CENTRO UNIVERSITÁRIO FEEVALE

GABRIEL PLEGGE DA SILVA

SISTEMA CONVERSOR DE IMAGENS MÉDICAS PARA O AMPLIA-I

Novo Hamburgo, novembro de 2008.

GABRIEL PLEGGE DA SILVA

SISTEMA CONVERSOR DE IMAGENS MÉDICAS PARA O AMPLIA-I

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor Orientador: Marta Rosecler Bez

Novo Hamburgo, novembro de 2008.

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial aos familiares e aos amigos próximos que colaboraram para o apoio e equilíbrio emocional necessários para o desenvolvimento deste trabalho.

RESUMO

Este projeto apresenta a proposta de desenvolvimento de um módulo de conversão de imagens, com o objetivo de transformar imagens de formatos distintos a ser inserido em um projeto maior chamado AMPLIA-I, que é a segunda fase do projeto AMPLIA. O projeto AMPLIA é um ambiente utilizado como ambiente virtual de aprendizagem na faculdade de medicina. Tem como objetivo aumentar os conhecimentos dos alunos baseados em casos reais, que encontram-se arquivados dentro do ambiente. A ferramenta é utilizada para criar simulações de casos de enfermidades onde o aluno deve interpretar o caso e solucionar a enfermidade. Como o sistema é carente de interpretação visual, por conter apenas casos textuais, está sendo desenvolvido o projeto AMPLIA-I, um módulo de exames por imagens que será acoplado ao AMPLIA. Desta forma, os alunos podem aprender também com exames e imagens médicas reais utilizadas no exercício da função de médico. O projeto AMPLIA-I necessitará de um módulo acoplado para que o usuário, médico, professor ou aluno, não precise se preocupar com a conversão dos formatos, e se preocupe apenas com a utilização da ferramenta.

Palavras-chave: Conversão de Imagem. Formatos de Imagem. AMPLIA-I. Bitmap. Processamento Digital de Imagens.

ABSTRACT

This project proposes the development of a module of image conversion, aiming the transformation of image files to other file formats. This project is inserted in a bigger project called AMPLIA-I, second level of the AMPLIA project. The AMPLIA project is an environment used as a virtual environment for learning medicine. Aims to increase knowledge among students based on real cases, which are filed within the environment. The tool is used to create simulated cases of diseases where the students must interpret and resolve the disease. As the system is poor for visual interpretation, because it contains only textual cases, the AMPLIA-I project is being developed, as a module of images that will be attached to AMPLIA project. In this way, students can also learn with examinations and medical images used in the actual exercise of medicine. The AMPLIA-I project will require a module attached to the user, doctor, teacher or student, does not have to worry about the conversion of formats, and worry only with the use of the tool.

Palavras-chave: Image Conversion. Image Formats. AMPLIA-I. Bitmap. Digital Image Processing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BMP	Bitmap
CS3	Creative Suite 3
DCT	Discret Cosine Transform
DICOM	Digital Imaging and Communicactions in Medicine
GIF	Graphics Interchange Format
GIMP	GNU Image Manipulation Program
GNU	GNU is Not Unix
JPEG	Joint Photographic Experts Group
LOCO-I	Low Complexity Low Compression for Images
LZW	Lempel-Ziv & Welch
RGB	Red, Green & Blue
RLE	Run-Length Encoding
PNG	Portable Network Graphics

SUMÁRIO

INTRODUÇÃO

Um Conversor de Imagens é um programa que deve transformar imagens em formatos diferentes dos originais. Este trabalho parte do princípio de que é possível criar um módulo de processamento de imagens digitais, com ênfase na conversão de imagens entre vários formatos existentes. No mercado existem softwares disponíveis que oferecem esta facilidade, porém, normalmente com conversão entre poucos formatos e, os que possuem vários formatos, têm preço elevado para que possam ser adquiridos por instituições públicas ou mais modestas, além de terem expansibilidade restrita devido ao seu código-fonte ser proprietário. (Bassani, 2003)

Imagens que são capturadas pelo olho humano são de natureza continua. (Bassani, 2003). Segundo Scuri (2002), para que estas imagens possam ser armazenadas e processadas, elas devem passar por uma digitalização. Este processo é chamado de discretização, e consiste em tornar a continuidade das cores de uma imagem em pontos de cores, tornando a imagem digitalizada e não mais contínua. Cada ponto é um conjunto de bytes que corresponde a uma cor. Este conjunto forma uma matriz de bytes que contém a imagem digitalizada.

Segundo Conci, Azevedo e Leta (2008), formatos de imagens são chamados de tipos de imagens como engano, pois formatos de imagem servem apenas para armazenar os dados das imagens enquanto os tipos de imagens referem-se ao conteúdo das imagens.

Este trabalho tem como objetivo estudar formatos de imagens a fim de construir um módulo de conversão de imagens para um projeto maior chamado AMPLIA-I. Os estudos serão feitos sobre os formatos de arquivos de imagem mais difundidos, além de estudos do padrão de arquivos DICOM (*Digital Imaging and Communicactions in Medicine*).

O desenvolvimento do módulo conversor para o AMPLIA-I contribuirá para áreas como medicina e ensino entre outras. O sistema conversor deve funcionar de forma transparente, sem que os usuários necessitem de conhecimentos na área da computação ou de conversão de arquivos.

Este trabalho está dividido em três capítulos. No primeiro é abordado todo o referencial teórico a ser utilizado no decorrer do trabalho. No capítulo dois, é apresentado o estudo realizado sobre ferramentas e trabalhos correlatos. O terceiro capítulo apresenta a proposta de trabalho a ser desenvolvida, seguida da conclusão.

1 IMAGENS DIGITAIS

Este capítulo tem como objetivo apresentar o estudo feito sobre formatos de imagens digitais, dando ênfase às propriedades de cores e de algoritmos de compressão. Além dos formatos de imagens digitais convencionais, este capítulo também aborda como assunto o padrão de formato DICOM e suas propriedades. A revisão bibliográfica visa também o conhecimento das fórmulas aplicadas pelos formatos digitais, a fim de serem utilizadas no sistema conversor de imagens.

O capítulo aborda primeiro os temas de formatos de arquivos digitais: Bitmap, JPEG, GIF, PNG, TIFF e DICOM. Na segunda etapa deste capítulo, são abordados os algoritmos de compressão utilizados nos formatos apresentados.

1.1 Formatos de arquivos

Especificações técnicas referentes a estruturas de arquivos são estudadas e apresentadas neste tópico como referencial teórico para conhecimento e aplicação na segunda etapa deste trabalho.

1.1.1 Mapeamento de bits (Bitmap)

O formato de arquivos Bitmap ou mapeamento de bits é o formato de imagens com o algoritmo mais simples, pois utiliza compressão de dados sem perda de qualidade ou não utiliza compressão. A qualidade da imagem é a melhor possível, por não haver perdas de bits no algoritmo de armazenamento dos dados. Como não contém algoritmos de compressão com perda de dados, os arquivos que utilizam este formato podem ficar maiores que os arquivos que utilizam outros formatos com compressão. Os bits de cor utilizados pelo formato Bitmap são: 4 bits para arquivos de imagem de 16 cores, 8 bits para arquivos de 256 cores, 16 bits

para 65.536 cores e 24 bits para imagens com 16 milhões de cores. O tamanho atingido pelo arquivo de formato Bitmap pode ser obtido através do cálculo do número de pixels horizontais multiplicado pelo número de pixels verticais da imagem, multiplicado pela quantidade de bits de cores utilizados e depois dividido pelo número 8, para que a fórmula tenha seu resultado em bytes. (Conci, Azevedo e Leta, 2008)

A estrutura dos arquivos bitmap se divide em quatro blocos. São eles, o *File Header*, *Image Header*, *Color Table* e *Pixel Data*. (Miano, 1999)

O *File Header*, ou cabeçalho de arquivo, é dividido em cinco seções. A primeira delas contém dois bytes com os caracteres ASCII “BM” para a confirmação do formato de arquivo ser bitmap. A segunda é um conjunto de quatro bytes que contém o tamanho do arquivo. Em seguida, as seções três e quatro, são espaços reservados, de dois bytes cada, que não são usados. Na última seção, carrega um conjunto de quatro bytes definindo o *offset* de início dos dados dos *pixels*. (Miano, 1999)

O *Image Header* é a seção que vem logo após o *File Header*. É composto por duas subseções: *Bitmap Info Header* e *Bitmap Core Header*. A seguir a tabela 1.1 e tabela 1.2 apresentam as características das duas subseções:

Tabela 1.1 – Propriedades do *Bitmap Info Header*

Propriedade	Tamanho (*bytes*)	Descrição
Tamanho	4	Tamanho do cabeçalho (no mínimo 40)
Largura	4	Largura da imagem
Altura	4	Altura da imagem
Planos	2	Deve conter o valor 1
Bit Count	2	Número de bits por pixel (Ex: 1,4,8,16,24 ou 32)
Compressão	4	Tipo de compressão
Tamanho da imagem	4	0 caso não esteja usando compressão
XPelsPerMeter	4	Melhor resolução em pixels por metro
YPelsPerMeter	4	Melhor resolução em pixels por metro
Cores Utilizadas	4	Número de cores utilizadas no mapa de cores
Cores Importantes	4	Número de cores significantes

Fonte: *Compressed Image File Formats*. John Miano, 1999.

Os bytes utilizados na propriedade de compressão do cabeçalho são: 0 para o tipo RGB, 1 para RLE8, 2 para RLE4 e 3 para *BitFields*.

Tabela 1.2 – Propriedades do *Bitmap Core Header*

Propriedade	Tamanho (*bytes*)	Descrição
Tamanho	4	Tamanho do Header (deve ser 12)
Largura	2	Largura da imagem
Altura	2	Altura da imagem
Planos	2	Deve ser 1
Bit Count	2	Bits por cor (1,4,8 ou 24)

Fonte: *Compressed Image File Formats*. John Miano, 1999.

Após os *Headers*, o arquivo bitmap apresenta a *Color Pallete*, quadro de cores utilizadas pela imagem. Este quadro pode ser apresentado em três formatos. O primeiro, *RGB Triple*, que contém uma estrutura de três campos, um para cada uma das cores: vermelho, verde e azul. Cada campo com um *byte* de tamanho recebe o respectivo valor de cor. O segundo formato é o *RGB Quad*, com os mesmos três atributos do *RGB Triple*, porém com um atributo adicionado chamado de "Reservado", que sempre recebe o valor zero. O terceiro formato pode ser encontrado em imagens que contém o *bitCount* com valores 16 ou 32 e o bit de compressão referenciando o formato *BitFields* (de valor 3). Este formato de estrutura não é igual aos outros, pois não é uma estrutura de tabela de *bytes*. Apresenta-se como uma máscara de *bits* na forma de um vetor (*array*) com três posições, uma para cada cor do RGB. Cada uma destas três posições é composta por um número que servirá de máscara para a sua respectiva cor. O primeiro sendo máscara para a cor vermelha, o segundo para a cor verde e o terceiro para a cor azul. Por exemplo, em uma imagem com 32 bits, o vetor conterá a seguinte informação:

Tabela 1.3 – Vetor de cores RGB com 32 bits

Posição no vetor	Cor correspondente
00000000000000000000001111111111	Vermelho
00000000000011111111110000000000	Verde
00111111111100000000000000000000	Azul

Fonte: *Compressed Image File Formats*. John Miano, 1999.

