UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA DE CONCENTRAÇÃO: CIÊNCIA DA COMPUTAÇÃO

FERNANDO JORGE CUTRIM DEMETRIO

VISUALIZAÇÃO DE DADOS VOLUMÉTRICOS COMPRIMIDOS BASEADO NA TRANSFORMADA DO COSSENO LOCAL

São Luís
2005

FERNANDO JORGE CUTRIM DEMETRIO

VISUALIZAÇÃO DE DADOS VOLUMÉTRICOS COMPRIMIDOS BASEADO NA TRANSFORMADA DO COSSENO LOCAL

Dissertação apresentada ao programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão para obtenção do título de mestre em Engenharia de Eletricidade.

Orientador: Prof. Dr. Anselmo Cardoso de Paiva

São Luís
2005

FERNANDO JORGE CUTRIM DEMETRIO

VISUALIZAÇÃO DE DADOS VOLUMÉTRICOS COMPRIMIDOS BASEADO NA TRANSFORMADA DO COSSENO LOCAL

Dissertação apresentada ao programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão para obtenção do título de mestre em Engenharia de Eletricidade.

Aprovada em: / /

BANCA EXAMINADORA

Prof. Dr. Anselmo Cardoso de Paiva
(Orientador)

Prof. Dr. Aristófanes Corrêa Filho
(Membro da Banca Examinadora)

Prof. Dr. Joaquim Bento Cavalcante Neto
(Membro da Banca Examinadora)

Dedico este trabalho, em especial, a minha mãe por estar comigo todos dias da minha vida sempre me segurando para não cair e se cair dando a mão para me reerguer.

AGRADECIMENTOS

Agradeço em primeiro lugar a minha mãe e meu irmão, por me apoiarem neste trabalho.

Ao professor doutor Anselmo Cardoso de Paiva, meu orientador, pela oportunidade, paciência e apoio científico.

A minha esposa, Andréya Ingryd, pelo incentivo depositado neste período.

A João Rodrigo e Leonardo Martins que ajudaram na elaboração do programa.

À professora Maria da Guia, pela atenção depositada neste período.

Ao grupo de Informática Aplicada – GIA, pelos momentos que dividimos nestes anos.

À Universidade Federal do Maranhão – UFMA, por desenvolver este programa de pós-graduação.

Ao Centro de Ensino Unificado do Maranhão –Ceuma, por permitir que desenvolvesse esse trabalho.

“Computação gráfica é matemática e arte. É uma ferramenta de concepção de arte, assim como piano ou o pincel.”

Eduardo Azevedo

RESUMO

Este trabalho descreve o desenvolvimento de um algoritmo para visualização de dados volumétricos comprimidos baseado na transformada do cosseno local, que minimiza os artefatos nos blocos gerados pelo método de compressão, e possibilita a visualização de grandes volumes em computadores com recursos limitados de memória. Nós apresentamos também os resultados obtidos pelo processo de visualização.

Palavras Chave:Transformada do Cosseno Local. Visualização de Dados Volumétricos Comprimidos. Compressão.

ABSTRACT

This work describes the development of an algorithm for visualization of compressed volumetric data with a local cosine transform scheme, that minimizes the blocking artefacts generated by transform compression methods, and makes possible the visualization of large volume data in computer with limited memory resources. We also present the results obtained for the visualization process .

Keywords: Local cosine transform. Visualization of Compressed Volumetric Data. Compression.

LISTA DE FIGURAS

Figura 2.1 – Representação por blocos e células..... 17

Figura 2.2 – *Pipeline* de visualização volumétrica..... 18

Figura 2.3 – Método de visualização volumétrica, adaptada de Paiva..... 19

Figura 2.4 – Lançamento do raio..... 21

Figura 2.5 – Tabela de *Footprint*..... 23

Figura 3.1 – Fluxo genérico de algoritmos de compressão sem perdas..... 26

Figura 3.2 – Exemplo de codificação usando o algoritmo de *Huffman*..... 28

Figura 3.3 – Fluxo genérico de algoritmos de compressão baseados em transformadas..... 34

Figura 3.4 – Extensão cosseno I para uma função *f*(t)..... 36

Figura 3.5: Extensão cosseno IV de uma função *f*(t)..... 37

Figura 3.6: Intervalos sobrepostos recobrindo o eixo do tempo..... 39

Figura 3.7: A função de janelamento *g(t)* e os subintervalos..... 40

Figura 3.8: As janelas sobrepostas dos intervalos *p*-1, *p* e *p*+1..... 40

Figura 3.9 – Representação gráfica da operação de dobragem..... 41

Figura 3.10: Visão geral do método de compressão de volumes baseado na transformada do cosseno local..... 43

Figura 3.11: ilustração do caminhamento em *zigzag* em um bloco do volume..... 47

Figura 4.1 – Ordem de Armazenamento dos blocos do volume comprimido..... 54

Figura 4.2 – Separador de blocos (*offset*)..... 55

Figura 4.3 – Estruturas *BlockOffSet* e *Uncomp*..... 56

Figura 4.4 – Descompressão do bloco (i, j, k) na direção k..... 57

Figura 4.5 – Imagem do volume utilizado..... 59

Figura 4.6 – Gráfico de Tempo de Visualização Puro..... 60

Figura 4.7 – Gráfico Tempo de Descompressão..... 61

Figura 4.8 – Gráfico com Tempo de Visualização Total..... 62

Figura 4.9 – Gráfico Taxa x Tempos (Codificação de *Huffman*)..... 63

Figura 4.10 – Gráfico Taxa x Tempos (Codificação Aritmética)..... 64

Figura 4.11 – Gráfico Tamanho dos Blocos x tempo..... 65

Figura 4.12 – Gráfico Tamanho dos Blocos x tempo..... 65

Figura 4.13 – Gráfico com a variação do *Iap*..... 66

SUMÁRIO

1 INTRODUÇÃO..... 12

2 VISUALIZAÇÃO VOLUMÉTRICA..... 17

2.1 Algoritmos de visualização volumétrica 19

2.1.1 *Ray Casting*..... 20

2.1.2 *Splatting*..... 22

3 COMPRESSÃO DE DADOS VOLUMÉTRICOS USANDO A TRANSFORMADA DO COSSENO LOCAL..... 25

3.1 Métodos de compressão sem perdas..... 25

3.1.1 Código de *Huffman*..... 27

3.1.2 Codificação Aritmética..... 29

3.2 Métodos de compressão com perdas..... 32

3.2.1 Compressão com a Transformada do Cosseno Local (LCT)..... 34

3.2.1.1 A Tranformada do Cosseno Local (LCT) 35

3.3.1.2 Compressão de Volumes Baseado na LCT 42

4 ALGORITMO DE VISUALIZAÇÃO DE VOLUMES COMPRIMIDOS COM A TRANSFORMADA DO COSSENO LOCAL..... 49

4.1 Trabalhos Correlatos..... 49

4.2 Estrutura do Dado Volumétrico Comprimido..... 54

4.3 Descompressão e Visualização..... 55

4.4 Resultados Obtidos..... 59

5 CONCLUSÕES..... 69

REFERÊNCIAS..... 72

ANEXO..... 75

1 INTRODUÇÃO

Visualização científica é a área da computação que trata das técnicas que utilizam imagens e animações para analisar e interpretar dados científicos, e permite resumir de modo gráfico os resultados gerados pelas aplicações de computação analisando os resultados, interpretando e formulando conclusões. É uma importante área de pesquisa e apresenta como forma de visualização a visualização volumétrica.

> A visualização volumétrica é uma das mais utilizadas técnicas de visualização científica e tem como principal objetivo a geração de imagens bidimensionais que representem a visualização de dados associados a regiões de um volume, dados volumétricos. Os dados volumétricos são provenientes de duas fontes principais, simulação e aquisição.

Os dados provenientes de simulação são construídos a partir de modelos matemáticos que representam o comportamento de fenômenos existentes na natureza. Algumas vezes, a simulação é a única forma possível de estudar um determinado fenômeno, por exemplo, se um evento é muito grande, muito pequeno, muito rápido ou muito lento para se observar na natureza. Técnicas de análise numéricas como o método dos elementos finitos podem ser utilizadas para simular eventos da natureza, constituindo uma fonte geradora de dados volumétricos.

Por outro lado, também existem dados volumétricos provenientes da digitalização de fenômenos e objetos da natureza. Como exemplo de processos de aquisição de dados volumétricos podemos destacar os processos de aquisição de imagens médicas (Ressonância Magnética (MRI), Tomografia Computadorizada (CT), Tomografia por Emissão de Pósitrons (PET)); aquisição de dados geológicos

(Sísmica de Reflexão) entre outros. Uma outra forma de aquisição de dados é a geração de dados volumétricos através da conversão de objetos geométricos na representação por *voxels* (*voxelisation*).

Em geral dados volumétricos necessitam de grande espaço de memória para seu armazenamento e manipulação. Como exemplo da magnitude de dados volumétricos podemos citar o projeto *Visible Human* da *National Library of Medicine* (NLM), que realizou a aquisição através de tomografia computadorizada (CT), ressonância magnética (MRI) e imagens coloridas de seções congeladas de dois cadáveres, um masculino e outro feminino, resultando num total de 15 Gbytes de dados volumétricos para o cadáver masculino e de 40 *Gbytes* para o cadáver feminino (ACKERMAN, 1996).

Outro exemplo do tamanho desses dados é proveniente dos métodos de aquisição de dados por sísmica de reflexão, intensivamente utilizados pela indústria de petróleo para investigar a presença de óleo na sub-superfície. Os dados volumétricos provenientes da sísmica de reflexão são gerados a partir da interação de ondas elásticas com a geologia do solo. Como exemplo do tamanho desses dados podemos citar um levantamento tridimensional de 800 km^2, relatado em (BROCKLEHURST, 1995), que necessitou de 2794 *Gbytes* para seu armazenamento.

Em um ambiente de visualização, o simples armazenamento e transmissão desses dados já se apresenta como um grande desafio. Assim, verificamos, que sempre que tivermos limitação de recursos para representar a informação teremos a necessidade de compressão a qual mantém um papel fundamental no sentido de elevar o potencial de armazenamento e de transmissão destas informações.

Ferramentas de visualização volumétrica necessitam da disponibilidade de modernas estações de trabalho e computadores pessoais com bom desempenho, grande quantidade de armazenamento tanto em disco como em memória, e facilidades gráficas. Podemos então verificar que a visualização de dados volumétricos recai nos dois clássicos problemas computacionais: a necessidade de longo tempo de execução e o alto consumo de memória.

Com base em experiências anteriores relatadas na literatura (CHAN *et al.*, 1991), (NING; HESSELINK, 1993) e (YEO; LIU, 1995) é constatável que a compressão é um grande desafio para possibilitar a visualização de dados volumétricos nos computadores pessoais ou estações de trabalho de uso convencional. Nesse caso se faz necessário um método que permita ao usuário carregar uma versão comprimida do dado volumétrico em uma pequena porção de memória e que possibilite que esse volume comprimido possa ser acessado e visualizado como se o usuário estivesse com o volume descomprimido completamente carregado na memória. Um esquema como esse deve permitir a descompressão local dos dados, à medida que eles são necessários para a visualização.

Em Paiva (2001) é apresentado um método de compressão de dados volumétricos, baseado na transformada do cosseno local (LCT), que apresenta a característica de permitir a descompressão local do dado sem visualizar. Faz-se necessário então, o desenvolvimento de um algoritmo para visualização de dados volumétricos comprimidos, segundo o método proposto em Paiva. Este algoritmo deve tratar os dados sem descomprimi-los, ou realizando somente uma descompressão do dado à medida que cada porção do mesmo é necessária. Para

tanto torna-se necessário a adaptação das técnicas de visualização de dados volumétricos ao esquema de compressão dos dados.

O objetivo deste trabalho é desenvolver um algoritmo, que aliado as técnicas de visualização e compressão de dados volumétricos, permita visualizar e manipular esses dados sem que haja descompressão total do volume, mantendo apenas uma parte do volume descomprimido, e assim minimizando os requisitos de memória para a visualização de volumes de grandes dimensões em máquinas com memória limitada.

Especificamente pretendemos implementar o algoritmo de *Splatting* (WESTOVER, 1990) para visualizar dados volumétricos comprimidos com o uso da LCT, sem que haja a necessidade da descompressão total do volume. Adicionalmente, iremos analisar os resultados obtidos, verificando tempo de execução e aplicabilidade do algoritmo desenvolvido e comparar os resultados obtidos com diferentes métodos de codificação utilizados e outros algoritmos existentes.

Esta dissertação está organizada da seguinte maneira:

No capítulo 2 apresentamos uma visão geral sobre as aplicações e algoritmos de visualização volumétrica. Neste capítulo, apresentamos os principais algoritmos de visualização volumétrica, destacando o algoritmo de *Splatting* o qual é utilizado para a visualização de dados comprimidos.

No capítulo 3 apresentamos uma revisão dos métodos de compressão sem perdas e dos métodos de compressão com perdas, destacando nos métodos que utilizam transformadas, a transformada do cosseno local.

O capítulo 4 descreve o algoritmo de visualização proposto o qual é baseado na transformada do cosseno local, as tecnologias utilizadas, o modelo de

implementação, com os diferentes métodos de quantização e codificação utilizados e os resultados obtidos pelo algoritmo.

Por fim, no capítulo 5 descrevemos as principais contribuições desta dissertação, futuras direções, ocasionando assim continuidade aos trabalhos aqui desenvolvidos.

2 VISUALIZAÇÃO VOLUMÉTRICA

Em (McCORMICK,1987) é relatado que a visualização volumétrica é o conjunto de técnicas utilizadas na visualização de dados associados a regiões de um volume, tendo como principal objetivo a exibição do interior de objetos volumétricos, a fim de explorar sua estrutura e facilitar sua compreensão.

Estes dados, quando associados a regiões de volumes, são denominados dados volumétricos. Assim, podemos também conceituar a visualização volumétrica como a classe de métodos de visualização relacionada com a representação, manipulação e visualização de conjuntos de dados volumétricos.

Os dados volumétricos são geralmente tratados como uma matriz de elementos de volumes (blocos ou células). Na abordagem por blocos (Figura 2.1 a) a área ao redor de um ponto do *grid* é considerada como possuindo o mesmo valor do ponto amostrado. Quando os dados são tratados como células (Figura 2.1 b) o volume é considerado como uma coleção de hexaedros com vértices nos pontos do *grid* e os valores variam dentro da célula através de uma interpolação dos valores dos vértices.

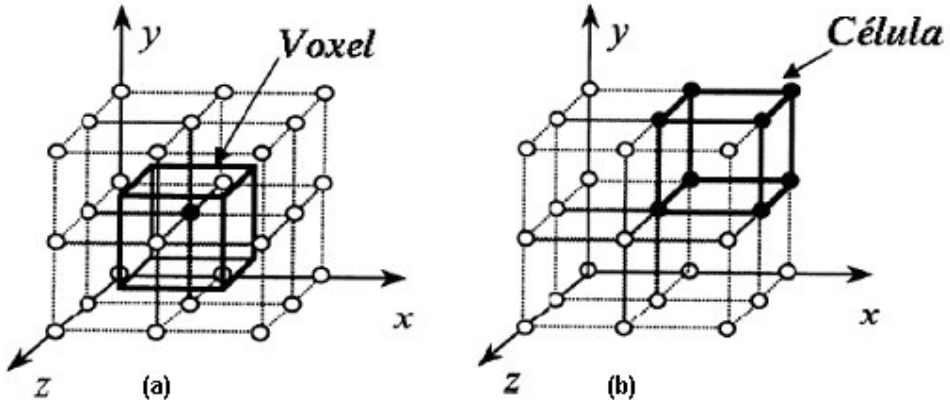


Figura 2.1 – Representação por blocos e células.

O processo de visualização passa por três etapas fundamentais. A primeira etapa é o processo de classificação que seleciona as características dos dados a partir de critérios quantitativos[1]. Em seguida é aplicado um modelo de iluminação, que calcula a tonalidade e cor de cada *voxel* de acordo com o material dos dados ou com as condições de iluminação externas, geralmente o modelo utilizado é o de Phong (1975). E por fim, a etapa de projeção dos *voxels* no plano da imagem que consiste em projetar o volume de acordo com a posição do observador em relação ao objeto e ao plano da imagem que determina assim a direção de projeção, conforme mostrado na Figura 2.2.

