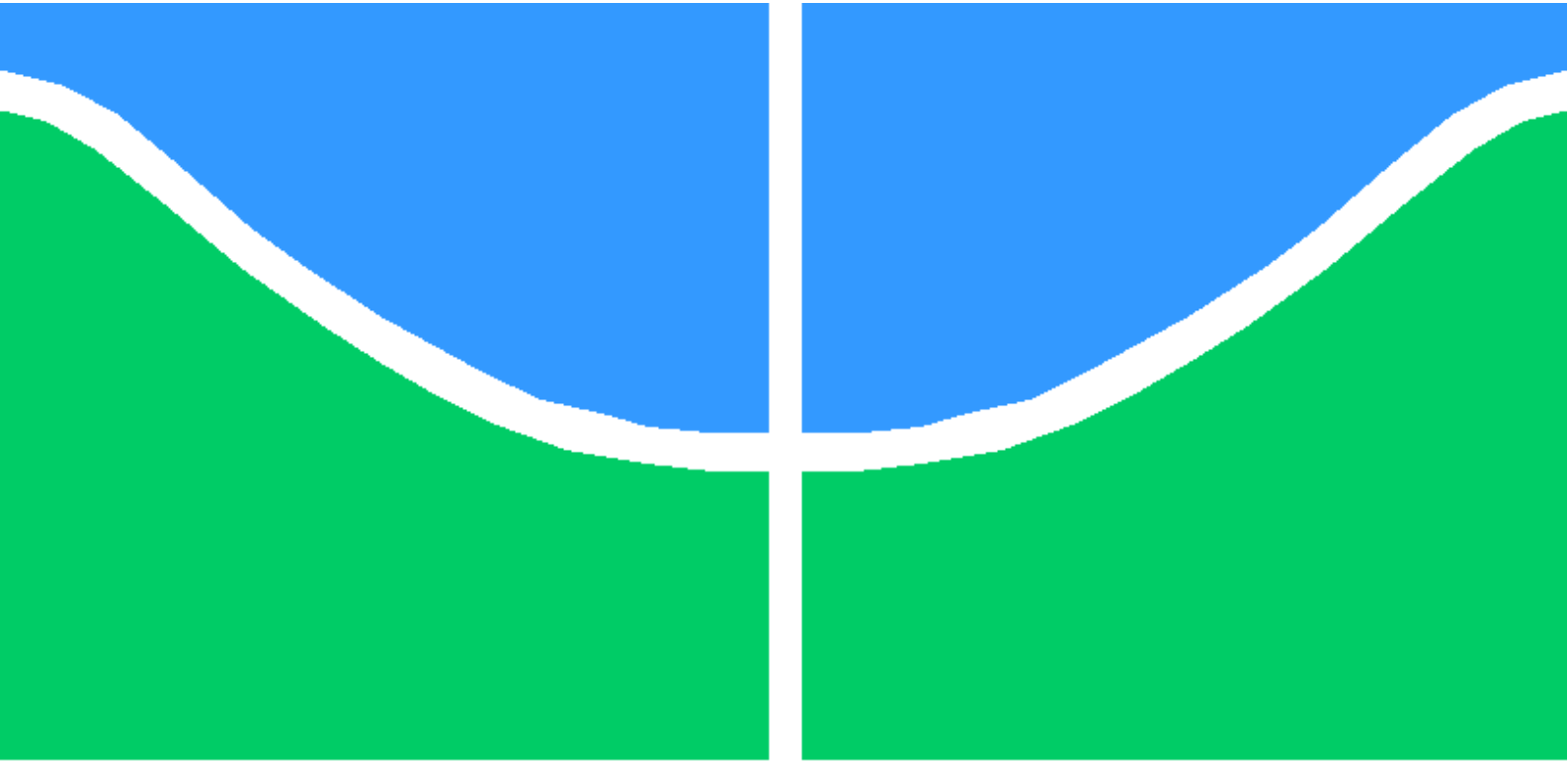


TRABALHO DE GRADUAÇÃO

CHAVEAMENTO DE STREAMS DE VÍDEO CODIFICADAS NO PADRÃO H.264/AVC

Francisco Borges de Pina Amorim

Brasília, dezembro de 2009

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

Dedicatória

Dedico este projeto à minha família, forte pilar responsável pela minha sustentação e meu crescimento.

Francisco Borges de Pina Amorim

Agradecimentos

Por esta conquista, agradeço à minha família por me dar dar suporte e incentivo para chegar a este ponto, em especial agradeço aos meus pais, Argemiro e Mânia, e ao meu irmão, Múcio. Agradeço também à minha namorada, Juliana, pelos anos de apoio e carinho. Importante ressaltar a presença de grandes amigos, como Germano, Rafael e Pedro (República do Iraque) e do curso de Engenharia Elétrica, que me apoiaram nas horas difíceis.

Também agradeço ao professor Ricardo Queiroz, ao Edson Mintsu e a todos do Grupo de Processamento Digital de Sinais da Universidade de Brasília pela paciência e orientação durante a execução do projeto.

Francisco Borges de Pina Amorim

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

CHAVEAMENTO DE STREAMS DE VÍDEO CODIFICADAS NO PADRÃO H.264/AVC

Francisco Borges de Pina Amorim

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Ricardo Lopes de Queiroz, ENE/UnB _____
Orientador

Edson Mintsu Hung, ENE/UnB _____
Examinador Interno

Prof. Ricardo Zelenovsky, ENE/UnB _____
Examinador Interno

RESUMO

O projeto realizado visou encontrar uma maneira de fazer o chaveamento de *streams* de vídeo codificadas no padrão H.264/AVC (MPEG-4 Part 10), de forma que se possa alternar entre estas, sem que haja erro causado pela interpretação errada de bits das sequências de vídeo durante o processo de decodificação. Para isso foram feitas pesquisas e testes visando encontrar as fontes de erros presentes quando o chaveamento é feito sem critérios e também visando encontrar quais os critérios que devem ser adotados.

ABSTRACT

The project aimed to find a way to switch between different video streams encoded with H.264/AVC (MPEG-4 Part 10) standard, without errors during the decoding process of the video sequence. Research and tests were done to find the sources of errors and also to find what criteria should be adopted to switch between these video streams.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A codificação de vídeo representa um grande avanço na mídia como um todo e os seus progressos propiciam cada vez mais a utilização deste recurso nos diversos meios de comunicação existentes. Aplicações de vídeo em sites na internet e telefones celulares, além da televisão digital, são exemplos de pontos em que a codificação de vídeo se mostra necessária, podendo adaptar a qualidade do vídeo, a taxa de bits, a resolução, entre outras características do vídeo, à finalidade, ao equipamento e à banda que está sendo utilizada.

A compressão permite que filmes e vídeos em geral possam ser comprimidos a tal ponto que se permita sua gravação em mídias como o DVD e o Blu-Ray, com alta qualidade. Além disso, sem a codificação de vídeo fica inviável a utilização de meios, como a internet, para sua transmissão de foma rápida, já que esta, apesar de termos crescentes taxas de envio e recepção de bits, não suportaria transmitir vídeos não comprimidos. Pode-se também dizer que a compressão torna mais eficiente o uso da taxa de transmissão disponível em algum recurso, já que diminui drasticamente a taxa de bits por segundo de uma sequência de vídeo.

Outro ponto importante que justifica o estudo e execução deste projeto é a expansão do uso da televisão digital no Brasil. Este fato é importante por representar que o país utiliza a mais moderna tecnologia de transmissão desse meio de comunicação presente até o momento. Mas o mais importante foi o passo dado quando foi tomada a decisão de se utilizar um sistema próprio de televisão digital, o qual seria construído baseado no sistema japonês ISBD, mas que teria muitas mudanças e inovações a serem feitas no Brasil, inclusive a utilização de um outro padrão de codificação de vídeo, o H.264, quando no Japão é utilizado o MPEG-2. Tal fato representou a abertura de um leque de oportunidades e incentivos dentro das universidades e empresas para o desenvolvimento de sistemas, aplicativos, hardware e software para alimentar este novo Sistema Brasileiro de Televisão Digital, o **SBTVD**.

Neste contexto, a codificação de vídeo é elemento fundamental para fornecer o conjunto "qualidade e eficiência" a esses meios de comunicação. Podemos dizer que, por exemplo, a codificação permite a transmissão de um vídeo, comprimido, praticamente em tempo real e em alta definição, por um canal de televisão aberta, que ocupa apenas 6MHz de banda. Um exemplo disso é a transmissão feita por canais da rede aberta quando transmitem um jogo de futebol, ao

vivo, em alta definição.

Dentro da realidade apresentada, o padrão de codificação de vídeo escolhido para o SBTVD foi o denominado **H.264/AVC** (*Advanced Video Coding*) ou **MPEG-4 Part 10**. Tal padrão pode ser considerado o mais completo em recursos para aplicações de vídeos, mas há, nesse caso, maior grau de complexidade e exigência quando o comparamos com o padrão MPEG-2 (mais usado nos demais países).

1.2 DEFINIÇÃO DO PROBLEMA

Um *stream* de vídeo codificado é uma sequência de bits reunidos e quando interpretados da forma correta nos dão como resultado uma sequência de *frames* que representam o vídeo. Um vídeo codificado no padrão H.264/AVC é um exemplo de *stream* e também precisa ser propriamente interpretado. Sendo assim, quando se quer chavear de um *stream* de vídeo a outro, não se pode simplesmente fazê-lo sem qualquer critério, já que os bits têm seu próprio significado e ordem dentro de sua sequência. Vários erros ocorrem quando se faz esse chaveamento desta forma, mostrando que, para o decodificador, não se pode simplesmente interromper um fluxo de bits de uma sequência de vídeo codificado e começar a receber os de uma outra sequência sem qualquer critério.

Durante o processo de codificação de um vídeo, são criados certos vínculos entre imagens de *frames* próximos, isto ocorre pois neste momento determina-se que algumas das imagens serão reconstruídas a partir de outras. Tais vínculos são transmitidos juntamente com dados do vídeo codificado e são usados pelo decodificador para resconstruir a sequência de vídeo original. Quando trocamos de um fluxo a outro de bits de duas sequências de vídeo diferentes em um ponto errado, estamos quebrando estes vínculos, gerando inúmeros erros durante o processo de decodificação. Assim, podemos dizer que existem alguns problemas relacionados ao se fazer o chaveamento de *streams* sem critério.

A Figura 1.1 mostra exemplos de sequências de vídeos, com as quais poderia se fazer o chaveamento. A tentativa seria de se alternar de uma sequência para outra como está mostrado no exemplo da Figura 1.2. Mas a realidade é que nem sempre é tão simples fazer o chaveamento desejado, já que, como já foi dito, a falta de critérios para se fazer o chaveamento causa erros no processo de decodificação da sequência formada após o processo.

(a) Quadro 1 (b) Quadro 2 (c) Quadro 3 (d) Quadro 4

(e) Quadro 1 (f) Quadro 2 (g) Quadro 3 (h) Quadro 4

Figura 1.1: Exemplos de sequencias de vídeo.

(a) Quadro 1 (b) Quadro 2 (c) Quadro 3 (d) Quadro 4

Figura 1.2: Exemplo de uma nova sequência formada pelas duas sequências da Figura 1.1.

1.3 OBJETIVOS DO PROJETO

O projeto visa encontrar uma forma de se alternar entre duas *streams* de vídeo codificado no padrão H.264, sem que haja erro no processo de decodificação. Para isso, existe o objetivo de estudo e pesquisa em livros e normas, para se ter uma forma de localização, na sequência de bits de vídeo codificado, do momento certo de se fazer o chaveamento, fornecendo à nova sequência de bits (composta por pedaços das outras sequências) uma lógica para que essa seja decodificada sem que haja erro. Nesta abordagem o foco não foi o de realizar este chaveamento de forma remota, e sim o de interpretar a estrutura de uma sequência de vídeo codificada no padrão H.264 e em quais pontos dessa o chaveamento pode ser feito sem que ocorram erros.