Arquivos do formato bitmap, que utilizam 16, 24 e 32 bits sem o campo "compressão" referenciando o padrão *BitFields*, não utilizam tabelas de paleta de cores.

O *Pixel Data* é a área onde estão armazenadas as informações referentes ao conteúdo visual do arquivo de imagem. Sua posição é determinada pelo atributo *OffBits* encontrado no *BitmapFileHeader*. As informações da imagem encontram-se dispostas em linhas, e são ordenadas de baixo para cima. O número de linhas da imagem é recebido no atributo altura,

provindo do *BitmapInfoHeader* ou do *BitmapCoreHeader*. A largura da linha é definida pelos atributos *BitCount* e largura, também provindos do *BitmapInfoHeader* ou do *BitmapCoreHeader*. O número de bytes é arredondado para cima para um número múltiplo de quatro. Segue a fórmula:

Bytes por linha = (((largura X *bit count* + 7) / 8) + 3) / 4

A apresentação dos dados do arquivo da imagem é feita de acordo com o número de *bits* por *pixels*. Se a quantidade de bits por *pixel* for entre 1 e 4, cada *byte* é dividido em dois ou oito atributos, que farão referência a uma cor que estará na paleta de cores. Se forem 8 *bits* por *pixel*, cada *byte* inteiro representará um índice da paleta de cores. Se forem imagens com 16 *bits* de cores por *pixel*, as imagens utilizarão dois *bytes* para fazer referência a uma cor da paleta. Se a compressão utilizada for *RGB*, a intensidade de cada cor terá sua representação feita por cinco *bits*, e o último *bit* não será utilizado. Para imagens de 24 *bits*, cada *pixel* será representado por 3 *bytes* e, para 32 *bits*, serão utilizados 4 *bytes*.

1.1.2 Joint Photographic Experts Group (JPEG)

O padrão Jpeg, sigla de *Joint Photographic Experts Group*, é normalmente utilizado em imagens fotográficas, é hoje o padrão de arquivos de imagem mais popular. Este padrão tem como principal característica o uso de uma compressão de dados com perda na qualidade da imagem. Esta perda é resultado da remoção de pontos da imagem original, diminuindo assim, o tamanho dos dados a serem comprimidos. Esta compressão com perda de qualidade permite ao arquivo ter tamanhos menores que os padrões Tiff e Bitmap. A taxa de compressão dos arquivos neste formato varia de 0 a 12. O número da taxa de compressão é inversamente proporcional ao nível de compressão da imagem. Quanto maior a taxa, menor a compressão, maior a qualidade da imagem e maior o tamanho do arquivo. Os arquivos formatados no padrão Jpeg têm por padrão as extensões JPEG, JFIF, JPE e JPG, sendo a última, a extensão mais comum de arquivos do formato. (Conci, Azevedo e Leta, 2008)

Segundo Agostini (2002), o padrão JPEG não contém uma estrutura de arquivo pré-definida como o padrão Bitmap, porém o padrão chamado JFIF (*JPEG File Interchange Format*) faz esta definição. O padrão JFIF é um padrão de imagem que utiliza algoritmo

JPEG, e foi criado por Eric Hamilton. As imagens que hoje contém a extensão JPG são arquivos que utilizam o padrão JFIF. Sendo assim, o formato JFIF acabou tornando-se um sinônimo de JPEG.

Os dados das imagens que utilizam o formato JPEG são divididos em grupos chamados de *scans*. Cada *scan* contém no seu cabeçalho além de uma marcação de início, alguns parâmetros em comum para qualquer arquivo que utilize codificação JPEG. Estão contidos nestas informações os dados sobre a largura do *scan*, o número de componentes de cor que estão contidos neste *scan* e o seletor da tabela *Huffman*. Estes dados são básicos do cabeçalho de *scans* do padrão JPEG. (In et all, 1999)

Segundo Agostini (1999), para o padrão JPEG podem ser utilizados quatro modos de compressão: seqüencial, progressivo, sem perdas e hierárquico. Nos dois primeiros, podem ser utilizados os algoritmos *Huffman* e Aritmético. O modo de compressão sem perda comporta o algoritmo JPEG-LS e Original. O modo de compressão hierárquico é pouco utilizado, pois é muito mais complexo em relação aos demais.

No modo seqüencial, as imagens têm seus dados codificados ordenados do topo para a base e pode ter seus dados de amostragem em 8 ou 12 bits. Cada componente de cor é comprimido em um único *scan*. *Huffman* e Aritmético são os dois tipos de codificação de entropia utilizáveis neste modo. O *baseline* é um subgrupo pertencente ao modo seqüencial e que aceita apenas dados de amostragem com 8 *bits*. (Agostini, 1999)

In et al. (1999), cita que modo progressivo também utiliza *Huffman* e o tipo Aritmético de codificação de entropia, aceitando também 8 e 12 *bits* de amostragem. A diferença visual do modo progressivo para o modo seqüencial pode ser observada na visualização de forma parcial da imagem. Este modo de operação faz com que as imagens possam ter uma pré-visualização do seu conteúdo de forma bruta logo após a decodificação do seu primeiro *scan*, tendo a visualização da imagem sendo melhorada gradualmente após a decodificação de cada *scan* posterior. Esta visualização não pode ser feita no modo de operação Seqüencial, pois a cada novo *scan* decodificado, são adicionados blocos de pixels à

imagem, sempre do topo para a base. Um exemplo prático e vantajoso do uso de imagens montadas através do modo progressivo é a utilização em ambientes de rede ou internet.

Segundo Wallace (1991), o modo hierárquico utiliza uma técnica de codificação "piramidal" da imagem em múltiplas resoluções. Cada uma destas resoluções se diferencia das adjacentes por um fator de dois, nas direções vertical, horizontal ou em ambas as direções. Agostini (1999) afirma que o modo hierárquico é muito complexo, e acaba por ser pouco utilizado. Também afirma que este modo, se for avaliado em baixa transmissão, tem um melhor desempenho quando comparado ao modo progressivo. A quantidade de dados contidos em uma imagem que utiliza este modo também é superior a quantidade de dados do modo progressivo. Os *scans* são agrupados em *frames*, que são divisões da imagem em grupos pedaços da imagem do modo hierárquico. Alguns *frames* podem conter apenas um *scan*.

De acordo com Agostini (1999), o modo de compressão sem perdas é o modo menos utilizado por possuir um algoritmo com poder de compressão inferior aos demais modos. Segundo Miano (1999), o formato original de compressão sem perdas sempre preserva a imagem original, fazendo com que a compressão não possa ser tão boa quanto a de um algoritmo que comprima com perda. Com a criação do modo de compressão chamado JPEG-LS, outro padrão de compressão sem perdas, a utilização do padrão sem perdas original tornou-se obsoleta.

O algoritmo JPEG-LS tem seu núcleo baseado em um algoritmo chamado LOCO-I (*Low Complexity Low Compression for Images*). Traz além da sua versão sem perdas, uma versão do algoritmo com perda de qualidade, para uma maior compressão. Esta versão é chamada de "quase sem perda" ou *near-lossless*. (Weinberg, 2000).

O núcleo da compressão JPEG pode ser apontado em três operações do algoritmo: a transformada discreta do coseno em duas dimensões (DCT 2-D), a quantização e a codificação de entropia. Estas operações são encontradas em imagens coloridas e em tons de cinza, porém, em imagens coloridas devem ser adicionados dois outros procedimentos antes

da execução da DCT 2-D. São os processos adicionais de imagens coloridas a conversão de espaços de cores e o *downsampling*.

A conversão de espaços de cores, aplicada em imagens coloridas, tem a função de converter o espaço de cores RGB (*Red, Green and Blue*) para o espaço de cores utilizado no padrão JPEG. Este modelo é denominado luminância e crominância. Esta conversão é feita pelo fato do elevado grau de correlação entre os componentes R, G e B, elevando o grau de dificuldade do processamento de cada uma das cores que compõe o padrão RGB. Um exemplo de modelo de cores de luminância e crominância é o YCbCr. Como imagens em tons de cinza possuem apenas um componente, acabam não necessitando fazer esta conversão.

Conforme Agostini (2001), YCbCr é um modelo de cores do tipo luminância e crominância, onde o Y refere-se ao fator de luminância e o Cb e Cr referem-se a crominância. A cor representada no componente Y é a tonalidade em cinza da imagem, enquanto que os outros dois componentes representam as cores da imagem. No componente Cb são representados dados da cor azul e no componente Cr são representados dados da cor vermelha. A base deste espaço de cores é o padrão YUV, que é utilizado no PAL, sistema de televisão europeu. A conversão de RGB para YCbCr é feita pixel a pixel e é dada pela formula a seguir:

$$Y_{i,j} = 0{,}299R_{i,j} + 0{,}587G_{i,j} + 0{,}114B_{i,j}$$
$$Cb_{i,j} = -0{,}169R_{i,j} - 0{,}331G_{i,j} - 0{,}5B_{i,j}$$
$$Cr_{i,j} = 0{,}5R_{i,j} - 0{,}419G_{i,j} - 0{,}081B_{i,j}$$

Figura 1.1 – Fórmula de conversão de *pixel* RGB para *pixel* YCbCr
Fonte: Agostini, 2001.

Segundo Miano (1999), o modelo de cor YCbCr, a luminância representada pelo componente Y é a representação da *grayscale* (ou escala em tons de cinza) da imagem, o que faz com que este modelo seja compatível com modelos em preto e branco.

O processo de *downsampling* é um procedimento não obrigatório e aplicável apenas em imagens coloridas. O processo de compressão com perdas começa nesta etapa do padrão JPEG. Imagens que passam pelo procedimento de *downsampling* sofrem alterações e jamais

são iguais as imagens originais. Estas perdas podem ser controladas e, com isso, podem se tornar imperceptíveis ao olho humano. O *downsampling* trabalha diretamente nos pontos de cores da imagem (Cb e Cr), onde a percepção do olho humano é menor em relação à luminância (Y), eliminando parte das informações dos componentes de cores da imagem. Um exemplo utilizado por Agostini e Bampi (2001) para o processo de *downsampling* é utilizar a relação de 4:1:1 ao invés de 1:1:1 para os componentes de luminância e crominância. Utilizando esta relação, têm-se 4 componentes de luminância (Y) para cada 1 componente de crominância de cada cor (Cb e Cr). Sendo assim, para cada 4 informações Y, tem-se 1 Cb e 1 Cr. Descrevendo esta alteração em dados, na relação 1:1:1 (antes do *downsampling*) utilizando uma fração de imagem de 4 pixels, contendo 8 bits para cada componente de cor (1x8:1x8:1x8), como resultado são obtidos 24 bits, que multiplicados pelo número de pixels da fração da imagem, resultam em 96 bits. Ao aplicar o *downsampling* será utilizada a proporção de 4:1:1, tem-se um total de 4 x 8 bits para as 4 informações Y somadas com 1 x 8 bits de Cb e 1 x 8 bits de Cr, totalizando em 48 bits os 4 pixels da imagem após o *downsampling*. Esta compressão, nesta proporção, gera uma redução de 50% no tamanho da imagem.