Dentre as projeções existem duas classes básicas, a perspectiva na qual todos os projetores (raios) passam por um ponto e a paralela na qual os projetores são linhas paralelas entre si.

Figura 2.2 *Pipeline* de visualização volumétrica.

Os métodos de visualização dos dados volumétricos se dividem em dois grupos: algoritmos de extração de superfícies (*surface fitting*) e algoritmos de

[1] Valores dos *voxels* são representados conforme sua densidade, variando sua cor de acordo com o valor apresentado.

rendering direto de volumes (*volume rendering*). Eles diferem basicamente pela utilização ou não de representações intermediárias dos dados volumétricos para gerar a visualização. Enquanto os de *rendering* direto, como o próprio nome denota, não precisam de uma etapa intermediária, ou seja a projeção é gerada diretamente a partir do volume, nos algoritmos de extração de superfície os dados volumétricos são convertidos para uma representação geométrica e após isso são aplicados técnicas de *rendering* de polígonos. A Figura 2.3 apresenta fluxo de processamento dos dois grupos de métodos.

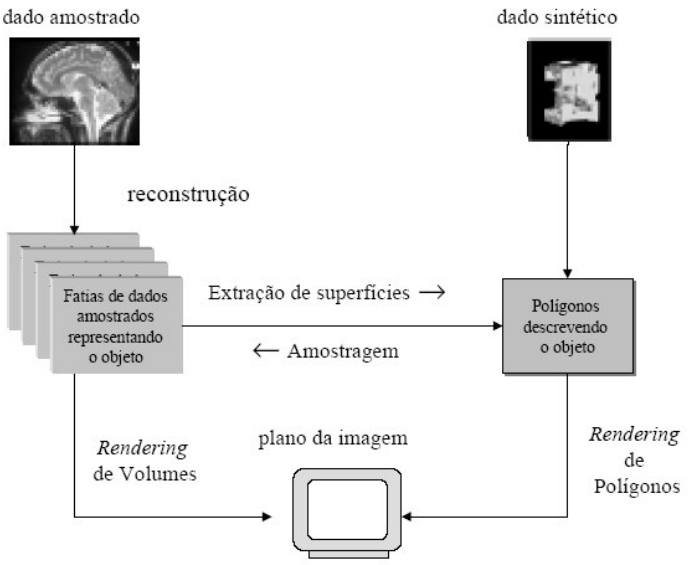


Figura 2.3 – Métodos de visualização volumétrica, adaptada de (Paiva, 2001).

2.1 Algoritmos de visualização volumétrica

O processo de visualização se inicia com a etapa de classificação, que está relacionada à identificação do material representado em cada *voxel*. Esta etapa possibilita a seleção de características dos dados, segundo um critério quantitativo,

definindo a região do volume que se deseja explorar. Em técnicas de Extração de Superfícies, classificar significa definir o valor de limiarização (*threshold*) utilizado para identificar a superfície que será poligonizada e posteriormente visualizada. Já para técnicas de visualização direta, a etapa de classificação envolve a definição da relação entre os valores dos dados do volume e os valores de cor e opacidade que serão utilizados no algoritmo de exibição, ou seja, a definição das funções de transferência.

Após a classificação é aplicado um modelo de iluminação que, com base nas propriedades do material de cada *voxel* e nas condições de iluminação externas, calcula a tonalidade da cor em cada ponto do volume.

Para a visualização direta, é comum utilizar uma adaptação do modelo de Phong (1975) com a substituição das normais à superfície por gradientes do campo escalar no *voxel*.

A última etapa do processo de visualização envolve a projeção dos *voxels*, na superfície de visualização e a conseqüente composição para determinar a imagem a ser visualizada.

Nas seções seguintes descreveremos dois algoritmos de visualização de dados volumétricos, um dos quais será utilizado no algoritmo proposto para a visualização dos dados comprimidos.

2.1.1 *Ray Casting*

O algoritmo de *Ray Casting* é um dos mais usados para a visualização de volumes quando necessitamos de imagens de alta qualidade (ELVINS,1992). O algoritmo percorre todos os *pixels* da imagem determinando a cor e a opacidade

para cada um. Um raio é disparado de cada *pixel* através do volume. As opacidades e cores encontradas ao longo do raio são acumuladas até se determinar a cor e a opacidade final do *pixel*.

Este algoritmo possui um alto custo computacional, mas pode ser paralelizado facilmente, uma vez que os valores dos *pixels* são determinados através do lançamento de raios independentes entre si.

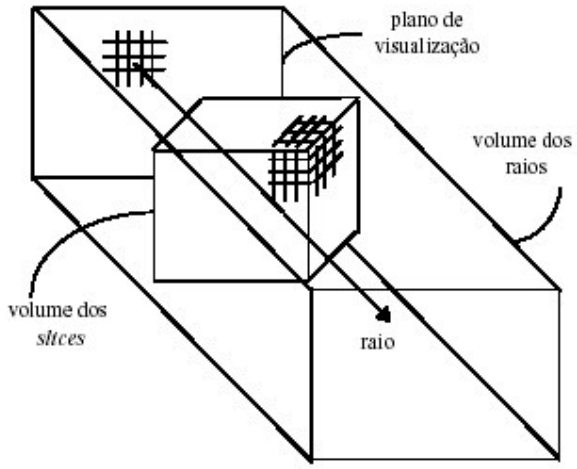


Figura 2.4 – Lançamento do raio

A tarefa fundamental deste algoritmo é o lançamento dos raios, o qual pode ser feito de maneira eficiente se considerarmos o *frustum*[2] de visão, lançarmos o primeiro raio como uma aresta desse *frustum* e, os demais, de maneira incremental.

Para gerar a imagem segundo o ponto de vista desejado aplicamos as transformações necessárias aos pontos que definem o *frustum* e lançamos os raios a partir do plano da imagem rotacionado e escalado para a resolução especificada.

[2] Volume que define os objetos a serem visualizados

2.1.2 *Splatting*

O algoritmo de visualização utilizado neste trabalho foi o de *Splatting,* por ser um algoritmo que obedece a ordem do objeto para composição da imagem. Neste algoritmo, cada elemento (*voxel*) é mapeado no plano da tela; em seguida, através de um processo de acumulação, tem sua contribuição adicionada à formação da imagem. O algoritmo termina quando todas as primitivas tiverem sido mapeadas na tela. Este algoritmo é definido por quatro etapas principais: transformação, classificação/iluminação, reconstrução e visibilidade.

O processo de transformação é composto pelo mapeamento das coordenadas (x,y,z) do volume em coordenadas de visualização (i,j,k).

O mapeamento é realizado através da definição da matriz de projeção, sendo efetuado de maneira mais eficiente através de cálculos incrementais.

O algoritmo pode então ler diretamente da matriz de transformação o tamanho dos incrementos de cada coordenada (i,j,k) para os incrementos de coordenadas (x,y,z). Este procedimento é o inverso do realizado no lançamento incremental dos raios no algoritmo de *Ray Casting*.

Os incrementos são constantes ao longo do volume, pois consideramos o *grid* uniforme com projeção ortográfica. Através da consulta ao sinal e à magnitude da mudança do valor de k, o algoritmo determina a ordem em que o volume será percorrido, de modo a garantir a ordem de composição desejada.

O eixo mais rápido é o que possui menor mudança em valores absolutos de k para cada passo ao longo de sua direção, e o sinal deste incremento define a extremidade inicial da composição. De maneira semelhante é definido o segundo

eixo mais rápido e o terceiro. Os dois primeiros formam um plano que será projetado e terá suas contribuições acumuladas em um buffer denominado *sheet*. A ordem de composição (de frente para trás ou de trás para frente) pode ser definida pelo usuário, bastando para isso utilizar o processo de composição adequado.

Uma vez conhecidos esses valores e definida a ordem de caminhamento, o algoritmo transforma o ponto inicial com a utilização da matriz de projeção. Em seguida os demais pontos são transformados pelo esquema incremental. Os pontos transformados são então enviados para o processo de classificação/iluminação. Este processo determina a cor e a opacidade de acordo com o valor do *voxel* e aplica o modelo local de iluminação gerando a cor e a opacidade final do *voxel*.

O passo seguinte, central ao algoritmo, é a reconstrução. Nele o algoritmo se propõe a reconstruir um sinal contínuo, a partir de um volume discreto, de modo a reamostrar o sinal na resolução desejada para gerar a imagem.

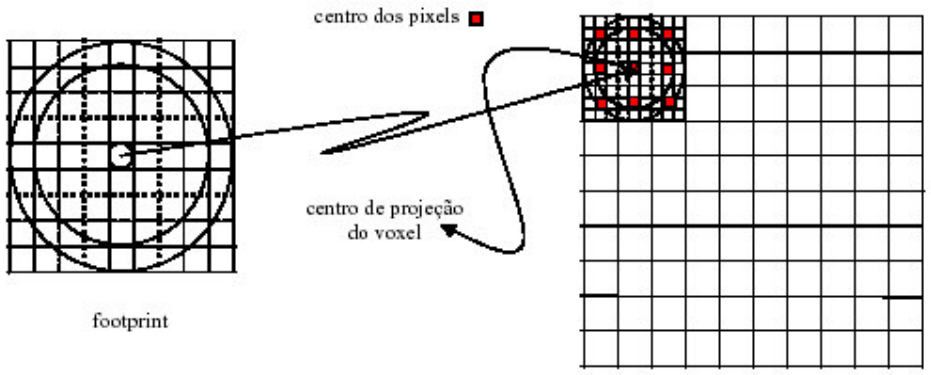


Figura 2.5 – Tabela de *Footprint*

Para isto é criada uma tabela representando a função de *footprint*, a qual é centralizada na posição de projeção de cada *voxel*. Em seguida determinamos os *pixels* cujos centros estão contidos sob a projeção da tabela.

Nessas posições pegamos o valor da tabela e ponderamos os valores de cor e opacidade do *voxel* compondo-os ao valor corrente no *sheet buffer*. A forma mais simples de construir a tabela que representa a função de *footprint* é determinar a extensão da projeção do núcleo do filtro de reconstrução (*kernel*) sobre o plano da imagem e selecionar uma resolução maior que a da imagem para amostrar a função.

Em seguida, o núcleo de reconstrução é integrado para cada um dos pontos gerados. Uma forma diferente de realizar este processo é através da modelagem do resultado com uma função simples (*e.g.* gaussiana), que é avaliada para cada ponto da tabela.

Para construir a tabela, o algoritmo calcula a extensão da projeção do núcleo de reconstrução e o mapeamento entre ela e a tabela normalizada básica gerada a partir de uma projeção ortográfica, sem operações de escala, da esfera unitária que contém o núcleo.

Para uma projeção ortográfica e um *grid* igualmente espaçado nas três direções, temos que o núcleo de reconstrução é mapeado em uma esfera. O algoritmo precisa criar uma tabela transformada apenas para levar em consideração as escalas uniformes que a transformação de visualização impôs aos *voxels*. O processo de visibilidade recebe a cor e a opacidade do *voxel* e pondera esses valores para gerar a contribuição em todos os *pixels* que estão sob a extensão do *footprint*. Os valores ponderados são compostos no *buffer* de acumulação, utilizando o esquema de acumulação apropriado à ordem de caminhamento no volume.

Como a reconstrução é um processo aditivo e a visibilidade não, ao finalizar a projeção de uma fatia do volume sobre o *sheet buffer*, realizamos a etapa de composição deste buffer com a imagem já existente no *accumulation buffer*.

3 COMPRESSÃO DE DADOS VOLUMÉTRICOS USANDO A TRANSFORMADA DO COSSENO LOCAL

A utilização de dados volumétricos está presente e vem em crescente uso, tornando imperativo o desenvolvimento de técnicas que viabilizem o armazenamento e a transmissão destes dados em redes de computadores.

A compressão é o processo pelo qual se consegue reduzir a quantidade de dados necessária para representar uma certa quantidade de informação pela eliminação de informações redundantes e/ou irrelevantes.

Os métodos de compressão dividem-se em dois grandes grupos: os sem perdas ou reversíveis e os com perdas ou irreversíveis. Algoritmos sem perdas eliminam somente as informações redundantes, possibilitando, assim, a reconstrução total do dado original após o processo de descompressão. Já os algoritmos de compressão com perdas eliminam, além das informações redundantes, as informações irrelevantes, possibilitando somente a reconstrução aproximada do dado original após a descompressão.

3.1 Métodos de compressão sem perdas

Algumas aplicações exigem que o processo de compressão e descompressão seja livre de perdas de informação, como imagens médicas digitais, transmissão de textos e imagens binárias de *fac-símile*, programas executáveis, banco de dados etc. Em imagens médicas, caso haja perda de informação após a descompressão, pode haver comprometimento na precisão do diagnóstico. No caso de documentos de *fac-símile*, a exigência é de caráter legal.

Os algoritmos de compressão de dados sem perdas são divididos, basicamente, em duas categorias: os métodos baseados em dicionário ou universal e os métodos estatísticos. Os métodos baseados em dicionário ou universal geram um arquivo comprimido contendo códigos de comprimento fixo, normalmente, com 12 a 16 bits, onde cada código representa uma seqüência particular de valores dos dados originais, não necessitando do conhecimento da freqüência com que os símbolos fonte ocorrem na informação original (conhecimento estatístico). Os métodos estatísticos utilizam códigos de comprimentos variáveis.

O princípio básico desses algoritmos é a diminuição da redundância existente nos dados. Existem vários algoritmos de compressão sem perda, os quais são baseados na estrutura estatística dos dados. Os algoritmos de *Huffman* (HUFFMAN*,* 1952), Codificação Aritmética (RISSANEM, *1979*) e *Lempel-Ziv* (ZIV, 1977) estão entre os mais conhecidos. Estes métodos são adequados a uma grande variedade de aplicações, embora as taxas de compressão obtidas sejam pequenas tornando-os pouco atrativos.

Para que o algoritmo funcione de maneira eficiente, é desejável a existência de um modelo preciso para os dados, o que permite que a codificação defina precisamente os tamanhos dos códigos de cada símbolo. Essa separação do método de compressão sem perda em modelagem e codificação foi inicialmente sugerida em Rissanem (1981).

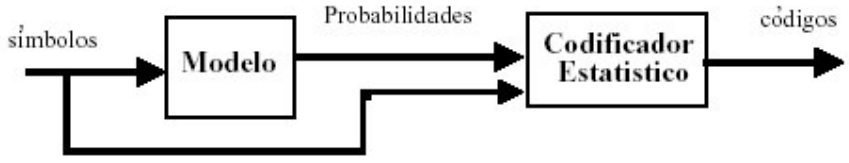


Figura 3.1 – Fluxo genérico de algoritmos de compressão sem perdas.

3.1.1 Código de *Huffman*

O Código de *Huffman* é um dos métodos clássicos de compressão de dados que utiliza as propriedades estatísticas dos dados para produzir os códigos para o conjunto de símbolos. Esses códigos variam de tamanho, sendo atribuídos códigos com mais bits para símbolos que possuem menor probabilidade de ocorrência e códigos menores para os símbolos com maior probabilidade. Esta simples idéia causa a redução do tamanho médio do código e, por conseguinte, o tamanho do dado codificado é menor que o dado original.

Esse algoritmo é baseado na construção de uma árvore binária, que guarda todos os símbolos existentes nos dados em suas folhas, com suas respectivas probabilidades de ocorrência ao lado.

O algoritmo que rege a construção da árvore de *Huffman* é definido pelos seguintes passos:

1. Cada símbolo presente nos dados forma uma árvore com somente um nó. Na raiz de cada árvore temos a probabilidade de ocorrência da árvore igual à soma das probabilidades das folhas da árvore.

2. As árvores são ordenadas de acordo com a sua probabilidade.

3. Duas árvores com as menores probabilidades são combinadas para formar uma nova árvore com um nó raiz como pai.

4. As árvores são continuamente reordenadas e combinadas até que uma única árvore binária contenha todos os símbolos dos dados.

5. O código representando um símbolo pode ser obtido saindo da raiz da árvore em direção à folha que o contém. Ao percorrer a árvore acrescentamos um 0

ao código quando descemos um ramo para a esquerda, e acrescentamos um 1, quando descemos à direita.

6. Com a utilização desse procedimento geramos um conjunto de códigos para os símbolos, cujos comprimentos refletem a sua distribuição de probabilidades de ocorrência.

