UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RECUPERAÇÃO DE QUADROS DE ARQUIVOS DE VÍDEO H.264/AVC CORROMPIDOS

RAFAEL FARNESE

ORIENTADOR: CAMILO CHANG DÓREA.

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

ÁREA DE CONCENTRAÇÃO: INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO

PUBLICAÇÃO: PPGENE.DM-106/12
BRASÍLIA/DF: 02/12

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RECUPERAÇÃO DE QUADROS DE ARQUIVOS DE VÍDEO H.264/AVC CORROMPIDOS

RAFAEL FARNESE

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA

APROVADA POR:

_____
CAMILO CHANG DÓREA, Dr. UnB
(ORIENTADOR)

_____
ALEXANDRE ZAGHETTO, Dr. UNB
(EXAMINADOR INTERNO)

_____
BRUNO LUIGGI MACCHIAVELLO ESPINOZA, Dr. UnB
(EXAMINADOR EXTERNO)

BRASÍLIA/DF, 29 DE FEVEREIRO DE 2012

FICHA CATALOGRÁFICA

FARNESE, RAFAEL

Recuperação de Quadros de Arquivos de Vídeo H.264/AVC Corrompidos [Distrito Federal] 2012.

xiii, 116p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2012). Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. H.264	2. MP4
3. ISO/IEC 14496	4. Recuperação de quadros
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

FARNESE, R. (2012). Recuperação de Quadros de Arquivos de Vídeo H.264/AVC Corrompidos. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGENE.DM--106/12, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 116p.

CESSÃO DE DIREITOS

AUTOR: Rafael Farnese

TÍTULO: Recuperação de Quadros de Arquivos de Vídeo H.264/AVC Corrompidos

GRAU: Mestre ANO: 2012

_____

Rafael Farnese
Instituto de Criminalística – PCDF – SAISO, Bloco C.
CEP 70.610-200 - Brasília – DF – Brasil.

AGRADECIMENTO

Ângelo Shimabuko, muito obrigado por toda a sua ajuda.

AC, Luciano, Maurel e Scoralick, obrigado por compartilharem da vontade de fazer mais do que só o esperado.

Prof. Camilo, sua orientação foi fundamental. Muito obrigado.

RESUMO

RECUPERAÇÃO DE QUADROS DE ARQUIVOS DE VÍDEO H.264/AVC CORROMPIDOS
Autor: Rafael Farnese
Orientador: Prof. Camilo Chang Dórea
Programa de Pós-graduação em Engenharia Elétrica
Brasília, 29 de fevereiro de 2012

Esta dissertação trata da recuperação de quadros (imagens) de *bitstreams* de vídeo H.264/AVC armazenados em arquivos contêineres ISO/IEC 14496 ("`.mp4`") corrompidos. O interesse em tais padrões de vídeo e de contêiner se dá pela prevalência por eles alcançada no mercado e pela grande quantidade de solicitações recebidas pelo Instituto de Criminalística da Polícia Civil do Distrito Federal para exame de equipamentos (sistemas de videossegurança, *smartphones*, filmadoras, etc.) que geram ou armazenam dados nesse formato. Em diversas situações, esses arquivos de vídeo são localizados em mídias investigadas, mas, por se encontrarem danificados, não podem ser reproduzidos por tocadores (*players*) de mídia. Nesses casos, a recuperação forense de quadros que os compõem se mostraria de grande utilidade para a formação de provas periciais. O trabalho apresenta teoria sobre o padrão H.264/AVC e sobre as diferentes especificações de contêineres ISO/IEC 14496 e, na sequência, detalha e interpreta os dados (*bitstream*) e metadados de um arquivo de vídeo. Em seguida, diferentes graus de corrupção são produzidos no arquivo, sendo expostas as consequências em sua reprodução. Como resultado, consegue-se comprovar que quadros integrantes de *bitstream* desacompanhado de seus metadados são irrecuperáveis. Mais do que isso, um procedimento para recuperação de quadros de arquivos corrompidos é apresentada e demonstrada, sendo também disponibilizado um algoritmo para localização de todos os quadros não sobrescritos, desde que presentes os metadados.

Palavras-chave: vídeo, H.264/AVC, ISO/IEC 14496, MP4, recuperação de quadros.

ABSTRACT

RECOVERY OF FRAMES FROM H.264/AVC BITSTREAMS STORED IN DAMAGED ISO/IEC 14496 CONTAINERS

Author: Rafael Farnese
Supervisor: Camilo Chang Dórea
Brasília, 29th February 2012

This thesis presents a procedure for the recovery of frames (images) contained in H.264/AVC bitstreams stored in corrupted ISO/IEC 14496 container files ("`.mp4`"). The focus on these container and bitstream formats is due to the fact that they are currently one of the most commonly used formats for recording, compression and distribution of video. Indeed, most video files found in seized evidence (e.g. smartphones, flash drives, CCTVs) under examination in the digital forensic laboratory of the Brazilian Federal District Police are in these formats. There are many cases, however, in which video files are partly retrieved, but they turn out to be corrupted in such ways that media players cannot reproduce them. In these situations, one plausible way to find out what their contents are and, depending on what is found, use them in a court of law, is to forensically recover their frames. This work starts by laying out the theory behind the H.264/AVC standard and the ISO/IEC 14496 containers. Next, it scrutinizes all the metadata of a video file. Different levels of corruption are intentionally inflicted on this same file, allowing the consequences on their playback to be shown and explained. Lastly, it is shown that it is not possible to recover frames from bitstreams that are not accompanied by their respective metadata, and a technique for the recovery of frames from corrupted video files is presented and demonstrated, along with an optimized algorithm to locate all the non-overwritten frames related to these files, regardless of where they are stored in the media.

Keywords: H.264/AVC, ISO/IEC 14496, MP4, frame recovery.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

AVC	*Advanced Video Coding*
CABAC	Codificação Aritmética Binária Adaptativa ao Contexto
CAVLC	Codificação Adaptativa de Comprimento de Palavra Variável
CODEC	*EnCOder/DECoder*
DPB	*Decoded Picture Buffer*
ISSO	*International Standard Organization*
ITU	*International Telecommunication Union*
MB	Macrobloco
MJPEG	*Motion* JPEG
MPEG	*Moving Picture Experts Group*
NAL	*Network Abstraction Layer* (Camada de Abstração de Rede)
PPS	*Picture Parameter Set*
QP	*Quantization Parameter* (Parâmetro de Quantização)
Qstep	*Quantizer Step Size* (Tamanho do Passo de Quantização)
QTFF	*Quick Time File Format*
SPS	*Sequence Parameter Set*
SEI	*Supplemental Enhancement Information*
VUI	*Video Usability Information*
VCL	*Video Coding Layer* (Camada de Codificação de Vídeo)
WMV	*Windows Media Video*

1 INTRODUÇÃO

Perícias criminais têm como objetivo examinar vestígios de crimes e elaborar laudos periciais que subsidiem a investigação criminal e a persecução penal. Busca-se identificar, por meio desse exame técnico, a materialidade (conjunto de elementos que tornam evidente a criminalidade de um ato), a autoria e a dinâmica de um ato criminoso, reconstituindo, assim, a cena do crime [1]. Perícias criminais de informática, por sua vez, lidam especificamente com vestígios digitais. Grande parte desses vestígios encontra-se em mídias de armazenamento de dados, como discos rígidos, cartões de memória, *pendrives* e memórias internas de *smartphones* e aparelhos de telefonia celular.

Dentre os tipos de dados encontrados em mídias de armazenamento apreendidas, arquivos de vídeo têm recebido grande destaque. Isso porque é cada vez maior a quantidade de dispositivos portáteis que permitem gravação de vídeos. Pesquisas em sítios de venda de aparelhos de telefonia celular, por exemplo, mostram que escassos são os modelos desprovidos de câmera embutida. O compartilhamento desses vídeos também tem aumentado recentemente, principalmente em função da popularização de redes sociais na Internet que disponibilizam essa funcionalidade. A onipresença dessas câmeras filmadoras portáteis faz com que, com razoável frequência, pessoas sejam filmadas em prática de atos delituosos. Consequentemente, a descoberta desses arquivos em mídias apreendidas pode levar à formação de provas irrefutáveis.

Ocorre, porém, que, com razoável frequência, os arquivos de vídeo encontrados nas mídias apresentam-se corrompidos. Essas situações acontecem, principalmente, nos casos onde somente é possível a recuperação parcial do conteúdo dos arquivos que haviam sido apagados antes da apreensão da mídia. Há, inclusive, situações onde o nome do vídeo é sugestivo, como, por exemplo, `12aninhos.3gp`, mas a corrupção do arquivo impede que o vídeo seja reproduzido pelos diversos tocadores disponíveis, como o Windows Media Player [2], o Media Player Classic [3], o Totem [4] ou o VLC Media Player [5]. Assim, por mais que haja a suspeita de que um arquivo contenha cenas incriminadoras, os peritos criminais ficam impossibilitados de comprovar a ação delituosa.

Nessas situações, sabemos que, pela natureza de sua corrupção, o arquivo de vídeo muito provavelmente não pode ser restaurado à sua integridade original ou a um estado que permita sua reprodução. No entanto, para a maioria dos casos bastaria a recuperação de um dos quadros (imagens) que o compõem para que a prova pudesse ser

produzida. Exemplos dessas situações são quadros comprovando i) filmagem do teclado de um terminal de autoatendimento; ii) porte ilegal de arma de fogo, iii) sedução de menor ou iv) tráfico de drogas (ver Figura 1.1). Em todos esses casos, a impossibilidade de reprodução dos vídeos pode ser compensada pela recuperação de quadros individuais.

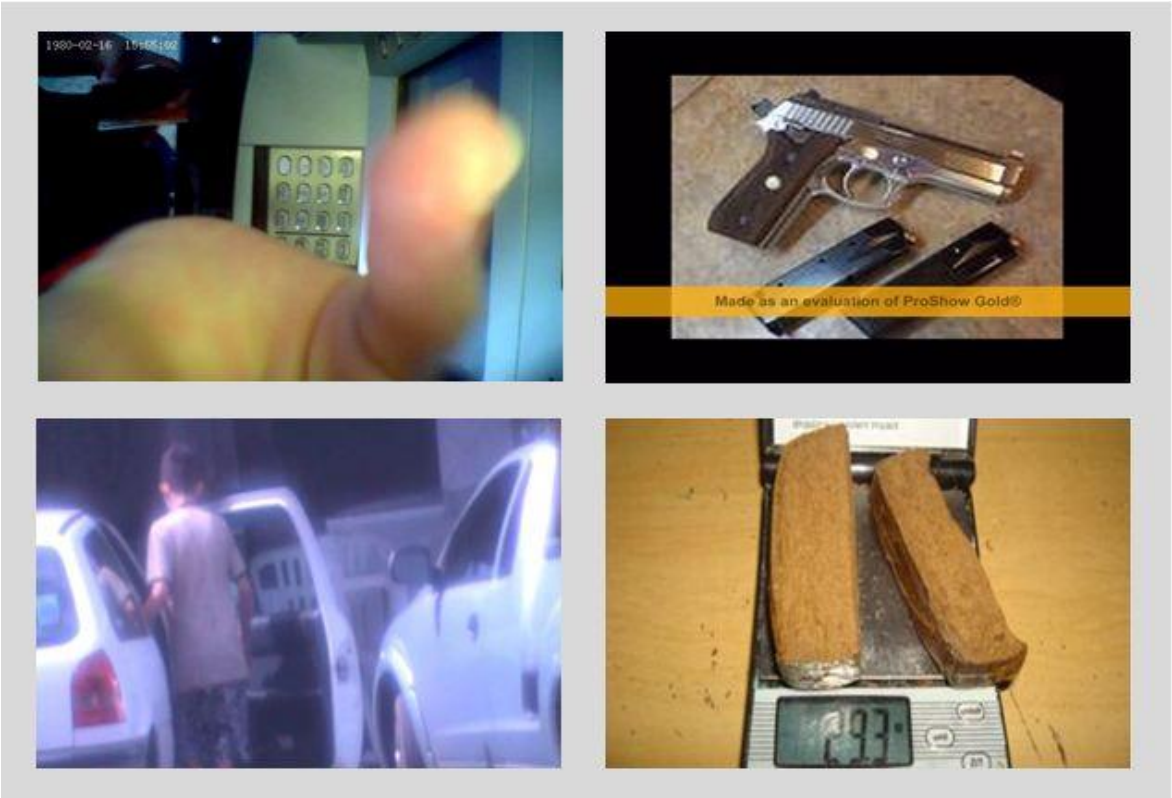

Figura 1.1: Quadros ilustrando filmagens de teclado de ATM, posse de arma de fogo, sedução de menor e comercialização de entorpecentes.

A extração dos quadros que compõem um arquivo de vídeo íntegro pode ser feita por ferramentas como VirtualDub [6], Adobe Premiere [7] ou ImageGrab [8]. No entanto, um pré-requisito para o uso desses aplicativos é que os vídeos estejam íntegros. A extração de quadros de vídeos corrompidos (irreproduzíveis) não é possível com as ferramentas hoje disponíveis, e tampouco foram identificadas publicações de trabalhos científicos que tratem especificamente desse tema.

No interesse de se realizar essa recuperação de quadros, há um fator básico que deve ser levados em consideração: o padrão de codificação utilizado para a criação do fluxo (*bitstream*) de vídeo. Todo fluxo de vídeo deve atender a algum padrão de codificação. H.262/MPEG-2 Part 2, MJPEG, Windows Media Video (WMV), MPEG-4

Part 2, H.264/AVC e VC-1 são exemplos de padrões. Um vídeo codificado de acordo com uma determinada especificação exige, para a sua reprodução, um decodificador compatível, uma vez que cada padrão possui sintaxe e semântica próprias. Da mesma forma, a análise pericial de fluxos de vídeo corrompidos deve necessariamente levar em consideração o padrão utilizado.

De todos os padrões existentes, um apresenta-se proeminente, do ponto de vista pericial: H.264/AVC [9] [10]. A destacada importância desse padrão se dá pelo motivo de ele ser amplamente suportado por variados equipamentos e aplicativos, incluindo sistemas de videossegurança, aparelhos de telefonia celular, smartphones, filmadoras e máquinas fotográficas digitais [11]. Consequentemente, uma grande parte dos arquivos de vídeo armazenados nesses tipos de equipamento, quando encaminhados para perícia, encontra-se nesse formato de codificação. Mais do que isso, o suporte a esse padrão tem crescido fortemente em comparação com outros padrões tradicionais, especialmente o MPEG-4 Part 2 (comumente conhecido como Xvid e DivX) [12], em grande parte devido a seu desempenho comprovadamente superior (melhor relação qualidade perceptual/*bitrate*) [13].

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Determinar a viabilidade técnica da recuperação de quadros constantes de arquivos de vídeo H.264/AVC corrompidos, estabelecendo o procedimento requerido, os pré-requisitos e as limitações.

1.1.2 Objetivos Específicos

- Analisar a sintaxe e a semântica de *bitstreams* H.264/AVC;
- Detalhar o processo de codificação de um vídeo H.264/AVC;
- Estudar os padrões de contêineres ISO/IEC 14496 ("`.MP4`") utilizados para armazenar *bitstreams* H.264/AVC;
- Detalhar e interpretar o conteúdo dos boxes de metadados existentes em um arquivo ISO/IEC 14496 contendo *bitstream* H.264/AVC, em um processo

exaustivo e pioneiro de decodificação dos bits que representam seus inúmeros campos e parâmetros;

- Decodificar e interpretar os dados de um *bitstream* H.264/AVC;
- Realizar testes de reprodução de arquivo de vídeo com diferentes tipos e graus de corrupção, dos metadados e do *bitstream* em si, a fim de entender as consequências na reprodução dos vídeos;
- Determinar a viabilidade de recuperação de quadros de *bitstreams* H.264/AVC, estando ausentes os seus metadados;
- Apresentar um procedimento de recuperação de quadros de arquivos de vídeo H.264/AVC corrompidos e irreproduzíveis.

1.2 ESTRUTURA DO TRABALHO

Esta dissertação está dividida em cinco capítulos principais. O primeiro é responsável por descrever a importância, a premência e a aplicabilidade do trabalho, relacionando-o às ciências forenses.

O segundo capítulo busca apresentar, de maneira objetiva, conceitos e informações relevantes para o embasamento teórico requerido, incluindo detalhamento do padrão de vídeo sob estudo, H.264/AVC, e dos diferentes tipos de contêineres ISO/IEC 14496 (arquivos "`.MP4`") utilizados para armazenamento desse tipo de vídeo.

O terceiro capítulo detalha e interpreta os campos de metadados presentes em arquivo ISO/IEC 14496 contendo *bitstream* H.264/AVC. O mesmo é feito com os dados contidos no *bitstream* em si, visando a determinar se é possível a recuperação de quadros de arquivo de vídeo que não possua metadados.

No quarto capítulo, diferentes versões de um arquivo de vídeo H.264/AVC em container ISO/IEC 14496 são geradas, com diferentes formas e graus de corrupção, no intuito de entender as consequências dessas corrupções em sua reprodução. Uma técnica de recuperação de quadros (imagens) constantes de arquivos corrompidos e não reproduzíveis é, então, apresentada, sendo comprovada sua eficácia por meio de um caso concreto. Apresenta-se, ao fim deste capítulo, um algoritmo que detalha os passos a serem seguidos para a recuperação de quadros de vídeo.

Encerrando a dissertação, o quinto capítulo apresenta as conclusões do trabalho, discute os resultados alcançados e sugere propostas para trabalhos futuros.

1.3 METODOLOGIA

A metodologia utilizada no presente trabalho é do tipo qualitativa mista, sendo composta por uma primeira parte, com elementos de pesquisa descritiva, e uma segunda parte experimental, tendo esta se subdividido, por sua vez, em duas frentes complementares: estudo de caso e pesquisa-ação.

A pesquisa descritiva tem a importância fundamental de permitir o entendimento da teoria por trás do objeto de estudo: arquivos de vídeo H.264/AVC em containers ISO/IEC 14496 ("`.MP4`"). Essa pesquisa engloba a especificação do padrão H.264/AVC, os procedimentos requeridos para a codificação de um vídeo nesse formato, a forma de organização dos dados (sintaxe e semântica) e os passos necessários para a correta decodificação. Igualmente importante, este passo de pesquisa descritiva abrange estudo das especificações dos diversos padrões de contêineres ISO/IEC 14496 (arquivos "`.MP4`").

O método de estudo de caso permite o entendimento de determinados fatos por meio da análise de casos isolados. A lógica por trás deste método é a de que o estudo de um caso em profundidade pode ser considerado representativo de muitos outros, ou mesmo de todos os casos semelhantes [14]. De fato, a análise e a interpretação aprofundadas dos metadados presentes em um arquivo ISO/IEC 14496 contendo *bistream* H.264/AVC, assim como do *bitstream* em si, permitiram o entendimento da sintaxe e da semântica dessas informações, com validade não somente para o caso específico, mas sim para todos os arquivos que igualmente atendam aos padrões ISO/IEC 14496 e H.264/AVC. Resposta a um questionamento de grande relevância (*é possível a recuperação de quadros de bitstream desacompanhado de seus metadados?*) pôde ser categoricamente alcançada por meio deste método.

A pesquisa-ação, como o nome indica, visa a produzir mudanças (ação) e compreensão (pesquisa). Possui uma base empírica que é concebida e realizada através de uma estreita relação com uma ação ou com a resolução de um problema [15]. Essa inquirição empírica permitiu que, por meio da análise de um arquivo de vídeo propositalmente corrompido de diferentes formas, fosse identificada não só a condição

mínima de integridade necessária para sua reprodução, mas também, e principalmente, um procedimento para recuperação de quadros de arquivos corrompidos irreproduzíveis.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada a fundamentação teórica requerida para o desenvolvimento deste trabalho. Conceitos e explicações detalhadas sobre vídeo digital e padrões de compressão, em específico o padrão H.264/AVC, são oferecidos (Seções 2.1 a 2.10). Arquivos contêineres, responsáveis por armazenar *bitstreams* de vídeo, são também discutidos (Seção 2.11). Por fim, é apresentado um resumo sobre conceitos de informática e sobre os aplicativos e ferramentas utilizadas (Seção 2.12).

2.1 VÍDEO DIGITAL

Vídeo digital é a representação de uma cena visual real realizada por meio de amostragens espaciais e temporais. As imagens estáticas bidimensionais que compõem um vídeo digital progressivo são denominadas quadros, sendo compostos, cada um, por um conjunto de pixels com largura (W) e altura (H) fixos. O tamanho das imagens é, então, determinado por W x H. Cada pixel representa, por meio de um número fixo de bits, o brilho (luminância) e a cor (crominância) da amostra espacial. Quanto mais bits forem utilizados para representar um pixel, maior será a quantidade de cores e a gradação de brilho passíveis de representação, e consequentemente, maior será o tamanho final do arquivo [16].

A representação digital do brilho e da cor contida em um pixel é normalmente feita utilizando-se o modelo de representação de cor YCbCr (popularmente conhecido como YUV). Esse modelo não é absoluto, e sim uma forma mais eficiente de representação de informação RGB, um modelo aditivo que se utiliza das três cores primárias vermelha, verde e azul (*Red*, *Green*, *Blue*) para, unindo-as em diferentes proporções, reproduzir as todas as possibilidades de cor [16].

YCrCb define uma determinada cor através de um componente de luminância (Y), responsável pela determinação do brilho (basicamente a imagem em tons de cinza), e de dois componentes de crominância (Cb e Cr), responsáveis pela determinação da cor. Além de reduzir a redundância contida na representação RGB, ao condensar toda a informação de luminância em um só componente, o modelo YCrCb aproveita-se de uma característica da visão dos seres humanos: menor sensibilidade à cor do que ao brilho/luminância. As informações de brilho, portanto, têm mais impacto na percepção de qualidade de uma

imagem do que as de cor. Por essa razão, a maior parte dos padrões de vídeo reduzem a amostragem dos sinais Cb e Cr, visando a uma maior compressão do fluxo de vídeo (*bitstream*) [17].

Para que a reprodução de um vídeo transmita a impressão de movimento a quem o assiste, as imagens que o compõem são exibidas, em sequência, a uma velocidade que normalmente varia entre 24 e 30 quadros por segundo.

Assim, o tamanho total de um vídeo se dá pelo produto dos seguintes fatores:

▪ quantidade de pixels em uma imagem (W x H);

▪ quantidade de bits por pixel;

▪ número de imagens por segundo (*frame rate*);

▪ duração do vídeo (em segundos).

Um vídeo que tenha duração de 10 minutos (600 segundos), *frame rate* de 25 quadros por segundo, resolução de 1024x768 pixels e que utilize 24 bits para representar cada pixel, ocupará, então, um tamanho de aproximadamente 283 Gbytes:

(600) * (25) * (1024 * 768) * (24) = ~ 283 Gbytes.

De forma análoga, pode-se calcular que um DVD (*Digital Versatile Disk*) que atenda à recomendação ITU-R BT.601 (SDTV, 720x480/576 pixels) comporta somente três minutos de vídeo sem compressão.

Percebe-se, portanto, que a representação digital de vídeos demanda grande espaço de armazenamento, havendo, com algumas exceções, a necessidade de aplicação de técnicas de compressão para viabilizar seu uso.

2.2 QUADROS PROGRESSIVOS E ENTRELAÇADOS

Um quadro (*frame*) de vídeo pode se dividir em dois campos (*fields*). As linhas horizontais pares compõem um dos campos e as linhas horizontais ímpares compõem o outro. No modo entrelaçado (*interlaced*/*interleaved mode*), os dois campos são separados temporalmente por um período de campo (metade de um período de quadro). Podem,

ainda, ser codificados separadamente, como dois campos de imagens distintas, ou agruparem-se, para a formação de uma única imagem. Esse formato é normalmente utilizado com o intuito de deixar os movimentos perceptualmente mais fluidos, ao permitir a exibição do primeiro campo em metade do tempo que seria requerido para exibição do quadro.

No modo progressivo (*progressive mode*), não há separação temporal entre os dois campos, ou seja, os dois campos retratam o mesmo instante de tempo. Um quadro progressivo é codificado e decodificado como uma unidade indivisível, sem levar em consideração a separação em dois campos [16].

2.3 COMPRESSÃO DE VÍDEO DIGITAL

Os algoritmos de compressão atuam removendo redundâncias e/ou irrelevâncias do sinal de vídeo e podem ser divididos em duas grandes famílias: compressão sem perda de dados *(lossless data compression*) ou compressão com perda de dados (*lossy data compression*).

Pelos métodos de compressão sem perda de dados, o vídeo obtido após a compressão é idêntico ao vídeo original. No entanto, essa fidelidade é alcançada ao custo de taxas de compressão muito mais baixas do que as conseguidas por meio de métodos com perda de dados.

Já os métodos de compressão com perda de dados baseiam-se na premissa de que os vídeos apresentam redundâncias e/ou irrelevâncias nos domínios do tempo, do espaço e/ou das frequências. Assim, muitos dos dados que o compõem podem ser simplesmente descartados com pouco ou nenhum prejuízo para a qualidade perceptual final.

A compressão envolve um par de sistemas que normalmente recebe o nome de CODEC, composto por um compressor (*enCOder*) e um descompressor (*DECoder*).

O compressor de diversos padrões conhecidos e utilizados atualmente, como o MPEG-2, MPEG4-Visual ou o H.264/AVC, subdivide-se conceitualmente em três partes: compressor temporal, compressor espacial e codificador de entropia [16], conforme ilustrado na Figura 2.1.

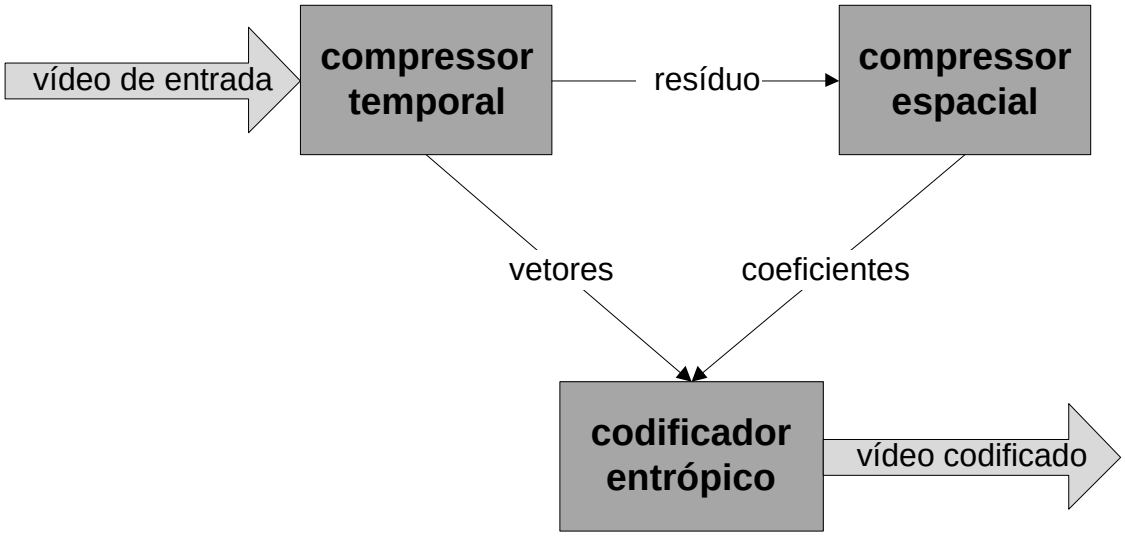

Figura 2.1: Subdivisão conceitual de um CODEC.

O compressor temporal, ou interquadro, recebe como entrada a sequência de quadros não comprimidos. Aproveitando-se das similaridades entre quadros vizinhos, esse compressor cria, a partir de quadros de referência, predições dos próximos quadros, sendo também calculada uma compensação de movimento entre a predição e os quadros base (predição com compensação de movimento). O resultado dessa compressão é um conjunto de parâmetros modelos (vetores de movimento), que descrevem como o movimento foi compensado, e quadros residuais (criados subtraindo-se a predição dos quadros base) [16].

O compressor espacial, também denominado compressor intraquadro, minimiza a redundância trabalhando a semelhança entre amostras próximas de um mesmo quadro. A taxa de compressão, nesses casos, é proporcional à quantidade de áreas homogêneas do quadro [18].

O procedimento de compressão envolve, ainda, a aplicação de uma transformada às amostras residuais e a quantização dos resultados. A transformada converte as amostras em outro domínio no qual elas são representadas por coeficientes. Os coeficientes são, então, quantizados para a remoção de valores "insignificantes", não fundamentais para a qualidade perceptual desejada. Os demais coeficientes, significantes, proveem uma representação mais compacta do quadro residual. O resultado desse passo é um conjunto de coeficientes quantizados [18].

Por fim, o codificador de entropia encerra a sequência de compressões. Esta etapa remove a redundância estatística dos dados e produz uma sequência final de bits

compactados que, em última instância, representam todos os dados: parâmetros de vetores de movimento codificados, coeficientes residuais codificados e informações/parâmetros de cabeçalho.

No processo inverso, de descompressão, o decodificador reconstrói os quadros de vídeo a partir da sequência de bits comprimidos (*bitstream*). Os vetores de movimento e os coeficientes são decodificados pelo decodificador entrópico e em seguida o decodificador espacial reconstrói os quadros residuais, conforme mostrado na Figura 2.2. O decodificador temporal utiliza os vetores de movimento e quadros anteriormente decodificados para recriar a predição dos quadros, aos quais se somam os quadros residuais [16].

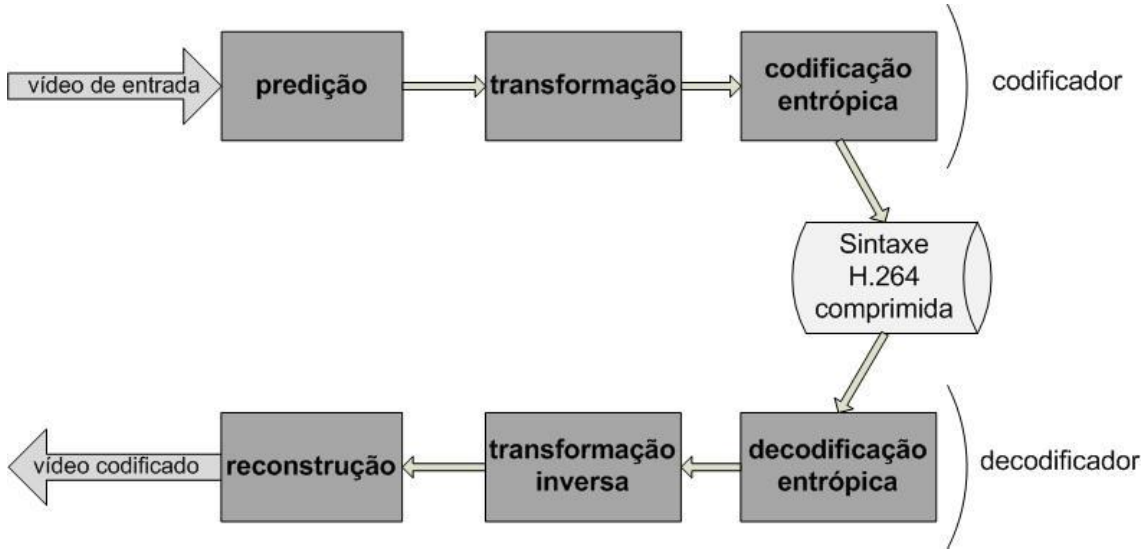

Figura 2.2: Diagrama de um codificador genérico.

2.4 PADRAO H.264/AVC

Para que equipamentos de vídeo digital de fabricantes distintos se comuniquem, é importante que a codificação de vídeo digital seja padronizada. Os aparelhos de DVD, por exemplo, utilizam o padrão MPEG-2 [19].

Há, atualmente, dois órgãos principais responsáveis por essa padronização: ITU-T *Video Coding Experts Group* (VCEG) [20] e ISO/IEC *Moving Picture Experts Group* (MPEG) [21]. H.264, também conhecido como MPEG-4 Parte 10 ou *Advanced Video Coding* (AVC), é um popular padrão de codificação de vídeo digital publicado

conjuntamente pelos dois citados órgãos, como *Recommendation* ITU-T H.264 [10] e ISO/IEC 14496-10 [9].

A especificação define uma sintaxe (formato) para o vídeo comprimido e o procedimento para decodificação dessa sintaxe, mas não estabelece como o vídeo digital deve ser codificado, deixando essa função a cargo dos desenvolvedores [22]. A Tabela 2.1 resume como se organiza o documento de especificação do padrão.

Tabela 2.1: Resumo dos capítulos da especificação do padrão H.264/AVC.

Capítulos/Anexos	Conteúdo
0 – Introdução	Descrição geral sobre o padrão.
1 – Escopo 2 – Referências 3 – Definições 4 – Abreviações 5 – Convenções	Delimita o documento e introduz as referências e terminologias utilizadas.
6 – Formatos de Dados e Relacionamentos	Define os formatos dos dados de entrada, codificados, decodificados e de saída, além dos relacionamentos entre as unidades codificadas.
7 – Sintaxe e Semântica	Define, por meio de uma série de tabelas, a sintaxe do fluxo de dados codificados e a semântica de cada um dos elementos sintáticos.
8 – Decodificação	Detalha os passos requeridos para a decodificação de um fluxo H.264/AVC.
9 – *Parsing*	Detalha os passos requeridos para a extração dos elementos sintáticos de um fluxo H.264/AVC.
A – Perfis e Níveis	Define subconjuntos de ferramentas de codificação e limites para a capacidade dos decodificadores.
B – Formato do Fluxo de Bytes	Especifica a sintaxe e a semântica de um fluxo de bytes NAL.
C – Decodificador de Referência Hipotético	Especifica um decodificador de referência hipotético e seu uso para determinar limites de desempenho.
D – Informação de Melhoria	Define formato de informação não essencial (controle

Suplementar (SEI)	de *buffer*, etc.) que pode ser acrescida ao fluxo H.264/AVC.
E – Informação de Usabilidade de Vídeo	Detalha informações não essenciais sobre a reprodução de vídeo.
G – Codificação de Vídeo Escalável	Especifica forma de codificação de vídeo escalável, para transmissões em múltiplas resoluções espaciais, taxas de quadros ou níveis de qualidade.

Uma sequência de vídeo codificado em H.264/AVC deve atender à sintaxe e à semântica descritas no Capítulo 7 da recomendação do padrão para ser compreendida pelos decodificadores. Os Capítulos 8 e 9, por sua vez, são igualmente fundamentais por estabelecerem os passos a serem seguidos para a reconstrução da sequência de vídeo reproduzível, através da correta decodificação do *bitstream* e da extração dos elementos semânticos.

