UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCELO SCHIAVON PORTO

Desenvolvimento Algorítmico e Arquitetural para a Estimação de Movimento na Compressão de Vídeo de Alta Definição

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. Sergio Bampi
Orientador

Prof. Dr. Luciano Volcan Agostini (UFPel)
Co-orientador

Porto Alegre, fevereiro de 2012.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Porto, Marcelo Schiavon

Desenvolvimento Algorítmico e Arquitetural para a Estimação de Movimento na Compressão de Vídeo de Alta Definição [manuscrito] / Marcelo Schiavon Porto. – 2012.

166 f.:il.

Orientador: Sergio Bampi; Co-orientador: Luciano Volcan Agostini.

Tese (Doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2012.

1. Estimação de movimento 2. Desenvolvimento algorítmico, 3. Desenvolvimento arquitetural 4. Vídeos de alta definição. I. Bampi, Sergio. II. Agostini, Luciano Volcan. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Álvaro Freitas Moreira
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Muitas pessoas foram importantes para mim durante todo este período de doutorado, e merecem ser lembradas nestes agradecimentos. Agradeço aos meus pais Roberto Porto e Ivone Porto pela criação que me deram e pelos valores que me ensinaram ao longo da vida. Muito obrigado pelo apoio dado ao longo de toda a minha formação acadêmica, e que não foi diferente durante este doutorado.

Gostaria de agradecer também a minha tia Maria Helena Schiavon, que sempre me apoiou em todas as etapas da minha vida e é como uma segunda mãe pra mim.

Um agradecimento especial a minha namorada e companheira Júlia Islabão, que esteve ao meu lado durante estes quatro anos de doutorado, enfrentando os momentos difíceis e a distância durante boa parte deste período. Muito obrigado por estar ao meu lado, te amo muito.

Gostaria de agradecer ao grande professor Sergio Bampi, meu orientador. Muito obrigado por todas as oportunidades que me destes e pelos conhecimentos que compartilhastes comigo durante estes últimos seis anos, desde o mestrado. Muito obrigado pela confiança que depositastes em mim quando me escolheste como teu aluno de doutorado, esta foi uma oportunidade única na minha carreira e espero ter correspondido as suas expectativas.

Ao Luciano Agostini, meu grande amigo e co-orientador desta tese, não tenho palavras para expressar todo o meu agradecimento. Não tenho dúvidas de que sem o apoio do Luciano, este trabalho não teria sido possível. Foi inspirado pelo Luciano que escolhi seguir carreira acadêmica, e foi através dele que conheci o professor Bampi e o professor Susin, que foram meus orientadores durante o mestrado. O Luciano não é excelente apenas como professor e orientador, mas também é um exemplo como pessoa, sempre disposto ajudar e sempre trabalhando por todos ao seu redor. Comigo, tem sido assim desde 2003, quando ele começou as atividades no GACI, e eu ingressei como bolsista IC no grupo. Espero que possamos continuar trabalhando juntos por muito tempo ainda, agora como colegas de profissão, e que eu possa retribuir de alguma forma por tudo que fizeste por mim, muito obrigado.

Gostaria de agradecer aos meus queridos colegas do grupo de pesquisa da UFRGS, o "lab 215", com quem convivi durante este período. Agradecimentos especiais aos colegas Cláudio Diniz, Bruno Zatt, Vagner Rosa, Roger Porto, Fábio Ramos, Dieison Deprá, André Martins, Débora Matos, Franco Valdez e Cristiano Thiele. Muito obrigado a todos pelo apoio, discuções técnicas, revisões de artigos, conversas na hora do cafesinho e churrascos do grupo. Este grupo é muito forte, e foi um prazer ter participado e contribuído com ele durante este período.

Tenho que agradecer também ao grupo de Arquiteturas e Circuitos Integrados da UFPel, o GACI, o qual tenho o prazer de participar desde sua criação, e que tenho participado mais ativamente desde que retornei a Pelotas para assumir como professor na UFPel. Gostaria de agradecer ao Diego Noble, Daniel Palomino e Felipe Sapaio, que estavam no GACI quando retornei a Pelotas, e que hoje estão no mestrado da UFRGS. Tenho que agradecer em especial aos bolsistas de IC Pargles D'all Oglio, Cássio Cristani e Gustavo Sanchez, que trabalharam comigo durante os dois últimos anos deste doutorado. O apoio deles foi fundamental para a realização deste trabalho, e mais uma vez tenho que agradecer ao Luciano, que disponibilizou os seus orientados e toda a infraestrutura do GACI para que eu pudesse trabalhar.

