados é obtida. No caso de imagens, valores de *pixels* individuais são considerados para representar símbolos individuais

e o conjunto de símbolos consiste de todos os valores de cinza. Para uma imagem de 8 bits por *pixel*, o tamanho do conjunto de símbolo é 2^8, ou seja 256 (Sahni *et al.*, 1997).

O código de Huffman é um código de bloco instantaneamente decodificável de maneira única. É considerado código de bloco, pois cada símbolo fonte é mapeado em uma seqüência fixa de símbolos de código. Como cada palavra de código em uma cadeia de símbolos de código pode ser decodificado sem referência aos símbolos sucessivos, o código de Huffman é considerado instantâneo. Pelo fato de que qualquer cadeia de símbolos de código pode ser decodificada de maneira única, o código Huffman é unicamente decodificável.

Durante o processo de criação do código, uma árvore binária representando estes códigos é criada. O primeiro passo na criação do código e Huffman é criar um array de caracteres de freqüências. A árvore binária pode ser facilmente construída pelo agrupamento recursivo dos caracteres de freqüências mais baixas e dos nós (Crane, 1997).

O algoritmo para construção da árvore de Huffman segue os passos descritos abaixo:

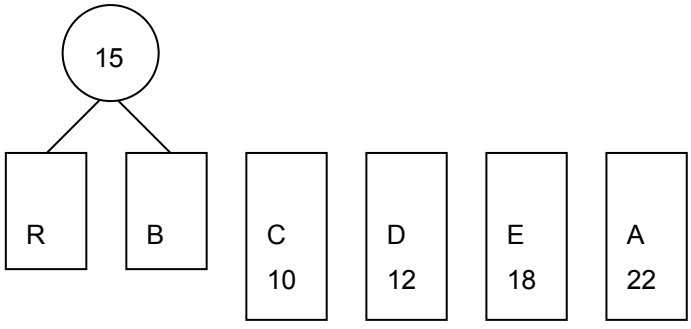
1. Determina-se a freqüência de ocorrência de cada símbolo. Todos os caracteres são inicialmente considerados nós livres;
2. Cria-se uma lista de prioridades classificada em ordem crescente de freqüência de ocorrência de cada símbolo;
3. Transfere-se para um nó pai os dois nós livres com as freqüências mais baixas. O peso desse novo nó será igual à soma dos dois nós filhos livres;
4. Remove-se da lista de nós livres os dois nós filhos. Cria-se o mais novo nó pai e adiciona-o à lista;
5. Repetem-se os passos 2 e 3 até existir somente um nó livre à esquerda. Este nó livre é a raiz da árvore.

Uma aplicação de árvore binária em codificação de dados é apresentada abaixo. Suponha que uma mensagem seja composta pelos caracteres A, B, C, D, E, R e que a freqüência de uso destas letras na mensagem seja 22, 8,

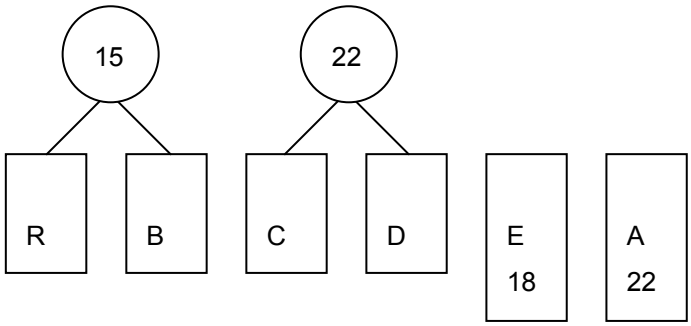
10, 12, 18, 7. A técnica de Huffman consiste em construir uma árvore binária baseada na freqüência de uso das letras de tal forma que as mais freqüentemente usadas (A, E) apareçam mais perto da raiz que as menos freqüentemente usadas (B, R). A construção desta árvore binária será feita de baixo para cima, começando a partir das letras menos usadas até atingir a raiz. Nesta árvore binária, as letras serão representadas nas folhas e os seus vértices internos conterão um número correspondente à soma das freqüências dos seus descendentes. Em cada passo do algoritmo tem-se uma coleção de árvores de Huffman da qual toma-se as duas com menor valor associado e transforma-as num só, cujo valor é a soma dos valores dos descendentes. Inicialmente, cada uma das letras será uma árvore composta apenas pela raiz e cujo conteúdo é o valor da freqüência de uso:

A	B	C	D	E	R
22	8	10	12	18	7

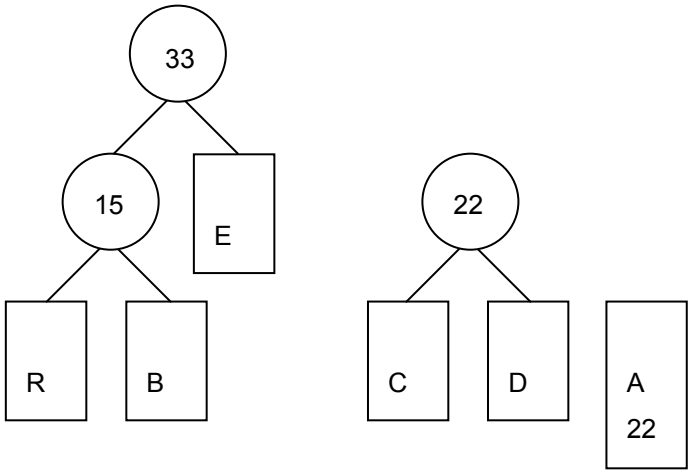
Selecionam-se as duas raízes de menor valor (7 e 8) e junta-as em uma nova árvore, que fará parte da coleção:



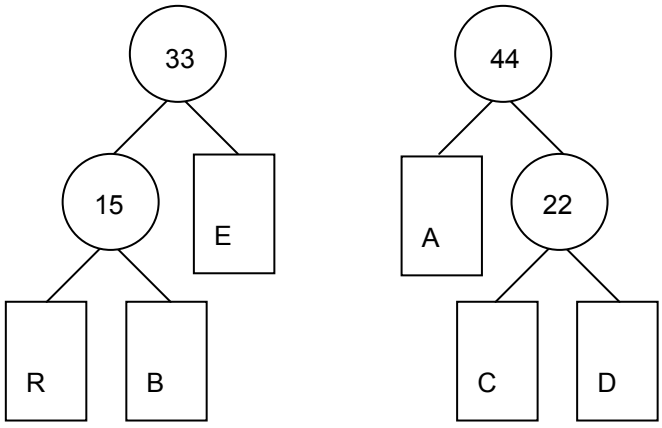
Repetindo o processo, escolhe-se as árvores com raízes 10 e 12, obtendo 22 na raiz. Neste ponto, a coleção de árvores tem a forma:



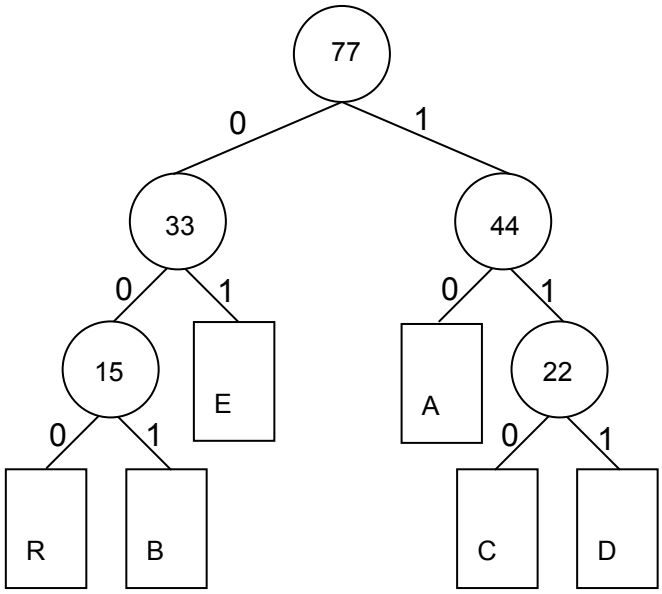
Continuando o mesmo processo, escolhe-se 18 e 15 formando:



Depois, 22 e 22 formando:



Finalmente, chega-se à:



Tem-se, então a árvore de Huffman completamente construída. Associam-se 0 às arestas que ligam um vértice com seu filho esquerdo e 1 às arestas que ligam um vértice com seu filho à direita. O código correspondente a cada letra será formado pelo número binário associado ao caminho da raiz até a folha correspondente. Com isso, obtém-se a seguinte tabela:

A	B	C	D	E	R
10	001	110	111	01	000

Para decodificar uma mensagem obtida através da tabela acima, por exemplo, 001 10 000 110 10, basta utilizar cada bit da seqüência para percorrer a árvore de Huffman desde a raiz até se atingir uma folha, quando se obtém o caracter correspondente. Deve-se, então voltar à raiz e continuar a percorrer a árvore para continuar a decodificação. Assim acontece com os demais caracteres, em ordem crescente do número de bits, conforme a prioridade. A codificação Huffman manteve a propriedade de permitir a decodificação direta, não permitindo que os bits da codificação de um caracter confundissem com a de outro.

No código de Huffman, nenhum código pode ser prefixo de outro código. Isto garante que a codificação binária dos dados é decifrável. O resultado do código tem o formato de uma árvore (Pitas, 1995).

Suponha que uma imagem tenha 8 níveis de intensidade. O número de bits pela codificação é *B* = 3. Assume-se que as probabilidades $p(i)$, i = 0, ..., 2^{B-1}, são conhecidas e criamos uma coluna de níveis de intensidade com probabilidades decrescentes $p(i)$ como mostra a FIGURA 1.9. As intensidades desta coluna constituem os níveis da árvore do código de Huffman. A árvore é construída em *B*–1 passos. Em cada passo os dois nós da árvore que possuem probabilidades mínimas são conectados para formar um nó intermediário. A probabilidade assumida por este nó é a soma das probabilidades dos dois galhos. Este procedimento é repetido até que todos os galhos (níveis de intensidade) são usados e a soma da probabilidade é 1. O código da árvore é arranjado para eliminar cruzamento sobreposto, como mostra a FIGURA 1.10. Os códigos são construídos pelo cruzamento da árvore de decodificação da raiz para seus níveis. Em cada nível, atribuímos 0 para o topo do galho e 1 para a base do galho. Este procedimento é repetido para todos os níveis da árvore. Cada folha corresponde a um único nível de intensidade. O código para esta intensidade consiste de 0´s e de 1´s que existe no caminho da raiz para sua folha específica. Como resultado, para os níveis de intensidade i = 1, 2 são atribuídos códigos pequenos , pois eles possuem alta probabilidade de ocorrência.

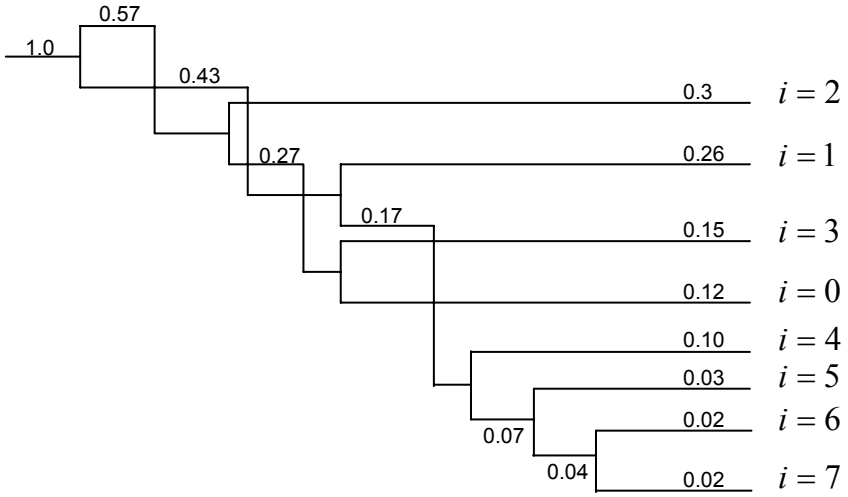


FIGURA 1.9 – Construção da árvore do Código de Huffman. (Pitas, 1995, p.178)

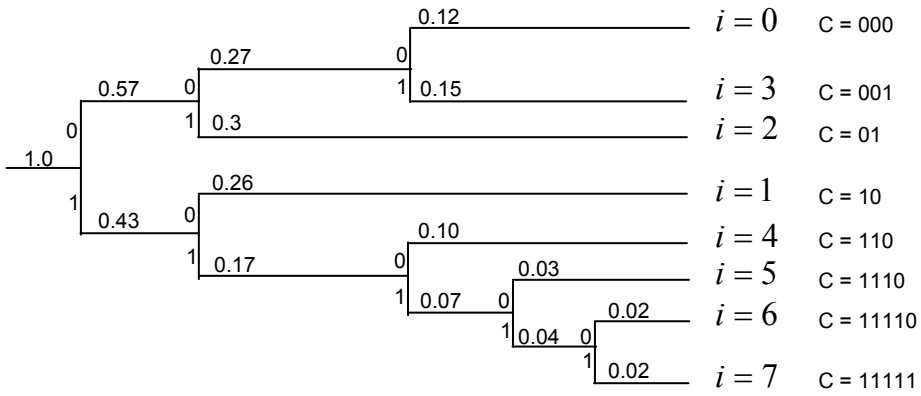


FIGURA 1.10 – Árvore do Código de Huffman arranjada. (Pitas, 1995, p.178)

Codificação Aritmética

Neste método, uma mensagem é codificada como um intervalo [0,1), onde [x, y) denota uma metade do intervalo aberto com inclusão de x, mas exclusão de y. Existem dois conceitos fundamentais na codificação aritmética: a probabilidade de um símbolo e o intervalo não codificado percorrido por um símbolo. A probabilidade de ocorrência de símbolos fonte determina a eficiência da compressão bem como o intervalo percorrido dos símbolos fonte para o processo não codificado. Esses intervalos percorridos estão contidos dentro do intervalo de 0 à 1 e determina a compressão de saída (Sahni *et al.*, 1997).

Na codificação aritmética não existe a correspondência um-a-um entre os símbolos-fonte e as palavras-código, ou seja, toda uma seqüência de símbolos-fonte é atribuída a uma única palavra de código aritmético. À medida que o número de símbolos na mensagem aumenta, o intervalo usado para representá-la diminui e o número de bits utilizados para representar o intervalo aumenta. Desta maneira, cada símbolo da mensagem reduz o tamanho do intervalo de acordo com a sua probabilidade de ocorrência.

A codificação aritmética difere-se da codificação de *Huffman* pelo fato da primeira não necessitar que cada símbolo-fonte seja traduzido em um número inteiro de símbolos-código.

1.6.2. Codificação por Planos de Bits

A codificação por planos de bits decompõe uma imagem monocromática ou colorida em uma série de imagens binárias, seguindo-se da compressão de cada imagem binária por um dos vários métodos para compressão binária.

Pode-se representar os níveis de cinza de uma imagem em níveis de cinza de *m* bits na forma de um polinômio de base 2. Esse código possui a propriedade de que palavras código sucessivas diferem apenas 1 bit de posição, afetando menos todos os *m* planos de bits.

1.6.3. Codificação Previsora sem Perdas

Outra abordagem de compressão livre de erro que não requer a decomposição da imagem em *m* planos de bits é a codificação previsora sem perdas. Esta técnica baseia-se na eliminação de redundâncias interpixels de *pixels* pouco espaçados através da extração e codificação apenas da nova informação em cada *pixel*, que é definida como a diferença entre o *pixel* real e um valor previsto daquele *pixel*.

1.7. Compressão Com Perdas

Ao contrário da abordagem livre de erro, a codificação com perdas baseia-se no comprometimento da precisão da imagem reconstruída em relação a um aumento de compressão.

As técnicas de compressão e descompressão com perdas da imagem são usadas quando a qualidade da imagem reconstruída poderá ser mantida em algum nível, mas não precisamente idêntica à imagem original.

Técnicas com perdas são utilizadas em imagens onde a medida exata ou o objetivo do processamento não deverá ser usado.

Para aplicações como vídeo conferência ou aplicações multimídia, alguma perda de informação é usualmente tolerada em troca para uma alta taxa de compressão. Alguns algoritmos são utilizados para a compressão de imagens com perda: o algoritmo EZW, proposto por Shapiro (Shapiro, 1993); o esquema de compressão EPIC, proposto por Simoncelli e colaboradores (Simoncelli *et al.*, 1990); o esquema AFB, proposto por Egger & Li (Egger & Li, 1995) e a compressão ASD, desenvolvido por Egger e colaboradores (Egger *et al.*, 1995). O melhor instrumento para medir a qualidade da imagem é o olho humano (Egger *et al.*, 1999).

Na compressão com perda, nos dados recriados é permitido diferir da origem de dados. Isto significa que a maioria dos mecanismos com perda é controlado por um parâmetro que emprega fidelidade em oposição ao tamanho compresso, e métodos bem diferentes são usados tanto para compressão com perda como para a compressão sem perda (Moffat, 1997).

A técnica de compressão com perdas da imagem proporciona fator de compressão maior de 100:1, isto significa que o primeiro conjunto de dados possui 100 unidades de transporte de informação (bits) para cada uma unidade nos conjuntos após a compressão. O fator de compressão depende do conteúdo da imagem, das operações utilizadas e da qualidade aceitável da imagem descompressa. Se a distorção resultante do processo de compressão puder ser tolerada, o aumento na compressão pode ser significativo.

Esquemas com perda são baseados em transformadas ortogonais e quantização proporcionando uma alta taxa de compressão na ordem de 100:1. A transformada *Wavelet* e codificação subbanda utilizam bancos de filtros que vem sendo extensivamente pesquisado para compressão de imagem sem perda (Ramaswamy *et al.*, 1996).

Excelentes resultados de compressão com perda têm sido obtidos usando a transformada *Wavelet*. Em um contexto de imagem, ela produz uma representação de multiresolução, que serão mostradas para serem naturalmente processadas para transmissão progressiva. Uma transformada de multiresolução para compressão com perda é conhecida na comunidade de imageamento médica como a transformada seqüencial (S-transform) (Said & Pearlman, 1996).

1.7.1. Codificação Previsora com Perdas

No modelo clássico de codificação previsora com perdas acrescenta-se um quantizador. Ele mapeia o erro de previsão em um intervalo limitado de saídas que estabelece a quantidade de compressão e de distorção associada à codificação previsora com perdas.

1.7.2. Codificação por Transformada

As técnicas de codificação previsora operam diretamente nos *pixels* de uma imagem, sendo conhecidas por métodos de domínio espacial. Na codificação por transformada, uma transformada linear reversível é utilizada para mapear a imagem a um conjunto de coeficientes de transformada, que são quantizados e codificados. No caso da maioria das imagens naturais, um número significativo de

coeficientes tem pequenas magnitudes, podendo ser quantizados grosseiramente (ou descartados completamente) com pouca distorção na imagem.

A meta do processo de transformada é descorrelacionar os *pixels* de cada subimagem, ou compactar o máximo possível a informação em um número menor de coeficientes de transformada. O estágio de quantização elimina ou quantiza de modo mais grosseiro seletivamente os coeficientes que carregam menos informação. Esses coeficientes apresentam menor impacto na qualidade da subimagem reconstruída. O processo tem término pela codificação dos coeficientes quantizados. Qualquer um ou todos os passos de codificação por transformada podem ser adaptados para o conteúdo local da imagem, o que é chamado de *codificação adaptativa por transformada*, ou fixos para todas as sub imagens, o que é chamado de *codificação não adaptativa por transformada*.

Os sistemas de codificação por transformada com base nas transformadas de Karhunen-Loève (KLT), Fourier Discreta (DFT), Cosseno Discreta (DCT), Walsh-Hadamard (WHT) e muitas outras têm sido construídas e/ou estudadas extensivamente. A escolha de uma transformada em particular em uma dada aplicação depende da quantidade de erro de reconstrução que pode ser tolerado, bem como dos recursos computacionais disponíveis. A compressão é alcançada durante a quantização dos coeficientes da transformada (e não durante a transformada) (Gonzales & Woods, 2002).

1.8. Classificação dos Algoritmos de Compressão

Pode-se classificar os algoritmos de compressão de imagens em 3 tipos, ou seja: por transformação do modelo, por discretização e por codificação.

1.8.1. Compressão por Transformação do Modelo

O conjunto de dados pode ser reduzido no tamanho pela utilização de padrões (ou ordem) existente nos dados. Um padrão é uma forma de repetição ou redundância que existe entre os dados. Se um conjunto de dados não contém padrões, então ele é totalmente aleatório e por isso não pode ser compresso.

Esta transformação, conforme ilustrado na FIGURA 1.11, consiste em analisar a imagem (usando filtros, transformadas, etc.) de forma a se obter um modelo mais conveniente para sua especificação e utilizá-lo para codificar uma imagem.

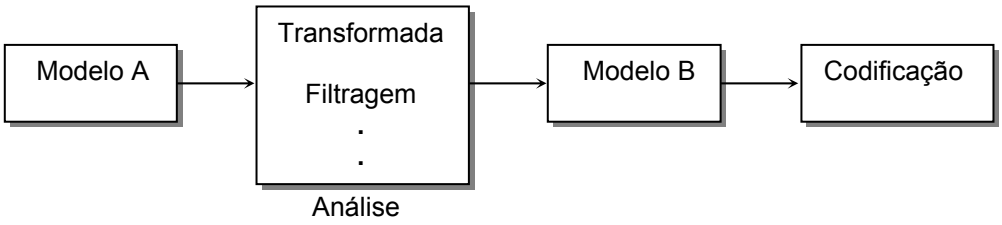


FIGURA 1.11 - Compressão por mudança de representação de uma imagem.
(Gomes & Velho, 1994, p.167)

Existem três principais técnicas usadas para explorar padrões em um conjunto de dados:

- redução do número de valores de dados no conjunto de dados por remoção de informação redundante;
- redução de algumas significâncias dos dados por descoberta de padrões que capturam a essência do conjunto de valores;
- redução da magnitude dos valores dos dados no conjunto de dados.

Quando o modelo final da imagem a ser codificado é contínuo, então sua codificação pode ser feita de modo independente da resolução. Isso ocorre, por exemplo, com a codificação usando a transformada fractal.

A compressão por aproximação é utilizada pelo método de compressão por transformação de modelo. Neste caso, calcula-se uma imagem *g* que seja uma aproximação da imagem original dada *f*, dentro de uma tolerância especificada. O problema de aproximar a função imagem, neste caso, é um problema de interpolação de dados esparsos.

1.8.2. Compressão por Discretização

O método de compressão por discretização utiliza-se de um modelo funcional da imagem e procura fazer a compressão reduzindo a informação no contra-domínio (resolução de cor), ou no domínio da imagem (resolução espacial da imagem). O processo de discretização de cor é conhecido pelo nome de quantização. A quantização de uma imagem permite uma redução do número de bits utilizados para armazenar o seu gamute de cores.

Os métodos mais simples de quantização por discretização fazem a quantização da representação espacial da imagem utilizando um método uniforme ou adaptativo. Esses métodos são também conhecidos por algoritmos de modulação de pulso ("*pulse code modulation*"). Quando a imagem possuir apenas informações de altas freqüências, utiliza-se este método para se fazer uma quantização com um número reduzido de bits.

1.8.3. Compressão por Codificação

Para os pesquisadores Smith e Rowe, o volume de dados a ser manipulado e a complexidade computacional de compressão e descompressão da imagem são os principais problemas encontrados quando pretende-se utilizar imagens compressas. Dados de vídeo são tipicamente transmitidos como uma seqüência de imagens compressas. Eles descreveram uma família de algoritmos que implementam operações na imagem digital compressa, rendendo performance de 50 para 100 vezes mais rápido do que algoritmos que devem fazer o processo de descompressão de imagens antes da aplicação e utilizar a compressão no resultado. Esta aceleração resulta da performance das operações diretamente nos dados compressos, visto que a compressão reduz o volume de dados significantemente. Juntamente com aceleração resultante do menor volume de dados, a maior parte da computação associada com compressão e descompressão é eliminada e o tráfego na memória é reduzido (Smith & Rowe, 1993).

Em 1997, Seales e colaboradores também exploraram idéia do processamento de dados sem descompressão buscando por padrões no domínio compresso sem impor um esquema de compressão de propósito especial. Isto permite integração com os tipos existentes de padrões para compressão de arquivos

de dados grandes incluindo parâmetros de perda da compressão no processo de busca. A perturbação da medida da comparação entre a imagem original e a compressa ocorre como um resultado da quantização dos dados transformados. O resultado mostrou um alto ganho para certas operações por escapar da descompressão (Seales *et al.*, 1997).

Uma técnica comum para a compressão da imagem é a transformada baseada na codificação. Um típico código de transformada trata os *pixels* da imagem como uma matriz de números. Uma transformada linear, como a transformada coseno discreta, é aplicada nesta matriz para criar uma nova matriz de coeficientes. Para recuperar a imagem original, aplica-se a transformação linear inversa. A transformação linear tem dois efeitos: primeiro, ela concentra a energia da imagem de tal modo que muitos dos coeficientes transformados são quase zero; segundo, ela decompõe espectralmente a imagem em freqüências altas e baixas. Visto que o sistema visual humano é menos receptivo para algumas freqüências que para outras, alguns coeficientes podem ser mais imperfeitamente aproximados que outros sem significante degradação da imagem. Uma maneira simples para utilizar-se posteriormente é através da quantização dos coeficientes, ou seja, truncando-se o bit de menor ordem de uma imagem (Smith & Rowe, 1993).

Outro método de compressão pode ser obtido fazendo-se uma subdivisão de cada linha da imagem em partições de intervalo e aproximando a imagem em cada intervalo por uma função constante. A imagem final fica assim aproximada, em cada linha, por uma função que é constante por partes.

A codificação dessa imagem constante por partes pode ser feita de modo eficiente para obter-se uma compressão. Ao invés de se armazenar a informação $f(x_i,y_i)$, em cada *pixel* (x_i,y_i), codifica-se, em cada linha da imagem o comprimento de cada intervalo onde a função é constante, juntamente com o valor da constante nesse intervalo. Esse método de codificação é chamado de codificação *RL*, ou codificação *Run Length*.

Na codificação *Run Length* obtém-se uma boa compressão da imagem se os intervalos da partição, onde a função imagem é constante, tiverem comprimentos suficientemente grandes, em relação à resolução horizontal da imagem.

A codificação *Run Length* decompõe o arquivo fonte em segmentos de símbolos idênticos, cada segmento é substituído por um par da forma (símbolo, número de ocorrências). Por exemplo, o arquivo fonte aaaabaaabbb é codificado como (a, 4), (b, 1), (a, 3), (b, 3). Este esquema de codificação trabalha bem quando existem muitos segmentos longos e trabalha fracamente quando existem muitos segmentos pequenos (Sahni *et al.*, 1997).

Em uma codificação uniforme o número de bits utilizado para codificar cada mensagem, que é constante, é chamado de taxa de bits. Sabe-se que diferentes níveis de quantização em uma imagem possuem diferentes probabilidades de ocorrência. Conseqüentemente, ter-se um conhecimento prévio da distribuição da probabilidade desses níveis pode-se utilizar uma codificação não uniforme da imagem de modo que os níveis mais freqüentes sejam codificados com um número menor de bits. Assim, consegue-se uma redução do número total de bits para se codificar uma imagem. Este tipo de codificação é chamado de codificação adaptativa.

1.9. Outros Padrões de Compressão

Os padrões de imagens são normalmente definidos por comitês que analisam o tipo de formato que se deseja criar (fotografias, desenhos, etc.), os tipos de algoritmos e variações disponíveis para o tipo de imagem desejada e, então, faz-se uma especificação de como será o algoritmo de codificação e de decodificação. Existem vários formatos disponíveis no mercado para os diversos tipos de imagem. Os mais utilizados na Internet são o GIF (*Graphic Interchange Format*), o JPEG (*Joint Photographic Experts Group*) e o MPEG (*Motion Photographic Experts Group*), devido ao pequeno tamanho do arquivo, e conseqüentemente alta velocidade para transmissão na rede.

Alguns esquemas de compressão combinam dois ou mais métodos para alcançar altas taxas de compressão as quais não seriam obtidas se os métodos fossem utilizados isoladamente.

Em muitos casos, a distinção entre esquemas de compressão com perda e sem perda é determinada pela exclusão ou inclusão do passo de quantização. Esquemas que incluem o passo de quantização são sempre com

perdas. Esquemas sem perda são limitados na soma de compressão que eles podem armazenar. Muitas aplicações, como as de vídeo, requerem altas taxas de compressão que somente os esquemas com perdas podem propiciar (Brown & Shepherd, 1954).

A padronização mais popular e completa de imagens estáticas e tons contínuos é chamada JPEG. Ela define três diferentes sistemas de codificação: (1) um sistema de codificação "linha-base", com perdas, que se baseia na DCT e é adequado à maioria das aplicações de compressão; (2) um sistema de codificação estendido para aplicações de maior precisão, maior compressão e de reconstrução progressiva; (3) um sistema de codificação independente sem perdas para a compressão reversível.

O padrão JPEG foi criado com a finalidade de estabelecer um formato padrão para compressão de imagens digitais, sendo um padrão ISO. Este padrão de compressão de imagem foi desenvolvido de forma especial para a compressão de fotografias. A principal característica deste algoritmo são as altas taxas de compressão que podem ser obtidas, sem que isso cause uma degradação de qualidade perceptível. Os estudos para o desenvolvimento do padrão foram iniciados no final da década de 70 e ao final da década de 80 já começaram a ser comercializados co-processadores gráficos em estações UNIX e Macintosh capazes de comprimir imagens em até 95%, sem perda de qualidade visível.

No padrão JPEG cada coeficiente não nulo é classificado segundo sua magnitude e segundo a quantidade de zeros que o precede. Uma vez classificados todos os coeficientes e levantada a função de densidade de probabilidade desta classificação, codifica-se a função usando o algoritmo de Huffman e acrescenta-se a informação da posição da posição relativa do coeficiente dentro de sua categoria e seu sinal. O artifício de classificação das magnitudes reduz enormemente a quantidade de entradas da função de densidade de probabilidade, que funciona como geratriz da tabela de códigos. O padrão JPEG utiliza uma tabela com 242 entradas, que resume a classificação dos coeficientes em 15 categorias de magnitude - cada categoria *i* agrupa coeficientes com magnitude entre 2^{i-1} e 2^{i} (exclusive), i=1,...,15 – sendo que cada uma delas pode ser precedida de 0 à 15 zeros, mais duas categorias especiais: a da seqüência de 16 zeros (zero precedido de 15 outros zeros) e do fim de bloco (representado pela entrada zero precedido de

nenhum zero). Este esquema de codificação está baseado na classificação da "corrida de zeros" (*zero run code*) (Oliveira *et al.*, 1998).

O formato GIF foi desenvolvido em 1987 com o objetivo de fornecer alta resolução a imagens na Internet. Isto foi necessário devido às baixas taxas de transmissão da rede naquela época (que ainda persiste em algumas regiões). Para então diminuir o tamanho das imagens enviadas na rede e conseqüentemente o tempo de espera do recebimento, um grupo na Compuserve[3] desenvolveu um esquema de compressão para mapas de bits.

O formato GIF, ao contrário do JPEG, é sem perdas e utiliza o algoritmo proprietário[4] LZW com índice variável para a codificação e decodificação do mapa de bits. O arquivo GIF consiste de duas partes, um cabeçalho com informações sobre a imagem e o conjunto de dados codificados. Quando uma imagem será mostrada, o cabeçalho é lido e o algoritmo é aplicado para decodificar a imagem e apresentá-la. Em 1989 a Compuserve acrescentou algumas novas características ao formato, como a transparência e forma animada do formato *(Aminated gif)*.

As imagens GIF são vastamente utilizadas na rede devido ao seu pequeno tamanho e as características espaciais, sendo então usadas para representar botões, ícones e qualquer outra imagem simples inserida na página. Um dos principais defeitos deste formato é que ele foi projetado para um número pequeno de cores (máximo 256). Quanto mais cores são aplicadas, é mais indicada a utilização de outros formatos específicos para fotos, como o JPEG. Outro problema é que como o algoritmo LZW é proprietário, podendo levar a problemas em futuras versões.

As padronizações mais comuns adotadas para a compressão e descompressão de imagens de seqüências de quadros são a H.261, MPEG I e MPEG II.

---

[3] Compuserve: A Compuserve é uma empresa que oferece soluções na área da informática.

[4] Algoritmo Proprietário: Um algoritmo proprietário é aquele desenvolvido e liberado para aplicações. Entretanto, seu código é protegido, ou seja, as atualizações realizadas nos algoritmos não são disponibilizadas.

A padronização H.261 foi projetada para aplicações de teleconferência por vídeo, em que vídeos são transmitidos por linhas com atraso de transmissão de menos de 150ms.

O padrão MPEG é um padrão ISO utilizado para compressão de seqüências de imagens (animação). A compressão de seqüências de imagens é de extrema relevância devido às suas diversas aplicações nas indústrias de vídeo, televisão e multimídia (Gomes & Velho, 1994).

O padrão MPEG é grande, complexo e utiliza basicamente o mesmo esquema de compressão do JPEG, exceto que este é designado para seqüências de vídeo e não imagens imóveis.

A MPEG I foi projetada para permitir taxas de bits maiores, bem como maiores codificações com qualidade.

A MPEG II suporta taxas de transferência de vídeo entre 5 e 10 Mbit/s, um intervalo apropriado para distribuição em TV a cabo e difusão por satélite de canal estreito.

Capítulo 2: Transformadas *Wavelets* e Espectro de Wiener no Processamento de Imagem Digital

2.1. Fundamentos

Geralmente assume-se que um sinal no domínio do tempo é um sinal original e um sinal que foi modificado por alguma transformação matemática como um sinal processado. A maioria dos sinais, na prática, estão no domínio do tempo na sua forma original, ou seja, qualquer sinal que é medido, é uma função de tempo. Em outras palavras, quando plota-se o sinal, um de seus eixos é o tempo (variável independente), e a outra (variável dependente) é usualmente a amplitude do sinal. Quando plota-se o sinal no domínio do tempo, obtém-se a representação da amplitude do sinal. Esta representação não é sempre a melhor representação do sinal para a maioria das aplicações relacionadas ao processamento de sinais. Em muitos casos, a representação não é explicitamente adequada. O espectro de freqüência de um sinal é basicamente formado pelos componentes de freqüência (componentes espectrais) daquele sinal.

Os fundamentos da teoria da análise em freqüência foram introduzidos por Jean Baptiste Joseph Fourier através da Transformada de Fourier (TF) (Fourier,1822; Preuss, 1982). Entretanto os pesquisadores voltaram-se da análise baseada na freqüência para a baseada em escala quando esta última demonstrou ser menos sensitiva a ruídos em diferentes escalas. As freqüências contidas num sinal podem ser encontradas através da Transformada de Fourier.

Para estudos dos sinais, a análise de Fourier é útil, pois a verificação das freqüências no espectro do sinal é de grande importância. Entretanto a análise de Fourier possui uma desvantagem, pois ao transformar-se o sinal para o domínio da freqüência, a informação de tempo é perdida, ou seja, ao analisar-se uma Transformada de Fourier é impossível dizer quando um evento detectado ocorreu. Se o sinal não muda muito no tempo, ou seja, se ele mantém-se aproximadamente estacionário, esta desvantagem não é relevante para a análise. Entretanto, muitos

sinais, contêm eventos não estacionários, de características transitórias. Com isso, a Transformada de Fourier não é utilizada por não se mostrar eficiente.

Na tentativa de suprir esta deficiência, a Transformada de Fourier foi adaptada em 1946 por Dennis Gabor (Gabor, 1946). No novo método, conhecido por *Short Time Fourier Transform* (STFT) o sinal é dividido em pequenos segmentos, onde estes segmentos (porções) do sinal podem ser assumidos serem estacionários. Para este propósito, uma função janela é aberta e analisa-se somente uma pequena seção do sinal a cada momento, numa técnica conhecida como janelamento do sinal, a qual mapeia o sinal para uma função bidimensional de tempo e freqüência. O principal problema desta técnica é que não se pode conhecer a exata representação de tempo-freqüência de um sinal, isto é, não se pode saber quais componentes espectrais existem em qual instante de tempo. O que se pode determinar são os intervalos de tempo nos quais certas bandas de freqüência existem, o que é parte do problema. As *Wavelets* representam o próximo passo, ou seja, a técnica de janelamento em regiões de tamanho variável (Burrus *et al.*, 1998).

