MAC0499 - Trabalho de Formatura Supervisionado
Compressão de Áudio Digital

Aluno: Marcio Masaki Tomiyoshi
Supervisor: Marcelo Gomes de Queiroz

Conteúdo

Parte I
Parte Técnica

1 Introdução

A utilização de compressores de arquivos sempre é interessante do ponto de vista da economia de espaço de armazenamento e utilização de banda desde que o processo de compressão e descompressão possam ser realizados rapidamente.

No caso de áudio digital, estamos lidando com arquivos que ocupam bastante espaço em disco e, por conseqüência, exigem uma grande largura de banda. O uso de técnicas genéricas de compressão neste tipo de arquivo pode levar a uma taxa de compressão muito mais baixa do que a possível quando nos aproveitamos da organização dos dados num arquivo de som.

Assim, o objetivo do trabalho foi estudar e implementar diversos métodos que foram desenvolvidos para explorar este maior potencial de compressão nos arquivos de áudio. Porém, antes de aprofundarmos no assunto proposto, abordaremos alguns conceitos de representação de áudio digital para que seja possível um melhor entendimento.

2 Representação de Áudio Digital

Para representarmos uma onda sonora em um computador, precisamos discretizar tanto a escala da amplitude da onda quanto a escala de tempo em que ela se situa já que ambas as escalas são naturalmente contínuas. O processo de discretização da amplitude da onda é chamado de *quantização*, enquanto a discretização do tempo é denominado *amostragem*.

Existem diferentes formas de se digitalizar uma onda sonora, porém, o padrão utilizado atualmente é o *Pulse Code Modulation* ou *PCM*.

2.1 Pulse Code Modulation

O *PCM* é o formato utilizado na maioria das mídias industriais existentes. É usado, por exemplo, no CD, nas fitas DAT, no HiMD e no DVD-Audio.

Essencialmente, o formato se baseia na divisão das escalas da amplitude e de tempo em partes de igual tamanho. Para isso, são definidos dois parâmetros: a *taxa de amostragem* e a *quantidade de bits por amostra*.

A taxa de amostragem indica quantas vezes por segundo o valor da amplitude da onda será armazenada. É medido em *hertz (Hz)*.

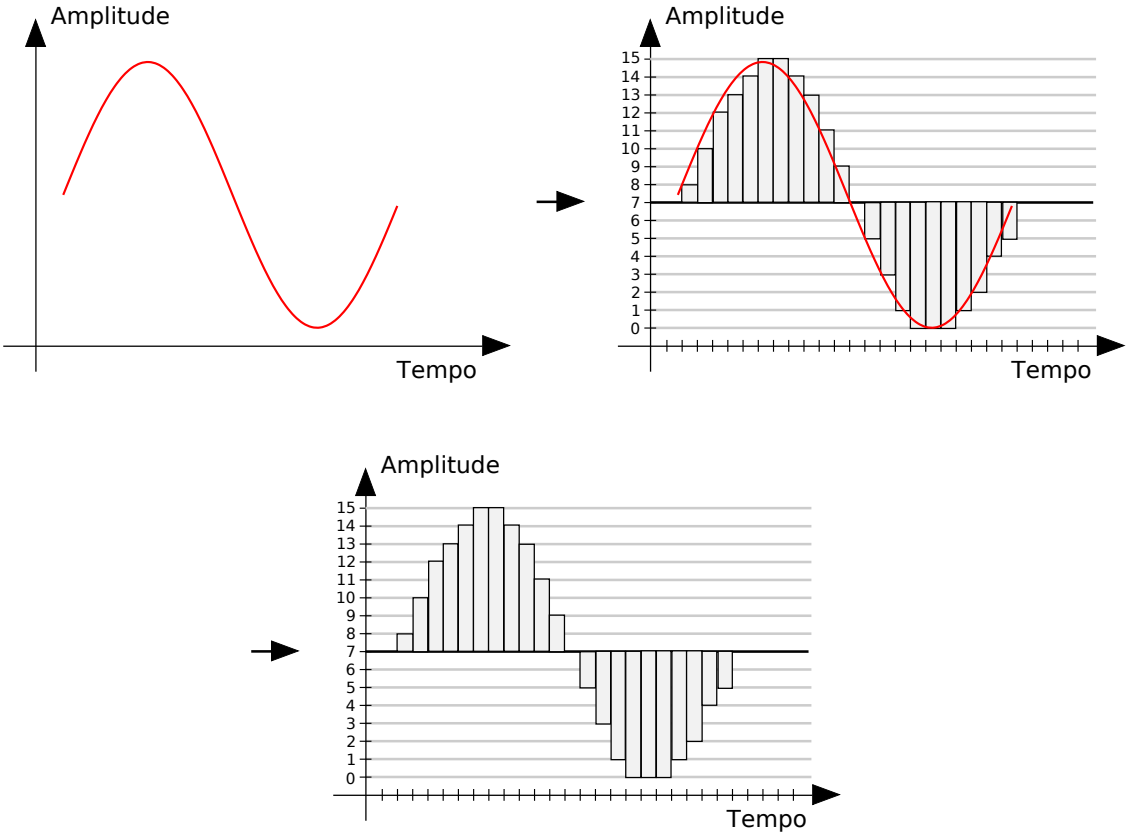

Figura 1: Processo de quantização com 4 bits.

A quantidade de bits por amostra indica o número total de intervalos de mesmo tamanho em que a amplitude da onda poderá ser armazenada.

Além disso, o formato possibilita armazenarmos quantos canais de som desejarmos.

O CD, padrão de qualidade atual, usa 16 bits para representar cada amostra, resultando em 2^{16} ou 65,536 intervalos. Sua taxa de amostragem é de 44100 Hz e 2 canais são armazenados. Com isso, o formato PCM gasta $16 \times 2 \times 44100$ Hz $= 1411$ kilobits por segundo ($kbps$), uma taxa de bits bastante elevada, existindo poucas aplicações que exijam uma taxa de bits ainda maior (gravações de vídeo digital é um dos exemplos).

A figura 1 ilustra a quantização de uma onda com 4 bits por amostra.

2.2 Aliasing

A figura 2 ilustra um possível problema ao realizarmos a amostragem. Através de uma mesma seqüência de amostras, conseguimos reconstruir duas ondas de freqüências distintas, o que obviamente não é desejado.

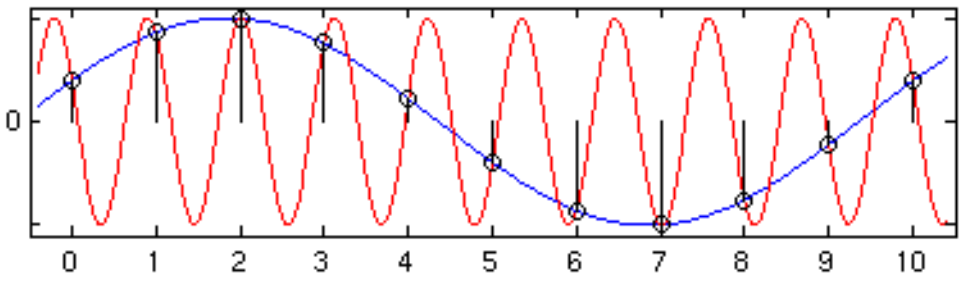

Figura 2: Fenômeno de aliasing.

Para não encontrarmos esse tipo de problema, fazemos uma filtragem das freqüências acima de $\frac{R}{2}$ Hz (eliminando-as), para uma taxa de amostragem de R Hz, pois o *teorema de Nyquist* (ou *teorema da amostragem*) nos garante que, para uma taxa de amostragem de R Hz, as freqüências de até $\frac{R}{2}$ Hz serão representadas corretamente [8].

Isto também explica a taxa de amostragem usada no CD (44100 Hz), possibilitando-nos representar corretamente freqüências de até 22050 Hz, valor acima do limiar superior da audição humana[1].

2.3 RIFF WAVE

O padrão *RIFF WAVE*, ou simplesmente *WAVE*, é o formato comumente utilizado para o armazenamento de arquivos em *PCM* em um computador. Neste caso, o formato consiste basicamente de um cabeçalho seguido das amostras. Além do *PCM*, outros formatos também são aceitos pelo padrão *RIFF WAVE*, fato que foi aproveitado durante o desenvolvimento de alguns dos compressores *lossy*.

O cabeçalho *WAVE* possui a forma descrita na tabela 1. Todos os números no cabeçalho *WAVE* estão no formato *little endian*[2]. Após o cabeçalho, temos os valores das amostras, que também estão em *little endian*.

[1] O ouvido humano consegue perceber freqüências entre 20 e 20000 Hz.

[2] No formato *little endian*, o byte mais significativo fica na primeira posição da memória. Exemplo: quatro bytes seriam posicionados como $Byte_3$ $Byte_2$ $Byte_1$ $Byte_0$.

Posição (em bytes)	**Campo**	**Descrição**
0	ChunkID	Contém a string "RIFF" em ASCII
4	ChunkSize	$= 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size)$
8	Format	Contém a string "WAVE"
12	SubChunk1ID	Contém a string "fmt " (o último caractere é um espaço)
16	SubChunk1Size	$= 16$ no caso do *PCM*
20	AudioFormat	$= 1$ no caso do *PCM*
22	NumChannels	Indica o número de canais (normalmente 2)
24	SampleRate	Indica a taxa de amostragem (44100 Hz para um CD)
28	ByteRate	$= \frac{SampleRate \times NumChannels \times BitsPerSample}{8}$
32	BlockAlign	$= \frac{NumChannels \times BitsPerSample}{8}$
34	BitsPerSample	Quantidade de bits por amostra (16 para um CD)
36	SubChunk2ID	Contém a string "data"
40	SubChunk2Size	$= \frac{NumSamples \times NumChannels \times BitsPerSample}{8}$
44	data	Valores das amostras

Tabela 1: Campos do cabeçalho *WAVE* como descrito em [2].

Para arquivos que usam 8 bits por amostra, os valores de $[-128, 127]$ são representados no intervalo $[0, 255]$. Já para arquivos de 16 bits, os valores são armazenados no formato conhecido como *complemento de 2*.

