Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Framework para Processamento Paralelo de Algoritmos de Aumento de Resolução de Vídeos

Pedro Garcia Freitas

Documentação apresentado como requisito para o exame
de qualificação : Mestrado em Informática

Orientadora
Prof.ª Dr.ª Mylene Christine Queiroz de Farias

Coorientadora
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Ricardo Pezzuol Jacobi

Banca examinadora composta por:

Prof.ª Dr.ª Mylene Christine Queiroz de Farias (Orientadora) — ENE/UnB
Prof. Dr. Eduardo A. B. da Silva — DEL/POLI/UFRJ
Prof. Dr. Ricardo Pezzuol Jacobi — CIC/UnB

CIP — Catalogação Internacional na Publicação

Freitas, Pedro Garcia.

Um Framework para Processamento Paralelo de Algoritmos de Aumento de Resolução de Vídeos / Pedro Garcia Freitas. Brasília : UnB, 2013.
144 p. : il. ; 29,5 cm.

Exame de Qualificação (Mestrado) — Universidade de Brasília, Brasília, 2013.

1. Aumento de resolução, 2. Super-resolução, 3. Computação Paralela

CDU 11/0068408

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Framework para Processamento Paralelo de Algoritmos de Aumento de Resolução de Vídeos

Pedro Garcia Freitas

Documentação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Mylene Christine Queiroz de Farias (Orientadora)
ENE/UnB

Prof. Dr. Eduardo A. B. da Silva
DEL/POLI/UFRJ

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Mestrado em Informática

Brasília, 19 de fevereiro de 2013

Resumo

O aumento dimensional de sinais visuais consiste na alteração do tamanho de uma imagem ou de um vídeo para dimensões espaciais maiores, utilizando técnicas de processamento digital de sinais. Geralmente, esse aumento é feito com a utilização de técnicas de interpolação. Contudo, essas técnicas de interpolação produzem distorções nas imagens aumentadas. Tais distorções ocorrem porque a imagem aumentada possui apenas as amostras da imagem original, de dimensões menores, que são insuficientes para reconstrução exata do sinal, o que gera efeitos de *aliasing*. Assim sendo, as técnicas de interpolação apenas estimam os coeficientes não-amostrados do sinal, o que muitas vezes produz resultados insatisfatórios para muitas aplicações, necessitando de outras técnicas para reconstituir os coeficientes não-amostrados com maior precisão.

Para melhorar a aproximação de uma imagem estimada com relação à imagem original, existem técnicas que reconstroem os coeficientes não-amostrados. Essas técnicas são chamadas de super-resolução. Elas consistem em aumentar a resolução utilizando, geralmente, informações de outras imagens em baixa ou alta-resolução para estimar a informação faltante na imagem que se deseja ampliar.

Super-resolução é um processo computacionalmente intenso, onde a complexidade dos algoritmos são, geralmente, de ordem exponencial no tempo em função do bloco ou do fator de ampliação. Portanto, quando essas técnicas são aplicadas para vídeos, é necessário que o algoritmo seja extremamente rápido. O problema é que os algoritmos mais computacionalmente eficientes, nem sempre são aqueles que produzem os melhores resultados visuais.

Sendo assim, este trabalho propõe um *framework* para melhorar o desempenho de diversos algoritmos de super-resolução através de estratégias de processamento seletivo e paralelo. Para isso, nesta dissertação são examinadas as propriedades dos resultados produzidos pelos algoritmos de super-resolução e os resultados produzidos utilizando-se técnicas de interpolação. Com essas propriedades, é encontrado um critério para classificar as regiões em que os resultados produzidos sejam visualmente equivalentes, não importando o método utilizado para ampliação. Nessas regiões de equivalência utiliza-se um algoritmo de interpolação, que é muito mais veloz do que os computacionalmente complexos de super-resolução. Assim, consegue-se reduzir o tempo de processamento sem prejudicar a qualidade visual do vídeo ampliado.

Além dessa abordagem, este trabalho também propõe uma estratégia de divisão de dados entre diferentes tarefas para que a operação de aumento de resolução seja realizada

de forma paralela. Um resultado interessante do modelo proposto é que ele desacopla a abstração de distribuição de carga da função de aumento dimensional. Em outras palavras, diferentes métodos de super-resolução podem explorar os recursos do *framework* sem que para isso seus algoritmos precisem ser modificados para obtenção do paralelismo. Isso torna o *framework* portável, escalável e reusável por diferentes métodos de super-resolução.

Palavras-chave: Aumento de resolução, Super-resolução, Computação Paralela

Abstract

The magnification of visual signals consists of changing the size of an image or a video to larger spatial dimensions, using digital signal processing techniques. Usually, this magnification is done using numerical interpolation methods. However, these interpolation methods tend to produce some distortions in the increased images. Such distortions occours because the interpolated image is reconstructed using only the original image samples, which are insufficients for the accurate signal reconstruction, generating aliasing effects. These interpolation techniques only approximate the non-sampled signal coefficients, producing unsatisfactory results for many applications. Thus, for these applications, others techniques to estimate the non-sampled coefficients are needed.

To improve the estimation accuracy of an image with respect to the original, the super-resolution techniques are used to reconstruct the non-sampled coefficients. Generally, these super-resolution techniques enhance the increased image using information of other images to estimate the missing information.

Super-resolution is a computationally intensive process, where the algorithms complexity are, generally, exponential in time as function of the block size or magnification factor. Therefore, when these techniques are applied for videos, it is required that the super-resolution algorithm be extremely fast. However, more computationally efficient algorithms are not always those that produce the best visual results.

Therefore, this work proposes a framework to improve the performance of various super-resolution algorithms using selective processing and parallel processing strategies. Thus, this dissertation examines the properties of the results produced by the super-resolution algorithms and the results produced by using interpolation techniques. From these properties, is achieved a criterion to classify regions wherein the results produced are equivalent (using both super-resolution or interpolation). In these regions of equivalence, the interpolation algorithms are used to increase the dimensions. In the anothers regions, the super-resolution algorithms are used. As interpolation algorithms are faster than the computationally complex super-resolution algorithms, the idea is decrease the processing time without affecting the visual quality of amplified video.

Besides this approach, this paper also proposes a strategy to divide the data among various processes to perform the super-resolution operation in parallel. An interesting result of the proposed model is the decoupling of the super-resolution algorithm and the parallel processing strategy. In other words, different super-resolution algorithms can explore the features of the proposed framework without algorithmic modifications to achieve

the parallelism. Thus, the framework is portable, scalable and can be reusable by different super-resolution methods.

Keywords: Increased resolution, Super-resolution, Parallel Computing

Sumário

Lista de Figuras

Lista de Tabelas

Dedico meu mestrado à memória do mestre Luiz Carlos Alborghetti.

Agradecimentos

Em primeiro lugar, certamente devo agradecer às minhas professoras e orientadoras, Aletéia Patrícia Favacho de Araújo e Mylene Christine Queiroz de Farias, que sempre me apoiaram durante o todo o período desse mestrado. Sem as cuidadosas leituras, críticas e sugestões feitas por elas, eu jamais conseguiria concluir este trabalho. Poucas linhas de texto jamais refletiriam a enorme valor e reconhecimento que tenho pela dedicação, incentivo e apoio recebido delas. Elas serão indubitavelmente as maiores referências na minha carreira acadêmica.

Também não poderia deixar dedicar enormes agradecimentos ao meu orientador da graduação e atual chefe, o Prof. José Felippe Beaklini. Sem o seu apoio, incentivo e amizade, provavelmente seria inviável para mim conciliar meu emprego com minhas atividades no meio acadêmico.

Aos professores do Instituto de Informática e Telecomunicações da Universidade de Granada, Rafael Molina e Salvador Villena Morales, por terem me cedido o código-fonte dos métodos que usei, em partes, nas simulações apresentadas neste trabalho.

Aos meus amigos e colegas do Laboratório de Cálculo Científico (LCC), que sempre me auxiliaram nos problemas profissionais, o que contribuiu muito com a elaboração desse trabalho.

Agradeço também ao órgão de Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio financeiro direto e indireto.

Capítulo 1

Introdução

O aumento dimensional de sinais visuais consiste na alteração do tamanho de uma imagem ou de um vídeo para dimensões espaciais maiores, utilizando técnicas de processamento digital de sinais [90]. Geralmente, esse aumento é feito com a utilização de técnicas de interpolação. Contudo, essas técnicas de interpolação produzem distorções nas imagens aumentadas. Tais distorções ocorrem porque a imagem aumentada possui apenas as amostras da imagem original, de dimensões menores, que são insuficientes para reconstrução exata do sinal, o que gera efeitos de *aliasing* [38]. Assim sendo, as técnicas de interpolação apenas estimam os coeficientes não-amostrados do sinal, o que muitas vezes produz resultados insatisfatórios para muitas aplicações, necessitando de outras técnicas para reconstituir os coeficientes não-amostrados com maior precisão.

No contexto do processamento de imagens, uma das técnicas para reconstituir esses coeficientes não-amostrados são as técnicas de super-resolução. Essas técnicas consistem em aumentar a resolução utilizando, geralmente, informações de outras imagens em baixa ou alta-resolução. O primeiro trabalho apresentado neste tópico foi publicado em 1984 no artigo *Multiframe image restoration and registration* [82], mas o termo "super-resolução" só foi incorporado na literatura em 1990 por Irani e Peleg [43]. Dessa forma, devido ao

interesse na indústria, após o artigo de Irani e Peleg, diversas outras abordagens de super-resolução foram desenvolvidas.

Para obter imagens em alta-resolução, uma grande quantidade de algoritmos de super-resolução são formulados utilizando o espaço de frequências. Esses métodos baseiam-se nas propriedades de deslocamento da transformada de Fourier e na relação entre os coeficientes transformados das imagens utilizadas para estimar a imagem a ser ampliada a partir de um conjunto de equações. Assim, diversos métodos foram propostos para estimar os coeficientes no espaço transformado. Tekalp *et al.* [79] utilizam análise de correspondência para mitigar os efeitos do ruído produzido pela reconstrução a partir de dados insuficientes. Uma abordagem que utiliza mínimos quadrados recursivos é proposta por Kim *et al.* [49]. Além disso, técnicas baseadas no teorema da amostragem multicanal é usada por Ur e Gross [83].

Além dos sistemas que trabalham no espaço de frequência, uma outra classe de métodos utilizam o domínio espacial para estimar a informação não-amostrada. Para isso, diferentes abordagens podem ser utilizadas, tais como a reconstrução de amostras não-uniformemente espaçadas, projeção inversa (*backprojection*) [10, 20, 45], modelos estocásticos [17, 85, 86], regularização de Tikhonov [65, 81], etc.

Algoritmos de super-resolução voltados para imagens são, geralmente, computacionalmente custosos, uma vez que envolvem um grande número de operações e trabalham com grande carga de dados. Assim, quando esses algoritmos são adaptados para o aumento dimensional dos quadros dos vídeos, tem-se que o esforço computacional de se processar uma única imagem é multiplicado pelo número de quadros total do vídeo. Assim sendo, para utilização de algoritmos de super-resolução para vídeos, é desejável que hajam recursos que permitam reduzir o tempo de processamento.

Nesse cenário, este trabalho propõe uma estratégia para reduzir o tempo de processamento do aumento dimensional dos quadros de vídeos, utilizando técnicas de processamento

seletivo que permitam que nem todos os dados sejam processados para gerar o vídeo de alta-resolução. Combinado com essas técnicas de processamento seletivo, um *framework* para distribuir e processar paralelamente os dados é proposto com o intuito de aumentar significantemente o desempenho dos algoritmos de aumento de resolução.

Para isso, o Capítulo 2 revisa os conceitos de super-resolução, computação paralela e apresenta a noção de simplificação. Em geral, nesse capítulo é feita uma ampla exposição dos conceitos que serão utilizados nos demais capítulos desta dissertação.

Em seguida, no Capítulo 3, são descritos os três algoritmos usados para testar a estratégia proposta para decomposição e processamento dos dados. Todos os algoritmos escolhidos funcionam no domínio espacial, sendo que os dois deles se utilizam modelos Bayesianos, e o terceiro busca uma representação esparsa da imagem de baixa-resolução que corresponda à uma projeção da imagem de alta-resolução.

O Capítulo 4 apresenta a estratégia de decomposição, a redução da carga de dados a ser processada, a distribuição dos dados e o processamento. Essa estratégia é descrita em forma de um *framework*, projetado de forma que não esteja acoplado ao algoritmo de aumento de resolução. Dessa forma, permite que essa estratégia possa ser utilizada por diferentes algoritmos de super-resolução sem que haja a necessidade de reprojetar ou modificar o algoritmo escolhido.

O Capítulo 5 apresenta os resultados das simulações feitas para testar o *framework* proposto. Nesse capítulo é feita uma exposição e uma análise do desempenho do *framework* com relação ao tempo e à qualidade do sinal ampliado.

Para finalizar, o Capítulo 6 discursa sobre as considerações finais do trabalho, apresentando uma reflexão sobre os resultados e expondo novas perspectivas da proposta para trabalhos futuros.

Capítulo 2

Referencial Teórico

Este capítulo traz uma breve apresentação dos conceitos relacionados ao desenvolvimento deste trabalho. Os conceitos são apresentados em seções, as quais foram divididas em Aumento de Resolução de Imagens e Vídeos (Seção 2.1), Simplificação de Algoritmos (Seção 2.2), Computação Paralela (Seção 2.3), Paralelismo em Processamento de Vídeos (Seção 2.4) e Métricas de Análise de Desempenho (Seção 2.5).

2.1 Aumento de Resolução de Imagens e Vídeos

Aumento de resolução de imagens envolve o estudo de métodos e algoritmos que produzem imagens de alta-resolução, a partir de imagens de resoluções menores. Geralmente, essa transformação baixa-para-alta resolução considera a transformação com a melhor qualidade possível. Assim, o nível de detalhes mantido num processo de comparação que utiliza imagens de alta resolução, que são reduzidas para imagens de resolução menores, e novamente transformadas em imagens de alta-resolução, definem o desempenho da qualidade do algoritmo.

Os métodos mais comuns de aumento da dimensão espacial (resolução) de imagens e vídeos consistem de técnicas de interpolação. Logo, diferentes técnicas de interpolação produzem diversos efeitos nas imagens geradas, como pode ser visto na Figura 2.1.

(a) (b)

(c) (d)

Figura 2.1: Exemplos de uso de diversas técnicas de interpolação: (a) original, (b) vizinho mais próximo, (c) bilinear e (d) bicúbica.

Na Figura 2.1-(b), nota-se que o método de interpolação do vizinho mais próximo deixa evidente o efeito de *aliasing*. Isso ocorre porque as informações de contornos e detalhes representam componentes de alta-frequência. Quando amplia-se a imagem, é reconstruída uma informação a partir de uma sub-amostragem.

Como pode ser visto nas Figuras 2.1-(c) e 2.1-(d), quando utilizam-se técnicas de interpolação bilinear e bicúbica o efeito de *aliasing* é amenizado, pois a informação das altas frequências perdidas é reconstruída a partir de uma composição suavizada. Este efeito também pode ser amenizado por meio de técnicas *antialiasing*, onde utilizam-se filtros que permitem a passagem de frequências baixas.

(a) (b)

Figura 2.2: (a) Imagem ampliada utilizando interpolação bilinear e (b) imagem obtida da filtragem da original com gaussiano passa-baixa.

Comparando as Figuras 2.2-(a) e 2.2-(b) é possível notar uma relação entre a operação de aumento dimensional por interpolação e as operações de filtragem. Destas figuras, é possível observar que o resultado ampliado por interpolação e a operação de suavização da imagem dão resultados semelhantes.

O redimensionamento que não utiliza interpolação é chamado de super-resolução. Este termo pode ser confuso, portanto o termo *zoom digital* é adotado para a operação de aumento nas dimensões da imagem, enquanto super-resolução é usado para a operação de fusão de diversas imagens em baixa resolução em outra com alta resolução [28]. O problema da super resolução é formulado da seguinte maneira: sendo $y_h \in \mathbb{R}^{N_h}$ a imagem original em alta-resolução, representada como um vetor de N_h *pixels* de largura. Sejam também o *operador de borramento* e o *operador de redução*, $\mathbf{H} : \mathbb{R}^{N_h} \rightarrow \mathbb{R}^{N_h}$ e $\mathbf{S} : \mathbb{R}^{N_h} \rightarrow \mathbb{R}^{N_l}$, respectivamente. O operador $\mathbf{H}$ realiza uma filtragem passa-baixa na imagem, enquanto que o operador $\mathbf{S}$ realiza uma redução por um fator inteiro s, diminuindo o tamanho da imagem. Sendo $z_l \in \mathbb{R}^{N_l}$ a versão de baixa resolução da imagem original, dada pela Equação 2.1 abaixo,

$$z_l = SHy_h + v, \tag{2.1}$$

onde v é um ruído gaussiano.

Dado z_l, o problema consiste em encontrar $\hat{y} \in \mathbb{R}^{N_h}$ tal que $argmax|\hat{y} - y_h| \leq \epsilon$, onde ϵ é um erro tolerado para a aproximação. Devido à característica gaussiana de v, a estimação da máxima verossimilhança é obtida pela minimização de $\|SH\hat{y} - z_l\|_2$. Portanto, é possível notar que a redução do ruído v é uma característica importante no problema do aumento de resolução de imagens. Contudo, como a composição dos operadores $\mathbf{SH}$ é retangular, com mais colunas do que linhas, este sistema não pode ser invertido diretamente, sendo um sistema linearmente dependente e, portanto, tendo diversas soluções.

Diferentes abordagens de trabalho foram tratadas para resolução do problema de aumento de escala, separando os operadores **S** e **H**. Alguns trabalhos assumem que não há efeito de borramento associado. Essas abordagens consistem de métodos de interpolação. Por outro lado, se há um efeito de borramento envolvido, então o processo de recuperação da imagem aumentada consiste de um processo de remoção de borramento (*deblurring*) [57, 94]. Sendo assim, diversos algoritmos para aumento de resolução de imagens foram propostos para resolver o problema da inversão de forma linearmente estável [16, 26, 31, 44, 55, 73, 76, 81].

2.2 Processamento Simplificado

Algoritmos são estratégias utilizadas para resolver um problema, considerando um conjunto de regras bem-definidas [19]. Assim, embora existam diversas formas para se solucionar um problema, distintos algoritmos podem ser eficazes para resolução com diferentes graus de eficiência.

No contexto de análise de algoritmos, eficiência consiste em uma baixa complexidade na utilização do espaço (memória e dados) ou na de execução (CPU). Devido à grande quantidade de informações que costumam ser manipuladas em aplicações de vídeo digital, a eficiência dos algoritmos associados a este tipo de aplicação é de extrema relevância. Portanto, para que um algoritmo seja eficiente e compatível com uma arquitetura de hardware, algumas estratégias devem ser tomadas. As estratégias de simplificação de processamento usadas neste trabalho são definidas em *nível de dados* e em *nível de instrução*.

2.2.1 Simplificação do Problema em Nível de Dados

A simplificação do processamento em nível dados é uma abordagem muito comum em em processamento de sinais, imagens e vídeos. Dessa forma, define-se essa simplificação

como:

1. A *redução da quantidade de dados* a ser utilizada para caracterizar a solução do problema.

2. A adoção na *divisão dos dados* simplifica o problema.

Os sinais de vídeo tem como característica uma grande quantidade de informação, o que faz com que os sistemas que os processem sejam capazes de executar grandes quantidades de informação, em curtos intervalos de tempo. Essa condição requer que algumas transformações sejam aplicadas para que haja uma redução na quantidade de informação a ser processada ou que tal informação seja dividida de forma que vários processadores dividam o esforço computacional.

A transformação nos dados a serem processados deve condicioná-los às condições definidas para a simplificação do processamento em nível de dados, podendo explorar tanto as dimensões espaciais quanto temporais do vídeo. Para a simplificação, três abordagens são utilizadas, as quais são *quantização, particionamento e processamento seletivo*.

A abordagem de *particionamento* consiste em dividir cada quadro do vídeo em sub-imagens (blocos) de maneira a permitir que estas sub-imagens sejam processadas de forma independente. O particionamento é uma simplificação que provê independência de dados, sendo uma estratégia dividir-para-conquistar, que permite a paralelização do cálculo. Os blocos particionados podem consistir de linhas ou colunas inteiras, ou sub-imagens de largura e altura arbitrárias. As partições também podem ser feitas considerando intervalos temporais, criando divisões igualmente distribuídas ou assimetricamente distribuídas. No padrão JPEG [47], a simplificação por particionamento é utilizada, em um passo denominado *Block splitting*, em que a imagem codificada é dividida em macro-blocos, que são transformados via DCT [34] e, em seguida, quantizados. Essa técnica é vastamente usada por diversos padrões de codificação de imagens e vídeos.

O *processamento seletivo* é uma abordagem de particionamento inteligente. Isto é, ao invés de apenas particionar geometricamente o vídeo em blocos, são extraídas informações de maior importância, classificadas como *regiões de interesse*, onde são aplicados os cálculos mais custosos. Assim, os algoritmos mais complexos são usados nas regiões de interesse, enquanto as demais regiões não são processadas ou recebem um processamento mais simples.

2.2.2 Simplificação do Problema em Nível de Instrução

Vídeos digitais possuem uma enorme quantidade de informação redundante, o que implica em um processamento com uma numerosa quantidade de operações também redundantes. Portanto, a densidade de instruções por dados é uma grandeza que deve ser considerada quando deseja-se otimizar sistemas de processamento de vídeos. Toda operação sobre um dado é relevante, uma vez que é alta a probabilidade dessa operação ser efetuada sobre outro dado. Assim, a redução na densidade de operações por dados consiste na *simplificação em nível de instrução*, que pode ser obtida por meio de *redução por transformada* ou de *redução por aproximação*.

A simplificação via *redução por transformada* consiste em aplicar uma transformação nos dados para reduzir a quantidade total de tempo consumido para um determinado processamento, mantendo o mesmo resultado numérico final. Em processamento de sinais, a utilização desse tipo de simplificação está presente em diversos problemas. Por exemplo, a suavização de bordas de imagens por filtros passa-baixa pode ser feita deslizando-se uma máscara sobre toda o espaço de *pixels*, gerando um processo iterativo que executa a mesma operação diversas vezes sobre diferentes regiões da imagem. Por outro lado, esse mesmo efeito pode ser obtido aplicando-se a transformada de Fourier sobre todo o domínio da imagem, gerando coeficientes nulos no espaço transformado, e retornando ao espaço original dos *pixels* pela transformada inversa [34].

Transformações de quaisquer tipos podem ser consideradas, mesmo que não tenham como definição as transformadas matemáticas. Por exemplo, considerando que as implementações em hardware de operações de divisão e multiplicação são muito mais complexas e lentas que outras, tais como deslocamento de bits e adição, a substituição das primeiras pelas segundas operações podem favorecer o desempenho em muitos casos. A transformação da função-objetivo em outra mais eficiente, muitas vezes é uma abordagem que favorece muito o desempenho computacional da resolução do problema [13].

Quando o resultado numérico é diferente daquele gerado pelo algoritmo não-simplificado, define-se a abordagem como sendo a *redução por aproximação*. Essa estratégia é semelhante à *simplificação por transformada*, mas neste caso toleram-se erros de aproximação. A ideia é minimizar os erros, mas considerando-se resultados sub-ótimos em um tempo menor, procurando-se otimizar a relação entre a precisão do resultado e tempo consumido. Exemplos dessa abordagem consistem na utilização de *lookup tables* para evitar cálculos complexos [68]; realização de operações inteiras sobre dados de ponto flutuante, tornando os cálculos mais rápidos, mas perdendo precisão [56]; e outras abordagens que toleram perdas sem prejudicar a semântica da informação.

Há uma relação entre a simplificação em nível de instrução e em nível de dados. A primeira parte da definição da simplificação em nível de dados implica na simplificação em nível de instrução, uma vez que reduzindo a quantidade de dados a serem processados, as instruções também são diminuídas por um fator de redução η tal que:

$$\eta = \frac{n(a)}{n(A)}, \tag{2.2}$$

onde $n(x)$ é o número de elementos do conjunto x.

2.3 Computação Paralela

Computação paralela é uma estratégia computacional em que o processamento é efetuado simultaneamente em uma arquitetura com mais de uma unidade de processamento. Esta estratégia parte do princípio de que problemas computacionais podem sem resolvidos a partir de programas formados por um conjunto de tarefas [23].

Essas tarefas, por sua vez, podem possuir ou não independência entre si. Caso possuam, elas formarão uma cadeia de processos que são dispostos em uma ordem, caracterizando uma sequência de passos a serem seguidos de forma ordenada, o que é denominado de *processamento sequencial*. Por outro lado, quando duas ou mais tarefas que compõe o programa que soluciona certo problema não dependem de ordem, elas podem ser executadas de forma independente por unidades de computação distintas. Assim, a execução independente dessas tarefas pode ser realizada em uma arquitetura paralela concomitantemente [5].

Essas arquiteturas paralelas funcionam sobre três tipos básicos de abstrações de processamento, que são o paralelismo de tarefas, o de instrução e o de dados. O desenvolvimento de um algoritmo em paralelo pode ser independente da arquitetura utilizada na implementação, contudo a estratégia de divisão do processamento é feita sob essas abstrações.

O paralelismo de instrução é utilizado na execução simultâneas de múltiplas operações independentes. Já o paralelismo de tarefas consiste na execução de distintas tarefas que atuam sobre dados independentes. Finalmente, o paralelismo de dados é adequado em aplicações onde a mesma tarefa executa sobre diferentes dados [23].

No paralelismo de dados a aplicação é decomposta subdividindo-se o espaço dos dados para que processadores independentes trabalhem em cada sub-espaço criado. Essa estratégia de decomposição envolve pouco compartilhamento dos dados, cabendo ao desenvolvedor garantir que este compartilhamento ocorra de forma sincronizada. Geralmente,

o paralelismo de dados é a abordagem mais usada nas implementações distribuídas, pois as tarefas operam sobre os distintos dados, sem exigir muita gerência e sem perder desempenho devido ao *overhead*.

Neste trabalho é dado ênfase a esse tipo de paralelismo, uma vez que os vídeos apresentam essa característica. Isto é, vídeos são compostos por grandes volumes de dados separáveis, onde são realizadas operações redundantes que podem ser processadas independentemente.

Após um algoritmo paralelo ser proposto, outro passo do desenvolvimento é definir o modelo de programação a ser usado. Esse modelo consiste na interface entre os recursos e as ferramentas que permitem a concretização do algoritmo por meio de uma implementação. Essa implementação, diferente do algoritmo, que é uma entidade abstrata, é dependente da arquitetura onde ele será executado.

Existem três principais modelos de programação para computação paralela, os quais são a implementação em memória compartilhada, a passagem de mensagem e o híbrido [35]. Cada modelo tende a ser mais adequado à arquitetura onde ocorrerá a execução, que pode ser classificada como arquitetura de memória compartilhada ou distribuída.

2.3.1 Arquiteturas Paralelas

Máquinas paralelas são computadores que suportam o processamento simultâneo de mais de uma tarefa, pois elas tem mais de uma unidade de processamento. Elas são classificadas de acordo com a organização do *hardware* que suporta o paralelismo, podendo ser de *memória compartilhada* ou de *memória distribuída*.

Memória Compartilhada

Máquinas paralelas com *memória compartilhada* consistem de um conjunto de processadores independentes que computam paralelamente, mas que concorrem no acesso aos

dados de memória utilizando um único barramento de acesso. Geralmente, essas arquiteturas são ideais para sistemas *CPU-bound*, onde a computação é mais frequente que o acesso à memória. Devido à essas características e ao barramento compartilhado, a presença de memória cache fornece uma grande melhoria no desempenho do sistema, uma vez que reduz a concorrência ao barramento. Essa organização é ilustrada na Figura 2.3.

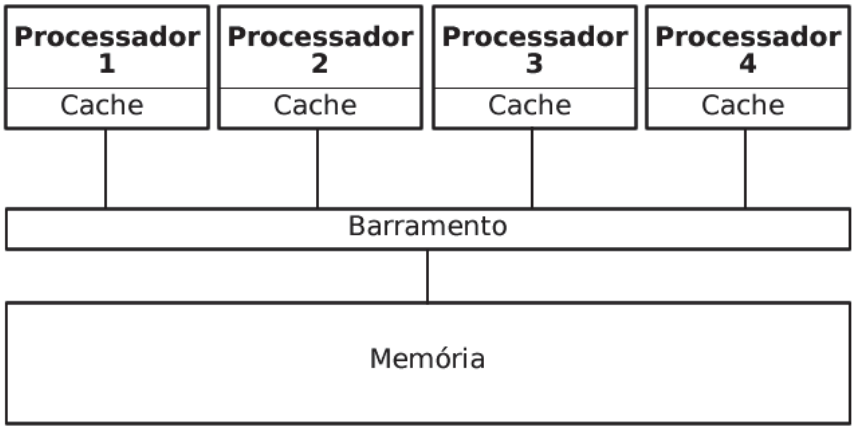


Figura 2.3: Diagrama esquemático de uma arquitetura de memória compartilhada.

Assim, arquiteturas de memória compartilhada apresentam uma complexidade no suporte escalável do número de processadores. Dessa forma, quanto maior o número de processadores concorrendo ao acesso à memória, maior será a concorrência pelo barramento, aumentando exponencialmente o efeito do *Gargalo de Neumann* [59], e prejudicando a eficiência. A maioria desses sistemas tem a quantidade de processadores limitada graças à contenção do barramento. Melhorias na utilização do barramento, tal como restrições de processadores por barramento combinados com múltiplos barramentos comutados, permitem um aumento de processadores simétricos [39].

Memória Distribuída

Máquinas paralelas com *memória distribuída* evitam a limitação das arquiteturas de memória compartilhada, utilizando memórias dedicadas para cada processador, conforme

ilustrado na Figura 2.4. A comunicação entre processador e sua memória local é feita exclusivamente por um barramento dedicado, sem concorrência. Quando um processador necessita de dados que estão sendo processados ou armazenados no espaço de outro processador, ele requisita e o processador proprietário dessa informação concede por passagem de mensagens. Portanto, há a comunicação entre processador-processador, que é feita por um canal (circuito de conexão), o qual pode ser um barramento específico ou uma rede.

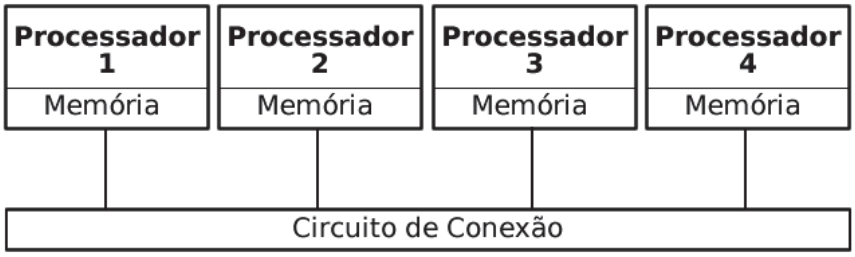


Figura 2.4: Diagrama esquemático de uma arquitetura de memória distribuída.

A principal vantagem da arquitetura de memória distribuída é que o acesso à memória local é muito mais rápido e sem conflito no acesso ao barramento, o que torna essa arquitetura muito mais escalável. Contudo, o compartilhamento de dados nessa arquitetura requer muito mais cuidado para que a sincronização dos dados seja mantida. Além disso, a sincronização de dados nos sistemas de memória distribuída costuma ser delegada ao programador, o que exige um maior esforço no desenvolvimento de aplicações para esses sistemas.

Nas máquinas de memória distribuída, a comunicação inter-processo é, geralmente, feita por meio de um paradigma de passagem de mensagem. Caso haja muita comunicação entre os processos para a sincronia dos dados, o sistema pode ter um prejuízo no desempenho devido ao *overhead* da comunicação.

Dessa forma, o principal problema em sistemas de memória distribuída está na gerência da comunicação entre processadores. Portanto, a escolha da arquitetura paralela – memória compartilhada versus memória distribuída – depende do comportamento dos dados. Sis-

temas com poucas estruturas de dados globais, onde os dados são executados localmente, e com pouca sincronização são candidatos a se beneficiarem de arquiteturas de memória distribuídas. Por outro lado, sistemas que possuem poucas estruturas localizadas e que precisam ter frequentes sincronizações de dados, se beneficiam da organização de memória compartilhada.

Para grandes sistemas de computação paralela, uma arquitetura formada por *clusters de multiprocessadores* é geralmente adotada. Esses *clusters* são organizados semelhantemente às arquiteturas de memória distribuída, mas os processadores individuais são substituídos por grupos de multiprocessadores simétricos. Essa disposição permite melhorar as limitações de ambas as arquiteturas, aumentando a eficiência de cada nó individual e permitindo escalar o sistema para acima dos milhares de processadores. A Figura 2.5 ilustra essa arquitetura, no qual cada unidade de processamento – da perspectiva da arquitetura de memória distribuída – é um grupo de processadores organizados em memória compartilhada.

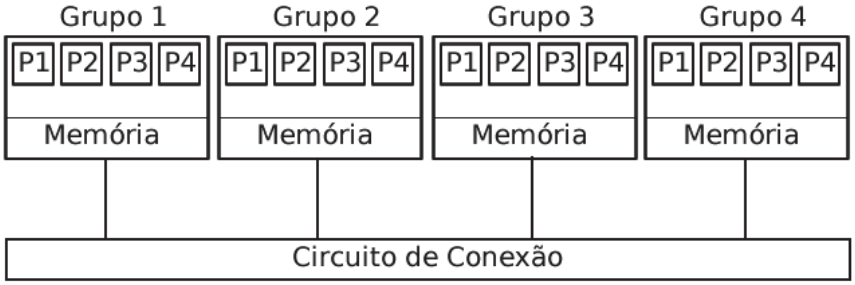


Figura 2.5: Diagrama esquemático de uma arquitetura híbrida.

Nesse cenário, o presente trabalho apresenta um *framework* para execução em uma arquitetura híbrida, distribuindo a carga de dados ao longo de um sistema de memória distribuída, onde cada nó possui um conjunto de processadores, distribuindo assim o esforço computacional em cada processador de cada nó.

2.3.2 Processadores de Propósito Geral

Computadores podem resolver diversos tipos de problemas, desde que lhes sejam submetidos diferentes programas, cada qual com um propósito específico. Com o intuito de reutilizar um mesmo hardware, os processadores de propósito geral (PPGs) podem executar diversos tipos de softwares, independente do propósito desses.

Atualmente, os PPGs possuem grande desempenho computacional, com recursos de paralelização de instruções e de tarefas. Além disso, alguns conjuntos de instruções dedicados para propósitos específicos têm sido incorporados nesses processadores. Por exemplo, tecnologias como MMX e SSE [2], foram desenvolvidas para melhorar o desempenho do processamento de mídias digitais. Instruções SIMD [75] têm sido incorporadas nos conjuntos de instruções para permitir explorar o paralelismo em nível de dados, o que permite operar sobre dados de imagens e vídeos em baixo nível de forma muito eficiente.

Ainda que os processadores utilizados em computadores pessoais, servidores e estações de trabalho tenham um grande poder de processamento, eles costumam ser grandes consumidores de energia. Portanto, não são ideais para a utilização em dispositivos com bateria. Além do consumo, o avanço nos processadores de propósito geral tem dado aos usuários uma razoável experiência em aplicações de mídia digital. Por isso, os recursos dos computadores pessoais tem sido referência para a industria de TV digital, no desenvolvimento de softwares de visão computacional e de processamento de vídeo [61].

Neste cenário, devido à grande influência das aplicações de mídia digital e computação gráfica voltadas ao usuário comum, que impulsionam a demanda por eficiência nos processadores de propósito geral, nos últimos 10 anos a indústria tem incorporado por padrão a tecnologia de múltiplos núcleos nas CPUs comercializadas para o aumento do desempenho. Porém, é possível observar que essa arquitetura de memória compartilhada, que já está incorporada na maioria dos computadores comercializados, ainda é vastamente aproveitada

pelos softwares de mídia atuais. Logo, há boas perspectivas para a pesquisa e o desenvolvimento de implementações de algoritmos paralelos para processamento de sinais [54].

2.3.3 Implementação para Arquiteturas Paralelas

Dada uma determinada arquitetura que suporte a execução de programas em paralelo, o desenvolvimento e a implementação de um algoritmo paralelo consiste de quatro etapas. A primeira consiste em identificar e criar os passos do algoritmo que podem ser executadas em paralelo. A segunda consiste em adotar estratégias de simplificação para implementar esses passos em paralelo. A terceira consiste na concretização do algoritmo em si, por meio da escrita de um programa, que envolve a escolha de um paradigma de programação, de um modelo, de uma linguagem de programação e de uma interface de implementação. Por último, um estilo de programação e de implementação deve ser adotado para que a aplicação funcione de acordo com o modelo, permitindo manutenibilidade do código.

Etapa 1 - Identificação do Paralelismo

A primeira etapa da paralelização consiste em identificar as porções de código onde o paralelismo pode ser explorado. Para isso, deve-se manter o compromisso de que a *corretude* do algoritmo deve ser a mesma, independente da implementação ser serial ou paralela. Isto é, para um programa que realiza cálculos em paralelo estar correto, ele deve produzir os mesmos resultados da versão serial. Mais especificamente, os resultados do programa não devem depender da quantidade de processadores usada. A mudança na quantidade de 1 para n processadores deve impactar somente no tempo necessário para a resolução, nunca sobre os resultados produzidos [52].

A identificação mais óbvia de condições de paralelismo consiste em determinar os estágios do programa que não compartilham dados. O processamento de dados independentes geram módulos bem definidos, que em uma implementação serial consistiria de uma sequên-

cia executável independente de ordem. Ou seja, se um algoritmo serial possui as etapas e_1, e_2, ..., e_n em que a produção do resultado seja o mesmo, independendo da ordem em que elas são executadas, então essas etapas são candidatas à executarem em paralelo. Contudo, quando há compartilhamento ou dependência dos dados, a identificação dessas condições é menos trivial. Se um programa serial escreve em um local de memória em um primeiro passo, e então lê esse local em um segundo, uma implementação paralela desses dois passos pode causar uma leitura que antecede a escrita. Essa é uma situação conhecida como *condições de corrida*, o que faz com que o algoritmo produza resultados incorretos [77].

Para identificar as situações em que o paralelismo não é possível, Bernstein [12] formalizou um conjunto de condições em que as condições de corrida podem ocorrer. Assim, para dois passos de computação, P_1 e P_2, eles podem ser executados em paralelo se nenhuma das seguintes condições ocorrerem:

1. *write-after-read* (WAR): P_1 lê de um local que, posteriormente, é escrito por P_2 (ou vice-versa);

2. *write-after-write* (WAW): P_1 escreve em um local que, posteriormente, é sobrescrito por P_2 (ou vice-versa);

3. *Read-after-write* (RAW): P_1 escreve em um local que, posteriormente, é lido por P_2 (ou vice-versa);

Etapa 2 - Estratégia de Decomposição

A segunda etapa na preparação do programa para execução em paralelo consiste em decompor os passos do algoritmo em partes que não gerem as condições de corrida mencionadas acima. A primeira forma de decomposição é identificar as fases em que exista dependência entre os dados. Esse tipo de decomposição permite que diferentes proces-

sadores executem diferentes tarefas. Essa abordagem explora o paralelismo de tarefas. Por exemplo, um algoritmo de visão computacional pode delegar a um processador a tarefa de segmentar regiões nos quadros, enquanto outro processador fica encarregado de realizar funções de filtragem.

Outra estratégia de decomposição em paralelo consiste em sub-dividir os dados em porções menores, aplicando a mesma operação nessas partições paralelamente, aproveitando o paralelismo de dados. Um exemplo dessa abordagem, em um vídeo, pode consistir de dividir um quadro com resolução 2000×2000, em imagens de 200×200, fazendo com que 100 processadores independentes realizem a mesma operação nessas imagens, ao invés de operar serialmente sobre o quadro inteiro.

Uma abordagem que combina o aproveitamento do paralelismo de dados com o de tarefa é o *pipelining*. Essa abordagem divide um diferente estágio de computação em outros sub-estágios sequenciais. Caso haja muitos dados independentes a serem passados pelo *pipeline*, cada estágio pode ser realizado em um conjunto de dados independente ao mesmo tempo. Por exemplo, em um vídeo, um algoritmo de visão computacional pode realizar em cada quadro quatro operações: conversão de cor para escala de cinza, binarização da imagem, segmentação e identificação das regiões. Cada uma dessas operações consiste de um estágio. Assim, enquanto o primeiro quadro está no último estágio, o segundo pode ser processado no terceiro, etc. Dessa forma, se cada um desses estágios for desenvolvido de forma que execute no mesmo tempo, este *pipeline* provê um ganho no desempenho.

Etapa 3 - Modelo de Desenvolvimento

A terceira etapa do desenvolvimento de um software para computação em paralelo consiste em escolher o modelo de programação. Essa escolha é de importância fundamental, pois afeta a seleção do paradigma da linguagem de programação e a biblioteca escolhida

para implementação. Existem dois tipos básicos de modelos para desenvolvimento, os quais são os modelos de memória compartilhada e os modelos com passagem de mensagem [58].

Modelos de programação de memória compartilhada, como sugere o nome, são voltados para o desenvolvimento em arquiteturas de memória compartilhada. Ou seja, todos os dados acessados pela aplicação são acessíveis por todos os processos. Assim, cada processador pode buscar ou armazenar os dados em qualquer posição da memória sem se comunicar com outro processador. Esse modelo se caracteriza pela necessidade de sincronização das estruturas de dados.

Modelos que utilizam passagem de mensagens correspondem ao modelo de abstração de arquiteturas de memória distribuída. Assim, cada processador possui um conjunto de dados particular e os dados são compartilhados por troca de mensagens. Nesse modelo, as primitivas de envio e recebimento são utilizadas para sincronização dos dados.

Esses dois modelos, embora formem duas abstrações que correspondem às arquiteturas correspondentes, sua utilização não é restrita. Isto é, há a possibilidade de implementar um sistema de memória compartilhada que execute em uma arquitetura de memória distribuída por meio de *middlewares* que realizem essa interface [7]. Por outro lado, o modelo de passagem de mensagens pode ser utilizado para funcionar sobre um sistema de memória compartilhada, abordagem muito comum em arquiteturas de multiprocessadores simétricos.

Etapa 4 - Implementação

Finalmente, esta é a última etapa, e ela é consequência da escolha do modelo de desenvolvimento, que consiste do estilo de implementação. Para explorar o paralelismo de tarefas, a implementação de um modelo cliente-servidor, onde um processo executa uma tarefa e delega a outro processo uma segunda tarefa, é um forte candidato.

Para aproveitar o paralelismo de dados, os programas devem paralelizar as etapas iterativas. Isso se dá porque iterações representam tarefas redundantes, que costumam ser utilizadas para separar passos de computação em programas seriais, mas que costumam caracterizar etapas separáveis em blocos independentes e processáveis em paralelo. Para esse objetivo, há a paralelização das iterações (operações redundantes) em memória compartilhada e para modelos de passagem de mensagens.

Em sistemas de memória compartilhada, essa decomposição pode ser representada como uma iteração paralela. Há uma série de tecnologias que permitem esse tipo de abordagens, tais como OpenMP [18], *pthreads* [74], CUDA [72] etc. Segundo Bernstein [12], iterações formam as condições que mais favorecem a paralelização de programas sem condições de corrida e sem sincronização. Para isso, basta que elas não tenham em seu corpo as condições WAR, WAW e RAW. Caso haja essas condições, as variáveis que geram conflitos devem ser tratadas como *regiões críticas* e sincronizadas. Essas regiões críticas são porções de memória que devem ter a garantia de serem acessadas somente por um processador de cada vez.

Modelos que utilizam passagem de mensagens implementam o estilo de desenvolvimento chamado *Single Program, Multiple Data* (SPMD) [46, 48]. Nesse paradigma todos os processadores executam exatamente o mesmo código, mas aplicando as instruções em porções de dados diferentes, exatamente como na definição de paralelismo de dados. No início da execução, as variáveis escalares são replicadas entre os processadores, que as tomam como propriedade e as isolam dos outros processadores. Se houver necessidade de compartilhamento dos valores dessas variáveis, caberá ao programador explicitar esse comportamento, fazendo com que a informação seja trocada por passagem de mensagens.

2.4 Paralelismo em Processamento de Vídeo

Operações em vídeo digital são classificadas em três níveis, as quais são de controle, de características e de *pixels*. Esses níveis se diferem pela forma como estão relacionados com os dados de entrada, consistindo de três níveis semânticos diferentes [22, 24, 67]. Processamento em nível de *pixels* é a abstração de mais baixo nível, consistindo em pegar um conjunto de *pixels* como entrada e produzir outro conjunto de *pixels* como saída. Processamento em nível de características consiste em ter como entrada um conjunto de *pixels* e extrair atributos como saída. O nível mais elevado de operações, o nível de controle, consiste em usar como entrada um conjunto de *pixels* e interpretar seus atributos, a fim de produzir um processamento baseado nessa informação.

Como pode ser notado na Figura 2.6, essa classificação é hierárquica porque o nível mais alto depende de um conjunto de operações de um nível abaixo. Assim, as operações em níveis de *pixels* são mais numerosas que em nível de características, que são mais numerosas que em nível de controle. Essa hierarquia deve ser considerada em uma implementação paralela, pois o ganho na velocidade nos níveis mais baixos implica em um aumento no desempenho das operações dos níveis superiores.

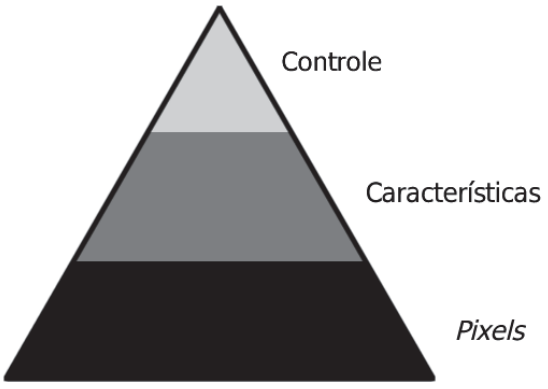


Figura 2.6: Hierarquia das operações em processamento de vídeos.

2.4.1 Paralelismo em Nível de *Pixels*

As operações de mais baixo nível são aquelas que simplesmente transformam dados de *pixels* em dados de *pixels*. Isso significa que essa operação tem como entrada e saída valores possíveis para os *pixels* de um *frame*. Operações em nível de *pixels* incluem transformações em espaços de cores, filtragens, realce de bordas, transformação de domínios, entre outros. Este é o principal escopo de estudo na área de processamento de imagens e vídeos. Assim, o principal objetivo das operações em nível de *pixels* é editar a imagem para uma determinada aplicação, a qual é definida em um nível superior.

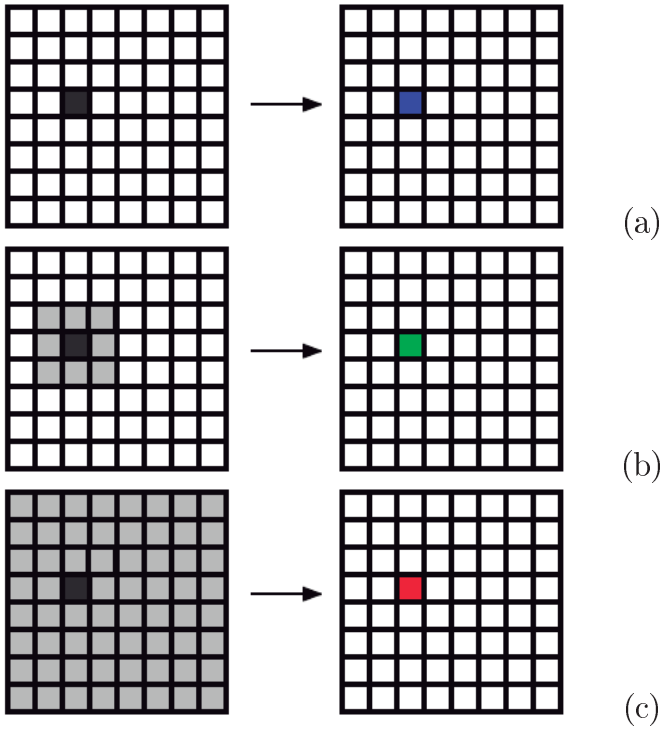


Figura 2.7: Paralelismo em nível de *pixel*: (a) ponto-a-ponto, (b) vizinhança e (c) dependência global.

Processamento em nível de *pixels* recebe três classificações, as quais são ponto-a-ponto (ou pontual), dependente da vizinhança local e globalmente dependente [51]. O processamento ponto-a-ponto é formado por operações que utilizam um *pixel* como entrada e retornam outro *pixel* como saída, independente da informação de qualquer outro *pixel* no

vídeo. Operações de aritmética de *pixels*, operações lógicas, e correção gama são exemplos de processamentos em nível de *pixels*. Operações desse tipo são candidatas óbvias à paralelização devido à independência dos dados por instrução. Na Figura 2.7-(a) é ilustrada uma operação pontual, onde a operação efetuada no *pixel* destacado é feita sem qualquer dependência com outros *pixels* do vídeo.

Processamento dependente da vizinhança local, como o nome sugere, é aquele em que as operações efetuadas em um dado *pixel* são dependentes dos dados da vizinhança deste *pixel*. Em outras palavras, as operações são mais complexas que aquelas efetuadas ponto-a-ponto e a paralelização dessas operações são mais restritas, pois a independência dos dados processados não é mais pontual, mas restringida às áreas inter-dependentes. A Figura 2.7-(b) ilustra uma operação que gera uma área de dependência. Nessa figura, o *pixel* mais escuro é o transformado e a área que o contorna forma a vizinhança, que também é entrada para o algoritmo usado no processamento. Esse conjunto formado por *pixel* e vizinhança gera o *pixel* ilustrado em verde no quadro de saída. Essa é uma característica muito comum em diversos problemas de processamento de imagens e vídeos, como, por exemplo, os algoritmos de filtragem espacial [50].