Enfim, gostaria de agradecer a todos os amigos, colegas e funcionários da UFRGS e UFPel que de uma forma ou de outra contribuíram para o desenvolviemnto deste trabalho, quatro anos se passaram desde o início deste doutorado e com certeza não consegui agradecer nominalmente a todos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABEM	*All Binary Motion Estimation*
ACC	Acumulador
ATME	*Adaptive True Motion Estimation*
ASIC	*Application-specific Integrated Circuit*
BC	Bloco Candidato
BCC	Bloco Candidato Calculado
BE	Bloco Escolhido
BMPDS	*Best Multi-Point Diamond Search*
BRAM	*Block* RAM
BBGDS	*Block-Based Gradient Descent Se*arch
Cb	*Chrominance blue*
CDL	Controle Dinâmico de Laços
CIF	*Common Intermediate Format*
CLB	*Configurable Logic Block*
Cr	*Chrominance red*
DCS	*Dual Cross Search*
DGDS	*Directional Gradient Descent Se*arch
DIC	*Dynamic Iteration Control*
D_M	Desvio Médio
DMPDS	*Dynamic Multi-Point Diamond Search*
DS	*Diamond Search*
DVD	*Digital Versatile Disk*
DVSS	*Dynamically Variable Step Search*
EDTV	*Enhanced-Definition Television*
EM	Estimação de Movimento
FDGDS	*Fast Directional Gradient Descent Searc*h
FPGA	*Field Programmable Gate Array*

FS	*Full Search*
HD	*High definition*
HDTV	*High Definition Digital Television*
HO-MIRS	*Hardware Oriented - Multiple Iterative Random Search*
MRSDS	*Multiple Reference Sub-sampled Diamond Search*
HS	*Hexagon Search*
HSI	*Hue, Saturation, Intensity*
IEC	*International Electrotechnical Commission*
IRS	*Iterative Random Search*
ISO	*International Organization for Standardization*
ITU-T	International Telecommunication Union - Telecommunication
LDSP	*Large Diamond Search Pattern*
LFSR	*Linear Feedback Shift Register*
LUT	*Look up Table*
MAE	*Minimum Absolute Error*
MBA	Memória do Bloco Atual
MBC	Memória de Bloco Candidato
MBCS	Memória de Bloco Candidato do SDSP
MC	*Motion Compensation*
ME	*Motion Estimation*
MIRS	*Multiple Iterative Random Search*
ML	Memória Local
MMEA	*Multi-resolution ME Algorithm*)
MPDS	*Multi-Point Diamond Search*
MPEG	*Moving Picture Experts Group*
MSE	*Minimum Square Error*
NSS	N *Step Search*
VM	Vetor de Movimento
OTS	*One at a Time Search*
PMVFAST	Predictive Motion Vector Field Adaptive Search Technique
PRR	Percentual de Redução do Resíduo
PSNR	*Peak-to-Signal Noise Ratio*
QCIF	*Quarter Common Intermediate Format*
QSDS	*Quarter Sub-sampled Diamond Search*
QFHD	*Quad Full High Definition*

RAM	*Random Access Memory*
RAR	*Roshal ARchive*
RD	Registrador de deslocamento
RDS	*Random Diamond Search*
RGB	*Red, Green, Blue*
RSP	*Random Search Plus*
SAD	*Sum of Absolute Differences*
SDS	*Sub-sampled Diamond Search*
SDSP	*Small Diamond Search Pattern*
SDTV	*Standart Definition Television*
T	Transformada
TSMC	*Taiwan Semiconductor Manufacturing Company*
TSS	*Three Step Search*
UFRGS	Universidade Federal do Rio Grande do Sul
UMD	*Universal Media Disc*
UMH	*Uneven Multi Hexagon*
UA	Unidade Aleatória
UP	Unidade de Processamento
VGA	*Video Graphics Adapter*
VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VMPDS	*Variable Multi-Point Diamond Search*
VQEG	*Video Quality Experts Group*
WQXGA	*Wide Quad eXtended Graphics Array*
Y	*Luminance*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A compressão de vídeo é um tema extremamente relevante no cenário atual, principalmente devido ao crescimento significativo da utilização de vídeos digitais. Sem a compressão, é praticamente impossível enviar ou armazenar vídeos digitais devido à sua grande quantidade de informações, inviabilizando aplicações como televisão digital de alta definição, vídeo conferência, vídeo chamada para celulares etc. O problema vem se tornando maior com o crescimento de aplicações de vídeos de alta definição, onde a quantidade de informação é consideravelmente maior. Diversos padrões de compressão de vídeo foram desenvolvidos nos últimos anos, todos eles podem gerar grandes taxas de compressão. Os padrões de compressão de vídeo atuais obtêm a maior parte dos seus ganhos de compressão explorando a redundância temporal, através da estimação de movimento. No entanto, os algoritmos de estimação de movimento utilizados atualmente não consideram as variações nas características dos vídeos de alta definição.

Neste trabalho uma avaliação da estimação de movimento em vídeos de alta definição é apresentada, demonstrando que algoritmos rápidos conhecidos, e largamente utilizados pela comunidade científica, não apresentam os mesmos resultados de qualidade com o aumento da resolução dos vídeos. Isto demonstra a importância do desenvolvimento de novos algoritmos focados em vídeos de altíssima definição, superiores à HD 1080p.