Dentro do projeto existe sempre a tentativa de identificar as fontes de erros presentes como forma de combatê-las e eliminá-las, elegendo, assim, a melhor forma de se chavear os *streams* de vídeo codificado no padrão H.264.

1.4 APRESENTAÇÃO DO MANUSCRITO

A organização deste trabalho foi feita de forma que o Capítulo 2 (Revisão Bibliográfica) mostre, de forma breve, alguns conceitos que envolvem o vídeo e seu codificação. Além disso, o segundo capítulo contém também informações retiradas de normas e livros a respeito da estrutura do vídeo codificado no padrão H.264.

No Capítulo 3, os passos do desenvolvimento do projeto estão detalhados, desde a procura teórica pela solução do problema até a forma como foram planejados os testes e obtidos os resultados.

Em seguida, no Capítulo 4, temos a descrição detalhada dos testes realizados, os resultados obtidos e análises feitas. Neste capítulo algumas bateria de testes foram realizadas.

No quinto e último capítulo, são feitas algumas conclusões a respeito dos testes realizados, contendo análises e considerações finais.

2 REVISÃO BIBLIOGRÁFICA

2.1 INTRODUÇÃO

Para a melhor compreensão deste trabalho como um todo, é importante esclarecer, de forma sucinta, alguns conceitos dentro da codificação de vídeo de forma que os interessados no projeto, mesmo não tendo estudado a fundo os temas que serão abordados, consigam compreender o problema, os objetivos, os métodos utilizados, os resultados dos testes e as conclusões que serão apresentadas no decorrer deste trabalho.

2.1.1 Amostragens Temporal e Espacial

Uma sequência de vídeo digital é formada por uma sequência de quadros (*frames*), ou seja, uma sequência de fotos tiradas em pequenos intervalos de tempo e que representam o vídeo. Desta forma, a captura deste vídeo é feita através de amostragens espacial e temporal.

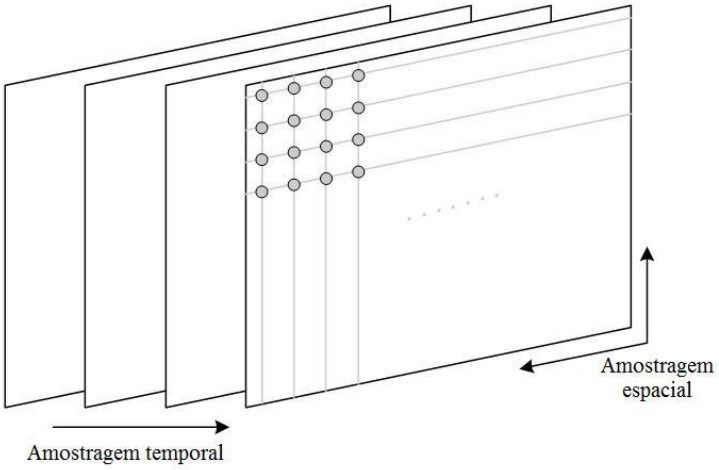


Figura 2.1: Amostragens temporal e espacial [1].

A **amostragem temporal** fornece as diversas imagens capturadas em certo período, ou seja, a amostragem é feita através da construção de uma sequência de quadros que irão representar o vídeo.

A **amostragem espacial** pode ser representada pela resolução da imagem/quadro. O quadro é representado por um conjunto de elementos de imagem chamados pixels (*picture elements*). A quantidade de pixels em uma imagem pode representar uma maior quantidade de detalhes, bem como uma maior quantidade de informações contidas na mesma.

2.1.2 Espaço de Cores

Outro conceito útil a ser apresentado seria o conceito de espaço de cores. Trata-se do método para a representação do brilho e das cores de uma determinada imagem. Para se representar um pixel colorido de forma precisa, são necessários, no mínimo, três números [1]. Desta forma, o espaço de cores pode ser definido como sendo uma forma de separar componentes de cores dos elementos da imagem.

Um conhecido método de espaço de cores é o **RGB**. A sigla se deve ao nome das cores vermelho (*red*), verde (*green*) e azul (*blue*). Este método consiste na existencia de três componentes distintos de cor que representam estas cores e que quando misturados, variando a proporção de cada, podem representar qualquer cor que se queira.

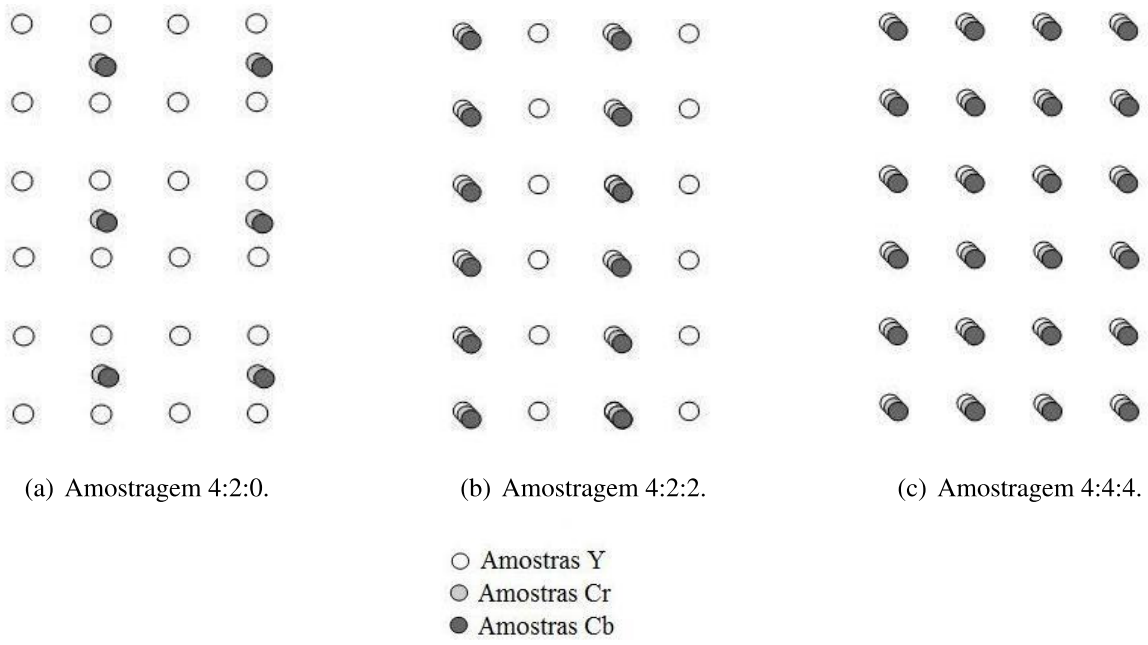
Outro espaço de cores existente é o chamado **YCbCr**, onde os três elementos agora são a luminância (Y), responsável pelo brilho do elemento da imagem e os elementos de crominância (Cb e Cr), responsáveis pelos elementos de cor, representando a diferença entre o valor da cor e o valor da luminância (Ex. Cb = B - Y).

2.1.3 Amostragens

Como dito em [1], descobriu-se que o brilho sensibiliza mais os olhos humanos do que as cores. Desta forma, visando a redução na quantidade de informação presente no vídeo, diferentes tipos de amostras são feitas. O tipo de amostragem utilizada no projeto foi a 4:2:0. Ela indica que para cada 4 amostras de Y, teremos 1 amostra de Cb e 1 amostra de Cr. A Figura 2.2 mostra com mais clareza o conceito, mostrando também exemplos de outros tipos de amostragens que também são usadas.

2.2 A CODIFICAÇÃO DE VÍDEO

A codificação de vídeo é elemento decisivo para tornar mais eficientes o uso de recursos como mídias de armazenamento, banda de transmissão, qualidade de vídeo, entre outros. Entrando em alguns dos conceitos que envolvem a codificação de vídeo, devemos ressaltar alguns.



(a) Amostragem 4:2:0. (b) Amostragem 4:2:2. (c) Amostragem 4:4:4.

(d) Tipos de amostras

Figura 2.2: Tipos de amostragem.

2.2.1 Codificador e Decodificador

Primeiramente, observa-se a presença de dois elementos base para a utilização dos recursos da codificação: o codificador e o decodificador. O codificador de vídeo tem o poder de comprimir ao máximo o vídeo ou manter a qualidade melhor possível. Esse balanço é feito de acordo com a capacidade e características do codificador, juntamente com as necessidades da aplicação que irá utilizar o vídeo. Assim, este vídeo poderá ser armazenado ou transmitido com eficiência, de acordo com o seu propósito. Um codificador pode ser constituído, basicamente, de três funções básicas: modelo temporal, modelo espacial e codificador de entropia [1]. Tal codificador será usado como exemplo por ser similar ao codificador do padrão H.264 que é utilizado no projeto.

2.2.1.1 Modelo Temporal

O **modelo temporal** utiliza algoritmos que visam eliminar redundâncias entre *frames* que sejam temporalmente próximos, explorando a correlação entre eles, resultando em redução da quantidade de informação a ser transmitida/armazenada. Desta forma, o modelo temporal utiliza *frames* anteriores e/ou posteriores a um *frame* atual para construir uma predição deste.

As mudanças entre *frames* podem ser causadas por movimentos de objetos, alterações na iluminação, movimento de câmera, entre outros. Quando há pequenas alterações entre *frames* dentro de uma sequência de vídeo, como o movimento de um objeto, pode ser considerado que os

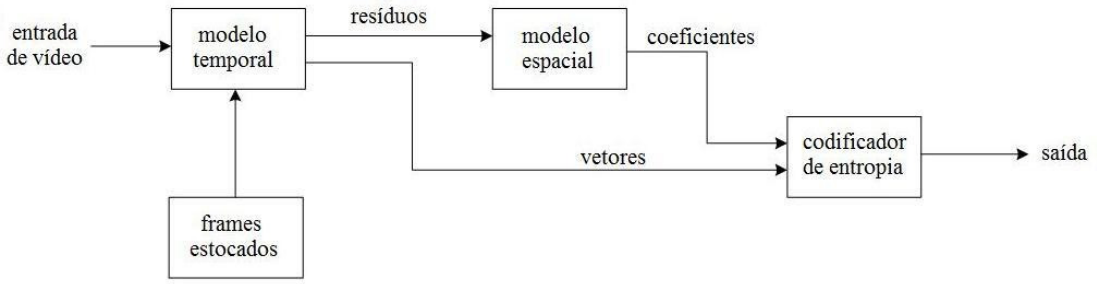


Figura 2.3: Um exemplo de codificador básico.

pixels que formam aquele objeto se moveram entre um *frame* e outro. Esse fluxo de pixels, entre os *frames*, pode ser representada por vetores, os quais fazem parte de um diagrama que chamado de *optical flow*. Este diagrama representa a movimentação de cada pixel entre um *frame* e outro.

Desta forma, utilizando um *frame* de referência juntamente com um diagrama como o da Figura 2.6, é possível se ter uma predição do que seria o *frame* atual.

A construção de um diagrama de vetores de movimento para todos os pixels torna-se praticamente inviável por se tratar de um processo que exige um esforço computacional muito grande, além de representar um acréscimo na quantidade de informação que deverá ser armazenada/transmitida. O que é feito nos padrões atuais (como o H.264) é utilizar-se de blocos de pixels para se fazer este processo. Dentro desta realidade existe uma unidade básica utilizada para este modelo: **o macrobloco**. Trata-se de uma região de 16x16 pixels, que é utilizada como elemento básico para se realizar as predições dentro dos codificadores. No caso do padrão H.264, trata-se especificamente de uma região com 16x16 amostras de luminância (Y) e 8x8 amostras de cada crominância (Cb e Cr). Mesmo sendo o elemento básico utilizado para as predições, dentro do padrão H.264 pode-se fazer predições utilizando blocos menores que um macrobloco.