Processamento com dependência global é formado por operações que utilizam todos os *pixels* do quadro como vizinhança para saída de um único *pixel*, conforme ilustrado na Figura 2.7-(c). Esse tipo de dependência costuma gerar iterações computacionalmente custosas, pois para cada *pixel* no quadro de entrada, todos os demais também são entradas. Além disso, há uma grande dependência dos dados, o que desfavorece a paralelização em nível de dados, requerendo algoritmos paralelos mais complexos. Contudo, esse tipo de problema é muito comum, tendo a transformada discreta de Fourier e as transformadas *wavelets* [34] como exemplos.

Os problemas mais notáveis relacionados ao processamento em nível de *pixels* são aqueles que utilizam operações matriciais. Álgebra linear é extensamente utilizada em proces-

samento de sinais digitais, sendo que a maioria dos algoritmos projetados para processar vídeos utilizam algum processamento de dados matriciais. Portanto, algoritmos que envolvem operações sobre matrizes são pontos-chave que merecem ser paralelizados.

O paralelismo no nível de *pixel* ocorre trivialmente quando aplica-se a simplificação de dados por *particionamento*, onde cada partição é um único *pixel* ou uma região independente. Assim, problemas definidos no nível mais baixo da hierarquia possuem como entrada dados facilmente estruturáveis, que favorecem o paralelismo em nível de dados.

2.4.2 Paralelismo em Nível de Características

Como o nome sugere, o nível de características consiste em utilizar as informações obtidas no nível de *pixels* para caracterizar propriedades do vídeo. Portanto, algoritmos que atuam neste nível de abstração se caracterizam por reduzir a quantidade de dados, quando comparados entrada e saída. Assim, a segmentação para detecção de regiões de interesse, extração de bordas, redução de dimensionalidade (LDA [29], PCA [66], etc.) e a extração de informações estatísticas são exemplos de operações definidas neste nível.

Dessa forma, enquanto algoritmos que trabalham em nível de *pixels* resolvem completamente alguns problemas, tais como correção gama e suavização de ruídos, os algoritmos que atuam no nível de características costumam ter como objetivo a redução e a preparação dos dados para um pós-processamento pelo nível de controle. No contexto do processamento simplificado, o nível de características é o que gera as regiões de interesse para o processamento seletivo. Algumas operações neste nível intermediário podem ser facilmente estruturadas para aproveitar o paralelismo em nível de dados.

2.4.3 Paralelismo em Nível de Controle

Processamento de controle é aquele que utiliza os dados do nível de característica para inferir informações. Essas informações interpretadas, geralmente, são utilizadas para o controle do fluxo de processamento da aplicação, inserindo uma compreensão semântica da informação. Sendo assim, não são raros os casos em que os algoritmos que trabalham em nível de controle sejam de inteligência artificial ou de aprendizado de máquina. Por isso, o nível de controle é o principal escopo da área de visão computacional.

Algoritmos em nível de controle realizam operações de classificação de regiões, reconhecimento de padrões ou tomam decisões com base nos dados gerados pelos algoritmos em nível de características. Esses tipos de tarefas são caracterizadas, principalmente, por operações de controle, com poucas repetições ou operações aritméticas redundantes. Assim, esses algoritmos possuem mais características seriais do que paralelas. Devido à essas propriedades, algoritmos no nível de controle tendem a ter dados irregularmente estruturados, com pouca redundância nos dados, o que os torna mais propensos a aproveitarem o paralelismo em nível de instrução.

2.5 Métricas de Análise

As métricas de análise buscam especificar critérios de comparação de alguma grandeza com o intuito de avaliar o desempenho do dado analisado com alguma referência. Essas métricas, geralmente, fornecem valores quantitativos que servem de indicadores para análise dessa grandeza.

Este conceito é muito relativo ao problema em que a métrica é aplicada. No contexto deste trabalho, tomam-se as métricas para desempenho do algoritmo em relação ao tempo e as métricas para analisar a desempenho de um vídeo em relação à sua qualidade visual.

2.5.1 Métricas de Desempenho de Algoritmos Paralelos

Implementar algoritmos paralelos é uma tarefa complexa, que consiste em otimizar a relação entre a divisão das tarefas em paralelo e a sincronização dos dados divididos. Essa relação ótima deve ser buscada porque, geralmente, o desempenho de um programa paralelo não aumenta automaticamente com o número de processadores.

Para obter um melhor desempenho, o desenvolvedor deve considerar diversos fatores. O primeiro é o balanceamento de carga entre os processadores que estão executando em paralelo, evitando que alguns deles consumam mais tempo de execução, enquanto outros fiquem ociosos. O segundo fator é o problema do algoritmo ser muito complexo, com muitos passos ou muitas estruturas de dados. Nesse caso, a implementação deve ser cuidadosamente escrita para evitar conflitos na sincronia dos dados ou cair no primeiro fator de influência. Outro fator que deve ser considerado é que todo algoritmo possui características paralelas e seriais em etapas distintas.

Assim, uma implementação pode reduzir muito o tempo de computação paralelizando algumas etapas do algoritmo. Entretanto, este ganho pode não compensar o esforço se a etapa serial for demorada demais. Por outro lado, uma implementação totalmente serial, mas que aproveita corretamente essa característica com *pipelining*, pode ter um custo-benefício melhor. Por isso, é importante a análise quantitativa de uma implementação em paralelo para conhecer o ganho obtido [71].

Um programa de computação em paralelo ideal é aquele em que o tempo de execução é reduzido por um fator inversamente proporcional ao número de processadores usados, ou seja, se há dois processadores executando em paralelo, o tempo de execução deve cair pela metade, se comparado com o tempo que seria gasto com um único processador. Assim, se três processadores são usados, o tempo deveria cair para um terço; se quatro, o tempo deve ser reduzido por quatro, e assim por diante. Formalmente, essa relação é calculada como uma grandeza chamada de *speedup* [21].

O *speedup* é definido como a razão do tempo de execução em um único processador e o tempo de execução em paralelo, conforme mostrado na Equação 2.3,

$$speedup(n) = \frac{\tau(1)}{\tau(n)}, \tag{2.3}$$

onde $\tau(n)$ é o tempo de execução para n processadores.

Uma aplicação é dita *escalável* se o *speedup* em n processadores é próximo a n. Ademais, a escalabilidade possui um limite. Isto é, a quantidade de operações em paralelo na aplicação é finita, e a distribuição em mais processadores começa a degradar o desempenho. Assim, o programador deve ter como objetivo desenvolver um sistema que otimize a escalabilidade.

Dada a importância da escalabilidade para o sucesso de uma implementação em paralelo, quando os tempos de execução de programas paralelos não executam conforme a Equação 2.3, é importante que eles possam ser mensurados por meio do *speedup*. Assim, Gustafson [37] apresenta uma definição de *speedup* escalável como sendo a medida que especifica se uma aplicação é escalável se, devido ao crescimento de processamento proporcional ao crescimento do número de processadores, o tempo de execução continuar sendo o mesmo.

Todavia, a escalabilidade pode não ser obtida por quatro razões principais. A primeira delas ocorre quando o algoritmo possui muitas etapas seriais. A segunda é quando uma das etapas seriais do algoritmo é predominante no tempo de execução. Assumindo-se que τ_s é a soma dos tempos das etapas seriais e τ_p é a soma dos tempos das etapas em paralelo, o *speedup* pode ser definido pela Equação 2.4,

$$speedup(n) = \frac{\tau_s + \tau_p}{\tau_s + \frac{\tau_p}{n}} \leq \frac{\tau(1)}{\tau(n)}. \tag{2.4}$$

Isso significa que o *speedup* total é limitado pela razão do tempo executado serialmente em

relação ao tempo executado em paralelo. Essa relação é conhecida como *lei de Ambdahl* [6].

O terceiro impedimento da escabilidade ocorre em aplicações com um alto número de comunicações ou sincronizações de dados. Isso é, as etapas seriais do algoritmo executam rapidamente e aparecem, frequentemente, entre as etapas paralelas, então o custo da sincronia aumenta significantemente. Esse comportamento pode ser modelado pela Equação 2.5,

$$speedup(n) = \frac{\tau(1)}{\tau_s + \frac{\tau_p}{n} + c\ln(n)}, \tag{2.5}$$

onde c é o número de comunicações realizadas para a sincronização. Portanto, da equação acima é possível notar que se c é um valor alto, o valor no denominador é predominante [30]. Isso significa que aumentando-se o número de processadores comunicantes, o valor do *speedup* começa a declinar.

Finalmente, o quarto impedimento para não se obter escalabilidade é a carga de dados desbalanceada. Se um sistema paralelo está desbalanceado, onde um único processador captura metade da carga de dados para o processamento, este sistema terá o *speedup* máximo de dois, não importando quanto outros processadores estejam envolvidos. Portanto, uma das maiores preocupações em sistemas distribuídos consiste em distribuir adequadamente a carga de dados para que todos os processadores trabalhem igualmente.

2.5.2 Métricas de Desempenho para Simplificações de Implementação

O objetivo do processamento simplificado é reduzir o espaço (consumo de memória) ou o tempo de processamento. Para esse último caso, este trabalho toma emprestado o conceito de *speedup* utilizado para mensurar o desempenho de programas paralelos. Assim, adapta-se a Equação 2.3 para medir o ganho ao adotar uma simplificação.

Para calcular o ganho em desempenho total, deve-se considerar o tempo de processamento utilizando-se 1 processador (serial) sem nenhuma simplificação. Considera-se que C é um algoritmo e S corresponda à versão deste algoritmo simplificado, $\tau_C(n)$ é o tempo de execução do algoritmo C e $\tau_S(n)$ é o tempo de execução do algoritmo S tal que $\tau_S(n) < \tau_C(n)$ para n processos. O desempenho total é dado pelo *speedup*, conforme a Equação 2.6,

$$speedup(n) = \frac{\tau_C(1)}{\tau_S(n)}, \tag{2.6}$$

tal que τ_c é o tempo de execução das partes complexas (não-simplificadas), e τ_s é o tempo de execução das partes simplificadas, considerando m simplificações.

2.5.3 Métricas Objetivas de Qualidade Visual

Métricas de qualidade tem o objetivo de em medir o nível de degradação da qualidade visual da imagem após qualquer tipo de processamento, incluindo transmissão e compresão. No caso de sinais que envolvem o sistema perceptual humano (tais como imagens e vídeos) há duas formas de mensurar a qualidade, as quais são qualitativamente (subjetiva) e quantitativamente (objetiva).

A análise subjetiva de um vídeo consiste em apresentar um conjunto de vídeos para um expectador e pedir a opinião sobre eles. Considerando que diferentes ações (de transmissão ou processamento) podem inserir distorções e artefatos, a ideia é perguntar qual a sequência (entre as apresentadas) possui melhor qualidade. Sendo assim, esse tipo de análise requer recursos humanos (voluntários), tempo e financeiros. Dessa maneira, modelos computacionais que mensurem a qualidade de vídeo de forma automática se tornam fundamentais para diversas aplicações.

De forma contrária à análise subjetiva, a análise objetiva de um vídeo consiste em utilizar apenas os dados para determinar a qualidade da informação, permitindo que al-

goritmos computacionais calculem essa qualidade. Assim, há duas abordagens a serem tomadas. A primeira consiste em considerar apenas o conteúdo dos dados analisados, o que mede a *fidelidade* do sinal analisado em relação ao sinal original. Métricas como o erro quadrático médio (*mean square error*, MSE) e a *relação sinal ruído de pico* (*peak signal noise ratio*, PSNR) são exemplos famosos desta abordagem.

A segunda abordagem de medidas objetivas consiste em considerar informações do sistema visual humano (SVH). Essas medidas objetivas são categorizadas de acordo com a relação entre o sinal original e o SVH. Ou seja, quanto mais informações sobre o sinal original, maior a capacidade de utilizar essas informações para a medida de qualidade. Assim, essas métricas são de referência completa (utilizam todo o sinal original), referência reduzida (parte da informação do sinal original) ou sem referência (considerando apenas o sinal analisado e conhecimentos do SVH) [27].

Dentro dessas métricas objetivas disponíveis, o tipo mais adequado depende do tipo de aplicação. Por exemplo, um sistema de transmissão em tempo real que precise monitorar a qualidade do vídeo recebido provavelmente não terá disponível todo o vídeo original para utilizar referência completa. Nesse caso, uma quantidade menor de informação pode ser enviada para que o receptor analise a qualidade do que está recebendo, permitindo que uma métrica de referência reduzida seja utilizada.

Dentro do contexto deste trabalho, onde as informações das simulações serão apresentadas no Capítulo 5, apenas métricas de referência completa são utilizadas. Dentro das métricas disponíveis, são utilizadas a PSNR e a SSIM [40].

Relação Sinal Ruído de Pico (PSNR)

A relação sinal ruído de pico (*peak signal to noise ratio*) é uma métrica de comparação da fidelidade que relaciona o máximo possível de potência de um sinal com a potência do ruído (erro). A unidade é o decibel (dB). Assim, embora seja uma métrica geral para

medir a distorção de sinais, no caso de vídeos, o PSNR é a relação entre a entrada e a saída de um processo de transmissão ou processamento [34]. Matematicamente, o PSNR é calculado utilizando pela Equação 2.7,

$$PSNR = 10log_{10}\left(\frac{PICO^2}{ERRO}\right), \tag{2.7}$$

onde $PICO$ é o valor máximo possível para o sinal (geralmente, 255 para vídeos), e MSE corresponde ao erro médio quadrático (*Mean Square Error*, MSE). O MSE é calculado como indicado na Equação 2.8,

$$MSE(x, y) = \frac{1}{M}\sum_{i=0}^{M} \|x_i - y_i\|^2, \tag{2.8}$$

onde M é a dimensão do sinal, x é o sinal de referência e y é o sinal analisado.

Assim, quanto maior o valor do PSNR, maior é a relação entre a potência do sinal e a potência do ruído, o que implica em uma maior fidelidade entre o sinal analisado e de referência. Valores de PSNR entre os 30dB e os 40dB são considerados aceitáveis. Em imagens, quando tem-se valores de PSNR acima de 40dB, os artefatos são quase imperceptíveis para o olho humano [41].

Índice de Similaridade Estrutural (SSIM)

SSIM (*structural similarity*) é uma métrica de referência completa onde o algoritmo utiliza a imagem original e sua versão degradada para obter uma estimação da qualidade visual da imagem degradada. O processo de comparação entre a referência e a imagem analisada é feito em um espaço de parâmetro tridimensional, onde os parâmetros são combinados a fim de gerar um único valor numérico, que permite definir a qualidade da imagem [89]. A ideia é esse valor concordar com os resultados obtidos a partir da análise subjetiva dos observadores humanos.

O algoritmo que computa a SSIM relaciona a imagem de referência x com a imagem analisada y, considerando informações de luminância $l(x, y)$, contraste $c(x, y)$ e estrutura $s(x, y)$, por meio das Equações 2.9–2.14

$$l(x, y) = \frac{2\mu_y\mu_x + C_1}{\mu_y^2 + \mu_x^2 + C_1}, \tag{2.9}$$

$$c(x, y) = \frac{2\sigma_y\sigma_x + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{2.10}$$

e

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.11}$$

onde

$$C_1 = K_1^2 L^2, \tag{2.12}$$

$$C_2 = K_2^2 L^2 \tag{2.13}$$

e

$$C_3 = \frac{C_2}{2} \tag{2.14}$$

são constantes tais que $K_1 < 1$ e $K_2 < 1$ e $L = 255$ [27]. Assim, a fórmula geral da métrica SSIM é dada pela Equação 2.15,

$$SSIM(x, y) = l(x, y) \cdot c(x, y) \cdot s(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2.15}$$

onde μ e σ corresponde à média e ao desvio padrão da amostra, respectivamente, e σ_{xy} à amostra de covariância.

Essa métrica foi escolhida para este trabalho porque consiste em um método robusto,

sendo popular devido ao seu desempenho. Por isso, alguns softwares já utilizam essa métrica em produção, tal como o software de referência do codificador H.264 [78, 88].

2.6 Considerações Finais

Neste capítulo foram apresentados os conceitos teóricos que servirão como embasamento e referência nos próximos capítulos deste trabalho. O primeiro conceito apresentado foi a descrição do problema de aumento de resolução de imagens e vídeos. Conforme descrito na Seção 2.1, matematicamente esse problema consiste em um modelo linear onde os dois operadores (**H** e **S**) precisam ser determinados a fim de obter a solução y_h.

Para obter esses operadores, diversas abordagens podem ser utilizadas, assim como apresentado no Capítulo 1. Além disso, o problema da determinação e modelagem desses operadores pode ser reduzido à um outro problema de otimização. Nesse trabalho serão adotadas três abordagens para solucionar esse problema da super-resolução, conforme será explicado no Capítulo 3.

Além da formulação geral do problema da super-resolução, este capítulo definiu e explanou sobre o conceito de simplificação, que consiste em encontrar estratégias para reduzir o tempo de processamento ou o consumo de memória. Esses conceitos serão utilizados no Capítulo 4, onde o *framework* proposto é finalmente detalhado. Também no Capítulo 4 serão utilizados os conceitos discutidos na Seção 2.3, uma vez que o *framework* proposto é desenvolvido para processamento paralelo. Finalmente, a Seção 2.5.1 explicou como funcionam as métricas que serão usadas no capítulo de resultados (Capítulo 5).

Capítulo 3

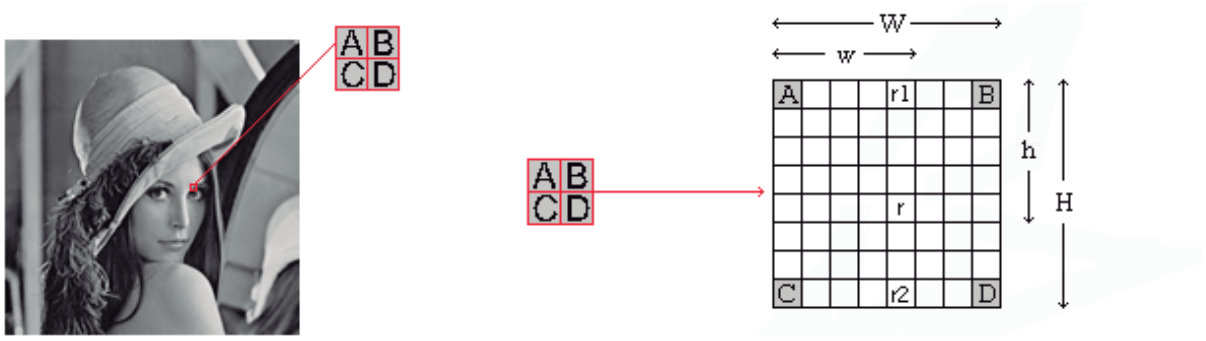
Algoritmos de Aumento de Resolução

O problema de aumentar a resolução espacial consiste em re-amostrar os *pixels* em uma imagem de dimensões maiores (alta-resolução), mantendo uma relação com os valores de *pixels* da imagem a ser aumentada (baixa-resolução). Isto é, para um fator de aumento P, a partir da imagem de baixa-resolução z_l de tamanho $h \times w$, deve-se "espalhar" os *pixels* em uma matriz y_h de dimensões $H \times W$, onde $W = Pw$ e $H = Ph$. Em seguida, a partir das informações espalhadas em y_h, deve-se preencher os *pixels* que formam "espaços em branco".

Essa ideia é ilustrada na Figura 3.1, a qual é mostrada como o bloco com dimensões 2×2, contendo os *pixels* A, B, C e D, são "espalhados" em um outro subconjunto, que equivalerá a esse bloco redimensionado com dimensões 8×8. Na Figura 3.1-(b), esse bloco de tamanho 2×2, ao ser "espalhado" no bloco 8×8, gera uma matriz "esparsa", o que introduz os "espaços em branco" que precisam ser preenchidos pelo algoritmo de aumento de resolução.

As coordenadas dos *pixels* A, B, C e D podem ser mapeadas da imagem de baixa-resolução para a imagem de alta-resolução. Uma vez feito o mapeamento dessas coordenadas, é necessário um algoritmo que re-amostre a intensidade dos *pixels* na imagem

transformada. Geralmente, isso é feito por um processo de interpolação. Assim, mesmo em algoritmos de super-resolução, que utilizam técnicas mais elaboradas para estimar esses "espaços em branco", a interpolação é comumente utilizada, sendo associada aos operadores de redução dimensional e de borramento, os quais foram apresentados na Seção 2.1.



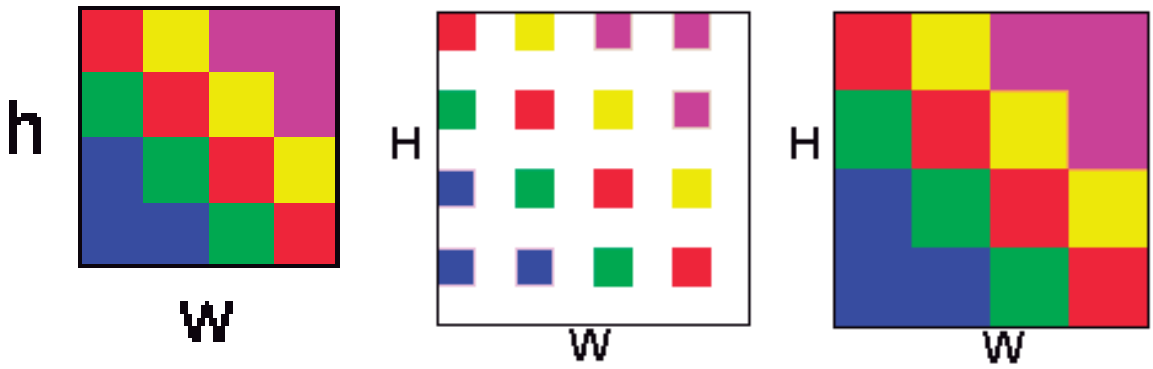
(a) (b)

Figura 3.1: Espalhamento dos *pixels* no aumento de resolução. (a) Um conjunto de *pixels* selecionado, (b) Espalhamento dos *pixels* A, B, C e D na imagem ampliada.

3.1 Redimensionamento de Imagens Usando Interpolação

Para os métodos de interpolação, a forma ingênua de re-amostrar consiste em preencher os "espaços vazios" com os valores dos *pixels* espalhados que estão mais próximos. Por isso, essa abordagem é chamada de *método do vizinho mais próximo*. As Figuras 3.2-(a) e 3.2-(b) ilustram, respectivamente, tal abordagem para um bloco de tamanho 4×4, sendo aumentado para outro bloco 8×8. Os espaços "vazios" (em branco) são preenchidos com os valores dos vizinhos mais próximos, resultando na Figura 3.2-(c). Nessa ampliação por um fator dois, tem-se a impressão de que o *pixel* simplesmente quadruplicou de tamanho. Sendo assim, esse método tem como vantagem o fato de não inserir cores diferentes da imagem original, embora apresente uma distorção evidente.

Outra abordagem é utilizar métodos derivados de *interpolação linear*. Esses são métodos numéricos que servem para estimar o valor de um ponto não-amostrado a partir do



(a) (b) (c)

Figura 3.2: Passos do algoritmo de interpolação do vizinho mais próximo. (a) imagem 4×4 a ser ampliada, (b) *pixels* espalhados em uma matriz 8×8 (fator de ampliação 2), e (c) "espaços em branco" preenchidos com valores do *pixel* mais próximo.

valor de outros pontos amostrados, utilizando uma função linear [69]. Contudo, para dados com representação bidimensional, como é o caso de imagens, é necessária uma generalização da interpolação linear, conhecida como interpolação bilinear [34].

No contexto de imagens, a ideia da interpolação bilinear é parecida com a do vizinho mais próximo. Isto é, preencher os "espaços em branco" da imagem aumentada com base nos valores dos *pixels* vizinhos. Contudo, como o próprio nome sugere, ao invés de simplesmente replicar o valor dos *pixels*, utiliza-se uma combinação de interpolações lineares para estimar uma pequena resolução da imagem original. Isso suaviza os efeitos de serrilhado observado na replicação dos vizinhos mais próximos, pois "dissolve" os valores das cores dos *pixels* nos "espaços em branco".

Para isso, utilizam-se os pontos A, B, C e D espalhados na Figura 3.1-(b), tendo como objetivo encontrar a cor do *pixel* r. Primeiramente, estima-se o valor temporário i, interpolando-se os *pixels* A e B, utilizando-se a seguinte equação:

$$r_i = A + \frac{w(B - A)}{W}. \tag{3.1}$$

De forma análoga, estima-se uma segunda variável temporária j, utilizando interpolação linear com os valores dos *pixels* C e D, conforme é mostrado na equação abaixo,

$$r_2 = C + \frac{w(D - C)}{W}. \tag{3.2}$$

Finalmente, combina-se os valores de r_1 e r_2, interpolados na dimensão horizontal, numa nova interpolação linear, que produz o valor interpolado de r, como mostrado na equação

$$r = r_1 + \frac{h(r_1 - r_2)}{H}. \tag{3.3}$$

Além da interpolação bilinear, existem outras abordagens para interpolar pontos em uma imagem re-amostrada. Teoricamente, qualquer método numérico que permita interpolar pontos em uma malha bidimensional pode ser usado para estimar os "espaços em branco" de uma imagem ampliada. Sendo assim, métodos derivados da interpolação de Lagrange e de Hermite [69] (como é a interpolação bilinear e a bicúbica) podem ser adaptados para aumento dimensional em imagens. Métodos como de Newton-Gregory [91], *splines*, método de Lanczos [25] e diversos outros algoritmos de interpolação multivariada também podem ser utilizados para este fim, conforme mostrado por Todd Wittman [92].

Em termos computacionais, os demais métodos de interpolação são mais custosos que o método do vizinho mais próximo, pois introduzem mais operações aritméticas do que simplesmente a cópia dos *pixels* ao lado. Porém, esses métodos produzem resultados com menor quantidade de artefatos de blocagem. Como consequência, eles tendem a produzir um efeito de suavização no lugar dos blocos introduzidos pelo método do vizinho mais próximo, o que também implica em alteração dos níveis de cor com relação à imagem original, onde a Figura 3.3 ilustra esses efeitos. A imagem apresentada na Figura 3.3-(a) é a referência original que, ao ser reduzida em um quarto do tamanho e re-ampliada pelos métodos do vizinho mais próximo, bilinear e bicúbico, produz as imagens apresentadas na

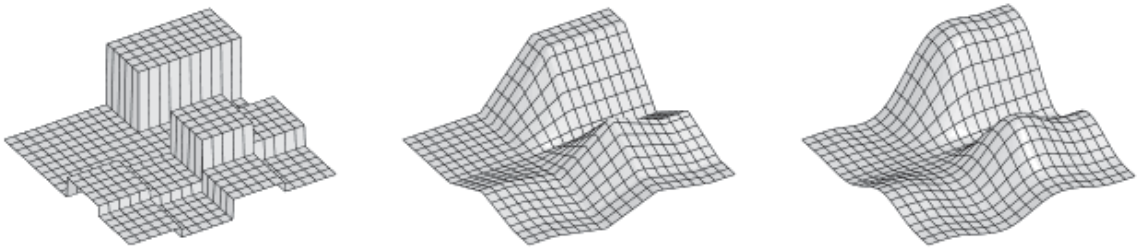
Figura 3.3-(b), (c) e (d). A Figura 3.4 ilustra graficamente o comportamento do resultado a partir desses métodos.

(a) (b)

(c) (d)

Figura 3.3: Efeitos dos métodos de interpolação utilizando um fator de redução e ampliação de 4. (a) Original, (b) Vizinho mais próximo, (c) Bilinear e (d) Bicúbico.

Assim, diferentes tipos de distorções são produzidas pelas diferentes abordagens. Tais distorções são evidenciadas quando observa-se a diferença absoluta entre a imagem de referência e a imagem re-amostrada com um algoritmo de interpolação, conforme ilustrado



(a) (b) (c)

Figura 3.4: Gráfico das curvas de valores das intensidades dos *pixels* utilizando os métodos de interpolação. (a) vizinho mais próximo, (b) bilinear e (c) bicúbico.

na Figura 3.5. Essa imagem apresenta quatro figuras, onde a primeira (a) ilustra o gradiente da imagem original, que realça as regiões de contorno e detalhes dessa imagem. As imagens (b), (c) e (d) mostram a diferença da interpolada com a imagem de referência, ilustrando como o processo de aumento por interpolação pode ser visto como um processo que introduz um erro no gradiente.

Dessa forma, nota-se na Figura 3.5 que quanto mais clara for a região, maior será o erro de estimação da intensidade da cor pelo algoritmo de interpolação. Como é possível notar, as principais informações perdidas pela re-amostragem são informações de contorno e detalhes, que equivalem às informações de alta-frequência.

Além disso, observando o gradiente da imagem (Figura 3.5-(a)), é possível notar que essa operação se assemelha muito com os erros (diferenças) das imagens interpoladas com a original. Por causa disso, algoritmos mais robustos de aumento de resolução de imagem funcionam tentando estimar parâmetros que minimizem essa diferença, relacionando o erro com o gradiente para, dessa forma, obter uma matriz tal que, ao somar com a matriz interpolada, se aproxime com mais precisão da imagem de referência. Por isso, entender o funcionamento dos algoritmos de interpolação é importante no contexto deste trabalho.

Contudo, embora existam diversos métodos de interpolação para se re-amostrar a imagem ampliada, o entendimento dos métodos do vizinho mais próximo e bilinear é suficiente

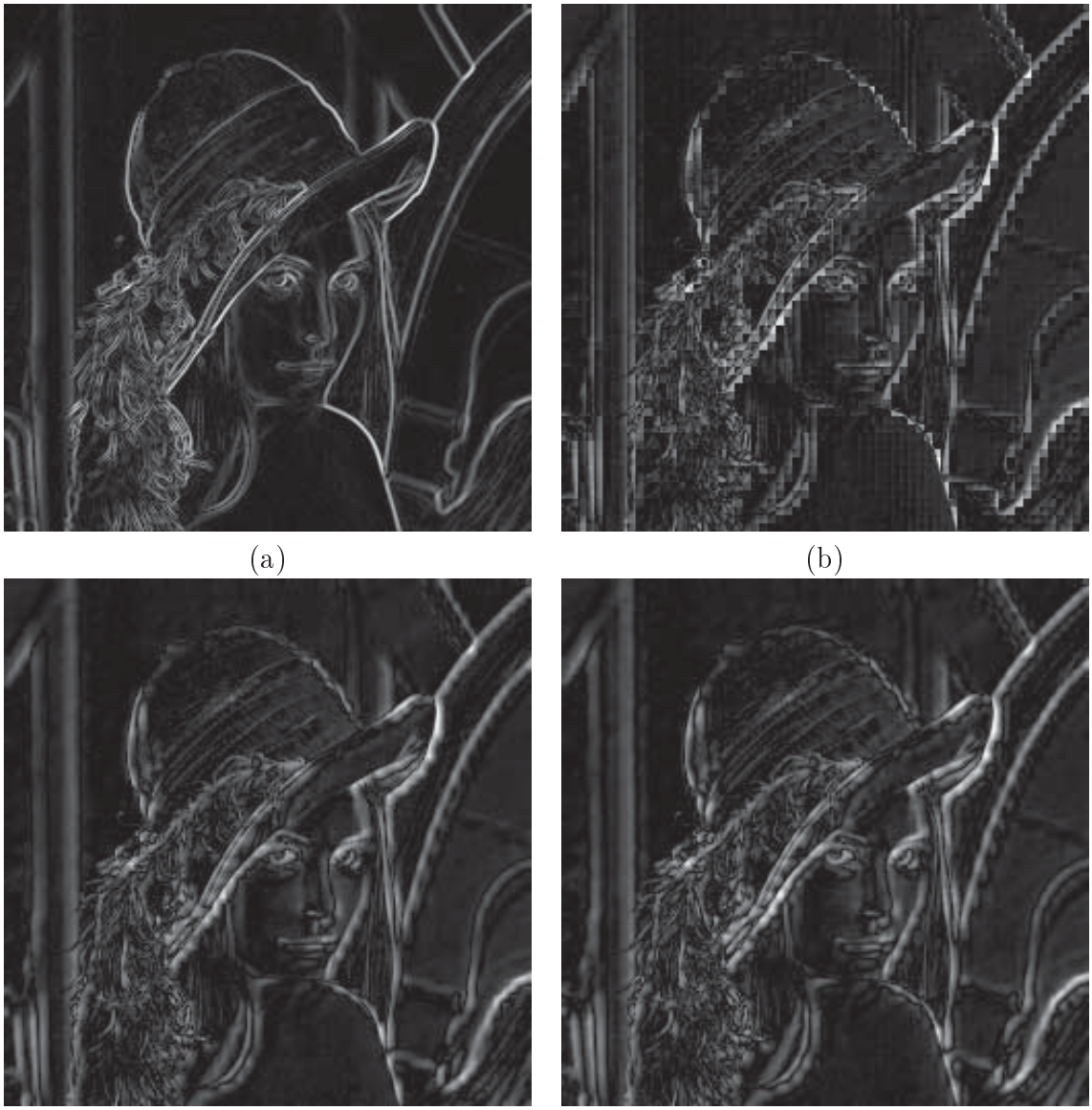


Figura 3.5: Gradiente da imagem original e a diferença entre o resultado por interpolação e a original no tamanho da ampliação. (a) Gradiente gaussiano da imagem, (b) Vizinho mais próximo, (c) Bilinear e (d) Bicúbico.

para o escopo deste trabalho, pois a interpolação bilinear já possui as propriedades de atuar como operador de redução espacial e inserir um ruído gaussiano (borramento) na imagem escalada. Por isso, os estágios dos algoritmos apresentados nas próximas seções que necessitam de interpolação utilizam o método bilinear.

3.2 Super-resolução via Métodos Bayesianos

O primeiro método utilizado foi proposto por Villena *et al.* [84], no artigo *Bayesian Combination of Sparse and Non-sparse Priors in Image Superresolution*. A abordagem desse trabalho parte do princípio de que as imagens de baixa-resolução obtidas de uma câmera provêm informações adicionais que permitem reconstruir uma imagem em alta-resolução. As informações consideradas são relacionadas ao *registro*, onde o movimento entre as imagens de baixa-resolução é estimado para a etapa de *reconstrução*, onde a imagem em alta-resolução pode ser recuperada a partir de um conjunto de imagens de baixa-resolução.

O trabalho de Villena consiste em uma abordagem Bayesiana. Nesse tipo de abordagem, um modelo é fundamental para reconstruir as imagens de alta-resolução . Assim, diversos modelos Bayesianos para resolver o problema da suavização gerada pela interpolação são mencionados, tais como aqueles que utilizam auto-regressão condicional (SAR), baseado em *wavelets* [9], variação total (TV) [8], etc. Todos esses modelos, contudo, possuem a característica de eliminar (parcialmente) ou corromper as componentes de baixa frequência do sinal. Como consequência, pode haver imagens com regiões de bordas suavizadas ou com presença de distorções de alta-frequência (ruído).

Villena assume que o processo de geração da imagem de alta-resolução y_h dependa de L imagens de baixa-resolução y_l^k, tal que $k = 1, \ldots, L$. Todas as imagens y_l^k possuem a mesma dimensão $N = h \times w$ *pixels*. A imagem de alta-resolução y_h possui PN *pixels*, onde $P > 1$ é o fator de aumento. Assim, as imagens y_l^k e x são transformadas para uma representação vetorial unidimensional de tamanho $N \times 1$ e $PN \times 1$, respectivamente. Além disso, um conjunto de vetores de movimento s_k modelam a translação e rotação da imagem y_h. Assim, o processo introduz efeitos de deslocamento, de borramento e de

sub-amostragem, modelados utilizando a seguinte equação:

$$y_l^k = AH_kC(s_k)y_h + n_k = B_k(s_k)y_h + n_k, \tag{3.4}$$

onde $B_k = AH_kC(s_k)$ é um sistema matricial de tamanho $N \times PN$, A é o operador de sub-amostragem de tamanho $N \times PN$, H_k é o operador de borramento de tamanho $PN \times PN$, $C(s_k)$ é o operador de distorção de tamanho $PN \times PN$, gerado pelo conjunto de vetores de movimento s_k, e n_k é um ruído de amostragem de tamanho $N \times 1$.

Os vetores de movimento $s_k = (\omega_k, c_k, d_k)$ consistem em movimentos translacionais e rotacionais, onde ω_k é o ângulo de rotação, c_k é o deslocamento horizontal e d_k o deslocamento vertical. Além disso, as matrizes de borramento H_k são assumidas como sendo conhecidas, podendo ser estimadas pela aplicação de um filtro passa-baixa. Assim, os efeitos de sub-amostragem, borramento e ruído são sintetizados na matriz $B_k(s_k)$, ba qual cada *pixel* da imagem de alta-resolução de y_h é mapeado para um *pixel* na imagem de baixa-resolução, y_l^k. Assim, o problema consiste em estimar a imagem de y_h (alta-resolução) a partir de um conjunto de imagens $\{y_l^k\}$ usando as informações conhecidas de $\{C(s_k)\}$ e $\{n_k\}$.

No caso dos vetores de movimento, como pressupõe-se que eles são gerados de movimentos em y_h, Villena propõe um modelo em dois estágios. O primeiro deles consiste em modelar o processo de aquisição, a imagem desconhecida y_h e os vetores $\{s_k\}$. Assim, ele supõe que para determinar y_h existem m modelos que possam ser combinados. Esses modelos são denotados por $p_i(y_h|\alpha_i)$, onde $i = 1, 2, \ldots, m$. O vetor $y_l = \{y_l^k\}$ também é um processo aleatório, correspondente à distribuição condicional $p(y_l|y_h, \{s_k\}, \{\beta_k\})$. Dessa maneira, essas distribuições dependem de dois novos parâmetros adicionais, α_i e $\{\beta_k\}$, que são modelados no segundo estágio do modelo.

Assumindo-se que n_k é um ruído-branco gaussiano de média zero e variância β_k, as probabilidades condicionais das imagens y_l^k são dadas por

$$p(y_l^k|y_h, \{s_k\}, \{\beta_k\}) = \beta_k^{N/2} exp\left(\frac{-\beta_k}{2}\|y_l^k - B_k(s_k)y_h\|^2\right). \tag{3.5}$$

Considerando-se também que há independência estatística entre as imagens de baixa-resolução, o que faz com que a probabilidade condicional do conjunto de imagens de baixa-resolução y_l possa ser expresso como

$$p(y_l|y_h, \{s_k\}, \{\beta_k\}) = \prod_{k=1}^{L} P(y_l^k|y_h, s_k, \beta_k). \tag{3.6}$$

Para calcular $C(s_k)x$, é denotada como (u_k, v_k) a coordenada dos *pixels* da imagem de alta-resolução distorcida. Nesse ponto, a distorção é obtida reduzindo-se a imagem de referência e redimensionando-se novamente, utilizando interpolação bilinear. Sendo (u, v) a coordenada de referência da imagem de alta-resolução, define-se Δu_k e Δv_k como sendo as coordenadas da k-ésima variação da distorção em relação à y_h,

$$\Delta u_k = u_k - u = ucos(\theta_k) - vsin(\theta_k) + c_k - u \tag{3.7}$$

e

$$\Delta v_k = v_k - u = usin(\theta_k) + vcos(\theta_k) + d_k - v. \tag{3.8}$$

Como a imagem em alta-resolução é indeterminada, a interpolação bilinear é utilizada para aproximá-la. Após essa aproximação, rotaciona-se para gerar as distorções.

Em seguida, denotando-se $[a_k(s_k), b_k(s_k)]^T$ como o vetor de diferença entre os *pixels* na posição (u_k, v_k) e o *pixel* na imagem de alta-resolução original. Esses vetores são calculados

pelas seguintes equações,

$$a_k(s_k) = \Delta u_k - \lfloor \Delta u_k \rfloor \tag{3.9}$$

e

$$b_k(s_k) = \Delta v_k - \lceil \Delta v_k \rceil. \tag{3.10}$$

Usando a interpolação bilinear, $C(s_k)x$ pode ser aproximado como

$$C(s_k)x \approx D_{b_k}\left(I - D_{a_k}\right) L_{bl} + \left(I - D_{b_k}\right) D_{a_k} L_{tr} + \left(I - D_{b_k}\right)\left(I - D_{a_k}\right) L_{tl} x + D_{b_k} D_{a_k} L_{br}, \tag{3.11}$$

onde D_{a_k} e D_{b_k} denotam as matrizes diagonais dos vetores $a_k(s_k)$ e $b_k(s_k)$. As matrizes L_{bl}, L_{tr}, L_{tl} e L_{br} correspondem aos operadores que geram os *pixels* ao redor da posição (u_k, v_k).

A qualidade da imagem de alta-resolução estimada, assim como a precisão da estimação dos outros parâmetros, depende da modelagem da distribuição. No trabalho de Villena [84], dois modelos são usados. O primeiro consiste em minimizar a variação total, enquanto que o segundo é uma minimização da diferença em norma ℓ_1. Para ambos, inicialmente, é definido um modelo auto-regressivo inicial, que é dado como

$$p_1(y_h|\alpha_1) = \alpha_1^{\frac{PN}{2}} exp\left(-\frac{\alpha_1}{2}\|\zeta y_h\|^2\right), \tag{3.12}$$

onde ζ é o operador Laplaciano e α_1 é um parâmetro a ser determinado na segunda etapa do modelo.

O modelo que utiliza a norma ℓ_1 (SARL1), é definido como

$$p_2(y_h|\alpha_2) = (\alpha_2^h \alpha_2^v)^{\frac{P\cdot N}{4}} exp\left[-\sum_{i=1}^{P\cdot N}\left(\alpha_2^h\|\Delta_i^h(y_h)\|_{\ell_1} + \alpha_2^v\|\Delta_i^v(y_h)\|_{\ell_1}\right)\right], \tag{3.13}$$

onde $\Delta_i^h(y_h)$ e $\Delta_i^v(y_h)$ representam a diferença de primeira ordem horizontal e vertical para

o *pixel* i, e $\alpha_2 = \{\alpha_2^h, \alpha_2^v\}$ é o parâmetro a ser determinado na segunda etapa do modelo hierárquico.

O modelo que utiliza a variação total (SARTV) é definido como

$$p_3(y_h|\alpha_3) = \alpha_3^{\frac{P \cdot N}{2}} exp\left[-\frac{\alpha_3}{2}\sum_{i=1}^{P \cdot N}\sqrt{F(y_h)}\right], \tag{3.14}$$

onde,

$$F(y_h) = (\Delta_i^h(y_h))^2 + (\Delta_i^v(y_h))^2, \tag{3.15}$$

e α_3 é o parâmetro a ser determinado na segunda etapa deste modelo. Ambos modelos TV e ℓ_1 são esparsos a priori, e muito efetivos em manter os contornos [17, 86]. A principal diferença entre eles é que no modelo ℓ_1, a segunda etapa consiste em estimar dois parâmetros, α_2^h e α_2^v, ao invés de apenas um.

Denota-se por s_k^{-p} a estimativa do vetor s_k, obtida das imagens de baixa-resolução, geradas a partir de um processo de degradação que consiste em rotacionar, borrar e reduzir as dimensões da imagem original. Assim, os parâmetros de movimento são modelados como sendo variáveis aleatórias seguindo uma distribuição Gaussiana, conforme a seguinte equação:

$$p(s_k) = \mathcal{N}(s_k|s_k^{-p}, \Lambda_k^p), \tag{3.16}$$

onde Λ_k^p é uma matriz de covariância a priori. Assim, os parâmetros s_k^{-p} e Λ_k^p incorporam informação dos parâmetros sobre os vetores de movimento no processo de estimação. Caso essas variáveis sejam desconhecidas, s_k^{-p} e $(\Lambda_k^p)^{-1}$ podem ser tomadas como zero.

3.2.1 Estimativa dos parâmetros

Os parâmetros α_1, α_2, α_3 e $\{\beta_k\}$ são modelados por meio da distribuição Gama, conforme mostrado na Equação 3.17 a seguir,

$$p(\omega) = \Gamma(\omega|a_\omega, b_\omega) = \frac{b_\omega^{a_\omega}}{\Gamma(a_\omega)}\omega^{a_\omega - 1} exp\left(-b_\omega \omega\right), \tag{3.17}$$

onde ω denota o parâmetro, e a_ω e b_ω são parâmetros de escala ajustáveis.

Assim, combinando-se as Equações 3.6, 3.12, 3.13, 3.14 e 3.17, obtem-se

$$p_l(\Omega_l, y) = p_l(y_h|\alpha_l)p(\alpha_l)\prod_{k=1}^{L} p(y_l^k|y_h, s_k, \beta_k)p(\beta_k)p(s_k), \tag{3.18}$$

onde l denota a distribuição selecionada (1 para SAR, 2 para ℓ_1 e 3 para TV) e $\Omega_l = \{y_h, s_k, \{\beta_k\}, \alpha_l\}$.

No trabalho de Villena *et al.* [86], a inferência Bayesiana usada é baseada na distribuição de $p(\Phi|y_l)$, onde denota-se $\Phi = \{\Omega, \{\alpha_l\}\}$. A ideia é aproximar essa distribuição, encontrando a distribuição da imagem de alta-resolução que minimiza a combinação convexa das divergências de Kullback-Leiber (minimização da divergência de informação) [85], modelada pela Equação 3.19 a seguir,

$$\hat{q}(\Phi) = \arg\min_{q(\Phi)} \sum_{l=1}^{m} \lambda_l C_{KL}\left(q(\Omega)q(\alpha_l)p_l(\Theta_l)|y\right), \tag{3.19}$$

onde m é o número de imagens de baixa-resolução combinadas no modelo, $\lambda_l \geq 0$ para $l = 1, 2, \ldots, m$ e $\sum_{l=1}^{m} \lambda_l = 1$. Além disso, $p(\Theta_l|y)$ é dada pela Equação 3.20,

$$p(\Theta_l|y_l) = \frac{p(\Theta_l, y_l)}{p(y_l)}, \tag{3.20}$$

que equivale à distribuição definida na Equação 3.18 para o respectivo modelo; $q(\Phi) =$

$q(\Omega)\prod_{l=1}^{m} q(\alpha_1)$, $q(\Omega) = q(y_h)\prod_{k=1}^{L} q(\beta_k)$ e as divergências de Kullback-Leiber são definidas pela Equação 3.21,

$$C_{KL}\left(q(\Omega)q(\alpha_l)p_l(\Theta_l)|y)l\right) = \int q(\Omega)q(\alpha_l)log\left(\frac{q(\Omega)q(\alpha_l)}{p_l(\Theta_l}\right)d\Omega d\alpha_l \tag{3.21}$$

Dessa forma, após algumas manipulações algébricas, Villena *et al.* obtém que a distribuição da imagem em alta-resolução é dada por

$$q_{\ell_1}(y_h) = exp\left(-\frac{1}{2}\left[\lambda_2 g_{\ell_1} + (1-\lambda_2)\langle\alpha_1\rangle\|\zeta y_h\|^2 + \sum_k \langle\beta_k\rangle\langle\|y_l^k - B_k(s_k)y_h\|^2\rangle\right]\right), \tag{3.22}$$

onde

$$g_{\ell_1} = \langle\alpha_2^h\rangle\sum_i \frac{(\Delta_i^h(y_h))^2 + w_{2i}^h}{\sqrt{w_{2i}^h}} + \langle\alpha_2^v\rangle\sum_i \frac{(\Delta_i^v(y_h))^2 + w_{2i}^v}{\sqrt{w_{2i}^v}}, \tag{3.23}$$

e

$$q_{TV}(y_h) = exp\left(-\frac{1}{2}\left[\lambda_3 g_{TV} + (1-\lambda_3)\langle\lambda_1\rangle\|\zeta y_h\|^2 + \sum_k \langle\beta_k\rangle\langle\|y_l^k - B_k(s_k)y_h\|^2\rangle\right]\right), \tag{3.24}$$

onde

$$g_{TV} = \lambda_3\langle\alpha_3\rangle\sum_i \frac{(\Delta_i^h(y_h))^2 + (\Delta_i^v(y_h))^2 + w_{3i}}{\sqrt{w_{3i}}}. \tag{3.25}$$

Nas equações acima, Δ^h e Δ^v são as matrizes de convolução associadas às diferenças de primeira ordem horizontal e vertical. Os parâmetros w são obtidos conforme as seguintes equações:

$$w_{2i}^h = E_{\ell_1}(y_h)(\Delta_i^h(y_h))^2, \tag{3.26}$$

$$w_{2i}^v = E_{\ell_1}(y_h)(\Delta_i^v(y_h))^2, \tag{3.27}$$

e

$$w_{3i} = E_{TV}(y_h)\left[(\Delta_i^h(y_h))^2 + (\Delta_i^v(y_h))^2\right]. \tag{3.28}$$

Nessas equações, $E_{\ell_1}(y_h)$ e $E_{TV}(y_h)$, são as esperanças das distribuições descritas nas Equações 3.22 e 3.24.

