CARLA CRISTINA FONSECA FERREIRA

UM ESTUDO SOBRE ARQUIVOS VETORIAIS PARA VISUALIZAÇÃO DE MAPAS NA *WEB*

DISSERTAÇÃO DE MESTRADO

TeCGraf

DEPARTAMENTO DE INFORMÁTICA

PUC-Rio

Rio de Janeiro, 11 de maio de 1998

CARLA CRISTINA FONSECA FERREIRA

UM ESTUDO SOBRE ARQUIVOS VETORIAIS PARA VISUALIZAÇÃO DE MAPAS NA *WEB*

Dissertação apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Mestre em Ciências em Informática.

Orientador: Marcelo Gattass

Co-orientador: Luiz Henrique de Figueiredo

TeCGraf/PUC-Rio

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 11 de maio de 1998

Agradecimentos

A minha família e a meus amigos pelo apoio, compreensão e paciência.

Ao orientador e professor Marcelo Gattass por todo conhecimento e apoio que me transmitiu.

Ao co-orientador Luiz Henrique de Figueiredo pela ajuda durante a construção deste trabalho.

Aos colegas do TeCGraf que de algum modo ajudaram na realização deste trabalho.

Ao CNPq pelo auxílio financeiro.

Este trabalho foi desenvolvido no TeCGraf, Grupo de Tecnologia em Computação Gráfica da PUC-Rio. O TeCGraf é suportado financeiramente através de projetos principalmente com a PETROBRAS/CENPES.

Resumo

Este trabalho apresenta um estudo sobre a transmissão de mapas na *World Wide Web*. Mapas são representações aproximadas da superfície terrestre, que projetam cada ponto do globo terrestre em uma superfície plana. A característica marcante das figuras que representam mapas é a presença de linhas poligonais com um grande número de pontos. Esses pontos, representados nos formatos vetoriais atuais, criam arquivos muito grandes para serem transmitidos pela *web*. Atualmente, a utilização de formatos de imagem na *web* é comum; porém as figuras geradas têm baixa resolução e não podem ser escaladas.

É apresentado um levantamento sobre os formatos de arquivos existentes, seguido por um estudo sobre algoritmos de compressão. Este trabalho propõe estratégias para reduzir o tamanho dos arquivos através da restrição da precisão dos pontos, da eliminação de pontos repetidos e da codificação eficiente das coordenadas. São mostrados exemplos reais para avaliar as estratégias propostas e são feitas recomendações baseadas nesses exemplos.

Abstract

This paper presents a study on the transmission of maps in the World Wide Web. Maps are approximate representations of the Earth surface in which points of the globe are projected on a plane. An important feature of figures that represent maps is that they have polygonal lines with a large number of points. With current vector formats these points generate archives that are too large to be transmitted through the web. The use of image formats, although common in the web today, only provide poor resolution figures that do not scale well.

A review of current metafile formats is presented, followed by a study on compression algorithms. The thesis proposes strategies to reduce the size of the archives by restricting the precision of the points, eliminating repeated points and performing an efficient coding of the coordinates. Real examples are shown to evaluate the proposed strategies and recommendations are made based on these examples.

Sumário

1. Introdução ____ *1*

1.1 Deficiência da Representação de Mapas por Imagens ____ **2**

1.2 Deficiência dos Formatos Vetoriais Atuais ____ **4**

1.3 Codificação Comprimida de Mapas ____ **4**

1.4 Trabalhos Correlatos ____ **7**

1.5 Objetivos da Dissertação ____ **9**

1.6 Organização da Dissertação ____ **9**

2. Formatos de Arquivos de Imagens e Figuras ____ *11*

3. Compressão ____ *16*

3.1 Huffman ____ **17**

3.2 LZW ____ **18**

3.3 LZO ____ **19**

4. TeCGraf Web Format ____ *22*

4.1 Quantização ____ **24**

4.2 Codificação Compacta das Coordenadas ____ **24**

4.3 Compressão ____ **37**

4.4 Descompressão e Decodificação ____ **38**

5. Casos Testes ____ *39*

6. Resultados ____ *43*

6.1 Quantização ____ **43**

6.2 Codificação Compacta ____ **44**

6.3 Compressão ____ **55**

7. Conclusão ____ *58*

A. Descrição dos Formatos de Arquivos e Figuras ____ *61*

A.1 Enhanced Metafile ____ **61**

A.2 Computer Graphics Metafile 61

A.3 CD Metafile 66

A.4 PLOT 68

A.5 Drawing Exchange Format 68

A.6 Drawing Web Format 70

A.7 Simple Vector Format 73

B. Descrição do TeCGraf Web Format *75*

C. Estimativa de Dimensão de Imagens *81*

Referências *82*

Lista de Figuras

Figura 1.1: Mapa da Cobertura Vegetal do Brasil ____ *2*
Figura 3.1: Construção da árvore de Huffman ____ *18*
Figura 4.1: Organização de um arquivo TWF ____ *22*
Figura 4.2: Etapas da criação e interpretação de um arquivo TWF ____ *23*
Figura 4.3: Organização de uma poligonal no TWF ____ *24*
Figura 4.4: Tipos de pontos na proposta 1 ____ *25*
Figura 4.5: Formato dos pontos absolutos ____ *27*
Figura 4.6: Formato dos pontos relativos por incremento ____ *29*
Figura 4.7: Formato da cadeia de bits ____ *31*
Figura 4.8: Direções ____ *31*
Figura 4.9: Exemplo de cadeia de bits ____ *33*
Figura 4.10: Número par de direções oeste consecutivas no meio da cadeia de bits ____ *34*
Figura 4.11: Número ímpar de direções oeste consecutivas no meio da cadeia de bits ____ *35*
Figura 4.12: Soluções para um número ímpar de direções oeste consecutivas no final da cadeia de bits ____ *36*
Figura 5.1: Mapa da África ____ *40*
Figura 5.2: Curvas de Nível de Madison ____ *40*
Figura 5.3: Curvas de Nível de Anchorage ____ *41*
Figura 5.4: Mapa de Municípios do Brasil ____ *41*
Figura 5.5: Mapa da Situação das Áreas Indígenas no Brasil ____ *42*
Figura A.1: Organização de um arquivo EMF ____ *61*
Figura A.2: Estrutura de um meta-arquivo ____ *62*
Figura A.3: Um meta-arquivo em diferentes níveis de detalhe ____ *62*
Figura A.4: Comando na forma pequena ____ *65*
Figura A.5: Comando na forma longa ____ *65*
Figura A.6: Sistemas de coordenadas do Canvas Draw ____ *67*
Figura A.7: Organização de um arquivo CD Metafile ____ *67*
Figura A.8: Organização de um arquivo DXF ____ *69*
Figura A.9: Organização de um arquivo DWF ____ *70*
Figura A.10: Bloco de dados do arquivo DWF ____ *70*
Figura A.11: Organização de um arquivo SVF ____ *73*
Figura C.1: Quantização das coordenadas em um grid uniforme ____ *81*

Lista de Tabelas

Tabela 1.1: Comparação entre os formatos CGM, DWF e PDF ____ *4*
Tabela 4.1: Valores para o header ____ *28*
Tabela 4.2: Valores para o footer ____ *28*
Tabela 4.3: Tamanho das coordenadas dos pontos absolutos ____ *28*
Tabela 4.4: Tamanho dos pontos absolutos ____ *29*
Tabela 4.5: Tamanho das coordenadas dos pontos relativos por incremento ____ *30*
Tabela 4.6: Tamanho dos pontos relativos por incremento ____ *30*
Tabela 4.7: Valores das direções na cadeia de bits ____ *32*
Tabela 5.1: Dados sobre os arquivos originais ____ *42*
Tabela 6.1: Quantização ____ *44*
Tabela 6.2: Comparação dos formatos na resolução 4096´4096 ____ *45*
Tabela 6.3: Comparação dos formatos na resolução 2048´2048 ____ *45*
Tabela 6.4: Comparação dos formatos na resolução 1024´1024 ____ *46*
Tabela 6.5: Comparação dos formatos na resolução 512´512 ____ *46*
Tabela 6.6: TWF Proposta 2 - Tipos de pontos no arquivo África ____ *51*
Tabela 6.7: TWF Proposta 2 - Tipos de pontos no arquivo Madison ____ *51*
Tabela 6.8: TWF Proposta 2 - Tipos de pontos no arquivo Anchorage ____ *51*
Tabela 6.9: TWF Proposta 2 - Tipos de pontos no arquivo de Municípios do Brasil ____ *52*
Tabela 6.10: TWF Proposta 2 - Tipos de pontos no arquivo de Vegetação do Brasil ____ *52*
Tabela 6.11: TWF Proposta 2 - Tipos de pontos no arquivo de Áreas Indígenas do Brasil ____ *52*
Tabela 6.12: TWF Proposta 2 - Número médio de bits por ponto na resolução 4096´4096 ____ *54*
Tabela 6.13: TWF Proposta 2- Número médio de bits por ponto na resolução 2048´2048 ____ *54*
Tabela 6.14: TWF Proposta 2 - Número médio de bits por ponto na resolução 1024´1024 ____ *55*
Tabela 6.15: TWF Proposta 2 - Número médio de bits por ponto na resolução 512´512 ____ *55*
Tabela 6.16: TWF Proposta 2 – Compressão por vários algoritmos na resolução 4096´4096 ____ *56*
Tabela 6.17: TWF Proposta 2 – Compressão por vários algoritmos na resolução 2048´2048 ____ *56*
Tabela 6.18: TWF Proposta 2 – Compressão por vários algoritmos na resolução 1024´1024 ____ *56*
Tabela 6.19: TWF Proposta 2 – Compressão por vários algoritmos na resolução 512´512 ____ *57*
Tabela C.1: Relação entre o tamanho das imagens e os diversos tipos de folha ____ *81*

Lista de Gráficos

Gráfico 6.1: Quantização 44
Gráfico 6.2: Comparação dos formatos no arquivo África 47
Gráfico 6.3: Comparação dos formatos TWF_P1, TWF_P2, DWF e GIF no arquivo África 48
Gráfico 6.4: África - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF 48
Gráfico 6.5: Madison - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF 49
Gráfico 6.6: Anchorage - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF 49
Gráfico 6.7: BR Municípios - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF 49
Gráfico 6.8: BR Vegetação - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF 50
Gráfico 6.9: BR Indígena - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF 50
Gráfico 6.10: TWF Proposta 2 - Pontos absolutos e relativos por incremento no arquivo África 53
Gráfico 6.11: TWF Proposta 2 - Pontos relativos por direção quantizada no arquivo África 53
*Gráfico 6.12: TWF Proposta 2 – Pontos eliminados (direções oeste consecutivas) do arquivo África.*53
Gráfico 6.13: TWF Proposta 2 - Número médio de bits por ponto 55
Gráfico 6.14: TWF Proposta 2 – Compressão por vários algoritmos do arquivo África 57
Gráfico 6.15: TWF Proposta 2 – Compressão por vários algoritmos do arquivo BR Vegetação 57

Lista de Siglas

ATA	*Airline Transport Association*
BMP	*Microsoft Windows Device Independent Bitmap*
CALS	*Computer-Aided Logistics Support*
CD	*Canvas Draw*
CDM	CD *Metafile*
CGM	*Computer Graphics Metafile*
DWF	*Drawing Web Format*
DXF	*Drawing Exchange Format*
EMF	*Enhanced Metafile*
GDI	*Graphics Device Interface*
GIF	*Compuserve Graphics Interchange Format*
GKS	*Graphical Kernel System*
LZO	*Lempel-Ziv-Oberhumer*
LZW	*Lempel-Ziv-Welch*
PDF	*Portable Document Format*
PHIGS	*Programmer's Hierarchical Interactive Graphics System*
PIP	*Petroleum Industry Profile*
SIG	Sistema de Informação Geográfica
SVF	*Simple Vector Format*
TeCGraf	Grupo de Tecnologia em Computação Gráfica da PUC-Rio
TWF	*TeCGraf Web Format*
VVS	*Virtual Visualization Surface*
W3C	*World Wide Web Consortium*
WMF	*Windows Metafile*

1. Introdução

Sistemas de Informação Geográfica, ou SIGs, são sistemas de informação construídos para armazenar, analisar e manipular dados geográficos, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente e indispensável para tratá-los. Dados geográficos são coletados a partir de diversas fontes e armazenados via de regra nos chamados *bancos de dados geográficos* [8].

SIGs têm ocupado um papel cada vez mais importante em diversas atividades humanas, como o controle do meio-ambiente, o gerenciamento de frotas e a difusão de informações turísticas. Em todas essas atividades, a Internet é um veículo fundamental para a divulgação dessas informações no âmbito municipal, federal ou mundial.

A visualização de mapas geográficos na *World Wide Web*, entretanto, não encontra um suporte adequado na atual tecnologia de arquivos. Mapas são representações aproximadas da superfície terrestre, que projetam cada ponto do globo terrestre em uma superfície plana. Na maioria dos servidores *web* que apresentam informações geográficas, os mapas são transmitidos como imagens, ou seja, há a transformação de suas primitivas gráficas do tipo linhas, textos e áreas preenchidas em uma matriz de *pixels*.

Normalmente, as imagens de mapas são transmitidas no formato GIF (*Compuserve Graphics Interchange Format*) [3]. Esse formato possui compressão e gera arquivos pequenos o suficiente para trafegar na rede sem muita demora, desde que o tamanho da imagem seja reduzido.

1.1 Deficiência da Representação de Mapas por Imagens

A apresentação de mapas como imagens de pequena resolução não é satisfatória. A representação impressa de todos os detalhes de um mapa pode facilmente requerer imagens de até 12000 pontos em cada direção (Apêndice C). Nesta resolução, é praticamente impossível gerar um arquivo GIF comprimido com os programas e equipamentos disponíveis no mercado. Na resolução 4096×4096, a imagem no formato GIF do mapa da Cobertura Vegetal no Brasil (Figura 1.1) possui 438 *Kbytes* de tamanho. Este tamanho torna inviável o tráfego na Internet atual para aplicações interativas. Na tela do computador, a resolução é de aproximadamente 75 *dpi*, logo não é necessário gerar uma imagem com esse tamanho apenas para visualização na *web*. Se as informações vetoriais originais do mapa forem preservadas, o arquivo pode ser menor e manter a mesma qualidade.

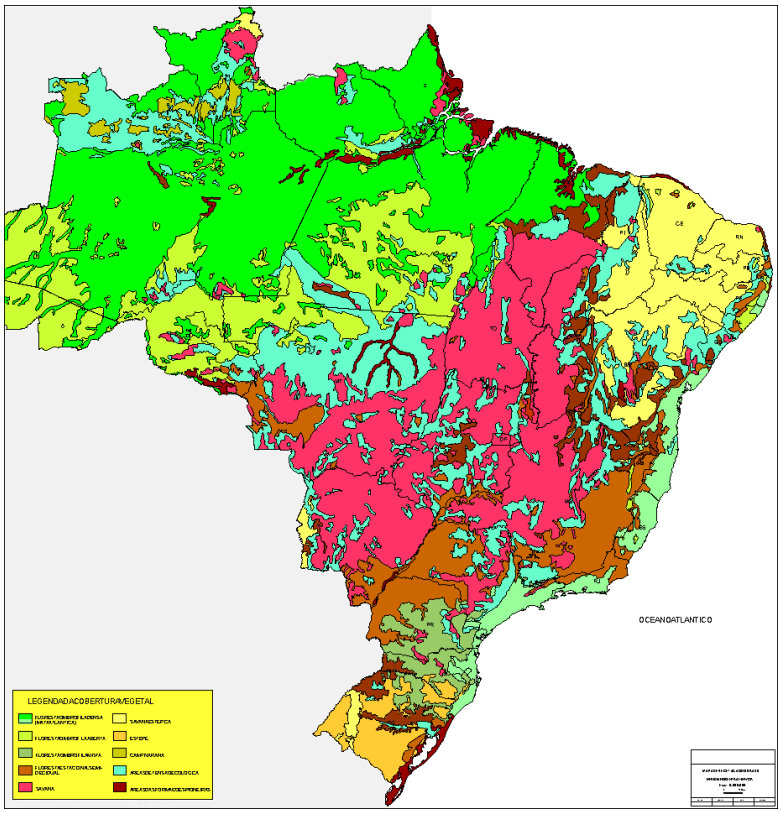


Figura 1.1: Mapa da Cobertura Vegetal do Brasil

Para contornar essa dificuldade, os sistemas atuais fornecem as imagens na resolução de uma parte de uma janela de um monitor, tipicamente menor que algo em

torno de 640×480. Nessa resolução, as imagens de mapas ficam com tamanhos variando tipicamente entre 5 e 30 *Kbytes.* Quando o usuário deseja ver algum detalhe, o sistema oferece uma facilidade de *zoom*. Porém, para obter-se um *zoom* razoável é necessário acessar novamente o servidor para buscar uma nova imagem com o mapa na nova escala. Alguns sistemas trabalham com determinadas escalas pré-definidas, enquanto outros regeram a imagem a partir da base de dados vetorial na escala especificada pelo usuário. Em ambos os casos, a interatividade do sistema pode ficar seriamente comprometida, pois a nova imagem deve ser reenviada através da rede, que geralmente tem baixa velocidade.

Outro problema importante decorrente da apresentação de mapas como imagens é a identificação de entidades. A extensão de uma entidade geofísica, como um rio ou uma estrada, pode ocupar uma parcela significativa da superfície de visualização. Por isto, as entidades são mais facilmente identificáveis através da seleção das primitivas gráficas que as compõem do que por áreas de imagens.

O W3C (*World Wide Web Consortium*) [45] é um consórcio fundado em outubro de 1994 com o propósito de direcionar a *web* para seu potencial completo através do desenvolvimento de protocolos comuns que promovam sua evolução e assegurem sua interoperabilidade. Trata-se de um consórcio industrial internacional, conjuntamente dirigido por: *Massachusetts Institute of Technology, Laboratory for Computer Science* (MIT/LCS) nos Estados Unidos; *Institut National de Recherche en Informatique et Automatique* (INRIA) na Europa; e *Keio University Shonan Fujisawa Campus* (SFC) no Japão. O W3C estabeleceu alguns requisitos [46] que gráficos que podem ser escalados devem atender para que sejam utilizáveis na *web*. Os principais requisitos podem ser divididos em dois grupos: facilidades gráficas e interação. As facilidades gráficas, entre outras coisas, definem que os gráficos devem ser vetoriais e devem possuir elementos curvos, texto, seleção de fonte de texto, modo *truecolor* (sem restrição a cores indexadas), transparência, controle de fim de linha, níveis de detalhe e dados *raster*. A interação inclui *zoom*, *pan*, seleção de elementos únicos, agrupamento de elementos em estruturas semânticas, *menus* ativados por seleção, *links* para outras vistas e outras figuras no mesmo arquivo, *links* para mídia externas (URL) e acesso através de mídia externas.

1.2 Deficiência dos Formatos Vetoriais Atuais

O tamanho dos arquivos das figuras armazenadas em formatos vetoriais depende fundamentalmente do número de primitivas gráficas neles contidas e da precisão do sistema de coordenadas utilizado. A representação vetorial de um mapa possui muitas linhas, muitos pontos e coordenadas descritas com muita precisão. A codificação de mapas realizada nos formatos atuais de arquivos de figuras, como o CGM (*Computer Graphics Metafile*) [5, 6], resulta em arquivos muito grandes, bem maiores que os respectivos GIFs.

Recentemente, outros formatos mais eficientes, como o DWF (*Drawing Web Format*) [28, 29, 37, 38] da *Autodesk* e o PDF (*Portable Document Format*) [30, 37] da *Adobe*, foram lançados, reduzindo o tamanho dos arquivos, sem contudo tirar proveito da natureza dos dados dos mapas. A menor coordenada armazenada, por exemplo, possui precisão de 8 *bits*, que ainda não é suficientemente pequena, como veremos.

A Figura 1.1 apresenta o mapa da Cobertura Vegetal no Brasil, que originalmente está no formato CGM. Na Tabela 1.1, há uma comparação entre o tamanho do arquivo original e os tamanhos dos respectivos arquivos nos formatos DWF e PDF, na resolução 32768×32768. Como pode ser observado, os arquivos deveriam ser menores, principalmente o DWF, para trafegarem na *web* sem muita demora.

Arquivo	CGM	DWF	PDF
BR Vegetação (Kb)	830	3135	482

Tabela 1.1: Comparação entre os formatos CGM, DWF e PDF

1.3 Codificação Comprimida de Mapas

Esta seção estuda os requisitos e propõe as etapas para a criação de arquivos vetoriais que sejam pequenos o suficiente para trafegar mapas na *web*. Em função do desafio de tentar reduzir significativamente o tamanho dos arquivos vetoriais contendo milhares de pontos, este trabalho adota a estratégia de eliminar quaisquer informações

desnecessárias e redundantes, sem que haja perda visual.

A primeira informação desnecessária é a precisão exagerada das coordenadas dos pontos, considerando que a figura é apenas para apresentação. Na maioria dos sistemas de informação, não se transmite a figura para que, sobre seus dados, sejam feitas análises numéricas precisas. Os requisitos de precisão nas coordenadas dos gráficos de apresentação adotados neste trabalho são os seguintes:

- as coordenadas devem suportar a exibição da figura em uma tela de monitor convencional, com um fator de *zoom* de 400%, sem erros de posicionamento dos vértices;
- fatores de *zoom* acima de 400% podem ou acarretar aproximações no posicionamento dos vértices ou solicitar ao servidor *web* novas coordenadas através do *plug-in* responsável pela tarefa;
- a figura deve poder ser impressa ou inserida em documentos, de modo que não haja perda de qualidade visual, diferentemente do que acontece com as imagens.

O número máximo de pontos endereçáveis nos monitores atuais geralmente não passa de 1280 na direção horizontal. Como nem todos os usuários da informação podem ter esse tipo de equipamento, é razoável hoje em dia supor que uma figura seja apresentada em janelas com menos de 1024 *pixels*. Com um fator de *zoom* de 400%, o número máximo de pontos endereçáveis não chegaria a 4096. Ao se imprimir em 300 *dpi* uma página *web* que contenha tais figuras, numa folha de papel geralmente com 11 polegadas de altura e 8 polegadas de largura, temos aproximadamente 3000 pontos endereçáveis. Logo, uma figura com resolução 4096×4096 possui um número de pontos endereçáveis suficiente para a sua apresentação com um fator de *zoom* de até 400% e para a sua impressão sem perda de informação.

Assim, a primeira etapa no processo de codificação com compressão de uma figura deve ser a imersão de seus pontos em um espaço de coordenadas inteiras de tamanho máximo M definido pela aplicação. Quanto menor for M, maior é a compressão e maior a perda de informação. Quanto maior for M, maior será o arquivo e menor será a perda de informação.

Quando os pontos de mapas são transformados para uma grade inteira que

varia de 0 até M, muitos pontos caem no mesmo lugar, gerando linhas de comprimento zero. A eliminação dessas linhas não afeta em nada o desenho do mapa e reduz substancialmente o tamanho do arquivo.

Uma descrição resumida de mapas na *web* pode ser feita adotando-se alguns conceitos oriundos dos primeiros sistemas gráficos, que foram feitos em tempos em que as restrições de memória e *bandwidth* entre o computador e o terminal gráfico eram bem maiores que as atuais. Nesses modelos, as figuras são compostas de objetos gráficos descritos por primitivas do tipo linhas, arcos, marcas, textos, imagens e poligonais, estas últimas podendo ser mostradas como linhas ou regiões preenchidas com uma determinada cor, textura ou padrão.

Os comandos que definem as primitivas contêm essencialmente as coordenadas dos vértices que as definem. A aparência das primitivas de um objeto é implícita em função da posição que essas primitivas ocupam na lista ou no arquivo. A aparência é definida por um conjunto de valores correntes de atributos que definem a cor, a espessura de linha, o tipo de preenchimento, o mapa de textura, etc. Esses valores começam com *defaults* implícitos e são redefinidos por comandos específicos, que mudam o valor corrente e afetam todas as primitivas que o seguem na lista, até a ocorrência de um novo comando do mesmo atributo. Desse modo, se vários objetos têm um mesmo estilo de linha e são contíguos na lista, então basta que esse estilo seja definido uma vez, antes do primeiro objeto.

A maneira em que as primitivas devem se sobrepor também é definida implicitamente. Os objetos desenhados primeiro podem ter partes obscurecidas pelos seguintes. O uso dessa propriedade pode ser importante para simplificar mapas. Considere, por exemplo, um desenho do mapa do Brasil com o objeto Mar aparecendo como uma região preenchida com a cor azul marinho. Nesse modelo, o Mar pode ser simplesmente representado por um retângulo desenhado no início da lista ou do arquivo. À medida que os estados e ilhas são desenhados, as praias vão se definindo. Em outras palavras, não é necessário descrever o polígono Mar pela costa, que é uma poligonal complicada, com muitos vértices. Uma função de seleção que respeite esta prioridade também pode ser implementada, fazendo testes de *pick* percorrendo os objetos da lista do fim para o começo (ou do começo para o fim, lembrando somente o último objeto selecionado).

Uma vez definido como os objetos gráficos são descritos, resta a questão fundamental de como eles devem ser codificados para o armazenamento e o transporte. Os comandos gráficos são codificados em símbolos que representam operadores e operandos, como, por exemplo, uma linha e suas coordenadas. Como este é o ponto mais delicado do processo, ele é descrito em detalhes no decorrer deste trabalho.

Mesmo que o processo de codificação dos comandos seja bem feito, com os símbolos mais freqüentes sendo representados por códigos menores, o arquivo resultante pode ainda ter redundâncias oriundas de repetições de padrões. Para reduzir mais ainda o arquivo, é importante que um algoritmo clássico de compressão de uso geral [4] seja utilizado.

É importante notar que o processo de codificação com compressão proposto neste trabalho não inclui a simplificação cartográfica dos dados, ou seja, supõe-se que os dados foram simplificados antes de serem submetidos ao processo em questão. Resumindo, o processo de codificação com compressão proposto possui três passos básicos:

1. Imersão da figura no espaço cartesiano dos inteiros positivos Z_+^2, escalando-se as coordenadas dos pontos para que seu valor mínimo seja zero e seu valor máximo M seja definido pela aplicação. Eliminação dos segmentos de tamanho zero resultantes desta tarefa.
2. Codificação das primitivas da figura em um formato compacto.
3. Compressão através de algoritmos clássicos de compressão sem perda.

A descompressão proposta possui apenas dois passos seqüenciais:

1. Descompressão.
2. Interpretação dos comandos.

1.4 Trabalhos Correlatos

O problema de reduzir o tamanho do arquivo que descreve um mapa, requer, necessariamente, uma investigação de como armazenar poligonais longas em um espaço pequeno e este é um foco importante deste trabalho. De certa maneira, esse

foco está relacionado com o assunto de *generalização* da área de SIGs.

Generalização é um assunto reconhecido como importante para a apresentação de mapas e consiste na criação de uma representação simplificada dos dados. Dettori e Puppo [8] procuram sintetizar e formalizar matematicamente as operações que transformam as representações de mapas em outras mais simples e eficazes para serem usadas na comunicação visual. Eles classificam os processos em:

- simplificação – eliminação de pontos que descrevem uma linha;
- seleção – eliminação de entidades;
- simbolização – representação através de símbolos, ao invés de desenhos;
- exagero – redução da precisão geométrica em benefício do significado;
- deslocamento – translação das entidades para evitar confusão;
- agregação – agrupamento de entidades que individualmente iriam desaparecer.