A Transformada de Fourier, assim como a Transformada *Wavelet*, é uma transformada reversível, isto é, ela permite avançar e retornar entre o sinal original e o processado (transformado). Entretanto, somente um deles é avaliado num dado tempo. Isto é, nada da informação da freqüência é avaliada no domínio do tempo do sinal, e nada da informação de tempo é avaliada no sinal da Transformada de Fourier (Polikar, 1994).

A análise *Wavelet* permite o uso de intervalos de longo tempo quando se deseja informação mais precisa de baixa freqüência, e intervalos mais estreitos quando se deseja informação de alta freqüência. Altas freqüências são melhores determinadas no tempo, e baixas freqüências são melhores determinadas na freqüência. Basicamente, necessita-se da Transformada *Wavelet* para se analisar sinais não estacionários, ou seja, os quais a resposta da freqüência varia no tempo. A análise *Wavelet* pode ser de sucesso na descrição da relação tempo-frequência, mediante o uso de escalas de resolução. Considerando-se esta última característica, multiresolução e *Wavelets* estão fortemente relacionadas.

Segundo o pesquisador Manduca, os coeficientes *Wavelet* são parcialmente localizados em ambos espaço e freqüência bem como formam uma

representação multiescala da imagem com um fator de escala constante, levando à localizar sub-bandas de freqüências com igual largura na escala logarítmica. Eles também possuem alguma orientação específica. Por causa destas propriedades, as transformadas *Wavelets* fazem um excelente emprego da codificação eficiente de imagens do mundo real. Esta transformada é teoricamente melhor acompanhada para compressão de imagens que outras transformadas comuns, incluindo a Transformada Coseno Discreta (DCT), na qual a compressão em padrão JPEG é baseada. A útil proporção de compressão alcançada através da compressão baseada em *Wavelet* ou outra técnica com perda depende da aplicação específica e quanto da degradação da imagem pode ser tolerada (Manduca, 1995).

Um razoável investimento em pesquisa para o desenvolvimento de novas técnicas e métodos de análise de sinais que possam ser empregados em um amplo domínio de aplicações tem ocorrido nos últimos anos. Um exemplo disto é o trabalho desenvolvido por Marar e colaboradores que investiram esforços para a construção de uma família de funções polinomiais denominada Polinômios Potências de Sigmóides (PPS), que é uma técnica analítica para encontrar os coeficientes em PPS para a geração de uma família de funções *Wavelets* polinomiais. Duas aplicações foram apresentadas, uma relacionada às redes neurais artificiais e outra à compressão de imagens através de transformadas ortogonais para a utilização em técnicas de análise por multiresolução (Marar *et al.*, 1996).

Numerosas técnicas têm sido desenvolvidas para fazer a informação de imagem digital mais facilmente acessível e executar a análise automaticamente. Dentre essas técnicas, as transformadas *Wavelets* têm sido utilizadas não somente em imagens médicas, mas também por sinais e processamento de imagens em geral. Dentre essas aplicações encontram-se métodos para compressão (Wang, 2001).

As *Wavelets* têm sido utilizadas na compressão de dados e na análise de sinais, obtendo altas taxas de compressão com pouca perda de informação e também na eliminação de ruídos (Kunt *et al.*, 1987).

Outros campos aplicados que estão fazendo uso de *Wavelets* são: astronomia, acústica, engenharia nuclear, problemas de computação gráfica, neurofisiologia, música, ressonância magnética, identificação de vozes, ótica,

fractais, turbulência, previsão de terremotos, radar, visão humana, e aplicações em matemática pura tais como resolução de equações diferenciais parciais (Lima, 2004).

As famílias de funções $\psi_{a,b}$, geradas a partir das operações de dilatação e translação da mesma função ψ (*Wavelet* "mãe"), tornaram-se uma ferramenta muito importante nas várias áreas mencionadas. Essas famílias, denominadas "*Wavelets*", são definidas pela equação:

$$\psi_{a,b}(x) = |a|^{-\frac{1}{2}} \psi\left(\frac{x-b}{a}\right), \qquad a,b \in \Re,\ a \neq 0, \tag{2.1}$$

onde a representa o parâmetro de dilatação e b representa o parâmetro de translação.

A primeira menção registrada ao termo *Wavelet* ocorreu em 1909, na tese do matemático alemão Alfred Haar (Haar, 1910). Entretanto, as *Wavelets* de Haar ficaram no anonimato por muitos anos e, por um período muito longo, elas continuaram a ser a única base ortonormal de *Wavelets* conhecida. Em 1985, Stephane Mallat (Mallat, 1987) deu às *Wavelets* um grande impulso através de seu trabalho em processamento digital de imagens e, inspirado nos resultados de Mallat, Y. Meyer construiu a primeira *Wavelet* não trivial (suave). Uma das propriedades das *Wavelets* de Haar é que elas têm suportes compactos. Contudo, elas não são continuamente diferenciáveis, o que de certa forma limita as suas aplicações. Ao contrário das *Wavelets* de Haar, as *Wavelets* de Meyer são continuamente diferenciáveis. Elas não têm suportes compactos. Alguns anos mais tarde, Ingrid Daubechies (Daubechies, 1988) usou os trabalhos de Mallat para construir um conjunto de bases ortonormais de *Wavelets* suaves, com suportes compactos. Os trabalhos de Daubechies são os alicerces das aplicações atuais de *Wavelets.*

As transformadas *Wavelets* podem ser vistas como mecanismos para decompor ou quebrar sinais nas suas partes constituintes, permitindo analisar os dados em diferentes domínios de freqüências com a resolução de cada componente amarrada à sua escala. Além disso, na análise de *Wavelets*, pode-se utilizar funções que estão contidas em regiões finitas, tornando-as convenientes na aproximação de dados com descontinuidades.

Os algoritmos *Wavelets* processam dados em diferentes escalas ou resoluções e, independentemente da função de interesse ser uma imagem, uma curva ou uma superfície, as *Wavelets* oferecem uma técnica para a representação dos níveis de detalhes presentes. Elas constituem uma ferramenta matemática para decompor funções hierarquicamente, permitindo que uma função seja descrita em termos de uma forma grosseira, mais outra forma que apresenta detalhes que vão desde os menos delicados aos mais finos.

Como o sinal original ou função pode ser representado em termos de uma expansão em *Wavelets*, as operações com dados podem ser feitas usando os coeficientes de *Wavelets*. Se for possível escolher as *Wavelets* que melhor se adaptam aos dados, ou se forem truncados os coeficientes de *Wavelets* menores do que um valor previamente estabelecido, os dados serão esparsamente representados. Esta “codificação esparsa” faz das *Wavelets* uma ferramenta de interesse para o campo da compressão de dados.

Segundo Young, a teoria *Wavelet* é a matemática associada com a construção de um modelo para um sinal, sistema, ou processo com um conjunto de sinais especiais. Os sinais especiais são justamente pequenas *Waves* (ondas) ou *Wavelets*. Estas pequenas ondas ou *Wavelets* devem ser oscilatórias e apresentar amplitudes às quais ligeiramente decaem para zero em ambas as direções, ou seja, tanto positiva como negativa (Young, 1994).

A condição requerida para a oscilação leva à sinuosidade. A condição de ligeira caída é uma diminuição gradual ou operação de janelamento. Estas duas condições devem ser simultaneamente satisfeitas para a função ser uma pequena *Wave* ou *Wavelet*. De forma mais geral, conjuntos de *Wavelets* são empregados para aproximar um sinal e cada elemento no conjunto *Wavelet* é construído de alguma função.

A FIGURA 2.1 ilustra um exemplo de uma *Wavelet* clássica, conhecida por "*Wavelet* mãe de Morlet".

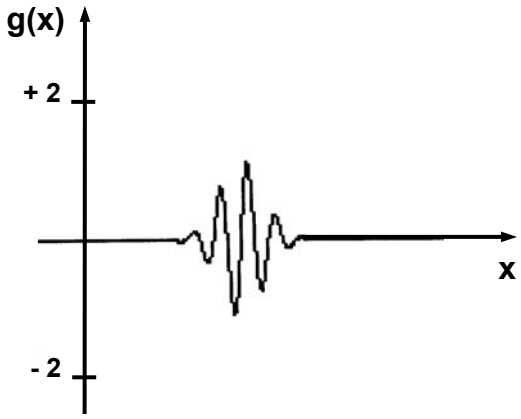


FIGURA 2.1 - *Wavelet* mãe de Morlet. (Young, 1994)

A FIGURA 2.2 ilustra uma sinusóide oscilante com amplitude $-\infty \leq t \leq \infty$ e, portanto, tendo energia infinita e com a *Wavelet* tendo sua energia finita concentrada em volta de um ponto (Burrus *et al.*, 1998).

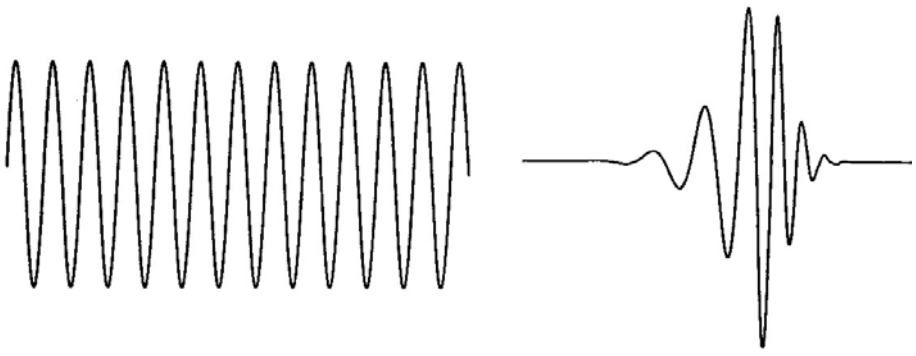


FIGURA 2.2 - Uma *Wave* (sinusóide) e uma *Wavelet.* (Young, 1994)

A transformada *Wavelet* é uma transformada que provê a representação tempo-freqüência, ou seja, é capaz de com a condição da informação de tempo e freqüência simultaneamente, fornecer uma representação tempo-freqüência do sinal.

O princípio mais geral na construção das *Wavelets* é o uso de dilatações e translações. As *Wavelets* mais usadas formam um sistema ortonormal[5] de funções com suportes compactos construído desta forma. Esta é a razão pela qual elas podem distinguir as características locais de um sinal em diferentes escalas e, por translações, elas cobrem toda a região na qual o sinal é estudado. Na análise de sinais não estacionários, a propriedade de localidade das *Wavelets* nos conduz às suas vantagens sobre a transformada de Fourier.

Deve-se distinguir duas versões diferentes da transformada de *Wavelet*, a contínua e a discreta. A primeira é análoga à Transformada de Fourier e é usada principalmente em análise e caracterização de detalhes de sinais. A segunda é análoga à Transformada de Fourier Discreta e é mais apropriada para a compressão de dados e reconstrução de sinais.

Um outro aspecto importante na análise dos sinais é quanto à avaliação da sua qualidade no que tange aos efeitos do processamento envolvido.

Existem diversos fatores que contribuem para a degradação de um sinal digital, dentre eles os mais comuns são ruído e perda por compressão. Daí a necessidade de avaliar o quanto desta degradação interfere no resultado final da imagem. Com este objetivo, a análise das variações *rms* (*root mean square*) no valor do *pixel* tem sido utilizada para descrever o ruído em imagens. Contudo, este método indica somente a magnitude do ruído, não fornecendo informações sobre sua característica. Por este motivo, as técnicas de análise espectral têm ganho espaço, uma vez que possibilitam obter informações sobre a magnitude e característica do ruído.

---

[5] Base Ortonormal: É aquela em que todos os vetores que definem os eixos têm tamanho 1 e, além disso, são ortogonais entre si.

2.2. Transformada Discreta de *Wavelets*

Denota-se por $L^p(\Re)$ o espaço das funções, $f(x)$, definidas em $\Re$, assumindo valores em C (ou $\Re$), tais que $\|f(x)\|_p \equiv \left(\int_{-\infty}^{\infty} |f(x)|^p dx\right)^{\frac{1}{p}} < \infty$. Para caso particular quando $p=2$, define-se o produto interno de duas funções $f(x)$ e $g(x)$ como $\langle f(x), g(x)\rangle = \int_{-\infty}^{\infty} f(x)\overline{g(x)}dx$, onde a barra representa o complexo conjugado. Diz-se que $f(x)$ e $g(x)$ são ortogonais se $\langle f(x), g(x)\rangle = 0$. Uma função $f(x)$ tem suporte compacto, se existe um intervalo fechado e limitado, fora do qual $f(x)=0$.

Uma *Wavelet* é uma função $\psi(x) \in L^1(\Re) \cap L^2(\Re)$, tal que a família de funções expressas pela equação (2.2) seja uma base ortonormal para $L^2(\Re)$, ou seja:

$$\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k) \tag{2.2}$$

onde j e k são inteiros arbitrários.

Da definição acima, se $\psi(x)$ é uma *Wavelet*, então, $\psi_{j,k}(x)$ também o será para qualquer $j,k \in \mathrm{Z}$.

2.3. A Wavelet de Haar

A *Wavelet* de Haar é definida como:

$$\psi(x) = \begin{cases} 1, \text{ se } x \in \left[0, \frac{1}{2}\right) \\ -1, \text{ se } x \in \left[\frac{1}{2}, 1\right) \\ 0, \textit{ caso contrário}. \end{cases} \tag{2.3}$$

A *Wavelet* Haar não tem boa localização tempo-freqüência e a sua transformada de Fourier $\hat{\psi}(\omega)$, decai muito lentamente, se comportando como $|\omega|^{-1}$ quando $\omega \to \infty$.

A família de *Wavelet* Haar constitui uma base ortonormal para $L^2(\Re)$. A FIGURA 2.3 ilustra a representação gráfica para a *Wavelet* de Haar, conforme fundamentada pela equação (2.3).

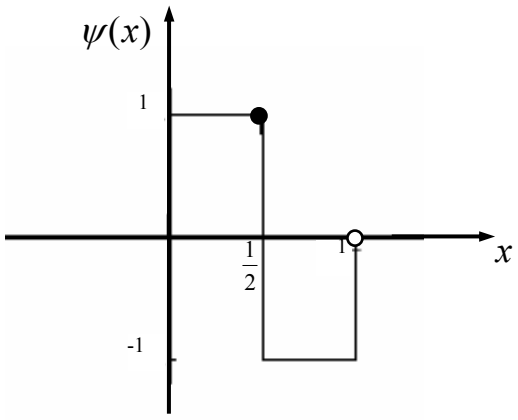


FIGURA 2.3 - A *Wavelet* de Haar, $\psi(x)$. (Haar, 1910)

2.4. Análise Multiresolução

A análise multiresolução permite que um sinal seja analisado em diferentes freqüências com diferentes resoluções. Ela é utilizada quando há uma boa resolução de tempo e pouca resolução em freqüência nas altas freqüências, bem como quando há boa resolução de freqüência e pouca resolução de tempo nas baixas freqüências. Esta abordagem faz sentido especialmente quando o sinal em análise possui altas componentes de freqüência para baixas durações e altas componentes de freqüência para longas durações (Polikar, 1994).

A simplicidade do esquema *Wavelet* despertou interesse nos matemáticos levando-os a uma alternativa para problemas não solucionáveis através da Análise de Fourier. Isto desencadeou a descoberta de *Wavelets,* a qual forma base ortonormal para quadrado integrável[6] e outras funções definidas por Meyer, Daubechies, Battle, Lemarié e outros. A formalização estabelecida por Mallat e Meyer criou uma estrutura para expansões *Wavelet,* conhecida por Análise Multiresolução, e estabeleceu ligações com os métodos usados em outras áreas (Vetterli & Kovacevic,1995).

Há muito tempo que em processamento de imagens e sinais, tem-se reconhecido que esquemas de decomposição de sinal através da multiresolução proporcionam convenientes e efetivos caminhos para processar informação. Pirâmides, *Wavelets*, bancos de filtros, granulometrização e esqueletização são algumas das ferramentas para construção de esquemas de decomposição de sinal de multiresolução. Embora tais ferramentas sejam construídas em diferentes paradigmas, reconhece-se que elas são partes da mesma teoria. Uma maneira popular de obter um esquema de decomposição de sinal de multiresolução é uniformizar um dado sinal, através de um filtro passa baixa linear, com o objetivo de remover altas freqüências, e sub amostrar o resultado em ordem de obter uma versão de escala reduzida do sinal original. Pela repetição deste processo, uma coleção de sinais na escala decrementada é então produzida. Estes sinais, empilhados no topo de cada outro, formam um esquema de decomposição de sinal básico, conhecido como uma pirâmide de multiresolução. Uma coleção de detalhes de sinais é também construída pela subtração de cada nível da pirâmide na versão interpolada do próximo nível inferior. Do ponto de vista da freqüência, os sinais de

---

[6] Quadrado Integrável: O espaço Lebesgue $L^2(\Re)$ das funções mensuráveis é o espaço das funções cujos quadrados são integráveis, isto é, $f(x) \in L^2(\Re)$ se e somente se $\|f(x)\|^2 = \int_{-\infty}^{+\infty} |f(x)|^2 dx < \infty$. Este conceito é importante, pois permite considerar espaços de funções normalizadas, e portanto, metrizáveis. A análise de multiresolução, chave para o estudo das *Wavelets*, fundamenta-se em subespaços do espaço de Lebesgue. Correspondente ao espaço de Lebesgue, no caso discreto, tem-se o conceito de espaço l^2 de seqüências complexas de quadrado integrável.

detalhes resultantes formam uma decomposição do sinal em termos de cópias de filtragem passa alta do sinal original. Pode-se mostrar que o sinal original pode ser unicamente reconstruído através de sinais de detalhes. Entretanto, os sinais de detalhes provêm uma representação do sinal de multiresolução que garante a perfeita reconstrução. O melhor exemplo deste esquema é a Pirâmide Laplaciana (Goutsias e Heijmans, 2000).

A teoria da análise da multiresolução foi originalmente desenvolvida por Mallat para representar funções definidas sobre $\Re^n$. A análise da multiresolução provê uma eficiente ferramenta para representação de figuras e análise de características em múltiplos níveis de detalhes. A idéia básica que conecta a análise multiresolução e a transformada *Wavelet* discreta é fundamentada em se representar uma função complicada com outra comparativamente simples de baixa resolução, entretanto com uma coleção de perturbações, com o aumento do nível de detalhes, necessário para recuperar a função original (DeRose *et al.*, 1993).

Técnicas de multiresolução são naturalmente aplicadas em imagens, onde noções como resolução e escala são muito intuitivas. Estas técnicas também têm sido usadas na visão computacional para trabalhos como reconhecimento e estimação de movimento, bem como na compressão de imagens com pirâmides e codificação com sub-bandas. Uma importante característica do seu uso nas técnicas de compressão de imagem é sua propriedade de aproximação sucessiva, ou seja, quanto mais altas forem as freqüências, altas resoluções de imagens serão obtidas. Nota-se que a aproximação sucessiva de multiresolução corresponde ao sistema visual humano com emprego de técnicas de multiresolução em termos de qualidade perceptível (Vetterli & Kovacevic, 1995).

Assim como na compressão de imagem, a aparência da imagem é a motivação fundamental para o processamento da multiresolução. Se os objetos na imagem são pequenos nos tamanhos ou baixos no contraste, utiliza-se altas resoluções para sua análise; se os objetos são grandes nos tamanhos ou altos no contraste, uma visão mais superficial é suficiente.

As *Wavelets* nos permitem recuperar os detalhes que seriam perdidos com a diminuição de resolução quando se passa de uma escala para a escala

seguinte, ou seja, elas medem as flutuações entre duas escalas consecutivas (Lima, 2004).

A uma determinada resolução j-1 de um sinal, as funções de escalonamento $\phi_{j-1,k}(t)$ formam uma base para um conjunto de sinais. Os detalhes são representados pelas *Wavelets* $\psi_{j-1,k}(t)$. O sinal somado aos detalhes combina-se em uma multiresolução a um nível mais fino j. As médias provêm das funções de escalonamento e os detalhes provêm das *Wavelets*, ou seja:

$$\underset{\text{(médias locais)}}{\text{sinal ao nível j-1}} + \underset{\text{(diferenças locais)}}{\text{detalhes ao nível j-1}} = \text{sinal ao nível j} \qquad (2.4)$$

Tem-se que o teorema da multiresolução é dado por:

$$\sum_k a_{jk}\phi_{jk}(t) = \sum_k a_{j-1,k}\phi_{j-1,k}(t) + \sum_k b_{j-1,k}\psi_{j-1,k}(t) \qquad (2.5)$$

onde o lado esquerdo da equação é a representação do sinal como uma combinação linear de funções de escalonamento $\phi_{jk}(t)$ ao nível j. No lado direito, tem-se a soma de combinações lineares, sendo uma de sinal ao nível j-1, usando funções de escalonamento $\phi_{j-1,k}(t)$ e outra de detalhes, também ao nível j-1, usando *Wavelets* $\psi_{j-1,k}(t)$.

Desta forma, tem-se a multiresolução de um sinal. Aplicando-a a todos os sinais, tem-se a multiresolução para espaços de funções. Denota-se por V_j o espaço de todas as combinações lineares dos $\phi_{jk}(t)$, ao nível j. Da mesma maneira, V_{j-1} denota o espaço de todas as combinações lineares dos $\phi_{j-1,k}(t)$, ao nível j-1. Denota-se por W_{j-1} o espaço de todas as combinações lineares dos $\psi_{j-1,k}(t)$ ao nível

j-1. Desta maneira, $V_j = V_{j-1} + W_{j-1}$. Tem-se também que V_{j-1} é perpendicular à W_{j-1}. Portanto, $V_{j-1} \cap W_{j-1} = \{0\}$, onde 0 é a função constante 0, e desta maneira verifica-se a soma direta (e ortogonal) $V_j = V_{j-1} \oplus W_{j-1}$, a qual é a declaração chave da multiresolução.

O uso repetido permite expressar um espaço de sinais utilizando os diversos níveis de detalhes, como na equação 2.6:

$$V_3 = V_2 \oplus W_2 = V_1 \oplus W_1 \oplus W_2 = V_0 \oplus W_0 \oplus W_1 \oplus W_2 \qquad (2.6)$$

onde no lado esquerdo tem-se o espaço de todas as funções representáveis em degraus de comprimento 1/8. No lado direito o mesmo espaço representado através de um espaço de sinal uniforme de comprimento 1 (correspondente à média global) e através dos espaços de detalhes a diferentes resoluções.

A análise clássica da multiresolução é baseada no conhecimento de um conjunto aninhado de espaços funcionais nos quais as sucessivas aproximações de uma dada função convergem para aquela função, e pode ser eficientemente computada. Bonneau e colaboradores propuseram em seu trabalho permitir a análise multiresolução sempre que os espaços funcionais não estão aninhados, assim como eles inibiram a propriedade onde sucessivas aproximações convergem para uma dada função. Em sua proposta introduziram uma nova análise multiresolução do grande conjunto de dados definido na grade uniforme e para isso foram utilizadas duas bases *Wavelets* conhecidas, a Haar e a base linear. Surgiu desta idéia uma família de análise multiresolução a qual é uma mistura da multiresolução Haar e a linear (Bonneau *et al.*, 1996). Também, utilizam-se as *Wavelets* numa série de expansão de sinais ou funções do mesmo modo que uma série de Fourier utiliza a *Wave* ou sinusóide para representar um sinal ou função. Os sinais são funções de uma variável contínua, as quais sempre representam tempo ou distância. Desta expansão em série, desenvolveu-se uma versão similar em

tempo discreto para a Transformada Discreta de Fourier, onde o sinal é representado através de uma *string* de números onde os números podem ser exemplos de um sinal, exemplos de outras *strings* de números, ou produto interno de um sinal com algum conjunto de expansão.

A expansão *Wavelet* estabelecida não é única. Existem diferentes sistemas *Wavelets* que podem ser usados eficientemente os quais seguem, em geral, três características básicas:

1. um sistema *Wavelet* é um conjunto de blocos construtores para construir ou representar um sinal ou função. Ele é um conjunto de expansão bi dimensional (usualmente uma base) para algumas classes de um (ou superior) sinal dimensional;
2. a expansão *Wavelet* fornece a localização tempo-freqüência do sinal. Isto significa que a maioria da energia do sinal é bem representada através dos coeficientes de expansão $a_{j,k}$;
3. o cálculo dos coeficientes do sinal pode ser feito eficientemente.

Todos os sistemas *Wavelet* possuem virtualmente estas características gerais. Entretanto, existem três características adicionais que são mais específicas para expansões de *Wavelets*, ou seja:

1. toda primeira geração de *Wavelet* conhecida foi gerada a partir de uma única função de escalonamento ou *Wavelet* por simples escalonamento e translação. A parametrização bi-dimensional é encontrada através da função *Wavelet* mãe;
2. quase todos os sistemas *Wavelets* satisfazem as condições de multiresolução;
3. a maior parte dos coeficientes de resolução podem ser calculados através de coeficientes de resolução por um algoritmo em estrutura de árvore conhecido por um banco de filtro.

2.5. Decomposição Unidimensional usando Transformada *Wavelet* Haar

Em 1909, Haar descreveu bases ortonormais, as quais são uma evolução da base Haar (Wang, 2001).

Por exemplo, para uma dada imagem unidimensional formada por oito *pixels* com os valores:

(7 5 2 8 9 5 3 3)

pode-se obter, por exemplo, uma imagem com menor resolução através da aplicação de uma média tomada par a par sobre a imagem anterior. Tem-se assim, a seguinte imagem resultante:

(6 5 7 3)

Pode-se observar que uma excessiva perda de informações ocorre nesta operação de média. Essas informações perdidas são, entretanto, fundamentais para uma posterior recuperação da imagem original. Para se recuperar as informações, faz-se necessário o armazenamento dos coeficientes de detalhes originários do processo de média.

Os coeficientes de detalhes podem ser obtidos armazenando-se os valores das diferenças entre o valor médio e os valores das intensidades nos *pixels* envolvidos nessa operação.

Toma-se, assim, o primeiro par do exemplo dado acima, ou seja, os valores 7 e 5. O valor da média é 6. Sabe-se que 6+1=7 e 6-1=5. Portanto, o coeficiente de detalhe necessário para a recuperação dos dados originais (primeiro par) tem o valor 1.

Para o segundo par, analogamente, pode-se chegar ao coeficiente de detalhe (-3), pois 5+(-3)=2 e 5-(-3)=8. Através desse processo, obtém-se um conjunto de informações, conforme ilustra a tabela 2.1:

Resolução	Médias	Coeficientes de detalhes
8	7 5 2 8 9 5 3 3	
4	6 5 7 3	1 -3 2 0
2	5,5 5	0,5 2
1	5,25	0,25

TABELA 2.1 – Processo de decomposição por *Wavelets* em várias resoluções

A Transformada *Wavelet* – Decomposição *Wavelet* – da imagem (7 5 2 8 9 5 3 3) baseada na *Wavelet* Haar é dada pela seqüência de números reais: (5,25 0,25 0,5 2 1 -3 2 0). Essa seqüência é composta pelo último elemento do processo de médias (5,25) e pela seqüência dos coeficientes de detalhes (0,25 0,5 2 1 -3 2 e 0).

Este processo, também chamado de banco de filtragens (*filter bank*) pode ser generalizado para outros tipos de *Wavelets*.

Através desse mesmo processo é possível codificar o *quadrature mirror filters* (QMF) do processamento de sinal da engenharia da comunicação. Esses sinais da engenharia da comunicação eram vistos quase sem relação com os resultados das *wavelets* – os sinais eram seqüências exatas, o tempo é discreto, enquanto as *wavelets* pertencem à $L^2(\Re)$ e os problemas na análise matemática que são altamente não discretos. Filtros duplos, ou mais genericamente, *subband filters*, foram concebidos na engenharia bem antes das *wavelets* na matemática das recentes décadas. Esses filtros duplos da engenharia têm sido usados na tecnologia, ainda mais amplamente para o contexto de *quadrature mirror filters.* Os *QMF´s* produzem perfeita reconstrução dos sinais que são passados através dos filtros dos algoritmos de síntese (composição) e análise (decomposição) do processamento de sinais (Walnut, 2002).

2.6. Compressão de Imagens Digitais Utilizando a Transformada *Wavelet*

As *Wavelets* encontraram na compressão de imagens uma das principais áreas de aplicação. Isto porque uma imagem original pode ser representada por uma combinação linear de funções base *Wavelets*, similar à análise de Fourier, a compressão pode ser representada pelos coeficientes *Wavelets*.

Os primeiros esforços voltados para a compressão de imagens levaram a uma superabundância de métodos, reduzindo o número de bits. Algumas alternativas guiadas pela teoria da informação foram propostas e resultados na razão até perto de 10:1 foram atingidos. A necessidade do aumento desta proporção tornou-se a base de muitas pesquisas. Sabe-se que os *pixels* são conseqüências de técnicas limitantes na transformação de uma cena análoga em dado digital e que qualquer esforço em descrever imagens em termos de entidades físicas mais próximas, como contornos ou regiões deveriam aumentar a compressão. Baseado nestes fatos surgiu uma segunda geração de métodos de compressão. Estes métodos tentam descrever uma imagem em termos de seu contorno e textura. Todo objeto em uma cena natural é caracterizado por suas bordas (contornos) cercando sua superfície (textura). Um problema é que a informação é considerada somente numa região muito limitada, não existindo caminho para distinguir entre o contorno correspondente das bordas do objeto para aquelas as quais não fazem parte. Os estudos mostraram que utilizando a codificação baseada na decomposição direcional alcançou-se uma compressão na ordem de 57:1, conforme mostra a FIGURA 2.4 (a), entretanto o reconhecimento de pessoas e detalhes pequenos ficou afetado. Outros modelos como o de Aproximações Polinomiais não obtiveram resultados muitos satisfatórios, com a compressão alcançando taxas de compressão de 32:1. Utilizando-se o modelo *Wavelet* para contorno de bordas, os resultados da compressão chegaram à 60:1 (FIGURA 2.4 (b)) e em outro caso à 120:1 (FIGURA 2.5 (a)), sendo que a diferença da qualidade mostrada em ambas foi pequena. Apesar de gerar uma compressão duas vezes maior, a qualidade não foi deteriorada em proporção. Entretanto, muitos detalhes são perdidos na alta compressão (Kunt *et al.*, 1987).

(a) (b) (c)

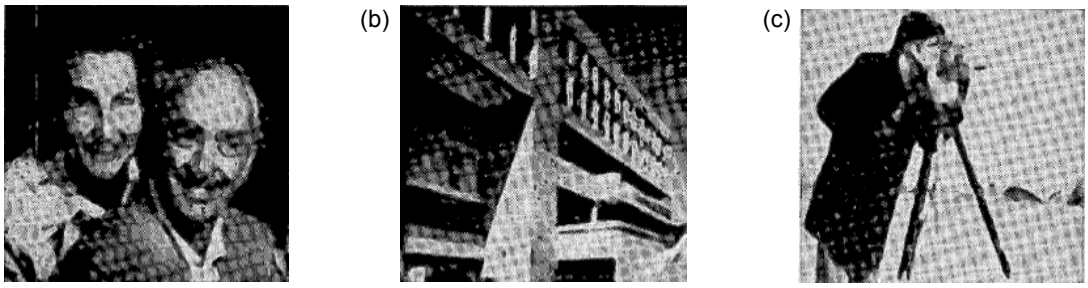


FIGURA 2.4 – Codificação baseado na decomposição usando o modelo *Wavelet*. As taxas de compressão são: (a) 57:1, (b) 59:1 e (c) 49:1. (Kunt *et al.*, 1987, p.1332)

(a) (b) (c)

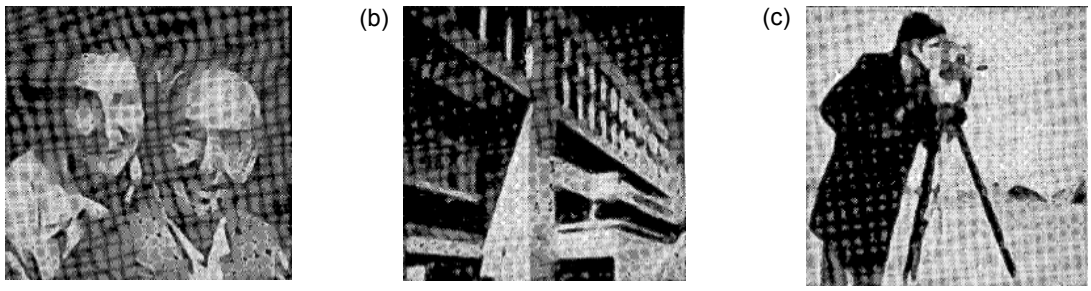


FIGURA 2.5 – Codificação baseado na decomposição usando o modelo *Wavelet*. As taxas de compressão são: (a) 118:1, (b) 87:1 e (c) 84:1. (Kunt *et al.*, 1987, p.1332)

A transformada *Wavelet* tem sido amplamente usada em compressão de imagens. Entretanto, seu uso limitava-se a aplicações de compressão com perda e este aspecto, segundo Sheng e colaboradores, deve-se ao fato de que a maioria das transformadas *Wavelet* produz coeficientes de ponto flutuante os quais não são bem processados para aplicações de codificação sem perda. Com a utilização da transformada *Wavelet* de inteiro reversível, ou seja, a transformada *Wavelet* que transforma inteiros para inteiros e permite perfeita reconstrução do sinal original, existe interesse na utilização da transformada *Wavelet* para codificação da imagem sem perda. Com a utilização do inteiro reversível da transformada *Wavelet* tem-se como principal vantagem, através do uso de técnicas apropriadas, um *bit stream*[7] embutido que pode ser gerado, ou seja, o decodificador pode extrair uma versão com perda da imagem, possivelmente com a redução da resolução, na desejada medida do *bit stream*, e continua decodificar nas mais altas medidas até a imagem

[7] Bit Stream: é uma seqüência de bits. Podem ser representados por letras, dígitos ou outro símbolo que é utilizado como parte da organização, controle ou representação de dados.

ser perfeitamente reconstruída. Esta medida de escalabilidade é valiosa em muitas aplicações. Através da integração da compressão com e sem perda em um estilo natural, um único método de compressão de imagem provê excelente performance sem perda bem como suporta muitas aplicações que requerem habilidade para recuperar exatamente a imagem original (Sheng e colaboradores, 1998).

A compressão de imagens pode ser realizada através de métodos de codificação, compressão do domínio espacial e métodos de compressão da transformada do domínio. As combinações desses métodos também têm sido exploradas. A compressão baseada na Transformada Coseno Discreta (DCT) e a Transformada *Wavelet* são dois exemplos de métodos de transformada do domínio (Sahni *et al.*, 1997).

Segundo Liao Supeng e colaboradores, o processo de compressão de codificação da imagem é uma técnica muito importante na área de processamento da informação. No esquema de compressão de imagem proposto pelos pesquisadores neste trabalho, utilizam-se as propriedades de análise *Wavelet* e visão humana. A Transformada *Wavelet*, a qual pode dar representações de multiresolução, é muito efetiva para análise do conteúdo da informação. Apesar dos resultados da pesquisa apresentarem altas taxas de compressão, nota-se nas imagens boa reconstrução das qualidades avaliadas em ambas as medidas objetivas e efeitos visuais subjetivos. O esquema de compressão foi dividido em três etapas: decomposição, quantização e codificação. O processo de decomposição utiliza a Transformada *Wavelet*. A decomposição da resolução de uma imagem digital com a Transformada *Wavelet* produz um conjunto de sub imagens, o qual é conhecido como uma representação *Wavelet* ortogonal em duas dimensões. A quantização é o passo chave da compressão de imagens, entretanto alguns quantizadores são ótimos no significado da razão do erro quadrático, mas não necessariamente fornece o melhor efeito visual (Supeng *et al.*, 1996).