Finalmente, os parâmetros α_i, α_2 e $\{\beta_k\}$ são gerados pelas distribuições a seguir. Para $\{\beta_k\}$, tem se que

$$q(\beta_k) = \beta_k^{\frac{N}{2}-1+a_{\beta_k}} exp\left[-\beta_k\left(b_{\beta_k} + \frac{E_l}{2}\|y_l^k - B_k(s_k)y_h\|^2\right)\right], \tag{3.29}$$

onde $l = \{\ell_1, TV\}$. O parâmetro α_1, necessário para a Equação 3.12 é obtido conforme a distribuição

$$q(\alpha_1) = \alpha_1^{\frac{PN}{2}-1+a_{\alpha_1}} exp\left[-\alpha_1\left(b_{\alpha_1} + \frac{E_l\|\zeta y_h\|^2}{2}\right)\right]. \tag{3.30}$$

Para a Equação 3.13, o parâmetro α_2 é obtido pelas distribuições

$$q(\alpha_2^h) = (\alpha_2^h)^{\frac{PN}{4}-1+a_{\alpha_2}} exp\left[-\alpha_2^h\left(b_{\alpha_2} + \sum_{i=1}^{PN}\sqrt{w_{2i}^h}\right)\right] \tag{3.31}$$

e

$$q(\alpha_2^v) = (\alpha_2^v)^{\frac{PN}{4}-1+a_{\alpha_2}} exp\left[-\alpha_2^v\left(b_{\alpha_2} + \sum_{i=1}^{PN}\sqrt{w_{2i}^v}\right)\right]. \tag{3.32}$$

Por fim, o parâmetro α_3, relativo à Equação 3.14, obedece a seguinte distribuição,

$$q(\alpha_3) = (\alpha_3)^{\frac{PN}{2}-1+a_{\alpha_3}} exp\left[-\alpha_3\left(b_{\alpha_3} + \sum_{i=1}^{PN}\sqrt{w_{3i}}\right)\right]. \tag{3.33}$$

Os passos de estimação do algoritmo de Villena *et al.* que utilizam ℓ_1 (SARL1) são descritos no Algoritmo 1. Os parâmetros iniciais para esse método são inicializados como $w_{2i}^h = (\Delta_i^h(y_h^0))^2$, $w_{2i}^v = (\Delta_i^v(y_h^0))^2$, $\alpha_2^h = \frac{PN}{2(\sum_i^{PN}\sqrt{w_{2i}^h})}$, $\alpha_2^v = \frac{PN}{2(\sum_i^{PN}\sqrt{w_{2i}^v})}$,

$\lambda_2 = 0.5$ e $\beta_k = \frac{N}{\|y_k - B_k(s_k)x^0\|^2}$. De forma análoga, a abordagem que utiliza variação total (SARTV) é descrita no Algoritmo 2. Nesse caso, os parâmetros são inicializados como $w_{3i} = (\Delta_i^h(x^0))^2 + (\Delta_i^v(x^0))^2$, $\alpha_3 = \frac{PN}{2\sum_i^{PN}\sqrt{w_{3i}}}$, e $\lambda_3 = 0.5$.

Algoritmo 1: Algoritmo iterativo para aumento de resolução utilizando ℓ_1.

Data: Imagem de baixa-resolução a ser ampliada y_1, e os parâmetros λ_1, λ_2 e P

Result: Imagem ampliada y_h

1 Gerar o conjunto de imagens de baixa-resolução $y_l = \{y_l^1, y_l^2, \cdots, y_l^m\}$, onde y_l^1 é a entrada original e y_l^j (para $j = 2..m$) são formados por rotações e translações de y_l^1.

2 Aumentar y_l^1 por um fator de aumento P, utilizando interpolação bilinear, e atribuir o resultado dessa operação à estimativa inicial da imagem de alta-resolução y_h^0.

3 **while** *o critério de convergência não for atingido* **do**

4 Estimar a distribuição da imagem usando a Equação 3.22.

5 Computar w_2^h e w_2^v, conforme as Equações 3.26 e 3.27.

6 Estimar os parâmetros $\{\beta_k\}$, α_1 e α_2, seguindo as Equações 3.29, 3.30, 3.31 e 3.32.

7 Atualizar a estimação de y_h^i utilizando os novos parâmetros calculados.

Algoritmo 2: Algoritmo iterativo para aumento de resolução utilizando TV.

Data: Imagem de baixa-resolução a ser ampliada y_l^1, e os parâmetros λ_1, λ_2 e P

Result: Imagem ampliada y_h

1 Gerar o conjunto de imagens de baixa-resolução $y_l = \{y_l^1, y_l^2, \cdots, y_l^m\}$, onde y_l^1 é a entrada original e y_l^j (para $j = 2..m$) são formados por rotações e translações de y_l^1.

2 Aumentar y_l^1 por um fator de aumento P, utilizando interpolação bilinear, e atribuir o resultado dessa operação à estimativa inicial da imagem de alta-resolução y_h^0.

3 **while** *o critério de convergência não for atingido* **do**

4 Estimar a distribuição da imagem usando a Equação 3.24.

5 Computar w_3^h, conforme a Equação 3.28.

6 Estimar os parâmetros $\{\beta_k\}$, α_1 e α_3, seguindo as Equações 3.29, 3.30 e 3.33.

7 Atualizar a estimação de y_h^i utilizando os novos parâmetros calculados.

3.3 Super-resolução via Codificação Esparsa (SRvSR)

O terceiro algoritmo usado para os testes é uma modificação do método proposto por Yang *et al.* [93]. A ideia é que para transformar uma imagem de baixa resolução z_l, em outra imagem de alta resolução y_h, deve haver um operador de verificação, o qual transforma a imagem em alta-resolução em outra de baixa resolução, chamado de *operador de sub-amostragem* S [70]. Além disso, supõe-se haver um operador de degradação H, responsável por transformar a imagem de alta-resolução y_h na imagem de baixa resolução z_l por meio da solução do sistema abaixo:

$$z_l = \mathbf{SH}y_h + v, \tag{3.34}$$

onde v é um ruído. Dessa maneira, a solução consiste em encontrar $\mathbf{S}$ e $\mathbf{H}$ tais que para a entrada z_l, y_h possa ser determinado. O problema é que para uma dada imagem de baixa-resolução z_l, existem diferentes imagens de alta-resolução y_h que podem satisfazer a restrição de reconstrução. A Figura 3.6 ilustra o resultado das operações $\mathbf{S}$ e $\mathbf{H}$.

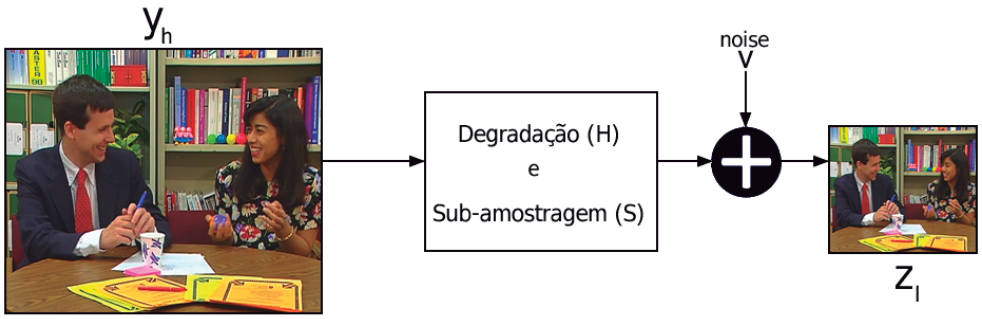


Figura 3.6: Resultado dos operadores de redução da resolução.

Como visto no início deste capítulo, sabe-se que o processo de reduzir as dimensões da imagem de alta-resolução y_h, e depois aumentar por técnicas de interpolação gera uma imagem degradada. Sendo assim, define-se o operador de interpolação $\mathbf{Q}$ que realize operações semelhantes à aplicação $\mathbf{S}$ e $\mathbf{H}$, gerando uma aproximação de y_h, denotada por y_l, tal que

$$y_l = \mathbf{Q} z_l, \tag{3.35}$$

conforme ilustrado na Figura 3.7. Para evidenciar a diferença entre y_l e y_h, a Figura 3.8 exibe essas imagens lado-a-lado.

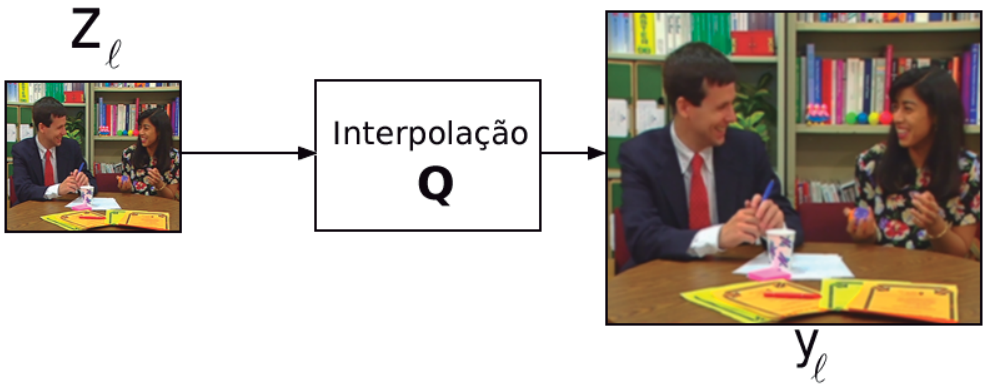


Figura 3.7: Resultado do operador de interpolação ($\mathbf{Q}$).

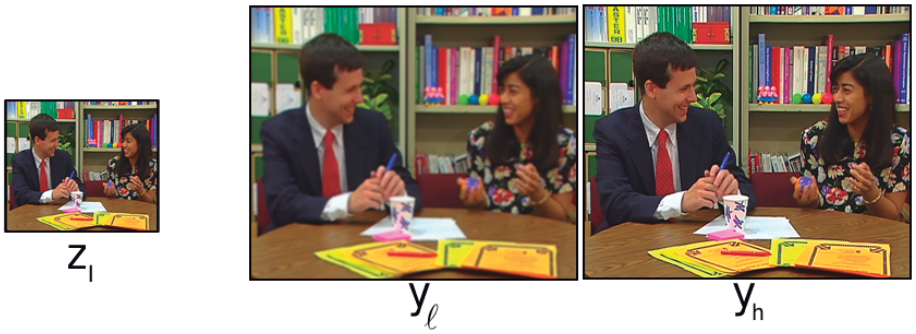


Figura 3.8: Resultados das operações.

Substituindo z_l da Equação 3.34 na Equação 3.35, a seguinte expressão:

$$\mathbf{Q}^{-1}y_l = \mathbf{SH}y_h + v. \tag{3.36}$$

Portanto, y_h e y_l se relacionam segundo a equação abaixo,

$$y_l = \mathbf{QSH}y_h + \mathbf{Q}v = \mathbf{QSH}y_h + \hat{v}, \tag{3.37}$$

que pode ser reescrita como

$$y_l = \mathbf{L}y_h + \hat{v}, \tag{3.38}$$

onde $\mathbf{L}$ é um operador que equivale ao processo de borramento, sub-amostragem e interpolação.

Como y_l é disponível a partir de z_l, basta $\mathbf{L}$ para obter y_h. Porém, esse problema é semelhante àquele de encontrar $\mathbf{S}$ e $\mathbf{H}$, apresentado para a Equação 3.34. Contudo, diferente daquele, agora y_h tem relação com z_l e y_l, o que reduz as possibilidades de solução. Assim, tem-se que o problema de encontrar a imagem de alta-resolução y_h é um problema de redução do ruído $\hat{v}$.

Conforme Candes e Tao [14], o problema de encontrar y_h poderia ser tratado por uma combinação de problemas de programação linear (PPL), resolvendo-se a seguinte minimização na norma ℓ_1:

$$\min_{g \in R^N} \|y_l - Lg\|_{\ell_1}, \tag{3.39}$$

onde g é uma decomposição tal que $g = y_h + s$, e

$$\min_{s \in R^n} \|\hat{v} - Ls\|_{\ell_1}. \tag{3.40}$$

Assim, com essa combinação de PPLs, é possível determinar y_h. Contudo, $\mathbf{L}$ é indeterminado, o que faz com que seja necessário determinar este operador.

Para evitar esse problema, um conjunto de imagens de alta-resolução é utilizada no treinamento da busca das distribuições que mais aproximam y_l e y_h. A ideia consiste em utilizar essa coleção de treinamento para determinar como uma imagem em alta-resolução se degrada com relação à outra interpolada.

Essa é uma abordagem comum em métodos de super-resolução [87, 93] e consiste em dividir as imagem de alta-resolução e as interpoladas em pequenos blocos, chamados de *retalhos*. Esses retalhos servem para criar um conjunto de dados que correlacionam uma pequena base da imagem de alta-resolução com a de baixa-resolução. O objetivo disso é criar um conjunto de dados com as imagens de treinamento, de forma que os elementos

desse conjunto possam atuar como o operador L nos retalhos de uma imagem qualquer a ser ampliada.

O problema é como relacionar os retalhos da imagem y_h e y_l, conforme mostra a Figura 3.9. Nessa figura, p_k^l representa o k-ésimo retalho da imagem interpolada (y_l), enquanto p_k^h é o retalho da imagem em alta-resolução disponível para o treinamento (y_h). Uma abordagem ingênua para relacionar esses dois retalhos pode consistir em relacioná-los por meio de um critério de qualidade, tal como o PSNR. Dessa forma, após calcular o PSNR entre p_k^l e p_k^h, salva-se em um banco de dados todo o conteúdo de p_k^h indexado pelo valor de PSNR. Dessa maneira, quando a imagem y_h não estiver disponível, basta utilizar a imagem y_l como referência, dividi-la em retalhos e buscar no banco os retalhos de alta-resolução.

Contudo, essa abordagem ingênua possui três grandes problemas. O primeiro consiste no critério de otimização escolhido. Escolher o PSNR máximo nessa abordagem é ruim porque diferentes retalhos de baixa-resolução podem ter o mesmo índice de qualidade com outro retalho de alta-resolução que não corresponda com o dado original. Assim, alguns retalhos podem ser substituídos erroneamente. O segundo critério consiste na informação armazenada. Como é armazenado um bloco para cada retalho, o banco terá que armazenar, em forma de retalhos, toda a imagem de alta-resolução. O último problema consiste no *sobreajuste* dos dados. Isto é, o banco ficará ajustado para o conjunto de imagens utilizadas no treinamento, mas quando uma imagem fora desse banco for utilizada, o bloco que maximiza o PSNR pode conter uma informação pouco coerente com a imagem de baixa-resolução.

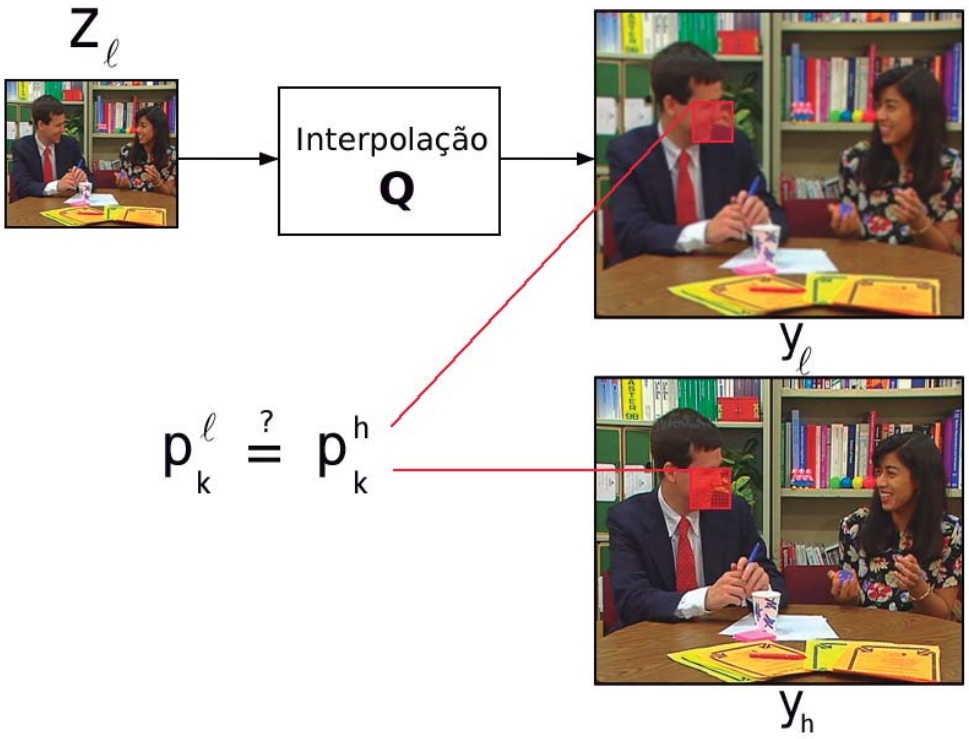


Figura 3.9: Resultados das operações.

Tendo em vista esses problemas, ao invés dos retalhos serem armazenados como *pixels*, eles são guardados em forma de dados que permitem aproximar a imagem de alta-resolução a partir de outro de baixa-resolução. Para isso, dois conjuntos de vetores normalizados, Ψ_h e Ψ_l, são criados. Esses conjuntos são chamados de *dicionários de bases* e atuam como operadores lineares.

Neste trabalho, o dicionário Ψ_h opera reconstruindo um sinal a partir vetor esparso. Dessa forma, sendo p_k^h o k-ésimo retalho obtido do sinal de alta-resolução e q_k um vetor esparso, então esse retalho pode ser reconstruído conforme a Equação 3.41,

$$p_k^h = \Psi_h q_k, \tag{3.41}$$

onde Ψ_h é uma matriz com dimensões $n \times m$, $\|q_k\|_{\ell_0} \ll n$ e p_k^h tem tamanho $n \times 1$.

A Figura 3.10 ilustra essa codificação para o k-ésimo retalho. Nessa figura, o bloco de

tamanho marcado $s \times s$ equivale ao vetor p_k^h de tamanho $s^2 \times 1$, onde $s = \sqrt{n}$. Portanto, a partir da base esparsa q_k, basta que Ψ_h esteja armazenado para que p_k^h possa ser recuperado.

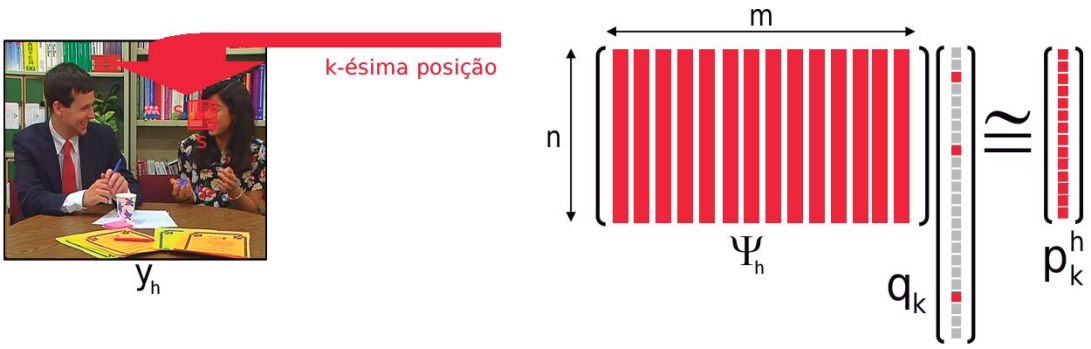


Figura 3.10: Obtenção do retalho p_k^h a partir do vetor esparso q_k.

Para determinar o vetor esparso q_k, o problema descrito pela Equação 3.37 é reescrito em termos dos retalhos p_k^h e p_k^l, relativos aos dados das imagens y_h e y_l, transformando-se no problema de minimizar a seguinte diferença:

$$\|\mathbf{L}p_k^h - p_k^l\|_2 < \epsilon. \tag{3.42}$$

Contudo, substituindo a Equação 3.41 na Equação 3.42, tem-se que

$$\|\mathbf{L}\Psi_h q_k - p_k^l\|_2 < \epsilon, \tag{3.43}$$

onde $\Psi_l = \mathbf{L}\Psi_h$, o que permite que o problema finalmente seja reescrito como

$$\begin{aligned} \min \quad & \|q_k\|_{\ell_1} \\ \text{s. a.} \quad & \|\Psi_l q_k - p_k^l\|_2 \end{aligned}. \tag{3.44}$$

Assim, a partir do retalho de baixa-resolução p_k^l e do dicionário Ψ_l, é possível determinar a base q_k. Dessa forma, resta apenas determinar os dicionários Ψ_h e Ψ_l para resolução geral do problema.

3.3.1 Geração dos Dicionários

O primeiro passo para a construção dos dicionários consiste em coletar os dados em um banco de imagens de alta-resolução e, a partir delas, gerar outro conjunto de imagens de baixa-resolução. Para isso, cada imagem desse banco é sub-amostrada, reduzindo-se suas dimensões. Em seguida, as imagens reduzidas são redimensionadas para o tamanho original. Dessa forma, é construído um segundo conjunto de imagens interpoladas. Assim, um conjunto de pares de retalhos $\{(pp_k^l, pp_k^h)\}$ é gerado, correspondendo aos *pixels* coletados da imagem interpolada e de alta-resolução, respectivamente. A Figura 3.11 ilustra esse passo.

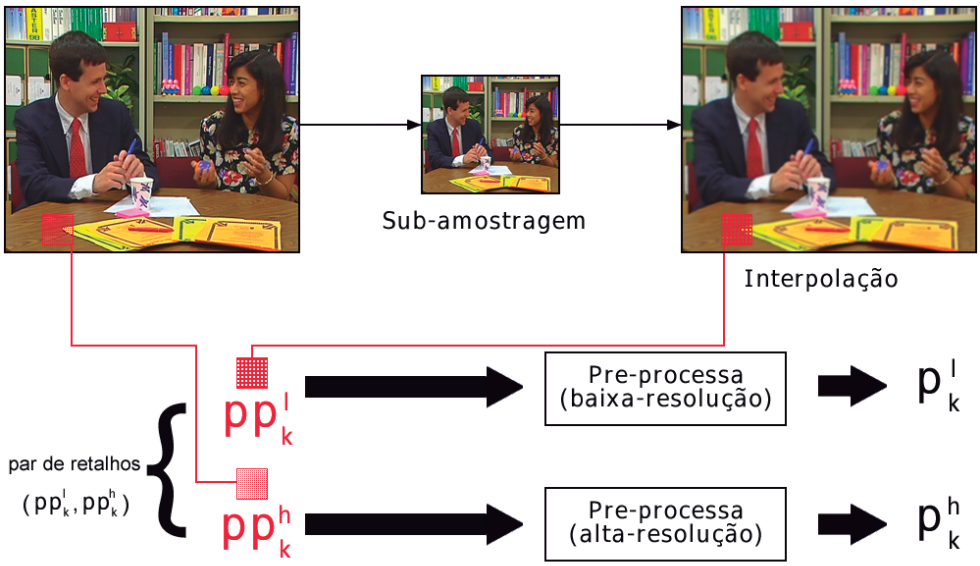


Figura 3.11: Resultados das operações.

Para gerar o dicionário Ψ_l, primeiramente gera-se os retalhos das bases de baixa-resolução, p_k^l, a partir dos pares de retalhos do gradiente de y_l, conforme mostrado na Equação 3.45,

$$p_k^l = \bigtriangledown pp_k^l. \tag{3.45}$$

A Figura 3.12 ilustra esse processo. Em seguida, p_k^l é redimensionado das dimensões $\sqrt{n} \times \sqrt{n}$ para as dimensões $n \times 1$, tornando-se um vetor que é inserido em Ψ_l. Assim, a k-ésima coluna de Ψ_l é formada pelo vetor p_k^l. Dessa forma, após construído Ψ_l e de posse dos retalhos p_k^l, utiliza-se a Equação 3.44 para obter os vetores q_k.

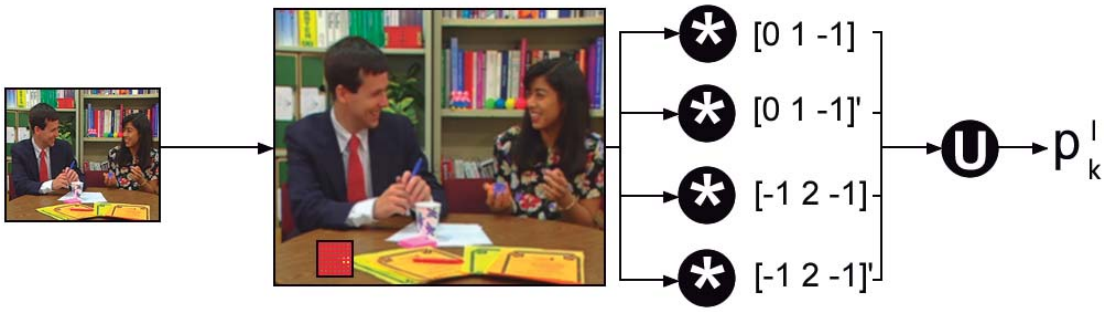


Figura 3.12: Geração de um retalho p_k^l.

Após obter os vetores q_k, os retalhos de alta-resolução p_k^h podem ser obtidos a partir da Equação 3.41. O problema é que até esse ponto Ψ_h ainda não foi determinado. Para determinar esse dicionário de alta-resolução, utiliza-se um conjunto de retalhos, $\hat{p_k^h}$, gerados a partir dos dados das imagens de treinamento. Para isso subtrai-se as imagens de alta-resolução das imagens de baixa-resolução, geradas no treinamento, e extrai-se os retalhos $\hat{p_k^h}$ desse resultado. A Figura 3.13 ilustra esse processo, que é descrito pela Equação 3.46,

$$\hat{p_k^h} = pp_k^h - pp_k^l. \tag{3.46}$$

Dessa maneira, o dicionário Ψ_h pode ser obtido minimizando-se $\|\hat{p_k^h} - \Psi_h q_k\|^2$, utilizando o método proposto por Lee *et al.* [53]. Assim, após a determinação de Ψ_h e Ψ_l, uma imagem qualquer pode ser ampliada seguindo os passos do Algoritmo 3.

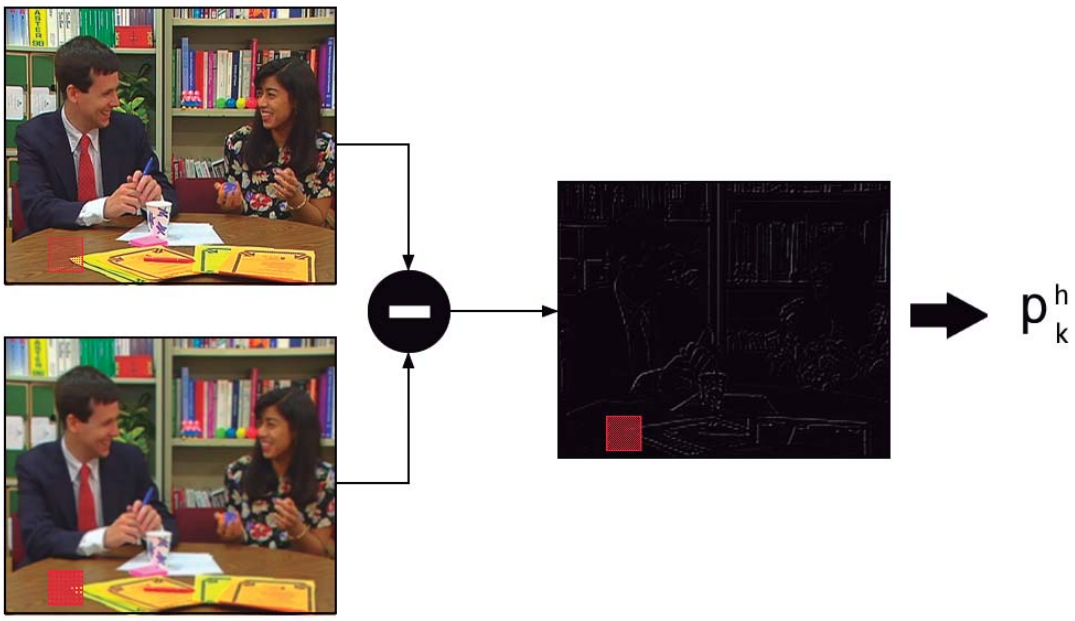


Figura 3.13: Geração de um retalho $\hat{p_k^h}$.

Algoritmo 3: Algoritmo iterativo para aumento de resolução utilizando ℓ_1.

Data: Imagem a ser ampliada z_l e fator de ampliação P

Result: Imagem ampliada y_h

1 Interpolar a entrada z_l e salvar o resultado em y_l.

2 Dividir a imagem em retalhos.

3 **foreach** *retalho* p_k^l **do**

4 Calcular q_k usando a Equação 3.44.

5 Calcular p_k^h utilizando a Equação 3.41.

6 Unir todos os retalhos p_k^h na imagem y_h.

3.4 Considerações Finais

Este capítulo apresentou três técnicas distintas para o aumento de resolução de imagens. A primeira consiste nas abordagens mais simples entre elas, que são as técnicas de interpolação. Conforme descrito, essas técnicas têm a vantagem de serem computacional-

mente simples, com pouco consumo de tempo e de memória. Por outro lado, a qualidade da informação produzida por essas técnicas apresenta distorções e artefatos que são notáveis ao olho humano. Sendo assim, existem técnicas que buscam reduzir essas distorções através de abordagens que minimizam os efeitos de sub-amostragem, borramento e ruído.

Essas técnicas são chamadas de super-resolução. Como exposto, diferentes abordagens podem ser utilizadas para minimizar a diferença entre uma imagem de baixa-resolução e sua equivalente amostrada em alta-resolução. Assim, primeiramente foi discutida uma técnica de super-resolução que consiste em encontrar as distribuições dos *pixels* da imagem ampliada seguindo um modelo Bayesiano. Dessa forma, na Seção 3.2 foi visto que distribuições distintas podem ser utilizadas para solucionar o mesmo problema (recuperar a imagem de alta-resolução a partir de imagens de baixa-resolução). Finalmente, na Seção 3.3, foi apresentada uma abordagem que utiliza codificação esparsa para relacionar a informação da imagem de baixa-resolução com a de alta-resolução. Para isso, dois conjuntos de dados foram criados para relacionar as informações da imagem de baixa-resolução com a de alta-resolução.

Tendo em vista as diferentes abordagens possíveis para a resolução do problema da super-resolução, é interessante haver um método que abstraia os detalhes dela, permitindo que os algoritmos de aumento de resolução possam ser implementados em arquiteturas paralelas sem que haja necessidade de modificação do algoritmo de super-resolução. Sendo assim, no capítulo seguinte será exposto um *framework* que permite a operação do aumento de resolução ser executada de forma paralela, independendo do algoritmo de super-resolução adotado.

Capítulo 4

Framework Paralelo e Simplificado para Aumento de Resolução em Vídeos

4.1 Simplificação

A carga de dados dos sinais de vídeo é muito grande. Além disso, uma particularidade desse tipo de sinal é que ela costuma ter muita redundância, o que faz com que as simplificações que visam reduzir a quantidade de processamento sejam abordagens necessárias. No contexto desse trabalho, as simplificações criadas consistem em reduzir a quantidade de processamento, gerando dados que divergem daqueles gerados pela abordagem não-simplificada, mas fazendo com que essa divergência não seja percebida pelo sistema visual humano. Assim, as simplificações propostas nas seções a seguir são a *seleção de informação visual significante*, o *processamento guiado por contorno* e a *codificação diferencial*.

4.1.1 Seleção de Informação Visual Significante (SIVS)

Os passos descritos no capítulo anterior descrevem o procedimento para aumentar as dimensões de um canal em uma imagem, com apenas uma profundidade de cor. Contudo,

vídeos coloridos geralmente possuem três canais de cores. Assim, o primeiro passo de simplificação consiste em aplicar a operação de aumento de resolução no canal que possui maior informação de detalhes.

Considerando que sinais de vídeo são, geralmente, codificados no formato YUV (um canal de luminância e dois canais de crominância), a primeira simplificação consiste em utilizar os algoritmos de super-resolução somente no canal de luminância (Y) e utilizar interpolação nos demais canais de cores. Isso é aceitável, porque o sistema visual humano é muito mais sensível aos detalhes de luminância do que aos detalhes de cores. Essa simplificação consiste em uma simplificação em nível de dados. Nesse caso, tem-se que a simplificação consiste em utilizar a função-objetivo f em apenas um canal ao invés de aplicá-la aos três canais. Como consequência, com essa diminuição dos dados, diminui o tempo de processamento.

4.1.2 Processamento Guiado por Contorno (PGC)

Uma segunda simplificação adotada foi o processamento guiado por contorno. Essa estratégia consiste em criar uma região de interesse que será ampliada pelo custoso algoritmo de super-resolução. Para isso, cria-se uma máscara, aplicando-se o filtro de Canny [15], que destaca as regiões de bordas e de maiores detalhes da imagem.

Assim, o processo de segmentação gera duas classes de regiões na imagem. Para a aplicação descrita neste trabalho, as regiões de interesse (R_i) são aquelas em que será utilizado um algoritmo de super-resolução, muito mais custoso que a simples interpolação. As demais regiões são classificadas como secundárias (R_s). A Figura 4.1-(b) ilustra essa classificação. Nessa imagem, a área em preto corresponde à R_s, enquanto a parte destacada em branco corresponde à R_i.

Após a segmentação e a classificação das regiões R_s e R_i, é necessário uniformizar os dados para que eles possam ser processados de acordo com os algoritmos de ampliação.

Para isso, particiona-se a imagem filtrada em blocos de tamanho $n \times n$, verificando se há bordas destacadas no bloco. Caso haja, marca-se toda a região do bloco. Caso contrário, o bloco todo recebe o valor falso. Dessa maneira, se um bloco possui uma região R_i, ele é marcado como sendo um *bloco complexo* (B_c), senão, ele é um *bloco simples* (B_s).

Tal procedimento gera como resultado uma máscara binária que é usada como guia para aplicação do algoritmo sobre os blocos. Assim, os blocos complexos são processados pelo algoritmo de super-resolução, enquanto os blocos simples são ampliados com um algoritmo de interpolação simples.

As Figuras 4.1-(c) e (d) ilustram o resultado da classificação dos blocos de acordo com diferentes tamanhos de n. Os blocos brancos são B_c e os pretos são B_s. A partir dessa figura é possível notar que quanto menor for o bloco de particionamento, maior será a área descartada e, portanto, menor a área a ser processada pelo algoritmo mais custoso. Uma visão geral que ilustra essa e as outras etapas de simplificação dos dados espaciais é ilustrada na Figura 4.2.

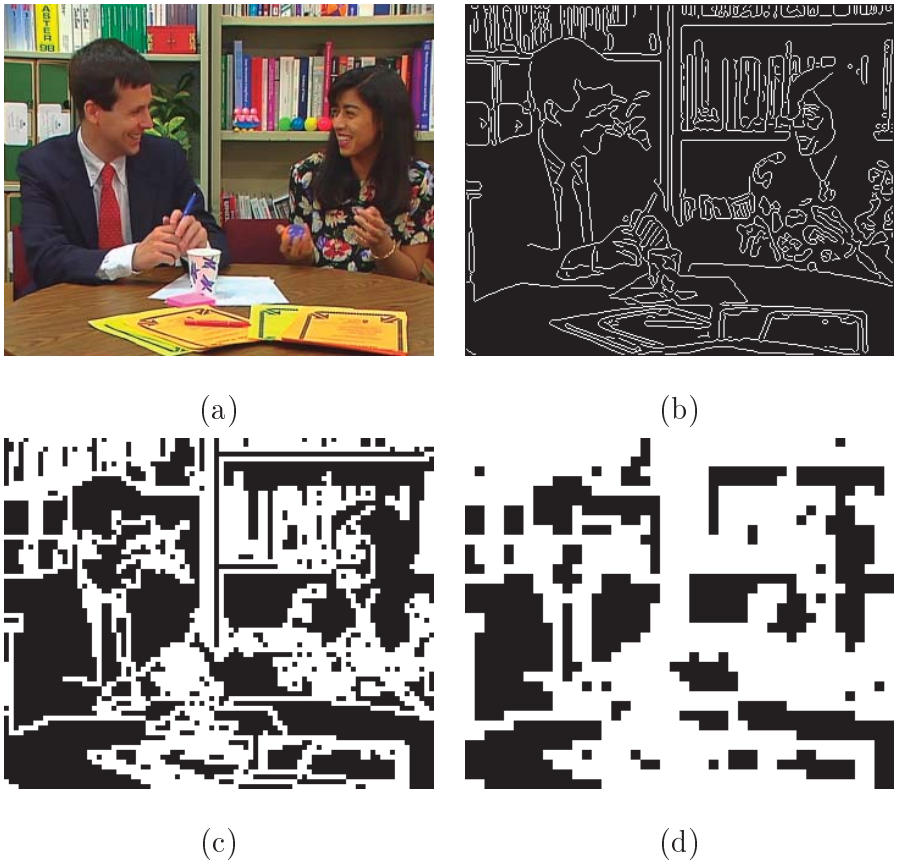


Figura 4.1: Seleção de regiões de interesse para processamento seletivo. (a) original, (b) segmentação com algoritmo de Canny, (c) partição com blocos de tamanho 4×4 e (d) partição com blocos de tamanho 8×8.

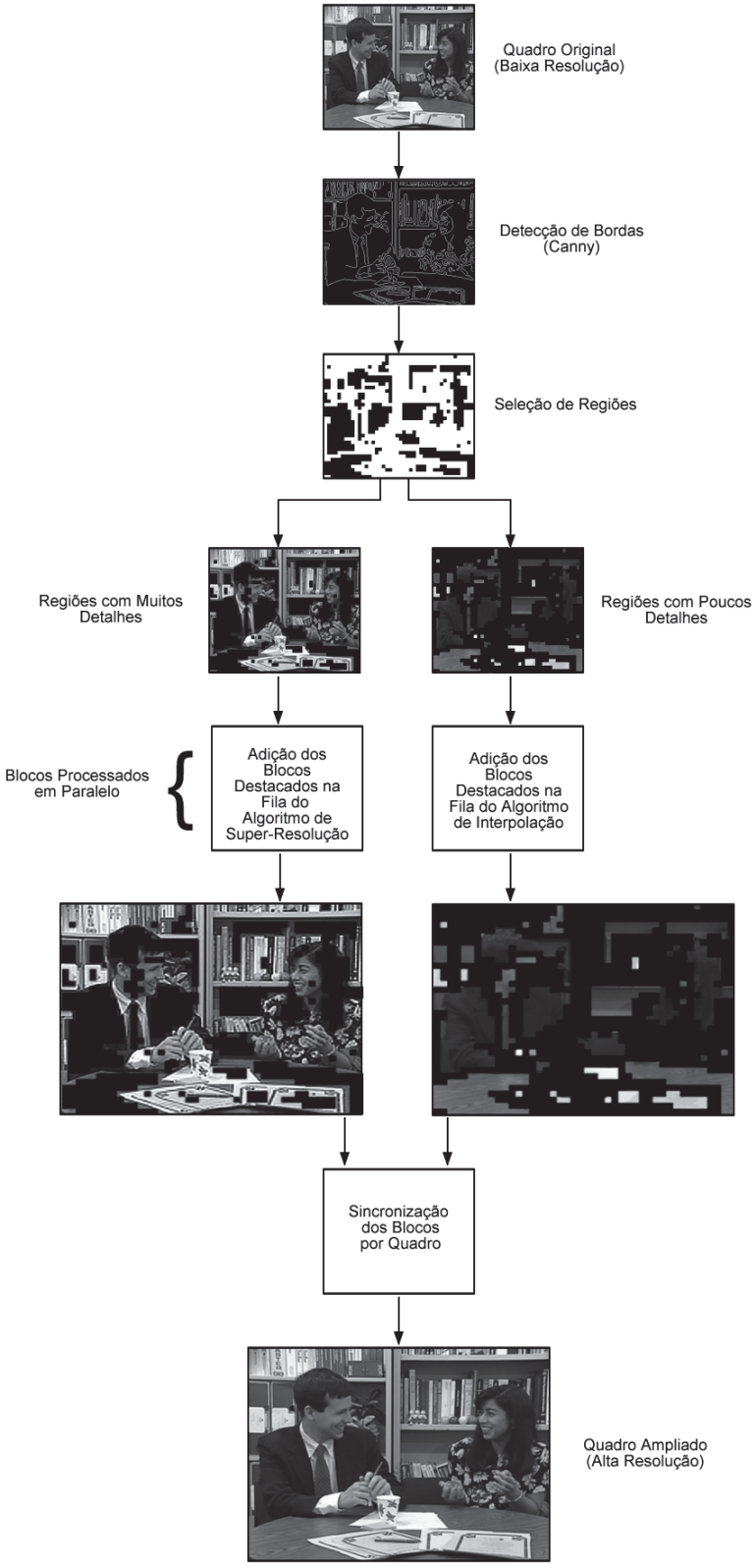


Figura 4.2: Diagrama geral do *framework* proposto.

4.1.3 Codificação Diferencial (CD)

Ambas as simplificações descritas na seção anterior (SIVS e PGC) exploram características em nível de dados e atuam exclusivamente sobre os dados espaciais. Logo, se cada quadro for processado de forma independente, haverá uma quantidade muito grande de informação a ser classificada. Além disso, a informação entre quadros muito próximos é muito redundante.

Essa redundância é principalmente notada em quadros consecutivos em um vídeo. Nesse tipo de sinal, a diferença entre os quadros deve ser pequena para que o observador tenha a sensação de movimento suave. Assim, há pouca variação entre um quadro e outro. A Figura 4.3 ilustra essa redundância na informação. Visualmente é possível notar que a diferença entre essas imagens é quase imperceptível. A imagem produzida pela diferença entre os quadros, ilustradas na Figura 4.3-(c), destaca a informação que foi alterada ao longo do tempo. Nesse caso, as partes pretas consistem na informação redundante, enquanto as regiões em branco são as regiões com a nova informação.

Matematicamente, essa redundância consiste em uma similaridade das distribuições entre quadros consecutivos. Isso é, entre esses quadros, a distribuição dos valores de cor são muito próximos. Além disso, é possível notar que essas distribuições não se caracterizam por um modelo matemático simples, o que dificulta modelar e quantificar a redundância.

Observando os histogramas da Figura 4.4, é possível notar como a frequência de ocorrências dos valores distintos é drasticamente reduzida no quadro diferencial. Nesse quadro, os valores ficam todos centrados em torno de um único valor. Adicionalmente, é possível notar que esse comportamento se assemelha a uma distribuição de Poisson [80], onde o valor mais provável é zero, que é o valor mais redundante. Dessa maneira, utilizando-se a informação diferencial é possível saber onde a informação é mais redundante. Assim, valores mais próximos do valor nulo são os mais redundantes, enquanto que os valores mais distantes dele são aqueles que agregam maior informação visual.

Para aproveitar essa redundância, a abordagem a ser adotada consiste em processar os quadros diferenciais. Esse tipo de abordagem é bastante conhecida em compactação de vídeos como *codificação diferencial* [60]. Como é possível notar, a eliminação da redundância ajuda muito na redução dos dados. Por isso, este trabalho toma esse termo emprestado para descrever essa etapa do *framework*.

Quando a etapa de PGC é aplicada nos quadros diferenciais, nota-se uma redução ainda maior das regiões classificadas como R_i e, por consequência, há um número muito menor de blocos B_c. A Figura 4.5 ilustra as regiões selecionadas, na qual é possível notar que na Figura 4.5-(a) o processamento sobre o quadro requererá uma quantidade muito maior de esforço computacional do que na Figura 4.5-(b), uma vez que há mais regiões classificadas.

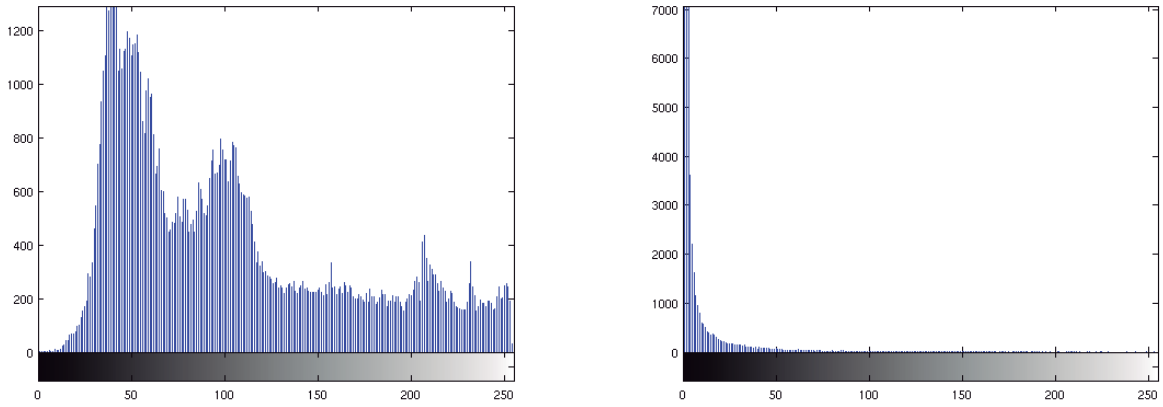
Por outro lado, a partir da imagem da Figura 4.5-(b), o número de regiões selecionadas é menor. Para esses quadros ilustrados, por exemplo, a quantidade de blocos a serem processados em cada quadro é 1.9 vezes a quantidade de blocos a serem processados utilizando-se codificação diferencial. Ou seja, neste caso, a codificação diferencial reduz pela metade a quantidade de dados a serem processados, o que implica em uma redução no custo do tempo em semelhante proporção.

Como exposto, a abordagem de utilizar os quadros diferenciais reduz a quantidade de informação nos quadros a serem processados pelos algoritmos mais custosos. Contudo, essa informação é muito mais sensível, pois ela consiste justamente em informações de alta-frequência, que são perdidas no processo de interpolação. A Figura 4.6 demonstra como essa informação é sensível. Ela compara a formação do primeiro quadro gerado a partir do diferencial do quadro de referência. As imagens da Figura 4.6-(a) e (b) são as reconstruções ampliadas do diferencial equivalente ao da Figura 4.3-(b). As imagens das Figuras 4.6-(c) e (d) ilustram o erro absoluto entre o quadro diferencial e o aumentado, utilizando interpolação e um algoritmo de super-resolução, respectivamente. As Figuras 4.6-(e) e (f) mostram o resultado da reconstrução com esses quadros.

(a)

(b) (c)

Figura 4.3: Codificação diferencial dos quadros: (a) primeiro quadro de referência, (b) segundo quadro, (c) diferença entre o primeiro e o segundo quadro.



(a) (b)

Figura 4.4: Histogramas dos *pixels* entre os quadros e de seus respectivos diferenciais: (a) segundo quadro, (b) diferencial entre o segundo e o primeiro quadro.

(a) (b)

Figura 4.5: Blocos selecionados como de interesse (em branco): (a) segundo quadro, (b) diferencial entre o segundo e o primeiro quadro.

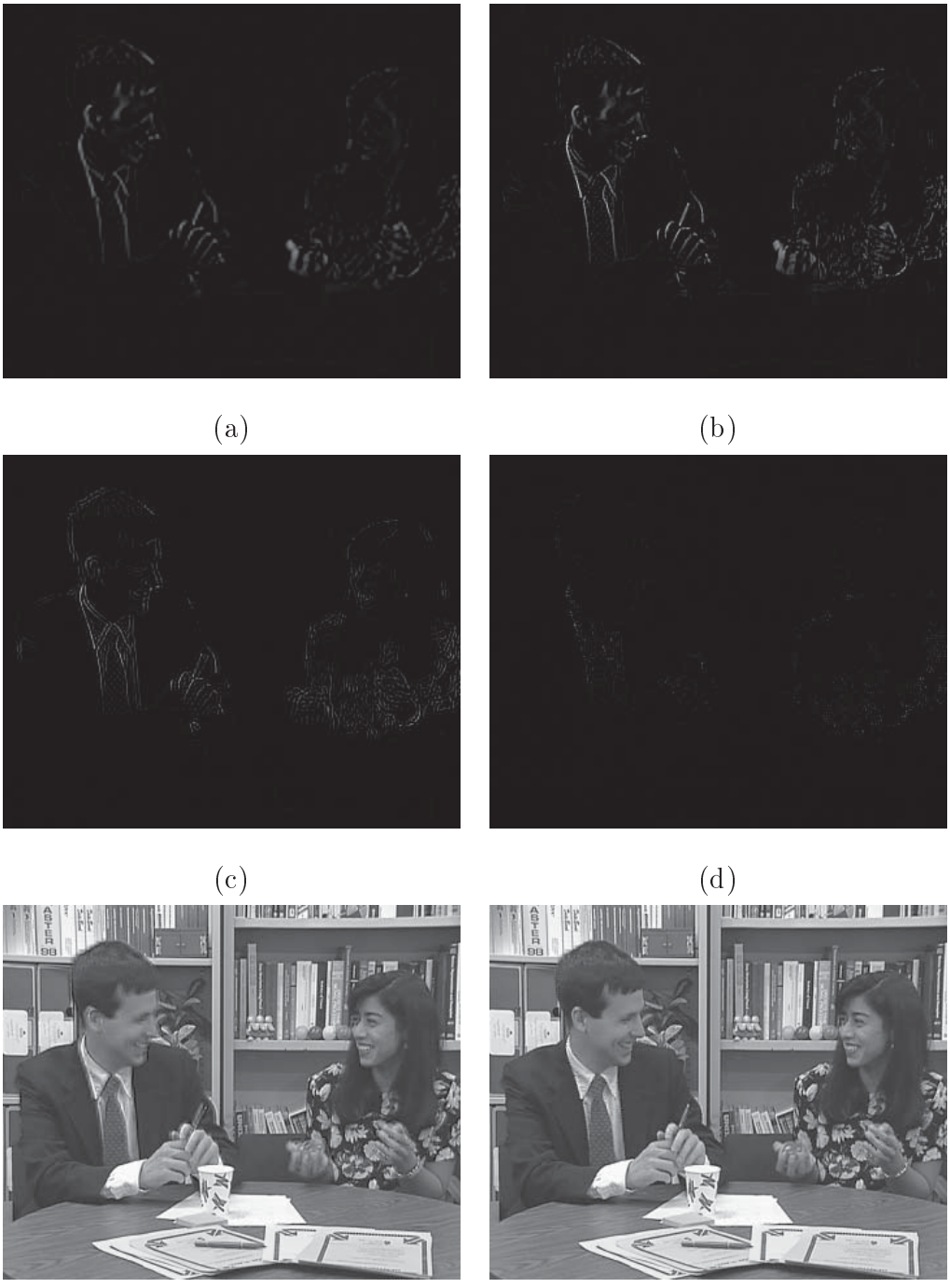


Figura 4.6: Resultado dos quadros diferenciais ampliados: (a) ampliado por interpolação, (b) ampliado utilizando um algoritmo de super-resolução, (c) erro entre o quadro diferencial original e o interpolado, (d) erro entre o quadro diferencial original e o aumentado por SR, (e) quadro compensado com o diferencial interpolado e (f) quadro compensado com o diferencial ampliado por SR.