Para se construir o *optical flow* utilizando os macroblocos, é feita uma procura em uma ou duas imagens de referência, do macrobloco que melhor combina com o macrobloco atual (pertencente ao *frame* atual). Um critério que é usado com frequência é o de se eleger o macrobloco do quadro de referência que, quando subtraído do macrobloco atual, resulta em macrobloco residual de menor energia possível. Essa procura e combinação é chamada de estimação de movimento.

Feita essa estimação, o macrobloco eleito pertencente à imagem de referência é subtraído do atual, resultando em um macrobloco residual. Assim, a saída do modelo temporal é constituída basicamente de vetores de movimento e de resíduos.

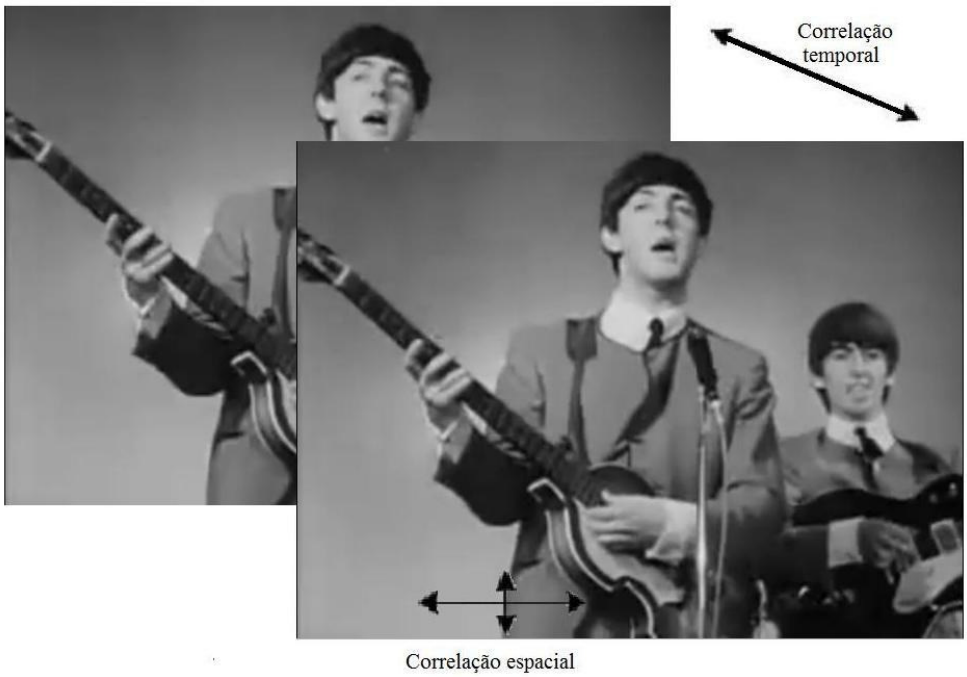


Figura 2.4: Correlação espacial e correlação temporal.

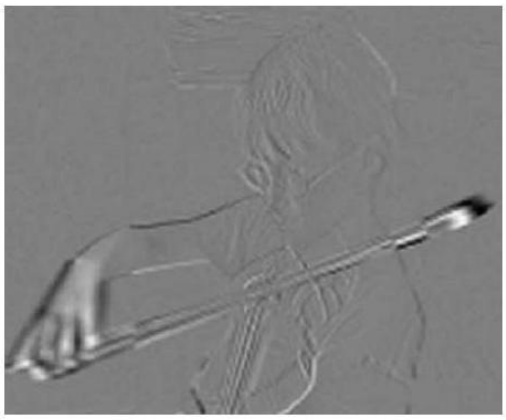
2.2.1.2 Modelo Espacial

O **modelo espacial** utiliza a correlação espacial dentro de um mesmo quadro para eliminar redundâncias presentes neste. Em muitos modelos, como no caso do modelo utilizado na codificação H.264, é utilizado o conjunto transformada e quantização para se fazer tal processo.

Aqui utiliza-se macroblocos dentro de um mesmo quadro ou resíduo codificados anteriormente como base para reconstruir o macrobloco atual de forma aproximada. Este macrobloco reconstruído é chamado de predito e pode ser formado por uma combinação dos demais macroblocos codificados anteriormente. Esse macrobloco predito é subtraído do atual formando um macrobloco residual, que é a informação que será codificada e transmitida. A transformada leva os valores dos resíduos e amostras para um outro domínio e serão representadas agora por seus coeficientes. Esses coeficientes são quantizados, um processo que diminui o alcance dos valores de entrada deste processo, diminuindo também a quantidade de bits que necessária para representar estes valores. Alguns valores insignificantes podem ainda ser removidos, reduzindo-se ainda mais a quantidade de informação presente no vídeo codificado.

(a) *Frame* 1.

(b) *Frame* 2.



(c) Diferença entre os *frames*

Figura 2.5: Exemplo de *frames* e a diferença entre os mesmos [1].

2.2.1.3 Codificador de Entropia

Tais coeficientes quantizados, juntamente com vetores de movimento originados no modelo temporal, são agora processados pelo **codificador de entropia**. Agora, são as redundâncias estatísticas dentro dos dados a serem eliminadas para se diminuir a quantidade de informação do vídeo.

O decodificador tem o papel de reconstruir o vídeo a partir da sequência de bits que representa o vídeo codificado. Tem basicamente funções que desfazem o que é feito no decodificador para reconstruir os quadros codificados. Ele utiliza um decodificador de entropia para restaurar os vetores gerados no modelo temporal e para reconstruir os coeficientes gerados no modelo espacial. Assim, com os coeficientes, ele reconstrói os resíduos e amostras do modelo espacial, utilizando macroblocos decodificados anteriormente. A partir daí é realizada a reconstrução

Figura 2.6: *Optical flow* entre os *frames* 1 e 2 da Figura 2.5 [1].

dos quadros e resíduos do modelo temporal. Utilizando os vetores de movimento, resíduos e imagens decodificadas anteriormente, os macroblocos e as imagens são reconstruídas de forma semelhante às originais.

2.3 O PADRÃO H.264/AVC OU MPEG-4 *PART* 10

O padrão H.264 representa uma possibilidade de se obter melhores resultados em termos de compressão e qualidade [1], mesmo representando um maior grau de complexidade, quando comparado com outros padrões comuns, como o caso do MPEG-2.

2.3.1 Perfis

O padrão H.264/AVC tem como característica a presença de alguns perfis em sua estrutura. Estes perfis determinam uma série de parâmetros particulares que estão presentes dentro de cada um deles. Inicialmente, foram criados três perfis dentro do H.264, chamdos de *Baseline*, *Main* e *Extended*. Mais tarde foram adicionados outros perfis, *High*, *High 10* e *High 4:2:2*, cada um com suas peculiaridades. A Tabela 2.1 mostra as características e diferenças básicas entre estes perfis.

Tabela 2.1: Perfis do H.264 [2]

Perfil	**Descrição**
Baseline	Voltado para aplicações com pequena complexidade computacional e mínimo de erro.
Main	Voltado para aplicações com máxima eficiência de compressão.
Extended	Voltado para aplicações de *streaming* de vídeo.
High	Semelhante ao *Main profile*, mais voltado para vídeos de alta definição
High 10	Semelhante ao perfil *High*, mas apresenta suporte a amostras com precisão maior que 10 bits.
High 4:2:2	Semelhante ao perfil *High*, mas apresenta suporte a amostragem no formato 4:2:2 (Seção 2.1.3).

2.3.2 Predições Intra e Inter

A sequência de vídeo é codificada utilizando-se sua sequência de imagens, mas quando nos referimos a imagens, não necessariamente estamos dizendo *frames*, já que existe um conceito dentro da terminologia da codificação de vídeo, que representa uma fatia de *frame*: são os chamados *slices*. Vimos anteriormente que existe uma unidade básica utilizada para se fazer predição, chamada de macrobloco (bloco formado por 16x16 pixels). Os *slices* são fatias de quadros constituídas por macroblocos. O número de macroblocos nos *slices* não precisa ser constante, podendo variar de 1 até todos os macroblocos do quadro. Dependendo do perfil utilizado, quando os *slices* são codificados, seus dados podem ainda ser divididos em partições diferentes (Partição A, B e C), para aumentar a robustez a erros (usado no perfil *Extended*).

Os macroblocos presentes em um *slice* podem ser preditos a partir dos dados já codificados anteriormente. Quando o macrobloco é predito utilizando-se amostras já codificadas presentes em seu próprio *slice*, diz-se que este foi predito através da predição do tipo **INTRA**. Quando este macrobloco é predito utilizando-se amostras presentes em *slices* codificados antes dos *slice* atual, foi utilizada a predição do tipo **INTER**.

2.3.3 Imagens e Listas de Referência

Um conceito importante presente na predição INTER, refere-se aos *slices* e imagens utilizadas para se fazer este tipo de predição. Estas são chamadas **imagens de referência** e estão presentes em **listas de referência** que tanto codificador quanto decodificador constroem no mo-

mento de sua operação. Estas imagens são utilizadas como base para se fazer a predição INTER. Assim, são formadas duas listas descritas como lista 0 e lista 1, com um conjunto de imagens recentemente codificadas/decodificadas, para se consultar no momento de realizar a predição, tanto na codificação quanto na decodificação. Estas listas estão em constantes mudanças, dado que as imagens vão sendo codificadas/decodificadas e entram nas listas, enquanto outras vão saindo da lista com o decorrer do processo.

Como já foi citado anteriormente, essas predições são utilizadas para se criar os resíduos, que são codificados e transmitidos para o decodificador, de forma que, juntamente com dados como vetores de movimento, possa ser reconstruído o macrobloco atual.

Desta forma, como está exibido na Tabela 2.2, os tipos de macroblocos são definidos de acordo com o seu processo de predição. A Tabela 2.3, por sua vez, exibe os tipos de *slices* possíveis.

Tabela 2.2: Tipos de Macroblocos

Tipo	**Descrição**
I (Intra)	São os macroblocos preditos utilizando-se a predição INTRA, ou seja, utilizando dados de outras amostras já codificadas pertencentes ao presente *slice*.
P (Predito)	São os macroblocos preditos utilizando-se a predição INTER, ou seja, utilizando dados de outras amostras já codificadas presentes em *slices* codificados anteriormente. Tais macroblocos só podem ser preditos a partir de uma das imagens que pertencem à lista 0 de referência.
B (Bi-predito)	Da mesma forma que os macroblocos do tipo P, utilizam a predição do tipo INTER, mas neste caso, podem ser preditos utilizando-se uma ou duas imagens pertencentes às listas 0 e 1 de referência.

2.3.4 IDR - *Instantaneous Decoder Refresh*

Um tipo essencial de imagem codificada é a chamada *Instantaneous Decoder Refresh*, ou simplesmente **IDR**. Esta imagem é constituída por *slices* do tipo I ou SI e é utilizada para se limpar as listas de referência do decodificador. Ela indica que todas as demais imagens que irão ser decodificadas a partir dela, não dependerão das imagens que foram decodificadas antes dela, ou seja, ela sinaliza que as demais imagens presentes nas listas de referência não serão mais úteis para a decodificação das que vêm a seguir. A primeira imagem da sequência de vídeo codificada

Tabela 2.3: Tipos de *slices* [1]

Tipo	Descrição
I (Intra)	São *slices* que contém apenas macroblocos do tipo I.
P (Predicted)	Contém macroblocos do tipo P e/ou I.
B (Bi-predicted)	Contém macroblocos do tipo B e/ou I.
SP (Switching P)	Contém macroblocos do tipo P e/ou I. São *slices* que facilitam o chaveamento entre *bitstreams*. (Presente apenas no perfil *Extended*)
SI (Switching I)	Contém macroblocos do tipo SI, um tipo especial de macrobloco I, que é usado para facilitar o chaveamento entre *bitstreams*. (Presente apenas no perfil *Extended*)

é sempre uma imagem do tipo IDR.