O algoritmo de Douglas-Peucker [10] é amplamente utilizado na simplificação de linhas. Os pontos resultantes dessa simplificação correspondem a um subconjunto dos pontos originais. Nesse algoritmo, os pontos inicial e final da linha poligonal são conectados por um segmento de reta. Depois, são calculadas as distâncias de todos os pontos contidos entre o ponto inicial e o ponto final da linha poligonal ao segmento de reta traçado. O ponto que apresenta a maior distância é identificado. Se a distância deste ponto ao segmento for menor que alguma tolerância pré-definida, então o segmento de reta é considerado adequado para representar a linha poligonal no modo simplificado. Caso contrário, o ponto com maior distância ao segmento de reta é selecionado e a linha poligonal é subdividida nesse ponto. O procedimento então se repete recursivamente para as duas partes da linha poligonal até que o critério de tolerância seja satisfeito. No final, os pontos selecionados são encadeados para formar a representação simplificada da linha poligonal original.

Na PUC-Rio, Mediano [12] abordou a utilização de V-trees, estruturas de multiresolução em disco, no armazenamento e recuperação de seqüências longas de pontos, em um dado espaço de coordenadas geo-referenciadas. A V-tree foi projetada para armazenar longas seqüências de pontos e permitir o acesso eficiente aos seus fragmentos. Essa estrutura otimiza o acesso a seqüências de pontos quando a consulta

envolve mudanças para escalas menores, pois permite gerar uma aproximação da seqüência de pontos que melhor se adapte à escala escolhida.

Também na PUC-Rio, Derraik [11] realizou um estudo comparativo de algumas representações em multiresolução para linhas poligonais. Foram estudadas as estruturas *strip tree*, *arc tree* e *box tree*, e suas variantes, comparando as velocidades de construção, de percorrimento (*drawing*), de operações de interseção e seleção (*pick*); e o custo de armazenagem em memória.

Na visão da linha de pesquisa na qual esta dissertação se insere, as idéias dos trabalhos citados acima são importantes para serem implementadas em um servidor *web* de mapas. O mapa generalizado (no sentido cartográfico) gerado pelo servidor, entretanto, ainda possui redundâncias que podem ser eliminadas e são o foco deste trabalho.

1.5 Objetivos da Dissertação

O objetivo deste trabalho é investigar métodos eficientes para representação de mapas vetoriais, de modo que os arquivos criados sejam suficientemente pequenos para trafegar rapidamente na Internet, permitindo a visualização vetorial no cliente. Para tanto, foram estudados os formatos de arquivos e os algoritmos de compressão existentes. Várias formas de armazenamento das coordenadas foram analisadas: coordenadas absolutas, coordenadas relativas por incremento e coordenadas relativas por direção quantizada. O objetivo desta análise é armazenar a informação sem perda, porém usando um mínimo de *bits*.

Para comparar os formatos existentes e testar as idéias propostas aqui, um conjunto de arquivos de dados reais é utilizado. Para uniformizar as comparações, os formatos de arquivos que não possuem compressão foram comprimidos através do método do dicionário, utilizando a biblioteca LZO [26].

1.6 Organização da Dissertação

A seguir é apresentado com mais detalhes o desenvolvimento deste trabalho. No capítulo 2, é feita uma breve explicação sobre os formatos de imagem e figura estudados, alguns dos quais foram utilizados nos testes. Uma análise mais detalhada de alguns desses formatos é feita no Apêndice A. O capítulo 3 trata de compressão, e são

apresentados os algoritmos Huffman [4, 13] e LZW [4, 17, 18, 19] e a biblioteca LZO. No capítulo 4, são apresentadas duas estratégias de armazenamento comprimido e uma proposta de um novo formato de arquivo, o TWF, *TeCGraf Web Format*, detalhado no Apêndice B. No capítulo 5, estão os casos testes, isto é, as imagens que foram utilizadas nos testes e os dados sobre as mesmas. O capítulo 6 mostra os resultados obtidos e o capítulo 7 apresenta as sugestões e conclusões para trabalhos futuros.

2. Formatos de Arquivos de Imagens e Figuras

Um arquivo de figuras é uma coleção de elementos. Estes elementos podem ser componentes geométricos da figura, como uma poligonal ou um polígono; também podem ser detalhes da aparência desses componentes, como a cor da linha; ou podem ser informações para o interpretador sobre como interpretar um determinado arquivo de figuras ou uma figura particular. As informações para o interpretador, normalmente, estão no início do arquivo de figuras e, a partir delas, o interpretador pode decidir se é capaz ou não realizar a interpretação. Exemplos de informações para o interpretador são o número da versão e uma lista dos elementos utilizados no arquivo de figuras. Caso o interpretador tenha sido criado para uma versão anterior à especificada no arquivo ou não consiga interpretar todos os elementos do mesmo, a interpretação é interrompida logo no início do arquivo.

Dentre o grande número de formatos de arquivos de figuras e imagens atualmente existentes, foram selecionados aqueles que são mais adequados para a representação de mapas na *web*. Como a limitação fundamental é a velocidade de transmissão na Internet, a principal característica destes arquivos discutida aqui é o tamanho dos mesmos. No caso de mapas vetoriais, este tamanho é geralmente associado ao grande número de vértices de poligonais; por isso, o armazenamento de seqüências de vértices é o foco central do estudo nestes formatos.

Os formatos considerados aqui como os mais relevantes para a transmissão de mapas podem ser divididos em dois grupos: os de imagem e os vetoriais. Os formatos de imagem estudados foram o GIF (*Compuserve Graphics Interchange Format*) [3] e o BMP (*Microsoft Windows Device Independent Bitmap*) [3]. O formato GIF é o mais utilizado na transmissão de mapas; o BMP foi escolhido porque, não sendo

comprimido, permite uma avaliação do tamanho da imagem sem compressão. O formato GIF suporta imagens com 1, 4 ou 8 *bits* por *pixel*. As imagens no formato BMP foram comprimidas *a posteriori* com algoritmos de compressão sem perda da biblioteca LZO [26], discutidos no capítulo 3.

O BMP, *Microsoft Windows Device Independent Bitmap* [3], suporta imagens com 1, 4, 8 ou 24 *bits* por *pixel*. Todas as imagens de 1, 4 e 8 *bits* por *pixel* possuem uma *palette* RGB associada. As imagens de 4 e 8 *bits* por *pixel* podem ser comprimidas com um algoritmo semelhante ao padrão RLE [4]. As imagens de 24 *bits* por *pixel* não são comprimidas e representam a cor RGB diretamente.

Os formatos de imagens não satisfazem os requisitos de capacidade de escala e identificação de objetos e só são apresentados aqui e nos resultados dos testes para efeito de comparação entre a proposta e as tecnologias atuais. Nos testes, as imagens GIF e BMP possuem 8 *bits* por *pixel*, isto é, no máximo 256 cores.

Os formatos vetoriais estudados podem ser divididos em dois grupos: os gerais e os feitos especificamente para a *web*. Os gerais estudados foram: EMF [22], CGM [5, 6], CDM [34], PLOT [23] e DXF [3, 27]. Os formatos vetoriais específicos para a *web* estudados foram: DWF [28, 29, 37, 38], *ActiveCGM* [36], SVF [31, 37, 38] e PDF [30, 37].

Como não foi possível obter informações detalhadas dos formatos proprietários *ActiveCGM* e PDF, eles são apresentados de forma sucinta a seguir. Os demais formatos serão estudados com mais detalhes e possuem uma apresentação mais completa no Apêndice A.

O *ActiveCGM* [36] da *Intercap* é uma família de perfis do *International Standard for Computer Graphics Metafiles* (ISO/IEC 8632: 1992, normalmente chamado de padrão CGM). A família de perfis *ActiveCGM* define uma metodologia para adicionar *hyperlinks*, animação e outras utilidades aos gráficos padrão CGM. A intenção é aumentar o extenso suporte existente para o padrão CGM, definindo como ativar gráficos CGM de modo que eles possam ser facilmente divulgados na *web*.

O PDF (*Portable Document Format*) [30, 37] da *Adobe* tem como objetivo trazer o *layout* da página para a *web*. A revolução de *Desktop Publishing* começou com o *Macintosh* (*quick draw*) e o *PostScript* (exibição WYSIWYG – *what you see is*

what you get). Isto permitiu ver na tela do computador exatamente o que será visto na página impressa. O PDF estende esse procedimento para a *web*. Essencialmente, o PDF é um *PostScript* com regras restritas, como o modo como é formatado e um *header* e um *footer* extras. O PDF é um formato binário e ainda se baseia nas entidades básicas presentes nos arquivos *PostScript*: texto, linhas e *splines* de Bézier. Há um controle preciso de como esses elementos são definidos, mas não há elementos de nível mais alto como arcos ou círculos (estes são criados através de Bézier). Desse modo, o interpretador pode se concentrar em um conjunto de entidades muito pequeno. Por outro lado, isso também significa que arcos e círculos podem ter um comportamento bastante ruim. O *Adobe Acrobat* oferece um método para criar arquivos PDF comprimidos, porém nenhum outro *software* parece ser capaz de criar ou ler esses arquivos.

O formato EMF (*Enhanced Metafile*) [22] é utilizado nos sistemas *Windows 32 bits* para armazenar primitivas gráficas. É vetorial, binário e as coordenadas das poligonais são salvas com 32 *bits*, sendo que há a opção de convertê-las para 16 *bits*, a fim de diminuir o tamanho final do arquivo. O formato suporta textos e primitivas geométricas, como arcos, círculos e polígonos. As cores estão contidas em uma *palette*, logo não é possível fazer uma codificação RGB das mesmas. A ordem das primitivas é definida implicitamente, ou seja, elas são desenhadas na mesma ordem em que aparecem no arquivo. Os atributos são correntes, isto é, não são associados a cada primitiva, e não há compressão no formato. O EMF não foi desenvolvido voltado para *web*, logo não satisfaz os requisitos de interação [46].

O padrão ANSI CGM (*Computer Graphics Metafile*) [5, 6] é vetorial e possui várias primitivas geométricas (linhas, polígonos, arcos circulares, arcos cônicos, curvas de Bézier, etc.), texto, fontes e atributos de texto, e primitivas *raster*. O conceito de atributo corrente também se aplica a um CGM, bem como a ordem implícita das primitivas. O CGM não possui compressão e também não satisfaz os requisitos de interação [46], pois não foi desenvolvido para a *web*. O *ActiveCGM*, apresentado anteriormente, é um perfil do CGM para ser utilizado na *web*.

O CGM possui três formas diferentes de codificação: a por caracter, a binária e a *clear text*. Os principais objetivos da codificação por caracter são assegurar que a codificação seja compacta e garantir uma transferência fácil de dados através das

redes. A ênfase da codificação binária é a fácil criação e interpretação dos arquivos CGM. Na codificação *clear text*, os dados armazenados são palavras legíveis, o que permite a edição dos arquivos, enquanto que, na codificação por caracter, apenas um caracter pode ter um significado específico. Nos testes, foram utilizados arquivos CGM binários.

Na codificação binária, as cores do CGM podem ser indexadas ou diretas. Quando são indexadas, as cores são representadas com um inteiro sem sinal, de 8, 16, 24 ou 32 *bits*, sendo que o padrão é 8 *bits*. Quando as cores são diretas, podem ser codificadas como três ou quatro inteiros sem sinal, dependendo do modelo de cores. As precisões válidas são de 8, 16, 24 ou 32 *bits*, sendo que o padrão também é 8 *bits*.

Os dados, na codificação binária do CGM, podem ser representados por inteiros de 8, 16, 24 e 32 *bits*, por reais de ponto fixo de 32 ou 64 *bits* ou por números reais de ponto flutuante de 32 ou 64 *bits*. O valor padrão para parâmetros inteiros é 16 *bits* e para números reais é ponto fixo de 32 *bits*. Nos testes, foram utilizadas coordenadas inteiras de 32 *bits*.

O formato CDM (CD *Metafile*) [34] é criado utilizando-se a biblioteca gráfica bidimensional CD – *Canvas Draw* – [33], desenvolvida pelo TeCGraf/PUC-Rio [32]. Esse formato possui o conceito de atributo corrente e a ordem das primitivas é implícita, porém não apresenta compressão. A codificação é textual e suas coordenadas são vetoriais e inteiras de 32 *bits*. O CDM possui, entre outras, as primitivas de linha, arco, texto, setor, marca e imagem *raster*. É possível selecionar atributos como tipo de fonte de texto, tamanhos de texto e de marca, etc. As cores podem ser indexadas em uma *palette* ou RGB. O CDM também não foi desenvolvido para *web*.

O formato PLOT [23] possui coordenadas cartesianas, suas informações são vetoriais e a codificação é binária. A ordem implícita é utilizada no desenho das primitivas e cada ponto ocupa 4 *bytes*, sendo as coordenadas números inteiros com sinal. O PLOT não possui compressão e não foi desenvolvido para ser utilizado na *web*. Algumas primitivas que esse formato representa são o arco e o círculo.

O formato DXF (*Drawing Exchange Format*) [3, 27] da *Autodesk* é uma descrição textual de um desenho. Os dados em um arquivo DXF são vetoriais e a

precisão dos números reais armazenados pode ser definida pelo usuário, sendo que são permitidas no máximo 16 casas decimais. É possível ter no máximo 256 cores indexadas e não há compressão dos dados. O DXF possui textos e as demais primitivas geométricas (linhas, polígonos, etc.). As primitivas podem ser agrupadas em blocos, porém a ordem de desenho continua sendo implícita. O conceito de atributo corrente também se aplica a esse formato. O DXF não foi desenvolvido para a *web*, mas a *Autodesk* desenvolveu o DWF, que tem como objetivo o tráfego na rede.

O DWF (*Drawing Web Format*) [28, 29, 37, 38] é proposto como padrão pela *Autodesk* para visualizar os desenhos do *AutoCAD* (DXF, DWG) na *web*. A maior parte de um arquivo DWF é composta por pares operadores/operandos, que podem ser de dois tipos: texto legível e código binário. Normalmente, o arquivo possui uma mistura dos dois tipos de pares operador/operando.

O DWF é vetorial e suas coordenadas podem ser absolutas (31 *bits* sem sinal) ou relativas (32 *bits* com sinal). Uma coordenada relativa é formada subtraindo-se a coordenada atual da coordenada absoluta anterior. A compressão utilizada no DWF baseia-se no reconhecimento de padrões repetidos. Muitas operações de desenho permitem que sejam utilizadas coordenadas relativas inteiras de 16 *bits* (valores relativos de 16 *bits* com sinal), para minimizar o tamanho final do arquivo. O DWF suporta, por exemplo, linhas, poligonais, polígonos, símbolos, imagens, círculos, arcos, elipses, textos e transparência variável. O conceito de atributo corrente e a ordem implícita das primitivas também são válidos para esse formato. Como foi desenvolvido para a *web*, o DWF permite que sejam feitos *zoom* e *pan* e que sejam definidos *links*.

O SVF (*Simple Vector Format*) [31, 37, 38] foi criado pela *SoftSource* e pela NCSA em 1994 para a utilização na *web*, logo possui *zoom* e *hyperlinks*, sendo que estes podem ser ativados ou desativados baseando-se na camada onde estão. As camadas possibilitam agrupar objetos, de modo a mostrar ou não partes do desenho. O SVF possui primitivas básicas como: pontos, linhas, círculos, arcos, curvas de Bézier e textos. Os parâmetros, como cores e coordenadas, são inteiros e os tamanhos variam de 1 a 8 *bytes*. As poligonais também podem possuir coordenadas relativas. Há uma tabela de cores padrão com 256 cores, mas também podem ser definidas cores específicas em RGB. O SVF não possui compressão, mas apresenta o conceito de atributo corrente.

3. Compressão

A compressão de dados compreende duas etapas distintas: modelagem e codificação [4]. O modelo é simplesmente uma coleção de dados e regras utilizada para processar os símbolos de entrada e determinar os códigos de saída. Um programa utiliza o modelo para definir as probabilidades de cada símbolo e o codificador para produzir um código apropriado baseado nessas probabilidades.

Na codificação, os símbolos têm que ser codificados exatamente com o número de *bits* de informação que contêm. Duas boas aproximações diferentes para este problema são a codificação de Shannon-Fano [4] e a codificação de Huffman [4, 13]: ambas geram códigos de tamanho variável a partir da tabela de probabilidades de um conjunto de símbolos e utilizam um número inteiro de *bits* em cada código.

Um provável sucessor da codificação de Huffman é a codificação aritmética [4], mais complicada em conceito e em implementação. A codificação aritmética não produz um único código para cada símbolo, mas um código para uma mensagem inteira. Cada símbolo adicionado à mensagem modifica incrementalmente o código de saída. Desse modo, o código de saída pode ter um número fracionário de *bits*.

A compressão de dados sem perda é geralmente implementada utilizando ou modelagem estatística ou modelagem baseada em dicionário. A modelagem estatística lê e codifica um único símbolo de cada vez, utilizando a probabilidade do caracter aparecer. A modelagem baseada em dicionário usa um único código para substituir um conjunto de caracteres. Nela, o problema de codificação tem sua importância reduzida, aumentando bastante a importância do modelo. Quando é utilizado um dicionário estático, este tem que ser colocado no arquivo comprimido, aumentando seu tamanho.

Exemplos de métodos de compressão que utilizam a modelagem baseada em dicionário são LZW [4, 17, 18, 19] e LZO [26].

Como foi mencionado, o formato GIF comprime suas imagens utilizando o algoritmo LZW. Além desse algoritmo, foram estudados com mais detalhes os algoritmos de Huffman estático e dinâmico e a biblioteca LZO.

3.1 Huffman

O método de codificação do algoritmo de Huffman [4, 13] baseia-se na árvore de Huffman. Os símbolos a serem codificados estão associados às folhas da árvore e o seu código é obtido pelo caminho que deve ser percorrido desde a raiz para chegar até o símbolo procurado. A partir de cada nó interno percorrido no caminho, se seguirmos pela esquerda, o *bit* de codificação deve ser 0; se seguirmos pela direita, o *bit* deve ser 1.

O algoritmo para construção da árvore de Huffman (Figura 3.1) baseia-se na freqüência de ocorrência dos símbolos. Esse algoritmo é executado em três passos:

(a) Colocar os *n* pesos associados aos nós terminais em ordem ascendente.

(b) Tirar os dois nós de menor valor da seqüência, somá-los e introduzir esta soma novamente na seqüência de forma a manter a ordem ascendente. A soma será associada a um nó interno do qual os dois que foram retirados vão passar a depender.

(c) Repetir o segundo passo até que exista apenas um valor na seqüência. Este valor será associado à raiz.

Uma vez terminada a construção da árvore de Huffman, cada símbolo está associado a uma folha e cada nó interno tem um peso que é a soma dos pesos de seus dependentes. Para estabelecer a codificação de cada símbolo, segue-se o caminho da raiz até a folha. Da mesma forma, na decodificação, também parte-se da raiz da árvore e segue-se até atingir uma folha. A compressão é realizada em dois passos: primeiro é feita uma análise dos dados a serem comprimidos e um levantamento da freqüência de ocorrência dos mesmos; depois de construída a árvore, ela é utilizada para codificar os caracteres contidos no arquivo. Para permitir a decodificação, a árvore que foi

utilizada é colocada no arquivo comprimido, antes dos códigos começarem a ser gerados. Na decodificação, primeiro lê-se a árvore e depois decodificam-se os dados.

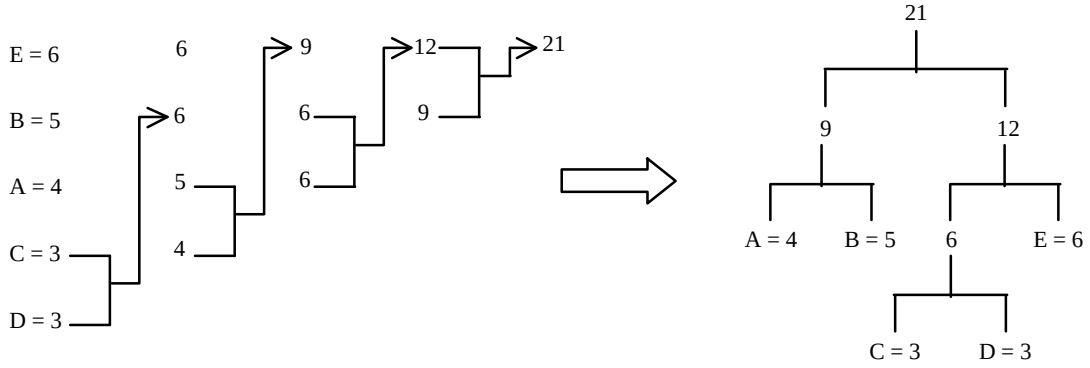


Figura 3.1: Construção da árvore de Huffman

O método de codificação do algoritmo FGK (*Faller, Gallager, Knuth*) [14, 15, 16] baseia-se na seguinte propriedade da árvore de Huffman: os nós se apresentam nível a nível, da esquerda para a direita, com seus pesos em ordem ascendente. A compressão dos dados é realizada através da atualização dinâmica da árvore. O algoritmo codifica um símbolo de forma análoga ao método de Huffman estático, mas, assim que o símbolo é codificado, o algoritmo retorna pelo caminho percorrido na árvore, atualizando o peso de todos os nós encontrados no caminho de volta até a raiz. Inicialmente, o algoritmo começava com uma árvore balanceada [14, 15], depois passou por uma melhora e a árvore inicial passou a ser nula [16].

3.2 LZW

O método de codificação do algoritmo LZW (*Lempel-Ziv-Welch*) [4, 17, 18, 19] não necessita conter uma tabela de codificação e é adaptativo, isto é, a codificação é construída e modificada dinamicamente a partir da própria informação de entrada, otimizando a compressão para vários tipos de informação. O algoritmo baseia-se em uma tabela de cadeias, na qual coletam-se cadeias de caracteres encontradas no arquivo de entrada (que contém a informação original, descomprimida), com a esperança de que essas cadeias ocorram novamente mais adiante. Quanto mais vezes as cadeias contidas na tabela ocorrerem no arquivo de entrada, maior será a redundância e melhor a taxa de compressão.

A tabela é iniciada com todas as cadeias de 1 caracter, ou seja, supondo que os caracteres sejam de 8 *bits*, ela começa já com 256 cadeias (0..255). Para cada cadeia nova encontrada na entrada (e inserida na tabela), é atribuído um código novo. Os códigos novos começam a partir de 257, pois o código 256 é reservado como um marcador. Nota-se que 257 necessita 9 *bits* e à medida que aumenta o número de códigos aumenta o número de *bits* na codificação.

Na compressão, a chave de busca na tabela é uma cadeia e a informação procurada é o seu código associado; na descompressão é exatamente o inverso: a chave de busca é um código e a informação procurada é a cadeia associada. Na compressão e na descompressão são utilizadas tabelas com as mesmas informações, porém a diferença da natureza da chave de busca exige que a estrutura de dados da tabela seja diferente nas duas fases. A fase de descompressão é a mais simples, pois pode ser utilizada uma tabela de acesso direto, onde o código da cadeia é usado como índice da entrada. Na fase de compressão, no entanto, a chave de busca é a própria cadeia e a melhor solução é usar uma tabela *hash*. Para encontrar a entrada correspondente à cadeia wK, por exemplo, calcula-se o *hash* (algum cálculo aritmético com os valores de w e K) e obtém-se a entrada wK. Para evitar o problema de incluir na tabela uma cadeia que acabou de ser inserida (repetição contínua, KwKwK, por exemplo), sempre compara-se o código atual com o código anterior processado para não ocorrer repetição. O LZW é o mais utilizado entre os algoritmos citados.

3.3 LZO

Na compressão dos arquivos criados pelo formato proposto neste trabalho foi utilizada a biblioteca LZO (*Lempel-Ziv-Oberhumer*) versão 1.00 [26]. Os arquivos devem ser transmitidos pela rede e a LZO é uma biblioteca de compressão e descompressão de dados em tempo real, logo tem como objetivo a velocidade. Além disso, de acordo com os testes realizados, ela apresenta resultados tão bons quanto o algoritmo LZW utilizado pelo formato GIF, mesmo porque ambos se baseiam na modelagem do dicionário. A LZO é uma biblioteca de domínio público e possui referência na rede, enquanto o LZW corresponde à patente 4,558,302 da Unisys [48]. Durante os testes, a LZO não apresentou problemas. Outro fator importante para a

escolha da LZO é sua portabilidade entre as plataformas e o fato de não utilizar muita memória. Essa biblioteca também dispõe de vários algoritmos, explicados mais adiante.

Na LZO, a velocidade do descompressor é favorecida em relação à velocidade do compressor. O código fonte é escrito em ANSI C e a biblioteca funciona em arquiteturas que possuam no mínimo inteiros de 32 *bits*, porém o suporte para os sistemas de 16 *bits* mais utilizados (DOS, *Windows* 3.11) também é oferecido.

A LZO utiliza a modelagem baseada em dicionário. Nesta modelagem, são feitos índices com as expressões mais utilizadas, ou seja, cada expressão obtém um número de referência. Nos locais onde a expressão aparece, é colocada apenas uma referência à mesma. Desse modo, quanto mais expressões repetidas possuir um arquivo, mais índices existirão, mais referências poderão ser feitas e melhor será a compressão.

Esta versão da LZO inclui oito algoritmos: LZO1, LZO1A, LZO1B, LZO1C, LZO1F, LZO1X, LZO1Y e LZO2A. Primeiro, o LZO1 e o LZO1A foram colocados na rede, em março de 1996. Estes continuam incluídos na biblioteca principalmente por razões de compatibilidade. O descompressor LZO2A é bastante lento, logo não há um compressor rápido. Os compressores de interesse são: LZO1B, LZO1C, LZO1F, LZO1X e LZO1Y. Cada um deles possui uma característica ligeiramente diferente e a princípio o algoritmo LZO1X é uma boa escolha para uma grande variedade de dados.

A convenção adotada para os nomes dos algoritmos na LZO é LZOxx_N, onde xx indica qual é o algoritmo e N é o nível de compressão. Por exemplo, o nome de algoritmo LZO1X_1, indica que se trata do algoritmo 1X da biblioteca LZO com nível de compressão 1. O intervalo de 1 a 9 indica os níveis de compressão padrão mais rápidos, utilizando 64 *Kbytes* de memória para compressão. O nível 99 oferece melhor compressão e é razoavelmente rápido, porém utiliza mais memória (256 *Kbytes*). O nível 999 oferece uma compressão ótima, contudo é lento e utiliza muita memória. Este último nível é recomendado para a geração de dados pré-comprimidos.

Outra biblioteca pública de compressão de dados sem perda é a *zlib* [47]. O formato de dados da *zlib* é portável entre as plataformas e, ao contrário do LZW, o método de compressão utilizado pela *zlib* nunca expande os dados. A memória

utilizada por ela é independente dos dados de entrada e pode ser reduzida, se necessário, mediante algum custo na compressão.

O LZO1X é aproximadamente de 4 a 5 vezes mais rápido que o mais rápido nível de compressão da biblioteca *zlib* [47], tanto na compressão quanto na descompressão.