4.2 Processamento Paralelo do *Framework* Proposto

Os primeiros estágios de simplificação são facilmente paralelizáveis. Para isso, inicialmente, cria-se um *buffer* contendo um conjunto de quadros consecutivos. Dependendo do tamanho total do vídeo e dos recursos computacionais disponíveis para distribuição dos processos, vários *buffers* podem ser criados e processados sequencialmente. Considerando a relação tamanho do vídeo versus quantidade de recursos, o tamanho e a quantidade dos *buffers* pode ser ajustada.

Esta seção descreve a estratégia de processamento distribuído de somente um *buffer*. A ideia é que se há mais de um *buffer*, basta que todas as etapas descritas sejam feitas em todos eles. Tais etapas de processamento paralelo descritas para cada *buffer* são a *simplificação*, a *classificação*, a *distribuição*, o *processamento* e a *reconstrução*, os quais são descritos nas seções abaixo.

4.2.1 Etapa1 - Simplificação e Classificação

A etapa de simplificação pega um conjunto de quadros e os enumera de acordo com sua posição no tempo, com o intuito de manter a ordenação da informação temporal. Essas posições são mantidas numa fila T, que é uma estrutura de dados compartilhada entre os processos.

Em seguida, atribui-se um processo a um conjunto de quadros de forma proporcional aos recursos disponíveis. Por exemplo, se um *buffer* contém K quadros distintos em um ambiente com K processadores, distribui-se um quadro por processador. Em seguida, define-se cada processo como $p(t, e_i)$, onde t é o processo responsável pelo quadro t, e_i é o estágio e i é a etapa de simplificação e classificação. Assim, há três estágios nesta etapa de simplificação e classificação, as quais são relativas à codificação diferencial, ao processamento guiado por contorno e à classificação dos blocos.

O **primeiro estágio**, e_1, consiste em calcular os quadros diferenciais. Para isso, cada processo, $p(t, e_1)$, calcula a diferença entre os quadros t e $t+1$, verificando a quantidade de quadros não-diferenciais que o usuário deseja manter. Isso é, se o usuário deseja manter r quadros de referência, todos os quadros cuja posição t corresponda à condição descrita pela Equação 4.1,

$$t \; mod \; \frac{K}{r} = 0 \tag{4.1}$$

serão mantidos. Para guardar a informação do tipo do quadro (diferencial ou quadro completo), uma fila B, com índice proporcional à T, é criada. Assim, B serve como um mapa de *bits*, indicando se um quadro t qualquer é diferencial ou não.

O **segundo estágio**, e_2, consiste na detecção e seleção das regiões de interesse. Para isso, cada processo, $p(t, e_2)$, processa independentemente o quadro t de acordo com o descrito na Seção 4.1.2. De forma semelhante, no **terceiro estágio**, e_3, o processo $p(t, e_3)$ detecta as regiões complexas e simples, classificando cada bloco como sendo "selecionado" ou "não-selecionado", respectivamente. Por fim, os estágios e_1, e_2 e e_3 estão ilustrados na Figura 4.7.

4.2.2 Etapa 2 - Distribuição e Processamento

O particionamento utilizado na simplificação e processamento seletivo demonstra que há uma clara independência de dados entre cada quadro do vídeo. Cada conjunto de operações é executado sobre cada bloco independente, como se fosse uma imagem independente. Isso permite que processos diferentes processem cada bloco independentemente e paralelamente.

Por outro lado, quando o processamento guiado por contorno é executado em diferentes quadros, a quantidade de blocos pode ser variável, uma vez que quadros distintos não terão a mesma quantidade de detalhes e informações de alta-frequência. Esse comportamento é notável quando observada a diferença entre os quadros completos e os quadros diferenciais,

conforme ilustram as Figuras 4.5-(a) e (c). Portanto, a divisão homogênea de blocos por quadro é pouco conveniente para a distribuição homogênea entre os diferentes processos.

Para homogeneizar os dados, após a etapa de simplificação, cada bloco é encapsulado em uma estrutura de dados chamada **célula**. A célula contém uma *flag* de *bit* único, indicando a classe do bloco (B_c ou B_s), uma variável inteira indicando o quadro ao qual o bloco pertence, duas variáveis inteiras de posição do bloco no quadro, e uma matriz contendo os valores dos *pixels*. Portanto, esta estrutura contém o instante, a posição do bloco no quadro e dados da sub-imagem do quadro ao qual o bloco endereça, onde x e y representam a posição horizontal e vertical do bloco, t é a posição temporal do quadro, s é o bit de seleção (indicando se é ou não uma região de interesse) e *matriz* contém os *pixels* do bloco. Essa estrutura é ilustrada na Figura 4.8.

Para gerar uma distribuição dos dados que melhor balanceie a carga entre os processos, duas filas, $\mathbf{F_{in}}$ e $\mathbf{G_{in}}$, são criadas de acordo com a classificação de cada bloco. Quando uma célula contém um *bit* de seleção com o valor verdadeiro, ela é armazenada na primeira fila disponível da fila $\mathbf{F_{in}}$. Caso contrário, esta operação é feita na fila $\mathbf{G_{in}}$. Portanto, nessa estrutura de dados, os blocos são dispostos de acordo com a demanda e não mais seguindo uma ordem relacionada à posição dos *pixels* ao longo do vídeo. Dessa maneira, a distribuição de dados é homogeneamente balanceada para ambas as filas.

Tanto $\mathbf{F_{in}}$ quanto $\mathbf{G_{in}}$ são estruturas acessíveis entre todos os processos. Contudo, embora cada célula enfileirada em uma dessas filas possa ser adicionada por cada um dos processos, cada processo cria duas filas próprias, $\mathbf{f_{in}(t)}$ e $\mathbf{g_{in}(t)}$, que não são compartilhadas entre eles.

Após as filas $\mathbf{f_{in}(t)}$ e $\mathbf{g_{in}(t)}$ estarem completas, com todos os blocos classificados, seus elementos são, finalmente, copiados para $\mathbf{F_{in}}$ e $\mathbf{G_{in}}$. O objetivo desta abordagem é evitar que a sincronia feita para a inserção em $\mathbf{F_{in}}$ e $\mathbf{G_{in}}$ bloqueie todos os estágios e_1, e_2 e e_3 dos outros processos.

Dessa forma, separando o processamento desses três estágios do enfileiramento nas estruturas globais, evita-se que uma máquina de menor poder computacional coloque outra com maior poder computacional em estado de bloqueio desnecessariamente. Além disso, como quadros diferentes possuem quantidades de blocos classificados com quantidades distintas, então o tamanho da fila $\mathbf{f_{in}(t)}$ pode ser diferente de $\mathbf{f_{in}(t+1)}$, o que poderia fazer com que o tempo de enfileiramento em $\mathbf{F_{in}}$ ficasse condicionado aos quadros com maiores regiões complexas, se não houvesse essa separação entre o processamento da simplificação, armazenamento em uma estrutura interna e compartilhamento com região concorrente.

A Figura 4.9 ilustra a classificação e distribuição nas estruturas internas e externas. Nos três quadros ilustrados, cada bloco destacado consiste em uma célula. As cores distintas servem para indicar qual o quadro, e qual a classe de bloco pertence. Por exemplo, as regiões vermelhas são de blocos complexos no quadro t, as regiões amarelas são blocos no quadro $t+1$, e assim por diante. Note que quando essas informações são enfileiradas em $\mathbf{F_{in}}$, os elementos podem ser tratados de maneira uniforme, reduzindo o contexto da posição, permitindo que o processamento paralelo dessa fila esteja descorrelacionado com a informação do quadro.

Em seguida, as filas $\mathbf{F_{in}}$ e $\mathbf{G_{in}}$ são distribuídas ao longo dos nós. Essa distribuição é feita entregando para cada nó uma malha. Neste caso, uma malha é um conjunto de células proporcional ao número de nós na rede. Isso é, um ambiente com K nós terá K malhas. Assim, dependendo da capacidade de cada nó, as malhas podem ter tamanhos diferentes. Por exemplo, um ambiente com dois nós, X e Y, onde o tempo de processamento seja $\tau(X) = z\tau(Y)$ e $z > 1$. Nesse caso, o tamanho da malha entregue ao processo em Y deve ser proporcional à z vezes o tamanho daquela entregue para X.

Para aproveitar arquiteturas heterogêneas, conforme ilustrado na Figura 2.5, cada nó possui um conjunto de processadores que executa uma malha independente. Cada um desses processadores captura uma célula dentro da malha ao qual ele está restrito, processa

e enfileira em uma estrutura interna do nó $\mathbf{f_{out}(t)}$.

Nesse ponto, $\mathbf{f_{out}(t)}$ contém as sub-imagens aumentadas, mas ainda descorrelacionadas da ordem do quadro. Após o processamento, cada célula de entrada salva o resultado em uma célula simétrica, contendo as mesmas informações de localização em x, y e t, mas com uma matriz de tamanho $mn \times mn$, onde m é a ordem do aumento dimensional, e n é o tamanho original do bloco, conforme ilustra a Figura 4.10.

A Figura 4.11 ilustra a distribuição e o processamento ao longo dos nós, destacando as malhas k e $k+1$, relativas aos nós k e $k+1$. Nessa figura, cada malha possui apenas quatro células, que são distribuídas para cada um dos quatro processadores dentro dos nós. A Figura 4.11 destaca ainda como é feito o processamento de $\mathbf{F_{in}}$ e $\mathbf{G_{in}}$ em uma disposição mestre-escravo.

No caso, $\mathbf{G_{in}}$ é processado sequencialmente para $\mathbf{G_{out}}$ pelo processo mestre, pois o tempo de processamento de cada célula dessa fila é muito menor que das células em $\mathbf{F_{in}}$. Assim, os nós-escravos são reservados ao processamento dos algoritmos custosos.

4.2.3 Etapa 3 - Reconstrução

Após os enfileiramentos de $\mathbf{f_{in}(t)}$ terem sido ampliados e enfileirados em $\mathbf{f_{out}(t)}$, esses blocos são copiados de volta para $\mathbf{F_{out}}$. Por sua vez, o processo de ordenamento inverso é realizado, percorrendo cada uma dessas filas, verificando a qual quadro cada bloco pertence e redistribuindo os blocos em suas devidas posições, conforme ilustra a Figura 4.12.

Quando todos os blocos aumentados forem reordenados dentro de um conjunto de quadros, tem-se que os quadros estão divididos entre quadros de referência e diferenciais. Sendo assim, o último estágio consiste em iterar sobre o mapa de *bits*, e verificar se o quadro é de referência ou não, somando a referência com os diferenciais a fim de ter todos os quadros reconstruídos.

A Figura 4.13 ilustra como funciona essa última etapa. Comparando essa figura com a Figura 4.7, é possível notar que o cálculo dos quadros diferenciais é claramente paralelizável. Por outro lado, o processo de restauração ilustrado na Figura 4.13 é intrinsecamente serial, pois o quadro restaurado $t+1$ depende de t, mas $t+2$ depende da reconstrução de $t+1$, o que ilustra um processo iterativo.

Como a reconstrução dos quadros diferenciais consiste apenas em somá-lo com o quadro de referência anterior, cada operação *pixel-a-pixel* é totalmente independente. Sendo assim, para longos *buffers*, pode-se paralelizar o processamento dividindo-se os quadros referenciais, proporcionalmente ao número de processadores por nós. Nesse caso, cada processador fica responsável por um trecho da sequência, que vai iterando ao longo dos quadros diferenciais.

A Figura 4.14 ilustra esse processo para um conjunto de quatro processadores por nós. Quatro processos são gerados, cada um processando paralelamente o primeiro, o segundo, o terceiro e o quarto quadrante de cada quadro. Note nessa figura que a estrutura B foi modificada. Ao invés de usá-la como mapa de *bits*, B guarda a posição dos quadros referenciais. Dessa maneira, é possível saber que todos os quadros entre B_j e B_{j+1} são diferenciais. Com essas informações, a partir da Figura 4.14 é possível concluir que os quadros $t+3$ e $t+6$ são referenciais. Portanto, é possível entregar todos os quadros no intervalo $[t, t+2]$ para um nó, os quadros em $[t+3, t+5]$ para outro nó, e assim por diante. No caso, $P_j(t, q)$ endereça o processo considerando os dois níveis de paralelização. No caso, j é o índice de B que aponta para o processo equivalente, t é o índice de qual quadro referencial está sendo processado (conteúdo de B_j) e q representa o quadrante.

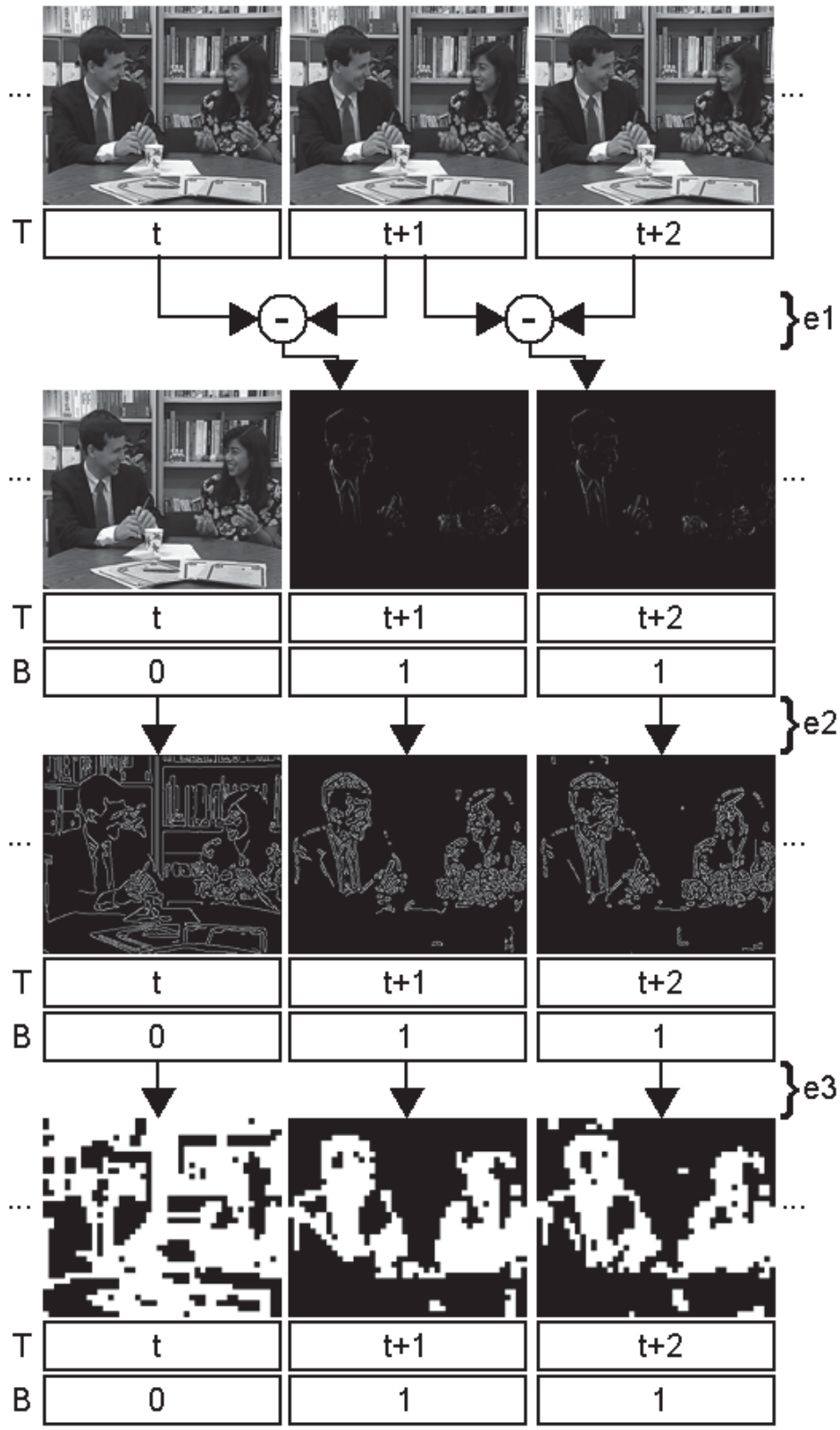


Figura 4.7: Etapas de simplificação e classificação processadas em paralelo.

Figura 4.8: Estrutura de endereçamento do bloco de entrada.

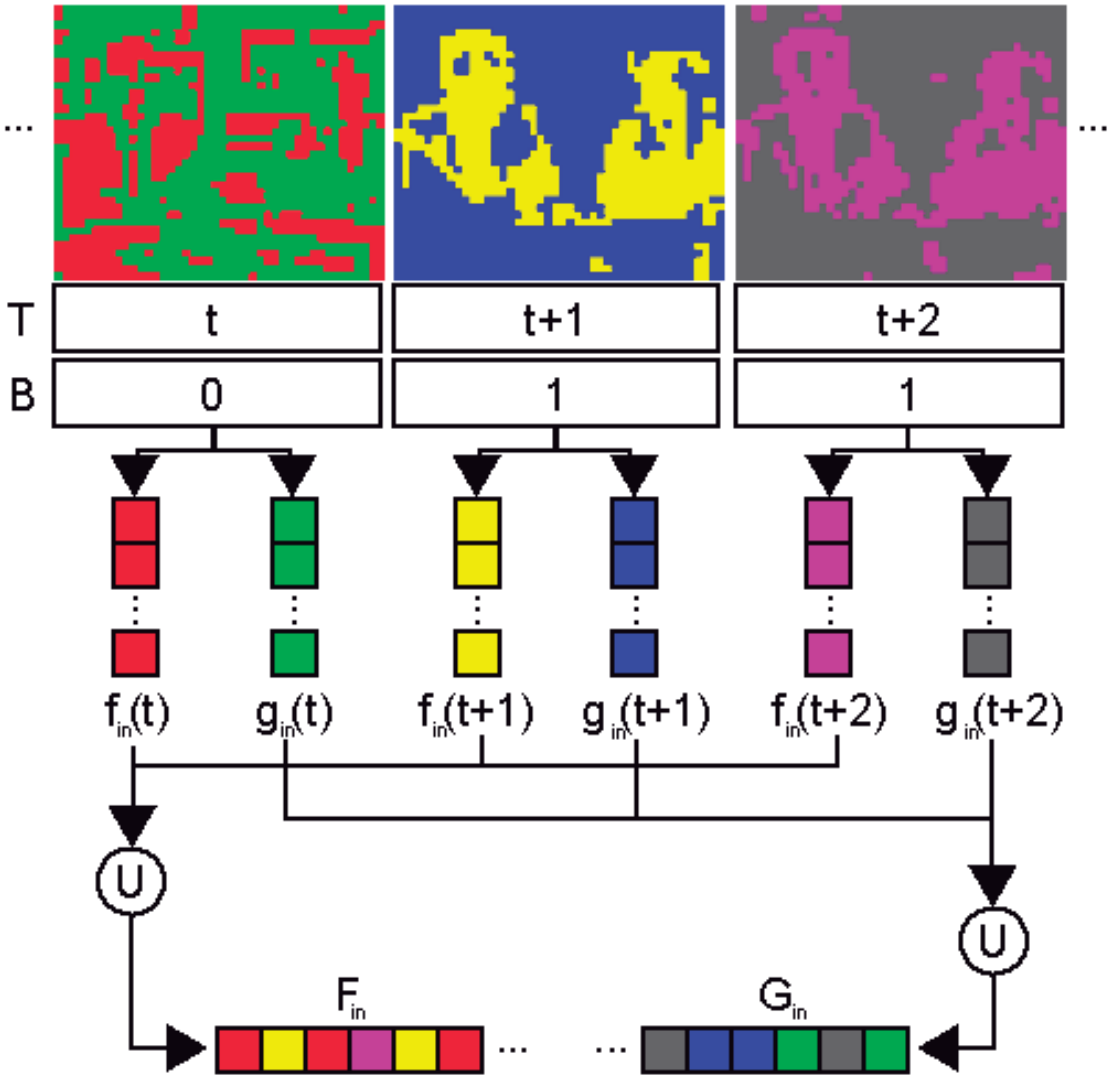


Figura 4.9: Distribuição dos dados ao longo dos processos.

Figura 4.10: Estrutura de endereçamento do bloco de saída.

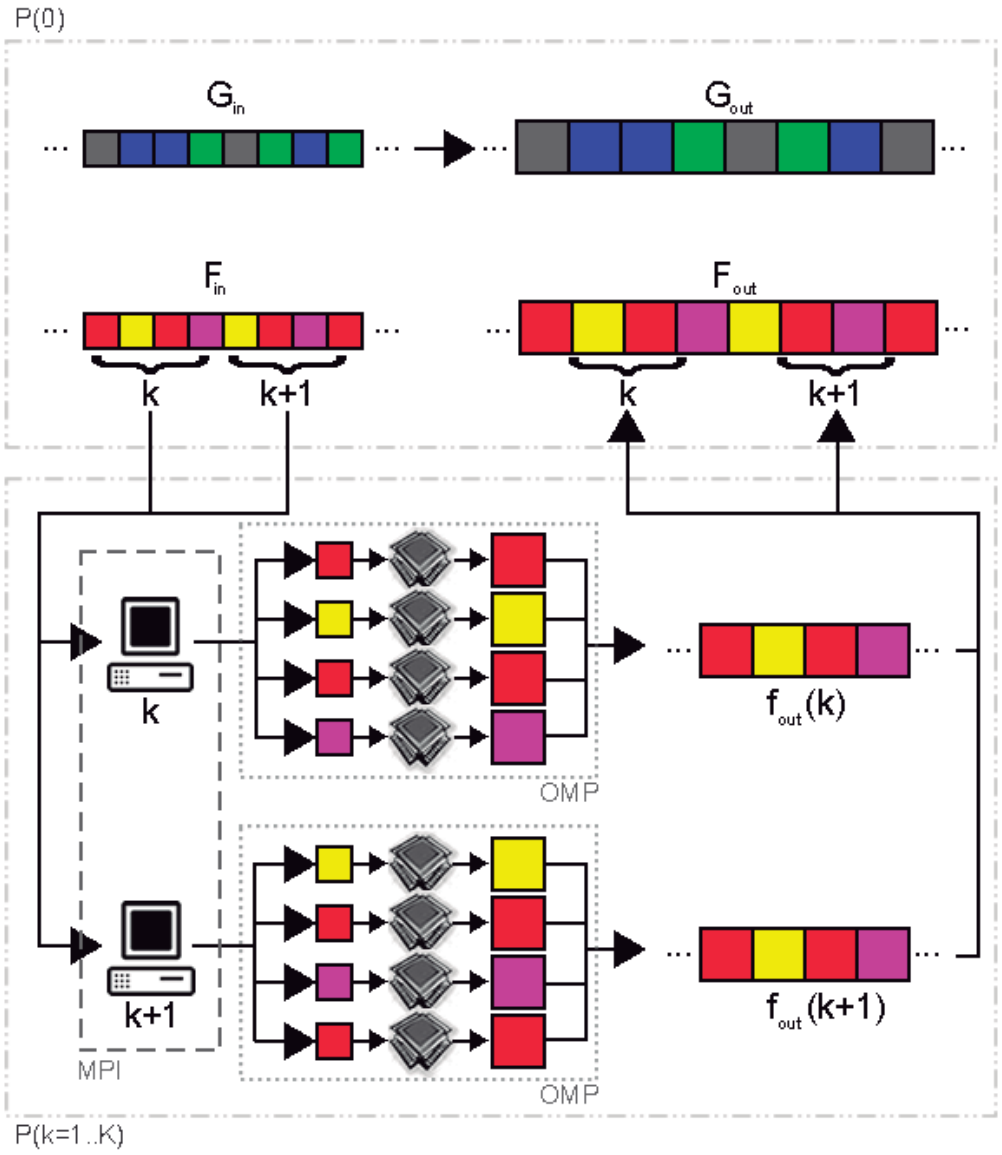


Figura 4.11: Distribuição dos dados ao longo dos processos.

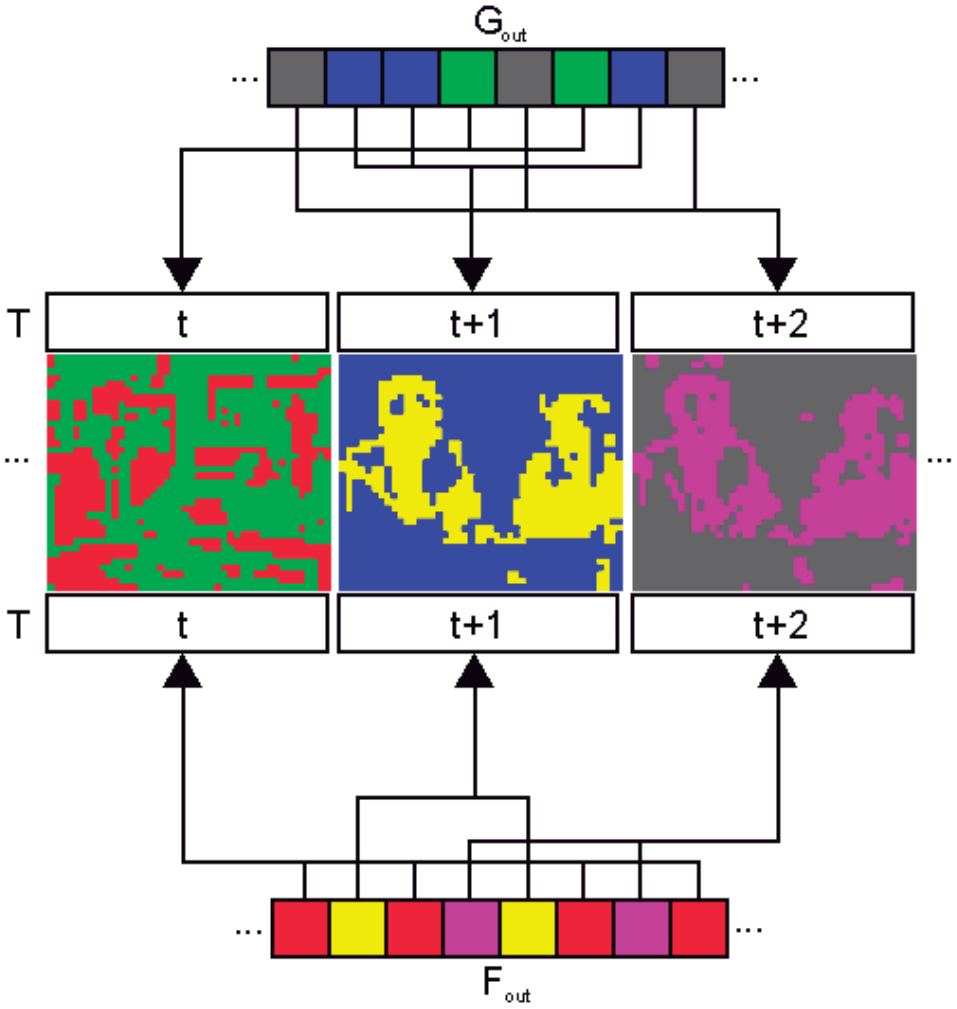


Figura 4.12: Reordenamento dos blocos nos quadros.

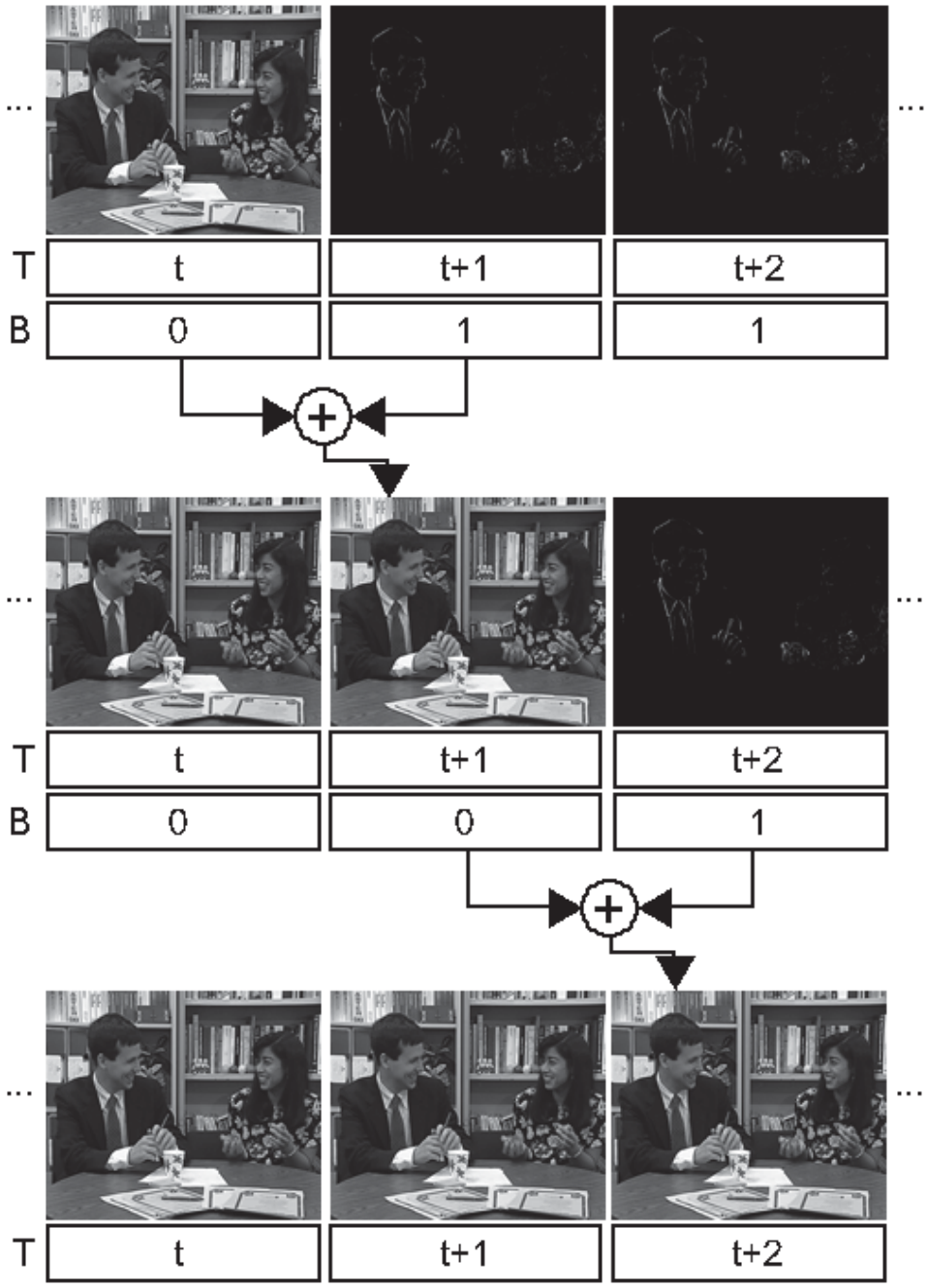


Figura 4.13: Reconstrução dos quadros diferenciais.

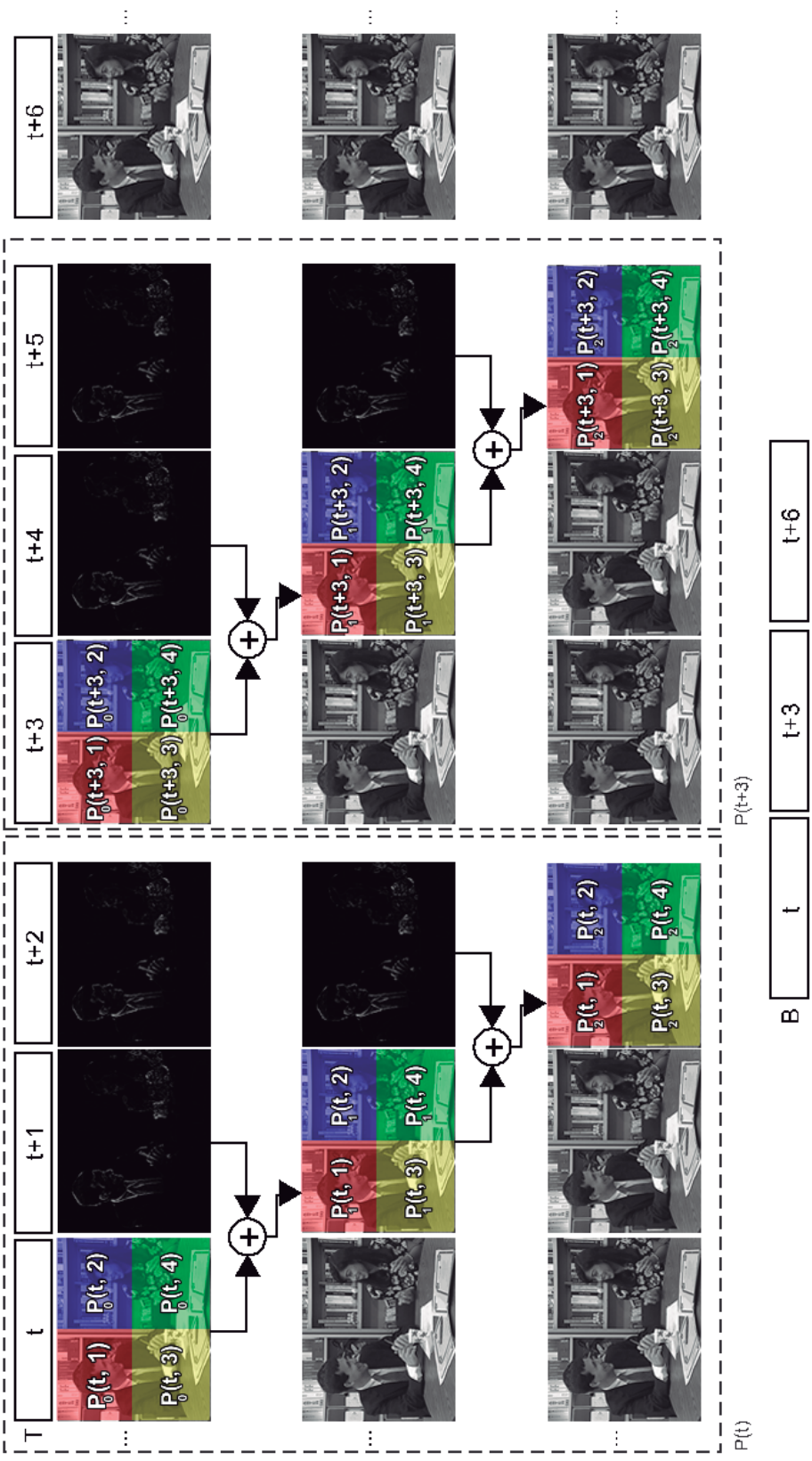


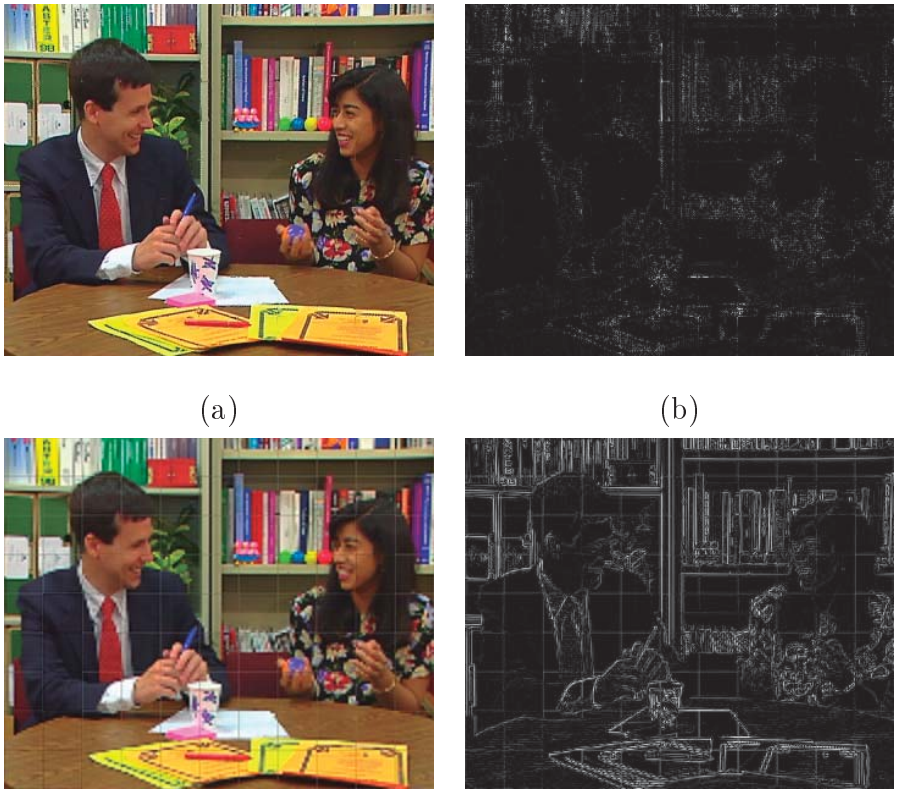
Figura 4.14: Reconstrução dos quadros diferenciais em paralelo.

4.3 Redução de Efeitos de Blocagem

A decomposição dos quadros do vídeo em blocos que são processados de forma independente apresenta como efeito colateral um efeito conhecido como blocagem (*blocking*). Esse efeito consiste na geração de artefatos em forma de contornos em regiões que não estavam na imagem original. O nome blocagem é dado porque o bloco fica visivelmente destacado na imagem processada, conforme ilustrado na Figura 4.15.

As Figuras 4.15-(a) e (c) apresentam os quadros após o processamento, utilizando um algoritmo de aumento de resolução e interpolação, respectivamente. Na Figura 4.15-(a) é possível notar bordas artificiais entre os blocos, como se a imagem fosse montada como um mosaico. Na Figura 4.15-(c) esse efeito fica ainda mais evidente. As Figuras 4.15-(b) e (d) exibem a diferença em relação à imagem original, destacando o erro gerado.

Para evitar esse efeito de blocagem produzido, basta introduzir uma borda de zeros contornando cada bloco. Assim, antes de chamar o algoritmo de ampliação das dimensões, o *framework* deve adicionar essa borda artificial, e, após o término do algoritmo em cada bloco, deve-se remover a borda gerada no bloco aumentado.



(a) (b)

(c) (d)

Figura 4.15: Exemplos do efeito de blocagem em quadros de vídeo: (a) ampliada utilizando super-resolução, (b) erro da imagem a com relação ao quadro original, (c) ampliada utilizando interpolação bilinear e (d) erro da imagem c com relação ao quadro original.

4.4 Considerações Finais

Este capítulo descreveu o *framework* proposto neste trabalho. Primeiramente foi descrito as técnicas de simplificação propostas para reduzir a carga de dados a serem processados pelo algoritmo mais custoso (super-resolução). Em seguida, foram traçados os passos do *framework* que realizam o processamento paralelo.

Com essa abordagem que combina simplificação e processamento paralelo, espera-se que o tempo necessário para ampliar todos os quadros de um vídeo seja reduzido, quando comparado com o tempo de processamento de todos os quadros de forma serial. Assim, o

próximo capítulo analisa o ganho do desempenho comparando-se as abordagens simplificada e não-simplificadas, e as implementações serial e paralela.

Capítulo 5

Resultados das Simulações

O esquema de paralelização foi desenvolvido para ser portável para diferentes arquiteturas paralelas. A distribuição da carga, que corresponde à divisão de F_{in} em malhas ao longo dos nós, pode ser feita com uma tecnologia que favoreça a entrega dos dados em ambiente multicomputador. Bibliotecas que utilizam passagens de mensagem, tais como a *Message Passing Interface* [58] e a *Parallel Virtual Machine* [11], são exemplos dessas tecnologias.

Os resultados apresentados neste capítulo utilizam um paradigma de único programa que manipulam múltiplos dados (*Single Program, Multiple Data*, SPMD) para explorar o paralelismo em memória distribuída. A implementação explora esses recursos no ambiente Matlab [63]. Os recursos providos para *SPMD* nesse ambiente distribuem os dados utilizando rotinas do MPI (MPICH2)[36, 62], mas fornecendo uma interface transparente ao programador.

Além disso, também utilizando Matlab, o processamento dentro de cada nó é dividido entre os processadores utilizando os recursos do *parfor* e *blkproc*. A ideia do *parfor* no Matlab é funcionar de forma semelhante aos recursos de paralelização das repetições no OpenMP [18]. Isso é, em uma estrutura de repetição, o conjunto de declarações no corpo da

repetição é executado de forma independente em cada repetição. Contudo, ao invés de cada execução independente ser separada pelo tempo (iteração), ela é separada pelo espaço, em diferentes processadores. Dessa maneira, diferentes tarefas processam diferentes intervalos da repetição.

Como cada iteração do *parfor* é executada independente de ordem, o *parfor* é útil quando há iterações nos quais os dados são independentes e não criam operações dependentes entre elas. Quando uma iteração depende do resultado de iterações anteriores, esse tipo de abordagem de paralelização deve ser evitada, pois gera as condições classificadas por Bernstein [12], apresentadas na Seção 2.3.3.

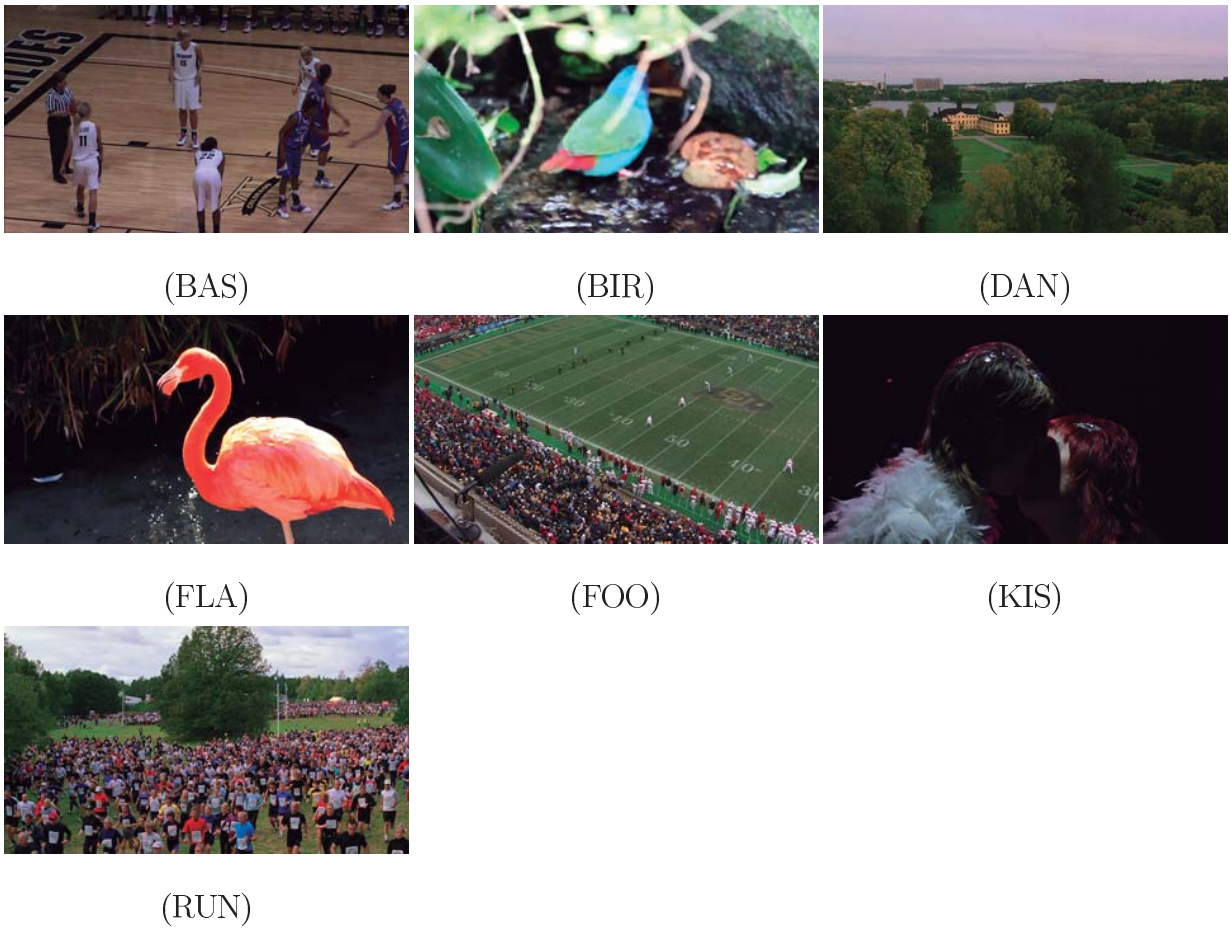
Outras conformações de arquiteturas e tecnologias podem ser utilizadas para melhorar a vazão de processamento. O *framework* pode, por exemplo, ser portado para ser utilizado em uma arquitetura mista com placas gráficas. Isto é, a fila de maior custo computacional pode ser distribuída ao longo dos nós, para serem processadas utilizando-se GPUs ao invés das CPUs. Por outro lado, as filas de menor custo podem ser processadas pelas CPUs ociosas em cada nó, paralelamente. Assim, embora outros tipos de arquiteturas possam aproveitar a estratégia de paralelização descrita, esse trabalho restringiu a implementação somente ao ambiente descrito no começo desta seção, pois o objetivo das simulações é apenas avaliar o desempenho do *framework* proposto. Implementações mais refinadas podem constituir possíveis trabalhos futuros.

5.1 Resultados Obtidos

Os resultados apresentados nesta seção foram obtidos a partir do processamento de um conjunto de sete vídeos, livres de direitos autorais, utilizando-se o *framework* proposto. Uma miniatura de um quadro de cada um desses vídeos é apresentada na Figura 5.1. Esses vídeos possuem características espaciais e temporais diferentes entre si. Isso é, alguns

possuem maior quantidade de texturas espaciais, outros possuem objetos com movimentos ou velocidades distintas, etc.

Com o intuito de padronizar os testes, todos esses vídeos escolhidos possuem 720 linhas e 1280 colunas e 24 quadros por segundo (720p24). O comprimento dos vídeos são variáveis, sendo que o vídeo *Basketball* (BAS) possui 14 segundos (337 quadros), *Birds* possui 24 segundos (593 quadros), *Dancing* possui 40 segundos (961 quadros), *Flamingo* possui 10 segundos (250 quadros), *Football* possui 19 segundos (457 quadros), *Kiss* possui 14 segundos (326 quadros) e *Running* possui 20 segundos (481 quadros).



(BAS) (BIR) (DAN)

(FLA) (FOO) (KIS)

(RUN)

Figura 5.1: Quadros dos vídeos utilizados: *Basketball* (BAS), *Birds* (BIR), *Dancing* (DAN), *Flamingo* (FLA), *Football* (FOO), *Kiss* (KIS) e *Running* (RUN).

5.1.1 Análise do Desempenho da Simplificação

O primeiro passo para testar o desempenho da simplificação proposta no capítulo anterior, consistiu em contar a quantidade de blocos obtida ao se classificar as regiões quadro-a-quadro, utilizando a codificação diferencial. Com isso é obtido o total de dados rotulados como "não-selecionado" e "selecionado". No caso, os blocos "selecionados" são aqueles processados pelo algoritmo de super-resolução, enquanto que os "não-selecionados" são ampliados por interpolação. Dessa maneira, quanto mais blocos "não-selecionados", menor a quantidade de recursos computacionais exigida.

A Figura 5.2 apresenta a contagem dos blocos feita para os vídeos testados utilizando diferentes tamanhos de blocos. Nessa figura, a coluna da esquerda ilustra os gráficos dessa contagem feita sobre os blocos classificados quadro-a-quadro, seguindo os passos descritos na Seção 4.1.2 sem considerar a codificação diferencial. Já a coluna da direita ilustra a contagem da classificação feita considerando os quadros diferenciais.

Como uma maior quantidade de blocos "não-selecionados" é uma propriedade desejável, pois implica em exigir menos recursos computacionais, a Figura 5.3 apresenta a porcentagem desse tipo de bloco em relação ao total de blocos. Assim, a partir dessa figura, é possível observar que, quanto menor o bloco, maior a quantidade de blocos simplificados. Isso ocorre porque o critério escolhido para classificação consiste apenas em verificar se dentro do bloco há uma região complexa. Portanto, quanto maior o bloco, maior a probabilidade de haver uma região desse tipo contida nele.

Ainda com a Figura 5.3, é possível notar que a utilização da codificação diferencial aumenta a porcentagem de quadros não-selecionados na maioria dos casos. Contudo, como pode ser observado para os vídeos KIS e DAN, em alguns casos a codificação diferencial seleciona mais regiões do que na abordagem quadro-a-quadro. Isso ocorre porque o processamento guiado por contorno utiliza o filtro de Canny [15] para detecção de bordas. A utilização desse filtro nos quadros diferenciais irá destacar mais regiões em vídeos com

maior quantidade de movimento, uma vez que o realce da diferença inter-quadros destaca regiões tanto do quadro anterior quanto do posterior.

Mesmo com essas considerações, a utilização da simplificação permite evitar que até metade dos dados seja processada pelo algoritmo custoso. Sendo assim, o próximo passo de análise consiste em verificar como se comporta essa simplificação com relação ao tempo. Em outras palavras, como a segregação dos blocos classificados e os respectivos algoritmos impactam na velocidade de resolução do problema.

Para isso, o primeiro passo consiste em buscar a relação entre o tamanho do bloco, a quantidade de blocos gerados e o impacto no tempo. Conforme ilustra a Figura 5.4, o tamanho do bloco é inversamente proporcional à quantidade de blocos, e isso não depende do processo de classificação ou dos quadros serem diferenciais. Portanto, o teste consiste em variar o tamanho do bloco e verificar qual o tempo total de processamento é necessário utilizando-se simplificações ou não.