Esta tese apresenta o desenvolvimento de novos algoritmos rápidos de estimação de movimento, focados na codificação de vídeos de alta definição. Os algoritmos desenvolvidos nesta tese apresentam características que os tornam menos suscetíveis à escolha de mínimos locais, resultando em ganhos significativos de qualidade em relação aos algoritmos rápidos convencionais, quando aplicados a vídeos de alta definição. Além disso, este trabalho também visa o desenvolvimento de arquiteturas de hardware dedicadas para estes novos algoritmos, igualmente dedicadas a vídeos de alta definição. O desenvolvimento arquitetural é extremamente relevante, principalmente para aplicações de tempo real a 30 quadros por segundo, e também para a utilização em dispositivos móveis, onde requisitos de desempenho e potência são críticos.

Todos os algoritmos desenvolvidos foram avaliados para um conjunto de 10 sequências de teste HD 1080p, e seus resultados de qualidade e custo computacional foram avaliados e comparados com algoritmos conhecidos da literatura. As arquiteturas de hardware dedicadas, desenvolvidas para os novos algoritmos, foram descritas em VHDL e sintetizadas para FPGAs e ASIC, em *standard cells* nas tecnologias 0,18µm e 90nm. Os algoritmos desenvolvidos apresentam ganhos de qualidade para vídeos de alta definição em relação a algoritmos rápidos convencionais, e as arquiteturas desenvolvidas possuem altas taxas de processamento com baixo consumo de recursos de hardware e de potência.

Palavras-Chave: Estimação de movimento, Desenvolvimento algorítmico, Desenvolvimento de arquiteturas de hardware, Vídeos de alta definição.

Algorithmic and Architectural Development for Motion Estimation on High Definition Video Compression

ABSTRACT

Video compression is an extremely relevant theme in today's scenario, mainly due to the significant growth in digital video applications. Without compression, it is almost impossible to send or store digital videos, due to the large amount of data that they require, making applications such as high definition digital television, video conferences, mobiles video calls, and others unviable. This demand is increasing since there is a strong growth in high definition video applications, where the amount of information is considerably larger. Many video coding standards were developed in the last few years, all of them can achieve excellent compression rates. A significant part of the compression gains in the current video coding standards are obtained through the exploration of the temporal redundancies by means of the motion estimation process. However, the current motion estimation algorithms do not consider the inherent variations that appear in high and ultra-high definition videos.

In this work an evaluation of the motion estimation in high definition videos is presented. This evaluation shows that some well know fast algorithms, that are widely used by the scientific community, do not keep the same quality results when applied to high resolution videos. It demonstrates the relevance of new fast algorithms that are focused on high definition videos.

This thesis presents the development of new fast motion estimation algorithms focused in high definition video encoding. The algorithms developed in this thesis show some characteristics that make them more resilient to avoid local minima, when applied to high definition videos. Moreover, this work also aims at the development of dedicated hardware architectures for these new algorithms, focused on high definition videos. The architectural development is extremely relevant, mainly for real time applications at 30 frames per second, and also for mobile applications, where performance and power are critical issues.

All developed algorithms were assessed using 10 HD 1080p test video sequences, and the results for quality and computational cost were evaluated and compared against known algorithms from the literature. The dedicated hardware architectures, developed for the new algorithms, were described in VHDL and synthesized for FPGA and ASIC. The ASIC implementation used 0.18μm and 90nm CMOS *standard cells* technology. The developed algorithms present quality gains in comparison to regular fast algorithms for high definition videos, and the developed architectures presents high processing rate with low hardware resources cost and power consumption.

Keywords: Motion estimation, Algorithmic development, Architectural hardware development, High definition videos.

1 INTRODUÇÃO

A compressão de vídeos digitais é um tema de extrema importância na atualidade, principalmente devido à grande quantidade de informações contidas em vídeos digitais sem compressão. Tarefas como armazenamento ou transmissão, são praticamente impossíveis se forem considerados vídeos digitais sem compressão. Isto fez com que diversos padrões de compressão de vídeo fossem desenvolvidos, para reduzir significativamente a quantidade de dados presentes nos vídeos digitais. Esta compressão significativa na quantidade de dados possibilitou a utilização de vídeo digital em diversos dispositivos como celulares, DVD *players*, televisão digital de alta definição (HDTV), entre outros. No entanto, este problema vem se tornando maior com o constante aumento das resoluções dos vídeos digitais. Atualmente, até mesmo dispositivos móveis, como celulares e pequenas câmeras digitais, podem trabalhar com vídeos de alta definição.