No sistema RGB, uma imagem digital é caracterizada atribuindo-se a cada *pixel*, um vetor com três componentes, cada uma das quais representando as intensidades das cores vermelho (R), verde (G) e azul (B), respectivamente. Os valores de cada componente é um número inteiro entre 0 e 255. No caso de uma imagem em tons de cinza, as três componentes são iguais e a imagem é

completamente caracterizada pelo escalar, que é o valor comum das três intensidades (Lima, 2004).

Dada uma função $f(x)$ em V_0, pode-se escrevê-la de maneira única como:

$$f(x) = \sum_k a_{0,k}\phi(x-k) \tag{2.7}$$

e através da decomposição dos sub-espaços, pode-se reescrevê-la como:

$$f(x) = f_J(x) + \sum_{j=1}^{J} \Delta_j(x) \tag{2.8}$$

onde $f_J(x) \in V_J$ e $\Delta_J \in W_J$, portanto, $f_J(x) = \sum_k a_{J,k}\phi_{J,k}$ e $\Delta_j(x) = \sum_k d_{j,k}\psi_{j,k}(x)$, onde os coeficientes a_J e d_j são calculados recursivamente a partir de a_0.

Dada uma imagem digital unidimensional, em tons de cinza, com 2^l *pixels*, onde l é um inteiro não negativo, sejam $\{a_{0,k}\}k = 0,....,2^l - 1$ os valores de cada um de seus *pixels*. Associa-se a esta imagem a seguinte função em V_0: $f(x) = \sum_k a_{0,k}\phi(x-k)$. Com tal definição, os algoritmos acima permitem calcular os coeficientes *Wavelets* de $f(x)$, $d_{j,k}$, $j = 1,....,J = 2^l$ e os coeficientes $a_{J,k}$. Ao projetar $f(x)$ sobre um dos subespaços V_j, consegue-se obter uma versão de baixa resolução de $f(x)$, reduz-se a resolução por um fator de 2^j e ao passar de V_j para V_{j+1} perde-se a resolução por um fator de 2 e os detalhes que seriam perdidos são representados por Δ_j. Desta maneira, ao decompor $f(x)$ de acordo com a equação 2.8, obtêm-se uma versão da imagem onde todos os *pixels* possuem o mesmo valor, que é a "média" de todos os *pixels*, mais os detalhes correspondentes às escalas intermediárias.

Sob o ponto de vista computacional, começa-se com uma imagem unidimensional com 2^l *pixels*, armazenada num vetor A, com 2^l posições. No primeiro passo passa-se de V_0 para V_1, usando-se as relações de recursão, gera-se

2^{l-1} coeficientes $\{a_{1,k}\}_{k=0}^{2^{l-1}-1}$ e 2^{l-1} coeficientes $\{d_{1,k}\}_{k=0}^{2^{l-1}-1}$; faz-se $A[k]=a_{1,k}$, para $k=0,...,2^{l-1}-1$ e $A[2^{l-1}+k]=d_{1,k}$, para $k=0,...,2^{k-l}-1$. Com isso, tem-se uma imagem de baixa resolução, $f_1(x)\in V_1\subset V_0$, com resolução diminuída por fator de dois, armazenada nas primeiras 2^{l-1} posições de A, nas posições seguintes estão os detalhes, correspondentes à passagem de V_0 para V_1, que é a projeção de $f(x)$ sobre W_1. Pode-se repetir este processo a $f_1(x)$ e, partindo-se de seus coeficientes $\{a_{1,k}\}_{k=0}^{2^{l-1}-1}$ e das relações de recursão, encontra-se os coeficientes $\{a_{2,k}\}_{k=0}^{2^{l-1}-2}$; estes serão armazenados nas 2^{l-1} primeiras posições de A, sendo que $A[k]=a_{2,k}$ para $k=0,...,2^{l-2}-1$ e $A[2^{l-2}+k]=d_{2,k}$ para $k=0,...,2^{k-2}-1$. Nas primeiras 2^{l-2} posições de A tem-se a versão de $f(x)$, $f_2(x)\in V_2\subset V_0$, onde a resolução foi diminuída por um fator de 2, em relação à versão anterior, nas posições seguintes os detalhes correspondem às escalas intermediárias, em ordem decrescente. Repetindo-se o processo l vezes, na l-ésima vez, a partir da versão $f_{l-1}(x)\in V_{l-1}\subset V_0$ de $f(x)$ e das relações de recursão, obtêm-se $a_{l,0}$ e $d_{l,0}$ que serão armazenados em $A[0]$ e $A[1]$, respectivamente. Com isso, passa-se de um vetor A para um vetor CW, sendo que neste temos os coeficientes de *Wavelets* de $f(x)$. Cada passo no processo acima pode ser implementado por uma matriz invertível (ortogonal, nos exemplos considerados), o que significa que a partir do vetor CW pode-se obter o vetor A e, portanto, reconstruir a imagem. No caso de uma imagem colorida, os coeficientes acima $a_{j,k}$ e $d_{j,k}$ serão vetores com três componentes e aplica-se o procedimento acima a cada componente, separadamente.

Dada uma imagem com 2^l x 2^l *pixels*, a qual pode ser armazenada numa matriz quadrada A, de ordem, 2^l, neste caso, trata-se cada linha ou coluna como se fosse uma imagem unidimensional, aplicando-se o processo acima no cálculo dos coeficientes de *Wavelets*.

O processo de decomposição de uma imagem pode ocorrer através da Decomposição Padrão ou da Decomposição Não-Padrão. Na Decomposição Padrão, no primeiro passo, associa-se a cada linha da imagem uma função em V_0, em seguida, para cada linha calcula-se os coeficientes *Wavelets*. A seguir, associa-

se a cada uma das colunas transformadas no processo anterior uma função em V_0 e aplica-se o processo descrito anteriormente às mesmas e obtemos os respectivos coeficientes *Wavelets*. Note que em $A[0,0]$ está a média dos *pixels*, nas demais posições estão armazenados os coeficientes *Wavelets*, propriamente ditos. Como estas operações são invertíveis, pode-se inverter o processo de decomposição e reconstruir a imagem inicial. A FIGURA 2.6 ilustra a Decomposição Padrão.

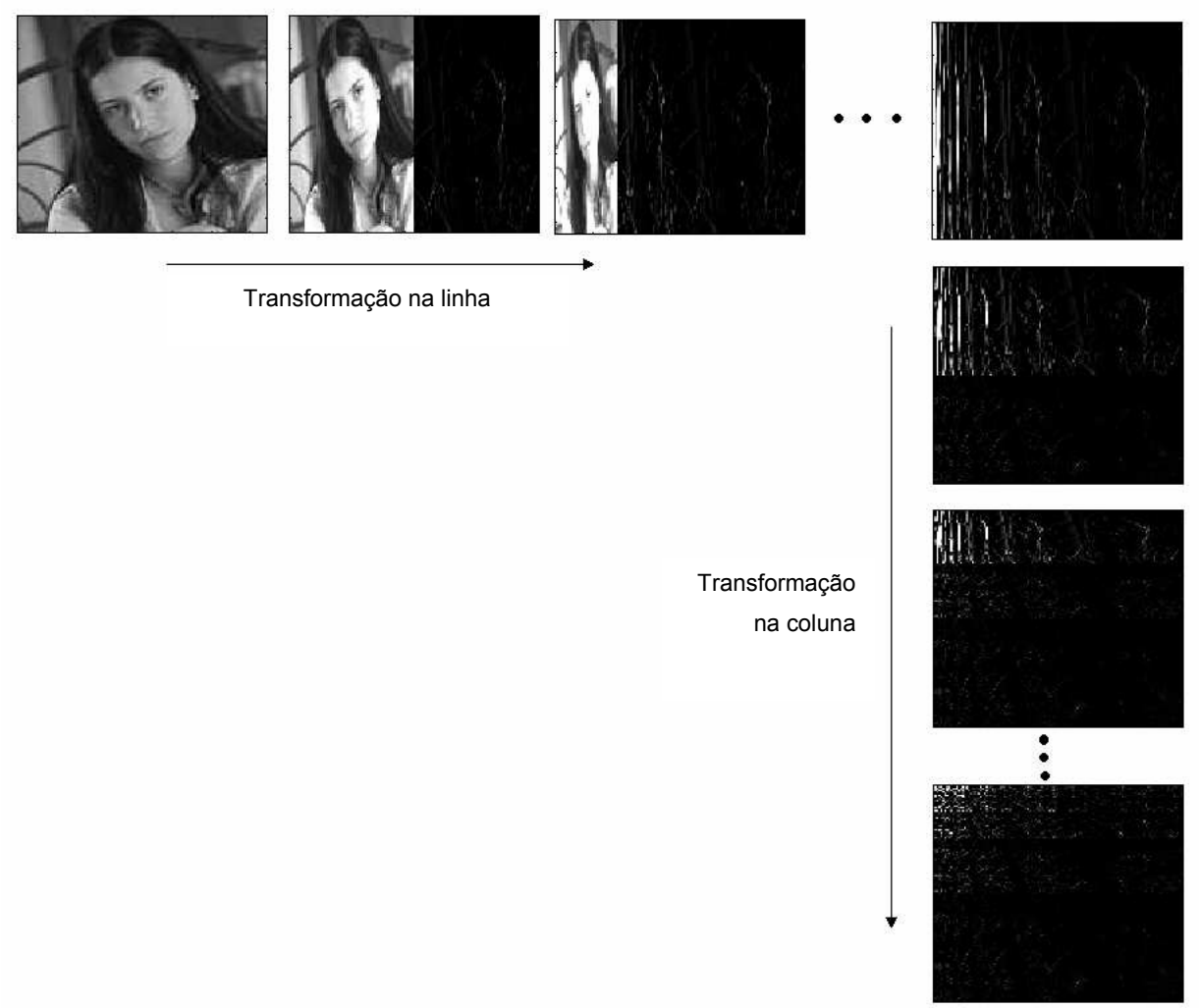


FIGURA 2.6 – Decomposição Padrão de uma imagem utilizando *Wavelets.* (Lima, 2004, p.35)

Na Decomposição Não-Padrão, aplica-se operações em linhas e colunas alternadamente. Associa-se à cada linha da imagem uma função em V_0; a seguir, decompõe-se cada linha aplicando-se apenas um passo, o processo descrito na passagem de V_0 para $V_1 \oplus W_1$, depois, trata-se cada coluna resultante como se fosse uma função em V_0 e as mesmas são decompostas, como feito no passo

anterior, onde as linhas foram substituídas pelas colunas. No passo seguinte, toma-se a versão de baixa resolução da imagem original a qual está armazenada numa submatriz, de A, restrita aos $A[i,j]$ com $i,j = 0,....,2^{l-1}-1$ (nas demais posições têm-se os coeficientes *Wavelets*). A seguir, repete-se o processo à versão de baixa resolução da imagem obtida no passo anterior e tem-se uma submatriz de A, restrita a $A_1[i,j]$, com $i,j = 0,....,2^{l-2}-1$, na qual está uma nova versão de baixa resolução da imagem (nas demais posições os coeficientes *Wavelets*). Prossegue-se desta forma e, após l passos, encontra-se uma submatriz de A, formada por $A[0,0]$ contendo a "média" de todos os *pixels* e nas demais posições estarão os coeficientes *Wavelets*. A FIGURA 2.7 ilustra a Decomposição Não-Padrão.

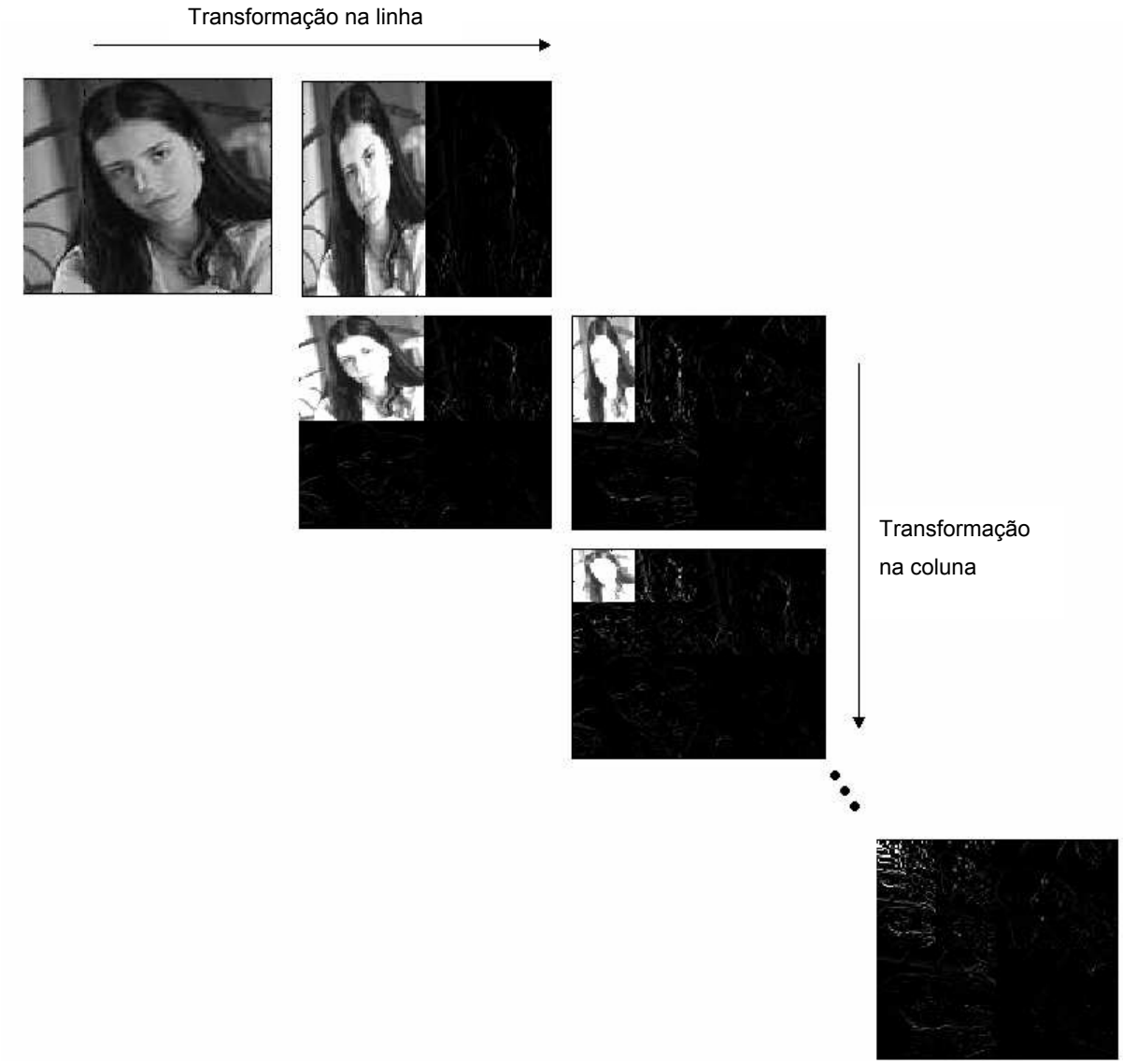


FIGURA 2.7 – Decomposição Não-Padrão de uma imagem utilizando *Wavelets.* (Lima, 2004, p.36)

2.7. O Espectro de Wiener

Considerando-se um processo estacionário ergótico $f(x,y)$, a função de autocorrelação pode ser definida como:

$$R_f(\xi,\eta) = \lim_{X,Y\to\infty} \frac{1}{2X2Y} \int_{-X}^{X} \int_{-Y}^{Y} f(x,y) f(x+\xi, y+\eta) dxdy \tag{2.9}$$

onde ξ e η representam as coordenadas espaciais da autocorrelação.

Na prática, define-se a função de autocorrelação em termos das flutuações de f, $\Delta f = f - \overline{f}$, onde $\overline{f}$ é definido por:

$$\overline{f} = \lim_{X,Y\to\infty} \frac{1}{2X2Y} \int_{-X}^{X} \int_{-Y}^{Y} f(x,y) dxdy \tag{2.10}$$

Redefinido a autocorrelação, tem-se:

$$R_{\Delta f}(\xi,\eta) = \lim_{X,Y\to\infty} \frac{1}{2X2Y} \int_{-X}^{X} \int_{-Y}^{Y} \Delta f(x,y) \Delta f(x+\xi, y+\eta) dxdy \tag{2.11}$$

Desta maneira, a autocorrelação refere-se à definição dada pela equação (2.11). Nota-se que para $R_{\Delta f}(0,0)$, tem-se:

$$R_{\Delta f}(0,0) = \lim_{X,Y\to\infty} \frac{1}{2X2Y} \int_{-X}^{X} \int_{-Y}^{Y} \Delta f(x,y)^2 dxdy = \sigma^2 \tag{2.12}$$

a qual representa a própria variância σ^2.

A função de autocorrelação fornece uma medida do grau em que a grandeza $f(x,y)$ depende de $f(x+\xi, y+\eta)$, o que fornece uma descrição da estrutura do ruído. Considerando-se uma imagem onde $f(x,y)$ representa a intensidade no pixel correspondente à coordenada espacial (x,y), pode-se dizer que:

- Uma função de autocorrelação que decresce rapidamente com a variação de (ξ,η), indica que o ruído é substancialmente independente, a menos de uma pequena distância. Se esta pequena distância coincide com um pixel da imagem, o ruído é branco.
- Uma função de autocorrelação que, ao contrário, decresce lentamente com a variação de (ξ,η) indica que o ruído é altamente correlaciondo, o qual é chamado de ruído estruturado.

O espectro de Wiener das flutuações de um processo aleatório ergótico relaciona-se com a função de autocorrelação segundo o teorema de Wiener-Khintchine, conforme mostram as equações (2.13) e (2.14), formando um par de transformadas de Fourier.

$$W(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} R(\xi,\eta)e^{-2\pi i(u\xi+v\eta)}d\xi d\eta \tag{2.13}$$

$$R(\xi,\eta) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} W(u,v)e^{+2\pi i(u\xi+v\eta)}d\xi d\eta \tag{2.14}$$

Das equações acima, pode-se tirar uma importante relação entre o espectro de Wiener e a variância σ^2, ou seja:

$$\sigma^2 = R(0,0) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} W(u,v)dudv \tag{2.15}$$

A propriedade de ergodicidade que se aplica a autocorrelação não se transfere a sua transformada. O espectro de Wiener guarda as informações sobre a correlação da imagem. O ruído branco, por exemplo, estaria representado por componentes de freqüência distribuídos por toda a base do espectro. Já o ruído estruturado, teria apenas informações em baixas freqüências.

Para o caso discreto, as equações (2.13) e (2.14) tornam-se:

$$W(u,v) = \frac{1}{MN} \sum_{\xi=0}^{N-1} \sum_{\eta=0}^{M-1} R(\xi,\eta) e^{-j2\pi\left(\frac{u\xi}{N}+\frac{v\eta}{M}\right)} \tag{2.16}$$

$$R(\xi,\eta) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} W(u,v) e^{+j2\pi\left(\frac{u\xi}{N}+\frac{v\eta}{M}\right)} \tag{2.17}$$

onde:

$$R(\xi,\eta) = \frac{1}{NM} \sum_{m=0}^{N-1} \sum_{n=0}^{M-1} (f(m,n) - \overline{f})(f^*(m+\xi, n+\eta) - \overline{f}) \tag{2.18}$$

onde ξ e m são amostrados em intervalos de Δx e η e n são amostrados em intervalos de Δy.

A relação entre os intervalos de amostragem no domínio espacial e da freqüência é dada pelas equações (2.19) e (2.20).

$$\Delta x = \frac{1}{N\Delta u} \tag{2.19}$$

$$\Delta y = \frac{1}{M\Delta v} \tag{2.20}$$

Capítulo 3: Método para Compressão de Imagens Digitais fundamentado em Procedimentos de Huffman e *Wavelets*

3.1. Características do Sistema

Utilizou-se a codificação e decodificação de Huffman para obter-se a compressão sem perda de conteúdo. Técnicas do processamento de imagens e modelagem matemática, com base na ferramenta matemática *Wavelet* para decomposição da imagem, foram usadas neste sistema. Através do Espectro de Wiener os resultados obtidos no processo de compressão e decomposição da imagem foram avaliados com o objetivo de verificar se houve alguma perda na qualidade da imagem compressa ou decomposta.

O sistema desenvolvido processa imagens no formato .bmp, em escala de cinza e com 8 bits.

A FIGURA 3.1 ilustra o Diagrama geral do sistema desenvolvido. O usuário escolhe a imagem no Banco de Imagens e faz a opção por uma das três situações: Compressão utilizando a Codificação de Huffman; Decomposição utilizando a Transformada *Wavelet* Haar ou o processo de Decomposição utilizando a Transformada *Wavelet* seguida da compressão com a Codificação de Huffman. Deste processo resulta a compressão da imagem no primeiro e no terceiro caso e a decomposição da imagem com a escolha da segunda opção. Posteriormente, a imagem compressa ou decomposta sofrerá o processo inverso com o objetivo de avaliar se a mesma sofreu alguma perda com a realização do processo escolhido. Neste momento, a imagem será submetida ao Espectro de Wiener para avaliar sua qualidade. Com esta análise têm-se os resultados dos coeficientes de variância tanto da imagem original quanto da imagem transformada. Avaliam-se os resultados da seguinte maneira: quanto mais próximos forem os valores dos coeficientes de variância tanto da imagem original como da imagem transformada, menor foi a perda resultante do processo realizado.

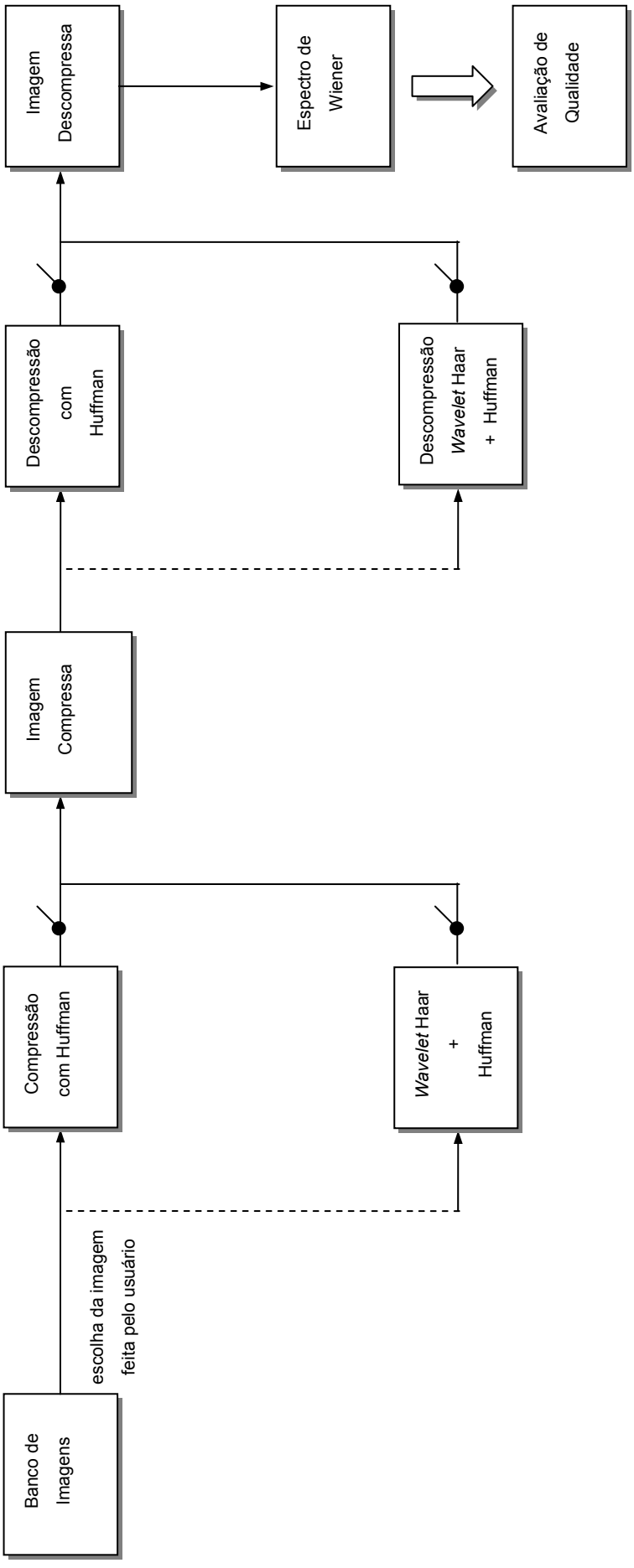


FIGURA 3.1 - Diagrama geral do Sistema desenvolvido.

A FIGURA 3.2 ilustra em pseudocódigo a estrutura algorítmica para o sistema de compressão de imagens digitais com base em Huffman e *Wavelet*:

```
Begin
Buscar a imagem escolhida pelo usuário no Banco de Imagens;
        Se a Opção for Compressão pela Codificação de Huffman
              Comprimir a imagem utilizando a codificação de Huffman;
              Salvar a imagem compressa;

        Se a Opção for Decomposição pela Transformada Wavelet
              Decompor a imagem utilizando a Transformada Wavelet Haar;
              Salvar a imagem decomposta;

        Se a Opção for Decomposição pela Transformada Wavelet seguida da
        Compressão pela Codificação de Huffman
              Decompor a imagem utilizando a Transformada Wavelet Haar;
              Comprimir a imagem utilizando a codificação de Huffman;
              Salvar a imagem compressa;

        Se a Opção for Descompressão pela Decodificação de Huffman
              Descomprimir a imagem utilizando a decodificação de Huffman;
              Salvar a imagem descompressa;

        Se a Opção for Composição pela Transformada Wavelet seguida da
        Descompressão pela Decodificação de Huffman
              Compor a imagem utilizando a Transformada Wavelet Haar;
              Descomprimir a imagem utilizando a decodificação de Huffman;
              Salvar a imagem descompressa;

Utilizar o Espectro de Wiener na imagem Descompressa/Decomposta com o
objetivo de avaliar a qualidade;
End;
```

FIGURA 3.2 - Pseudocódigo para busca, compressão, descompressão e análise de qualidade de imagens digitais.

3.2. Compressão e Descompressão utilizando Huffman

No módulo de compressão da imagem utilizando o código de Huffman, o usuário escolhe a imagem desejada para o processo da compressão. O sistema busca a Imagem no banco de imagens e comprime-a. A imagem compressa poderá ser salva no banco de imagem compressa e estará disponível para a operação inversa, ou seja, a descompressão.

Após o processo de descompressão, realizado através da decodificação de Huffman, as imagens podem submeter-se ao Espectro de Wiener, processo pelo qual avalia-se a qualidade da imagem.

Verifica-se com o Espectro de Wiener se houve variância entre as imagens original e a descompressa através da decodificação de Huffman. Se os valores gerados através do Espectro de Wiener forem iguais, significa que não houve perda de informação com o processo de compressão/descompressão realizado.

A FIGURA 3.3 ilustra o diagrama geral da compressão utilizando a codificação e decodificação de Huffman:

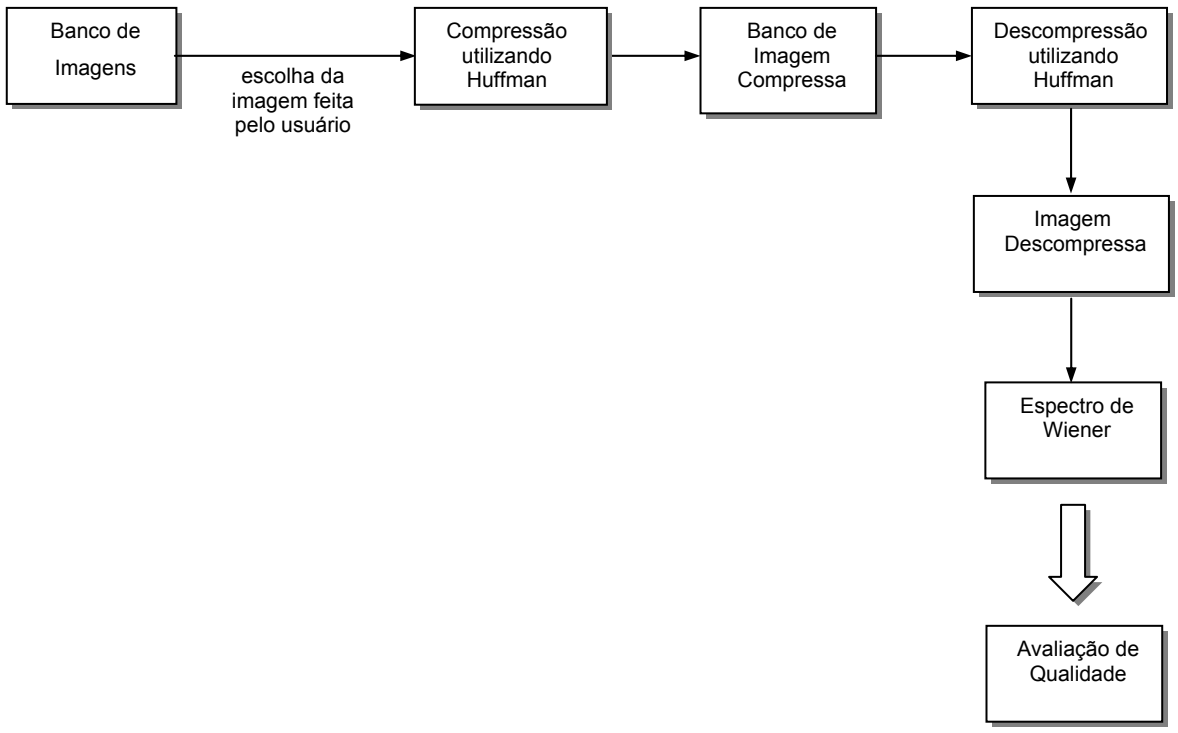


FIGURA 3.3 - Diagrama do Módulo de Compressão e Descompressão utilizando Huffman.

A FiGURA 3.4 ilustra em pseudocódigo a estrutura algorítmica para o sistema de Compressão e Descompressão utilizando o código de Huffman.

Begin

Buscar a imagem escolhida pelo usuário no Banco de Imagens;

Se a Opção for Compressão pela Codificação de Huffman

Construir a árvore de Huffman com os passos abaixo:

1. Determina-se a freqüência de ocorrência de cada símbolo;
2. Cria-se uma lista classificada em ordem crescente de freqüência de ocorrência de cada símbolo;
3. Transfere-se para um nó pai a soma das freqüências dos dois nós livres com freqüências mais baixas;
4. Remove-se da lista de nós livres os dois nós filhos e adiciona-se o mais novo nó pai;
5. Repetem-se os passos 2 e 3 até existir somente um nó livre à esquerda. Este nó livre é a raiz da árvore;

Salvar a imagem compressa;

Se a Opção for Descompressão pela Decodificação de Huffman

Descomprimir a imagem utilizando o processo inverso ao descrito na codificação de Huffman;

Salvar a imagem descompressa;

Utilizar o Espectro de Wiener na imagem Descompressa com o objetivo de avaliar a qualidade;

End;

FIGURA 3.4 - Pseudocódigo para a implementação do Módulo de Compressão e Descompressão utilizando Huffman.

3.3. Decomposição da imagem utilizando a Transformada *Wavelet* Haar

No módulo de decomposição da imagem utilizando a Transformada *Wavelet* Haar, o usuário escolhe a imagem desejada para o processo da decomposição. O sistema busca a Imagem no banco de imagens e a decompõe. A imagem decomposta poderá ser salva no banco de imagens descompostas.

Após o processo de decomposição, as imagens foram submetidas ao processo de validação efetuado pelo Espectro de Wiener.

Verifica-se com o Espectro de Wiener se houve alguma alteração na variância entre as imagens original e a decomposta através da decomposição pela *Wavelet* Haar. Se os valores gerados através do Espectro de Wiener forem iguais significa que não houve perda de informação com o processo de decomposição realizado.

A FIGURA 3.5 ilustra o diagrama geral da decomposição utilizando a Transformada *Wavelet* Haar:

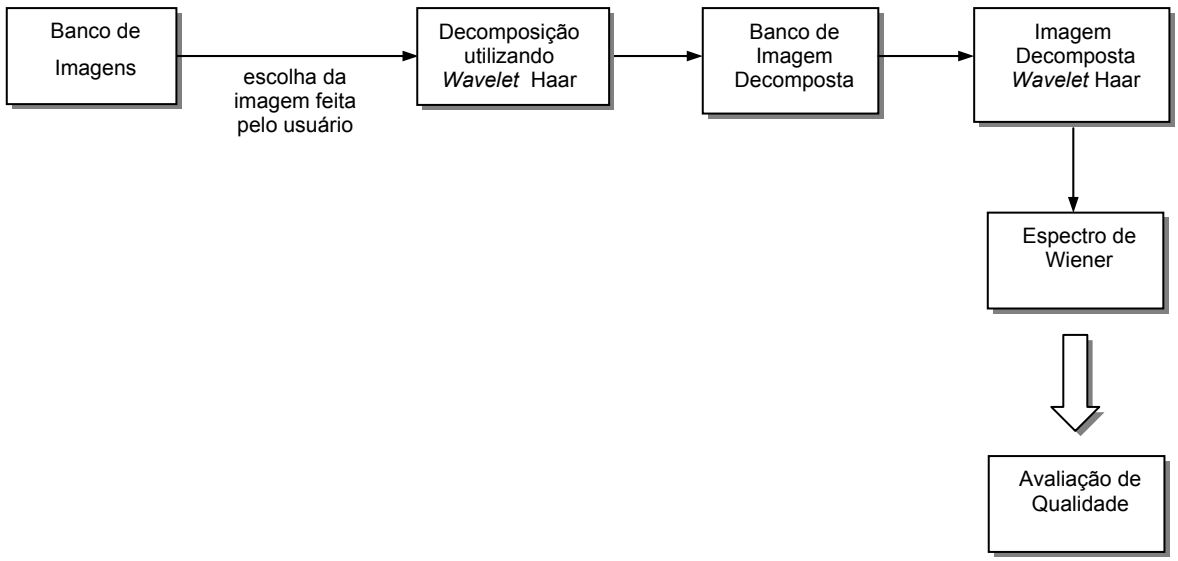


FIGURA 3.5 - Diagrama do Módulo de Decomposição utilizando *Wavelet* Haar.

A FiGURA 3.6 ilustra em pseudocódigo a estrutura algorítmica para o processo de Decomposição da imagem utilizando a Transformada *Wavelet* Haar.

```
Begin

Buscar a imagem escolhida pelo usuário no Banco de Imagens;
        Se a Opção for Decomposição com a Transformada Wavelet Haar
              Decompor a imagem;
              Salvar a imagem decomposta;

Utilizar o Espectro de Wiener na imagem Decomposta com o objetivo de
avaliar a qualidade;

End;
```

FIGURA 3.6 - Pseudocódigo para a implementação do Módulo de Decomposição utilizando a *Wavelet* Haar.

Neste trabalho utiliza-se a análise espectral de Wiener para verificar a possível perda de informação ocasionada pelos processos de compressão através da codificação de Huffman e da decomposição com a Transformada *Wavelet* Haar.

Os estudos de caso para avaliação dos resultados foram desenvolvidos considerando as imagens Lena e Girl, clássicas do Processamento Digital de Imagens (ambas nas resoluções 1024x1024, 512x512, 256x256 e 128x128), e uma imagem de solo e outra de phantom, ambas imagens tomográficas obtidas no minitomógrafo de raios X e Gama da Embrapa Instrumentação Agropecuária.

Lena (ou Lenna) é uma das imagens mais amplamente utilizada para testes em algoritmos de compressão e processamento de imagens em geral. Lena

Sodeberg vivia na Suécia e sua imagem foi originalmente digitalizada da revista Playboy de novembro de 1972, conforme ilustra a FIGURA 3.7.

FIGURA 3.7 – Imagem Lena.