Nos testes, os tamanhos de blocos utilizados foram de 8×8, de 16×16, de 32×32, de 64×64 e de 128×128. Com essa variação de tamanho, foi coletado o tempo de resolução em cada quadro independente (quadro-a-quadro) e utilizando-se quadros diferenciais. O objetivo disso é verificar se a informação em cada bloco têm influência sobre o tempo do algoritmo de super-resolução. Em outras palavras, o teste buscou verificar se a eliminação de redundância reduz o tempo de processamento sem a necessidade do processamento guiado por contorno. Os algoritmos de super-resolução usados nos testes foram aqueles descritos no Capítulo 3 (SRvSR, SARTV e o SARL1).

A Figuras 5.5 e 5.6 ilustram o tempo de processamento em função do tamanho do bloco, para todos os blocos, com e sem a simplificação do processamento guiado por contorno. E como pode ser notado desses gráficos, o conteúdo da informação não influencia no desempenho dos algoritmos de super-resolução. Contudo, utilizando o processamento guiado por contorno, onde as regiões menos complexas são aumentas por interpolação, observa-se um

ganho no desempenho.

Considerando a Figura 5.7, que mostra o tempo médio para que um único bloco seja processado, é esperado que o tempo total de processamento ilustrado nas Figuras 5.5 e 5.6 também sejam crescentes em função do crescimento do bloco. No entanto, isso não ocorre, conforme fica claro nos gráficos do SARTV e SARL1, pois o tempo total parece decrescer com o aumento no tamanho do bloco.

Isso ocorre porque a quantidade de blocos gerados é inversamente proporcional ao tamanho desses blocos. Assim, aumentando-se a quantidade de blocos, aumenta-se o *overhead* das operações de classificação e de entrada e saída da memória. Como o método SRvSR possui ordem de grandeza maior do que SARTV e SARL1, o tempo de processamento do algoritmo é suficientemente grande para compensar o tempo gasto nas demais operações, o que caracteriza a semelhança assintótica do gráfico do método SRvSR nas Figuras 5.5, 5.6 e 5.7.

Por outro lado, os métodos SARTV e SARL1 possuem um decrescimento no tempo total de processamento porque o aumento do custo em função do tamanho do bloco compensa o custo com operações secundárias. Dessa maneira, é possível concluir que há uma configuração que melhor balanceie o custo do algoritmo de aumento de resolução com os custos das demais operações. Pelas Figuras 5.5 e 5.6, é possível notar que o valor do tempo mínimo para a maioria dos casos se dá com blocos de tamanho 32×32.

A Figura 5.8 ilustra os *speedups* da simplificação utilizando-se a classificação quadro-a-quadro e a classificação usando quadros diferenciais. Como essa figura ilustra, o ganho no tempo é tão menor quanto maior for o tamanho do bloco. Isso ocorre porque quanto menor o bloco, menor será a probabilidade dele conter uma região classificada para processar o algoritmo de maior custo. Contudo, as análises feitas nos parágrafos anteriores deve ser considerada. Assim, embora os blocos menores apresentem maior *speedup*, esses valores servem somente para ilustrar o ganho no desempenho ao se utilizar as simplificações, sendo

que o tamanho do bloco que atinge o tempo ótimo ainda é de 32×32.

Tendo esse tamanho ótimo para o bloco, é necessária uma análise do impacto da simplificação sobre as propriedades visuais dos vídeos. Ou seja, é necessário analisar os valores da relação sinal ruído (PSNR) e do índice de similaridade estrutural do vídeo (SSIM) para verificar o quanto a qualidade visual do vídeo é afetada pela abordagem simplificada. Essa análise é feita na Seção 5.1.2.

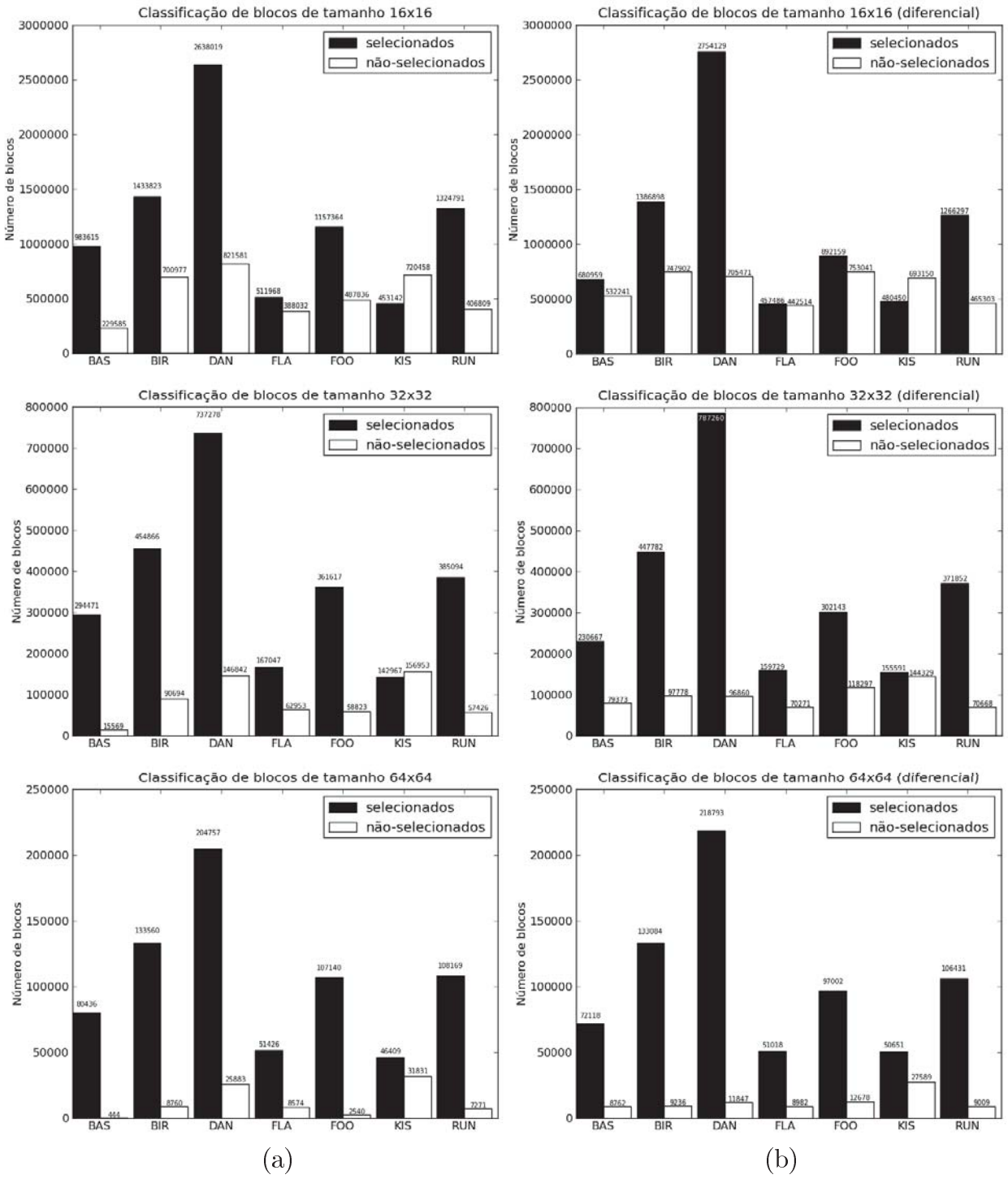


Figura 5.2: Relação dos blocos selecionados para serem processados pelo algoritmo mais custoso versus os blocos não-selecionados (simplificados), que serão processados pelo algoritmo mais simples. (a) Classificação feita quadro-a-quadro e (b) classificação feita utilizando quadros diferenciais (referencial é mantido a cada 5 quadros).

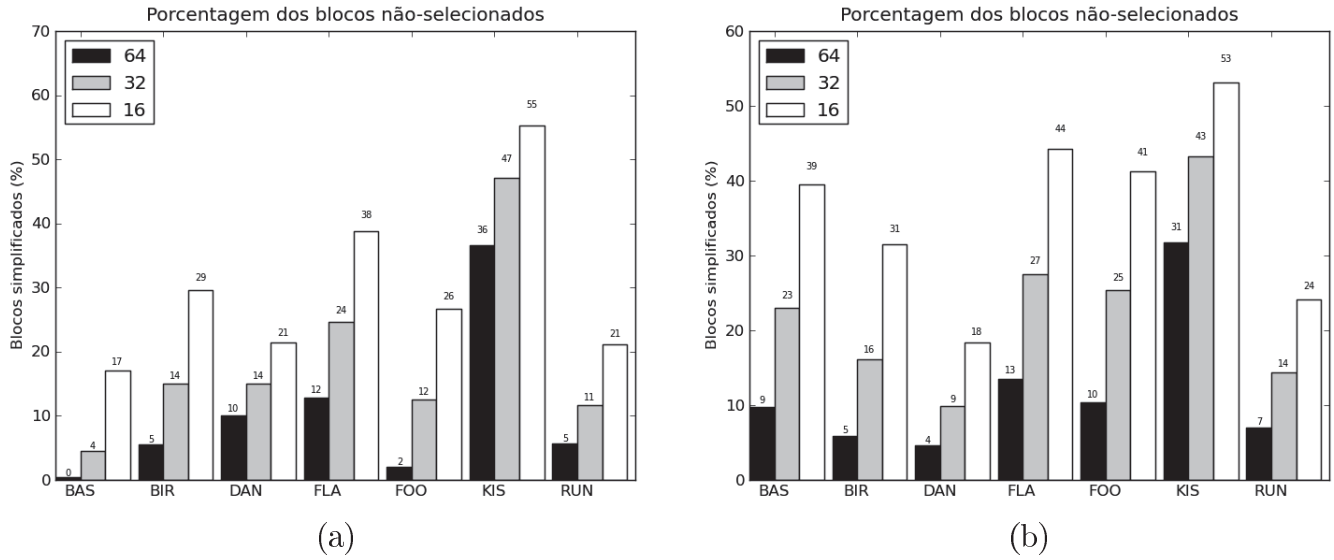


Figura 5.3: Percentagem dos blocos classificados como "não-selecionados": (a) classificação feita quadro-a-quadro e (b) classificação feita utilizando quadros diferenciais.

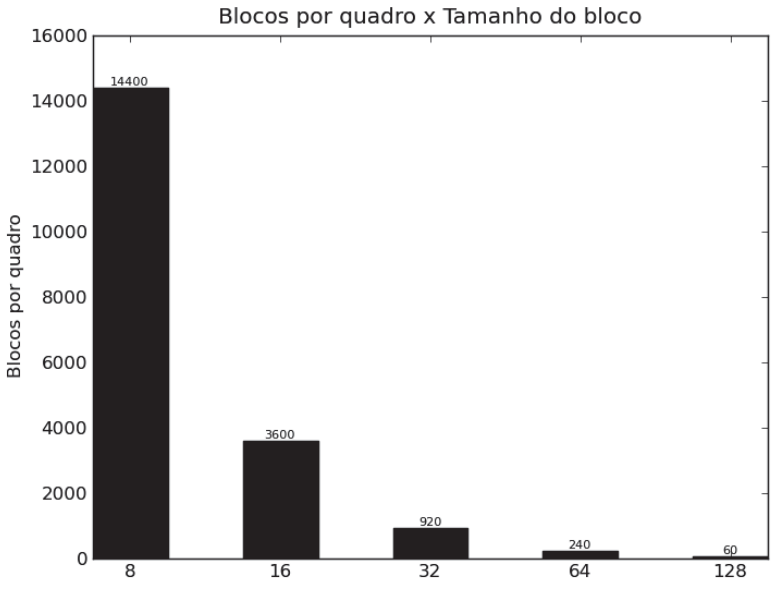


Figura 5.4: Blocos por quadros em função do tamanho dos blocos.

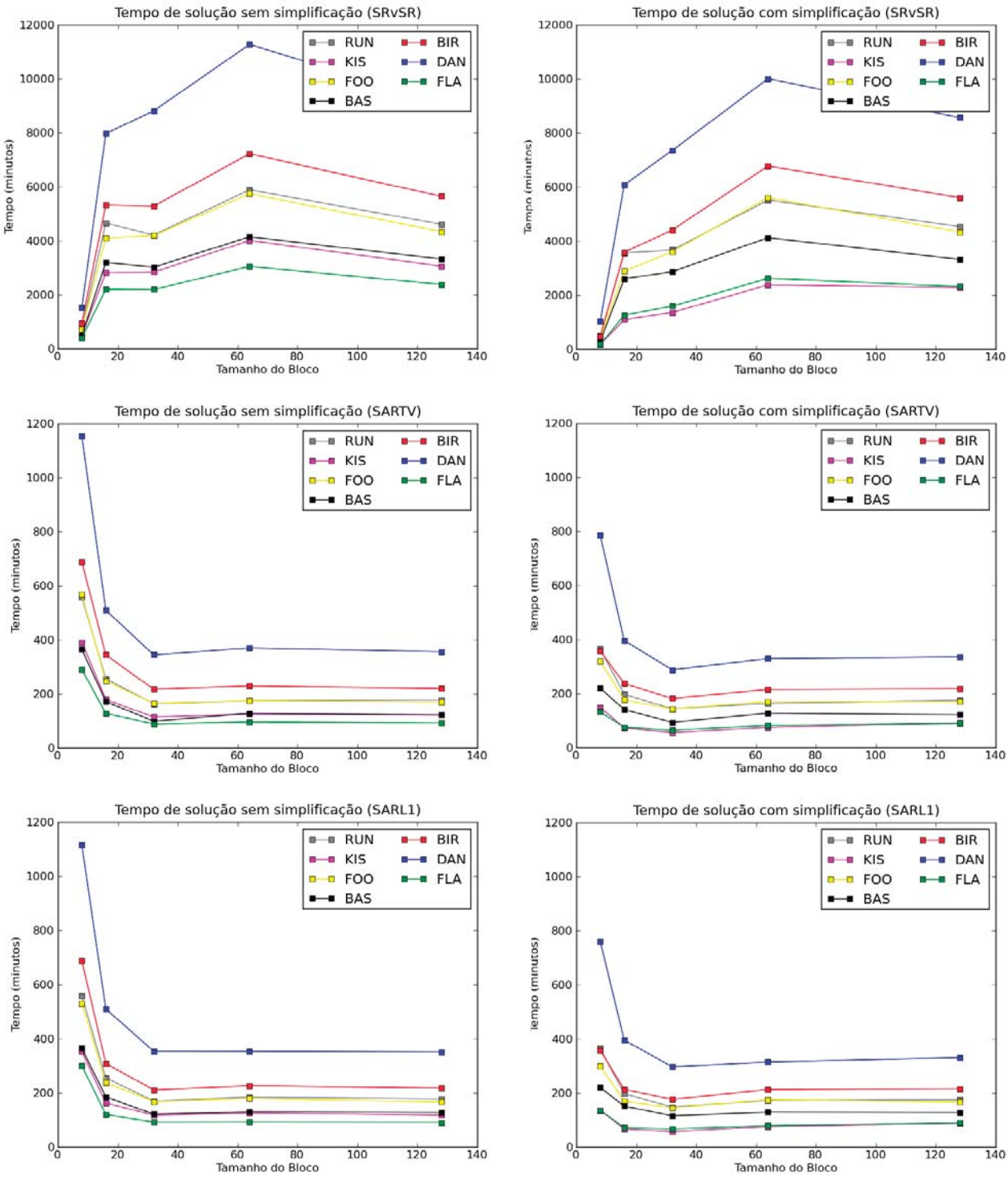


Figura 5.5: Tempo de solução de acordo com o tamanho dos quadros processados quadro-a-quadro: (a) sem simplificação e (b) com simplificação.

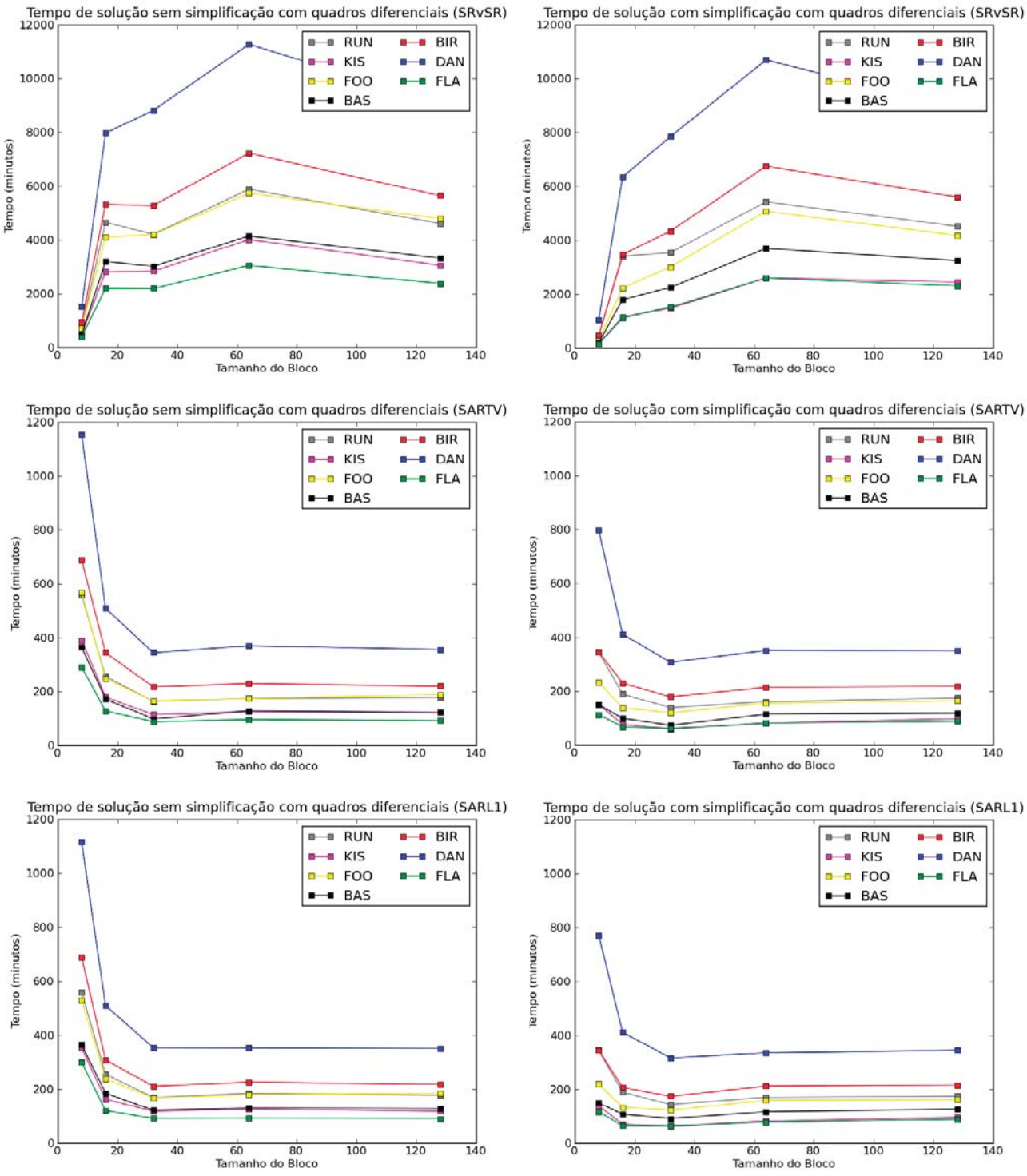


Figura 5.6: Tempo de solução de acordo com o tamanho dos quadros processados utilizando quadros diferenciais: (a) sem simplificação e (b) com simplificação.

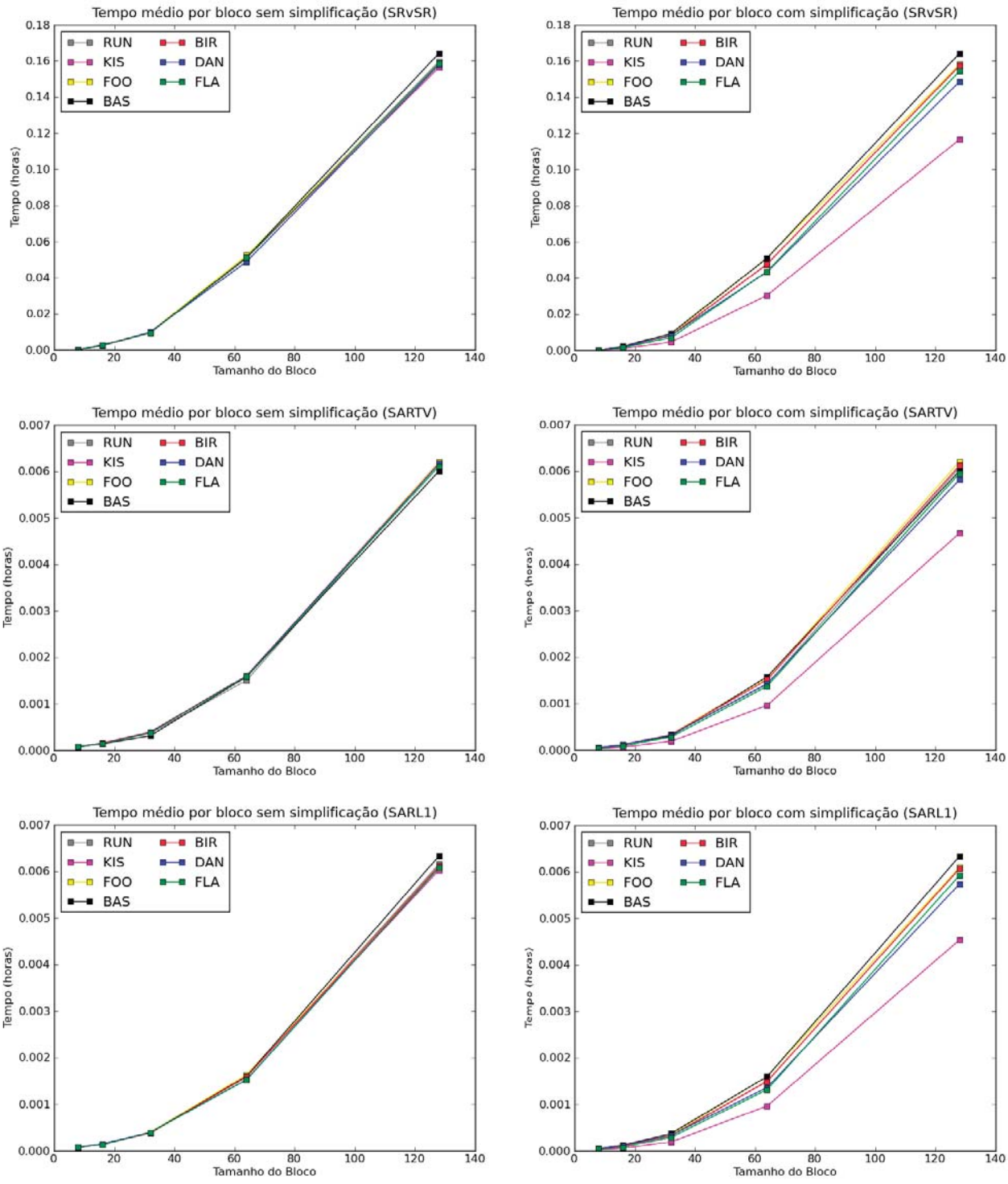


Figura 5.7: Tempo médio de processamento por bloco: (a) sem simplificação (b) com simplificação.

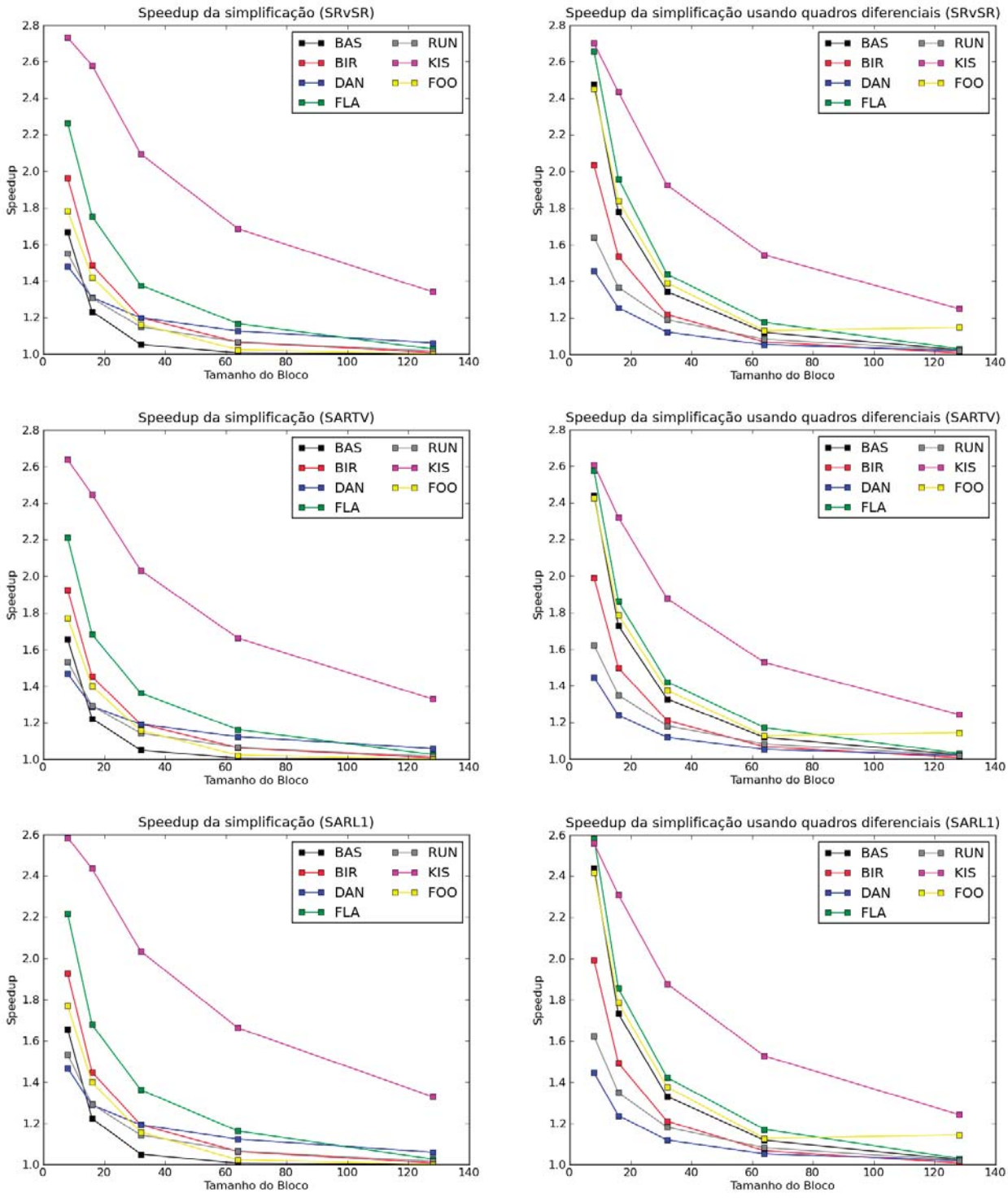


Figura 5.8: *Speedups* da simplificação. (a) Quadro-a-quadro (b) Quadros diferenciais.

5.1.2 Análise da Qualidade Visual

Sendo 32×32 o tamanho ótimo do bloco para a simplificação, é necessário verificar se esse valor corresponde a um valor aceitável de qualidade visual. Como observado na seção anterior, quanto menor o bloco, maiores serão as regiões que serão aumentadas por interpolação. Sendo assim, espera-se que a degradação seja maior devido a isso. Por isso, a métrica PSNR é utilizada. Como o PSNR é uma métrica de fidelidade, é possível medir o quanto um vídeo se degrada quando comparado com a referência original.

Por outro lado, quando menor o bloco, mais "inteligente" é a aplicação do algoritmo de super-resolução, uma vez que seleciona mais regiões de detalhes (heterogêneas) e mais regiões homogêneas. Com essa diferente classificação, é possível que mais regiões homogêneas, aumentadas por interpolação, não tenham tanto impacto visual, ainda que o PSNR dessa região seja menor do que seria se essas regiões fossem aumentadas por outro método. Sendo assim, a segunda métrica escolhida para a análise é o SSIM. A utilização dessa segunda métrica se dá porque ela é projetada para produzir valores de comparação melhores que o PSNR quando considera-se a percepção visual humana.

Para ilustrar essa diferença, a Figura 5.9 lista dois blocos e suas respectivas reconstruções utilizando diferentes métodos de ampliação. Os blocos usados para teste possuem duas características muito distintas entre si. Enquanto os blocos da primeira linha caracterizam-se por serem totalmente homogêneos, os blocos da segunda linha dessa figura possuem uma informação totalmente heterogênea.

Comparando-se os blocos de referência da Figura 5.9-(a) com os demais, é possível notar que para a informação homogênea, há pouca diferença visual entre o original e os blocos aumentados por meio dos algoritmos de super-resolução. Além disso, o bloco aumentado utilizando-se interpolação difere dos demais apenas por um pequeno efeito de borda, quase imperceptível.

Em contrapartida, ao comparar-se os blocos da segunda linha da Figura 5.9, é possível

notar que o resultado visual difere de maneira significante do bloco de referência, dependendo do algoritmo de resolução usado. Sendo assim, os valores das métricas objetivas desses blocos reconstruídos são listados na Figura 5.10.

Assim, a partir dessa figura, é possível notar que os valores de PSNR divergem bastante para a imagem homogênea, quando são comparados os valores entre a ampliação pelos métodos de super-resolução e o de interpolação. Já para os valores da imagem heterogênea, esses valores não são muito distintos. Contudo, esses valores contradizem o observado, pois as imagens da primeira linha da Figura 5.9 são visualmente muito mais semelhantes entre si do que as imagens da segunda linha. Portanto, a diferença entre os valores do PSNR deveria ser muito maior para as imagens com informação heterogênea do que para aquelas com informação homogênea.

Contrariamente a esse comportamento do PSNR, o SSIM produz valores muito mais coerentes, conforme mostrado na Figura 5.10-(b). Nesse gráfico, é possível notar que a divergência dos valores do SSIM da imagem homogênea aumentada por interpolação, quando comparada com os métodos de super-resolução, é muito menor do que a divergência da imagem imagem heterogênea aumentada utilizando-se esse método. Isto posto, o SSIM é utilizado para mensurar a qualidade visual do vídeo de acordo com a variação dos parâmetros, sendo esse método o mais coerente para mensuração no contexto deste trabalho.

O cálculo do PSNR dos vídeos foi feito *pixel-a-pixel*. Isto é, concatenando-se cada linha dos quadros em uma linha única e tratando-se os vídeos como um sinal unidimensional. Os valores produzidos foram calculados utilizando-se a Equação 2.7.

No caso do SSIM, essa transformação feita na estrutura do sinal para o cálculo do PSNR poderia comprometer as propriedades psico-visuais que essa métrica utiliza. Dessa forma, os índices de similaridade estrutural foram calculados para cada quadro de forma independente e, em seguida, foi calculada a média dos valores de SSIM de cada quadro.

Além disso, visando obter uma referência de comparação do impacto da simplificação,

primeiramente as métricas foram calculadas para os vídeos ampliados sem simplificação. Ou seja, todos os quadros foram ampliados independentemente, sem dividi-los em blocos e sem utilizar diferentes algoritmos para diferentes regiões. A Figura 5.11 apresenta os valores do PSNR e SSIM utilizando essa abordagem. Os resultados dessa figura foram produzidos reduzindo-se as dimensões dos vídeos de referência de 720p pela metade. Dessa forma, os vídeos de referência reduzidos para dimensões de 640×360 foram ampliados para as dimensões originais de 1280×720 utilizando os diferentes algoritmos comparados.

De forma semelhante, para os vídeos ampliados considerando as simplificações, a proporção de redução e ampliação por um fator de dois foi escolhida como padrão para os testes. Sendo assim, a Figura 5.12 ilustra os resultados do teste da qualidade para os diferentes algoritmos. Nessa figura, é possível constatar que, como suposto, a qualidade do vídeo aumenta de acordo com o aumento no tamanho do bloco. Observando os valores de PSNR é possível observar esse comportamento. Assim, quanto maior o bloco, menor é a quantidade de regiões aumentadas por interpolação, sendo o resultado final mais fiel aos valores dos *pixels* do vídeo de referência. Como o PSNR é uma métrica de fidelidade, esse comportamento é totalmente coerente.

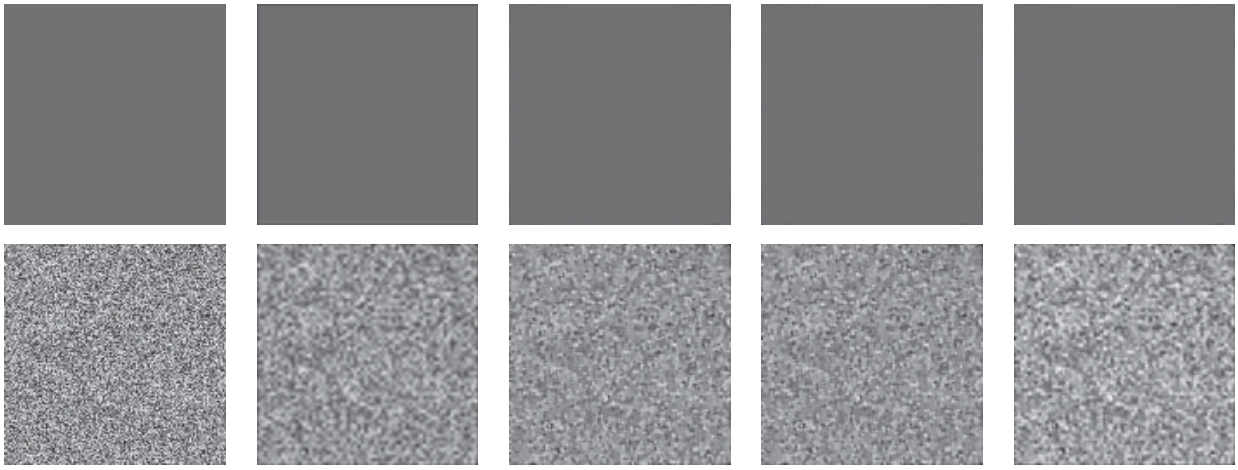
Ainda na Figura 5.12, é possível notar que os valores de SSIM dos vídeos analisados também crescem em função do tamanho do bloco. Porém, diferente do que ocorre com o PSNR, os gráficos com essa métrica possuem um crescimento muito mais homogêneo e suave. Dado esse comportamento, os gráficos da Figura 5.13 ilustram a proporção das métricas dos vídeos simplificados em relação às métricas dos vídeos não-simplificados da Figura 5.11. É possível concluir que grandes variações nos valores do PSNR não implicam necessariamente em grandes variações no SSIM. Portanto, em termos de resultados visuais, ainda que um vídeo seja pouco fiel à referência (pela escala do PSNR), as distorções e artefatos das regiões interpoladas são pouco perceptíveis.

Para ilustrar esse comportamento, a Figura 5.14 exibe o resultado visual de diferentes

técnicas usadas para aumentar o primeiro quadro de um dos vídeos usados nos testes. A Figura 5.14-(a) é o quadro de referência. Ou seja, o quadro original que foi reduzido e depois ampliado para as dimensões originais. A Figura 5.14-(b) é o resultado da ampliação de todo o quadro utilizando-se somente interpolação. A Figura 5.14-(c) é o resultado da ampliação utilizando-se o algoritmo SARTV em todo o quadro, sem nenhuma simplificação. Finalmente, a Figura 5.14-(d) é o resultado da ampliação utilizando-se simplificação, onde os blocos das regiões de contorno e heterogêneas são ampliadas com o algoritmo SARTV e os blocos das regiões homogêneas são ampliados com interpolação. Além disso, a Figura 5.14-(d) foi gerada com blocos de 8×8, tamanho esse que resultou nos piores resultados de PSNR e SSIM, conforme mostrados na Figura 5.12.

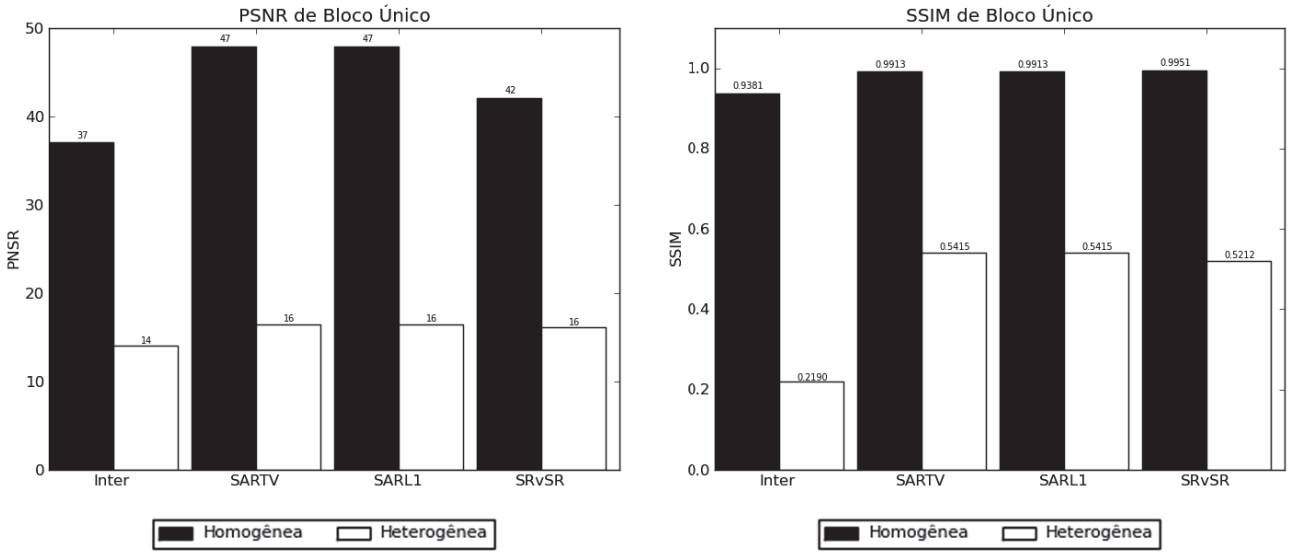
Dessa forma, embora exista uma diferença considerável dos valores do PSNR medidos para as Figuras 5.14-(c) e (d), é possível notar que visualmente é quase imperceptível perceber as diferenças entre essas imagens. Para reforçar essa ideia, a Figura 5.15 ilustra o resultado do mesmo quadro do vídeo FOO ampliado, considerando-se as técnicas de simplificação, com blocos de diferentes tamanhos.

Dessa forma, resultados visuais são pouco perceptíveis ao se variar o tamanho dos blocos, conforme ilustrado na Figura 5.15. Além disso, considerando a Figura 5.13, que mostra como os valores do SSIM passam a diferir pouco entre os blocos maiores que 32×32, é coerente usar esse tamanho de bloco, uma vez que ele minimiza o tempo do processamento simplificado.



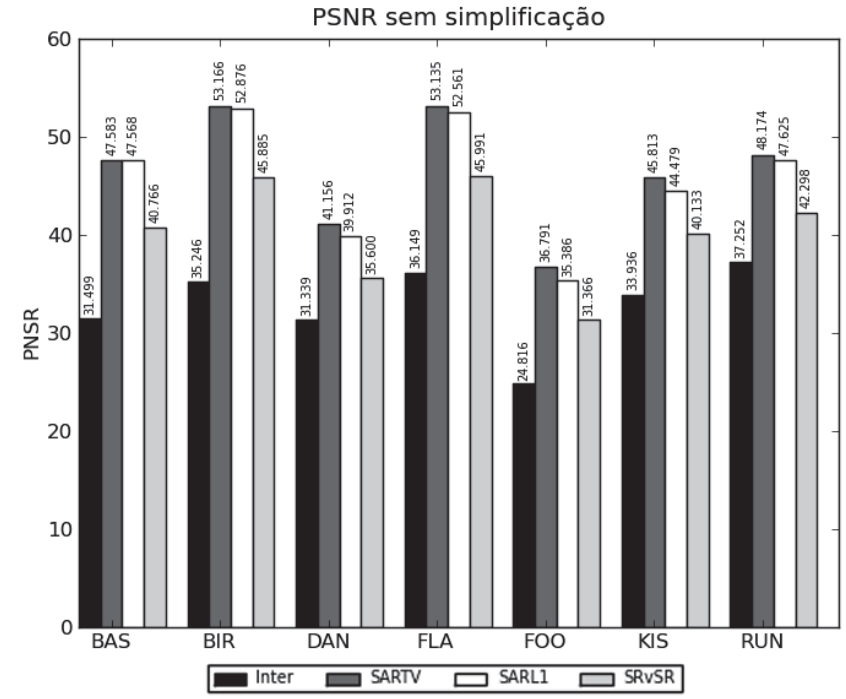
(a) (b) (c) (d) (e)

Figura 5.9: Blocos de 128×128 aumentados utilizando diferentes algoritmos: (a) original, (b) interpolação bilinear, (c) SARTV, (d) SARL1 e (e) SRvSR.

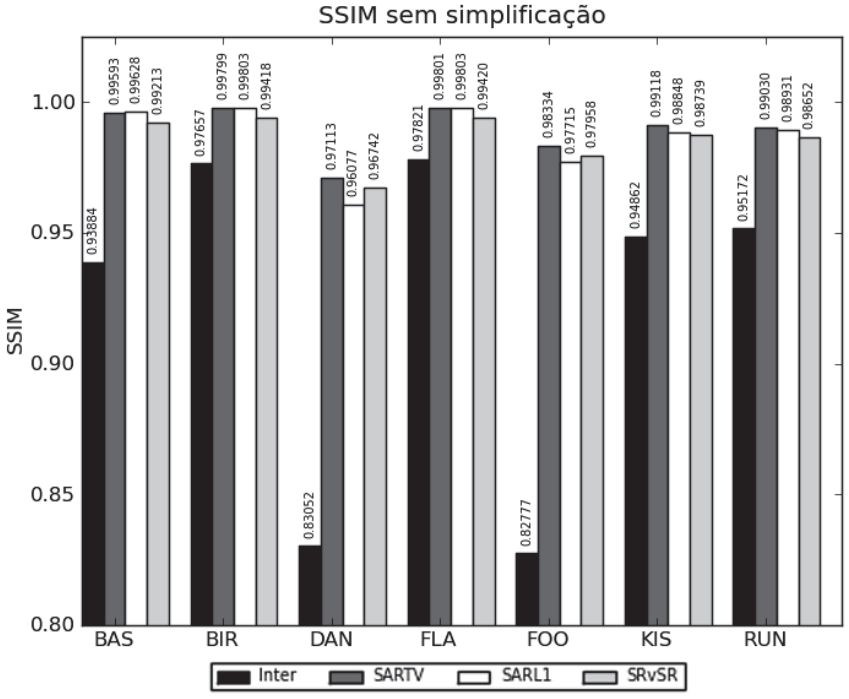


(a) (b)

Figura 5.10: Métricas dos blocos homogêneos e heterogêneos: (a) PSNR e (b) SSIM.



(a)



(b)

Figura 5.11: Métricas dos vídeos aumentados utilizando interpolação, SARTV, SARL1 e SRvSR sem simplificação. (a) PSNR (b) SSIM.

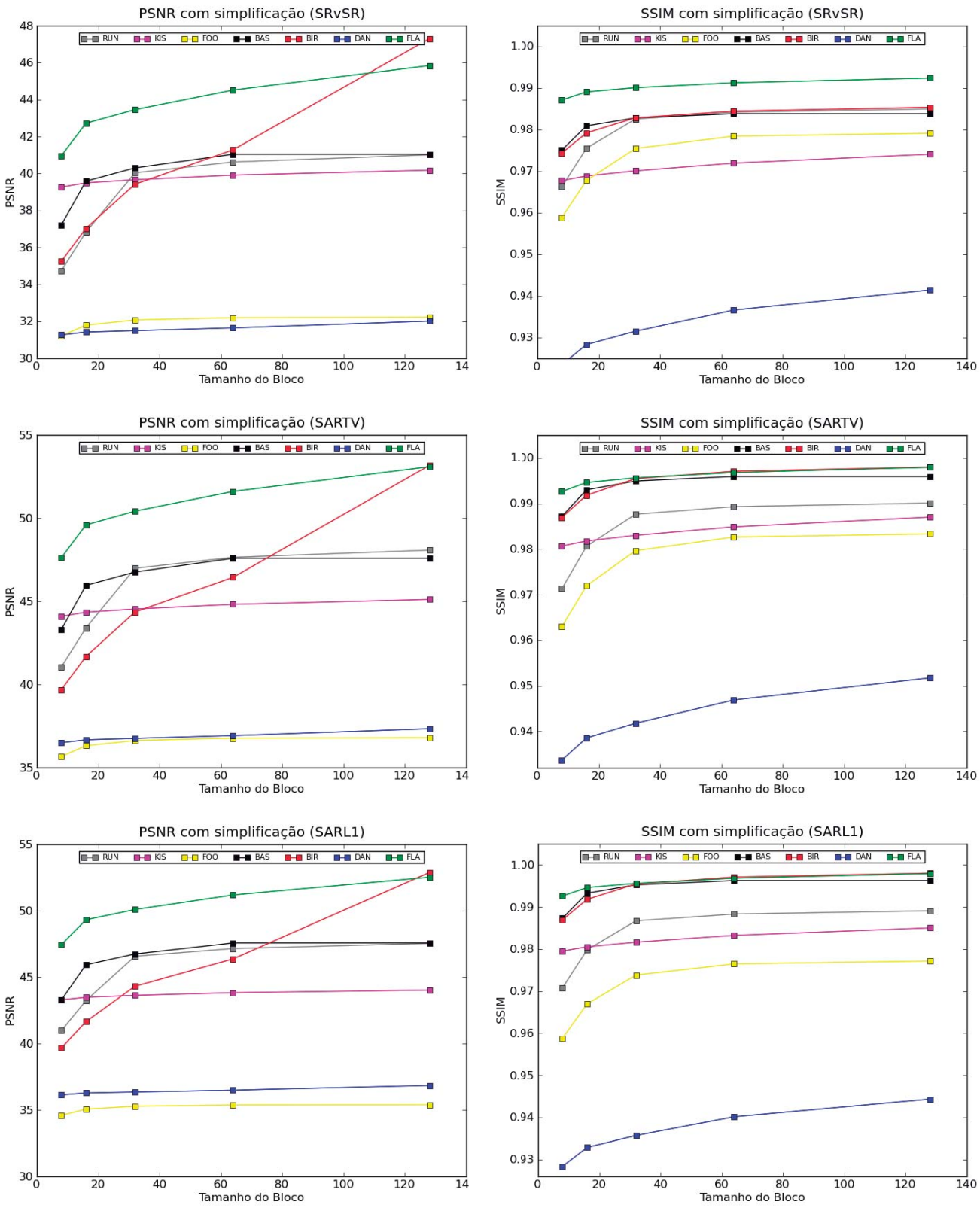


Figura 5.12: Qualidade dos vídeo utilizando simplificação em função do tamanho do bloco: (a) PSNR e (b) SSIM.

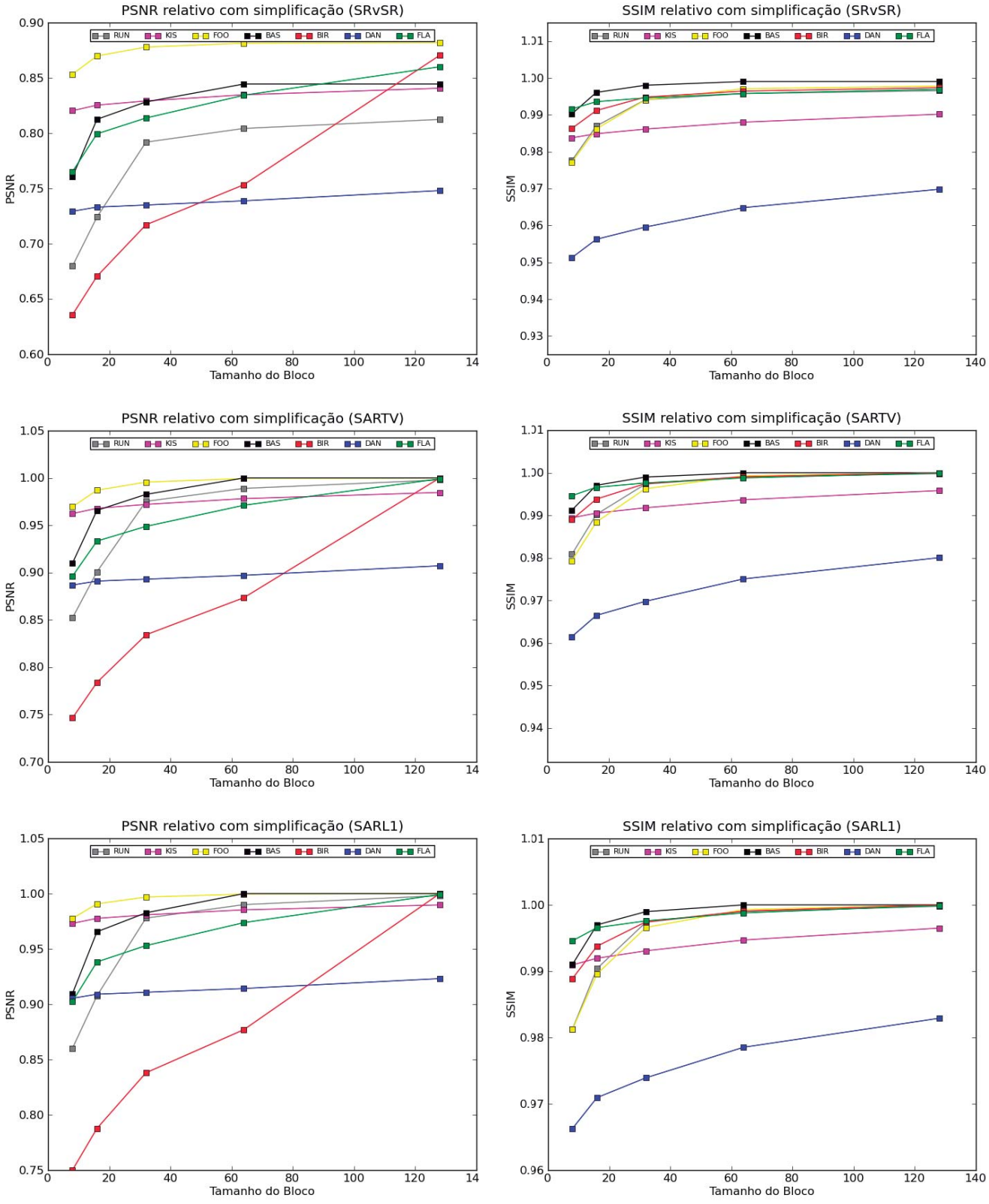


Figura 5.13: Valores relativos às métricas sem simplificação com uso da simplificação: (a) PSNR e (b) SSIM.