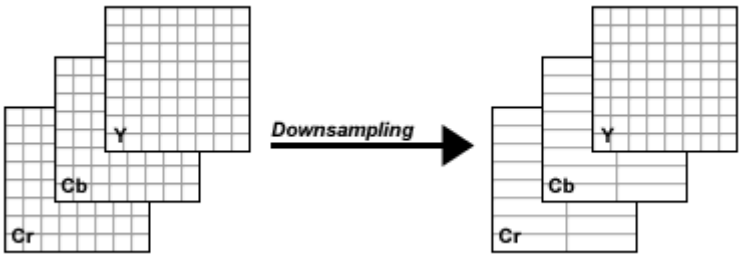

Figura 1.2 – Matrizes de YCbCr antes e depois do *Downsampling*
Fonte: Agostini, 2001.

É chamado de Transformação Discreta do Coseno em duas dimensões, o processo de conversão de dados de um domínio espacial para um domínio de freqüência. O objetivo desta transformação é preparar os dados para o próximo passo da compressão, a quantização. Para que ocorra este procedimento, a imagem deve ser dividida em matrizes de duas dimensões, contendo 8 x 8 *pixels*. Para a diminuição da complexidade dos cálculos da DCT 2-D, são efetuados cálculos de DCT 1-D para cada uma das duas dimensões da matriz. Existem vários algoritmos utilizados para efetuar o cálculo em uma dimensão, e conforme Agostini (2002), o mais rápido é chamado de Arai.

Segundo Wallace (1991), a quantização tem o objetivo de descartar informações que não têm valor visual significativo. Agostini (2002) diz que após executados os passos da DCT 2-D, as informações provindas de freqüências mais elevadas são atenuadas ou removidas pela quantização. O objetivo deste passo é aplicar uma divisão inteira dos coeficientes da DCT 2-D por uma constante, chamada constante de quantização, arredondando o resultado sempre para um número inteiro menor. Esta divisão tem a intenção de aumentar o número de freqüências reduzidas a zero, assim, removendo as freqüências que não interferem na imagem. As constantes de quantização são armazenadas em matrizes de 8x8. Estas matrizes são chamadas de tabelas de quantização. Para imagens coloridas, são utilizadas duas tabelas de quantização, uma para a luminância (Y), e outra para a crominância (Cb e Cr). Estas tabelas são adicionadas ao arquivo JPEG comprimido, para que as informações possam ser remontadas na descompressão.

A Codificação de entropia é o passo responsável pela aplicação da compressão, utilizando várias técnicas de compressão em conjunto. Dois exemplos de técnicas utilizadas são *Huffman* e *Run-Length Encoding* (RLE).

1.1.3 Graphics Interchange Format (GIF)

Segundo Miano (1999), o formato GIF (*Graphics Interchange Format*) também é um padrão bastante difundido e utilizado. O GIF possui muitas características que fazem com que ele seja diferente dos outros formatos. Por exemplo, o formato armazena apenas imagens com até 256 cores em RGB ou imagens em tons de cinza. Outro exemplo de diferencial do GIF é a utilização de um mapa de cores, que é um índice onde se encontram todas as cores presentes na imagem. O formato permite que se escolha uma cor específica da imagem a ser utilizada como transparente, e também permite que se escolha uma cor para ser o plano de fundo.

A publicação do padrão GIF foi feita em 1987, pela CompuServe, apresentando as características da sua primeira versão, chamada de GIF87a. Em seguida, foi lançada a sua segunda versão chamada GIF89a, que além de ser compatível com outra versão, trazia muitas melhoras ao padrão do formato, como a possibilidade de adicionar mais de uma imagem no arquivo, possibilitando a criação de animações mais simples. Outra adição de melhoria foi a

possibilidade de adicionar uma cor definida como plano de fundo da imagem, para que não fossem ocupados tantos pixels com a mesma cor, e assim, economizando no tamanho do arquivo da imagem. (CompuServe, 1987)

De acordo com Miano (1999), o formato GIF também possui na sua estrutura de arquivo um bloco chamado de *Global Color Table* ou tabela global de cores. Esta tabela contém um *array* com todos os índices de cores utilizados de forma genérica. Por possibilitar a existência de múltiplas imagens em um único arquivo, o GIF tem também a possibilidade de configurar uma *Local Color Table* (tabela de cores local) para cada imagem. A existência ou não desta tabela é definida por uma *flag* no cabeçalho de cada bloco de imagem do arquivo. Caso a imagem não possua uma *Local Color Table*, automaticamente, ela utiliza a *Global Color Table*.

A estrutura do arquivo contém um bloco chamado *Graphic Control Extension*, que guarda algumas infomações referentes à configuração do arquivo. Entre elas está a *flag* que indica se existe ou não uma cor que deve ficar transparente, juntamente com o índice da cor que deve ser transparente.

Arquivos no formato GIF têm por padrão serem armazenados do topo para a base, da esquerda para a direita. O cabeçalho da imagem possui uma *flag* chamada de *Interlace* na qual habilita o modo entrelaçado de armazenamento da imagem. Este modo faz com que a imagem seja guardada no bloco de dados da imagem com um algoritmo de mesma função do *Progressive JPEG*, fazendo com que a pré-visualização do seu conteúdo já traga a imagem em uma qualidade baixa, e conforme os dados são recebidos e interpretados, pixels são adicionados e a qualidade da imagem aumenta.

O algoritmo de compressão do formato de arquivos GIF é o LZW (*Lempel-Ziv* & *Welsh*), método conhecido como *Dictionary-Based Compression Scheme*, ou sistema de compressão baseado em dicionário, que é abordado no item 1.2.2.

1.1.4 Portable Network Graphics (PNG)

Segundo Boutell et. al. (1997), o *Portable Network Graphics* é um padrão de arquivo relativamente novo. Foi desenvolvida em um curto período de tempo de um ano e meio, tendo sua publicação liberada em dezembro de 1996. O padrão foi desenvolvido para um melhor desempenho em aplicações online. Suporta padrões de *RGB Triple* (utilizado no padrão *Bitmap*), *Indexed-Color* ou *Color Pallete* (como o GIF), *Grayscale*, *Truecolor* e suporta também canal *alpha*. O PNG é um padrão de arquivos que praticamente foi desenvolvido para substituir o GIF, por ser livre de patentes. Os espaços de cores *Grayscale* e RGB são os espaços suportados neste formato.

O canal *alpha* é a representação da transparência. É suportado no modo *Grayscale* e no modo RGB. Este recurso pode ser aplicado em imagens que possuem 8 e 16 bits por canal de cor. Este canal de transparência utiliza a mesma faixa de bits utilizada para as cores (ou para a luminosidade, caso o arquivo seja comprimido no modo *Grayscale*).

Como o *Progressive JPEG* e o *Enterlaced GIF*, os dados das imagens PNG também podem ser gravados de forma a serem apresentados com exibição progressiva.

No padrão PNG, os arquivos são divididos em blocos seqüenciais chamados *chunks*. Os *chunks* padrões dos arquivos PNG são encontrados na definição do padrão PNG. O grupo que desenvolve este padrão de arquivo, *PNG Development Group*, também mantém uma lista de tipos de *chunks* públicos definidos. Existem também os *chunks* que são desenvolvidos de maneira privada e definidos por aplicações. Caso seja criado um *chunk* que possa ser de interesse ou uso geral, este pode ser submetido ao grupo que desenvolve, para que eles avaliem e publiquem na sua lista. Estes blocos são desenvolvidos e nomeados para que sejam feitos algoritmos de leitura para os mesmos em decodificadores de arquivos PNG. Assim, pelos nomes, os decodificadores podem pular ou ignorar blocos de tipos desconhecidos ou que são julgados como sem importância.

Para cada *chunk* são dados nomes únicos compostos por quatro letras provindas da tabela ASCII. São utilizados padrões de letras maiúsculas e minúsculas nos nomes para definir alguns aspectos. A primeira letra quando é maiúscula define que o bloco é crítico. A segunda letra define que o bloco é público. O terceiro caractere do nome é reservado, e deve

ser encontrado sempre em letra maiúscula ou, do contrário, deve ser ignorado pelos decodificadores. O quarto caractere define se o *chunk* é ou não *Safe to Copy* ou, em outras palavras, se o bloco pode ser copiado de uma imagem fonte, e colocado em uma segunda imagem destino.

O bloco *header* do formato PNG é denominado IHDR *chunk*. Contém a altura e largura da imagem, a quantidade de bits utilizados por cada canal, o modo de armazenamento (*Grayscale*, *Pallete*, RGB, *Grayscale* com canal *Alpha*, RGB com canal *Alpha*), e se o arquivo encontra-se em formato entrelaçado.

Segundo Miano (1999), PLTE é o *chunk* que contém a *Color Pallete*. É utilizado de forma obrigatória em imagens que têm o modo *Pallete* definido no IHDR. Caso a imagem seja RGB ou RGB com canal *Alpha*, a imagem pode conter um bloco PLTE opcional, a ser usado caso a imagem necessite ser quantizada para 256 cores. Arquivos que utilizam o *Grayscale* não possuem PLTE nem de forma opcional, pois este padrão utiliza luminosidade, ao contrário da paleta de cores, que utiliza cores.

Os dados da imagem são comprimidos e encontrados em *chunks* chamados IDAT. Todos os blocos IDAT de um arquivo PNG devem ser consecutivos, não havendo nenhum outro *chunk* entre eles. Podem ser localizados em qualquer lugar do arquivo PNG entre o IHDR e o IEND (*chunk* de finalização do arquivo), exceto quando o arquivo contém *Color Pallete*. Quando isso ocorre, os blocos IDAT devem ser encontrados entre o PLTE e o IEND.

Existem outros *chunks* não críticos que são definidos pelo padrão. Um exemplo é o bloco bKGD, que contém a cor que deve ser utilizada de plano de fundo da imagem PNG. A informação contida neste chunk depende do modo do arquivo, definida no IHDR (*Grayscale*, RGB, etc).

Conforme Miano (1999), os blocos de dados IDAT são comprimidos utilizando o *Deflate Compression*, que são comprimidos e descomprimidos utilizando uma variante do processo de compressão chamado LZ77. Os algoritmos referentes a este processo são encontrados em uma biblioteca de nome ZLIB. O *Deflate*, também é utilizado no formato

GZIP de compressão de arquivos, além de ser utilizado nos dados do PNG. Em conjunto com o *Deflate*, o PNG utiliza o algoritmo *Huffman*.

1.1.5 Digital Imaging and Communications in Medicine (DICOM)

O formato de arquivos DICOM (*Digital Imaging and Communications in Medicine*), desenvolvido pelo Colégio Americano de Radiologia (ACR) em conjunto com o NEMA (*National Electrical Manufacturers Association*) é um padrão de arquivo criado para a comunicação e troca de informações entre equipamentos digitais para diagnósticos médicos. As tomografias computadorizadas são exemplos de diagnósticos médicos que podem utilizar o formato de arquivo DICOM. O formato consiste na criação de um arquivo que pode conter informações diversas sobre os diagnósticos, de maneira a serem lidos e interpretados por outros equipamentos digitais sem a perda ou alteração de informação. Em outras palavras, o formato foi inventado para transferir informação médica entre dispositivos eletrônicos de forma padronizada. Dentre estas informações podem estar contidos dados sobre o paciente, informações sobre o aparelho no qual o exame foi feito, imagens referentes a diagnósticos, etc. (NEMA, 1999).

Segundo Santos (2003), o padrão é mais do que um formato para transferência de imagens médicas, pois suporta dados e informações relacionadas, serviços relacionados a comunicação, consulta de imagens em uma base de dados que será vista a seguir, dentre outros.