A imagem Girl, conforme ilustra a FIGURA 3.8, é outra imagem também muito utilizada na área de processamento de imagens digitais.

FIGURA 3.8 – Imagem Girl.

As imagens tomográficas de uma amostra de solo e outra de um phantom, conforme ilustram as FIGURAS 3.9 e 3.10, respectivamente, foram obtidas através do minitomógrafo da Embrapa Instrumentação Agropecuária.

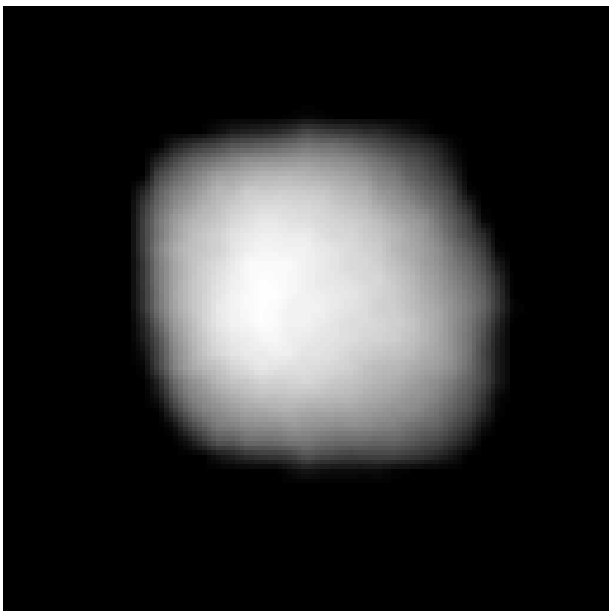


FIGURA 3.9 – Corte transversal de uma amostra de solo reconstruído pelo método de retroprojeção filtrada.

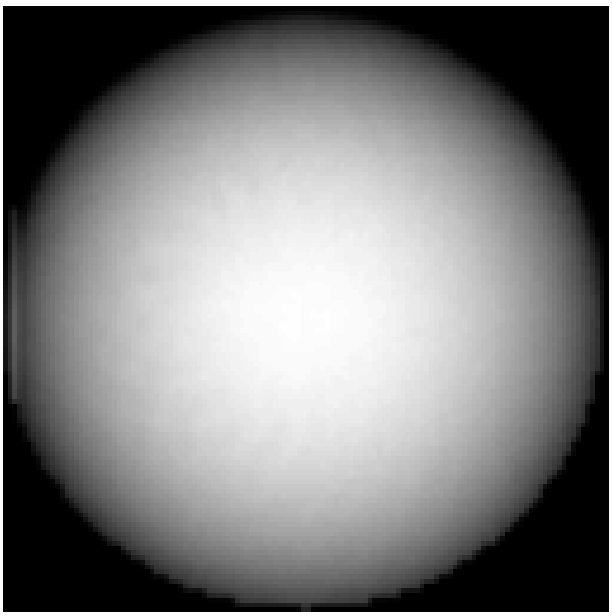


FIGURA 3.10 – Phantom homogêneo de nylon reconstruído por retroprojeção filtrada. ($\mu = 0.213 cm^{-1}$)

Capítulo 4: Resultados, Discussões e Conclusões

4.1. Resultados e Discussões

Para avaliação dos resultados da aplicação do método para compressão de imagens digitais fundamentado em procedimentos de Huffman e *Wavelets* foram desenvolvidos dez estudos de casos, sendo quatro casos envolvendo a imagem Lena, quatro casos envolvendo a imagem Girl e dois casos envolvendo imagens obtidas com o tomógrafo de raio-X e Gama da Embrapa Instrumentação Agropecuária.

- **1º estudo de caso:** Imagem Lena com resolução de 128x128 pixels:

A imagem Lena (com resolução de 128x128 pixels, conforme ilustra a FIGURA 4.1) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.1 – Lena128x128.bmp

A TABELA 4.1 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Lena128x128	17462	15900	8,95%	15852	9,22%

TABELA 4.1 – Lena128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,27% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.2 apresenta o Espectro de Wiener resultante da imagem Lena em resolução 128x128 pixels. As FIGURAS 4.3 e 4.4 ilustram respectivamente a imagem Lena descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.5 e 4.6 apresentam respectivamente a imagem Lena descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

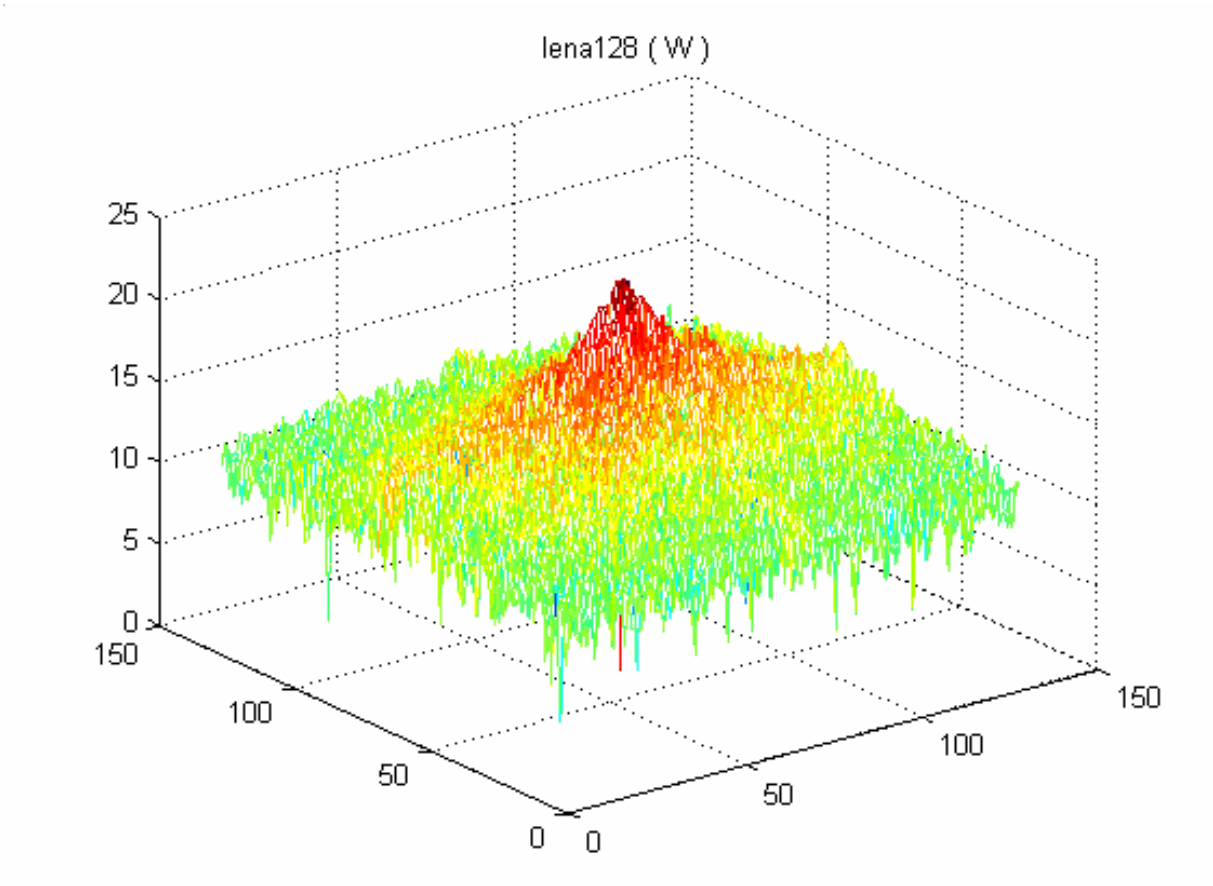


FIGURA 4.2 – Lena128x128: Espectro de Wiener resultante da imagem original.

FIGURA 4.3 – Lena128x128: Resultado da imagem descompressa após a codificação Huffman.

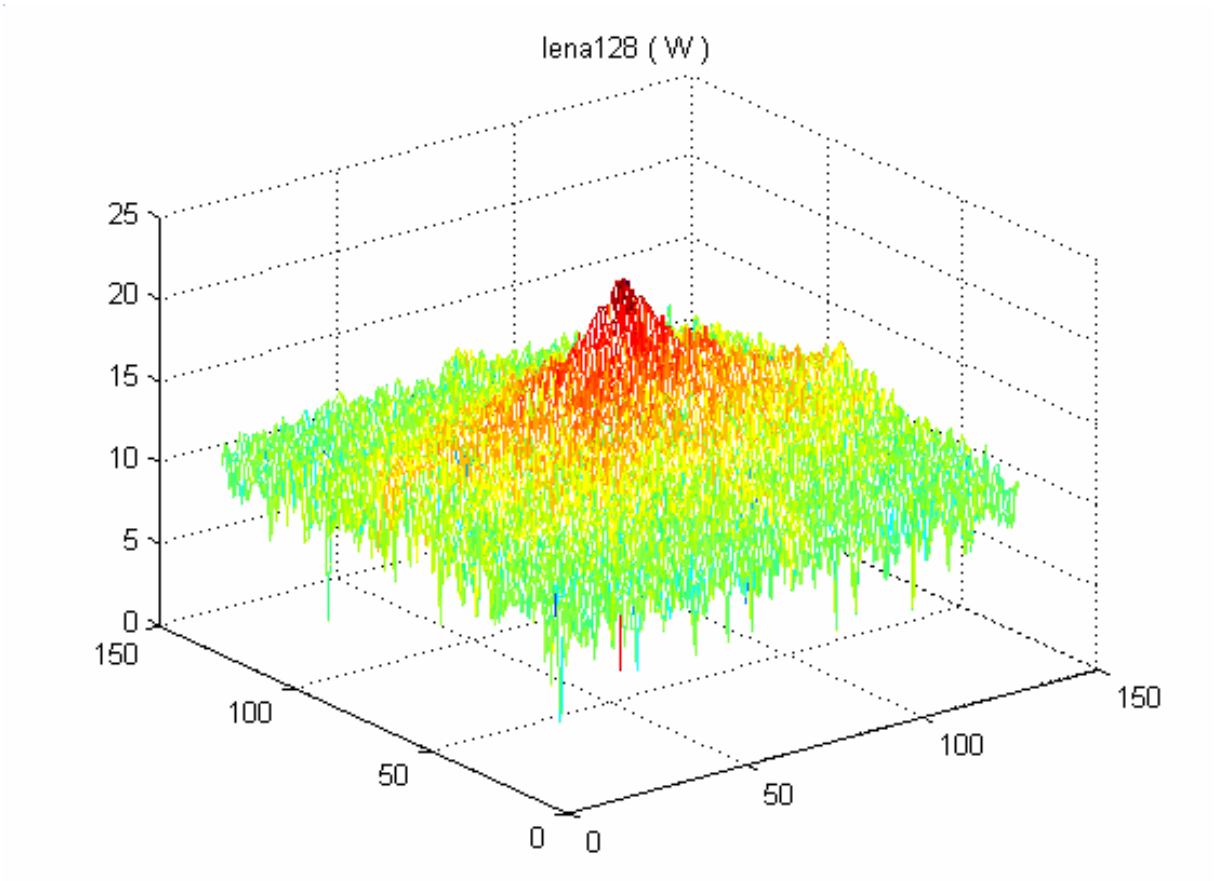


FIGURA 4.4 – Lena128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.5 – Lena128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

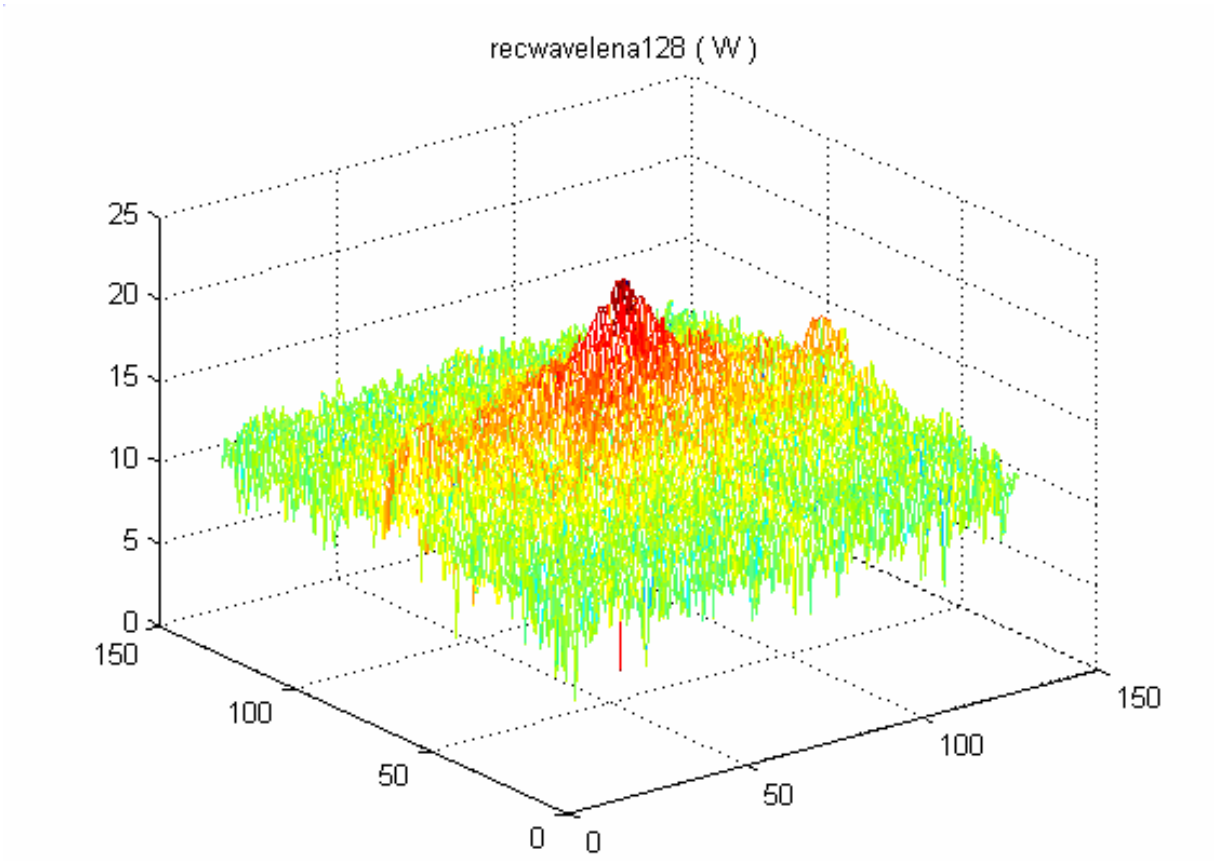


FIGURA 4.6 – Lena128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.2 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Lena Original	17462	1.1027e -006
Lena Compressa por Huffman	15900	1.1027e -006
Lena Compressa Decomposta por *Wavelet* e codificada por Huffman	15852	2.0862e -007

TABELA 4.2 – Lena128x128: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.2, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $8{,}94 \times 10^{-7}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **2º estudo de caso:** Imagem Lena com resolução de 256x256 pixels:

A imagem Lena (com resolução de 256x256 pixels, conforme ilustra a FIGURA 4.7) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.7 – Lena256x256.bmp

A TABELA 4.3 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Lena256x256	66616	61492	7,69%	61004	8,42%

TABELA 4.3 – Lena256x256: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,73% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.8 apresenta o Espectro de Wiener resultante da imagem Lena em resolução 256x256 pixels. As FIGURAS 4.9 e 4.10 ilustram respectivamente a imagem Lena descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.11 e 4.12 apresentam respectivamente a imagem Lena descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

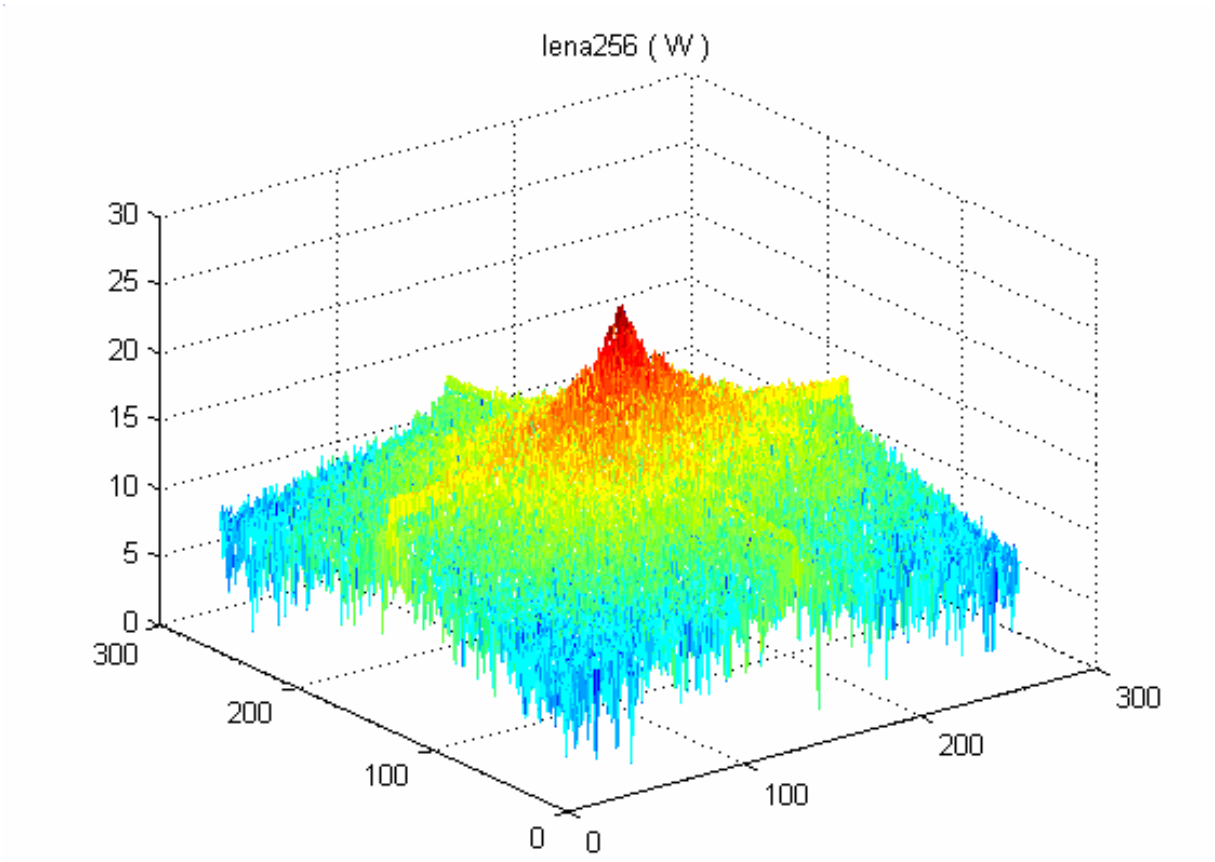


FIGURA 4.8 – Lena256x256: Espectro de Wiener resultante da imagem original.

FIGURA 4.9 – Lena256x256: Resultado da imagem descompressa após a codificação Huffman.

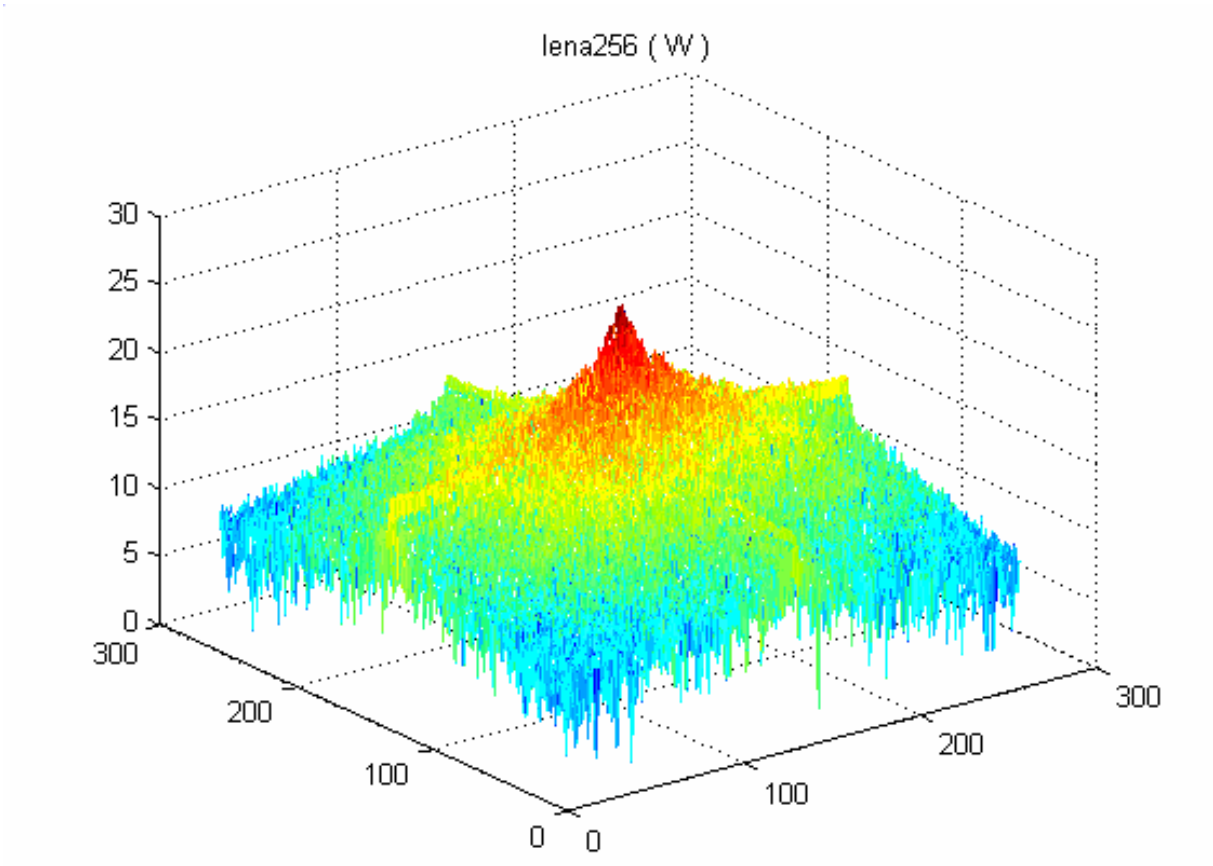


FIGURA 4.10 – Lena256x256: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.11 – Lena256x256: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

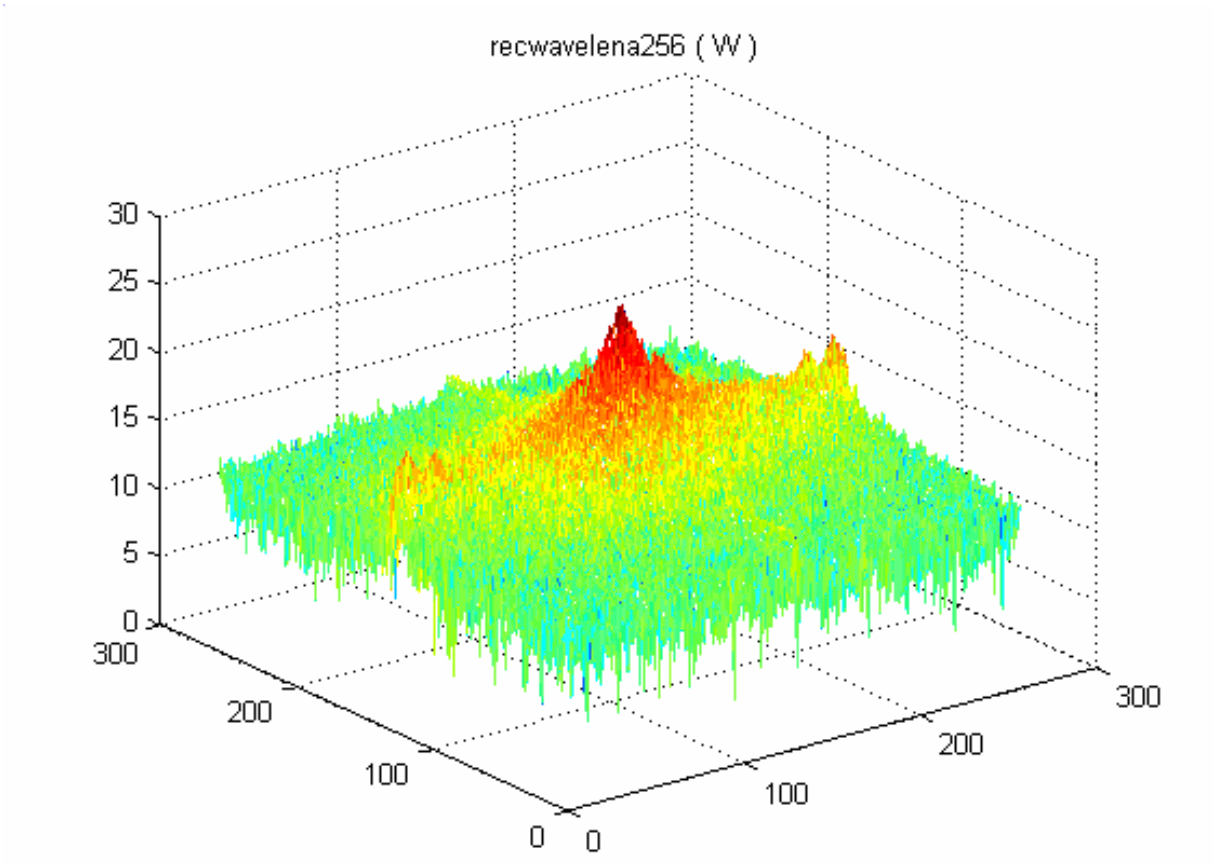


FIGURA 4.12 – Lena256x256: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.4 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Lena Original	66616	7.6294e -006
Lena Compressa por Huffman	61492	7.6294e -006
Lena Compressa Decomposta por *Wavelet* e codificada por Huffman	61004	4.2915e -006

TABELA 4.4 – Lena256x256: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.4, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $3{,}34 \times 10^{-6}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **3º estudo de caso:** Imagem Lena com resolução de 512x512 pixels:

A imagem Lena (com resolução de 512x512 pixels, conforme ilustra a FIGURA 4.13) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.13 – Lena512x512.bmp

A TABELA 4.5 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Lena512x512	263222	244760	7,01%	243568	7,47%

TABELA 4.5 – Lena512x512: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,46% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.14 apresenta o Espectro de Wiener resultante da imagem Lena em resolução 512x512 pixels. As FIGURAS 4.15 e 4.16 ilustram respectivamente a imagem Lena descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.17 e 4.18 apresentam respectivamente a imagem Lena descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

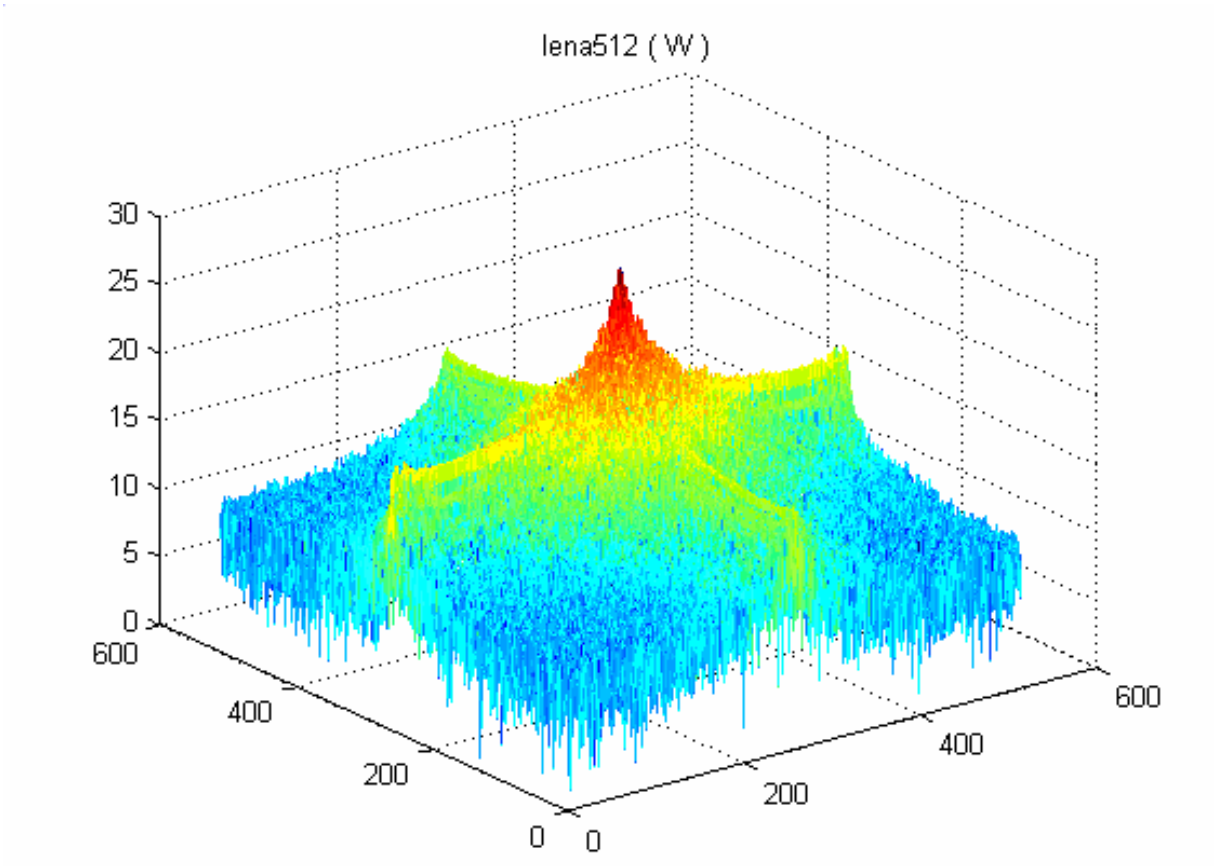


FIGURA 4.14 – Lena512x512: Espectro de Wiener resultante da imagem original.

FIGURA 4.15 – Lena512x512: Resultado da imagem descompressa após a codificação Huffman.

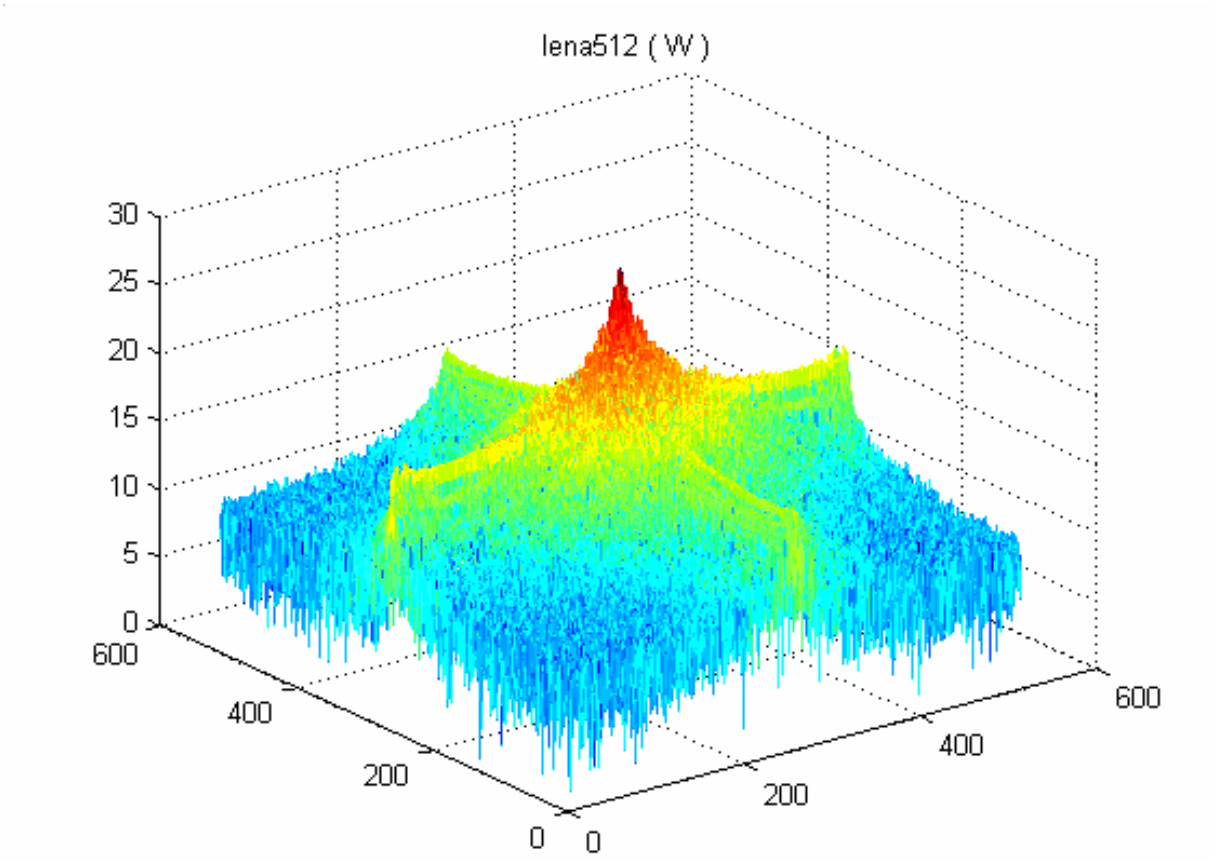


FIGURA 4.16 – Lena512x512: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.17 – Lena512x512: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

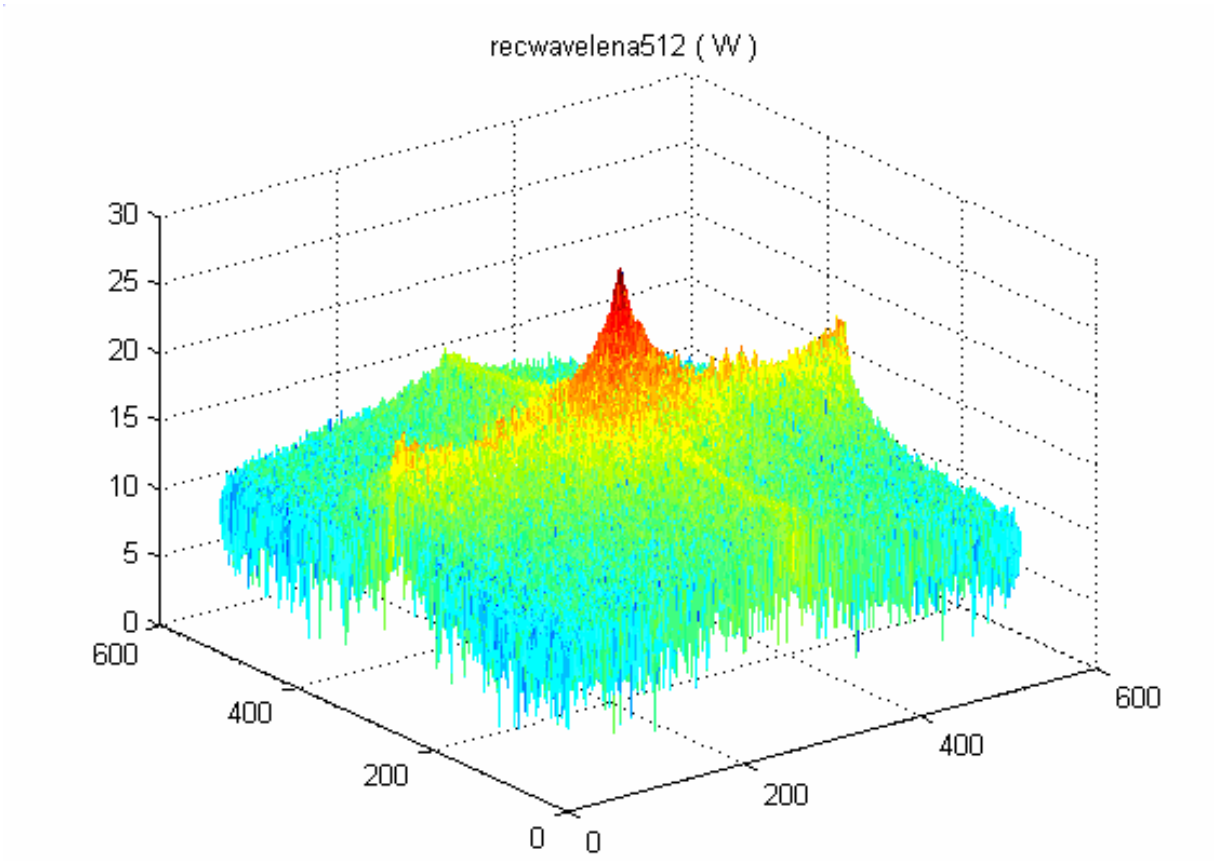


FIGURA 4.18 – Lena512x512: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.6 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Lena Original	263222	9.7275e -005
Lena Compressa por Huffman	244760	9.7275e -005
Lena Compressa Decomposta por *Wavelet* e codificada por Huffman	243568	3.8147e -006

TABELA 4.6 – Lena512x512: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.6, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $9{,}35 \times 10^{-5}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **4º estudo de caso:** Imagem Lena com resolução de 1024x1024 pixels:

A imagem Lena (com resolução de 1024x1024 pixels, conforme ilustra a FIGURA 4.19) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.19 – Lena1024x1024.bmp

A TABELA 4.7 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Lena1024x1024	1049654	976384	6,98%	973488	7,26%

TABELA 4.7 – Lena1024x1024: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,28% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.20 apresenta o Espectro de Wiener resultante da imagem Lena em resolução 1024x1024 pixels. As FIGURAS 4.21 e 4.22 ilustram respectivamente a imagem Lena descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.23 e 4.24 apresentam respectivamente a imagem Lena descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

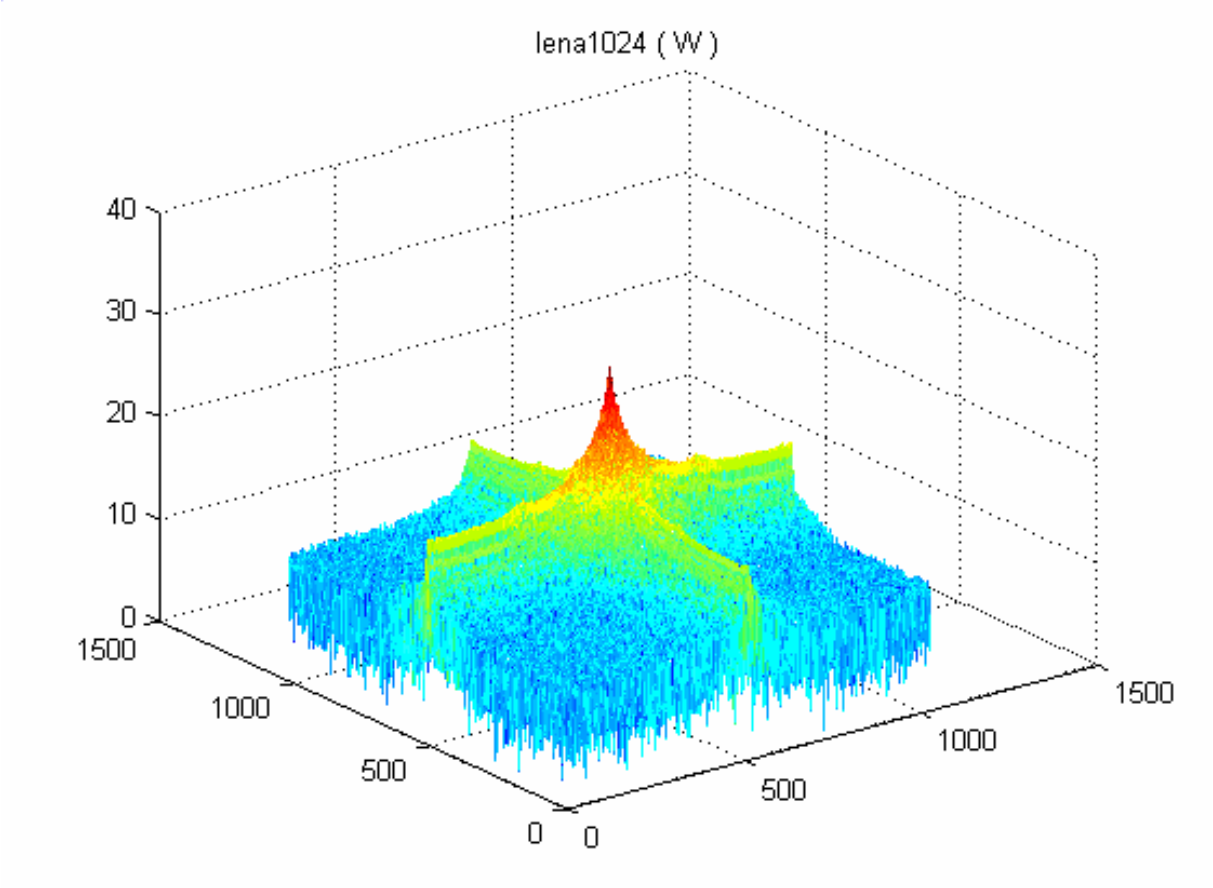


FIGURA 4.20 – Lena1024x1024: Espectro de Wiener resultante da imagem original.