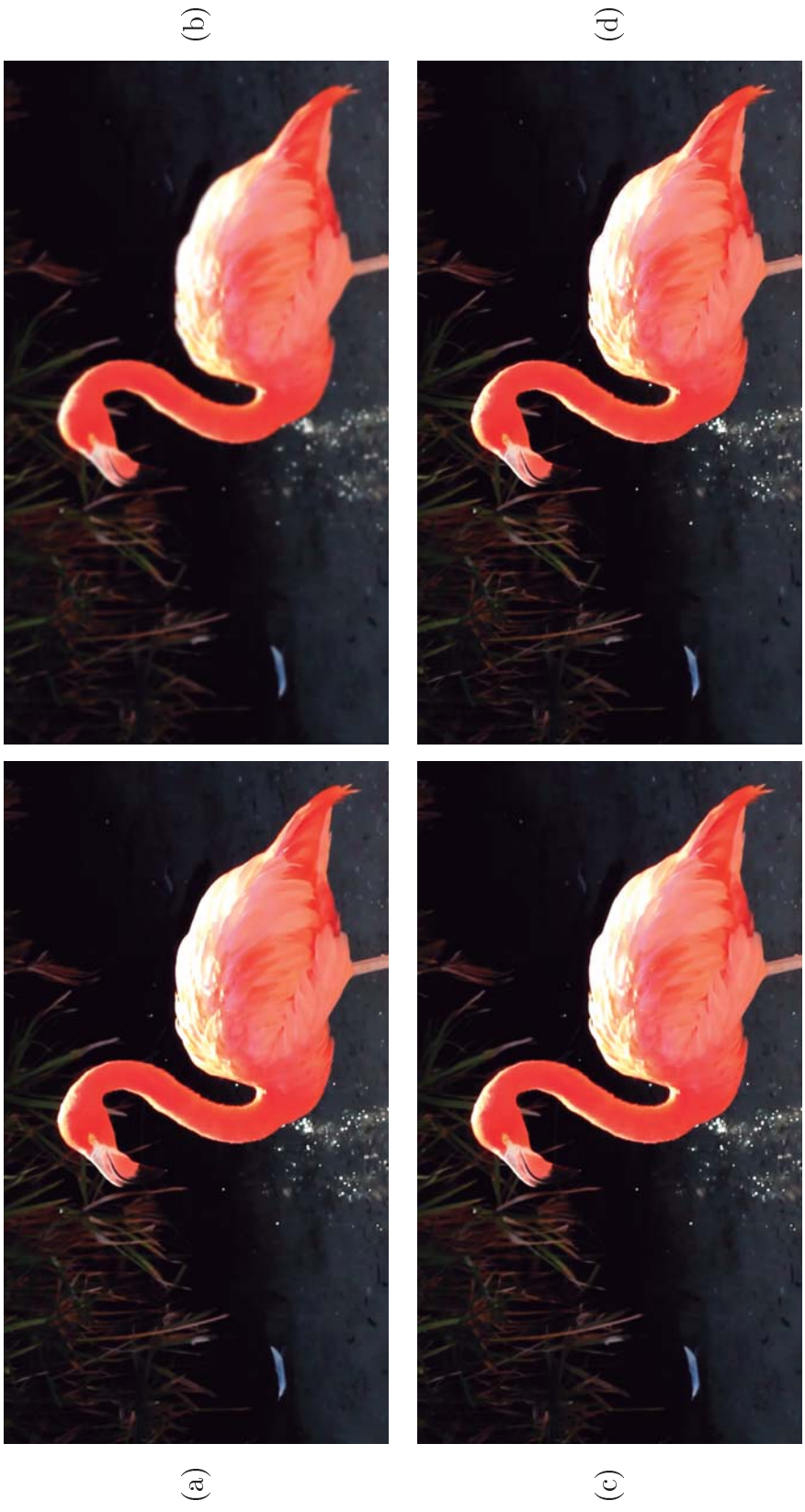


Figura 5.14: Resultado visual do aumento com simplificação: (a) original (referência), (b) quadro aumentada por interpolação, (c) quadro aumentado com super-resolução e (d) quadro aumentado com a combinação de blocos interpolados e super-resolução (blocos 16×16).

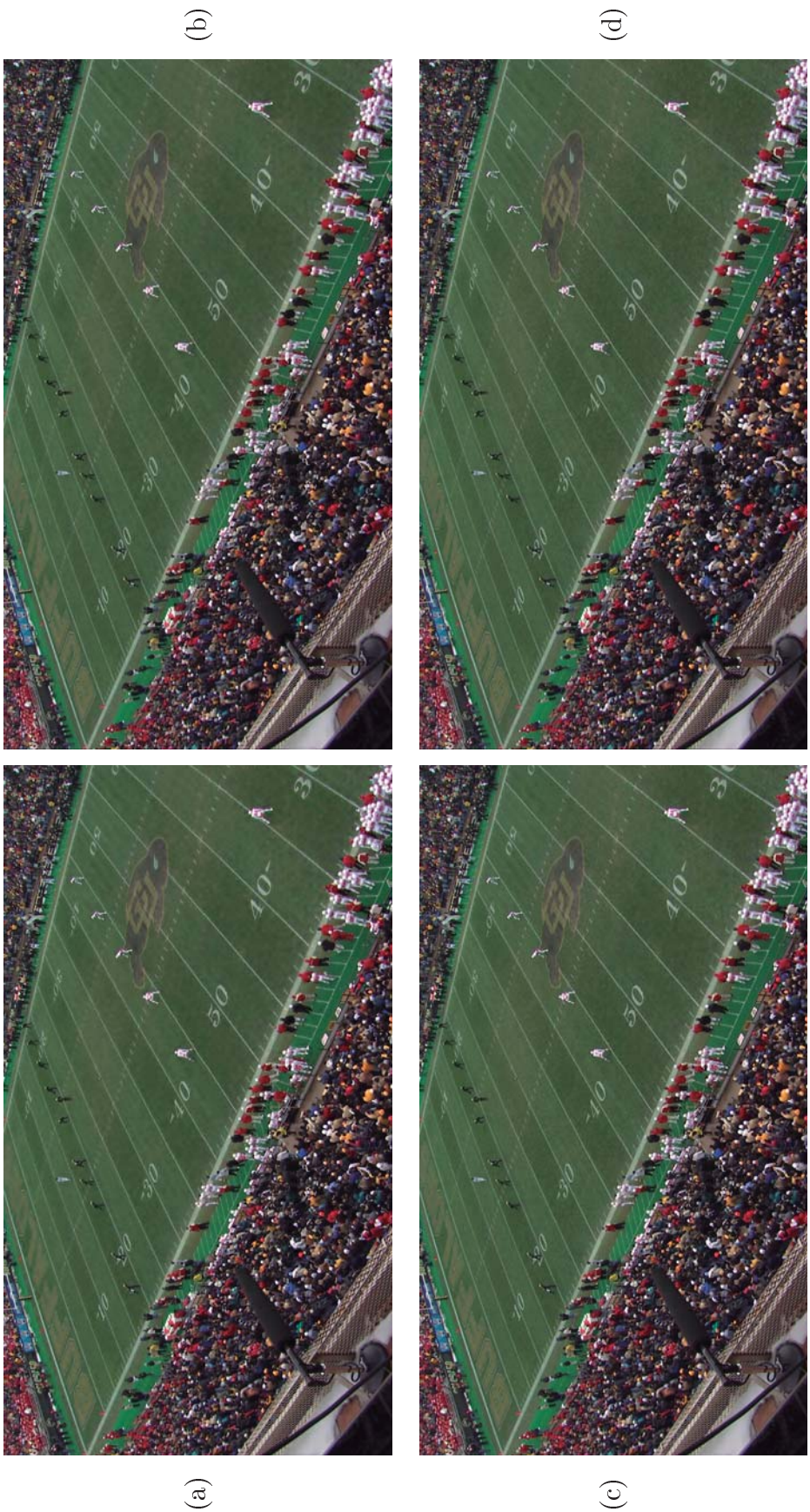


Figura 5.15: Resultado visual do aumento simplificado com blocos de tamanho (a) 8×8, (b) 16×16, (c) 32×32 e (d) 64×64.

5.1.3 Análise do Desempenho do *Framework* Paralelo

Após a análise das informações sobre o comportamento da qualidade e da eficiência da simplificação, resta analisar o ganho do desempenho com a execução paralela do *framework*. Para isso, é possível realizar diversas simulações, variando múltiplos parâmetros dos algoritmos envolvidos e mensurando o tempo de execução para cada uma das variações. Contudo, quando o objetivo central da análise é o desempenho da paralelização, é mais interessante fixar os demais parâmetros envolvidos e avaliar o tempo em função do número de processos.

Sendo assim, conforme exposto na Seção 5.1.1, o tamanho de bloco que melhor balanceia o tempo de processamento da simplificação é de 32×32. Além disso, como foi analisado na Seção 5.1.2, sabe-se que para blocos maiores que 32×32 o aumento dos valores de SSIM é mínimo, significando uma melhoria visual quase imperceptível. Portanto, com essas considerações, é plausível fixar o tamanho dos blocos nessas dimensões, uma vez que resulta em um melhor equilíbrio entre tempo de simplificação e qualidade dos resultados visuais.

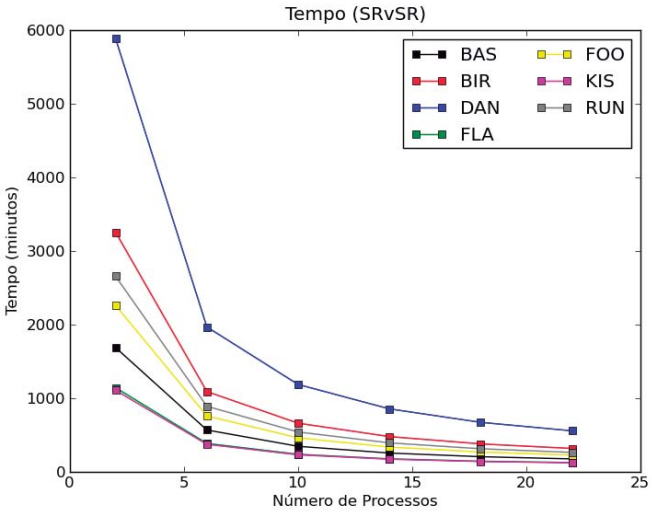
Como as simulações foram executadas em um máquina contendo quatro processadores Intel Xeon X5690, que contabilizam um total de 24 núcleos, executou-se o programa variando entre 2 e 24 o número de processos executando em paralelo. Em cada uma dessas execuções, foi mensurado o tempo gasto de resolução do algoritmo para os blocos fixados no tamanho 32×32 e considerando todas as simplificações, conforme ilustra a Figura 5.16.

De posse do tempo da solução serial, apresentado na Figura 5.6, foi calculado o *speedup* da implementação paralela. Tais valores são ilustrados na Figura 5.17. Com os valores de *speedup* apresentados nessa figura, divide-se pelo número de processos correspondentes para obter a eficiência do *framework*, conforme ilustrado na Figura 5.18.

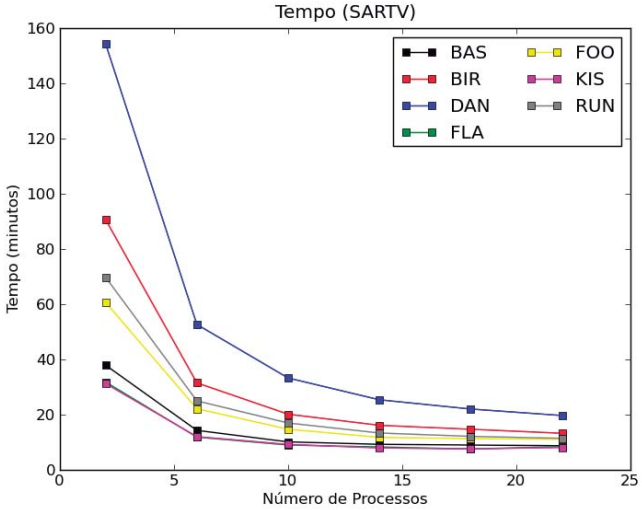
Analisando a Figura 5.6, que contém o tempo em função do número de processos, é possível perceber que o *framework* proposto obteve um significante ganho de desempenho

com o processamento paralelo. Tomando, por exemplo o gráfico do vídeo DAN, é possível notar que o tempo de processamento caiu de quase 6000 minutos para menos de 1000 minutos, utilizando-se o método SRvSR, representando um ganho de aproximadamente 84 horas (3,5 dias).

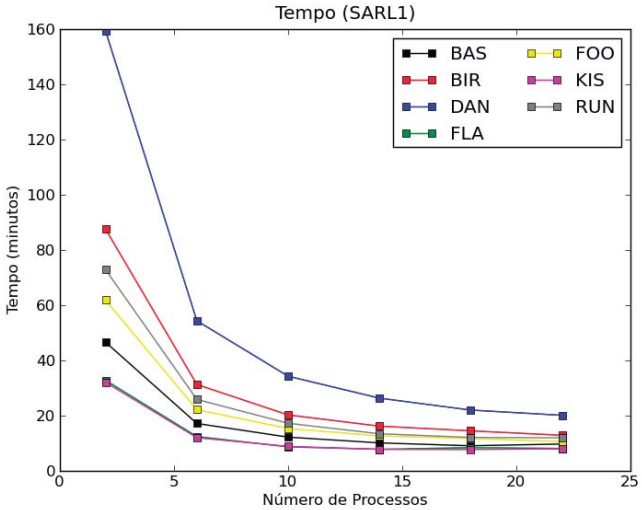
A Figura 5.17 também ilustra um padrão interessante. Como é possível notar dessa figura, o *speedup* se manteve crescente em função do aumento do aumento do número de processos executando em paralelo na maioria dos casos. Assim, embora o *speedup* não tenha se comportado muito próximo do comportamento ideal (linear com $speedup(n) = n$), é possível notar que o ganho de performance ainda é bem relevante.



(a)

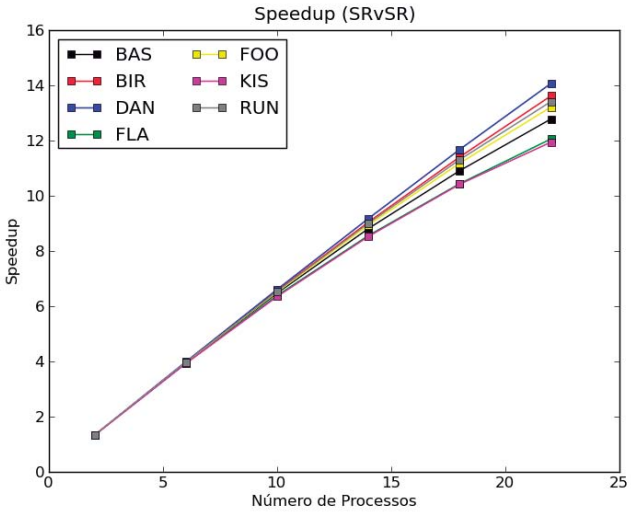


(b)

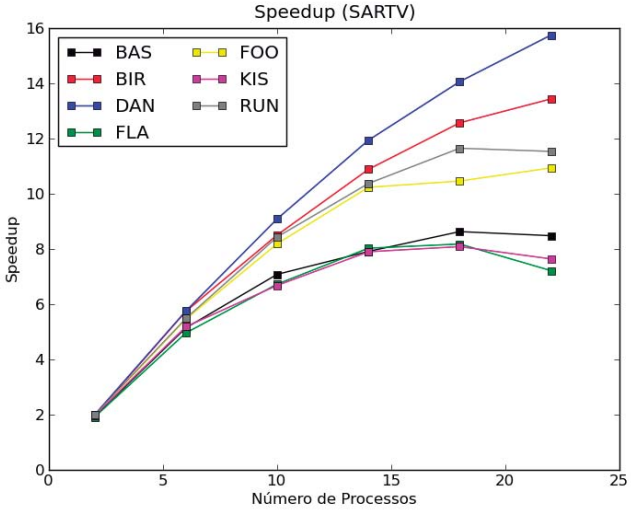


(c)

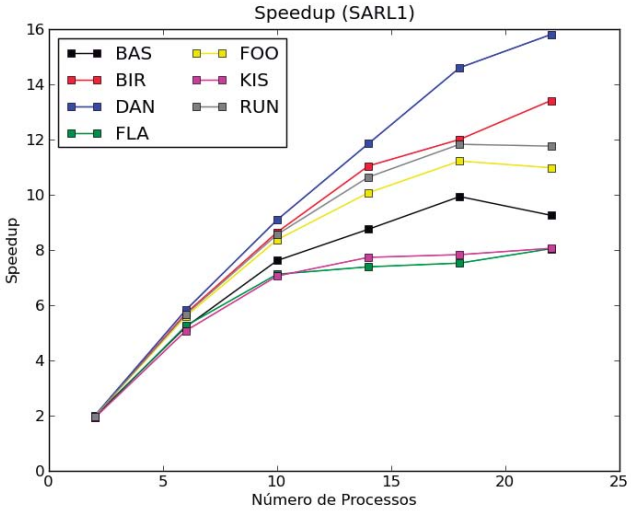
Figura 5.16: Tempo de execução dos algoritmos em paralelo em função do número de processos paralelos: (a) SRvSR, (b) SARTV e (c) SARL1.



(a)

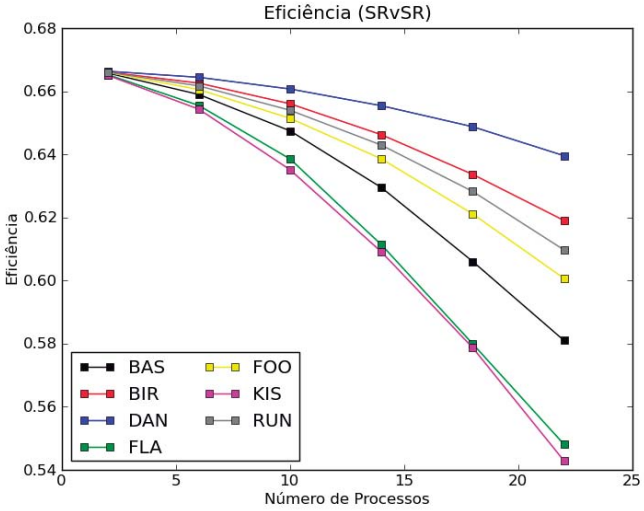


(b)

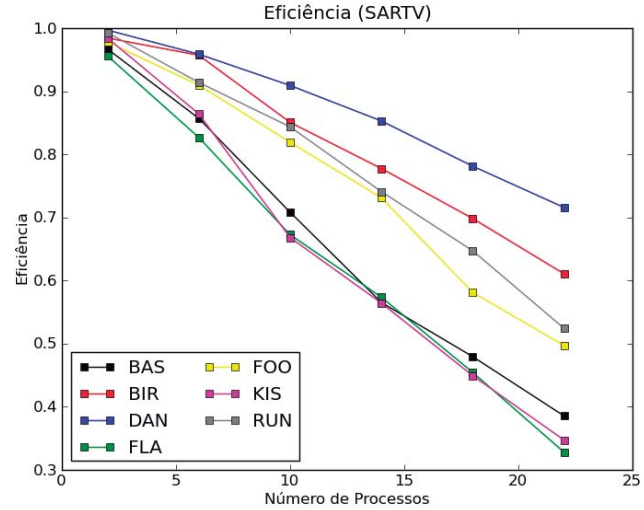


(c)

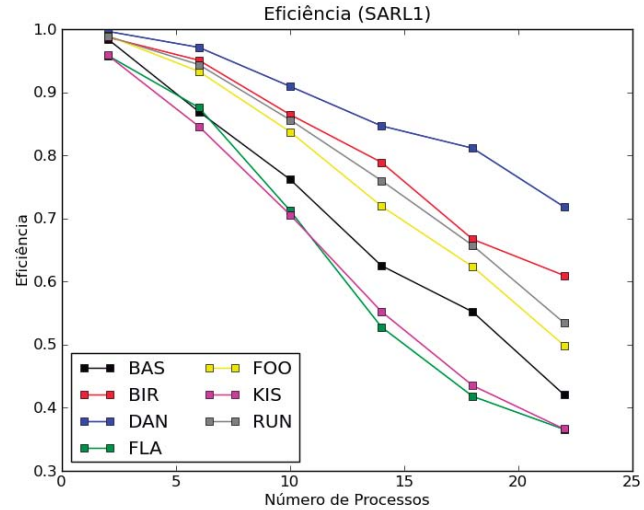
Figura 5.17: *Speedup* em função do número de processos paralelos: (a) SRvSR, (b) SARTV e (c) SARL1.



(a)



(b)



(c)

Figura 5.18: Eficiência em função do número de processos paralelos: (a) SRvSR, (b) SARTV e (c) SARL1.

5.2 Considerações Finais

Nesse capítulo foi feita a análise do *framework* sob os aspectos do desempenho da simplificação e como ela impacta na qualidade do resultado visual produzido. Conforme discutido na Seção 5.1.1, tem-se que a abordagem simplificada fornece uma relevante economia de recursos computacionais, uma vez que isola partes do sinal de vídeo e evita que elas sejam processadas pelos custosos algoritmos de super-resolução. Dessa seção é possível notar que diferentes métodos tendem a se comportar de maneira diferente de acordo com o tamanho do bloco escolhido. Contudo, concluiu-se que um tamanho adequado de bloco possui tamanho 32×32, que foi fixado posteriormente nas simulações que utilizaram processamento paralelo.

Tendo verificado a economia no tempo pela utilização da simplificação, o próximo passo consistiu em analisar como a qualidade do vídeo ampliado se comporta de acordo com a quantidade de regiões descartadas do processo de super-resolução. Assim, essa análise foi feita na Seção 5.1.2, onde foi possível perceber que a métrica de análise consiste em um importante fator na determinação da eficácia da simplificação.

Essa seção iniciou introduzindo uma pequena simulação, onde duas imagens contendo informações semanticamente opostas (uma sendo totalmente homogênea e a outra totalmente heterogênea) são ampliadas utilizando-se os diferentes métodos testados. Em seguida, cada uma dessas imagens foi comparada com a referência original para verificar a resposta gerada pelas métricas PSNR e SSIM. O resultado foi que o PSNR forneceu valores menos coerentes com o que foi observado visualmente, enquanto que o SSIM forneceu valores mais coerentes, o que justificou o pressuposto critério adotado na classificação, o que faz com que somente as regiões mais heterogêneas do quadro sejam processadas pelos algoritmos de super-resolução.

Finalmente, na Seção 5.1.3 são discutidos os resultados do processamento paralelo do

framework. Embora esse processamento não tenha impacto na qualidade, é de grande importância para viabilizar o aumento dimensional de vídeos que possuam muitos quadros. Conforme mostrado nessa seção, o *framework* proposto possui uma escalabilidade razoável, sendo uma estratégia de paralelização que pode ser adotada para diversas aplicações.

Capítulo 6

Conclusões

Neste trabalho foi apresentado um novo *framework* para processar de forma paralela algoritmos de super-resolução, visando aumentar as dimensões espaciais de sinais de vídeo. A abordagem apresentada segmenta e distribui os dados do vídeo de forma que a estratégia de paralelização e o método de super-resolução sejam independentes. Dessa forma, o *framework* permite que diferentes algoritmos de super-resolução possam aproveitar a estratégia apresentada neste trabalho, uma vez que a paralelização não requer modificações específicas nesses algoritmos.

Além disso, este trabalho propôs uma estratégia de simplificação a qual permite que partes menos evidentes da informação visual não sejam processadas pelos algoritmos de super-resolução. Conforme exposto no Capítulo 5, é possível notar que esse tipo de algoritmo é computacionalmente custoso, tendo comportamento exponencial em função do tamanho das dimensões espaciais do quadro a ser processado. Assim sendo, essa estratégia de simplificação permite economizar recursos computacionais, uma vez que processa uma quantidade reduzida de dados, quando comparada com a abordagem que processa todo o sinal utilizando algoritmos de super-resolução.

Ainda no Capítulo 5, quando comparados os quadros ampliados por super-resolução

com os quadros ampliados por interpolação, é possível notar a diferença entre eles, pois a interpolação gera um resultado com evidente distorção (borramento, no caso da interpolação bilinear). Por outro lado, quando comparam-se os quadros produzidos totalmente por super-resolução com aqueles produzidos utilizando partes interpoladas e partes com super-resolução (simplificados), a diferença é menos perceptível. Esse resultado é coerente, principalmente, quando é utilizada a análise feita dos valores do SSIM que, conforme explicado no Capítulo 2, considera elementos do sistema visual humano na percepção de distorções.

Dessa forma, é importante ressaltar que os resultados deste trabalho são aproveitáveis em diversas aplicações. A aplicação mais evidente que poderia aproveitar o método proposto consiste no simples aumento de resolução de vídeos. Exemplos práticos desse tipo de aplicação são os populares serviços de hospedagem de vídeo *online*, tais como o Youtube [4], o Vimeo [3] e Dailymotion [1]. Todos esses sistemas tem como ponto em comum hospedar e oferecer vídeos enviados por diversos usuários. Contudo, para oferecer seu conteúdo em alta-resolução, esses serviços dependem que o usuário submeta o conteúdo dessa forma. Todavia, utilizando o método proposto neste trabalho, serviços como esses poderiam processar e fornecer vídeos em alta-resolução, ainda que os usuários submetam conteúdos em resolução menor.

Aplicações e algoritmos que utilizam métodos de super-resolução como etapa secundária também podem se beneficiar do *framework* proposto. Por exemplo, no artigo *Fast Inverse Halftoning Algorithm for Ordered Dithered Images* [32], o autor deste trabalho propõe um método para aproximar os níveis de cor da imagem original a partir de uma versão em meio-tom com qualidade visual superior à outros métodos semelhantes. Contudo, um dos principais problemas desse método é que ele aproxima os níveis de cor da imagem original em uma imagem reconstruída com dimensões menores. Então, para escalar a imagem reconstruída para as dimensões da imagem original, esse método utiliza interpolação, o

que acaba inserindo um nível de degradação. Dessa forma, a utilização de algoritmos de super-resolução para aumentar as dimensões da imagem, pode permitir uma reconstrução com maior qualidade.

No contexto de vídeos, essa técnica de *inverse halftoning* descrita no parágrafo anterior possui outras aplicações. Um exemplo do aproveitamento desse algoritmo em vídeos é apresentado no artigo de mesma autoria, intitulado *Error Concealment Using a Halftone Watermarking Technique* [33]. Nesse artigo, é apresentada uma técnica para encobrimento (mitigação) dos erros gerados pela perda de pacotes do sinal de vídeo devido à problemas na transmissão. Assim, mesmo que o vídeo seja danificado durante uma transmissão na rede, os dados perdidos não danificam o resultado reproduzido, pois os erros são encobertos de forma que o expectador que recebe o sinal não perceba os artefatos gerados.

Assim sendo, como apresentado nesses exemplos, o problema de aumento dimensional tem relação com diversas aplicações. Logo, o *framework* pode ser utilizando como proposto ou modificado com diferentes objetivos.

Uma modificação a ser feita e analisada em trabalhos futuros pode consistir em otimizar o critério de seleção na etapa de simplificação. No Capítulo 4, o critério proposto foi processamento guiado por contorno, que utiliza um filtro de Canny para realçar as bordas e classificar as regiões que contenham essas bordas como sendo de maior relevância.

Contudo, sabe-se que o sistema visual humano não observa as regiões de imagens e vídeos de forma aleatória. Ao contrário, o sistema cognitivo humano tende a direcionar a atenção para informações específicas dentro dos sinais visuais, conforme apresenta Nadenau *et al.* [64]. Assim, uma possível abordagem para reduzir as regiões a serem processadas pelos algoritmos de super-resolução é utilizar um algoritmo que modele o sistema visual humano. Dessa forma, ao invés dos algoritmos de super-resolução redimensionarem todas as regiões de bordas, eles podem ser utilizados somente nas regiões que são direcionadas pelo processo de percepção visual, enquanto todas as outras regiões são aumentadas por

interpolação.

Conforme descrito no parágrafo anterior, melhorias na produção de resolução variável proposta para esse *framework* ainda podem ser investigadas. Além disso, como outras aplicações podem ser beneficiadas pelos algoritmos de aumento de resolução de imagens, o *framework* apresentado nesse trabalho surge como uma importante contribuição para favorecer o desempenho desses algoritmos na utilização em vídeos.

Referências

[1] Dailymotion. `http://www.dailymotion.com/`, January 2013. [Online; accessed 31-Jan-2013]. 118

[2] Using Intel® Streaming SIMD Extensions and Intel® Integrated Performance Primitives to Accelerate Algorithms. `http://goo.gl/c1HhU`, 2013. [Online; accessed 31-Jan-2013]. 16

[3] Vimeo. `http://vimeo.com/`, January 2013. [Online; accessed 31-Jan-2013]. 118

[4] Youtube. `http://www.youtube.com/`, January 2013. [Online; accessed 31-Jan-2013]. 118

[5] G. S. Almasi and A. Gottlieb. *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989. 11

[6] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 1967 (Spring Joint Computer Conference)*, 23(4):483–485, 1967. 29

[7] Cristiana Amza, Alan L. Cox, Hya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29:18–28, 1996. 20

[8] S. D. Babacan, R. Molina, and A.K. Katsaggelos. Variational bayesian super resolution. *IEEE Transactions on Image Processing*, 20(4):984 – 999, 2011. 42

[9] Mark R. Banham and Aggelos K. Katsaggelos. Spatially adaptive wavelet-based multiscale image restoration. *IEEE Transactions on Image Processing*, 5(4):619–634, 1996. 42

[10] M.N. Bareja and C.K. Modi. An effective iterative back projection based single image super resolution approach. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 95 –99, may 2012. 2

[11] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. A user"s guide to pvm parallel virtual machine. Technical report, Knoxville, TN, USA, 1991. 87

[12] Arthur Bernstein. Analysis of programs for parallel processing. *IEEE Transactions on Electronic Computers*, 15(5):757–763, 1966. 18, 21, 88

[13] Richard E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, January 1985. 10

[14] E. J. Candes and T. Tao. Decoding by linear programming. *Information Theory, IEEE Transactions on*, 51(12):4203–4215, December 2005. 54

[15] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, November 1986. 63, 90

[16] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-resolution through neighbor embedding. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:275–282, 2004. 7

[17] G. Chantas, N. Galatsanos, R. Molina, and A.K. Katsaggelos. Variational bayesian image restoration with a product of spatially weighted total variation image priors. *IEEE Transactions on Image Processing*, 19(2):351–362, February 2010. 2, 46

[18] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007. 21, 87

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* McGraw-Hill Science/Engineering/Math, July 2001. 7

[20] Shengyang Dai, Mei Han, Ying Wu, and Yihong Gong. Bilateral back-projection for single image super resolution. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1039 –1042, july 2007. 2

[21] C. Denis, R. Couturier, and F. Jézéquel. *Parallel Scientific Computing and Optimization*, volume 27, pages 47–56. Springer, New York, 2009. PEQUAN LIP6. 27

[22] Radu Dobrescu, Matei Dobrescu, and Dan Popescu. Parallel image and video processing on distributed computer systems. *WSEAS Trans. Sig. Proc.*, 6:123–132, July 2010. 22

[23] Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors. *Sourcebook of parallel computing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. 11

[24] A. Downton and D. Crookes. Parallel architectures for image processing. *Electronics Communication Engineering Journal*, 10(3):139 –151, jun 1998. 22

[25] Claude E. Duchon. Lanczos Filtering in One and Two Dimensions. *J. Appl. Meteor.*, 18(8):1016–1022, August 1979. 38

[26] Michael Elad and Dmitry Datsenko. Example-Based Regularization Deployed to Super-Resolution Reconstruction of a Single Image. *The Computer Journal*, 52(1):15–30, January 2009. 7

[27] Mylène C. Q. Farias. Visual-quality estimation using objective metrics. *Journal of the Society for Information Display*, 19(11):764–770, 2011. 31, 33

[28] Sina Farsiu, Sina Farsiu, Sina Farsiu, Dirk Robinson, Dirk Robinson, Michael Elad, Michael Elad, Peyman Milanfar, and Peyman Milanfar. Advances and challenges in super-resolution. *IEEE Transsations Acoust Speech Signal Process*, 29:1153–1160, 1981. 6

[29] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936. 25

[30] Horace P. Flatt and Ken Kennedy. Performance of parallel processors. *Parallel Computing*, pages 1–20, 1989. 29

[31] William T. Freeman and Egon C. Pasztor. Learning low-level vision. *International Journal of Computer Vision*, 40:2000, 2000. 7

[32] P.G. Freitas, M.C.Q. Farias, and A.P.F. de Araujo. Fast inverse halftoning algorithm for ordered dithered images. In *Graphics, Patterns and Images (Sibgrapi), 2011 24th SIBGRAPI Conference on*, pages 250 –257, aug. 2011. 118

[33] P.G. Freitas, R. Rigoni, M.C.Q. Farias, and A.P.F.d. Araujo. Error concealment using a halftone watermarking technique. In *Graphics, Patterns and Images (SIBGRAPI), 2012 25th SIBGRAPI Conference on*, pages 308 –315, aug. 2012. 119

[34] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB, 2nd ed.* Gatesmark Publishing, 2nd edition, 2008. 8, 9, 24, 32, 37

[35] J R Graham. Comparing parallel programming models. *Journal of Computing Sciences in Colleges*, 23(6):65–71, 2008. 12

[36] William Gropp. Mpich2: A new start for mpi implementations. In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 7–, London, UK, UK, 2002. Springer-Verlag. 87

[37] John L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31:532–533, 1988. 28

[38] Fredric J. Harris. *Multirate Signal Processing for Communication Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. 1

[39] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann, 4 edition, September 2006. 13

[40] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ICPR '10, pages 2366–2369, Washington, DC, USA, 2010. IEEE Computer Society. 31

[41] Quan Huynh-Thu and Mohammed Ghanbari. The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommunication Systems*, 49:35–48, 2012. 32

[42] Intel. Intel pentium processors with mmx technology for embedded computing. `http://www.intel.com/p/en_US/embedded/hwsw/hardware/pentium-mmx`, 2013. [Online; accessed 31-Jan-2013].

[43] M. Irani and S. Peleg. Super Resolution From Image Sequences. *ICPR-C*, 90:115–120. 1

[44] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *CVGIP: Graph. Models Image Process.*, 53:231–239, April 1991. 7

[45] Michal Irani and Shmuel Peleg. Motion analysis for image enhancement: Resolution, occlusion, and transparency. *Journal of Visual Communication and Image Representation*, 4:324–335, 1993. 2

[46] L. Jamieson, D. Gannon, and R. Douglas, editors. *The Characteristics of Parallel Algorithms*. MIT Press, 1987. 21

[47] Jechang Jeong. Digital consumer electronics handbook. In Ronald K. Jurgen, editor, *Digital Consumer Electronics Handbook*, chapter The JPEG standard, pages 91–99. McGraw-Hill, Inc., Hightstown, NJ, USA, 1997. 8

[48] Ken Kennedy, Charles Koelbel, and Hans Zima. The rise and fall of high performance fortran: an historical object lesson. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, HOPL III, pages 7–1–7–22, New York, NY, USA, 2007. ACM. 21

[49] S.P. Kim, N.K. Bose, and H.M. Valenzuela. Recursive reconstruction of high resolution image from noisy undersampled multiframes. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(6):1013 –1027, jun 1990. 2

[50] E.R. Komen. *Low-level Image Processing Architectures: Compared for Some Non-linear Recursive Neighbourhood Operations*. Technische Universiteit Delft, 1990. 24

[51] Shorin Kyo, Shin'ichiro Okazaki, and Tamio Arai. An integrated memory array processor architecture for embedded image recognition systems. *SIGARCH Comput. Archit. News*, 33:134–145, May 2005. 23

[52] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comput.*, 28(9):690–691, September 1979. 17

[53] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19*, pages 801–808. 2007. 59

[54] J. Lenzer and G. Wieber. On design strategies for parallel algorithms in signal processing using graph models. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '80.*, volume 5, pages 939 – 942, apr 1980. 17

[55] Xin Li and Michael T. Orchard. New edge-directed interpolation. *IEEE Transactions on Image Processing*, 10:1521–1527, 2001. 7

[56] Chris Lomont. Understanding Quake's Fast Inverse Square Root. 10

[57] Stéphane Mallat and Guoshen Yu. Super-resolution with sparse mixing estimators. *IEEE Transactions on Image Processing*, 19(11):2889–2900, 2010. 7

[58] Marc Snir and Steve Otto and Steven Huss-Lederman and David Walker and Jack Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998. 20, 87

[59] Joey Markgraf. The Von Neumann bottleneck. `http://aws.linnbenton.edu/cs271c/markgrj/`, 2007. [Online; accessed 31-Jan-2013]. 13

[60] Dave Marshall. Differential encoding. `http://www.cs.cf.ac.uk/Dave/Multimedia/node232.html`, 2001. [Online; accessed 31-Jan-2013]. 68

[61] Judit Martínez, Eva Costa, Paco Herreros, Xavi Sánchez, and Ramon Baldrich. A modular and scalable architecture for pc-based real-time vision systems. *Real-Time Imaging*, 9:99–112, April 2003. 16

[62] MathWorks. Implementing data-parallel applications using the toolbox and matlab distributed computing server. `http://www.mathworks.com/products/parallel-computing/description7.html`, January 2012. [Online; accessed 31-Jan-2013]. 87

[63] MATLAB. *version 7.14.0 (R2012a)*. The MathWorks Inc., Natick, Massachusetts, 2012. 87

[64] Marcus N., Marcus J. Nadenau, David Alleysson, and Murat Kunt. Human Vision Models for Perceptually Optimized Image Processing – A Review. In *Proc. of the IEEE*, 2000. 119

[65] V. Patanavijit. A recursive resolution-enhancement using multiframe srr based on meridian filter with meridian-tikhonov regularization. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTICON), 2011 8th International Conference on*, pages 1047 –1050, may 2011. 2

[66] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901. 25

[67] William K. Pratt. *Digital Image Processing.* Wiley-Interscience, 4 edition, February 2007. 22

[68] G.P. Qiu. Interresolution look-up table for improved spatial magnification of image. 11(4):360–373, December 2000. 10

[69] Anthony Ralston. *A first course in numerical analysis; 1st ed.* International series in pure and applied mathematics. McGraw-Hill, New York, NY, 1965. 37, 38

[70] Holger Rauhut, Karin Schnass, and Pierre Vandergheynst. Compressed sensing and redundant dictionaries. *CoRR*, abs/math/0701131, 2007. 51

[71] Sartaj Sahni and Venkat Thanvantri. Parallel computing: Performance metrics and models. Technical report, University of Florida, 1995. 27

[72] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional, 2010. 21

[73] R.R. Schultz and R.L. Stevenson. A bayesian approach to image expansion for improved definition. *Image Processing, IEEE Transactions on*, 3(3):233 –242, may 1994. 7

[74] Richard W. Stevens and Stephen A. Rago. *Advanced Programming in the UNIX(R) Environment (2nd Edition).* Addison-Wesley Professional, 2005. 21

[75] Jon Stokes. SIMD architectures. `http://arstechnica.com/features/2000/03/simd/`, 2000. [Online; accessed 31-Jan-2013]. 16

[76] Jian Sun. Image super-resolution using gradient profile prior. *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, pages 1–8, 2008. 7

[77] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007. 18

[78] Luis Teixeira and Luis Corte-Real. Coding of multiple h.264 video streams using ssim as quality metric. In *2nd IEEE International Workshop on Multimedia Analysis and Processing (IMAP)*, August 2008. 34

[79] A. M. Tekalp, M. K. Ozkan, and M. I. Sezan. High-resolution image reconstruction from lower-resolution image sequences and space-varying image restoration. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 3, pages 169–172 vol.3. IEEE, March 1992. 2

[80] William J. Thompson. Poisson distributions. *Computing in Science and Engg.*, 3(3):78–82, May 2001. 67

[81] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. V. H. Winston & Sons, Washington, D.C.: John Wiley & Sons, New York„ 1977. 2, 7

[82] R. Tsai and T. Huang. Multiframe image restoration and registration. In *Advances in Computer Vision and Image Processing*, 1984. 1

[83] Hanoch Ur and Daniel Gross. Improved resolution from subpixel shifted pictures. *CVGIP: Graph. Models Image Process.*, 54(2):181–186, March 1992. 2

[84] S. Villena, M. Vega, D. Babacan, R. Molina, and A. Katsaggelos. Bayesian combination of sparse and non sparse priors in image super resolution. *Digital Signal Processing*, 2012. 42, 45

[85] S. Villena, M. Vega, S.D. Babacan, R. Molina, and A.K. Katsaggelos. Using the kullback-leibler divergence to combine image priors in super-resolution image reconstruction. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 893 –896, sept. 2010. 2, 47

[86] S. Villena, M. Vega, R. Molina, and A. K. Katsaggelos. Image prior combination in super-resolution image reconstruction. In *European Signal Processing Conference (EUSIPCO 2010)*, pages 616–620. Aalborg (Denmark), August 2010. 2, 46, 47

[87] Jinjun Wang, Shenghuo Zhu, and Yihong Gong. Resolution enhancement based on learning the sparse association of image patches. *Pattern Recogn. Lett.*, 31:1–10, January 2010. 54

[88] Yuxia Wang, Yuan Zhang, Rui Lu, and Pamela C. Cosman. Ssim-based end-to-end distortion modeling for h.264 video coding. In Weisi Lin, Dong Xu, Anthony Ho, Jianxin Wu, Ying He, Jianfei Cai, Mohan S. Kankanhalli, and Ming-Ting Sun, editors, *PCM*, volume 7674 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2012. 34

[89] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(4):600–612, 2004. 32

[90] C. Weerasinghe, M. Nilsson, S. Lichman, and I. Kharitonenko. Digital zoom camera with image sharpening and suppression. *Consumer Electronics, IEEE Transactions on*, 50(3):777 – 786, aug. 2004. 1

[91] E.T. Whittaker. *The Calculus of Observations - A Treatise on Numerical Mathematics*. Read Books Design, 2012. 38

[92] Todd Wittman. Mathematical techniques for image interpolation. `www.math.ucla.edu/~wittman/thesis/Poral2.pdf`, 2005. [Online; accessed 31-Jan-2013]. 38

[93] Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010. 51, 54

[94] Xiangjun Zhang and Xiaolin Wu. Image interpolation by adaptive 2-d autoregressive modeling and soft-decision estimation. *IEEE Transactions on Image Processing*, 17(6):887–896, 2008. 7

Apêndice A

Trabalhos Publicados

2011 24th SIBGRAPI Conference on Graphics, Patterns and Images

Fast Inverse Halftoning Algorithm for Ordered Dithered Images

Pedro Garcia Freitas, Mylène C.Q. Farias, and Aletéia P. F. de Araújo
Department of Computer Science,
University of Brasília (UnB),
Brasília, Brazil
Email: sawp@sawp.com.br, mylene@ieee.org, and aleteia@cic.unb.br

***Abstract*—In this paper, we present a simple and fast inverse halftoning algorithm, targeted at reconstructing halftoned images generated using dispersed-dot ordered dithering algorithms. The proposed algorithm uses a simple set of linear filters combined with a stochastic model in order to predict the best intensity values for the binary image pixels. The algorithm produces images with a better perceptual quality than the available algorithms in the literature, preserving most of the fine details of the original gray-level image. It has a high performance, which can be further improved with the use of parallelization techniques.**

***Keywords*-halftoning, inverse halftoning, dispersed-dot ordered dithering, dithering.**

I. Introduction

Halftoning is the technique of converting continuous-tone images into binary images using patterns of white and black dots. This technique is useful for creating the illusion of seeing multiple intensity levels in a binary image, which makes it suitable for applications where a reduced number of levels is needed, such as newspapers, fax machines, and document printing processes. Halftoning algorithms can be roughly classified as *dithering* or *error diffusion* [1].

Dithering algorithms generate halftoned images by comparing the pixel intensity values of the original image with threshold scalar values or matrices, which can be generated by various methods. Error diffusion algorithms compare the pixel intensity values with a fixed threshold. The resulting error between the output value and the original value is distributed to neighboring pixels according to predefined weights. Both the ordered dithering and the dithered with error diffusion have advantages and disadvantages for specific applications. The choice of algorithm often depends on the application with which it is associated.

Inverse halftoning is the technique that allows the restoration of the original information of an image (with multiple levels of intensity) from a binary (black-and-white) representation of it. This technique can be used in several applications when the only available version of the image is a binary one. For example, it is well known that dithered images suffer great degradations when subject to simple operations, such as filtering, decimation, interpolation, sharpening, etc. In these applications, it is necessary to use inverse halftoning techniques to convert the binary images into gray-level images and, then, apply the desired operation.

Different solutions have been proposed to solve the inverse halftoning problem, such as iterative projection [2], neural networks [3], vector quantization [4], lookup table [5], [6], wavelet estimation [7][8], and least square methods [9]. Although the above techniques produce satisfactory results for some cases, they all have a high computational cost, which is mainly due to their high mathematical complexity. Among the algorithms in the literature, the work by Damera-Venkata *et al.* is one of the most computationally efficient algorithms [10][11].

A more recent and non-conventional application of inverse halftoning is in error concealment or recovery [12] and data hiding. For example, in the work by Adsumilli *et al.* a dithered version of the original video is embedded into the original itself using a spread-spectrum watermarking technique. If parts of this original are lost in the transmission or compression stages, the receiver can extract the corresponding dithered version of the original from the received frame. Then, an inverse halftoning technique is used to obtain an approximation of the lost data of the frame.

In this paper, we present a simple and fast inverse halftoning algorithm, targeted at reconstructing halftoned images generated using dispersed-dot ordered dithering algorithms. The proposed algorithm uses a simple set of linear filters combined with a stochastic model to predict the corresponding intensity values of the binary image pixels. Our goal is to obtain the best approximation of the original image with the lowest computational cost. Our target application is the recovery of lost information (e.g. error concealment) in real-time processing applications, such as video decoding.

This paper is organized as follows. In Section II, we present a description of the dithering algorithm considered in this work. In Section III, we describe the details of the proposed inverse halftoning algorithm. In Section IV, we present the results and, finally, in Section V, we give our conclusions.

II. Ordered Dithering

In this work, we focus on *ordered dithering* algorithms. These algorithms have the characteristic of generating halftoned images with sets of pixel clusters that have a predictable pattern. This, in turn, generates binary images that

1530-1834/11 $26.00 © 2011 IEEE
DOI 10.1109/SIBGRAPI.2011.14

have an adequate redundancy that allows successful inverse halftoning operations. Given our target applications, generating a more easily predictable image is very important, and has a significant impact on the performance of the technique.

The ordered dithering technique can be classified into two types: dispersed-dot ordered dithering and clustered-dot ordered dithering. In the first type, the less relevant dots for the representation (e.g., white pixels in a dark area) are spread out in an apparently random order, but the density is varied in order to match the gray-level intensity of the original image. In the second type, geometric shapes of different sizes are used to create a pattern that is proportional to the gray-level intensity of the original image. That is, the pixels of the binary image are clustered to create a visual sensation of different tones.

Fig. 1 shows examples of these two types of dithering techniques. In Fig. 1, an image with decreasing gray-level intensities (top) is depicted, along with dithered versions of this image obtained using the dispersed-dot (middle) and clustered-dot dithering algorithms (bottom).

The dots pattern model used in this work is the dispersed dithering. The ten 3×3-pixels dot patterns used to generate the dithered images are depicted in Fig. 2. These patterns were generated using the following Bayer Matrix (M_B) [13]:

$$M_B = \begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix}.$$

To generate the dithered image according to this model, we first quantize the image intensity levels using ten intervals shown in Table I. The image with quantized levels, $I_{quantized}$, is obtained from the original image, $I_{original}$, using the following expression:

$$I_{quantized} = \left\lfloor \frac{10}{255} I_{original} \right\rfloor.$$

Then, we filter the quantized image using the Bayer matrix. The values of each cell of the matrix are used as thresholds. If the normalized output values corresponding to the pixel

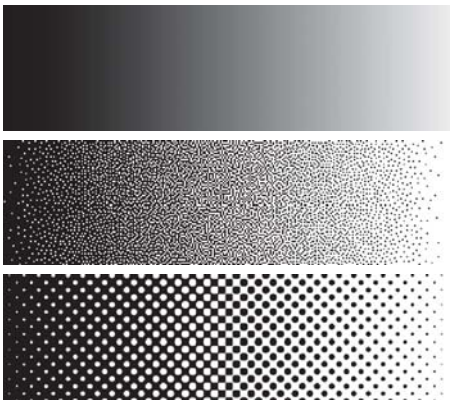


Fig. 1. Original gray-level image (top) and dithered images generated using dispersed-dot (middle) and clustered-dot (bottom) ordered algorithms.