Os padrões de compressão de vídeo são formados por um conjunto de algoritmos e técnicas que tentam reduzir o número de dados necessários para representar um vídeo. Dentro dos compressores de vídeo digital atuais, como o MPEG-4 (ISO/IEC, 1999) e H.264 (ITU-T, 2005), por exemplo, existem diversas etapas nas quais as informações que compõem o vídeo são submetidas, dentre elas as principais são: estimação de movimento (EM), compensação de movimento, transformadas, quantização e codificação de entropia (BHASKARAN, 1999). Dentre as etapas dos compressores atuais, a mais complexa, e que resulta no maior impacto sobre a taxa de compressão, é a estimação de movimento. A EM é responsável por encontrar uma correlação entre quadros, mapeando a redundância temporal entre os quadros vizinhos de uma cena. O processo de estimação é extremamente complexo, o que torna indispensáveis o uso de arquiteturas de hardware dedicadas para a codificação de vídeos de alta definição em tempo real.

A grande complexidade do processo de estimação fez com que diversos algoritmos rápidos fossem desenvolvidos. Os algoritmos rápidos possuem heurísticas que aceleram o processo de estimação. Em geral, estas heurísticas repetem um determinado padrão de busca até que uma dada condição de parada seja atingida. No entanto, a característica gulosa destas heurísticas faz com que elas sejam extremamente suscetíveis à escolha de mínimos locais. Toda vez que uma iteração obtém um resultado inferior ao obtido na última iteração, o algoritmo encerra sua busca, desconsiderando a possibilidade de encontrar um resultado melhor após algumas iterações. O número de mínimos locais em uma área de busca cresce significativamente com o aumento da resolução dos vídeos, logo, esta característica deve ser considerada para a estimação de movimento em vídeos de alta definição.

Existem muitos artigos científicos publicados na literatura que apresentam algoritmos rápidos de EM, bem como soluções arquiteturais para estes algoritmos. No entanto, a grande maioria dos artigos apresentam seus dados de avaliação para vídeos de baixa resolução. Em geral, os algoritmos rápidos de EM podem gerar resultados de

qualidade similares aos obtidos pelo algoritmo ótimo, quando aplicados a vídeos de baixa resolução. Estes algoritmos, quando aplicados a vídeos de alta definição, tendem a apresentar perdas significativas de qualidade em relação ao algoritmo ótimo, o que diminui a relevância destes algoritmos neste contexto.

Esta tese visa o desenvolvimento de algoritmos rápidos e arquiteturas dedicadas para a estimação de movimento com foco em vídeos de alta definição. Um dos focos deste trabalho é o desenvolvimento de novos algoritmos para a EM, voltados para a codificação de vídeos de alta definição. Estes algoritmos devem apresentar características que os tornem mais resistentes à escolha de mínimos locais, aproximando os resultados de qualidade aos obtidos pelo algoritmo ótimo, no entanto, mantendo uma redução significativa no custo computacional. Este trabalho também foca no desenvolvimento de arquiteturas de hardware dedicadas para os novos algoritmos. Estas arquiteturas tem foco em desempenho para a codificação de vídeos de alta definição (mínimo HD 1080p) em tempo real, a 30 quadros por segundo. Esta é uma área importante, principalmente para aplicações em tempo real para dispositivos móveis, onde os requisitos de taxa de processamento e consumo de energia são críticos.

Os algoritmos rápidos de EM que foram desenvolvidos podem ser classificados em duas classes: Multiponto e Aleatória. Ambas apresentam características que visam à redução da incidência da escolha de mínimos locais na estimação de movimento em vídeos de alta definição. A classe de algoritmos multiponto utiliza a estratégia de dividir a busca em diferentes pontos da área de busca. Nesta classe de algoritmos, foram desenvolvidos os algoritmos *Multi-Point Diamond Search* (MPDS) e *Dinamic Multi-Point Diamond Search* DMPDS. Os algoritmos MPDS e DMPDS utilizam cinco núcleos de busca, um aplicado a região central da área de busca e outros quatro divididos entre os quatro setores. A distância dos núcleos de busca dos setores, em relação ao centro, é dada por um parâmetro d, que possui um valo fixo para o algoritmo MPDS e dinâmico para o algoritmo DMPDS. Os algoritmos da segunda classe (Aleatória) utilizam a aleatoriedade no processo de busca como estratégia para diminuir a incidência da escolha de mínimos locais. Todos os algoritmos aleatórios desenvolvidos possuem uma etapa em comum, o sorteio e a avaliação de N blocos candidatos aleatórios dentro da área de pesquisa. Além da etapa de exploração aleatória, cada algoritmo apresenta uma forma diferente de exploração da região central da área de busca e, também, diferentes abordagens de refinamento final que é aplicado sobre os resultados obtidos com a etapa aleatória.

Dez sequências de teste HD 1080p foram utilizadas para a avaliação dos algoritmos desenvolvidos. Os resultados de qualidade e custo computacional foram avaliados e comparados a outros algoritmos da literatura. Os resultados destas avaliações mostraram que os algoritmos desenvolvidos apresentam ganhos de qualidade significativos em relação a outros algoritmos rápidos conhecidos. Desta forma, foi possível reduzir as perdas em relação ao resultado ótimo e, ainda assim, manter ganhos significativos em redução do custo computacional. Os algoritmos desenvolvidos também foram avaliados para sequências de teste WQXGA (2560x1600 pixels) e os resultados demonstram que a eficiência dos algoritmos desenvolvidos se mantém e que os ganhos obtidos sobre algoritmos rápidos convencionais são ainda maiores dos que os obtidos em HD 1080p.