Os arquivos DICOM adotam o método de programação de orientação a objetos. Assim, cada informação referente a um objeto DICOM pode conter métodos relacionados a ela. Estes métodos ou funções em DICOM são chamados serviços. O conjunto do objeto de informação com o serviço é chamado de *Service Object Pair* (SOP) ou Paridade Serviço-Objeto. As informações dos SOPs são definidas em IODs (*Information Object Definition*), ou definições de objetos de informação.

A seguir, algumas definições que são comumente utilizadas no padrão DICOM:

Tabela 2.1 – Elementos da estrutura de arquivos DICOM.

Nome	Definição
BASIC OFFSET TABLE	Tabela de ponteiros que indicam *frames* de uma imagem encapsulada no arquivo, e que contém *multi-frames*.
CHARACTER REPERTOIRE	Grupo finito de caracteres que é considerado como necessário para um determinado propósito, sem importar a codificação de caracteres do arquivo.
DATA ELEMENT	Dados referentes a elementos do arquivo (IODs codificados)
DATA ELEMENT TAG	Identificador numérico único do *Data Element*
DATA ELEMENT TYPE	Utilizado para demarcar obrigatoriedade, obrigatoriedade sob condições ou opcional. Utilizando tanto para atributos de IOD quanto para atributos de SOP.
PIXEL CELL	Recipiente de informação de um único pixel
PIXEL DATA	Dados gráficos compostos por variáveis codificadas em *Pixel Data Element*. Podem conter *Data Elements* descritivos com informação sobre a imagem.
PRIVATE DATA ELEMENT	Data Element adicional definido pelo criador do arquivo DICOM para transmitir informações que não são definidas no padrão do formato do arquivo.
STANDARD DATA ELEMENT	Elemento definido no padrão DICOM e encontrado no dicionário de dados de elementos do formato.
VALUE	Componente do *Value Field*, que pode ser composto por um ou mais *Values*.
VALUE FIELD	Campo que contém um ou mais valores do *Data Element*.
VALUE LENGTH	Campo que contém o comprimento do *Value Field*.
VALUE MULTIPLICITY (VM)	Especifica o número de valores contidos no *Value Field*.
VALUE REPRESENTATION (VR)	Especifica os tipos e formatos dos dados do *Value Field*.
VALUE REPRESENTATION FIELD	Local de armazenamento dos dados codificados do *Value Field* com VR explícito no *Data Element*.

Fonte: NEMA, 1999.

Conforme já mencionado anteriormente, o formato de arquivos DICOM não é um formato desenvolvido para arquivos de imagem. Este padrão foi desenvolvido como metodologia para comunicação entre equipamentos. Além de conter informações referentes a imagens, ele também contém informações técnicas referentes ao conteúdo do arquivo.

O formato contém duas maneiras de armazenamento de arquivos de imagem. A primeira forma é a nativa (não comprimida), e a segunda é a encapsulada, que utiliza padrões definidos fora do formato DICOM, e que podem utilizar compressão. Em arquivos que contém imagem no formato nativo, os dados da imagem são trazidos em *Pixel Cells*. Para dados encapsulados, são suportados tanto arquivos com um único *frame* quanto arquivos com

mais de um *frame*. Através do modo de encapsulamento, o DICOM provê um suporte ao formato JPEG, mas com algumas limitações devido à perda de informações na compressão JPEG. O padrão também provê suporte a imagens codificadas com compressão RLE.

1.2 Algoritmos de Compressão

Neste item são abordados os algoritmos de compressão de imagem utilizados nos formatos de arquivos estudados no item 1.1. O estudo destes algoritmos auxiliará o sistema conversor de forma a facilitar o entendimento das operações a serem realizadas.

1.2.1 Algoritmo *Run-Length Encoding* (RLE)

De acordo com Fridrich et al. (2004), o algoritmo de compressão *Run-Length Encoding* caracteriza-se por substituir seqüências de caracteres repetidos por dados que contém a quantidade da repetição do caractere, seguida pela informação que refere o caractere replicado. Este algoritmo é utilizado em bitmaps que contém mais de 256 cores.

Miano (1999) explica que o algoritmo de compressão RLE aplica-se em imagens de formato Bitmap, utilizando 4 ou 8 *bits* por *pixel*. A primeira utilizará RLE4 se o atributo "compressão" da estrutura de cabeçalho *BitmapInfoHeader* conter o valor 2 (que equivale ao tipo de compressão RLE4). A segunda utilizará RLE8 caso seu atributo "compressão" contenha o valor 1 (indicando compressão RLE8). As compressões não podem ser utilizadas por imagens com outros padrões de *bits* por *pixel*.

Segundo Fridrich et al. (2004), a forma de decodificação é bastante simples. São cinco as principais regras que definem o funcionamento do algoritmo de descompressão. Para a codificação dos dados, cada dado contido no corpo da estrutura deve ser interpretado. Por exemplo: *n* B, onde o *byte* é repetido por *n* vezes quando *n* for maior que 0. Se *n* for igual a zero, devem ser aplicadas as demais regras. Ao capturar um dado onde o *n* é 0, e o segundo *byte* também é 0, o dado deve ser interpretado como EOL (*End of Line*). Caso *n* seja 0 e o segundo *byte* seja 1, a informação recebida é de EOB (*End of Bitmap*). Ao receber 0 2 o decodificador deve esperar para receber mais dois dados referentes às posições *x* e *y*, que

completam a regra *Delta*. Esta regra indica que o ponteiro deve ser movido *x pixels* para a direita e *y pixels* para baixo.

Quando o segundo *byte* for igual ou maior que o número 3, o decodificador utiliza a regra intitulada Modo Absoluto, a qual o segundo número apontará a quantidade de dados que seguirão fazendo referência a dados únicos, que não repetem mais de uma vez. Este modo é utilizado para economizar tamanho. Por exemplo, se forem dados referentes a três *pixels* de cores diferentes, o modo absoluto é ativado, para que não haja a necessidade de referenciar o *n* sendo 1. Neste exemplo, economiza-se as posições do *n* = 1 em 3 vezes apenas, mas se for comparada em arquivos grandes, a redução de informação é considerável. Devido ao emprego ou não desta regra, os algoritmos compactadores de RLE podem gerar arquivos com diferentes tamanhos e contendo a mesma informação.

A imagem a seguir representa a codificação do RLE sendo utilizada por algoritmos com e sem o emprego do modo absoluto.

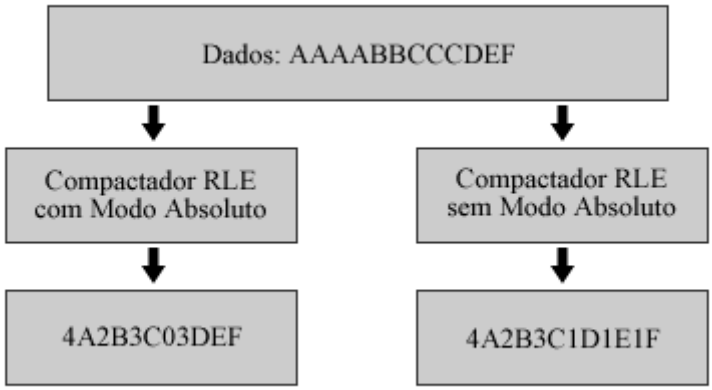

Figura 2.1 – Codificação de Run-Length Encoding
Fonte: Friedrich, 2004.

1.2.2 Algoritmo LZW (Lempel-Ziv & Welch)

Segundo Nelson (1989), a versão do algoritmo LZW é baseada nos algoritmos LZ77 e LZ78, que foram publicados respectivamente em 1977 e 1978, ambos por Abraham Lempel e Jacob Ziv. Mais tarde, Terry Welch aplicou refinamentos ao algoritmo, publicando o LZW em 1984.

O algoritmo LZW baseia sua compressão em um dicionário de termos. Seu objetivo é tentar retornar o máximo de índices possíveis referentes a termos já conhecidos no dicionário. Este dicionário inicialmente já vem preenchido com os caracteres básicos. Por exemplo, quando utilizados caracteres de 8 *bits*, os primeiros 256 caracteres são por padrão adicionados ao dicionário, ocupando a posição de 0 a 255. Entre a posição 256 e a posição final do dicionário, são adicionados termos existentes nos dados de entrada do algoritmo.

O algoritmo consiste em sempre adicionar palavras novas ao dicionário, fazendo com que blocos de informação tornem-se referências de termos do dicionário. Novos termos são constantemente adicionados ao dicionário para que sempre exista um aumento do número de termos conhecidos e uma diminuição de caracteres a serem repetidos. A rotina de captura de termos e de compressão pode ser entendida pela figura 2.2 a seguir.

```
CODE:

  1. STRING = get input character
  2. WHILE there are still input characters DO
  3.     CHARACTER = get input character
  4.     IF STRING+CHARACTER is in the string table then
  5.         STRING = STRING+character
  6.     ELSE
  7.         output the code for STRING
  8.         add STRING+CHARACTER to the string table
  9.         STRING = CHARACTER
 10.     END of IF
 11. END of WHILE
 12. output the code for STRING
```

Figura 2.2 – Exemplo de rotina de compressão LZW
Fonte: Nelson, 1989.

O algoritmo de descompressão usado no LZW utiliza dados provindos da compressão do primeiro algoritmo, para recriar os dados exatamente iguais. Dois dos fatores que fazem com que a compressão LZW seja bastante eficiente são o fato de não haver perda de dados e a inteligência do algoritmo em não necessitar enviar a tabela de termos (dicionário) para a descompressão. Esta tabela pode ser recriada exatamente como foi criada na compressão. Essa possibilidade existe por sempre retornar palavras e caracteres já utilizados anteriormente. Com isso, o tamanho do arquivo acaba sendo preenchido com uma maior quantidade de informações realmente pertencentes ao arquivo, e não com metadados utilizados na compressão, como tabelas de dicionário.

```
CODE:

 1. Read OLD_CODE
 2. output OLD_CODE
 3. WHILE there are still input characters DO
 4.     Read NEW_CODE
 5.     STRING = get translation of NEW_CODE
 6.     output STRING
 7.     CHARACTER = first character in STRING
 8.     add OLD_CODE + CHARACTER to the translation table
 9.     OLD_CODE = NEW_CODE
10. END of WHILE
```

Figura 2.3 – Exemplo de rotina de descompressão LZW
Fonte: Nelson, 1989.

O algoritmo de descompressão deve remontar a tabela de termos ao longo do seu processo. A tabela recriada neste segundo momento acaba sendo exatamente igual à tabela de termos do algoritmo de compressão, incluindo os primeiros caracteres fixos contendo o conjunto de caracteres utilizados na compressão (no caso os primeiros 256 caracteres).

O algoritmo de descompressão apresentado na figura 2.3 demonstra uma idéia de como deve funcionar a retirada das informações do conteúdo compactado. Porém, se for implementado com exato algoritmo, o programa de descompactação pode encontrar problemas na montagem do dicionário por causa de exceções. A seguir, na figura 2.4 uma estrutura de algoritmo mais complexa para a resolução dos problemas encontrados na implementação do processo de descompactação LZW simples.

```
CODE:

 1. Read OLD_CODE
 2. output OLD_CODE
 3. CHARACTER = OLD_CODE
 4. WHILE there are still input characters DO
 5.     Read NEW_CODE
 6.     IF NEW_CODE is not in the translation table THEN
 7.         STRING = get translation of OLD_CODE
 8.         STRING = STRING+CHARACTER
 9.     ELSE
10.         STRING = get translation of NEW_CODE
11.     END of IF
12.     output STRING
13.     CHARACTER = first character in STRING
14.     add OLD_CODE + CHARACTER to the translation table
15.     OLD_CODE = NEW_CODE
16. END of WHILE
```

Figura 2.4 – Exemplo de rotina mais robusta de descompressão LZW
Fonte: Nelson, 1989.