4. TeCGraf Web Format

O ponto de partida para criar uma base de testes para as idéias expostas anteriormente foi fazer mudanças no CDM (CD *Metafile*) [34]. A biblioteca CD [33] foi escolhida como referência porque o seu formato possui características desejáveis, como o conceito de atributos correntes, coordenadas vetoriais, texto, escolha da fonte de texto, dados *raster* e cores RGB. Além disso, a CD já possui grande parte do código necessário para os testes pronto. O novo formato de arquivo resultante dessas modificações é chamado aqui de TWF, *TeCGraf Web Format*, na esperança de que ele venha a ser a base de um produto TeCGraf/PUC-Rio [32] de figuras na *web*.

Como a preocupação maior deste trabalho é o tamanho do arquivo, a primeira modificação importante foi trocar a codificação de *clear text* para arquivos essencialmente binários. A Figura 4.1 apresenta o esquema de um arquivo TWF, onde *Id* é o identificador do formato, *w* e *h* são os valores máximos de largura e altura, respectivamente, e *Função* corresponde a uma função do CD. Cada função possui um *Índice* (operador) para identificá-la e algumas funções possuem seus *Parâmetros* (operandos). O Apêndice B apresenta as funções do TWF.

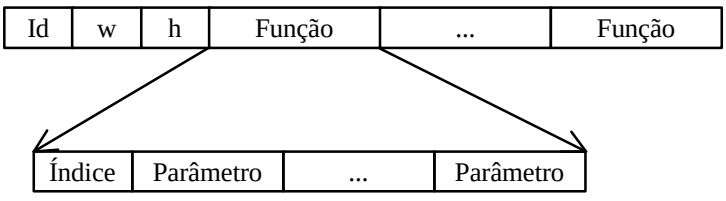


Figura 4.1: Organização de um arquivo TWF

A largura e a altura são números inteiros de 16 *bits* (*short*) e os índices ocupam um *byte* (*unsigned char*). Cada função possui quantidades e tipos de dados diferentes.

A grande mudança em relação ao formato CDM é o modo de codificação dos vértices de uma poligonal, ou melhor, das coordenadas do mapa. Dado esse aspecto central do tema, este trabalho estuda duas propostas diferentes de codificação compacta de poligonais. Então, as etapas de codificação e decodificação são divididas em proposta 1 e proposta 2. As demais etapas (quantização, compressão e descompressão) são sempre iguais, independente da proposta. Deve ficar claro que ou se utiliza a proposta 1 ou se utiliza a proposta 2 para a criação e interpretação de um TWF, isto é, ou há um TWF_P1 ou um TWF_P2. A Figura 4.2 mostra as etapas de criação e interpretação de um arquivo no formato TWF, sendo que ou a linha pontilhada ou a linha tracejada podem ser seguidas para a criação desse arquivo.

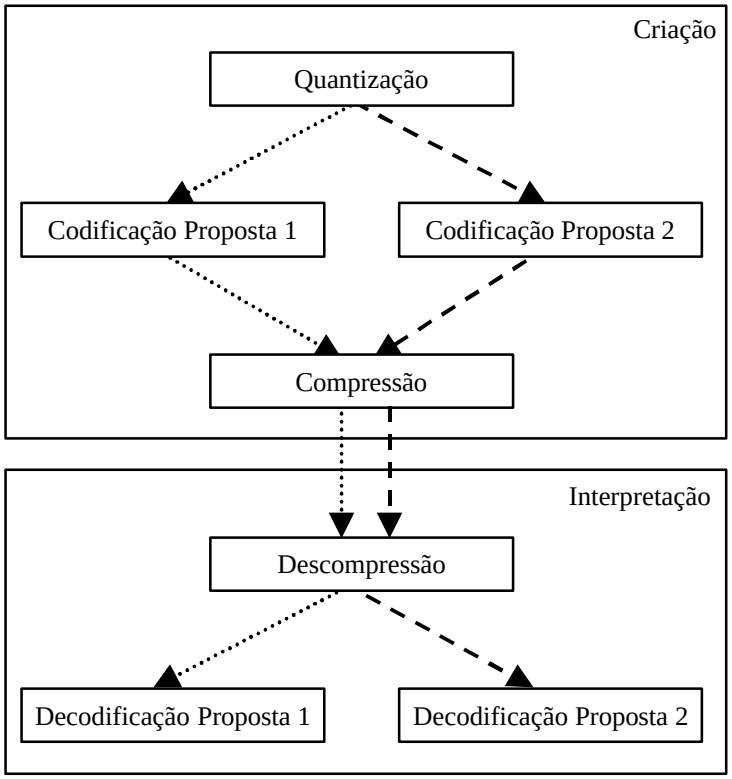


Figura 4.2: Etapas da criação e interpretação de um arquivo TWF

4.1 Quantização

Como já foi discutido, as coordenadas dos mapas são geralmente definidas com precisão excessiva para apresentação, tendo como conseqüência um número de pontos bem maior que o número máximo de pontos endereçáveis nos monitores. A etapa de quantização tem a função de reduzir a precisão das coordenadas e o número de pontos que descrevem as poligonais.

A idéia básica é converter as coordenadas originais do mapa para o espaço cartesiano dos inteiros positivos Z_+^2, onde o valor máximo de uma coordenada é igual à resolução da imagem em *pixels*. Assim, o maior valor admitido para uma coordenada *x* é igual à largura da imagem e o maior valor admitido para uma coordenada *y* é igual à altura da imagem.

Nessa conversão, muitos pontos do mapa podem corresponder ao mesmo ponto no sistema inteiro. Quando uma poligonal possui pontos consecutivos iguais no sistema de coordenadas inteiras, apenas um ponto é considerado, eliminando-se segmentos de tamanho zero.

4.2 Codificação Compacta das Coordenadas

A codificação compacta consiste em armazenar as poligonais do mapa da forma mais compacta possível. Essa codificação pode ser realizada segundo duas propostas diferentes a serem apresentadas, sendo que a diferença entre as propostas está no modo de armazenamento dos pontos que compõem a poligonal. Dessa forma, a organização da poligonal é a mesma tanto para a proposta 1 quanto para a proposta 2 (Figura 4.3).

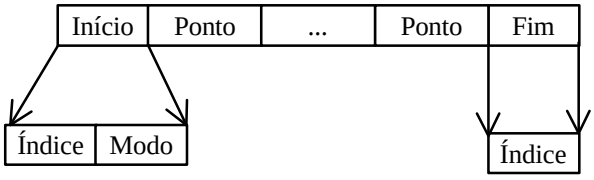


Figura 4.3: Organização de uma poligonal no TWF

Cada poligonal possui uma função que indica o seu início e outra função que indica o seu término. A função da biblioteca CD que indica o início de uma poligonal é a *cdBegin*, que é composta por um índice e pelo modo da poligonal. Como ocorre no CD *Metafile*, o modo da poligonal pode ser: CD_CLOSED_LINES, CD_OPEN_LINES, CD_FILL ou CD_CLIP. O modo CD_OPEN_LINES cria uma poligonal aberta, enquanto o modo sCD_CLOSED_LINES cria uma poligonal fechada, conectando seu último ponto ao primeiro. O CD_FILL também cria uma poligonal fechada e a preenche com o estilo interior corrente. O CD_CLIP cria um polígono para definir uma região de *clipping* não retangular. A marcação de fim de poligonal é formada apenas pelo índice correspondente à função *cdEnd*.

Os índices da marcação de início e da marcação de fim da poligonal ocupam 1 *byte* (*unsigned char*) e o modo da poligonal ocupa 2 *bytes* (*short)*. As duas propostas de codificação dos vértices da poligonal são vistas a seguir.

4.2.1 Proposta 1 – Codificação Simples

Nesta proposta, os vértices da poligonal são codificados utilizando-se coordenadas relativas por incremento. Desse modo, obtém-se dois tipos de ponto: absoluto e relativo.

O ponto absoluto ocupa 4 *bytes* e o relativo 2 *bytes*. Em ambos os casos, os dois *bits* mais significativos indicam que tipo de ponto está sendo analisado e os outros *bits* são divididos entre a coordenada *x* e a coordenada *y*. Em um ponto absoluto, os dois *bits* mais significativos têm valor 10 e, em um ponto relativo, têm valor 11. Assim, cada coordenada de um ponto absoluto possui 15 *bits* e cada coordenada de um ponto relativo possui 7 *bits* (Figura 4.4).

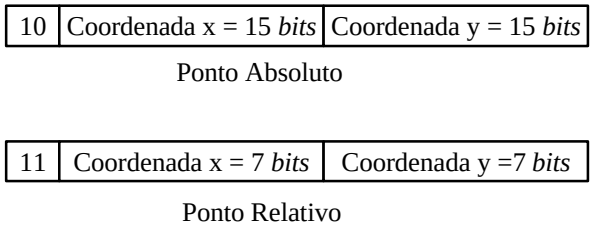


Figura 4.4: Tipos de pontos na proposta 1

O primeiro ponto de uma poligonal é sempre absoluto; os demais podem ser de qualquer tipo. Se for possível armazenar as coordenadas do ponto corrente utilizando-se 7 *bits*, então ele será relativo; senão, será absoluto. As coordenadas de um ponto relativo sempre são calculadas levando-se em consideração as coordenadas do ponto anterior no sistema de coordenadas inteiro da quantização, independentemente do ponto anterior ser absoluto ou relativo. Dessa forma, dentro de uma poligonal existe o conceito de posição corrente, que corresponde à posição do ponto corrente no sistema inteiro de coordenadas utilizado na quantização. A posição corrente começa com o primeiro ponto e termina no último ponto de cada poligonal.

As coordenadas de um ponto absoluto nunca são negativas, pois na quantização garante-se que as coordenadas são inteiras e positivas. Assim, a maior resolução permitida por esta codificação é de $2^{15} \times 2^{15}$, sendo $2^{15}-1=32767$ a maior coordenada possível de ser armazenada em um ponto absoluto. As coordenadas de um ponto relativo podem ser negativas; logo seu intervalo de valores válidos é de -2^6 a 2^6-1, ou seja, de –64 a +63.

Para se ter uma idéia, a distância máxima entre dois pontos da Terra, ou seja, o semiperímetro do Equador, é de aproximadamente 20037 km. Logo, na resolução máxima, é possível armazenar o semiperímetro do Equador em quilômetros. Desse modo, a menor unidade que se consegue representar é 1 km, ou seja, o maior erro admitido é igual a 1 km. Supondo que o erro máximo em um mapa seja igual a 0.1 mm na escala da carta, a maior escala que se consegue representar utilizando a codificação apresentada é $1:10^7$.

4.2.2 Proposta 2 – Codificação Adaptativa

Nesta proposta, os pontos da poligonal podem possuir coordenadas de três tipos: absolutas, relativas por incremento e relativas por direção quantizada. Os pontos com coordenadas absolutas podem aparecer isolados na poligonal, isto é, os pontos anterior e posterior ao ponto absoluto corrente podem ou não ser pontos absolutos. Da mesma forma, um ponto relativo por incremento pode aparecer isolado na poligonal. Porém, os pontos relativos por direção quantizada formam uma cadeia, a chamada *cadeia de pixels*. Esses pontos são codificados em conjunto, sendo que cada

conjunto, ou melhor, cada cadeia de *pixels* possui pelo menos dois pontos com coordenadas relativas por direção quantizada.

O primeiro ponto de cada poligonal é sempre um ponto absoluto, assim o conceito de posição corrente fica restrito a cada poligonal. Desta forma, o conceito de posição corrente não precisa ser definido para outros objetos gráficos, como textos e imagens. Apesar de gerar uma proposta mais elaborada, a estratégia é simples: todos os pontos são sempre armazenados no menor tipo possível, a fim de minimizar o tamanho final do arquivo criado.

4.2.2.1 Coordenadas Absolutas

Nas coordenadas absolutas, os dados são armazenados segundo o resultado da etapa de quantização, isto é, com os valores do sistema de coordenadas inteiro utilizado na quantização. Um ponto com coordenadas absolutas é composto por um *header*, pelas coordenadas *x* e *y* e pelo *footer* (Figura 4.5).

header	coordenada x	coordenada y	*footer*

Figura 4.5: Formato dos pontos absolutos

Há mais de um tamanho de ponto com coordenadas absolutas; por isso, é necessário ter um *header* e um *footer* para cada ponto. O objetivo do *header* é indicar qual o tamanho da codificação do ponto absoluto: enorme, grande, normal ou pequeno. O *footer* indica o que há depois do ponto corrente: um ponto absoluto, um ponto relativo por incremento, uma cadeia de *bits* ou fim de poligonal. Tanto o *header* quanto o *footer* ocupam dois *bits* e seus possíveis valores podem ser vistos na Tabela 4.1 e na Tabela 4.2, respectivamente. Essas tabelas se aplicam tanto a pontos absolutos quanto a pontos relativos por incremento.

Conforme dito anteriormente, o conceito de posição corrente é interno a uma poligonal e só existe após o primeiro ponto, que possui coordenadas absolutas. Essa convenção dispensa a necessidade de um *footer* que indique o tipo do primeiro ponto. Os tamanhos em *bits* das coordenadas dos pontos absolutos podem ser visto na Tabela 4.3

Header	Tamanho
00	Enorme
01	Grande
10	Normal
11	Pequeno

Tabela 4.1: Valores para o *header*

Footer	Próximo
00	Fim de poligonal
01	Ponto com coordenadas absolutas
10	Ponto com coordenadas relativas por incremento
11	Cadeia de *bits*

Tabela 4.2: Valores para o *footer*

Ponto absoluto	Coordenada x	Coordenada y
Enorme	30 *bits*	30 *bits*
Grande	18 *bits*	18 *bits*
Normal	14 *bits*	14 *bits*
Pequeno	10 *bits*	10 *bits*

Tabela 4.3: Tamanho das coordenadas dos pontos absolutos

As coordenadas dos pontos absolutos nunca possuem valores negativos, pois são referentes ao sistema inteiro Z_+^2 obtido após a quantização. Desse modo, os valores das coordenadas de um ponto absoluto enorme, por exemplo, variam de 0 a $2^{30}-1 \approx 10^{10}$. Por isso, a maior resolução admitida nesta proposta é $2^{30} \times 2^{30}$, isto é, a maior figura criada pode ter tamanho $2^{30} \times 2^{30}$. Com esta resolução, o semiperímetro do Equador (20037 km) pode ser representado em decímetros, ou seja, o maior erro admitido é igual a 1 dm. Supondo que o erro máximo em um mapa seja igual a 0.1 mm na escala da carta, a maior escala que se consegue representar utilizando a codificação apresentada é $1:10^3$. Quanto menor for a resolução, maior será a perda e menor será o tamanho do arquivo gerado.

O tamanho total ocupado pelos vários pontos absolutos pode ser obtido adicionando-se os tamanhos do *header*, do *footer* e das respectivas coordenadas. Os tamanhos totais desses pontos podem ser vistos na Tabela 4.4.

Ponto absoluto	*Header* (*bits*)	Coord. x (*bits*)	Coord. y (*bits*)	*Footer* (*bits*)	**Total (*bytes*)**
Enorme	2	30	30	2	**8**
Grande	2	18	18	2	**5**
Normal	2	14	14	2	**4**
Pequeno	2	10	10	2	**3**

Tabela 4.4: Tamanho dos pontos absolutos

4.2.2.2 Coordenadas Relativas por Incremento

Nas coordenadas relativas por incremento, o valor das coordenadas colocadas no arquivo é igual à diferença das coordenadas do ponto atual com relação às coordenadas do ponto anterior (no sistema inteiro de coordenadas da quantização).

O formato de um ponto com coordenadas relativas por incremento (Figura 4.6) é igual ao formato de um ponto com coordenadas absolutas, ou seja, é composto por um *header*, pelas coordenadas *x* e *y* e pelo *footer*.

header	coordenada x	coordenada y	*footer*

Figura 4.6: Formato dos pontos relativos por incremento

Da mesma forma que nos pontos com coordenadas absolutas, há mais de um tamanho de ponto com coordenadas relativas por incremento. As funções do *header* e do *footer* são as mesmas dos pontos com coordenadas absolutas, sendo que o *header* indica qual o tamanho do ponto com coordenadas relativas por incremento que será analisado a seguir. O *header* e o *footer* continuam ocupando dois *bits* e seus possíveis valores são iguais aos dos pontos absolutos, mostrados na Tabela 4.1 e na Tabela 4.2, respectivamente.

Os tamanhos em *bits* das coordenadas dos pontos relativos por incremento podem ser vistos na Tabela 4.5. As coordenadas desses pontos podem assumir valores negativos, logo as coordenadas de um ponto relativo por incremento enorme, por exemplo, podem variar de -2^{13} a $2^{13}-1$, isto é, de –8192 a 8191.

Ponto relativo por incremento	Coordenada x	Coordenada y
Enorme	14 *bits*	14 *bits*
Grande	10 *bits*	10 *bits*
Normal	6 *bits*	6 *bits*
Pequeno	2 *bits*	2 *bits*

Tabela 4.5: Tamanho das coordenadas dos pontos relativos por incremento

O tamanho total ocupado pelos vários pontos relativos por incremento pode ser obtido adicionando-se os tamanhos do *header*, do *footer* e das respectivas coordenadas, como mostra a Tabela 4.6.

Os pontos sempre são armazenados no menor tamanho. Os pontos absolutos normal e pequeno e os pontos relativos por incremento enorme e grande possuem o mesmo tamanho, respectivamente. Nesse caso, o ponto corrente sempre é considerado ponto relativo, a não ser que seja o primeiro ponto da poligonal.

Ponto relativo por incremento	*Header* (*bits*)	Coord. x (*bits*)	Coord. y (*bits*)	*Footer* (*bits*)	**Total (*bytes*)**
Enorme	2	14	14	2	**4**
Grande	2	10	10	2	**3**
Normal	2	6	6	2	**2**
Pequeno	2	2	2	2	**1**

Tabela 4.6: Tamanho dos pontos relativos por incremento

4.2.2.3 Coordenadas Relativas por Direção

Uma cadeia de *pixels* é composta por um conjunto de pontos adjacentes no sentido 8-conexo. Nesse trabalho, as cadeias de *pixels* são codificadas como cadeias de *bits*. (*Freemam chain codes*) [1].

Um ponto com coordenadas relativas por direção sempre pertence a uma cadeia de *bits*. Cada cadeia de *bits*, cujo formato é ilustrado na Figura 4.7, termina com um código de marcação de fim de cadeia. Como, no formato proposto, a cadeia de *bits* pode ser interna a uma poligonal, é necessária a presença de um *footer* (Tabela 4.2). O *header* não é necessário, porque as cadeias de *bits* só podem ter uma formação, isto é, não há diferentes formatos dessas cadeias.

direção	direção	...	fim cadeia	*footer*

Figura 4.7: Formato da cadeia de *bits*

As direções são calculadas segundo a rosa dos ventos (Figura 4.8). Por exemplo, supondo o segmento P_iP_{i+1} (Figura 4.8), o ponto inicial do segmento é colocado na origem da rosa dos ventos e de acordo com a localização do ponto P_{i+1} verifica-se qual a direção correspondente. Nesse caso, a direção corresponde é a noroeste. Os valores das direções são apresentados na Tabela 4.7.

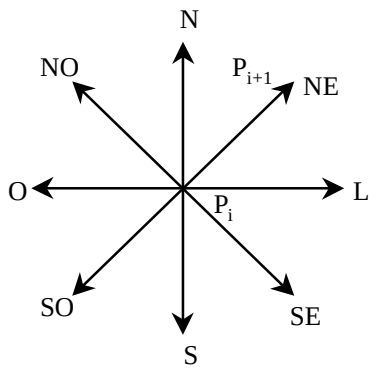


Figura 4.8: Direções

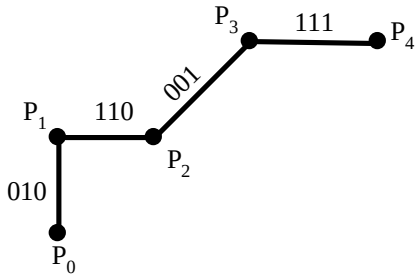
NO	N	NE	=	3	2	1	=	011	010	001
O	P_i	L		4	P_i	0		100	P_i	000
SO	S	SE		5	6	7		101	110	111

Tabela 4.7: Valores das direções na cadeia de *bits*

A primeira direção de cada cadeia de *bits* é armazenada segundo a rosa dos ventos. Da segunda direção em diante, é feita a relação da direção atual com a sua anterior. Esta relação é realizada verificando-se quantas direções têm que ser percorridas da direção anterior para chegar à direção atual, seguindo-se sempre o sentido anti-horário. Por exemplo, supondo que a direção anterior é norte e a atual é leste, o número de direções percorridas da direção norte até a direção leste, segundo a rosa dos ventos, é igual a 6, logo a direção atual relativa a anterior vale 6. A conta realizada é [(8–direção anterior) + direção atual] mod 8, no exemplo a conta é [(8–2) + 0] mod 8=6. A motivação para essa codificação relativa é aumentar a ocorrência das direções leste em detrimento das oeste. Com isso, espera-se que o algoritmo de compressão da fase posterior possa ser mais eficiente e cria-se um código improvável (oeste) que pode ser utilizado para codificar o fim de cadeia.

A marcação de fim da cadeia de *bits* é composta por duas direções oeste seguidas. Uma direção oeste relativa corresponde a uma volta de 180°. Por exemplo, um segmento é composto pelos pontos P_1 e P_2 e o próximo segmento é P_2P_1. A direção relativa do segmento P_2P_1 em relação ao seu anterior é oeste. Na codificação proposta neste trabalho, duas direções oeste não podem ocorrer no meio de uma cadeia.

Um exemplo de cadeia de *pixels* codificada como uma cadeia de *bits* pode ser visto na Figura 4.9. A primeira direção da cadeia de *bits* não pode ser relativa, pois não há direção anterior. Desse modo, a primeira direção da cadeia é a direção norte (P_0P_1). As próximas direções são sempre relativas às suas anteriores. A direção P_1P_2 é leste segundo a rosa dos ventos, porém esta deve ser relativa à direção de P_0P_1. Desse modo, da direção norte até a direção leste são percorridas 6 direções, logo a segunda direção da cadeia de *bits* é 110. O procedimento é o mesmo até a última direção (P_3P_4). Depois da última direção (111), acrescenta-se à cadeia a marcação de fim que é composta pelas duas direções oeste seguidas (direções sublinhadas). No fim da cadeia, é colocado o *footer* indicando o que está a seguir.



010, 110, 001, 111, 100, 100, *footer*

Figura 4.9: Exemplo de cadeia de *bits*

O algoritmo de Bresenham [2, 24, 25] também trabalha com direções e poderia ter sido utilizado na codificação de todos os segmentos de uma poligonal. Assim, a codificação de um segmento seria feita com 1 ou mais códigos de direção. Quando a distância entre pontos fosse 1, o segmento seria uma direção, quando a distância fosse maior, o segmento seria *rasterizado*, gerando uma cadeia de direções.

Como cada direção ocupa 3 *bits*, uma linha horizontal com tamanho 10, por exemplo, seria codificada com 30 *bits* (≈ 4 *bytes*), utilizando-se o algoritmo de Bresenham. Isto sem considerar nenhuma marcação de início ou de fim. Essa linha é muito melhor armazenada em um ponto relativo por incremento normal (2 *bytes*). Desse modo, descartou-se nesse trabalho a idéia de *rasterizar* segmentos.

A cadeia de *pixels* não possui um tamanho fixo, pois não é possível saber quantas direções esta vai possuir. Forma-se uma cadeia de *bits* a partir de uma cadeia de *pixels* sempre que o valor absoluto da diferença entre as coordenadas quantizadas *x* e *y* de dois pontos consecutivos em relação aos seus anteriores for menor ou igual a 1. Como cada direção da cadeia ocupa três *bits*, uma cadeia de *pixels* só é criada com no mínimo duas direções (total de 2 *bytes*), pois caso contrário é mais econômico guardar as coordenadas como um ponto relativo por incremento pequeno (1 *byte*).

Os casos em que no meio de uma cadeia de *bits* ocorrem direções oeste consecutivas podem ser confundidos com a marcação de fim de cadeia e precisam ser eliminados. Esta eliminação, além de diminuir o tamanho do arquivo, não causa perda de informação, desde que certos cuidados sejam tomados. Esses cuidados são diferentes para números pares e ímpares de direções oeste internas consecutivas.

Um número par de direções oeste consecutivas não causa nenhum problema,

pois um número 2*n* de direções oeste indica que o mesmo segmento da poligonal foi percorrido 2*n*+1 vezes seguidas (1 direção qualquer mais 2*n* direções oeste), mas terminou no mesmo ponto. Nesse caso, nenhuma direção oeste é considerada. Apenas a primeira direção, que é diferente de oeste, é considerada. Com isso, 2*n* pontos relativos são descartados, o arquivo criado fica menor e sem informações repetidas. A Figura 4.10 ilustra essa situação, onde a direção qualquer é a sul (101) e ocorrem 2 direções oeste (100) que são eliminadas. Nessa figura, as direções sublinhadas indicam a marcação de fim de cadeia e as direções em negrito são as direções oeste no meio da cadeia.

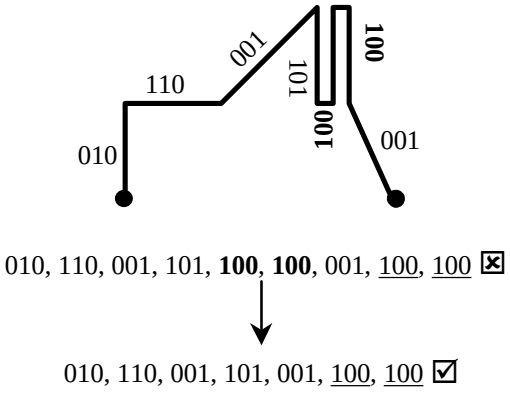


Figura 4.10: Número par de direções oeste consecutivas no meio da cadeia de *bits*

Quando ocorre um número ímpar (2*n*+1) de direções oeste, apenas uma direção oeste é considerada. Dessa forma, o arquivo contém apenas a primeira direção, que pode ser qualquer uma diferente de oeste, e uma direção oeste. Como ocorre na primeira situação, o arquivo criado fica menor, pois 2*n* pontos relativos são descartados (Figura 4.11).

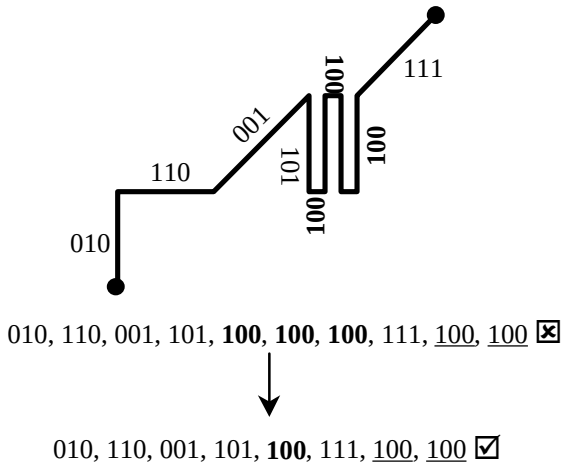


Figura 4.11: Número ímpar de direções oeste consecutivas no meio da cadeia de *bits*

Um número ímpar de direções oeste consecutivas causa um problema, quando a direção oeste considerada é a última direção de uma cadeia de *bits*, porque no final da cadeia ficam três direções oeste consecutivas: a considerada devido à seqüência ímpar de direções oeste e duas que marcam o fim de cadeia. O problema ocorre no momento de interpretação do arquivo, pois duas cadeias oeste indicam fim de cadeia. As duas primeiras direções serão interpretadas como fim de cadeia e a partir da terceira direção oeste, a leitura será errada, comprometendo a interpretação. Quatro opções para resolução deste problema foram analisadas (Figura 4.12):

(a) terminar a cadeia e substituir a última direção (oeste) por um ponto relativo por incremento pequeno;

(b) eliminar da cadeia a última direção diferente de oeste e as últimas direções oeste, perdendo o último ponto;

(c) aumentar o tamanho de cada direção para 4 *bits*, de modo a criar outra codificação de fim de cadeia;

(d) colocar no início de cada cadeia de *bits* o número de direções que esta possui.