FIGURA 4.21 – Lena1024x1024: Resultado da imagem descompressa após a codificação Huffman.

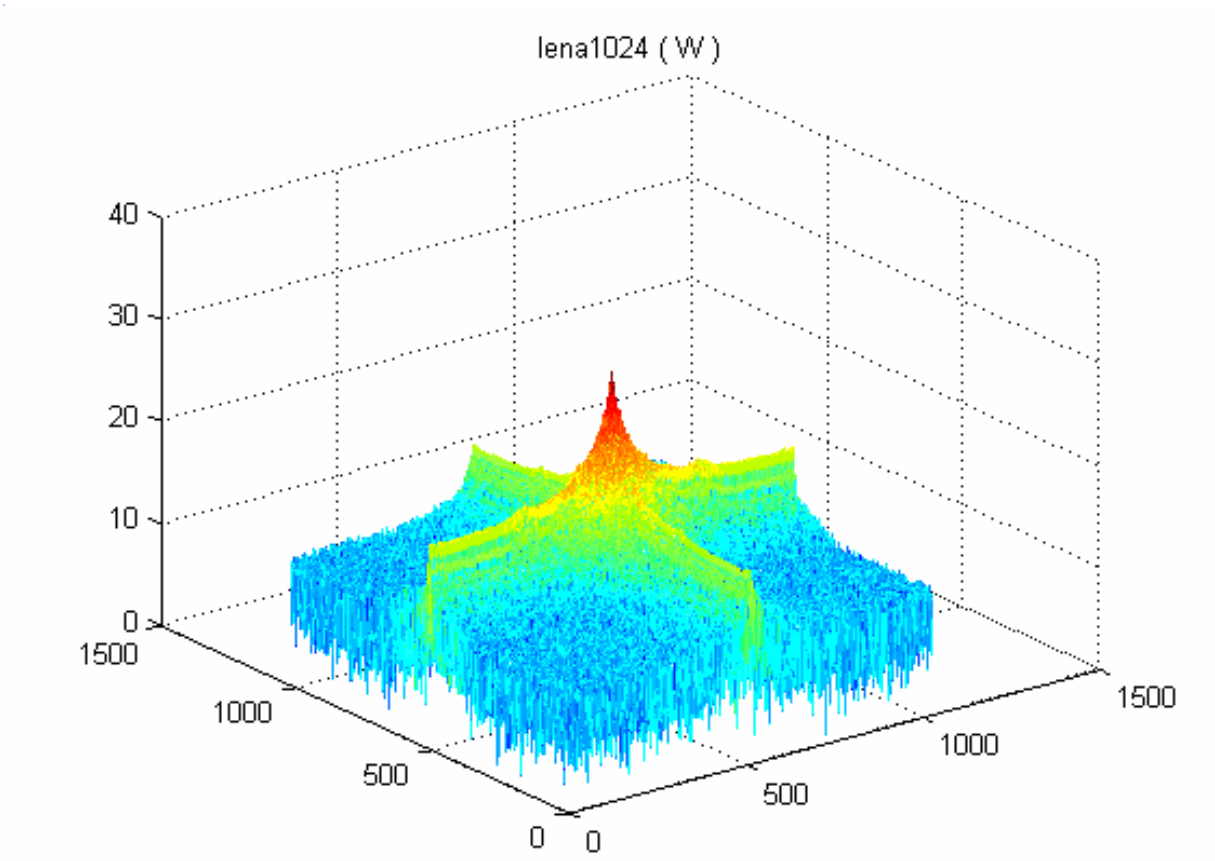


FIGURA 4.22 – Lena1024x1024: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.23 – Lena1024x1024: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

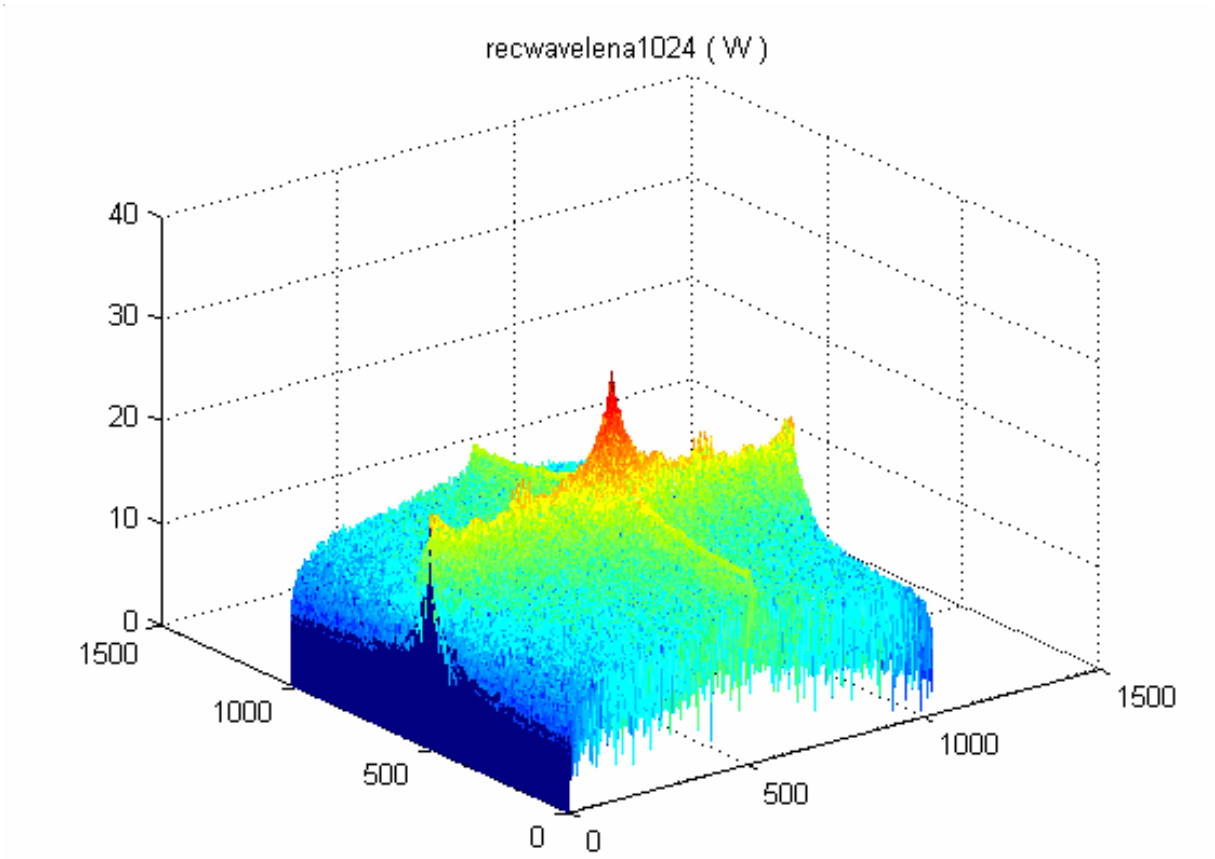


FIGURA 4.24 – Lena1024x1024: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.8 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Lena Original	1049654	0.0024
Lena Compressa por Huffman	976384	0.0024
Lena Compressa Decomposta por *Wavelet* e codificada por Huffman	973488	0.001

TABELA 4.8 – Lena1024x1024: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.8, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $1{,}40 \times 10^{-3}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **5º estudo de caso:** Imagem Girl com resolução de 128x128 pixels:

A imagem Girl (com resolução de 128x128 pixels, conforme ilustra a FIGURA 4.25) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.25 – Girl128x128.bmp

A TABELA 4.9 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Girl128x128	17462	15200	12,95%	14960	14,33%

TABELA 4.9 – Girl128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 1,38% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.26 apresenta o Espectro de Wiener resultante da imagem Girl em resolução 128x128 pixels. As FIGURAS 4.27 e 4.28 ilustram respectivamente a imagem Girl descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.29 e 4.30 apresentam respectivamente a imagem Girl descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

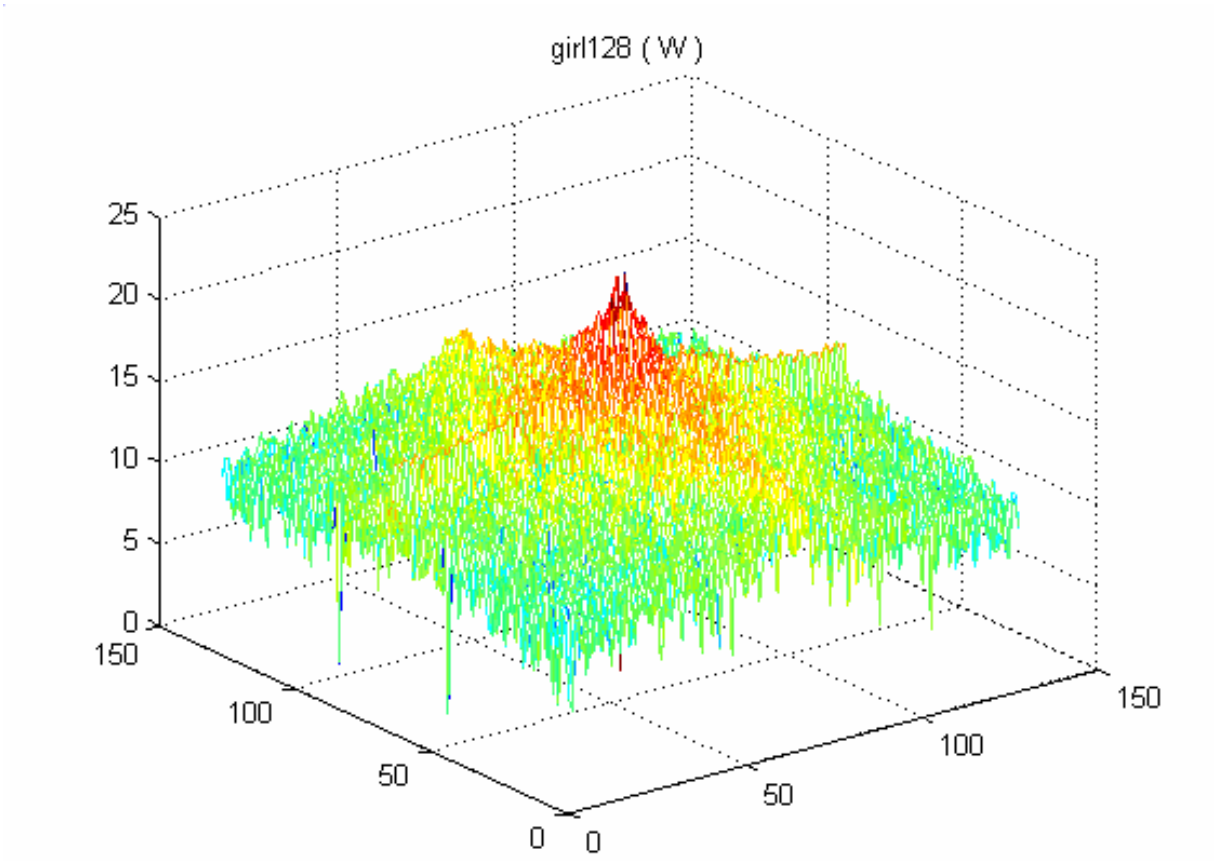


FIGURA 4.26 – Girl128x128: Espectro de Wiener resultante da imagem original.

FIGURA 4.27 – Girl128x128: Resultado da imagem descompressa após a codificação Huffman.

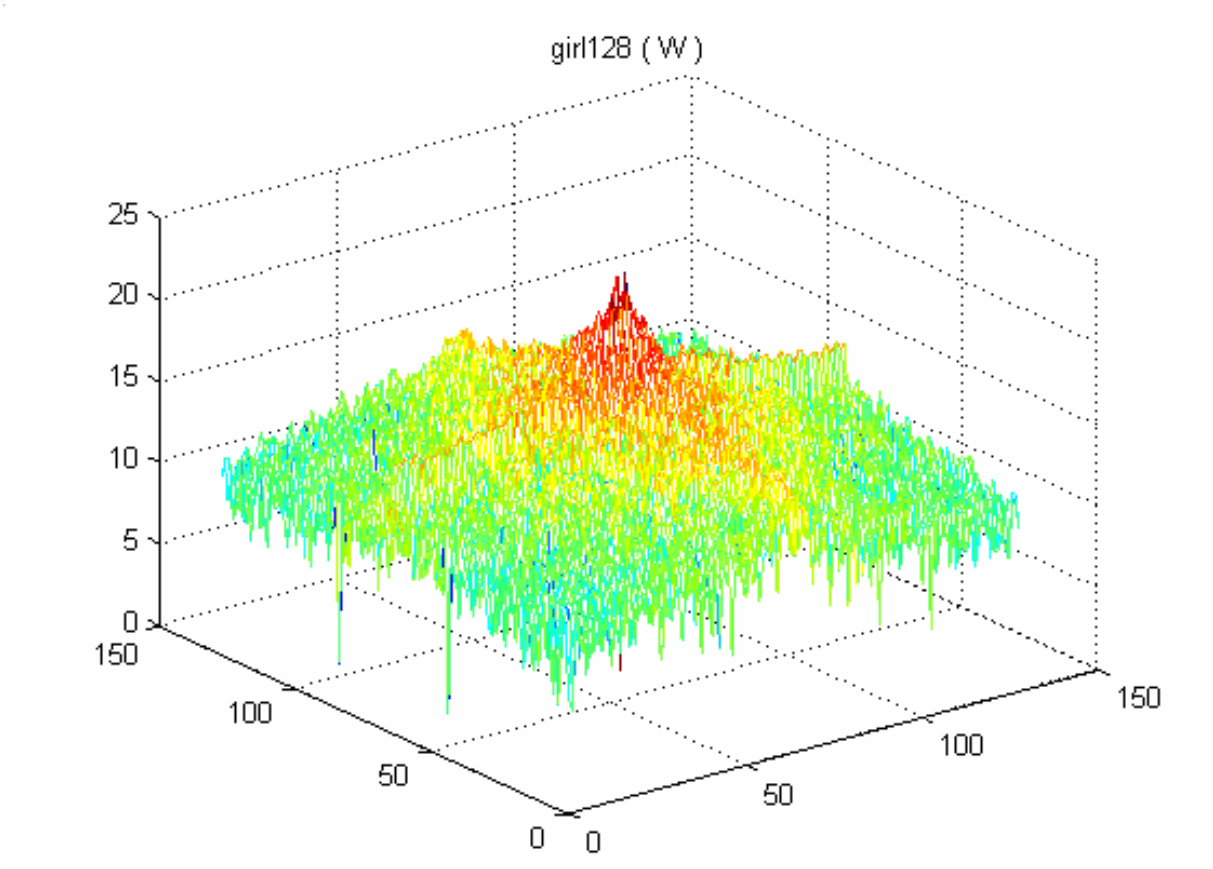


FIGURA 4.28 – Girl128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.29 – Girl128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

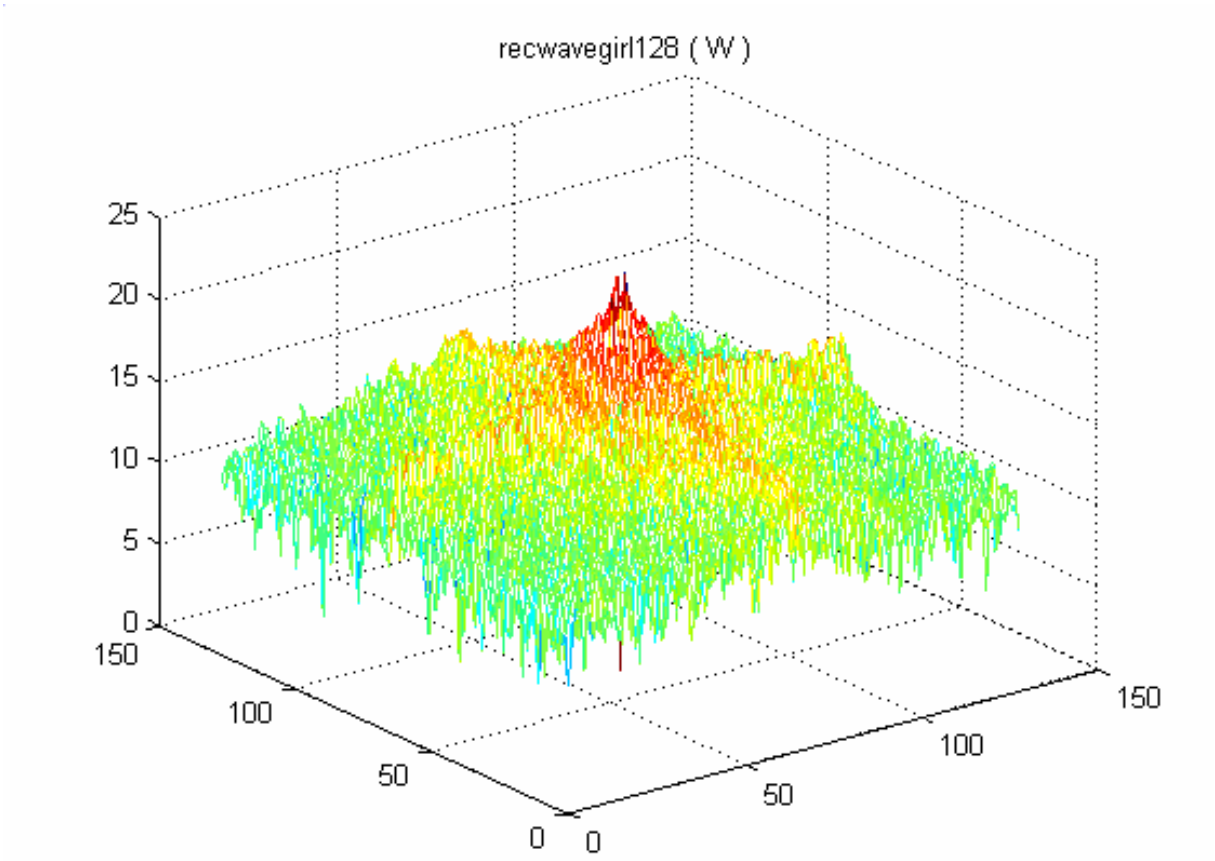


FIGURA 4.30 – Girl128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.10 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Girl Original	17462	1.5795e -006
Girl Compressa por Huffman	15200	1.5795e -006
Girl Compressa Decomposta por *Wavelet* e codificada por Huffman	14960	7.4506e -007

TABELA 4.10 – Girl128x128: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.10, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $8{,}34 \times 10^{-7}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **6º estudo de caso:** Imagem Girl com resolução de 256x256 pixels:

A imagem Girl (com resolução de 256x256 pixels, conforme ilustra a FIGURA 4.31) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.31 – Girl256x256.bmp

A TABELA 4.11 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Girl256x256	66614	58956	11,50%	58564	12,08%

TABELA 4.11 – Girl256x256: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,58% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.32 apresenta o Espectro de Wiener resultante da imagem Girl em resolução 256x256 pixels. As FIGURAS 4.33 e 4.34 ilustram respectivamente a imagem Girl descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.35 e 4.36 apresentam respectivamente a imagem Girl descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

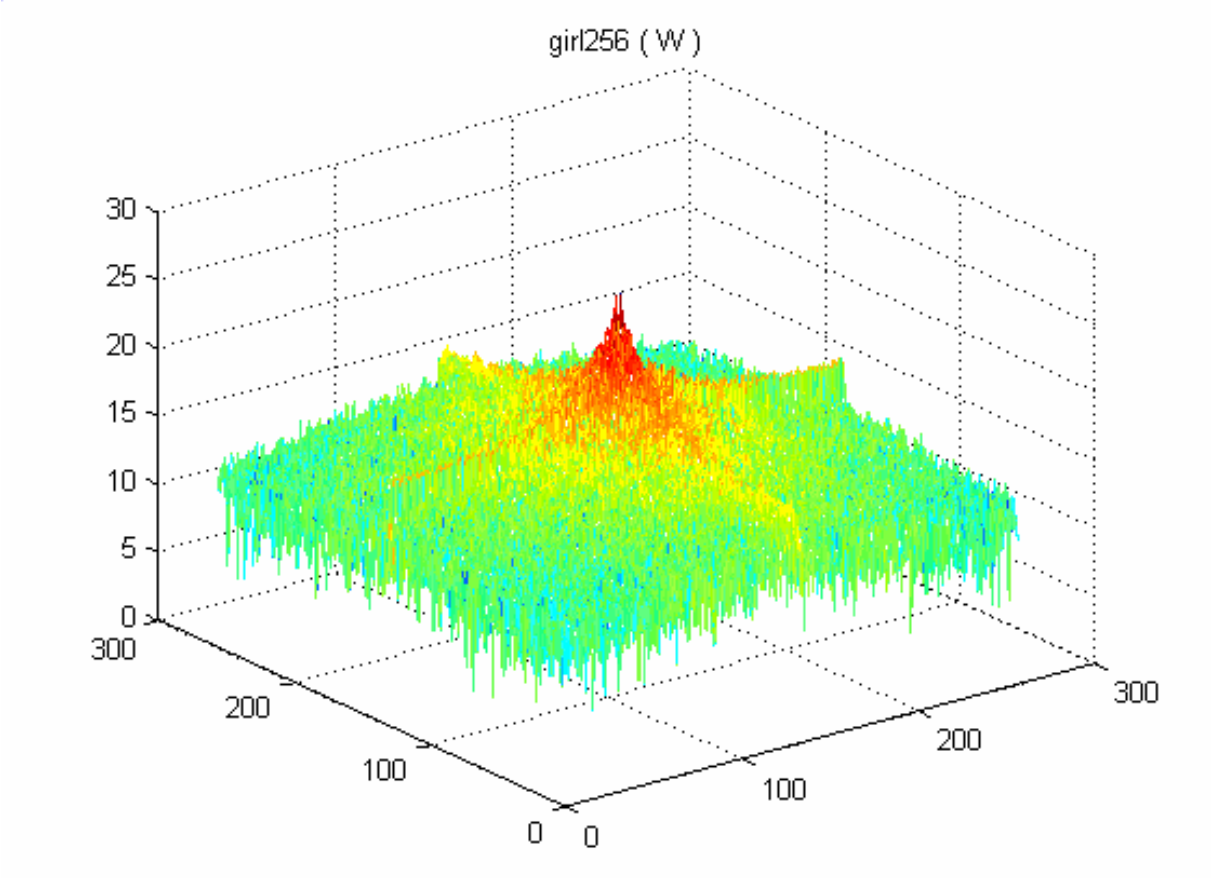


FIGURA 4.32 – Girl256x256: Espectro de Wiener resultante da imagem original.

FIGURA 4.33 – Girl256x256: Resultado da imagem descompressa após a codificação Huffman.

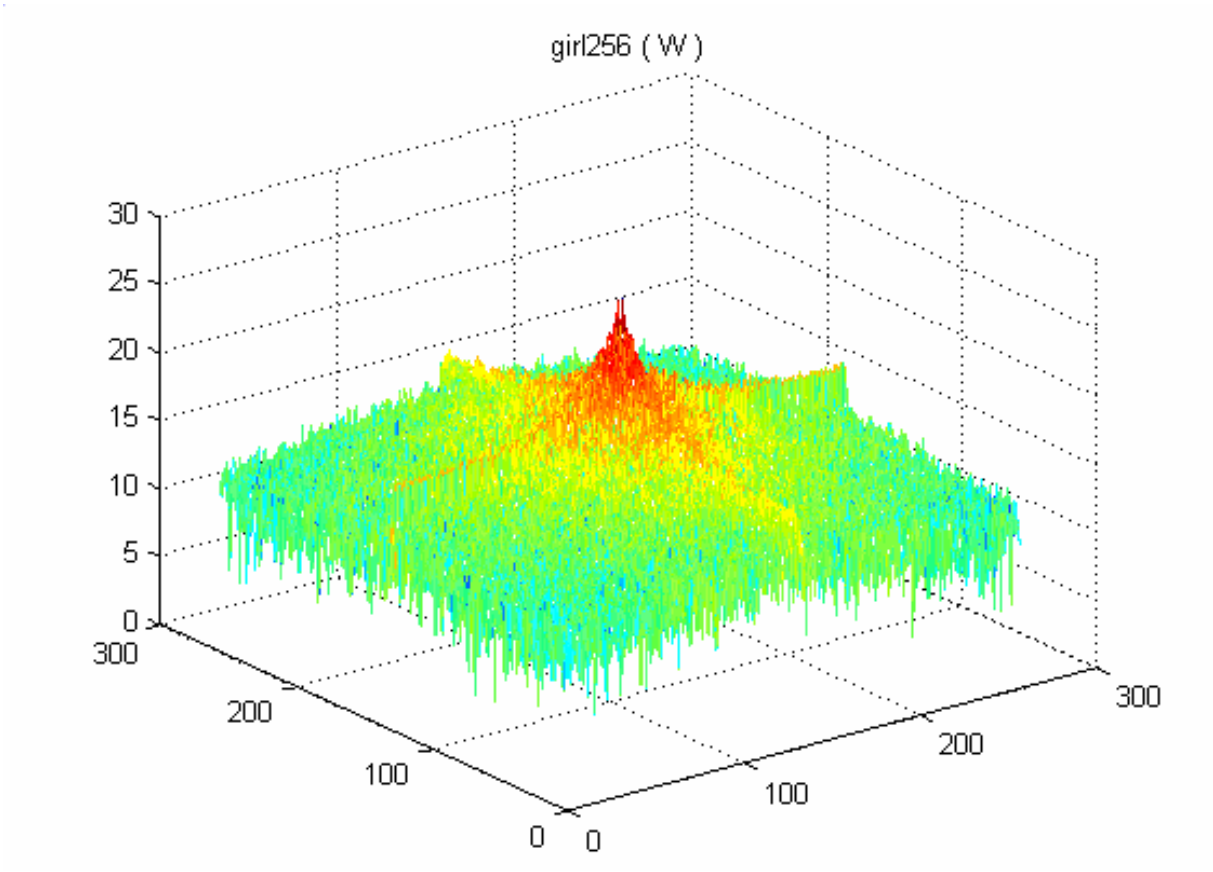


FIGURA 4.34 – Girl256x256: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.35 – Girl256x256: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

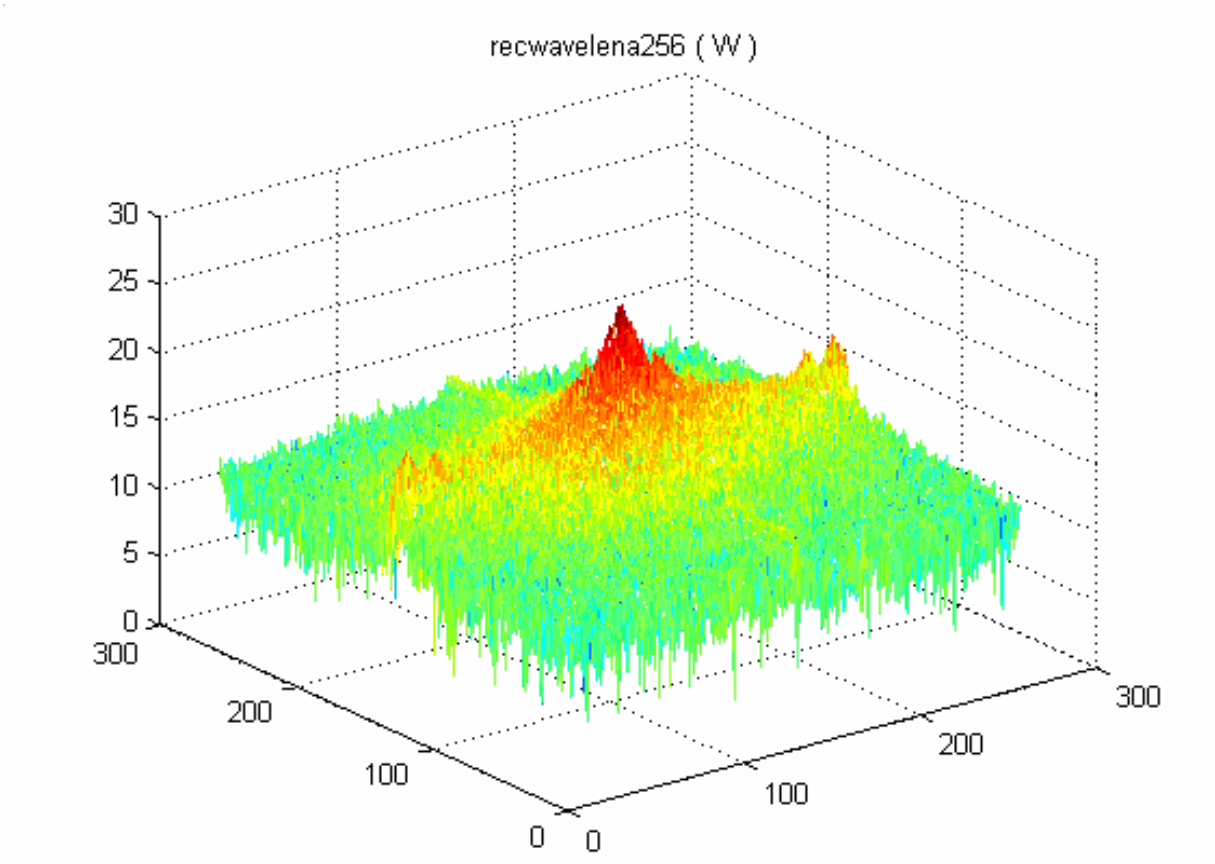


FIGURA 4.36 – Girl256x256: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.12 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Girl Original	66614	1.1683e -005
Girl Compressa por Huffman	58956	1.1683e -005
Girl Compressa Decomposta por *Wavelet* e codificada por Huffman	58564	7.1526e -007

TABELA 4.12 – Girl256x256: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.12, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $1{,}10 \times 10^{-5}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **7º estudo de caso:** Imagem Girl com resolução de 512x512 pixels:

A imagem Girl (com resolução de 512x512 pixels, conforme ilustra a FIGURA 4.37) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.37 – Girl512x512.bmp

A TABELA 4.13 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Girl512x512	263222	231644	12,00%	230648	12,38%

TABELA 4.13 – Girl512x512: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,38% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.38 apresenta o Espectro de Wiener resultante da imagem Girl em resolução 512x512 pixels. As FIGURAS 4.39 e 4.40 ilustram respectivamente a imagem Girl descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.41 e 4.42 apresentam respectivamente a imagem Girl descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

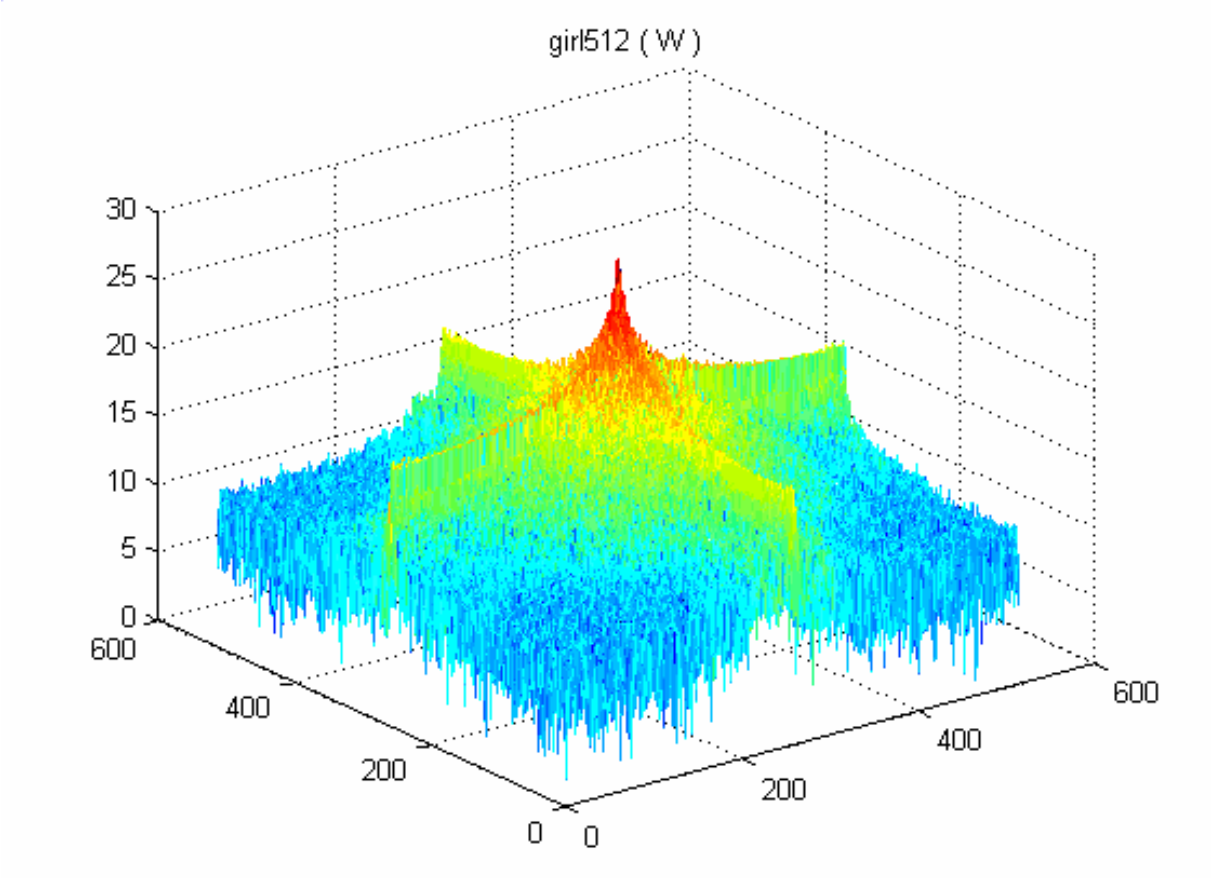


FIGURA 4.38 – Girl512x512: Espectro de Wiener resultante da imagem original.