O padrão, um documento com 676 páginas (*Recommendation* ITU-T H.264) [10], foi publicado pela primeira vez em 2003, sendo seguido desde então por diversas revisões, ampliações e aprimoramentos, tendo a última delas sido lançada em 2010. Os conceitos utilizados nos padrões MPEG-2 e MPEG-4 Visual, seus predecessores, foram tomados como ponto de partida [23]. O resultado, no entanto, é um padrão com eficiência de compressão bastante superior. Em comparação com esses dois outros padrões, o H.264/AVC é capaz de fornecer uma melhor qualidade de imagem com o mesmo *bitrate* (taxa de transferência de bits) ou um *bitrate* menor para a mesma qualidade de imagem [13].Um exemplo dessa afirmação pode ser visto na Figura 2.3. Um DVD de camada única, por exemplo, é capaz de armazenar duas horas de vídeo em formato MPEG-2, ao passo que, se o codificador utilizado for o H.264/AVC, o mesmo disco seria capaz de armazenar quatro horas ou mais de vídeo com a mesma qualidade perceptual.

Figura 2.3: Compressão nos padrões MPEG-2, MPEG-4 Visual e H.264/AVC, com *bitrate* fixo de 150kbps [24].

O padrão H.264/AVC, com seu ganho de desempenho, tornou-se o sucessor dos padrões MPEG-1, MPEG-2 e H.263 em inúmeros equipamentos, como aparelhos de telefonia celular, filmadoras digitais, reprodutores de filmes e sistemas de videossegurança, entre outros [12] [25] [26].

2.5 PERFIS E NÍVEIS

Diversos são os algoritmos e os processos suportados pelo padrão H.264/AVC. Alguns são essenciais, como as transformadas 4x4, ao passo que outras são opcionais e nem mesmo diretamente relacionadas ao processo de codificação de vídeo, como as mensagens SEI (*Supplemental Enhancement Information*) e VUI (*Video Usability Information*). Um codificador H.264/AVC pode escolher entre diversas ferramentas, essenciais ou não, para proceder à codificação de um fluxo de vídeo [10].

O decodificador, no entanto, pode não ser capaz de processar um fluxo H.264/AVC que utilize processos ou algoritmos por ele não implementados. Os perfis H.264/AVC, apresentados no Anexo A da especificação e ilustrados na Figura 2.4, são os responsáveis por padronizar quão amplo tem de ser o suporte ao padrão. Assim, um decodificador que atenda a um perfil determinado só será capaz de decodificar vídeos produzidos com processos e algoritmos estipulados por esse perfil. A intenção dos perfis é estabelecer limites operacionais às ferramentas de codificação e às capacidades computacional e de armazenamento requeridas para a decodificação de uma sequência. Isso é feito determinando quais ferramentas de decodificação o decodificador deve ser capaz de utilizar. Assim, com a correta determinação do perfil, o padrão de compressão

H.264/AVC pode ser eficientemente utilizado tanto em limitados *smartphones* quanto em decodificadores de alta resolução, como tocadores Blu-Ray [24].

Os perfis *baseline* (*baseline profile*) e *baseline* restrito (*constrained baseline profile)* são normalmente utilizados em aplicações de baixa complexidade, como transmissões de vídeo por aparelhos de telefonia celular. O perfil estendido (*extended profile*) adiciona suporte a novas ferramentas, em comparação com o perfil *baseline*, sendo usualmente recomendado para *streamings*. O perfil principal (*main profile*) estende as ferramentas suportadas pelo perfil *baseline* restrito, e é destinado a aplicações de entretenimento, como TV digital e reprodução de filmes [10] (Ver Figura 2.4).

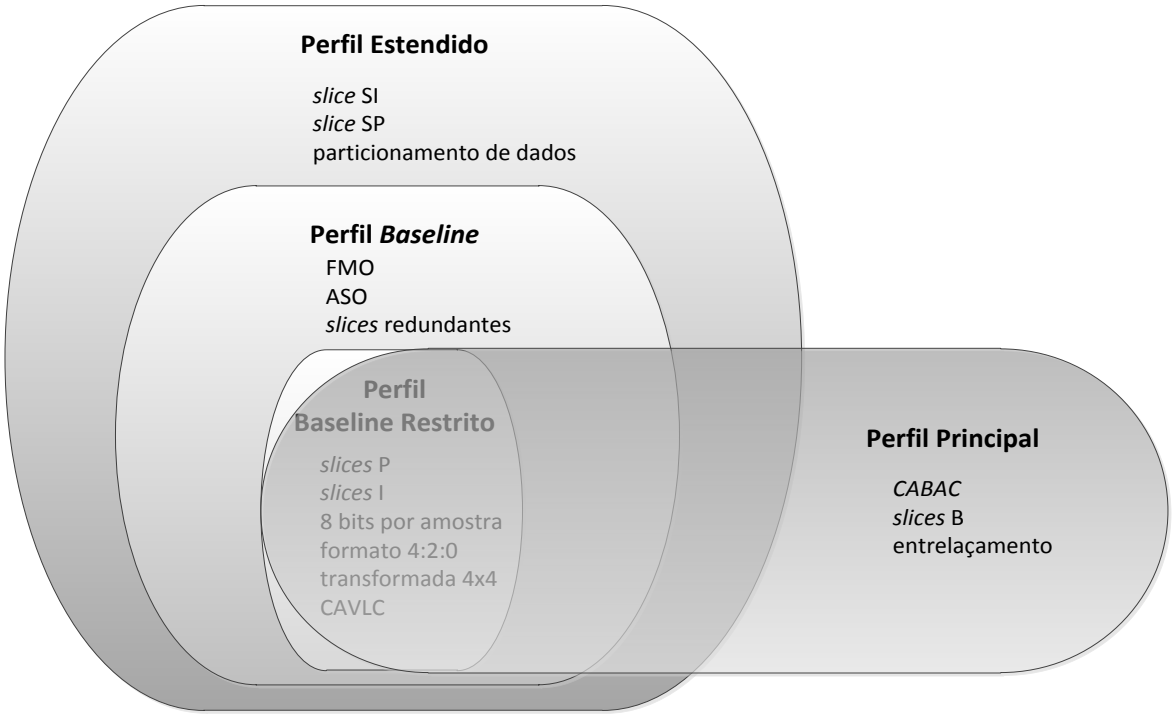

Figura 2.4: Perfis *Baseline*, *Baseline* Restrito, Estendido e Principal.

Além dos perfis, os níveis são também responsáveis por impor limites à codificação. A diferença é que, neste caso, as restrições são impostas aos valores dos elementos de sintaxe do *bitstream* H.264/AVC. Os níveis são numerados de 1 a 5, com subníveis de uma casa decimal. Um decodificador que seja compatível com o nível 2.1, por exemplo, suportará, além dos fluxos de vídeo com nível 2.1, aqueles com níveis inferiores, como 2 ou 1.2 [24].

Há, entre os diversos parâmetros vinculados aos níveis, restrições no decodificador quanto i) ao número máximo de macroblocos capaz de ser processado por segundo; ii) ao número máximo de macroblocos em um quadro decodificado; iii) ao espaço máximo de memória requerido para armazenamento dos quadros de referência; iv) ao máximo *bitrate* suportado; e v) ao espaço máximo de memória requerido para armazenamento dos dados antes da decodificação. A Tabela 2.2 ilustra algumas das limitações impostas a determinados níveis, quanto à resolução luma e o número máximo de frames por segundo.

Tabela 2.2: Limitações de determinados níveis.

Nível	Resolução luma	N.º máx. de quadros por segundo
1	176x144 (QCIF)	15
1.1	176x144 (QCIF)	30
1.3, 2	352x288 (CIF)	30
3	720x480 (525 SD)	30
3	720x576 (625 SD)	25
4, 4.1	1920x1080 (1080p HD)	30
4.2	1920x1080 (1080p HD)	60
5.1	4096x2048 (4Kx2K)	30

A escolha de um perfil e de um nível determinará o pico de poder computacional e de uso de memória que serão exigidos do decodificador. A conformidade a um determinado conjunto perfil/nível é verificada por meio de um modelo teórico denominado "Decodificador de Referência Hipotético" (*Hypothetical Reference Decoder*), e um decodificador H.264/AVC pode, analisando certos parâmetros do *bitstream*, determinar se é capaz de decodificá-lo [24], conforme ilustrado na Figura 2.5.

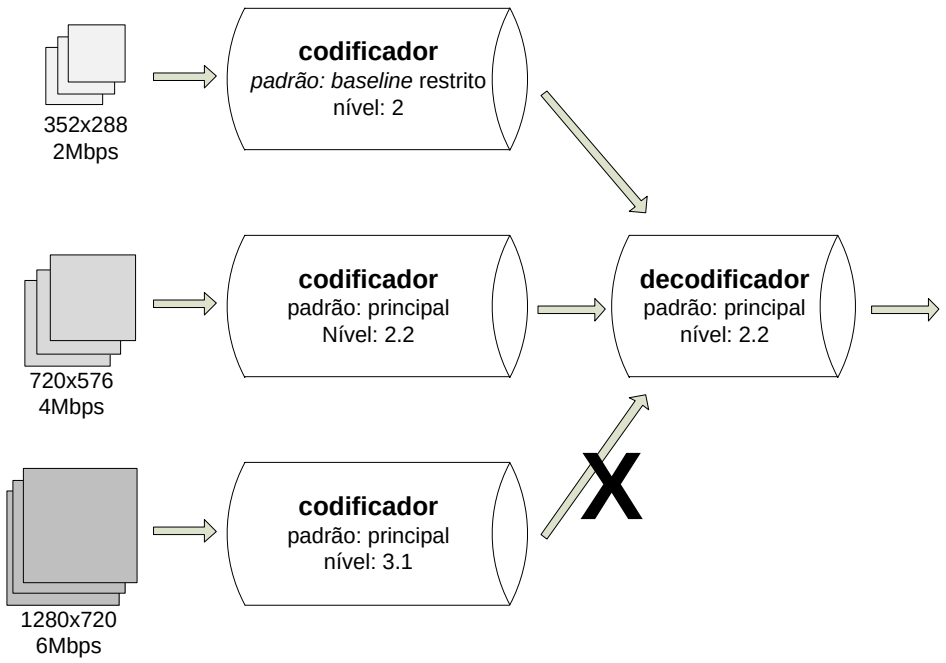

Figura 2.5: Checagem de conformação a determinados conjuntos de perfis/níveis.

2.6 SINTAXE DE UM BITREAM H.264/AVC

O Capítulo 7 da especificação do padrão H.264/AVC estabelece a sintaxe requerida de um *bitstream* H.264/AVC, ou seja, o formato e a ordem que os dados que compõem o fluxo devem seguir. Esses dados incluem não só os quadros codificados em si, mas também os diversos parâmetros de controle. O decodificador analisa essa sintaxe e, extraindo os parâmetros e os elementos de dados, decodifica o vídeo [10].

A estrutura determinada pelo padrão para a correta decodificação do fluxo é hierárquica. No mais alto nível, há a sequência de vídeo em si, e dela depreendem-se, em níveis abaixo, as unidades de acesso (*access units*), representando quadros codificados, e seus subconjuntos, *slice*s, até o último nível, onde se encontram os macroblocos e os blocos [27]. Ver Figura 2.6.

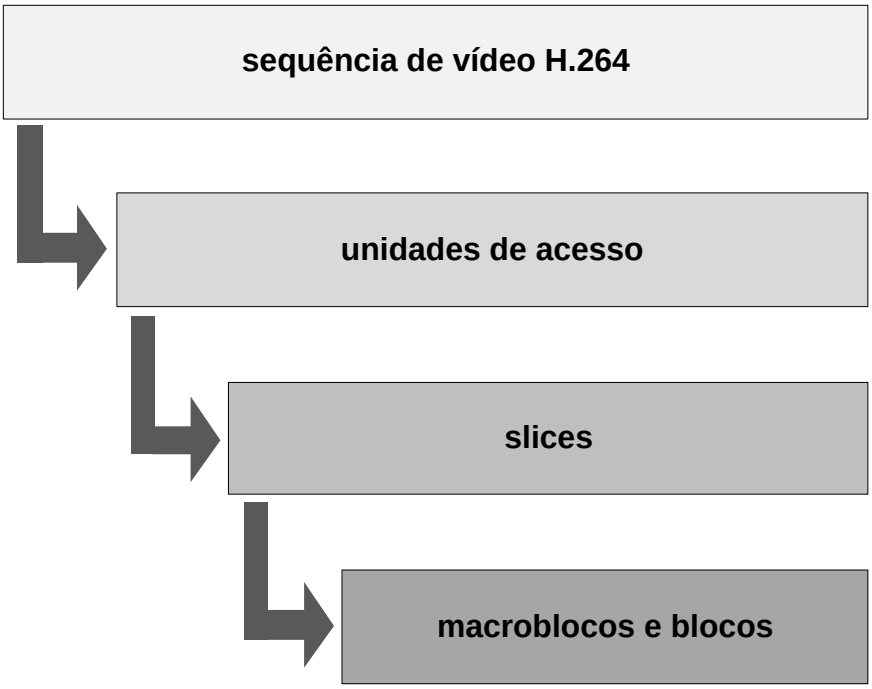

Figura 2.6: Estrutura hierárquica de um bitstream H.264/AVC.

A Camada de Codificação de Vídeo (*Video Coding Layer*, VCL) é especificada pelo padrão para representar, de forma eficiente, os dados do vídeo. Os requerimentos de flexibilidade e customização exigidos por vários aplicativos são atendidos por meio dessa camada. A Camada de Abstração de Rede (*Network Abstraction Layer*, NAL), por sua vez, é especificada para formatar os dados e prover informações de cabeçalho de uma maneira apropriada para transmissão ou armazenamento. Tanto as *slices* quanto as *access units* específicas de controle são denominadas NAL *units*. Há, no entanto, duas subdivisões: VCL e não-VCL NAL *units*. As VCL NAL *units* contêm a representação dos valores das amostras dos quadros, ao passo que as não-VCL NAL *units* contêm as demais informações, tais como conjuntos de parâmetros (dados de cabeçalho que podem se aplicar a diversas VCL NAL *units*) e informações suplementares não obrigatórias, as quais, se presentes, são usadas para aprimorar a usabilidade do sinal de vídeo decodificado [10].

As NAL *units*, incluindo as VCL NAL *units*, possuem uma sequência de bytes que descrevem os elementos de sintaxe. Essa sequência, que recebe o nome de *Raw Byte Sequence Payload* (RBSP), tem tamanho variável, e pode ser sucedida por bits de finalização (RBSP *trailing bits*) para forçar que tenha um tamanho inteiro de bytes. Cada elemento de sintaxe pode ser subdividido em subelementos. Por exemplo, o elemento de sintaxe *Picture Parameter Set* pode ou não conter o subelemento *Scaling List*, mas sempre

conterá o elemento *rbsp_trailing_bits*. Os elementos e subelementos são também denominados seções e subseções [10].

Cada quadro de uma sequência de vídeo é particionado em macroblocos de tamanho fixo que agrupam uma área quadrangular da imagem de 16x16 pixels, contendo 16x16 amostras do componente luma e 8x8 amostras de cada um dos dois componentes de crominância (Cr e Cb), no formato 4:2:0. Todas essas amostras de luminância e crominância são preditas, espacial e/ou temporalmente [28].

Os macroblocos são organizados em *slices*, que representam subconjuntos de um determinado quadro que podem ser decodificados independentemente. A ordem de transmissão dos macroblocos no *bitstream* depende de um mapa denominado *Macroblock Allocation Map* e não é necessariamente *raster-scan*. H.264/AVC suporta cinco tipos diferentes de *slices*. A primeira e mais simples é a I-*slice* (Intra *slice*). Nesse caso, todos os macroblocos são codificados sem referenciarem outros quadros da sequência de vídeo. P-*slice*s (predictive *slice*s) e B-*slice*s (*bi-predictiveslice*s), por outro lado, utilizam-se de quadros previamente codificados para formar predições de macroblocos. SP-*slice*s (*switching* P-*slice*s) e SI-*slice*s (switching I-*slice*s) são especificadas para permitir alternância entre *bitstreams* codificados em taxas de bits variáveis [18].

2.7 ETAPAS DE PREDIÇÃO

Para criar a predição de um determinado macrobloco, o codificador se baseia em macroblocos previamente codificados. Uma vez criado o macrobloco predito, o codificador forma um macrobloco residual subtraindo o macrobloco predito do macrobloco original (macrobloco residual = macrobloco original – macrobloco predito), conforme ilustrado pela Figura 2.7. Uma predição bem calculada minimiza a variância dos dados armazenados em macroblocos residuais, o que se traduz, em última instância, em um *bitstream* com compressão otimizada. A predição é uma etapa *lossless*, ou seja, é um procedimento completamente reversível, sem qualquer perda de dados [29].

Figura 2.7: Macrobloco residual = Macrobloco original – Macrobloco predito.

O primeiro quadro de uma sequência ou os pontos de acesso randômico são tipicamente "Intra" codificados, ou seja, utilizam somente informações constantes do próprio quadro para a codificação. Em quadros "Intra", cada bloco é predito a partir de blocos vizinhos já codificados. Os quadros que não o primeiro e os pontos de acesso randômico, por sua vez, são usualmente "Inter" codificados. Nesse tipo de predição, o codificador busca um bloco similar em algum quadro já codificado, chamado de quadro de referência, e, por meio de um processo denominado estimação de movimento, calcula um vetor de movimento que aponta para o bloco referenciado [18].

A diferença entre os blocos originais e os blocos preditos formam os blocos residuais da predição, tanto "Inter" quanto "Intra" (ver Figura 2.8). Esses blocos residuais são transformados e quantizados. Os coeficientes quantizados da transformada são entropicamente codificados e transmitidos ou armazenados juntamente com as informações acessórias das predições intraquadros ou interquadros [18].

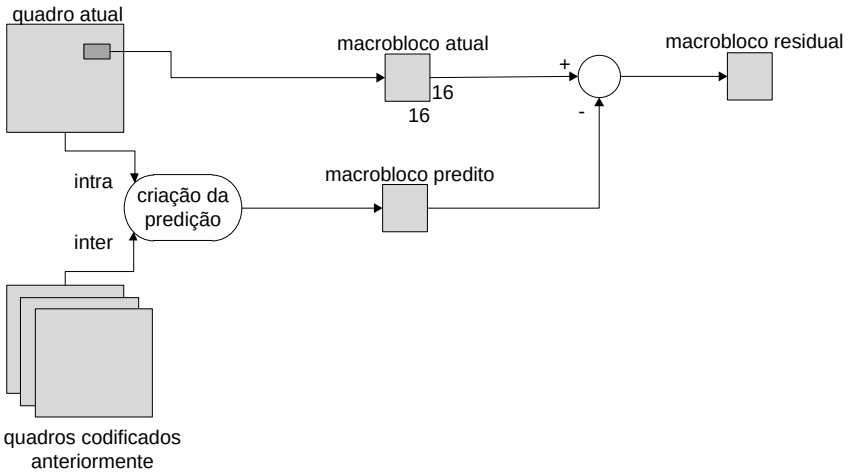

Figura 2.8: Processo de predição interquadros e intraquadros.

2.7.1 Predição Intraquadros

A predição intraquadro prediz macroblocos a partir de outros já codificados e pertencentes ao mesmo quadro. O primeiro quadro de uma sequência de vídeo ou os quadros classificados como quadros de acesso randômico (*random access points*) utilizam-se somente deste tipo de predição. Essa forma se mostra eficiente em muitas situações porque há uma grande chance de que exista uma alta correlação entre as amostras luma e chroma de blocos adjacentes. Qualquer tipo de *slice* pode conter macroblocos Intra, com a restrição de que, se uma *slice* é Intra, todos os seus macroblocos devem ser Intra [16].

De todos os tipos de codificação de *slice*s existentes no padrão H.264/AVC, há quatro opções relacionadas à codificação intraquadro: CHROMA, INTRA-4x4, INTRA-8x8 e INTRA-16x16. Em contraste com padrões de codificação de vídeo anteriores, que conduziam a predição no domínio das transformadas, a predição no padrão H.264/AVC é sempre conduzida no domínio do espaço, referindo-se a amostras vizinhas de blocos já codificados. Com vistas a manter todas as *slice*s independentes entre si, não são permitidas predições intraquadros que avancem os limites da *slice* [24].

O tamanho de bloco CHROMA gera um bloco de predição para cada par de componentes chroma, usando blocos de quatro tamanhos possíveis. No uso do tamanho de bloco luma 4x4 (INTRA-4x4), cada bloco utiliza um modo de predição, dentre nove possíveis (uma predição DC e oito modos de predições direcionais). Para cada bloco 4x4 de amostras luma é gerado um bloco predito 4x4. O tamanho de bloco luma 8x8 (INTRA-8x8) também suporta nove modos de predição, e, nesse caso, para cada bloco luma 8x8 é gerado um bloco predito 8x8. Com tamanho de bloco luma 16x16 (INTRA-16x16), indicado para áreas com imagens muito semelhantes ou áreas de transições graduais, uma predição uniforme é aplicada para todo o componente de luminância do macrobloco. São suportados, nesse caso, quatro modos de predição, e é gerado um único bloco predito de tamanho 16x16. As amostras de crominância de um macrobloco são preditas usando uma técnica de predição similar à do componente de luminância em macroblocos INTRA-16x16 [24].

A escolha de qual tamanho de bloco é mais conveniente exige uma ponderação sobre dois fatores geralmente antagônicos: eficiência de cálculo da predição x custo de sinalização do modo de predição escolhido. Blocos de tamanho 16x16 normalmente produzem mais dados residuais em consequência de predições menos precisas. Como

vantagem, menos bits são necessários para codificar a predição em si. No outro extremo, blocos de tamanho 4x4 precisam individualmente sinalizar ao decodificador o modo de predição utilizado, requerendo, assim, um maior número de bits. A vantagem é a maior acuidade da predição e o consequente menor tamanho dos dados residuais. Isso porque, com tamanho menor de bloco, há uma maior chance de os blocos atuais e preditos serem muito aproximados [24].

Conjuntamente à decisão de qual o melhor tamanho de bloco a ser utilizado, o codificador deve selecionar o modo de predição Intra a ser utilizado. Os modos têm o propósito de minimizar o número de bits dos blocos preditos e residuais.

As três tabelas abaixo resumem os modos compatíveis com os tamanhos de bloco luma 4x4/8x8, 16x16 e CHROMA, nessa ordem.

Tabela 2.3: Codificação de *slices* INTRA-4x4 e INTRA-8x8.

Modos compatíveis com tamanhos de bloco luma 4x4 e 8x8[1]	Descrição
0 (Vertical)	Extrapolação de amostras acima
1 (Horizontal)	Extrapolação de amostras à esquerda
2 (DC)	Média de amostras acima e à esquerda
3 (Diagonal Down-Left)	Interpolação a um ângulo de 45º entre amostras abaixo e à esquerda e acima e à direita
4 (Diagonal Down-Right)	Extrapolação a um ângulo de 45º entre amostras abaixo e à direita
5 (Vertical Left)	Extrapolação a um ângulo de 26.6º à esquerda da vertical (largura/altura = ½)
6 (Horizontal-Down)	Extrapolação a um ângulo de 26.6º à abaixo da horizontal
7 (Vertical-Right)	Extrapolação ou interpolação a um ângulo de 26.6º à direita da vertical
8 (Horizontal-Up)	Interpolação a um ângulo de 26.6º acima da horizontal

Tabela 2.4: Codificação de *slices* INTRA-16x16.

[1] Predições Intra-bloco com tamanho de bloco 8x8 só são utilizados nos perfis *High*.

Modos compatíveis com tamanho de bloco luma 16x16	Descrição
0 (Vertical)	Extrapolação de amostras acima
1 (Horizontal)	Extrapolação de amostras à esquerda
2 (DC)	Média de amostras acima e à esquerda
3 (Plane)	Função plana linear envolvendo as amostras acima e à esquerda. (Apropriada para áreas com variações sutis de luminância.)

Tabela 2.5: Codificação de *slices* CHROMA.

Modos compatíveis com tamanho de bloco luma CHROMA	Descrição
0 (DC)[2]	Média de amostras acima e à esquerda
1 (Horizontal)	Extrapolação de amostras à esquerda
2 (Vertical)	Extrapolação de amostras acima
3 (Plane)	Função plana linear envolvendo as amostras acima e à esquerda. (Apropriada para áreas com variações sutis de luminância.)

A forma de cálculo do modo DC modifica-se dependendo de quais amostras estão disponíveis para o cálculo. Os demais modos listados nas tabelas, apesar de suportados, só podem ser usados se as amostras requeridas para a predição estiverem disponíveis [24].

As escolhas de modo de predição Intra para cada bloco devem ser sinalizadas para o decodificador. Para predições com tamanho de bloco 16x16 ou CHROMA, o modo de predição é sinalizado na própria sintaxe do macrobloco. No caso de blocos luma de tamanhos 4x4 e 8x8, contudo, o padrão H.264/AVC utiliza-se de um estilo de codificação preditiva para evitar que seja necessário um grande número de bits para esse fim. Essa solução diminui o número de bits necessários para a sinalização justamente porque blocos vizinhos tendem a ser bastante correlacionados. Assim, se um determinado bloco foi mais

[2] Neste caso os quatro modos de predição são os mesmos dos modos compatíveis com o tamanho de bloco luma 16x16, havendo mudança somente na numeração dos modos.

bem predito usando o modo 1, é grande a probabilidade de que o próximo bloco também seja mais bem predito por esse modo [24].

2.7.2 Predição Interquadros

Tipicamente, predições interquadros são utilizadas em todos os quadros de uma sequência de vídeo que não o primeiro quadro, denominado um *random access point*. O padrão H.264/AVC especifica vários tipos de codificação por compensação de movimento para P-*slice*s. Cada macrobloco do tipo P, com tamanho de 16x16 pixels, é particionado em blocos de tamanho fixo usados para a descrição do movimento. São suportados particionamentos com tamanhos de bloco luma de 16x16, 16x8, 8x16 e 8x8, com as respectivas amostras chroma associadas, os quais correspondem, respectivamente, a tipos de macrobloco P Inter-16x16, Inter-16x8, Inter-8x16 e Inter-8x8. Assim, há quatro distribuições possíveis: uma partição 16x16, duas partições 8x16, duas partições 16x8 ou quatro partições 8x8. O particionamento em 16x16 (Inter-16x16) cobre todo o macrobloco, por ter exatamente o seu tamanho (ver Figura 2.9) [24].

Quando um macrobloco é particionado em blocos de amostras luma de tamanho 8x8 (com as amostras chroma associadas), são criados quatro submacroblocos. Cada um desses submacroblocos subdivide-se em uma partição de submacrobloco, com tamanho 8x8 (igual ao do submacrobloco), em duas partições com tamanhos 4x8 ou 8x4, ou em quatro partições com tamanho 4x4. Nesse caso, um elemento extra de sintaxe tem de ser transmitido para cada um das quatro partições de submacroblocos. A função desse elemento é especificar se cada partição de submacrobloco está codificada usando predição por compensação de movimento com blocos luma de tamanho 8x8, 8x4, 4x8 ou 4x4 [24].

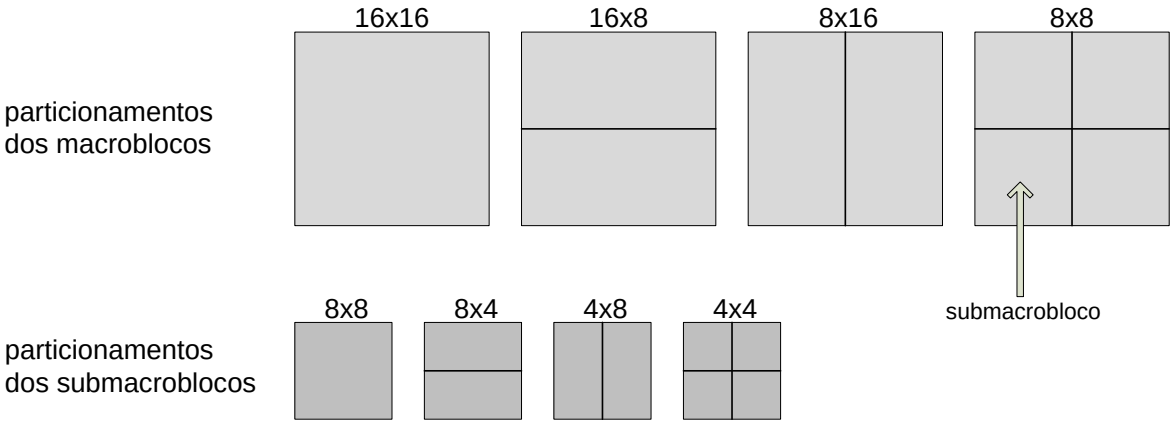

Figura 2.9: Particionamento de macroblocos e submacroblocos.

Uma partição de macrobloco ou de submacrobloco de uma *slice* P possui um vetor de movimento que aponta para uma área de mesmo tamanho em um quadro de referência. Sendo uma *slice* B, no entanto, a partição pode ter um ou dois quadros de referência e, consequentemente, um ou dois vetores de movimento. O(s) vetor(es) de movimento devem apontar para uma área de mesmo tamanho da partição sendo predita. A seleção de quais quadros de referência serão usados é feita no nível das partições do macrobloco. Diferentes partições de macrobloco podem referenciar quadros distintos para gerarem suas predições. As partições de submacrobloco, no entanto, devem referenciar sempre o(s) mesmo(s) quadro(s) [24].

Como exemplo, se um macrobloco for codificado usando-se o tipo Inter-8x8, e, por sua vez, cada partição de submacrobloco for codificado usando-se o tipo Inter-4x4, dezesseis vetores de movimento seriam transmitidos para um único macrobloco P-*slice*. Um macrobloco luma de 16x16 seria, então, particionado em 16 blocos de tamanho 4x4.

O esquema de subparticionamento descrito exige um maior número de bits para a descrição dos vetores de movimento e para os dados extras do tipo de partição. Em compensação, tendo sido feita a escolha correta de particionamento, reduzem-se os dados residuais a serem transmitidos. Assim, a escolha do tipo de particionamento depende diretamente das características do vídeo de entrada. De forma geral, tamanhos maiores de sub-blocos são indicados para áreas homogêneas de um quadro, ao passo que tamanhos menores normalmente se mostram mais adequados para áreas com mais detalhes. Alguns formatos antecessores, como o H.263 e o MPEG-4 Visual, suportam apenas blocos de dois tamanhos, 16x16 e 8x8 [18].

O sinal de predição para cada bloco luma MxN é obtido deslocando-se uma área do quadro de referência correspondente. Essa área é especificada por um vetor de movimento translacional e um índice de referência de quadro. Esse vetor de movimento pode apontar para posições que sejam amostras inteiras, metade de amostras ou mesmo 1/4 de amostras dos componentes luma de um quadro de referência. As amostras fracionadas (1/2 e 1/4) são calculadas pela interpolação de amostras do quadro de referência. Cada vetor de movimento é diferencialmente codificado a partir de vetores de movimento de quadros vizinhos [24].

Essa citada interpolação dos quadros de referência se dá da forma a seguir descrita. Cada partição de um macrobloco intercodificado é predito a partir de uma área do mesmo tamanho pertencente ao quadro de referência. O vetor de movimento (distância relativa entre as duas áreas) tem resolução de 1/4 de pixel, para o componente luma, e 1/8 de pixel, para os componentes chroma, no formato 4:2:0. As amostras luma e chroma em posições subpixel, por não existirem no quadro de referência, são criadas por meio de interpolação de regiões vizinhas do quadro de referência. Se os componentes vertical e horizontal do vetor de movimento são inteiros, as amostras relevantes no bloco de referência existem, e podem ser diretamente referenciadas. Se um ou ambos os componentes do vetor de movimento têm valores fracionados, as amostras preditas são geradas por meio de interpolação entre amostras adjacentes no quadro de referência [24].

A sintaxe do padrão H.264/AVC geralmente permite vetores de movimento irrestritos, ou seja, permite que os vetores de movimento apontem para fora da área de imagem. Quando isso ocorre, o quadro de referência é estendido para além dos limites da imagem. Isso é feito repetindo-se os pixels da fronteira da imagem antes da interpolação. Uma ressalva é que predições por compensação de movimento não podem extrapolar os limites da *slice* [24].

Os vetores de movimento são calculados e transmitidos para cada bloco, e têm a função de informar qual área de um de um determinado quadro já codificado foi usada para a predição. Em padrões MPEG anteriores, o quadro de referência é o quadro prévio mais recente. O padrão H.264/AVC, no entanto, abre a possibilidade de referências a múltiplos quadros. Assim, um bloco pode ser predito a partir de uma única região de um quadro de referência, no caso de macroblocos P ou B, ou a partir de duas regiões pertencentes a dois quadros de referência, no caso de macroblocos B (bipredição) [24].

Outra forma de predição, denominada predição “com peso” (*weighted prediction*), é usada em macroblocos pertencentes a *slice*s P ou B. A ideia é atribuir um fator (peso) a cada amostra antes de predição a ser calculada. O cálculo do fator se dá de duas formas distintas: explícito ou implícito. No modo explícito, suportado tanto por *slice*s P quanto por *slices* B, o fator é calculado pelo codificador e transmitido no cabeçalho da *slice*. No modo implícito, suportado somente por *slice*s B, o fator é calculado com base nas posições temporais dos quadros de referência. Assim, um fator mais pesado pode ser aplicado se o quadro de referência está temporalmente próximo do quadro atual. Quadros temporalmente mais distantes recebem, dessa forma, fatores mais leves. Ambas as formas, explícita e implícita, têm a função de permitir controle sobre as contribuições relativas dos quadros de referência. Um uso típico seria na transição entre duas cenas com efeito de *fading* [30].

A transmissão de um vetor de movimento para cada partição de macrobloco ou de submacrobloco tem um impacto negativo no tamanho do *bitstream*. Quanto mais particionados forem os macroblocos, maior será o custo em bits. Ocorre que, com grande frequência, partições vizinhas têm vetores de movimento bastante correlacionados. Utilizando-se dessa correlação, os vetores de movimento podem ser preditos a partir de outros vetores já calculados para partições vizinhas. O que é codificado e transmitido, nesse caso, é a diferença entre o vetor atual (que deve ser calculado) e o vetor predito. Esse tipo de predição de vetor de movimento, para ser utilizado, depende da disponibilidade de vetores de movimento próximos e do tamanho das partições [18].

Macroblocos P-*slice* (macroblocos do tipo P) podem ser codificados em um modo conhecido como *skip*. Um macrobloco é marcado como *skipped* se i) contiver somente coeficientes da transformada quantizados que tenham valor zero e ii) possuir características que permitam que seu movimento seja predito a partir do movimento de macroblocos vizinhos (predição espacial). O ganho de eficiência se dá porque se retira a necessidade de transmissão dos vetores de movimento, dos parâmetros do índice de referência e do sinal quantizado do erro de predição. Uma vantagem desse modo em relação a modos similares de padrões antecessores é o fato de o vetor de movimento usado para a reconstrução do macrobloco *skipped* ser inferido das propriedades de movimento de macroblocos vizinhos, ao invés de ser deduzido como zero. A decodificação se dá de maneira similar à de um macrobloco predito de tamanho 16x16 que referencie o quadro, localizado na posição zero do índice do buffer [24].