Além disto, as amostras dos diferentes canais são armazenadas da seguinte forma:

$$amostra_0 - canal_0,\ amostra_0 - canal_1, \ldots,\ amostra_0 - canal_m,$$

$$amostra_1 - canal_0,\ amostra_1 - canal_1, \ldots,\ amostra_1 - canal_m,$$

$$\vdots$$

$$amostra_n - canal_0,\ amostra_n - canal_1, \ldots,\ amostra_n - canal_m.$$

(n = número de amostras e $m = NumChannels$)

Em [1, 2, 3, 6, 7] mais informações sobre o formato podem ser encontradas. Nas seções 3.2.1 e 3.2.2, outros detalhes úteis para a implementação serão explicados.

3 Compressão de Áudio Digital

A compressão de arquivos de áudio se divide em duas categorias:

- Sem perdas (*lossless*)

  Ao efetuarmos a compressão de um arquivo, o arquivo obtido ao descomprimirmos será exatamente igual ao original, bit por bit.

- Com perdas (*lossy*)

  Neste caso, ao realizarmos a descompressão, teremos como resultado final um arquivo diferente daquele que foi comprimido.

3.1 Compressão sem perdas: *AudioPaK*

O padrão de compressão *lossless* estudado e implementado foi baseado no *AudioPaK* [5]. O formato foi criado com o objetivo de ser pouco exigente computacionalmente, então diversos compromissos foram tomados, simplificando as operações executadas em detrimento de uma maior taxa de compressão.

A idéia por trás do método é que, através de uma estimativa do valor da próxima amostra, é possível armazenarmos apenas a diferença entre ela e o valor estimado. Assim, quanto melhor o estimador utilizado, teremos que o valor estimado estará mais próximo do valor amostrado e, por conseqüência, a diferença será um número menor que o valor da amostra, possibilitando que economizemos bits.

3.1.1 Divisão em blocos

A primeira etapa do processo de codificação envolve a divisão em blocos ou *frames*, que agrupam uma certa quantidade de amostras.

Por se tratar de um método em que armazenamos apenas a diferença entre o valor amostrado e o valor estimado, caso haja corrupção dos dados no início do arquivo, por exemplo, o erro se propagaria até o final do processo e todos os valores decodificados a partir dali seriam incorretos. A vantagem de se dividir o arquivo original em diversos blocos é que adicionamos mais robustez ao formato, pois agora, mesmo que tenhamos corrupção de alguma amostra, a divisão feita garante que, no próximo bloco, os valores decodificados estarão corretos.

Em contrapartida, ao realizarmos este processo, precisamos adicionar um certo *overhead* em cada *frame*, gastando alguns bits a mais para compensar o aumento de robustez.

Com a divisão em blocos gostaríamos de saber se existe um tamanho de bloco ideal. Em [5], testes com diversos tamanhos de blocos foram feitos e os arquivos comprimidos tiveram pouca variação de tamanho. Assim, o artigo sugere que, por melhor se adequar a outros padrões já estabelecidos, blocos de tamanho 1,152 sejam utilizados para compactarmos arquivos de som com qualidade de CD.

3.1.2 Estimadores utilizados

Conforme dito anteriormente, um bom estimador tem grande importância na compressão que pode ser atingida pelo formato. O *AudioPaK*, por ter como foco o baixa complexidade de computação, propõe estimadores que exijam uma quantidade baixa de cálculos e consigam uma taxa de compressão boa.

Os estimadores utilizados são os seguintes:

$$\begin{cases} \hat{x}_0[n] = 0 \\ \hat{x}_1[n] = x[n-1] \\ \hat{x}_2[n] = 2x[n-1] - x[n-2] \\ \hat{x}_3[n] = 3x[n-1] - 3x[n-2] + x[n-3] \end{cases}$$

Chamaremos de residual a diferença entre o valor amostrado e o valor estimado, ou seja, $e[n] = x[n] - \hat{x}[n]$, onde e é o residual, x é o valor amostrado e $\hat{x}$ o estimado. Fica claro que o residual é exatamente o que será armazenado pelo formato. Com isso, temos uma propriedade interessante desses estimadores que facilita o modo de calculá-los:

- $e_0[n] = x[n] - \hat{x}[n] = x[n]$

- $e_1[n] = x[n] - x[n-1] = e_0[n] - e_0[n-1]$

- $$\begin{aligned} e_2[n] &= x[n] - 2x[n-1] + x[n-2] \\ &= x[n] - x[n-1] - x[n-1] + x[n-2] \\ &= e_0[n] - e_0[n-1] - e_0[n-1] + e_0[n-2] \\ &= (e_0[n] - e_0[n-1]) - (e_0[n-1] - e_0[n-2]) \\ &= e_1[n] - e_1[n-1] \end{aligned}$$

- $e_3[n] = x[n] - 3x[n-1] + 3x[n-2] - x[n-3]$
  $= (x[n] - 2x[n-1] + x[n-2]) - (x[n-1] - 2x[n-2] + x[n-3])$
  $= e_2[n] - e_2[n-1]$

Ou seja, podemos calcular o residual recursivamente da seguinte forma:

$$\begin{cases} e_0[n] = x[n] \\ e_1[n] = e_0[n] - e_0[n-1] \\ e_2[n] = e_1[n] - e_1[n-1] \\ e_3[n] = e_2[n] - e_2[n-1] \end{cases} \quad (1)$$

Note que com estes residuais, não é necessário realizarmos nenhuma multiplicação para calculá-los. A figura 3 ilustra a interpretação gráfica dos estimadores:

- $\hat{x}_0$ é sempre 0, então o residual é exatamente o valor real da amostra, o que é útil para os casos em que a correlação entre as amostras é baixa.
- $\hat{x}_1$ utiliza o valor da amostra anterior para estimar a próxima e tem um bom desempenho quando as amostras estão todas próximas umas das outras.
- $\hat{x}_2$ toma as duas amostras anteriores e estipula que a próxima amostra estará na reta gerada por elas.
- $\hat{x}_3$ toma as três amostras anteriores e estipula que a próxima amostra estará na parábola gerada por elas.

De posse deste conjunto de estimadores, podemos prosseguir na descrição do algoritmo de compressão.

Primeiro, armazenamos a amostra de referência do bloco. Em seguida, para cada amostra pertencente ao bloco faremos o cálculo de todos os quatro residuais. Para cada estimador, acumularemos o valor em módulo dos residuais calculados. A partir desta informação, escolhemos o estimador com o menor valor acumulado, que será o estimador utilizado no bloco.

A motivação por trás desta escolha é que se, no total, a soma dos residuais em módulo do estimador foi a menor de todas, o valor que será armazenado para cada amostra pelo formato (o próprio residual) é, em média, menor que os residuais dos outros estimadores. Assim, será possível utilizar uma quantidade de bits menor para representarmos a amostra.

Para conseguirmos economizar bits na representação, precisamos codificar os valores de uma forma diferente. O *AudioPaK* utiliza, para isso, a codificação de Golomb que será discutida a seguir.

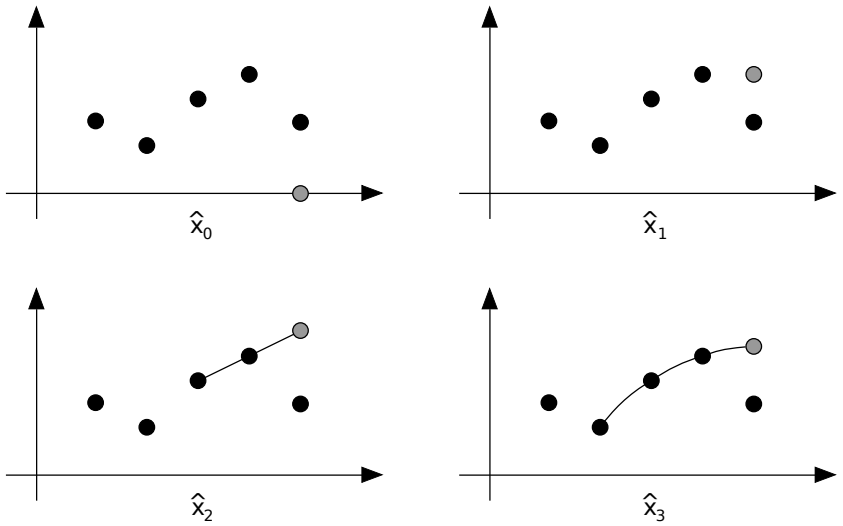

Figura 3: Interpretação gráfica dos estimadores. As amostras são os círculos pretos e o valor estimado o círculo cinza.

3.1.3 Codificação de Golomb e de Rice

O *AudioPaK* sugere a utilização da codificação de Golomb juntamente com um remapeamento dos números inteiros para o armazenamento das amostras residuais. A codificação de Golomb, através de um parâmetro m, divide um inteiro *positivo* em duas partes: uma parte é a representação binária de $(n \bmod m)$ e a outra é uma representação unária de $\left(\left\lfloor \frac{n}{m} \right\rfloor\right)$.

Por representar apenas números positivos e os valores dos residuais poderem ser negativos, é preciso aplicar o seguinte mapeamento:

$$r[n] = \begin{cases} 2e[n] & \text{se } e[n] \geq 0 \\ 2e[n] - 1 & \text{se } e[n] < 0 \end{cases}$$

Um caso particular da codificação de Golomb é quando o parâmetro m é uma potência de 2. O método para $m = 2^k$, é conhecido como codificação de Rice. No código implementado, foi utilizado este tipo de código.

A codificação de Rice também difere da de Golomb na representação dos inteiros negativos. O código se divide em três partes: um bit de sinal, os k bits menos significativos de n e uma representação unária dos bits mais significativos de n.

A representação unária de um inteiro n é apenas uma seqüência de n zeros, seguido de um 1 que indica o final do código.

Exemplo Vamos codificar $n = -19$ em Rice para $k = 3$. Temos que $((19)_{10} = (10\underbrace{011}_{k})_2)$. Como n é negativo, ajustamos o bit de sinal para 1. Em seguida, tomando os k bits menos significativos, obtemos a segunda parte do código: 011. Por fim, codificamos os bits mais significativos de n $((10)_2 = (2)_{10})$ unariamente: $\underbrace{00}_{2}1$.