As arquiteturas de hardware dedicadas para os novos algoritmos de EM também são apresentadas. As arquiteturas foram desenvolvidas com foco em alto desempenho, sendo o requisito mínimo o processamento de 30 quadros HD 1080p por segundo. As arquiteturas foram descritas em VHDL e sintetizadas para FPGAs e *standard cells*. As

arquiteturas desenvolvidas apresentam baixa frequência de operação e baixa utilização de recursos de hardware em relação a trabalhos relacionados. Estas características também garantiram os baixos resultados de consumo de energia para a síntese ASIC. Estes resultados foram obtidos devido à eficiência dos algoritmos e das arquiteturas desenvolvidas.

O capítulo 2 deste trabalho apresenta alguns conceitos básicos da codificação de vídeo digital. Estes conceitos serão importantes para a compreensão do restante do texto. No capítulo 3 serão apresentados alguns conceitos sobre a estimação de movimento, discutindo o papel da EM no contexto da codificação de vídeo. Alguns algoritmos clássicos de EM serão apresentados, bem como uma revisão bibliográfica sobre o estado da arte em termos de algoritmos e arquiteturas dedicadas à EM. O capítulo 4 apresenta um estudo mais aprofundado sobre a EM em vídeos de alta definição. Neste estudo, o algoritmo DS (*Diamond Search*) é comparado com o algoritmo de busca completa (FS – *Full Search*) com a aplicação em vídeos de diferentes resoluções. Um estudo sobre os mínimos locais também é apresentado, bem como a avaliação do uso de diferentes tamanhos de bloco e níveis de subamostragem de pixel na EM em vídeos de alta definição. O capítulo 5 apresenta os resultados do desenvolvimento algorítmico. Os algoritmos desenvolvidos são descritos e os resultados das avaliações e comparações com outros algoritmos são apresentados. No capítulo 6 são apresentados os resultados do desenvolvimento arquitetural para a EM. As arquiteturas desenvolvidas para os novos algoritmos são descritas, bem como os resultados de síntese para FPGAs e *standard cells* na tecnologia TSMC 0,18um e 90nm. Este capítulo também apresenta as comparações destes resultados com trabalhos relacionados. Por fim, o capítulo sete apresenta as conclusões e trabalhos futuros apontados por esta tese.

2 CONCEITOS DE COMPRESSÃO DE VÍDEO DIGITAL

Neste capítulo serão apresentados alguns conceitos básicos de vídeos e compressão de vídeos digitais, que serão importantes para a compreensão dos demais capítulos apresentados nesta tese.

2.1 Conceitos Básicos de Vídeo Digital

Um vídeo digital é formado por uma sequência de imagens independentes, captadas com um intervalo de tempo determinado. Em geral, para que se obtenha sensação de movimento contínuo (tempo real) é necessária à captação de 24 a 30 imagens (quadros) por segundo (GONZALEZ, 2003). Os quadros são formados por pontos (pixels). Pontos pertencentes ao mesmo quadro e espacialmente vizinhos são chamados de pontos vizinhos. Imagens, também chamadas de quadros (em inglês *frames*), pertencentes à mesma cena e temporalmente próximas, são chamadas de imagens vizinhas. Pontos e imagens vizinhas são geralmente muito similares e esta é uma característica importante de imagens e vídeos digitais. Esta similaridade resulta em grande redundância de informações na sua representação. A redundância em vídeos digitais surge de diferentes formas e todos os codificadores de vídeo atuais consideram as diferentes formas de redundância para aumentar a sua eficiência de compressão.

Todo vídeo digital pode ser dividido em uma sequência de quadros. Os codificadores de vídeo ainda dividem, em geral, os quadros em blocos. Estes blocos podem ter tamanhos variados, mas comumente são utilizados tamanhos de 4x4, 8x8 ou 16x16 pixels. A Figura 2.1 ilustra uma sequência de quadros temporalmente vizinhos de um vídeo digital, destacando a divisão de um quadro em blocos de pixels.