Para que haja uma melhor compreensão do Código de *Huffman*, segue um exemplo que apresenta uma situação hipotética onde desejamos codificar a seqüência BANANA. Os símbolos a considerar e suas respectivas freqüências estão dispostos na Figura 3.2.

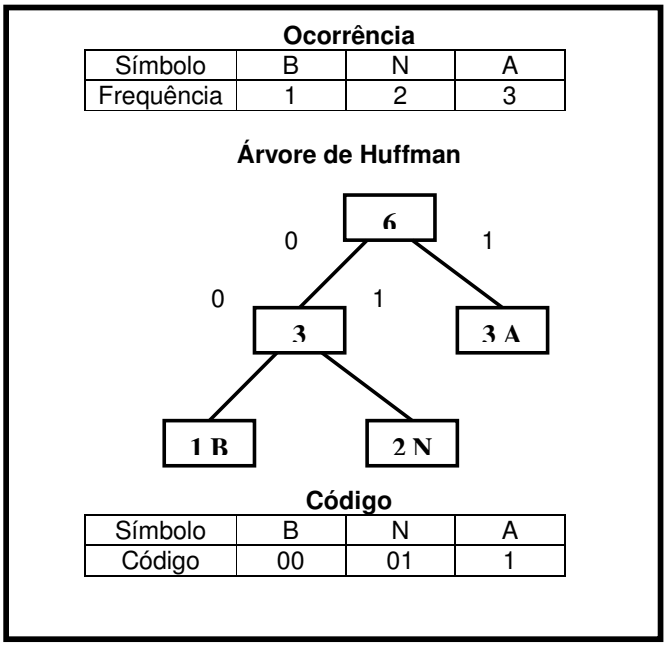


Figura 3.2 – Exemplo de codificação usando o algoritmo de *Huffman*

Seguindo os passos necessários para a codificação, o seguinte resultado é obtido: 001011011. Uma medida fundamental da qualidade do código é o comprimento do texto codificado, isto é, o número de dígitos binários na codificação. Nesse caso, o comprimento é de 9 bits. Enquanto a seqüência BANANA possui comprimento equivalente a 48 bits (cada símbolo corresponde a 8 bits).

3.1.2 Codificação Aritmética

Este método de codificação é baseado na idéia de que é mais eficiente gerar códigos para grupos de seqüências, que gerar um código para cada símbolo em uma seqüência. Como vantagens da codificação aritmética podemos citar:

- Sua otimalidade teórica, i.e. a geração de códigos para um símbolo com probabilidade *p*, que possua comprimento muito próximo de *log(p)*, o comprimento de código ótimo de acordo com Shannon (1948).
- Adequação aos casos onde existem probabilidade de aparecimento de símbolos muito desbalanceadas, como imagens de dois tons (preto e branco) ou coeficientes de transformadas.

Por outro lado, a codificação aritmética possui as seguintes desvantagens:

- É um algoritmo lento;
- Nem sempre gera um código prefixado;
- Precisa que se indique com um símbolo especial o final da seqüência;
- Possui pouca resistência ao aparecimento de erros.

A codificação aritmética pode ser conceitualmente dividida em duas fases. Na primeira fase é gerado um identificador único (*tag*) para uma dada seqüência de símbolos. Então, na fase subseqüente, é associado um código binário para esse identificador. Outra desvantagem deste método, decorrente da não garantia da geração de um código sem prefixos, é a impossibilidade de sua paralelização.

O algoritmo básico para gerar o identificador para uma seqüência de símbolos se inicia com um intervalo corrente com valor inicial definido para [0; 1). Para cada símbolo na mensagem, é associado um subintervalo com tamanho proporcional a sua probabilidade de aparecimento. Então, para cada símbolo na seqüência que está sendo codificada, o intervalo corrente é substituído pelo subintervalo associado ao símbolo corrente, sendo novamente subdividido em intervalos proporcionais às probabilidades dos símbolos. Podemos verificar que o intervalo final gerado para um seqüência particular de símbolos é disjunto de todos os outros intervalos que poderiam ser gerados para outras seqüências de símbolos. Podemos escolher qualquer número dentro desse intervalo pra representar a seqüência de símbolos que desejamos codificar. Em geral, escolhemos o limite inferior do intervalo ou o seu ponto médio. Agora, somente precisamos gerar um código binário para esse número.

Em Nelson (1991) é apresentado um exemplo de codificação aritmética com a mensagem “BILL GATES”. A distribuição de probabilidade é dada na Tabela 3.1.

Tabela 3.1 – Exemplo de codificação aritmética.

Caractere	Probabilidade	Intervalo
	1/10	0,0 – 1,0
Espaço	1/10	0,0 – 1,0
A	1/10	0,1 – 0,2
B	1/10	0,2 - 0,3
E	1/10	0,3 – 0,4
G	1/10	0,4 – 0,5
I	1/10	0,5 – 0,6
L	2/10	0,6 – 0,8
S	1/10	0,8 – 0,9
T	1/10	0,9 – 1,0

A cada símbolo na mensagem é associado um sub-intervalo do intervalo [0; 1). Assim, verificamos que, para identificar uma mensagem que comece com o

símbolo “B", o número que a representa deve estar no intervalo [0,2; 0,3). O próximo símbolo a ser codificado, letra “I", está associado ao sub-intervalo [0,5; 0,6), dentro do novo intervalo corrente [0,2; 0,3). Assim, o número que representa a mensagem “BI" está associado ao sub-intervalo entre 50% e 60% do intervalo corrente, ou seja, um número entre 0,25 e 0,26. Seguindo esse processo, obtemos os intervalos gerados durante a codificação de todos os símbolos da mensagem representados na Tabela 3.2 a seguir. Verificamos, pois, que o número 0,2572167752 codifica de maneira única a mensagem. É relativamente simples verificar como é feito o processo de decodificação. O primeiro símbolo é identificado como aquele cujo intervalo contém o número; no exemplo verificamos que o código está no intervalo entre 0,2 e 0,3, logo, o primeiro símbolo é “B". Então devemos retirar esse símbolo do código. Isto é feito subtraindo o limite inferior do intervalo associado a “B", o que resulta no número 0,0572167752. Então, dividimos o número resultante pelo comprimento do intervalo associado a “B" (0;1), o que resulta em 0,572167752. Para identificar o próximo símbolo, basta verificar em que intervalo esse número cai, no caso símbolo “I". Esse processo continua até ser encontrado um símbolo de marcação de final de mensagem.

A implementação básica da codificação aritmética possui dois problemas centrais: a definição do intervalo corrente requer o uso de muita precisão aritmética e, nenhuma saída é gerada até que a completa seqüência de símbolos seja codificada. A solução direta para esse problema é, a medida que conhecemos o valor de um *bit* que descreve o intervalo, colocarmos esse valor na saída. Podemos decidir que um *bit* pode ser colocado na saída quando o limite inferior do intervalo possuir o mesmo *bit* mais significativo que o limite superior. Após escrever esse *bit* já

conhecido na saída, podemos expandir o intervalo corrente de modo que ele represente somente a porção não conhecida do intervalo final.

Tabela 3.2 – Exemplo de codificação aritmética – Mensagem "BILL GATES".

Caractere	Limite Inferior	Limite Superior
	0,0	1,0
B	0,2	0,3
I	0,25	0,26
L	0,256	0,258
L	0,2572	0,2576
G	0,25720	0,257220
A	0,257216	0,2572168
T	0,2572164	0,2572168
E	0,25721676	0,257216776
S	0,2572167752	0,2572167756

3.2 Métodos de compressão com perdas

As técnicas de compressão com perdas envolvem a perda de parte da informação durante o processo de compressão dos dados, o que resulta na impossibilidade de reconstrução exata do dado original. Em muitas aplicações, as perdas no processo de compressão são aceitáveis. Isto resulta do fato do usuário do dado comprimido não perceber as perdas, ou delas não influenciarem a utilização que se fará do dado reconstruído. Esses algoritmos permitem o ajuste na fidelidade de reconstrução para possibilitar um aumento na taxa de compressão.

Esses métodos trabalham no sentido de reduzir parte dos dados que podem ser considerados irrelevantes para uma determinada aplicação. Quando aplicados a sinais, podemos verificar que os métodos de compressão com perda podem trabalhar no sentido de reduzir dois tipos de informação irrelevante, a redundância espacial (temporal) que é a correlação entre amostras vizinhas

espacialmente (temporalmente) no sinal e a redundância espectral que é correlação entre diferentes bandas espectrais.

Quando tratamos de compressão de sinais, temos basicamente os métodos preditivos e baseados em transformadas. Nos métodos preditivos, utilizamos a informação já codificada para prever valores futuros, e somente a diferença é codificada. Como essa codificação é aplicada no domínio espacial, é relativamente simples de implementar e se adapta de maneira eficiente às características locais do sinal. Um exemplo de algoritmo dessa classe é o DPCM (*Diferential Pulse Code Modulation*).

Por outro lado, os métodos baseados em transformadas inicialmente transformam o sinal do domínio espacial para um outro domínio, utilizando uma transformada conhecida, e nesse novo domínio realizam a codificação dos valores transformados (coeficientes). Esses métodos propiciam taxas de compressão maiores que os métodos preditivos, com a desvantagem de necessitarem de um esforço computacional maior.

A compressão baseada em transformadas fundamenta-se na idéia de que podemos transformar o sinal para um domínio apropriado (representação compacta), descartar os coeficientes que estão próximos de zero, quantizar com poucos *bits* os coeficientes pequenos e concentrar a representação nos coeficientes que contém a informação mais importante. Dessa maneira, ao reconstruir o objeto, a informação perdida foi a de menor importância.

A Figura 3.3 apresenta o fluxo básico de compressão baseada em transformada composto por três estágios. A transformação da representação do sinal para o domínio mais adequado ao processamento dos estágios seguintes, é o

primeiro e mais importante estágio. Em seguida temos os estágios de quantização e codificação.

Em geral, esses métodos envolvem uma divisão do sinal em blocos que são transformados separadamente, para simplificar o processo de transformação. Porém, essa blocagem pode causar considerável variação no valor reconstruído nas bordas dos blocos.

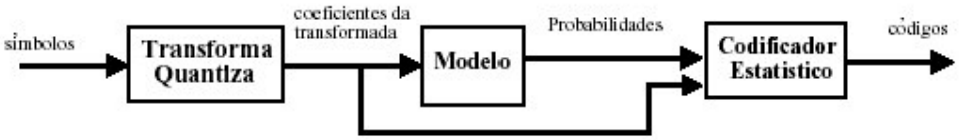


Figura 3.3 – Fluxo genérico de algoritmos de compressão baseados em transformadas.

Na compressão com perdas, temos um aumento na taxa de compressão proporcional a diminuição da qualidade da imagem reconstruída.

3.2.1 Compressão com a Transformada do Cosseno Local (LCT)

O ponto central nos métodos de compressão com perda é a escolha da transformada utilizada para obter a representação compacta do sinal. Em geral, são usadas transformadas lineares com as seguintes características:

- Base de funções fixa;
- Algoritmos eficientes para o cálculo;
- Separáveis;
- Características de redução da correlação entre os coeficientes;
- Boas propriedades de compactação da energia.

Uma transformada que possui base fixa de funções, um algoritmo eficiente para seu cálculo e boa compactação de energia é a transformada de *Fourier,* mas a mesma não possui suporte compacto. No entanto, a transformada discreta do cosseno (DCT) (AHMED; RAO, 1974) é uma das transformadas mais usadas para compressão. Essa popularidade se baseia no fato da DCT aproximar a transformada de *Karhunem-Loéve* (*KL*), a qual descorrelaciona todos os valores do sinal, mas demanda de muito esforço computacional por ser dependente dos dados.

A transformada dos cossenos locais é uma transformada com base ortonormal de classe C^{∞}, suporte compacto e que fornece uma representação em escala para os dados. Adicionalmente, essa transformada possui a vantagem de permitir que a partição do domínio espacial seja feita de maneira adaptativa, baseada nas características do sinal no domínio tempo x freqüência.

3.2.1.1 A Tranformada do Cosseno Local (LCT)

No estudo de uma representação para um sinal que apresente a menor autocorrelação possível, por exemplo, para usar essa representação em um método de compressão de sinais, podemos verificar que a correlação presente nos sinais é geralmente um fenômeno local. Assim, observamos que as bases de *Fourier* não são adequadas para esse tipo de representação, em razão das funções dessa base se estenderem por todo o domínio, ou seja, ser uma base de funções não locais.

Esta característica apresentada, resulta na necessidade de dividir o eixo do tempo em intervalos e calcular a base de *Fourier* separadamente, que propicia

uma perda de correlação entre os intervalos e o aparecimento dos chamados efeitos borda ou efeito blocagem.

A introdução das bases trigonométricas locais como funções da base em substituição à base de *Fourier*, as quais são multiplicadas por uma função janela diferenciável, conseguem assim maior correlação local na representação do sinal, minimizando o aparecimento dos efeitos borda e, adicionalmente, com liberdade para a escolha da partição do eixo do tempo que melhor represente as características do sinal.

Bases do Cosseno

Seja $f \in L^2[0;1]$ e $f(0) \neq f(1)$ uma função diferenciável, então a base de cosseno I pode atenuar os efeitos de borda através da restauração de uma extensão periódica $\hat{f}$ de *f* (Figura 3.4). Dessa forma, as altas freqüências introduzidas pela interpretação de uma mudança brusca nas bordas são menores que na base de *Fourier*.

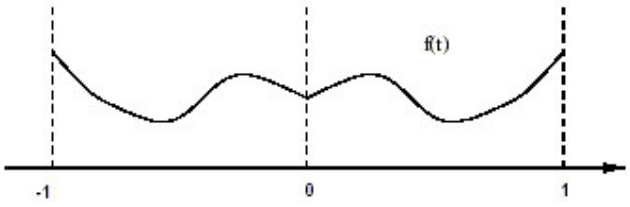


Figura 3.4: Extensão cosseno I para uma função *f*(t).

A função $\hat{f}$, que é simétrica em torno de 0, igual a *f* sobre o intervalo [0;1] e possui período 2, é dada por:

$$\hat{f}(t) = \begin{cases} f(t) & para\ t \in [0;1] \\ f(-t) & para\ t \in (-1;0] \end{cases} \qquad (3.1)$$

Outras bases de cosseno são geradas a partir de diferentes extensões de *f* sobre o intervalo [0,1]. Contudo neste trabalho, as bases de cossenos IV, serão as que demonstraremos maior interesse pois são utilizadas para construir bases de cossenos locais mais suaves, elas aparecem quando a função *f* é estendida em uma função $\hat{f}$ de período 4, simétrica em torno de 0 e antisimétrica em torno de 1 e –1.

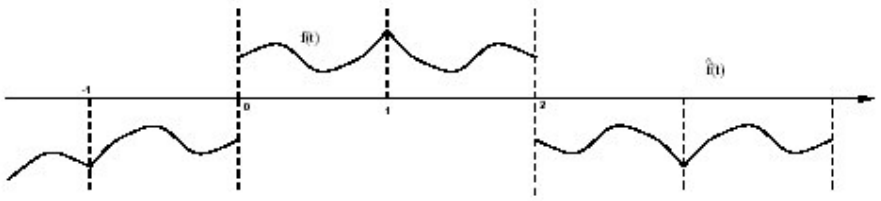


Figura 3.5: Extensão cosseno IV de uma função *f*(t).

Essas bases possuem problemas similares aos apresentados pelas bases de *Fourier*, os coeficientes de uma função suave possuem decaimento lento em altas freqüências. Isto acontece em razão da decomposição corresponder a uma extensão descontínua de *f* em cada intervalo. A importância dessas bases é dada por seu uso na construção de bases de cossenos locais que não apresentam esse problema.