Segundo Nelson (1989), é difícil de caracterizar os resultados de uma técnica de compressão. Sua eficiência nos resultados depende de diversos fatores. A utilização do algoritmo de compressão LZW funciona quando utilizada para textos, tendo expectativa de comprimir o conteúdo em 50% ou mais. Já a compressão de dados binários pode ou não trazer resultados não tão eficazes.

1.2.3 Algoritmo *Huffman Coding*

Segundo Miano (1999), este algoritmo foi inventado em 1952 por David A. Huffman. O *Huffman Coding* tem suas técnicas utilizadas nos padrões de arquivos de imagem JPEG e PNG. Sua base está na medida de freqüência de utilização dos caracteres. A utilização da referência da posição dos caracteres ao invés de utilizar os mesmos, faz com que o número de bytes utilizados na referência seja dinâmico, enquanto que caracteres em ASCII, por exemplo, contém um tamanho padrão de *bytes* pré-definido. Em outras palavras, caracteres do conjunto ASCII têm a mesma quantidade de *bytes*, mesmo que não necessitem da sua utilização por completo. Já as referências à determinadas posições de tabela não têm sua largura variável. Sendo assim, os números que apontam para as primeiras posições contendo apenas um caractere numérico, terão a utilização de menos *bits* do que os que apontam para uma posição de número maior. Espera-se que o número de *bits* economizados na apresentação dos códigos dos elementos mais freqüentes em uma informação seja maior do que o déficit de *bits* dos caracteres menos freqüentes e que possuam uma maior quantidade de *bits* para sua representação.

A idéia de classificar as letras por freqüência foi utilizada na criação do Código Morse, onde as letras mais freqüentes têm seus códigos mais curtos. Conforme Miano (1999), o *Huffman Coding* é a melhor técnica para se gerar códigos com tamanhos variáveis para símbolos.

Para sua execução, é criada uma árvore binária onde são colocados de baixo para cima, os símbolos de maior freqüência. É importante ressaltar que ao trabalhar com a árvore binária, o algoritmo de *Huffman* acaba gerando códigos binários.

A codificação inicia com todos os elementos e suas respectivas freqüências em uma fila e nenhum nodo. A construção desta árvore é feita dentro de uma estrutura de repetição que verifica o término dos elementos da fila. Enquanto existem elementos na fila, são executados quatro passos. O primeiro passo é localizar os dois valores ou nodos com a menor freqüência para removê-los da fila. O segundo passo é a criação de um novo nodo, que recebe os dois valores selecionados no passo anterior. O terceiro passo é colocar como freqüência do nodo a soma das freqüências dos seus filhos (adicionados no passo 2). E, por último, o nodo criado é adicionado à fila novamente, utilizando a freqüência que lhe foi atribuída.

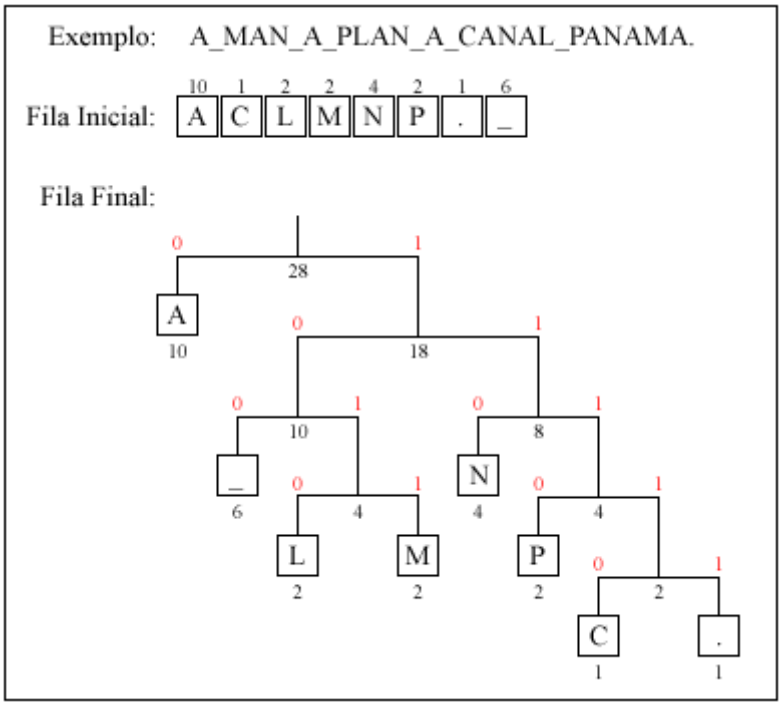

Figura 3.1 – Exemplo de Árvore binária de *Huffman Coding*
Fonte: Miano, 1999.

Após criada toda a árvore binária, são adicionados os *bits* 0 e 1 a cada símbolo da árvore. A representação de cada símbolo é dada pelo caminho a ser percorrido na árvore, da raiz até o símbolo. A seguir, a tabela com a relação de símbolos utilizados no exemplo da figura 3.1 e suas respectivas propriedades geradas pelo *Huffman Coding*.

Tabela 3.1 – Códigos Huffman para os símbolos exibidos no exemplo da figura 3.1.

Valor	Código *Huffman*	Tamanho	Freqüência	*Bits* utilizados
A	0	1	10	10
C	11110	5	1	5
L	1010	4	2	8
M	1011	4	2	8
N	110	3	4	12

Valor	Código *Huffman*	Tamanho	Freqüência	*Bits* utilizados
P	1110	4	2	8
_	100	3	6	18
.	11111	5	1	5
				Total: 74 *bits*

Fonte: Miano, 1999.

Existe um aspecto que faz com que os códigos gerados pelo algoritmo possam depender da implementação do seu algoritmo. O posicionamento de símbolos que aparecem com a mesma freqüência não tem uma definição específica. Isso faz com que a regra de posicionamento seja definida e implementada ao se programar o algoritmo.

1.2.4 Algoritmo *Deflate*

Segundo Deutsch (1996), o algoritmo de compressão chamado de *Deflate* é uma combinação do algoritmo LZ77 com *Huffman Coding*. O conjunto de dados a serem comprimidos utilizando *Deflate*, é separado em blocos sucessivos de dados de entrada. Cada bloco tem seu conteúdo comprimido com LZ77 e *Huffman Coding*. Cada bloco de informação contém sua própria árvore binária, independente das informações do restante das informações. Já a compressão LZ77 aplicada ao bloco pode utilizar termos já utilizados em um bloco anterior, e que ocorrem dentro da condição de se encontrar a uma posição a menos de 32 *Kbytes* de distância.

Cada bloco comprimido contém a sua árvore binária criada através do *Huffman Coding* com os seus códigos referentes à informação do bloco, e uma segunda área contendo dados comprimidos em LZ77. A área de conteúdo comprimido é composta por elementos de dois tipos, conforme o algoritmo utilizado. São eles, *bytes* literais contendo termos não repetidos anteriormente, e ponteiros para termos encontrados anteriores a menos de 32kb atrás. Os ponteiros pares de informação que contém o tamanho do termo e o início de onde o termo se encontra.

Segundo Soares et al. (1999), o *header* de cada bloco de dados comprimido utilizando o algoritmo *Deflate* contém duas informações. A primeira é uma *flag* que indica se o bloco é o último bloco do arquivo, e a segunda indica o tipo de compressão. Os modos de compressão para o *Deflate* são três. O representado pelos *bits* 00 que significa sem

compressão, o de *bits* 01, compressão LZ77 com *Huffman* estático e 10 que representa a compressão LZ77 com *Huffman* semi-adaptativo. A diferença entre os dois tipos está na tabela de códigos *Huffman*, onde no modelo estático se tem apenas uma tabela geral para todos os blocos comprimidos, enquanto na segunda, é enviada a tabela de códigos *Huffman* para cada bloco que é comprimido utilizando este algoritmo.

Segundo Soares et. al. (1999), na utilização do *Huffman* para o *Deflate* são necessárias algumas variações no algoritmo. Isso ocorre pelo fato de um mesmo conjunto de símbolos poder originar árvores de *Huffman* diferentes. Assim, para evitar a ambigüidade das árvores de *Huffman*, são aplicadas duas restrições ao algoritmo. Os códigos contendo o mesmo comprimento precedem lexicograficamente códigos que contenham o comprimento maior. Um exemplo é o código 10, que precede 110, que precede 1110. A segunda regra é baseada na ordem dos códigos com mesmo comprimento. A regra aplica a ordenação destes códigos de acordo com a ordenação dos símbolos os quais eles codificam.

Além de arquivos de imagens no formato PNG, o *Deflate* é utilizado em arquivos comprimidos com GZip. (Soares et. al., 1999)

2 CONVERSORES

Este capítulo tem como objetivo apresentar sistemas conversores de imagens existentes no mercado. São apresentados, em uma primeira etapa, alguns dos softwares que fazem estas conversões. No segundo tópico deste capítulo são apresentadas bibliotecas de programação utilizadas para conversão de imagens.

A importância do conhecimento gerado pelo estudo feito a partir deste capítulo está na pesquisa de *benchmark*, na qual são estudadas as melhores aplicações e práticas referentes à conversão de imagens, tanto em softwares, quanto em *frameworks*, bibliotecas de programação e em trabalhos semelhantes a este.

2.1 Softwares de conversão de imagens

No mercado existem muitos softwares que provêem soluções para os problemas de conversão de arquivos de imagens. Alguns têm o objetivo específico de converter imagens, enquanto que outros têm seus objetivos focados na edição de imagem, mas possuem esta solução como característica adicional. Softwares específicos para edição de imagens como Adobe Photoshop, Adobe Fireworks e GIMP são exemplos que tem esta característica como ferramenta adicional. Como software de objetivo específico de conversão de imagens, podem ser citados o Image Converter Plus e o 36 Image Converter.

Os programas de edição de imagens têm seu foco em profissionais da área de design. Alguns direcionados à manipulação de fotografia, como o Photoshop e GIMP, e outros com ideais voltados à criação e produção de material gráfico, como o Fireworks.

2.1.1 Adobe Photoshop CS3

O Adobe Photoshop CS3 é tido como referência se tratando de programas de manipulação de imagem. Possui um formato de arquivo específico que utiliza a extensão PSD para armazenar os arquivos-fonte das criações feitas pelo aplicativo. O conteúdo dos arquivos desta extensão contém informações organizadas da mesma maneira a qual foram geradas na aplicação, permitindo que sejam alteradas, copiadas, etc. A estrutura deste arquivo permite que conteúdos gráficos sejam criados a partir de camadas.

Apesar de conter uma extensão de arquivo própria, o software tem como propriedade a possibilidade de abrir imagens de diversos formatos, entre eles os formatos JPEG, GIF, PNG, TIFF e BMP, conforme a figura 4.1 a seguir.