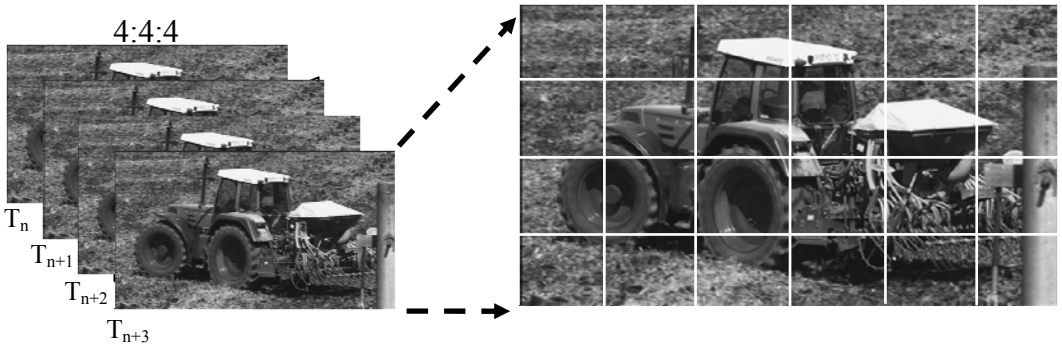


Figura 2.1: Sequência de quadros e blocos em um vídeo digital

Uma cena visual natural é contínua no tempo e no espaço, sendo que a representação digital de uma cena visual envolve amostragem espacial e temporal. A amostragem espacial é realizada, normalmente, como uma matriz retangular na imagem do vídeo. Quanto maior esta matriz (resolução), melhor será a definição (qualidade) da imagem, visto que mais pixels serão usados para a representação. A qualidade da imagem é influenciada pela quantidade de pixels. Quanto mais espaçados são os pixels,

menor a resolução da imagem amostrada, assim como quanto mais próximos são os pixels, maior é a resolução da imagem amostrada e, consequentemente, melhor será o resultado visual da imagem.

A amostragem temporal é realizada como uma série de quadros, estáticos, amostrados a certos intervalos de tempo, conforme ilustrado na Figura 2.1. Quanto maior for a taxa de amostragem, melhor será a percepção de movimento no vídeo. Um vídeo captado e exibido a 30 quadros por segundo transmite a sensação de movimento contínuo (tempo real), no entanto, ao assistir este vídeo em câmera lenta (*slow motion*), através da redução da velocidade de exibição (maior tempo entre quadros), alguns movimentos mais rápidos podem ficar prejudicados. Alguns efeitos indesejados como borrões ou *motion blur* podem se tornar perceptíveis. Estes efeitos podem ser evitados com o aumento da taxa de amostragem temporal, por exemplo, para 120 quadros por segundo. Algumas câmeras podem captar centenas de quadros por segundo (FASTEC, 2011) tornando, assim, possível a percepção de movimentos que nunca haviam sido captados por câmeras de vídeos convencionais (cinema, televisão, etc..) que amostram imagens em taxas na ordem de dezenas de quadros por segundo.

Vídeo digital é a representação de uma cena de vídeo amostrada de forma digital. Cada amostra temporal-espacial, chamada elemento de figura (*picture element* ou pixel) é representada como um número, ou um conjunto de números, que representa as componentes de luminância ou cor da amostra, de acordo com o espaço de cor utilizado. Para se obter uma imagem amostrada bidimensionalmente, a câmera foca uma projeção bidimensional da cena de vídeo em um sensor.

2.2 Compressão com Perdas e sem Perdas

Os codificadores de vídeo podem ser classificados entre os que fazem a compressão sem perdas (*lossless*) e os que geram perdas durante o processo de compressão (*lossy*) (RICHARDSON, 2002).

Na compressão sem perdas, os codificadores utilizam apenas técnicas de compressão que não eliminem informações. Isto garante que o arquivo resultante do processo de descompressão é idêntico ao arquivo original. Este tipo de compressão pode ser muito eficiente para diferentes tipos de dados, principalmente para os que apresentam grande quantidade de redundância estatística. Um exemplo de compressão sem perdas muito utilizado é o compressor de arquivos ZIP (SALOMON, 2008). Estes algoritmos reestruturam o arquivo, atribuindo códigos menores para símbolos com maior ocorrência e códigos maiores para símbolos de menor ocorrência. Isto reduz o tamanho do arquivo sem perda de informação. No entanto, imagens e vídeos digitais possuem uma distribuição de probabilidade muito próxima entre os símbolos (neste contexto os valores dos pixels). Compressores de imagens sem perdas eficientes como o JPEG-LS (RICHARDSON, 2003), por exemplo, atingem apenas taxas de compressão da ordem de 3 a 4 vezes.

A compressão com perdas é a mais utilizada para a codificação de imagens e vídeos digitais. Além da relativa ineficiência dos compressores sem perdas neste contexto, o volume de dados a ser processado, armazenado ou transmitido, principalmente em vídeos, é muito elevado e requer taxas de compressão elevadas. Os compressores de vídeo atuais estão baseados no princípio de remover informações de baixa relevância visual (RICHARDSON, 2003). Desta forma, é possível alcançar elevadas taxas de compressão com uma pequena redução na qualidade visual, sendo que muitas vezes esta

perda em qualidade é imperceptível ao sistema visual humano. Compressores de vídeo atuais, como o MPEG-2 e o H.264/AVC podem atingir taxas de compressão entre 24:1 e 50:1, com a qualidade visual muito próxima a do vídeo original (GHANBARI, 2003).