A transformada associada com essa base é denominada DCT-IV (*Discrete Cosine IV Transform*), e a transformada discreta F(*u*) de uma função f(j), j=0,1,...,N - 1 é definida como:

$$F(u) = \sqrt{\frac{2}{N}} \sum_{j=0}^{N-1} f(j) \cos\left[\frac{\pi}{N}\left(j+\tfrac{1}{2}\right)\left(u+\tfrac{1}{2}\right)\right] \qquad u = 0,1,...,N-1. \tag{3.2}$$

A extensão para dimensões maiores que 1 pode ser feita observando que a transformada é separável. Seguindo essa abordagem, podemos afirmar que, para três dimensões, a transformada discreta do cosseno IV é definida por:

$$F(u,v,w) = \frac{2}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} f(i,j,k) \cos\left[\frac{\pi}{N}\left(i+\tfrac{1}{2}\right)\left(u+\tfrac{1}{2}\right)\right] \tag{3.3}$$

$$\cos\left[\frac{\pi}{N}\left(j+\tfrac{1}{2}\right)\left(v+\tfrac{1}{2}\right)\right]$$

$$\cos\left[\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(w+\tfrac{1}{2}\right)\right]$$

Em razão da propriedade de separabilidade, temos que qualquer transformada *k*-dimensional pode ser calculada através do cálculo de *k* aplicações sucessivas da transformada unidimensonal ou sua inversa. Logo, podemos calcular a transformada de um volume (transformada 3D) aplicando a DCT-IV unidimensional nas linhas do volume, nas colunas e ao longo de sua profundidade.

Uma das desvantagens desta transformada para compressão é o fato da mesma não possuir um coeficiente representando o valor médio do sinal transformado. Esta propriedade é denominada resolução de constante.

Bases de cossenos em Blocos

Bases de cossenos locais são bases em blocos que segmentam o eixo do tempo em intervalos, cujos tamanhos são adaptados às estruturas dos sinais. Esses blocos são obtidos através da multiplicação de janelas escaladas e transladadas ao longo desse eixo com as funções cosseno.

Bases ortogonais em blocos são obtidas dividindo o eixo do tempo em intervalos consecutivos $\left[a_p, a_{p+1}\right]$, com tamanhos arbitrários $l_p = a_{p+1} - a_p$. Dada a função $g = 1_{[0,1]}$, temos que um intervalo arbitrário é recoberto pela janela retangular escalada.

Essas bases em bloco são construídas com janelas retangulares descontínuas, as quais particionam o espaço em intervalos disjuntos, gerando

artefatos devidos a essas descontinuidades, diferenças na reconstrução em torno de um ponto oriundas de dois intervalos adjacentes naquele ponto. Para tentar evitar essas descontinuidades é necessária a utilização de janelas diferenciáveis com decaimento suave nas bordas.

Bases de Cossenos Locais

Malvar mostrou em (MALVAR, 1988) e (MALVAR, 1990) que era possível a criação de bases ortogonais discretas com janelas contínuas diferenciáveis moduladas por uma base cosseno-IV. Coifman e Meyer (COIFMAN *et al.* , 1991) redescobriram esse resultado para funções contínuas.

Seja I_p uma partição do eixo do tempo em intervalos que se sobrepõem uns aos outros:

$$I_p = \left[a_p - n_p, a_{p+1} + n_{p+1}\right], \tag{3.4}$$

podemos considerar que cada intervalo I_p é dividido em subintervalos. Dois desses subintervalos, O_p e O_{p+1}, representam a porção em que há sobreposição na borda direita e esquerda (Figura 3.6) e o terceiro representa um intervalo central C_p. Esses subintervalos podem ser definidos por:

$$\begin{aligned} O_p &= \left[a_p - n_p, a_p + n_p\right] \\ O_{p+1} &= \left[a_{p+1} - n_{p+1}, a_{p+1} + n_{p+1}\right] \\ C_p &= \left[a_p + n_p, a_{p+1} - n_{p+1}\right] \end{aligned} \tag{3.5}$$

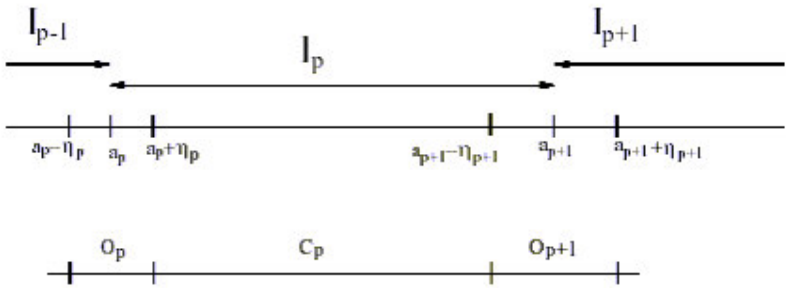


Figura 3.6: Intervalos sobrepostos recobrindo o eixo do tempo.

Podemos definir uma função de janelamento (também denominada função sino) g_p, cujo suporte é I_p, e que possui perfil decrescente em O_p e perfil crescente em O_{p+1} (Figura 3.7).

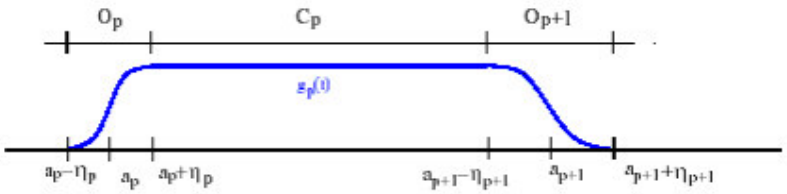


Figura 3.7: A função de janelamento *g(t)* e os subintervalos.

Agora, podemos definir uma base de cossenos locais de L^2 (R) como a base derivada da base de cosseno-IV de L^2[0,1], através da multiplicação de versões transladadas e escaladas de cada função da base de cossenos com a função de janelamento g_p (Figura 3.8).

Figura 3.8: As janelas sobrepostas dos intervalos *p*-1, *p* e *p*+1.

Uma base de cossenos locais pode ser representada graficamente como um conjunto de retângulos recobrindo todo o plano tempo x freqüência. Isto gera uma grade nesse plano, cujas células possuem tamanhos variando no tempo.

Para decompor uma função *f* na base cossenos locais, devem ser calculados os produtos internos da função com os vetores discretos da base (f, $g_{p,k}$). Um algoritmo rápido, introduzido em (MALVAR, 1992), subtitui os cálculos de (f, $g_{p,k}$) por cálculos de produtos internos na base original, que podem ser calculados com o algoritmo rápido da DCT-IV, adicionando um procedimento de dobragem (*folding*).

A operação de dobragem pode ser representada graficamente através do espelhamento da porção da função modulada (Figura 3.9b), que se encontra sobre o intervalo $[a_p - n_p, a_p]([a_{p+1}, a_{p+1} + n_{p+1}])$, em torno do eixo $t = a_p (t = a_{p+1})$ (Figura 3.9c), seguido da adição da porção espelhada à função com suporte no subintervalo $[a_p, a_p + n_p]([a_{p+1}, a_{p+1} + n_{p+1}])$ (Figura 3.9d).

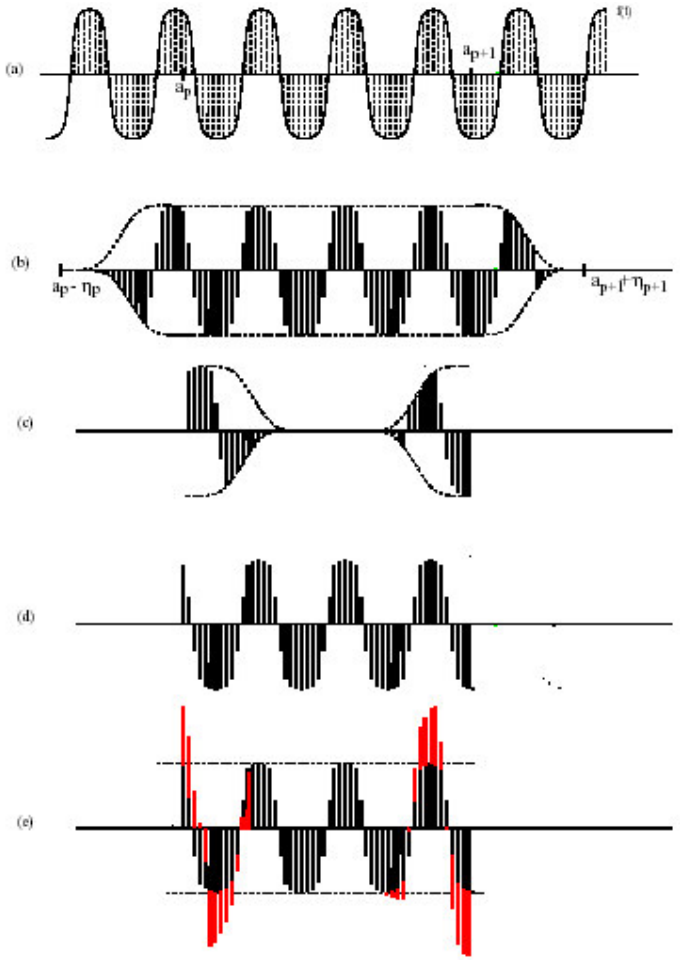


Figura 3.9: Representação gráfica da operação de dobragem. (a) A função *f*(t), que será dobrada. (b) A versão de *f*(t) modulada pela função de janelamento. (c) A porção espelhada da função *f*(t) modulada, que possuía suporte fora do intervalo. (d) A porção de *f*(t) com suporte no intervalo. (e) A porção espelhada, adicionada, à porção de *f*(t) com suporte no intervalo.

Assim verificamos que os coeficientes $\langle f, g_{p,k} \rangle$, que representam a transformada do cosseno local de *f*, podem ser calculados através da dobragem da função *f*, gerando a função f_{fold}. Em seguida, basta calcular o produto interno da função dobrada com a base ortogonal do cosseno-IV, o que pode ser feito através da utilização do algoritmo da transformada rápida do cosseno-IV.

A transformada inversa do cosseno local reconstrói a função *f* sobre $[a_p, a_{p+1}]$, a partir dos produtos internos da função dobrada f_{fold} com as funções da base de cossenos-IV. Aplicando a transformada rápida DCT-IV inversa, igual à transformada DCT-IV, reconstruímos a função f_{fold}, a qual, após a aplicação da operação de desdobramento, gera a função *f* reconstruída. É necessário aplicar a DCT-IV inversa nos blocos associados aos intervalos $[a_{p-1}, a_p]$, $[a_p, a_{p+1}]$ e $[a_{p+1}, a_{p+2}]$, antes de realizar a operação de desdobramento. Logo, para a completa reconstrução de um bloco do sinal é necessária a reconstrução parcial de seus vizinhos.

Podemos entender melhor o processo de cálculo da transformada inversa dos cossenos locais, verificando que a mesma é aplicada, para cada intervalo, em três etapas:

1. Aplicação da DCT-IV;
2. Desdobramento das porções sobrepostas do intervalo;
3. Adição da contribuição das porções sobrepostas.

Assim, verificamos que, para a completa reconstrução de um intervalo I_p, é necessária a aplicação das fases 1 e 2 nos intervalos vizinhos I_{p-1} e I_{p+1}.

3.2.1.2 Compressão de Volumes Baseado na LCT

A Figura 3.10 descreve o fluxo geral do método de compressão proposto em (Paiva, 2001), o qual segue um fluxo genérico de métodos baseados em transformação. O estágio inicial é a aplicação da transformada do cosseno local, gerando o volume transformado. Em seguida, temos dois estágios típicos de um codificador com perdas: quantização dos coeficientes restantes para restringi-los a um pequeno número de possibilidades; e, finalmente, a codificação dos dados transformados.

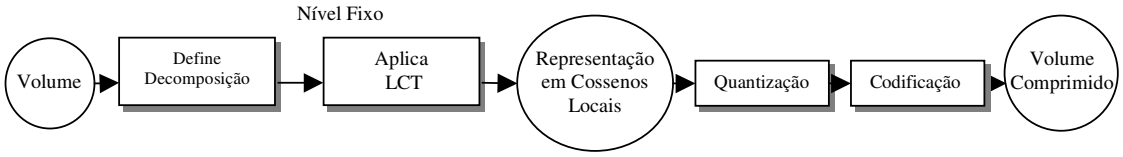


Figura 3.10: Visão geral do método de compressão de volumes baseado na transformada do cosseno local.

O estágio inicial do método de compressão é a definição da partição do volume para aplicação da transformada do cosseno local. Essa partição pode ser definida usando o método de nível de decomposição fixo, o qual subdivide o volume em subvolumes de tamanhos iguais (*nx, ny, nz*), ou usando o método baseado em decomposição adaptativa que utiliza o algoritmo de *best basis* (COIFMAN, 1992) para encontrar de maneira adaptativa a melhor subdivisão para representação do volume. Nesse caso, temos que a descrição da decomposição do volume deve ser codificada de uma maneira que seja possível ao método de descompressão recuperar o arranjo utilizado para os blocos da decomposição.

É desejável que a codificação do arranjo dos blocos permita a sua recuperação em qualquer ordem. Esta característica é fundamental para tornar

possível o desenvolvimento de algoritmos de visualização baseados na estrutura dos dados comprimidos, os quais descomprimem o volume localmente, minimizando os requisitos de memória para a visualização.

Usando a abordagem de método de decomposição fixo, especificamos o nível de decomposição diádica que será utilizado e o tamanho da porção sobreposta dos intervalos (*lap*). Com base nessas definições, o algoritmo calcula a partição do volume. Um volume de tamanho $N_x \times N_y \times N_z$ possui 2^{3j} blocos no nível *j* de decomposição diádica, cada bloco com tamanho $\frac{N_x}{2^j} \times \frac{N_y}{2^j} \times \frac{N_z}{2^j}$. O nível máximo de decomposição diádica $j_{máx}$ é determinado pelo tamanho da porção sobreposta das janelas de cossenos locais (*lap*), sendo calculado através da expressão $j_{máx} = \log_2(2 * lap).$

Para cada bloco resultante da decomposição do volume, aplicamos três transformadas de cossenos locais unidimensionais, uma ao longo de cada eixo, adicionando as contribuições de dobragem dos seis vizinhos. Seguimos a aplicação da transformada unidimensional com 2^j intervalos às colunas de cada fatia resultante da transformada anterior, seguido do mesmo processo aplicado às linhas de cada fatia resultante do processo precedente.

A segunda abordagem é baseada na busca por uma partição do volume que seja adaptada às características de freqüência do volume. Nesse método, é definido um valor para o tamanho do intervalo de sobreposição, o qual define o nível máximo de decomposição que pode ser utilizado. Em seguida, as transformadas do volume são armazenadas em um volume de blocos, cada bloco sendo um nó da árvore de cossenos locais. O processo é repetido para o nível de decomposição anterior. Assim temos dois volumes de nós da árvore de cossenos locais, um

representando um nível de decomposição *L* e o outro o nível *L*-1. Então para cada nó (b_i^L, b_j^L, b_k^L) do volume de blocos do nível *L*-1, calculamos o custo da representação e comparamos com seus oito filhos no nível *L*, os blocos:

$(b_{sonk}, b_{soni}, b_{sonj})$ $(b_{sonk}, b_{soni+1}, b_{sonj})$ $(b_{sonk}, b_{soni}, b_{sonj+1})$ $(b_{sonk}, b_{soni+1}, b_{sonj+1})$ $(b_{sonk}, b_{soni}, b_{sonj})$ $(b_{sonk}, b_{soni+1}, b_{sonj})$ $(b_{sonk}, b_{soni}, b_{sonj+1})$ $(b_{sonk}, b_{soni+1}, b_{sonj+1})$, onde $soni = 2*i, sonj = 2*j$ e $sonk = 2*k$.

Caso o somatório dos custos dos filhos seja menor que o custo do pai, então escolhemos a representação do volume que está associada aos nós da árvore de cossenos locais associados aos filhos. Isto significa retirar da árvore de cossenos locais o bloco de transformada associado ao nó pai, e substituir seu custo pelo somatório dos custos dos filhos. Caso contrário os filhos são descartados, tornando o nó pai uma folha da árvore. Ao final do processo temos uma *octree* recortada, com os nós que foram escolhidos para representar o volume localizados em suas folhas. Para o processo de compressão, realizamos um caminhamento em profundidade na *octree*, para cada nó da árvore descemos nas sub-árvores segundo uma ordem fixa, primeiro a fatia de blocos zero, em cada fatia, primeiro a linha zero e, em cada linha, primeiro a coluna zero até a última coluna. Em cada folha da árvore, o bloco de transformada associado é enviado para o processo de quantização/codificação. Em essência, esse é o funcionamento do algoritmo de compressão de volumes baseado em *best basis*.