Assim, o código de Rice para $n = -19$ com $k = 3$ é $(1011001)_2$.

Entendido o processo de codificação, falta apenas calcularmos o valor de k que será utilizado em um bloco. Idealmente, poderíamos testar, para cada *frame* de um arquivo com b bits por amostra, qual o valor de k entre 0 e $b - 1$ que minimizaria a quantidade total de bits do bloco. Porém, isso seria consideravelmente custoso computacionalmente, o que conflitaria com a proposta do *AudioPaK*.

Então, o valor de k é estimado a partir da seguinte fórmula:

$$k = \lceil \log_2(E(|e_i[n]|)) \rceil, \text{ onde } E \text{ é a função esperança.} \tag{2}$$

Esta estimativa é baseada no valor médio dos residuais de um bloco, apenas tomando o logaritmo na base 2 sobre este valor médio para indicar qual a seria a quantidade de bits utilizada, em média.

Ainda em [5], é dado uma outra fórmula para estimar k, muito semelhante à primeira:

$$k = \lceil \log_2(\ln(2)E(|e_i[n]|)) \rceil \tag{3}$$

Como $\ln(2) \approx 0,69$, podemos ver que esta estimativa basicamente diminui o valor médio através da constante $\ln(2)$. Em testes práticos realizados durante o desenvolvimento, foi observado que a estimativa (3) produzia arquivos com tamanho um pouco menor que a estimativa (2).

É fácil observar que, dada a natureza da codificação de Golomb e Rice, o *AudioPaK* tem uma taxa variável de bits por amostra.

3.1.4 Detalhes de Implementação

Nesta seção, uma descrição mais completa do formato implementado será realizada.

Cabeçalhos Para que o arquivo possa ser corretamente decodificado, o formato do cabeçalho do arquivo deve seguir a tabela 2.

Descrição	Tamanho
String “COMP” em ASCII	4 bytes
Tamanho do bloco	15 bits
Quantidade de canais	4 bits
Bits por amostra	5 bits
Cabeçalho WAVE	44 bytes

Tabela 2: Cabeçalho de um arquivo comprimido em *AudioPaK*.

A string “COMP” é utilizada para identificar o arquivo e dificultar tentativas de descompactação de arquivos que não tenham sido codificados corretamente.

O tamanho do bloco é fixo para todos os blocos, com exceção do último, que pode armazenar menos amostras que o usual. Com 15 bits em sua representação este tamanho pode variar entre 0 e 32,768 amostras por bloco.

A quantidade de canais e bits por amostra poderiam ser inferidos da cópia do cabeçalho *WAVE*, porém isso exigiria que fosse realizado uma análise do cabeçalho *WAVE*, então, armazenamos ambas por se tratar de uma informação pequena.

A cópia do cabeçalho *WAVE* foi feita para garantir que o arquivo decodificado seja idêntico ao original. Se analisarmos a estrutura do cabeçalho *WAVE* (tabela 1) notaremos diversas informações redundantes que poderiam ser geradas no momento da descompactação. Dependendo da consistência do cabeçalho original, isto poderia resultar em um cabeçalho ligeiramente diferente. Por estar interessado em ter o arquivo original e o decodificado exatamente iguais, bit a bit, este procedimento se mostrou mais adequado.

Além do cabeçalho do arquivo, também temos um cabeçalho para cada bloco, indicado na tabela 3.

Descrição	Tamanho
Indica se é o último bloco	1 bit
Índice do estimador utilizado no bloco	2 bits
Parâmetro k do estimador	5 bits
Caso seja o último bloco, indica o tamanho dele	15 bits

Tabela 3: Cabeçalho de um bloco no *AudioPaK*.

Codificação O processo de compactação do arquivo pode ser descrito brevemente da seguinte forma:

```
escreva o cabeçalho do AudioPaK
para cada amostra faça
    se for a primeira amostra do bloco
        teste se é o último bloco do arquivo
        para cada um dos canais
            leia a amostra (servirá de referência)
            armazene a amostra para cada um dos estimadores
    senão
        para cada um dos canais
            leia uma amostra
            armazene-a para o estimador 0
            para os outros estimadores
                calcule o residual
        se a amostra for a última do bloco
            escolha o estimador com menor residual acumulado
            estime o parâmetro k para o código de Rice
            escreva o cabeçalho do bloco
            escreva a amostra de referência
            escreva os residuais com o código de Rice
```

Exemplo Uma simulação do cálculo dos estimadores é exibido na tabela 4.

n	0	1	2	3	4
e_0	10	-22	-13	23	
e_1	10	-32	9	36	
e_2	10	-42	41		
e_3	10	-52	83		

$\rightarrow$

n	0	1	2	3	4
e_0	10	-22	-13	23	
e_1	10	-32	9	36	
e_2	10	-42	41	27	
e_3	10	-52	83		

Tabela 4: Uma iteração do cálculo de residuais.

n	0	1	2	3	4	$\sum \lvert e_i[n] \rvert$
e_0	10	-22	-13	23	32	100
e_1	10	-32	9	36	9	96
e_2	10	-42	41	27	-27	147
e_3	10	-52	83	-14	-54	213

Tabela 5: Todos os residuais calculados e estimador do bloco escolhido.

Decodificação Uma breve descrição do algoritmo de descompactação:

```
 1    leia o cabeçalho do AudioPaK
 2    escreva o cabeçalho WAVE
 3    leia o cabeçalho do bloco
 4    enquanto o bloco não for o último
 5        para cada um dos canais
 6            leia a amostra de referência
 7            armazene a amostra para cada um dos estimadores
 8        para cada residual no bloco
 9            para cada um dos canais
10                leia o residual e armazene no estimador adequado
11            para os estimadores menores faça
12                para cada canal
13                    calcule o valor amostrado
14       para cada amostra no bloco
15           para cada canal
16                escreva a amostra
17       leia o cabeçalho do bloco
18   repita o processo para o último bloco
```

Na linha 13, sabendo os valores dos estimadores para $n-1$ (já calculados na iteração anterior) e o valor atual do residual, podemos reconstruir as amostras reorganizando as equações (1) da seguinte forma:

$$\begin{cases} x[n] = e_0[n] \\ e_0[n] = e_1[n] + e_0[n-1] \\ e_1[n] = e_2[n] + e_1[n-1] \\ e_2[n] = e_3[n] + e_2[n-1] \end{cases}$$

Exemplo Uma simulação da reconstrução das amostras é exibida na tabela 6.

n	0	1	2	3	4
e_0	10	-22	-13		
e_1	10	-32	9		
e_2	10	-42	41	27	
e_3	10	-52	83	-14	

$\rightarrow$

n	0	1	2	3	4
e_0	10	-22	-13		
e_1	10	-32	9	36	
e_2	10	-42	41	27	
e_3	10	-52	83	-14	

Tabela 6: Uma iteração da reconstrução de amostras.

3.1.5 Desenvolvimento Futuro

Diversas modificações podem ser aplicadas ao código produzido, desde simples refatorações, até a incorporação de outras técnicas para melhorar a taxa de compressão e/ou a velocidade da compactação e descompactação de arquivos.

Por exemplo, existem técnicas que se aproveitam da correlação existente entre os diferentes canais em um mesmo arquivo, além da correlação entre as amostras de um mesmo canal que é normalmente aplicada. Através de testes, [5] mostra que há pouca melhora quando esta técnica é aplicada, sugerindo que a compressão seja feita sem considerar a correlação entre os canais de áudio.

Um pequeno aumento na taxa de compressão poderia ser observado ao adicionarmos um bit no cabeçalho do frame que indicasse que todas as amostras do bloco tem valor 0. Existem muitos casos em que o início e/ou final de uma música são compostos destes valores e com isso conseguiríamos economizar mais alguns bits no arquivo final.

Além disso, da forma como está implementado atualmente, a função que codifica o inteiro para o formato proposto por Rice realiza inúmeras chamadas a uma função que escreve apenas um bit no arquivo por vez. Uma sugestão para melhorar o desempenho seria modificá-la para escrever, por exemplo, os m bits menos significativos de um inteiro dado.

Outras melhorias poderiam ser feitas mediante um estudo de como é realizado o acesso às matrizes que armazenam os residuais de cada estimador e otimizando este processo para se aproveitar melhor do *cache* do processador.

3.2 Compressão com perdas

Nesta seção, serão abordadas as técnicas utilizadas em três padrões de compressão com perdas: *μ-law*, *IMA ADPCM* e *MPEG-1*.

O nível de complexidade varia desde processos muito simples, como o *μ-law*, que se baseia em uma forma alternativa de se graduar a escala de amplitude da onda sonora, até processos mais sofisticados, como o *MPEG-1*, que se aproveita de certas características auditivas do ser humano para permitir uma maior compactação dos arquivos sonoros, sem que haja uma queda muito grande na qualidade do som percebido.

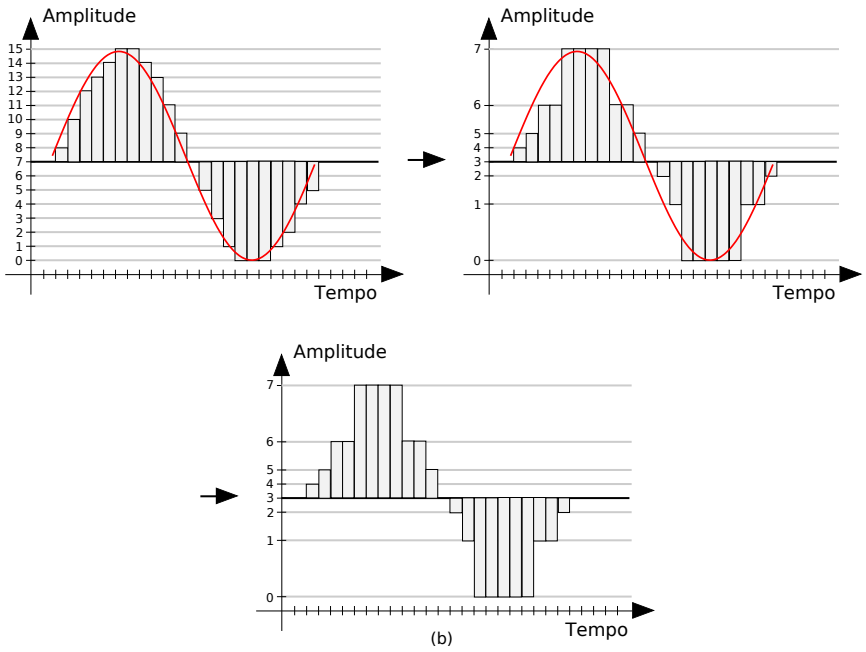

Figura 4: Comparação do *PCM* com o *μ-law*.