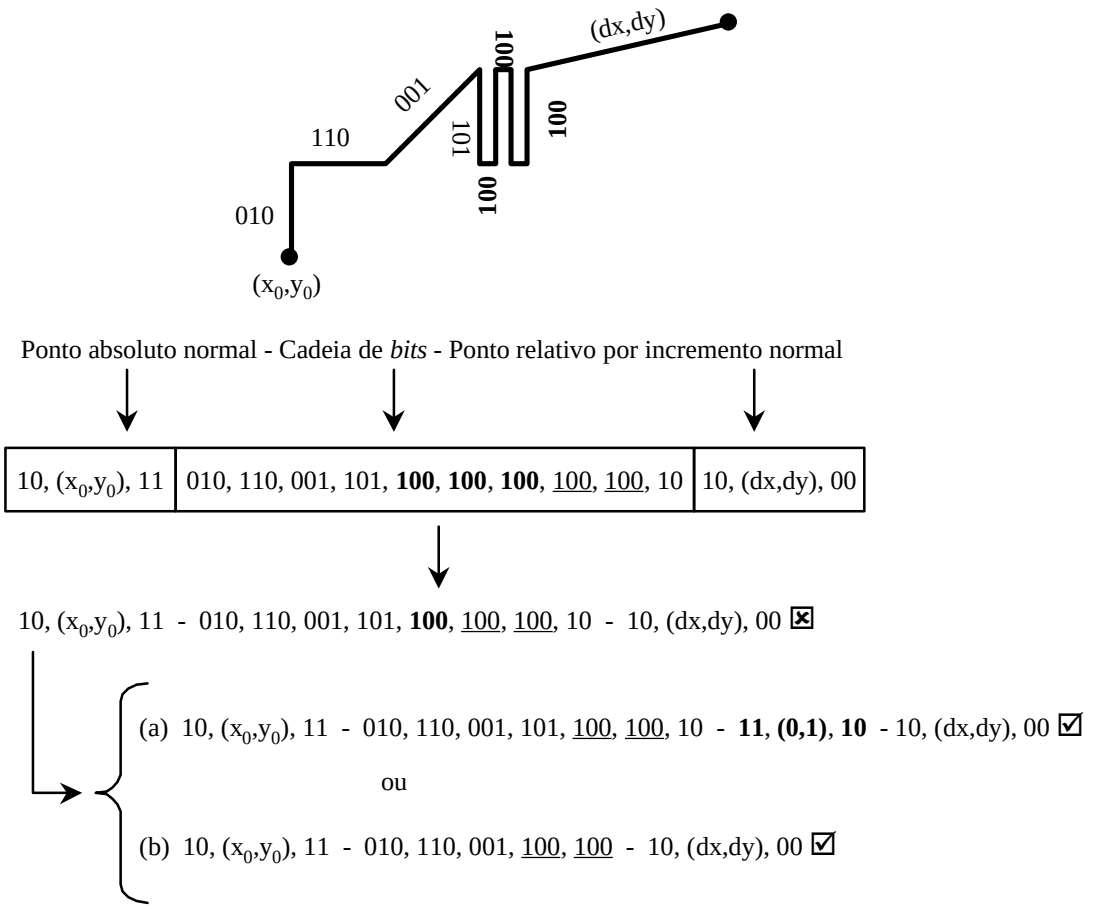


Figura 4.12: Soluções para um número ímpar de direções oeste consecutivas no final da cadeia de *bits*

A segunda opção eliminaria um segmento da poligonal original. Mesmo sendo um segmento pequeno, isto poderia ocorrer várias vezes em um único mapa, o que causaria a perda de algumas características originais. Por isso, este procedimento não foi considerado uma boa solução.

A terceira opção aumentaria o tamanho de todas as direções, logo também aumentaria bastante o tamanho final do arquivo, o que não é objetivo dessa proposta.

Sendo assim, das opções apresentadas, restam a primeira e a quarta a serem analisadas. Nos testes que serão apresentados adiante, foram realizadas algumas estatísticas. Em relação à primeira opção, uma estatística obtida foi o maior número de direções oeste que seriam substituídas por um ponto relativo por incremento pequeno. Sobre a quarta opção, obteve-se o tamanho da maior cadeia de *pixels*, ou seja, o maior número de direções que uma cadeia de *pixels* apresentou, considerando-se todas as cadeias de todos os arquivos utilizados nos testes, e o maior número de cadeias que

um arquivo possuiu. Com estas estatísticas, foi possível observar qual das opções apresenta melhor resultado para os casos testes estudados.

Analisando a quarta opção, o maior número direções de uma cadeia de *bits*, sem considerar as direções que marcam o fim da cadeia, foi igual a 1063. Assim, seriam necessários no mínimo 11 *bits* no início de cada cadeia para guardar o número de direções que essa possui. No pior caso, um arquivo chegou a ter 78083 cadeias de *pixels*. Desse modo, seriam gastos 11 *bits* × 78083 cadeias = 858913 *bits* ≈ 107364 *bytes* ≈ 104 *Kbytes* a mais no arquivo final para colocar o número de direções no início de cada cadeia. Com a inserção do número de direções no início de cada cadeia de *bits*, não seria mais necessário colocar as duas direções oeste que marcam o fim da mesma, ou seja, seriam diminuídos 6 *bits* × 78083 cadeias = 468498 *bits* ≈ 58562 *bytes* ≈ 57 *Kbytes*. Finalizando, o arquivo criado teria 104 *Kbytes* – 57 *Kbytes* = 47 *Kbytes* a mais, se a quarta opção fosse utilizada.

Na primeira opção, o maior número de direções oeste convertidas em pontos relativos por incremento pequenos, que ocupam 1 *byte*, foi igual a 792. Logo, no pior caso da primeira opção, seriam utilizados 792 *bytes* a mais no arquivo criado, enquanto a quarta opção gastaria mais 47 *Kbytes* no arquivo final. Desse modo, concluiu-se que a melhor solução é a primeira. Uma análise mais detalhada da quarta opção mostra que mesmo utilizando uma codificação adaptativa para o número de pontos da poligonal, dificilmente essa geraria melhores resultados que a primeira opção.

Sendo assim, na segunda proposta do *TeCGraf Web Format*, quando ocorre um número ímpar de direções oeste consecutivas no final de uma cadeia de *bits*, a cadeia termina na última direção diferente de oeste e uma direção oeste é colocada logo após a cadeia utilizando um ponto relativo por incremento pequeno.

4.3 Compressão

A etapa de compressão consiste em aplicar o algoritmo LZO1X_1 da biblioteca LZO no arquivo gerado pela codificação compacta. O LZO1X é o algoritmo de uso geral da LZO, isto é, o algoritmo que apresenta resultados melhores na maioria dos casos. O LZO1X_1 é o nível mais rápido dentre os algoritmos LZO1X e utiliza 64 *Kbytes* de memória para compressão. Na descompressão, todos os algoritmos apresentam um bom desempenho.

Para comprimir, basta passar para o algoritmo os *bytes* e o número de *bytes* a serem comprimidos e uma área de memória adicional para trabalho (64 *Kbytes* de memória). O algoritmo retorna os *bytes* e o número de *bytes* comprimidos.

4.4 Descompressão e Decodificação

A etapa de descompressão é a primeira etapa executada na interpretação de um arquivo TWF. A segunda e última etapa é a decodificação.

A descompressão é feita utilizando-se a função de descompressão relativa ao algoritmo LZO1X_1 da biblioteca LZO. Para tanto, bastar passar para a função os *bytes* e o número de *bytes* comprimidos, que são retornados os *bytes* e o número de *bytes* descomprimidos.

Para decodificar um arquivo, basta seguir os passos da codificação de modo inverso, ou seja, os últimos passos da codificação são os primeiros da decodificação e assim por diante. A decodificação é a etapa inversa da codificação.

5. Casos Testes

Nos testes, foram utilizados arquivos origem de dois tipos: BIN e CGM. O formato BIN, utilizado na dissertação de mestrado de Derraik [11], é binário e cada poligonal é composta pelo número de pontos (*int*) que possui, seguido pelos pontos em coordenadas reais (*float*). O formato CGM utilizado é binário e com coordenadas inteiras de 32 *bits*. Os arquivos utilizados nos testes foram:

- Mapa da África [39] (Figura 5.1), com os limites internos e as fronteiras.
- Curvas de Nível de Madison [40] (Figura 5.2).
- Curvas de Nível de Anchorage [40] (Figura 5.3).
- Mapa de Municípios do Brasil [41] (Figura 5.4).
- Mapa da Cobertura Vegetal do Brasil (Figura 1.1).
- Mapa da Situação das Áreas Indígenas no Brasil (Figura 5.5).

Os mapas originais da Cobertura Vegetal do Brasil e da Situação das Áreas Indígenas no Brasil estão no formato CGM, enquanto que os demais estão no formato BIN. Todos os arquivos BIN foram convertidos para CGM utilizando-se a biblioteca CD, porém o inverso não ocorreu, porque o formato BIN não é muito utilizado na transmissão de mapas. A comparação desses mapas em várias resoluções no formato CGM será analisada no capítulo 6.

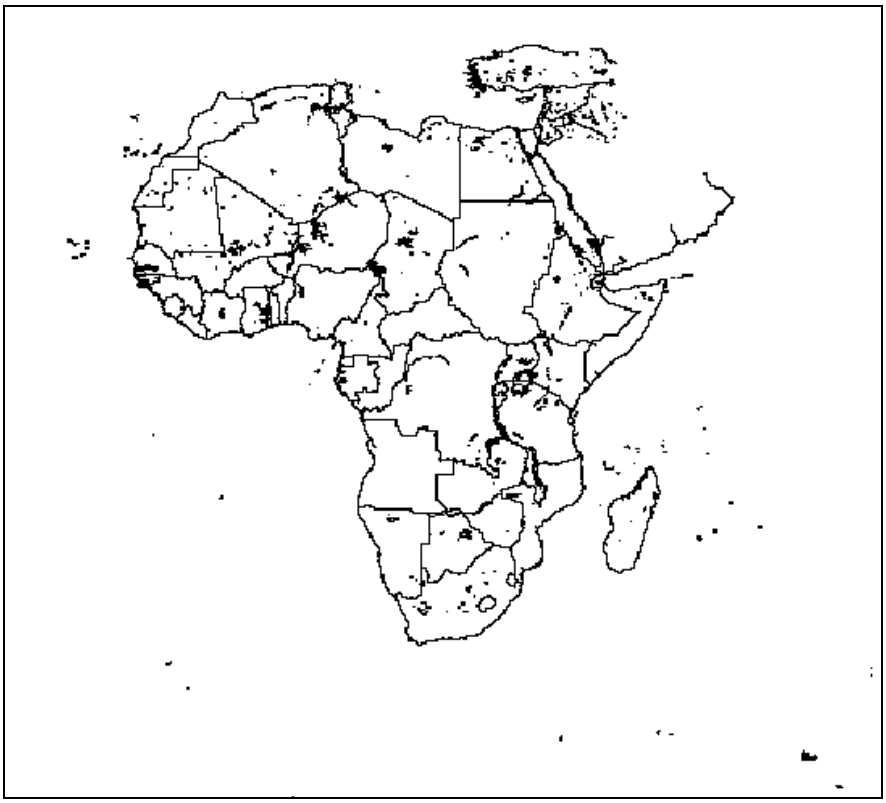


Figura 5.1: Mapa da África

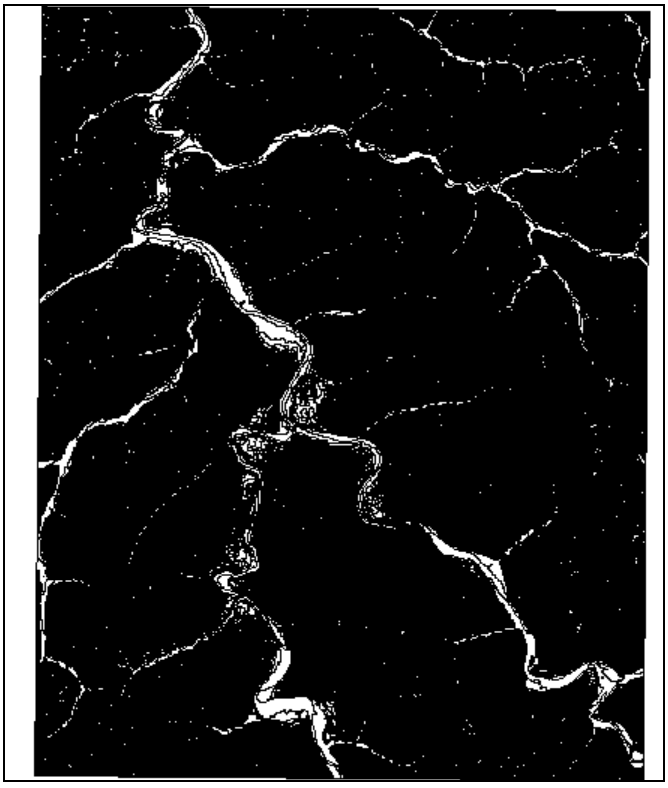


Figura 5.2: Curvas de Nível de Madison

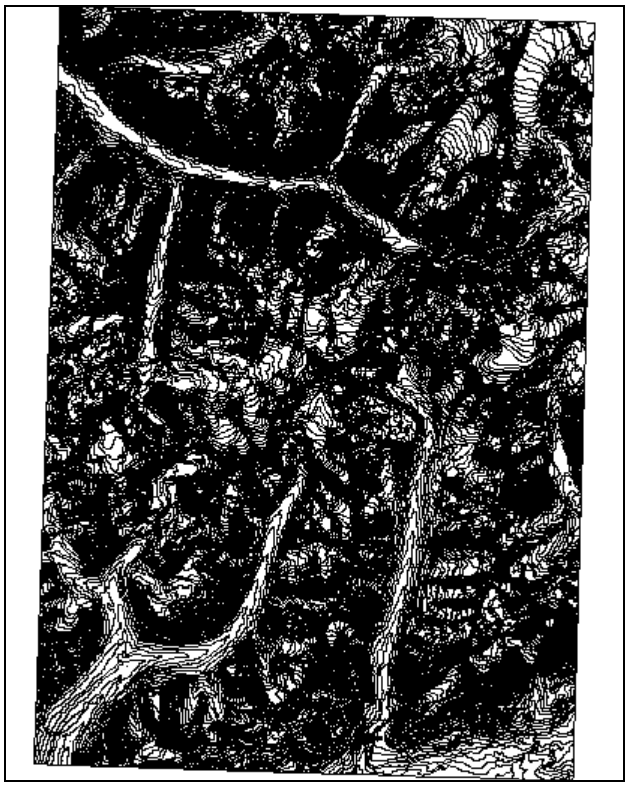


Figura 5.3: Curvas de Nível de Anchorage

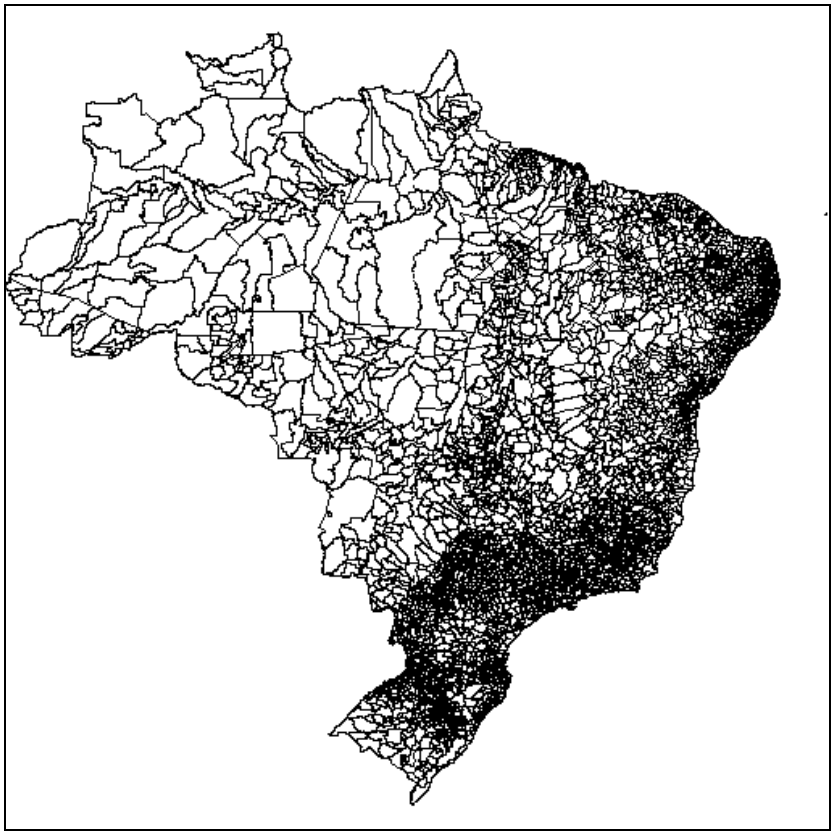


Figura 5.4: Mapa de Municípios do Brasil

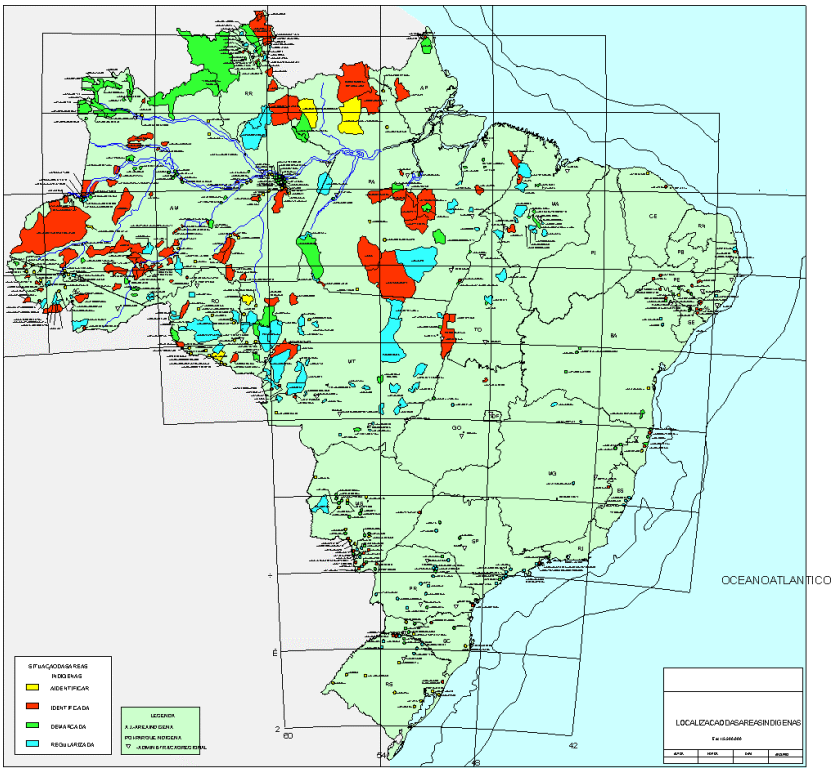


Figura 5.5: Mapa da Situação das Áreas Indígenas no Brasil

Na Tabela 5.1, são apresentados alguns dados sobre esses arquivos, como o formato original, o tamanho, o número de pontos, o número de poligonais e a média de pontos por poligonal (número de pontos/número de poligonais). Estes dados são parâmetros para avaliação dos resultados que serão apresentados a seguir. Na escolha dos casos testes, houve a preocupação em trabalhar com arquivos de características diferentes, principalmente a média de pontos por poligonal.

Arquivo	Formato	Tamanho (Kb)	Número de pontos	Número de poligonais	Média de pontos por poligonal
África	BIN	3341	426516	2149	198
Madison	BIN	5212	665674	2840	234
Anchorage	BIN	1745	222551	1582	141
BR Municípios	BIN	5666	716933	16586	43
BR Vegetação	CGM	830	196977	2781	71
BR Indígena	CGM	715	164760	1643	100

Tabela 5.1: Dados sobre os arquivos originais

6. Resultados

Foram utilizados os formatos CGM, DWF, DXF, CDM, EMF, BMP e GIF na comparação com as duas propostas do *TeCGraf Web Format* apresentadas. Os arquivos nos formatos citados, exceto os DWF, foram criados através das bibliotecas CD – *Canvas Draw* – Versão 3.6 [33] e IM – Biblioteca de Acesso a Arquivos de Imagens *Bitmaps* – Versão 2.2 [35], ambas desenvolvidas pelo TeCGraf – Grupo de Tecnologia em Computação Gráfica [32]. Os arquivos no formato DWF foram criados a partir dos respectivos DXF utilizando-se o *AutoCAD Release 14.0*.

A partir dos arquivos originais foram geradas as imagens para comparar o tamanho das mesmas. Foram criadas imagens em todos os formatos nas resoluções 512 ×512, 1024×1024, 2048×2048 e 4096×4096. Serão apresentados os resultados obtidos nas etapas de quantização, que vale para todos os formatos; de codificação, comparando as duas propostas do TWF com tamanhos dos arquivos nos vários formatos; e de compressão.

6.1 Quantização

Na quantização, os pontos dos arquivos originais são convertidos para um sistema inteiro, cujos valores variam de zero até a resolução do arquivo a ser gerado. Como já foi visto, quanto menor a resolução, mais pontos irão coincidir e mais pontos serão eliminados do arquivo resultante.

O resultado da quantização consiste no número de pontos que foram realmente codificados no TWF em comparação com o número de pontos originais dos arquivos (Tabela 6.1 e Gráfico 6.1).

Arquivo	Núm. pontos originais	512x512		1024x1024		2048x2048		4096x4096	
		Núm. Pontos	% original	Núm. pontos	% original	Núm. pontos	% original	Núm. pontos	% original
África	426516	14387	3	26657	6	50161	12	95763	22
Madison	665674	402919	61	551159	83	632607	95	658471	99
Anchorage	222551	154369	69	200114	90	217877	98	221880	100
BR Municípios	716933	78840	11	133756	19	231710	32	383925	54
BR Vegetação	196977	93571	48	147996	75	178032	90	188622	96
BR Indígena	164760	52490	32	77943	47	101746	62	129693	79

Tabela 6.1: Quantização

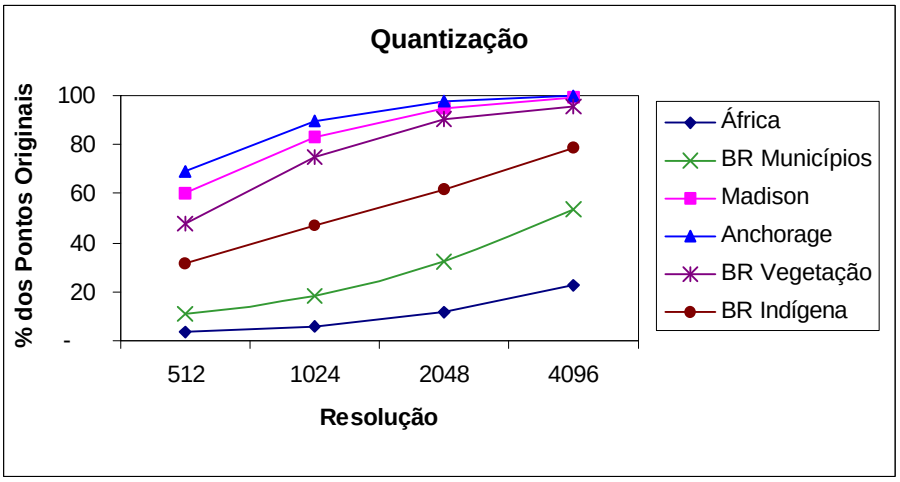


Gráfico 6.1: Quantização

6.2 Codificação Compacta

A seguir serão apresentadas as tabelas com as comparações dos tamanhos dos arquivos em vários formatos e em várias resoluções. TWF_P1 e TWF_P2 são as propostas 1 e 2, respectivamente, do *TeCGraf Web Format*. Os formatos que não possuem compressão foram comprimidos com o algoritmo LZOX_1 da biblioteca LZO citada anteriormente. O tamanho final dos arquivos depois de passarem pela compressão são comparados com o tamanho do arquivo GIF correspondente, isto é, verifica-se a qual a porcentagem do arquivo GIF corresponde o tamanho do arquivo no formato que está sendo analisado.