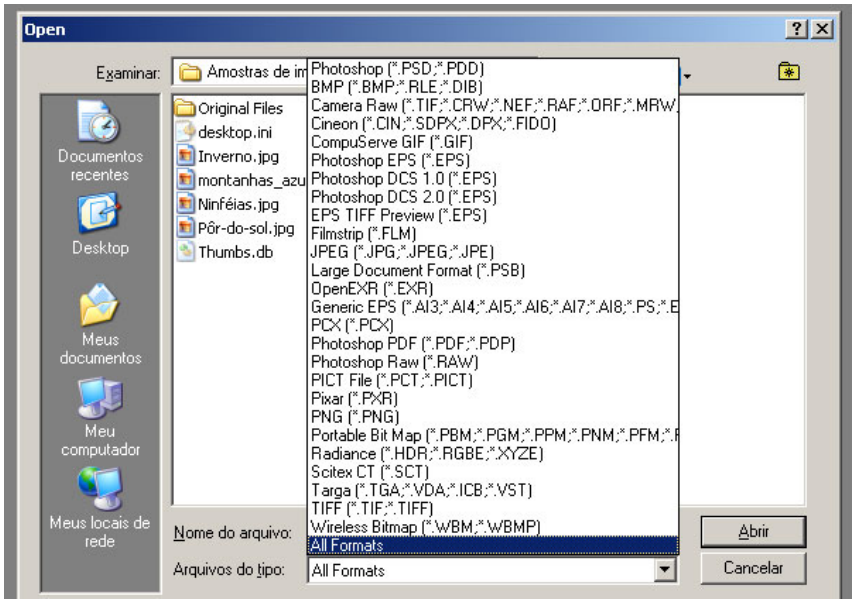

Figura 4.1 – Formatos de imagens aceitos pelo Adobe Photoshop CS3
Fonte: Adobe Photoshop, 2007.

Referente à conversão de imagens, o Photoshop possui três maneiras diferentes de se converter imagens. Como modo mais normal de se proceder, os arquivos de extensões legíveis pelo software podem ser abertos e salvos em outras extensões conforme é apresentado na figura 4.2.

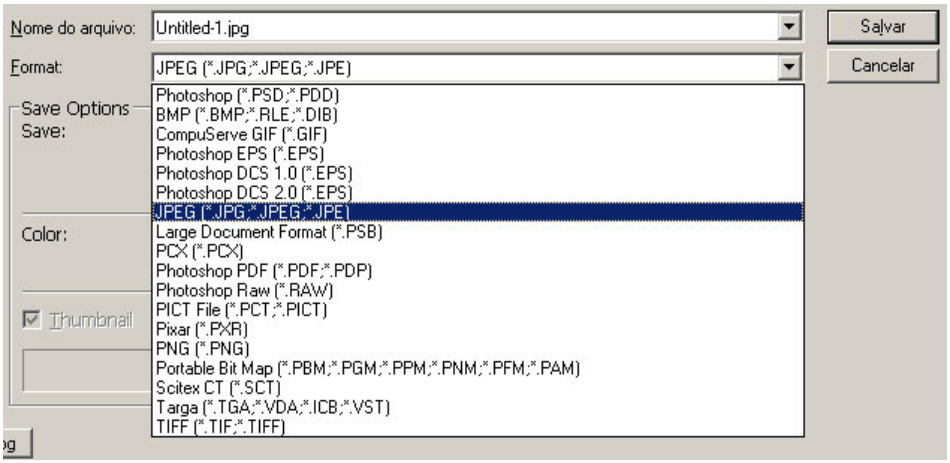

Figura 4.2 – Formatos de imagens aceitos para exportação pelo Adobe Photoshop CS3
Fonte: Adobe Photoshop, 2007.

O comando *Save for Web & Devices*, encontrado no menu *File* do software é a segunda alternativa, bastante utilizada para exportar arquivos de imagens em formatos diferentes. Informações referentes à estrutura de arquivos e propriedades de compactação podem ser configuradas neste modo. Este modo de salvar os arquivos permite que sejam escolhidos alguns formatos como GIF, PNG e JPG. Propriedades como qualidade da compressão, paleta de cores e transparência, além de outras, também podem ser manipuladas nesta forma de exportar, conforme é apresentado na imagem a seguir.

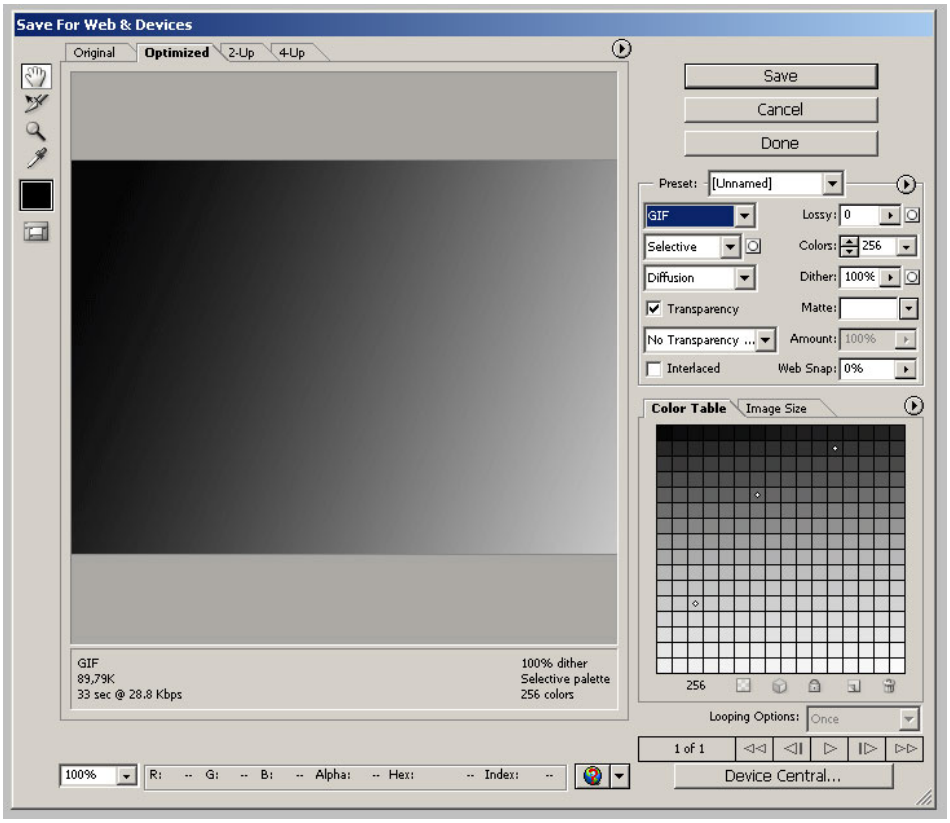

Figura 4.3 – Tela de *Save for Web & Devices*, Adobe Photoshop CS3
Fonte: Adobe Photoshop, 2007.

2.1.2 Adobe Fireworks CS3

O software Fireworks foi desenvolvido pela empresa Macromedia, que dominava o mercado de produção de softwares voltados ao desenvolvimento para a web. A suíte de softwares de desenvolvimento com foco em internet da Macromedia, além do Fireworks, contava com o software Flash para animações e interatividade, e com o software Dreamweaver, ambiente completo de desenvolvimento para programadores. Em 2006 a Adobe Systems Incorporated fez a aquisição da empresa Macromedia Inc. Sendo assim, estes e os demais programas da Macromedia Inc. passaram a pertencer à lista de produtos da Adobe Systems Incorporated. Com esta mudança, alguns produtos como o Macromedia Freehand deixaram de ser aperfeiçoados, e outros como o Fireworks, continuaram, tendo apenas o nome do seu pacote, que antes era intitulado Macromedia Studio, passando a se chamar Adobe Creative Suite.

Conforme Adobe Systems Incorporated (2008), o principal objetivo deste aplicativo é prover aos seus usuários um ambiente propício para um rápido desenvolvimento de interfaces para websites e aplicações.

Ao contrário do Adobe Photoshop, o Adobe Fireworks não possui uma extensão de arquivos própria. Utiliza o padrão de arquivos de imagem PNG, que, conforme visto no tópico 1.1.4, é composto por blocos de arquivos chamados de *chunks*. Estes blocos podem ser personalizados pelos programas, que acabam tendo um arquivo de imagem padrão, mas com propriedades específicas.

Além dos padrões de arquivos PNG, o Fireworks também suporta os outros formatos mais comuns de imagem como JPEG, GIF e Bitmap. Possui a opção de salvar suas imagens nestes formatos e outros formatos de arquivos de imagem. Além do modelo padrão de conversão baseado em salvar os arquivos, o Fireworks possui uma ferramenta chamada *Optimize*, que possibilita uma configuração a fundo em formatos de arquivos para exportar imagens ou segmentos de imagens. A seguir, na figura 4.4 é apresentada uma demonstração de configuração de paleta de cores, e de propriedades do padrão de arquivos GIF.

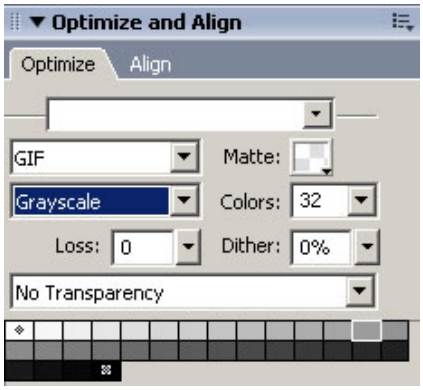

Figura 4.4 – Ferramenta *Optimize* do Adobe Fireworks
Fonte: Adobe Fireworks, 2007.

O Fireworks também permite fazer *scripts* de processos em lote, ou *Batch Process*. Estes scripts são formas pré-programadas de aplicar configurações a conjuntos de imagens, para facilitar certos tipos de trabalhos. Junto com a funcionabilidade do *Batch Process*, existe a possibilidade de exportar estas imagens selecionadas para outros padrões de arquivos. A imagem a seguir apresenta o formulário de configuração do processo em lote do Fireworks.

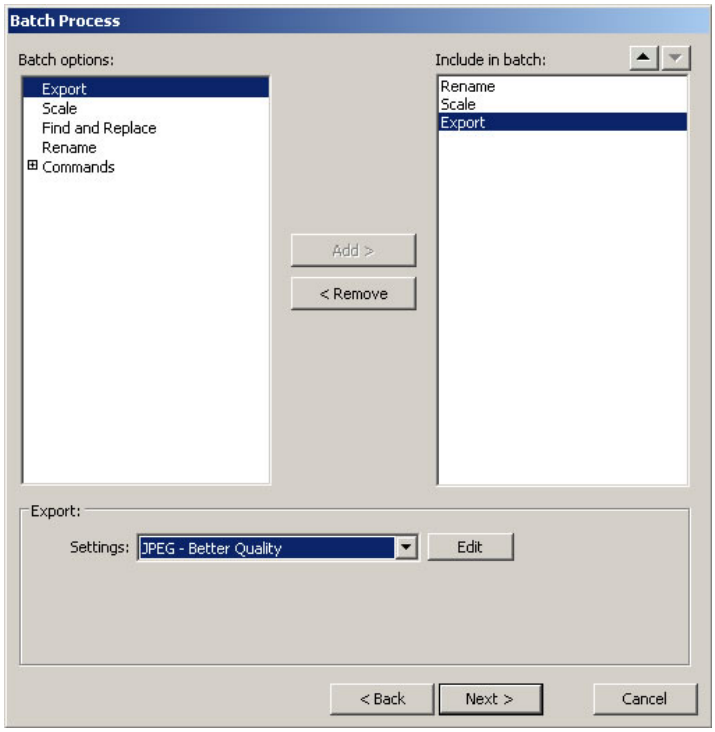

Figura 4.5 – Ferramenta *Batch Process* Adobe Fireworks
Fonte: Adobe Fireworks, 2007.