3.2.1 *μ*-law

O formato *μ-law* baseia-se em um jeito diferente de se graduar a escala da amplitude da onda sonora. Ao invés de dividí-la em intervalos igualmente espaçados, como no *PCM*, o *μ-law* divide a escala em espaços logarítmicos.

Através desta escala logarítmica, o *μ-law* passa a representar mais fielmente ondas de baixa amplitude enquanto as ondas de amplitudes maiores acabam tendo erros maiores de quantização. Como os sinais de grandes amplitudes tendem a mascarar o ruído introduzido, o resultado final é aceitável [8].

Além disso, é possível, utilizando-se menos bits, cobrir uma faixa dinâmica[3] em que seriam necessários uma quantidade maior de bits caso fosse feita uma quantização linear. No caso do *μ-law*, com 8 bits conseguimos representar ondas em que seriam necessários 14 bits numa escala linear [12], para obter a mesma proporção entre os sons mais fortes e os mais fracos.

A figura 4 ilustra a diferença entre a quantização linear utilizada pelo *PCM* e a logarítmica usada pelo *μ-law*.

Para converter um arquivo *PCM* para *μ-law*, para cada amostra, nor-

[3] *Dynamic range*, em inglês. Trata-se da diferença entre a menor e a maior amplitude do sinal que pode ser representado.

malizamos ela no intervalo $-1 \leq x \leq 1$ e aplicamos a seguinte fórmula, com $\mu = 255$:

$$y = \begin{cases} 255 - \dfrac{127}{\ln(1+\mu)} \times \ln(1+\mu|x|) & \text{para } x \geq 0 \\ 127 - \dfrac{127}{\ln(1+\mu)} \times \ln(1+\mu|x|) & \text{para } x < 0 \end{cases}$$

Após a conversão, podemos armazenar cada amostra com 8 bits, assim, um CD ficaria com metade do seu tamanho original, após comprimido para *μ-law*.

Detalhes de Implementação No caso do *μ-law*, apenas o codificador teve de ser implementado, pois foi possível aproveitar-se do padrão *WAVE*, que já provê suporte ao formato, permitindo a reprodução do arquivo compactado por tocadores de áudio existentes.

Isso trouxe vantagens: por se tratar de um formato *lossy*, não seria possível realizar facilmente testes para termos certeza de que o arquivo produzido realmente se adequava ao *μ-law*, pois seria possível codificar e decodificar um arquivo de alguma forma consistente mas errônea, em que, subjetivamente, os arquivos compactados soariam de forma semelhante aos originais.

Cabeçalho Assim, foi preciso adequar o cabeçalho *WAVE* (tabela 1). Os seguintes campos foram modificados:

- AudioFormat = 7 (*μ-law*)
- BitsPerSample = 8
- ChunkSize, SubChunk2Size, BlockAlign e ByteRate foram recalculados de acordo com os novos parâmetros.

As amostras são armazenadas da mesma forma que no *PCM*.

Codificação Uma descrição breve do código implementado:

```
1    escreva o cabeçalho Wave
2    para cada amostra faça
3        leia a amostra
4        normalize e aplique a transformação
5        escreva a amostra
```

3.2.2 IMA ADPCM

O *IMA ADPCM* utiliza-se de idéias semelhantes ao compressor *lossless* estudado na seção 3.1, armazenando as diferenças entre as amostras ao invés dos valores reais delas. Porém, ao invés de armazenar a diferença precisamente, como fazia o *AudioPaK*, o *IMA ADPCM* permite que um valor próximo da diferença seja armazenado, causando as perdas no processo de codificação.

Além disso, o *IMA ADPCM* vai se adaptando ao sinal de entrada (daí o nome, *Adaptive Differential Pulse Code Modulation*), modelando seus parâmetros de acordo com as características da onda encontradas.

Assim como no *AudioPaK*, novamente é preciso um estimador para a próxima amostra, a fim de que a diferença entre o valor estimado e o valor amostrado seja armazenado. No *IMA ADPCM*, o valor estimado é simplesmente o valor da amostra imediatamente anterior decodificado.

Também é importante notar que a saída codificada possui 4 bits por amostra, compactando um CD para 25% do tamanho original, e que o código não é uma representação da diferença, mas sim uma informação que permite reconstruir a amostra codificada (veremos isso com mais detalhes adiante).

Uma limitação do formato é que, por seguir valores pré-calculados, os algoritmos de codificação e decodificação funcionam corretamente apenas para arquivos originais com 16 bits por amostra.

Codificação Para inicializar o algoritmo, tomamos 0 como índice do passo e armazenamos nossa amostra de referência. O passo inicial não precisa ser necessariamente 0, porém este é o valor normalmente utilizado. A partir daí, a cada amostra lida efetuamos as seguintes instruções:

1. Calculamos a diferença entre a amostra atual e a anterior;
2. Quantizamos o valor da diferença;
3. Desquantizamos o valor recém-quantizado (será explicado a seguir e utilizado no item 1 da próxima iteração);
4. Ajustamos o índice do passo;
5. Escrevemos a amostra;

Quantização Agora, vamos detalhar o processo de quantização, que resultará no código de 4 bits que irá para o arquivo de saída e servirá para

ajustarmos o tamanho do passo na iteração seguinte. Seja x o valor da diferença:

```
se x < 0
    bit3 = 1
    x = -x
senão
    bit3 = 0
se x >= tamanho do passo
    bit2 = 1
    x = x - tamanho do passo
senão
    bit2 = 0
se x >= tamanho do passo/2
    bit1 = 1
    x = x - tamanho do passo/2
senão
    bit1 = 0
se x >= tamanho do passo/4
    bit0 = 1
senão
    bit0 = 0
```

Desquantização O processo de desquantização é o inverso do descrito acima. Através da amostra anterior, do código quantizado e do tamanho do passo construíremos uma aproximação da amostra:

```
se bit2
    diferença = diferença + tamanho do passo
se bit1
    diferença = diferença + tamanho do passo/2
se bit0
    diferença = diferença + tamanho do passo/4
diferença = diferença + tamanho do passo/8
se bit3
    diferença = -diferença
amostra aproximada = amostra anterior + diferença
limitamos o valor da amostra aproximada entre -32768 e 32767
```

Com o código de 4 bits, tomamos os 3 menos significativos, consultamos a tabela 7 e adicionamos o valor ao índice do passo, que será utilizado na

iteração seguinte, realizando o processo adaptativo do algoritmo. O valor do índice do passo é limitado entre 0 e 88 em toda iteração. A tabela 8 contém os tamanhos dos passos e seus respectivos índices.

Bits	**Ajuste**
000	-1
001	-1
010	-1
011	-1
100	2
101	4
110	6
111	8

Tabela 7: Tabela de ajuste de índice.

Fica claro que o processo de desquantização é parte fundamental da decodificação, o que é bastante interessante, pois, ao desenvolvermos o codificador, acabamos por praticamente desenvolver o decodificador simultaneamente.

Como dito anteriormente, os 4 bits emitidos no arquivo de saída não armazenam o valor da diferença. Ao examinar a forma como o quantizador funciona, observamos que a diferença será representada por frações do tamanho do passo que possuem os seguintes valores: $0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, \frac{5}{4}, \frac{3}{2}, \frac{7}{4}$. Então, para cada diferença, teremos 8 valores relativos ao tamanho do passo para escolher aquele que mais se aproxima da diferença real.

O processo adaptativo do codificador então analisa qual desses 8 valores melhor representou a diferença e efetua a modificação necessária ao passo. Se o código de 4 bits indicou que o tamanho do passo atual foi grande demais para representar a diferença (frações menores que 1), na iteração seguinte o tamanho do passo será reduzido. Caso contrário, se o tamanho do passo atual não foi suficiente para representar a diferença (frações maiores ou iguais a 1), o algoritmo passa a aumentar o tamanho do passo para poder representar melhor a diferença.

Índice	**Tamanho**	**Índice**	**Tamanho**	**Índice**	**Tamanho**
0	7	30	130	60	2,272
1	8	31	143	61	2,499
2	9	32	157	62	2,749
3	10	33	173	63	3,024
4	11	34	190	64	3,327
5	12	35	209	65	3,660
6	13	36	230	66	4,026
7	14	37	253	67	4,428
8	16	38	279	68	4,871
9	17	39	307	69	5,358
10	19	40	337	70	5,894
11	21	41	371	71	6,484
12	23	42	408	72	7,132
13	25	43	449	73	7,845
14	28	44	494	74	8,630
15	31	45	544	75	9,493
16	34	46	598	76	10,442
17	37	47	658	77	11,487
18	41	48	724	78	12,635
19	45	49	796	79	13,899
20	50	50	876	80	15,289
21	55	51	963	81	16,818
22	60	52	1,060	82	18,500
23	66	53	1,166	83	20,350
24	73	54	1,282	84	22,358
25	80	55	1,411	85	24,623
26	88	56	1,552	86	27,086
27	97	57	1,707	87	29,794
28	107	58	1,878	88	32,767
29	118	59	2,066		

Tabela 8: Tabela de ajuste do passo.

Detalhes de Implementação O formato *IMA ADPCM* também é suportado pelos arquivos *WAVE*, o que motivou a criação do codificador apenas. Porém, desta vez, diversas modificações na forma como os arquivos eram gravados em disco tiveram de ser feitas para se adequar ao padrão especificado em [7].