	4096 x 4096								
Arquivo	TWF_P1	TWF_P2	CGM	DWF	DXF	MF	EMF	BMP	GIF
África (Kb)	200	53	1701	98	23894	5787	1726	16386	88
Comp. LZO1X_1 (Kb)	86	45	533		323	812	564	263	
(%) GIF	98	51	606	111	367	923	641	299	100
Madison (Kb)	1304	1233	2645	1221	33119	8843	2649	16386	1512
Comp. LZO1X_1 (Kb)	1095	1080	2476		2856	3959	2514	4957	
(%) GIF	72	71	164	81	189	262	166	328	100
Anchorage (Kb)	444	431	895	438	11148	2981	909	16386	877
Comp. LZO1X_1 (Kb)	389	387	851		2014	1378	874	2986	
(%) GIF	44	44	97	50	230	157	100	340	100
BR Municípios (Kb)	848	335	3055	521	37017	9813	3255	16386	337
Comp. LZO1X_1 (Kb)	468	294	1612		1557	2576	1823	1073	
(%) GIF	139	87	478	155	462	764	541	318	100
BR Vegetação (Kb)	426	378	839	311	10120	2752	1003	16386	438
Comp. LZO1X_1 (Kb)	284	277	634		1620	1062	682	781	
(%) GIF	65	63	145	71	370	242	156	178	100
BR Indígena (Kb)	316	209	722	197	8439	2312	755	16386	284
Comp. LZO1X_1 (Kb)	159	118	452		1173	778	466	586	
(%) GIF	56	42	159	69	413	274	164	206	100

Tabela 6.2: Comparação dos formatos na resolução 4096×4096

	2048 x 2048								
Arquivo	TWF_P1	TWF_P2	CGM	DWF	DXF	MF	EMF	BMP	GIF
África (Kb)	111	35	1701	63	21113	5532	248	4098	36
Comp. LZO1X_1 (Kb)	50	28	319		609	434	170	105	
(%) GIF	139	78	886	175	1692	1206	472	292	100
Madison (Kb)	1253	1037	2645	1013	32751	8475	2548	4098	486
Comp. LZO1X_1 (Kb)	870	826	2243		5423	3497	2247	1512	
(%) GIF	179	170	462	208	1116	720	462	311	100
Anchorage (Kb)	435	386	895	370	11013	2846	893	4098	337
Comp. LZO1X_1 (Kb)	315	305	790		1885	1229	807	1085	
(%) GIF	93	91	234	110	559	365	239	322	100
BR Municípios (Kb)	550	225	3055	356	36583	9380	1344	4098	131
Comp. LZO1X_1 (Kb)	291	182	1148		725	1744	880	410	
(%) GIF	222	139	876	272	553	1331	672	313	100
BR Vegetação (Kb)	404	300	839	260	10010	2643	961	4098	172
Comp. LZO1X_1 (Kb)	229	208	569		1496	943	603	324	
(%) GIF	133	121	331	151	870	548	351	188	100
BR Indígena (Kb)	261	172	722	155	8345	2220	645	4098	115
Comp. LZO1X_1 (Kb)	106	89	379		960	628	369	251	
(%) GIF	92	77	330	135	835	546	321	218	100

Tabela 6.3: Comparação dos formatos na resolução 2048×2048

	1024 x 1024								
Arquivo	TWF_P1	TWF_P2	CGM	DWF	DXF	MF	EMF	BMP	GIF
África (Kb)	65	24	1701	40	20593	5013	148	1026	15
Comp. LZO1X_1 (Kb)	31	18	184		362	232	96	43	
(%) GIF	207	120	1227	267	2413	1547	640	287	100
Madison (Kb)	1094	662	2645	693	32051	7775	2229	1026	71
Comp. LZO1X_1 (Kb)	644	541	1924		4736	2940	1818	232	
(%) GIF	907	762	2710	976	6670	4141	2561	327	100
Anchorage (Kb)	401	267	895	264	10778	2611	824	1026	106
Comp. LZO1X_1 (Kb)	243	211	695		1691	1048	686	329	
(%) GIF	229	199	656	249	1595	989	647	310	100
BR Municípios (Kb)	359	164	3055	237	35739	8536	947	1026	48
Comp. LZO1X_1 (Kb)	184	123	761		479	1049	564	145	
(%) GIF	383	256	1585	494	998	2185	1175	302	100
BR Vegetação (Kb)	344	141	839	177	9762	2398	844	1026	67
Comp. LZO1X_1 (Kb)	168	111	480		1250	770	472	129	
(%) GIF	251	166	716	264	1866	1149	704	193	100
BR Indígena (Kb)	214	116	722	113	8141	2016	552	1026	47
Comp. LZO1X_1 (Kb)	80	49	296		727	454	283	105	
(%) GIF	170	104	630	240	1547	966	602	223	100

Tabela 6.4: Comparação dos formatos na resolução 1024×1024

	512 x 512								
Arquivo	TWF_P1	TWF_P2	CGM	DWF	DXF	MF	EMF	BMP	GIF
África (Kb)	41	19	1701	24	20562	4981	92	258	7
Comp. LZO1X_1 (Kb)	19	13	107		111	136	55	18	
(%) GIF	271	186	1529	343	1586	1943	786	257	100
Madison (Kb)	804	288	2645	432	31895	7618	1649	258	8
Comp. LZO1X_1 (Kb)	419	263	1547		1239	2345	1235	23	
(%) GIF	5238	3288	19338	5400	15488	29313	15438	288	100
Anchorage (Kb)	311	119	895	169	10740	2574	643	258	20
Comp. LZO1X_1 (Kb)	163	106	569		487	873	492	60	
(%) GIF	815	530	2845	845	2435	4365	2460	300	100
BR Municípios (Kb)	252	138	3055	156	35657	8454	702	258	16
Comp. LZO1X_1 (Kb)	122	91	501		423	658	372	46	
(%) GIF	763	569	3131	975	2644	4113	2325	288	100
BR Vegetação (Kb)	238	84	839	112	9743	2379	630	258	24
Comp. LZO1X_1 (Kb)	107	63	373		266	582	300	47	
(%) GIF	446	263	1554	467	1108	2425	1250	196	100
BR Indígena (Kb)	164	75	722	75	8124	1999	451	258	19
Comp. LZO1X_1 (Kb)	49	31	221		185	332	199	41	
(%) GIF	258	163	1163	395	974	1747	1047	216	100

Tabela 6.5: Comparação dos formatos na resolução 512×512

Uma outra visão dos dados apresentados nas tabelas de comparação dos formatos pode ser obtida analisando-se os gráficos seguintes. Cada gráfico corresponde a um arquivo teste. O Gráfico 6.2 utiliza o arquivo África para mostrar a evolução do tamanho dos arquivos gerados nos vários formatos com o crescimento da resolução.

De acordo com o Gráfico 6.2, os formatos que apresentam os melhores resultados são: TWF_P1, TWF_P2, DWF e GIF. Dessa forma, apenas estes formatos serão analisados nos próximos gráficos. O Gráfico 6.3 apresenta a evolução do tamanho dos arquivos gerados nos formatos de melhor resultado com o crescimento da resolução, ainda utilizando o arquivo teste África como exemplo.

No Gráfico 6.3, observa-se que a inclinação das curvas relativas a cada um dos formatos aumenta à medida que a resolução cresce, sendo que a curva do formato GIF é a que apresenta o maior aumento na inclinação. As curvas dos formatos TWF_P1 e DWF apresentam um aumento intermediário e a curva do formato TWF_P2 mostra o menor aumento na inclinação. Logo, o formato que apresenta menor aumento no tamanho do arquivo com o crescimento da resolução é o TWF_P2.

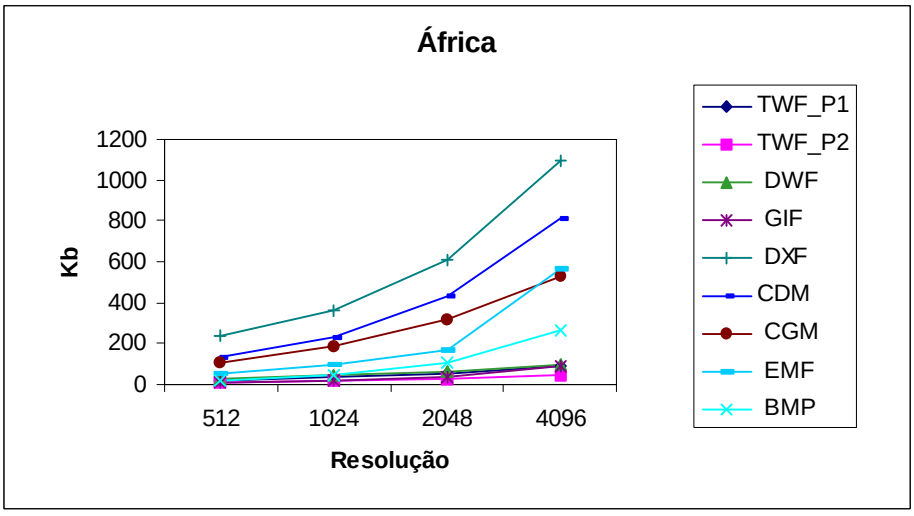


Gráfico 6.2: Comparação dos formatos no arquivo África

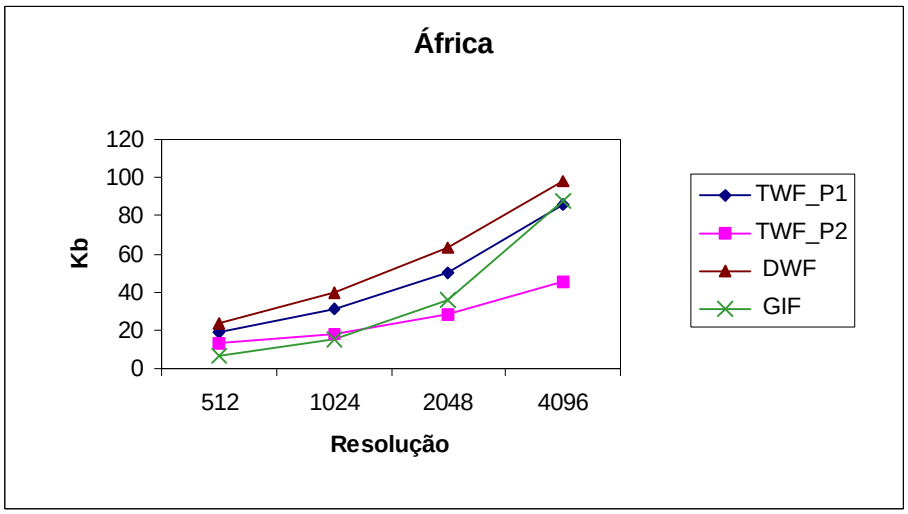


Gráfico 6.3: Comparação dos formatos TWF_P1, TWF_P2, DWF e GIF no arquivo África

Nos próximos gráficos, continuará sendo realizada a comparação entre os formatos TWF_P1, TWF_P2, DWF e GIF para os vários arquivos de teste, sendo que os formatos TWF_P1, TWF_P2 e DWF serão analisados em relação ao GIF. O tamanho do arquivo GIF servirá de unidade para comparação, ou seja, será analisada a razão entre os tamanhos dos arquivos em um determinado formato e o tamanho do arquivo GIF na resolução correspondente. Note que esta razão para o formato GIF será sempre igual a 1.

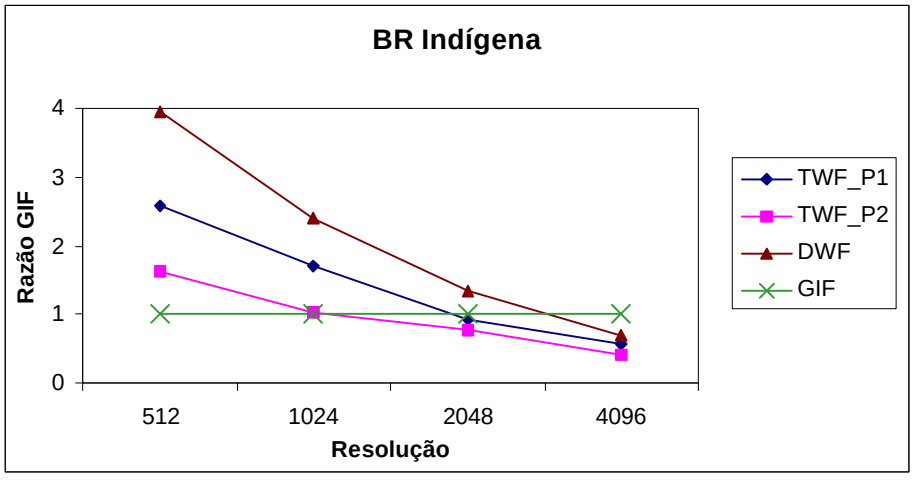


Gráfico 6.4: África - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF

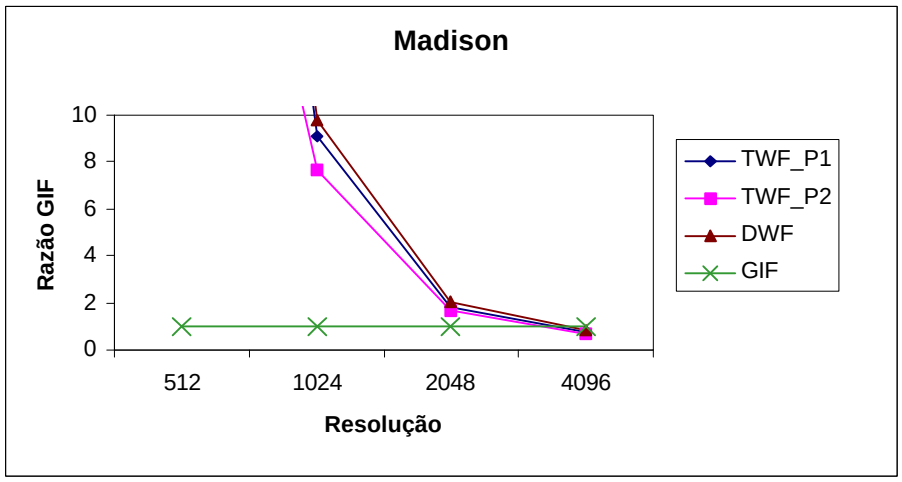


Gráfico 6.5: Madison - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF

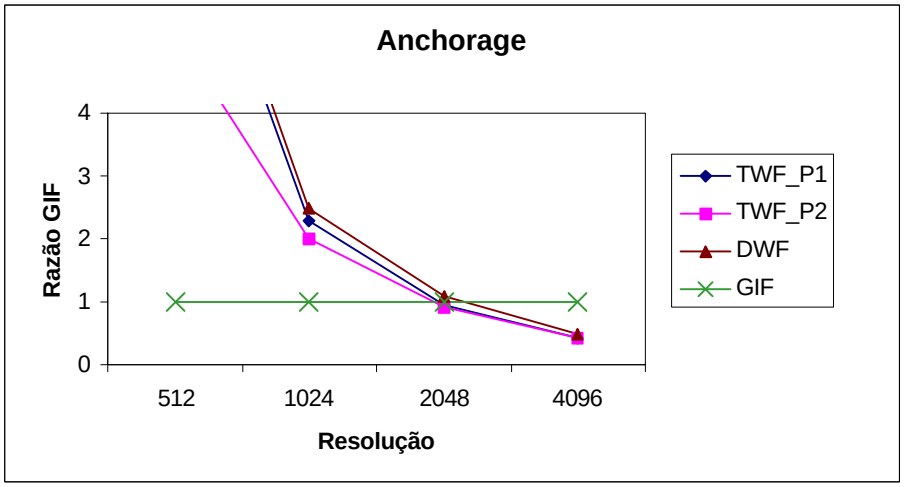


Gráfico 6.6: Anchorage - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF

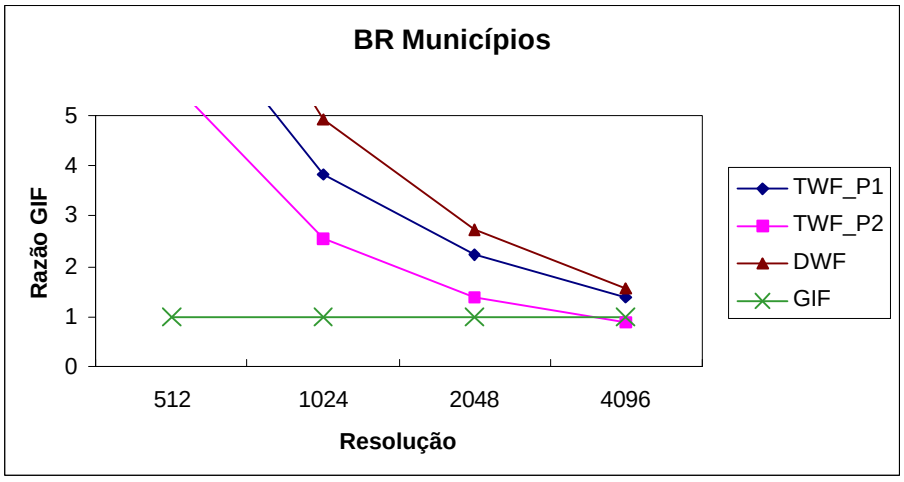


Gráfico 6.7: BR Municípios - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF

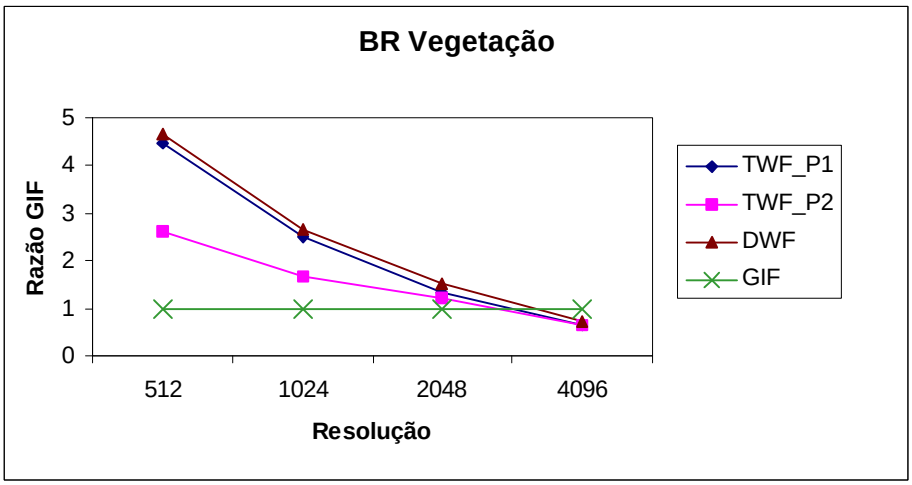


Gráfico 6.8: BR Vegetação - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF

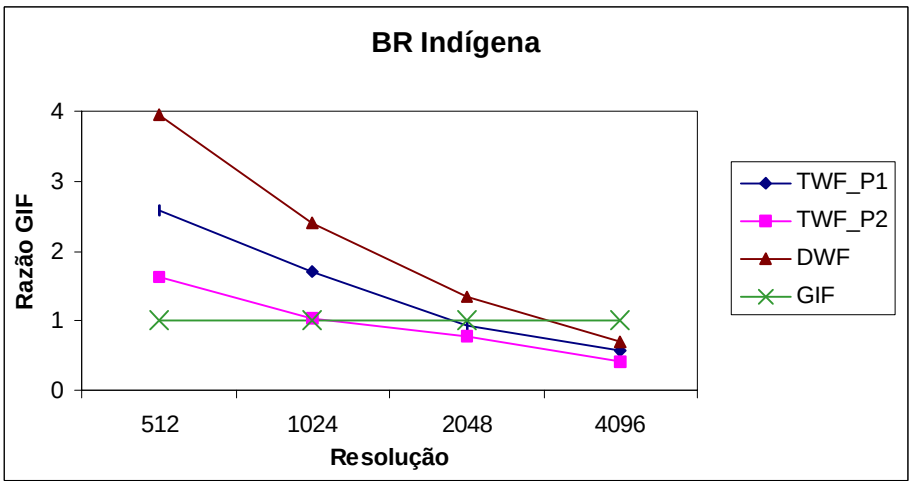


Gráfico 6.9: BR Indígena - Razão dos formatos TWF_P1, TWF_P2 e DWF em relação ao GIF

Como pôde ser observado, a segunda proposta do TWF apresentou resultados melhores ou tão bons quanto os outros formatos. A segunda proposta do TWF apresentou melhores resultados em comparação com a primeira proposta. Por isso, serão apresentados mais dados sobre a proposta 2. Em ambas as propostas, quanto maior a resolução, melhor o resultado no TWF em relação aos arquivos GIF.

A proposta 2 do *TeCGraf Web Format* possibilita uma análise da quantidade de cada tipo de ponto que foi gerada, para cada arquivo origem, em cada uma das resoluções apresentadas. Nas tabelas seguintes, também podem ser revistos os resultados da etapa de quantização: número original de pontos de cada arquivo e número de pontos gerados para cada resolução. São apresentados o número de pontos

e de poligonais originais de cada arquivo, além do número de pontos armazenados para cada arquivo em cada uma das resoluções, sendo que este último corresponde à coluna Pontos nas tabelas. As colunas AP, AN, AG e AE correspondem aos tipos de ponto absoluto pequeno, normal, grande e enorme, respectivamente. As colunas RP, RN, RG e RE correspondem aos tipos de ponto relativo por incremento pequeno, normal, grande e enorme, respectivamente. A coluna RDr indica o número de pontos relativos por direção quantizada e a DrO indica o número de pontos que foram eliminados, isto é, que não foram gravados porque correspondiam a direções oeste consecutivas em uma cadeia de *bits*.

África	Núm. Original de Pontos = 426516					Núm. Original de Poligonais = 2149					
Resolução	Pontos	AP	AN	AG	AE	RP	RN	RG	RE	RDr	DrO
4096x4096	95763	4	2145	0	0	321	122	0	0	92539	632
2048x2048	50161	294	1855	0	0	433	26	0	0	46239	1314
1024x1024	26657	2149	0	0	0	467	11	0	0	22240	1790
512x512	14387	2149	0	0	0	482	5	0	0	10275	1476

Tabela 6.6: TWF Proposta 2 - Tipos de pontos no arquivo África

Madison	Núm. Original de Pontos = 665674					Núm. Original de Poligonais = 2840					
Resolução	Pontos	AP	AN	AG	AE	RP	RN	RG	RE	RDr	DrO
4096x4096	658471	225	2615	0	0	46796	586493	596	0	21744	2
2048x2048	632607	699	2141	0	0	90770	421643	11	0	117323	20
1024x1024	551159	2840	0	0	0	69418	187949	0	0	290860	92
512x512	402919	2840	0	0	0	19806	42417	0	0	337680	176

Tabela 6.7: TWF Proposta 2 - Tipos de pontos no arquivo Madison

Anchorage	Núm. Original de Pontos = 222551					Núm. Original de Poligonais = 1582					
Resolução	Pontos	AP	AN	AG	AE	RP	RN	RG	RE	RDr	DrO
4096x4096	221880	69	1513	0	0	8567	208793	369	0	2569	0
2048x2048	217877	457	1125	0	0	27168	169103	23	0	19999	2
1024x1024	200114	1582	0	0	0	28505	83356	6	0	86641	24
512x512	154369	1582	0	0	0	8379	18301	0	0	126013	94

Tabela 6.8: TWF Proposta 2 - Tipos de pontos no arquivo Anchorage

BR Municípios	Núm. Original de Pontos = 716933					Núm. Original de Poligonais = 16596					
Resolução	Pontos	AP	AN	AG	AE	RP	RN	RG	RE	RDr	DrO
4096x4096	383925	0	16596	0	0	11637	28069	221	0	322526	4876
2048x2048	231710	1263	15333	0	0	3727	8045	33	0	194567	8742
1024x1024	133756	16596	0	0	0	3209	3772	2	0	100549	9628
512x512	78840	16596	0	0	0	4638	1569	0	0	48365	7672

Tabela 6.9: TWF Proposta 2 - Tipos de pontos no arquivo de Municípios do Brasil

BR Vegetação	Núm. Original de Pontos = 196977					Núm. Original de Poligonais = 2781					
Resolução	Pontos	AP	AN	AG	AE	RP	RN	RG	RE	RDr	DrO
4096x4096	188622	26	2755	0	0	12552	159425	1427	19	12384	34
2048x2048	178032	225	2556	0	0	27848	98977	672	11	47655	88
1024x1024	147996	2781	0	0	0	7243	17515	226	9	120066	156
512x512	93571	2781	0	0	0	1610	3855	73	0	84906	346

Tabela 6.10: TWF Proposta 2 - Tipos de pontos no arquivo de Vegetação do Brasil

BR Indígena	Núm. Original de Pontos = 164760					Núm. Original de Poligonais = 1643					
Resolução	Pontos	AP	AN	AG	AE	RP	RN	RG	RE	RDr	DrO
4096x4096	129693	17	1626	0	0	4743	66188	602	35	56406	76
2048x2048	101746	229	1414	0	0	9661	45793	158	16	44383	92
1024x1024	77943	1643	0	0	0	5507	17290	105	12	53102	284
512x512	52490	1643	0	0	0	860	2565	101	0	46583	738

Tabela 6.11: TWF Proposta 2 - Tipos de pontos no arquivo de Áreas Indígenas do Brasil

Como foi visto, a quantização diminui consideravelmente o número de pontos e grande parte dos pontos foram armazenados como pontos relativos por direção quantizada, o que indica uma melhora em relação aos tipos de armazenamentos nos formatos vetoriais existentes.

Os dados das tabelas que mostram os tipos de pontos dos arquivos no formato TWF_P2 também podem ser visualizados graficamente. Para exemplificar, os dados do arquivo África são apresentados nos gráficos seguintes.

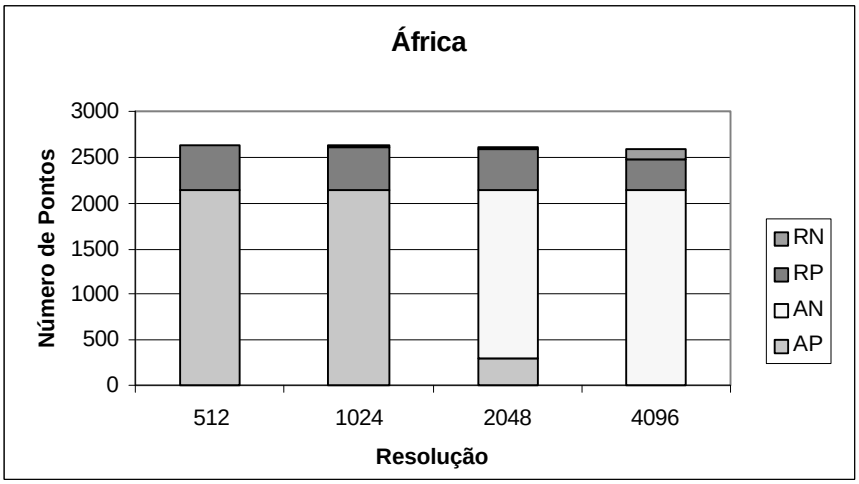


Gráfico 6.10: TWF Proposta 2 - Pontos absolutos e relativos por incremento no arquivo África

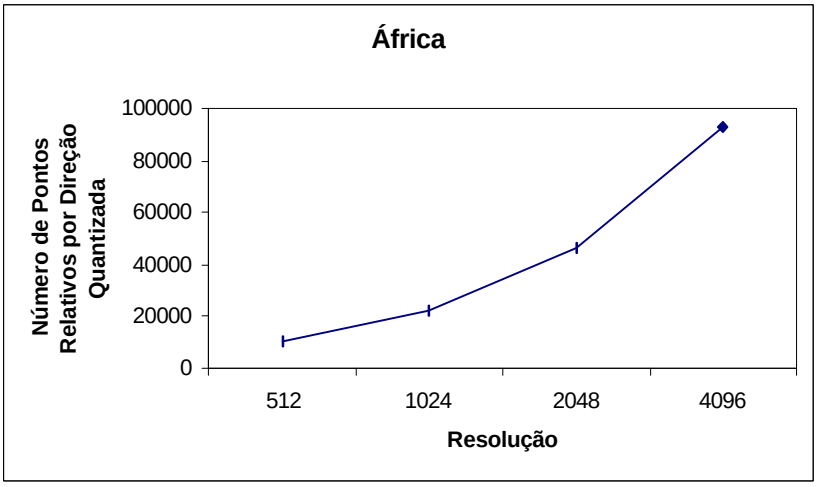


Gráfico 6.11: TWF Proposta 2 - Pontos relativos por direção quantizada no arquivo África

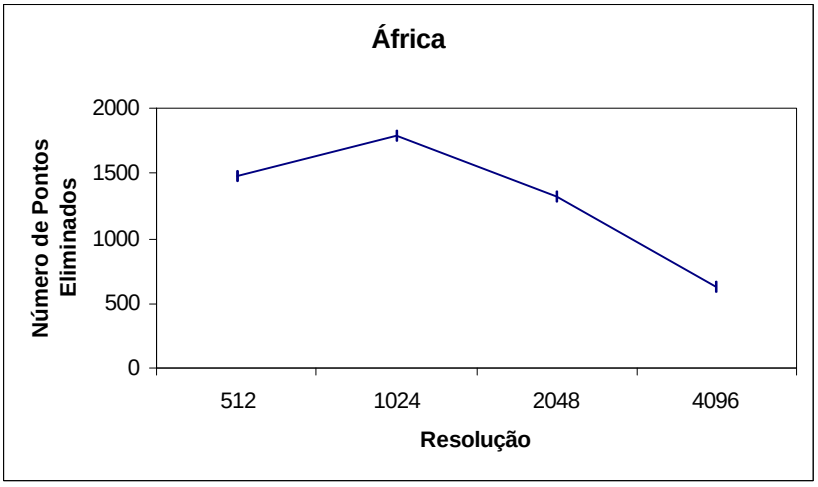


Gráfico 6.12: TWF Proposta 2 – Pontos eliminados (direções oeste consecutivas) do arquivo África.