2.3.5 GOP - *Group of Pictures*

O conceito de *Group of Pictures* (GOP) indica qual são os tipos de imagens e a sua ordem dentro de uma sequência codificada de vídeo. Por exemplo, pode-se codificar um vídeo no padrão H.264 com um GOP do tipo IBBPBBI, o que indica que a primeira imagem da sequência será do tipo I (contendo apenas macroblocos I), as duas próximas serão do tipo B (contendo macroblocos do tipo I e B), a próxima será do tipo P (com macroblocos do tipo I e P) e assim por diante.

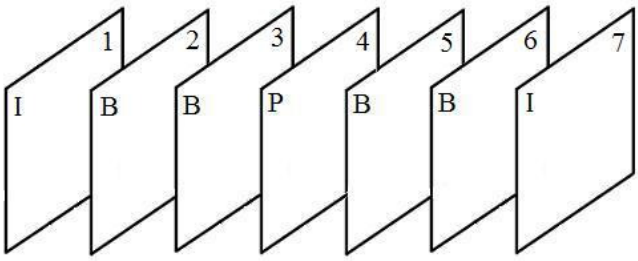


Figura 2.7: Exemplo de GOP (Ordem de exibição).

A ordem da codificação não obedece, necessariamente, à ordem de exibição das imagens, como a Figura 2.8 indica. Esta ordem de codificação das imagens é a responsável para que se possa usar como referência imagens que estão depois da imagem atual na ordem de exibição. A

figura também nos mostra, através da indicação das setas, a ligação entre as imagens do tipo B e P com suas respectivas referências.

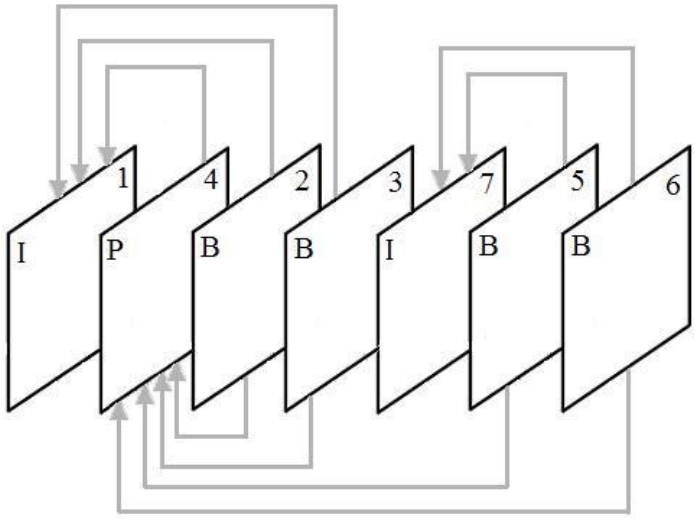


Figura 2.8: Exemplo de GOP (Ordem de codificação).

2.4 A ESTRUTURA DO *BITSTREAM* DE VÍDEO CODIFICADO NO PADRÃO H.264

Para a execução do projeto, foi extremamente importante o estudo da estrutura do *bitstream* de vídeo codificado no padrão H.264, já que para se fazer o chaveamento das *bitstreams* seria necessário conhecer sua sequência para construir ferramentas que fizessem o chaveamento no momento e do jeito correto.

Uma característica importante da codificação feita pelo padrão H.264 é a presença de duas camadas distintas, chamadas de *Video Coding Layer* (VCL) e *Network Abstraction Layer* (NAL). Estas duas camadas são úteis para a diferenciação os dados específicos da codificação, presentes na VCL, dos dados específicos do transporte, presentes na NAL. O processo da codificação fornece os dados para a VCL e tais dados são encapsulados através de um mapeamento e se tornam **unidades NAL** (NAL *units*). Assim, a sequência de vídeo codificado é representada por uma sequência de unidades NAL, as quais contêm um *header* que identifica o conteúdo desta, além de conter uma carga de sequência de bytes, chamada de *Raw Byte Sequence Payload*, ou

simplesmente RBSP, que contém dados do vídeo codificado, como dados de início e fim da sequência, parâmetros da sequência, formato do vídeo, *slices* codificados, entre outros.

A norma [3] determina características para a construção e a estrutura de cada unidade NAL. Algumas destas características são essenciais para a compreensão da saída encontrada para o problema proposto neste projeto, e serão listadas e explicadas a seguir.

2.4.1 *Start Code Prefix*

O *start code prefix* ou código de início representa uma sequência de bits que deverá sempre constar antes do início de cada unidade NAL dentro de uma *bitstream* de vídeo codificado no padrão H.264. Este código de início está representado pela sequência de bits a seguir:

00000000 00000000 00000001

Ou também:

0x00 0x00 0x01

Vale ressaltar que os codificadores têm sistemas que evitam esta sequência de bytes dentro de uma NAL unit, antes que essa termine. Assim, não é possível que o decodificador receba esta sequência sem que represente o ínicio de uma unidade NAL.

2.4.2 NAL *Header*

Outra importante característica padronizada pela norma é o formato do *header* da unidade NAL, um byte que sempre virá, no *bitstream*, depois do código de início e deverá obedecer ao seguinte formato:

Primeiro bit: forbidden_zero_bit (Um bit de sinalização que deverá ser sempre zero)

Segundo e terceiro bits: nal_ref_idc (Dois bits que sinalizam que a unidade NAL contém ou não conteúdo de uma imagem de referência. Se nal_ref_idc = 00, então o conteúdo não pertence a uma imagem de referência).

Quarto ao oitavo bits: nal_unit_type (Indicam o tipo da unidade NAL)

2.4.3 Tipos de unidades NAL

As unidades NAL podem ser de diversos tipos [3], como estão listados na Tabela 2.4. Para se saber qual é o tipo de uma unidade NAL dentro da sequência, pode-se utilizar a informação

contida em seu cabeçalho.

Tabela 2.4: Tipos de unidades NAL [3]

Tipo da Unidade NAL	**Identificador**
Não especificado	0
Slice codificado não IDR	1
Slice codificado da partição A	2
Slice codificado da partição B	3
Slice codificado da partição C	4
Slice codificado IDR	5
Supplemental enhancement information (SEI)	6
Parâmetro da sequência	7
Parâmetro da imagem	8
Delimitador da unidade de acesso	9
Fim da sequência	10
Fim do *stream*	11
Dado de preenchimento	12
Extensão de parâmetro de sequência	13
Prefixo de unidade NAL	14
Parâmetro de subconjunto de sequência	15
Reservado	16 ao 18 e 21 ao 23
Slice codificado de uma imagem auxiliar codificada não particionada	19
Extensão de *slice* codificado	20
Não específico	24 ao 31

3 DESENVOLVIMENTO

3.1 INTRODUÇÃO

O vídeo digital codificado nada mais é do que uma sequência de bits que carregam toda as informações necessárias para que um decodificador compatível com o padrão de codificação consiga realizar a tarefa de decodificar, recuperando a sequência de vídeo.

Desta forma, quando temos mais de um *bitstream* de vídeo codificado e queremos fazer o chaveamento entre eles, não podemos fazê-lo sem critérios. Cada bit tem a sua posição e a sua importância dentro de sua sequência e tirá-lo dessa é fazer com que o decodificador os interprete de forma errada. Se o decodificador não recebe a informação necessária para interpretar de forma correta os bits que estão sendo decodificados, o erro certamente ocorre.

3.2 LISTAS DE REFERÊNCIA E IMAGEM IDR

Como vimos em seções anteriores, tanto no processo de codificação quanto no processo de decodificação, é essencial a presença das imagens de referência para se realizar a predição INTER, responsável pela diminuição das redundâncias presentes entre quadros subsequentes com correlação temporal. O mecanismo presente neste tipo de predição utiliza imagens presentes nas listas de referência.

Também foi visto que o GOP (Seção 2.3.5) de uma sequência de vídeo pode contar com diversas imagens que são codificadas e decodificadas utilizando-se uma ou duas imagens presentes nas listas de referência. A Figura 3.1 mostra um exemplo de duas sequências de vídeo que, assim como na Figura 2.8, possui imagens que são codificadas e decodificadas com auxílio de imagens de referência. Como pode ser visto na Figura 3.2, que mostra um exemplo de chaveamento feito entre as sequências da Figura 3.1, o chaveamento entre as *streams* feito sem critérios tem grandes chances de gerar erros para o processo de decodificação. Um exemplo de erro pode ser notado quando se nota que, na Figura 3.2, o quadro 3, pertencente à sequência Y, irá ser decodificado a partir dos quadros 1 e 4, pertencentes à sequência X.

Ao se chavear entre duas *streams* em bytes quaisquer da primeira e da segunda, ocorre que o decodificador está com uma lista de referência contendo diversas imagens da primeira sequência de vídeo, quando começamos a decodificar imagens da segunda. Se não recebeu alguma infor-

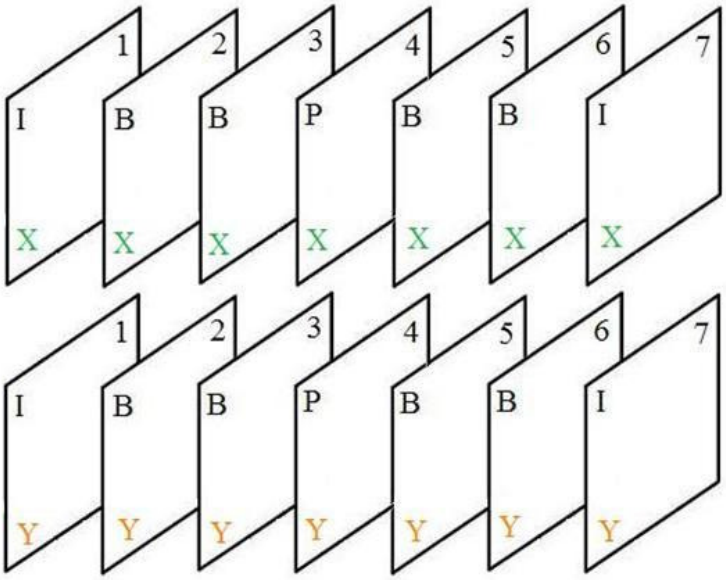


Figura 3.1: Exemplo de duas sequências de vídeo (Sequências X e Y).

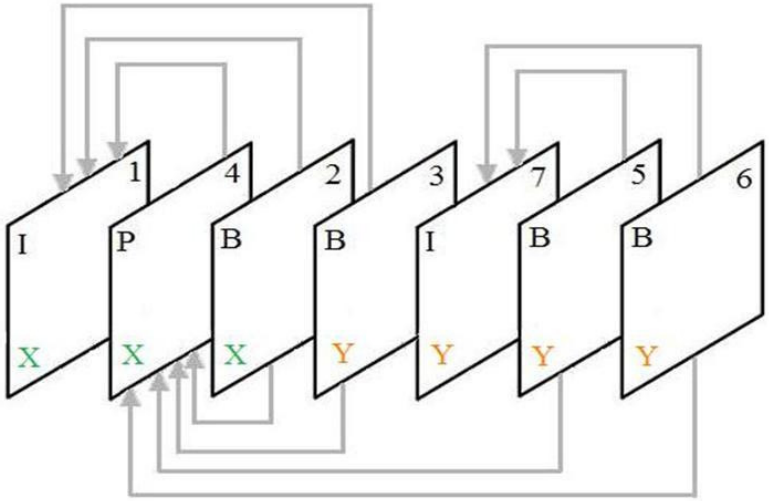


Figura 3.2: Nova sequência formada após o chaveamento entre as sequências X e Y da Figura 3.1.

mação anterior, o decodificador irá utilizar imagens da primeira sequência, que estão na lista de referência, para decodificar e reconstruir as imagens da segunda. Feito tal processo, o erro é praticamente certo.