Quantização e Codificação

O processo de quantização utilizado foi uniforme. Assim, foi definido um passo de quantização constante para cada bloco do volume, o qual foi calculado de

maneira semelhante à proposta em (CHIUEH *et al*, 1997). Para todos os elementos em um bloco, calculamos o intervalo dinâmico dos coeficientes através da expressão:

$$R_{bi,bj,bk} = \max Value_{bi,bj,bk} - \min Value_{bi,bj,bk}\,, \qquad (3.6)$$

determinado o valor do intervalo o passo seguinte é dado pela seguinte expressão de quantização

$$Q_{bi,bj,bk} = \frac{R_{bi,bj,bk}}{2^c}\,, \qquad (3.7)$$

com *c*, definido como uma constante fixa. Podemos definir a constante *c* como o número de bits que usaremos para representar os coeficientes quantizados da transformada antes da codificação. Cada um dos coeficientes é então quantizado utilizando a equação

$$F^Q(i,j,k) = [\frac{F(i,j,k)}{Q_{bi,bj,bk}} + 0.5] - F_{\min}\,, \qquad (3.8)$$

ou seja, o valor do coeficiente quantizado é resultante do arredondamento da razão entre o coeficiente original e o passo de quantização correspondente ao bloco em que se encontra, além de uma translação para colocar o coeficiente de menor valor no bloco (F_{min}) na origem.

Após a quantização, os coeficientes sofrem uma translação igual ao valor do menor coeficiente quantizado do bloco. Isto é necessário para obtermos todos os coeficientes como valores positivos, o que torna o algoritmo de codificação aritmética mais eficiente. Assim, para cada bloco do volume é também codificado o passo de quantização utilizado e o valor do menor coeficiente quantizado. O valor do passo de quantização é armazenado sem codificação, enquanto o coeficiente mínimo é codificado junto com os demais coeficientes.

A compressão se dá, além da quantização, através da manutenção de apenas um pequeno número de coeficientes de cada bloco. Para selecionar os coeficientes mais significantes seguimos a abordagem de Chen e Pratt (1984), que sugeriram que o caminhamento ao longo do bloco ocorresse segundo uma ordem em *zigzag*. A ordem em *zigzag* tridimensional é o caminhamento ao longo de um bloco de modo que $(u1, v1, w1)$ é visitado antes de $(u2, v2, w2)$ se $u1 + v1 + w1 < u2 + v2 + w2$; as tuplas (u, v, w) contidas no plano $u + v + w = K$ são percorridas segundo uma ordem *zigzag* bidimensional. A Figura 3.11 ilustra essa ordem de caminhamento ao longo dos blocos.

Essa abordagem é baseada na suposição que um grande número dos coeficientes encontrados por último, segundo essa ordem, será formada basicamente por zeros, isso porque, em geral, os coeficientes de alta freqüência possuem menor amplitude.

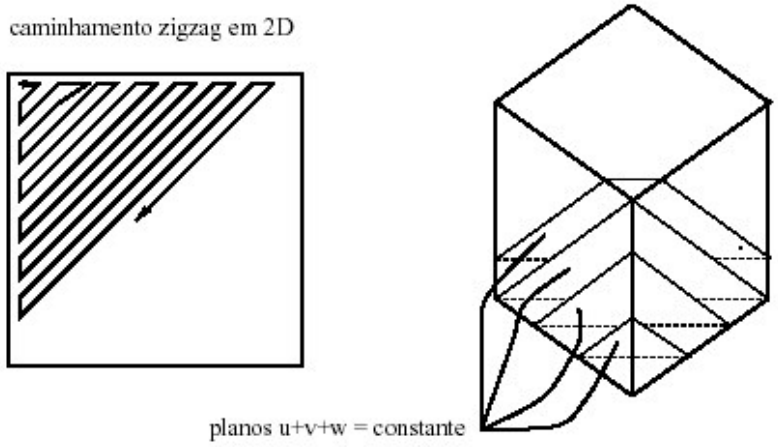


Figura 3.11: ilustração do caminhamento em *zigzag* em um bloco do volume.

A seqüência linear de coeficientes resultante do caminhamento em *zigzag* dos blocos da transformada é submetida a um corte, sendo mantidos apenas uma porcentagem dos coeficientes de mais baixa freqüência, definindo implicitamente os

demais como zero. Os coeficientes são, então, enviados para um processo de geração de códigos com baixa entropia por codificação aritmética ou codificação de *huffman*. A codificação aritmética utilizada é aplicada sobre todos os coeficientes retidos do volume transformado. Há uma primeira etapa em que um modelo estatístico de primeira ordem estima as freqüências de cada coeficiente, sendo essa etapa seguida pela geração dos códigos de cada coeficiente. Ao final de cada bloco é colocado um valor adicional para determinar fim de bloco. Já a codificação de *huffman* constrói uma árvore binária, que guarda todos os símbolos existentes nos dados em suas folhas, com suas respectivas probabilidades de ocorrência ao lado.

4 ALGORITMO DE VISUALIZAÇÃO DE VOLUMES COMPRIMIDOS COM A TRANSFORMADA DO COSSENO LOCAL

Este capítulo aborda a estratégia de adaptação de um algoritmo de visualização de dados volumétricos para a utilização dos dados comprimidos com a transformada do cosseno local, o que pode reduzir a necessidade de largura de banda para transmissão do dado e de memória necessária para a sua visualização.

Inicialmente descreveremos trabalhos já publicados sobre a visualização de dados volumétricos comprimidos. Em seguida é apresentado o formato dos dados volumétricos comprimidos segundo o esquema proposto em Paiva (2001) e o processo de visualização destes dados desde a descompressão até a visualização total do volume. Por fim apresentamos os resultados obtidos com o algoritmo implementado.

4.1 Trabalhos Correlatos

Ning e Hesselink (1993) propuseram um método que divide o volume em blocos, os quais são comprimidos usando quantização vetorial, e podem ser visualizados diretamente, sem necessidade de descompressão. No entanto, o processo de visualização usando a representação comprimida do volume resulta em um grande esforço de processamento, e a qualidade das imagens geradas não é garantida.

Na busca por uma representação comprimida do dado volumétrico que facilitasse também o processo de visualização, Whilhelms e Gelder (1994) realizaram algumas experiências usando representações hierárquicas, variações da

estrutura *octree*. No método proposto, cada nó da *octree* contém uma representação aproximada dos dados e, medidas do erro dessa representação e da importância dessa região na visualização do volume. O caminhamento na *octree* permite a visualização dos dados sem um grande acréscimo no tempo de processamento, através da possibilidade de visualização de regiões do volume como um único objeto, possível quando o erro dessa aproximação é baixo ou a importância da região não é grande. Esta representação obtém compressão quando alguns ramos da *octree* são cortados com base nos critérios de importância.

Em (WESTERMANN, 1994) é apresentada uma abordagem para a visualização de volumes baseada na transformada de *wavelets*, demonstrando como resolver a integral de *rendering* diretamente no domínio das freqüências. Assim, a visualização pode ser realizada sem necessidade de calcular a transformada inversa do volume. Também são apresentadas técnicas para aceleração do algoritmo de visualização baseadas nas propriedades de multiresolução da representação por *wavelets*. Foi verificado que o tempo de visualização depende diretamente do suporte da *wavelet* utilizada, o que também influencia significativamente a capacidade de compressão da transformada. Logo, para altas taxas de compressão, o tempo para a visualização não é satisfatório.

Em Yeo e Liu (1995) foi proposto um esquema para a visualização de dados volumétricos a partir de uma representação baseada na blocagem dos dados, e na descompressão dos blocos à medida que são necessários. Essa abordagem propicia uma utilização eficiente da memória, permitindo a visualização de volumes que de outro modo não caberiam na memória disponível. O esquema de compressão dos dados é uma adaptação para 3D do algoritmo JPEG, sendo baseado na DCT e na codificação diferencial seguida por codificação de *Huffman*

para os coeficientes da transformada. Esse método apresenta boas taxas de compressão e tempo razoável para descompressão e visualização do volume.

Outra abordagem para visualização de dados volumétricos usando descompressão local foi proposta em Haley e Blake (1996). Nesse trabalho foi proposto um algoritmo para a visualização incremental do dado volumétrico, representado em uma estrutura hierárquica (*octree*), como alternativa para a redução dos requisitos de memória do algoritmo de visualização volumétrica *Shear Warping* (LACROUTE; LEVOY, 1995).

> Em Ihm e Park (1998) é proposto um esquema que se baseia na partição do volume em blocos de 163 *voxels*, os quais são transformados para uma representação de *wavelets*. Em seguida, os coeficientes são submetidos a um processo de limiarização, e codificação. Na codificação, os coeficientes são organizados de modo a permitir o acesso randômico a qualquer coeficiente do bloco. Isto favorece o desenvolvimento de algoritmos de visualização adequados à estrutura do dado comprimido, os quais necessitam apenas da descompressão local do volume.

Seguindo essa linha de trabalho Kim e Shin (1999) desenvolveram um método de compressão que permite a visualização de grandes volumes de dados, os quais são divididos em blocos de 83 *voxels*. Em seguida, os coeficientes são normalizados e re-ordenados de acordo com a ordem de reconstrução.

Em Rodler (2002) foi desenvolvido o uso de eficientes métodos de armazenamento de dados espaciais combinados com rápida recuperação baseado na transformada de *wavelets* e no algoritmo de codificação por subbandas tridimensionais desenvolvido para codificação de seqüências de vídeos por Chen e

Pearlman (1996). O algoritmo proposto possibilita o acesso randômico aos *voxels* do volume, e altas taxas de compressão.

Li e Kaufman (2003) propuseram um algoritmo chamado *box-growing* que particiona os espaços vazios da textura. A idéia básica é agrupar os *voxels* com propriedades similares com mesmo domínio do volume e função de transferência no mesmo sub-volume. Cada sub-textura inclui *voxels* vizinhos com densidades e magnitudes de gradientes similares. Essas sub-texturas são empacotadas dentro de maiores para reduzir o número de texturas. A partição e o empacotamento são independentes da função de transferência. Durante o *rendering*, a visibilidade dos boxes são determinadas caso qualquer *voxel* incluso é assumido valor não zero de opacidade através da função de transferência. Apenas as sub-texturas dos boxes visíveis são misturadas e apenas pacotes de texturas contendo sub-texturas visíveis residem na memória. As densidades e os gradientes são organizadas em texturas separadas para evitar o armazenar gradientes de texturas em regiões vazias, independente da função de transferência. A partição e o empacotamento podem ser consideradas como uma compressão baseada em textura sem perdas com uma taxa de compressão média de 3,1:1 para os gradientes de textura.

Schneider e Westermann (2003) apresentam um método de compressão volumétrica com aceleração de hardware que decodifica e renderiza os dados volumétricos. Portanto, foi desenvolvido um esquema de quantização hierárquico baseado na análise de covariância de multiresolução do campo original capaz de codificar eficientemente dados estáticos e multidimensionais que variam com o tempo, permitindo assim eficientes codificações de dados em grande escala. A decodificação e o rendering de dados volumétricos comprimidos podem ser feitos através de programação de hardwares usando processadores gráficos. Desta forma

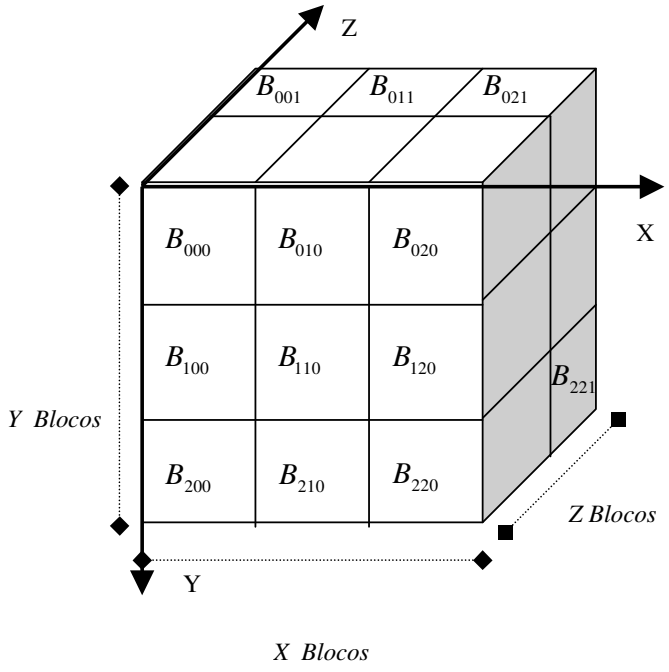
a transferência entre a CPU e a unidade de processamento gráfico (GPU) pode ser minimizado habilitando memória flexível e eficiente, para as opções de *rendering* em tempo-real.

Krishnan *et al* (2004) apresentam um esquema para compressão de grandes dados volumétricos usando *rendering* interativo em ambientes distribuídos que explora as distorções das propriedades de escalas e de multi-resolução apresentadas em JPEG2000. O cliente interativo apresenta aumento nas escalas de resolução, posição e progressiva melhoria da qualidade. O servidor explora as localidades de espaço oferecidas pela Transformada Discreta de Wavelet (DWT) e empacota as informações indexadas para transmitir, assim que possível, os dados volumétricos comprimidos relevantes aos clientes. Uma vez que os clientes identificam o volume de interesse (VOI), o volume é progressivamente refinado dentro do VOI. Foi utilizado como experimento um segmento de um dado volumétrico obtido através de CT de um abdômen com o tamanho de 512x512x256 *voxels* obtido do *Siemens Medical Solutions.*

Verificamos que a estratégia de blocagem dos dados volumétricos é amplamente utilizada, e que em razão disto inúmeros métodos de descompressão sofrem do conhecido efeito de blocagem, que é o problema apresentado pela divisão do volume em blocos para realizar a compressão do volume e ao reconstruir os blocos ocorre perda de correlação entre os blocos adjacentes. Em função disto e da potencialidade do algoritmo de compressão baseado na LCT pode resultar em utilização de dados volumétricos comprimidos desde que seja desenvolvida uma estratégia eficiente para sua compressão e visualização.

4.2 Estrutura do Dado Volumétrico Comprimido

Segundo o esquema de compressão utilizado os dados volumétricos comprimidos estão armazenados em blocos segundo a ordem de caminhamento natural dos blocos (primeiro eixo x, segundo eixo y, depois eixo z) conforme Figura 4.1. Cada bloco tem suas dimensões definidas pelo nível de decomposição diádica, conforme descrito na Seção 3.2.1.2.



B_{000}	B_{010}	B_{020}	B_{100}	B_{110}	B_{120}	B_{200}	B_{210}	B_{220}	$\cdots$	$B_{X,Y-1,Z}$	$B_{X,Y,Z}$

Figura 4.1 – Ordem de Armazenamento dos blocos do volume comprimido.

Cada bloco a ser salvo no arquivo comprimido é submetido primeiro ao cálculo da LCT. Em seguida os *voxels* do bloco transformado são ordenados através do caminhamento do mesmo segundo a ordem *zigzag* 3D (conforme descrito na

Seção 3.2.1.2). Para aumentar a taxa de compressão podem ser armazenados apenas um percentual dos primeiros coeficientes da transformada, os quais são considerados os mais significantes. Assim, cada bloco comprimido tem um percentual de coeficientes retidos associado a ele. Esta seqüência de coeficientes é quantizada e em seguida codificada por um algoritmo de compressão sem perdas (Codificação de *Huffman* ou Codificação Aritmética).

Verificamos assim, que após o final do processo de codificação a seqüência de *bits* que representa cada bloco comprimido do volume terá um tamanho variável, dependente do processo. O tamanho dos blocos são então armazenados em um arquivo de configuração juntamente com o endereço inicial de cada um, em bytes. A Figura 4.2 apresenta uma representação do arquivo gerado onde Sb representa o tamanho de cada bloco no volume comprimido.

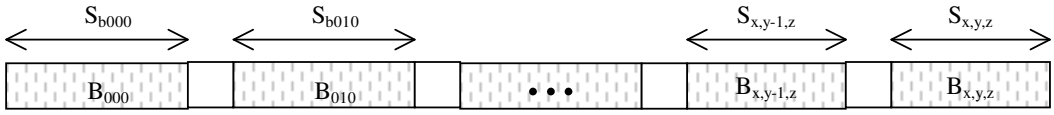


Figura 4.2 – Separador de blocos (*offset*).

4.3 Descompressão e Visualização

O algoritmo de visualização necessita percorrer o volume em uma ordem que depende da posição do observador. Para determinar esta ordem na qual deve ser percorrido o volume, devemos levar em consideração a direção de visualização, que é dada pelo vetor Normal ao Plano de Visualização (VPN). As coordenadas do VPN identificam qual a direção para o qual o plano de visualização está voltado, determinando assim a ordem correta de composição a qual por sua vez, define a ordem de caminhamento do volume.