Em macroblocos do tipo B-*slice*, as predições temporais podem ser calculadas através de dois vetores de movimento representando duas estimativas do movimento por partição de macrobloco ou de submacrobloco. São utilizados, nesse caso, dois buffers de quadros de referência, gerenciados pelo controle de buffer. Em uma inovação em relação a padrões anteriores, no H.264/AVC essas predições podem se dar a partir de quadros de referência tanto anteriores quanto posteriores na ordem de exibição. A predição, nesse caso, é calculada através de uma média ponderada dos valores dos pixels dos quadros de referência. Assim como ocorre com as P-*slice*s, há um limite máximo no número de quadros utilizáveis para essa estimativa de movimento, especificado para cada nível (*level*), e as partições de submacrobloco devem referenciar os mesmos quadros [24].

2.7.3 Passos da codificação de macrobloco interpredito

A codificação de um macrobloco interpredito segue os seguintes passos (não necessariamente na ordem apresentada) [18]:

- Interpolação dos quadros no DPB (*Decoded Picture Buffer*) para geração de posições com 1/4 do tamanho da amostra do componente luma e 1/8 do tamanho das amostras dos componentes chroma;
- Escolha das imagens de referência, entre as constantes do DPB;
- Escolha dos tamanhos das partições de macroblocos e de submacroblocos;
- Escolha do tipo de predição entre as opções abaixo:
  - predição a partir de um quadro de referência contido na lista 0, para macroblocos P ou B, ou nas lista 1, para macroblocos B;
  - bipredição a partir de dois quadros de referência, uma na lista 0 e outro na lista 1, para macroblocos B, opcionalmente com uso de predição ponderada;
- Escolha do vetor de movimento para cada partição de macrobloco ou de submacrobloco (se forem utilizados dois quadros de referência, serão gerados dois vetores de movimento);
- Predição dos vetores de movimento a partir de vetores previamente transmitidos (vetor predito = vetor atual – vetor previamente transmitido);

▪ Codificação das informações: tipo de macrobloco, escolha das referências de predição, partições residuais e vetores de movimento (preditos ou não);

▪ Aplicação de um filtro *deblocking* antes da disponibilização do quadro reconstruído ao buffer de predição.

2.8 LISTAS DE QUADROS DE REFERÊNCIA

*Slice*s, ao serem recebidas, são decodificadas para a recriação dos quadros, que são,só então, reproduzidos/apresentados. Esses quadros decodificados são também armazenados em um buffer denominado *Decoded Picture Buffer* (DPB), para possível futuro uso como quadros de referência. Também o codificador precisa armazenar os quadros de referência, decodificados, em um buffer similar, denominado *Coded Picture Buffer* (CPB), para que sejam usados durante a codificação de quadros futuros [24].

A armazenagem dos quadros de referência deve seguir as especificações de tipo de buffer e de operações de controle de gerenciamento de memória (MMCO) contidas no *bitstream*. Sendo o tamanho do buffer configurado para mais de um quadro, deve-se sinalizar o índice para localização do quadro de referência no buffer. Assim, um parâmetro de índice adicional deve ser transmitido para cada um dos vetores de movimento das partições de macroblocos de tamanhos 16x16, 16x8, 8x16, ou 8x8 [24].

Os quadros de referência podem ser marcados como de curto ou de longo prazo. Essas marcações são alteráveis e ambos os tipos de quadro podem ser removidos do DPB por meio de um comando explícito no *bitstream*. A diferença entre eles é que o de curto prazo, a depender de configuração, pode também ser removido de forma automática quando o DPB estiver cheio. Para a codificação ou decodificação de uma *slice*, os frames de referência são ordenados em uma lista (lista 0), no caso das *slice*s P, ou em duas listas (lista 0 e lista 1), no caso das *slice*s B [24].

Nas listas, são elencados primeiros os frames de referência de curto prazo e em seguida os de longo prazo. Os frames de longo prazo ordenam-se por meio de um índice denominado *LongTermFrameIdx*. Os de curto prazo ordenam-se por meio de um índice denominado *frame_num* ou POC. Por default, o primeiro quadro na lista 0 é o quadro mais recentemente decodificado, em caso de *slice* P. Em caso de *slice* B, por default o primeiro

quadro nas listas 0 e 1 é, respectivamente, o quadro anterior e o posterior ao atual, em ordem de exibição. Esse ordenamento se dá de forma que frames temporalmente mais próximos do frame corrente apareçam primeiro na lista. Como é provável que possuam a melhor predição, e consequentemente sejam usados, serão necessários menos bits para referencia-los [24].

2.9 TRANSFORMAÇÃO E QUANTIZAÇÃO

A etapa de transformação é responsável por converter blocos de amostras de imagens em um domínio diferente, criando coeficientes de transformada. A etapa de quantização, por sua vez, reduz a precisão desses coeficientes de transformada e, consequentemente, diminui o espaço por ele ocupado. Isso é alcançado dividindo-os por um valor inteiro, com o resultado arredondado para o inteiro mais próximo. Um parâmetro de quantização (QP) é utilizado, de forma que quanto maior esse parâmetro, maior será a compressão alcançada, já que uma maior parte dos coeficientes assumirá valor zero, e haverá menos coeficientes com valores diferentes de zero. Esse parâmetro de quantização indexa um total de 52 valores de tamanho de passo de quantização (*quantizer step size*) suportados pelo padrão. A operação básica da etapa de quantização, como pode ser visto na Figura 2.10, é a seguinte: Coeficiente quantizado = *round*(coeficiente da transformada / Qstep) [18].

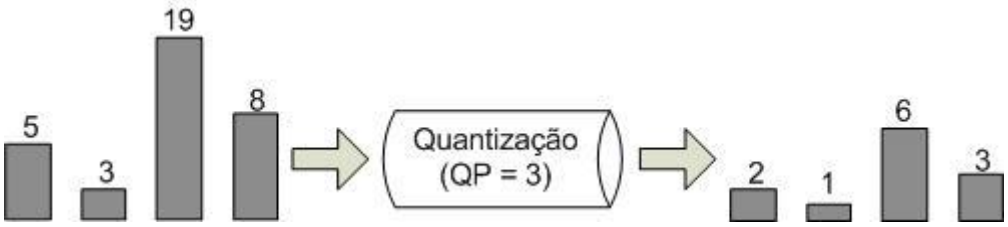

Figura 2.10: Etapa de Quantização.

É no passo de quantização, e somente neste, que o processo de compressão envolve perda de dados (*lossy compression*). As etapas de predição, transformada e codificação entrópica são totalmente reversíveis (*lossless compression*), apesar de sua grande capacidade de redução de redundância [18].

Nos padrões MPEG-2 e MPEG4-Visual, a etapa de transformação se dá pela aplicação, nos blocos residuais, de uma equação de transformada discreta de cosseno (2-D DCT). Um problema gerado por esse método é o aparecimento de discrepâncias entre as referências de predição do codificador e do decodificador, além de acumulação de distorções na saída do decodificador [31]. Isso se dá porque os processadores necessitam fazer arredondamentos em multiplicações de fatores irracionais. Para a mitigação desse problema, os padrões especificaram critérios de acuidade para as transformadas inversas, baseados no Padrão IEEE 1180-1990, e determinaram uma periodicidade reduzida de renovação de dependências, por meio de codificação Intra [32].

Padrões mais recentes, como o VC-1 e o próprio H.264/AVC, implementaram modificações nas etapas de transformação e quantização visando evitar essas discrepâncias nas referências geradas pelo codificador e pelo decodificador. Além disso, estabeleceram suporte a aritmética de inteiros, com precisão limitada, na intenção de minimizar a complexidade dos cálculos. A especificação do H.264/AVC detalha os processos de transformada inversa e de quantização inversa requeridos, fazendo com que os problemas causados pelas diferentes implementações de transformadas deixem de existir, uma vez que as diferentes implementações de decodificadores devem gerar resultados idênticos [18].

As etapas anteriores correspondentes (transformação e quantização), no entanto, não são descritas na especificação do padrão H.264/AVC, devendo ser inferidas dos processos de transformada inversa e de reescalonamento. A transformada básica, ou *core*, é uma transformada inteira em blocos residuais de tamanho 4x4 ou 8x8. Pode haver, em casos específicos, o cálculo da transformada de Hadamard (transformada DC) em parte da saída da transformada básica. Os coeficientes finais da transformada são, então, escalonados e quantizados [18]. Ver Figura 2.11.

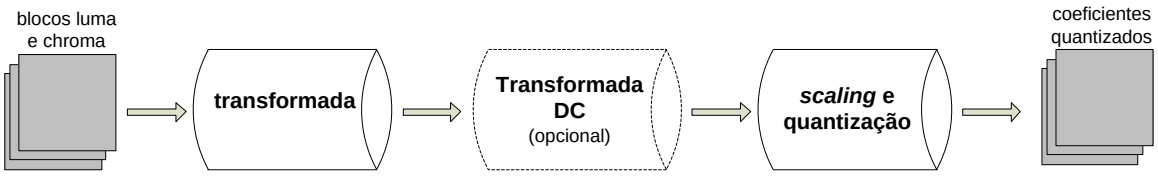

Figura 2.11: Etapas para codificação dos coeficientes quantizados.

Para decodificação, a especificação do padrão determina que a quantização inversa e o reescalonamento sejam calculados após a transformada inversa DC, se existente. Transformadas inteiras inversas 4x4 ou 8x8 devem, então, ser aplicadas nos coeficientes reescalonados. Ver Figura 2.12.

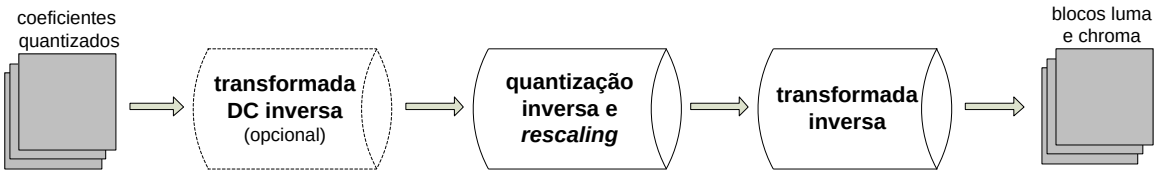

Figura 2.12: Etapas para decodificação dos coeficientes quantizados.

2.9.1 Procedimentos de transformação de amostras luma

Há três formas distintas de cálculo de transformada das amostras luma dos blocos residuais. O primeiro é utilizado quando os macroblocos são codificados usando predição Intra com partição de tamanho 16x16. O segundo se aplica somente quando está sendo utilizado o perfil *High* e a transformada 8x8 (opcional) está habilitada (só disponível para macroblocos codificados com modo de predição Intra-8x8 ou qualquer modo de predição Inter). A terceira forma é a padrão, ou seja, aplica-se a todos os demais casos [18].

No caso padrão, ilustrado pela Figura 2.13, cada bloco de tamanho 4x4 constante de uma região luma de tamanho 16x16 de um macrobloco é transformado, escalonado e quantizado para a produção de um bloco de tamanho 4x4 de coeficientes de transformada quantizados. A Figura 2.14 ilustra o procedimento inverso.

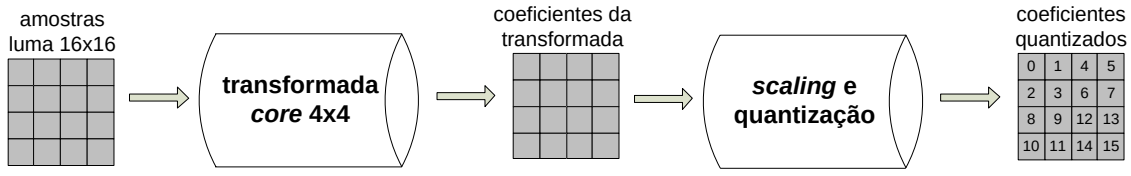

Figura 2.13: Procedimento padrão para cálculo dos coeficientes quantizados.

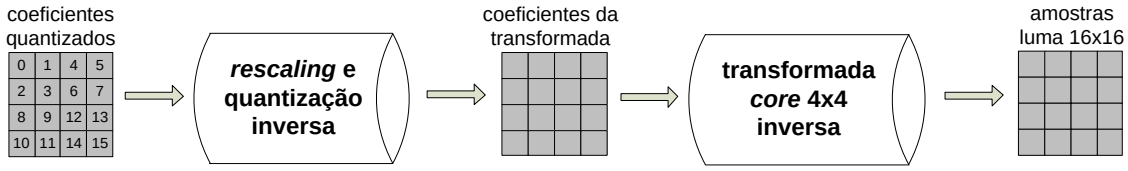

Figura 2.14: Procedimento padrão de reconstrução de amostras luma.

No primeiro caso (predição Intra com partição de tamanho 16x16), após o passo de transformação geral, é aplicada uma segunda transformação aos coeficientes com valores mais baixos (coeficientes de frequência "DC") entre os gerados pela primeira transformada. Isso é feito para aprimorar o desempenho da codificação, uma vez que esses coeficientes DC tendem a ser bastante correlacionados. Para isso, a transformada padrão é aplicada a cada bloco 4x4 de amostras luma. Os coeficientes DC de cada bloco 4x4 de coeficientes são agrupados para formarem um bloco 4x4 de coeficientes DC. Este bloco é, então, novamente transformado por meio de uma transformada Hadamard 4x4. Neste ponto, o bloco de coeficientes DC transformados e os outros 15 coeficientes AC em cada bloco são escalonados, quantizados e transmitidos em uma determinada ordem. Ver Figura 2.15. Uma observação importante é que no processo inverso, de decodificação, a transformada DC 4x4 inversa acontece antes dos passos de escalonamento e quantização inversos, no intuito de otimizar a faixa dinâmica [18]. Ver Figura 2.16.

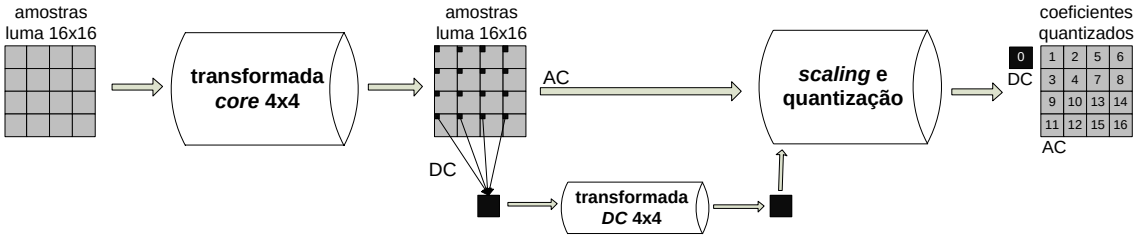

Figura 2.15: Procedimento para cálculo dos coeficientes quantizados (predição Intra 16x16).

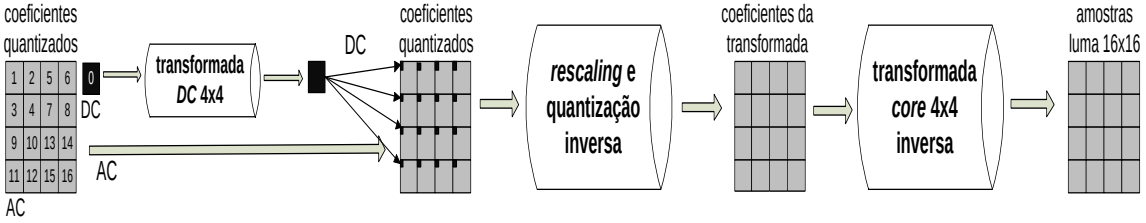

Figura 2.16: Procedimento para reconstrução das amostras luma (predição Intra 16x16).

No segundo caso (perfil *High* e escolha por transformada inteira 8x8), cada bloco 8x8 de amostras luma é transformado, escalonado e quantizado. Ver Figura 2.17. O processo inverso pode ser visto na Figura 2.18.

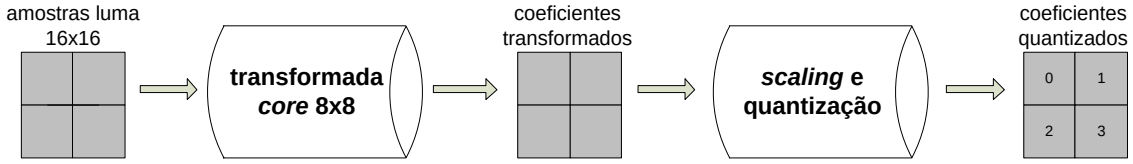

Figura 2.17: Procedimento para cálculo dos coeficientes quantizados (perfil *High* e transformada 8x8).

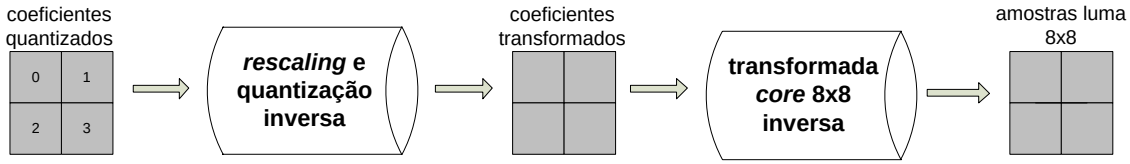

Figura 2.18: Procedimento para reconstrução das amostras luma (perfil *High* e transformada 8x8).

2.9.2 Procedimentos de transformação de amostras chroma

Os três diferentes formatos 4:2:0, 4:2:2 e 4:4:4 para representação dos componentes luma e chroma, quando usados para a codificação de um macrobloco, geram formas distintas de cálculo de transformação dos componentes chroma. Os componentes luma, como se mantêm constantes nos três formatos (a diferença se dá somente na quantidade de amostras chroma), não sofrem interferência.

No formato 4:2:0, para cada 16x16 amostras luma, há 8x8 amostras de crominância Cr e 8x8 amostras de crominância Cb. Nesse caso, o passo de transformada recebe como entrada blocos de 4x4 amostras de crominância Cr ou Cb. Cada um dos

blocos tem seus quatro coeficientes DC novamente transformados (DCT ou Hadamard 2x2). Os dois blocos DC e os oito blocos AC são, por fim, escalonados, quantizados e transmitidos/armazenados. Assim como no procedimento de transformada de amostras luma com predição Intra e partição de tamanho 16x16, no processo inverso a transformada DC 2x2 inversa ocorre antes dos passos de escalonamento e quantização inversos. Ver Figura 2.19 e Figura 2.20.

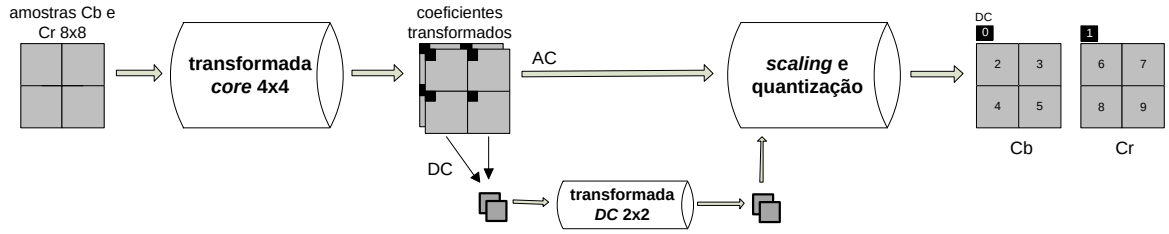

Figura 2.19: Procedimento para cálculo dos coeficientes quantizados das amostras chroma dos blocos residuais (formato 4:2:0).

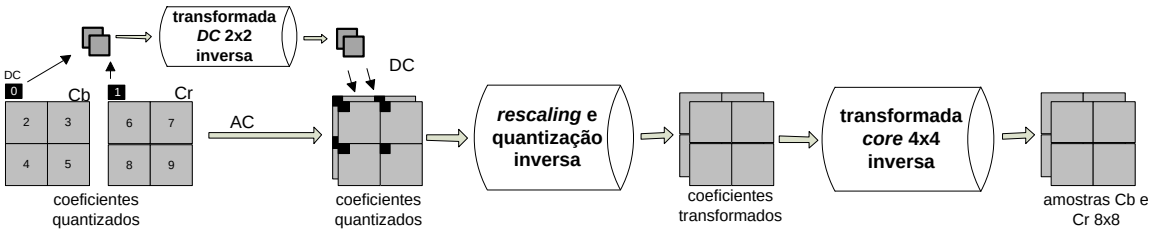

Figura 2.20: Procedimento para reconstrução das amostras chroma dos blocos residuais (formato 4:2:0).

No formato 4:2:2, para cada 16x16 amostras luma, há 8x16 amostras de crominância Cr e 8x16 amostras de crominância Cb. Quando esse é o caso, cada bloco de tamanho 4x4 é transformado, sendo os oito coeficientes DC novamente transformados (Hadamard 2x4). Os dois blocos DC e os dezesseis blocos AC são, então, escalonados, quantizados e transmitidos/armazenados. Novamente o processo inverso segue a ordem: transformada DC 2x4 inversa, escalonamento inverso e quantização inversa. Ver Figura 2.21 e Figura 2.22.

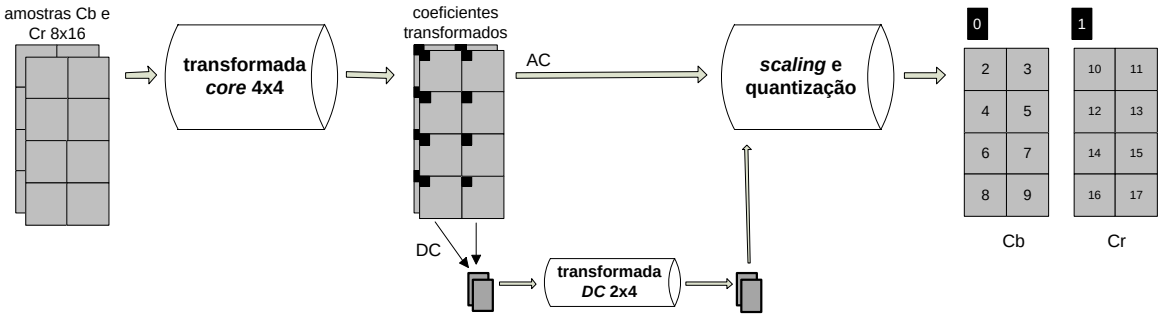

Figura 2.21: Procedimento para cálculo dos coeficientes quantizados das amostras chroma dos blocos residuais (formato 4:2:2).

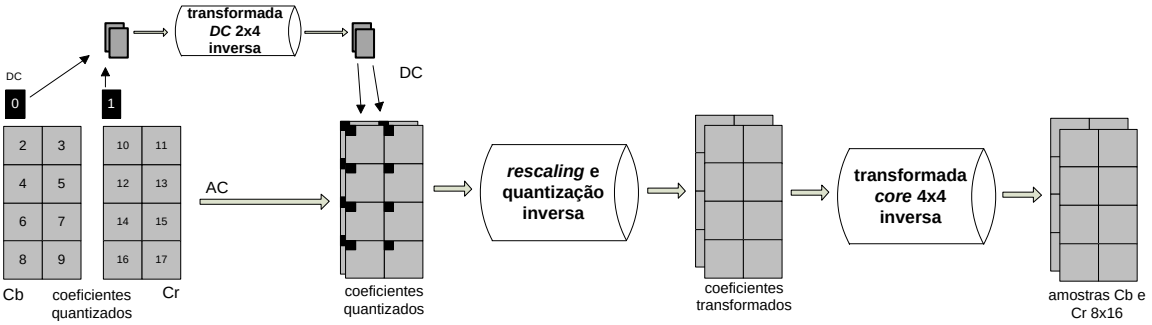

Figura 2.22: Procedimento para reconstrução das amostras chroma dos blocos residuais (formato 4:2:2).

Por fim, no formato 4:4:4, para cada 16x16 amostras luma há equivalentes 16x16 amostras de crominância Cr e 16x16 amostras de crominância Cb. Quando esse é o formato em uso, há três formas de cálculo de transformada das amostras de crominância, as quais são idênticas às três formas válidas para amostras luma.

2.9.3 Reorganização dos coeficientes de transformada quantizados

Antes de serem entropicamente codificados, os coeficientes de transformada quantizados de cada bloco são lidos (*scanned*) e rearranjados em uma lista de forma tal que os coeficientes com valor zero, normalmente a grande maioria, fiquem agrupados no fim da lista, sendo precedidos pelos coeficientes com valores diferentes de zero (coeficientes significativos) [33].

Os blocos pertencentes a quadros progressivos normalmente possuem os coeficientes significativos agrupados na extremidade superior esquerda (coeficientes DC),

e, portanto, a leitura dos coeficientes segue uma ordem em zigzag [33]. A Figura 2.23ilustra essa ordem, em um bloco de tamanho 4x4.

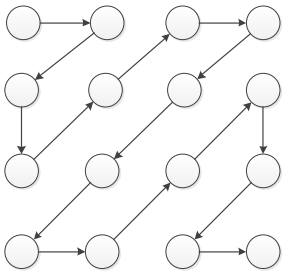

Figura 2.23: ordem dos coeficientes em um bloco de tamanho 4x4 pertencente a um quadro progressivo.

Em blocos de um campo entrelaçado, no entanto, os coeficientes significativos tendem a ocorrer tanto na extremidade superior quanto em direção à esquerda, exigindo, conforme ilustrado na Figura 2.24, uma ordem distinta de leitura [33].

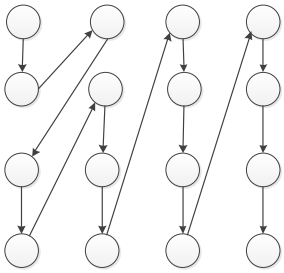

Figura 2.24: ordem dos coeficientes em um bloco de tamanho 4x4 pertencente a um quadro entrelaçado.

2.10 CODIFICAÇÃO ENTRÓPICA

O último passo da codificação de um fluxo H.264/AVC é a codificação entrópica, que tem como função comprimir os símbolos (elementos de sintaxe) calculados. Essa etapa recebe como entrada esses símbolos, que englobam os coeficientes de transformada, vetores de movimento codificados, tipos de predição, parâmetros, identificadores e códigos de delimitação, e, a partir desses dados, gera um fluxo de vídeo H.264/AVC codificado, que pode então ser armazenado ou transmitido. Os passos da codificação entrópica são totalmente reversíveis, sem qualquer perda de dados, tratando-se, portanto, de um método

de compressão *lossless*. A lógica nesse caso é, ao invés de designar uma *codeword* (símbolo codificado) de igual tamanho a todos os símbolos, usar modelos de probabilidade para designar *codewords* baseadas na quantidade esperada de ocorrências de cada um dos símbolos ou mesmo de conjuntos de símbolos [34].

O padrão H.264/AVC especifica quatro métodos possíveis de codificação entrópica: codificação com tamanho fixo (*fixed length code*), codificação de tamanho variável do tipo Golomb-Exponencial (*Exponential-Golomb variable length code*), CAVLC e CABAC. Os dois primeiros esquemas de codificação descritos, tamanho fixo e Golomb-Exponential, são utilizados para a codificação de símbolos que estejam acima do nível das *slices*, sintaticamente falando. Para os símbolos no nível das *slices* ou abaixo delas, a seleção de qual esquema é utilizado se dá da seguinte forma: se o modo CABAC está selecionado, os símbolos o usam para codificação entrópica. Caso contrário, valores de coeficientes são codificados por CAVLC e os outros símbolos são codificados usando codificação de tamanho fixo ou Golomb-Exponencial [34].

O método de codificação com tamanho fixo, como o próprio nome informa, gera um código binário com tamanho fixo para cada símbolo de entrada. O esquema de codificação Golomb-Exponencial de ordem k é um tipo de codificação universal, prefixal, parametrizada por um inteiro não-negativo k. O *codeword* consiste de um prefixo de x bits 0 (x >=0) o valor 1 e x bits de informação [34].

Para a codificação de um valor binário code_num, devem ser seguidos os seguintes passos [34]:

- retirar os últimos k bits do valor binário a ser codificado, code_num;
- adicionar 1 a esse valor, aritmeticamente;
- contar o número de bits desse valor e subtrair 1, aritmeticamente;
- sendo x o número de bits desses resultado, preceder (code_num + 1) com x bits 0s;
- suceder esse resultado com os k bits, em binário.

No padrão H.264/AVC, esses são os passos seguidos para a codificação de um code_num, com as particularidades de que k é sempre tomado como 0 e o valor zero é designado para o *codeword* binário 0. Assim, como exemplo, os *codewords* para os valores 0, 1, 2, 3 e 4, são, respectivamente, 1, 010, 011, 00100 e 00101. Para a codificação de um parâmetro p, o codificador faz um mapeamento desse valor para um code_num e o cálculo

acima descrito, com k=0, é realizado. Para a decodificação, o decodificador lê x bits 0 consecutivos, calcula o tamanho total da *codeword* como 2x + 1 bits, lê os x + 1 bits remanescentes e calcula o code_num, o qual é então mapeado para o parâmetro p [34].

Esse esquema de codificação é utilizado no intuito de que os símbolos que ocorram com frequência sejam representados por *codewords* curtas. A designação símbolo->*codeword* é feita logicamente e a decodificação pode ser calculada algoritmicamente sem a necessidade de pesquisa em tabelas [34].

CAVLC (*Context Adaptive Variable Length Coding*) é um modo de compressão suportado em todos os perfis H.264/AVC e requer menos poder computacional para decodificação do que o modo CABAC. Isso se dá, no entanto, ao custo de uma menor eficiência de compressão. É utilizado exclusivamente para codificação entrópica de blocos de coeficientes de transformada. Isso porque o modo se mostra mais eficiente exatamente nesse tipo de dado, que contém, em sua maioria, valores zero, seguidos, em quantidade e em ordem de ocorrência, de valores +1 e -1, denominados *trailing* 1s. Os valores diferentes de 0 e +1/-1, além de serem minoria, tendem a ser correlacionados entre blocos vizinhos e a agruparem-se no início da lista, próximo ao coeficiente DC. Nesses casos, CAVLC codifica os valores utilizando tabelas VLC. A escolha de qual tabela é usada em determinado momento depende da quantidade de coeficientes diferentes de zero encontrada em blocos anteriores. Um bloco de coeficientes, ao ser codificado pelo método CAVLC, tem seus valores lidos e convertidos em uma série de VLCs *(variable length codes*). A escolha de certas tabelas VLC baseia-se em estatísticas locais, ou seja, na quantidade de coeficientes significativos em blocos vizinhos e na magnitude de coeficientes recentemente codificados [34].

A codificação CABAC (*Context Adaptive Binary Arithmetic Coding*) é suportada somente nos perfis *Main* e superiores, por requerer um maior poder computacional para decodificação, se comparado com os demais modos suportados (tamanho fixo, Golomb Exponential e CAVLC). O processo de codificação envolve quatro passos: “binarização”, modelação do contexto, codificação aritmética binária e atualização de probabilidade [35].

No primeiro deles, “binarização”, símbolos não binários são mapeados para sequência binárias unívocas, denominadas *bin strings*. Isso é feito porque CABAC utiliza codificação aritmética binária e, portanto, somente decisões binárias (0 ou 1) são codificadas. No segundo passo, modelação do contexto, um modelo de contexto é

escolhido entre os disponíveis, segundo um critério que leva em conta os dados de símbolos recentemente codificados. Esse modelo de contexto é um modelo probabilístico para um ou mais *bins* do símbolo "binarizado", e tem como missão armazenar a probabilidade de um determinado *bin* ser 0 ou 1. O terceiro passo, codificação aritmética binária, é responsável por codificar cada *bin* de acordo com o modelo probabilístico selecionado, com a limitação de que há somente duas subfaixas para cada bin, correspondendo aos possíveis valores de 0 ou 1. O quarto e último passo, atualização de probabilidade, é responsável por atualizar o modelo de contexto selecionado, de acordo com o valor codificado atual. Para cada 0 ou 1 encontrado em um bin, o respectivo contador de frequência é atualizado [35].

2.11 CONTÊINERES DE BITSTREAMS H.264/AVC

Em terminologia de vídeo, containers são arquivos encapsuladores com a função de armazenar (ou informar onde estão armazenados) um ou mais *bitstreams* de vídeo e possíveis *bitstreams* de áudio, legendas e informações sobre capítulos, além de metainformações que permitam ao reprodutor decodificar os *bitstreams* e sincronizá-los. Os metadados, no entanto, não detalham como os *bitstreams* devem ser decodificados, restringindo-se a informar quais padrões de codificação e parâmetros foram utilizados [36] [37].

Um aplicativo interessado em reproduzir um arquivo de vídeo deve, primeiramente, ser capaz de interpretar o contêiner e dele extrair os metadados do(s) *bitstream*(*s*). Em um segundo momento, verificará se suporta o padrão de codificação utilizado pelo(s) *bitstream*(*s*) e só então, em caso positivo, procederá à decodificação utilizando os parâmetros informados pelo contêiner.

A família de padrões ISO/IEC 14496 MPEG-4 define diversos padrões de contêineres para armazenamento de vídeo. O principal deles é o padrão especificado em sua parte 12 (ISO/IEC 14496-12), *ISO Base Media File Format* [38], que, por sua vez, é diretamente baseado no padrão *QuickTime File Format* (QTFF) [39]. Sua destacada importância reside no fato de ele ser utilizado como base para outros padrões de contêiner mais específicos, como o *MP4 File Format* (ISO/IEC 14496-14) [40], o *Advanced Video Coding File Format* (ISO/IEC 14496-15) [41], e mesmo padrões que não pertencem à

família ISO/IEC 14496, como o *3GPP File Format* [42] e o *3GPP2 File Format* [43]. A relação entre esses contêineres é ilustrada na Figura 2.25.

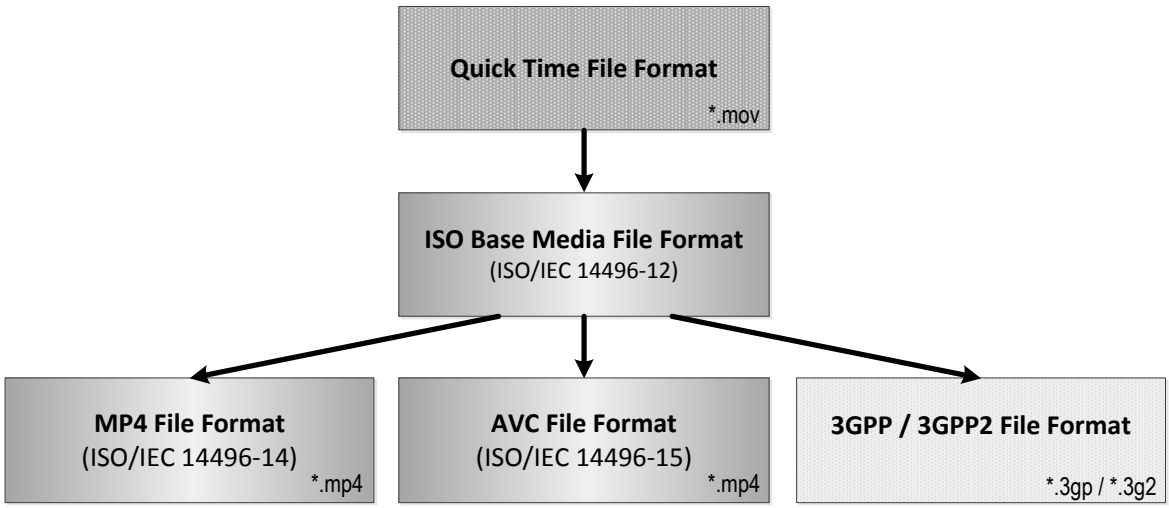

Figura 2.25:Relação entre contêineres.