A partir disso, a busca por uma solução que conseguisse sanar este problema passa pela tentativa de se enviar a informação necessária para que o decodificador não utilize as imagens da

sequência de vídeo anterior ao chaveamento como referência para reconstruir imagens da atual sequência. Seria importante também garantir que ao se fazer o chaveamento, as demais imagens a serem decodificadas não dependam de imagens anteriores ao ponto de chaveamento da sequência atual para serem reconstruídas, já que estas imagens não foram transmitidas para o decodificador e por isso não estarão entre as imagens de sua lista de referência. Em um chaveamento feito sem critérios, ou seja, feito em bytes ou bits quaisquer entre as duas *streams* de vídeo, as chances de conseguirmos satisfazer estas condições são muito pequenas.

Assim, o cenceito da imagem do tipo IDR, como já foi apresentada na Seção 2.3.4, parece ser ideal, já que quando esta chega ao codificador marca todas as imagens presentes nas atuais listas de referência como inúteis para referência. Assim, o que ocorre é que a partir da IDR, as listas de referência são praticamente refeitas e repreenchidas com o decorrer da codificação. Da mesma forma funciona para o processo de decodificação. Além disso, a imagem IDR é composta apenas por macroblocos do tipo I (só utilizam predição INTRA).

3.3 ESTRUTURA DO *STREAM* DE VÍDEO

Para se tentar fazer o chaveamento em imagens do tipo IDR, procurou-se, então, conhecer a estrutura do *stream* a ponto de saber como identificar o byte, dentro da segunda sequência de vídeo codificado, para se fazer o chaveamento de forma que um quadro IDR fosse o primeiro da segunda sequência a ser decodificado.

Nesse ponto, o uso e estudo de um codificador de vídeo (JM 16.0) e seus processos para codificar/decodificar o *stream* de vídeo foi importante para se entender o formato e os dados das sequências codificadas, além de representar uma motivação extra que é gerada quando se pode codificar e decodificar uma sequência de vídeo e ver na prática os resultados destes processos. Através desta abordagem, contando com referências como [1] e [3], o conceito de unidade NAL aparece, como descrito na Seção 2.4.

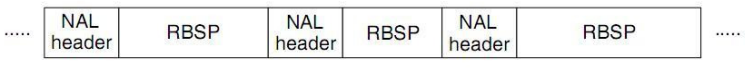


Figura 3.3: Sequência de unidades NAL [1].

O conhecimento da estrutura do *bitstream*, bem como as unidades NAL e seus tipo (Seção 2.4), permite a construção de um *parser* destas unidades, como primeiro passo prático para

começar a solucionar o problema. Um *parser* das unidades NAL se trata de um programa que foi construído para receber arquivos de vídeo codificados no padrão H.264 e fornecer dados a respeito de suas unidades NAL.

Para a construção deste *parser*, fez-se uso das informações descritas na Seção 2.4. Primeiramente, para se separar as unidades NAL entre si, foi necessário o uso do código de início destas unidades. Além disso, com a identificação das unidades NAL que está presente em seu *header*, tornou-se possível também reconhecer cada unidade pertencente ao *bitstream*.

Na construção do *parser*, notou-se que, ao contrário do esperado, cada unidade NAL não estava antecedida pela sequência de bytes 0x00 0x00 0x01, e sim pela sequência de bytes 0x00 0x00 0x00 0x01. Ou seja, um byte de valor igual a 0x00 aparecia antes do código de início de todas as unidades NAL.

Voltando as atenções novamente para a norma [3], foi encontrada uma informação que mostra que na estrutura da sequência de bytes que compõem o vídeo codificado, este byte igual a 0x00 pode aparecer antes do código de início nos seguintes casos:

1: Quando se trata da primeira unidade NAL do *stream*. Nesse caso, este byte chama-se *leading_zero_8bits*.

2: No começo de uma unidade de acesso (*access unit*[4]), que seria o conjunto de unidades NAL que representam uma imagem codificada, ou seja, um conjunto que, quando decodificado, representa uma imagem decodificada. Neste caso, esse byte 0x00 recebe o nome de *zero_byte*. Além disso, o *zero_byte* também aparece quando as unidades NAL são dos tipos 7 e 8 (Tabela 2.4).

Nesta hora, observou-se que o codificador JM, para os parâmetros escolhidos para nossos testes, utiliza sempre unidades NAL que se encaixam nos quesitos acima e por isso sempre apresentam o byte extra 0x00 antes do prefixo de código de início. Tendo isto em mente para prever qualquer problema, o projeto seguiu e este fato não atrapalhou o desenvolvimento do raciocínio nem a obtenção de resultados.

Assim, o programa construído separa as unidades NAL e fornece cada uma dessas informações: tipo de unidade NAL, se faz parte de uma imagem de referência ou não, tamanho em bytes e o byte de início. Este *parser* foi construído com o uso da linguagem C/C++ e fornece os dados necessário para se realizar os testes.

Foram elaborados alguns testes que procuram a identificação de erros e suas fontes dentro do processo de chaveamento dos *streams* feito sem critérios. Os testes consistiram na concatenação em diferentes pontos dos *streams*, tanto o *stream* de saída como o de chegada. A tentativa foi de

tentar verificar quais eram os resultados para estas diferentes combinações, notando principalente a quantidade de erros gerados. Para poder comparar as combinações, tentou-se fazê-las de forma que os pontos nos *streams* que estavam variando ficassem o mais próximo possível um do outro, respeitando as condições impostas por cada teste.

Para verificar como estes testes foram feitos, bem como seus resultados, o Capítulo 4 mostra tais informações de forma detalhada.

4 TESTES E RESULTADOS EXPERIMENTAIS

4.1 INTRODUÇÃO

Os testes que foram realizados tiveram como processo o chaveamento de um primeiro *stream* e para um segundo através de um simples programa feito em linguagem C/C++. O chaveamento foi feito entre bytes destas sequências. Um parâmetro adotado para uma simplificação dos testes foi que um *slice* tem o tamanho de uma imagem inteira, ou seja, ele contém todos os macroblocos desta imagem. Outro fato importante a ser citado é o uso do *player* MPlayer 1.0, utilizado juntamente com o sistema operacional Ubuntu 8.10, nos testes realizados.

Quando, nos testes a seguir, é afirmado que o ponto de entrada da segunda sequência de vídeo é uma imagem (tipo B, P, I ou IDR - vide Seção 2.3), isso quer dizer que a segunda sequência será utilizada a partir do byte 0x00 anterior ao prefixo código de ínicio da unidade NAL. Por exemplo, se dizemos que em determinado teste iremos fazer o chaveamento para uma imagem do tipo P da segunda sequência de vídeo, isto quer dizer que o primeiro byte que iremos utilizar da segunda sequência será o byte 0x00 antes do código de início de uma unidade NAL, no meio da sequência de unidades NAL, que representa uma imagem do tipo P.

Foram usadas sequências de vídeo com alguns GOPs diferentes (Seção 2.3.5). A tabela 4.1 mostra como são estes GOPs.

Tabela 4.1: GOPs dos vídeos

Sequência	**GOP**	**Período de imagem IDR (*frames*)**
mobile.264 (352x288 pixels)	IBBPBBPBBI	30
coastguard.264 (352x288 pixels)	IBBBPBBBI	18
parkrun.264 (1280x720 pixels)	IBBPBBPBBPBBPBBI	18
mobcal.264 (1280x720 pixels)	IBBPBBPBBI	36

Para descrever e mostrar os testes que foram feitos neste projeto, os procedimentos que foram realizados serão indicados e com auxílio de algumas figuras os resultados obtidos serão exemplificados.

4.2 TESTE 1: CHAVEAMENTO DE VÍDEOS NA RESOLUÇÃO 352X288 PIXELS.

A primeira bateria de testes foi realizada com vídeos de baixa resolução através de algumas tentativas de chaveamento utilizando informações obtidas através do *parser* que havia sido construído e um outro simples programa construído em C/C++ que fizesse a concatenação dos *bitstreams* em pontos diferentes para que se pudesse observar os resultados das diferentes combinações.

Aqui foram utilizados dois vídeos de resolução CIF (*Common Intermediate Format* - 352x288 pixels), chamados *coastguard.cif* e *mobile.cif*. As Figuras 4.1 e 4.2 apresentam as sequências utilizadas nesta bateria de testes.

(a) (b) (c) (d) (e)

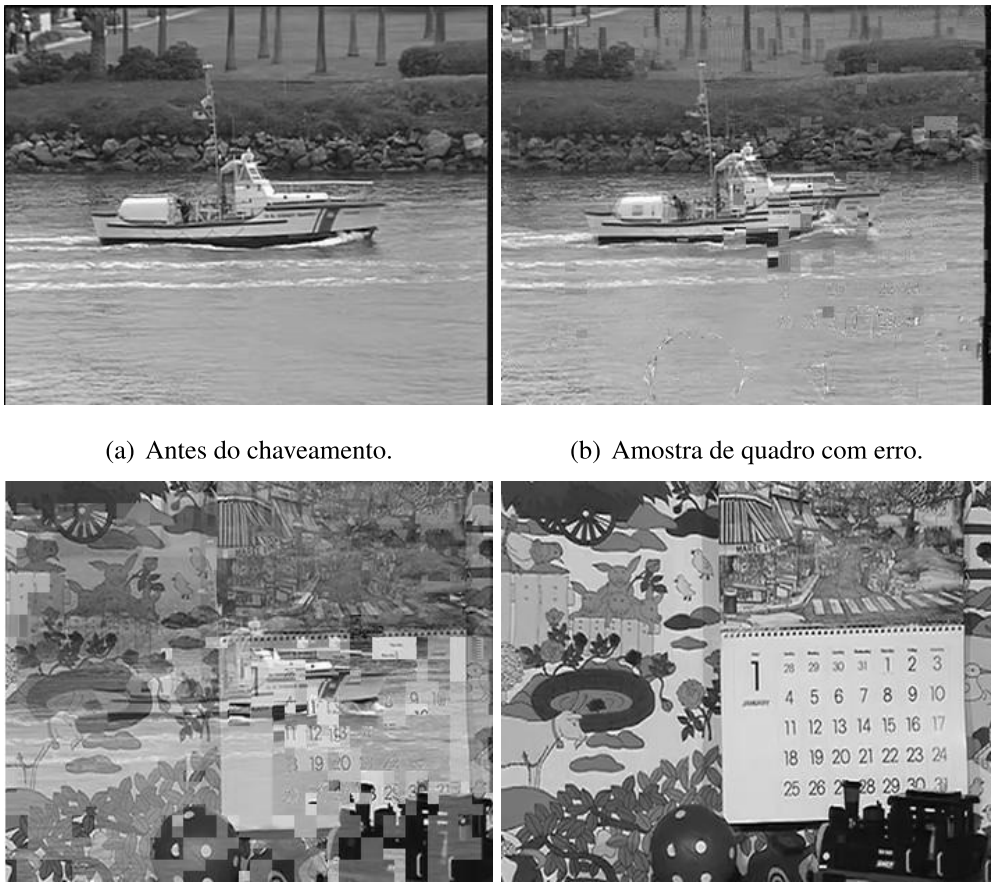
Figura 4.1: Alguns quadros da primeira sequência de vídeo CIF: *coastguard.cif*.