O passo seguinte será descomprimir os blocos no arquivo comprimido, os quais, conforme apresentado na Figura 4.2, estão armazenados em uma seqüência de *bits* codificados. Para iniciar o processo de visualização é necessário descomprimir e reconstruir esse arquivo, contudo é importante ressaltar que a compressão gera um conjunto de informações necessárias à descompressão, tais como: número de *voxels* em cada direção, *lap*, tamanho de cada bloco comprimido, número de *bits* por coeficiente, percentual de coeficientes retidos, algoritmo de codificação utilizado e valores máximo, mínimo e endereço inicial de cada bloco.

Com esses parâmetros o processo de descompressão inicia com a alocação de espaço de memória M para duas fatias de blocos de *voxels,* onde $M = (BS_x \times BS_y \times BS_z)$ *voxels,* e BS_x é o número de *voxels* do bloco na direção x, BS_y o número de *voxels* do bloco na direção y e BS_z o número de *voxels* do bloco na direção z. A estrutura *BlockOffSet* é responsável por receber os parâmetros do necessários a descompressão, nela são alocados os valores do maior coeficiente (*max*) e do menor coeficiente (*min*) do bloco, a seqüência de coeficientes do bloco (armazenados na estrutura *TFVolB *b*) e a posição inicial do bloco no arquivo (*offSetInicial*). Uma outra estrutura, chamada *Uncomp* recebe os parâmetros tamanho do bloco (*blocksize*), *lap*, percentual de coeficientes retidos (*percent*), número de voxels em cada direção (v_x, v_y, v_z) e a codificação utilizada(*cod*), conforme Figura 4.3.

```
typedef struct
{
int bi,bj,bk;
long int offSetInicial;
float max,min;
TFVolB *b;
bool aberto;
bool DCT_IS, DCT_Col, DCT_Lin;
}BlockOffSet;
```

```
typedef struct {
 char nome[100], nome_def[100];
 int nb, lap, blocksize, t_stream, nbits, cod,
 percent, nx,ny,nz, vx,vy,vz, tam_seq, header;
}Uncomp;
```

Figura 4.3 – Estruturas *BlockOffSet* e *Uncomp.*

Conhecido o tipo de codificação utilizada, realizamos o processo de decodificação do bloco. O bloco decodificado é armazenado em um vetor que contém os valores dos coeficientes da transformada, o qual em seguida é submetido ao processo inverso da quantização através da expressão:

$$F(i,j,k) = F^{Q}(i,j,k) \times Q_{bi,bj,bk} \qquad (4.1)$$

onde, $F(i,j,k)$ é o valor do coeficiente reconstruído, $F^{Q}(i,j,k)$ é o valor do coeficiente quantizado e $Q_{bi,bj,bk}$ é o passo de quantização correspondente ao bloco em que se encontra.

Para descomprimir um bloco na posição (i,j,k) é necessário calcular a DCT-IV e o desdobramento nos seus seis vizinhos. Como a LCT 3D e sua inversa são aplicadas através do cálculo de três transformadas unidimensionais, temos que realizar esse processo ao longo da três direções. Inicialmente é aplicado a DCT-IV e o desdobramento ao longo da profundidade do volume nos blocos (b_i, b_j), (b_i, b_{j+1}), (b_{i+1}, b_j) e (b_{i+1}, b_{j+1}) das fatias k e k+1. Em seguida são adicionadas as contribuições

entre os blocos adjacentes, $lap^{-}_{i,j,k+1}$ é adicionado ao bloco (b_i, b_j, b_k), conforme Figura 4.4.

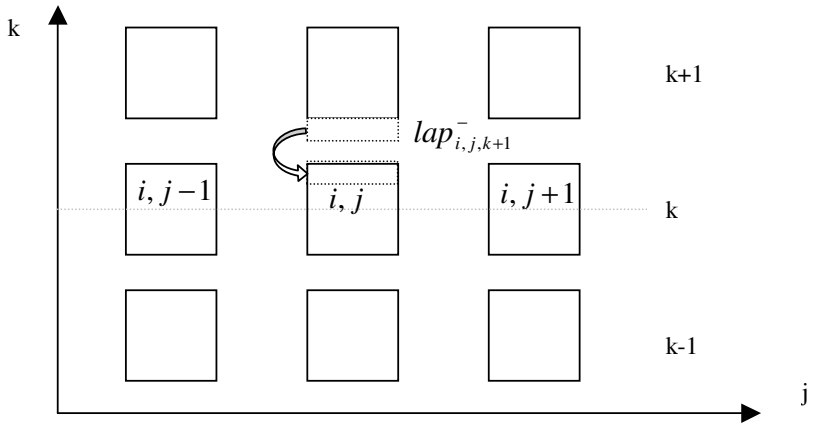


Figura 4.4 – Descompressão do bloco (i, j, k) na direção k.

Então, aplicamos a DCT-IV e o desdobramento ao longo da direção y nos blocos (b_i, b_j), (b_{i+1}, b_j), (b_i, b_{j+1}) e (b_{i+1}, b_{j+1}) da fatia k, adicionando as contribuições entre os blocos adjacentes ao longo de y. Finalmente, é aplicada a DCT-IV e o desdobramento ao longo de x nos blocos (b_i, b_j) e (b_i, b_{j+1}) da fatia k, e, logo após, é adicionada as contribuições entre esses blocos. Assim, o bloco (b_i, b_j, b_k) está completamente reconstruído podendo ser, por exemplo, enviado para um processo de visualização. Além disto, outros blocos envolvidos na reconstrução do bloco (b_i, b_j, b_k) estão parcialmente reconstruídos. Podemos verificar, para exemplificar, que o bloco (b_{i+1}, b_j, b_k) já foi submetido a duas transformadas de cossenos locais unidimensionais, faltando para sua completa reconstrução apenas a aplicação da transformada ao longo de x.

O cálculo da transformada inversa do cosseno necessita, portanto, de memória para manter os blocos parcialmente reconstruídos. No pior caso essa

memória é equivalente a duas fatias de blocos, ou seja, duas vezes a resolução das fatias do volume vezes o número de fatias em cada bloco.

Após descomprimido o bloco é individualmente enviado para o processo de visualização, através do algoritmo de visualização *Splatting* proposto em (WESTOVER, 1990), que é do modelo que segue a ordem de percorrimento do objeto, cada elemento do volume é mapeado em uma posição da tela, ou melhor, provisoriamente em um *buffer* de memória denominado tabela de *footprint* e têm suas participações na formação da imagem gradualmente compostas através de pesos, que são, na verdade, suas opacidades e distâncias com relação ao observador.

4.4 Resultados Obtidos

Para realizar os testes do algoritmo proposto foi utilizado um volume do projeto *Visible Man* que foi o utilizado em (IHM; PARK, 1998), (RODLER, 1999) e (KIM; SHIN, 1999). O volume é uma parte do volume original com 512x512x256 *voxels*, reamostrado a partir dos dados de CT adquiridos com o cadáver fresco. Esta reamostragem se fez necessária em razão da existência de diferentes espaçamentos entre fatias e tamanhos de *pixels* nos dados originais. Os valores dos *voxels* variam no intervalo [0, 4095] (12 *bits*), armazenados em 16 *bits*. A Figura 4.5 representa uma imagem do dado volumétrico utilizado.

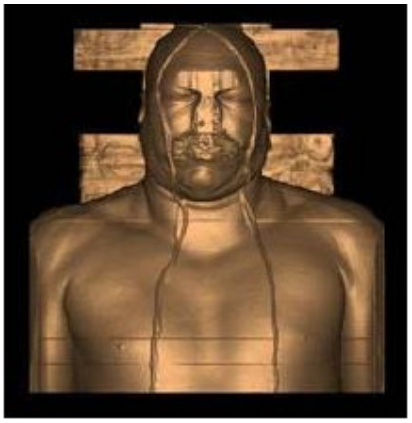


Figura 4.5 - Imagem do volume utilizado.

O algoritmo proposto foi implementado utilizando a linguagem C++ e a biblioteca de interface FLTK (*Fast Light Toolkit)* (FLTK, 2005). A plataforma para a devida aquisição dos dados foi um computador Pentium IV com 1.6Ghz e 256Mb de Ram, em ambiente operacional *Linux*.

Nos testes realizados foram quantificados alguns parâmetros com o propósito de verificar a qualidade e capacidade do algoritmo proposto. Foi quantificado o tempo de visualização total (TVT), que contempla o tempo que o algoritmo leva para carregar os dados para a memória e visualizá-los. Medimos também o tempo de descompressão (TD), que representa o tempo que o algoritmo leva para carregar os dados comprimidos do disco, descomprimir todos os blocos do volume e reconstruir os coeficientes. O tempo de visualização puro (TVP) é uma outra medida importante, pois representa o tempo gasto somente para visualizar o volume descomprimido. Outras duas medidas são: a taxa de compressão que representa o quanto foi comprimido o volume, sendo calculada dividindo o tamanho original do volume pelo tamanho do volume comprimido; e a razão sinal-ruído (SNR) que representa uma medida do ruído acrescentado aos dados pelo processo de compressão, é calculada dividindo a média dos quadrados dos valores do sinal

original pelo erro médio quadrático. O tempo de visualização do dado não comprimido (VSC) é também representado nos gráficos com curvas de tempo.

Como variáveis de controle o algoritmo utiliza: O tipo de codificação, que pode ser a codificação de *Huffman* ou a codificação Aritmética, o número de coeficientes retidos após o processo de quantização, variando de 0% a 100% em intervalos de 25%; o *blocksize* ou tamanho dos blocos (BS); e o *lap*, parâmetro utilizado pela função de dobragem da transformada conforme mencionado na Seção 3.2.1.2.

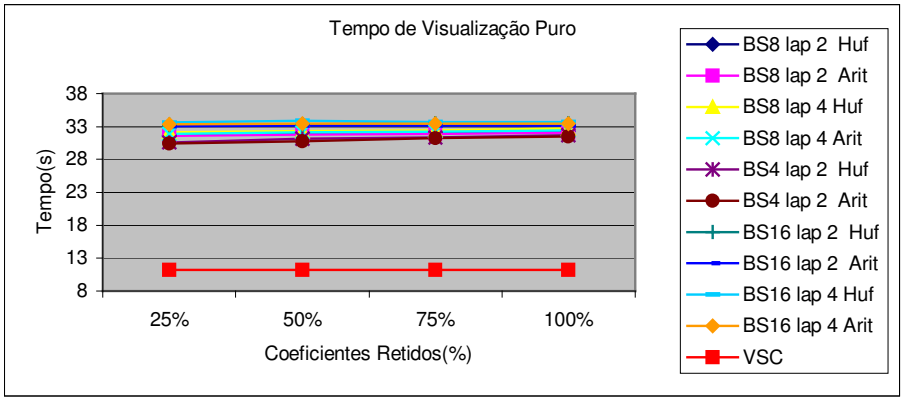


Figura 4.6 – Gráfico de Tempo de Visualização Puro.

A Figura 4.6 apresenta um gráfico com resultados do tempo de visualização puro obtidos variando os parâmetros mencionados, este gráfico apresenta as curvas de Tempo x Percentual de Coeficientes Retidos. As curvas diferem segundo os parâmetros: tamanho dos blocos (BS4, BS8 e BS16) onde BS4 representa blocos com 8 x 8 x 4 *voxels*[3], BS8 blocos com 16 x 16 x 8 *voxels* e BS16 blocos com 32 x 32 x 16 *voxels*; *lap;* codificação utilizada e os coeficientes retidos. O tempo de visualização puro sofre influência insignificante dos parâmetros da

[3] Em razão do volume ter dimensões diferentes ao longo dos eixos e ter sido usada uma implementação da LCT, que utiliza partição diádica dos intervalos, os blocos possuem números de *voxels* distintos em cada direção.

compressão apresentando um desvio padrão calculado de 1,13s em relação a média dos tempos.

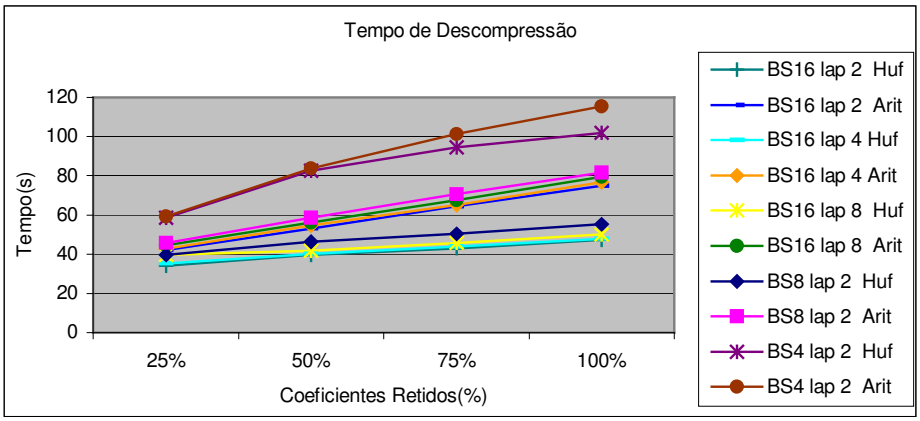


Figura 4.7 – Gráfico de Tempo de Descompressão.

Quando medimos os tempos de descompressão (Figura 4.7), os resultados indicam que quanto menor o tamanho dos blocos maior o tempo necessário para descompressão de todo o volume, isso se deve ao fato da diminuição do tamanho dos blocos resultar em aumento do número de blocos e no processo de descompressão existirem determinadas operações relacionadas a inicialização e finalização do processo que independem do tamanho do bloco. A codificação utilizada também influencia no tempo de descompressão, pois conforme descrito na literatura, verificamos que a codificação de *Huffmam* é mais rápida que a codificação Aritmética, apesar desta obter melhores resultados na taxa de compressão. O número de coeficientes retidos também influenciam no resultado pois quanto maior o número de coeficientes retidos maior é o tamanho do bloco comprimido e mais tempo é consumido para transferência do disco para a memória.

Foi verificado também que, quanto maior o *lap* maior o tempo necessário para descompressão, isto está ligado diretamente ao número de *voxels* que é

utilizado no processo de desdobramento dos coeficientes na aplicação da LCT inversa.

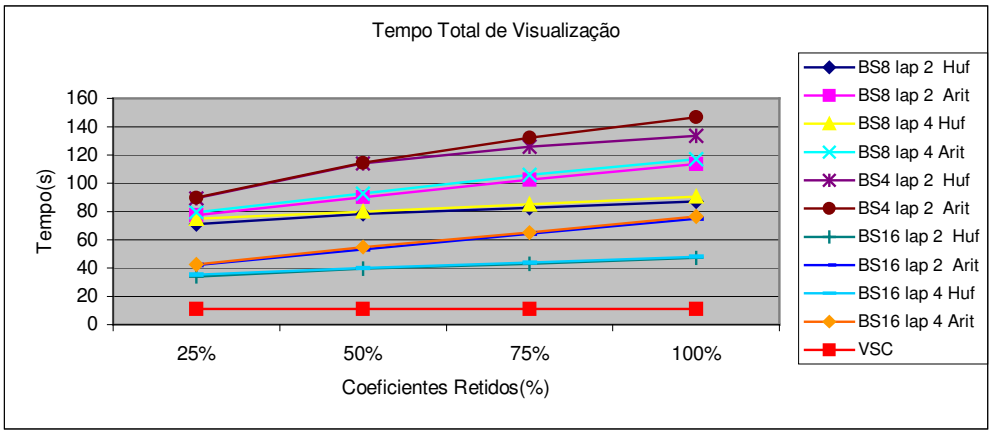


Figura 4.8 – Gráfico de Tempo Total de Visualização.

Resultados semelhantes são apresentados quando analisamos o tempo total de visualização (Figura 4.8), que contempla o tempo de visualização puro e o tempo de descompressão. Contudo quando analisamos o tempo de visualização puro notamos que o tempo aumenta cada vez que aumenta o tamanho do bloco e em relação ao tempo de descompressão o tempo diminui quando aumentamos o tamanho do bloco, por outro lado esse tempo medido no processo de descompressão é relativamente bem maior em relação ao tempo de visualização puro, pois enquanto a variação em percentual é de 11% para tempo de visualização, para o tempo de descompressão é de 137% em relação a valores médios. Isso significa que os resultados obtidos no tempo total de visualização apresentam curvas que coincidem mais com as do tempo de descompressão por sofrerem uma maior influência.