Cabeçalho O cabeçalho *WAVE* foi modificado e recebeu alguns novos parâmetros. As alterações feitas foram as seguintes:

- AudioFormat = 17 (*IMA ADPCM*)
- BitsPerSample = 4
- BlockAlign = 2048
- SubChunk1Size = SubChunk1Size + 4
- ByteRate, SubChunk2Size e ChunkSize foram atualizados para manter a consistência

Além disso, novos campos tiveram de ser adicionados antes do *SubChunk2* (byte 36 em diante). O novo posicionamento está na tabela 9.

Posição (em bytes)	**Campo**	**Descrição**
36	CbSize	= 2
38	SamplesPerBlock	$= \frac{(BlockAlign - (4 \times NumChannels)) \times 8}{BitsPerSample \times NumChannels} + 1$
40	Fact	Contém a string “fact”
44	FactSize	= 4
48	TotalSample	Número de samples no arquivo
52	SubChunk2ID	Contém a string “data”
56	SubChunk2Size	Tamanho do campo data
60	data	Valores das amostras

Tabela 9: Campos do cabeçalho *IMA ADPCM* a partir do byte 36.

Posição	Descrição
Byte 0	Byte menos significativo da amostra de referência
Byte 1	Byte mais significativo da amostra de referência
Byte 2	Índice do passo
Byte 3	Reservado

Tabela 10: Cabeçalho de um bloco *WAVE IMA ADPCM*.

Blocos Assim como o *AudioPaK*, no formato *WAVE*, o *IMA ADPCM* é dividido em blocos. Cada bloco possui um cabeçalho da forma mostrada na tabela 10.

Para cada canal do arquivo de entrada um cabeçalho de bloco é gerado e armazenado seqüencialmente no início de cada bloco.

Além disso, os valores quantizados são organizados de uma forma ligeiramente diferente da vista na seção 2.3. No *IMA ADPCM*, o bloco de dados tem o seguinte formato:

$$palavra_0 - canal_0,\ palavra_0 - canal_1, \ldots,\ palavra_0 - canal_m,$$

$$palavra_1 - canal_0,\ palavra_1 - canal_1, \ldots,\ palavra_1 - canal_m,$$

$$\vdots$$

$$palavra_n - canal_0,\ palavra_n - canal_1, \ldots,\ palavra_n - canal_m.$$

$$(n = BlockAlign \text{ e } m = NumChannels)$$

Onde cada palavra possui 4 bytes, representando 8 valores quantizados, e cada byte tem a estrutura mostrada na tabela 11.

Byte	Posição	Descrição
Byte i	Bits mais significativos	Amostra $2i + c, c = 8p + 1$
$(0 \leq i \leq 3)$	Bits menos significativos	Amostra $2i + 1 + c$

Tabela 11: Formato de um byte da palavra p.

O algoritmo implementado segue a descrição dada anteriormente.

3.2.3 MPEG-1 Audio

O *MPEG-1* revolucionou o segmento de compressão de áudio digital *lossy*, além de ter afetado a indústria fonográfica como um todo com o *MP3* (*MPEG-1 layer III*). Grande parte da popularidade do formato se deve a sua elevada eficiência, unindo altas taxas de compressão e alta qualidade sonora.

Desenvolvido pelo *Moving Pictures Expert Group*, o padrão *MPEG-1* é, na verdade, composto de diversas partes, incluindo modelos para codificação de vídeo. Nesta seção será estudado apenas a parte relacionada à codificação de áudio, assim, toda referência a *MPEG-1* diz respeito apenas à parte de áudio.

Ao contrário dos formatos estudados anteriormente, o *MPEG-1* leva em consideração diversas características do sistema auditivo humano e não apenas características das ondas sonoras. Explorando diversas técnicas, o *MPEG-1* consegue descartar uma maior quantidade de informação que seria inaudível para nós e, com isso, alcança qualidades próximas a de um CD com cerca de 15% do tamanho do mesmo.

O *MPEG-1* é dividido em três camadas ou *layers* que vão aumentando sua complexidade computacional progressivamente ao mesmo tempo que a qualidade de compressão melhora, mas ainda assim é mantida a compatibilidade com as camadas anteriores, isto é, um decodificador da *layer III* possui todos os componentes necessários para decodificar arquivos da *layer I* e *II*.

Embora seja preciso seguir regras estritas para se adequar ao formato, garantindo que a saída decodificada esteja dentro do esperado, o padrão é abrangente o suficiente para permitir diversas implementações de codificação diferentes, o que permite uma evolução na qualidade final de um arquivo *MPEG-1*, conforme novas técnicas vão sendo desenvolvidas [11].

A principal área de melhorias relacionadas ao *MPEG-1* está no modelo psicoacústico utilizado [9], que determina quais informações sonoras seriam menos relevantes para o ouvido humano. É possível utilizar diferentes modelos psicoacústicos em diferentes implementações de codificadores pois o modelo utilizado não é necessário durante o processo de decodificação de um arquivo *MPEG-1*.

Características A taxa de amostragem em um arquivo *MPEG-1* pode ser de 32000, 44100 ou 48000 Hz.

As taxas de bits aceitas variam de 32 a até 224 kbps (kilobits por segundo) por canal de som, dependendo da *layer* em uso. Portanto, podemos ter taxas de até $2 \times 224 = 448$ kilobits por segundo. A lista completa das taxas de bits aceitas por cada camada pode ser vista na tabela 12. Além destas, o formato ainda aceita taxas de bits que não fazem parte do padrão, porém

não há garantias de que não haverá incompatibilidades na decodificação.

Layer I	**Layer II**	**Layer III**
32	32	32
64	48	40
96	56	48
128	64	56
160	80	64
192	96	80
224	112	96
256	128	112
288	160	128
320	192	160
352	224	192
384	256	224
416	320	256
448	384	320

Tabela 12: Listagem das taxas de bits (em kbps) aceitas por cada camada *MPEG-1*.

Além do modo que utiliza uma taxa de bits constante para o arquivo inteiro (CBR ou *Constant Bit Rate*) que era originalmente utilizado pelos codificadores *MP3*, ainda existem modos que variam a taxa de bits de acordo com o necessário para garantir a qualidade final do arquivo (VBR ou *Variable Bit Rate*) e modos que variam a taxa de bits em torno de uma média pré-estabelecida (ABR ou *Average Bit Rate*).

O *MPEG-1* suporta diversas configurações de canais de som: apenas um canal (mono), dois canais mono independentes, estéreo e *joint-stereo*. *Joint-stereo* é um modo que se aproveita das correlações entre os canais estéreo para diminuir a taxa de bits utilizada, através de uma técnica chamada *intensity stereo* [9, 13], que codifica a soma dos dois canais ao invés de enviá-los separadamente, mas utiliza diferentes escalas na hora da decodificação, se aproveitando da forma como o ouvido humano percebe sinais estéreo em freqüências acima de 2000 Hz. Este modo foi idealizado para a codificação a baixas taxas de bits e só é aplicado quando necessário, ou seja, também são usados dois canais distintos se houverem bits suficientes.

Além destes modos, a *layer III* ainda conta com um modo estéreo que armazena, ao invés dos canais esquerdo e direito, a soma e a diferença deles, reconstruindo a informação estéreo sem perdas a partir destes dados.

Estrutura Básica O processo realizado durante a codificação envolve, basicamente, as seguintes etapas:

1. Transformação da onda sonora para o espectro de freqüências;
2. Divisão do espectro de freqüências em subbandas;
3. Simultaneamente, o som é analisado pelo modelo psicoacústico;
4. Quantização das amostras, através das informações passadas pelo modelo psicoacústico;
5. Empacotamento dos dados, para se adequar ao padrão *MPEG-1*.

A decodificação é, essencialmente, o procedimento inverso:

1. Desempacotamos os dados;
2. Reconstruímos os valores das subbandas;
3. Juntamos todas as subbandas e fazemos a transformação para o domínio do tempo.

Note que, para a decodificação, o modelo psicoacústico não é necessário.

Modelos Psicoacústicos Os modelos psicoacústicos analisam certas características do sinal de entrada para determinar quais sons serão ou não ouvidos pelo sistema auditivo humano.

Isso é possível devido a diversas limitações que nossos ouvidos possuem. Na presença de sons mais fortes em uma dada freqüência, sinais situados em freqüências próximas acabam se tornando inaudíveis devido às *regiões críticas*, fenômeno chamado de *mascaramento simultâneo*.

Mesmo sob o silêncio, a resposta de nosso ouvido para as mais diversas freqüências não é linear, variando ainda conforme a nossa idade. Sabendo deste fato e do mascaramento simultâneo, podemos concluir que algumas faixas de freqüência devem receber mais bits do que outras durante sua quantização.

Na figura 5, podemos observar estes fenômenos: todos os sinais abaixo dos limites indicados não serão ouvidos pelos humanos e podem ser codificados com uma menor precisão.

Além disso, a presença de um sinal mais forte também pode mascarar sons imediatamente anteriores (*pré-mascaramento*) e imediatamente posteriores (*pós-mascaramento*), fenômeno que é conhecido como *mascaramento*

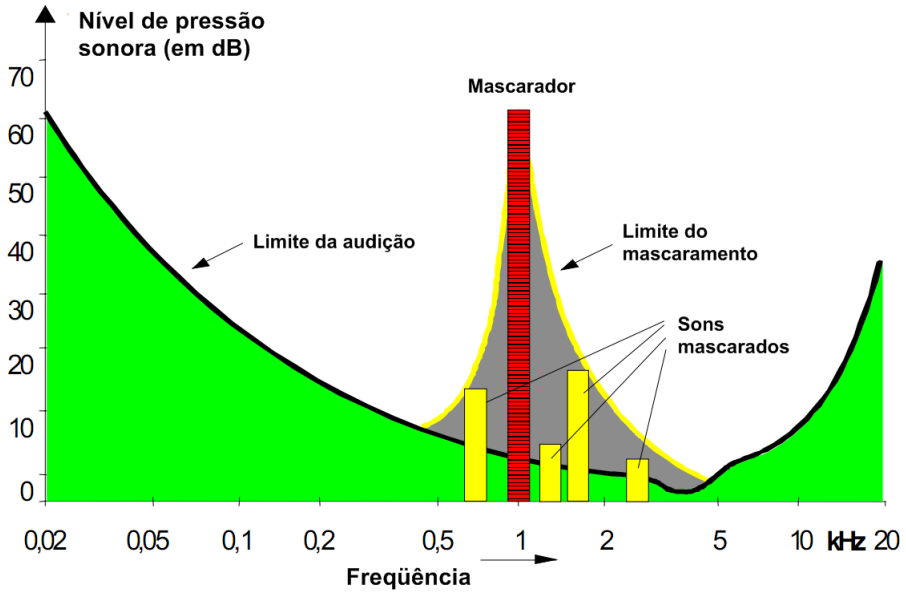

Figura 5: Mascaramento simultâneo. Imagem adaptada de [10].