O Gráfico 6.10 mostra que com o aumento da resolução a quantidade de pontos absolutos pequenos diminui e a quantidade de pontos absolutos normais aumenta, porém o número de pontos absolutos se mantém aparentemente constante. Isto ocorre porque o número de pontos absolutos deve ser no mínimo igual ao número de poligonais do arquivo. O número de pontos relativos por incremento também se mantém aparentemente constante, sendo que com o aumento da resolução aumenta o número de pontos relativos por incremento normal e diminui o número de pontos relativos por incremento pequeno.

No mapa da África, as poligonais possuem muitos segmentos, apresentam muitos detalhes . Assim, quanto maior a resolução mais esses detalhes aparecem, favorecendo a codificação com pontos relativos por direção quantizada (Gráfico 6.11). O Gráfico 6.12 mostra que o número de pontos relativos por direção eliminados, não possui uma correspondência direta com o aumento da resolução.

Ainda baseado na segunda proposta, foi feita uma avaliação do número médio de *bits* ocupado por cada ponto nos arquivos. Esta média é feita dividindo-se o tamanho ocupado por todos os pontos (sem compressão) pelo número total de pontos. O valor obtido por esta divisão é aproximadamente o número de *bits* por ponto.

	4096 x 4096					
	África	Madison	Anchorage	BR Municípios	Br Vegetação	BR Indígena
Pontos (Kb)	44	1214	424	263	344	163
Núm. Pontos	95763	658741	221880	383925	188622	129693
Bits / Ponto	3.8	15.1	15.7	5.6	14.9	10.3

Tabela 6.12: TWF Proposta 2 - Número médio de *bits* por ponto na resolução 4096×4096

	2048 x 2048					
	África	Madison	Anchorage	BR Municípios	Br Vegetação	BR Indígena
Pontos (Kb)	26	979	372	162	255	123
Núm. Pontos	50161	632607	217877	231710	178032	101746
Bits / Ponto	4.2	12.7	14.0	5.7	11.7	9.9

Tabela 6.13: TWF Proposta 2- Número médio de *bits* por ponto na resolução 2048×2048

	1024 x 1024					
	África	Madison	Anchorage	BR Municípios	Br Vegetação	BR Indígena
Pontos (Kb)	15	581	236	102	99	66
Núm. Pontos	26657	551159	200114	133756	147996	77943
Bits / Ponto	4.6	8.6	9.7	6.2	5.5	6.9

Tabela 6.14: TWF Proposta 2 - Número médio de *bits* por ponto na resolução 1024×1024

	512 x 512					
	África	Madison	Anchorage	BR Municípios	Br Vegetação	BR Indígena
Pontos (Kb)	11	251	102	79	50	29
Núm. Pontos	14387	402919	154369	78840	93571	52490
Bits / Ponto	6.3	5.1	5.4	8.2	4.4	4.5

Tabela 6.15: TWF Proposta 2 - Número médio de *bits* por ponto na resolução 512×512

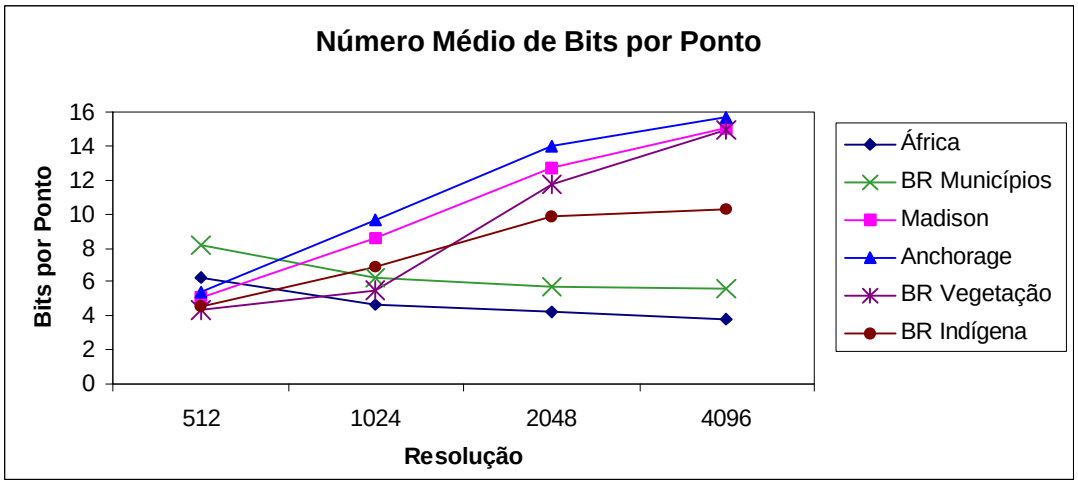


Gráfico 6.13: TWF Proposta 2 - Número médio de *bits* por ponto

Os arquivos África e BR Municípios apresentaram uma diminuição no número médio de *bits* por ponto com o aumento da resolução (Gráfico 6.13), porque houve um maior número de pontos relativos por direção quantizada com o aumento da resolução (Gráfico 6.11). Para os demais arquivos, o número médio de *bits* por ponto aumenta com a resolução, sendo que o maior ponto armazenado ocupa menos de 2 *bytes*.

6.3 Compressão

Na compressão dos arquivos, foi utilizado o algoritmo LZO1X_1 da biblioteca LZO. Esta biblioteca possui outros algoritmos e a seguir serão apresentadas tabelas

com os arquivos resultantes da etapa de codificação da proposta 2 do TWF comprimidos com vários algoritmos da LZO, a fim de fazer uma comparação entre o tamanho dos arquivos em *Kbytes*. Também são apresentados dois gráficos que confirmam que não há algoritmo que seja ótimo em todos os casos. Outra observação a ser feita é que os arquivos não sofrem uma grande compressão, o que indica que a codificação foi bem sucedida, conseguindo eliminar muitas redundâncias. Nas tabelas e nos gráficos seguintes, 1A, 1B_1, 1B_5, 1B_9, 1C_1, 1C_5, 1C_9, 1F_1, 1X_1, 1X_1_1 e 1Y_1 correspondem aos algoritmos LZO1A, LZO1B_1, LZO1B_5, LZO1B_9, LZO1C_1, LZO1C_5, LZO1C_9, LZO1F_1, LZO1X_1, LZO1X_1_1 e LZO1Y_1, respectivamente.

	4096 x 4096										
Arquivo (Kb)	1A	1B_1	1B_5	1B_9	1C_1	1C_5	1C_9	1F_1	1X_1	1X_1_11	1Y_1
África	45	45	44	44	45	44	44	45	45	46	46
Madison	1077	1064	1021	1023	1063	1014	1012	1094	1080	1130	1119
Anchorage	385	380	366	367	380	364	363	389	387	403	400
BR Municípios	294	292	288	286	292	287	285	294	294	301	300
BR Vegetação	279	275	259	257	274	257	254	288	277	288	287
BR Indígena	143	109	104	94	139	131	130	141	118	137	120

Tabela 6.16: TWF Proposta 2 – Compressão por vários algoritmos na resolução 4096×4096

	2048 x 2048										
Arquivo (Kb)	1A	1B_1	1B_5	1B_9	1C_1	1C_5	1C_9	1F_1	1X_1	1X_1_11	1Y_1
África	28	28	28	28	28	28	28	28	28	28	28
Madison	835	821	777	776	821	774	767	858	826	874	862
Anchorage	308	303	286	285	303	285	282	317	305	321	318
BR Municípios	184	183	180	179	183	180	178	183	182	187	186
BR Vegetação	212	209	197	194	208	195	191	215	208	216	214
BR Indígena	110	83	78	70	106	100	99	109	89	97	91

Tabela 6.17: TWF Proposta 2 – Compressão por vários algoritmos na resolução 2048×2048

	1024 x 1024										
Arquivo (Kb)	1A	1B_1	1B_5	1B_9	1C_1	1C_5	1C_9	1F_1	1X_1	1X_1_11	1Y_1
África	19	19	18	18	19	18	18	18	18	18	18
Madison	549	539	514	514	539	512	509	557	541	571	561
Anchorage	214	210	199	199	210	199	197	219	211	222	219
BR Municípios	129	128	126	126	128	126	125	124	123	127	126
BR Vegetação	111	110	108	107	109	107	106	110	111	113	110
BR Indígena	71	51	48	47	59	56	56	60	49	56	49

Tabela 6.18: TWF Proposta 2 – Compressão por vários algoritmos na resolução 1024×1024

	512 x 512										
Arquivo (Kb)	1A	1B_1	1B_5	1B_9	1C_1	1C_5	1C_9	1F_1	1X_1	1X_1_11	1Y_1
África	14	14	13	13	14	13	13	13	13	13	13
Madison	264	262	255	255	261	254	254	263	263	271	269
Anchorage	107	106	103	103	106	103	102	107	106	110	109
BR Municípios	101	100	99	98	100	98	97	93	91	94	93
BR Vegetação	63	63	62	62	63	62	61	63	63	64	62
BR Indígena	37	32	32	31	35	35	34	36	31	33	31

Tabela 6.19: TWF Proposta 2 – Compressão por vários algoritmos na resolução 512×512

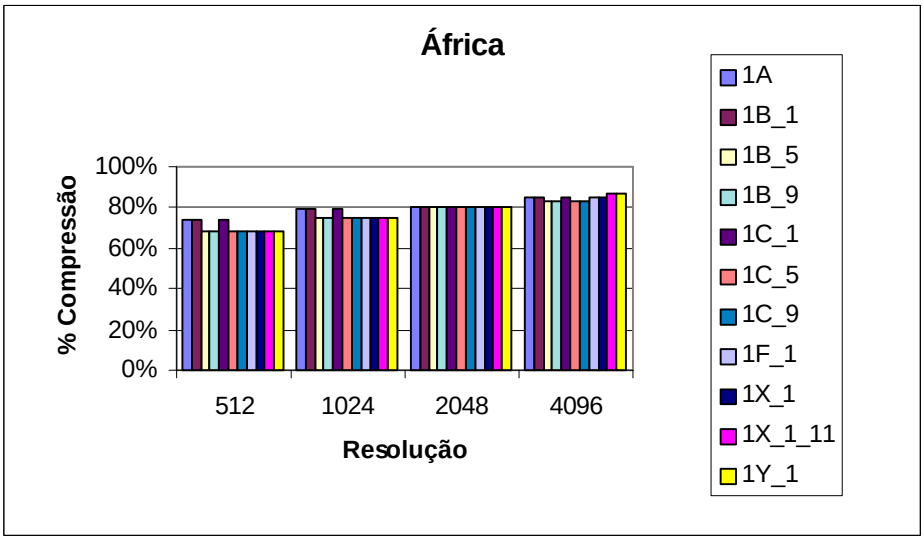


Gráfico 6.14: TWF Proposta 2 – Compressão por vários algoritmos do arquivo África

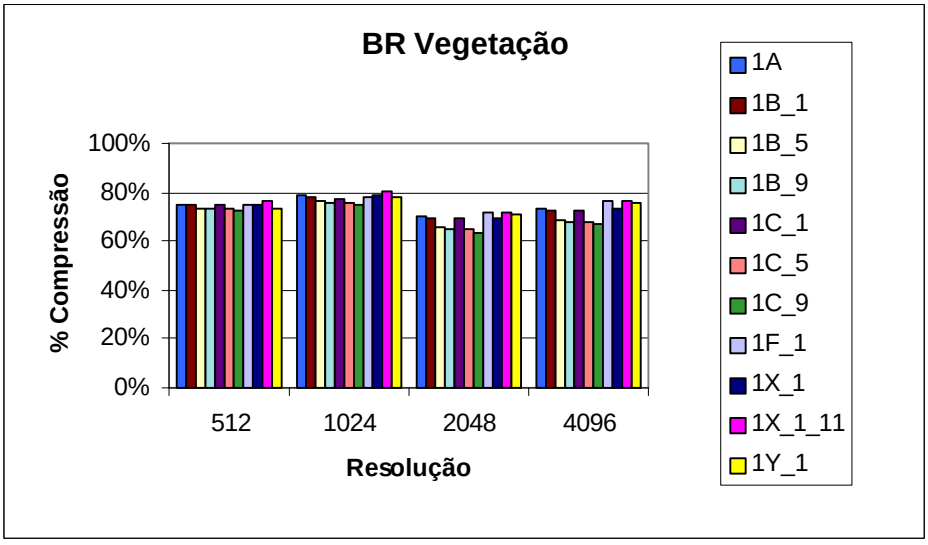


Gráfico 6.15: TWF Proposta 2 – Compressão por vários algoritmos do arquivo BR Vegetação

7. Conclusão

Atualmente, na visualização de mapas na *web*, é comum utilizar arquivos de imagem de baixa resolução, como os arquivos GIF. Esses mapas visualizados através de imagens não podem ser escalados para adaptar-se a diferentes densidades de monitor e de impressora, nem podem sofrer *zoom* ou suportar a seleção individual de entidades, como rodovias e rios.

Gráficos vetoriais podem ser escalados e satisfazem a muitos dos requisitos dos mapas, mas, geralmente requerem arquivos muito grandes. Os formatos de arquivos vetoriais existentes necessitam de pelo menos 1 *byte* para armazenar um ponto. Mapas, normalmente, possuem milhares de pontos e esquemas de compressão sem perda não apresentam um fator de compressão maior do que três. Desse modo, os arquivos criados têm tamanho igual a muitas centenas de *Kbytes*, que são grandes demais para trafegar na *web*.

Técnicas cartográficas, como simplificações [10], podem ser utilizadas para reduzir o número de pontos de um mapa, mas não respondem à questão de como codificar esses pontos de uma forma compacta. Nesse trabalho, foram apresentadas duas técnicas de codificação: uma sem cadeias de *bits* e outra com cadeias de *bits*. A última é sempre melhor que a padrão e aconselha-se a utilização da codificação com cadeias de *bits* na proposta final do TWF. Se as poligonais são simplificadas, no sentido cartográfico, para serem exibidas em determinada resolução, é como se seus vértices pudessem ficar em *pixels* adjacentes. Poligonais com *pixels* adjacentes são representadas de modo mais compacto através de uma cadeia de *bits* e não através de coordenadas absolutas ou relativas por incremento.

Um ponto importante no esquema proposto neste trabalho é o mapeamento das coordenadas reais em um subconjunto do espaço dos inteiros positivos. Isto não reduz somente o tamanho da representação computacional dos pontos, mas também estabelece uma base para o conceito de vizinhança utilizado na codificação por cadeia de *bits*. As coordenadas máximas *x* e *y* também são um modo conveniente para controlar a perda na precisão e o tamanho do arquivo final.

Neste trabalho, foi apresentada uma técnica de codificação vetorial que cria arquivos de mapa pequenos com uma resolução adequada para apresentação. É importante notar que essa codificação de dados tem perda de precisão e talvez não seja boa para computações cartográficas. O objetivo é somente a apresentação através da *web*.

A comparação com os arquivos GIF, apresentada no capítulo 6, indica que a técnica proposta é boa para resoluções de apresentação por volta de 4096×4096. Isto é, se a resolução é pequena, por exemplo 512×512, o arquivo GIF é definitivamente menor. Por outro lado, para resoluções muito grandes (32768×32768), quase nenhum segmento de poligonal pode ser codificado como uma cadeia de *bits* e a estratégia proposta perderá sua principal vantagem. Contudo, para a resolução de apresentação atual (4096×4096), a estratégia proposta produz gráficos que podem ser escalados com tamanho reduzido.

O foco deste trabalho é a codificação compacta de poligonais longas. Foram utilizados os conceitos de atributos correntes e prioridade implícita do pacote gráfico *Canvas Draw*, que parece ter produzido bons resultados. Há, no entanto, outras questões que devem ser observadas, como:

(a) Inclusão de dados *raster*.

(b) Inclusão de *hyperlinks* com agrupamento de elementos em estruturas semânticas.

(c) Textura e sobreposição com o plano alfa.

(d) Criação de camadas (*layers*), *stencil* e *mask*.

(e) Controle de estilo de junção e fim de linha.

(f) Exibição progressiva.

Acredito também que o TWF pode ser baseado no modelo *Canvas Draw*, mas

deve superar e ficar independente deste, a fim de se tornar realmente proveitoso. Por exemplo, é possível fazer algo parecido com o *Open GL 2D*.

Deve ser realizado um estudo melhor sobre a compressão, para verificar se é possível obter um algoritmo melhor, talvez adaptativo aos dados de entrada. Além disso, a etapa de quantização deverá aumentar e incluir a simplificação cartográfica dos dados, a fim de tornar o formato mais genérico, eliminando o requisito que estabelece a simplificação prévia dos dados.

A. Descrição dos Formatos de Arquivos e Figuras

A.1 Enhanced Metafile

O formato binário EMF (*Enhanced Metafile*) [22] possui algumas melhoras em relação ao antigo *Windows Metafile* (WMF), como: um cabeçalho expandido, uma descrição, uma *palette* e um aumento no número e nos tipos de funções de dispositivos de interface gráfica (GDI - *Graphics Device Interface*) que podem ser gravadas.

Esse formato é independente de plataforma e consiste em um cabeçalho, uma descrição, uma *palette* e um vetor de registros (Figura A.1).

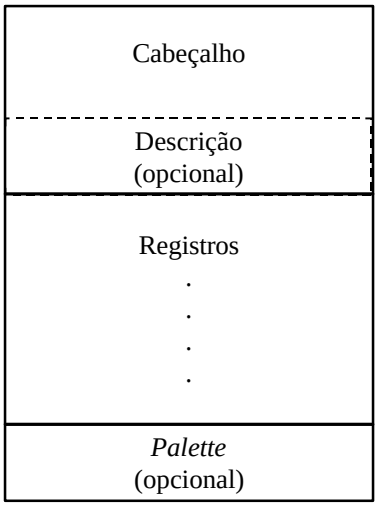


Figura A.1: Organização de um arquivo EMF

O cabeçalho do EMF contém informações de tamanho, versão, dimensão e resolução. O tamanho da descrição, caso a mesma exista, e o número de entradas na *palette* também são colocados no cabeçalho.

A descrição informa o que há no arquivo e onde esse foi criado, ou seja, informa o objetivo do arquivo (nome da figura) e o nome da aplicação que criou a figura. Deve ser colocado um caracter nulo entre o nome da figura e o nome da aplicação e dois caracteres nulos no fim da descrição.

Um registro do EMF é composto por um campo tipo, um campo tamanho e um vetor de parâmetros, sendo que cada campo, bem como cada um dos componentes do vetor, ocupa 32 *bits*. Nas poligonais, há a opção de converter os pontos para 16 *bits*, para diminuir o tamanho final do arquivo.

A.2 Computer Graphics Metafile

O padrão ANSI CGM (*Computer Graphics Metafile*) [5, 6], especifica que elementos podem estar em quais posições do arquivo. A estrutura de um meta-arquivo

pode ser vista na Figura A.2, onde o meta-arquivo é considerado uma série de níveis de abstração.

O conjunto básico de elementos de um meta-arquivo possui a capacidade de adição de dados dependentes da aplicação, que não têm significado gráfico e para os quais não se espera que os resultados de interpretação sejam descritos. Para satisfazer a esses requisitos, o CGM foi dividido em duas funções principais. Primeiro, são definidos os componentes individuais, chamados *elementos* (Figura A.3), que precisam estar no meta-arquivo e seguem algumas regras sobre o modo como esses elementos são combinados em um meta-arquivo válido. Depois, é definido o modo como esses elementos são gravados no meta-arquivo.

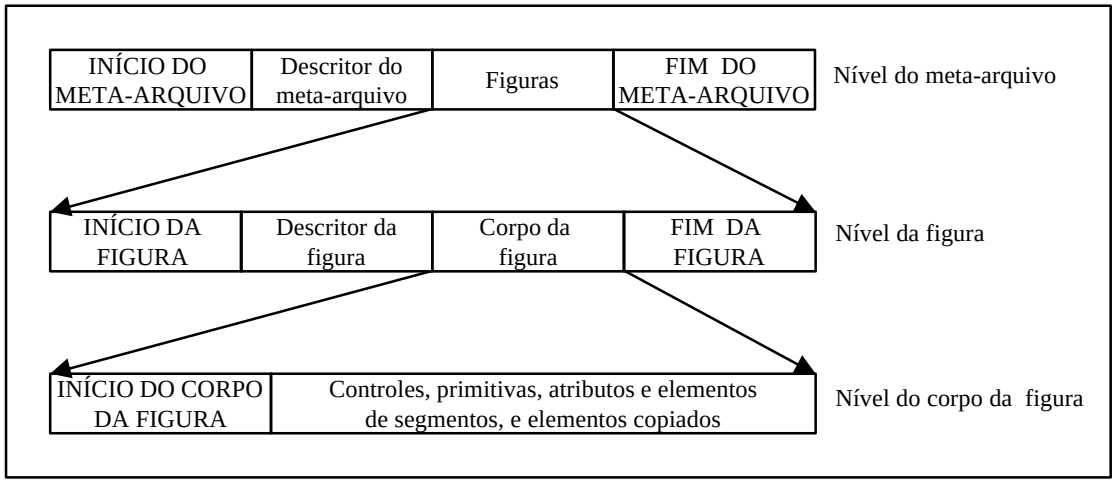


Figura A.2: Estrutura de um meta-arquivo

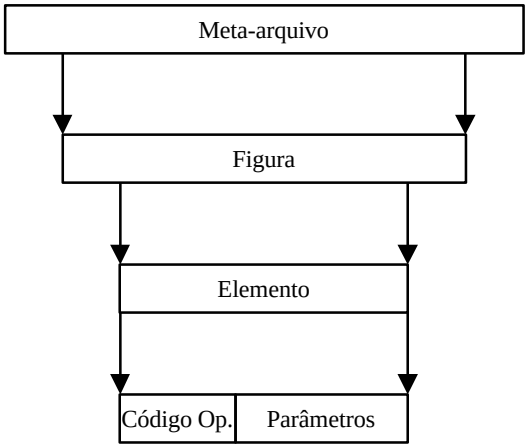


Figura A.3: Um *meta-arquivo* em diferentes níveis de detalhe

A primeira regra é conhecida como *descrição funcional* do meta-arquivo e nessa parte são descritos os elementos que estão incluídos no padrão. Esses elementos incluem elementos de desenho, como poligonal e texto. Também há informações sobre a aparência das primitivas, os chamados *atributos*, como cor de linha e tamanho de texto.

A descrição funcional é insuficiente sem o método de armazenamento dos elementos, ou seja, sem a *codificação* dos elementos. No CGM, há três formas diferentes de codificação: a por caracter, a binária e a *clear text*. O principal objetivo da codificação por caracter é assegurar que a codificação seja compacta e garantir uma transferência fácil de dados através das redes. A ênfase da codificação binária é a fácil criação e interpretação dos arquivos CGM. Na codificação *clear text*, os dados armazenados são legíveis, o que permite a edição dos arquivos.

O padrão CGM define um meta-arquivo que é completamente especificado e consistente com a filosofia de definições de figuras estáticas de armazenamento. A definição de meta-arquivo será analisada como uma série de camadas de detalhes. Essas camadas são: meta-arquivo, figuras e elementos, sendo que um elemento é composto por um código de operação e por seus parâmetros (Figura A.3).

No nível mais alto da estrutura está o meta-arquivo. Cada meta-arquivo está contido entre o “Início do meta-arquivo” e “Fim do meta-arquivo” (Figura A.2). No início de cada meta-arquivo está um “Descritor do meta-arquivo”, efetivamente um cabeçalho do meta-arquivo. Este descritor contém informações relevantes para todo o meta-arquivo. Essas informações são necessárias para que o interpretador seja capaz de entender o meta-arquivo e incluem: o número da versão e alguma descrição como data e nome do autor; uma lista dos elementos do meta-arquivo de modo que o interpretador possa decidir no início se o meta-arquivo pode ser interpretado; etc.

Cada meta-arquivo contém pelo menos uma figura. Essas figuras são quase totalmente auto-suficientes, isto é, só precisam das informações definidas nelas mesmas e no “Descritor do meta-arquivo” para serem interpretadas. Esta independência das figuras faz com que elas possam ser interpretadas tanto seqüencialmente quanto aleatoriamente. A figura também possui um “Descritor da figura”, ou seja, um cabeçalho. Dentre outras, esse descritor tem as seguintes funções: declarar os modos dos parâmetros para alguns elementos na figura (cores indexadas, por exemplo); definir as coordenadas da *window* a ser utilizada; definir a cor de fundo; etc. Cada figura é uma representação estática; logo qualquer elemento com efeitos dinâmicos é definido fora da figura.

Cada um dos itens a serem armazenados em um meta-arquivo são guardados como um *elemento*. Elementos podem ter várias funções diferentes. O elemento é o nível mais baixo de informações úteis do meta-arquivo. Cada elemento pode ter uma lista de dados que serão chamados de parâmetros. Cada elemento também possui um código de operação para identificá-lo. Esse código é seguido pelos parâmetros relativos ao elemento corrente.

O código de operação é um par de inteiros na codificação binária, uma palavra com significado na codificação *clear text*, e um ou dois códigos de caracter na codificação por caracter. Os parâmetros da especificação funcional são codificados para cada uma dessas três codificações. Os parâmetros codificados são colocados após o código de operação para formar um elemento codificado completo.

Algumas codificações são limitadas por si mesmas, pois muitos elementos CGM possuem um número fixo de parâmetros. Outros elementos podem ter uma quantidade variável de dados, como uma poligonal. Uma codificação que é limitada por si mesma, utiliza algum padrão ou indicação especial para designar o fim de um elemento. O designador do final de um elemento é chamado *terminador*.

Na codificação binária, qualquer padrão pode ocorrer como dado válido. Desse modo, não é possível designar um terminador para o fim de cada elemento. Por esta razão, a codificação binária possui um pedaço extra de informação logo após cada código de operação, um contador da quantidade de dados em cada elemento codificado. Este contador é o elemento de tamanho. A codificação binária é freqüentemente referenciada como a codificação de manipulação do tamanho.

O código de operação de um elemento codificado é chamado de *cabeçalho do comando* na codificação binária, e o elemento codificado é chamado de *comando*. Há duas maneiras de codificar elementos na codificação binária: a forma *pequena* e a forma *longa*. Comandos na forma pequena (Figura A.4) podem codificar até 30 *bytes* de dados de parâmetros. Há apenas uma única partição de dados seguindo o único cabeçalho do comando.

Comandos na forma longa (Figura A.5) podem codificar uma quantidade ilimitada de dados. Esse tipo de comando contém uma ou mais partições de dados. Cada partição de dados possui o seu próprio campo de tamanho, que conta o número de *bytes* de dados na partição e indica se há ou não mais partições. As partições podem variar de tamanho de 0 a 32767 *bytes* de dados codificados.