(a) (b) (c) (d) (e)

Figura 4.2: Alguns quadros da segunda sequência de vídeo CIF: *mobile.cif*.

4.2.1 Saindo de um byte qualquer do primeiro *stream* de vídeo para um byte qualquer do segundo.

Neste teste, foi escolhido um byte do primeiro *stream* de vídeo de forma que não fosse o começo nem o fim de nenhuma imagem desta sequência de vídeo. Fez-se isso utilizando as informações dadas pelo *parser*. Além disso, a concatenação dos *streams* foi feita em um byte que também não pertencesse nem ao início nem ao fim de uma imagem da segunda sequência de vídeo.



(a) Antes do chaveamento. (b) Amostra de quadro com erro.

(c) Amostra de quadro com erro. (d) Após erros.

Figura 4.3: Resultado da concatenação em bytes quaisquer dos dois *streams* CIF.

Desta forma, na Figura 4.3, mostra-se o resultado em alguns dos *frames* da sequência resultante. Nesta figura, entre o *frame* (a) e o *frame* (d), há 12 *frames*. Destes, praticamente todos apresentam erros de decodificação como os mostrados nos quadros presentes em (b) e (c). Algumas das imagens neste período são decodificadas de forma correta por se tratarem de imagens do tipo I, que na utilizam de predição INTER para serem decodificadas. As imagens do tipo P e B, que contam com este tipo de predição para serem decodificadas, são as imagens decodificadas de forma errada na nova sequência de vídeo.

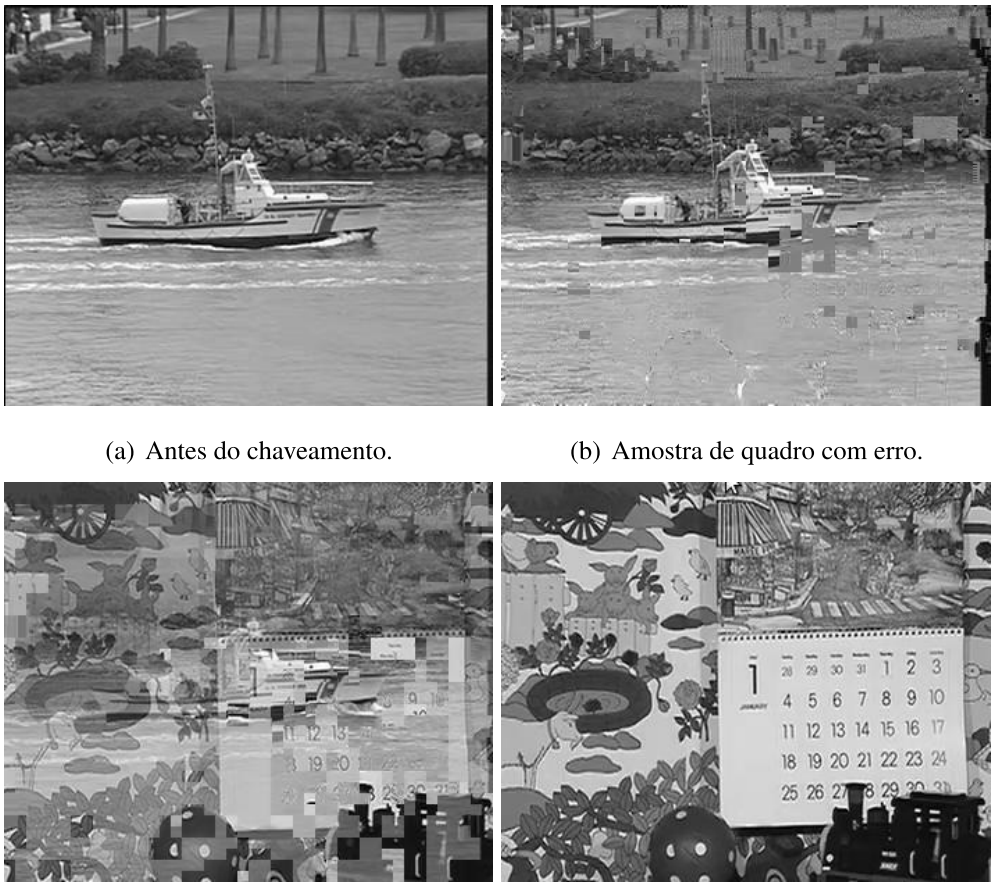
Nota-se que os erros começam depois do ponto em que a concatenação foi feita. Isto se deve porque o decodificador começa a receber bytes de imagens da segunda sequência de vídeo, enquanto ainda guarda e utiliza imagens decodificadas anteriormente como referência, causando uma confusão de blocos, misturando as imagens, resultando em erros grosseiros. Tais erros são tão evidentes, que mesmo quando exibidos a uma taxa de 30 *frames* por segundo, podem ser percebidos, mostrando que o chaveamento feito desta forma não pode ser considerado correto.

A qualidade da imagem só foi restaurada e estabilizada no momento em que uma imagem do

tipo IDR da nova sequência passa pelo decodificador, trazendo em sua sequência um conjunto de imagens que não utilizaria mais as imagens erradas presentes na listas de referência.

4.2.2 Saindo de um byte qualquer do primeiro *stream* de vídeo para imagem do tipo B do segundo.

Neste teste, é usado o mesmo byte de saída do primeiro *stream* do teste anterior. Mas agora a entrada na segunda sequência de vídeo está em um byte específico escolhido: no início do prefixo de uma unidade NAL de um quadro do tipo B (ver Seção 2.3). Foi escolhido o quadro do tipo B cuja posição de início estivesse depois e o mais próximo possível da posição escolhida no teste da Seção 4.2.1.



(a) Antes do chaveamento. (b) Amostra de quadro com erro.

(c) Amostra de quadro com erro. (d) Após erros.

Figura 4.4: Resultado da concatenação saindo de um byte qualquer do primeiro e entrando em uma imagem do tipo B do segundo *stream* CIF.

O resultado, como se pode ver na Figura 4.4, apresentou praticamente os mesmo erros de decodificação resultantes do teste anterior. O período dos erros também se repetiu, 12 *frames*, o que evidenciou que este método de chaveamento e concatenação também não é o ideal. Quando

a nova sequência, resultado da concatenação das outras duas, passa pelo decodificador, este armazena parte das imagens já decodificadas para realizar a predição das que estão por vir. Depois da concatenação, este utiliza imagens da sequência anterior ao byte de concatenação para auxiliar na decodificação das novas imagens, o que gera um erro evidente. Mais uma vez, a sequência de vídeo só se estabiliza quando a imagem do tipo IDR passa pelo decodificador.

4.2.3 Saindo de um byte qualquer do primeiro *stream* de vídeo para imagem do tipo P do segundo.

Este teste é muito semelhante ao teste anterior, com a diferença de que o ponto escolhido da segunda sequência para se fazer o chaveamento representa uma imagem do tipo P (ver Seção 2.3). Foi escolhida a imagem do tipo P cuja posição de início estivesse depois e o mais próximo possível da posição escolhida no teste da Seção 4.2.1.

Mais uma vez, como esperado, vários erros ocorreram. Após o chaveamento, esses erros de decodificação prejudicaram muito a qualidade do vídeo, como podemos notar na Figura 4.5.

4.2.4 Saindo de um ponto qualquer do primeiro *stream* de vídeo para imagem do tipo I do segundo.

Este teste foi feito para se mostrar que, mesmo que a imagem do tipo I não utilize a predição INTER, o chaveamento feito para esta não evita os erros da decodificação, dado que as imagens presentes nas listas de referência continuam sendo as imagens da sequência anterior, fonte dos erros no processo de decodificação de imagens do tipo B e P que vêm depois da imagem tipo I usada como ponto de chaveamento na segunda sequência de vídeo. Foi escolhida a imagem do tipo I cuja posição de início estivesse depois e o mais próximo possível da posição escolhida no teste da Seção 4.2.1.

Como a Figura 4.6 mostra, os erros são praticamente os mesmos, causados pelos mesmos motivos já descritos nos testes anteriores. Assim, o chaveamento feito para uma imagem do tipo I não representa uma boa forma de se fazê-lo.

4.2.5 Saindo de um byte qualquer do primeiro *stream* de vídeo para imagem do tipo IDR do segundo.

Agora, um importante teste realizado, utilizando o chaveamento feito de um byte qualquer dentro da primeira sequência de vídeo, ligado a uma imagem do tipo IDR da segunda. Mais uma

(a) Antes do chaveamento.

(b) Após erros.

(c) Amostra de quadro com erro.

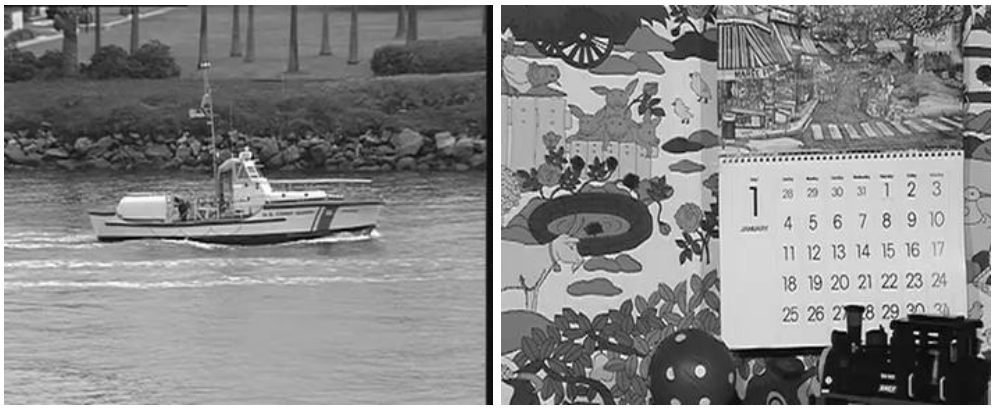
Figura 4.5: Resultado da concatenação saindo de um ponto qualquer do primeiro e entrando em uma imagem do tipo P do segundo *stream* CIF.

vez, foi escolhida a imagem do tipo IDR cuja posição de início estivesse depois e o mais próximo possível da posição escolhida no teste da Seção 4.2.1.

O resultado foi o de nenhum erro aparente, mesmo quando o vídeo é visto em uma taxa de *frames* por segundo pequena. Mas, a exemplo dos testes anteriores, o *player* usado acusou um erro no momento do chaveamento, mostrando que, apesar de nenhum erro aparente, algo errado aconteceu.

Aqui entra o erro causado por se sair de um byte qualquer da primeira sequência de vídeo. Afinal, se é feito o chaveamento de um *stream* para outro antes do final das informações da imagem atual, isto representa que esta última imagem da primeira sequência ficou prejudicada em algum ponto. Por mais que o erro não tenha sido percebido pelos olhos, o software o percebe, o que pode acarretar em algum erro mais grave, dependendo do software ou da aplicação que é utilizada.