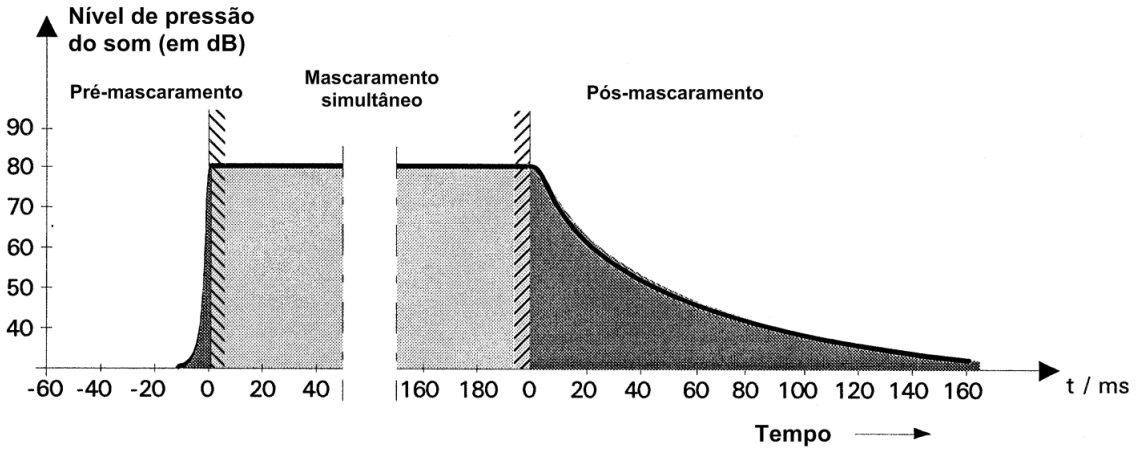

Figura 6: Mascaramento temporal. Imagem adaptada de [9].

temporal. A figura 6 ilustra o mascaramento temporal. Usualmente, o pré-mascaramento é menor que o pós-mascaramento, como pode ser visto na figura.

As regiões críticas do ouvido, apresentadas na tabela 13, indicam aproximadamente, as áreas que são afetadas pelo mascaramento simultâneo. Por exemplo, um sinal de freqüência 13000 Hz, pode tornar sinais de 11625 Hz até 15375 Hz inaudíveis dependendo de sua amplitude.

Índice	**Limite Superior da Banda (em Hz)**	**Índice**	**Limite Superior da Banda (em Hz)**
1	50	15	1,970
2	95	16	2,340
3	140	17	2,720
4	235	18	3,280
5	330	19	3,840
6	420	20	4,690
7	560	21	5,440
8	660	22	6,375
9	800	23	7,690
10	940	24	9,375
11	1,125	25	11,625
12	1,265	26	15,375
13	1,500	27	20,250
14	1,735		

Tabela 13: Limites aproximados das regiões críticas do ouvido.

Com um bom modelo psicoacústico, o *MPEG-1* consegue classificar com quais subbandas é necessário utilizar uma quantidade maior de bits para mascarar o ruído introduzido pela quantização, representando-as com uma maior precisão e tornando o ruído inaudível.

Idealmente, um codificador perfeito ajustaria a quantidade de bits para cada subbanda de forma que o ruído de quantização fosse inaudível. Na prática, porém, é melhor utilizar uma quantidade maior do que a necessária de bits, pois em muitos casos, o usuário final deseja realizar algum tipo de pós-processamento, como equalização, o que poderia tornar o ruído perceptível [10].

Divisão em Subbandas Para dividir o espectro de freqüências em subbandas é utilizado um *banco de filtros*, comum a todas as *layers*, que divide o espectro de freqüências em 32 subbandas de tamanhos iguais. Para um ar-

quivo com taxa de amostragem de 48000 Hz, por exemplo, freqüências de até 24000 Hz são representadas e cada subbanda cobrirá uma faixa de $\frac{24000}{32} = 750$ Hz.

Um problema que isto causa é que, ao considerarmos que as regiões críticas do ouvido não possuem tamanho constante, uma mesma subbanda pode abrigar diversas regiões críticas.

Outros compromissos foram tomados no projeto deste banco de filtros: a transformação e sua inversa geram perdas no sinal, isto é, mesmo que o sinal não seja quantizado posteriormente, a onda original não será reconstruída perfeitamente, porém a diferença entre elas é pequena e inaudível. Além disso, as subbandas adjacentes se sobrepõem em uma certa faixa de freqüência, ou seja, pode ocorrer que um sinal em uma dada freqüência seja representado em duas subbandas distintas.

Apesar disso, o banco de filtros utilizado cumpre bem o papel dado a ele, sendo rápido de calcular e providenciando uma boa resolução temporal e de freqüência; a cada 32 amostras da entrada, o banco de filtros produz 32 amostras de freqüência, uma para cada subbanda.

Layers Apesar de as três camadas do *MPEG-1* serem baseadas no mesmo princípio, elas possuem diferenças significativas entre elas. Em especial, a *layer III* introduz métodos muito mais sofisticados para alcançar uma taxa de compressão mais alta.

Layer I Utilizando-se do banco de filtros descrito anteriormente, a *layer I* agrupa 12 amostras de cada uma das 32 subbandas, empacotando em um *frame* 384 amostras.

Para cada um dos grupos de 12 amostras é atribuído a quantidade de bits alocada para representá-las e um fator de escala.

O fator de escala tem como propósito utilizar totalmente os níveis de quantização disponíveis. É um fator de multiplicação que normaliza a maior amostra da subbanda para 1, sendo escolhido um dentre os 62 valores pré-definidos pelo padrão.

A quantidade de bits alocada para cada subbanda é determinada pelo modelo psicoacústico de forma a minimizar o ruído de quantização introduzido pelo processo, podendo utilizar de 0 a 15 bits por subbanda.

Além disso, um bloco da *layer I* contém o cabeçalho e, opcionalmente, valores para checagem de erros (CRC).

Layer II A *layer II* é uma versão da primeira camada com algumas pequenas melhorias. Ao invés de agrupar as subbandas de 12 em 12 amostras,

agora um mesmo bloco contém 36 amostras por subbanda, fazendo com que cada *frame* armazene 1,152 amostras.

Os fatores de escala e a quantidade de bits alocada por subbanda são armazenados de forma mais eficaz, fazendo com que a quantização possua mais bits disponíveis para realizar a compactação.

Cada grupo de 36 amostras é na verdade dividido em 3 subgrupos de 12 amostras. A cada subbanda é atribuída a quantidade de bits alocados, mas os fatores de escala agora podem ser compartilhados ou não entre os subgrupos. Em cada bloco, cada subbanda pode receber de um a três fatores de escala e, se o codificador julgar adequado, o mesmo fator de escala pode ser compartilhado por dois ou mais subgrupos.

Este compartilhamento é realizado quando os fatores de escala são suficientemente próximos ou o codificador considerar que o ruído introduzido pelo uso do mesmo fator para diversos subgrupos será mascarado pelo sistema auditivo.

Por causa disso, um campo adicional é criado no cabeçalho do *frame*, que indica como os fatores de escala devem ser compartilhados entre os subgrupos.

Outra modificação da *layer II* é a imposição de algumas restrições na alocação de bits para as subbandas médias e altas, economizando alguns bits e limitando as opções de quantização nestas subbandas.

Layer III Nesta camada, o processo utilizado mostra-se muito mais complexo e refinado que o das camadas anteriores. Após a divisão em subbandas pelo banco de filtros, a saída dos filtros é novamente processada, mas desta vez por uma transformada discreta de cosseno modificada (em inglês, *modified discrete cosine transform* ou *MDCT*) que produz uma melhor resolução espectral do sinal. Assim, o *MP3* consegue se adaptar melhor às regiões críticas do ouvido.

Processando-se os valores obtidos através da *MDCT* ainda é possível tratar a sobreposição de subbandas causada pelo banco de filtros original.

A quantização realizada não é uniforme, tornando o ruído introduzido pela quantização melhor distribuído entre os valores possíveis. Os valores quantizados ainda usam a codificação de *Huffman* para uma melhor compressão de dados.

Por causa disso, os códigos produzidos têm tamanho variável, fazendo com que os valores codificados não necessariamente preencham exatamente o tamanho fixo de um bloco, o que levou à criação de um reservatório de bits. Conforme necessário, o codificador doa ou empresta bits deste reservatório.

4 Resultados Experimentais

Os testes foram realizados com os seguintes CDs, abrangendo uma diversidade de gêneros musicais:

1. *Awake* de 1994, da banda *Dream Theater* (*Rock/Heavy Metal*)
2. *Uncle Charles* de 2007, da *SoundScape Big Band Jazz* (*Jazz*)
3. *Time Out* de 1959, do *The Dave Brubeck Quartet* (*Jazz*)
4. *Cartola* de 1976, de *Cartola* (*MPB*)
5. *Acabou Chorare* de 1974, dos *Novos Baianos* (*MPB*)
6. *Heitor Villa-Lobos - Obra Completa para Violão Solo* de 1978, de *Sérgio e Odair Assad* (Música Clássica)
7. *Tchaikovsky Festival* de 1994, da *Royal Philharmonic Orchestra* (Música Clássica)

Cada faixa do CD foi convertida nos formatos desenvolvidos e estudados no trabalho, além de serem comprimidas com o *ZIP* e *FLAC*[4], ambos *lossless*, para efeito de comparação com o *AudioPaK*.

Para o *FLAC*, foi usada a opção -5, padrão do codificador. Modificando essa opção, é possível aumentar ou diminuir a compressão em troca do consumo de CPU na codificação e decodificação.