Em 2001, a empresa Apple publicou a especificação do padrão de contêiner *QuickTime File Format*. Ainda nesse ano foi publicada a norma ISO/IEC 14496-1 [44], especificando a primeira versão do padrão *MPEG-4 File Format*. Este padrão é essencialmente idêntico ao anterior, diferenciando-se, na prática, somente pelo acréscimo de suporte a algumas características específicas de codificações de vídeos padronizadas pela ISO/IEC 14496. Em 2003, uma segunda versão do padrão *MPEG-4 File Format* substituiu a primeira, tendo sido publicada como ISO/IEC 14496-14 [40]. Esta segunda versão foi generalizada para a criação do padrão *ISO Base Media File Format*, cuja especificação foi publicada como ISO/IEC 14496-12 [38]. Esse padrão base age como fundação para outros formatos, como o próprio *MPEG-4 File Format* [40], o *3GPP/3GPP2 File Format* [42] [43], e o *Advanced Video Coding* (AVC) *File Format* [38].

O padrão *MPEG-4 File Format* segue fielmente a normatização contida na especificação do padrão *ISO Base Media File Format*, expandindo-a, contudo, para permitir melhor gerenciamento dos metadados dos *bitstreams*. A extensão oficial para arquivos que atendam a esse padrão é "`.MP4`". A distinção de outros padrões que também se utilizam dessa extensão se dá por meio de código no cabeçalho do arquivo: "`mp41`", se

em conformidade com a primeira versão (ISO/IEC 14496-1:2001), ou "`mp42`", se em acordo com a segunda versão (ISO/IEC 14496-1:2003) [40].

O padrão *AVC File Format* foi normatizado pela ISO/IEC 14496-15 [41] em 2004 visando o armazenamento de *bitstreams* H.264/AVC (ISO/IEC 14496-10). É plenamente compatível com o padrão *ISO Base Media File Format*, diferenciando-se por acrescentar suporte a aspectos específicos do padrão de codificação H.264/AVC. Arquivos que atendam a essa especificação devem possuir a extensão "`.MP4`" e diferenciam-se de outros formatos que também utilizam essa extensão por meio do código "`avc1`" contido em campo de cabeçalho [41].

Os padrões 3GPP e 3GPP2 são amplamente utilizados em equipamentos de telefonia celular e diferenciam-se entre si basicamente pelo fato de o primeiro ser direcionado a aparelhos com tecnologia UMTS/GSM, ao passo que o segundo é utilizado em aparelhos com tecnologia CDMA, o que exige algumas alterações no formato. Ambos são diretamente baseados no padrão *ISO Base Media File Format*, com alterações que permitem uma melhor adaptação dos contêineres a sua utilização em dispositivos portáteis com limitações de banda. As extensões oficiais dos arquivos contêineres 3GPP e 3GPP2 são, respectivamente,"`.3gp`" e "`.3g2`", apesar de alguns aparelhos utilizarem a extensão "`.MP4`". Há uma grande quantidade de aparelhos de telefonia celular que se utilizam desse tipo de contêiner para armazenamento de *bitstreams* H.264/AVC. Nesses casos, a especificação dos formatos 3GPP e 3GPP2 remete à ISO/IEC 14496-15 (*AVC File Format*) para o armazenamento dos metadados específicos a esse padrão de codificação de vídeo [42] [43].

Todos esses padrões de contêineres que se baseiam no *ISO Base Media File Format* são compostos por uma coleção de compartimentos de diversos tipos, os quais podem ter seus dados lidos e utilizados, ou serem simplesmente ignorados, se o aplicativo não reconhecer seu tipo. Essa particularidade permite que os padrões mantenham-se compatíveis com versões futuras que optem por ampliar os tipos de compartimentos. Não há dados nos arquivos contêineres que não os contidos nos compartimentos. As unidades básicas do contêiner, ou seja, seus compartimentos, são denominadas boxes[3]. Conforme ilustrado na Figura 2.26, boxes são hierárquicos, de forma que um box pode conter outros boxes, e assim sucessivamente. Boxes que contenham outros boxes são denominados

[3] No padrão *QuickTime File Format*, os objetos possuem nomenclatura distinta, sendo chamados de átomos.

container boxes. Sua função é auxiliar a estruturação hierárquica dos dados e metadados, os quais, por sua vez, são apresentados, em forma de campos ou tabelas, pelos boxes finais (*leaf boxes*), que não contêm outros boxes [38].

Figura 2.26: Organização hierárquica dos boxes.

Um box contém, em seu cabeçalho, dois campos: tamanho e tipo. O campo tipo informa a natureza dos dados contidos no box e, por conseguinte, o formato em que se encontram, e é especificado por um número inteiro de 32 bits, em ordenamento *big-endian* (bytes mais significantes são armazenados/transmitidos primeiro), comumente interpretado como um código de quatro caracteres ASCII. A compatibilidade entre diferentes versões da especificação que incorporem novas funcionalidades e novos boxes é facilitada pela determinação, contida na especificação, de que boxes com tipo não reconhecido devem ser simplesmente ignorados.

Três importantes códigos são o "`moov`" (0x6D6F6F76), que informa que em seu interior estão armazenados, em sub-boxes, os metadados necessários para a localização,

decodificação e sincronização dos *bitstreams*, o "`mdat`" (0x6D646174), que informa se tratar de box contendo um *bitstream* (percebe-se, dessa informação que os *bitstreams* e seus metadados são armazenados separadamente), e o "`ftyp`" (0x66747970).

Esse último tipo, denominado *File Type Box*, tem como função informar a qual especificação derivada do ISO Base Media File o arquivo contêiner atende. Desse código pode-se depreender, dentre outras informações, a codificação utilizada no(s) *bitstream(s)*, particularidades de armazenamento dos dados e restrições ou extensões implementadas no arquivo [38].

O tamanho, por sua vez, informa o número total de bytes ocupados pelo box, incluindo o cabeçalho e os sub-boxes, por meio de um número inteiro de 32 bits. Duas particularidades ocorrem se os valores forem 0 ou 1. Sendo 0, deve-se entender que o box é o último do arquivo e estende-se até o seu fim. Sendo 1, transfere-se a informação sobre o tamanho do box para um campo denominado tamanho estendido, com 64 bits. A utilidade desse campo opcional é permitir que boxes com dados de mídia com tamanhos superiores a 2^{32} bytes possam ser expressos, e, segundo a especificação, só deve ser utilizado, se houver real necessidade. [38].

2.12 CONCEITOS BÁSICOS DE INFORMÁTICA

Diversos conceitos de informática necessários para o entendimento deste trabalho são expostos nesta subseção. Entre os tópicos cobertos, veem-se informações sobre os padrões de notações decimal e binária, sistemas de arquivos e funções *hash*, além de um resumo sobre as diversas ferramentas utilizadas.

2.12.1 Notação decimal e binária

No decorrer desta dissertação, as potências de 10 serão indicadas de acordo com as normas do Sistema Internacional (SI) de medidas [45]. Para as potências de dois, serão usadas as recomendações da norma IEC 80000-13 [46]. A Tabela 2.6mostra os valores, prefixos e nomes correspondentes.

Tabela 2.6: Prefixos decimais e binários.

SI		IEC 80000-13	
Potência	Prefixo (nome)	Potência	Prefixo (nome)
10^3	k (quilo)	2^{10}	Ki (kibi)
10^6	M (mega)	2^{20}	Mi (mebi)
10^9	G (giga)	2^{30}	Gi (gibi)
10^{12}	T (tera)	2^{40}	Ti (tebi)
10^{15}	P (peta)	2^{50}	Pi (pebi)
10^{18}	E (exa)	2^{60}	Ei (exbi)
10^{21}	Z (zeta)	2^{70}	Zi (zebi)
10^{24}	Y (iota)	2^{80}	Yi (yobi)

O bit (0 ou 1, a menor unidade de informação) e o byte (um octeto ou conjunto de 8 bits) não fazem parte do SI. Portanto, quando associados a prefixos decimais, devem ser designados por extenso. Fabricantes de discos rígidos e *pendrives*, por exemplo, usam potências de 10, e costumam usar a notação GB para gigabyte e TB para terabyte. O correto seria usar Gbyte e Tbyte, pois o "B" tem como significado mais comum o bel, cujo uso é aceito pelo SI, mesmo não sendo um padrão [45].

Quando associado a um prefixo binário, o byte deve ser representado pela letra "B" [46]. Módulos de memória usam potência de dois. Assim, a notação correta é MiB (mebibyte) e GiB (gibibyte). Se um computador tem 512 MiB ou 4 GiB de memória RAM instalada, é (duplamente) incorreto usar as expressões 512 MB ou 4 GB, respectivamente.

2.12.2 Blocos e setores: endereçamento

Um meio de armazenamento comum é o disco rígido magnético (HD – *hard disk*), o qual consiste, basicamente, de um ou mais pratos (discos) cobertos por um material magnético. *Pendrives,* cartões de memória e discos de estado sólido (SSDs – *solid state discs*), por outro lado, utilizam-se de memória flash, sendo os bits armazenados em circuitos eletrônicos. Ambas as formas de armazenamento, no entanto, compõem-se de unidades mínimas de armazenamento, de tamanho fixo (geralmente 512 bytes), denominadas setores. O termo bloco (ou unidade de alocação) é mais usado no contexto do sistema operacional e consiste em um grupo de setores contíguos que formam a menor unidade gerenciável pelo sistema de arquivos. Tipicamente, o bloco possui um tamanho de 4096 bytes (NTFS, EXT2), mas pode ter 512 bytes (NTFS) ou até 32KiB (FAT), dependendo do tamanho do volume.

2.12.3 Sistemas de arquivos

Um arquivo é um conjunto de bytes. Um sistema de arquivos, por sua vez, é um conjunto de estruturas lógicas que permitem o gerenciamento dos arquivos – criação, destruição, leitura, gravação, controle de acesso. Exemplos de sistemas de arquivos são o FAT, amplamente usado em dispositivos portáteis, como *pendrives* e cartões de memória, o NTFS, usado em sistemas operacionais da Microsoft, o EXT2, desenvolvidos para o sistema operacional Linux, e o HFS+, destinado a sistemas operacionais Mac OS, da empresa Apple, Inc.

2.12.4 Assinatura digital: função *hash*

Uma função *hash* ou função de mão única é definida como $h: A \rightarrow B$ tal que, para todo x (mensagem) pertencente a A, $h(x)$ (valor de *hash*) é computacionalmente fácil[4] e possui as seguintes propriedades [47]:

a) qualquer que seja y pertencente a B, é computacionalmente inviável[5] encontrar x em A tal que $h(x) = y$ (*pre-image resistance*);

b) para um determinado x pertencente a A, é computacionalmente inviável encontrar x', também pertencente a A, diferente de x, tal que $h(x') = h(x)$ (*second pre-image resistance*);

c) é computacionalmente inviável encontrar x *e* x', pertencentes a A e diferentes entre si, tais que $h(x) = h(x')$ (*collision resistance*).

Embora "b" e "c" pareçam iguais, computacionalmente são propriedades diferentes. Existem demonstrações de violação de "c" para o algoritmo MD5 [47], mas não há violação de"b" para esses e outros algoritmos.

O algoritmo MD5 [48] pode ser usado para efetuar a verificação de integridade de um arquivo. A assinatura (*hash*) gerada tem 128 bits, e a aplicação md5sum (para Linux) mostra esse *hash* em 32 caracteres hexadecimais. Exemplo de uso da ferramenta md5sum:

```
$ md5sum nomeArquivo > ./md5sum.txt
```

[4] Computacionalmente fácil é o oposto de computacionalmente inviável.
[5] Computacionalmente inviável significa que o tempo de processamento é muito grande (podendo ser da ordem de centenas ou milhares de anos, considerando-se a capacidade computacional atual de um único computador, mesmo multiprocessado).

O arquivo de texto `./md5sum.txt` conterá a assinatura (*hash*) do arquivo `nomeArquivo`.

2.12.5 Preservação da mídia original

A análise pericial dos dados de uma mídia de armazenamento permanente (como um disco rígido ou um *pendrive*) pode ser dividida em duas etapas: (i) geração de uma imagem da mídia, para preservação da evidência digital; (ii) pesquisa e recuperação de dados, realizados na imagem [49].

A geração de uma imagem da mídia (procedimento de cópia) deve preencher dois requisitos fundamentais: (i) assegurar a geração de uma imagem fiel à original (algumas vezes a produção dessas imagens é chamada de cópia bit a bit); (ii) garantir que a mídia original não tenha sido alterada. Um sistema Linux (ou outro sistema compatível com UNIX) corretamente configurado bloqueia a montagem de um sistema de arquivos. Assim, pode-se utilizar uma ferramenta que somente efetue a leitura do meio, criando a cópia desejada. Uma ferramenta aprovada pelo NIST é o `dd` [50]. Por fim, uma técnica que permite avaliar ambos os requisitos é a utilização de um algoritmo de integridade, como uma função *hash* (Ver Seção 2.12.4).

2.12.6 Data carving

Data carving é o nome dado ao processo de recuperação de arquivos com base em cabeçalhos, trechos finais e estruturas de dados internas. Essa é a forma utilizada para reconstrução de arquivos quando os metadados do sistema de arquivos encontram-se corrompidos ou ausentes. Como exemplo, tome-se um arquivo de imagem JPEG. Sabe-se que, por padrão, arquivos desse tipo possuem, em seus dois primeiros bytes, o valor 0xFFD8 (cabeçalho), e finalizam-se com os bytes 0xFFD9. Assim, em um procedimento de recuperação de uma imagem JPEG por *data carving*, a mídia sob exame seria lida sequencialmente até que o marcador 0xFFD8 fosse encontrado. Todos os dados entre esse cabeçalho e o marcador de fim de arquivo 0xFFD9, incluindo-os, seriam então copiados e salvos como um arquivo de imagem JPEG recuperado. Há diversas ferramentas que automatizam esse processo, entre elas uma de nome `Foremost` (ver Seção 2.12.7).

2.12.7 Comandos e aplicativos para o sistema operacional Linux

São apresentados, nesta subseção, os comandos e aplicativos para o sistema operacional Linux que foram utilizados neste trabalho.

▪ `cat`: concatena arquivos, em sua função original, mas presta-se também para a simples exibição do conteúdo de um arquivo no monitor.

▪ `dd`: copia um arquivo ou dispositivo usando tamanhos de bloco especificados (o padrão é 512 bytes). Alguns de seus parâmetros são:

`count=`: número de blocos a serem lidos;

`if=`: arquivos de entrada;

`of=`: arquivo de saída;

`seek=`: a escrita irá ocorrer após o número de blocos determinado;

`skip=`: número de blocos desconsiderados, na leitura;

`bs=`: tamanho do bloco de entrada/saída, em bytes.

▪ `file`: indica o tipo de arquivo ou diretório indicado pelo usuário conforme os padrões do sistema operacional.

▪ `ls`: lista arquivos e diretórios, informando nome, tamanho e datas, entre outros metadados.

▪ `av_mpeg`: apresenta diversas informações sobre um arquivo de vídeo qualquer.

▪ `foremost`: recupera (reconstrói) arquivos baseando-se em cabeçalhos, trechos finais e estruturas de dados internas. Recebe os seguintes parâmetros: tamanho do bloco usado pelo sistema de arquivos; tipo do arquivo procurado (os tipos aceitos são informados no manual da ferramenta); nome do diretório onde serão gravados os arquivos recuperados; nome do arquivo (geralmente uma imagem `dd`) que contém a imagem do volume a ser recuperado.

▪ `SleuthKit`: conjunto de ferramentas, agrupadas por categorias (denominadas camadas – *layers*), muitas delas voltadas para a visualização e pesquisa de mídia [51]. Neste trabalho, foi usada uma única ferramenta deste pacote:

  ▫ `sigfind`: procura por uma assinatura em um bloco. Os seus parâmetros são os seguintes: número de bytes por bloco (o padrão é 512, sendo usual o valor 4096); *offset*/deslocamento (número de bytes ignorados antes de pesquisar a assinatura do bloco); -l (indica ordenamento *little endian*); assinatura

(sequencia a ser pesquisada, em formato hexadecimal); nome do arquivo a ser pesquisado.

▪ `ffmpeg`: ferramenta de linha de comando, de código aberto, que grava, converte e cria *streams* de áudio e vídeo em diversos formatos [52].Utiliza-se das bibliotecas `libavcodec`, com codificadores e decodificadores de áudio e vídeo, e `libavformat`, com multiplexadores e demultiplexadores para diversos formatos de contêineres multimídia. O aplicativo lê o(s) arquivo(s) de entrada, especificados pela opção *`-i`*, realiza os processamentos determinados via opções e gera o(s) arquivo(s) de saída. O comando `$ ffmpeg -i video.mp4 -vframes 10 quadros_%02d.jpeg`, por exemplo, extrai os primeiros 10 quadros do arquivo de entrada,`vídeo.mp4`.

3 ANÁLISE DE ARQUIVO DE VÍDEO

Neste capítulo é examinado um *bitstream* de vídeo H.264/AVC armazenado em um contêiner ISO/IEC 14496 (Seção 3.1). Todos os boxes contidos no contêiner são apresentados e têm seu conteúdo determinado e interpretado, à luz de suas especificações, em um processo exaustivo e pioneiro de decodificação dos bits que representam seus inúmeros campos e parâmetros. A análise desse arquivo, e da teoria a ele vinculada, foi dividida em três partes: metadados (box "`moov`" e sub-boxes) (Seção 3.2), dados (box "`mdat`") (Seção 3.3) e *bitstream* (Seção 3.4). A Seção 3.5 finaliza este capítulo comprovando que, sem as informações contidas nos metadados, a recuperação dos quadros (imagens) de um arquivo de vídeo não é possível.

3.1 CARACTERÍSTICAS DO ARQUIVO

O arquivo sob estudo, `j10_3786k.mp4`, encontrado dentro de um cartão de memória encaminhado para perícia no Instituto de Criminalística da PCDF, é um contêiner cujo formato – MPEG-4 parte 12, *ISO base media* – é definido pela norma ISO/IEC 14496-12:2008. Seu conteúdo consiste em dois fluxos (*streams*), um de vídeo – H.264/AVC – e outro de áudio, conforme pode ser visto na Figura 3.1. O par vídeo-áudio é denominado apresentação (*presentation*) pela norma.

```
$ ls -lgG j10_3786k.mp4
-rw-r--r-- 1 3877672 2011-10-03 17:31 j10_3786k.mp4

$ file j10_3786k.mp4
j10_3786k.mp4: ISO Media, MPEG v4 system, version 1

$ ./av_mpeg j10_3786k.mp4
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'j10_3786k.mp4':
Duration: 00:01:01.90, start: 0.000000, bitrate: 501 kb/s
Stream #0.0(und): Video: h264, yuv420p, 480x360, 29.97 tbr,[...]
Stream #0.1(und): Audio: aac, 22050 Hz, stereo, s16
```

Figura 3.1: Identificação de um vídeo MPEG-4, através dos comandos `file`[6] e `av_mpeg`[7].

Um contêiner MPEG-4 (parte 12) constitui-se de compartimentos (*boxes*), os quais, por sua vez, podem conter subcompartimentos. Os dois principais compartimentos

[6] Identifica o tipo de arquivo.
[7] Provê informações sobre o arquivo de vídeo.

de mais alta hierarquia são o "mdat" (para os fluxos – *streams* – de dados) e o "moov" (contendo os metadados). Não há ordem pré-definida entre eles. No arquivo em análise, "moov" aparece primeiro. Um compartimento possui um cabeçalho com oito bytes – quatro para o tamanho e quatro para o tipo/identificador.

As figuras seguintes mostram a saída do comando "$ dd if=j10_3786k.mp4 2>/dev/null | xxd[8]", que foi editada e dividida em várias partes para facilitar o entendimento do formato do arquivo e seu conteúdo. A primeira coluna mostra o endereço do primeiro byte da linha (em hexadecimal); as colunas intermediárias mostram 16 bytes por linha, agrupados de dois em dois (um byte é representado por dois caracteres hexadecimais); a última coluna mostra, quando existente, a representação ASCII dos valores mostrados nas colunas intermediárias.

Os primeiros quatro bytes ("0000 0014") do arquivo mostram o tamanho do cabeçalho – neste caso, 0x0014 = 20 bytes. Os quatro bytes seguintes ("6674 7970") identificam o arquivo como sendo um contêiner MPEG-4 – "ftyp". Os identificadores "isom" ("6973 6f6d") representam a "marca" (*brand*) e compatibilidade – "isom" corresponde à primeira versão do formato de arquivo "*ISO Base Media*", definido pela norma (ver Figura 3.2).

```
$ dd if=j10_3786k.mp4 2>/dev/null | xxd
0000000: 0000 0014 6674 7970 6973 6f6d 0000 0001  ....ftypisom....
0000010: 6973 6f6d                                isom..:.
```

Figura 3.2: Assinatura de um contêiner MPEG-4.

A norma ISO/IEC 14496-15:2004 estipula que, se as extensões previstas por ela forem requeridas, a marca deve ser "avc1", o que não ocorreu neste caso. A identificação do tipo H.264/AVC está indicada apenas no compartimento "stsd", descrito na seção 3.2.3.2.3.3.1.

Todos os compartimentos (boxes) presentes no arquivo j10_3786k.mp4 encontram-se lançados na Tabela 3.1, assim como a informação de em qual seção deste trabalho eles foram tratados.

[8] Copia os dados do arquivo para a saída padrão (monitor) e apresenta-os em formato hexadecimal.

Tabela 3.1: Boxes presentes no arquivo j10_3786k.mp4.

Compartimentos (boxes)							Seção
ftyp							atual
moov							3.2
	mvhd						3.2.1
	iods						3.2.2
	trak						3.2.3
		tkhd					3.2.3.1
		mdia					3.2.3.2
			mdhd				3.2.3.2.1
			hdlr				3.2.3.2.2
			minf				3.2.3.2.3
				vmhd			3.2.3.2.3.1
				dinf			3.2.3.2.3.2
					dref		3.2.3.2.3.2
					url		3.2.3.2.3.2
				stbl			3.2.3.2.3.3
					stsd		3.2.3.2.3.3.1
						avc1	3.2.3.2.3.3.1.1
						avcc	3.2.3.2.3.3.1.2
						btrt	3.2.3.2.3.3.1.3
					stts		3.2.3.2.3.3.2
					stss		3.2.3.2.3.3.3
					stsc		3.2.3.2.3.3.4
					stsz		3.2.3.2.3.3.5
					stco		3.2.3.2.3.3.6
	trak						3.2.4
mdat							3.3

3.2 ANÁLISE DOS METADADOS

O box "moov", apresentado na Figura 3.3, armazena os metadados do fluxo de vídeo (*bitstream*). Seu tamanho é de 14989 bytes (“0000 3a8d”: 0x3a8d). Somando-se o endereço do seu início (0x0014) ao tamanho (0x3a8d), temos o endereço (0x3aa1) de “mdat” (ver Seção 3.3).

```
0000014:           0000 3a8d 6d6f 6f76 0000 006c      ....moov...l
0000020: 6d76 6864 0000 0000 c665 eccf c665 eccf  mvhd.....e...e..
```

Figura 3.3: Início do box “moov”.

3.2.1 Box "mvhd" – “movie header box”

É o primeiro subcompartimento (sub-box) de “moov”. Contém as declarações gerais sobre a apresentação, considerada como um todo (*movie header, overall declarations*), e começa imediatamente após o cabeçalho de “moov”, a partir do endereço 0x001c, conforme visto na Figura 3.4.

```
000001c:                                  0000 006c              ...l
0000020: 6d76 6864 0000 0000 c665 eccf c665 eccf  mvhd.....e...e..
0000030: 0000 0258 0000 9116 0001 0000 0100 0000  ...X............
0000040: 0000 0000 0000 0000 0001 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0001 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 4000 0000 0000 0000  ........@.......
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0003                      ............
```

Figura 3.4: Box “mvhd”.

Os quatro primeiros bytes indicam o tamanho deste box: 108 bytes (0x006c) e mostram que o próximo box estará no endereço 0x0088 (= 0x001c + 0x006c) – o identificador é “iods” (ver Seção 3.2.2). Os valores lidos após o cabeçalho são mostrados na Tabela 3.2:

Tabela 3.2: Box “mvhd”.

Valor Hexadecimal	Significado	Descrição
00	versão 0[9]	rótulos de tempo em 32 bits
000000	zeros	campo reservado
c665 eccf	3328568527 s 2009-06-23 02:22:07 (UTC)[10]	rótulo de tempo de criação
c665 eccf	3328568527 s 2009-06-23 02:22:07 (UTC)[11]	rótulo de tempo de modificação
0000 0258	600 Hz	unidade de tempo
0000 9116	37142 -> 61,9 s ~ 62 s[12]	duração do vídeo
0001 0000	1.0(16.16)[13]	taxa ("*rate*": 1.0

[9] A versão 0 implica o uso de inteiros de 32 bits para rótulos de tempo; a versão 1 usa 64 bits.

[10] Os rótulos de tempo são registrados em segundos decorridos desde 01-01-1904 (à zero hora).

[11] Idem.

[12] Este valor corresponde ao número de unidades de tempo definido no campo anterior (600, neste caso).

		significa "*normal playback*"
0100	1.0 (8.8)[14]	volume (1.0 indica "*full volume*")
0000 0000 0000 0000 0000	zeros	campo reservado
0001 0000	{0x00010000,0,0, 0,0x00010000,0, 0,0,0x40000000}	"*unity matrix*"
0000 0000		
0000 0000		
0000 0000		
0001 0000		
0000 0000		
0000 0000		
0000 0000		
4000 0000		
0000 0000 (6x)	zeros	campo reservado
0000 0003	3	"*next_track_ID*"

3.2.2 Box "iods" – "initial object descriptor"

De acordo com a norma ISO/IEC 14496-14, contém os descritores de objetos iniciais. Começa no endereço 0x0088 e seu tamanho é de 21 bytes (0x0015), conforme visto na Figura 3.5; o próximo compartimento ("trak", s. 3.2.3) começa em 0x009d (= 0x0088 + 0x0015). Os valores são mostrados na Tabela 3.3.

```
0000088:                    0000 0015 696f 6473          ....iods
0000090: 0000 0000 1007 004f ffff 2815 ff         .......O..(..
```

Figura 3.5: Box "iods".

[13] Alguns valores são definidos por números decimais e serão indicados por (x.y), onde x e y representam a quantidade de bits para a parte inteira e a fração decimal, respectivamente.

[14] Idem.

Tabela 3.3: Box "iods".

Valor Hexadecimal	Significado	Descrição
00	0	versão
000000	0	flags
10	16	Tipo
07	07 bytes	Tamanho dos descritores
004f	0b0000000001001111, onde os bits são divididos da seguinte forma: 0b0000000001 = 1 0b0 = 0 0b0 = 0 0b1111 = valor fixo	 -> ObjectDescriptorID -> URL_Flag -> includeInlineProfilesFlag -> bits reservados
Ff	"no scene capability required"	"sceneProfile value"
Ff	"no audio capability required"	"audioProfile value"
28	"reserved for ISO use"	"visualProfile value"
15	"reserved for ISO use"	"graphicsProfile value"
Ff	sem descrição em ISO 14496-1:1998	

3.2.3 Box "trak" – "*track box*"

Compartimento para uma trilha (*track*), de vídeo ou de áudio, de uma apresentação. Este compartimento, mostrado na Figura 3.6, refere-se a uma trilha de vídeo (Seção "hdlr", 3.2.3.2.2). O seu tamanho (0x2146), somado ao endereço inicial (0x009d), determina o endereço do próximo compartimento: 0x21e3 ("trak" de áudio, ver Seção3.2.4). Possui dois boxes: "tkhd" (Seção 3.2.3.1) e "mdia" (Seção 3.2.3.2).

```
000009d:                                    00 0021                ..!
00000a0: 4674 7261 6b00 0000 5c74 6b68 6400 0000  Ftrak...\tkhd...
```

Figura 3.6: Box "trak".

3.2.3.1 Box "tkhd" – "*track header box*"

É o primeiro subcompartimento de "trak", tendo como função especificar características da trilha. Segue imediatamente o cabeçalho anterior, começando no

endereço 0x00a5, conforme mostrado na Figura 3.7. O seu tamanho é 0x005c, que somado ao seu início, aponta para o próximo compartimento: "mdia", a partir de 0x0101 (ver Seção 3.2.3.2). Seus valores estão descritos na Tabela 3.4.

```
00000a5:             00 0000 5c74 6b68 6400 0000       ...\tkhd...
00000b0: 0f7c 25b0 80c6 65ec cf00 0000 0100 0000  .|%...e.........
00000c0: 0000 0090 7000 0000 0000 0000 0000 0000  ....p...........
00000d0: 0000 0000 0000 0100 0000 0000 0000 0000  ................
00000e0: 0000 0000 0000 0100 0000 0000 0000 0000  ................
00000f0: 0000 0000 0040 0000 0001 e000 0001 6800  .....@........h.
0000100: 00                                       ..
```

Figura 3.7: Box "tkhd".

Tabela 3.4: Box "tkhd".

Valor Hexadecimal	Significado	Descrição
00	versão 0[15]	rótulos de tempo em 32 bits
00000f	15 (= 1 + 2 + 4 + 8)[16]	sinalizadores
7c25 b080	2082844800 s 1970-01-01 00:00:00 (UTC)[17]	rótulo de tempo de criação
c665 eccf	3328568527 s 2009-06-23 02:22:07 (UTC)[18]	rótulo de tempo de modificação
0000 0001	1	"*track ID*"
0000 0000	Zeros	campo reservado
0000 9070	36976 -> 61,6 s[19]	duração da trilha
0000 0000 0000 0000	Zeros	campo reservado
0000	0	"*layer*"
0000	0	"*alternate_group*"
0000	0	volume (0x0100 se áudio)
0000	0	campo reservado
0001 0000 0000 0000 0000 0000 0000 0000	{0x00010000,0,0, 0,0x00010000,0, 0,0,0x40000000}	"*unity matrix*"

[15] A versão 0 implica inteiros de 32 bits para rótulos de tempo; a versão 1 usa 64 bits.

[16] Os sinalizadores deste (*flags*) são somados, e seus significados são dados pelos bits. Neste caso, os valores individuais são: 0x0001 – "track_enabled"; 0x0002 – "track_in_movie; 0x0004 – "track_in_preview"; o valor 0x0008 não foi definido pela norma ISO/IEC 14496-12:2008.

[17] Os rótulos de tempo são registrados em segundos decorridos desde 01-01-1904 (à zero hora).

[18] Idem.

[19] Usa o mesmo número de unidades de tempo de "mvhd" – ver nota de rodapé na seção 3.2.1.

0001 0000		
0000 0000		
0000 0000		
0000 0000		
4000 0000		
01e0 0000	480 pixels (16.16)[20]	largura da imagem
0168 0000	360 pixels (16.16)[21]	altura da imagem

3.2.3.2 Box "mdia" – *"media box"*

Compartimento para informações que definem o tipo de mídia. Seu tamanho (0x20e2), somado ao seu início (0x0101) determina o endereço do próximo compartimento: 0x21e3 ("trak", Seção 3.2.4) (ver Figura 3.8). Observe-se que este compartimento está contido e termina junto com o compartimento-pai: "trak" (de vídeo). Contém três boxes: "mdhd" (Seção 3.2.3.2.1), "hdlr" (Seção 3.2.3.2.2) e "minf" (Seção 3.2.3.2.3).

```
0000101:   00 0020 e26d 6469 6100 0000 206d 6468   .. .mdia... mdh
0000110: 6400 0000 007c 25b0 80c6 65ec cf00 0075  d....|%...e....u
```

Figura 3.8: Box "mdia".

3.2.3.2.1 Box "mdhd" – "*media header box*"

Define informações gerais de mídia, independentes do seu tipo (vídeo ou áudio). Seu tamanho (0x0020), somado ao seu endereço inicial (0x0109), resulta no valor 0x0129, o qual endereça o próximo subcompartimento de "mdia": "hdlr" (Seção3.2.3.2.2) (ver Figura 3.9). Seus valores são descritos na Tabela 3.5.

```
0000109:                         00 0000 206d 6468           ... mdh
0000110: 6400 0000 007c 25b0 80c6 65ec cf00 0075  d....|%...e....u
0000120: 3000 1c36 0f55 c400 00                   0..6.U...
```

Figura 3.9: Box "mdhd".

[20] Alguns valores são definidos por números decimais e serão indicados por (x.y), onde x e y representam a quantidade de bits para a parte inteira e a fração decimal, respectivamente.

[21] Idem.

Tabela 3.5: Box "mdhd".

Valor Hexadecimal	Significado	Descrição
00	versão 0[22]	rótulos de tempo em 32 bits
000000	zeros	campo reservado
7c25 b080	2082844800 s 1970-01-01 00:00:00 (UTC)[23]	rótulo de tempo de criação
c665 eccf	3328568527 s 2009-06-23 02:22:07 (UTC)[24]	rótulo de tempo de modificação
0000 7530	30000 Hz	unidade (ou escala) de tempo
001c 360f	1848847 -> 61,6 s[25]	Duração
55c4	0b(0 10101 01110 00100) -> "21" "14" "04" -> ”und”	linguagem indeterminada, de acordo com a norma ISO 639-2/T
0000	0	pré-definido

3.2.3.2.2 Box"hdlr" – "handler reference box"

Declara os processos usados na trilha, portanto define sua natureza. Seu tamanho (0x002e), somado ao endereço 0x0129, resulta no endereço (0x0157) do próximo compartimento: "minf" (Seção 3.2.3.2.3) (ver Figura 3.10). A Tabela 3.6 descreve os valores lidos. O campo "handler_type" identifica esta trilha como sendo de vídeo.

```
0000129:                        00 0000 2e68 646c          ....hdl
0000130: 7200 0000 006d 686c 7276 6964 6500 0000  r....mhlrvide...
0000140: 0000 0000 0000 0000 000c 5669 6465 6f48  ..........VideoH
0000150: 616e 646c 6572 00                        andler.
```

Figura 3.10: Box “hdlr”.

Tabela 3.6: Box "hdlr".