FIGURA 4.39 – Girl512x512: Resultado da imagem descompressa após a codificação Huffman.

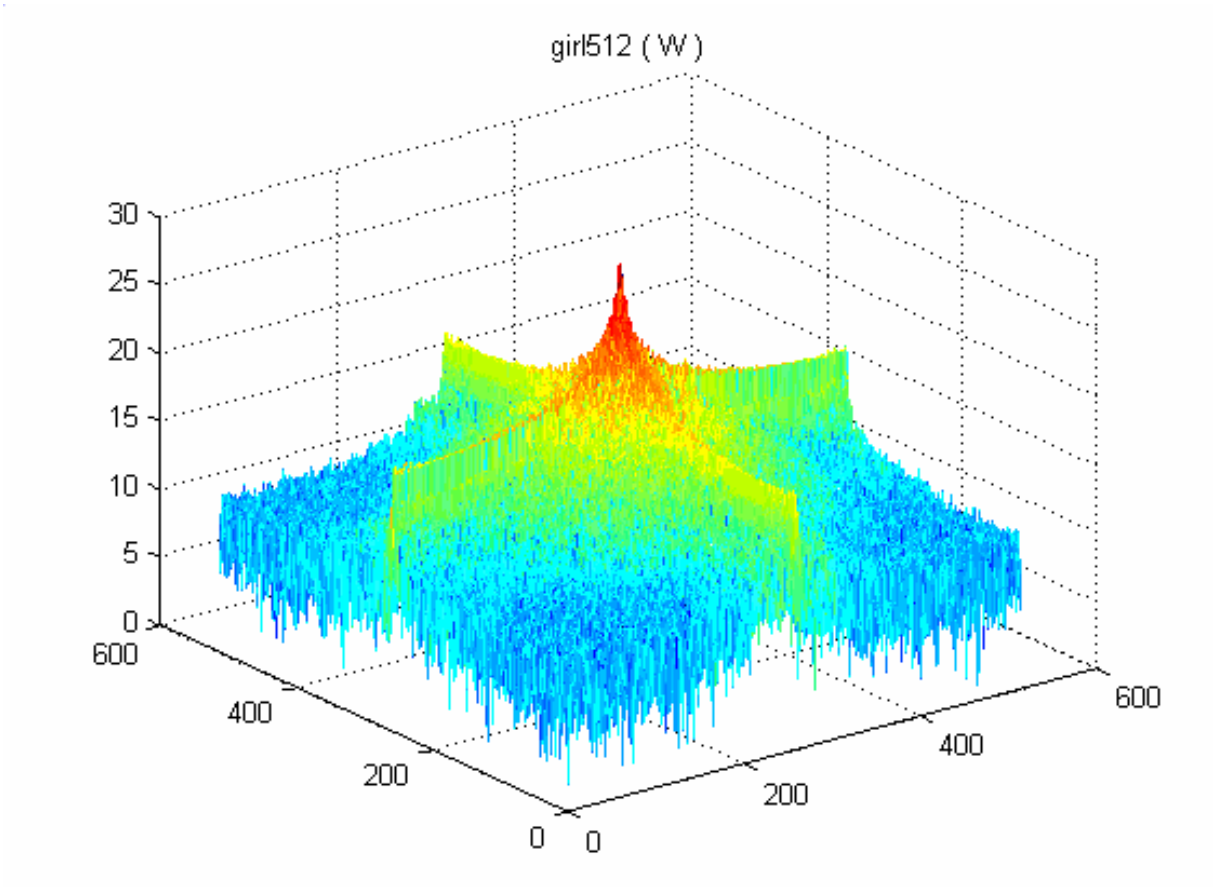


FIGURA 4.40 – Girl512x512: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.41 – Girl512x512: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

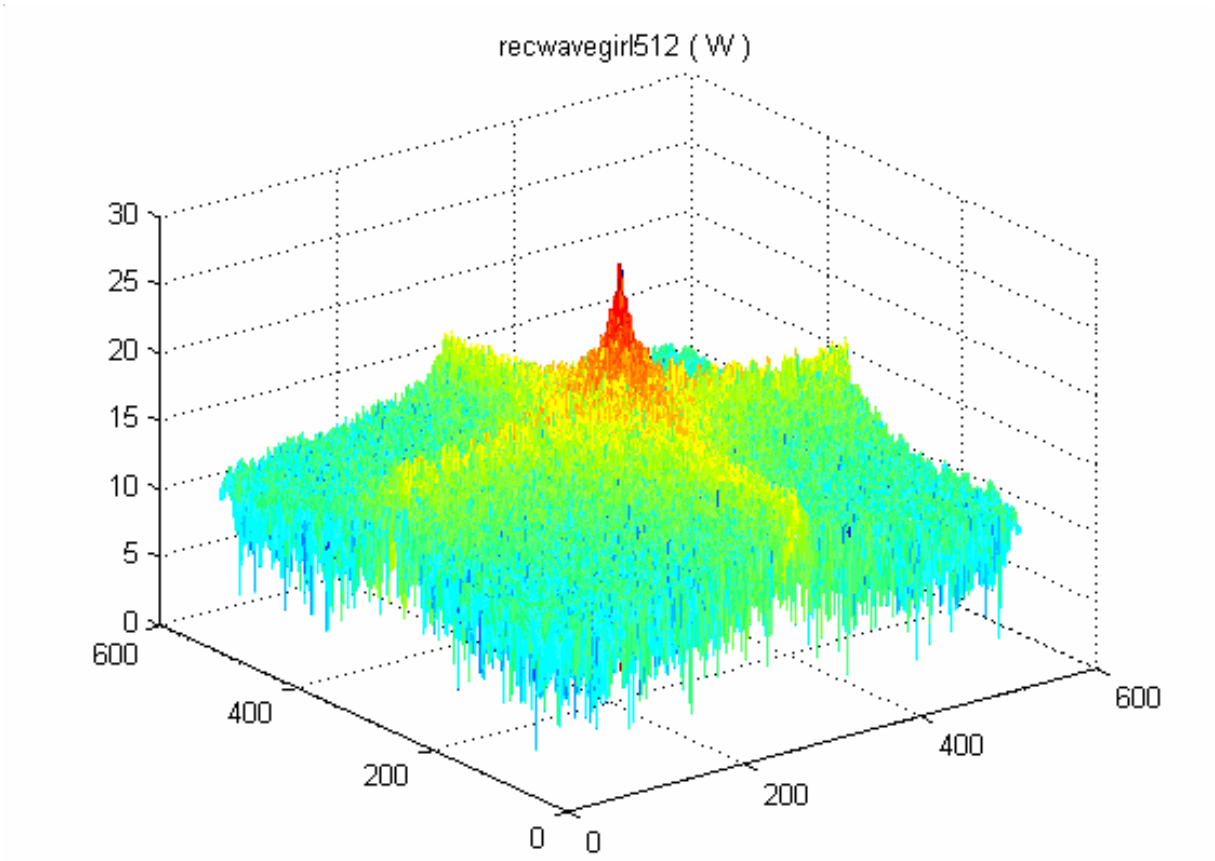


FIGURA 4.42 – Girl512x512: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.14 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Girl Original	263222	1.5450e -004
Girl Compressa por Huffman	231644	1.5450e -004
Girl Compressa Decomposta por *Wavelet* e codificada por Huffman	230648	4.7684e -005

TABELA 4.14 – Girl512x512: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.14, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $1{,}07 \times 10^{-4}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **8º estudo de caso:** Imagem Girl com resolução de 1024x1024 pixels:

A imagem Girl (com resolução de 1024x1024 pixels, conforme ilustra a FIGURA 4.43) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

FIGURA 4.43 – Girl1024x1024.bmp

A TABELA 4.15 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Girl1024x1024	1049654	920844	12,27%	913968	12,93%

TABELA 4.15 – Girl1024x1024: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 0,66% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.44 apresenta o Espectro de Wiener resultante da imagem Girl em resolução 1024x1024 pixels. As FIGURAS 4.45 e 4.46 ilustram respectivamente a imagem Girl descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.47 e 4.48 apresentam respectivamente a imagem Girl descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

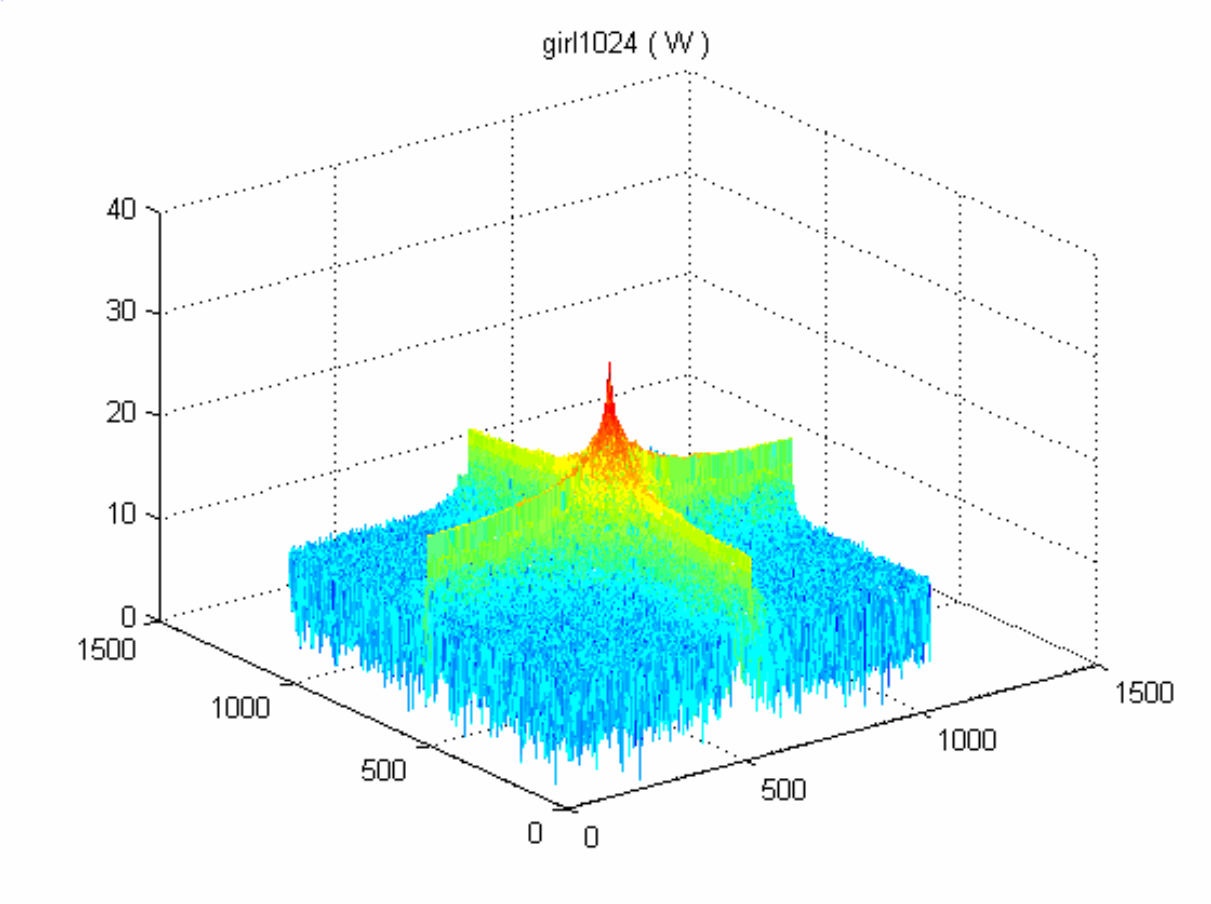


FIGURA 4.44 – Girl1024x1024: Espectro de Wiener resultante da imagem original.

FIGURA 4.45 – Girl1024x1024: Resultado da imagem descompressa após a codificação Huffman.

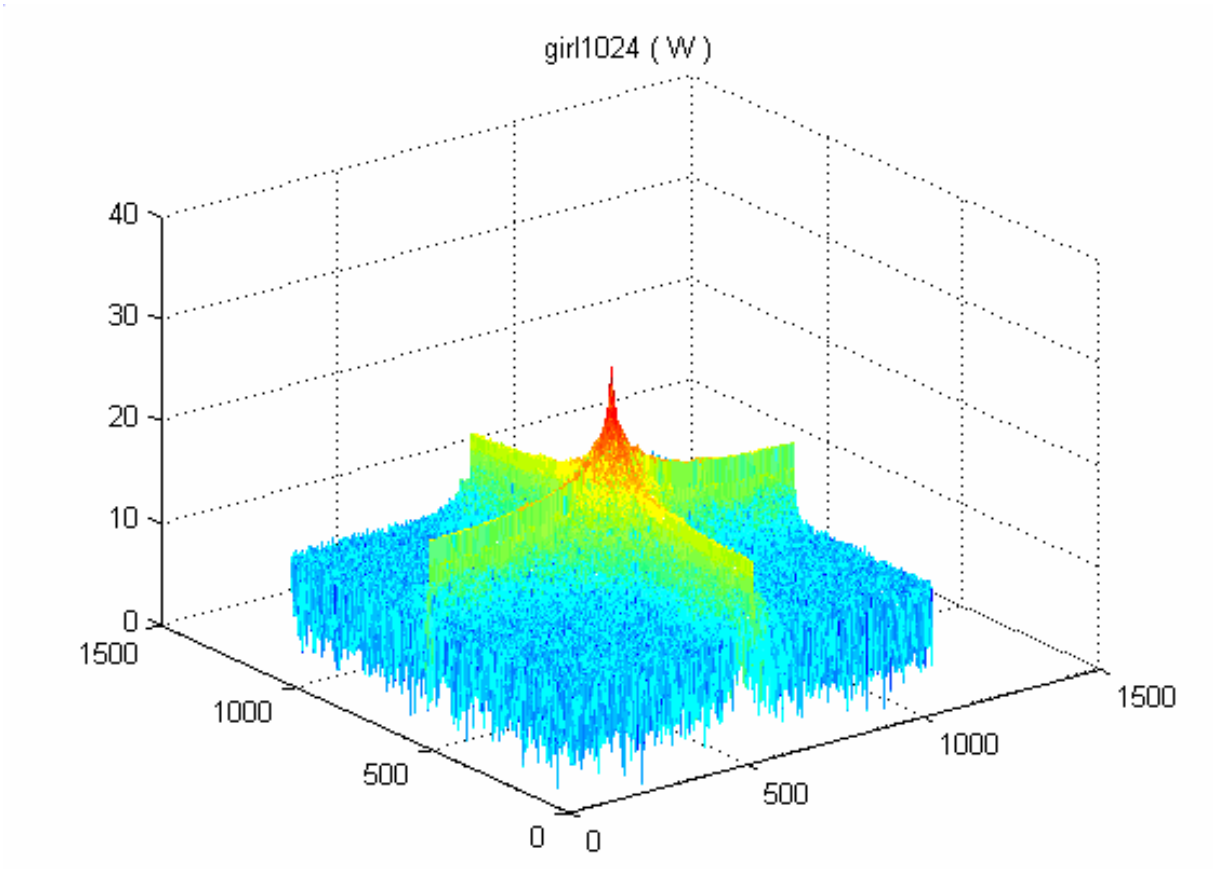


FIGURA 4.46 – Girl1024x1024: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

FIGURA 4.47 – Girl1024x1024: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

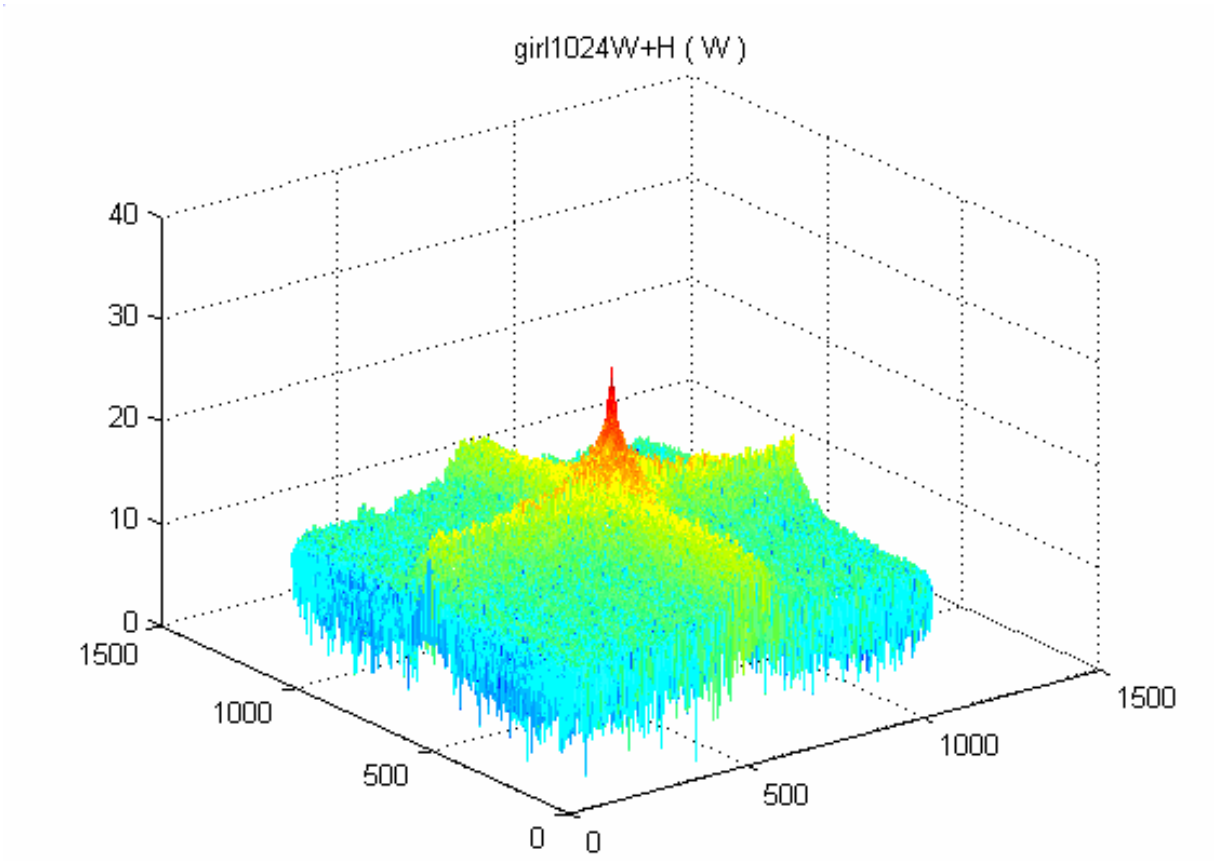


FIGURA 4.48 – Girl1024x1024: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.16 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Girl Original	1049654	0.0013
Girl Compressa por Huffman	920844	0.0013
Girl Compressa Decomposta por *Wavelet* e codificada por Huffman	913968	1.9836e -004

TABELA 4.16 – Girl1024x1024: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.16, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $1{,}10 \times 10^{-3}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **9º estudo de caso:** Imagem Solo com resolução de 128x128 pixels:

A imagem Solo (com resolução de 128x128 pixels, conforme ilustra a FIGURA 4.49) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

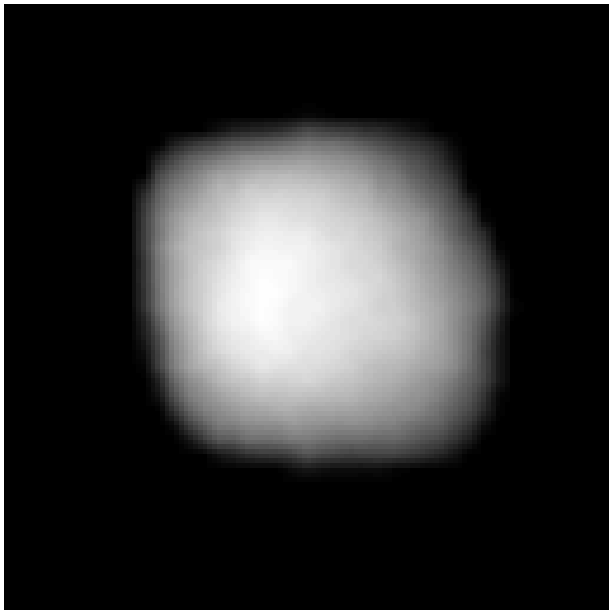


FIGURA 4.49 – Solo128x128.bmp

A TABELA 4.17 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Solo128x128	17462	9056	48,14%	6956	60,16%

TABELA 4.17 – Solo128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 12,02% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.50 apresenta o Espectro de Wiener resultante da imagem Solo em resolução 128x128 pixels. As FIGURAS 4.51 e 4.52 ilustram respectivamente a imagem Solo descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.53 e 4.54 apresentam respectivamente a imagem Solo descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

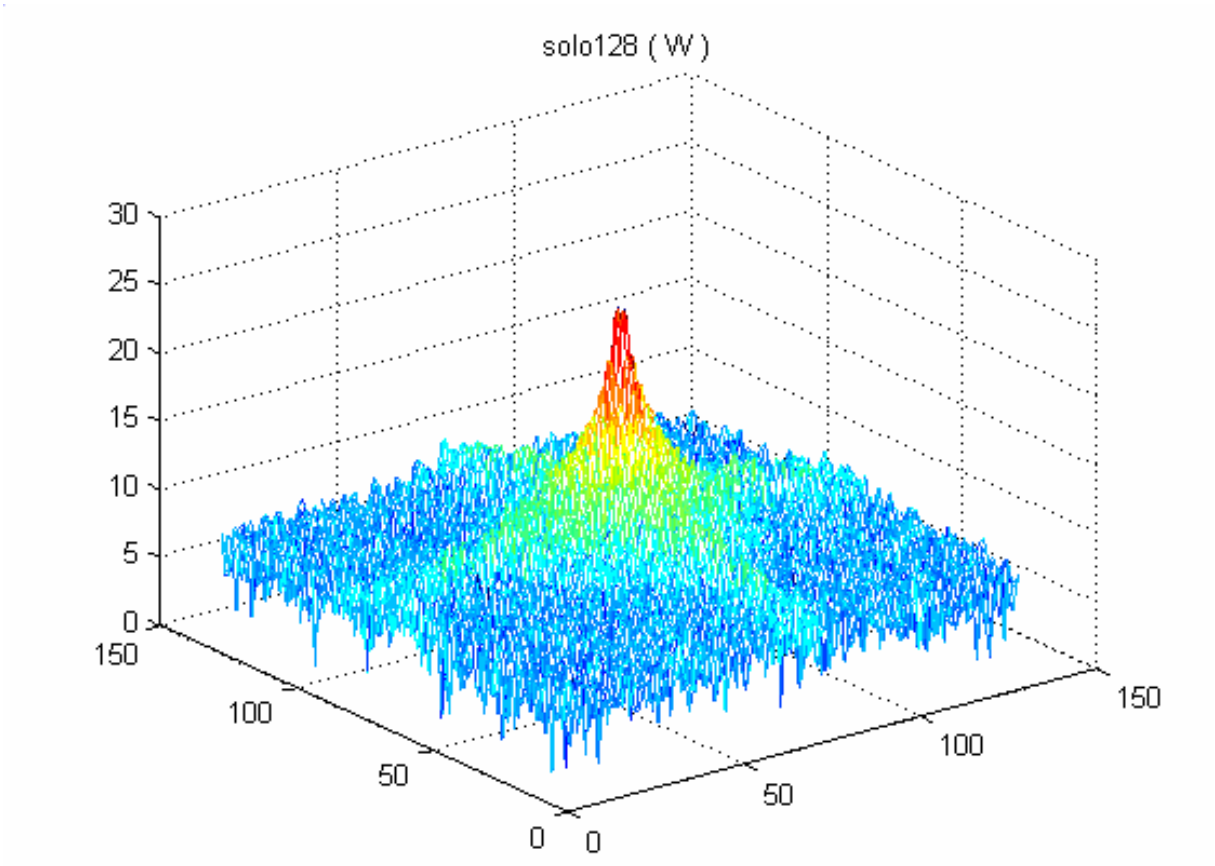


FIGURA 4.50 – Solo128x128: Espectro de Wiener resultante da imagem original.

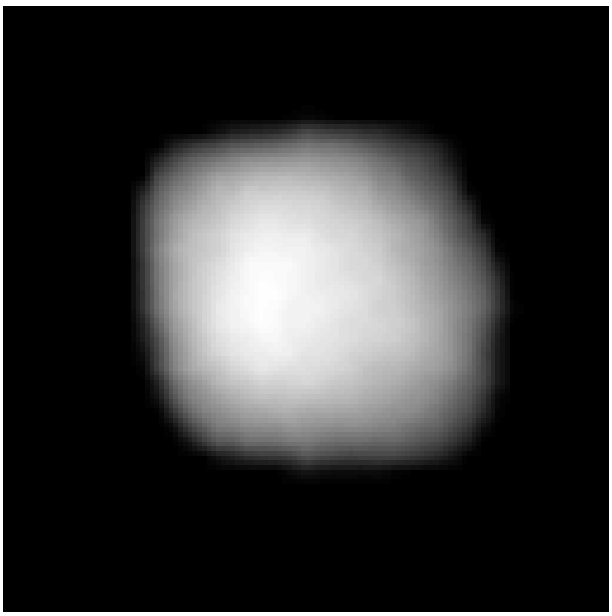


FIGURA 4.51 – Solo128x128: Resultado da imagem descompressa após a codificação Huffman.

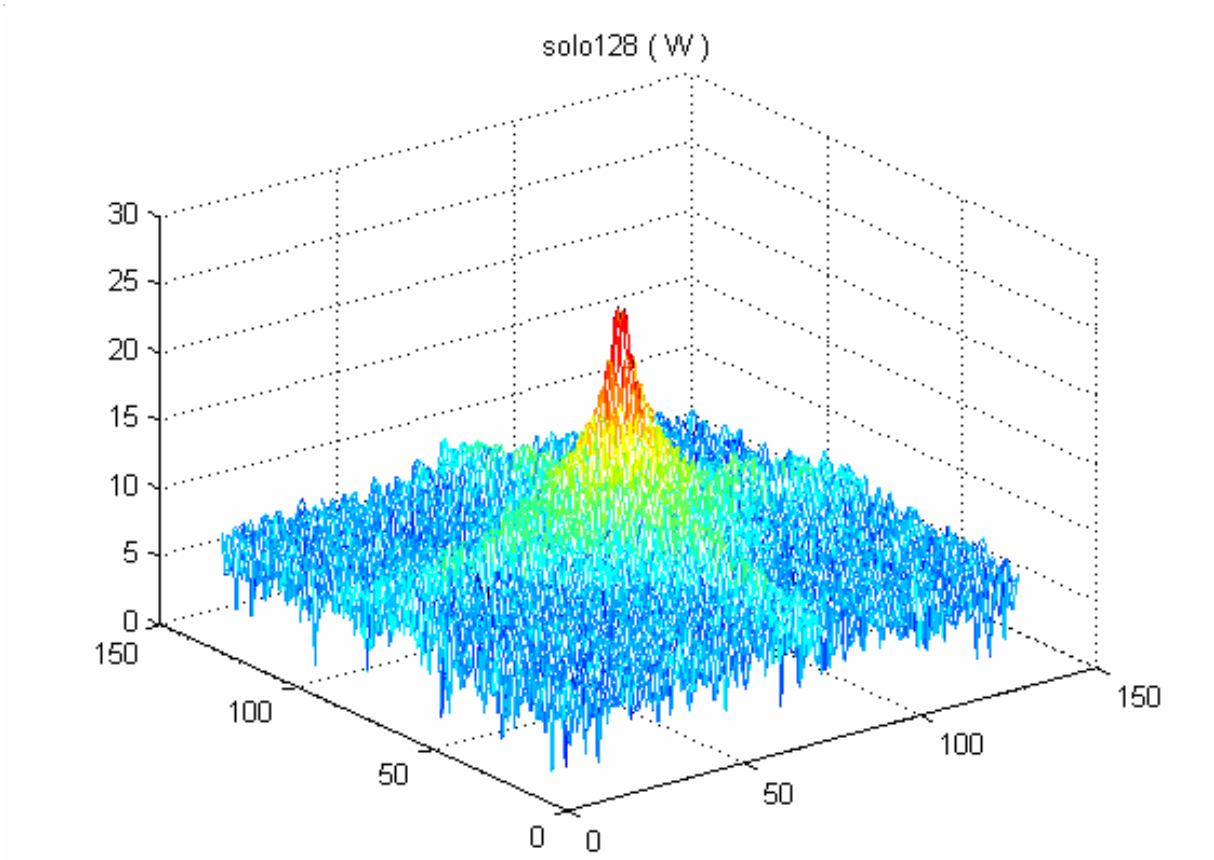


FIGURA 4.52 – Solo128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

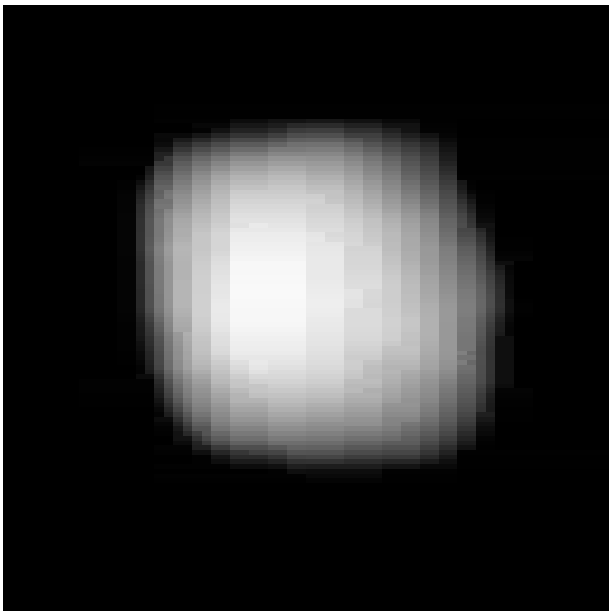


FIGURA 4.53 – Solo128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

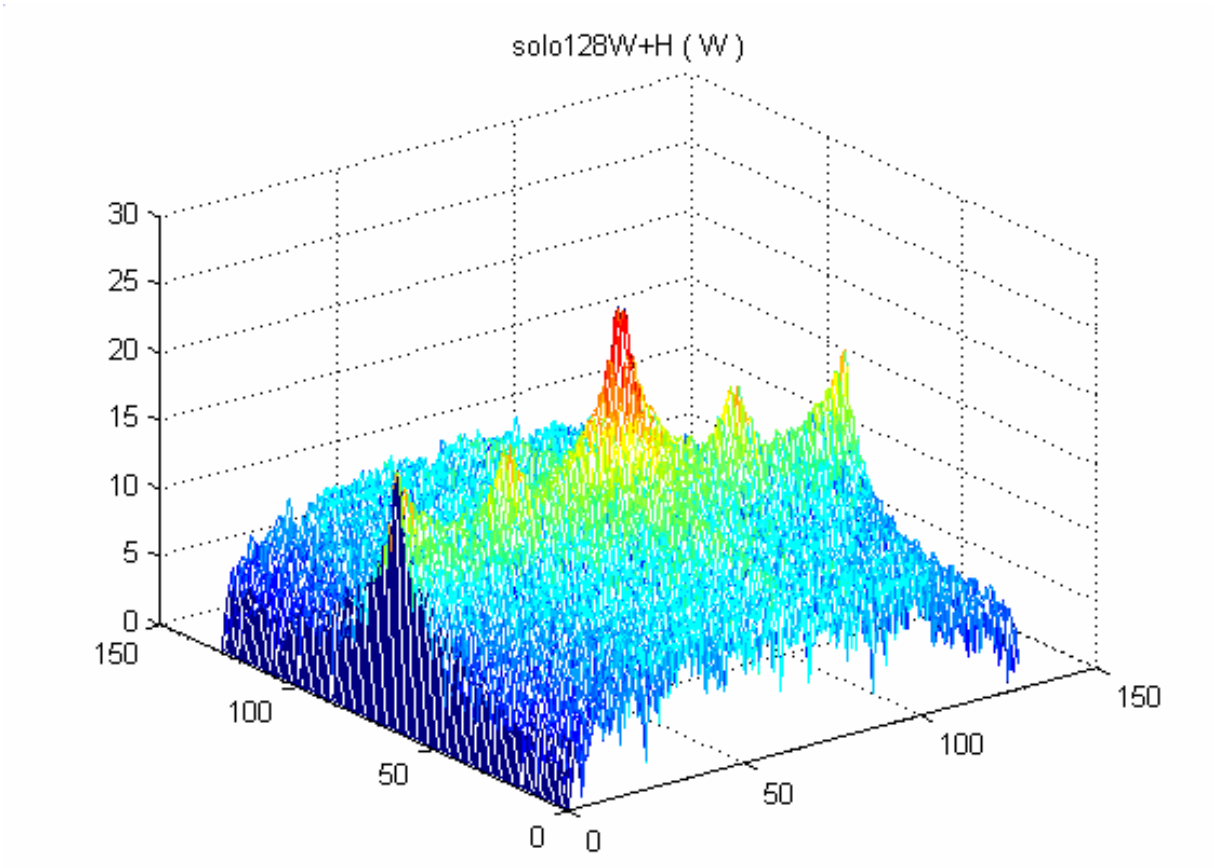


FIGURA 4.54 – Solo128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.18 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Solo Original	17462	1.0431e -006
Solo Compressa por Huffman	9056	1.0431e -006
Solo Compressa Decomposta por *Wavelet* e codificada por Huffman	6956	5.3644e -007

TABELA 4.18 – Solo128x128: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.18, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $5{,}07 \times 10^{-7}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

- **10º estudo de caso:** Imagem Phantom com resolução de 128x128 pixels:

A imagem Phantom (com resolução de 128x128 pixels, conforme ilustra a FIGURA 4.55) foi submetida à compressão pela codificação de Huffman, bem como pela decomposição *Wavelet* seguida pela decodificação de Huffman.