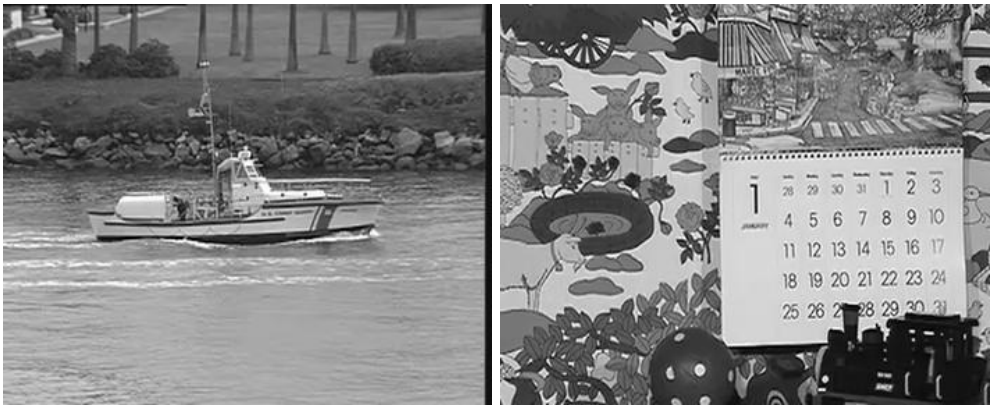
De qualquer forma, o erro apresentado não é comparável com os erros apresentados nos testes



(a) Antes do chaveamento. (b) Após erros.

(c) Amostra de quadro com erro.

Figura 4.6: Resultado da concatenação saindo de um ponto qualquer do primeiro e entrando em uma imagem do tipo I do segundo *stream* CIF.



(a) Antes do chaveamento. (b) Após chaveamento.

Figura 4.7: Resultado da concatenação saindo de um ponto qualquer do primeiro e entrando em uma imagem do tipo IDR do segundo *stream* CIF.

anteriores, já que em todos os testes o *player* continuou a rodar o vídeo mesmo com erros, mas no caso do chaveamento para a imagem IDR, não pudemos notar o erro apenas assistindo a nova

sequência de vídeo.

4.2.6 Saindo ao final de uma imagem qualquer do primeiro *stream* de vídeo para imagem do tipo IDR do segundo.

Para finalizar, foi feito um teste que visou fazer a concatenação dos *streams* de vídeo no final dos bytes de uma unidade NAL que representa uma imagem da primeira sequência. Assim, feito testes com o byte de chaveamento do primeiro *stream* variando no final da unidade NAL de alguns tipos de imagens, notou-se que, a exemplo do teste anterior, não é possível se notar algum erro presente de decodificação apenas assistindo-se a nova sequência de vídeo. Além disso, o *player* também não acusou erro algum quando as imagens de chaveamento passam pelo processo de decodificação. Sendo assim, este método de chaveamento foi eleito como o melhor possível para a bateria de testes desta Seção 4.2.

4.3 TESTE 2: CHAVEAMENTO DE VÍDEOS NA RESOLUÇÃO 1280X720 PIXELS.

Esta segunda bateria de testes foi feita visando a confirmação das suposições e afirmações feitas na bateria de testes da Seção 4.2. Além disso, o fato do vídeo ser de uma maior resolução poderia ajudar na detecção de erros.

Neste teste foram utilizados vídeos codificados a partir de dois vídeos de resolução 1280x720 pixels, chamados parkrun.yuv e mobcal.yuv. As Figuras 4.8 e 4.9 apresentam as sequências desta segunda bateria de testes.

(a) (b) (c) (d) (e)

Figura 4.8: Alguns quadros da primeira sequência de vídeo de resolução 1280x720 pixels: *parkrun.yuv*.

(a) (b) (c) (d) (e)

Figura 4.9: Alguns quadros da segunda sequência de vídeo de resolução 1280x720 pixels: *mobcal.yuv*.

4.3.1 Chaveamento entre bytes quaisquer dentro das sequências.

Neste teste, assim como o teste realizado na Seção 4.2.1, os *streams* foram ligados através de bytes que estavam situados em meio a dados de uma imagem qualquer das sequências.

Os resultados apresentaram, mais uma vez, muitos erros, sendo que desta vez o intervalo de quadros entre o último *frame* decodificado com sucesso da primeira sequência e o último *frame* com erros foi de 32 *frames*. Mais uma vez, observou-se que somente houve a interrupção dos erros quando um quadro IDR decorrente da segunda sequência de vídeo passou pelo decodificador, inutilizando a lista de referência e evitando novos erros.

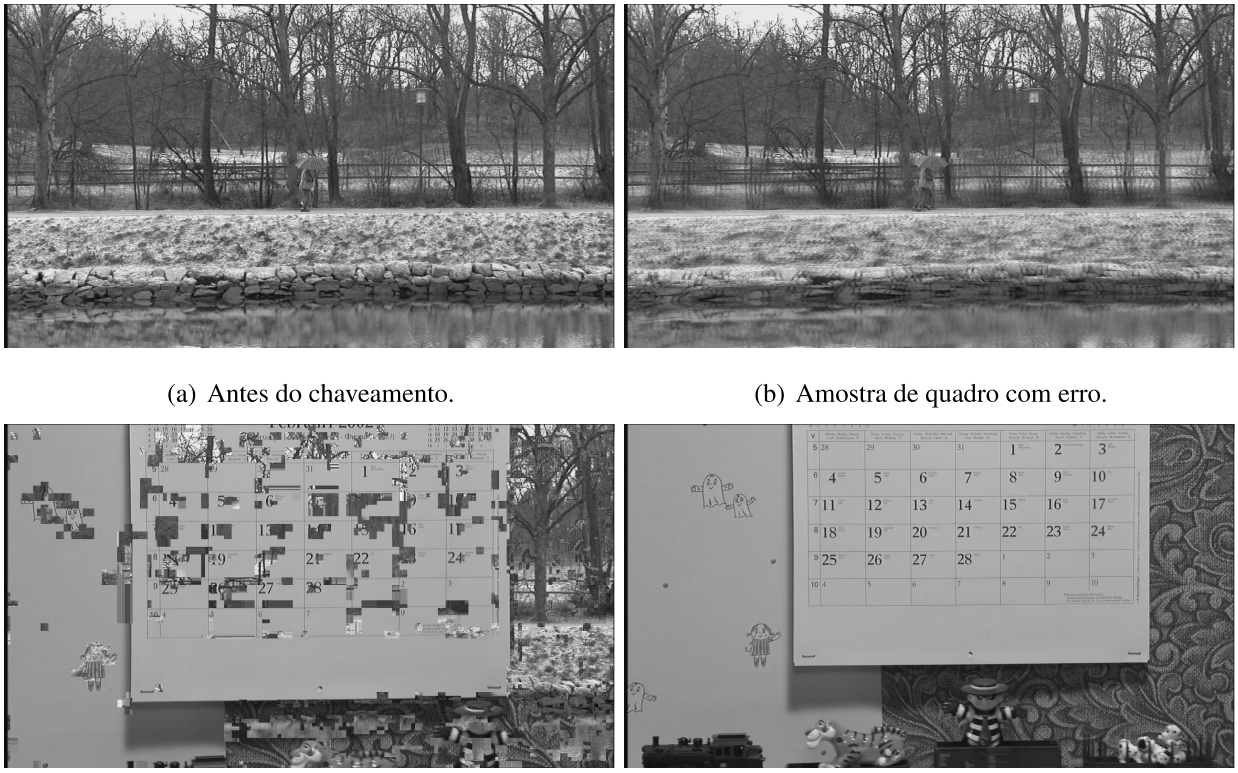
A Figura 4.10 mostra alguns quadros da nova sequência, evidenciando um quadro antes dos erros, um depois dos erros e alguns durante os erros. Nota-se que o resultado foi bem similar ao apresentado na Seção 4.2.

4.3.2 Saindo de um byte qualquer do primeiro *stream* de vídeo para imagens do tipo B, P ou I do segundo.

Como estes testes já foram realizados na primeira bateria de testes, localizados na Seção 4.2, e os resultados apresentados foram muito parecidos, estes testes para esta segunda bateria foram agrupados.

Lembrando que a escolha das imagens para onde o chaveamento foi feito na segunda sequência de vídeo foi a imagem, do tipo desejado, depois e mais próxima na ordem das unidades NAL do ponto escolhido para a realização do chaveamento no teste da Seção 4.3.1.

Assim como nos testes realizados com vídeos de baixa resolução, os resultados obtidos nestes foram muito semelhantes ao primeiro teste desta segunda bateria, tanto em número de quadros com erros na decodificação, quanto em termos de visualização. As imagens presentes no bloco da Figura 4.11 mostram erros apresentados, sendo que nesta figura, exibimos quadros da sequência resultante da concatenação feita para uma imagem do tipo P. Como os testes feitos com imagens



(a) Antes do chaveamento.

(b) Amostra de quadro com erro.

(c) Amostra de quadro com erro.

(d) Após erros.

Figura 4.10: Resultado da concatenação em bytes quaisquer dos dois *streams* de resolução 1280x720 pixels.

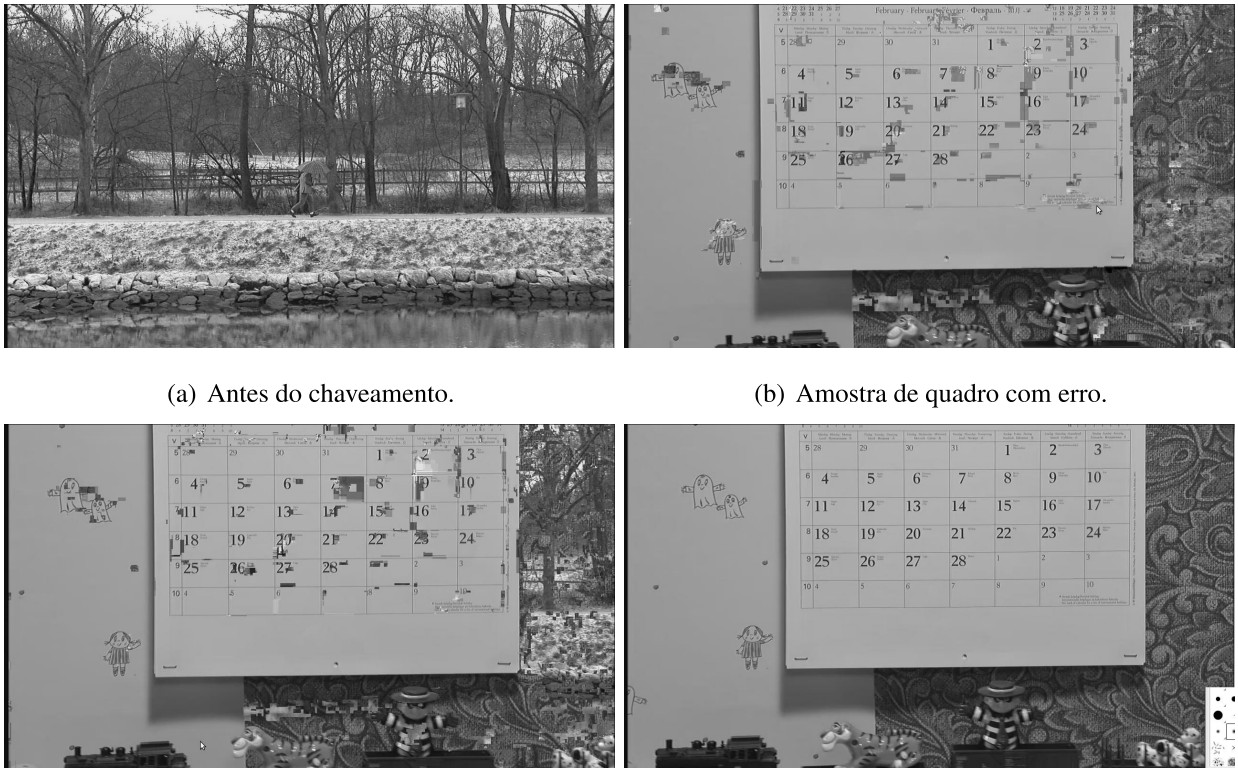
do tipo B e I se mostraram quase que idênticos, as imagens destes não serão apresentadas.