Conforme o gráfico da Figura 4.8, verificamos que quanto maior o tamanho dos blocos menor o número de blocos e conseqüentemente, menor o tempo de visualização total, pelo mesmo motivo descrito nos resultados do gráfico da Figura 4.7, a quantidade de acesso a disco que o número de blocos provoca para

a aquisição dos dados do arquivo comprimido. Outra variável que influencia consideravelmente é o percentual de coeficientes retidos, pois apesar de no gráfico tempo de visualização puro, o número de coeficientes retidos não ter influenciado nos tempos medidos, esta variável influenciou nos tempos de descompressão, influenciando assim no tempo de visualização total. A codificação de *Huffmam* que apresenta em sua estrutura um processo mais simples que a Aritmética e codifica os dados volumétricos com tempo um pouco menor que a codificação Aritmética e o *lap* que apresenta um pequeno aumento a medida que aumentamos o seu tamanho.

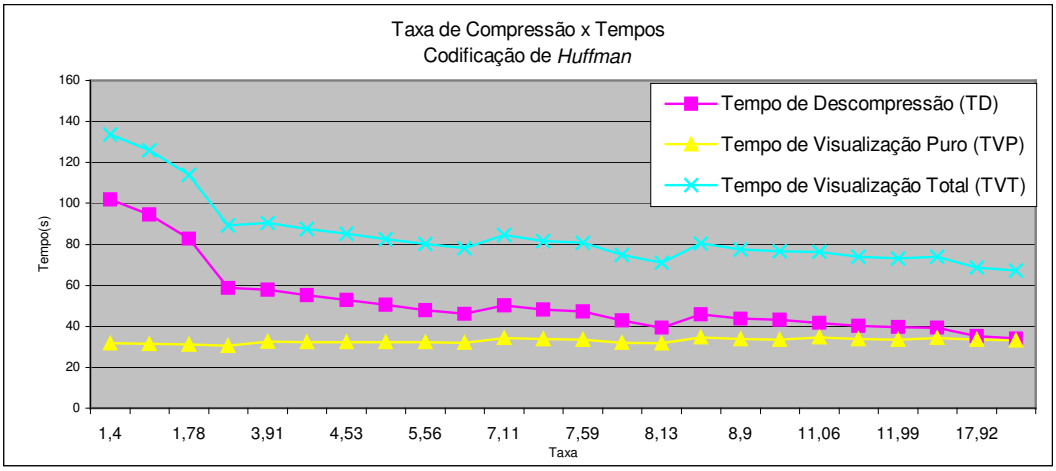


Figura 4.9 – Gráfico Taxa de Compressão x Tempos de Descompressão e Visualização (Codificação de *Huffman*).

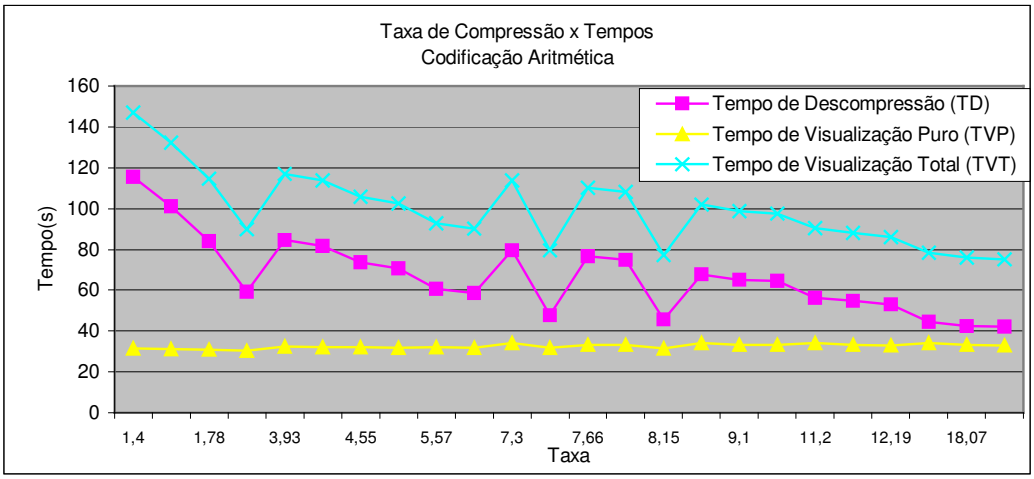


Figura 4.10 – Gráfico Taxa de Compressão x Tempos de Descompressão e Visualização (Codificação Aritmética).

Os gráficos das Figuras 4.9 e 4.10 apresentam a variação dos tempos em relação às taxas de compressão. Percebemos que o tempo de visualização puro apresenta uma curva constante comprovando que os parâmetros de compressão não influenciam nos tempos de visualização puro.

Em relação aos tempos de descompressão e de visualização total as taxas de compressão diminuem à medida que diminui o percentual de coeficientes retidos, o *lap* e o tamanho dos blocos. Essa variação da taxa é inversamente proporcional aos tempos, pois quanto maior a taxa de compressão menor o tempo gasto, fato este que ocorre devido principalmente ao número de coeficientes retidos, o qual diminui o tempo de carregamento dos dados para a memória.

Os gráficos apresentados mostram também, que o *lap* e o número de coeficientes retidos influencia mais nas curvas do gráfico da codificação aritmética que nas curvas do gráfico da codificação de *Huffman*. As variações apresentadas no gráfico 4.10 decorrem da influência do *lap*, pois quanto maior o *lap,* mantendo o percentual de coeficientes retidos iguais, maior o tempo de descompressão.

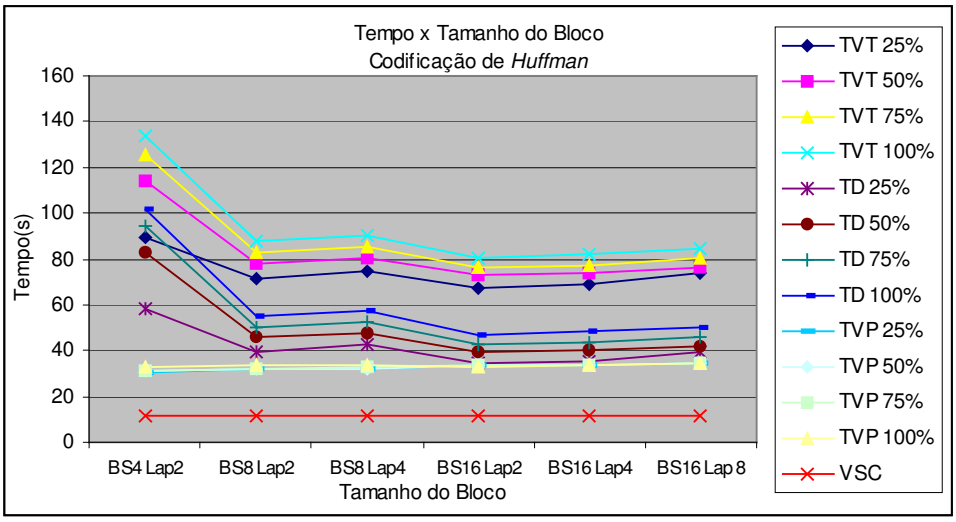


Figura 4.11 – Gráfico de Tamanho dos Blocos x Tempos.

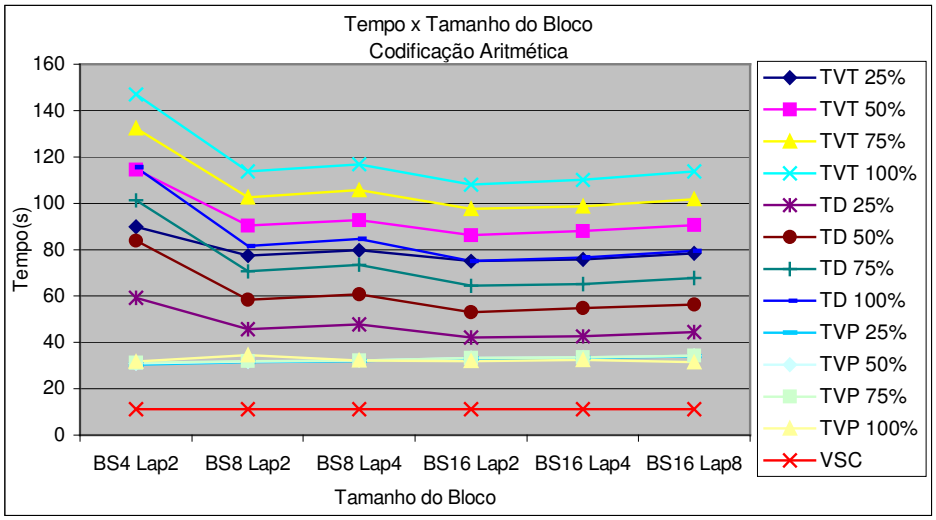


Figura 4.12 – Gráfico de Tamanho dos Blocos x Tempos.

Os gráficos das Figuras 4.11 e 4.12 apresentam as curvas do tempo x tamanho do bloco. Como resultado, verificamos que o tempo de visualização puro não varia com nenhum parâmetro de compressão. Entretanto o tempo de descompressão apresenta resultados diferentes em relação a contribuição dos parâmetros da compressão, como já esperado e mencionado, o tempo de descompressão aumenta a medida que o número de blocos, percentual de coeficientes retidos e o *lap* aumentam e também em funçao da codificação utilizada. Os gráficos das Figuras 4.11 e 4.12 apresentam ainda o uma característica entre as curvas dos tempos medidos, percebemos que o tempo de visualização puro acrescido do tempo de descompressão resulta exatamente no tempo de visualização total.

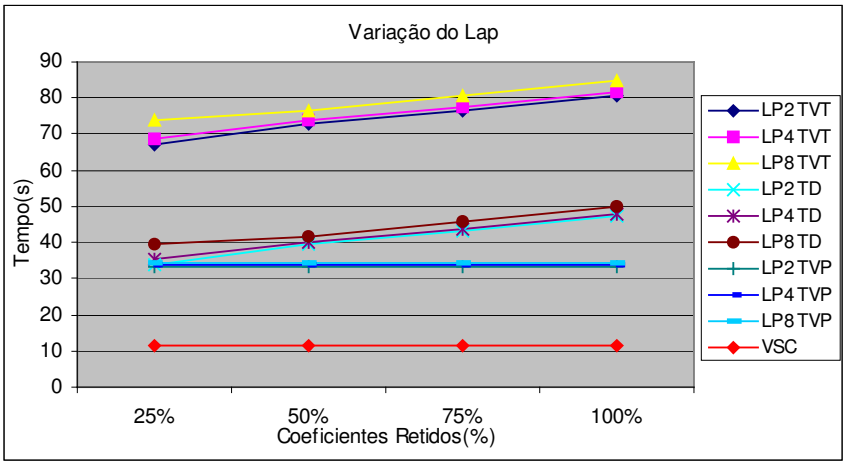


Figura 4.13 – Gráfico com a variação do *lap.*

O gráfico da Figura 4.13, possui em sua representação curvas com a dos tempos de visualização total, tempos de visualização puro e tempos de descompressão com o propósito de analisar o desvio destes tempos devido o aumento ou diminuição do *lap*. Constatamos que a medida que o *lap* aumenta, o tempo de visualização total e o tempo de descompressão aumentam também, isso ocorre devido a quantidade de *voxels* necessários para a reconstrução dos coeficientes pela transformada. Por outro lado o tempo de visualização puro não varia em função do *lap*.

Em relação aos resultados obtidos, identificamos que os parâmetros de compressão, como esperado, influenciam consideravelmente nos tempos medidos pelo algoritmo desenvolvido. Contudo, a variação apresentada mostra que quanto mais comprimido é o dado, maior a relação sinal-ruído e menor o tempo de visualização total. O contrário também é verificado através dos resultados, quanto menor a compressão, melhor será a relação sinal-ruído e menor o tempo de visualização total. O objetivo deste trabalho foi apresentar um algoritmo que visualize os dados comprimidos de maneira satisfatória ao usuário em computadores com

limitações de memória, caso semelhante ao utilizado nos testes. Com essas informações verificamos por exemplo que, o tempo de visualização total gasto para processar o dado volumétrico foi de 72,94s, com uma taxa de 11,99:1, relação sinal-ruído de 29,18dB (decibéis), codificação de *Huffman* e tamanho dos blocos igual a 32 x 32 x 16. Tempo este satisfatório, assim como as variáveis de controle, haja vista que o tempo de visualização deste volume sem o processo de descompressão é de 11,2s, mas utilizando 256Mb de memória Ram, enquanto que o proposto utiliza, para blocos com tamanho 32 x 32 x 16 apenas 16Mb da memória.

Tabela 4.1 - Quantidade de maior e menor taxa de compressão obtida para uma quantidade de memória usada para a visualização.

Quantidade de Memória (Mb)	Taxa de Compressão	Tempo de Visualização Total (TVT)
16	19,07 (maior taxa)	75,122
16	7,11 (menor taxa)	84,558
8	8,15 (maior taxa)	77,295
8	3,91 (menor taxa)	90,448
4	2,65 (maior taxa)	89,211
4	1,4 (menor taxa)	146,984

Tabela 4.1 – Quantidade de maior e menor tempo de visualização obtida para uma quantidade de memória usada para a visualização.

Quantidade de Memória (Mb)	Tempo de Visualização Total (TVT)	Taxa de Compressão
16	113,711 (maior tempo)	7,3
16	67,323 (menor tempo)	18,91
8	117,04 (maior tempo)	3,93
8	71,183 (menor tempo)	8,13
4	146,984 (maior tempo)	1,4
4	89,211 (menor tempo)	2,65

As tabelas 4.1 e 4.2 apresentam as variações da quantidade de memória utilizada pelo algoritmo de descompressão e visualização comparando também as variações de taxas de compressão e dos tempos de visualização total. Percebemos que a maior taxa de compressão, 19,07:1, apresentou um tempo de visualização total de 75,122s, bem próximo do menor tempo medido, 67,323s. Isso ocorre principalmente, devido ao percentual de coeficientes retidos (25%), por outro lado, ocorre um aumento no uso de memória para o processo de descompressão e visualização (16Mb), devido ao tamanho do bloco, que para esses dados medidos foi de 32x32x16 *voxels*. Ainda sobre as taxas de compressão do maior bloco, verificamos que o maior tempo de visualização total foi de 113,711s para uma taxa de compressão de 7,3:1, enquanto que para uma taxa de compressão de 7,11:1 o tempo de visualização total foi de 84,558, essa diferença de tempo em relação a taxas de compressão com valores aproximados ocorre devido a codificação utilizada, pois como já citado anteriormente a Codificação Aritmética é bem mais lenta que a Codificação de *Huffman.* Em relação a menor taxa de compressão medida, foi apresentado o maior tempo de visualização total, 146,984s, para uma taxa de compressão de 1,4:1, fato este ocasionado também pelo percentual de coeficientes retidos (100%) e pelo tamanho do bloco comprimido, contudo a quantidade de memória utilizada é a menor, 4Mb, devido a quantidade de coeficientes que o algoritmo utiliza no processo de descompressão.

5 CONCLUSÕES

Esse trabalho descreve um método para a descompressão e visualização de dados volumétricos comprimidos com transformada do cosseno local. O método de visualização proposto representa um algoritmo promissor para a visualização de dados volumétricos comprimidos, o qual permite a descompressão bloco a bloco do volume, sendo adequado ao desenvolvimento do algoritmo de visualização que permite a visualização de volumes com grandes dimensões em máquinas com reduzida capacidade de memória. Mais especificamente temos como contribuição principal deste trabalho o desenvolvimento do algoritmo de visualização descrito no Capítulo 4 na qual a principal dificuldade foi a reconstrução dos coeficientes dos blocos comprimidos, devido às porções sobrepostas dos seus vizinhos, para então, enviar ao processo de visualização.