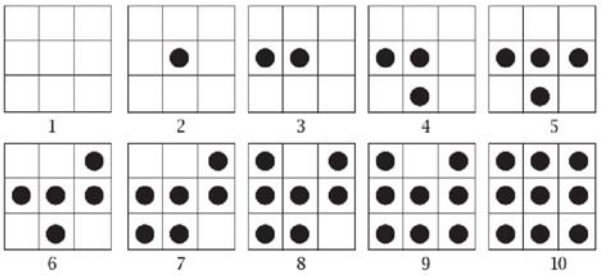


Fig. 2. Dot patterns corresponding to each level of quantization.

are smaller than the number in the matrix cell, the pixel will be substituted by a black value. Otherwise, if the values are greater than the number in matrix cell, the pixel will be replaced by a white value. In other words, the intervals are mapped into one of the patterns shown in Fig. 2.

The algorithm for generating the dithered image from the original 8-bit image is presented Algorithm 1. The resulting dithered version of the image Lena is shown in Fig. 3.

Fig. 3. Halftoned Lena image using Dispersed-dot Ordered Dithering technique.

TABLE I
MAPPED INTERVALS

Interval	Level
$[0, 25.5)$	1
$[25.5, 51)$	2
$[51, 76.5)$	3
$[76.5, 102)$	4
$[102, 127.5)$	5
$[127.5, 153)$	6
$[153, 178.5)$	7
$[178.5, 204)$	8
$[204, 229.5)$	9
$[229.5, 255]$	10

Algorithm 1 Produce 1-bit dithered image from 8-bit image

Input: 8-bit gray-scale input image I_{gray}
Output: 1-bit binary output image I_{dither}
1: Generate $pattern$ as a dictionary data structure that maps a level-value to a dot-pattern, as shown in Fig. 2.
2: Filter the image I_{gray} with a high-pass unsharp filter to obtain $I_{unsharp}$
3: Define a new image $I_{quantized} = \lceil \frac{10}{255} I_{unsharp} \rceil$, which is $I_{unsharp}$ quantized with 10 gray levels.
4: **for all** pixel $p \in I_{quantized}$ **do**
5: $\quad level = I_{quantized}[p]$
6: $\quad I_{dither}[p] = pattern[level]$
7: **end for**

III. Proposed Fast Inverse Halftoning Algorithm

Inverse Halftoning is the process of finding the most appropriate value in a gray-level interval to represent a black-and-white pixel of a dithered image, i.e. it is basically the inverse of the process used for obtaining the dithered image discussed in the previous section. Let us define $M(p)$ as a spatial mask of size 3×3 surrrounding a pixel p of the image, I_{gray} as the original gray-level, I_{dither} as the the dithered image, and $D(p)$ as the distribution of the area surrounding the pixel p. To reconstruct an 8-bit pixel image from an 1-bit pixel halftoned image, we first calculate the local distribution $D(p)$ for all pixels in I_{dither}. Then, we find the most probable mapped intervals corresponding to the original pixel intensity levels in I_{gray}. Once this interval is found, we randomly select a value within it, following a *random walk on fluctuating lattice model* [14].

In order to randomly select the best value in an interval, the random walk model uses a *renormalization group* transformation, which is a transformation of the space of spatially connected pixels. In this work, we chose to use a simple model with only spatial dependencies for the renormalization transformation. With this model, the transformation rescales the value of a particular pixel depending on the levels (see Table I) of the pixels in the surrounding 3×3 area. More specifically, for a given pixel p, we must first find the most frequent level value in the neighborhood. If this value is the same as the level of p, or it is not in the immediate vicinity, we choose a random value in the level interval. If the most frequent level is higher than the level of p, we set the upper-bound as mid-value of the upper interval and the lower-bound as the current value of the pixel. If the most frequent is lower than the level of p, we set the upper-bound as the current value of the pixel and lower-bound as the mid-value of the lower interval. From these new bounds, we choose a new random value for pixel.

The result of applying this inverse halftoning technique to the dithered version of the image Lena (see Fig. 3) is shown in Fig. 4(b). For comparison, the original Lena image is shown in Fig. 4(a). As expected, although a consistent gray-level image was obtained, lots of false-contours defects are present as a consequence of the quantization of the intensity levels.

To mitigate the problem of false contours, we apply an unsharp filter before the dithering process. The unsharp filter is used to highlight the image details and, therefore, to avoid the loss of important information. Then, we classify the regions of the image as 'smooth' or 'active' using a simple standard deviation measure over a 3×3 pixel area. If the region has a low value of standard deviation, it is replaced by its average. Otherwise, it remains unchanged. Using this small modification, we obtain a much better result. The result of applying this technique for the Lena dithered image is depicted in Fig. 4(c).

Fig. 4. Inverse Halftoning and Filtering Effects: (a) Original, (b) Random Walk, (c) Random Walk with unsharp masking, and (d) Random walk with unsharp masking and Gaussian filtering.

As can be observed in Fig. 4(c), unfortunately, the random choice of gray level intensity produces a fair amount of visible noise in the reconstructed image. We can smooth it out by applying a low-pass filter after the reconstruction process. We use a Gaussian blurring filter, generating the image in Fig. 4(d). As can be seen in this image, excess noise was greatly reduced in this final image, without affecting the image details. The proposed algorithm for inverse halftoning is presented Algorithm 2.

To find better filter parameters for the proposed algorithm, we conducted a set of tests, which basically consisted of choosing the optimal filter configurations to maximize the PSNR values for the resulting reconstructed images. The spatial masks that offered the results for the unsharp masking and Gaussian filters are:

$$M_{unsharp} = \begin{bmatrix} -0.489 & -0.022 & -0.489 \\ -0.022 & 3.044 & -0.022 \\ -0.489 & -0.022 & -0.489 \end{bmatrix}$$

and

$$M_{Gaussian} = \begin{bmatrix} 0.052 & 0.124 & 0.052 \\ 0.124 & 0.297 & 0.124 \\ 0.052 & 0.124 & 0.052 \end{bmatrix}.$$

Algorithm 2 Reconstruct 8-bit image from 1-bit image

Input: 1-bit binary input image I_{dither}.
Output: 8-bit gray-scale output image I_{gray}.

1: Generate $inversePattern$ as a dictionary data structure that maps a 3×3 dot-pattern to a level-value, as shown in Fig. 2.
2: **for all** pixel $p \in I_{dither}$ **do**
3: Define as $mask$ the 3×3 matrix populated with values of pixels around $p = I_{dither}(x, y)$

$$mask = \begin{bmatrix} p(x-1, y-1) & p(x-1, y) & p(x-1, y+1) \\ p(x, y-1) & p(x, y) & p(x, y+1) \\ p(x+1, y-1) & p(x+1, y) & p(x+1, y+1) \end{bmatrix}$$

4: Use $mask$ as key in $inversePattern$, obtaining the correspondent $level$

$$level = inversePattern[mask]$$

5: Calculate the level corresponding to pixel p

$$I_{levels}[p] = level$$

6: **end for**
7: **for all** pixel $p \in I_{dither}$ **do**
8: Get $level = I_{levels}[p]$
9: With $level$, set min and max as the lower and upper limits of range mapped by $level$, as illustrated in Table I.
10: Choose a random value between min and max, storing the result in a new variable $grayvalue$.
11: Create a matrix v containing the pixels in 3×3 neighborhood around p.
12: Call the $renormalization(v)$ procedure and store the result in $I_{reconstructed}[p]$ (random walk model)

$$I_{reconstructed}[p] = renormalization(v)$$

13: **end for**
14: Filter the $I_{reconstructed}$ with a gaussian low-pass filter, save the result in I_{gray} and return.

$$I_{gray} = gaussianLowPassFilter(I_{reconstructed})$$

IV. Experimental Results

We tested the proposed algorithms using a set of gray-level images. The results obtained for the images 'Rose', 'Einstein', 'Chester Cathedral', 'Paper machine', 'Bear', 'Hurricane', 'Peppers' and 'Pills' are depicted in Figs. 5 and 6. In both figures, the first column shows the original gray level images, the middle column shows the dithered version, and the last column shows the reconstructed image using the proposed algorithm.

The images chosen to test the proposed algorithm have a high level of details, what represents a big challenge for inverse dithering algorithms. Notice that the proposed algorithm is able to recover the details of the original image, even in high contrast areas (see 'Rose' and 'Bear' images). It is also able to recover the large variations in luminance (see 'Paper machine' image). The algorithm does add some graininess to the uniform areas. This is a characteristic of most sharpness enhancement algorithms, including the sharpness algorithm used in the proposed technique used compensate for the blurring effect of the dithering.

We also compared the algorithm proposed in this paper with other inverse halftoning algorithms. The following state-of-the-art algorithms were considered for comparison: *Fast Blind Inverse Halftoning* (FBIH) [10] and *Wavelet-based Inverse Halftoning via Deconvolution* (WinHD) [7]. Fig. 7 shows the reconstruction images using the proposed algorithm and these two reference algorithms. We can observe from this figure that the image reconstructed using our approach preserves much more details than the images reconstructed using the other two methods.

In our simulations, we also used three metrics for estimating the objective quality of the reconstructed images: Peak signal-to-noise ratio (PSNR), Universal Image Quality Index (UIQI) [15] and Structural similarity (SSIM) [16].

The results obtained with PSNR are listed in Table II. From the data, we can observe that the proposed method has significantly lower PSNR scores than the other two algorithms. As the results in Fig. 7 show, the low PSNR values listed in the Table II do not necessarily represent a lower visual quality, but only bigger differences in relation to the original. PSNR is a fidelity metric that simply estimates error differences between original and test images, not being able to identify if these differences cause an improvement or a degradation in quality. Over the years, PSNR have been widely criticized for not correlating with quality as perceived by human viewers [17].

TABLE II
Results using Peak signal-to-noise ratio (PSNR)

Image	FBIH	WinHD	Proposed
Cameraman	24.7800	31.1112	24.2410
Galaxy	33.6986	35.1189	25.6677
Peppers	27.4723	31.0780	24.8490
Chester cathedral	20.9075	23.5957	22.1560
Hurricane	26.8154	26.0245	25.4556
Pills	28.0535	31.3776	25.7641
Dollar	22.1636	21.4012	24.2334
Lena	29.5776	32.4471	25.3655
Bear	27.0785	30.5968	25.9224
Einstein	31.3399	33.0679	24.2774
Paper machine	27.6452	29.9093	25.4409
Rose	31.0464	30.2284	29.6955

Table III shows the results using the metric UIQI as a comparison criteria. In this case, we observed a strong fluctuation, making the results obtained with UIQI inconclusive.

Table IV shows the results obtained with the quality metric SSIM. The SSIM metric is one of the most reliable quality metrics available in the literature today, being able to give a better estimate of quality instead of only measuring the error differences between a pair of images [17]. When compared to the WinHD algorithm, a very computationally intense algorithm, the proposed method presents better results in around 60% of the tested images. When compared to the FBIH algorithm, the proposed method presents better results in all tested images. This is in agreement with the visual (perceptual) results presented in Fig. 7. In particular, notice

that the proposed method performed very well (according to SSIM) for images with a high level of detail, such as 'Rose', 'Pills' and 'Papermachine'.

TABLE III
RESULTS USING UNIVERSAL IMAGE QUALITY INDEX (UIQI)

Image	FBIH	WinHD	Proposed
Cameraman	0.9413	0.9675	0.9978
Galaxy	0.9993	0.9992	0.9290
Peppers	1.0003	0.9945	0.9688
Chester cathedral	0.99867	1.0006	0.9915
Hurricane	0.9840	0.9822	0.8949
Pills	0.9976	1.0054	0.9742
Dollar	0.9973	0.9981	0.9914
Lena	0.9989	0.9999	0.9804
Bear	0.9158	0.9427	0.9092
Einstein	0.9998	0.9999	0.9875
Paper machine	0.9710	0.9995	0.9596
Rose	1.0513	0.6299	0.5358

TABLE IV
RESULTS USING STRUCTURAL SIMILARITY (SSIM)

Image	FBIH	WinHD	Proposed
Cameraman	0.7375	0.9139	0.8599
Galaxy	0.8640	0.9191	0.9076
Peppers	0.7894	0.8469	0.8590
Chester cathedral	0.6648	0.8337	0.8352
Hurricane	0.7451	0.6842	0.8529
Pills	0.8597	0.9157	0.9279
Dollar	0.7352	0.6783	0.8921
Lena	0.8318	0.8896	0.8698
Bear	0.8092	0.8807	0.8418
Einstein	0.8513	0.9173	0.8988
Paper machine	0.8384	0.8788	0.9169
Rose	0.6697	0.6590	0.8278

We also compared the elapsed time for executing the proposed algorithm and the two others algorithms. In Table V, we list these times (in seconds) for all images in our dataset. As can be observed from the results, the proposed model has a much better performance than the other two reference methods, which makes it very adequate to any application that requires real-time processing. It is worth pointing out that both the dispersed-dot ordered algorithm and the proposed inverse halftoning algorithms are highly parallelizable algorithms. So, the processing time could be reduced even further for both the codification and the decodification phases.

TABLE V
ELAPSED TIME FOR RECONSTRUCTION (IN SECONDS).

Image	Proposed	FBIH	WinHD
Cameraman	1.79	2.46	35.14
Dollar	1.01	3.00	52.27
Einstein	1.53	3.00	53.36
Galaxy	1.44	3.01	65.78
Hurricane	1.24	2.99	75.48
Rose	1.24	2.02	73.13
Lena	1.26	2.40	36.29

V. CONCLUSIONS AND FUTURE WORK

We have presented a simple and fast approach for reconstructing halftoned images generated using dispersed-dot ordered dithering algorithms. The proposed algorithm produces images with a better perceptual quality than the available algorithms in the literature, while preserving most of the fine details of the original gray-level image. The proposed method has a high performance, which can be further improved with the use of parallelization techniques. Also, perceptual results would certainly benefit from the use of low-pass and high-pass filters optimized by taking into account a quality metric that is better correlated with perceptual quality, as opposed to PSNR.

ACKNOWLEDGMENT

This work was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), in part by a grant by Fundação de Empreendimentos Científicos e Tecnológicos (Finatec), and in part by a grant from DPP - University of Brasília (UnB).

REFERENCES

[1] J.-M. Guo and H. Lee, "Watermarking in halftone images with mixed halftone techniques," *Int. J. Imaging Syst. Technol.*, vol. 17, pp. 303–314, February 2007.

[2] M. Analoui and J. Allebach, "New results on reconstruction of continuous-tone from halftone," *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, vol. 3, pp. 313–316, 1992.

[3] Y. Mita, S. Sugiura, and Y. Shimomura, "High quality multi-level image restoration from bi-level image," in *Proc. Sixth Int. Congress Advances in Non-impact Printing Technologies for Black-and-White and Color*, 1990, pp. 791–802.

[4] J. Z. C. Lai and J. Y. Yen, "Inverse error-diffusion using classified vector quantization," *IEEE Transactions on Image Processing*, vol. 7, no. 12, pp. 1753–1758, 1998.

[5] M. Yuan, E. A. Riskin, T. Eve, and A. Riskin, "Error diffused image compression using a binary-to-grayscale decoder and predictive pruned tree-structured vector quantization," *IEEE Transactions on Image Processing*, vol. 3, pp. 854 – 858, 2002.

[6] Z. Karni, D. Freedman, and D. Shaked, "Fast inverse halftoning," *31st International Congress on Imaging Science*, 2010.

[7] R. Neelamani, R. D. Nowak, and R. G. Baraniuk, "Winhd: Wavelet-based inverse halftoning via deconvolution," *Rejecta Mathematica*, pp. 84–103, July 2009.

[8] Z. Xiong, M. T. Orchard, and K. Ramch, "Inverse halftoning using wavelets," in *in Proc. IEEE Int. Conf. Image Proc*, 1996, pp. 569 – 572.

[9] P.-C. Chang, C.-S. Yu, and T.-H. Lee, "Hybrid lms-mmse inverse halftoning technique," *IEEE Transactions on Image Processing*, pp. 95–103, 2001.

[10] N. Damera-venkata, T. D. Kite, M. Venkataraman, and B. L. Evans, "Fast blind inverse halftoning," in *in Proc. IEEE Int. Conf. Image Proc*, 1998, pp. 64–68.

[11] T. D. Kite, N. Damera-venkata, B. L. Evans, and A. C. Bovik, "A high quality, fast inverse halftoning algorithm for error diffused halftones," in *in Proc. IEEE Int. Conf. Image Proc*, 1998, pp. 59–63.

[12] C. Adsumilli, M. C. Q. de Farias, S. K. Mitra, and M. Carli, "A robust error concealment technique using data hiding for image and video transmission over lossy channels," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 15, no. 11, pp. 1394–1406, 2005.

[13] D. E. Knuth, "Digital halftones by dot diffusion," *ACM Trans. Graph.*, vol. 6, pp. 245–273, October 1987.

[14] C. D. Levermore, W. Nadler, and D. L. Stein, "Random walks on a fluctuating lattice: A renormalization group approach applied in one dimension," *Physical Review E*, vol. 51, no. 4, pp. 2779–2786, Apr 1995.

[15] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, pp. 81–84, 2002.

[16] Z. Wang, L. Lu, and A. C. Bovik, "Video Quality Assessment Based on Structural Distortion Measurement," *Signal Processing: Image Comm.*, vol. vol19, pp. 121–132, 2004.

[17] Z. Wang and A. C. Bovik, "Mean Squared Error : Love It or Leave It?" *IEEE Signal Processing Magazine*, no. January, pp. 98–117, 2009.

Fig. 5. Results obtained for the images 'Rose', 'Einstein', 'Chester cathedral ' and 'Paper machine': original image (left), halftoned image (center), and reconstructed image using the proposed algorithm (right).

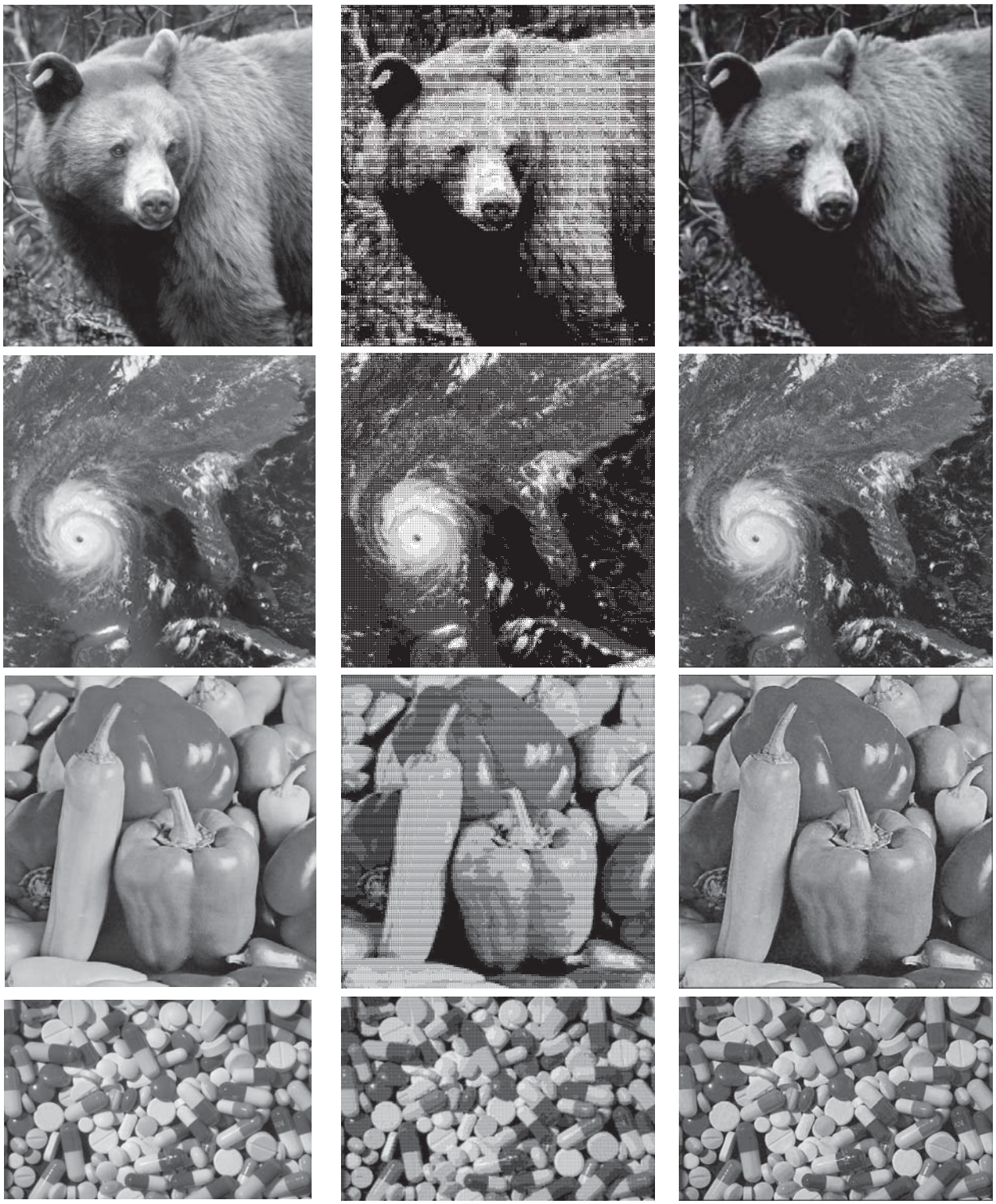


Fig. 6. Results obtained for the images ‘Bear’, ‘Hurricane’, ‘Peppers’ and ‘Pills’: original image (left), halftoned image (center), and reconstructed image using the proposed algorithm (right).

Fig. 7. Results obtained using the Proposed (left), FBIH (center), and WinHD (right) inverse halftoning methods for the images 'Bear', 'Chester cathedral', 'Lena' and 'Peppers'.

Error Concealment Using a Halftone Watermarking Technique

Pedro Garcia Freitas, Ronaldo Rigoni, Mylène C.Q. Farias, and Aletéia P.F. de Araújo
Department of Computer Science,
University of Brasília (UnB),
Brasília, Brazil

***Abstract*—In this paper, we propose an error concealment technique for H.264 coded videos. The algorithm is targeted at compressed videos degraded after packet losses caused by transmission over an unreliable channel. We use a combination of watermarking and halftoning (dithering) techniques. At the encoder side, a dithered version of each video frame is embedded into the video using a watermarking technique. The watermarking technique used by the proposed algorithm is a modified version of the Quantization Index Modulation (QIM) algorithm, which provides a good data hiding capacity. At the decoder side, the algorithm identifies which packets of the video were lost, extracts the corresponding mark, and applies an inverse halftoning technique to estimate the original content. The algorithm is fast and has a good performance, being able to restore content with a better quality than the default H.264 error concealment algorithm.**

***Keywords*-Error concealment, H.264, QIM, watermarking, halftoning.**

I. Introduction

The Internet provides a best effort model, which means losses may happen when packets are being transmitted. In the majority of applications, errors caused by losses are undesirable. In particular, for video transmission, errors may decrease the quality of received content, affecting the level of acceptability and popularity of the service. Error concealment algorithms are frequently used to minimize the effect of transmission errors on video content. These algorithms use the temporal and spatial information of the received pixels to estimate the lost data. Most error concealment methods are implemented as a part of coding and decoding algorithms, like H.264 [1].

Various approaches have been proposed to conceal transmission errors in videos and images, with interpolation being the most common technique [2], [3]. Approaches using interpolation have as great advantage the combination of algorithms simple, with low complexity, which does not requires changes in the encoder-side. However, the conceal errors capacity is lower than those techniques that insert some information on the encoder-side.

A different approach consists of using data-hiding techniques embed redundant information into the video or image, what allows to recover lost content. Examples of works that use watermarking techniques to self-embed a dithered halftoned version of the content include the works of Adsumilli *et al.* [4] and Phadikar and Maity [5]. The work of Nayak *et al.* [6] uses a hybrid method composed of multiple techniques, including watermarking, to conceal lost information in compressed videos. The disadvantage of inserting the information at the encoder side is that the a small amount of degradation is added to the signal. Furthermore, it is necessary that both the encoder and the decoder are synchronized, what increases the complexity of both the encoder and the decoder. However, the performance of data hiding based error concealment algorithms is superior than what is obtained with interpolation methods.

In this paper, we propose an error concealment technique for H.264 compressed videos that is simple, fast, and recovers lost content with a very good quality. The algorithm can be implemented together with the H.264 codec, what reduces complexity and avoids further distortion of the signal. The paper is divided as follows. In Section II, we describe the proposed system, detailing the halftoning, watermark embedding and extraction, inverse halftoning and reconstruction stages. In Section III, we present the simulation results. Finally, in Section IV, we present our conclusions.

II. Proposed Algorithm

The proposed algorithm has the goal to recover portions of the video lost in a transmission over an unreliable channel. At the encoder side, a halftoning algorithm generates a dithered halftoned version of each video frame. This information is embedded into the video using a fragile watermarking technique that is a modified version of the Quantization Index Modulation (QIM) algorithm [7]. At the decoder side, the algorithm identifies which packets of the video were lost. Then, for each lost packet, the original content is recovered by extracting the mark corresponding to the affected area and using an inverse halftoning algorithm to convert the dithered version of the frame into a colored multi-level approximation of the original.

The block diagram of the proposed algorithm is depicted in Fig. 1. Notice that this block diagram depicts stages of the H.264 codec combined with stages of the proposed error concealment algorithm. The idea here is to show that the proposed algorithm is intended to be implemented together with the H.264 codec (or any other compression algorithm). In this section, we describe in further details the stages of the proposed error concealment algorithm.

1530-1834/12 $26.00 © 2012 IEEE
DOI 10.1109/SIBGRAPI.2012.50

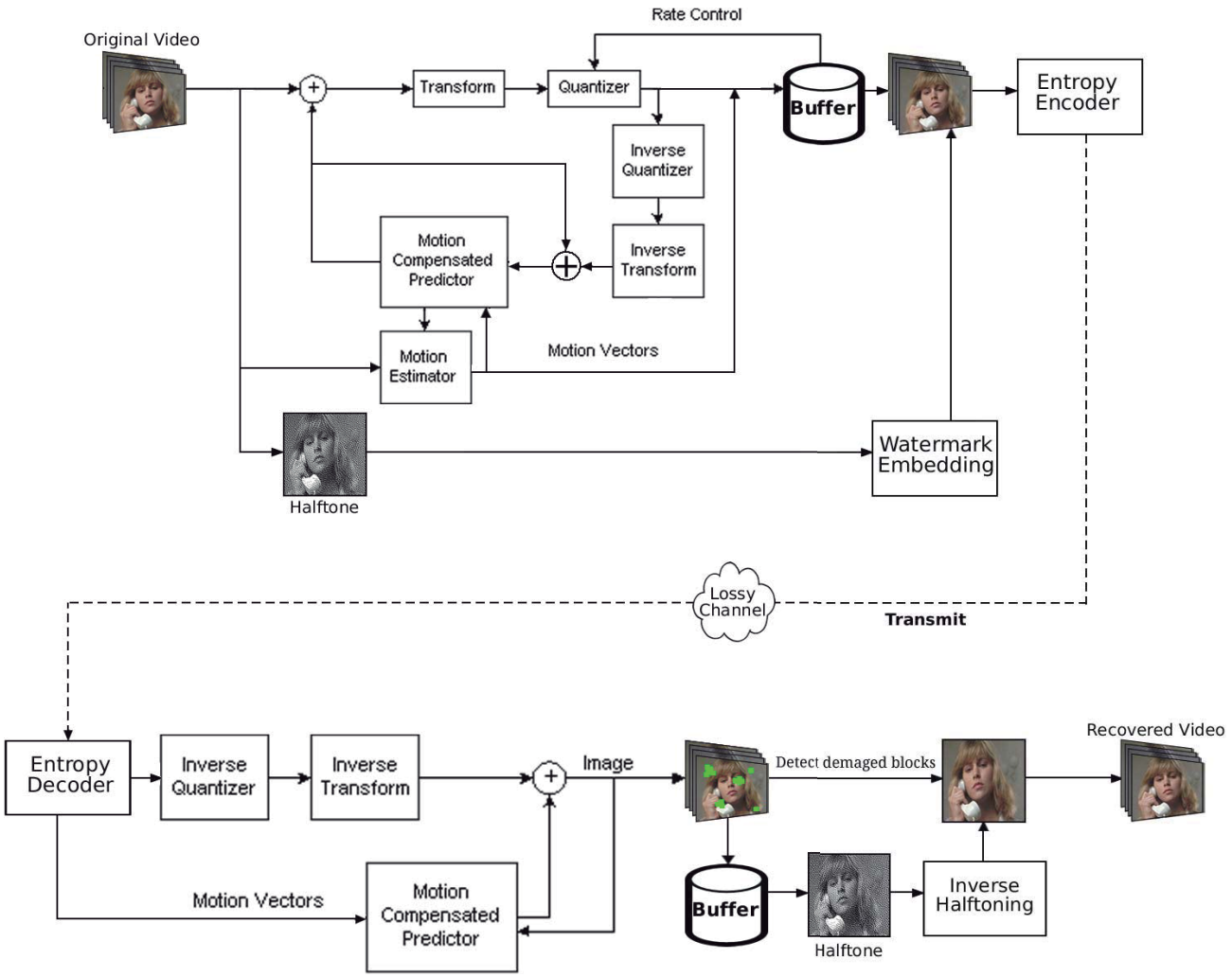


Fig. 1. Block diagram of the proposed system, which detects lost regions in a video and restores the original content.

A. *Halftoning Stage*

Halftoning is a technique for converting multi-level images into binary images using patterns of white and black dots [8]. This technique creates the illusion of seeing multiple intensity levels in a binary image, what makes it suitable for applications where only a reduced number of levels is available, such as newspapers, fax machines, and document printing processes. In the literature, there are several halftoning algorithms that can be roughly divided into two major classes: ordered dithering and error diffusion [9], [10].

Error diffusion algorithms compare the pixel intensity values with a fixed threshold. The resulting error between the output value and the original value is distributed among the neighboring pixels according to predefined weights. This provides interesting visual results and eliminates the local correspondence between the halftoned image and original gray level. *Ordered dithering* algorithms generate halftoned images by comparing each pixel value with a set of pixel clusters. The spatial patterns used by the chosen set of pixel clusters determine the quality of the dithered halftoned image. Both ordered dithering and error diffusion techniques have advantages and disadvantages. The choice of the type of halftoning algorithm often depends on the target application.

In this work, an ordered dithering algorithm is used to generate a dithered halftoned version of each picture frame, which is later embedded into the video itself. This type of halftoning technique is chosen because it uses sets of pixel clusters that have a predictable pattern. For each pixel of the picture, the algorithm only uses the pixel value and a fixed number of spatial patterns to generate the halftoned picture. Given that our target application requires generating a multi-level picture from a halftoned picture with (possibly) lost content, using an easily predictable pattern is very important.

As pointed out previously, in case of content loss the halftoned version of the picture frame is used to restore the video back to its original state. The quality of the restored video depends on the halftoning algorithm and of the number

of intensity levels it is able to represent. So, in order to guarantee a good quality, the halftoning algorithm must be able to represent the largest possible number of intensity levels with a minimum number of bits. Unfortunately, the data hiding capacity of the host picture frame is relatively low. Our tests show that a maximum of 3 bits per pixel can be embedded in the host picture frame without causing visible degradations.

Because of this limited data hiding capacity, we cannot use the classical ordered dithering patterns. So, we use a modified method of [11], proposed as *combinatorial dispersed-dot patterns* which is capable of generating all combinations of bits necessary to represent the intensity levels, but require less bits than classical ordered dithered patterns. This way, we can increase the number of mapped intervals without increasing the size of the dot-pattern matrix. For example, using a 3×3 Bayesian matrix to generate the dispersed-dot dithering, we obtain 10 distinct levels. On the other hand, using a 3×3 combinatorial matrix allows for $2^{3\times3}$ distinct configurations, that translates into up to 512 intervals. Since 3 bits are available for our application, we can have 2^3 combinations and, consequently, 8 different intervals, as shown in Fig. 2.

Pictures reconstructed from dispersed-dot dithered halftoned images can be slightly blurred. So, before we generate the halftoned picture we apply an unsharp-mask edge enhancement filter to the original picture frame (I_o), generating an image with enhanced details (I_{eh}). In Fig. 3(b), an example of this enhancement step is depicted for a frame of the video "Foreman" presented in Fig. 3(a).

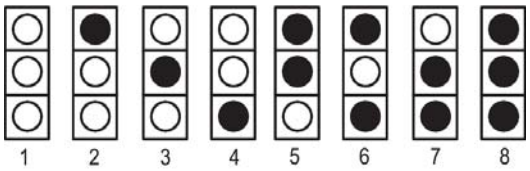


Fig. 2. Combinatorial dispersed-dot patterns used for generating dithered images with 3 bits, which allows mapping 8 intervals.

Then, we quantize I_{eh} using 8 distinct intervals, using the following equation:

$$I_q(x, y, c) = \left\lceil \frac{8}{255} I_{eh}(x, y, c) \right\rceil, \tag{1}$$

where x and y are the horizontal and vertical spatial dimensions, respectively, c refers to the color channel ($1 \leq c \leq 3$), and $\lceil . \rceil$ is the floor operation, and I_q is the quantized image. Next, we substitute the value of each pixel in I_q by the corresponding combinatorial dispersed-dot patterns, showed in Fig. 2, generating the halftoned image I_{dth}. Finally, for each color channel, the 3 bits are converted to an integer number. In Fig. 3(c) the halftoned picture corresponding to the frame shown in Fig. 3(a) is presented.

B. Watermarking Embedding Stage

After generating the mark using the techniques described in the previous section, the next stage consists of embedding it into the host image or video frame. But, in order to make it possible to recover the original content, the dithered mark corresponding to a specific region cannot be embedded in the same spatial and temporal position of the host image or video frame. Therefore, we spatially distribute the mark over the host picture using a *split-flip operation*, which consists of splitting the halftoned image into sub-blocks and flipping them to a different spatial region. More specifically, we divide the halftoned picture in 64×64 blocks, rotate each sub-block by 180 degrees, and shuffle the regions. Fig.4 shows one example of the process for the three color channels of the image "Lena".

For videos signals, we also perform a temporal distribution of the mark by inserting the mark corresponding to the current picture frame in a previous picture frame, located 1 second before. By distributing the mark spatially and temporally, a specific region does not store the mark necessary to restore it, what increases the probability that the algorithm is able to restore the content to its original version.

Among the available watermarking methods, the Quantization Index Modulation (QIM) algorithm is one of the methods with the best performance [7]. The QIM algorithm inserts a mark into a host picture by quantizing it with a uniform scalar quantizer. The standard quantization operation with step size δ is defined by the following equation

$$Q(x, \delta) = \left\lfloor \frac{x}{\delta} \right\rfloor \cdot \delta, \tag{2}$$

where $\lfloor . \rceil$ denotes the mathematical operation of rounding a value to the nearest integer. The watermarked pixel is obtained using the following equation

$$s(x) = Q(x, \delta) + d(m), \tag{3}$$

where $d(m)$ is the perturbation value, m is the mark signal to be embedded, and $Q(x, \delta)$ is the quantization function defined in equation 2. The value of the step size δ determines the data hiding capacity of the algorithm. The higher δ, the more bits per pixel can be inserted in the host. The value of δ also affects the level of distortion introduced in the original by the watermarking algorithm. In this paper, we used a value of δ equal to 3.

In this work, we propose a modification of the QIM algorithm that inserts an integer mark, instead of a function of a binary mark. In other words, the modulation function in equation 3 is set to $d(m) = m$. Therefore, the integer-halftoned mark, I_{dth}, is inserted in each pixel of the corresponding color channel of the original picture frame using the following equation:

$$I_m(x, y, c) = Q\left(I_o(x, y, c), \delta\right) + I_{dth}(x, y, c), \tag{4}$$

where I_m is the resulting watermarked picture frame, I_o is the original picture frame, δ is the quantization step, and c is the corresponding color channel.

C. Watermarking Extraction Stage

The watermarking extraction is performed at the receiver or decoder side, after the transmission of the video. First, we

(a) (b) (c)

Fig. 3. Halftoning algorithm: (a) Original frame, (b) Edge enhanced frame, (c) Halftoned frame.

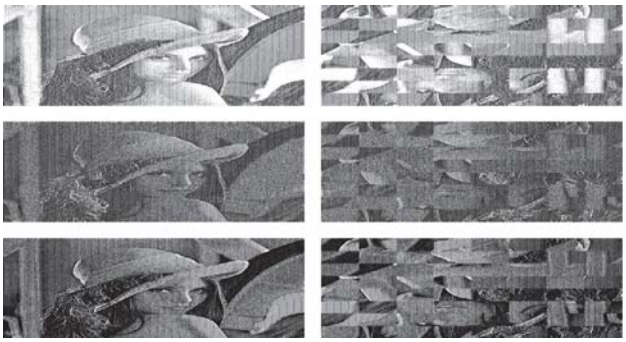


Fig. 4. Original dithered halftoned versions of each color channel of the image "Lena" (left) and the corresponding split-flip versions (right).

create a buffer containing the current picture frame and all the frames in the previous 1 second interval. After that, we are ready to restore losses in the next picture frames.

When a packet loss is detected by the H.264 decoder, we extract the corresponding mark from the buffer using the following equation:

$$\hat{I}_{dth}(x,y,c) = I_m(x,y,c) \mod \delta, \tag{5}$$

where I_m is the watermarked color channel and $\hat{I}_{dth}$ is the recovered integer-halftoned mark. Then, the extracted integer-halftoned mark is converted to a binary number of $\delta = 3$ bits in order to restore the halftoned picture frame.

D. Inverse Halftoning Stage

If any region is classified as "lost" in the current frame, we search the correspondent mark in the appropriate location of the buffer. Then, we use the inverse halftoning algorithm to generate a multi-level colored picture frame, which is an approximation of the original picture frame.

Given that I_{dth} is the halftoned picture frame, $D(p)$ is the distribution of the area surrounding the pixel p in I_{dth}. To reconstruct an 8-bit pixel from the halftoned picture, we first calculate the local distribution $D(p)$ for all pixels in I_{dth}. From this distribution, we find the corresponding mapped interval that contains the most probable pixel value in the corresponding color channel, according with the indices of the dot-patterns, shown in Fig.2. Once this interval is found, we randomly select a value within it, generating a slightly noisy picture I_{inv}.

Then, we filter I_{inv} with a Gaussian lowpass and a Laplacian-of-Gaussian filter [12]. The idea here is to spatially decorrelate the pixels in order to be able to process each pixel independently. This allows for an independent reconstruction, even when the neighboring pixels are missing. The resulting picture frames, I_{gauss} and I_{log}, are used to compose another picture, I_{bld}, given by the following equation:

$$I_{bld}(x,y,c) = \Upsilon\ I_{gauss}(x,y,c) + (1-\Upsilon)\ I_{log}(x,y,c), \tag{6}$$

where Υ is the blending-ratio matrix that determines the proportion of each input filtered picture in the output.

In our simulations, we observed that using $\Upsilon = I_{inv}$ preserves the edges of the pictures. Unfortunately, when we combine all 3 channels, the resulting picture frame contains visible color distortions. If, on the other hand, we use Υ as a constant matrix, I_{bld} is a blurred version of I_{inv}. An example of this can be seen in Fig. 5. In Figs. 5(a)-(d), the extracted halftoned picture, the filtered images, and their combination are shown, respectively. Notice that the combination I_{bld} is a very blurred picture (see Fig. 5(d)).

Hence, it is necessary to make another composition using I_{inv} and I_{bld}, with the goal of minimizing color distortion and keeping the details of the original image. The composition, $\hat{I}_o$, is the final result of inverse halftoning process and is given by the following equation:

$$\hat{I}_o(x,y,c) = \left\lfloor \sqrt{I_{inv}(x,y,c)\ I_{bld}(x,y,c)} \right\rceil, \tag{7}$$

where $\hat{I}_o$ is the recovered 8-bit version of original video frame, I_o. This way, we recover the original content with the best visual quality possible. An example of this process for the pictures in Fig. 5 is shown in Fig. 5(e).

III. Test Results

First, we tested the degradation of the original content caused by the proposed method, in particular by the insertion of the watermark. In Fig. 6 we show a comparison between original and watermarked picture frame for the videos "Carphone", "Foreman", "Mobile", and "Suzie". These videos are

(a) I_{dth} (b) I_{inv}

(c) I_{gauss} (d) I_{log}

(e) I_{bld} (f) $\hat{I}_o$

Fig. 5. Inverse halftoning algorithm steps: (a) extracted halftoned mark, (b) random inverse estimative, (c) image filtered using Gaussian low-pass filter, (d) image filtered using LoG filter, (e) blind combination of I_{gauss} and I_{log}, (f) restored image.

publicly available of format YUV 4:4:4 color CIF (352 $\times$288, progressive) with around 300 frames each. From this figure, we can observe that the addition of the mark causes very small distortions in the colors of the frame due to quantization. However, this effect is not very visible without having the original frame as reference.

In Table I, we show the values of the Peak signal-to-noise ratio (PSNR), the Structural similarity (SSIM) [13], and Universal Image Quality Index (UQI) [14] output values of the marked videos corresponding to the four originals. The values obtained with these metrics suggest that the marked videos have a good quality and that the small degradations present in these videos are visually acceptable.

Second, we tested our algorithm using a set of still images modified with a percentage of 16$\times$16 blocks deleted. To implement this test, we divided the images in blocks of 16$\times$16 pixels. Then, we deleted (substituted the original content by the value 0) of a fixed percentage of the blocks, with spatial positions chosen randomly. The percentage of lost blocks used in this test varied from 5% to 25%.

Fig. 7 shows three examples of the restoration of lost blocks using the proposed error concealment algorithm. The images on the upper line have 20% of the blocks deleted, while the images on the bottom are the corresponding restored versions. Notice that is difficult to identify the location of the restored blocks. We calculated the PSNR and SSIM values between the original and restored images of the test cases. Figs. 8 and

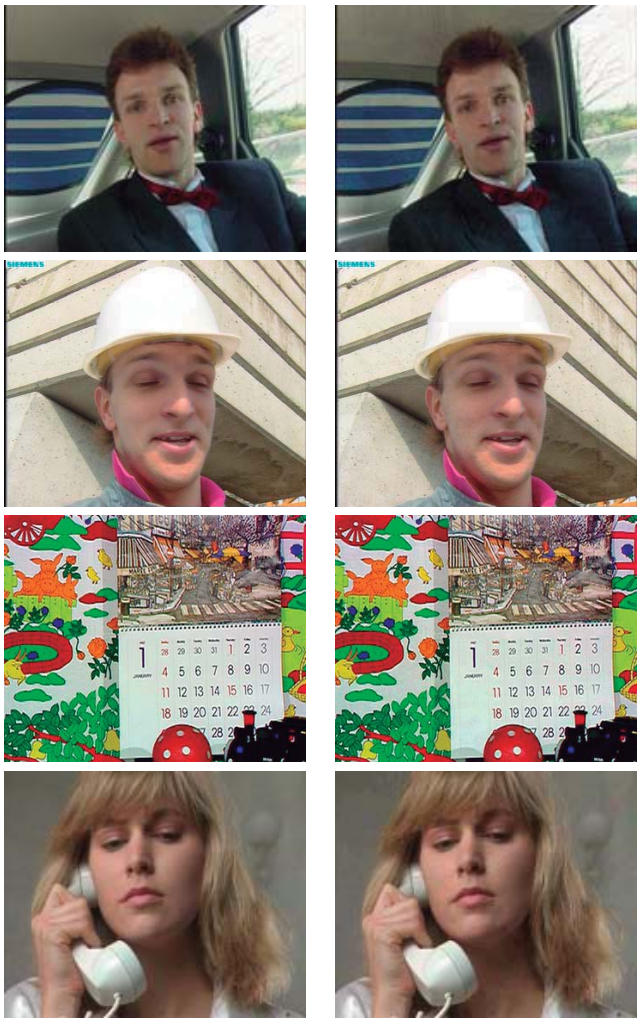


Fig. 6. Comparison of the original (left) and marked (right) images. From top to down: “Carphone”, “Foreman”, “Mobile” and “Suzie”.

9 depict the graphs of SSIM and PSNR versus the percentage of deleted blocks for all test images. It can be observed that, as expected, the quality of the images decreases with the percentage of deleted blocks.

Third, we tested the ability of the proposed algorithm to recover lost packets from H.264 compressed videos. We embed the proposed error concealment algorithm in a H.264 codec to obtain a protected video stream. In order to simulate packet losses in a given bitstream, we use a simple model that simulates packet losses over error-prone channels. The loss model discards groups of H.264 packets (Network Abstraction Layer units – NALs) from the middle of the bitstream, according to the desired packet loss rate (PLR) values. All packets are treated equally regarding their susceptibility to loss, i.e. we did not focus on specific types of packets, such as those carrying intra-coded slices. We tested PLR values of 0.5%, 1%, 3%, 5%, and 10%.

For these test cases, we calculated the PSNR and the SSIM

Fig. 7. Illustration of restoration of lost blocks using proposed error concealment algorithm: (top) pictures with 20% of content lost and (bottom) restored pictures using the proposed algorithm.

Video	UQI	PSNR	SSIM
Suzie	0.90069	41.20822	0.97837
Carphone	0.89881	38.91160	0.97383
Foreman	0.88333	39.17669	0.97625
Mobile	0.93128	38.19763	0.97615

TABLE I
UQI, PSNR AND SSIM VALUES CALCULATED BETWEEN ORIGINAL AND WATERMARKED VIDEOS.

values between the original and restored videos. The Figs. 11 and 12 depict the graphs of the values of PSNR and SSIM, respectively, versus the PLR values for all tested videos. In these graphs, the continuous lines represent the comparisons using the proposed method and the dashed lines represent the comparisons using the default H.264 error concealment algorithm. It can be observed that, as expected, the quality of the videos decreases with the PLR value. Also, the proposed algorithms always has a better performance (both in terms of PSNR and SSIM) than the default H.264 error concealment algorithm.

Fig. 10 depicts an example of the use of the proposed algorithm to mitigate lost packets for the videos "Foreman" and "Mobile". The first column of the figure shows sample frames of the original videos. The second column shows sample frames of the videos recovered with the default H.264 error concealment algorithm with 3% PLR. The third column shows sample frames of the videos restored using the proposed algorithm, also for 3% PLR. The method is able to get rid of common visible distortions that are commonly seen on videos restored using the standard H.264 error concealment algorithm, like for example blocking effects, packet losses, and false contours. In fact, we observed that, for PLR values below 5%, the restored videos present few distortions in comparison

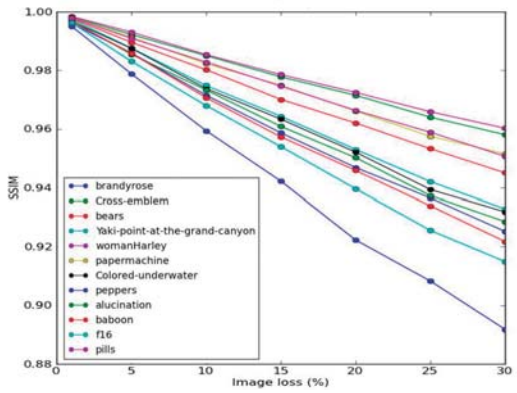


Fig. 8. SSIM values for reconstructed images versus percentage of deleted blocks.

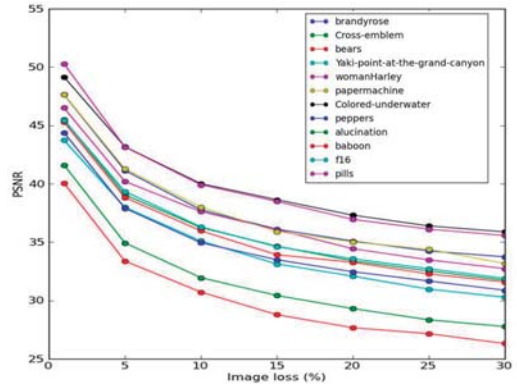
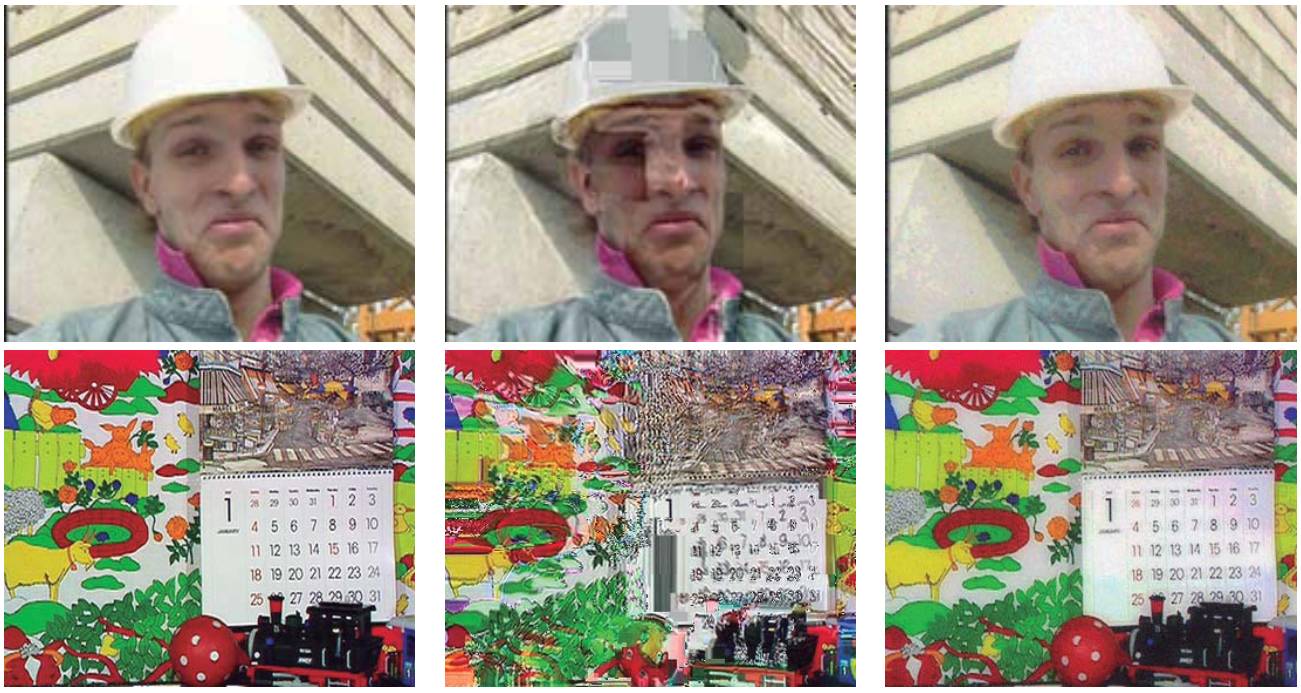


Fig. 9. PSNR values for reconstructed images versus percentage of deleted blocks.