Mais uma vez o decodificador não toma conhecimento de que o chaveamento foi feito, usa imagens decodificadas da sequência anterior ao chaveamento de sua lista de referência para recuperar outras da nova sequência (que precisam de imagens de referência para ser decodificadas), gerando *frames* decodificados cheio de erros.

4.3.3 Saindo de um byte qualquer do primeiro *stream* de vídeo para imagem do tipo IDR do segundo.

Para este teste, o chaveamento foi feito para um ponto em que a imagem presente no segundo *stream* de vídeo fosse do tipo IDR, a mais próxima após, na ordem das unidades NAL, do byte usado no teste da Seção 4.3.1.

O resultado, como a Figura 4.12 mostra, foi o de praticamente nenhum erro apresentado, a não ser um quadro presente no ponto do chaveamento que se mostra um pouco alterado. Com uma taxa de exibição de 30 *frames* por segundo, este erro ficou praticamente imperceptível, mas não para o *player*, que acusou erro no momento de decodificar este quadro, mesmo não parando



(a) Antes do chaveamento.

(b) Amostra de quadro com erro.

(c) Amostra de quadro com erro.

(d) Após erros.

Figura 4.11: Resultado da concatenação saindo de um byte qualquer da primeira e entrando em uma imagem do tipo P da segunda sequência (1280x720 pixels).

de funcionar. Olhando quadro a quadro, foi possível observar alguns erros neste *frame*, como pode ser visto na Figura 4.12(c).

Mais uma vez, vem a questão de se sair do primeiro *stream* antes que se acabe de transmitir/armazenar a imagem atual. Este processo pode causar erros, como é o atual caso, já que o decodificador começa a construir um novo *frame*, mas antes que todas as informações cheguem, ele é forçado a começar a reconstruir outro, pois as informações deste último foram perdidas.

4.3.4 Saindo ao final de uma imagem qualquer do primeiro *stream* de vídeo para imagem do tipo IDR do segundo.

Por fim, foi feito um teste que visou fazer a concatenação dos *streams* de vídeo no final dos bytes da unidade NAL de uma imagem da primeira sequência, para uma imagem IDR da segunda. Aqui notou-se, a exemplo do teste anterior, que não é possível notar nenhum erro presente de decodificação apenas assistindo-se a nova sequência de vídeo a uma taxa de 30 *frames* por segundo.

Mas quando o *player* é configurado para exibir a nova sequência de vídeo a uma taxa de

(a) Amostra de quadro anterior ao chaveamento.

(b) Amostra de quadro posterior ao chaveamento.

(c) Último quadro da primeira sequência: pequenos erros.

Figura 4.12: Resultado da concatenação saindo de um byte qualquer da primeira e entrando em uma imagem do tipo IDR da segunda sequência (1280x720 pixels).

frames por segundo pequena (por exemplo 2 *frames* por segundo), não há o quadro defeituoso que há quando o chaveamento é feito como na Seção 4.3.3. A Figura 4.13 mostra os últimos quadros antes do chaveamento, para a sequência feita no teste anterior (com alguns erros) e para a sequência feita neste teste (sem erros).

Além disso, o *player* também não acusou erro algum quando as imagens de chaveamento passam pelo processo de decodificação. Por isso, o chaveamento feito desta forma foi eleito como o melhor método.

(a) Último quadro da primeira sequência: teste da Seção 4.3.4.

(b) Último quadro da primeira sequência: teste da Seção 4.3.3.

Figura 4.13: Comparação entre os últimos quadros antes do chaveamento, para os testes das seções 4.3.3(a) e 4.3.4(b).

5 ANÁLISES FINAIS E CONCLUSÕES

5.1 CONCATENAÇÃO FEITA EM BYTES QUAISQUER DOS DOIS *STREAMS*

Depois de feitos os testes, alguns aspectos ficaram claros quando se trata de se fazer a concatenação de *bitstreams* de vídeo. Um deles é o erro gerado no momento em que se faz a concatenação em bytes aleatórios nas duas sequências. Quando fazemos este tipo de chaveamento estamos simplesmente emendando dados de unidades NAL diferentes, sendo que ficamos com uma unidade contendo um pedaço de informações de uma unidade da primeira sequência e um pedaço de informações de uma da segunda sequência. Logicamente, o decodificador não possui mecanismos para detectar este tipo de mudança brusca, já que ele é construído para simplesmente ler e interpretar os bits do *stream* que está sendo decodificado.

Além disso, o decodificador ainda possui em seu *buffer* imagens decodificadas de referência pertencentes a primeira sequência de vídeo. Quando uma imagem chega e indica ao decodificador que ela necessita de uma imagem que está numa posição específica de uma das listas de referência para ser decodificada, o decodificador apenas vai até esta posição na lista, utiliza a imagem que está ali para fazer a predição e reconstruir o *frame* atual. Como na lista estão imagens de referência da primeira sequência de vídeo e os quadros a serem decodificados depois do chaveamento são da segunda sequência, o processo de decodificação começa a gerar quadros cheios de erros e problemas.

Assim, a seguir estão enumeradas algumas fontes de erros para a concatenação feita entre bytes quaisquer das sequências:

Fonte de erro 1: Saindo de um byte qualquer da primeira sequência, impede-se que a informação pertencente à última unidade NAL, desta sequência, a ser transmitida seja entregue de forma correta.

Fonte de erro 2: As imagens do tipo P e B que chegam pela segunda sequência de vídeo, logo após o ponto de concatenação dos *streams*, irão utilizar imagens das listas de referência do decodificador que não são as mesmas com as quais foram codificados. Aliás, elas têm grande chance de ser muito diferentes, já que as listas de referência, neste momento, estão repletas de imagens da primeira sequência de vídeo. As imagens do tipo I, por não precisarem da predição INTER para ser decodificadas, conseguem sucesso em tal tarefa, mas não conseguem evitar que

este erro aconteça com as demais imagens.

Fonte de erro 3: A concatenação feita para o segundo *stream* de vídeo em um byte qualquer, aumenta a geração de erros. Isto acontece pois quando esses bytes são inseridos na nova sequência, são colocadas informações sem critério algum nessa nova *stream* de vídeo. Ou seja, quando os *streams* são ligadas neste ponto, os bytes que são inseridos não possuem identificação nenhuma e, por isso, provavelmente não serão interpretados da maneira correta.

5.2 CONCATENAÇÃO FEITA EM BYTE QUALQUER DO PRIMEIRO *STREAM* PARA IMAGEM DOS TIPOS P, B OU I DO SEGUNDO *STREAM*

Quando o chaveamento é feito a partir de um byte qualquer do primeiro *stream* e concatenando no começo de uma unidade NAL que representa uma imagem dos tipos P, B ou I, teremos problemas semelhantes ao da seção anterior.

Nestes testes, nota-se que apenas eliminamos uma das fontes de erros possíveis, que seria a fonte de erro número 3 mostrada na Seção 5.1.

Com relação à fonte de erro número 2, uma concatenação feita para uma imagem do tipo I do segundo stream não representa uma solução, já que, por mais que esta imagem seja decodificada corretamente, as imagens B e P que vêm a seguir, utilizam diversas imagens das listas de referência para serem decodificadas. Como já foi dito, essas imagens não são as imagens corretas e os vários erros aparecem.

Com os resultados mostrados nos testes, apesar da eliminação de uma fonte de erro, continua a presença de fortes problemas durante o processo de decodificação da nova *stream* de vídeo formada. Esta forma de concatenação praticamente não mostrou melhora nenhuma quando comparada com o chaveamento feito em bytes quaisquer das duas sequências.

5.3 CONCATENAÇÃO FEITA EM BYTE QUALQUER DO PRIMEIRO *STREAM* PARA IMAGEM DO TIPO IDR DO SEGUNDO *STREAM*

Fazendo a ligação dos *streams* de vídeo saindo-se do primeiro em um byte qualquer e chegando-se ao segundo em um byte que representa o começo de uma imagem do tipo IDR, elimina-se a

fonte de erro número 2. Como visto no Capítulo 4, a melhora obtida foi considerável, mostrando que a grande fonte de erros presente era realmente o uso de imagens erradas para se tentar fazer a decodificação de imagens do tipo P e B depois do ponto de chaveamento. O uso da imagem IDR elimina esse erro.

Mesmo não possuindo erros perceptíveis quando é realizado este tipo de concatenação utilizando-se vídeos de baixa resolução (352x288 pixels), nota-se que o *player* acusa um erro durante o processo de decodificação, quando passa pelo ponto de chaveamento.

Além disso, quando o teste foi feito em alta definição, usando um vídeo de resolução 1280x720 pixels, o erro é notado, em um quadro antes da concatenação, quando a nova sequência de vídeo é assistida utilizando-se uma taxa pequena de *frames* por segundo. Ele apresenta poucos erros, como é visto na Figura 4.13(b).

5.4 CONCATENAÇÃO FEITA NO FIM DE UMA IMAGEM DO PRIMEIRO *STREAM* PARA IMAGEM DO TIPO IDR DO SEGUNDO *STREAM*

Este último teste retornou resultados esperados: decodificação sem nenhum tipo de erro, tanto os erros visíveis, quanto os indicados pelo *player* que exibe vídeo. A Figura 4.13 apresenta a comparação dos dois quadros equivalentes para as duas últimas formas de chaveamento.

Neste caso, a forma de se fazer o chaveamento sem qualquer tipo de erro foi demonstrada, já que todas as fontes de erro foram eliminadas do processo.

5.5 CONSIDERAÇÕES EXTRAS

A partir deste trabalho, pode-se ser realizada uma outra abordagem sobre o tema, incluindo o estudo de como o chaveamento é feito nas transmissões de televisão atual. Com o conhecimento adquirido a partir deste projeto, a construção de *softwares* que façam o chaveamento de forma correta torna-se uma possibilidade. Com informações sobre os *streams* gerados pelas redes de televisão, como tipo de GOP ou período de quadros IDR, pode-se construir um *switch* que possa até ser comercializado, carregando *softwares* que façam esse chaveamento para alternar entre vídeos diferentes, como programas, propagandas, jornais, entre outros, de acordo com a preferência da emissora. Pode-se, por exemplo, se ter um repetidor de sinal digital de televisão para alguma cidade de forma que, nesse repetidor possa se fazer o chaveamento entre

propagandas locais e nacionais, no momento desejado.

Outro ponto notado foi a respeito do período de uma imagem IDR dentro de uma sequência de vídeo. Este também é um fator importante para que os chaveamentos sejam feitos de forma mais fácil, afinal, quanto menor a distância do ponto em que se deseja fazer o chaveamento até a próxima imagem IDR, menor é o tempo que se deve esperar para que o chaveamento correto seja feito.

Além das considerações já feitas, pode-se afirmar também que os vídeos de alta definição ajudam a mostrar e detectar erros, já que um dos erros apontados só se permitiu notar claramente quando utilizamos o vídeo de resolução 1280x720 pixels. Podemos afirmar isso tendo como base nos resultados vistos nos testes realizados nas Seções 4.2.5 e 4.3.3, quando um mesmo método de chaveamento apresentou erro para os dois testes (detectado pelo *player*), mas só pôde ser visto assistindo-se o teste realizado com o vídeo de resolução 1280x720 pixels.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] RICHARDSON, I. E. G. *H.264 and MPEG-4 Video Compression*. [S.l.]: John Wiley & Sons Ltd, 2003.

[2] HUNG, E. M. *Compensação de Movimento Utilizando Blocos Multi-escala e Forma Variável em um Codec de Vídeo Híbrido*. 2007.

[3] ITU-T. *Advanced video coding for generic audiovisual services*. fourth. [S.l.: s.n.], 2009.

[4] PROJETO Rede H.264 SBTVD: Questões sobre interfaces. 2009. Disponível em: <http://www.lapsi.eletro.ufrgs.br/h264/wiki>.