A compressão em *MP3* foi feita usando o *LAME*[5], com a opção -V2, que fica com taxas de bits médias próximas a 192 kbps e é considerado com qualidade de CD pela maioria das pessoas.

Nas tabelas a seguir estão dispostos o tamanho do CD compactado nos diversos formatos testados, além da menor e maior taxa de bits por segundo registrada nas faixas individuais.

Dependendo do gênero musical e estilo da música, as taxas de compressão *lossless* variam bastante, como também pode ser visto em [4]. De fato, a música com pior taxa de compressão no teste realizado em [4] faz parte do CD *Awake* que foi utilizado neste teste. Esse fenômeno deve-se principalmente aos estimadores utilizados: as características sonoras de alguns genêros musicais como o *Rock* acabam resultando em amostras residuais maiores que outros genêros como a música clássica, por exemplo, e geram arquivos com uma menor taxa de compressão ao final do processo.

[4] http://flac.sourceforge.net
[5] http://lame.sourceforge.net

Dream Theater - Awake				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	793.922	100%	1411	1411
ZIP	745.946	94%	1309	1374
AudioPaK	598.156	75,3%	957	1138
FLAC (-5)	558.540	70,3%	881	1071
μ-law	396.961	50%	705	705
IMA ADPCM	198.480	25%	352	352
MP3 (LAME -V2)	123.691	15,6%	197	234

Tabela 14: Resultados para os diversos formatos para o CD *Awake*.

SoundScape Big Band Jazz - Uncle Charles				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	682.790	100%	1411	1411
ZIP	620.422	90,9%	1219	1307
AudioPaK	448.943	65,7%	772	989
FLAC (-5)	399.336	58,5%	697	892
μ-law	341.395	50%	705	705
IMA ADPCM	170.697	25%	352	352
MP3 (LAME -V2)	89.729	13,1%	173	202

Tabela 15: Resultados para os diversos formatos para o CD *Uncle Charles*.

The Dave Brubeck Quartet - Time Out				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	398.754	100%	1411	1411
ZIP	359.439	90,1%	1240	1313
AudioPaK	243.759	61,1%	785	935
FLAC (-5)	227.041	56,9%	725	862
μ-law	199.377	50%	705	705
IMA ADPCM	99.688	25%	352	352
MP3 (LAME -V2)	53.661	13,5%	187	194

Tabela 16: Resultados para os diversos formatos para o CD *Time Out*.

Cartola - Cartola				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	368.052	100%	1411	1411
ZIP	328.960	89,4%	1219	1288
AudioPaK	213.364	58%	739	886
FLAC (-5)	185.202	50,3%	659	761
μ-law	184.026	50%	705	705
IMA ADPCM	92.013	25%	352	352
MP3 (LAME -V2)	54.580	14,8%	193	220

Tabela 17: Resultados para os diversos formatos para o CD *Cartola*.

Novos Baianos - Acabou Chorare				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	408.497	100%	1411	1411
ZIP	386.929	94,7%	1313	1354
AudioPaK	273.251	66,9%	746	1048
FLAC (-5)	236.806	58%	652	907
μ-law	204.248	50%	705	705
IMA ADPCM	102.124	25%	352	352
MP3 (LAME -V2)	52.075	12,7%	170	189

Tabela 18: Resultados para os diversos formatos para o CD *Acabou Chorare*.

Heitor Villa-Lobos - Obra Completa para Violão Solo				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	741.108	100%	1411	1411
ZIP	669.347	90,3%	1188	1301
AudioPaK	373.013	50,3%	662	742
FLAC (-5)	321.670	43,4%	555	659
μ-law	370.554	50%	705	705
IMA ADPCM	185.277	25%	352	352
MP3 (LAME -V2)	92.620	12,5%	163	188

Tabela 19: Resultados para os diversos formatos para o CD *Heitor Villa-Lobos - Obra Completa para Violão Solo*.

Royal Philharmonic Orchestra - Tchaikovsky Festival				
Formato	**Tamanho (em kB)**	**Porcentagem do original**	**Menor taxa de bits (kbps)**	**Maior taxa de bits (kbps)**
PCM WAV	610.103	100%	1411	1411
ZIP	508.383	83,3%	1163	1229
AudioPaK	295.792	48,5%	639	765
FLAC (-5)	271.801	44,5%	588	708
μ-law	305.051	50%	705	705
IMA ADPCM	152.525	25%	352	352
MP3 (LAME -V2)	82.377	13,5%	188	195

Tabela 20: Resultados para os diversos formatos para o CD *Tchaikovsky Festival*.

Comparativamente, o *FLAC* obteve uma diferença na taxa de bits em torno de 80 a 100 kbps em relação ao *AudioPaK*, fazendo com que todos os CDs testados ficassem menores quando comprimidos com o *FLAC*. Um caso extremo pode ser observado na tabela 19, onde a maior taxa de bits obtida pelo *FLAC* ainda era menor que a taxa de bits mínima do *AudioPaK*.

É necessário levar em consideração que o *FLAC*, apesar de se basear em uma técnica semelhante ao *AudioPaK*, não é uma implementação tão ingênua quanto à realizada no trabalho e, além disso, ainda se aproveita de outras técnicas mais avançadas, justificando o desempenho de compressão ligeiramente superior obtido.

Porém, mesmo com as melhorias que o *FLAC* possui, nos testes realizados a diferença de compressão obtida entre os arquivos codificados em ambos os formatos específicos para compressão *lossless* de áudio foi de apenas 4% a 8% do tamanho original do arquivo, valor pequeno em relação à diferença entre os compressores *lossless* de áudio e o compressor de uso geral *ZIP*, que foi de cerca de 40% do tamanho original do arquivo.

Outro ponto a se destacar é a baixa taxa de compressão alcançada pelo *μ-law*. Mesmo sendo um formato *lossy*, em alguns CDs, os compressores *lossless* obtiveram uma taxa de compressão maior ou muito próxima a do *μ-law*, como pode ser visto nas tabelas 17, 19 e 20.

Nas tabelas ainda é possível observar a alta eficiência do formato *MP3*, obtendo arquivos finais com cerca de 15% do original e a grande melhoria em se usar técnicas de compressão *lossless* específicas para áudio comparado a técnicas como o *ZIP* (que tem arquivos comprimidos em torno de 90% do tamanho original apenas, comparado aos 76% obtidos no pior caso do *AudioPaK* e do *FLAC*).

5 Conclusões

Arquivos de som estão dentre os que mais utilizam espaço em disco atualmente. A baixa taxa de compressão atingida com algoritmos como o *ZIP* mostram que o desenvolvimento de técnicas específicas para se aproveitar das características das ondas sonoras pode ser interessante.

Os resultados obtidos com compressões *lossless* como o *AudioPaK* mostram que é possível extrair uma taxa de compressão muito maior ao se explorar as correlações entre as amostras. Mesmo comparando o desempenho de um codificador extremamente simples como o *AudioPaK* com codificadores mais robustos como o *FLAC*, a diferença de compressão ao final do processo é consideravelmente próxima, o que pode indicar que o limite de compressão *lossless* está muito próximo de ser atingido.

Para o uso do dia-a-dia, isto é, quando não se deseja realizar algum tipo de pós-processamento ou ter alguma finalidade de arquivamento, por exemplo, a compressão *lossless* não possui atrativos suficientes, já que os arquivos *lossy* mais sofisticados têm arquivos finais de tamanho muito inferior e uma qualidade sonora por muitas vezes indistinguível da original.

Para isso, modelos como o μ-*law* e *IMA ADPCM* acabam não cumprindo bem seu papel. Apesar de terem sido importantes numa época em que o *MP3* era complexo demais para ser decodificado em tempo real, atualmente, os processadores ultrapassaram e muito a necessidade de cálculos exigida pelo *MP3*.

O uso de modelos psicoacústicos que se aproveitam das deficiências do sistema auditivo humano mudou completamente as técnicas de compressão *lossy*. Antes baseadas apenas nas características das ondas sonoras, hoje, os codificadores modernos usam modelos psicoacústicos para reduzir o tamanho do arquivo final sem comprometer sua qualidade.

Referências

[1] Wave File Format. http://www.sonicspot.com/guide/wavefiles.html. Acessado em 21 de novembro de 2008.

[2] WAVE PCM soundfile format. http://ccrma.stanford.edu/CCRMA/Courses/422/projects/WaveFormat/. Acessado em 21 de novembro de 2008.

[3] RIFF WAVE (.WAV) file format. http://sox.sourceforge.net/AudioFormats-11.html#ss11.6. Acessado em 21 de novembro de 2008.

[4] FLAC - comparison. http://flac.sourceforge.net/comparison.html. Acessado em 30 de novembro de 2008.

[5] M. Hans and RW Schafer. Lossless compression of digital audio. *Signal Processing Magazine, IEEE*, 18(4):21–32, 2001.

[6] P. Kabal. Audio File Format Specifications. http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html, 2006. Acessado em 21 de novembro de 2008.

[7] Microsoft Corporation. *New Multimedia Data Types and Data Techniques*, April 1994. Revision 3.0.

[8] F.R. Moore. *Elements of computer music*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1990.

[9] P. Noll. MPEG digital audio coding. *Signal Processing Magazine, IEEE*, 14(5):59–81, 1997.

[10] P. Noll and T. Liebchen. DIGITAL AUDIO: FROM LOSSLESS TO TRANSPARENT CODING. In *IEEE Signal Processing Workshop*, 1999.

[11] D. Pan, M. Inc, and IL Schaumburg. A tutorial on MPEG/audio compression. *Multimedia, IEEE*, 2(2):60–74, 1995.

[12] D.Y. Pan. Digital Audio Compression. *Digital Technical Journal*, 5(2):28–40, 1993.

[13] S. Shlien. Guide to MPEG-1 audio standard. *Broadcasting, IEEE Transactions on*, 40(4):206–218, 1994.

Parte II
Parte Subjetiva

6 Dificuldades Encontradas

Durante o desenvolvimento dos diversos codificadores produzidos várias adversidades acabaram tendo de ser superadas. Vários detalhes de implementação não eram suficientemente explicados nos artigos lidos, o que possibilitava algumas suposições erradas que foram corrigidas conforme iam sendo detectadas, assim como soluções próprias para os problemas encontrados.