Adicionalmente, apresentamos um estudo sobre o comportamento desse algoritmo de descompressão/visualização, a partir do qual podemos concluir que os melhores tempos de visualização são obtidos quando trabalhamos com um pequeno percentual de coeficientes retidos, com blocos de volume de dimensões maiores e altas taxas de compressão. Por outro lado, foi verificado que os blocos com pequenas dimensões precisam de pouca memória para a descompressão e visualização do dado volumétrico, apresentando um aumento no tempo de visualização total em dados com pequenas taxas de compressão. Assim, verificamos que o algoritmo apresenta características vantajosas em relação ao uso por computadores com limitações de memória, já que, os tempos para visualizar os dados volumétricos de grande capacidade e a quantidade de memória necessária

para esse processo são considerados satisfatórios, embora não adequados para visualização interativa em tempo real.

Um outro parâmetro importante que analisamos foi à influência da codificação utilizada no tempo de visualização total. Pela sua própria característica, a codificação aritmética, como previsto, apresenta tempos de visualização total muito maiores que a codificação de *Huffman*, por outro lado as taxas de compressão são maiores, conforme apresentado na Seção 4.4. Constatamos também que as diferenças entre os tempos de visualização total medido entre as codificações aumentam a medida que aumentam o percentual de coeficientes retidos e o tamanho dos blocos.

Observamos por fim, que o algoritmo de descompressão local baseado na LCT é de alta complexidade devido a influência das porções sobrepostas (*lap)* entre blocos vizinhos, que é parte fundamental na reconstrução dos coeficientes da transformada.

Como sugestão para trabalhos futuros propomos o desenvolvimento uma adaptação deste algoritmo para *Web*, onde os blocos comprimidos seriam enviados para o cliente na ordem de visualização adequada, e no computador do cliente o algoritmo de visualização descomprime e visualiza bloco a bloco, minimizando assim o consumo de memória e de largura de banda. Torna-se vantajoso este processo devido a menor quantidade de dados que seriam enviados pela rede que resultaria em um tempo de transmissão menor que enviar o dado no seu tamanho original. Um exemplo prático seria a necessidade de visualização de um dado volumétrico por um médico em setores de hospitais ou clínicas, após realizada a tomografia computadorizada do paciente, esse dado seria armazenado em formato comprimido em um servidor e a medida que o médico necessitasse deste dado para

diagnosticar, o algoritmo de descompressão e visualização para *Web*, iniciaria o processo de descompressão e visualização do dado na ordem adequada, pela rede seria transmitido somente os dados comprimidos que só sofrem a descompressão no computador do médico, melhorando assim o tráfego de informações na rede e a exigência de computadores com grande capacidade de memória.

Outra sugestão é apresentada nos resultados obtidos deste trabalho, onde os tempos de visualização puro (TVP) são maiores em relação ao tempo de visualização sem compressão (VSC), então o trabalho sugerido é de verificar o que influencia no tempo de visualização puro após o processo de descompressão e diminuir este tempo, melhorando assim a performance total do algoritmo de visualização. O que motiva este trabalho é conseguir altas taxas de compressão com baixos tempos de visualização mantendo a qualidade da imagem, apresentando assim resultados bem mais satisfatórios.

REFERÊNCIAS

ACKERMAN, M. A. Visible Woman: High Resolution Companion to the Visible Man. **Interactive Healthcare Newsletter**, v. 12 n. 7/8, p. 5-6. 1996

AHMED, N.; RAO, K. R. Discrete Cosine Transform. **IEEE Transaction on Computers**, p. 90-93, jan. 1974 (C-23)

BROCKLEHURST, A. **Data Compression**. Tech. Rept. British Petroleum Exploration. 1995

CHAN, K. K. et al. Visualization and Volumetric Compression. Image Capture, Formatting and Display, **SPIE**, v. 1444, p. 250-255. 1991.

CHEN, W. H.; PRATT, W. K. Scene Adaptive Coder. **IEEE Transactions on Communications**, p. 225-232, mar. 1984.

CHEN, Y.; PEARLMAN, W. A. Three-dimensional Sub-band Coding of Video Using the Zero-tree Method. **SPIE**, p. 1302-1310, mar. 1996. (of: Proceedings of SPIE-Visual Communications and Image Processing '96)

CHIUEH, T. C. et al. Inte-grated Volume Compression and Visualization, **IEEE Visualization'97**, oct, 1997.

COIFMAN, R. R.; MEYER, Y.; WICKERHAUSER, M. V. Wavelet Analysis and Signal. In: JONES and BARTLETT. **Wavelet and Their Applications**, Boston: edited by B. Ruskai et al. p. 153-178. 1991

ELVINS, T. T. A Survey of Algorithms for Volume Visualization. **Computer Graphics**, v. 26, n. 3, Aug. 1992.

FLTX, User Manual. Fast Light Toolkit. Disponível em: http://www.fltk.org. Último Acesso em: 25/10/2004.

HALEY, M. B.; BLAKE, E. H.Incremental Volume Rendering Using Hierarchical Compression. **Computer Graphics Forum**, v. 15, n. 3, p. 44-55. 1996

HUFFMAN, D. A. A Method for the Construction of Minimum Redundancy Codes. **Proceedings IRE**, v. 40, n. 10, p. 1098-1101. 1952

IHM, I.; PARK, S. Wavelet-Based 3D Compression Scheme for Very Large Volume Data. **Graphics Interface**, p. 107-116. 1998

KIM, T. Y.; SHIN, Y. G. 1999 (October). An Eficient Wavelet-Based Compression Method for Volume Rendering. **Pacific Graphics**, oct. 1999.

KRISHNAN, Karthik et al. Compression/Decompression Strategies for Large Volume Medical Imagery. University Arizona. **SPIE**, v. 5371, p. 152-159, apr. 2004

LACROUTE, P.; LEVOY, M. Fast Volume Rendering using a Shear-Warp Factorization of the Viewing Transformation, **Computer Graphics**, v. 28, n. 3, p. 451-458, 1995.

LI , Wei; KAUFMAN, Arie. **Texture Partitioning and Packing for Accelerating Texture-based Volume Rendering**. NY(USA): Center for Visual Computing (CVC) and Department of Computer Science Stony Brook University, Stony Brook, 2003.

MALVAR, H. S. The LOT: A Link Between Block Transforms Coding and Multirate Filter Banks. Proceedings of the IEEE International Symposium on Circuits and Systems, p. 835-838, june. 1988.

MALVAR, H. S. The LOT: Transform Coding Without Blocking E_ects. **IEEE Transactions on Acoustic**, Speech, and Signal Processing, n. 38, p. 969-978. 1990.

MALVAR, H. S. **Signal Processing with Lapped Transform**. Norwood, MA.: Artech House. 1992.

MALVAR, H. S.; STAELIN, D. H. Lapped Transforms for Eficient Transform/Subband Coding. **IEEE Transactions on Acoustic**, Speech, and Signal Processing, v. 37, n.4, p. 553-559. 1989.

MCCORMICK, B.; DEFANTI, T.; BROWN, M. Visualization in Scientific Computing. **Computer Graphics**, v. 21, n. 6, nov. 1987.

NELSON, Mark. **THe Data Compression Book**. [S.L]: Prentice Hall, 1991.

NING, P.; HESSELINK, L. Fast Volume Rendering of Compressed Data. p. 11-18 of: Visualization 1993.

NATIONAL LIBRARY OF MEDICINE - NLM. **The Visible Human Project**. Disponível em:http:www.nlm.nih.gov/research/visible/visible human.html. Acesso em: 1 jan. 05

PAIVA, A. C. de. Um Método para Compressão de Dados Volumétricos Comprimidos Baseado na Transfomada do Cosseno Local. 2001. 77 f. Tese Doutorado; Departamento de informática da Pontifícia Universidade Católica, Rio de Janeiro, 2001.

PHONG, B. T. Illumination for Computer Generated Pictures. **Communications of the ACM ,** v. 18, n. 6, p. 311-317. 1975.

RISSANEM, J. J.; LANGDON, G. G. Arithmetic Coding. **"IBM"Journal of Research and Development**, p. 149-162, mar. 1979.

RISSANEM, J. J.; LANGDON, G. G. Universal Modeling and Coding. **IEEE Transactions on Information Theory**, v.27, n.1, p. 12-22. 1981.

RODLER, F. R. **Wavelet Based 3D Compression for Very Large Volume Da-**

ta Supporting Fast Random Acces. Tech. rept. BRICS RS-99-34. Aarhus C, Denmark: Department of Computer Science, University of Aarhus. Disponível em:http://www.brics.dk. 1999.

RODLER, F. F. Wavelet Based 3D Compression with Fast Random Access for Very Large Volume Data. Pacific Conference on Computer Graphics and Applications 2002.

SHANNON, C. E. A Mathematical Theory of Communication. **Bell System Technical Journal**, v. 27, n3, p. 379-423.1948.

SCHNEIDER, Jens; WESTERMANN, Rudiger. **Compress Domain Volume Rendering**. Technical University Munich. 2003.

WESTERMANN, R. A Multiresolution Framework for Volume Rendering. p. 51-58 In: SYMPOSIUM ON VOLUME VISUALIZATION. Oct. 1994.

WESTOVER, L. Footprint Evaluation for Volume Rendering. **Computer Graphics** (Proc. SIGGRAPH), v. 24, n. 4, p. 367-376. 1990.

WHILHELMS, J.; GELDER, A. V. Multidimensional Trees for Controlled Volume Rendering and Compression. **ACM Siggraph Symposium on Volume Visualization**. p. 27-34. 1994.

YEO, B. L.; LIU, B. Volume Rendering of DCT-based Compressed 3D Scalar Data. **IEEE Transactions on Visualization and Computer Graphics**, v. 1, n. 1, p. 29-43. 1995.

ZIV, J.; LEMPEL, J. A Universal Algorithm for Sequential Data Compression. **IEEE Transaction on Information Theory**, v. 23, n. 3, p. 337-343.1977.

ANEXO A - Algoritmo de Descompressão

Comentários adicionais:

Para calcular a LCT Inversa de um bloco, são necessários os seguintes passos: primeiro, calculamos a DCT Interslice do bloco, em seguida, é realizado o unfolding com o bloco vizinho, em relação à intersice. Em seguida, é necessário aplicar o mesmo processo em reação às colunas. Porém, antes de fazer o desdobramento das colunas, é necessário verificar se o vizinho em relação às colunas já realizou a DCT da interslice e a DCT das colunas. Para isso, é necessária a definição de flags em cada bloco, os quais indicam se já foram realizadas as operações de DCT do mesmo.
Para realizar o desdobramento das linhas, igualmente é necessário, antes de fazer a dobragem, assegurar que o vizinho já realizou as operações de DCT de interslice , colunas e linhas, bem como as respectivas dobragens com seus vizinhos. Todos estes testes tem que ser realizados *para todo bloco*, pois a mínima variação dessa ordem invalida a transformada, e consequentemente acrescenta ruído inaceitável ao dado original.

Algoritmo de Descompressão

Algoritmo InvLct3D

Entrada: slc, fatia de blocos que está sendo usada
Saída: Uma fatia de blocos reconstruídos

```
{
 i,j,k, NB: inteiro; // k: percorre as fatias; i: as linhas ;j : as colunas
 k <-- slc; //recebe o valor da fatia passada como parametro
 NB <- numero_de_blocos; //número de blocos em todas as direções

para (i = 0; i< NB; passo 1)
 para (j = 0; j< NB; passo 1){
   {CALCULA A LCT INV. INTERSLICE}
   if (B[k][i][j].DCT_IS = false){//verifica se já foi calculada a DCT IS de B[k][i][j]
     DctIS(k,i,j);  //calcula a DCT interslice de B[k][i][j]
     B[k][i][j].DCT_IS <- true; //seta a operação de DCT interslice de B[k][i][j] como já
realizada
    }
    if(k != NB-1){ //verifica se não é o último na direção das fatias
     if(B[k+1][i][j].DCT_IS = false){//verifica se já foi calculada a DCT IS de B[k+1][i][j]
       DctIS(k+1,i,j); //calcula a DCT interslice de B[k+1][i][j]
     B[k+1][i][j].DCT_IS   <- true; //seta a operação de DCT interslice de B[k+1][i][j]
como já realizada
       UnfoldInterslice(B[k][i][j],B[k+1][i][j]); //Unfold entre B[k][i][j] e B [k+1][i][j]
       }
    }
     {CALCULA A LCT INV. DAS COLUNAS}
     if (B[k][i][j].DCT_Col = false){//verifica se já foi calculada a DCT Col. de B[k][i][j]
```

```
        DctColuna(k,i,j); //calcula a DCT coluna de B[k][i][j]
        B[k][i][j].DCT_Col <- true; //seta a operação de DCT col. de B[k][i][j] como já
realizada
      }
  {Agora, é necessário fazer o desdobramento das colunas de B[k][i][j] com
B[k][i][j+1]. Porém, antes disso é necessário preparar B[k][i][j+1], verificando se ele já
passou pelas etapas anteriores. Se não, é necessário realizar a DCT IS, unfold IS
(realizando a DCT IS do vizinho caso seja necessário) com seu vizinho B[k+1][i][j+1],
e finalmente a DCT coluna. }
      if (j != NB-1){
         if (B[k][i][j+1].DCT_IS = false){
             DctIS(k,i,j+1);
             B[k][i][j+1].DCT_IS <- true;
          }
        if (k != NB-1){
         if (B[k+1][i][j+1].DCT_IS = false){
            DctIS(k+1,i,j+1);
            B[k+1][i][j+1].DCT_IS <- true;
            UnfoldInterslice(B[k][i][j+1],B[k+1][i][j+1]);
          }
         }
         if (B[k][i][j+1].DCT_Col = false){
           DctColuna(k,i,j+1);
           B[k][i][j+1].DCT_Col <- true;
            {Podemos fazer o unfold das colunas agora}
           UnfoldCols(B[k][i][j],B[k][i][j+1]);
          }
       }

    {CALCULA A LCT INV. DAS LINHAS}
    if (B[k][i][j].DCT_Lin = false){//verifica se já foi calculada a DCT Lin. de B[k][i][j]
       DctLinha(k,i,j);
       B[k][i][j].DCT_Lin <- true; //seta a operação de DCT linha de B[k][i][j] como já
realizada
      }
{Agora, é necessário fazer o desdobramento das linhas de B[k][i][j] com B[k][i+1][j].
Mais uma vez, é necessário preparar B[k][i+1][j], verificando se ele já passou pelas
etapas anteriores.  Cada vez que formos realiza uma operação de Unfold é
necessário verificar se o bloco já está pronto, ou seja, se já passou pelas etapas
anteriores. }

      if (i != NB-1){
        if (B[k][i+1][j].DCT_IS = false){
          DctIS(k,i+1,j);
          B[k][i+1][j].DCT_IS <- true;
          }
       if (k != NB-1){
         if (B[k+1][i+1][j].DCT_IS = false){
           DctIS(k+1,i+1,j);
           B[k+1][i+1][j].DCT_IS <-  true;
```

```
            UnfoldInterslice(B[k][i+1][j],B[k+1][i+1][j]);
          }
        }

        if (B[k][i+1][j].DCT_Col = false){
           DctColuna(k,i+1,j);
           B[k][i+1][j].DCT_Col  <- true;
     }
    {Preparação do bloco B[k][i+1][j+1] para o unfold das colunas com B[k][i+1][j]}
    if (j != NB-1){
         if (B[k][i+1][j+1].DCT_IS = false){
           DctIS(k,i+1,j+1);
            B[k][i+1][j+1].DCT_IS <-  true;
          }

          if (k != NB-1){
             if (B[k+1][i+1][j+1].DCT_IS = false){
                DctIS(k+1,i+1,j+1);
                B[k+1][i+1][j+1].DCT_IS  <- true;
                UnfoldInterslice(B[k][i+1][j+1],B[k+1][i+1][j+1]);
              }
        }
        if (B[k][i+1][j+1].DCT_Col = false){
           DctColuna(k,i+1,j+1);
          B[k][i+1][j+1].DCT_Col <- true;
        }
        UnfoldCols(B[k][i+1][j], B[k][i+1][j+1]);
    }
{resta a agora o cálculo da DCT das linhas de B[k][i+1][j] para podermos realizar o
unfold com B[k][i][j]. Com isso, B[k][i][j] está completamente reconstruído}

       if (B[k][i+1][j].DCT_Lin = false){
         DctLinha(k,i+1,j);
         B[k][i+1][j].DCT_Lin <- true;
         {Podemos fazer o unfold das linhas agora}
         UnfoldLines(B[k][i][j],B[k][i+1][j]);
        }
   }
 }//fim para
}//Fim Algoritmo
```