2.3 Espaço de Cores

A representação digital de um vídeo colorido está associada à interpretação das cores pelo sistema visual humano. Existem muitas formas de representar as cores de forma digital. Um sistema para representar cores é chamado de espaço de cores e a definição do espaço de cor a ser utilizado para representar um vídeo é essencial para a eficiência da codificação deste vídeo. São vários os espaços de cores usados para representar imagens digitais, tais como: RGB, HSI e YCbCr (SHI, 1999). O espaço de cores RGB é um dos mais comuns, tendo em vista que é este o espaço de cores utilizado nos monitores coloridos. O RGB representa, em três matrizes distintas, as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul. A grande maioria das cores perceptíveis pelo olho humano podem ser derivadas a partir de combinações destas três cores primárias. Cada um dos canais R, G e B representa a intensidade de sua respectiva cor e a informação de intensidade luminosa, ou luminância, está distribuída entre os três canais.

No espaço de cores YCbCr, as três componentes utilizadas são: luminância (Y), que define a intensidade luminosa ou o luminância; crominância azul (Cb); e crominância vermelha (Cr) (MIANO, 1999), mas todas as cores representadas pelo espaço RGB são possíveis de serem representadas também no espaço YCbCr. Os componentes R, G e B do espaço de cores RGB possuem um elevado grau de correlação, isto torna difícil o processamento de cada uma das informações de cor de forma independente. Este é o principal fator que faz com que os padrões de compressão de vídeo utilizem espaços de cores do tipo luminância e crominância, como o YCbCr (RICHARDSON, 2002). Outra vantagem do espaço de cor YCbCr sobre o espaço RGB é que, no espaço YCbCr, a informação de cor está completamente separada da informação de luminância. Deste modo, estas informações podem ser tratadas de forma diferenciada pelos compressores de imagens estáticas e vídeos. O canal de luminância (Y) possui apenas as informações de luminância, os canais de crominância azul e vermelha (Cb e Cr) possuem as informações de cor azul e vermelha, respectivamente.

2.4 Subamostragem de Cores

O sistema visual humano possui menor resolução para informações de cor contidas nas imagens (GONZALEZ, 2003). Logo, os padrões de compressão de imagens estáticas e vídeos podem explorar esta característica humana para aumentar a eficiência de codificação através da redução da taxa de amostragem dos componentes de crominância, em relação aos componentes de luminância (RICHARDSON, 2002). Esta operação é chamada de subamostragem de cores e é realizada sob o espaço de cores YCbCr nos padrões de compressão de vídeos atuais.

Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a subamostragem. Os formatos mais comuns são o 4:4:4, o 4:2:2 e o 4:2:0. No formato 4:4:4, para cada quatro amostras de luminância (Y), existem quatro amostras de crominância azul (Cb) e quatro amostras de crominância vermelha (Cr). Desta forma, não existe subamostragem de cor e não existirá nenhuma perda de qualidade no vídeo. No formato 4:2:2, para cada quatro amostras de Y, apenas duas amostras de crominância Cb e duas amostras de

crominância Cr são representadas. Com isso, 50% das informações de cor serão descartadas, reduzindo em 25% o tamanho total do vídeo, antes mesmo de qualquer etapa de compressão. O formato 4:2:0, considera que para cada quatro amostras de luminância Y, apenas uma amostra de crominância Cb e uma de Cr são representadas. A Figura 2.2 apresenta estes três formatos de subamostragem de cores.

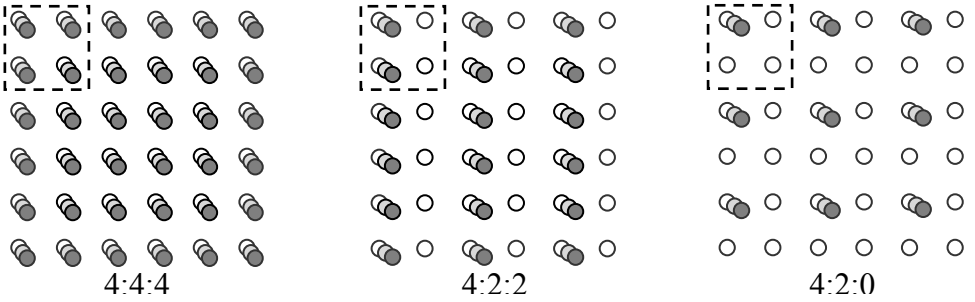


Figura 2.2: Formatos de subamostragem de crominância (RICHARDSON , 2002)

A subamostragem de cor aumenta significativamente a eficiência da compressão, uma vez que parte da informação da imagem é simplesmente descartada, sem causar impacto visual perceptível. Considerando o formato 4:2:0 como exemplo, apenas 25% das informações de crominância serão consideradas, um quarto das amostras presentes no componente de luminância. Assim, um vídeo YCbCr no formato 4:2:0 irá utilizar exatamente a metade das amostras necessárias para um vídeo RGB ou YCbCr no formato 4:4:4. Isso implica em uma redução de 50% no volume de informações que representam o vídeo, considerando apenas a subamostragem de cores, antes da aplicação de qualquer técnica de compressão de vídeo.