2.1.3 *GNU Image Manipulation Program* (GIMP)

O GIMP é um software de edição de imagens desenvolvido inicialmente para o ambiente Linux. Seu desenvolvimento iniciou em 1995, e teve a versão 1.0 publicada em

junho de 1998. Hoje encontra-se com a versão estável 2.4, tendo versões para os sistemas operacionais Windows e Mac OS.

O software possui um padrão de arquivos próprio chamado XCF, mas também permite que os arquivos sejam salvos nos formatos mais difundidos, como JPEG, GIF, PNG, BMP, etc. Além dos formatos comuns de arquivos de imagem, o software possibilita salvar imagens no formato de arquivos DICOM, conforme a figura 4.6 a seguir.

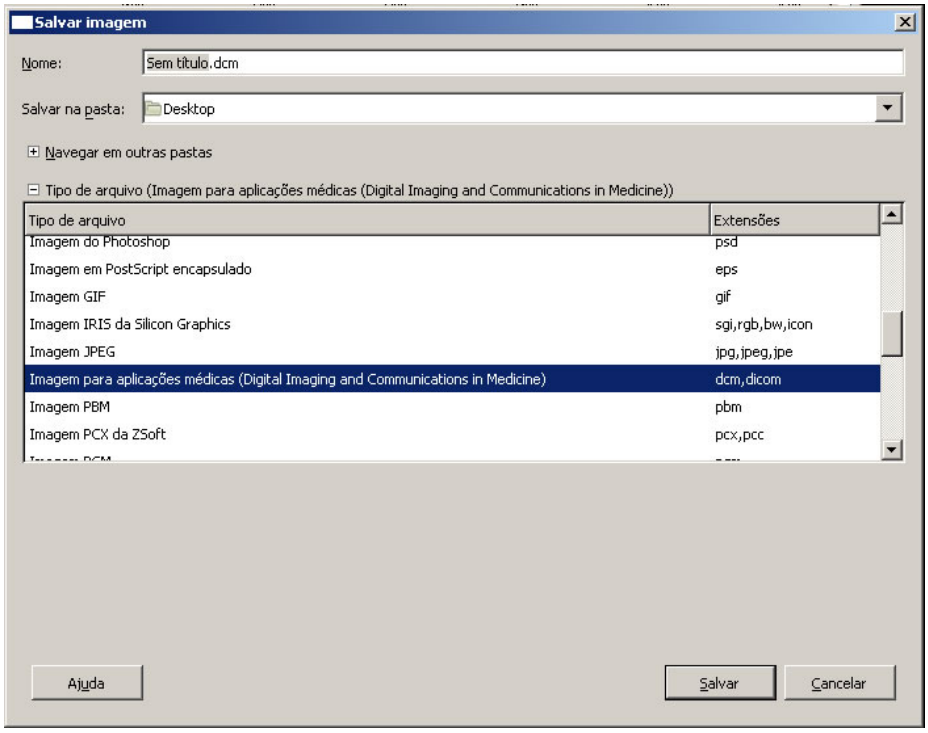

Figura 4.6 – Diálogo para salvar imagens do GIMP
Fonte: GIMP, 2007.

Além de possibilitar que arquivos sejam salvos com determinadas extensões, assim como o formato DICOM, o software também possibilita que sejam abertos arquivos neste e em diversos outros formatos.

2.1.4 Apresentação de “Ferramentas para visualização de imagens médicas em hospital universitário”

Segundo Caritá et al. (2004), foi desenvolvida uma biblioteca para manipulação de arquivos DICOM. Além desta biblioteca, é apresentado um software *Desktop* chamado *VDTApplication* (*Visual DICOM Toolkit Application*) e um software *Applet* chamado

VDTApplet (*Visual DICOM Toolkit Applet*) criado para ser executado via navegador. Este software utiliza a biblioteca e tem a função de visualizar arquivos DICOM.

Diferente da proposta deste trabalho, a linguagem utilizada na ferramenta foi o Java da empresa *SUN Microsystems*. Em sua implementação foram considerados importantes muitos aspectos que fizeram com que o projeto se tornasse genérico, tendo a leitura de uma maior variedade de imagens gravadas no padrão DICOM. Esta qualidade tornou o projeto hábil a ler e interpretar arquivos no formato DICOM gerados por diversos fabricantes. No *VDTApplet* foram implementadas rotinas que possibilitam a visualização de arquivos encontrados nos formatos JPEG, PNG, BMP, GIF entre outros, além do formato DICOM.

Segundo Caritá et al. (2004), o VDTApplet foi integrado a um software chamado RIS (*Radiology Information System*), cujo objetivo é melhorar o cadastramento e consulta de laudos através da rede *ethernet* do hospital, visando diminuir o tempo do processo. Os cadastros feitos através deste sistema contêm informações de exames, ligados a outras informações referentes aos pacientes, sobre os laudos e sobre a realização dos exames. Relatórios técnicos e de produtividade podem ser extraídos do sistema, facilitando o gerenciamento e o controle de qualidade do serviço.

A pesquisa feita sobre imagens de duas modalidades: tomografia computadorizada e ressonância magnética. As ferramentas foram avaliadas de forma qualitativa por dois médicos do hospital HCFMRP-USP, um radiologista e um neurologista. Como resultado das avaliações das ferramentas, os médicos chegaram a uma conclusão positiva, solicitando algumas melhorias na ferramenta de ajuste de brilho e contraste das imagens. Foram avaliados quatro aspectos do sistema: interface, tempo de recuperação da imagem, qualidade da imagem e ferramenta de manipulação da imagem. As imagens utilizadas no sistema foram escolhidas aleatoriamente pelos médicos. Os índices utilizados nas avaliações dos aspectos do sistema foram de 1 a 5, sendo 1 muito ruim e 5 muito bom. A avaliação da ferramenta de ajuste de imagem recebeu o índice 3, considerado satisfatório, pelos dois médicos. Os demais tópicos avaliados receberam pontuação 4, com exceção da interface do sistema que recebeu índice 5 para um dos médicos.

2.2 Bibliotecas de programação para formatos de arquivos de imagem

A evolução das linguagens de programação e das suas metodologias, em especial a orientação a objetos, trouxe uma facilidade para criação de estruturas genéricas para organização e reaproveitamento de código. Estas estruturas de programação podem ser utilizadas em diversos projetos, por conter apenas métodos relacionados à manipulação de informações pertinentes a si mesmas e por serem genericamente desenvolvidas para um aproveitamento geral. Este é o princípio das bibliotecas de programação, disponibilizar a execução de algoritmos para programas diversos não necessitarem de esforço para executar determinadas funções.

As bibliotecas pesquisadas neste trabalho referem-se a estruturas de programação que contém rotinas de manipulação de arquivos de imagem. Os estudos feitos são, em grande maioria, sobre códigos-fonte desenvolvidos na linguagem C#, que utiliza a orientação a objetos, no caso, as bibliotecas Free Image Lib.NET, *.NET Image Lib* e *openDICOM.NET*.

2.2.1 Biblioteca Free Image Lib.NET

O projeto *Free Image Lib.NET* foi registrado em 2006. Consiste em um conjunto de classes desenvolvidas na linguagem *C#*, que contém algoritmos para codificar e decodificar imagens nos formatos PNG, TGA e BMP.

Um dos motivos da seleção desta biblioteca para estudo é o desenvolvimento de todas as estruturas necessárias para trabalhar com o formato PNG, que entre os demais formatos de imagem, é um dos mais complicados. As rotinas de *Huffman Table*, *Deflate*, *Pallete*, *Chunk*, entre outras mostram a maturidade da biblioteca.

2.2.2 Biblioteca OpenDICOM.NET

Como referencial teórico para os estudos sobre conversão de imagens médicas, também são relevantes pesquisas de bibliotecas que trabalham com os arquivos de padrão DICOM. A biblioteca estudada para manipulação destes arquivos é a *openDICOM.NET*, que é

desenvolvida na linguagem *C#*, suportando ser compilada em *.MONO framework*, plataforma de desenvolvimento utilizada em ambientes Linux.

Esta biblioteca contém uma classe de extração de informações de arquivos DICOM, criando um conjunto de elementos de uma segunda classe, chamada de *DataElement*. Tem mapeados muitos tipos de *DataElements* que pertencem ao formato DICOM.

O projeto *openDICOM.NET* traz, além da biblioteca de manipulação de arquivos DICOM, alguns softwares para extração de informação via console que utilizam a biblioteca, e também um software para visualização de arquivos DICOM.

3 PROPOSTA DE SISTEMA CONVERSOR DE IMAGENS

Este trabalho tem como proposta a criação de uma biblioteca expansível que tenha o princípio de converter imagens entre diversos formatos de arquivos. Os estudos realizados sobre os padrões de arquivos de imagens, suas estruturas e seus algoritmos de compressão trazem o referencial teórico necessário para o conhecimento do escopo tratado no desenvolvimento da proposta.

No tópico 3.1 são apresentadas as metodologias a serem aplicadas para o desenvolvimento do projeto, quanto no tópico 3.2 são apresentadas informações relacionadas à modelagem do projeto.

3.1 Metodologia

A implementação da biblioteca atende necessidades relacionadas a um projeto maior chamado AMPLIA-I. A portabilidade relacionada ao projeto contempla a possibilidade da inclusão de padrões de arquivos não estudados neste trabalho, de forma a não restringir a evolução da ferramenta de conversão dentro do projeto AMPLIA-I. O sucesso do fator portabilidade depende da modelagem e da previsão de eventuais problemas relacionados à grande diferença entre os padrões de arquivos.

O desenvolvimento da biblioteca, assim como o projeto AMPLIA-I, é feito através da linguagem Microsoft Visual C# utilizando o *.NET Framework*. O projeto maior no qual o sistema conversor de imagens deve ser inserido também é desenvolvido com esta linguagem, criando assim a necessidade de implementação da mesma sobre a mesma plataforma, e nos mesmos padrões de desenvolvimento dos demais módulos do AMPLIA-I.

As técnicas utilizadas no desenvolvimento da biblioteca do sistema conversor de imagens são as mesmas aplicadas pelos outros desenvolvedores do projeto AMPLIA-I, para que haja uma maior homogeneidade na sua codificação. A plataforma .NET e a linguagem C# utilizam da estrutura de codificação no padrão de orientação a objetos, trazendo portabilidade ao projeto e um ganho no reaproveitamento de código-fonte.

Além da criação da biblioteca para ser utilizada em um sistema de conversão de arquivos de imagens acoplado ao projeto AMPLIA-I, a biblioteca poderá ser disponibilizada de forma a poder contribuir para outros projetos que necessitem da conversão.

3.2 Princípios do algoritmo genérico do conversor de imagens

A conversão de imagens tem como objetivo ter uma entrada de um arquivo em um determinado formato e a saída de um novo arquivo em um padrão diferente. Para que a conversão possa ser feita entre diversos formatos, um formato deve ser escolhido para fazer o intermédio entre os demais. No caso, a escolha do formato Bitmap será feita por conter o formato de arquivo de imagem mais simples, por conter a compressão sem perda de dados e por ter manipulação nativa na linguagem C#.