Os primeiros 16 *bits* de qualquer comando distinguem se esse está na forma pequena ou longa. Estes *bits* são compostos por três campos. Da posição mais significativa para menos significativa, temos:

- um campo de classe de elemento de 4 *bits* (*bits* 15-12);
- um campo de identificação de elemento de 7 *bits* (*bits* 11-5);
- um campo de tamanho de 5 *bits* (*bits* 4-0).

Se todos os *bits* possuírem valor 1 no campo de tamanho (valor inteiro 31), então o comando está na forma longa. Caso contrário, é um comando de forma pequena e seus 16 *bits* iniciais compreendem todo o cabeçalho de comando. O campo de tamanho conta os dados de parâmetros (de 0 a 30 *bytes*), e o primeiro *byte* dos dados de parâmetros começa imediatamente após o cabeçalho do comando (Figura A.4).

Se o comando está na forma longa (Figura A.5), então a primeira partição de dados está imediatamente após os primeiros 16 *bits* do comando. Cada partição de dados começa com 16 *bits* de controle de partição, que consiste em:

- um indicador (*bit* 15) de partição de 1 *bit* (continuação);
- um lista de dados de 15 *bits* (*bits* 14-0).

O *bit* mais significativo dos 16 *bits* é um *bit* de continuação. Quando seu valor é 1, significa que há mais dados após a partição corrente.

Na codificação binária, as cores do CGM podem ser indexadas ou diretas. Quando são indexadas, as cores são representadas com um inteiro sem sinal, de 8, 16, 14 ou 32 *bits*, sendo que o padrão é 8 *bits*. Quando as cores são diretas, podem ser codificadas como três ou quatro inteiros sem sinal, depende do modelo de cores. As precisões válidas são de 8, 16, 14 ou 32 *bits*, sendo que o padrão também é 8 *bits*.

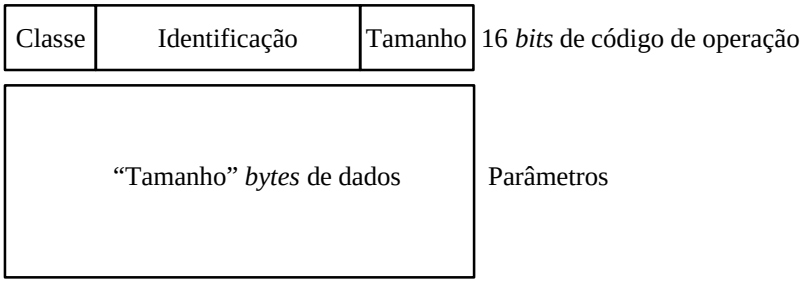


Figura A.4: Comando na forma pequena

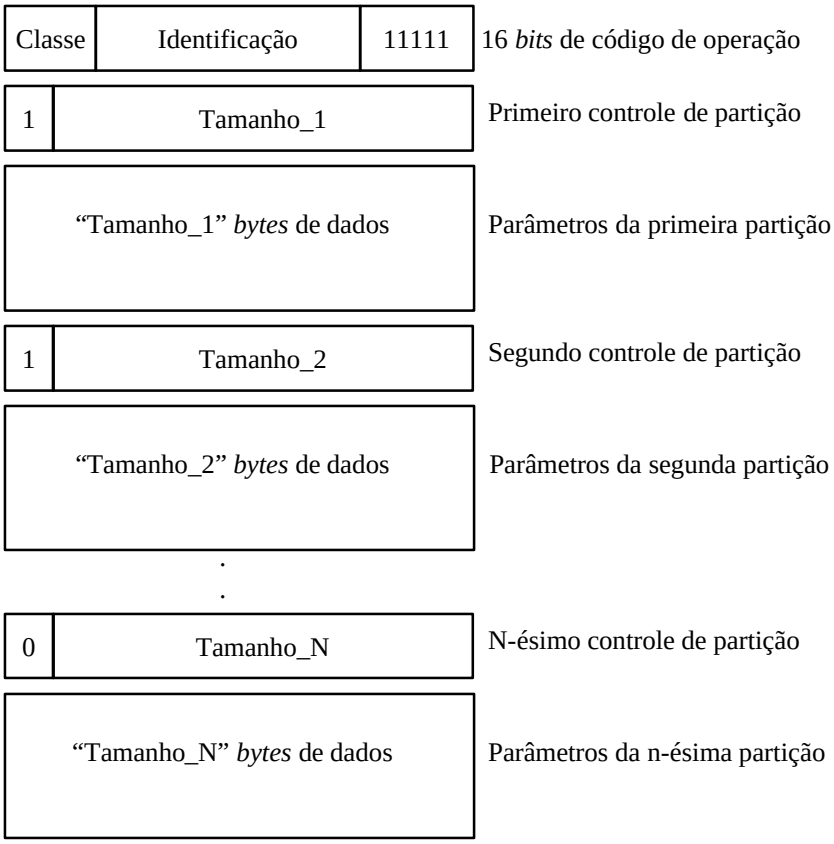


Figura A.5: Comando na forma longa

Os dados podem ser representados por inteiros de 8, 16, 24 e 32 *bits*, por reais de ponto fixo de 32 ou 64 *bits*, ou por números reais de ponto flutuante de 32 ou 64 *bits*. O valor padrão para parâmetros inteiros é 16 *bits* e para números reais é ponto fixo de 32 *bits*.

Há subconjuntos do formato CGM utilizados em áreas industriais diferentes, principalmente nas indústrias de aeronaves (ATA - *Airline Transport Association* [42]), de petróleo (PIP - *Petroleum Industry Profile* [44]) e de defesa (CALS – *Computer-Aided Logistics Support* [43]). Esses subconjuntos limitam as entidades usadas no arquivo, a fim de assegurar a compatibilidade entre os programas. Uma das limitações pode ser a representação das coordenadas, utilizando-se somente uma parte das possibilidades apresentadas.

A.3 CD Metafile

O formato CDM (CD *Metafile*) [34], é criado utilizando-se a biblioteca gráfica bidimensional CD – *Canvas Draw* – [33] desenvolvida pelo TeCGraf – Grupo de Tecnologia em Computação Gráfica da PUC-Rio [32].

O CD *Metafile* contém primitivas do sistema gráfico *Canvas Draw*, ou seja, ele armazena uma sessão de desenho e não objetos estruturados. Os conceitos que regem o CD *Metafile* são os mesmos do CD. Assim, por exemplo, a geometria das linhas e polígonos preenchidos é definida por funções chamadas de primitivas. As aparências destes objetos, como estilo de linha ou cor, são definidas pelo conceito de atributo global corrente.

Este formato só é possível, porque o CD possui um conceito abstrato de superfície de visualização virtual, VVS (*Virtual Visualization Surface*), inspirado no GKS [7, 20] e PHIGS [21]. A biblioteca CD fornece serviços para desenhos em janelas, *buffers, clipboards,* gerenciadores de impressão e meta-arquivos. Todos estes locais de desenho são considerados VVS.

Existem duas funções que controlam o fluxo de primitivas: uma de criação de uma VVS e outra de ativação da mesma. A função de criação tem como primeiro parâmetro um ponteiro para o controlador (*driver*) da superfície de visualização em questão (janela, *offscreen buffer, clipboard*, etc.) e retorna um ponteiro para uma VVS. Toda referência a esta superfície passa a ser feita por este ponteiro, reduzindo ao máximo a dependência do código da aplicação em relação ao dispositivo específico onde o desenho está sendo feito. A função de ativação faz um papel de distribuidor das primitivas para as diversas VVS. Quando uma superfície de visualização é ativada, a anterior é desativada e todas as primitivas passam a ser "desenhadas" na nova VVS.

Assim, este arquivo contém todos os comandos do CD em que a aplicação "desenhou" na VVS CD *Metafile*. Uma função do CD, chamada *cdPlay*¸ lê este arquivo e "desenha" os comandos do CD do arquivo na VVS que estiver corrente. O CD também tem controladores para outros formatos de arquivo, como o CGM, logo a biblioteca CD permite a conversão de formatos de arquivos. Basta que a aplicação dê um *cdPlay* em um arquivo de determinado formato com a VVS sendo outro arquivo de diferente formato.

Um ponto delicado do processo de armazenamento de comandos em arquivo é o sistema de coordenadas. As coordenadas no CD e, conseqüentemente, no CD *Metafile*, são valores inteiros. A VVS é suposta ser retangular com um sistema de coordenadas no canto inferior esquerdo como ilustra a Figura A.6.

A função que indica a criação de um CD *Metafile* tem como parâmetros o nome do arquivo e, opcionalmente, a largura, a altura e a resolução da VVS, que são números reais. A largura e a altura correspondem ao tamanho da VVS em milímetros e a resolução é o número de *pixels* por milímetro. Caso não sejam fornecidos, a largura e a altura recebem o valor do maior inteiro possível e a resolução recebe valor 1.

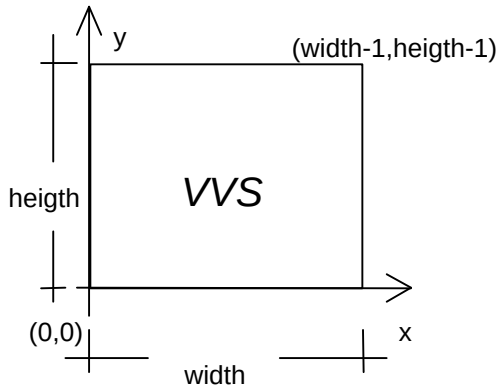


Figura A.6: Sistemas de coordenadas do *Canvas Draw*

No arquivo CD *Metafile*, só são gravados os valores *width* e *height* mostrados na Figura A.6. Esses valores são obtidos truncando-se o produto dos valores de largura e altura em milímetros pela resolução de *pixel* por milímetros.

As coordenadas do CD *Metafile* são inteiras. Esse formato é textual, isto é, armazena todos os números e textos das primitivas como textos, incluindo as imagens. Isto aumenta bastante o tamanho final do arquivo. Na Figura A.7, é apresentado o esquema de um arquivo no formato CD *Metafile*, onde *Id* é o identificador do formato, *width* e *height* são a largura e a altura da VVS, respectivamente, e Função corresponde a uma função do CD.

Na primeira linha do arquivo, estão o identificador do formato, que é o texto "CDMF", e a largura e altura da VVS em *pixels*. Cada linha seguinte corresponde a uma função do CD, composta pelo seu índice e seus dados. O índice identifica qual função será analisada e, conseqüentemente, se há dados a serem obtidos. As funções do CD com seus respectivos dados estão listadas em [33].

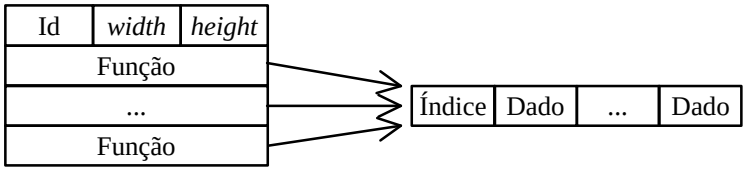


Figura A.7: Organização de um arquivo CD *Metafile*

A largura e a altura da VVS e os índices das funções são números inteiros, ou seja, cada um desses dados ocupa 4 *bytes*. Cada função possui quantidades e tipos de dados diferentes. Uma poligonal possui uma função que indica o seu início (*cdBegin*) e outra que indica o seu fim (*cdEnd*). Entre essas marcações de início e fim, estão as funções relativas aos pontos da poligonal (*cdVertex*). As coordenadas do CD *Metafile* são vetoriais.

A função *cdBegin* tem um parâmetro que indica o modo da poligonal. O modo pode ser CD_CLOSED_LINES, CD_OPEN_LINES, CD_FILL ou CD_CLIP. O modo CD_CLOSED_LINES conecta o último ponto da poligonal com o primeiro ponto da mesma. O CD_FILL conecta o último ponto da poligonal com o primeiro

ponto da mesma e preenche o polígono resultante de acordo com o estilo interior corrente. O CD_CLIP não cria um polígono a ser desenhado, cria um polígono para definir uma região de *clipping* não retangular. A marcação de fim de poligonal é formada apenas pelo índice correspondente à função *cdEnd*.

As coordenadas dos pontos da poligonal são valores inteiros, logo, após cada índice relativo à função *cdVertex*, obtém-se as duas coordenadas inteiras do ponto corrente da poligonal.

A.4 PLOT

Os arquivos PLOT [23] são arquivos de dados reduzidos para coordenadas cartesianas. Esses arquivos contêm dois tipos básicos de informação: vetorial e de atributos. As informações vetoriais são as linhas e as informações de características possuem as localizações e as dimensões.

Um arquivo gráfico é um conjunto de instruções de desenho. Cada instrução é composta por uma letra ASCII, usualmente seguida por *bytes* de informação binária. As instruções são executadas em ordem. Cada ponto ocupa 4 *bytes*, sendo que cada coordenada, *x* e *y*, é um inteiro com sinal. O último ponto é o ponto corrente para próxima instrução, ou seja, caso a próxima função precise de um ponto de referência, esse sempre será o último ponto analisado.

Algumas primitivas do PLOT são o arco e o círculo. No caso do arco, a letra "a" identifica a instrução, os primeiros 4 *bytes* são o centro do arco, os próximos 4 *bytes* indicam o ponto inicial e os últimos 4 *bytes* são o ponto final do arco. A coordenada menos significativa do ponto final é utilizada somente para determinar o quadrante. O arco é desenhado no sentido horário. A letra "c" indica o círculo. Os primeiros 4 *bytes* indicam o centro do círculo e os próximos 2 *bytes*, o raio do mesmo.

A.5 Drawing Exchange Format

O formato DXF (*Drawing Exchange Format*) [3, 27] da *Autodesk* é uma descrição textual de um desenho, possibilitando a troca de arquivos entre um sistema de CAD e outro programa.

Os dados em um arquivo DXF são vetoriais e a precisão dos números reais armazenados pode ser definida pelo usuário, sendo que são permitidas no máximo 16 casas decimais. É possível ter no máximo 256 cores indexadas e não há compressão dos dados.

Esse formato é flexível, pois possui opções na sua criação. Uma opção é gravar somente os objetos selecionados pelo usuário; outra é gravar os objetos e as entidades. Nenhuma dessas opções grava o estado do arquivo de desenho, que é armazenado no cabeçalho. O cabeçalho inclui todas as tabelas e as definições de variáveis do sistema. Na maioria dos casos, o arquivo é gravado completo, isto é, com todas as informações disponíveis.

Um arquivo DXF completo possui todos os componentes de um arquivo de desenho, incluindo definições de bloco (símbolos), informação de camadas, estilos de texto, estilos de dimensão, vistas nomeadas, sistemas de coordenadas nomeados e

configurações de *viewport* nomeadas. Um arquivo parcial é utilizado quando o importante é analisar apenas a geometria e as entidades.

Um arquivo DXF é organizado em seções: cabeçalho, classes, tabelas, blocos, entidades e objetos (Figura A.8). A seção de cabeçalho contém informações gerais sobre o desenho, que são o número da versão do banco de dados do *AutoCAD* e o número de variáveis de sistema. Cada parâmetro possui um nome de variável e o valor associado.

A seção de classes armazena as informações de classes definidas da aplicação, cujas instâncias aparecem nas seções de blocos, de entidades e de objetos do banco de dados. Uma definição de classe é permanentemente fixa na hierarquia de classes.

A seção de tabelas contém as definições das seguintes tabelas de símbolos:

- Tabela de tipos de linha (LTYPE).
- Tabela de camadas (LAYER).
- Tabela de estilos de texto (STYLE).
- Tabela de vistas (VIEW).
- Tabela do sistema de coordenadas do usuário (UCS).
- Tabela de configuração da *viewport* (VPORT).
- Tabela de estilos de dimensão (DIMSTYLE).
- Tabela de identificação da aplicação (APPID).
- Tabela de referência de bloco (BLOCK_REFERENCE).

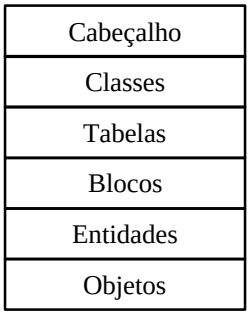


Figura A.8: Organização de um arquivo DXF

A seção de blocos é composta pela definição dos blocos e das entidades de desenho que compõem cada bloco referenciado no desenho. A seção de entidades possui os objetos gráficos (entidades) do desenho, incluindo referências de bloco (entidades inseridas).

Os objetos não gráficos do desenho, ou seja, todos os objetos que não são entidades nem tabela de símbolos, são definidos na seção de objetos. Exemplos da seção de objetos são os dicionários que contém estilos *mline* e grupos.

Um arquivo DXF é formado por muitos grupos, cada um dos quais ocupa duas linhas do arquivo. A primeira linha é o código do grupo e a segunda é o valor do mesmo, em um formato que depende do tipo do grupo especificado pelo seu código.

A.6 Drawing Web Format

O DWF (*Drawing Web Format*) [28, 29, 37, 38], é proposto como padrão pela *Autodesk* para disponibilizar os desenhos do *AutoCAD* (DXF, DWG) na *web*. O propósito é criar vistas eficientes de desenhos, similares a gráficos eletrônicos. Os componentes não visuais, como atributos, propriedades e comportamento de objetos complexos, são removidos do desenho. Devido à falta de dados não visuais, o DWF não foi desenvolvido para ser lido na aplicação de CAD original onde foi criado, embora um arquivo DWF possa fazer referência a outro arquivo no formato nativo da aplicação através de um *link* ou de uma operação implícita.

Os arquivos DWF são organizados em três partes principais (Figura A.9): o cabeçalho de identificação, o bloco de dados e uma marcação de fim de arquivo.

Os dados no cabeçalho e na marcação de fim de arquivo são codificados como textos legíveis. O conteúdo do bloco de dados é delimitado por operadores e pelos argumentos utilizados por esses operadores, os operandos (Figura A.10).

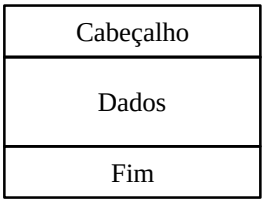


Figura A.9: Organização de um arquivo DWF

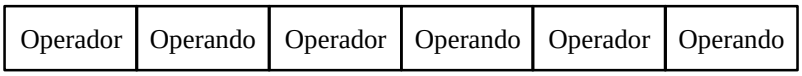


Figura A.10: Bloco de dados do arquivo DWF

O cabeçalho tem duas funções básicas: permitir a fácil identificação do arquivo DWF e identificar qual a revisão da especificação DWF foi utilizada na codificação do arquivo. Um exemplo de cabeçalho que ocupa 12 *bytes* é "(DWF V00.34)". Uma aplicação que gere um formato DWF deve especificar o menor número de revisão possível que uma aplicação que leia o formato deva precisar para usar os dados apropriadamente.

A marcação de fim de arquivo é simplesmente um operador especial que indica o final do arquivo seqüencial de dados DWF. Uma aplicação pode armazenar dados não relativos ao formato DWF depois do operador de término. Este operador, na codificação ASCII, é “(EndOfDWF)” e ocupa 9 *bytes*.

No bloco de dados, existem dois tipos de pares operadores/operandos: texto legível e código binário. Todas as operações no DWF têm uma forma

operador/operando legível, e a maioria das operações também possui uma forma operador/operando com codificação binária. Utilizando a forma de operador apropriada, é possível criar um arquivo legível ou um arquivo que seja mais eficiente do ponto de vista de processamento e de armazenamento. Normalmente, o arquivo possui uma mistura dos dois tipos de pares operador/operando.

Uma aplicação lendo um DWF pode não entender um conjunto de operadores, principalmente quando a aplicação que está lendo o arquivo é mais antiga que a aplicação que criou o mesmo. Por isso, o DWF é desenvolvido de modo a permitir que o leitor do arquivo ignore a maioria dos operadores. Para que seja possível ignorar um operador, deve-se saber o tamanho do seu operando. O DWF possui três categorias de operadores:

(a) Operadores de um *byte*: que devem ser reconhecidos por razões de eficiência e não podem ser ignorados. Uma aplicação que leia um DWF não precisa implementar esses operadores, mas tem que ser capaz de computar o tamanho de seu operando; portanto o operador deve ser reconhecido. Caso um operador de um *byte* não seja reconhecido, o restante do arquivo não pode ser lido.

(b) Operadores textuais estendidos (legíveis) que são delimitados e aninhados. Uma aplicação que leia um DWF pode ignorar este par operador/operando sem entender a operação ou seus conteúdos.

(c) Operadores binários estendidos que indicam o tamanho de seus operandos, de modo que a aplicação que lê o DWF pode facilmente ignorar as operações e os dados desconhecidos.

Para melhorar a legibilidade dos arquivos DWF, um operador pode ser precedido por qualquer quantidade de caracteres do tipo espaço, tabulação, retorno de linha ou fim de linha. Quando atribui-se um operador a uma operação, deve-se seguir as seguintes regras:

- Por padrão, uma nova operação é atribuída a um operador textual estendido. Sendo este legível, pode ser facilmente ignorado por aplicações com versões antigas do DWF, e não utiliza os poucos operadores de um *byte*. Se o operando desta operação representa uma informação binária, essa informação deve ser convertida para uma forma textual legível para ser armazenada no arquivo DWF.
- Se o operador desta operação representa uma grande quantidade de informação binária que seria ineficiente converter para uma forma textual legível, então deve ser atribuído um operador binário estendido para a operação.
- Se a operação deve ser utilizada freqüentemente no arquivo DWF e tem um operador relativamente pequeno, esta deve receber um operador de um *byte*.

Por exemplo, o operador de um *byte*, "(", caracter de abre-parênteses, indica que a seguir virá um operador textual legível estendido e possivelmente aninhado. Após o caracter de abre-parênteses, "(", está um operador de símbolos de caracteres de múltiplos *bytes* seguido por espaços em branco, seguidos por zero ou mais operandos, seguidos por um terminador de fecha-parênteses, ")", isto é, "(Origin 240 120)". Operadores textuais estendidos podem ser aninhados: "(Owner (FirstName Brian)(LastName Mathews))".

O operador de um *byte* correspondente ao caracter de abre-chaves, "{", indica que a seguir virá uma seção binária de dados estendidos e possivelmente aninhados. Imediatamente após o caracter de abre-chaves, está um inteiro de 4 *bytes* que representa o tamanho em *bytes* dos dados binários. Seguindo o campo de tamanho, está um operador binário estendido de 2 *bytes*, que permite até 65536 operações. Finalmente, o conjunto binário é terminado com um caracter de fecha-chaves, "}". Por exemplo, dados binários para um *raster* de *pixels* pode ser representado como "{cccceexxxxxxxxx}", onde "cccc" é o tamanho do dado binário, "ee" é o operador para *raster*, e "xxxxxxxxx" é o dado *raster*. O operador binário estendido e o caracter de fim, "}", são considerados parte do conjunto de dados binários. Logo, o valor de "cccc" é 12 (nove "x", dois "e", e um "}") nesse exemplo, que pode ser codificado como um valor binário *little-endian* (*byte* menos significativo no final).

A maioria das coordenadas especificadas em arquivos DWF são coordenadas lógicas, ao contrário das coordenadas da tela ou do dispositivo. Coordenadas lógicas são especificadas no intervalo positivo de inteiros de 32 *bits* com sinal (31 *bits* de precisão) com um intervalo válido de 0 até o máximo de 2147483647 (2^{31}–1). Normalmente, uma aplicação que escreve um DWF deve escalar as primitivas geométricas da ilustração que está sendo armazenada, de modo que uma grande parte do intervalo de 31 *bits* seja utilizado. Isto permite que uma aplicação que interpreta um DWF escale a ilustração para o visualizador desejado ou que um usuário faça *zoom* no desenho com suficiente precisão para ver detalhes com qualidade.

Dependendo do operador em uso, os valores das coordenadas lógicas podem ser codificados como coordenadas absolutas ou como coordenadas relativas. Enquanto as coordenadas lógicas absolutas podem variar somente de 0 a 2147483647 (31 *bits* sem sinal), as coordenadas relativas podem variar de –2147483647 a +2147483,647 (32 *bits* com sinal). Uma coordenada relativa é formada subtraindo-se a coordenada atual da coordenada absoluta anterior.

Coordenadas relativas são utilizadas para aumentar a eficiência do algoritmo de compressão de dados do DWF, que tenta achar padrões repetidos de dados. Desenhos comuns têm objetos representados por seqüências de linhas, círculos, e assim por diante, que podem ocorrer múltiplas vezes. Desse modo, o conjunto de coordenadas relativas será repetido e conseqüentemente a compressão encontrará vários padrões repetidos.

Para muitas aplicações, o maior nível de detalhe permitido pelas coordenadas lógicas de 31 *bits* de um DWF não é necessário e somente aumentará o tamanho do arquivo ao armazenar valores tão grandes. Desse modo, muitas operações de desenho permitem que coordenadas relativas inteiras de 16 *bits* (valores relativos de 16 *bits* com sinal) sejam utilizadas. Sempre que um arquivo DWF é interpretado, tanto com coordenadas relativas de 16 ou 32 *bits*, suas coordenadas são convertidas para coordenadas lógicas absolutas de 31 *bits* antes de serem usadas. Esta é uma forma de compressão sem perda, pois os 31 *bits* de precisão são preservados mesmo quando o valor foi armazenado em 16 *bits*.

O DWF suporta, por exemplo, linhas, poligonais, polígonos, símbolos, imagens, círculos, arcos, elipses, curvas de Bézier, textos, visibilidade, sombreamento de Gouraud, mapeamento de textura, regiões cortadas e transparência variável.

A.7 Simple Vector Format

O SVF (*Simple Vector Format*) [31, 37, 38] foi criado pela *SoftSource* e pela NCSA em 1994. O primeiro *plug-in* para *web* estava disponível em janeiro de 1995.

O SVF foi o primeiro formato gráfico proposto explicitamente para uso na *web* e o primeiro formato vetorial disponível na rede. Este foi desenvolvido para ser um formato simples para descrição de imagens vetoriais. Os objetos de desenho básicos incluem pontos, linhas, círculos, arcos, curvas de Bézier e textos. Características do formato incluem camadas (para o controle da visibilidade relativa dos objetos), *hyperlinks*, notificações (para enviar mensagens quando o usuário passa por um certo nível de *zoom*), preenchimentos e a habilidade de esconder as cores padrão.

O arquivo é dividido em três seções: a *intro*, o cabeçalho e a parte principal (Figura A.11). A *intro* é simplesmente um texto identificando o arquivo como um SVF. O cabeçalho inclui informações gerais úteis para visualização e manipulação da imagem, como camadas, extensões, e cores. A parte principal descreve como desenhar a imagem e os *hyperlinks*. Os objetos são desenhados na ordem em que estes ocorrem no arquivo.