Valor Hexadecimal	Significado	Descrição
00	0	versão
000000	zeros	campo reservado
6d68 6c72	mhlr	deveriam ser zeros
7669 6465	vide	"*handler_type*": vídeo track

22 A versão 0 implica inteiros de 32 bits para rótulos de tempo; a versão 1 usa 64 bits.
23 Os rótulos de tempo são registrados em segundos decorridos desde 01-01-1904 (à zero hora).
24 Idem.
25 Este valor corresponde ao número de unidades de tempo definido no campo anterior (30000). Portanto, a duração em segundos é determinada pela razão entre esses números.

0000 0000 (3x)	0	campo reservado
001c 360f	1848847 -> 61,6 s[26]	duração do vídeo
0c	<form feed>	sem significado
5669 6465 6f48 616e 646c 6572 00	VideoHandler	sequência de caracteres em UTF-8 terminada em NULL

3.2.3.2.3 Box "minf" – "media information box"

Contém objetos que definem as características da mídia na trilha. O seu tamanho (0x208c), acrescentado ao endereço inicial (0x0157), aponta para o próximo compartimento: "trak" (endereço 0x21e3, ver Seção 3.2.4) – "minf" é o último box desta trilha, neste nível (ver Figura 3.11). Contém três subcompartimentos: "vmhd" (Seção 3.2.3.2.3.1), "dinf" (Seção 3.2.3.2.3.2) e "stbl" (Seção 3.2.3.2.3.3).

```
0000157:                  00 0020 8c6d 696e 6600          .. .minf.
0000160: 0000 1476 6d68 6400 0000 0100 0000 0000  ...vmhd.........
```

Figura 3.11: Box "minf".

3.2.3.2.3.1 Box "vmhd" – "video media header box"

Contém informações gerais, independentes de codificação, para a mídia de vídeo. Segue-se imediatamente ao compartimento-pai ("minf") e é seguido pelo compartimento-irmão "dinf" (Seção3.2.3.2.3.2), cujo endereço (0x0173) é dado pela soma do seu próprio (0x015f) com seu tamanho (0x0014) (Seção Figura 3.12). Seus poucos valores são mostrados na Tabela 3.7.

```
000015f:                                          00                .
0000160: 0000 1476 6d68 6400 0000 0100 0000 0000  ...vmhd.........
0000170: 0000 00                                  ...
```

Figura 3.12: Box "vmhd".

26 Este valor corresponde ao número de unidades de tempo definido no campo anterior (30000).

Tabela 3.7: Box "vmhd".

Valor Hexadecimal	Significado	Descrição
00	0	versão
000001	1	pré-definido
0000	0	modo gráfico, pré-definido
0000 (3x)	{0, 0, 0}	"*opcolor*" (red, green, blue)

3.2.3.2.3.2 Box "dinf" – "data information box"

Contém os boxes "dref" e "url ". Somando-se o seu tamanho (0x0024) ao seu endereço (0x0173), obtém-se o endereço (0x0197) do próximo compartimento: "stbl" (Seção3.2.3.2.3.3). "dref" (*data reference box*) e "url " (observe-se o espaço após o "l") declaram a localização dos dados, conforme mostrado na Figura 3.13. Neste caso, ambos contêm apenas um campo, de três bytes (24 bits), com o mesmo valor: 0x000001. Isto significa que os dados estão no próprio arquivo. Caso contrário, seriam identificados os nomes e/ou URLs de onde os dados seriam encontrados.

```
0000173:        00 0000 2464 696e 6600 0000 1c64     ...$dinf....d
0000180: 7265 6600 0000 0000 0000 0100 0000 0c75  ref............u
0000190: 726c 2000 0000 01                        rl ....
```

Figura 3.13: Box "dinf".

3.2.3.2.3.3 Box "stbl" – "sample table box"

Contém vários compartimentos, que indexam e descrevem os quadros (amostras). Por meio das tabelas contidas neste box, é possível, por exemplo, localizar um quadro específico, determinar seu tipo, descobrindo, por exemplo, se se trata de um quadro intrapredito (ver Seção2.7.1), e determinar seu tamanho, o contêiner onde está localizado e o offset[27] neste contêiner. Segundo a ISO/IEC 14496-12:2008, um quadro consiste em todos os dados associados a um único rótulo de tempo – dois quadros em uma trilha (*track*) não podem compartilhar o mesmo rótulo.

[27] Posição em relação a um ponto de referência.

O tamanho (0x204c), adicionado ao seu endereço (0x0197), aponta para o próximo compartimento: “trak” (endereço 0x21e3, ver Seção3.2.4), conforme mostrado na Figura 3.14.

```
0000197:                  00 0020 4c73 7462 6c00          .. Lstbl.
00001a0: 0000 ac73 7473 6400 0000 0000 0000 0100  ...stsd.........
```

Figura 3.14: Box “stbl”.

3.2.3.2.3.3.1 Box "stsd" – "*sample description box*"

Primeiro subcompartimento de “stbl”, informa detalhes sobre o tipo de codificação usada e outras informações relevantes. Somando-se sua origem (0x019f) ao seu tamanho (0x00ac), obtém-se o endereço (0x024b) de “stts” (Seção 3.2.3.2.3.3.2) (ver Figura 3.15).

```
000019f:                                        00                .
00001a0: 0000 ac73 7473 6400 0000 0000 0000 0100  ...stsd.........
00001b0: 0000 9c61 7663 3100 0000 0000 0000 0100  ...avc1.........
```

Figura 3.15: Box “stsd”.

3.2.3.2.3.3.1.1 Box "avc1"

Este primeiro compartimento de "stsd" e os seguintes são definidos pela cláusula 5.3.4.1 da norma ISO/IEC 144496-15:2010 [9]. Seu conteúdo começa em 0x01b7, conforme pode ser visto na Figura 3.16, e seus campos são descritos na Tabela 3.8.

```
00001b0: 0000 9c61 7663 3100 0000 0000 0000 0100  ...avc1.........
00001c0: 0000 0046 464d 5000 0002 0000 0002 0001  ...FFMP.........
00001d0: e001 6800 4800 0000 4800 0000 0000 0000  ..h.H...H.......
00001e0: 0100 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000200: 0000 18ff ff                             .....
```

Figura 3.16: Box “avc1”.

Tabela 3.8: Box “avc1”.

Valor Hexadecimal	Significado	Descrição
0000 0000 0000	0	campo reservado
0001	1	"*data_reference_index*"
0000	0	
0000	0	
4646 4d50 0000 0200 0000 0200	FFMP ???? ????	segundo a ISO/IEC 14496-12 (de 2008), este campo deveria conter zeros
01e0	480	largura
0168	360	altura
0048 0000	72 dpi (16.16)	
0048 0000	72 dpi (16.16)	
0000 0000	0	
0001	1	"*frame_count*"
0000 0000 (8x)		"*compressorname*"[28]
0018	0x0018 -> imagens coloridas, *no alpha*	"*depth*"
ffff	-1	pré-definido

3.2.3.2.3.3.1.2 Box "avcC"

Seu conteúdo começa em 0x020d, conforme mostrado na Figura 3.17, e seus campos são descritos na Tabela 3.9.

```
0000200: 0000 18ff ff00 0000 3261 7663 4301 42c0  ........2avcC.B.
0000210: 1eff e100 1b67 42c0 1e9a 740f 05ff 2ffe  .....gB...t.../.
0000220: 0012 0010 2000 007d 2000 1d4c 11e2 c5d4  .... ..} ..L....
0000230: 0100 0468 ce3c 80                        ...h.<.
```

Figura 3.17: Box "avcC".

Tabela 3.9: Box "avcC".

Valor Hexadecimal	Significado	Descrição
01	1	"*configurationVersion*"
42	66[29]	"*AVCprofileIndication*"
c0	192[30]	"*profile_compatibility*"
1e	30[31]	"*AVCLevelIndication*"
ffe1	0b1111111111100001, onde:	

[28] Segundo a norma, deveria conter o nome do codificador usado para compressão do vídeo (ver seção 2.3).
[29] Informa o código do perfil (*profile_idc*), conforme definido na ISO/IEC 14496-10 (ISO-IEC, 2010) (ver seção 2.5 - PERFIS E NÍVEIS).
[30] Definido exatamente como o byte que ocorre entre os parâmetros profile_idc e level_idc em um conjunto de parâmetros de sequência, conforme definido na ISO/IEC 14496-10 (ISO-IEC, 2010) (ver seção 2.5).
[31] Informa o código do nível (level_*idc*), conforme definido na ISO/IEC 14496-10 (ISO-IEC, 2010) (ver seção 2.5).

	0b111111 0b11 = 3 => 4 bytes 0b111 0b00001 = 1	 -> reservado (sempre uns) -> "*lengthSizeMinusOne*"[32] -> reservado (sempre uns) -> "*numOfSequenceParameterSets*"[33]
001b	27 bytes	"*sequenceParameterSetLength*"[34]
67 42 c0 1e 9a 74 0f 05 ff 2f fe 00 12 00 10 20 00 00 7d 20 00 1d 4c 11 e2 c5 d4		"*sequenceParameterSetNALUnit*"[35]
01	1	"*numOfPictureParameterSets*"[36]
0004	4 bytes	"*pictureParameterSetLength*"[37]
68 ce 3c 80		"*pictureParameterSetNALUnit*"[38]

3.2.3.2.3.3.1.3 Box "btrt"

Seu conteúdo começa em 0x023f, conforme mostrado na Figura 3.18, e é descrito na Tabela 3.10.

```
0000230: 0100 0468 ce3c 8000 0000 1462 7472 7400  ...h.<.....btrt.
0000240: 005c 9500 0eed e000 067b c8              .\.......{.
```

Figura 3.18: Box "btrt".

Tabela 3.10: Box "btrt".

Valor Hexadecimal	Significado	Descrição
0000 5c95	23701	"*bufferSizeDB*"[39]
000e ede0	978400	"*maxBitrate*"[40]
0006 7bc8	424904	"*avgBitrate*"[41]

[32] Indica se a profundidade das NAL Units de um quadro de vídeo H.264/AVC encontra-se codificada com 1, 2 ou 4 bytes (ver seção 2.6 - SINTAXE DE UM BITREAM H.264/AVC).
[33] Informa a quantidade de conjuntos de elementos de sintaxe constantes no *bitstream* H.264/AVC (ver seção 2.6).
[34] Indica o tamanho em bytes da estrutura de sintaxe *Sequence Parameter Set NAL unit*, conforme definido na ISO/IEC 14496-10 (ver seção 2.6).
[35] Conteúdo de uma SPS NAL unit, conforme definido na ISO/IEC 14496-10 (ver seção 2.6).
[36] Indica o número de conjuntos de parâmetros de imagens (*PPSs*) utilizados para a decodificação de um *bitstream* H.264/AVC (ver seção 2.6).
[37] Indica o tamanho em bytes da *PPS NAL unit*, conforme definido na ISO/IEC 14496-10 (ver seção 2.6).
[38] Conteúdo de uma *PPS NAL unit*, conforme definido na ISO/IEC 14496-10 (ver seção 2.6).
[39] Informa o tamanho do buffer a ser usado na decodificação (ver seção 2.8 - LISTAS DE QUADROS DE REFERÊNCIA).
[40] Determina a taxa máxima de decodificação, em bits/s, para qualquer intervalo de 1 segundo.

3.2.3.2.3.3.2 Box "**stts**" – "decoding time to sample box"

Somando-se sua origem (0x024b) ao seu tamanho (0x0018), obtém-se o endereço (0x0263) de "stss" (Seção3.2.3.2.3.3.3) (ver Figura 3.19).

```
000024b:                                 00 0000 1873                ....s
0000250: 7474 7300 0000 0000 0000 0100 0007 3700  tts...........7.
0000260: 0003 e900 0000 3c73 7473 7300 0000 0000  ......<stss.....
```

Figura 3.19: Box "stts".

Seu conteúdo é mostrado na Tabela 3.11. "sample_count" indica a quantidade de quadros (1847 neste caso) e "sample_delta" a duração de cada um (neste caso, a mesma para todas os quadros: 1001) – a unidade é definida em "mdhd", na unidade de tempo (*timescale*), sendo, neste caso, 30.000 Hz. Portanto, são definidas ~30 (29,97 = 1:(1001:30000)) quadros por segundo.

Tabela 3.11: Box "stts".

Valor Hexadecimal	Significado	Descrição
00	0	versão
000000	0	sinalizadores
0000 0001	1	"*entry_count*"
0000 0737	1847	"*sample_count*"
0000 03e9	1001	"*sample_delta*"[42]

3.2.3.2.3.3.3 Box "**stss**" – "*sync sample box*"

Somando-se sua origem (0x0263) ao seu tamanho (0x003c), obtém-se o endereço (0x029f) de "stsc" (Seção3.2.3.2.3.3.4) (ver Figura 3.20).

[41] Determina a taxa média de decodificação, em bits/s, para o vídeo como um todo (Ver exemplo na Figura 3.1).

[42] Esse valor determina o tempo de duração do quadro, como múltiplo da unidade de tempo (escala) determinada no *box* "mdhd" (seção 3.2.3.2.1). Assim, o produto da quantidade de quadros (1847) pelo seu delta (1001) resulta no valor 1848847. Este último, dividido pela escala (30000 Hz), determina a duração do vídeo em 61,6 s.

```
0000263:        00 0000 3c73 7473 7300 0000 0000     ...<stss.....
0000270: 0000 0b00 0000 0100 0000 fb00 0001 e500  ................
0000280: 0002 9100 0003 5d00 0003 ce00 0004 1600  ......].........
0000290: 0004 dd00 0005 6900 0006 4000 0006 bc    ......i...@.....
```

Figura 3.20: Box "stss".

Seu conteúdo é mostrado na Tabela 3.12. Há 11 entradas (ver "*entry_count*"), e cada uma identifica um quadro de acesso direto (*random access point*), i.e., cada quadro identificado será um ponto de entrada no fluxo do vídeo (ver Seção2.7.1).

Tabela 3.12: Box "stss".

Valor Hexadecimal	Significado	Descrição
00	0	versão
000000	0	sinalizadores
0000 000b	11	"*entry_count*": número de entradas
0000 0001	1	entrada 01
0000 00fb	251	entrada 02
0000 01e5	485	entrada 03
0000 0291	657	entrada 04
0000 035d	861	entrada 05
0000 03ce	974	entrada 06
0000 0416	1046	entrada 07
0000 04dd	1245	entrada 08
0000 0569	1385	entrada 09
0000 0640	1600	entrada 10
0000 06bc	1724	entrada 11

3.2.3.2.3.3.4 Box "stsc" – "sample to chunk box"

Os quadros (amostras) de vídeo (ou áudio) são agrupadas em pedaços, ou nacos (*chunks*), que podem ter (e geralmente têm) diferentes tamanhos. Este compartimento, mostrado na Figura 3.21, forma uma tabela que permite localizar o pedaço que contém determinado quadro, assim como sua posição e descrição. Somando-se seu endereço de origem (0x029f) ao seu tamanho (0x0034), obtém-se o endereço (0x02d3) do próximo compartimento: "stsz" (Seção3.2.3.2.3.3.5).

```
000029f:                                                00                .
00002a0: 0000 3473 7473 6300 0000 0000 0000 0300  ..4stsc.........
00002b0: 0000 0100 0000 0f00 0000 0100 0000 0200  ................
00002c0: 0000 0e00 0000 0100 0000 8400 0000 0c00  ................
00002d0: 0000 01                                  ...
```

Figura 3.21: Box "stsc".

Seu conteúdo é mostrado na Tabela 3.13. O primeiro pedaço (*chunk*) contém 15 quadros; há 130 pedaços com 14 quadros cada; o último pedaço tem 12 quadros.

Tabela 3.13: Box "stsc".

Valor Hexadecimal	Significado	Descrição
00	0	versão
000000	0	sinalizadores
0000 0003	3	"*entry_count*": contador de entradas
0000 0001	1	entrada 1
0000 000f	15	"*samples_per_chunk*"
0000 0001	1	"*sample_description_index*"
0000 0002	2	entrada 2
0000 000e	14	"*samples_per_chunk*"
0000 0001	1	"*sample_description_index*"
0000 0084	132	entrada 3
0000 000c	12	"*samples_per_chunk*"
0000 0001	1	"*sample_description_index*"

3.2.3.2.3.3.5 Box "stsz" – "sample sizes box"

Contabiliza os quadros, i.e., fornece sua quantidade e o tamanho de cada um. Somando-se sua origem (0x02d3) ao seu tamanho (0x1cf0), obtém-se o endereço (0x1fc3) do próximo compartimento: "stco" (Seção3.2.3.2.3.3.6) (ver Figura 3.22).

```
00002d3:        00 001c f073 7473 7a00 0000 0000     ....stsz.....
00002e0: 0000 0000 0007 3700 0030 4c00 0003 6200  ......7..0L...b.
00002f0: 0004 3e00 0004 c600 0005 c000 0005 8200  ..>.............
0000300: 0007 bf00 0006 1f00 0005 db00 0006 7700  ..............w.
```

Figura 3.22: Box "stsz".

Seu conteúdo é parcialmente mostrado na Tabela 3.14. Cada entrada identifica o tamanho do respectivo quadro, em bytes.

Tabela 3.14: Box "`stsz`".

Valor Hexadecimal	Significado	Descrição
00	1	versão
000000	0	sinalizadores
0000 0000	0	"*sample_size*"
0000 0737	1847	"*sample_count*"
0000 304c	12364	tamanho do 1° quadro (bytes)
0000 0362	866	tamanho do 2° quadro (bytes)
0000 043e	1086	tamanho do 3° quadro (bytes)
0000 04c6	1222	tamanho do 4° quadro (bytes)
[...]	[...][43]	[...]

3.2.3.2.3.3.6 Box "`stco`" – "chunk offset box"

Fornece o endereço, a partir do início do arquivo (e não do compartimento), de cada conjunto (*chunk*) de quadros. Somando-se sua origem (0x1fc3) ao seu tamanho (0x0220), obtém-se o endereço (0x21e3) da próxima trilha (Seção3.2.4, "trak") (ver Figura 3.23).

```
0001fc3:        00 0002 2073 7463 6f00 0000 0000     ... stco.....
0001fd0: 0000 8400 003a a900 00d4 d800 012a fc00  .....:.......*..
0001fe0: 016e 7f00 01a8 d400 01f7 c600 0269 7700  .n...........iw.
0001ff0: 02f9 8600 0367 b700 03b0 3200 03f3 6400  .....g....2...d.
0002000: 0439 ea00 04a9 8700 0534 ba00 059b 8d00  .9.......4......
[...]
00021c0: 35aa 0300 369c bd00 371a 7e00 37a9 2300  5...6...7.~.7.#.
00021d0: 383f 0e00 38ce 4b00 396c c200 3a09 ec00  8?..8.K.9l..:...
00021e0: 3a8f 8200 0018 be74 7261 6b00 0000 5c74  :......trak...\t
00021f0: 6b68 6400 0000 01c6 65ec cfc6 65ec cf00  khd.....e...e...
```

Figura 3.23: Box "`stco`".

Seu conteúdo é mostrado na Tabela 3.15, de forma parcial. Observe-se que os endereços, que estão dispostos em ordem crescente e a partir do início do arquivo, estão contidos na área de dados – após o cabeçalho "`mdat`", cujo endereço é 0x3aa1.

[43] Estão sendo mostradas apenas as quatro primeiras entradas, pois o número de quadros (1847) é muito grande – a lista completa é mostrada no apêndice.

Tabela 3.15: Box "stco".

Valor Hexadecimal	Significado	Descrição
00	1	versão
000000	0	sinalizadores
0000 0084	132	"*entry_count*"
0000 3aa9	15017	posição (em bytes) do 1o. conjunto
0000 d4d8	54488	posição (em bytes) do 2o. conjunto
[...]	[...][44]	[...]
003a 09ec	3803452	posição do penúltimo conjunto
003a 8f82	3837826	posição do último conjunto

Todos os endereços armazenados neste recipiente estão listados na Tabela 3.16.

Tabela 3.16: Endereços dos conjuntos de quadros de "stco".

Endereços de início dos *chunks* (box "stco")										
-	1	2	3	4	5	6	7	8	9	10
0	00003aa9	0000d4d8	00012afc	00016e7f	0001a8d4	0001f7c6	00026977	0002f986	000367b7	0003b032
1	0003f364	000439ea	0004a987	000534ba	00059b8d	0005e5b9	000635ae	00069179	000715a6	000780d2
2	0007f1d6	00084aa5	00089380	0008dfd5	00093ce0	00098a46	0009d23e	000a2130	000a910a	000afbb3
3	000b4a18	000ba44b	000bf914	000c5bed	000cb9fd	000d48dc	000d7c9b	000d9fcb	000dc9d0	000dfedf
4	000e2957	000e5883	000e8729	000ebd26	000ef336	000f258b	000f5cd9	00102071	001056be	0010d191
5	00116972	0011fe59	00125593	0012a1ee	0012fe3f	00134e35	0013a41c	0013f359	0014430e	00148e44
6	0014dadb	00153a45	0015f3a1	001632ed	001663e7	0016a18b	00170b58	0017994f	00181ced	00188cc4
7	0019d622	001a79b4	001b2266	001bbfdb	001c6c74	001d259b	001d6428	001d9574	001df6c0	001e4d85
8	001ec4fa	001efda0	001f6ed1	001fbfea	00200dc3	002058cb	0020b1e1	0021067a	002146ac	0021f4dd
9	00228116	00231d8e	0023e8a5	0024cb73	00256522	0025e4c7	00264eb8	0026e705	00277a3d	00287656
10	0028d7e0	00292784	002996ee	002a05d2	002a52f4	002aa802	002b1c41	002b8ce6	002bef36	002c6215
11	002ceb74	002d522e	002d8d47	002df9ee	002ea66b	002fa17d	00301eed	0030c680	0031c150	0032d70d
12	0033a355	00344b3d	0034ee89	0035aa03	00369cbd	00371a7e	0037a923	00383f0e	0038ce4b	00396cc2
13	003a09ec	003a8f82	-	-	-	-	-	-	-	-

3.2.4 Box "trak"

É uma nova trilha, análoga àquela já estudada. Somando-se sua origem (0x21e3) ao seu tamanho (0x18be), obtém-se o endereço (0x3aa1) de "mdat" (Seção3.3) (ver Figura 3.24).

[44] A lista completa de entradas (132) é mostrada na Tabela 3.16.

```
00021e3:        00 0018 be74 7261 6b00 0000 5c74     ....trak...\t
00021f0: 6b68 6400 0000 01c6 65ec cfc6 65ec cf00  khd.....e...e...
0002200: 0000 0200 0000 0000 0091 1600 0000 0000  ................
0002210: 0000 0000 0000 0001 0000 0100 0000 0000  ................
0002220: 0000 0000 0000 0000 0000 0100 0000 0000  ................
0002230: 0000 0000 0000 0000 0000 0040 0000 0000  ...........@....
0002240: 0000 0000 0000 0000 0018 5a6d 6469 6100  ..........Zmdia.
0002250: 0000 206d 6468 6400 0000 00c6 65ec cfc6  .. mdhd.....e...
0002260: 65ec cf00 0056 2200 14d4 0055 c400 0000  e....V"....U....
0002270: 0000 3768 646c 7200 0000 0000 0000 0073  ..7hdlr........s
0002280: 6f75 6e00 0000 0000 0000 0000 0000 0047  oun............G
0002290: 5041 4320 4953 4f20 4175 6469 6f20 4861  PAC ISO Audio Ha
00022a0: 6e64 6c65 7200 0000 17fb 6d69 6e66 0000  ndler.....minf..
```

Figura 3.24: Segundo box "`trak`".

Uma verificação superficial mostra o seguinte: (i) os dois bytes a partir do endereço 0x2217 (dentro do compartimento "`tkhd`" – ver Seção3.2.3.1) contém o valor 0x0100, referente ao volume da trilha, e que conterá esse valor (0x0100) apenas se a trilha for de áudio; (ii) no compartimento "`hdlr`" (endereço 0x2273), o campo "`handler_type`" (endereço 0x227f) possui o valor "`soun`", que confirma que esta é uma trilha de áudio (Seção3.2.3.2.2); (iii) veja-se a expressão "*Audio Handler*" no fim do compartimento "`hdlr`". O estudo de áudio foge ao escopo deste trabalho, portanto esta parte do arquivo não será analisada.

3.3 ANÁLISE DA ÁREA DE DADOS

O box "`mdat`" armazena os dados. O seu tamanho (0x3af058), somado ao seu início (0x3aa1), leva ao final da área de dados – o resultado anterior, 0x3b2af9, corresponde ao início (bytes "00 0000") de um "*trailer*", i.e., um rótulo que marca o fim do arquivo, conforme mostrado na Figura 3.25. Portanto, os dados úteis encontram-se antes dos 3.877.625 bytes – o tamanho do arquivo gravado é de 3.877.672 bytes.

```
0003aa1:   00 3af0 586d 6461 7400 0030 4865 8880   .:.Xmdat..0He..
0003ab0: 2000 5e23 0b80 0083 2b2a 38c9 890e 08d6   .^#....+*8.....
[...]
03b2af0: 0022 2a02 4298 8c94 3800 0000 2f66 7265  ."*.B...8.../fre
03b2b00: 6549 736f 4d65 6469 6120 4669 6c65 2050  eIsoMedia File P
03b2b10: 726f 6475 6365 6420 7769 7468 2047 5041  roduced with GPA
03b2b20: 4320 302e 342e 3400                      C 0.4.4.
```

Figura 3.25: Início e fim da área de dados ("`mdat`").

3.3.1 Localização dos conjuntos (*chunks*) de quadros

A Tabela 3.15, mostrada na seção 3.2.3.2.3.3.6 (box "stco"), lista os endereços de todos os conjuntos de quadros – efetivamente são os endereços do primeiro quadro de cada conjunto. A Figura 3.26 mostra o início dos dois primeiros e dos dois últimos conjuntos.

```
000aa9:                       00 0030 4865 8880           ..0He..
0003ab0: 2000 5e23 0b80 0083 2b2a 38c9 890e 08d6   .^#....+*8.....
[...]
000d4d8:                     0000 0761 419a 0f07           ...aA...
000d4e0: b3dd 65d7 e18b f793 c9af 7b70 8c23 7bb1  ..e.......{p.#{.
[...]
03a09ec:                               0000 07fb               ....
03a09f0: 419a 6231 3067 f137 d777 d7f2 dffd fa52  A.b10g.7.w.....R
[...]
03a8f82:      0000 0941 419a 7038 3067 abbf 754f    ...AA.p80g..uO
03a8f90: a7ff fa25 a6bc 6f88 7e9d efff ba49 7fdb  ...%..o.~....I..
[...]
```

Figura 3.26: Dois primeiros e dois últimos "chunks".

Os quatro bytes iniciais formam um cabeçalho que informa o tamanho do conteúdo do quadro, excluindo-se o próprio cabeçalho. Portanto, o primeiro valor – "0000 3048" (12.360 bytes) indica o tamanho do conteúdo do primeiro quadro. O box "stsz" (seção 3.2.3.2.3.3.5) armazena o tamanho de todos os quadros, que são mostrados parcialmente na Tabela 3.14 e por inteiro no Apêndice. O valor para o primeiro quadro é de 12.364 bytes (valor 0x304c) – que totaliza o tamanho do cabeçalho e do conteúdo do quadro. Somando-se o valor 0x304c ao endereço 0x3aa9, obtém-se o valor 0x6af5, que é o endereço do próximo quadro, e não do próximo conjunto. Portanto, cada conjunto é localizado apenas pelos valores contidos em "stco".

3.3.2 Localização dos quadros

Dado o endereço do primeiro quadro de um conjunto, os demais serão localizados em uma lista encadeada: o tamanho do primeiro quadro informa a posição do próximo e assim por diante. Há duas formas de efetuar as contas: (i) somar ao endereço de início o valor correspondente no box "stsz"; (ii)somar ao endereço de início o valor armazenado no próprio cabeçalho e acrescentar 0x04. A Figura 3.27 apresenta os dois primeiros quadros do primeiro conjunto.

```
0003aa9:                       00 0030 4865 8880           ..0He..
0003ab0: 2000 5e23 0b80 0083 2b2a 38c9 890e 08d6   .^#....+*8.....
[...]
0006af5:             00 0003 5e41 9a01 00b3 c9ae       ...^A......
0006b00: fefd 7ffe fffb dfff ffff ffff ffff ffff  ................
[...]
```

Figura 3.27: Os dois primeiros quadros do primeiro conjunto.

O box "`stsc`" informa que o primeiro conjunto possui 15 entradas. Observe-se, na Figura 3.28, que a 14ª entrada, que começa no endereço 0xb21f, com tamanho de 1731 bytes – os 1727 bytes (0x6bf) informados no próprio cabeçalho mais quatro bytes– leva à 15ª entrada, que começa em 0xb8e2 (= 0xb21f + 0x04 + 0x06bf), com tamanho total de 1765 bytes – igual a 1761 (0x06e1) mais quatro. Entretanto, esta entrada leva ao endereço 0xbfc7 (= 0xb8e2 + 0x04 + 0x06e1), que não corresponde a um quadro.

```
[...]
000b21f:                                         00                .
000b220: 0006 bf41 9a0d 06b3 d57f ebff ffff ffff  ...A............
[...]
000b8e2:      0000 06e1 419a 0e07 33f4 bfff ffff    ....A...3.....
000b8f0: ffff ffff ffff bfff ffff e5af ffff 7cff  ..............|.
[...]
000bfc7:                   de 0400 006c 6962 6661         ....libfa
000bfd0: 6163 2031 2e32 3500 0042 171f ffff ffff  ac 1.25..B......
```

Figura 3.28: Os últimos quadros do primeiro conjunto.

Ainda na Figura 3.28, vê-se que, no endereço 0xbfc7, os quatro primeiros bytes informam um valor – 0xde040000 (= 3.724.804.096 bytes, ou mais de 3552 MiB) – que não pode ser o tamanho de um quadro, pois é maior que o tamanho do próprio arquivo (~3,7 MiB). Além disso, a sequência seguinte é uma expressão que provavelmente refere-se a uma biblioteca ("`libfac 1.25`").

Concluindo, a localização de um quadro depende do conhecimento conjunto de três informações: (i) o endereço do primeiro quadro de um conjunto (contido no recipiente "`stco`"); (ii) a quantidade de quadros por conjunto (que está em "`stsc`"); (iii) o tamanho de cada quadro (obtido no cabeçalho do próprio quadro ou em "`stsz`").

3.4 ANÁLISE DO FLUXO DE BITS (*BITSTREAM*)

Os dados mostrados na Seção 3.3 (ANÁLISE DA ÁREA DE DADOS) referem-se exclusivamente ao *bitstream*. Entretanto, uma sequência de quadros necessita de alguns parâmetros ("*sequence parameter sets*" – SPS – e "*picture parameter sets*" – PPS). Esses parâmetros não estão no *bitstream*, mas nos metadados, especificamente, neste caso, em "`avcC`" (3.2.3.2.3.3.1.2), contido em "`stsd`" (3.2.3.2.3.3.1), nos campos "*sequenceParameterSetNALUnit*" (SPS) e "*pictureParameterSetNALUnit*" (PPS).

Por exemplo, o perfil do vídeo ("*profile*") é indicado no mesmo box("`avcC`"), no campo "*AVCprofileIndication*" , e seu valor é 66 (0x42), que corresponde ao "*baseline profile*" (norma ISO/IEC 14496-10, A.2.1 [9]).

3.4.1 SPS

O SPS em "`avcC`" é mais bem entendido na depuração a seguir, com os valores hexadecimais mostrados na Figura 3.29. O primeiro byte (0x67 = '0110 0111') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é "7" (SPS – *sequence parameter set*).

```
67 42 c0 1e 9a 74 0f 05 ff
2f fe 00 12 00 10 20 00 00
7d 20 00 1d 4c 11 e2 c5 d4
```

Figura 3.29: SPS – *sequence parameter set.*

Os valores do SPS são interpretados (como bits) usando a codificação entrópica definida em 2.10 – e são mostrados na Tabela 3.17, onde a primeira coluna contém os valores em notação binária. Nesta unidade NAL são usadas apenas a codificação com tamanho fixo e a codificação de tamanho variável do tipo Golomb-exponencial.

Tabela 3.17: SPS – *sequence parameter set*.

Bits	Cod.[45]	Valor[46]	Parâmetro[47]
0	f(1)	0	forbidden_zero_bit
11	u(2)	3	nal_ref_idc
00111	u(5)	7	nal_unit_type
01000010	u(8)	66	profile_idc
1	u(1)	1	constraint_set0_flag
1	u(1)	1	constraint_set1_flag
0	u(1)	0	constraint_set2_flag
00000	u(5)	-	reserved_zero_5bits
00011110	u(8)	30	level_idc
1	ue(v)	0	seq_parameter_set_id
00110	ue(v)	5	log2_max_frame_num_minus4
1	ue(v)	0	pic_order_cnt_type
00111	ue(v)	6	log2_max_pic_order_cnt_lsb_minus4 [2]
010	ue(v)	1	num_ref_frames
0	u(1)	0	gaps_in_frame_num_value_allowed_flag
000011110	ue(v)	29	pic_width_in_mbs_minus1
000010111	ue(v)	22	pic_height_in_map_units_minus1
1	u(1)	1	frame_mbs_only_flag
1	u(1)	1	direct_8x8_inference_flag
1	u(1)	1	frame_cropping_flag
1	ue(v)	0	frame_crop_left_offset
1	ue(v)	0	frame_crop_right_offset
1	ue(v)	0	frame_crop_top_offset
00101	ue(v)	4	frame_crop_bottom_offset
1	u(1)	1	vui_parameters_present_flag
1	u(1)	1	aspect_ratio_info_present_flag
11111111	u(8)	255	aspect_ratio_idc
00000000 00001001	u(16)	9	sar_width
00000000 00001000	u(16)	8	sar_height
0	u(1)	0	overscan_info_present_flag
0	u(1)	0	video_signal_type_present_flag
0	u(1)	0	chroma_loc_info_present_flag
1	u(1)	1	timing_info_present_flag
00000000 00000000 00000011 11101001	u(32)	1001	num_units_in_tick
00000000 00000000 11101010 01100000	u(32)	60000	time_scale
1	u(1)	1	fixed_frame_rate_flag
0	u(1)	0	nal_hrd_parameters_present_flag
0	u(1)	0	vcl_hrd_parameters_present_flag
0	u(1)	0	pic_struct_present_flag
1	u(1)	1	bitstream_restriction_flag
1	u(1)	1	motion_vectors_over_pic_boundaries_flag
1	ue(v)	0	max_bytes_per_pic_denom
1	ue(v)	0	max_bits_per_mb_denom
0001011	ue(v)	10	log2_max_mv_length_horizontal
0001011	ue(v)	10	log2_max_mv_length_vertical
1	ue(v)	0	num_reorder_frames
010	ue(v)	1	max_dec_frame_buffering
1	f(1)	'1'	rbsp_stop_one_bit
00	f(1)	'00'	rbsp_alignment_zero_bit

[45] f(x), u(x): tamanho fixo, usando x bits; ue(v): tamanho variável, usando Golomb-exponencial.

[46] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.