6.1 AudioPaK

Para a implementação do primeiro formato, foi necessário primeiramente fazer a leitura correta de um arquivo *WAVE*, o que seria aproveitado para o restante dos codificadores. Apesar de ser um formato simples, que consiste somente de um cabeçalho e as amostras necessárias, a leitura correta dos arquivos no formato causaram diversas dificuldades.

A fim de facilitar os testes, o cabeçalho e as amostras lidas eram impressas em um arquivo em formato ASCII e seus valores eram analisados. Alguns arquivos lidos acabavam imprimindo amostras com números absurdos em determinados instantes.

Por exemplo, no meio do arquivo, um canal de som que vinha se comportando como esperado repentinamente passava a exibir valores completamente aleatórios e esses números começavam a aparecer em ambos os canais até que novamente, os valores depois de uma certa quantidade de amostras voltavam a imprimir o que era esperado.

A suspeita inicial estava, obviamente, nas funções que faziam a leitura das amostras, fazendo as conversões necessárias para tratar a ordem dos bytes e para adequar os valores negativos e positivos como sendo complemento de 2. Acreditando que elas estavam causando os erros, diversas modificações delas foram testadas, todas sem sucesso.

Assim, foi feita uma análise de quando as amostras passavam a mostrar valores que não condiziam com o esperado. Através do valor numérico, foi notado que esse comportamento só era observado ao serem encontrados dois valores nas amostras, os que correspondiam a "\n" e "\r" na tabela ASCII. O sistema operacional, ao ler estes caracteres, tratava como um caractere de nova linha padrão do mesmo, ou seja, ambos eram convertidos para "\n\r". Com isso, o problema foi facilmente resolvido, bastando abrir o

arquivo *WAVE* em modo binário para a leitura ocorrer conforme o esperado. Apesar de parecer um problema simples, ele reapareceu mais à frente, como será explicado.

Durante o desenvolvimento do codificador, outros problemas surgiram. Inicialmente, o arquivo codificado era criado a partir da saída padrão redirecionada para um arquivo. Seguindo o artigo, o algoritmo para o cálculo dos residuais foi facilmente implementado; a codificação de Golomb, no entanto, gerou alguns problemas.

As funções de leitura e escrita de bits foram criadas, assim como as funções que codificavam e decodificavam números no código de Golomb. Inicialmente, um pequeno trecho de código disponibilizado no artigo do *AudioPaK* foi utilizado para estimar o valor do parâmetro k sem a necessidade de se utilizar operações de ponto flutuante, mas durante testes realizados, a taxa de compressão dos arquivos gerados deste modo se mostravam maiores do que os que usavam a fórmula dada pela equação (3).

Por outro lado, ao utilizarmos esta equação, dependendo do arquivo codificado, o programa terminava com erros. Ao analisar a fórmula, podemos observar que, da forma que ela é dada, k pode assumir valores negativos e, em casos extremos, ter um valor inválido, já que se um bloco for composto apenas por 0, o logaritmo devolvia o valor máximo do inteiro, o que não era desejado.

Assim, foram tomados alguns cuidados com o cálculo de k e, com estes aspectos da codificação funcionando, conseguíamos obter um arquivo final, mas após a decodificação, o arquivo de saída não era idêntico ao de entrada.

Através de um editor hexadecimal, foi possível notar que os valores estavam corretos até certo ponto, quando passavam a ter valores completamente diferentes dos originais. Isso lembrou o problema enfrentado durante a leitura dos arquivos *WAVE* e realmente estava relacionado a ele. Ao gerar o arquivo através de um redirecionamento da saída padrão, novamente o arquivo não estava sendo tratado como binário e conversões com os caracteres de nova linha eram realizados.

Descoberto e consertado este problema, ainda assim o arquivo final diferia. Comparando os arquivos, foram observadas pequenas diferenças que indicavam que a função que realizava a conversão de números codificados em complemento de 2 não estava correta.

Outro pequeno inconveniente encontrado foi em relação ao alinhamento do cabeçalho, já que algumas funções utilizadas precisavam que o último bloco fosse alinhado em relação a um byte, o que teve de ser modificado para que elas manipulassem o mesmo buffer de bits que outras funções.

Superadas todas as dificuldades enfrentadas no desenvolvimento, finalmente obtive um compressor *lossless* que não demonstrou ter outros proble-

mas durante os testes realizados.

6.2 μ-law

O desenvolvimento do codificador de *μ-law* ocorreu sem maiores transtornos, pois a maioria das funções já estavam implementadas e testadas, sendo necessário somente realizar a transformação adequadamente e colocá-la na saída.

6.3 IMA ADPCM

O *IMA ADPCM* também ocasionou problemas durante o desenvolvimento, especialmente pela dificuldade em se encontrar a forma adequada de se organizar o arquivo final.

O código inicialmente desenvolvido armazenava as informações no arquivo de saída da mesma forma que em um arquivo *PCM*, apenas modificando o valor do tipo de arquivo no cabeçalho, para indicar que era do tipo *IMA ADPCM*. Como nenhum tocador de arquivos de som conseguia identificar o arquivo, foi iniciada uma busca pela internet para descobrir qual poderia ser o problema, sem muito sucesso, inicialmente.

Ao encontrar o site [1], novos campos foram adicionados ao cabeçalho *WAVE*, mas ainda assim, o arquivo final não era lido por nenhum programa.

Após muitas buscas, um site foi encontrado, que descrevia o desejado, mas continha diversas imagens que não podiam mais ser abertas e dificultava a compreensão da informação necessária. Porém, a partir do título de tal site foi encontrado, finalmente, a referência que precisava: um documento da *Microsoft* que descrevia os formatos de todos os tipos de arquivos *WAVE* [7].

A partir deste documento, além de poder codificar corretamente o arquivo para ser lido por outros programas, ele continha uma descrição do algoritmo *IMA ADPCM*, o que permitiu que alguns erros cometidos fossem corrigidos.

Diversos campos do cabeçalho *WAVE* tiveram seus valores modificados, fazendo com que os arquivos gerados entrassem dentro das especificações. Além disso, como as amostras eram organizadas de forma razoavelmente diferente, novas estruturas e novas funções de escrita em arquivo tiveram de ser criadas.

6.4 MPEG-1

Infelizmente, o codificador *MPEG-1* não pôde ser desenvolvido por falta de algumas informações necessárias ao desenvolvimento que só estão disponíveis

mediante a compra da especificação. Os artigos encontrados na ACM ou na IEEE não entravam em detalhes de implementação. Essa foi provavelmente a maior frustração do trabalho, pois seria bastante interessante desenvolver pelo menos um codificador que se adequasse à *layer I*.

7 Conceitos Estudados no Curso e Disciplinas Relevantes para o Trabalho

Diversas disciplinas acabam sendo relevantes para o trabalho, mesmo que às vezes de forma indireta. A matéria de *Computação Musical*, que certamente seria muito relevante para o trabalho, infelizmente não pôde ser cursada durante a graduação, devido a diversos conflitos de horário ocorridos.

Apesar disso, algumas matérias como *Visão e Processamento de Imagens* também acabaram auxiliando em algumas idéias que, embora não usadas diretamente no trabalho realizado, têm bastante uso na área de computação musical, como por exemplo, a transformada de Fourier.

Muito das técnicas utilizadas no desenvolvimento vêm do aprendido em disciplinas mais básicas, como *Estrutura de Dados* e *Princípios de Desenvolvimento de Algoritmos*. Em particular, na disciplina de *Estrutura de Dados*, um exercício-programa exigido foi um compressor de arquivos baseado no *LZ78*, fazendo com que precisássemos pela primeira vez ler arquivos bit a bit.

As disciplinas de *Laboratório de Programação* ensinaram a modularizar o código, utilizando *Makefile* para auxiliar na compilação, além de introduzir o uso de LaTeX na produção de textos, ferramenta de altíssima utilidade.

Além destas, matérias como *Cálculo*, *Estatística* e *Física* são necessárias para o entendimento de algumas técnicas utilizadas na compressão de dados, além de conceitos básicos sobre o som.

Também não posso deixar de agradecer ao apoio do professor Marcelo Queiroz, que me incentivou a pesquisar sobre a área com a qual eu tivesse mais afinidade, sanando as dúvidas que vieram a aparecer durante diversas etapas do trabalho de formatura com bastante clareza e paciência.

8 Medidas para se Aprofundar na Área do Trabalho

A área de compressão de áudio ainda possui muito mais técnicas além das estudadas neste trabalho. Uma breve visita aos sites de codificadores de

código aberto como o *FLAC* (*lossless*) ou o *Vorbis*[6] (*lossy*), já é possível encontrar diversos outros métodos desenvolvidos no intuito de se melhorar a taxa de compressão e/ou nível de qualidade dos codificadores.

Ainda que o *MP3* seja até hoje praticamente o padrão de compressão digital de áudio, formatos mais novos tendem a suprir algumas deficiências do formato e conseguem uma qualidade maior em taxas de bits mais baixas (menores que 128 kbps), como por exemplo, o próprio Vorbis e o AAC (Advanced Audio Coding) com suas inúmeras variantes, que faz parte do padrão MPEG-2 e é empregado na TV digital brasileira.

Outra área que seria de bastante interesse é no ramo de compressão de sons multicanal, formato bastante utilizado em filmes e que deve possibilitar um maior aproveitamento das similaridades entre os diversos canais, produzindo bons resultados.

Mais estudos sobre psicoacústica e processamento digital de sinais estão dentre os temas que precisariam ser aprofundados para ser possível continuar na área, já que, como foi visto, a maior parte do cenário atual de compressão *lossy* se baseia em bons modelos psicoacústicos.

Outros ramos de compressão também poderiam ser estudados, como compressão de voz, que se utiliza de outras técnicas, aproveitando-se de modelos de produção da fala e compressão de vídeo, uma área que tem arquivos de tamanhos maiores ainda e atualmente conta com processos de compressão que exigem altíssimo poder computacional, mesmo para os dias de hoje.

[6] http://www.vorbis.com