Com a definição do formato base do sistema conversor, acaba por ser definido por conseqüência o formato de entrada dos métodos de conversão de todos os padrões de arquivos de imagem assim como o formato de saída de todos os métodos de captura de dados da imagem de todos os padrões.

3.3 Modelagem de dados da biblioteca de conversão

Como núcleo do sistema conversor de imagens, a biblioteca de conversão deve ser estudada de maneira a fornecer futuramente uma boa maturidade para ter continuidade. Esta continuação será feita através do acréscimo de padrões de arquivos para conversão. A seguir é apresentado o diagrama de classes no padrão UML, exemplificando a proposta de uma modelagem de padronização de arquivos exeqüível e proporcionando a possibilidade de extensão futura da biblioteca.

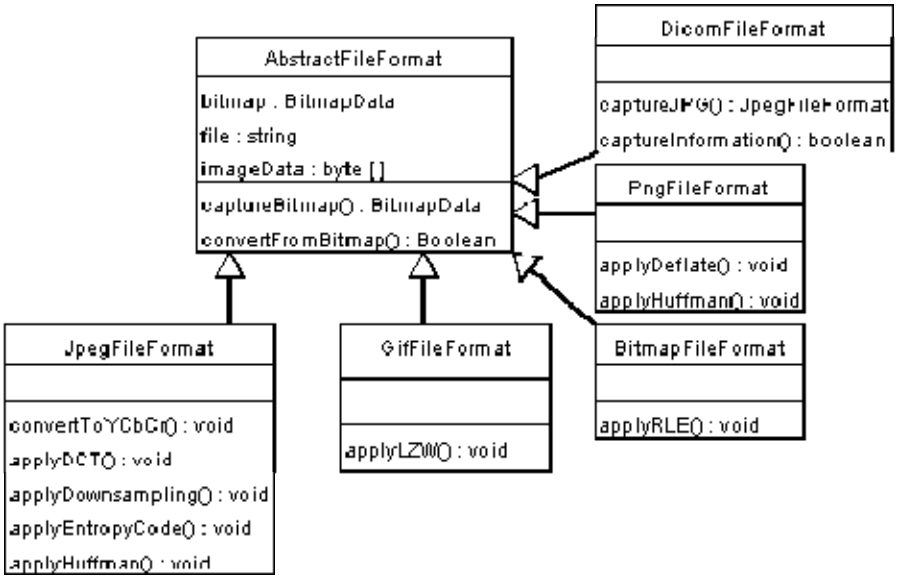

Figura 5.1 – Exemplo de diagrama de classes da biblioteca de conversão

O modelo da figura 5.1 apresenta a idéia da abstração dos formatos de arquivos de imagem, mas não apresenta os métodos que realmente serão utilizados no desenvolvimento da biblioteca. A classe abstrata de formato de arquivos contém as informações necessárias que serão utilizadas em todos os formatos de arquivo adicionados na biblioteca. Esta classe conterá também os métodos utilizados para fazer a conversão do formato para bitmap, e do bitmap para o formato. Ao generalizar a classe, a herança trará os métodos, que devem ser sobrescritos com os métodos realmente utilizados pelo formato.

O *.NET Framework* será a base do desenvolvimento desta biblioteca. Alguns dos algoritmos referentes aos formatos de arquivos JPEG, GIF e Bitmap já estão codificados no framework, não sendo necessária a reimplementação dos mesmos. Nestes casos, serão inseridas formas de adequação à execução dos algoritmos do *framework*, organizados em classes padronizadas semelhantes às do modelo proposto na figura 5.1.

3.4 Modelagem de dados do sistema conversor

Após a implementação da biblioteca de conversão, o sistema conversor será o *framework* da utilização da biblioteca. Este sistema é a composição da interface gráfica, com as funcionalidades da biblioteca.

A integração do sistema conversor com o AMPLIA-I poderá ser feita de duas formas. A primeira forma é transparente para o usuário do sistema, onde o diálogo de seleção de imagem permitirá a seleção de imagens pertencentes aos padrões implementados na biblioteca. Estas imagens serão abertas para o usuário sem a necessidade do conhecimento de padrões de arquivos de imagem.

A segunda forma é a adição de um item no menu do programa, onde conteriam comandos de importação e exportação de imagem. Para esta forma o usuário necessitaria de conhecimentos básicos de formatos de arquivos de imagem para selecionar os tipos de arquivos a serem importados ou para selecionar o formato de arquivo para o qual a imagem deve ser exportada.

Estas duas formas são hipóteses da implementação do sistema conversor de imagens integrado ao AMPLIA-I. As definições das formas de desenvolvimento deverão ser feitas após a construção da biblioteca de conversão.

Um foco importante para este trabalho é o tratamento de dados extraídos de arquivos DICOM. As informações contidas em arquivos de formato DICOM, além das imagens, podem ter diversas formas de dados. Os conteúdos textuais deverão ser extraídos e apresentados ao usuário na forma de arquivos XML (Extensive Markup Language), enquanto que imagens serão apresentadas como os demais formatos.

CONCLUSÃO

Foram realizados estudos sobre diversos formatos de arquivos de imagens. Os estudos mostraram a complexidade destas estruturas, em especial o formato PNG, utilizando modelos de dados contendo informações gerais lidas por aplicativos em geral e dados específicos criados especificamente para determinados programas.

O formato DICOM, um dos focos da proposta, mostrou-se bastante completo por conter, além de imagem, informações referentes ao laudo, ao equipamento, bem como servir como protocolo de comunicação entre alguns equipamentos médicos.

Foram estudados os algoritmos utilizados na compressão de imagens, nos quais utilizam muitas técnicas. Alguns com algoritmos que geram uma redução de tamanho através de perda de dados da imagem, outros utilizam dicionários de dados, outros com algoritmos mais simples e sem perda de dados, e no caso de arquivos em formatos PNG, utilizando algoritmos aplicados em softwares de compactação de arquivos em geral.

A pesquisa abrangeu estudos sobre programas que possuem rotinas de conversão de arquivos de imagem. A ferramenta de visualização de arquivos DICOM implantada em um hospital e avaliada por médicos também contribuiu para conhecimentos que podem ser aplicados futuramente ao projeto AMPLIA-I.

O desenvolvimento da ferramenta será dividido em duas etapas. A primeira será a biblioteca de classes, na qual haverá uma tentativa de se possibilitar a conversão de imagens entre todos os formatos desenvolvidos. Para conversões de arquivos DICOM para demais formatos, serão gerados arquivos XML contendo as informações textuais que acompanham as imagens no arquivo. Na segunda fase será desenvolvido o *framework* onde a biblioteca será aplicada, com o mesmo padrão utilizado no projeto AMPLIA-I.

Em função do curto espaço de tempo para desenvolvimento, não é possível saber se o acoplamento do sistema conversor de imagens médicas ao projeto AMPLIA-I será possível, porém o desenvolvimento do sistema já utilizará o padrão do projeto para facilitar a incorporação do conversor à ferramenta AMPLIA-I.

REFERÊNCIAS BIBLIOGRÁFICAS

ADOBE SYSTEMS INCORPORATED. **Adobe Fireworks CS4**. Disponível em http://www.adobe.com/products/fireworks/. Acesso em: 12 nov. 2008.

AGOSTINI, Luciano Volcan. **Estudo de Padrões de Compressão de Imagens para Aplicações VLSI.** CPGCC-UFRGS, 1999. p.47-48

AGOSTINI, Luciano Volcan. BAMPI, Sergio. **Arquitetura Integrada para Conversor de Espaço de Cores e Downsampler para a Compressão de Imagens JPEG.** VII Workshop IBERCHIP, 2001. p.3.

AGOSTINI, Luciano Volcan. **Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG.** Porto Alegre: PPGC-UFRGS, 2002. p.97.

ALI, Hamza. NE'MA, Bashar. **Effective Variations on Opened GIF Format Images.** IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.5, May 2008.

BASSANI, Hanseclever de França. FREITAS, Alan Neiva. **SAPPI Sistema de Auxílio à Pesquisa em Processamento de Imagens.** Brasília, 2003.

CARITÁ, Edilson Carlos. MATOS, André Luiz Mendes. DE AZEVEDO-MARQUES, Paulo Mazzoncini. **Ferramentas para visualização de imagens médicas em hospital universitário.** Radiol Brás. 2004 Nov/Dec; vol 37 no.6.

CompuServe GIF 89a Specification. Disponível em http://www.w3.org/Graphics/GIF/spec-gif89a.txt. Acessado em 25 nov. 2008.

CONCI, Aura. AZEVEDO, Eduardo. LETA, Fabiana. **Computação Gráfica: Teoria e Prática.** Rio de Janeiro: Editora Elsevier, 2008.

DEUTSCH, Peter. **DEFLATE Compressed Data Format Specification version 1.3.** Maio, 1996.

FRIDRICH, Jessica. GOLJAN, Miroslav. CHEN, Qing. PATHAK, Vivek. **Lossless data embedding with file size preservation.** Proceedings of SPIE, 2004.

GONZALEZ, Rafael. WOODS, Richard. **Processamento de imagens digitais.** São Paulo, Editora Blucher, 2000.

IN, Jaehan. SHIRANI, Shahram. KOSSENTINI, Faouzi. **On RD Optimized Progressive Image Coding Using JPEG*.** University of British Columbia. Vancouver, Canada, 1999. p.

MIANO, John. **Compressed image file formats : JPEG, PNG, GIF, XBM, BMP.** Reading, MA : Addison-Wesley, 1999. p.23-28

NELSON, Mark. **LZW Data Compression.** Dr. Dobb's Journal, 1989.

NEMA, **Digital Imaging and Communications in Medicine (DICOM), Part 5: Data Structures and Encoding**. National Electrical Manufacturers Association. EUA, 1999.

openDICOM.NET. The DICOM Library Project. Disponível em http://opendicom.sourceforge.net/. Acessado em 13 nov. 2008.

SANTA-CRUZ, Diego. EBRAHIMI, Touradj. ASKELÖF, Joel. LARSSONB, Mathias. CHRISTOPOULOS, Charilaos. **JPEG 2000 still image coding versus other standards.** Applications of Digital Image Processing XXIII, volume 4115. San Diego, California, Jul. 2000.

SANTOS, Diego Ferreira dos. **Interface DICOM para captura e transmissão de imagens médicas.** Campinas, SP. 2003.

SCURI, Antonio. **Fundamentos da Imagem Digital.** Tecgraf / PUC-Rio, 2002.

SOARES, Marco. MARTINS, Pascoal. PEREIRA, Ricardo. COUTINHO, David. **Compressão de Dados com o Algoritmo Lempel-Ziv: Um caso Estudado.** DEEC, ISEL, Lisboa, 1999.

WALLACE, Gregory K. **The JPEG Still Picture Compression Standard.** IEEE Transactions on Consumer Electronics. Dez.1991. p.13

WEINBERGER, Marcelo. SEROUSSI, Gadiel. SAPIRO, Guillermo. **The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS.** Image Processing, IEEE Transactions on, 2000. p.1, 3.

ZIEBELL, Sebastian. Free Image Lib.NET. Disponível em http://sourceforge.net/projects/netimagelib/. Acesso em 13 nov. 2008.