[47] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

3.4.2 PPS

O PPS em "avcC" é mostrado na Figura 3.30, e contém apenas quatro bytes (mostrados em notação hexadecimal). O primeiro byte (0x68 = '0110 1000') identifica o tipo de unidade NAL (nal_unit_type) nos cinco bits menos significativos, que neste caso é "8" (PPS – *picture parameter set*).

```
68 ce 3c 80
```

Figura 3.30: PPS (*picture parameter set*).

Analogamente aos valores do SPS, os do PPS também são interpretados (como bits) usando a codificação entrópica definida em 2.10– e são mostrados na Tabela 3.18, onde a primeira coluna contém os valores em notação binária. Nesta unidade NAL são usadas apenas a codificação com tamanho fixo e a codificação de tamanho variável do tipo Golomb-exponencial.

Tabela 3.18: PPS – *picture parameter set*

Bits	Cod.[48]	Valor[49]	Parâmetro[50]
0	f(1)	0	forbidden_zero_bit
11	u(2)	3	nal_ref_idc
01000	u(5)	8	nal_unit_type
1	ue(v)	0	pic_parameter_set_id
1	ue(v)	0	seq_parameter_set_id
0	u(1)	0	entropy_coding_mode_flag
0	u(1)	0	pic_order_present_flag
1	ue(v)	0	num_slice_groups_minus1
1	ue(v)	0	num_ref_idx_l0_active_minus1
1	ue(v)	0	num_ref_idx_l1_active_minus1
0	u(1)	0	weighted_pred_flag
00	u(2)	0	weighted_bipred_idc
1	se(v)	0	pic_init_qp_minus26
1	se(v)	0	pic_init_qs_minus26
1	se(v)	0	chroma_qp_index_offset
1	u(1)	1	deblocking_filter_control_present_flag
0	u(1)	0	constrained_intra_pred_flag
0	u(1)	0	redundant_pic_cnt_present_flag
1	f(1)	'1'	rbsp_stop_one_bit
0000000	f(1)	'0000000'	rbsp_alignment_zero_bit

[48] f(x), u(x): tamanho fixo, usando x bits; ue(v): tamanho variável, usando Golomb-exponencial; se(v): tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3 (ISO-IEC, 2010).
[49] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.
[50] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

3.4.3 Primeiro quadro do primeiro *chunk*

O primeiro quadro (amostra), encontrado no endereço 0x3aa9 (ver Seção 3.3), é uma unidade NAL *slice* (ver Seção 2.6), cujos bytes iniciais são mostrados na Figura 3.31, em notação hexadecimal.

```
0000 3048 6588 8020
005e 230b 8000 832b
[...]
```

Figura 3.31: Início do primeiro quadro.

Os primeiros quatro bytes (“0000 3048”) determinam o tamanho do quadro (12.360 bytes). O byte seguinte (0x65 = '0110 0101') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é “5” (“*IDR picture*”, sem particionamento – ver Tabela 7-1: “*NAL unit type codes*” no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

Uma camada *slice* sem particionamento compõe-se de três partes: (i) o cabeçalho (`slice_header`); (ii) os bits de dados (`slice_data`); (iii) os bits de finalização para alinhamento do último byte (`rbsp_slice_trailing_bits`) (ver seção 7.3.2.8 do capítulo 7 da norma ISO/IEC 14496-10:2010 [9]). A Tabela 3.19 mostra o detalhamento do quadro, a partir do byte 0x65 (i.e., a partir da posição 0x3aad, desconsiderando-se os quatro bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*.

Tabela 3.19: Parâmetros do primeiro quadro (pos. 0x3aad)

Bits	Cod.[51]	Valor[52]	Parâmetro [53]
0	f(1)	0	forbidden_zero_bit
11	u(2)	3	nal_ref_idc
00101	u(5)	5	nal_unit_type (*IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
0001000	ue(v)	7	slice_type (*I slice*)
1	ue(v)	0	pic_parameter_set_id
000000000	u(v)[54]	0	frame_num

[51] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.
[52] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.
[53] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

1	ue(v)	0	idr_pic_id
0000000000	u(v)[55]	0	pic_order_cnt_lsb
0	u(1)	0	no_output_of_prior_pics_flag
0	u(1)	0	long_term_reference_flag
00101	se(v)	-2	slice_qp_delta
1	ue(v)	0	disable_deblocking_filter_idc
1	se(v)	0	slice_alpha_c0_offset_div2
1	se(v)	0	slice_beta_offset_div2
<slice_data>			

3.4.4 Segundo quadro do primeiro *chunk*

O segundo quadro, encontrado no endereço 0x6af5 (ver Seção 3.3), é uma unidade NAL *slice*, cujos bytes iniciais são mostrados na Figura 3.32, em notação hexadecimal.

```
0000 035e 419a 0100
b3c9 aefe fd7f feff
[...]
```

Figura 3.32: Início do segundo quadro.

Os primeiros quatro bytes (“0000 035e”) determinam o tamanho do quadro (862 bytes). O byte seguinte (0x41 = '0100 0001') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é “1” *(“non-IDR picture*”, sem particionamento – ver Tabela 7-1: “*NAL unit type codes* ” no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

A Tabela 3.20 mostra o detalhamento do quadro, a partir do byte 0x41 (i.e., a partir da posição 0x6af9, desconsiderando-se os bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*.

[54]O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.
[55]O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

Tabela 3.20: Parâmetros do segundo quadro (pos. 0x6af9)

Bits	Cod.[56]	Valor[57]	Parâmetro[58]
0	f(1)	0	forbidden_zero_bit
10	u(2)	2	nal_ref_idc
00001	u(5)	1	nal_unit_type (*non-IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
00110	ue(v)	5	slice_type (*P slice*)
1	ue(v)	0	pic_parameter_set_id
000000001	u(v)[59]	1	frame_num
0000000010	u(v)[60]	2	pic_order_cnt_lsb
1	u(1)	1	num_ref_idx_active_override_flag
1	ue(v)	0	num_ref_idx_l0_active_minus1
0	u(1)	0	ref_pic_list_reordering_flag_l0
0	u(1)	0	adaptive_ref_pic_marking_mode_flag
1	se(v)	0	slice_qp_delta
1	ue(v)	0	disable_deblocking_filter_idc
1	se(v)	0	slice_alpha_c0_offset_div2
1	se(v)	0	slice_beta_offset_div2
<slice_data>			

3.4.5 Terceiro quadro do primeiro *chunk*

O terceiro quadro, encontrado no endereço 0x6e57 (ver Seção 3.3), é uma unidade NAL *slice*, cujos bytes iniciais são mostrados na Figura 3.33, em notação hexadecimal.

```
0000 043a 419a 0201
33fa ffff ffc4 cbff
[...]
```

Figura 3.33: Início do terceiro quadro.

Os primeiros quatro bytes (“0000 043a”) determinam o tamanho do quadro (1082 bytes). O byte seguinte (0x41 = '0100 0001') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é “1” *(“non-IDR picture*”, sem particionamento – ver Tabela 7-1: “*NAL unit type codes* ” no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

---

[56] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.

[57] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.

[58] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

[59] O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.

[60] O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

A Tabela 3.21 mostra o detalhamento do quadro, a partir do byte 0x41 (i.e., a partir da posição 0x6e5b, desconsiderando-se os bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*. Os parâmetros após `pic_order_cnt_lsb` não são mostrados, por serem iguais aos do segundo quadro.

Tabela 3.21: Parâmetros do terceiro quadro (pos. 0x6e5b)

Bits	Cod.[61]	Valor[62]	Parâmetro[63]
0	f(1)	0	forbidden_zero_bit
10	u(2)	2	nal_ref_idc
00001	u(5)	1	nal_unit_type (*non-IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
00110	ue(v)	5	slice_type (*P slice*)
1	ue(v)	0	pic_parameter_set_id
000000010	u(v)[64]	2	frame_num
0000000100	u(v)[65]	4	pic_order_cnt_lsb
[...]			
<slice_data>			

3.4.6 Primeiro quadro do segundo *chunk*

O primeiro quadro do segundo *chunk*, encontrado no endereço 0xd4d8 (ver Seção 3.3), é uma unidade NAL *slice*, cujos bytes iniciais são mostrados na Figura 3.34, em notação hexadecimal.

```
0000 0761 419a 0f07
b3dd 65d7 e18b f793
[...]
```

Figura 3.34: Início do primeiro quadro do segundo *chunk*.

Os primeiros quatro bytes (“0000 0761”) determinam o tamanho do quadro (1889 bytes). O byte seguinte (0x41 = '0100 0001') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é “1” *(“non-IDR*

[61] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.

[62] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.

[63] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

[64] O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.

[65] O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

picture", sem particionamento – ver Tabela 7-1: "*NAL unit type codes* " no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

A Tabela 3.22 mostra o detalhamento do quadro, a partir do byte 0x41 (i.e., a partir da posição 0xd4dc, desconsiderando-se os bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*. Observe-se que este é o 16º quadro do vídeo, pois o primeiro *chunk* tem 15 quadros (ver Seção 3.2.3.2.3.3.5 – box "`stsz`"). Os parâmetros após `pic_order_cnt_lsb` não são mostrados, por serem iguais aos do segundo e do terceiro quadros.

Tabela 3.22: Parâmetros do primeiro quadro do segundo *chunk* (pos. 0xd4dc)

Bits	Cod.[66]	Valor[67]	Parâmetro[68]
0	f(1)	0	forbidden_zero_bit
10	u(2)	2	nal_ref_idc
00001	u(5)	1	nal_unit_type (*non-IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
00110	ue(v)	5	slice_type (*P slice*)
1	ue(v)	0	pic_parameter_set_id
000001111	u(v)[69]	15	frame_num
0000011110	u(v)[70]	30	pic_order_cnt_lsb
[...]			
<slice_data>			

3.4.7 250º quadro

O 250º quadro, encontrado no endereço 0x06c331, corresponde ao último quadro do 17º *chunk*, e seus bytes iniciais são mostrados na Figura 3.35, em notação hexadecimal.

```
0000 0528 419a f97c
b3d6 a097 5249 e4e8
[...]
```

Figura 3.35: Início do 250º quadro.

[66] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.

[67] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.

[68] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

[69] O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.

[70] O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

Os primeiros quatro bytes (“0000 0528”) determinam o tamanho do quadro (1320 bytes). O byte seguinte (0x41 = '0100 0001') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é “1” *(“non-IDR picture*”, sem particionamento – ver Tabela 7-1: “*NAL unit type codes* ” no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

A Tabela 3.23 mostra o detalhamento do quadro, a partir do byte 0x41 (i.e., a partir da posição 0x06c335, desconsiderando-se os bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*. Os parâmetros após `pic_order_cnt_lsb` são iguais aos dos quadros do mesmo tipo (`nal_unit_type` = 1) e não são mostrados.

Tabela 3.23: Parâmetros do 250º quadro (pos. 0x06c335)

Bits	Cod.[71]	Valor[72]	Parâmetro[73]
0	f(1)	0	forbidden_zero_bit
10	u(2)	2	nal_ref_idc
00001	u(5)	1	nal_unit_type (*non-IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
00110	ue(v)	5	slice_type (*P slice*)
1	ue(v)	0	pic_parameter_set_id
011111001	u(v)[74]	249	frame_num
0111110010	u(v)[75]	498	pic_order_cnt_lsb
[...]			
<slice_data>			

3.4.8 251º quadro

O 251º quadro (1º quadro do 18º *chunk)*, encontrado no endereço 0x06c85d, é uma unidade NAL *slice*, cujos bytes iniciais são mostrados na Figura 3.36, em notação hexadecimal.

[71] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.
[72] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.
[73] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.
[74] O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.
[75] O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

```
0000 3631 6588 8010
001f 88c3 e000 20ef
[...]
```

Figura 3.36: Início do 251º quadro.

Os primeiros quatro bytes (“0000 3631”) determinam o tamanho do quadro (13.873 bytes). O byte seguinte (0x65 = '0110 0101') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é “5” (“*IDR picture*”, sem particionamento – ver Tabela 7-1: “*NAL unit type codes* ” no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

A Tabela 3.24 mostra o detalhamento do quadro, a partir do byte 0x65 (i.e., a partir da posição 0x06c861, desconsiderando-se os bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*.

Tabela 3.24: Parâmetros do 251º quadro (pos. 0x06c861)

Bits	Cod.[76]	Valor[77]	Parâmetro[78]
0	f(1)	0	forbidden_zero_bit
11	u(2)	3	nal_ref_idc
00101	u(5)	5	nal_unit_type (*IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
0001000	ue(v)	7	slice_type (*I slice*)
1	ue(v)	0	pic_parameter_set_id
000000000	u(v)[79]	0	frame_num
010	ue(v)	1	idr_pic_id
0000000000	u(v)[80]	0	pic_order_cnt_lsb
0	u(1)	0	no_output_of_prior_pics_flag
0	u(1)	0	long_term_reference_flag
00111	se(v)	-3	slice_qp_delta
1	ue(v)	0	disable_deblocking_filter_idc
1	se(v)	0	slice_alpha_c0_offset_div2
1	se(v)	0	slice_beta_offset_div2
<slice_data>			

[76] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.

[77] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.

[78] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.

[79] O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do PPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.

[80] O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do PPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

3.4.9 1724º quadro

O 1724º quadro, encontrado no endereço 0x35aa03, é uma unidade NAL *slice*, cujos bytes iniciais são mostrados na Figura 3.37, em notação hexadecimal.

```
0000 53ee 6588 8005
8001 be38 8f78 ff06
[...]
```

Figura 3.37: Início do 1724º quadro.

Os primeiros quatro bytes ("0000 53ee") determinam o tamanho do quadro (21.486 bytes). O byte seguinte (0x65 = '0110 0101') identifica o tipo de unidade NAL (`nal_unit_type`) nos cinco bits menos significativos, que neste caso é "5" ("*IDR picture*", sem particionamento – ver Tabela 7-1: "*NAL unit type codes* " no capítulo 7 da norma ISO/IEC 14496-10:2010 [9]).

A

Tabela 3.25 mostra o detalhamento do quadro, a partir do byte 0x65 (i.e., a partir da posição 0x35aa07, desconsiderando-se os bytes do tamanho, que não fazem parte do *bitstream*) para o cabeçalho do *slice*.

Tabela 3.25: Parâmetros do 1724º quadro (pos. 0x35aa07)

Bits	Cod.[81]	Valor[82]	Parâmetro[83]
0	f(1)	0	forbidden_zero_bit
11	u(2)	3	nal_ref_idc
00101	u(5)	5	nal_unit_type (*IDR picture*)
<slice_header>			
1	ue(v)	0	first_mb_in_slice
0001000	ue(v)	7	slice_type (*I slice*)
1	ue(v)	0	pic_parameter_set_id
000000000	u(v)[84]	0	frame_num
0001011	ue(v)	10	idr_pic_id
0000000000	u(v)[85]	0	pic_order_cnt_lsb
0	u(1)	0	no_output_of_prior_pics_flag

[81] `f(x)`, `u(x)`: tamanho fixo, usando `x` bits; `ue(v)`: tamanho variável, usando Golomb-exponencial; `se(v)`: tamanho variável, usando Golomb-exponencial e mapeamento conforme a norma ISO/IEC 14496-10:2010, capítulo 9, tabela 9-3; `u(v)`: tamanho fixo, com `v` dependendo do contexto.
[82] Os valores desta coluna são decimais, exceto quando entre aspas simples ('), que indicam a notação binária.
[83] São mostrados apenas os parâmetros encontrados neste arquivo. A interpretação destes e dos demais é mostrada na norma ISO/IEC 14496-10.
[84] O valor de `v` é determinado pelo parâmetro `log2_max_frame_num_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 5`, portanto, `v = x + 4= 9`.
[85] O valor de `v` é determinado pelo parâmetro `log2_max_pic_order_cnt_lsb_minus4` (seja `x`) do SPS, acrescido de quatro bits. `X = 6`, portanto, `v = x + 4= 10`.

0	u(1)	0	long_term_reference_flag
00110	se(v)	3	slice_qp_delta
1	ue(v)	0	disable_deblocking_filter_idc
1	se(v)	0	slice_alpha_c0_offset_div2
1	se(v)	0	slice_beta_offset_div2
<slice_data>			

3.5 INTERPRETAÇÃO

O vídeo analisado compõe-se de 11 sequências de vídeo, identificadas no box "stss" (ver Seção 3.2.3.2.3.3.3). Cada uma contém uma *slice* IDR do tipo I (*Intra-coded slice*) e várias *slices* não-IDR do tipo P (*predictive slice*). O primeiro quadro de cada sequência, do tipo I, é um ponto de acesso randômico (ver Seções 2.7.1 e 2.7.2).

Os quadros do tipo I possuem uma ordem, definida no parâmetro idr_pic_id. No primeiro quadro o valor é "0" (zero), conforme pode ser visto na Tabela 3.19, e no 1724º quadro, que corresponde ao primeiro da última sequência, o valor é "10" (dez), conforme mostrado na

Tabela 3.25. Em cada sequência, os quadros são ordenados pelo parâmetro frame_num. Observe-se que, na primeira sequência, o primeiro quadro possui o valor "0" (ver Tabela 3.19), o segundo o valor "1" (ver Tabela 3.20), o terceiro o valor "2" (ver Tabela 3.21) e o 16º (1º quadro do segundo *chunk*) possui o valor "15" (ver Tabela 3.22).

A análise dos quadros (box "mdat") e dos metadados (box "moov") permite concluir que, estando ausentes os metadados, não é possível recuperar nenhum quadro. Isso ocorre porque alguns parâmetros específicos para cada quadro são dependentes de valores que se encontram nos metadados, como, por exemplo, os parâmetros "log2_max_frame_num_minus4" e "log2_max_pic_order_cnt_lsb_minus4", localizados no box "avcC", especificamente no campo SPS (ver Seção 3.2.3.2.3.3.1.2 e Seções 3.4.1 e 3.4.2). Além disso, a codificação entrópica é estritamente sequencial (ver Seção 2.10), i.e., cada parâmetro somente será decodificado após a decodificação do anterior. Assim, não se conseguindo decodificar corretamente parâmetros como os dois citados, a decodificação de todos os dados subsequentes fica inviabilizada.

Essa conclusão corrobora e complementa diversos artigos que abordam questões sobre a transmissão de *bitstreams* H.264/AVC por redes sem garantia de entrega dos dados [53] [54] [55]. Com frequência, esses trabalhos mencionam que certos tipos de metadados, como os contidos nos campos SPS e PPS, têm destacada importância no processo de

decodificação dos quadros que compõem um vídeo e devem, por padrão, serem transmitidos por meio de um canal seguro. Explicitações sobre qual o papel desses dados na reconstrução, assim como sobre quais as reais implicações de sua ausência, no entanto, não são apresentadas.

4 RECUPERAÇÃO DE QUADROS

A tentativa de recuperação de arquivos apagados é procedimento padrão em um exame pericial de mídias digitais. Essa recriação é muitas vezes possível porque, por questões de eficiência, a maioria dos sistemas de arquivos (ver Seção 2.12.3) não realmente apaga os dados, reescrevendo sobre a área por eles ocupada, mas sim apenas sinaliza que o arquivo foi "apagado" e disponibiliza a área para reuso. Até que essa área seja reutilizada, os dados encontram-se lá, apesar de não mais estarem diretamente acessíveis.

Esses arquivos "apagados" podem ser recuperados tanto por meio dos metadados do sistema de arquivos quanto por *data carving* (ver Seção 2.12.6). Em ambos os casos, contudo, é comum que os arquivos sejam recriados somente de forma parcial, corrompidos. Isso se dá, por exemplo, porque os dados encontravam-se armazenados de forma fragmentada no disco, dificultando ou impedindo os procedimentos de recuperação, ou mesmo porque parte dos dados acabou sendo sobrescrita por novo arquivo.

Este capítulo analisa e interpreta os efeitos desses tipos de corrupções, as quais foram intencionalmente causadas ao arquivo de vídeo original, `j10_3786k.mp4`. A intenção, neste caso, é determinar qual a condição mínima do arquivo que permite a visualização ou recuperação de seus quadros (imagens). Obviamente, conforme demonstrado no Capítulo 3, a parte do arquivo que contém o compartimento "`moov`" deve estar presente.

A forma escolhida para corromper o arquivo original, `j10_3786k.mp4`, foi a escrita de zeros binários por sobre parte(s) dos seus dados (box "`mdat`"). Salienta-se, contudo, que a escrita de quaisquer outros tipos de dados (e.g. bits aleatórios) simularia as situações reais de forma igualmente fiel. Em todos os casos, os metadados (box "`moov`") foram mantidos íntegros, uma vez que ficou provado que, estando ausentes, a recuperação é inviável (ver Seção 3.5).

Na Seção 4.1.1, removeu-se toda a parte de dados, com exceção do primeiro quadro. Na Seção 4.1.2, a parte corrompida foi o box "`trak`" de áudio. Na Seção 4.1.3, corrompeu-se um trecho intermediário da parte dos dados. Por fim, na Seção 4.1.4, o arquivo de vídeo foi propositalmente fragmentado e teve dois de seus trechos de dados zerados. Esse último foi o modelo de corrupção escolhido para a demonstração da técnica

de recuperação de quadros por representar fielmente os casos reais encontrados no dia-a-dia pericial: arquivos corrompidos (com perda de trechos de dados) e armazenados de forma fragmentada. A técnica, no entanto, não se restringe a arquivos com esse tipo de corrupção, sendo válida para quaisquer tipos de corrupção que mantenham os metadados (box "`moov`") e ao menos um quadro na área de dados (box "`mdat`"). Apresenta-se, ao final dessa Seção 4.1.4, um algoritmo que permite localizar todos os quadros não sobrescritos de um vídeo.

4.1.1 Arquivo contendo somente um quadro

O primeiro teste envolveu a remoção de toda a parte de dados, exceto o primeiro quadro (até o endereço 0x6af5), produzindo um arquivo com 27.381 bytes (denominado `j10_0x6af5.mp4`) (ver Figura 4.1).

```
$ dd if=j10_3786k.mp4 bs=1 count=27381 of=j10_0x6af5.mp4
```

Figura 4.1: Geração de um arquivo para testes.

Os aplicativos testados, Totem (versão 0.5.1), Media Player Classic (versão 6.4.9), Windows Media Player (versão 12.0) e VLC Media Player (versão 1.1.0), exibiram corretamente um quadro, sem quaisquer erros. Conforme mostrado na Figura 4.2, o comando utilizado na ferramenta `ffmpeg` (versão 0.5.1), produziu apenas uma imagem JPEG (o primeiro quadro do vídeo), como esperado, apesar de terem sido solicitados 99 quadros (opção "`-vframes 99`").

```
$ ffmpeg -re -i j10_0x6af5.mp4 -vframes 99 j10_0x6af5-%02d.jpeg
$ ls -lgG *.jpeg
-rw-r--r-- 1 15064 2011-12-14 12:14 j10_0x6af5-01.jpeg
```

Figura 4.2: Recuperação de apenas um quadro do arquivo de testes.

4.1.2 Arquivo sem o "`trak`" de áudio

Foi criado um arquivo, com o mesmo tamanho do original (3.877.672 bytes), gravando zeros binários no compartimento "`trak`" de áudio (ver Seção 3.2.4) – entre os

bytes 8675 (0x21e3) e 15009 (0x3aa1). Os passos da criação do arquivo são mostrados Figura 4.3.

```
$ dd if=j10_3786k.mp4 bs=1 count=8675 of=j10_sem_audio.mp4
$ dd if=/dev/zero bs=1 count=6334 >> j10_sem_audio.mp4
$ dd if=j10_3786k.mp4 bs=1 skip=15009 >> j10_sem_audio.mp4
```

Figura 4.3: Criação de um arquivo MPEG-4, sem áudio.

O vídeo foi mostrado correta e integralmente pelos aplicativos testados (Totem, Media Player Classic, Windows Media Player e VLC Media Player), sem áudio, obviamente.

4.1.3 Arquivo contíguo com um trecho de dados zerado

Foram zerados (com zeros binários) 2660 setores após o segundo quadro – o arquivo foi mantido íntegro nos 56 setores iniciais e a partir do 2716º setor (até o final). Os comandos contidos na Figura 4.4 mostram a criação do arquivo, denominado `j10_frag.mp4`, e possuindo o mesmo tamanho do arquivo original. A Figura 4.5, por sua vez, ilustra o conteúdo desse arquivo.

```
$ dd if=j10_3786k.mp4 bs=512 count=56 of=j10_frag.mp4
$ dd if=/dev/zero bs=512 count=2660 >> j10_frag.mp4
$ dd if=j10_3786k.mp4 bs=512 skip=2716 >> j10_frag.mp4
$ ls -lgG j10_*
-rw-r--r-- 1 3877672 2011-10-03 17:31 j10_3786k.mp4
-rw-r--r-- 1 3877672 2011-12-08 16:18 j10_frag.mp4
```

Figura 4.4: Criação de um arquivo MPEG-4, parcialmente corrompido.

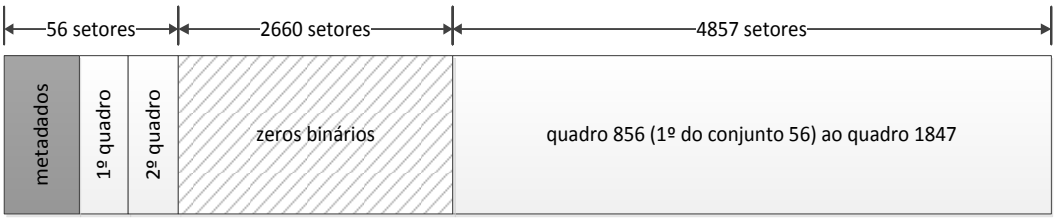

Figura 4.5: Ilustração do conteúdo do arquivo j10_frag.mp4 (sem escala).

O trecho de setores zerados pode ser arbitrário, mas foi escolhido de forma que o primeiro setor após os zeros contivesse o primeiro quadro de um conjunto (no caso, o quadro 856,1º do conjunto 62).

Com os aplicativos testados (Totem, Media Player Classic, Windows Media Player e VLC Media Player), o vídeo não conseguiu ser decodificado, tendo sido exibidos somente os dois primeiros quadros. Entretanto, a ferramenta `ffmpeg` (versão 0.5.1) conseguiu extrair quadros. A figura seguinte mostra, inicialmente, a extração dos 1847 quadros do arquivo original (íntegro), gravando-os em arquivos JPEG; em seguida, foram extraídos todos os quadros possíveis do arquivo corrompido, tendo sido extraídos 1846 imagens JPEG.

```
$ mkdir img-ok/; cd img-ok/
$ ffmpeg -re -i ../j10_3786k.mp4 -vframes 2000 j10_ok_%04d.jpeg
$ ls | wc -l
1847
$ mkdir ../img-rec/; cd ../img-rec/
$ ffmpeg -re -i ../j10_frag.mp4 -vframes 2000 j10_rec_%04d.jpeg
$ ls | wc -l
1846
```

Figura 4.6: Extração de quadros dos arquivos de vídeo.

A tabela seguinte compara os *hashes* MD5 (ver Seção 2.12.4) de algumas imagens extraídas. Os dois primeiros são coincidentes, o que é esperado, pois o arquivo de vídeo foi copiado até o segundo quadro, tendo sido zerado a partir daí. O terceiro quadro recuperado do arquivo corrompido não corresponde ao do original, assim como todos até o 859º (inclusive), do arquivo corrompido.

Tabela 4.1: Fragmentos de um arquivo de vídeo.

Arquivo original		Arquivo corrompido	
j10_ok_0001	aa6dbc7ac87e5904f6521cad194255ad	j10_rec_0001	aa6dbc7ac87e5904f6521cad194255ad
j10_ok_0002	f3215a84b6bf528817dd8e7d93b4a7c0	j10_rec_0002	f3215a84b6bf528817dd8e7d93b4a7c0
j10_ok_0003	c460678061e79ec75ac50b929ab49304	j10_rec_0003	84a0148f92ccad15deda5218fbd5d17e
j10_ok_0004	2585bcc43e5def14001d064e5ca0ceae	j10_rec-0004	224b69497003da24a7167172b3574fe8
j10_ok_0005	2acb91f47dc3348b1eceb4a70c76ae3c	j10_rec-0005	fbc410ab04fb276dea282569127f9014
j10_ok_0006	1004ae50621a9055b40e79f43d871fb1	j10_rec-0006	25cab69c82ceb1fa1773950cf68931c0

j10_ok_0007	5790e6eb3503ae110d6c18d7f0559a0a	j10_rec-0007	b5ddcf952d48f6c31a37a88fc4b5e0fc
j10_ok_0008	46006b3048ed1e5785c2802cd7857df3	j10_rec-0008	afa0aac2d3d92d0800c1a60a8e1a83f9
j10_ok_0009	5b4898db0586784b8e6985c0643511cc	j10_rec-0009	2a2c046bb175417dd6e9bd04753bafd7
j10_ok_0010	785b0a9212f8da54b69686098d9c563a	j10_rec-0010	7a69ea7a7d0b0abe2d5c846e71028476
j10_ok_0011	4ba3563bc612b6cd633e42969ba9051f	j10_rec-0011	1fc2e0954e4634b06423622c38d97234
j10_ok_0012	7ca377d97fda37d0d14e533a8f7b9a9f	j10_rec-0012	aea32e5b8756131c39f2b839ca776ed2
j10_ok_0013	90381da3b71e4a96574ec5ff75f9fbf9	j10_rec-0013	aea32e5b8756131c39f2b839ca776ed2
[...]	-	[...]	-
j10_ok_0855	dddea49e806085c23bf40bffcae05405	j10_rec-0855	aea32e5b8756131c39f2b839ca776ed2
j10_ok_0856	4addd058effede412bbdfec94ad4ab86	j10_rec-0856	8d64dfa463e0f546f8bd2f129ac75bb6
j10_ok_0857	ec155698179ebc2be0a11adf284c35ea	j10_rec-0857	4ad505fd727c884dc3ee5fd005b6fda3
j10_ok_0858	71a7ab56e7f3e9de8a15ce743a939235	j10_rec-0858	9d0d9669d90fd89400071bd2bfde1020
j10_ok_0859	d27f731f29cfdfb3ddd247f7d5efad35	j10_rec-0859	635cff43bdddfd8de9164a2f7889129f
j10_ok_0860	6b473084fdef37df50210ee9642a98c2	j10_rec-0860	4da6b0b3f65e5bc7aa938f861fea741b
j10_ok_0861	4da6b0b3f65e5bc7aa938f861fea741b	j10_rec-0861	5c2d2d4364e6fec23d598089c86ed1c5
j10_ok_0862	5c2d2d4364e6fec23d598089c86ed1c5	j10_rec-0862	d46e10218feb82f0f51dda6db456f99a
[...]	-	[...]	-
j10_ok_1845	fe8cf0c84c3ffad90823d5a6d24db0ef	j10_rec-1845	7b9b6240c6c82ca303b11c9f50944b3f
j10_ok_1846	7b9b6240c6c82ca303b11c9f50944b3f	j10_rec-1846	6c050a631c9ad71e67912c57498e2d3b
j10_ok_1847	6c050a631c9ad71e67912c57498e2d3b	nihil	-

Os *hashes* dos quadros recuperados 12 até 855, inclusive, são iguais. Isso decorre da predição – uma vez que os setores seguintes ao 56º estão zerados, os resíduos tendem a zero, portanto, em algum momento os quadros irão se repetir (neste caso, a partir do 12º quadro).

O quadro "perdido", não recuperado, foi o 857º (do vídeo original), que corresponde à última imagem de uma sequência e que, por causa disso, não pode ser reconstruída. Portanto, o 857º quadro recuperado corresponde ao 858º quadro do vídeo original.

O *hash* MD5 do 860º quadro recuperado é igual ao do 861º quadro do vídeo original. Este quadro corresponde à quinta entrada de acesso direto listada no box "`stss`", como mostrado na Tabela 3.12. Desse ponto, até o fim, os *hashes* coincidem – levando-se em conta que o 861º recuperado corresponde ao 862º original e assim sucessivamente.

4.1.4 Arquivo fragmentado e com dois trechos de dados ausentes

Nesta Seção, o teste envolve a gravação, em uma mídia digital, dos dados de um arquivo de vídeo, a qual foi feita propositalmente de forma fragmentada e incompleta (Seção 4.1.4.1). Em seguida, tentou-se recuperar o arquivo por meio de um procedimento de *data carving* (Seção 4.1.4.2). Esse arquivo recuperado mostrou-se incompleto/ corrompido (Seção 4.1.4.3), o que ensejou a busca por dados (quadros) que dele faziam parte, mas que estavam armazenados em área diversa da mídia (Seções 4.1.4.4 a 4.1.4.6). De posse dos dados encontrados, o arquivo de vídeo foi, então, reconstruído (Seção 4.1.4.7). Um algoritmo para localização dos quadros é apresentado, sintetizando os passos seguidos nesse exemplo (Seção 4.1.4.8). Por fim, apresenta-se uma análise do resultado desse procedimento de reconstrução do arquivo de vídeo e recuperação dos quadros que o integram (Seção 4.1.4.9).

As Seções 4.1.4.4 a 4.1.4.6, conforme dito, tratam da localização dos quadros do arquivo corrompido e seguem, de forma geral, o algoritmo a ser apresentado na Seção 4.1.4.8. De forma simplificada, o que é feito nessas seções (e posteriormente sistematizado pelo algoritmo) é a checagem de validade de informações contidas nos metadados (endereços e tamanhos de amostras), de forma a se descobrir se todos os quadros se encontram nos endereços informados. No cenário mais favorável, em que o arquivo não estivesse corrompido, a checagem comprovaria a sua integridade. Na presente situação, no entanto, o arquivo encontra-se propositalmente corrompido (ver Seção 4.1.4.1) e, portanto, nem todos os quadros podem ser localizados dessa forma. Parte-se, então, para uma leitura de todos os dados da mídia onde o vídeo se encontra salvo no intuito de se localizar os quadros faltosos. Nesse caso, o endereço dos quadros não é levado em consideração. Os parâmetros de busca utilizados são o tamanho dos quadros e um offset específico (resto da divisão inteira do endereço do quadro pelo tamanho do bloco utilizado pelo sistema de arquivos).

4.1.4.1 Gravação do arquivo na mídia, de forma fragmentada e incompleta

O vídeo contém 947 blocos, cada um com 4096 bytes (3.877.672 bytes correspondem a 946 * 4096 + 2856). Esses blocos foram agrupados em fragmentos, como mostrado na Tabela 4.2,e gravados em um volume de 1024 MiB contendo diversos dados

pré-existentes, simulando uma situação real de corrupção. A Figura 4.7 mostra, graficamente, a forma de gravação apresentada na tabela.

Tabela 4.2: Fragmentos de um arquivo de vídeo.