(a) (b) (c)

Fig. 10. Visual result of a reconstructed frame after a lossy transmition. (a) Original frames, (b) restored frames using default H.264 error concealment algorithm, (c) restored frames using the proposed algorithm.

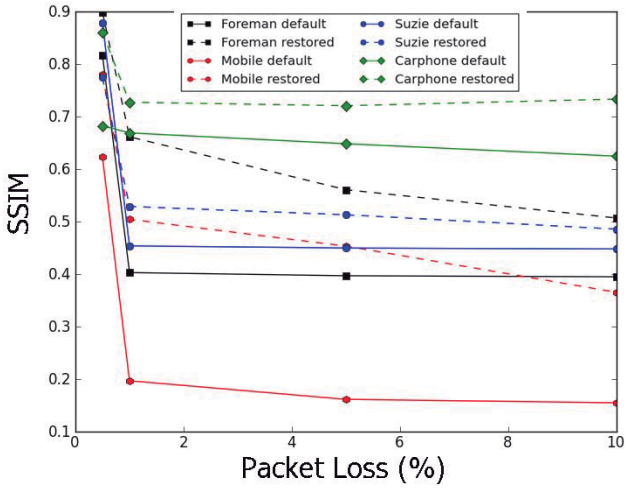


Fig. 11. SSIM values for for reconstructed videos versus packet loss rate (PLR) values.

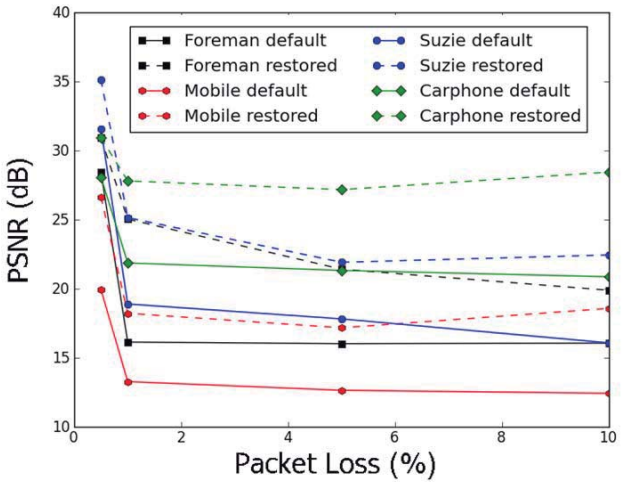


Fig. 12. PSNR values for reconstructed videos versus packet loss rate (PLR) values.

to the original videos.

IV. Conclusions

In this paper, we present a new technique for error concealment, which combines watermarking and halftoning algorithms. The proposed system is able to recover the original content with very good visual quality. The protected (watermarked) videos present good quality, showing few visible distortions. The proposed algorithm is fast and restores the picture frames with a better visual quality than the standard H.264 error concealment algorithm.

Future works include further investigating techniques to enhance the quality of marked pictures by reducing noise degradations created in the watermarking stage. Furthermore, in this paper the packages are removed randomly, which does not reflect the real errors that happens in a transmission. To

improve the quality of the reconstructed video, future works might include tuning the the algorithm to adjust the parameters according to the network loss model. Also, the inverse halftoning technique can be improved in order to provide a restored picture with a better quality. Moreover, the proposed algorithm can be modified to take into account aspects of the human visual system (HSV), such as the characteristics of color resolution and motion sensitivity. For example, we can insert the watermark in regions outside attention points, which makes it possible to quantize only the regions which are less perceptible to the viewer.

ACKNOWLEDGMENT

This work was supported in part by Conselho Nacional de Desenvolvimento Cientfico e Tecnológico (CNPq) Brazil and in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) Brazil.

REFERENCES

[1] S.-H. Yang and J.-C. Tsai, "A fast and efficient h.264 error concealment technique based on coding modes," in *Broadband Multimedia Systems and Broadcasting (BMSB), 2010 IEEE International Symposium on*, march 2010, pp. 1 –4.

[2] J. Zhou, B. Yan, and H. Gharavi, "Efficient motion vector interpolation for error concealment of h.264/avc," *Broadcasting, IEEE Transactions on*, vol. 57, no. 1, pp. 75 –80, march 2011.

[3] H. Sun, P. Liu, J. Wang, and S. Goto, "An efficient frame loss error concealment scheme based on tentative projection for h.264/avc," pp. 394–404, 2011, 10.1007/978-3-642-15696-037. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15696-037

[4] C. Adsumilli, M. Farias, S. Mitra, and M. Carli, "A robust error concealment technique using data hiding for image and video transmission over lossy channels," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 15, no. 11, pp. 1394 – 1406, nov. 2005.

[5] A. Phadikar and S. Maity, "Roi based error concealment of compressed object based image using qim data hiding and wavelet transform," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 2, pp. 971 –979, may 2010.

[6] C. K. Nayak, J. Surendran, S. N. Merchant, U. B. Desai, and S. Sanyal, "Error concealment of h.264 encoded video through a hybrid scheme," in *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, ser. MEDES '10. New York, NY, USA: ACM, 2010, pp. 189–195. [Online]. Available: http://doi.acm.org/10.1145/1936254.1936286

[7] B. Chen and G. Wornell, "Quantization index modulation: A class of provably good methods for digital watermarking and information embedding," *Information Theory, IEEE Transactions on*, vol. 47, no. 4, pp. 1423–1443, 2001.

[8] D. E. Knuth, "Digital halftones by dot diffusion," *ACM Trans. Graph.*, vol. 6, pp. 245–273, October 1987. [Online]. Available: http://doi.acm.org/10.1145/35039.35040

[9] J.-M. Guo and H. Lee, "Watermarking in halftone images with mixed halftone techniques," *Int. J. Imaging Syst. Technol.*, vol. 17, no. 5, pp. 303–314, Feb. 2007. [Online]. Available: http://dx.doi.org/10.1002/ima.v17:5

[10] D. Anastassiou and K. S. Pennington, "Digital halftoning of images," *IBM Journal of Research and Development*, vol. 26, no. 6, pp. 687 –697, nov. 1982.

[11] P. G. Freitas, M. C. Q. Farias, and A. P. F. d. Araújo, "Fast inverse halftoning algorithm for ordered dithered images," in *Proceedings...*, T. Lewiner and R. Torres, Eds., Conference on Graphics, Patterns and Images, 24. (SIBGRAPI). Los Alamitos: IEEE Computer Society Conference Publishing Services, 2011. [Online]. Available: http://urlib.net/sid.inpe.br/sibgrapi/2011/06.29.23.44

[12] J. S. Chen, A. Huertas, and G. Medioni, "Fast convolution with laplacian-of-gaussian masks," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-9, no. 4, pp. 584 –590, july 1987.

[13] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600 –612, april 2004.

[14] Z. Wang and A. C. Bovik, "A universal image quality index," *Signal Processing Letters, IEEE*, vol. 9, no. 3, pp. 81–84, 2002. [Online]. Available: http://dx.doi.org/10.1109/97.995823

Elsevier Editorial System(tm) for Pattern Recognition Letters

Manuscript Draft

Manuscript Number: PRLETTERS-D-13-00184

Title: Using Halftoning and Watermarking Techniques for Error Concealment and Tampering Detection

Article Type: Special Issue:SIBGRAPI12

Keywords: Quantization Index Modulation; Watermarking; Error Concealment; Inverse Halftoning; Tampering Detection

Corresponding Author: Mr. Pedro Garcia Freitas,

Corresponding Author's Institution: University of Brasília

First Author: Pedro Garcia Freitas

Pattern Recognition Letters
Authorship Confirmation

Please save a copy of this file, complete and upload as the "Confirmation of Authorship" file.

As corresponding author I, Pedro Garcia Freitas, hereby confirm on behalf of all authors that:

1. This manuscript, or a large part of it, has not been published, was not, and is not being submitted to any other journal. If presented at a conference, the conference is identified. If published in conference proceedings, the publication is identified below and substantial justification for re-publication must be presented.

2. All text and graphics, except for those marked with sources, are original works of the authors, and all necessary permissions for publication were secured prior to submission of the manuscript.

3. All authors each made a significant contribution to the research reported and have read and approved the submitted manuscript.

Date February, 8th 2013.

Previous conference presentation

SIBGRAPI 2012 – Conference on Graphics, Patterns and Images

Previous conference proceedings publication

SIBGRAPI Digital Libray Archive

Justification for re-publication

The submitted document is an extended version of previous proceeding publication. New results were added to this new version of the paper, which included a new application of the algorithm (a tampering application for video), and, most importantly, new tests and procedures.

Highlights

- We presented an approach to protect image and videos against digital forgery.
- We propose a technique to restore lost or tampered information in visual signals.
- The restoration is done using a proposed inverse halftoning technique.
- The halftone of signal is embedded in the signal itself using a QIM modification.

Using Halftoning and Watermarking Techniques for Error Concealment and Tampering Detection

Pedro Garcia Freitas[c], Ronaldo Rigoni[c], Mylène C.Q. Farias[d], Aletéia P.F. Araújo[c]

[a]*Department of Computer Science, University of Brasília, Brazil*
[b]*Department of Electrical Engineering, University of Brasília, Brazil*

Email addresses: sawp@sawp.com.br, Phone: +55 (61) 3107-3661, Fax: +55 (61) 3107-3662 (Pedro Garcia Freitas), rrigoni@gmail.com (Ronaldo Rigoni), mylene@ieee.org (Mylène C.Q. Farias), aleteia@cic.unb.br (Aletéia P.F. Araújo)

URL: www.sawp.com.br (Pedro Garcia Freitas), www.ronaldorigoni.com.br (Ronaldo Rigoni), www.pgea.unb.br/~mylene/ (Mylène C.Q. Farias)

Using Halftoning and Watermarking Techniques for Error Concealment and Tampering Detection

Pedro Garcia Freitas[c], Ronaldo Rigoni[c], Mylène C.Q. Farias[d], Aletéia P.F. Araújo[c]

[c]*Department of Computer Science, University of Brasília, Brazil*
[d]*Department of Electrical Engineering, University of Brasília, Brazil*

Abstract

In this paper we present a system with the goal of protecting and restoring *lost* or *tampered* information in images or videos. With the proposed system it is possible to implement both an *error concealment algorithm* and a *tampering detection algorithm*. The system is based on watermarking and halftoning techniques. At the encoder side, the algorithm generates a binary version (mark) of the original image or video frame (picture) using a halftoning technique. Then, a watermarking technique is used to embed this mark into the host picture. At the decoder side, after the lost or tampered regions are identified, the original content is recovered by extracting the dithered mark corresponding to the affected areas. An inverse halftoning algorithm is used to convert the dithered version of the picture into a good quality colored multi-level approximation of the original content.

Keywords: Quantization Index Modulation, Watermarking, Error Concealment, Inverse Halftoning, Tampering Detection

Email addresses: sawp@sawp.com.br, Phone: +55 (61) 3107-3661, Fax: +55 (61) 3107-3662 (Pedro Garcia Freitas), rrigoni@gmail.com (Ronaldo Rigoni), mylene@ieee.org (Mylène C.Q. Farias), aleteia@cic.unb.br (Aletéia P.F. Araújo)

URL: www.sawp.com.br (Pedro Garcia Freitas), www.ronaldorigoni.com.br (Ronaldo Rigoni), www.pgea.unb.br/~mylene/ (Mylène C.Q. Farias)

1. Introduction

The flexibility of digital images and videos is both a blessing and a curse. Digital technologies make it possible to create high quality pictures, animations, games, and special effects with an amazing realism. Digital pictures (images and videos) can be enhanced, compressed, transmitted, translated across different standards, and displayed in a variety of devices. Then, because of the significant advances in compression and transmission techniques, it is possible to deliver high quality visual content to the end user in many different ways. As a consequence, a variety of delivery services have been created in the last years, such as direct TV broadcast satellite, digital broadcast television, and IP-based video streaming.

Unfortunately, the transmission of visual content over wired and wireless channels still introduces multiple losses into the transmitted data that manifest themselves as various types of visible degradations. It is worth pointing out that, up to date, the Internet provides a best effort model, which means losses may happen when packets are being transmitted. In fact, during the course of a transmission errors are introduced according to the channel conditions, which causes disturbing variations in the quality of the received content.

In particular, video signals are transmitted in compressed format (bitstreams)[1], which means that a lot of visual information is placed into a few packets that are not equally important. Therefore, videos are particularly sensitive to transmission losses that affect the quality of the received video and, consequently, the level of acceptability and popularity of the service. To minimize the effect of transmission errors, error concealment algorithms are frequently used at the decoder or receiver side. Most error concealment algorithms use error prediction techniques like interpolation [2, 3]. Other algorithms use watermarking techniques to embed redundant information in the signal. For example, the work of Adsumilli *et al.* [4] uses a spread-spectrum watermarking technique to embed a dithered version of the picture frame into the host video. The work of Navak *et al.* [5] improved on Adsumilli *et al.*'s work

by adding motion vector estimation and edge-correlated information. Unfortunately, both works still produce reconstructed areas with a low quality. For these approaches, the quality of the reconstructed areas can only be improved by increasing the data hiding capacity and the quality of the inverse halftoning techniques.

Another very important concern for image and video transmission and storage applications is tamper detection and copyright protection [6]. Powerful softwares are currently available, making it easy to alter (tamper) visual digital content without leaving any clear sign of these modifications. As a consequence, automatic methods for checking the authenticity and integrity of digital images and videos are, undoubtedly, very important. Several techniques have been proposed with the goal of detecting tampering of digital content [6]. These techniques can be divided in approaches that do not require the original (no-reference) and approaches that do require the reference (full reference). Since in most transmission applications the original is not available, techniques no reference approaches are the most adequate ones.

Most of the no-reference tampering detection techniques are specialized in detecting only one type of modification [7, 8], what is not always useful in practical applications. One possible approach used by no-reference tampering detection techniques consists of using watermarking to embed invisible information into the host video [10-11]. To verify if the original content was tampered, the embedded information is extracted and its integrity is verified. In this approach, the *fragility* of the embedded mark is a key element that determines the amount of tampering that the algorithm is able to detect. Among the tampering detection algorithms that use this approach, we can cite the work of Wolfgang and Delp [9] that detects tampered areas by adding m-sequences of -1 and 1 to 8x8 blocks, and the work of Yeung and Mintzer [10] that detects individual pixel modifications using a verification key. Among the methods available in the literature, few address the problem of detecting

the location of tampered areas and restoring the original content with good quality [11]. Up to our knowledge, there are no watermark-based tampering detection algorithms for digital videos that is able to detect restored areas with a good quality.

In this paper, we present a system with the goal of protecting and restoring *lost* or *tampered* information in images or videos. With the proposed system it is possible to implement both an *error concealment algorithm* and a *tampering detection algorithm*. The system is based on watermarking and halftoning techniques. At the encoder side, the algorithm generates a binary version (mark) of the original image or video frame (picture) using a halftoning technique. Then, a watermarking technique is used to embed this mark into the host content. The watermarking technique used by the system is a simple modification of the Quantization Index Modulation (QIM) algorithm, which allows a slightly higher data hiding capacity [12]. For tampering detection, a ciphered key is also embedded into the host video to allow spatial and temporal localization of tampered regions. At the decoder side, after the lost or tampered regions are identified, the original content is recovered by extracting the dithered mark corresponding to the affected areas. An inverse halftoning algorithm is used to convert the dithered version of the picture into a good quality colored multi-level approximation of the original picture.

The paper is divided as follows. In Sections 2 and 3, the halftoning and watermarking embedding stages are described. In Sections IV and V, the watermarking extraction and inverse halftoning stages are detailed. In Sections VI and VII, we describe the implementation and simulation results of the two main applications of the proposed system: an error concealment and a tampering detection algorithm, respectively. Finally, in Section VIII the conclusions are discussed.

2. Halftoning stage

Halftoning is a technique for converting multi-level images into binary images using patterns of white and black dots [13]. This technique creates the illusion of seeing multiple intensity levels in a binary image, what makes it suitable for applications where only a reduced number of levels is available, such as newspapers, fax machines, and document printing processes.

In this work, a halftoning algorithm is used to generate a dithered version of each picture (still image or video frame), which is later embedded into the host picture itself. At the decoder, if any loss or tampered area is detected, the dithered version of the picture is recovered and used to restore the content back to its original state. Therefore, the quality of the restored image or video depends strongly on the efficiency of the halftoning algorithm. One of the contributions of this work is the design of a halftoning and an inverse-halftoning algorithms that are able to generate simple dithered images that can be later inverted with a very good quality.

In order to guarantee a good quality, the halftoning algorithm must be able to represent the largest possible number of intensity levels with minimum number of bits. Also, since (at the decoder/receiver side) parts of the picture might be lost, the halftoning algorithm needs to use only local information to generate the dithered picture and revert it. The simplest halftoning technique that satisfies these requirements is the *ordered dithering algorithm*. This class of algorithms generates dithered pictures with sets of pixel clusters that have a predictable pattern, what is very important for our target applications (transmission and storage of videos and images).

Given that the data hiding capacity of the host picture frame is relatively low, we do not use classical ordered dithering algorithm. We use combinatorial dispersed-dot patterns because they are capable of generating all combinations of bits necessary to represent a

number of intensity levels [14, 15]. This way, we can increase the number of mapped intervals without increasing the size of dot-pattern matrix. For example, using a 3x3 Bayesian matrix to generate a dispersed-dot dithering we can represent 10 distinct intensity levels. On the other hand, using a 3x3 combinatorial matrix allows for up to 512 intensity levels. Since our tests showed that a maximum of 3 bits per pixel (per color channel) can be embedded in the host picture without causing visible degradations, we can use combinatorial matrices to generate 2x3 combinations, allowing mapping up to 8 different intervals.

Since pictures reconstructed from dispersed-dot dithered images can be slightly blurred, before we generate the halftoning picture we apply an unsharp-masking edge enhancement filter to the original picture frame (I_o), generating an image with enhanced details (I_{eh}). Then, we quantize I_{eh} using 8 distinct intervals, using the following equation:

$$I_Q(x, y, c) = \left\lceil \frac{8}{255} I_{eh}(x, y, c) \right\rceil , \tag{1}$$

in which x and y are the horizontal and vertical spatial dimensions, respectively, c refers to the color channel ($1 \leq c \leq 3$), and I_Q is the resulting dithered image. As mentioned before, up to 3 bits can be embedded in each color channel. This means that the dithered mark (I_Q) must be represented with a total of 9 bits per pixel. So, we substitute each value of $I_Q(x, y, c)$ by the corresponding 3-bits combinatorial dispersed-dot patterns shown in Figure 1(a).

In case the target application requires that the spatial and temporal position of the lost or tampered areas be identified before restoration, 1 bit out of the available 9 bits should be used to store a ciphered key code. In these cases, we are left with 8 bits to represent the dithered version of each color channel of the image or video frame. Since the Human Visual System (HVS) is less sensitive to degradations in the blue channel, we opt to use 2 bits to represent the dithered version of the blue channel and 3 bits to represent the dithered version of the red and green channels. In other words, for the red and green channels, we substitute

each 8-bit pixel of I_Q by one of the 3-bits dot-patterns shown in Figure 1(a), while for the blue channel we substitute each 8-bit pixel of I_Q by one of the 2-bits dot-patterns shown in Figure 1(b).

Before embedding this information into the host picture, we concatenate the resulting 3 bits corresponding to each pixel and generate an integer number. This way, the resulting halftoning image is actually an integer dithered image (I_{dth}), with values ranging from 0 to 7. This integer mark can be easily embedded in the original image, without requiring extra space.

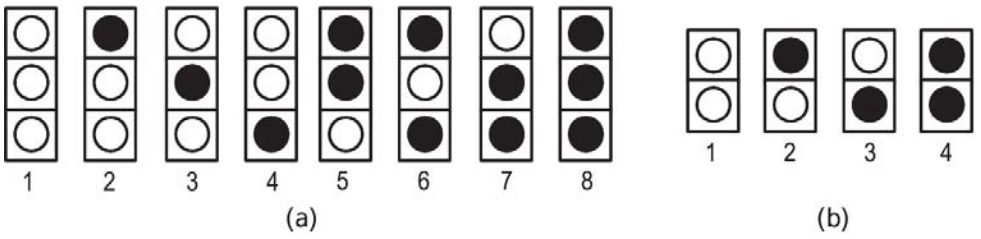


Figure 1: Combinatorial dispersed-dot patterns: (a) 2-bits dispersed-dot patterns used for mapping 4 intensity levels and (b) 3-bits dispersed-dot patterns used for mapping 8 intensity levels

3. Watermarking Embedding Stage

After generating the mark using the techniques described in the previous section, the next stage consists of embedding it into the host image or video. But, in order to make it possible to recover the original content, the dithered mark corresponding to a specific region cannot be embedded in the same spatial and temporal position of the host image or video. Therefore, we spatially distribute the mark over the host image or video frame using a *split-flip* operation, which consists of splitting the halftone image into sub-blocks and flipping them to a different spatial region. More specifically, we divided the picture in 32x32 sub-blocks, rotated each sub-block by 180 degrees, and shuffled the regions. Figure 2 shows one example of the process for the three color channels of the image Lena.

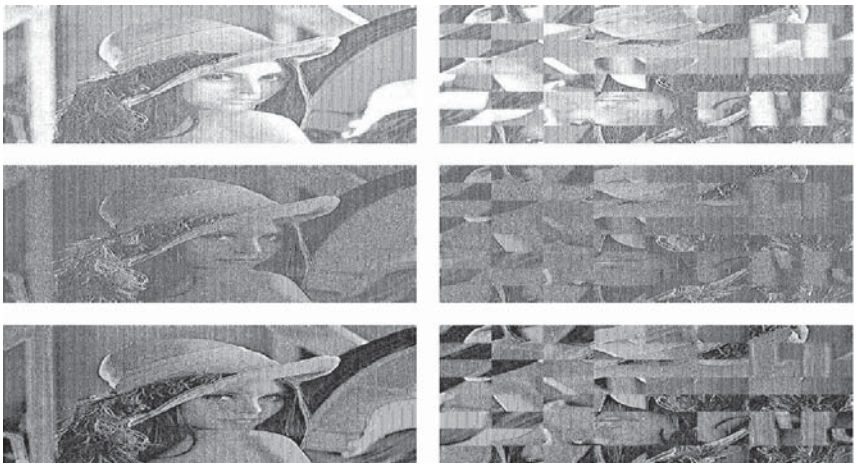


Figure 2: Original dithered versions of each color channel of the image Lena (left) and the corresponding split-flip versions (right).

For videos signals, we also perform an embedding temporal distribution by inserting the mark corresponding to the current picture frame in a previous picture frame, located 1 second before. By distributing the mark spatially and temporally, a region does not store the mark necessary to restore it, what increases the probability that the algorithm is able to restore the content to its original version. The mark is also encrypted with AES-256 block Cipher to protect it from being extracted by an unauthorized user [13].

Among the available watermarking methods, the Quantization Index Modulation (QIM) algorithm is one of the methods with the best performance [12]. The QIM algorithm inserts a mark into a host signal by quantizing it with a uniform scalar quantizer. The standard quantization operation with step size δ is given by the following equation:

$$Q(x, \delta) = \text{round}\left(\frac{x}{\delta}\right),$$

where $round(.)$ denotes the mathematical operation of rounding a value to the nearest integer. The watermarked pixel is obtained using the following equation:

$$s(x) = Q(x, \delta) + d(m),$$

in which $d(m)$ is the perturbation value, which depends on the mark signal m to be embedded.

In this work, we propose a modification of the QIM algorithm that inserts an integer mark, instead of a function of the 1-bit mark. Also, the modulation function is set to $d(m) = m$. So, the integer dithered image (I_{dth}) is inserted in each pixel of the corresponding color channel of the original picture using the following equation:

$$I_m(x, y, c) = Q(I_o(x, y, c), \delta) + I_{dth}(x, y, c),$$

where I_o is the original color channel of the picture frame, I_m is the resulting watermarked color channel, δ is the quantization step, and c is the corresponding color channel.

4. Watermarking Extraction Stage

The watermarking extraction is performed at the receiver or decoder side, after a possible transmission or processing of the image or video. When a lost or tampered region is detected by the decoder, we extract the corresponding mark using the following equation:

$$\hat{I}_{dth}(x, y, c) = I_m(x, y, c) \bmod \delta,$$

where I_m is the watermarked color channel and $\hat{I}_{dth}$ is the recovered integer dithered mark. Then, the extracted integer dithered mark is converted to a 3-bit binary number that results in the recovered dithered picture.

After the extracted mark is recovered, an authorized user can decipher the encrypted mark using the symmetric key, provided by the owner of the content. To ensure security, this symmetric key can be shared among authorized users using an asymmetric-key approach.

This guarantees that an unauthorized user is not able to identify or remove the embedded mark.

To detect if a region of the image or video is lost or tampered with, we compare the cipher detection key with the secret key contained in the mark extracted from every pixel of the blue channel of the dithered image. If differences are found, the pixels where the differences were found are classified as tampered. In applications like error concealment, where the position of lost areas are identified by the decoder, the key is not needed and the extra bit can be used to convey a better quality for the reconstructed areas.

5. Inverse Halftoning Stage

If any region is classified as lost or tampered, we search the correspondent mark in the appropriate location. Then, to generate a multi-level colored picture frame, we propose the following inverse halftoning algorithm [15]. Given that I_{dth} is the dithered picture frame, $D(p)$ is the distribution of the area surrounding the pixel p in I_{dth}. To reconstruct an 8-bit pixel from the dithered picture, we first calculate the local distribution $D(p)$ for all pixels in I_{dth}. From this distribution, we find the corresponding mapped interval that contains the most probable pixel value in the corresponding color channel, according with the indices of the dot-patterns.

Once this interval is found, we randomly select a value within it, generating a slightly noisy picture I_{inv}. Then, we filter I_{inv} with 2 filters: a Gaussian lowpass and a Laplacian-of-Gaussian. The idea here is to spatially decorrelate the pixels in order to be able to process each pixel independently. This allows for an independent reconstruction, even when the neighboring pixels are missing. The resulting pictures, I_{gauss} and I_{log}, are used to compose another picture, I_{bld}, given by the following equation:

$$I_{bld}(x, y, c) = \gamma I_{\text{gauss}}(x, y, c) + (1 - \gamma) I_{\log}(x, y, c),$$

where γ is the blending-ratio matrix that determines the proportion of each input filtered picture in the output. In our simulations, we observed that using $\gamma = I_{inv}$ preserves the edges of the pictures.

Unfortunately, when we combine all 3 channels, the resulting picture frame contains visible color distortions. If, on the other hand, we use γ as a constant matrix, $Ibld$ is a blurred version of I_{inv}. Therefore, with the goal of minimizing color distortion and keeping the details of the original image, it is necessary to make another composition of I_{inv} and I_{bld}, given by the following equation:

$$\hat{I}_o(x, y, c) = \left\lfloor \sqrt{I_{\text{col}}(x, y, c) I_{bld}(x, y, c)} \right\rfloor$$

where $\hat{I}_o$ is the recovered 8-bit version of original video frame, I_o. The final result of inverse halftoning process, $\hat{I}_o$, is a picture the best visual quality possible, as compared to the techniques available in the literature. Obviously, the procedure can be applied to videos and images alike. In the next sections, we describe two target applications of the proposed system: an error concealment and a tamper detection algorithm.

6. Error Concealment Algorithm

We used the techniques described in the previous sections to design an error concealment algorithm. The algorithm is designed to be integrated into a compression codec, but can be also used independently of the codec technology. Since decoders are able to identify which packets were lost, there is no need for a key code. We used the system proposed in the previous sections to insert the dithered version of the image or video frames using 3 bits for each color channel (see Section II, Figure 1(a)) to increase the quality of restored areas.

First, we tested our algorithm using a set of still images modified with a percentage of 16×16 blocks deleted. To implement this test, we divided the image in blocks of 16×16.

Then, we randomly chose a percentage of these blocks and substitute the original content by the value 0. We, then, varied the percentage of lost blocks from 5% to 25%. Figure 3 shows three examples of the restoration of lost blocks using proposed error concealment algorithm. The images on the left column have 20% of blocks discarded, while the images on the right column are the corresponding restored versions. Notice that the quality of the reconstructed image is excellent and it is very difficult to identify the location of the restored blocks.

We calculated the Peak signal-to-noise ratio (PSNR) and the Structural similarity (SSIM) [16] values between the original and restored images of the test cases. Figures 4 (a) and (b) depict the graphs of SSIM and PSNR, respectively, versus the percentage of deleted blocks for all test images. It can be observed that, as expected, the quality of the images decreases with the percentage of deleted blocks.

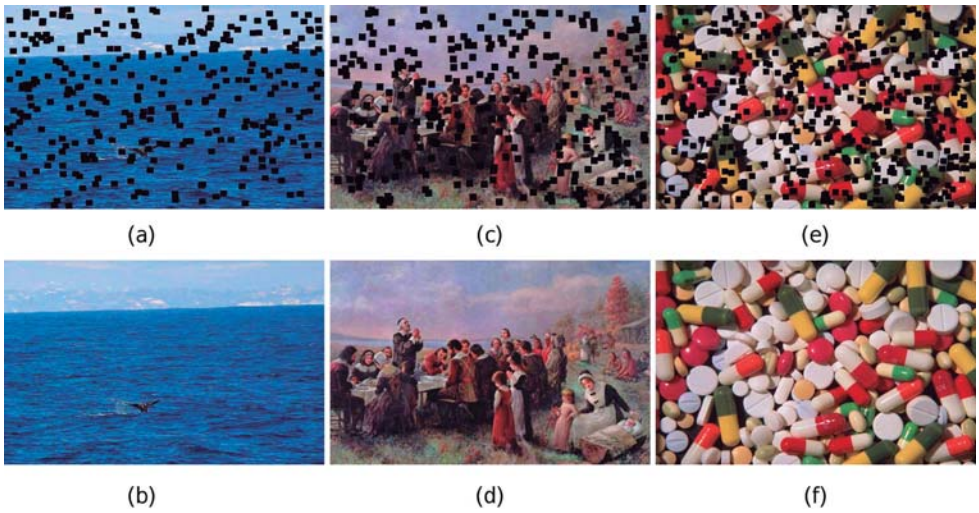


Figure 3: Illustration of restoration of lost blocks using proposed error concealment algorithm: (a), (c) and (e) pictures with 20% of content lost; (b), (d), and (e) restored pictures using the proposed algorithm.

Second, we tested the ability of the proposed algorithm to recover lost packets from H.264 compressed videos. We used publicly-available videos in YUV 4:4:4 color CIF (352 x 288, progressive) format with around 300 frames each. The videos used were 'Foreman', 'Mobile','Carphone', and 'Suzie' [17]. We embedded the proposed error concealment algorithm

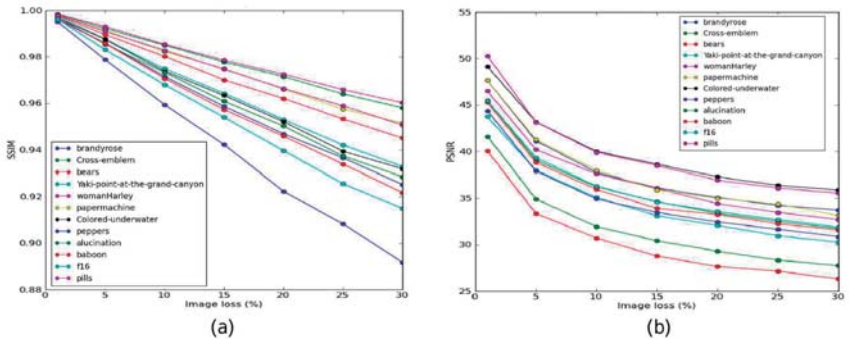


Figure 4: (a) SSIM and (b) PSNR for reconstructed images versus percentage of deleted blocks.

in a H.264 codec to obtain a protected video stream. In order to simulate packet losses in a given bitstream, we use a simple model that simulates packet losses over error-prone channels. We tested Packet Loss Rates (PLR) of 0.5%, 1%, 3%, 5%, and 10%. In simulations, all packets were treated equally regarding their susceptibility to loss.

Figure 5 depicts an example of the use of the proposed algorithm to mitigate lost packets for the videos Foreman and Mobile. The first row of the figure shows sample frames of the original videos. The second row shows sample frames of the videos recovered with the *default* H.264 error concealment algorithm [1] with 3% PLR. The third row shows sample frames of the videos restored using the proposed algorithm, also with 3% PLR. As can be noticed from the sample frames in Figure 3, the method is able to get rid of common visible distortions that are commonly seen on videos restored using the standard H.264 error concealment algorithm, like for example blocking effects, packet losses, and false contours. In fact, we observed that, for PLR values below 5%, the restored videos present very few distortions in comparison to the original videos.

We, then, calculated the PSNR and SSIM values between the original and restored videos for all test cases. Figures 6(a) and (b) depict the graphs of SSIM and PSNR, respectively, versus the PLR values for all videos. In these graphs, the continuous line represents the comparisons using the proposed method and the dashed lines represent the comparisons

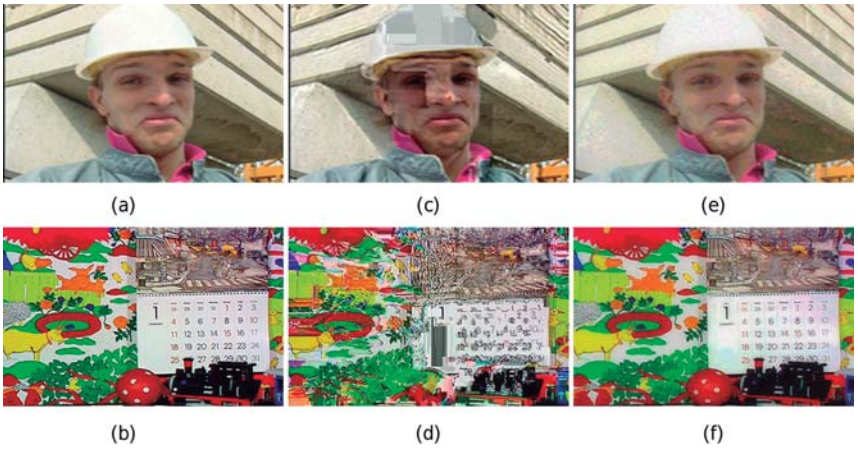


Figure 5: From top to bottom: (a) and (b) original frames, (c) and (d) restored using the default H.264 error concealment algorithm, (e) and (f) restored using the proposed algorithm.

using the default H.264 error concealment algorithm. It can be observed that, as expected, the quality of the videos decreases with the PLR. In summary, the proposed algorithm has always a much better performance than the default H.264 error concealment algorithm, both in terms of PSNR and SSIM.

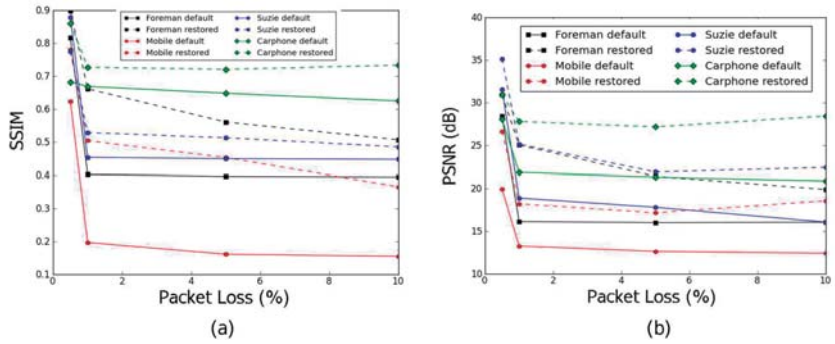


Figure 6: (a) SSIM and (b) PSNR for reconstructed videos versus packet loss rate values. The label 'default' refers to the standard H.264 error concealment algorithm, while 'restored' refers to the proposed error concealment algorithm.

7. Tampering detection Algorithm

As pointed in the Section I, tampering detection of video and image content is also a crucial problem in multimedia applications. We used the techniques described in Sections I-V

to design a tampering detection algorithm that is able not only to detect tampered regions, but also to recover the original content. For that, we used one of the 9 bits embedded in the host image or video frame to store a secret key code (see Section II). As described earlier, the dithered versions of the red and green channel use 3 bits each, while the dithered version of the blue channel uses only 2 bits.

First, we tested the proposed algorithm using still images with different characteristics: high-detailed, low-detailed, color, grayscale, documents, landscape, person pictures, etc. Different kinds of attacks were applied to these images: blurring of selected/random areas, noise addition, cut-and-paste, region deletion, and resizing. For all test cases, we were able to detect tampered regions in 100% of the cases. Also, there were no false-positives or false-negatives. In terms of reconstruction, the algorithm was able to recover tampered regions with good quality, as long as the tampered regions did not exceed 50% of the image.

Figures 7 and 8 depict two examples of the use of the proposed algorithm to detect tampered regions in still images. The first row of the examples (from left to right) shows the original (untampered) images, the tampered images, and the regions detected as tampered. The second row (from left to right) shows the recovered tampered regions, an image recovered without the key, and the image reconstructed with original content restored. As can be observed, the algorithm is able to detect tampered regions, independent of their size. Also, since the embedded halftone watermark is encrypted, an unauthorized user is unable to know if an image was tampered and, consequently, unable to reconstruct the image back to its original state (see Figures 7(e) and 8(e)). The algorithm is theoretically able to restore content of tampered images with good quality, if at least 50% of the regions of the embedded halftone are intact (see Figures 7(f) and 8(f)).

We also tested the performance of the algorithm for tampering detection and recovery of digital videos. As in the previous examples, we used publicly-available videos in YUV

4:4:4 color CIF (352 x 288, progressive) format with around 300 frames each, downloaded from the Video Trace Library [17] and from the Consumer Digital Video Library [18]. We tested the following attacks: blurring of selected areas, cut-and-paste, region-deletion, and object-addition. Again, for all test cases, we were able to detect tampered regions in 100% of the cases. There were no false-positives or false-negatives. In terms of reconstruction, the algorithm was able to recover tampered regions with good quality, as long as the tampered regions did not exceed 50% of the frame.

The Figure 9 depict examples of the use of the proposed algorithm to detect tampered regions for the 3 different types of attacks mentioned on paragraph above. In this figure, the image (a) shows the original frame content. The Figures 9(b), (e) and (h) shows the attacked frames using blurring, cut-and-paste and object-addition, respectively. In addiction, the Figures 9(c), (f) and (i) shows the tampered areas using these attacks, while the Figures 9(d), (g) and (j) shows the frames with the original contents restored. Notice that the algorithm is able to detect tampered regions, independent of their size, position or the level of difference in comparison to the original. Also, the proposed algorithm is able to restore content of tampered images with admissible quality.

8. Conclusions and future work

In this paper, we presented a system with the goal of protecting and restoring lost or tampered information in images or videos. With the proposed system it is possible to implement both an error concealment algorithm and a tampering detection algorithm. The system is based on watermarking and halftoning techniques. In order to increase the data hiding capacity, the work proposed a simple modification of the QIM watermarking algorithm. To obtain higher quality restored areas, improved inverse halftoning algorithms are also proposed. A secret key code is embedded to the host content to identify the spatial and temporal positions of tampered regions, taking advantage of the lower sensitivity of the

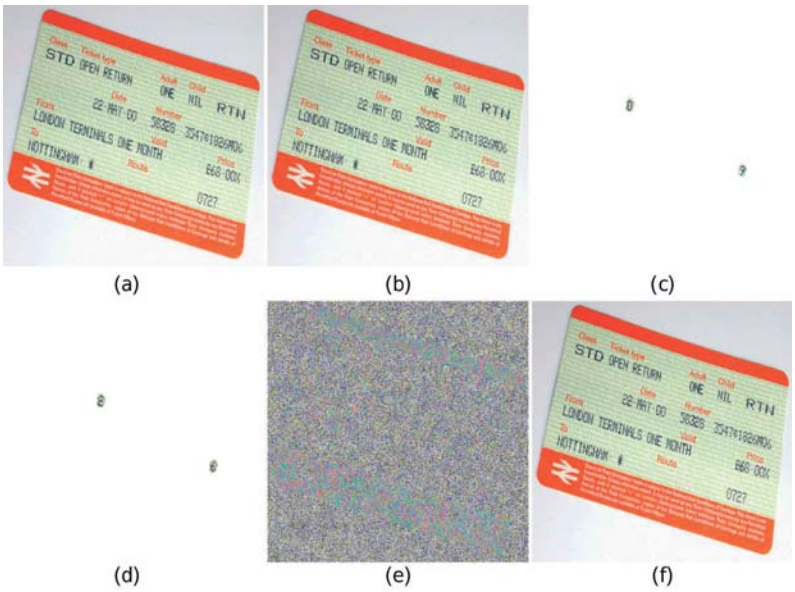


Figure 7: Identification and restoration of tampered regions in the 'train ticket' image, attacked using 'cut-and-paste': (a) original, (b) tampered, (c) regions identified as tampered, (d) restored regions, (e) unauthorized recovered mark, (f) restored image.

HVS to degradations in the blue color channel. The proposed algorithm presented good performance, being able to identify tampered or lost areas and restore them with very good quality.

Future work includes an increase of the data hiding capacity with the goal of embedding more information. With that, the quality of the restored content can be increased by adding. Additional bits can also be used to improve the protection of the data against tampering. For example, using some bits to embed additional temporal information can help counter other attacks, such as frame shuffle.

Acknowledgements

This work was supported in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) Brazil and in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) Brazil.

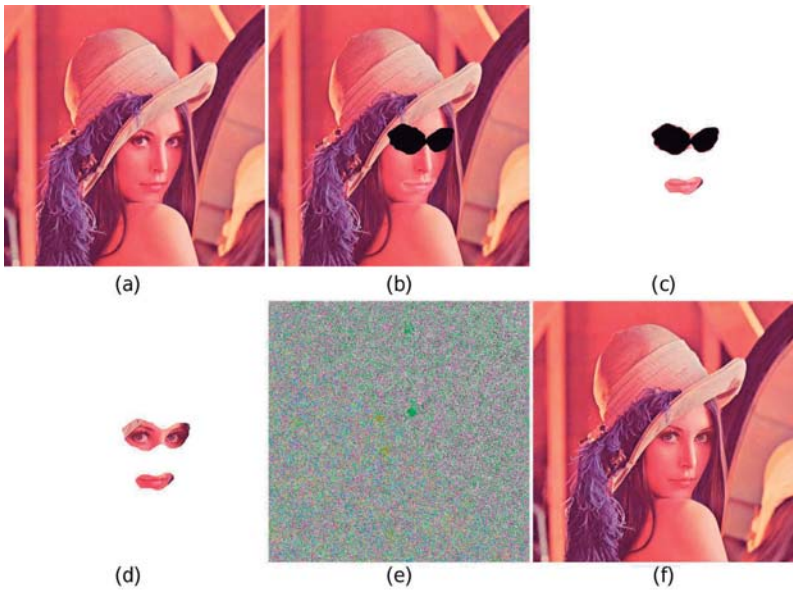


Figure 8: Identification and restoration of tampered regions of 'Lena' image, attacked using 'blurring' and 'data elimination': (a) original, (b) tampered, (c) regions identified as "tampered", (d) restored regions, (e) unauthorized recovered mark, (f) restored image.

References

[1] S.-H. Yang, J.-C. Tsai, A fast and efficient h.264 error concealment technique based on coding modes, in: Broadband Multimedia Systems and Broadcasting (BMSB), 2010 IEEE International Symposium on, 2010, pp. 1 –4.

[2] J. Zhou, B. Yan, H. Gharavi, Efficient motion vector interpolation for error concealment of h.264/avc, Broadcasting, IEEE Transactions on 57 (1) (2011) 75 –80.

[3] H. Sun, P. Liu, J. Wang, S. Goto, An efficient frame loss error concealment scheme based on tentative projection for h.264/avc (2011).

[4] C. Adsumilli, M. Farias, S. Mitra, M. Carli, A robust error concealment technique using data hiding for image and video transmission over lossy channels, Circuits and Systems for Video Technology, IEEE Transactions on 15 (11) (2005) 1394 – 1406.

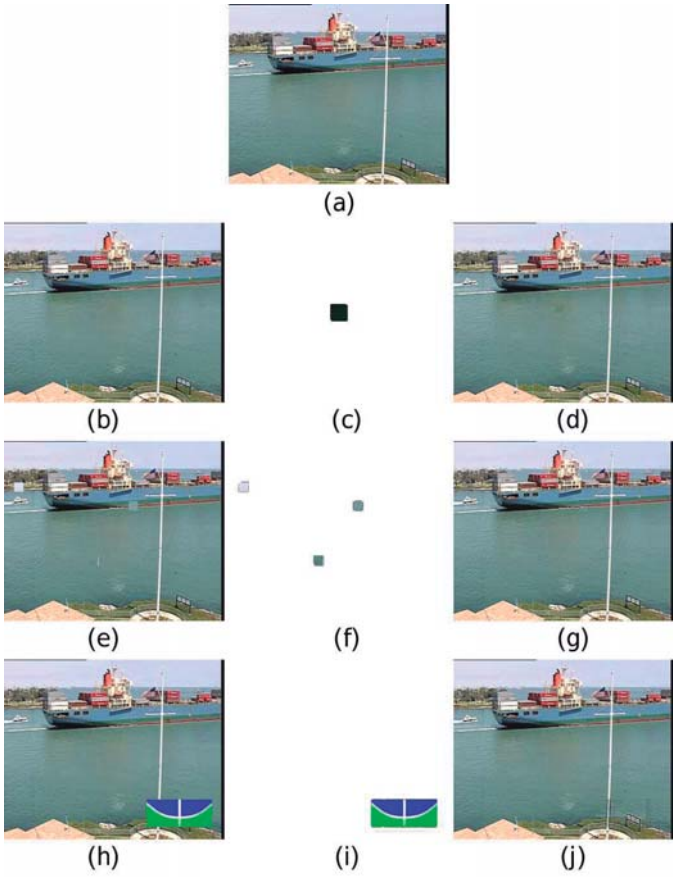


Figure 9: Identification and restoration of tampered regions for sample frames of video "Container": (a) original; (b), (e) and (h) tampered frames using blurring, cut-and-paste, and 'adding a new object' attacks; (c), (f), and (i) identified tampered regions; (d), (g), and (j) restored frames.

[5] C. K. Nayak, J. Surendran, S. N. Merchant, U. B. Desai, S. Sanyal, Error concealment of h.264 encoded video through a hybrid scheme, in: Proceedings of the International Conference on Management of Emergent Digital EcoSystems, MEDES '10, ACM, New York, NY, USA, 2010, pp. 189–195.

[6] Digital image forensics: a booklet for beginners, Multimedia Tools and Applications 51.

[7] T.-T. Ng, S.-F. Chang, Q. Sun, Blind detection of photomontage using higher order statistics, in: Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on, Vol. 5, 2004, pp. V–688 – V–691 Vol.5.

[8] F. Peng, X. lan Wang, Digital image forgery forensics by using blur estimation and abnormal hue detection, in: Photonics and Optoelectronic (SOPO), 2010 Symposium on, 2010, pp. 1 –4.

[9] R. Wolfgang, E. Delp, A watermark for digital images, in: Image Processing, 1996. Proceedings., International Conference on, Vol. 3, 1996, pp. 219 –222 vol.3.

[10] M. Yeung, F. Mintzer, An invisible watermarking technique for image verification, in: Image Processing, 1997. Proceedings., International Conference on, Vol. 2, 1997, pp. 680 –683 vol.2.

[11] A. Phadikar, S. P. Maity, M. Mandal, Novel wavelet-based qim data hiding technique for tamper detection and correction of digital images, Journal of Visual Communication and Image Representation 23 (3) (2012) 454 – 466.

[12] B. Chen, G. Wornell, Quantization index modulation: A class of provably good methods for digital watermarking and information embedding, Information Theory, IEEE Transactions on 47 (4) (2001) 1423–1443.

[13] H. Soleimany, A. Sharifi, M. Aref, Improved related-key boomerang cryptanalysis of aes-256, in: Information Science and Applications (ICISA), 2010 International Conference on, 2010, pp. 1 –7.

[14] D. E. Knuth, Digital halftones by dot diffusion, ACM Trans. Graph. 6 (1987) 245–273.

[15] P. Freitas, M. Farias, A. de Araujo, Fast inverse halftoning algorithm for ordered dithered images, in: Graphics, Patterns and Images (Sibgrapi), 2011 24th SIBGRAPI Conference on, 2011, pp. 250 –257.

[16] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, Image quality assessment: From

error visibility to structural similarity, IEEE Transactions on Image Processing 13 (4) (2004) 600–612.

[17] Video trace library of arizona state university (asu), http://trace.eas.asu.edu/yuv.

[18] The consumer digital video library (cvdl), http://www.cdvl.org.