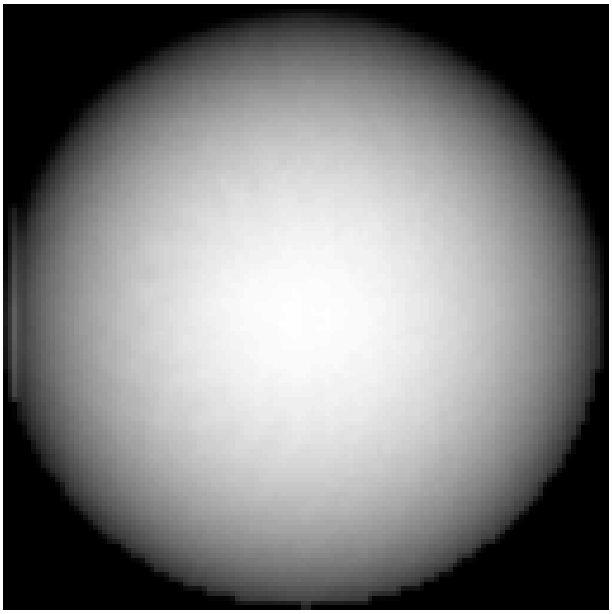


FIGURA 4.55 – Phantom128x128.bmp

A TABELA 4.19 ilustra os resultados obtidos considerando para estes dois processos a taxa de compressão resultante.

Imagem Original	*Tamanho da Imagem Original (bytes)*	*Tamanho da Imagem após a compressão com Huffman (bytes)*	*Taxa de Compressão (Huffman)*	*Tamanho da Imagem após a decomposição Wavelet seguida da compressão com Huffman (bytes)*	*Taxa de Compressão (Wavelet + Huffman)*
Phantom128x128	17462	15512	11,17%	12360	29,22%

TABELA 4.19 – Phantom128x128: Resultados obtidos considerando a compressão codificada por Huffman e a compressão obtida resultante da decomposição *Wavelet* seguida da codificação Huffman.

Conforme resultado apresentado, observa-se que houve um ganho de 18,05% na taxa de compressão obtida pela aplicação da decomposição *Wavelet* em conjunto com a compressão pela codificação de Huffman.

Para avaliação da qualidade da imagem após a descompressão, elaborou-se os Espectros de Wiener resultantes da imagem compressa pela codificação e Huffman e da imagem compressa pela decomposição *Wavelet* e codificação de Huffman, após suas descompressões.

A FIGURA 4.56 apresenta o Espectro de Wiener resultante da imagem Phanton em resolução 128x128 pixels. As FIGURAS 4.57 e 4.58 ilustram respectivamente a imagem Phanton descompressa da versão codificada por Huffman e seu Espectro de Wiener resultante. As FIGURAS 4.59 e 4.60 apresentam respectivamente a imagem Phanton descompressa da versão onde se aplicou a decomposição *wavelet* e a compressão Huffman e seu Espectro de Wiener resultante.

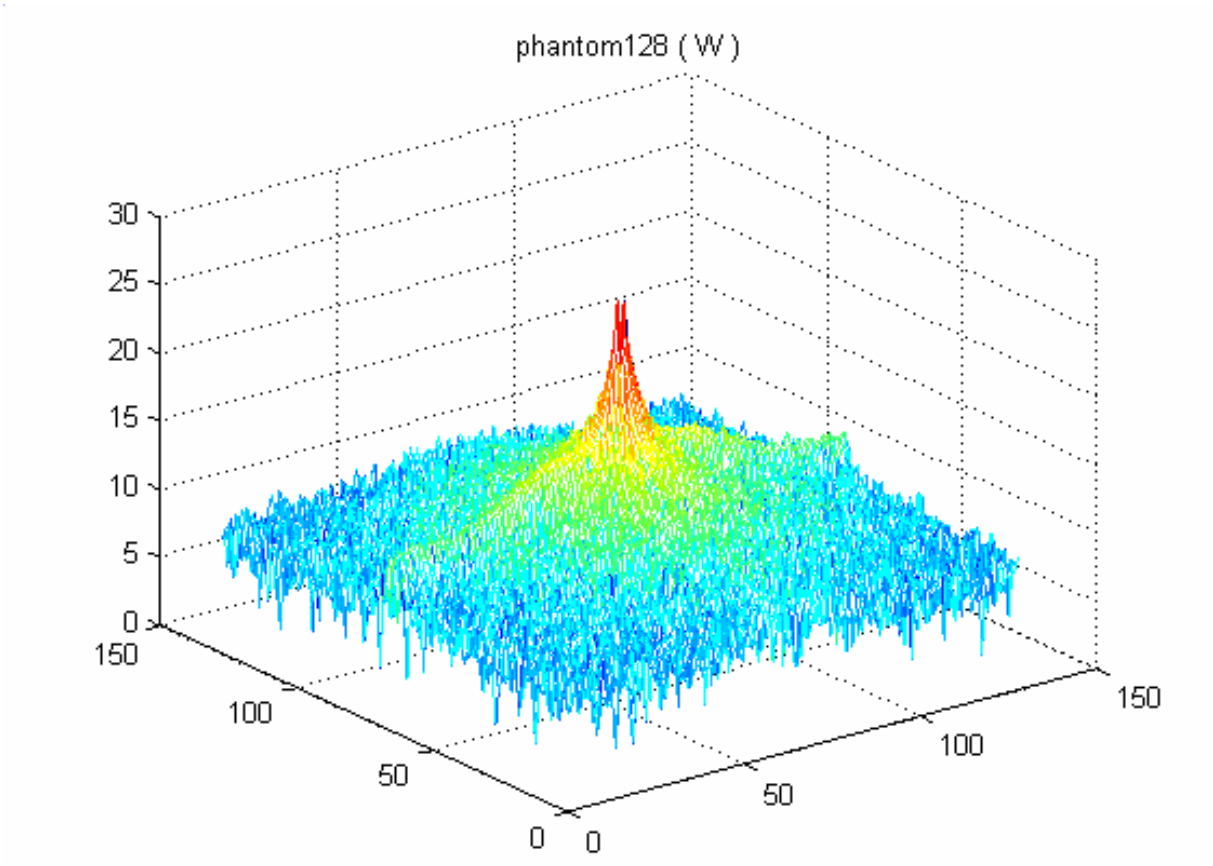


FIGURA 4.56 – Phantom128x128: Espectro de Wiener resultante da imagem original.

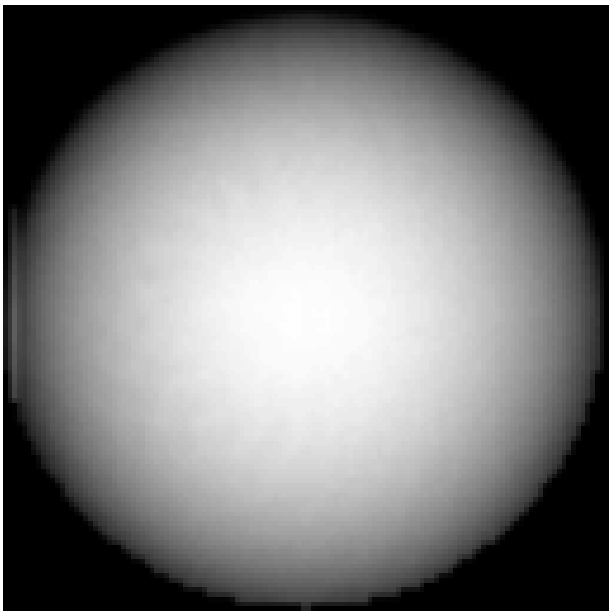


FIGURA 4.57 – Phantom128x128: Resultado da imagem descompressa após a codificação Huffman.

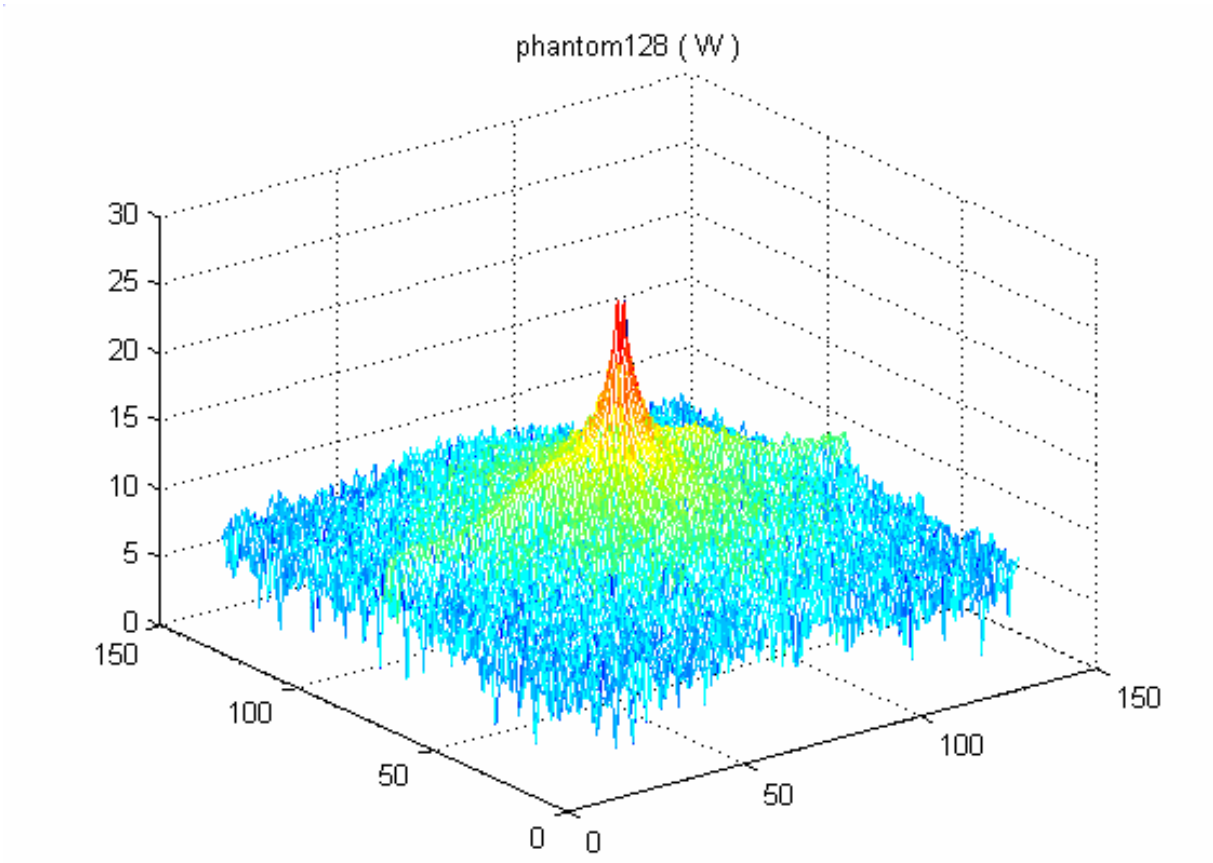


FIGURA 4.58 – Phantom128x128: Espectro de Wiener resultante da imagem descompressa após a codificação Huffman.

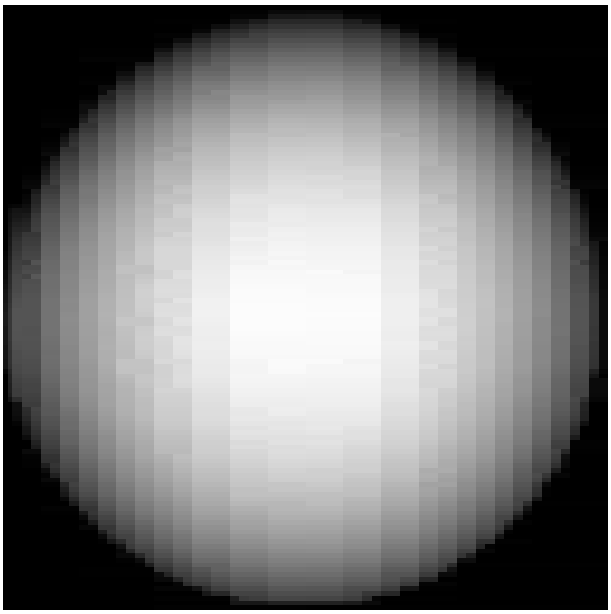


FIGURA 4.59 – Phantom128x128: Resultado da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

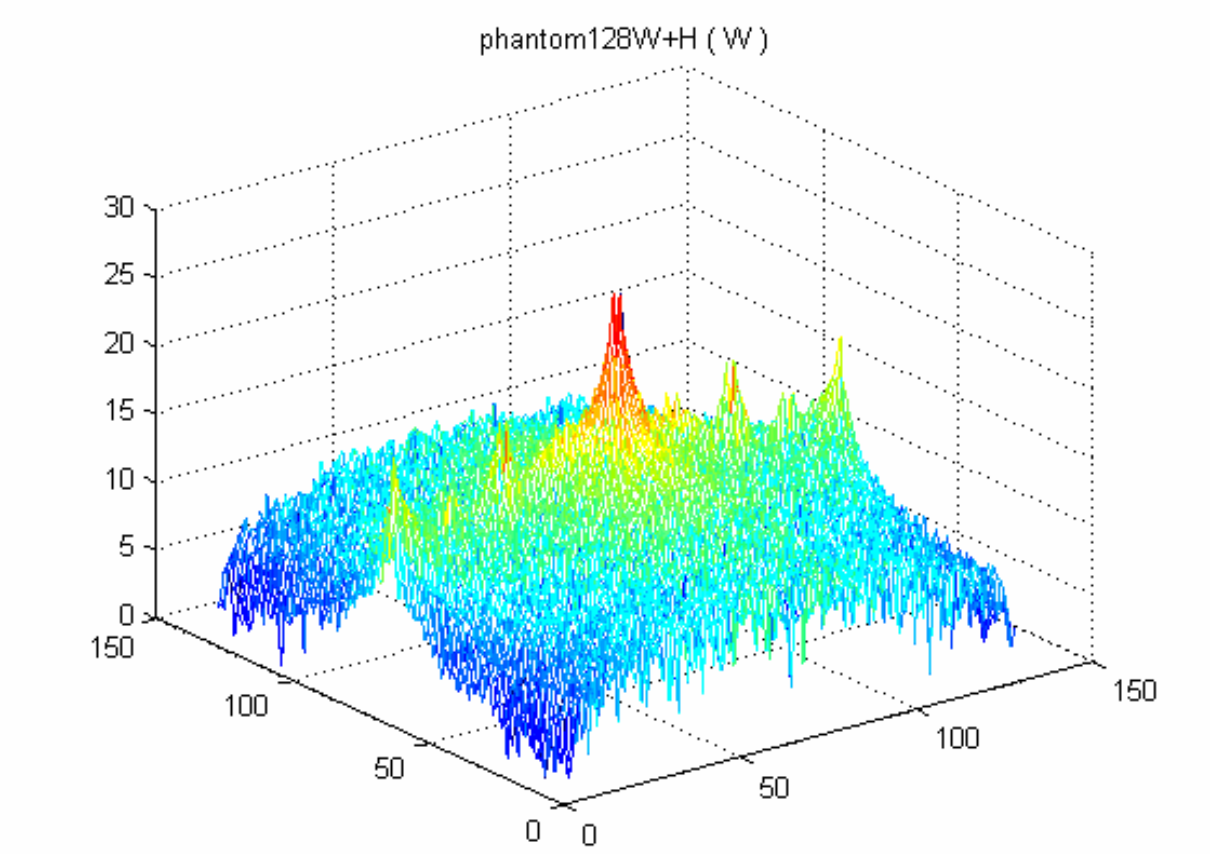


FIGURA 4.60 – Phantom128x128: Espectro de Wiener resultante da imagem descompressa após a decomposição *wavelet* e a compressão Huffman.

A TABELA 4.20 ilustra o resultado obtido quanto à análise da qualidade das imagens como indicadores da qualidade dos métodos de compressão indicados, o que é obtido com base no coeficiente de Wiener apresentado.

Imagem	***Tamanho da Imagem (bytes)***	***Coeficiente Wiener***
Phantom Original	17462	1.3113e -006
Phantom Compressa por Huffman	15512	1.3113e -006
Phantom Compressa Decomposta por *Wavelet* e codificada por Huffman	12360	7.1526e -007

TABELA 4.20 – Phantom128x128: Resultados obtidos quanto à análise da qualidade das imagens.

Conforme observado nos resultados apresentados na TABELA 4.20, o coeficiente de Wiener obtido após descompressão da imagem pelo método Compresso por Huffman não se distanciou do coeficiente de Wiener da imagem original. Por outro lado, o método de compressão com base em Huffman e *Wavelets*, em que pese à obtenção de uma taxa de compressão maior, proporcionou uma diferença entre os coeficientes de Wiener da imagem descompressa em relação à imagem original de $5{,}96 \times 10^{-7}$. Isto indica a ordem de degradação decorrente do processo de compressão do método.

4.2. Síntese sobre os resultados obtidos nos estudos de casos considerados

A TABELA 4.21 ilustra os resultados obtidos considerando as taxas de compressão em termos percentuais para as imagens Lena e Girl nas resoluções espaciais de 128x128, 256x256, 512x512 e 1024x1024 pixels, resultantes da compressão por codificação Huffman e do uso conjunto da decomposição *Wavelet* e codificação Huffman

Imagem	**Taxa de Compressão Huffman (%)**	**Taxa de Compressão *Wavelet* + Huffman (%)**
Lena 128x128 pixels	8,95	9,22
Lena 256x256 pixels	7,69	8,42
Lena 512x512 pixels	7,01	7,47
Lena 1024x1024 pixels	6,98	7,26
Girl 128x128 pixels	12,95	14,33
Girl 256x256 pixels	11,50	12,08
Girl 512x512 pixels	12,00	12,38
Girl 1024x1024 pixels	12,27	12,93

TABELA 4.21 – Resultados das taxas de compressão dos métodos utilizados.

A FIGURA 4.61 ilustra graficamente a síntese quanto ao ganho em termos percentuais sobre a taxa de compressão para as imagens Lena e Girl nas resoluções espaciais de 128x128, 256x256, 512x512 e 1024x1024pixels.

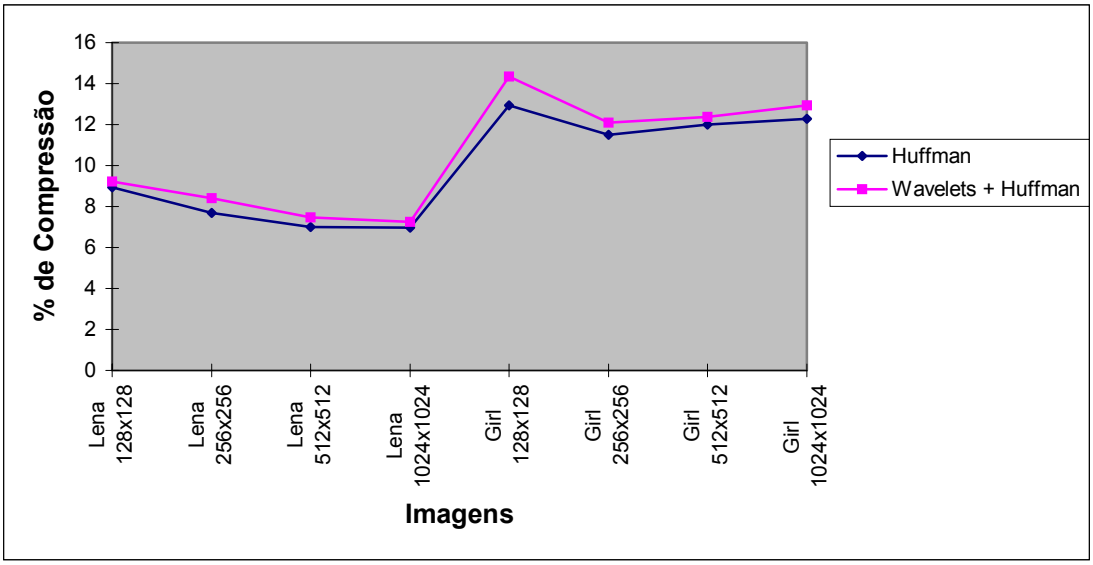


FIGURA 4.61 - Resultados das taxas de compressão.

A TABELA 4.22 ilustra os coeficientes de Wiener obtidos da imagem original, da decodificação por Huffman e do uso conjunto da decomposição *Wavelet* e decodificação Huffman para as imagens Lena e Girl nas resoluções espaciais de 128x128, 256x256, 512x512 e 1024x1024 pixels.

Imagem	**Coeficiente Wiener Imagem Original**	**Coeficiente Wiener Huffman**	**Coeficiente Wiener *Wavelet* + Huffman**
Lena 128x128 pixels	1.1027e -006	1.1027e -006	2.0862e -007
Lena 256x256 pixels	7.6294e -006	7.6294e -006	4.2915e -006
Lena 512x512 pixels	9.7275e -005	9.7275e -005	3.8147e -006
Lena 1024x1024 pixels	0.0024	0.0024	0.0010
Girl 128x128 pixels	1.5795e - 006	1.5795e - 006	7.4506e -007
Girl 256x256 pixels	1.1683e -005	1.1683e -005	7.1526e -007
Girl 512x512 pixels	1.5450e -004	1.5450e -004	4.7684e -005
Girl 1024x1024 pixels	0.0013	0.0013	1.9836e -004

TABELA 4.22 – Resultados dos coeficientes de Wiener dos métodos utilizados.

A FIGURA 4.62 ilustra graficamente a síntese sobre a variação do coeficiente de Wiener obtidos com a análise dos Espectros de Wiener das imagens originais e decompressas decorrentes dos métodos de Huffman e *Wavelets*.

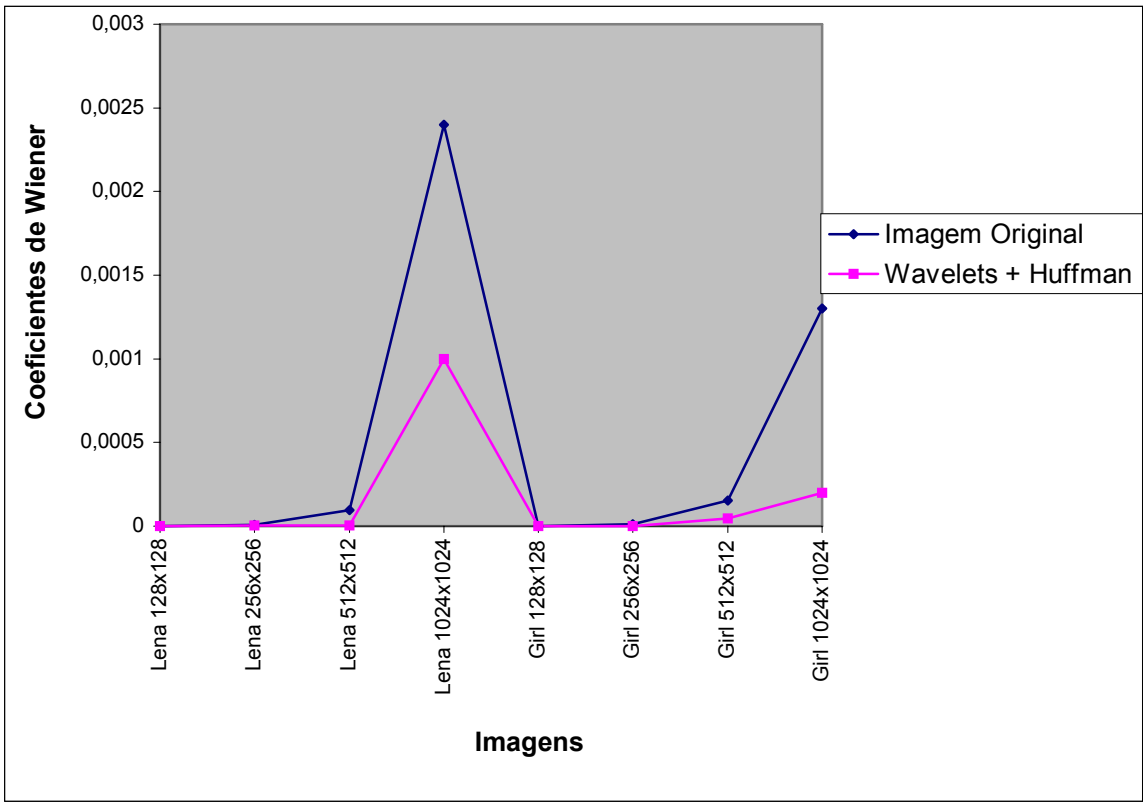


FIGURA 4.62 - Resultados dos coeficientes de Wiener.

A TABELA 4.23 ilustra o tempo utilizado para o processo de compressão das imagens digitais utilizando o uso conjunto das técnicas de decomposição *Wavelet* e codificação por Huffman para as imagens Lena e Girl nas resoluções espaciais de 128x128, 256x256, 512x512 e 1024x1024 pixels e solo e phantom na resolução 128x128 pixels.

Imagem	Tempo de Compressão
Lena 128x128 pixels	65 ns
Lena 256x256 pixels	160 ns
Lena 512x512 pixels	823 ns
Lena 1024x1024 pixels	1.765 ns
Girl 128x128 pixels	42 ns
Girl 256x256 pixels	180 ns
Girl 512x512 pixels	606 ns
Girl 1024x1024 pixels	1.662 ns
Solo 128x128 pixels	66 ns
Phantom 128x128 pixels	34 ns

TABELA 4.23 – Resultados dos tempos utilizados para a compressão.

4.3. Conclusões

Os resultados obtidos e apresentados nos estudos de caso considerados ilustram que o uso conjunto das técnicas de decomposição *Wavelets* seguido da compressão pelo método de Huffman proporcionam significativos ganhos quanto ao percentual de compressão de imagens digitais. Desta forma, quanto à taxa de compressão obtida no uso conjunto das duas técnicas citadas acima, para as imagens Lena e Girl, observou-se para as resoluções de 128x128, 256x256, 512x512 e 1024x1024 pixels, taxas de compressão da ordem de 9,22%, 8,42%, 7,47% e 7,26%, bem como 14,33%, 12,08%, 12,38% e 12,93% respectivamente.

Por outro lado, a análise da qualidade com base no Espectro de Wiener indicou pelos resultados obtidos pelos estudos de caso que o método com base no uso conjunto das técnicas de decomposição *Wavelets* seguido pela compressão Huffman proporcionou degradação nas imagens compressas, variando entre os percentuais de 43,75% a 96,08% para o conjunto de imagens com resoluções estudadas.

Assim, conclui-se que a junção dos métodos de decomposição com *Wavelets*, originalmente com perda e, compressão de Huffman, originalmente sem perda, proporcionou um método que amplia a taxa de compressão. Esses percentuais são maiores se comparados às taxas de compressão obtidas utilizando-se isoladamente o método de compressão com base no algoritmo de Huffman. A degradação da imagem resultante da junção dos métodos citados anteriormente também foi de menor ordem quando comparado com o uso do método de decomposição fundamentado exclusivamente em técnica de decomposição *Wavelets*.

Adicionalmente, a estruturação do método desenvolvido encontrou baixo custo computacional devido à diminuição da complexidade proporcionada pelo uso da Transformada *Wavelet* Haar no processo de decomposição das imagens, bem como pelo uso da codificação de Huffman.

4.4. Sugestões de trabalhos futuros

1. Explorar outras famílias de *Wavelets* na composição de alternativas para a junção com o método de Huffman objetivando maximizar a compressão de imagens digitais.

2. Utilizar a Transformada *Wavelet* com a finalidade de reduzir ruído para melhoria de figura de método de perdas por compressão.

APÊNCIDE A

Lista de Símbolos

Z	Conjunto dos Números Inteiros
$\Re$	Conjunto dos Números Reais
∞	Infinito
$\subset$	Contém
$\cap$	Intersecção
$\in$	Pertence
$\oplus$	Soma Ortogonal
$\sum_{j=1}^{J}$	Somatória com valor j variando de forma crescente de 1 à J
ψ	*Wavelet* mãe
$\|\psi\|$	Norma de *Wavelet*
$\langle \psi_{j,k}, \psi_{j,k`} \rangle$	Produto Interno de $\psi_{j,k}$ e $\psi_{j,k'}$
ϕ	Função de Escalonamento
V_j	Espaço de todas as combinações lineares dos $\phi_{jk}(t)$ ao nível j
W_{j-1}	Espaço de todas as combinações lineares dos $\psi_{j-1,k}(t)$ ao nível $j-1$
Δ_j	Detalhes perdidos no processo de multiresolução

APÊNCIDE B

Lista de Fórmulas

Redundância de dados relativa: $R_D = 1 - \frac{1}{C_R}$

Taxa de compressão: $C_R = \frac{n_1}{n_2}$

Probabilidade de ocorrência dos níveis de cinza de uma imagem: $p_r(r_k) = \frac{n_k}{n}$

Número médio de bits necessários para representar cada *pixel*: $L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$

Média do comprimento do código: $\overline{L} = \sum_{i=0}^{2^B - 1} L(i) p(i)$

Famílias de funções *Wavelets*: $\psi_{a,b}(x) = |a|^{-\frac{1}{2}} \psi\left(\frac{x-b}{a}\right)$

Teorema da multiresolução: $\sum_k a_{jk} \phi_{jk}(t) = \sum_k a_{j-1,k} \phi_{j-1,k}(t) + \sum_k b_{j-1,k} \psi_{j-1,k}(t)$

APÊNCIDE C

Códigos

Abaixo, segue o código do processo de Compressão e Descompressão da imagem utilizando a codificação e decodificação de Huffman:

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciências da Computação
Departamento de Computação

"Método para Compressão de Imagens Digitais fundamentado em Procedimentos de Huffman e Wavelets"