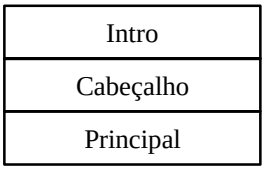


Figura A.11: Organização de um arquivo SVF

Os parâmetros dos comandos, como cores e coordenadas, são números inteiros. A maioria desses parâmetros podem ser de tamanhos diferentes, permitindo criar arquivos mais eficientes. Os tamanhos variam de 1 a 8 *bytes*, sendo que as versões de 1 e 2 *bytes* estão implementadas e a versão de ponto flutuante de 8 *bytes* está implementada para escala e *base point*. Os dois *bits* mais significativos do *byte* de um comando determina o tamanho dos parâmetros. Se os *bits* possuírem o valor 00 então o tamanho é 1 *byte*; se o valor for 01, então o tamanho é 2 *bytes*; caso o valor dos *bits* seja 10, o tamanho é 4 *bytes*; e caso seja 11, o tamanho é 8 *bytes*.

As coordenadas são especificadas em inteiros positivos, os *offsets* também podem ser negativos. No caso das poligonais, deve-se decidir entre utilizar a forma pequena (0..255) ou a forma longa (0..65535) do comando. Cada comando completo de poligonal consiste em um comando descrevendo quantos pontos serão especificados, seguido por cada um dos pontos. O último parâmetro especifica quantos *bytes* cada valor ocupa. Uma poligonal com modo *fill* é exibida como um polígono preenchido. O último segmento de linha é sempre uma linha implícita do primeiro ao último ponto. As poligonais também podem ser definidas de modo relativo, onde cada ponto descreve um *offset* do ponto corrente. Os *offsets* podem ser positivos ou negativos.

Há uma tabela de cores padrão com 256 cores, mas também podem ser definidas cores específicas em RGB. Os *hyperlinks* podem ser ativados ou desativados

baseando-se na camada onde eles estão. As camadas possibilitam agrupar objetos, de modo a mostrar ou não partes do desenho.

Os *links* são definidos na parte principal. Com eles, pode-se criar regiões de *link* de uma ou duas dimensões, uma ação de *link*, e um texto explicativo opcional. As regiões de *link* são exibidas como qualquer outra entidade, porém sua cor é invisível. Na *web*, a ação poderia ser um URL, que será ativado pelo *link*. Se o texto é uma palavra vazia, o *link* associado está inativo. O texto explicativo pode ser utilizado para fornecer mais informações sobre o *link*. Se a camada associada está desligada, o *link* está desativado.

Um *link* de uma dimensão significa que uma linha ou curva pode ser selecionada. Um *link* de duas dimensões é selecionado escolhendo-se um ponto dentro de uma região. Um *link* de um ponto é selecionado escolhendo-se o ponto. *Links* de textos são definidos como uma caixa em torno do texto.

B. Descrição do TeCGraf Web Format

Como já foi visto, no início de um arquivo TWF está o identificador do formato (*Id*), seguido pelos valores máximos da largura (*w*) e altura (*h*) em *pixels* da imagem (Figura 4.1), nesta ordem.

O *Id* é composto pelo tamanho do texto que identifica o formato, seguido pelo próprio texto. Sendo o texto de identificação igual a “CDMFBV”, o *Id* é composto pelo número 6, que ocupa 16 *bits* (*short*), seguido por “CDMFBV”. A largura e a altura ocupam cada uma 32 *bits* (*int*). Como a maior resolução admita na primeira proposta do TWF é $2^{15}\times2^{15}$ e na segunda proposta é $2^{30}\times2^{30}$, 32 *bits* são suficientes para armazenar a altura e a largura.

Depois do *Id*, da largura e da altura, o arquivo possui as funções, que não têm uma ordem pré-estabelecida de como devem ser chamadas. Cada Função possui obrigatoriamente um índice que a identifica, porém pode ou não ter parâmetros e a quantidade de parâmetros varia de uma Função para outra. Os índices das funções ocupam um *byte* (*uchar*).

A seguir são listadas as funções com seus objetivos e parâmetros. O nome da função é seguido por seus parâmetros, isto é, o nome de cada função é seguido pelos dados que devem ser passados ao se chamar a mesma. Logo após, o objetivo da função, os valores válidos para seus parâmetros e algum dado auxiliar são apresentados. No fim da explicação sobre cada função, pode ser vista uma tabela que representa a ordem em que os dados são armazenados no arquivo, o primeiro dado armazenado é sempre o índice da respectiva função. Entre as funções que indicam o início e o fim de uma poligonal estão os pontos que compõem a mesma. Esses pontos são armazenados de acordo com as especificações das propostas 1 ou 2 do TWF.

- *Flush ( )*: envia informações para dispositivos *buffered*. Não possui parâmetros.

0

- *Clear ( )*: limpa a VVS utilizando a cor de fundo corrente. Não possui parâmetros.

1

- *Clip (uchar mode)*: habilita ou desabilita o *clipping*. O parâmetro *mode* pode possuir os valores CD_CLIPOFF, CD_CLIPAREA ou CD_CLIPPOLYGON, que são iguais a 0, 1 e 2, respectivamente. O CD_CLIPOFF, que é o valor padrão, desabilita o *clipping*. O CD_CLIPAREA habilita o *clipping* e o CD_CLIPPOLYGON ativa um polígono como região de *clipping*.

2	*mode*

- *ClipArea (int xmin, int xmax, int ymin, int ymax)*: define um retângulo para *clipping*. No retângulo de *clipping*, apenas os pontos no intervalo *xmin* <= x <= *xmax*, *ymin* <= y <= *ymax* serão impressos. A região padrão é (0, *w*–1, 0, *h*–1),

onde *w* e *h* são a largura e a altura da VVS, respectivamente.

3	*xmin*	*xmax*	*ymin*	*ymax*

- *Line (int x1, int y1, int x2, int y2)*: desenha uma linha, utilizando a cor de frente, a espessura e o estilo de linha correntes. A linha começa em *(x1,y1)* e termina em *(x2,y2)*.

4	*x1*	*y1*	*x2*	*y2*

- *Box (int xmin, int xmax, int ymin, int ymax)*: preenche um retângulo de acordo com o estilo interior corrente. São preenchidos com o estilo interior corrente todos os pontos no intervalo *xmin* <= x <= *xmax*, *ymin* <= y <= *ymax*.

5	*xmin*	*xmax*	*ymin*	*ymax*

- *Arc (int xc, int yc, int w, int h, double angle1, double angle2)*: desenha um arco de uma elipse alinhado com o eixo utilizando a cor de frente, a espessura e o estilo de linha correntes. O arco de elipse desenhado tem o par de coordenadas *(xc, yc)* como centro da elipse. As dimensões *w* e *h* são os eixos elípticos X e Y, respectivamente. Os ângulos *angle1* e *angle2* definem o início e o fim do arco. O arco começa no ponto *(xc+(w/2)*cos(angle1),yc+(h/2)*sin(angle1))* e termina em *(xc+(w/2)*cos(angle2),yc+(h/2)*sin(angle2))*. Os ângulos são dados em graus.

6	*xc*	*yc*	*w*	*h*	*angle1*	*angle2*

- *Sector (int xc, int yc, int w, int h, double angle1, double angle2)*: preenche um arco de uma elipse, onde o centro da última é a coordenada *(xc, yc)*, as dimensões *w* e *h* são os eixos elípticos X e Y, respectivamente e os ângulos *angle1* e *angle2* definem o início e o fim do arco. O arco começa no ponto *(xc+(w/2)*cos(angle1),yc+(h/2)*sin(angle1))* e termina em *(xc+(w/2)*cos(angle2),yc+(h/2)*sin(angle2))*. Os ângulos são dados em graus.

7	*xc*	*yc*	*w*	*h*	*angle1*	*angle2*

- *Text (int x, int y, char *text)*: inclui um texto na posição (*x*, *y*) de acordo com a opacidade de fundo, a fonte e o alinhamento de texto correntes. Antes do *text*, é armazenado no arquivo o tamanho do mesmo (*short size*).

8	*x*	*y*	*size*	*text*

- *Begin (uchar mode)*: começa a definição de um polígono a ser desenhado ou preenchido. O *mode* pode ter os valores CD_FILL, CD_OPEN_LINES CD_CLOSED_LINES ou CD_CLIP, que são iguais a 0, 1, 2 e 3, respectivamente. O modo CD_CLOSED_LINES conecta o último ponto ao primeiro. O CD_FILL conecta o último ponto ao primeiro e preenche o polígono resultante de acordo com o estilo interior corrente. Ao invés de criar um polígono a ser desenhado,

CD_CLIP cria um polígono para definir uma região de *clipping* não retangular.

9	*mode*

- *End ( )*: finaliza a definição do polígono e o desenha ou preenche.

10

- *BackOpacity (uchar opacity)*: configura a opacidade do fundo para as primitivas de preenchimento baseadas em cores de frente e fundo. O parâmetro *opacity* pode assumir os valores CD_OPAQUE (0) ou CD_TRANSPARENT (1). Se ela for opaca, a primitiva de texto, por exemplo, apagará tudo o que estiver na caixa envolvente com a cor de fundo. Se for transparente, apenas a cor de frente é pintada. O valor padrão é o CD_TRANSPARENT.

12	*mode*

- *WriteMode (uchar mode)*: define o modo de escrita para todas as primitivas de desenho. O *mode* pode ter os valores CD_REPLACE (0), CD_XOR (1) ou CD_NOT_XOR (2). O padrão é o CD_REPLACE.

13	*mode*

- *LineStyle (uchar style)*: configura o estilo de linha corrente. O estilo de linha corrente pode ter os valores CD_CONTINUOUS, CD_DASHED, CD_DOTTED, CD_DASH_DOT ou CD_DASH_DOT_DOT, iguais a 0, 1, 2, 3 e 4, respectivamente. O padrão é CD_CONTINUOUS.

14	*style*

- *LineWidth (uchar width)*: o parâmetro *width* indica a espessura de linha corrente (em *pixels*). O valor padrão é 1. Apenas são válidas espessuras maiores que 1.

15	*width*

- *InteriorStyle (uchar style)*: configura o estilo corrente para as primitivas de preenchimento de área. O estilo corrente pode ter os valores CD_SOLID, CD_HATCH, CD_STIPPLE ou CD_PATTERN, que são iguais a 0, 1, 2 e 3, respectivamente. O CD_HATCH e o CD_STIPPLE são afetados pela opacidade de fundo corrente. O padrão é CD_SOLID.

16	*style*

- *Hatch (uchar style)*: seleciona um estilo de *hatch* pré-definido e configura o estilo interior para CD_HATCH. O estilo de *hatch* pré-definido pode ser CD_HORIZONTAL, CD_VERTICAL, CD_FDIAGONAL, CD_BDIAGONAL, CD_CROSS ou CD_DIAGCROSS, iguais a 0, 1, 2, 3, 4 e 5, respectivamente. O

valor padrão é CD_HORIZONTAL.

17	*style*

- *Stipple (int w, int h, uchar *fgbg)*: define a matriz *fgbg* de zeros e uns, onde *w*×*h* é o tamanho mesma. Os zeros são mapeados para a cor de fundo ou são transparentes, de acordo com o atributo de opacidade do fundo. Os uns são mapeados para a cor de frente. O estilo interior é configurado para CD_STIPPLE. O padrão é uma matriz 32×32 cheia de zeros.

18	*w*	*h*	*fgbg[0]*	*fgbg[1]*	…	*fgbg[wxh–1]*

- *Pattern (int w, int h, long int *color)*: define a matriz de cores *color*, onde *w*×*h* é o tamanho da mesma. Cada cor possui as componentes R, G e B (*uchar r*, *uchar g*, *uchar b*) e uma componente alfa (*uchar alpha*) que não é armazenada no arquivo, pois ainda não foi totalmente implementada. O padrão é uma matriz 32×32 toda com a cor preto.

19	*w*	*h*	*r[0]*	*g[0]*	*b[0]*	…	*r[wxh–1]*	*g[wxh–1]*	*b[wxh–1]*

- *Font (uchar typeface, uchar style, uchar size)*: seleciona uma fonte de texto. O *typeface* pode ser CD_SYSTEM, CD_COURIER, CD_TIMES_ROMAN ou CD_HELVETICA, iguais a 0, 1, 2 e 3, respectivamente. O *style* pode ser CD_PLAIN, CD_BOLD, CD_ITALIC ou CD_BOLD_ITALIC. O *size* é dado em pontos (1/72 polegada) e, para facilitar a escolha, há três constantes CD_SMALL=8, CD_STANDARD=12 e CD_LARGE=18. Os valores padrão são CD_SYSTEM, CD_PLAIN, CD_STANDARD.

20	*typeface*	*style*	*size*

- *NativeFont (char* font)*: seleciona uma fonte de texto nativa (*font*), cuja descrição é dependente do *driver* e da plataforma. Antes do texto que indica a fonte, é armazenado no arquivo o tamanho desse texto (*short size*).

21	*size*	*font*

- *TextAlignment (uchar alignment)*: define o alinhamento vertical e horizontal de um texto. Esse alinhamento pode ser CD_NORTH, CD_SOUTH, CD_EAST, CD_WEST, CD_NORTH_EAST, CD_NORTH_WEST, CD_SOUTH_EAST, CD_SOUTH_WEST, CD_CENTER, CD_BASE_LEFT, CD_BASE_CENTER, ou CD_BASE_RIGHT, que variam de 0 a 11, respectivamente. O padrão é CD_BASE_LEFT.

22	*alignment*

- *MarkType (uchar type)*: configura o tipo de marca corrente. O tipo de marca pode ser CD_PLUS, CD_STAR, CD_CIRCLE, CD_X, CD_BOX, CD_DIAMOND, CD_HOLLOW_CIRCLE, CD_HOLLOW_BOX ou CD_HOLLOW_DIAMOND, que variam de 0 a 8, respectivamente. O padrão é CD_STAR.

23	*type*

- *MarkSize (uchar size)*: configura o tamanho da marca em *pixels*. O valor padrão em *pixels* é 10. O intervalo de espessura válido é maior que 1.

24	*size*

- *Palette (uchar n, long int *color, uchar mode)*: em sistemas limitados a 256 cores de paleta, esta função busca inserir as *n* cores na paleta *color* do sistema. O *mode* pode ser CD_POLITE (0) ou CD_FORCE (1). O modo CD_FORCE ignora as cores do sistema e os elementos de interface, pois os menus e diálogos podem estar em cores ilegíveis, porém existirão mais cores disponíveis. Recomenda-se utilizar CD_POLITE. Cada cor possui as componentes R, G e B (*uchar r*, *uchar g*, *uchar b*) e uma componente alfa (*uchar alpha*) que não é armazenada no arquivo, pois ainda não foi totalmente implementada.

25	*n*	*mode*	*r[0]*	*g[0]*	*b[0]*	…	*r[n–1]*	*g[n–1]*	*b[n–1]*

- *Background (long int color)*: configura a nova cor de fundo corrente para *color*. O valor padrão é a cor branca. Como ocorre para as outras funções, a cor possui as componentes R, G e B (*uchar r*, *uchar g*, *uchar b*) e uma componente alfa (*uchar alpha*) que não é armazenada no arquivo.

26	*r*	*g*	*b*

- *Foreground (long int color)*: configura a nova cor de frente corrente para *color*. Esta cor é utilizada em todas as primitivas (linhas, áreas, marcas e texto). O valor padrão é o preto. A cor possui as componentes R, G e B (*uchar r*, *uchar g*, *uchar b*) e uma componente alfa (*uchar alpha*) que não é armazenada no arquivo.

27	*r*	*g*	*b*

- *PutImageRGB (int iw, int ih, uchar *r, uchar *g, uchar *b, int x, int y, int w, int h)*: coloca, em uma área especificada da VVS, uma imagem que tem os componentes vermelho, verde e azul definidos nas três matrizes *r*, *g* e *b*, respectivamente. Cada componente destas matrizes ocupa 1 *byte* (*uchar*). Os componentes estão organizados na matriz linha a linha, ou seja, primeiro são definidos todos os componentes da primeira linha da coluna zero até a coluna *iw*–1, depois os componentes da segunda linha da coluna zero até a coluna *iw*–1 e assim por diante. O *pixel* (0,0) fica no canto inferior esquerdo e o *pixel* (*iw*–1, *ih*–1) fica no canto superior direito do retângulo da imagem. Os parâmetros *w* e *h* referem-se ao retângulo-destino da VVS, de forma que é possível reduzir ou expandir a

imagem desenhada.

28	*iw*	*ih*	*x*	*y*	*w*	*h*	*r[0]*	*g[0]*	*b[0]*	…	*r[iwxih–1]*	*g[iwxih–1]*	*B[iwxih–1]*

- *PutImageMap (int iw, int ih, uchar *index, long int *colors, int x, int y, int w, int h)*: coloca, em uma área especificada da VVS, uma imagem cujas cores são dadas através de uma matriz de índices (*index*), um mapa de cores (*colors*). A cor correspondente a um dado índice é dada em *colors[index]*. Essas cores indexadas são decodificadas para o modo RGB (*uchar r*, *uchar g*, *uchar* b) e então armazenadas no arquivo. O *pixel* (0,0) fica no canto inferior esquerdo e o *pixel* (*iw*–1, *ih*–1) fica no canto superior direito do retângulo da imagem. Os parâmetros *w* e *h* referem-se ao retângulo-destino da VVS, de forma que é possível reduzir ou expandir a imagem desenhada. Na tabela seguinte, *mind* indica o maior índice da matriz de índices (*index*).

29	*iw*	*ih*	*x*	*y*	*w*	*h*	*index[0]*	…
index[iwxih–1]	*r[0]*	*g[0]*	*b[0]*	…	*r[mind–1]*	*g[mind–1]*	*b[mind–1]*	

- *Pixel (int x, int y, long int color)*: o *pixel (x, y)* recebe a cor definida por *color*, sendo *color* composta pelas componentes R, G e B (*uchar r*, *uchar g*, *uchar b*) e uma componente alfa (*uchar alpha*) que não é armazenada no arquivo.

30	*x*	*y*	*r*	*g*	*b*

- *ScrollImage (int xmin, int xmax, int ymin, int ymax, int dx, int dy)*: copia o retângulo definido pelas coordenadas *(xmin, ymin)* e *(xmax, ymax)* no retângulo definido por *(xmin+dx, ymin+dy*) e *(xmax+dx, ymax+dy)*.

31	*xmin*	*xmax*	*ymin*	*ymax*	*dx*	*dy*

- *TextOrientation (double angle)*: define a orientação do texto. O parâmetro *angle* define o ângulo em graus do texto em relação à linha horizontal. O valor padrão é 0.

32	*angle*

C. Estimativa de Dimensão de Imagens

Supondo que os mapas apresentados em papel possuem um erro máximo ε de 0.1 mm na escala da carta, a quantização das coordenadas em um *grid* uniforme deve possuir um espaçamento máximo de:

$$\Delta_x = \Delta_y = e\sqrt{2} = 0.1\sqrt{2} = \sqrt{0.02},$$

conforme ilustra a Figura C.1.

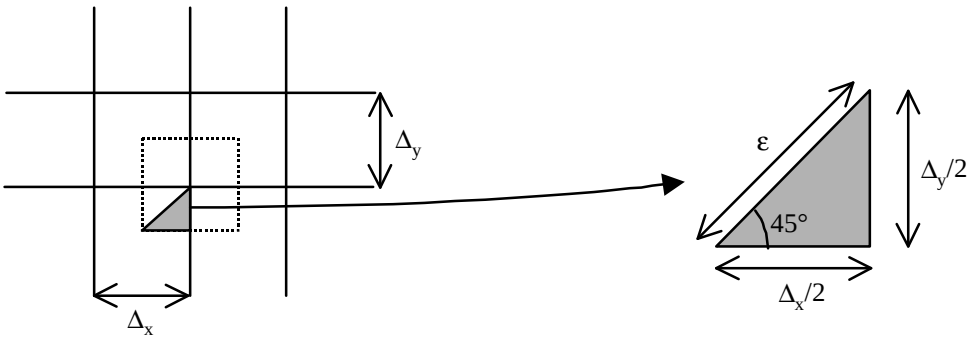


Figura C.1: Quantização das coordenadas em um *grid* uniforme

A Tabela C.1 apresenta o tamanho das imagens em relação aos mapas apresentados em papel A0, A1, A2, A3 e A4, com este espaçamento.

Folha	Altura × Largura (mm)	Altura × Largura (pixel)
A0	1188 × 840	11700 × 8268
A1	594 × 840	5850 × 8268
A2	594 × 420	5850 × 4125
A3	297 × 420	2925 × 4125
A4	297 × 210	2925 × 2075

Tabela C.1: Relação entre o tamanho das imagens e os diversos tipos de folha

Referências

1. GONZALEZ, R. C., WOODS, R. E., *Digital Image Processing*, Addison-Wesley Publishing Company, 1992.
2. FOLEY, J. D., DAM, A., FEINER, S. K., HUGHES, J. F., *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, 1996.
3. MURRAY, J. D., VANRYPER, W., *Encyclopedia of Graphics File Formats*, O'Reilly & Associates, Inc., 1994.
4. NELSON, M., GAILLY, J., *The Data Compression Book*, *M&T Books*, Segunda Edição, 1996.
5. HENDERSON, L. R., MUMFORD A. M., *The CGM Handbook*, Academic Press Inc., 1993.
6. *Information Technology - Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information*, Part 1 - Functional Specification (ISO8632-1), Part 2 - Character Encoding (ISO8632-2), Part 3 - Binary Encoding (ISO8632-3), Part 4 - Clear Text Encoding (ISO8632-4), 1992.
7. ENDELE, G., KANSY, K. e PFAFF, G., *Computer Graphics Programming - GKS The Graphical Standard*, Springer-Verlag, Berlim, 1984.
8. CÂMARA, G., CASANOVA, M. A., HEMERLY, A. S., MAGALHÃES, G. C., MEDEIROS, C. M. B., Anatomia de Sistemas de Informação Geográfica, 10° Escola de Computação, Campinas, 1996.
9. PUPPO, E., DETTORI, G., *Towards a Formal Model for Multiresolution Spatial Maps*, in 4th International Simposium on Large Spatial Databases, pp. 152-169, 1995.
10. DOUGLAS, D. H., PEUCKER, T. K., *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*, The Canadian Cartographer, Volume 10, 2, pp. 112-122, 1973.
11. DERRAIK, A. L. B., Um Estudo Comparativo de Representações de Multi-Resolução para Linhas e Poligonais, Dissertação de Mestrado, Departamento de Informática PUC-Rio, 1997.
12. MEDIANO, M. R., V-tree: Um Método de Armazenamento para Dados Vetoriais Longos, Dissertação de Mestrado, Departamento de Informática PUC-Rio, 1995.
13. HUFFMAN, D. A., *A Method for the Construction of Minimum Redundancy Codes*, Proceedings of IRE 40, 1951, pp. 1098-1101.
14. FALLER, N., Sistema Adaptativo para Compressão de Dados, Dissertação de Mestrado, COPPE-UFRJ, 1973.
15. GALLAGER, R. G., *Variations on a Theme of Huffman*, IEEE Transactions on Information Theory IT-24, 6, November 1978, pp. 668-674.
16. KNUTH, D. E., *Dynamic Huffman Coding*, *Journal of Algorithms 6*, 1985, pp. 163-180.
17. ZIV, J., LEMPEL, A., *A Universal Algorithm for Sequential Data Compression*,

IEEE Transactions on Information Theory IT-23, 3, May 1977, pp. 337-343.

18. ZIV, J., LEMPEL, A., *Compression of Individual Sequences via Variable-Rate Coding*, IEEE Transactions on Information Theory IT-24, 5, September 1978, pp.530-536.
19. WELCH, A. T., *A Technique for High Performance Data Compression*, IEEE Computer,17, 6, June 1984, pp.8-19.
20. *Computer Graphics – Graphical Kernel System (GKS) Functional Description*, ANSI X3.124, June 1985.
21. PHIGS+ Committee, Andries van Dam, chair, *PHIGS+ Functional Description*, Revision 3.0, Computer Graphics, 22(3), June 1988, 125-218.
22. CRAIN, D., *Microsoft Developer Network Technology Group*, June 1993.
23. *Sun Release 4.1*, Manual On Line, 19 de outubro de 1987.
24. *The Bresenham Line-Drawing Algorithm*: http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html.
25. *Algorithm Vault - Bresenham's Line-Drawing Algorithm*: http://www.dbn.lia.net/users/greg/grfxbsl.htm.
26. OBERHUMER, M. F. X. J., LZO: http://wildsau.idv.uni-linz.ac.at/mfx/lzo.html.
27. *Autodesk - AutoCAD: DXF: The Open Standard For Drawing Exchange*: http://www.autodesk.com/products/acadr14/features/dxf.htm.
28. *WHIP! and DWF - A Primer*: http://www.autodesk.com/products/whip/primer.htm.
29. Using DWF Files: http://www.autodesk.com/products/acadr14/features/usedwf.htm.
30. *Adobe PDF*: http://www.adobe.com/prodindex/acrobat/adobepdf.html.
31. *Technical Information on SVF*: http://www.softsource.com/svf/tech.html.
32. TeCGraf: http://www.tecgraf.puc-rio.br/.
33. CD – Canvas Draw – Uma Biblioteca Gráfica 2D: http://www.tecgraf.puc-rio.br/manuais/cd/.
34. *Driver CD Metafile*: http://www.tecgraf.puc-rio.br/manuais/cd/drv/mf.html.
35. IM – Biblioteca de Acesso a Arquivos de Imagens *Bitmaps*: http://www.tecgraf.puc-rio.br/manuais/im/.
36. *Answers to Frequently Asked Questions About CGM*: http://www.intercap.com/CGMinfo/cgmfaq.htm.
37. *Web Vector Format Comparison*: http://www.tailormade.com/WebFormatCompare.htm.
38. *Web Vector Format Comparison*: http://www.intercap.com/CGMinfo/Papers/vector.htm.
39. CIA World Data Bank: ftp://setftp.stanford.edu/pub/World_Map/.
40. Curvas de Nível: ftp://spectrum.xerox.com/ds9/map/dlg/1:24,000/optional/.

41. Malha Municipal Digital do Brasil - Situação em 1991 e 1994, IBGE (Instituto Brasileiro de Geografia e Estatística), Base de dados em CD, ISBN: 85-240-0591-2.

42. *Airline Transport Association Specifications*: http://www.lmengineering.com/ataspecs.html.

43. CALS: http://arriwww.uta.edu/ccc/calsover.html.

44. CGM*PIP: http://www.posc.org/technical/cgmpip/pip_index.html.

45. W3C: http://www.w3.org/.

46. *W3C Scalable Graphics Requirements*: http://www.w3.org/Graphics/ScalableReq.

47. GAILLY, J., ADLER, M., ZLIB: http://www.cdrom.com/pub/infozip/zlib/.

48. *What about patents on data compression algorithms*?: http://www.faqs.org/faqs/compression-faq/part1/section-6.html

UM ESTUDO SOBRE ARQUIVOS VETORIAIS PARA VISUALIZAÇÃO DE MAPAS NA *WEB*

Dissertação de Mestrado apresentada por CARLA CRISTINA FONSECA FERREIRA em 11 de maio de 1998 ao Departamento de Informática da PUC-Rio e aprovada pela comissão julgadora formada por:

_____

Prof. Marcelo Gattass (DI/PUC-Rio)
Orientador

_____

Prof. Luiz Henrique de Figueiredo (LNCC)
Co-orientador

_____

Prof. Arndt von Staa (DI/PUC-Rio)

_____

Prof. Gilberto Câmara Neto (INPE)

Visto e permitida a impressão.
Rio de Janeiro,

_____

Coordenador dos Programas de Pós-Graduação
do Centro Técnico Científico da PUC-Rio