Fragmento	Blocos (com 4 KiB) do vídeo	Localização (blocos de 4 KiB) no volume
01	1 - 60	104100 - 104159
02	61 - 180	110000 - 110119
03	181 - 360	nihil
04	361 - 600	160000 - 160239
05	601 - 900	nihil
06	901 - 947	250000 - 250046

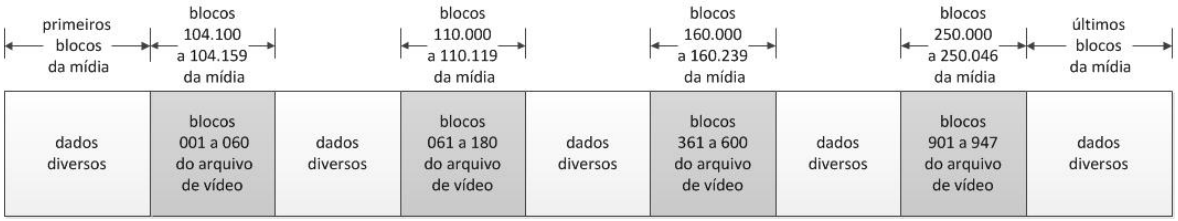

Figura 4.7: Gravação dos dados do arquivo de vídeo na mídia de armazenamento (sem escala).

A cópia dos dados (sem nome de arquivo e sem os outros metadados para o sistema de arquivos) foi efetuada com o comando `dd`, conforme mostrado na Figura 4.8. O volume em questão está formatado com o sistema de arquivos FAT32, mas isso não é relevante, pois não serão usadas as estruturas e nem ferramentas daquele sistema para a recuperação dos dados. A única informação importante é que foram usados blocos de 4096 bytes[86] [49]. Por fim, foi feita uma cópia do volume produzindo o arquivo-imagem `imagem-1g.dd` (Ver Seção 2.12.5).

[86] A maioria dos sistemas de arquivos (NTFS, Ext3, XFS...) usa blocos de 4096 bytes, por padrão. O FAT32 usa blocos de 4096 bytes quando o volume tem entre 32 MiB e 8192 MiB. Ver Seção 2.12.2.

```
$ dd if=j10_3786k.mp4 bs=4096 count=60 of=/dev/sdd1 seek=104100
$ dd if=j10_3786k.mp4 bs=4096 count=120  skip=60 of=/dev/sdd1 seek=110000
$ dd if=j10_3786k.mp4 bs=4096 count=240 skip=360 of=/dev/sdd1 seek=160000
$ dd if=j10_3786k.mp4 bs=4096 skip=900 of=/dev/sdd1 seek=250000
```

Figura 4.8: Cópia dos dados do arquivo de vídeo para outro volume.

4.1.4.2 Tentativa de recuperação do arquivo por *data carving*

Nessa imagem foi executado um *data carving*[87] usando o aplicativo `foremost`[88] (versão 1.5.6), que identificou e recuperou sete arquivos MPEG-4 e outros arquivos não relevantes[89], conforme mostrado na Figura 4.9

```
$ cat audit.txt
Foremost version 1.5.6 [...]
Foremost started at Tue Dec 13 10:50:41 2011
Invocation: foremost imagem-1g.dd
Output directory: [...]/output
Configuration file: /etc/foremost.conf
-------------------------------------------------------------
File: imagem-1g.dd
Start: Tue Dec 13 10:50:41 2011
Length: 1 GB (1073741824 bytes)

Num      Name (bs=512)         Size      File Offset     Comment

0:      00832800.mp4          19 MB       426393600
1:      02570380.mp4          19 MB      1316034560
2:      02661692.mp4         354 KB      1362786304
3:      02738284.mp4          19 MB      1402001408
4:      02741916.mp4          19 MB      1403860992
5:      02747660.mp4          19 MB      1406801920
6:      02751356.mp4          19 MB      1408694272
[...]
Finish: Tue Dec 13 10:50:48 2011
```

Figura 4.9: Resultados do *data carving* com o foremost.

O primeiro arquivo de vídeo MPEG-4 recuperado foi encontrado no setor 832.800 (o `foremost` nomeia o arquivo usando o número do setor onde foi encontrado), tendo sido "recuperados" 19 MiB (o padrão do `foremost`). O setor 832.800 corresponde ao

[87] Processo de recuperação de dados a partir de seus cabeçalhos, rodapés e estruturas de dados internas. Ver Seção 2.12.6.

[88] Ferramenta de console para realização de *data carving*, gratuita e com código fonte aberto, disponível em http://foremost.sourceforge.net/. Ver Seção 2.12.7.

[89] O volume que recebeu o arquivo de vídeo modificado estava propositalmente não vazio, no intuito de se simular uma situação real. Essa é a explicação para terem sido recuperados diversos outros arquivos que não só o do vídeo de interesse.

bloco 104.100 (considerando-se um tamanho de bloco de 4096 bytes – ver Seção 2.12.2). A Figura 4.10 mostra o início do arquivo recuperado, cujo box "mvhd" (Seção 3.2.1) informa o mesmo valor para os rótulos de criação e de modificação, "c665 eccf", idêntico ao do vídeo procurado (data: 2009-06-23, horário: 02:22:07).

```
0000000: 0000 0014 6674 7970 6973 6f6d 0000 0001  ....ftypisom....
0000010: 6973 6f6d 0000 3a8d 6d6f 6f76 0000 006c  isom..:.moov...l
0000020: 6d76 6864 0000 0000 c665 eccfc665 eccfmvhd.....e...e..
0000030: 0000 0258 0000 9116 0001 0000 0100 0000  ...X............
[...]
```

Figura 4.10: início do arquivo recuperado.

Os visualizadores Totem, Media Player Classic, Windows Media Player e VLC Media Player mostram cerca de quatro segundos de vídeo antes de apresentarem um erro de decodificação.

4.1.4.3 Constatação de que o arquivo recuperado por *data carving* encontra-se corrompido

O vídeo original possui um tamanho de ~3,7 MiB, e o último conjunto (132) começa no endereço 0x3a8f82, conforme box "stco" (Seção 3.2.3.2.3.3.6). Neste endereço do arquivo recuperado não é encontrado um quadro. O valor que aparece nos quatro primeiros bytes – 0x3e7db15f – é maior que o tamanho do arquivo original inteiro (ver Figura 4.11). Deveria ser o tamanho deste quadro (de número 1836[90]), que é dado no box "stsz" e encontra-se no apêndice – o valor é 0x0945.

```
03a8f82:      3e7d b15f f6a6 e0b8 98a5 e4b3 1af7    >}._..........
03a8f90: 7f4b 5a28 db2d 1ce7 6b83 5919 df1b 8df6  .KZ(.-..k.Y.....
[...]
```

Figura 4.11: Dados no endereço 0x3a8f82 do arquivo recuperado.

[90] O último conjunto (*chunk*) começa após 15 + 130 * 14 = 1835 quadros – ver *box* "stsc" (Seção 3.2.3.2.3.3.4).

4.1.4.4 Constatação de que há quadros recuperáveis em outras áreas da mídia (fragmentação dos dados)

Da informação do Totem (cerca de quatro segundos recuperados) e do fato de haver cerca de 30 quadros (amostras) por segundo (ver box "`mdhd`", Seção 3.2.3.2.1), conclui-se que entre 100 e 150 quadros fazem parte do trecho recuperado do vídeo (de um total de 1847). O box "`stco`" (Seção 3.2.3.2.3.3.6) lista o início de cada conjunto. O décimo conjunto inicia-se com o quadro 128 (posição 0x03b032) e o décimo-primeiro começa com o quadro 142 (posição 0x03f364), de acordo com o box "`stsc`" (Seção 3.2.3.2.3.3.4), conforme mostrado na Figura 4.12.

```
003b032:      0000 03b5 419a 7f3f b3c2 37ff ffff    ....A..?..7...
003b040: ffff efff f5fc 65fb 62f2 f2a8 b3f4 547d  ......e.b.....T}
[...]
03f364:           8b3f 24f9 310e f330 d0d9 73f0      .?$.1..0..s.
003f370: 7790 e4cf c2df 6b24 7f3a fede 25f9 af64  w.....k$.:..%..d
```

Figura 4.12: Início dos quadros 128 e 142.

O tamanho do quadro 128 é 0x03b9 e o do quadro 142 é 0x0373 – ver Apêndice. Note-se que o valor mostrado na Figura 4.12 para o quadro 128 é 0x03b5. A diferença de 0x04 corresponde aos quatro bytes do cabeçalho (que identifica o tamanho). Também é mostrado o início do quadro 129.

No endereço 0x03f364 aparece o valor 0x8b3f24f9, quando deveria ser 0x0373. Há duas possibilidades para explicar porque o quadro 142 não se encontra nesse endereço: (i) o arquivo foi fragmentado (é o que ocorre neste caso, conforme a preparação feita); (ii) a parte do arquivo correspondente a esse quadro foi sobrescrita. A recuperação de imagens somente é possível no primeiro caso [56].

Sabendo-se que os dados do arquivo estão armazenados em blocos de 4096 bytes (ver Seção 2.12.2), pode-se concluir que o valor "0000 036f" (= 0x0373 – 0x04) começa no byte 868 de um bloco (considerando-se que o primeiro byte é o zero). O endereço 0x03f364 do quadro 142 equivale ao byte 258.916, cujo resto da divisão inteira por 4096 é 868. Usando o comando `sigfind`[91] do pacote `SleuthKit`[92], o valor "0000 036f" foi

[91] Coleção de ferramentas de linha de comando, gratuitas e com código fonte aberto, para análise forense de mídias de armazenamento digitais. Disponível em http://www.sleuthkit.org/.
[92] Ferramenta destinada a localizar assinaturas binárias em arquivos.

localizado no bloco 110.003 na posição 868 (hexadecimal 0x0364)[93], conforme mostrado na Figura 4.13.

```
$ sigfind -b 4096 -o 868 0000036f imagem-1g.dd
Block size: 4096  Offset: 868  Signature: 36F
Block: 110003 (-)
```

Figura 4.13: Comando para localizar o valor "0000 036f".

A visualização do bloco (ver Figura 4.14) confirma que o quadro foi localizado. Na posição 0x0364 encontra-se o valor 0x036f, que é quatro unidades menor que 0x0373 – mostrado no box "`stsz`" (ver Apêndice). Além disso, o quadro seguinte (143) deve começar no endereço 0x06d7 (= 0x0364 + 0x04 + 0x036f). Nesta posição encontra-se o valor 0x032d – o tamanho do quadro – e o valor no box "`stsz`" é 0x0x331.

```
$ dd if=imagem-1g.dd bs=4096 count=1 skip=110003 2>/dev/null | xxd
[...] (amostra 142: tam. 0x036f (+ 0x04 = 0x0373 -> ok em "stsz")
000 0364:           0000 036f 419a 8d46 b3fa ffe1      ...oA..F....
0000370: 4d7f ffff ff7f dc5b a7ee 12de e873 2963  M......[.....s)c
[...] (amostra 143: tam. 0x032d (+ 0x04 = 0x0331 -> ok em "stsz")
000 06d7:                 00 0003 2d41 9a8e 4733         ...-A..G3
00006e0: cfaf ffff ffff ffff ffc4 6e33 537a ddbc  ..........n3Sz..
```

Figura 4.14: Vista parcial do bloco 110.003 de imagem-2g.dd.

4.1.4.5 Determinação de quais quadros estão armazenados no 1º e no 2º fragmentos do arquivo

O quadro 142 e seguintes, como visto, fazem parte de um segundo fragmento. Ainda falta estabelecer qual o último quadro do primeiro fragmento e qual é o primeiro quadro do segundo fragmento. Existem 13 quadros entre o 128 e o 142, os quais fazem parte do décimo *chunk*. Neste caso, não há endereços intermediários nos metadados (no box "`stco`" – Seção 3.2.3.2.3.3.6 - são listados apenas os endereços do primeiro quadro de cada conjunto). O endereço de cada um desses quadros deve ser obtido

[93] Como pode ser visto na Figura 4.13, o valor 868 é utilizado como *offset*, informando o deslocamento de interesse, dentro de cada bloco, para localização da assinatura.

sequencialmente, somando-se o tamanho de um quadro conhecido (neste caso, o quadro 128) ao seu endereço de início para chegar ao endereço do próximo quadro.

```
03b032:      0000 03b5 419a 7f3f b3c2 37ff ffff    ....A..?..7...
003b040: ffff efff f5fc 65fb 62f2 f2a8 b3f4 547d  ......e.b.....T}
[...]
03b3eb:                           00 0003 e741             ....A
003b3f0: 9a80 4033 c75f dfff efff fe23 8bdc 45ea  ..@3._.....#..E.
[...]
03b7d6:                0000 0423 419a 8140 b3c7        ...#A..@..
003b7e0: daff fafe afff ffff ffff f8ad 54ba b88a  ............T...
[...]
03bbfd:                                 00 0004              ...
003bc00: 5041 9a82 4133 f592 fffc 97ff ffff c56f  PA..A3.........o
[...]
03c051:   06 55f3 7803 8bb0 225e 3fb0 4eff 82eb   .U.x..."^?.N...
003c060: 545d 1822 78e8 5f74 1872 2834 ee33 31be  T]."x._t.r(4.31.
```

Figura 4.15: Início dos quadros 128, 129, 130 e 131.

A Figura 4.15 mostra o início dos quadros 128 a 131. A Tabela 4.3 mostra os endereços e tamanhos calculados dos quadros 129 a 132, com base no quadro 128.

Tabela 4.3: Lista dos quadros 128 a 132.

# quadro	endereço (x)	tamanho (y)	x+y+4
128	0x03b032	0x03b5	0x03b3eb
129	0x03b3eb	0x03e7	0x03b7d6
130	0x03b7d6	0x0423	0x03bbfd
131	0x03bbfd	0x0450	0x03c051
132	0x03c051	0x0655f378 (?)	na

Os quadros 129, 130 e 131 mostram valores coerentes, mas o quadro 132 não. Voltando ao box "`stsz`" (Seção 3.2.3.2.3.3.5), vê-se que o tamanho deste quadro é 0x03be; portanto, o valor no endereço 0x03c051 deveria ser 0x03ba (= 0x03be – 0x04). O endereço 0x03bbfd encontra-se no 60º bloco a partir do 104.100. Portanto, o primeiro fragmento do vídeo ocupa os blocos 104.100 a 104.159.

O endereço 0x03c051 corresponde ao byte 245.841. O resto da divisão inteira desse valor por 4096 é 81 (0x51), que representa a posição do quadro em relação ao início de um bloco. Usando novamente o `sigfind`, localiza-se o valor "0000 03ba" no bloco 110.000, como pode ser visto na Figura 4.16.

```
$ sigfind -b 4096 -o 81 000003ba imagem-1g.dd
Block size: 4096  Offset: 81  Signature: 3BA
Block: 110000 (-)
```

Figura 4.16: Localização do valor "0000 03ba" na posição 81.

A Figura 4.17 mostra, parcialmente, o bloco 110.000, onde se pode ver o início dos quadros 132 e 133.

```
$ dd if=imagem-2g.dd bs=4096 skip=110000 count=1 2>/dev/null | xxd
[...]
0000051:   00 0003 ba41 9a83 41b3 c73e 0f7c 1d46  ....A..A..>.|.F
0000060: ebff 9ab2 de9e 5937 beee c4ee 85bf 78cf  ......Y7......x.
[...]
000040f:                                       00                .
0000410: 0003 8741 9a84 4233 c76a f5ff fe6a d579  ...A..B3.j...j.y
[...]
```

Figura 4.17: Visualização parcial do bloco 110.000.

Os valores mostrados na Figura 4.17 estão resumidos na Tabela 4.4.

Tabela 4.4: Quadros 132 e 133.

# quadro	endereço (x)[94]	tamanho (y)	x+y+4
132	0x0051	0x03ba	0x040f
133	0x040f	0x0387	0x079a

O quadro 142 foi encontrado na posição 868 (0x0364) do bloco 110.003. Então, sua distância do quadro 132 é igual a 13.075 (= 868 + 12288[95]- 81) bytes. No arquivo original, o endereço do quadro 142 é 0x03f364 e o do quadro 132 é 0x03c051. A diferença entre esses dois valores é 0x3313, que corresponde a 13.075 bytes, o que prova que os blocos 110.000 até 110.003 fazem parte do mesmo fragmento. Portanto, foram localizados os quadros até o de número 142 (1º quadro do 11º conjunto).

Para agilizar o processo, e não ser necessário avaliar o restante dos 1847 quadros, pode-se usar os endereços do primeiro quadro de cada conjunto, usando os valores do box "`stco`" (Seção 3.2.3.2.3.3.6). O próximo conjunto é o 12º, cujo endereço é 0x0439ea (do

[94]Endereços relativos ao bloco 110.000.
[95] 12288 = (110.003 – 110.000) * 4096.

quadro 156 = 1 + 15 + 10 * 14, segundo o box "stsc", Seção 3.2.3.2.3.3.4). O endereço relativo ao início do bloco 110.000 é 0x79ea (0x0439ea – 0x03c000). O valor encontrado no box "stsz" (Seção 3.2.3.2.3.3.5) para o quadro 156 (1º quadro do 12º conjunto) é 0x0434. O valor que aparece na Figura 4.18 é 0x0430, o que comprova que o quadro foi localizado.

```
000 79ea:                          0000 0430 419a            ...0A.
00079f0: 9b4d b3c7 ebcf ffff ffff ffff ffff fc4e  .M.............N
[...]
```

Figura 4.18: Quadro 156 encontrado no byte 31210 a partir do início do bloco 110.000.

Pode-se agilizar ainda mais o processo, efetuando-se uma pesquisa binária[96]. Seja 72 (= 12 + (132 - 12) / 2) o número do conjunto intermediário entre o conjunto já localizado, 12º, e o último conjunto do vídeo, 132º. O seu endereço é 0x1a79b4. Subtraindo-se 0x03c000 (ou 245.760 = 60 * 4096), tem-se a posição relativa 0x16b9b4 desde o início do bloco 110.000. Nesta posição não foi encontrado o quadro 996 (= 1+ 15 + 14 * 70), que deveria conter o valor 0x00000c75 – ver Figura 4.19.

```
0 16b9b4:           0a87 4383 65af 56a2 eeff e76f      ..C.e.V....o
016b9c0: ba65 00ac 96d3 e550 4118 c1a5 b2fc 6e45  .e.....PA.....nE
[...]
```

Figura 4.19: Quadro não encontrado na posição relativa (ao bloco 110.000) 0x16b9b4.

O conjunto 42 (= 12 + (72 - 12) / 2) começa no endereço 0x0e5883, cujo valor relativo ao bloco 110.000 é 0x0a9883 (ver Figura 4.20).

[96] Algoritmo de busca em lista ordenada que requer acesso aleatório aos seus elementos. São realizadas sucessivas divisões do espaço de busca, comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor; se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor.

```
00a9883:        d3 7a53 9048 53e8 d661 9fd3 14ef     .zS.HS..a....
00a9890: 6731 43aa 61fc d6e1 32a6 6bb6 3d75 4dac  g1C.a...2.k.=uM.
[...]
```

Figura 4.20: Quadro não encontrado na posição relativa (ao bloco 110.000) 0x0a9883.

O conjunto 27 (= 12 + (42 - 12) / 2) começa em 0x09d23e, cujo valor relativo é 0x06123e. O tamanho do conteúdo do quadro 366 (= 1 + 15 + 25 * 14) é 0x0235 (= 0x0239 - 0x04). A Figura 4.21 confirma que o quadro foi encontrado.

```
0061240: 0235 419a 7339 b3c5 5ffe bfff ffac baff  .5A.s9.._.......
[...]
```

Figura 4.21: Quadro 366 encontrado na posição relativa (ao bloco 110.000) 0x06123e.

O conjunto 35 (= 27 + (42 - 27 + 1) / 2) começa no endereço 0x0cb9fd, cujo valor relativo ao bloco 110.000 é 0x08f9fd, onde não foi encontrado um quadro (ver Figura 4.22).

```
008f9fd:                                  d8 1721                ..!
008fa00: 2179 7254 2f9d 4467 f85b 5fa4 98a4 508c  !yrT/.Dg.[_...P.
[...]
```

Figura 4.22: Quadro não encontrado na posição relativa (ao bloco 110.000) 0x08f9fd.

O conjunto 31 (= 27 + (35 - 27) / 2) começa no endereço 0x0b4a18, cujo valor relativo ao bloco 110.000 é 0x078a18, onde não foi encontrado um quadro (ver Figura 4.23).

```
0078a18:                      0697 b528 8d7a c09e          ...(.z..
0078a20: 61e0 99c2 e8bb 9d5d e584 039a 1d61 4cb4  a......]......aL.
[...]
```

Figura 4.23: Quadro não encontrado na posição relativa (ao bloco 110.000) 0x078a18.

O conjunto 29 começa no endereço 0x0a910a, cujo valor relativo ao bloco 110.000 é 0x06d10a. O tamanho do conteúdo do quadro 394 (= 1 + 15 + 27 * 14) é 0x057b (= 0x057f - 0x04). A Figura 4.24 confirma que o quadro foi encontrado.

```
006d10a:                          0000 057b 419a             ...{A.
006d110: 8f47 b3fa ffff 25ff ff5f ff5f ffff ffff  .G....%.._._....
[...]
```

Figura 4.24: Quadro 394 encontrado na posição relativa (ao bloco 110.000) 0x06d10a.

O conjunto 30 começa no endereço 0x0afbb3, cujo valor relativo ao bloco 110.000 é 0x073bb3. O tamanho do conteúdo do quadro 408 (= 15 + 14 * 28 + 1) é 0x03f7 (= 0x03f3b - 0x04). A Figura 4.25 confirma que o quadro foi encontrado.

```
0073bb3:        00 0003 f741 9a9d 4eb3 c257 ffff  oC.....A..N..W..
0073bc0: ffff ffff fffa d4da ad5b c58a e730 4afb  .........[...0J.
[...]
```

Figura 4.25: Quadro 408 encontrado na posição relativa (ao bloco 110.000) 0x073bb3.

Então existe uma descontinuidade entre os quadros 408 e 422 (início do conjunto 31). Neste caso não é possível uma pesquisa binária, pois a localização de um quadro do conjunto depende do conhecimento do quadro anterior. Na Figura 4.26 são mostradas as oito primeiras entradas do conjunto 30 (quadros 408 a 415 do vídeo), as quais endereçam quadros.

```
0073bb3:        00 0003 f741 9a9d 4eb3 c257 ffff  oC.....A..N..W..
[...]
0073fae:                                  0000                ..
0073fb0: 049b 419a 9e4f 33f5 ad7f ffbd 7e11 ac9a  ..A..O3.....~...
[...]
07444d:                                     00 0004               ...
0074450: 8e41 9a9f 4fb3 c56b ffff ffdf dfff ffff  .A..O..k........
[...]
0748df:                                       00                .
00748e0: 0004 a841 9aa0 5033 c90d f81f bfff ffeb  ...A..P3........
[...]
074d8b:                               00 0005 0141              ....A
[...]
075290: 0000 0477 419a a251 33f5 9f5f ffe1 dfe1  ...wA..Q3.._....
[...]
07570b:                               00 0004 6941              ...iA
[...]
075b78:                         0000 0477 419a a452          ...wA..R
[...]
```

Figura 4.26: Os oito primeiros quadros do conjunto 30.

A Figura 4.27 mostra os quadros 08 a 14 do conjunto 30 (quadros 415 a 421 do vídeo) e o endereço 0x0776c4, onde termina o quadro 421 – o endereço 0x0776c5 (= 0x07734a + 0x04 + 0x0377) fica após o quadro, mas não é o endereço do quadro 422, que é dado no box "`stco`" (Seção 3.2.3.2.3.3.6).

```
075b78:                         0000 0477 419a a452          ...wA..R
[...]
075ff3:        00 0004 5341 9aa5 52b3 c95f ffff     ...SA..R.._..
[...]
07644a:                               0000 03be 419a            ....A.
[...]
07680c:                                    0000 0389              ....
[...]
076b99:                            00 0003 f441 9aa8           ....A..
[...]
076f91:   00 0003 b541 9aa9 54b3 cbaf c12b 831a   ....A..T....+..
[...]
07734a:                               0000 0377 419a            ...wA.
[...]
00776c0: 3079 9699 9221 4ffe ffff 98a3 05aa 944a  0y...!O........J
[...]
```

Figura 4.27: Os últimos quadros do conjunto 30.

Os endereços 0x07734a (início do quadro 421) e 0x0776c5 (byte seguinte ao fim do quadro 421) estão no 120º bloco a partir do bloco 110.000, i.e., estão no bloco 110.119 (uma vez que o primeiro bloco do volume é o bloco zero). O endereço relativo 0x078a18 (0xb4a18 – 0x3c000) está situado no 121º bloco a partir do bloco 110.000, mas não aponta

para o quadro 422 (primeiro do conjunto 31). Portanto, até o momento, foram identificados 180 blocos do vídeo: 60 blocos nos blocos 104.100 a 104.159 e 120 blocos nos blocos 110.000 a 110.119.

4.1.4.6 Determinação da existência de outros quadros fora os encontrados nos dois primeiros fragmentos do arquivo

Usa-se a Tabela 4.5 para calcular os parâmetros de pesquisa para a ferramenta `sigfind` (usada anteriormente) localizar um quadro (neste caso, o primeiro quadro de cada conjunto) considerando-se o seu tamanho. O comando geral é mostrado na Figura 4.28.

```
$ sigfind -b 4096 -o y xxxxxxxx imagem-1g.dd
```

Figura 4.28: Comando para localizar o bloco que contém o valor "xxxxxxxx" na posição "y".

O valor "`xxxxxxxx`" é um número hexadecimal com oito caracteres, cujo valor é obtido subtraindo-se quatro do tamanho listado na Tabela 4.5. O valor de "`y`" é o resto da divisão inteira do endereço do quadro por 4096 – nesta operação, todos os números são decimais.

Tabela 4.5: Endereços dos conjuntos de quadros de "stco" a partir conjunto 31.

Contém, em cada célula, o nº do quadro, endereço do quadro e tamanho total. Os índices na 1ª coluna concatenados com a 3ª linha dão o nº do conjunto.										
-	1	2	3	4	5	6	7	8	9	10
3	422 000b4a18 (0x0337)	436 000ba44b (0x07ac)	450 000bf914 (0x06b1)	464 000c5bed (0x0456)	478 000cb9fd (0x058d)	492 000d48dc (0x0578)	506 000d7c9b (0x0148)	520 000d9fcb (0x0106)	534 000dc9d0 (0x02cf)	548 000dfedf (0x00c5)
4	562 000e2957 (0x015f)	576 000e5883 (0x00b2)	590 000e8729 (0x0342)	604 000ebd26 (0x01e5)	618 000ef336 (0x035f)	632 000f258b (0x02e9)	646 000f5cd9 (0x0257)	660 00102071 (0x0160)	674 001056be (0x010c)	688 0010d191 (0x0453)
5	702 00116972 (0x023e)	716 0011fe59 (0x04ef)	730 00125593 (0x03aa)	744 0012a1ee (0x03fe)	758 0012fe3f (0x06e3)	772 00134e35 (0x0461)	786 0013a41c (0x0354)	800 0013f359 (0x069c)	814 0014430e (0x0465)	828 00148e44 (0x01f8)
6	842 0014dadb (0x0828)	856 00153a45 (0x03da)	870 0015f3a1 (0x06c2)	884 001632ed (0x019b)	898 001663e7 (0x020e)	912 0016a18b (0x0762)	926 00170b58 (0x01c0)	940 0017994f (0x02f6)	954 00181ced (0x1a81)	968 00188cc4 (0x08bd)

A Tabela 4.5 é parcial, para facilitar a visualização e os cálculos, e contempla apenas os conjuntos (*chunks*) 31 (quadro 422) a 70 (quadro 968). Fazendo um resumo do

procedimento adotado para localizar os quadros a partir do 422º, a ferramenta `sigfind` não encontrou nenhum valor para os quadros dos conjuntos 31 (quadro 422) até o 65 (quadro 898). Foi obtido um resultado positivo para o quadro 912 do conjunto 66. A Figura 4.29 mostra esses três resultados.

```
$ sigfind -b 4096 -o 2584 00000333 imagem-1g.dd
Block size: 4096  Offset: 2584  Signature: 333
[nenhum resultado]

[...]

$ sigfind -b 4096 -o 999 0000020a imagem-1g.dd
Block size: 4096  Offset: 999  Signature: 20A
[nenhum resultado]

$ sigfind -b 4096 -o 395 0000075e imagem-1g.dd
Block size: 4096  Offset: 395  Signature: 75E
Block: 160002 (-)
```

Figura 4.29: Busca de valores usando a ferramenta sigfind.

O quadro 912 foi localizado no bloco 160.002, na posição (byte) 395, cujo valor hexadecimal é 0x018b, conforme mostrado na Figura 4.30.

```
$ dd if=imagem-1g.dd bs=4096 skip=160002 2>/dev/null | xxd
[...]
000018b:                              00 0007 5e41  Kv....3...p...^A
0000190: 9a33 19b3 c5d6 a4bb b7f8 8bbd ddda eff0  .3..............
[...]
```

Figura 4.30: Início do quadro 912 no bloco 160.002.

O endereço 0x16a18b do quadro (ver Tabela 4.5), cujo valor decimal é 1.483.147, está no 362º bloco do arquivo original[97] (1483147 / 4096 = ~362,096). Isso significa que os blocos 120 a 361 (começando-se do bloco zero) foram perdidos[98]. O procedimento para localizar os quadros 912 em diante é o mesmo adotado anteriormente e não será mostrado. Basta informar que os quadros até o 1318 (o primeiro do conjunto 95) foram localizados e estão íntegros. A Figura 4.31 mostra a localização desse primeiro quadro do conjunto 95.

[97] Contagem dos blocos inicia-se em zero.
[98] Em uma situação real, a explicação mais usual para essa perda é a sobrescrição dos dados.

```
$ sigfind -b 4096 -o 1314 000009d1 imagem-1g.dd
Block size: 4096  Offset: 1314  Signature: 9D1
Block: 160238 (-)
```

Figura 4.31: Localização do quadro 1318 no bloco 160.238.

4.1.4.7 Reconstrução do arquivo de vídeo acrescentando a ele todos os quadros já recuperados

A reconstrução do vídeo até o quadro 1317, zerando-se (com zeros binários) os blocos contendo os quadros 422 a 911, é mostrada na Figura 4.32.

```
$ dd if=imagem-1g.dd bs=4096 skip=104100 count=60 of=video-rec.mp4
$ dd if=imagem-1g.dd bs=4096 skip=110000 count=120 >> video-rec.mp4
$ dd if=/dev/zero bs=4096 count=182 >> video-rec.mp4
$ dd if=imagem-1g.dd bs=4096 skip=160002 count=23
```

Figura 4.32: Construção do arquivo de vídeo video-rec.mp4.

4.1.4.8 Apresentação de algoritmo para localização de quadros

Os procedimentos executados anteriormente formam um algoritmo que permite localizar todos os quadros não sobrescritos do vídeo, o qual é apresentado na Figura 4.33.

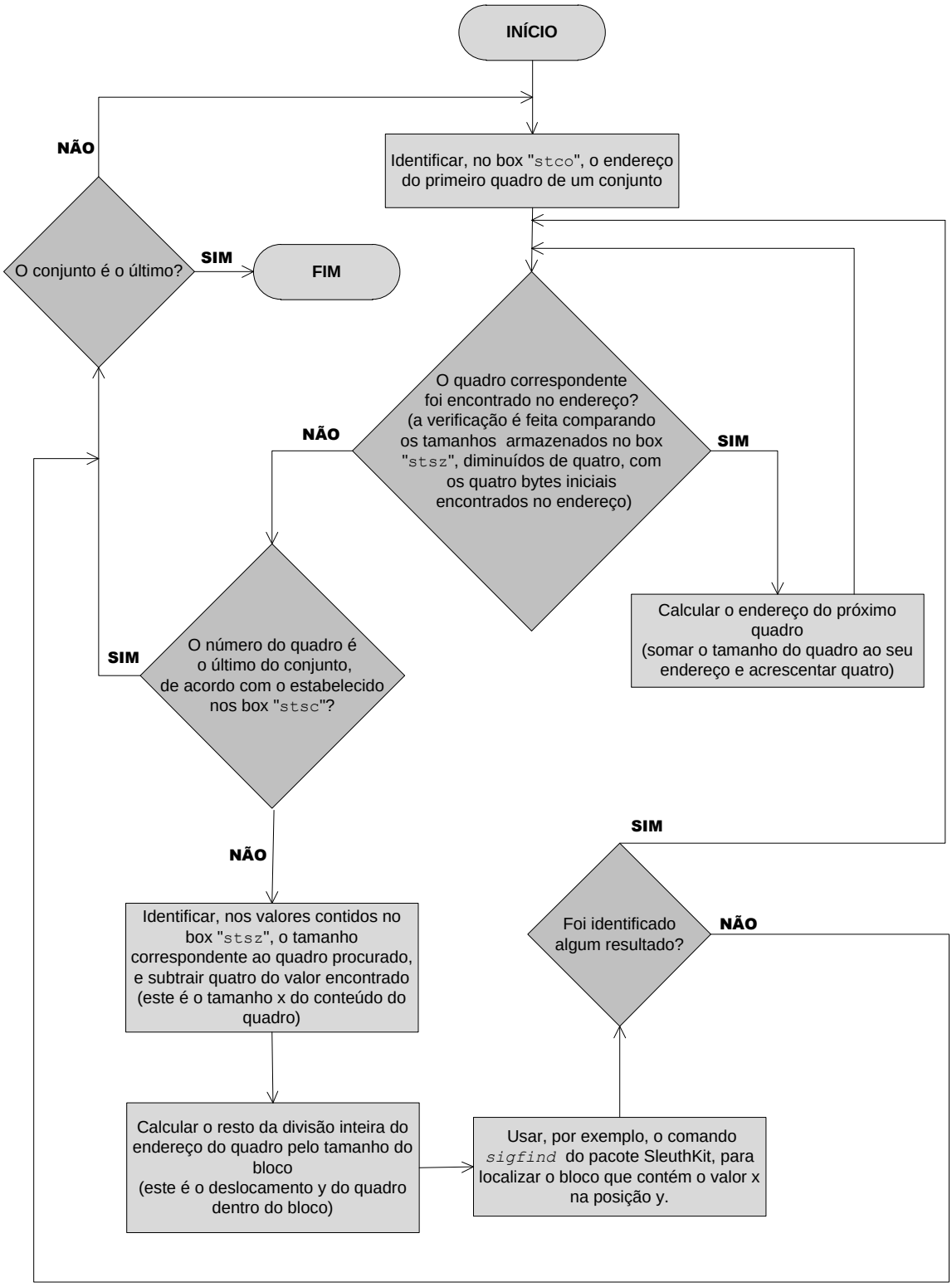

Figura 4.33: Algoritmo para localização dos quadros de um arquivo de vídeo.