Mestranda: Luciana Aparecida de Oliveira Betetto
Orientador: Prof. Dr. Paulo Estevão Cruvinel

```
#include <stdio.h>
#include <windows.h>
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
#include <search.h>
#include <mmsystem.h>
#include <classes.hpp>
#include <vcl.h>

typedef struct tagHuffTreeNode {
BYTE symbol;
DWORD freq;
tagHuffTreeNode *pLeft;
```

```
tagHuffTreeNode *pRight;
tagHuffTreeNode *pNext; // utilizado na construção da FILA
} HuffTreeNode; // Estrutura do nó da Árvore e da FILA

typedef struct tagHuffCode {
WORD code; // Codigo referente ao símbolo [combinação dos rótulos do caminho] (Codificação da cor (0&1))
BYTE size; // // Tamanho do codigo
BYTE symbol; // Tons de cinza
} HuffCode; // Estrutura da Tabela codificada

typedef struct tagLogPal{
  TLogPalette lpal;
  TPaletteEntry dummy[256];
} LogPal;
#define FOURCC_HUFF 0x46465548  // 'HUFF'

// Remove nó (pNode) da FILA (ppQueue)
void PQueueRemove (HuffTreeNode *pNode, HuffTreeNode **ppQueue) {
                HuffTreeNode *pBack = NULL;
                HuffTreeNode *pTemp = *ppQueue;
                while (pNode != pTemp) {
                        pBack = pTemp;
                        pTemp = pTemp->pNext;
                }
                if (pNode == pTemp) {
                        if (pBack != NULL)
                                pBack->pNext = pTemp->pNext;  //remoção do meio ou extremidade
                        else
                                *ppQueue = pTemp->pNext; //remoção do primeiro elemento
                        pNode->pNext = NULL;
                }
}
// ---------------------------------------------------------------------
```

```
// Insere nó (pNode) na FILA (ppQueue)
void PQueueInsert (HuffTreeNode *pNode, HuffTreeNode **ppQueue) {
        HuffTreeNode *pTemp, *pBack;
                if (*ppQueue == NULL) {
                *ppQueue = pNode;
                pNode->pNext = NULL;
        }
                else {
                pBack = NULL;
                pTemp = *ppQueue;
                while (( pTemp->pNext != NULL)  && ( pTemp->freq < pNode-
>freq)) {
                        pBack = pTemp;
                        pTemp = pTemp->pNext;
                }
                if (pTemp->pNext == NULL && pTemp->freq < pNode->freq) {
                   pTemp->pNext = pNode; // inserção na extremidade
                        } else {
                        pNode->pNext = pTemp;
                        if (pBack != NULL)
                                pBack->pNext = pNode; // inserção no meio
                        else
                                *ppQueue = pNode; // inserção no inicio
                }
            }
}
// ---------------------------------------------------------------------
DWORD PQueueGetSize(HuffTreeNode *pQueue) {
        DWORD i = 0;
        while (pQueue != NULL) {
                i++;
                pQueue = pQueue->pNext;
        }
        return i;
}
```

```
// ---------------------------------------------------------------------
// Constrói a FILA a partir da frequência
// QuantidadeSimbolosUnicos = BuildPQueue (&pPQueue, freq);
// pPQueue = fila
// freq = vetor com a quantidade de ocorrência de cada cor

DWORD BuildPQueue(HuffTreeNode **ppPQueue, DWORD freq[256]) {
        DWORD i, elements = 0;
        HuffTreeNode *pNewNode, *pQueue = NULL;
        for (i=0; i<=255; i++) {
            if (freq[i] > 0) {
                pNewNode = (HuffTreeNode *)malloc (sizeof(HuffTreeNode));
                pNewNode->freq = freq[i];
                pNewNode->symbol = (BYTE)i;
                pNewNode->pLeft = NULL;
                pNewNode->pRight = NULL;
                pNewNode->pNext = NULL;
                PQueueInsert (pNewNode, &pQueue); // Insere NewNode na fila
                elements++;
            }
}
*ppPQueue = pQueue;
return elements; // retorna a quantidade de nós da fila (símbolos da figura)
}
// ---------------------------------------------------------------------
// Converte a FILA em ÁRVORE de Huffman
void ConvertPQueueToTree (HuffTreeNode **ppPQueue, DWORD QueueSize) {
        DWORD i;
        HuffTreeNode *pNewNode, *pLeftNode, *pRightNode, *pPQueue;
        pPQueue = *ppPQueue;
        // Construção dos "pais" da ÁRVORE
            for (i=1; i<QueueSize; i++) {
            pNewNode = (HuffTreeNode *)malloc (sizeof(HuffTreeNode));
            pLeftNode = pPQueue;
            PQueueRemove (pLeftNode, &pPQueue);
```

```
                    pRightNode = pPQueue;
                    PQueueRemove (pRightNode, &pPQueue);
                    pNewNode->pLeft = pLeftNode;
                    pNewNode->pRight = pRightNode;
                    pNewNode->pNext = NULL; // Retira o argumento da fila
                    pNewNode->freq = pLeftNode->freq + pRightNode->freq;
                    pNewNode->symbol = 0;
                    PQueueInsert (pNewNode, &pPQueue);
                    }
*ppPQueue = pPQueue; // ppPQueue retorna como o nó raiz da árvore
}
// ---------------------------------------------------------------------
HuffCode *gpHuffTable; // Tabela obtida apartor da arvore de Huffman
DWORD gHuffTableElements = 0; // Indice do vetor da tabela
DWORD CompressedFileSizeInBits = 0;
void ConvertToTable (HuffTreeNode *pTree, DWORD code, DWORD &CodeLen) {
            if (pTree->pLeft != NULL) {
            CodeLen++;
            code = code << 1; // multiplica por 2
            ConvertToTable (pTree->pLeft, code, CodeLen);
            code += 0x01; // soma 1 em hexa
            ConvertToTable (pTree->pRight, code, CodeLen);
            CodeLen-- ;
            }
            else {
            gpHuffTable[gHuffTableElements].symbol = pTree->symbol; // Tons de
cinza
            gpHuffTable[gHuffTableElements].code = (WORD)code; // Codificação
da cor (0&1)
            gpHuffTable[gHuffTableElements].size = (BYTE)CodeLen; // Tamanho
do código
            CompressedFileSizeInBits += pTree->freq * CodeLen;
            gHuffTableElements++; // Indice do vetor da tabela
            // remova da árvore e desaloque da memória
            free (pTree);
            }
```

```
}
// ----------------------------------------------------------------------
// Comparação utilizada no Quick Sort [funcao qsort()]
int __cdecl compfn (const void *p1, const void *p2) {
        return (((HuffCode *)p1)->size - ((HuffCode *)p2)->size);
}
// ----------------------------------------------------------------------
void GetCode (HuffCode *pHuffTable, BYTE symbol, DWORD *pCode, DWORD *pLen) {
        while (pHuffTable->symbol != symbol)
            pHuffTable++; // Percorre a tabela até achar o símbolo
        *pCode = pHuffTable->code; // Codificação da cor (0&1)
        *pLen  = pHuffTable->size; // Tamanho do codigo
}
// ----------------------------------------------------------------------
void AppendBits (BYTE *pData, DWORD BitLoc, DWORD code, DWORD len) {
DWORD i;
        for (i = 0; i < len; i++) {
            DWORD WhichByte = BitLoc / 8;
            DWORD WhichBit = 7 - (BitLoc & 0x07);
            BYTE Bit = (BYTE)( code >> (len - i - 1) ) & 0x01;
            pData[WhichByte] |= Bit << WhichBit;
            BitLoc++;
}
}
// ----------------------------------------------------------------------
// Método de Huffman
// NumeroBytesComprimidos = HuffCompress (&pCompressedData, pData, FileSize);
// onde pCompressedData = Dados comprimidos
//      pData = Area de memória alocada para a imagem
//      FileSize = Tamanho da imagem

int HuffCompress (BYTE * *ppCData, BYTE *pData, DWORD NumBytes) {
DWORD i, NumUniqueSymbols;
DWORD freq[256];
```

```
HuffTreeNode *pTreeRoot;
HuffTreeNode *pPQueue;
BYTE *pCompressedData;

// Etapa 1: Construa o vetor da freqüência por o byte
        ZeroMemory (freq, 256 * sizeof(unsigned int));
        for (i=0; i < NumBytes; i++)
        freq[ pData[i] ]++; // Lembre-se que pData[i] = cor

// Etapa 2: Faça uma fila de prioridades baseada na utilização somente de freqüências diferente de zero
        NumUniqueSymbols = BuildPQueue (&pPQueue, freq);

// Etapa 3: Faça uma árvore usando a fila de prioridade
        ConvertPQueueToTree(&pPQueue, NumUniqueSymbols);
        pTreeRoot = pPQueue; // nó raiz da árvore

// Etapa 4: Faça a tabela baseada na árvore
        gpHuffTable = (HuffCode *)malloc(NumUniqueSymbols * sizeof(HuffCode));
        // ConvertToTable modifica as variáveis globais:
        // CompressedFileSizeInBits, gpHuffTable, gHuffTableElements, pTreeRoot
        ConvertToTable (pTreeRoot, 0, 0); // pTreeRoot foi desalocado

// Etapa 5: Classifique a tabela para acesso rápido baseado na tabela da freqüência
        qsort(gpHuffTable, NumUniqueSymbols, sizeof(HuffCode), compfn);

// Etapa 6: Determine qual tamanho do arquivo comprimido e aloque na memória
DWORD HeaderSize = NumUniqueSymbols * sizeof(HuffCode) + 4 * sizeof(DWORD);
//
          int bitsComplementares = (CompressedFileSizeInBits % 8 );
          if ( bitsComplementares != 0 )
          {
              bitsComplementares = 8 - bitsComplementares;
          }
```

```
DWORD CompressedFileSizeInBytes = (CompressedFileSizeInBits + bitsComplementares) / 8 + HeaderSize;
        int bytesComplementares = (CompressedFileSizeInBytes % sizeof(DWORD) );
        if ( bytesComplementares != 0 )
        {
            bytesComplementares = sizeof(DWORD) - bytesComplementares;
        }
//
            DWORD CompressedFileSizeInDWORDS = (CompressedFileSizeInBytes + bytesComplementares ) / sizeof(DWORD);
            pCompressedData = (BYTE *)malloc(CompressedFileSizeInDWORDS * sizeof(DWORD));
            if (pCompressedData == NULL) {
            printf ("Falha na locação de %d bytes para dados comprimidos.\n", CompressedFileSizeInDWORDS * sizeof(DWORD));
            return 0;
}
ZeroMemory (pCompressedData, CompressedFileSizeInBytes);

// Etapa 7: Escreva o cabeçalho (header) no buffer
DWORD *pdwCD = (DWORD *)pCompressedData;
pdwCD[0] = FOURCC_HUFF; // 'HUFF'
        pdwCD[1] = NumBytes; // Tamanho do arquivo original
        pdwCD[2] = CompressedFileSizeInBits; // Contagem de bits comprimidos
        pdwCD[3] = NumUniqueSymbols; // Número de elementos no HuffTable
CopyMemory (pCompressedData + 16, gpHuffTable, NumUniqueSymbols * sizeof(HuffCode));

// Etapa 8: Converta os dados para bits e escreva no buffer
DWORD CurrentBitLoc = 0;
BYTE *pBits = pCompressedData + HeaderSize;
for (i = 0; i < NumBytes; i++) {
        DWORD code, len;
        GetCode(gpHuffTable, pData[i], &code, &len); // retorna em code o código de pData[i]
        AppendBits(pBits, CurrentBitLoc, code, len); //
```

```
            CurrentBitLoc += len;
            }
            *ppCData = pCompressedData;
printf("Tamanho original dos dados      %d bytes\n", NumBytes);
printf("Tamanho comprimido dos dados    %d bytes\n", (CompressedFileSizeInBits +
7) /8);
printf("Bytes originais nos dados       %d\n", NumUniqueSymbols);
printf("Tamanho do cabeçalho            %d bytes\n", HeaderSize);
printf("Tamanho total comprimido        %d bytes\n", CompressedFileSizeInBytes);
printf("Relação da compressão           %d%%\n", 100 - (CompressedFileSizeInBytes
* 100 / NumBytes) );
return CompressedFileSizeInDWORDS * sizeof(DWORD);
}
// -------------------------------------------------------------------
int HuffDescompress (BYTE * *ppCData, BYTE *pData, DWORD NumBytes) {
DWORD i, NumUniqueSymbols;
            DWORD freq[256];
            HuffTreeNode *pTreeRoot;
            HuffTreeNode *pPQueue;
            BYTE *pCompressedData;
}
bool VerificaCodigo( WORD CodigoBinario, BYTE &CodByte , int
NumberOfSymbols, int PalavraAtualTam)
{
            bool Achou = false; // busca a localização na tabela de simbolos
            for (int iSimboloAtual=0   ; ( Achou == false ) && iSimboloAtual <
NumberOfSymbols ; iSimboloAtual++ )
{
                  if ( (gpHuffTable[ iSimboloAtual ].code == CodigoBinario ) &&
(gpHuffTable[ iSimboloAtual ].size == PalavraAtualTam ))
                  {
                        CodByte = gpHuffTable[ iSimboloAtual ].symbol;
                        Achou = true;
                  }
                  else if (  gpHuffTable[ iSimboloAtual ].size > PalavraAtualTam )
                  {
                        iSimboloAtual = NumberOfSymbols;
```

```
                    }
}
return Achou;
}
int main( int argc, char* argv[])
{
char filename[128];
HANDLE hFile;
DWORD BytesRead, BytesWritten;
BYTE *pData, *pCompressedData;
DWORD FileSize, NumCompressedBytes;
DWORD StartTime, ElapTime;
BYTE *pLinha;
if ( argc < 3 )
{
            printf("Entre com os parametros de compressao(c) ou descompressão(d)
e o nome do arquivo \n" ) ;
            return -1;
}
else if ( strcmp( argv[1],"c" ) == 0 )
{

Graphics::TBitmap * bmp = new Graphics::TBitmap();
bmp->LoadFromFile((AnsiString ) argv[2]); // lendo apenas os dados
pData = new BYTE[bmp->Height* bmp->Width];
for ( int l =0 ; l < bmp->Height;l++ )
{
pLinha =( BYTE * ) bmp->ScanLine[l];
            for ( int c = 0; c < bmp->Width; c++ )
              {
                  pData[l * bmp->Width + c ] = pLinha[c];
              }
}
FileSize =  bmp->Height* bmp->Width;
strcpy(filename, argv[2]);
StartTime = timeGetTime();
```

```
NumCompressedBytes = HuffCompress (&pCompressedData, pData, FileSize);
// Determina a duração do processo de compressão
ElapTime = timeGetTime() - StartTime;
free (pData);
// escreva os dados comprimidos no disco
strcat (filename, ".huff");
hFile = CreateFile(filename, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
WriteFile(hFile, pCompressedData, NumCompressedBytes, &BytesWritten, NULL);
CloseHandle(hFile);
free (pCompressedData);
printf("Arquivo comprimido          '%s'\n", filename);
printf("Tempo da compressão          %d ms\n\n", ElapTime);

/*********************************************************************/
printf("\n\n");
BYTE *pDescompressedData;
DWORD NumDescompressedBytes;
//-------------------------------------------------------------------
strcpy(filename, argv[2]);
printf("Codigo de Huffman           '%s'\n", filename);

// Carregue o arquivo compactado
strcat (filename, ".huff");
hFile = CreateFile(filename, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
        printf ("Não foi possível abrir o arquivo.\n");
        return -1;
}
// Aloca memória para o arquivo
FileSize = GetFileSize(hFile, NULL);
pData = (BYTE *)malloc(FileSize);
if (pData == NULL) {
        printf ("Falha na alocação de %d bytes.\n", FileSize);
        return -1;
```

```
}
// Carrega o arquivo na memória
if (ReadFile(hFile, pData, FileSize, &BytesRead, NULL) == 0) {
                printf ("Falha na leitura do arquivo.\n");
                return -1;
}
CloseHandle(hFile); // Fecha o arquivo
StartTime = timeGetTime();
NumDescompressedBytes = HuffDescompress (&pDescompressedData, pData, FileSize);
// Determina a duração do processo de compressão
ElapTime = timeGetTime() - StartTime;
free (pData);
// escreva os dados comprimidos no disco
strcat (filename/*argv[1]*/, "HUFF.bmp");
hFile = CreateFile(filename, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
WriteFile(hFile, pDescompressedData, NumDescompressedBytes, &BytesWritten, NULL);
CloseHandle(hFile);
free (pDescompressedData);
printf("Arquivo descomprimido          '%s'\n", filename);
printf("Tempo da descompressão         %d ms\n\n", ElapTime);
int z;
scanf(" %d", &z);
/*********************************************************************/
return 0;
}

// trabalhando com a descompressão
else if ( strcmp(argv[1], "d" ) == 0 )
{
FILE *arqComprimido;
BYTE *pDadosComprimidos;
BYTE *pDadosDescomprimidos;
DWORD CabItem;
```

```
char *bmpCadeia;
int LenOriginalFile;
int LenOfBitsCompressFile;
int NumberOfSymbols;
char CabHuff[4];
FILE *arqBMP;
long int TamArquivo;
arqComprimido = fopen(argv[2], "rb");
if ( arqComprimido == NULL )
{
        printf ("Não foi possível abrir o arquivo.\n");
        return -1;
}
TamArquivo = ftell(arqComprimido);
fseek(arqComprimido,  0, SEEK_END );
TamArquivo = ftell(arqComprimido) ;
fseek(arqComprimido, 0, SEEK_SET );

// lendo cabeçalho de arq HUFF
fread(CabHuff,sizeof(BYTE),4,arqComprimido);

// lendo a informação de tamanho original
fread(&CabItem, sizeof(DWORD),1, arqComprimido);
LenOriginalFile =  ( int )CabItem;

// lendo o numero de bits do arquivo Comprimido
fread(&CabItem, sizeof(DWORD),1, arqComprimido);
LenOfBitsCompressFile = ( int ) CabItem;

// lendo o numero de bits do arquivo Comprimido
fread(&CabItem, sizeof(DWORD),1, arqComprimido);
NumberOfSymbols = ( int ) CabItem;
gpHuffTable = new HuffCode[NumberOfSymbols];
fread(gpHuffTable, sizeof(HuffCode), NumberOfSymbols,arqComprimido );
```

```
// verificando a quantidade de bytes que deverão ser lidos
TamArquivo -= ftell(arqComprimido);

// Alocando espaço para arquivo comprimido em memoria
pDadosComprimidos = new BYTE[TamArquivo];
ZeroMemory(pDadosComprimidos,TamArquivo*sizeof(BYTE));

// lendo dados do arquivo
fread(pDadosComprimidos, sizeof(BYTE),TamArquivo, arqComprimido);

// alocando espaço para o arquivo original
pDadosDescomprimidos = new BYTE [LenOriginalFile];
ZeroMemory(pDadosDescomprimidos,LenOriginalFile*sizeof(BYTE));

// verifica qual o menor tamanho na tabela
int PalavraMenorTam = gpHuffTable[0].size;
int PalavraMaiorTam = gpHuffTable[ NumberOfSymbols - 1].size;
int PalavraAtualTam =0;
int iPalavra=0;
WORD Code;
BYTE Symbol;
BYTE Um=1;
BYTE Res;
BYTE Palavra;
int pByteAtualDescomprimidos = 0 ;
int Contador = 0;
for ( int i = 0 ; i < LenOfBitsCompressFile; )
{
        PalavraAtualTam =0;
        //Palavra = 0 ;
        Code = 0;
        while ( PalavraAtualTam <  PalavraMenorTam  )
        {
            if ( ( Contador % 8 ) == 0 )
            {
```

```
                    Palavra = pDadosComprimidos[iPalavra++];
                    Contador=0;
                }
                // fazendo uma operação and com uma palavra que forneça o
primeiro bit
              Res = Palavra & 0x80 ;
                Contador++;
                i++;
                Palavra = Palavra << 1;
                PalavraAtualTam++;
                Code = Code << 1;
                Res = Res >> 7;
                Code += Res;
              }

            // Procurando a palavra na tabela
            while ( VerificaCodigo(Code,Symbol,NumberOfSymbols,
PalavraAtualTam )== false )
            {
                if ( ( Contador % 8 ) == 0 )
                {
                    Palavra = pDadosComprimidos[iPalavra++];
                    Contador=0;
                }
                Res = Palavra & 0x80 ;
                Contador++;
                i++;
                Palavra = Palavra << 1;
                PalavraAtualTam++;
                Code = Code << 1;
                Res = Res >> 7;
                Code += Res;
            }
            pDadosDescomprimidos[pByteAtualDescomprimidos++] = Symbol;
Graphics::TBitmap *bmp = new Graphics::TBitmap();
bmp->PixelFormat = pf8bit;
```

```
bmp->Height = 256;
bmp->Width = 256;
BYTE *pLinha;
// inserindo a paleta com as cores em cinza
LogPal SysPal;
SysPal.lpal.palVersion = 0x300;
SysPal.lpal.palNumEntries = 256;

// inserindo dados na paleta de cores
for ( int Cor=256; Cor>0;Cor-- )
{
		SysPal.lpal.palPalEntry[Cor].peRed = Cor;
		SysPal.lpal.palPalEntry[Cor].peBlue = Cor;
		SysPal.lpal.palPalEntry[Cor].peGreen = Cor;
		SysPal.lpal.palPalEntry[Cor].peFlags = 1;
}
bmp->Palette = CreatePalette((const tagLOGPALETTE *)&SysPal);
for ( int l =0 ; l < bmp->Height;l++ )
{
		pLinha =( BYTE * ) bmp->ScanLine[l];
		for ( int c = 0; c < bmp->Width; c++ )
		{
			pLinha[c] = pDadosDescomprimidos[l * bmp->Width +c ];
		}
}
bmp->SaveToFile("rec.bmp");
return 0;
}
}
```

Abaixo, segue o código do processo de Decomposição da imagem utilizando a Transformada *Wavelet* Haar:

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciências da Computação
Departamento de Computação

"Método para Compressão de Imagens Digitais fundamentado em Procedimentos de Huffman e Wavelets"

Mestranda: Luciana Aparecida de Oliveira Betetto
Orientador: Prof. Dr. Paulo Estevão Cruvinel

```
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#include <stdio.h>
//---------------------------------------------------------------------------

#pragma argsused

typedef double* Tipo_Linha;
typedef double* Tipo_Imagem;

#define max(a, b)  ((abs(a) > abs(b)) ? (a) : (b))
#define min(a, b)  ((abs(a) < abs(b)) ? (a) : (b))


void DecompositionStep (Tipo_Linha C, int h)
{
                int i;
                Tipo_Linha CAux;
                CAux = new double[h];
                for (i = 0; i < h; i++)
                        *(CAux+i) = *(C+i);

                 for (i = 0; i < ((int)h/2); i++)
                 {
                        *(CAux+i)               =  (*(C+2*i) + *(C+2*i+1))/2;
                        *(CAux+((int)h / 2) + i) =  (*(C+2*i) - *(C+2*i+1))/2;
                 }; // of for i

                 for (i = 0; i < h; i++)
                        *(C+i) = *(CAux+i);
```

```
}

void FilterDec ( Tipo_Linha C,
                                        int h,
                                        double x1,
                                        double x2,
                                        double THRMin,
                                        double THRMax,
                                        double Min_Res,
                                        double Filt_pc)
{
                int n;
                double xx1, xx2 , hh, Limite;

                 xx1 = x1 / 2;
                 xx2 = x2 / 2;
                 hh = h / 2;
                 Limite = Min_Res;
                 if (h >= Limite)
                 for (n = 0; n < h; n++)
                 {
                        if (
                                ( n > (hh+xx1)) &&
                                ( n < (hh+xx2)) &&
                                (*(C+n) <= THRMax) &&
                                (*(C+n) >= THRMin)
                           ) *(C+n) = C[n]*((double)Filt_pc/100);
                 }; // of for
};

void DecompositionFilter (
                                                        Tipo_Linha C,
                                                        int h,
                                                        double x1,
                                                        double x2,
                                                        double THRMin,
                                                        double THRMax)

{
                 double xx1, xx2, TTHRMin , TTHRMax;

                 xx1 = x1;
                 xx2 = x2;
                 TTHRMin = THRMin;
                 TTHRMax = THRMax;
                 double Min_Res = 1;
                 double Filt_pc = 0;
                 while (h > 1)
                 {
                        DecompositionStep(C, h);
```

```
                FilterDec(C,h, xx1, xx2, TTHRMin, TTHRMax, Min_Res,Filt_pc );
                h = (int) h / 2;
                xx1 = xx1 / 2;
                xx2 = xx2 / 2;
                TTHRMin = TTHRMin / 2;
                TTHRMax = TTHRMax / 2;
            }; // of while
};


void CompositionStep ( Tipo_Linha C, int h)
{
            int i;
            Tipo_Linha CAux;
            TMemoryStream *CAM = new TMemoryStream();
            CAM->SetSize(2*h);
            CAux = (Tipo_Linha) CAM->Memory;

            for (i = 0; i < 2*h; i++) *(CAux+i) = *(C+i);

                for (i = 0; i < h; i++)
                {
                    *(CAux+(2*i))       = *(C+i) + *(C+h+i);
                    *(CAux+(2*i+1))     = *(C+i) - *(C+h+i);
                }; // of for i

            for (i = 0; i < 2*h; i++) *(C+i) = *(CAux+i);
                CAM->Free();
};

void Composition (Tipo_Linha C, int NTotal)
{ int  h;
            h = 1;
            while (h <= NTotal)
            {
                CompositionStep(C,h);
                h = h * 2;
            }; // of while
};


void ImageDecomposition (Tipo_Imagem Imagem2D, int Size, int Passos)
{
            int Coluna, Linha, h , Cont;
            Tipo_Linha Imagem1D;
            Imagem1D = new double[Size];
            h = Size;
            Cont = 0;
            while ((h > 1) && (Cont < Passos))
            {
```

```
                        for (Coluna = 0; Coluna < h; Coluna++)
                        {
                                for (Linha = 0; Linha < h; Linha++)
                                {
                                        *(Imagem1D+Linha)                                       =
*(Imagem2D+Size*Linha+Coluna);
                                }; // of for Linha

                                DecompositionStep(Imagem1D , h);
                                for (Linha = 0; Linha < h; Linha++)
                                {
                                  *(Imagem2D+Size*Linha+Coluna)=*(Imagem1D+Linha);
                                }; // of for Linha
                        }; // of for Coluna
                        for (Coluna = 0; Coluna < h; Coluna++)
                        {
                                for (Linha = 0; Linha < h; Linha++)
                                {
                                 *(Imagem1D+Linha) = *(Imagem2D+Size*Coluna+Linha);
                                }; // of for Linha

                                DecompositionStep(Imagem1D , h);

                                for (Linha = 0; Linha < h; Linha++)
                                {
                                  *(Imagem2D+Size*Coluna+Linha)=*(Imagem1D+Linha);
                                }; // of for Linha
                         }; // of for Coluna

                         h = h / 2;
                         Cont = Cont + 1;
                 }; // of while h>1

};

void ImageComposition (Tipo_Imagem Imagem2D, int Size, int Passos)
{
                int Coluna, Linha, h;
                Tipo_Linha  Imagem1D;
                Imagem1D = new double[Size];
                h = Size / (ceil(pow(2,Passos-1)));
                while (h <= Size)
                {
                        for (Coluna = 0; Coluna < h; Coluna++)
                        {
                                for (Linha = 0; Linha < h; Linha++)
                                {
                                 *(Imagem1D+Linha) = *(Imagem2D+Size*Coluna+Linha);
                                }; // of for Linha
```

```
                        CompositionStep(Imagem1D , h / 2);

                        for (Linha = 0; Linha < h; Linha++)
                        {
                         *(Imagem2D+Size*Coluna+Linha)=*(Imagem1D+Linha);
                        }; // of for Linha
                    }; // of for Coluna

                    for (Coluna = 0; Coluna < h; Coluna++)
                    {
                        for (Linha = 0; Linha < h; Linha++)
                        {
                         *(Imagem1D+Linha) = *(Imagem2D+Size*Linha+Coluna);
                        }; // of for Linha
                        CompositionStep(Imagem1D , h / 2);

                        for (Linha = 0; Linha < h; Linha++)
                        {
                          *(Imagem2D+Size*Linha+Coluna)                    =
*(Imagem1D+Linha);
                        }; // of for Linha
                    }; // of for Coluna

                    h = h * 2;
                }; // of while h>1
};

int main(int argc, char* argv[])
{
        BYTE *pData;
        Tipo_Imagem Imagem;
        BYTE *pLinha;
        AnsiString NomeArquivoSaida;
        Graphics::TBitmap *bmp= new Graphics::TBitmap();
        double ValorTh = 10.5;

        if ( argc < 2 )
        {
                printf("Entre com o nome da imagem \n");
                return  -1;
        }
        bmp->LoadFromFile((AnsiString ) argv[1]);
        // lendo apenas os dados
        pData = new BYTE[bmp->Height* bmp->Width];
        Imagem = new double[bmp->Height* bmp->Width];

        for ( int l =0 ; l < bmp->Height;l++ )
        {
                pLinha =( BYTE * ) bmp->ScanLine[l];
                for ( int c = 0; c < bmp->Width; c++ )
```

```
                {
                        pData[l * bmp->Width + c ] = pLinha[c];
                        Imagem[l * bmp->Width + c ] = (double) pData[l * bmp-
>Width + c ];
                }
        }
        int h = bmp->Width;
        int i = 1;
        for ( int l =0 ; l < bmp->Height;l++ )
        {
                DecompositionFilter(&Imagem[l*bmp->Width],h,0,bmp->Width,-
ValorTh, ValorTh );
                i = 1;
                while ( i < h )
                {
                        CompositionStep(&Imagem[l*bmp->Width],i);
                        i = i << 1;
                }
        }
        /*

        // bmp->Width deve ser uma potencia de 2
        ImageDecomposition(Imagem,bmp->Width,5);

        double dMax,dMin;
        dMax = Imagem[0];
        dMin = Imagem[0];
        for ( int l =0 ; l < bmp->Height;l++ )
                for ( int c = 0; c < bmp->Width; c++ )
                {
                        dMax = max(Imagem[l * bmp->Width +c ],dMax);
                        dMin = min(Imagem[l * bmp->Width +c ],dMin);
                }

        */

        // salvando imagem decomposta
        for ( int l =0 ; l < bmp->Height;l++ )
        {
                pLinha =( BYTE * ) bmp->ScanLine[l];
                for ( int c = 0; c < bmp->Width; c++ )
                {
                        pLinha[c] = (BYTE) ceil( Imagem[l * bmp->Width +c ]);
                }
        }
        NomeArquivoSaida =  (AnsiString ) argv[1];
        NomeArquivoSaida          =          NomeArquivoSaida.Insert("wavelet",
NomeArquivoSaida.Pos("."));
        // nome arquivo de saida
        bmp->SaveToFile(NomeArquivoSaida);
```

```
                // salvando imagem decomposta
                double Dif;
                for ( int l =0 ; l < bmp->Height;l++ )
                {
                        pLinha =( BYTE * ) bmp->ScanLine[l];
                        for ( int c = 0; c < bmp->Width; c++ )
                        {
                                Dif = (int) pData[l * bmp->Width + c ] - Imagem[l * bmp-
>Width +c ];
                                if ( abs(Dif) < 0 )
                                        pLinha[c] = (BYTE) 0;
                                else if ( ( abs(Dif) > 0 ) && ( abs(Dif) <= ValorTh/2 ) )
                                        pLinha[c] = (BYTE) 100;
                                else if ( abs(Dif) > ValorTh/2 )
                                        pLinha[c] = (BYTE) 255;
                        }
                }
                // criando uma imagem com as diferenças
                bmp->SaveToFile("dif.bmp" );


                /*
                // recompondo a imagem
                ImageComposition(Imagem,bmp->Width,5);



                // salvando imagem decomposta
                for ( int l =0 ; l < bmp->Height;l++ )
                {
                        pLinha =( BYTE * ) bmp->ScanLine[l];
                        for ( int c = 0; c < bmp->Width; c++ )
                        {
                                pLinha[c] = (BYTE) ceil( Imagem[l * bmp->Width +c ]);
                        }
                }

                bmp->SaveToFile("Wave2dRec.bmp" );

                */

                return 0;
}
//---------------------------------------------------------------------------
```

Abaixo, segue o código do Espectro de Wiener:

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciências da Computação
Departamento de Computação

"Método para Compressão de Imagens Digitais fundamentado em Procedimentos de Huffman e Wavelets"

Mestranda: Luciana Aparecida de Oliveira Betetto
Orientador: Prof. Dr. Paulo Estevão Cruvinel

```
function [ W , R , Var] = Wiener_Khintchine(Data, TitleFigure )

f_barra = mean(mean(Data));
% dimensoes do arranjo
M = size(Data,2);
N = size(Data,1);
x = 1:N;
y = 1:M;
R= zeros(N,M);
R2 = zeros(N,M);
W2 = zeros(N,M);
temp_var = R;
matriz_var = Data(1:N,1:M) - f_barra;
R = fft2(matriz_var);
R = R .* conj(R) ;
R = ifft2(R);
R = fftshift(R);
TitleW =   strcat( TitleFigure , ' ( W ) ' );
TitleR =   strcat( TitleFigure , ' ( R ) ' );
W = fft2(R);
figure;
mesh(log(abs(fftshift(W))+1) ); title( TitleW );
```

```
figure;
mesh(real(R));title( TitleR );

% valor de autocorrelacao do ponto com s , na
function [ valor_R ] = autocorrelacao( matriz_variancias, s , na , N, M )
matriz_temp = zeros(size(matriz_variancias,1), size(matriz_variancias,2) );
Tot = M .* N;
matriz_temp( (1:N), (1:M) ) = matriz_variancias((1:N), (1:M)) .* matriz_variancias(
mod(((0:N-1) + s),N) + 1 ,mod((0:M-1) + na,M) +1);
valor_R = sum(sum(matriz_temp)) / Tot ;
```

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, M. P.; ALBUQUERQUE, M. P. **Processamento de Imagens: Métodos e Análises.** Rio de Janeiro, 1998.

BAXES, G. A. **Digital Image Processing – Principles and Applications.** John Wiley & Sons, Inc., Canadá, 1994.

BONNEAU, G. P.; HAHMANN, S.; NIELSON, G. M. **BlaC-*Wavelets*: A Multiresolution Analysis With Non-Nested Spaces.** IEEE, 1996.

BROWN, C. W.; SHEPHERD, B. J. **Graphics File Formats – Reference and Guide.** Prentice Hall,1954.

BURRUS, C. S.; GOPINATH, R. A.; GUO, H. **Introduction to *Wavelets* and *Wavelet* Transforms.** Prentice Hall, New Jersey, 1998.

CRANE, Randy. **A Simplified Approach to Image Processing.** Prentice Hall, New Jersey, 1997.

DAUBECHIES, I. **Orthonormal Basis of Compactly supported *Wavelets*.** Comm. Appl. Math. 41, pp. 909-996, 1988.

DeROSE, T. D.; LOUNSBERY, M.; WARREN, J. **Multiresolution Analysis for Surfaces of Arbitrary Topological Type.** Technical Report, October 1993.

EGGER, O.; LI, W. **Subband coding of images using asymmetrical filter banks.** IEEE Trans. Image Processing, 1995.

EGGER, O., LI, W.; KUNT, M. **High compression image coding an adaptive morphological subband decomposition.** Proc. IEEE, 1995.

EGGER, O.; FLEURY, P.; EBRAHIMI, T.; KUNT, M. **High-Performance Compression of Visual Information – A Tutorial Review – Part I: Still Pictures.** IEEE, June 1999.

FOURIER, J. B. J. **La Théorie Analitique de la Chaleur.** 1822.

GABOR, D. **Theory of communication.** Journal of the Institute for Electrical Engineers, 1946.

GIROD, B.; GRAY, R.; KOVACEVIC, J.; VETTERLI, M. **The Past, Present, and Future of Image and Multidimensional Signal Processing – Image and Video Coding.** IEEE Signal Processing Magazine, pg. 21-58, March 1998.

GOMES, J.; VELHO, L. **Computação Gráfica: Imagem.** IMPA/SBM, Rio de Janeiro, 1994.

GONZALES, R. C.; WOODS, R. E. **Digital Image Processing.** Prentice Hall, New Jersey, 2002.

GOUTSIAS, John; HEIJMANS, Henk J. A. M. **Nonlinear Multiresolution Signal Decomposition Schemes – Part I: Morphological Pyramids.** IEEE Transactions on Image Processing, Vol. 9, nº 11, November 2000.

HAAR, A. **Zur Theorie der Orthogonalen Funktionen-Systeme**, Math. Ann, 69, pp.331-371, 1910.

KUNT, M.; BÉNARD, M.; LEONARDI, R. **Recent Results in High-Compression Image Coding.** IEEE Transactions on Circuits and Systems, Vol. 34, nº 11, pg. 1306-1336, November 1987.

LIMA, Paulo Cupertino. ***Wavelets*: uma introdução.** Departamento de Matemática - UFMG, 2003.

LIMA, Paulo Cupertino. ***Wavelets*: Teoria, Algoritmos e Aplicações.** Departamento de Matemática, UFMG, 2004.

MALLAT, S. A. **Compact Multiresolution Representation: The *Wavelet* Model.** Proc. IEEE Computer Society Workshop on Computer Vision, IEEE Computer Society Press, Washington, D.C., pp.2-7, 1987.

MANDUCA, A. **Compressing Images with *Wavelet*/Subband Coding.** IEEE Engineering in Medicine and Biology, p. 639-646, 1995.

MARAR, J. F.; FILHO, E. C. B. C.; VASCONCELOS, G. C. ***Wavelets* Polinomiais: Uma Família de Funções Splines para Aplicações em Processamento de Sinais e Imagens.** Anais do IX SIBGRAPI, 1996.

MOFFAT, A. **Lossless Compression.** The Computer Journal, Vol. 40, 1997.

OLIVEIRA, L. F.; OLIVEIRA, A. A. F.; CAVALCANTI, P. R.; ESPERANÇA, C. **Compressão de Imagens usando Transformadas *Wavelet* e Curva de Peano-Hilbert.** Anais do XI Sibgrapi, 1998.

PINHO, A. J. **A Comparison of Methods for Improving the Lossless Compression of Images with Sparse Histograms.** IEEE International Conference on Image Processing, Vol. 2, pg. 673-676, 2002.

PITAS, Ioannis. **Digital Image Processing Algorithms.** Prentice Hall, 1995.

POLIKAR, R. **The *Wavelet* Tutorial.** Second Edition, 1994.

PREUSS, R. D. **Very fast computation of the Radix-2 Discrete Fourier Transform.** IEEE Acoustic, Speech and Signal Processing, 30, p. 515-607, 1982.

RAMASWAMY, V. N.; NAMUDURI, K. R.; RANGANATHAN, N. **Lossless Image Compression Using *Wavelet* Decomposition.** IEEE Proceedings of ICPR, 1996.

SAHNI, S.; VEMURI, B. C.; CHEN, F.; KAPOOR, C.; LEONARD, C.; FITZSIMMONS, J. **State of the Art Lossless Image Compression Algorithms.** IEEE Transactions on Image Processing, October 1997.

SAID, A.; PEARLMAN, W. A. **An Image Multiresolution Representation for Lossless and Lossy Compression.** IEEE Transactions on Image Processing, Vol. 5, September 1996.

SEALES, W. B.; YUAN, C. J.; BROWN, M. **Efficient Content Extraction In Compressed Images.** IEEE Workshop on Content Based Access of Image and Video Libraries, pg. 52-58, June 1997.

SHANNON, C. E. **A Mathematical Theory of Communication.** The Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656, July, October 1948.

SHAPIRO, J. M. **Embedded image coding using zerotrees of Wavelet coefficients.** IEEE Trans. Signal Processing, vol.41, 1993.

SHENG, F.; BILGIN, A.; SEMENTILLI, P. J.; MARCELLIN, M. W. **Lossy and Lossless Image Compression Using Reversible Integer *Wavelet* Transforms.** Proceedings, International Conference on Image Processing, Vol.3, pg. 876-880, Chicago, Illinois, October 1998.

SIMONCELLI, E. P., ADELSON, E. H. **Subband Transforms.** Norwell, MA: Kluwer, 1990.

SMITH, B. C.; ROWE, L. A. **Algorithms for Manipulating Compressed Images.** IEEE Computer Graphics & Applications, Vol. 13, pg. 34-42, September 1993.

SOUZA, I. A. **Interpolação 3-D de Imagens usando Teoria de Estimação.** São Carlos, UFSCar, Tese de Mestrado, 1999.

SUPENG, L.; WENZHI, D.; YUWEN, X.; SHIXIAN, P. **A Scheme of High-Compression Image Coding.** Signal Processing, Vol. 2, pg. 1039-1042, October 1996.

VETTERLI, Martin; KOVACEVIC, Jelena. ***Wavelets* and Subband Coding.** Prentice Hall Signal Processing Series, 1995.

WALNUT, David F. **An Introdution to *wavelet* analysis.** Applied and Numerical Harmonic Analysis, Birkhäuser, Boston – Basel – Berlin, 2002.

WANG, James Z. ***Wavelets* and Imaging Informatics: A Review of the Literature.** Journal of Biomedical Informatics, p. 129-141, 2001.

YOUNG, R. K. ***Wavelet* Theory and Its Applications.** Kluwer Academic Publishers, Massachusetts, 1994.