4.1.4.9 Análise do resultado da reconstrução do arquivo e recuperação de seus quadros

Os aplicativos testados (Totem, Media Player Classic, Windows Media Player e VLC Media Player) haviam conseguido mostrar (decodificar) aproximadamente 4 segundos de vídeo do arquivo recuperado pela ferramenta `foremost` (ver Seção 4.1.4.2), `00832800.mp4`. Esses 4 segundos correspondem aos 131 quadros iniciais contidos no primeiro trecho do arquivo original que fora fragmentado conforme mostrado na Seção 4.1.4.1.

Do arquivo reconstruído por meio da técnica apresentada (`video-rec.mp4`), no entanto, os mesmos aplicativos conseguiram mostrar aproximadamente 14 segundos de vídeo. Esses 14 segundos representam os 131 quadros iniciais, contidos no primeiro trecho da fragmentação, e também os quadros subsequentes de número 132 a 421, que foram identificados em área não contígua da mídia. Sabe-se que o arquivo `video-rec.mp4` contém diversos outros quadros que não só os 421 primeiros. Esses demais quadros, porém, encontram-se separados dos 421 primeiros por trecho de zeros binários, e, exatamente como aconteceu em teste distinto (Seção 4.1.3 - Arquivo contíguo com um trecho de dados zerado), constatou-se que, nessa situação, os citados aplicativos (*players*) não conseguem reproduzi-los.

A ferramenta `ffmpeg`, no entanto, conseguiu recuperar os outros quadros além dos 421 primeiros. Efetivamente, por meio do comando mostrado na Figura 4.34 foram recuperados, de forma íntegra, todos os quadros que haviam sido encontrados na mídia e acrescentados ao arquivo `video-rec.mp4`, contanto que retrocedessem a um quadro de acesso randômico (*random access point* – ver Seção 2.7.1), os quais agem como pontos de entrada no fluxo de vídeo.

```
$ ffmpeg -re -i video-rec.mp4 -r 1 dir-rec/video-rec-%02d.jpeg

$ ls -gG dir-rec/
-rw-r--r-- 1 15064 2011-12-14 22:38 video-rec-01.jpeg
-rw-r--r-- 1 19679 2011-12-14 22:38 video-rec-02.jpeg
-rw-r--r-- 1 24774 2011-12-14 22:38 video-rec-03.jpeg
-rw-r--r-- 1 25326 2011-12-14 22:39 video-rec-04.jpeg
[...]
-rw-r--r-- 1 19757 2011-12-14 22:39 video-rec-43.jpeg
-rw-r--r-- 1 17121 2011-12-14 22:39 video-rec-44.jpeg
-rw-r--r-- 1 18978 2011-12-14 22:39 video-rec-45.jpeg
-rw-r--r-- 1 16561 2011-12-14 22:39 video-rec-46.jpeg
[...]
```

Figura 4.34: Recuperação de quadros em video-rec.mp4.

5 CONCLUSÕES

Arquivos de vídeo H.264/AVC em contêineres ISO/IEC 14496 são constantemente identificados em mídias de armazenamento de dados periciadas no Instituto de Criminalística da Polícia Civil do Distrito Federal, e, com frequência, mostram-se importantes fontes de evidências. Diversas são as situações, contudo, em que esses arquivos se encontram corrompidos. Desse problema nasceu a motivação para o trabalho, qual seja determinar a possibilidade de recuperação de quadros (imagens) que compõem arquivos de vídeo danificados e irreproduzíveis.

Um abrangente referencial teórico sobre o padrão de vídeo H.264/AVC e sobre os contêineres ISO/IEC 14496 foi inicialmente apresentado, ao qual se seguiram análises e interpretações detalhadas de um arquivo de vídeo resgatado de um cartão de memória encaminhado para perícia. Alcançou-se, por meio desse estudo, um amplo entendimento sobre os diversos compartimentos (boxes) responsáveis pelo armazenamento de fluxos de vídeo (*bitstreams*) e de seus respectivos metadados. Em sequência, o *bitstream* em si também foi examinado, à luz do documento normatizador do padrão. A soma de todos esses esforços resultou na conclusão de que quadros integrantes de *bitstream* desacompanhado de seus metadados são irrecuperáveis.

Em frente de igual relevância, buscou-se determinar o estado mínimo de integridade de um arquivo de vídeo que permita a reprodução de seu conteúdo ou a recuperação de quadros que dele façam parte. Para isso, diferentes tipos e graus de corrupção foram intencionalmente infligidos a um arquivo de vídeo, tendo as consequências sido expostas e interpretadas. Passo a passo, foi apresentado um procedimento para recuperação de quadros de um arquivo não íntegro. A descrição desse procedimento, embasada por considerações teóricas, permitiu o estabelecimento da conclusão de que, estando presentes os metadados do arquivo (box "`moov`"), os quadros não sobrescritos podem ser recuperados. Mais do que isso, há a possibilidade de vinculá-los a informações de data e hora de criação e última modificação do arquivo do qual faziam parte. De forma a facilitar a replicação da técnica, foi apresentado ainda um algoritmo que resume os passos requeridos para a localização dos quadros de um vídeo.

Por fim, visando a aumentar o entendimento das diferentes possíveis formas de organização dos dados e metadados de um arquivo de vídeo H.264/AVC, sugere-se, como trabalhos futuros, que esse estudo seja replicado em *bitstreams* armazenados em

contêineres que não os baseados na família de padrões ISO/IEC 14496, como, por exemplo, os contêineres Matroska (MKV) e Audio Video Interleave (AVI).

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Associação Brasilieira de Criminalística, “Associação Brasilieira de Criminalística,” 2011. [Online]. Available: http://www.abcperitosoficiais.org.br/. [Acesso em 10 Dezembro 2011].

[2] Microsoft, “Windows Media Player,” 2012. [Online]. Available: http://windows.microsoft.com/en-US/windows/products/windows-media. [Acesso em 02 12 2011].

[3] GABEST, “Media Player Classic,” 2012. [Online]. Available: http://mpc-hc.sourceforge.net/. [Acesso em 02 12 2011].

[4] gnome, “Totem,” 2011. [Online]. Available: http://projects.gnome.org/totem/. [Acesso em 02 12 2011].

[5] Videolan, “VLC Media Player,” 2011. [Online]. Available: http://www.videolan.org/vlc/. [Acesso em 02 12 2011].

[6] Free Software Foundation, “VirtualDub,” 2011. [Online]. Available: http://virtualdub.org/. [Acesso em 27 12 2011].

[7] Adobe, “Adobe Premiere Pro CS5.5,” 2011. [Online]. Available: http://www.adobe.com/br/products/premiere.html.

[8] P. Glagla, “ImageGrab,” 2011. [Online]. Available: http://paul.glagla.free.fr/imagegrab_en.htm. [Acesso em 27 12 2011].

[9] ISO-IEC, “ISO/IEC 14496-10 (Coding of Audio-Visual Objects - Part 10: Advanced Video Coding),” 2010.

[10] “Recommendation ITU-T H.264, Edition 5.0,” 2010.

[11] MPEGLA, “The Standard for Standards,” 2011. [Online]. Available: http://www.mpegla.com/main/programs/AVC/Pages/Licensees.aspx. [Acesso em 02 12 2011].

[12] Mefeedia, “HTML5 Video Available on the Web,” 06 2011. [Online]. Available: http://blog.mefeedia.com/html5-june-2011. [Acesso em 05 11 2011].

[13] D. Vatolin, D. Kulikov, A. Parshin e M. Arsaev, “MPEG-4 AVC/H.264 Video Codecs Comparison,” CS MSU Graphics&Media Lab, Video Group, Moscow, 2011.

[14] A. Gil, “Como Elaborar Projetos de Pesquisa,” *Atlas,* 1996.

[15] K. Andaloussi, “Pesquisas-ações. Ciências, desenvolvimento e democracia,” 2004.

[16] S. Gary J. e T. Wiegand, “Video Compression - From Concepts to the H.264/AVC Standard,” *Proceedings of The IEEE,* vol. 93, n. 1, 2005.

[17] A. Gibson, “The H.264 Video Compression Standard,” 2002.

[18] I. E. Richardson, The H.264 Advanced Video Compression Standard, Wiley, 2010.

[19] E. T. M. Manoel, “Codificação de Vídeo H.264 - Estudo de Codificação Mista de Macroblocos,” 2007.

[20] International Telecommunication Union, 2012. [Online]. Available: http://www.itu.int/ITU-T/. [Acesso em 02 01 2012].

[21] “International Organization for Standardization,” 2012. [Online]. Available: http://www.iso.org/iso/home.html. [Acesso em 02 01 2012].

[22] J. R. Corbera, P. A. Chou e S. L. Regunathan, “Generalized Hypothetical Reference Decoder for H.264/AVC,” *IEEE Transactions on Circuits and Systems for Video Techology,* vol. 13, n. 7, 2003.

[23] Y. Su, J. Xin, A. Vetro e H. Sun, “Efficient MPEG-2 to H.264/AVC Intra Transcoding in Tranform-Domain,” *Mitsubishi Electric Research Laboratories, Inc.,* 2005.

[24] I. E. G. Richardson, H.264 and MPEG-4 Video Compression, Wiley, 2003.

[25] gigaom, “H.264 is still winning the codec war,” 07 2011.

[26] Wikipedia, “List of video services using H.264/MPEG-4 AVC,” 2011.

[27] T. Wiegand, G. J. Sullivan, G. Bjontegaard e A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, n. 7, 2003.

[28] H. Schwarz, D. Marple e T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard,” *IEEE Transactions on Circuits and Systems*

] *for Video Technology,* vol. 17, n. 9, 2007.

[29] I. Richardson, “An Overview of H.264 Advanced Video Coding,” *Onecodec - The Future of Video Coding,* 2011.

[30] M. Flierl e B. Girod, “Generalized B Pictures and the Draft H.264/AVC Video Compression Standard,” *IEEE Transactions Circuits and Systems for Video Technology,* 2003.

[31] X. yu e E.-h. Yang, “Rate Distortion Optimization in H.264,” 2007.

[32] H. S. Malvar, A. Hallampuro, M. Karczewicz e L. Kerofsky, “Low-Complexity Transform and Quantization in H.264/AVC,” *IEEE Transactions on Circuits Systems For Video Technology,* vol. 13, n. 7, 2003.

[33] S. Sethuraman, “The H.264 Advanced Video Compression Standard,” *IEEE Multimedia Compression Workshop,* 2005.

[34] X. Tian, T. Le e Y. Lian, Entropy Coders of the H.264/AVC Standard, 2011.

[35] D. Marple, H. Schwarz e T. Wiegand, “Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video COmpression Standard,” *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, n. 7, 2003.

[36] Wikipedia, “Digital container format,” 2011. [Online]. Available: http://en.wikipedia.org/wiki/Digital_container_format. [Acesso em 22 12 2011].

[37] M. Hannuksela, “Multiple meta boxes in ISO base media file format,” *ISO/IECJTC1/SC29/WG11,* 2006.

[38] ISO/IEC, “ISO/IEC 14496-12 (Coding of Audio-Visual Objects – Part 12: ISO Base Media File Format),” 2008. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c051533_ISO_IEC_14496-12_2008.zip.

[39] Apple Inc., “QuickTime File Format Specification,” 2011.

[40] ISO/IEC, “ISO/IEC 14496-14 (Coding of audio-visual objects -- Part 14: MP4 file format),” 2003. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38538.

[41 ISO/IEC, “ISO/IEC 14496-15 (Coding of Audio-Visual Objects - Part 15: Advanced

] Video Coding File Format),” 2010. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=55980.

[42] 3rd Generation Partnership Project, “3GPP File Formats for Multimedia Services,” 2011.

[43] 3rd Generation Partnership Project 2, “3GPP2 File Formats for Multimedia Services,” 2003.

[44] ISO/IEC, “ISO/IEC 14496-1 (Coding of Audio-Visual Objects -- Part 1: Systems),” 2001. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=34903.

[45] B. -. B. I. d. P. e. Mesures, “The International System of Units (SI),” 2006.

[46] International Electrotechnical Commission, “Quantities and Units - Part 13: Information Science and Technology,” 2008.

[47] J. Black, M. Cochran e T. Highland, “A Study of the MD5 Attacks: Insights and Improvements,” 2006.

[48] R. Rivest, “RFC 1321 - The MD5 Message_Digest Algorithm,” 1992.

[49] B. Carrier, “File System Forensic Analysis,” 2005.

[50] National Institute of Standards and Technology, “Test Results for Disk Imaging Tools: dd GNU fileutils 4.0.36, provided with Red Hat Linux 7.1,” 2002.

[51] B. Carrier, “The Sleuth Kit,” 2012.

[52] F. Bellard e M. Niedermayer, “ffmpeg,” 2012.

[53] X.-J. Zhang, X.-H. Peng, R. Haywood e T. Porter, “Robust Video Transmission over Lossy Network by Exploiting H.264/AVC Data Partitioning,” 2008.

[54] F. Boulos, B. Parrein, P. Le Callet e D. Hands, “Perceptual Effects of Packet Loss on H.264/AVC Encoded Videos,” 2009.

[55] X. Zhang e X. Peng, “An Unequal Packet Loss Protection Scheme for H.264/AVC

] Video Transmission," 2009.

[56 D. Feenberg , "Can Intelligence Agencies Read Overwritten Data?," 10 2011.
]

APÊNDICE

Tamanhos dos Quadros – box "stsz"

```
  0 0000304c 00000362 0000043e 000004c6 000005c0 00000582 000007bf 0000061f 000005db 00000677
  1 0000077a 0000083a 0000059e 000006c3 000006e5 00000765 00000664 00000588 00000499 00000468
  2 000003fa 000003dc 00000586 00000503 00000533 000004d0 000004c0 00000444 000003da 0000040a
  3 000002ff 0000045f 00000381 0000025d 0000054a 000003af 000003ee 000002f8 00000307 00000330
  4 000003b0 00000447 00000350 000003f5 000003fb 00000340 000002db 0000042f 00000336 000002c0
  5 000002a6 000002dc 0000034d 000002cd 0000038c 000003f7 000002f2 000002f8 000003b1 0000030b
  6 00000425 00000458 0000042b 00000438 0000039d 000004c5 000004c2 000006ea 000005c7 00000630
  7 000005b4 00000598 0000057d 00000516 000004a6 0000042e 00000486 000005ba 00000588 0000075c
  8 000008e2 00000988 0000096a 00000a04 00000a32 00000a14 00000a43 000009c4 000009a8 00000868
  9 00000838 000008e8 0000096c 00000884 000007b6 00000810 000008c5 0000086e 00000717 000006f9
 10 00000777 00000773 0000064d 000006bc 000006c6 00000723 000006df 0000069d 00000683 000006ad
11 00000641 000005af 000005dd 000004c5 00000450 0000036d 000002e0 00000294 000001f3 000002c7
12 000003b0 00000406 00000469 00000481 000004a2 00000492 0000043c 000003b9 000003eb 00000427
13 00000454 000003be 0000038b 0000029b 00000271 0000031b 000002be 00000314 0000032f 0000034a
14 0000032d 00000373 00000331 00000392 000003e3 000003ef 00000355 00000378 000003b9 000003c9
15 00000383 000003cf 00000325 0000037a 0000037a 00000434 000004be 00000463 0000046b 000004a5
16 0000055b 00000565 00000558 0000052e 0000061b 00000944 000009a7 00000ba3 00000c29 00000c36
17 00000aa0 000009c5 000008f8 00000912 000007f4 0000087f 000008e8 000008d4 000009f5 000009e6
18 00000513 00000504 000005b2 000005fe 0000069d 0000074e 000007ad 0000082d 000007ab 0000070a
19 000006cb 00000678 00000588 0000053e 00000552 0000051e 00000545 00000506 000004cf 00000471
20 0000034b 00000386 0000035d 000003ca 000003ee 000003dc 000003d8 00000466 0000043b 00000474
21 000003f8 000002f3 000003ad 00000610 00000598 0000054a 000004fa 000003e2 00000496 000004ea
22 0000053d 00000520 00000502 00000541 00000520 00000534 00000551 000005c9 000005b2 000005b6
23 000005ec 00000643 0000069b 00000636 0000055d 00000572 00000565 0000055d 00000533 000004e6
24 00000538 0000057a 00000586 000004d5 00000493 000004d4 00000532 00000474 000004b8 0000052c
25 00003635 00000390 000004fe 00000565 00000582 00000549 000005aa 00000629 00000614 00000655
26 000006a7 0000073f 00000639 00000725 000006c0 0000068e 0000086b 000007b4 000007bf 00000741
27 0000065f 000005fe 000005bc 00000602 000006c9 0000070a 0000073d 0000075e 00000764 000006e6
28 000005c7 000004d4 000004b5 0000046a 00000456 00000498 0000050a 00000466 0000049e 0000042b
29 000004b1 000004bd 000004b9 000004fd 00000496 00000421 00000508 000004fb 00000451 00000422
30 000003a6 000003f1 000003ea 00000429 00000440 000003c2 00000387 00000339 00000368 000002b4
31 00000422 00000418 0000039f 000003c1 0000039a 000003e4 0000039e 00000300 000002ee 00000738
32 000005e1 00000562 000004d4 00000559d 00000673 000007ce 00000787 00000694 000005a5 000004bb
33 000003e0 0000043c 00000444 00000426 00000484 00000547 00000565 0000046f 0000039c 000003d3
34 0000046d 00000459 00000428 000004e2 0000056e 000002f9 00000309 0000033d 00000368 00000311
35 00000394 00000453 00000488 0000050a 000004d5 0000046c 000004a9 000004ca 00000449 0000040b
36 00000378 0000035d 0000029e 000002ab 00000205 00000239 000002a8 000003b1 0000044e 00000412
37 000004cf 0000048d 000004ee 00000507 00000554 00000538 00000533 000004b9 00000598 0000061c
38 0000064f 000006c6 0000076a 00000611 000006ec 00000734 000004f8 0000055a 00000663 00000900
39 00000868 00000701 00000661 0000057f 00000604 000005c2 0000063b 000005a0 0000062b 000006bf
40 000006f7 000007a0 000007fe 00000777 000006dc 0000061e 000004b6 000003fb 0000049f 00000492
41 000004ac 00000505 0000047b 0000046d 0000047b 00000457 000003c2 0000038d 000003f8 000003b9
42 0000037b 00000337 00000365 000003d8 00000350 000003a0 000003fb 0000052f 000005d8 000005df
43 00000616 00000643 0000072e 00000797 0000080d 000007ac 000004f8 00000496 000005ba 000005f6
44 0000058f 000003b2 00000476 00000456 00000490 000004b6 00000437 000004c8 00000614 000006b1
45 00000677 00000571 00000410 000003fd 0000059c 00000732 00000764 000007c8 00000748 000006c2
46 0000061c 00000567 00000543 00000456 0000043d 000004ab 000004d6 000004da 00000534 000005cb
47 00000603 000005e1 0000061a 0000063c 00000644 0000062b 0000059f 0000058d 0000069d 0000060f
48 00000617 00001a3b 000001e7 00000178 00002e80 000006a4 00000445 000002ed 00000299 000002ca
49 000004ff 00000578 0000047d 000002a7 000001b8 00000230 00000304 000002df 000001ca 00000167
50 000000dc 00000149 0000029a 00000294 000001bc 00000148 000000f2 00000141 000001b3 000000e6
51 0000018c 0000012e 000000d7 00000129 00000309 00000320 000001ba 00000136 000000d2 00000106
52 00000318 000002c2 0000019e 00000145 000000dd 00000126 00000314 00000309 000001b7 00000133
53 000000f1 000002c2 00000300 000002cf 000001bc 00000324 00000201 000001f2 00000282 0000018d
54 0000025e 000001f9 00000197 000001d7 000004be 00000451 000002b2 000000c5 000001a6 000001dd
55 000003c3 0000034e 00000176 00000112 000000b1 00000111 0000032a 000002dc 000001f7 00000130
56 000000d1 0000015f 000002c8 000002c1 000001e6 00000150 000002b0 0000026a 000002e2 000001d7
57 00000148 000001fd 000001a2 000001fb 00000233 000000b2 00000270 000003c5 00000100 00000144
58 00000144 000000ed 00000227 00000376 00000137 00000149 000001bd 0000027a 00000385 00000342
59 0000012f 00000172 00000347 000001ea 00000322 00000418 000001fa 0000023d 00000253 000001a6
60 0000024c 000001e1 00000168 000001e5 0000025f 0000038d 000004e9 00000215 00000125 0000014b
61 00000190 00000378 0000046a 00000210 0000011d 000013d 000001b5 0000035f 00000432 00000230
62 00000163 00000189 000001a4 000002b5 0000037e 00000253 00000300 00000283 00000287 0000019e
63 0000016a 000002e9 00000208 00000215 00000269 00000395 000004eb 00000354 0000021e 00000220
64 0000020c 000002eb 000003fe 000002e8 00000228 00000257 0000028d 00000371 00000431 00000305
65 00000208 0000021e 000001e2 000046a6 00000a32 00000798 00003db5 00000328 0000032f 0000043d
66 00000364 000001e2 0000025d 00000485 00000740 000005e7 0000032e 00000162 00000178 00000160
67 000000c4 000000d6 00000212 0000010c 00000175 0000016c 000011e2 000015ff 000003d9 000001f5
68 00000216 00000375 00001125 0000179e 00000519 000002d8 00000339 00000453 0000131c 000015f8
```

69 00000524 00000258 000002d5 0000043a 00001bf7 00001c4c 00000524 0000032b 00000531 000003e4
70 000001c2 0000023e 0000041b 0000023c 0000027f 0000027e 00001bd9 00001d0c 0000058f 000002e9
71 000004ec 000003f7 00001218 00001387 00000917 000004ef 000002d3 00000559 000005da 000001b2
72 000004f4 00000958 000006d2 00000576 000003ac 00000222 000004c3 000008fc 000002c4 000003aa
73 000003ef 0000022b 00000424 00000249 00000135 000002c1 00000877 000008f0 000004f9 0000027a
74 000001a2 0000059f 000005ef 000003fe 00000638 00000810 00000308 0000056e 000005b9 00000302
75 000005f7 000007a6 0000032d 0000044a 0000055d 00000305 0000059d 000006e3 0000029a 0000048a
76 00000573 000002e3 00000538 0000032b 000001ad 000003f9 000004ac 000003a4 00000577 000007a2
77 000002a7 00000461 0000049a 000002ff 0000059f 0000061f 000003ba 000005e0 000005eb 0000032e
78 00000608 0000074a 00000362 000004d9 00000593 00000354 00000703 000006ea 000002f9 000004b0
79 0000056a 00000327 000005d9 00000349 000001e0 00000432 00000518 0000032b 00000701 0000069c
80 0000023e 000003c2 00000459 000003e7 0000063f 00000644 00000369 00000455 000004c3 0000039a
81 0000062d 00000610 0000026d 00000465 00000512 0000042d 00000492 00000305 0000054f 000004a3
82 00000507 000001f2 00000287 000005fb 000002ae 0000037c 00000429 000001f8 00000224 000005a8
83 00000871 00000460 000004f0 00000220 00000220 00000664 000009ac 0000047a 000004d7 0000031d
84 0000029c 00000831 00000828 00000520 00000442 00000313 000002fb 0000089d 00000857 00000590
85 000004e4 00000392 00000373 0000073e 0000027b 000003da 000003d2 00002f14 000004b6 00000251
86 00003d02 000003a8 000006e7 000008e1 00000629 0000036b 000003a1 0000047a 00000695 000006c2
87 000004a7 000002ba 000001ed 000002ab 0000044a 00000513 000002d9 0000018c 000001d6 00000242
88 0000033a 0000048f 000002e8 0000019b 00000282 0000029d 000000b8 000000e2 00000252 0000013e
89 000001c0 000001df 00000379 0000054e 0000038c 000001e0 00000203 0000020e 000002aa 000003d1
90 00000319 0000032b 000002ad 0000032c 000003c7 000004c9 000003c4 000001fc 0000023d 0000023c
91 0000056c 00000762 000003de 00000242 00000257 0000029a 00000111 0000016c 00000336 0000022b
92 00000224 0000023f 0000169e 00001a9d 0000057f 000001c0 00000341 00000310 00001297 000019fc
93 000005f6 0000038a 000003eb 00000377 00001517 000017e1 000005cc 000002c7 00000367 000002f6
94 00001c06 00001c40 00000528 0000032e 000004d8 00000399 000001e0 0000015c 00000313 0000016d
95 0000021f 000001f1 00001bbf 00001a81 000004dc 00000291 0000048b 000003ea 00000c49 00000e47
96 0000062e 00000318 000001a8 000004e2 00000369 000001cb 00000481 000008bd 00000298 00000378
97 0000530d 000010aa 00000fd1 000040d0 00000c44 00001351 00000b9d 00001689 00001435 00000f5d
98 00000d79 00001290 00000afc 0000103a 00000c6c 000005ea 0000074a 000007b1 0000096f 00000c12
99 00000a74 000008ed 000008ac 000008fe 0000089c 00000c79 00000dc0 00000a66 00000a58 00000b74
100 00000768 00000933 00000dfc 00000976 00000a0f 00000b65 00000b93 00000938 00000a34 00000880
101 0000089c 00000970 000006d1 00000b90 00000d29 00000916 0000095d 00000b39 0000097b 00000ac2
102 00000c5e 000009ac 00000a17 00000881 0000091b 00000e30 00000eb1 00000abe 00000a6b 00000a5e
103 0000085d 00000b07 00000e65 00000af9 00000b1b 00000c42 00000896 00000636 00000b23 000007b2
104 00000775 00000882 00001e44 00000258 00000291 00003a47 00000657 00000760 00000669 00000582
105 00000518 0000097b 000004c9 000003b3 000002d0 0000037f 000002a7 0000016b 000021a 000002c0
106 00000395 00000405 00000306 0000024c 00000232 00000111 000001d2 0000021f 0000010e 00000105
107 00000696 00000430 00000264 000001b3 000002b0 00000259 000001d7 00000292 0000052f 000003fb
108 00000344 000002eb 0000044a 000002e5 000001e6 0000019e 000002d5 0000023b 0000016b 0000149d
109 0000175d 0000060f 00000332 00000392 00000539 00000391 00000356 00000338 0000035a 00001bc5
110 000003bb 000003c4 000003ca 000002c1 00000245 00000291 00000351 00000266 00001218 000014d1
111 000005a9 000002d7 000003f8 0000032e 0000032d 000002fe 000001b6 00000290 00001c56 0000034f
112 0000042e 0000048c 000003a2 0000022f 000001dc 00000320 000002ff 00000180 00000158 0000040e
113 000002d3 000001fd 0000011d 0000026a 000002b7 00000174 000000ef 00001ccb 000003ba 0000054e
114 000003f8 00000610 00000513 00000303 000003d0 000003b4 00000c4f 00000e40 00000634 00000520
115 00000551 000002c9 0000036e 000003c4 00000362 0000042d 00000420 000003b5 00000443 00000402
116 000009a5 000004b6 00000376 000002f6 0000041e 00000410 00000479 00000273 0000044c 0000085f
117 00000662 0000030a 0000037e 00000344 000003ac 0000023f 00000342 00000385 000002f2 000002cc
118 0000039c 0000029c 000001fa 00000200 000003fa 00000b0d 00000325 00000352 00000465 000006e2
119 00000482 000002ef 00000476 000005cc 00000398 00000432 00000632 000004fb 00000702 000007d3
120 0000066f 00000541 000003a6 00000583 000005ce 00000411 000004d4 00000660 000008a1 000005f4
121 00000335 00000444 00000501 0000043a 00000415 00000587 000002dd 00000420 0000038d 0000038c
122 0000036b 0000024b 0000039d 000003d8 000002e8 000002d3 000002de 000003b4 00000394 000001de
123 000001dd 0000038e 0000032c 00000282 00000236 000003ef 00000670 0000044f 0000022e 00000357
124 000002fd 00002cda 00000e3b 00001388 000020c7 000006e4 00000970 000009e6 00000897 0000077b
125 00000830 00000afc 00000a74 0000098a 000007d7 00000627 00000760 00000910 00000978 0000079e
126 00000837 000005a5 000009ca 00000b92 00000b0e 000007c5 000009d1 000005c8 00000acd 00000c74
127 00000bb3 00000a72 00000d51 000009f0 00000c08 00000b0b 00000bb6 00000ade 00000d0e 00000824
128 00000c1d 00000c57 00000eb8 00000f55 00000e68 00000aae 000011fb 0000111c 0000132a 0000109b
129 00000f16 00000c1a 00000fb7 00000eb6 00001075 00000fd3 00000e55 00000c2b 000010da 00000ebd
130 00001055 00000eff 00000da2 00000976 00000b62 00000a73 00000a04 000008e5 00000902 000007c3
131 00000924 00000ba0 00000b2e 00000aaa 00000ab7 0000090f 00000924 000009d5 000009ce 000009ca
132 0000085d 000006b7 000006ee 00000847 00000837 00000866 000008b5 00000692 000006bb 000007f9
133 00000831 0000061c 000005aa 0000050c 00000665 00000758 000006be 000004dd 00000628 00000517
134 0000054d 00000782 00000809 0000079b 000008a9 000007be 000009d9 00000acd 00000aad 00000973
135 00000c26 00000820 0000090f 000009ad 000009e5 00000ac1 00000bec 00000925 0000071b 000008a8
136 00000b6a 000009ba 00000886 000008a9 0000094e 00000a39 00000ad0 000009d6 00000a67 0000075b
137 00000862 0000089c 00000894 000008d7 00000a2f 000007ed 000006e4 000006ee 0000074f 00000777
138 00000875 00003485 000003ce 000003e2 00005c95 00000573 000006b4 000006d2 000006ca 0000082a
139 00000842 000007cb 000004d1 000003f5 0000051a 00000866 00000219 0000037a 000003c2 000003f6
140 00000499 000004e0 00000403 000003a5 000003f3 000003dc 0000033b 0000037b 000003f7 000005da
141 0000044f 00000555 000004da 000004a1 000004ac 00000569 00000720 0000068a 000006c4 00000713
142 0000060a 000004d3 00000578 0000091b 000005c2 00000634 000006f5 000007fa 0000074e 00000602
143 00000822 000006e6 00000745 00000727 00000584 000005aa 00000b4c 000007e0 000006a8 00000611
144 000006ca 00000633 0000029b 000003a4 00000451 00000423 00000415 00000518 000004f9 0000046d
145 000004f4 000004c3 0000049e 00000401 000003ab 00000463 0000038d 00000331 00000382 00000416

146 00000323 0000042b 000004d9 00000499 00000410 00000878 00000665 0000063f 000005e2 0000054d
147 00000535 000005c9 00000572 00000706 0000064b 00000623 000006ba 00000770 00000760 000007c8
148 000007bf 000006e0 000006ce 0000075b 000007c6 000007f2 00000866 00000813 00000741 000006fd
149 000006fa 00000718 000007cc 000006d1 000006f9 00000719 000006b3 000003b2 00000431 0000040b
150 0000047e 000004cd 0000059c 000005e3 0000060f 000005c0 000005ef 0000068b 0000065e 00000727
151 000006cf 00000635 00000681 00000648 00000688 00000762 0000078c 000005d7 000005e5 000005d5
152 00000585 0000069f 000006d1 00000636 00000772 00000838 00000e65 00000abc 000009c6 0000081e
153 000008a5 0000083a 000007c6 000008af 000008f2 000007ff 0000086d 0000091c 00000846 000008c8
154 000008aa 000008ae 00000785 0000071b 000006e8 000007b2 000006c7 00000588 00000653 000006c8
155 000005cb 000005a5 000003b9 00000420 00000347 00000314 0000035e 0000039a 00000319 00000297
156 0000027c 0000031f 00000342 000002c0 00000397 00000351 000002db 00000312 000002fd 000002d7
157 00000707 00000497 0000056c 000006d6 0000063c 0000059c 00000622 0000096c 00000906 00000829
158 00000877 0000060c 000005f8 00000590 000008e6 00000736 000006e8 0000068d 00000844 0000066e
159 000002f7 000003e6 00000468 000004d4 00000541 00000527 00004d8d 000012b4 00001798 0000488e
160 0000053f 00000650 00000881 00000a27 00000dd3 000008ad 00000cc5 00000ca6 00000a5a 0000095f
161 000011d3 000005d6 00000713 00000818 00000891 00000699 0000089f 00000991 00000723 000007d6
162 0000089e 000006ea 00000812 00000871 0000086b 00000850 00000883 00000718 00000dec 00000822
163 0000088f 000008fb 000009ab 00000860 00001039 00000b14 00000f1a 00000c72 00000fc7 00000a5f
164 00000ffb 00000b1a 00001439 0000126e 0000144c 00000e63 0000157e 00000bcb 000013eb 00001024
165 000014b8 00000ef6 00001079 000018a5 0000158d 00001211 000016cd 00000e0b 0000102c 0000174c
166 00001288 000013a1 00001445 00000bd0 00000cd3 00001371 000010c8 0000105f 0000105f 00000c67
167 00000bb8 00000f73 00000c81 00000c7b 00000e9b 00000a3b 00000a1a 00000eb8 00000c80 00000c32
168 00000eca 000006c1 00000aa2 00000e13 00000c18 00000b5a 00000eaf 0000074c 00000953 00000adb
169 00000ae4 00000bc3 00000def 000007c7 000009de 00000bab 000008b6 00000820 00000900 00000547
170 000009eb 00000b4c 00000d99 000009d0 00000ca0 000006ea 00000bb3 00000e81 000009f6 000007e8
171 000008da 000005b0 000007ce 00000ac4 00000807 00000ad4 00000a0b 000006f7 000009e7 00000b98
172 00003cd3 00000370 0000060c 000053f2 0000068a 000008ef 00000ff0 000009f8 00000b21 00000c07
173 00000d9e 00000fab 00000c12 0000081c 00000cff 000004d5 000007e5 00000870 00000a2c 0000064a
174 000006d5 000008a9 0000085e 000004e3 00000826 00000564 00000684 00000610 0000077e 000007cd
175 00000da4 00000704 00000812 00000783 0000098a 000009cb 00000c63 00000775 00000732 00000701
176 00000bca 00000b7f 00000a08 0000070e 000008f3 0000096b 00000b23 000009e7 00000b87 00000881
177 0000074f 000008ca 00000a48 00000b1d 00000b32 00000a65 0000097f 0000091a 000007d0 0000080e
178 0000094f 0000079c 000007b1 00000858 00000949 000007e4 000009a1 00000a74 000009c3 000009c2
179 00000984 00000a73 00000c2f 00000d0b 00000b25 00000b58 00000bbd 00000a77 00000aae 00000a86
180 00000a19 0000090f 000007f8 000007e7 00000991 00000aa3 000009dc 000008df 00000a7f 00000a47
181 00000b63 000009c1 00000994 0000090b 000009f0 00000a0f 00000c07 00000a4f 00000a5d 00000881
182 0000097a 000007ff 0000099d 00000a6f 000008ce 00000725 000006da 0000076a 000007f1 000009ac
183 00000721 000006ff 00000737 000007f2 000008a3 00000945 000008a3 000007fd 0000075d 000006d9
184 0000095a 00000a74 000008c7 00000a26 00000a69 000009e9 00000b36