2.5 Redundância de Informação em Vídeos Digitais

A compressão de vídeo é baseada na eliminação de dados redundantes. Um dado é considerado redundante quando seu valor não representa uma nova informação relevante para a representação da imagem. Basicamente, existem três tipos diferentes de redundâncias exploradas na compressão de vídeos: redundância espacial, redundância temporal e redundância entrópica.

A redundância espacial, também chamada de redundância intraframe (GHANBARI, 2003), é resultado da correlação existente entre os pixels vizinhos do mesmo quadro. Pixels vizinhos tendem a possuir valores semelhantes e, por consequência, com elevado grau de redundância de informação, gerando pouco acréscimo de informação visual a cada novo ponto em relação aos seus vizinhos. As características do sistema visual humano fazem com que alguns tipos de informações que podem estar presentes na imagem não sejam captadas. Além disso, algumas informações da imagem, como o luminância, por exemplo, são mais importantes para o sistema visual humano do que outras, como as cores. Para explorar este tipo de redundância, parte da informação original da imagem é eliminada de forma irreversível pelo codificador. Existem três tipos principais de processos utilizados para este fim. O primeiro, chamado de subamostragem, que já foi explicado. . O segundo, conhecido por quantização, normalmente é aplicado no domínio das frequências e pode eliminar ou atenuar as frequências de menor relevância para o sistema visual humano. Por isso, este tipo de processo de codificação é chamado de codificação com perdas. É importante destacar que a eliminação destas informações contribui para que o codificador atinja elevadas taxas de compressão, com pequeno impacto na qualidade visual da imagem

que, eventualmente, pode mesmo ser nulo. O terceiro método de compressão amplamente utilizado nos codificadores de vídeo atuais, para reduzir a redundância espacial, é a codificação intra-quadro.

A redundância temporal, também chamada de redundância inter-quadros ou interframe (GHANBARI, 2003), é causada pela correlação existente entre quadros vizinhos. Na verdade, a redundância temporal poderia ser classificada como apenas mais uma dimensão da redundância espacial, como faz (GONZALEZ, 2003). Muitos blocos de *pixels* simplesmente não mudam de valor de um quadro para outro em um vídeo, como por exemplo, em um fundo que não foi alterado de um quadro para outro. Outros pixels apresentam uma pequena variação de valores causada, por exemplo, por uma variação de iluminação. Também é possível que o bloco de pixels simplesmente tenha se deslocado de um quadro para o outro, como por exemplo, em um movimento de um objeto em uma cena. Estes fatores resultam em um pequeno acréscimo de informação visual de cada imagem com relação às imagens vizinhas. Todos os padrões de codificação de vídeo atuais visam eliminar ou diminuir a redundância temporal. A exploração eficiente da redundância temporal conduz a elevadas taxas de compressão e é fundamental para o sucesso dos codificadores.

A redundância entrópica está relacionada com a forma de representação computacional dos símbolos codificados e não se relaciona diretamente ao conteúdo da imagem. A entropia é uma medida da quantidade média de informação transmitida por símbolo do vídeo (SHI, 1999). A quantidade de informação nova transmitida por um símbolo diminui na medida em que a probabilidade de ocorrência deste símbolo aumenta. Então, os codificadores que exploram a redundância entrópica têm por objetivo transmitir o máximo de informação possível por símbolo codificado e, deste modo, representar mais informações com um número menor de símbolos. A codificação de entropia, como é chamada, utiliza diferentes técnicas e algoritmos de compressão sem perdas para atingir este objetivo.

2.6 Avaliação de Qualidade em Vídeos Digitais

A avaliação de qualidade em vídeos digitais é um problema muito complexo. Existem diversas métricas objetivas para a avaliação de qualidade e a maioria delas utiliza equações matemáticas aplicadas aos valores dos pixels, antes e depois da compressão, para definir a qualidade do processo. No entanto, nenhuma das métricas existentes considera de maneira eficiente as questões subjetivas que um observador humano consegue perceber.

O critério de avaliação de qualidade objetivo mais utilizado é o PSNR (*Peak Signal-to-Noise Ratio*) (GHANBARI, 2003), apresentado na equação (1). O valor de *MAX* corresponde ao valor máximo que pode ser assumido por uma amostra e normalmente este valor é igual a 255, que é o valor máximo para amostras de oito bits. *MSE* é a sigla para *Mean Squared Error*, que calcula o erro médio quadrático dos pixels de um quadro, conforme a equação (2). As variáveis *R* e *O* são os quadros original e reconstruído (após a compressão e descompressão), respectivamente. As variáveis *m* e *n* representam as dimensões do quadro. O PSNR pode ser calculado para um quadro ou apenas para um bloco deste quadro. Seus resultados podem ser acumulados para a geração do PSNR de um quadro completo ou de um vídeo. Os valores de PSNR são medidos em decibéis (dB), sendo que para quadros idênticos, que resultam em um valor de MSE = 0, o valor do PSNR será indefinido. Em geral, é possível afirmar que altos valores de PSNR indicam uma qualidade relativamente alta da imagem e baixos valores

de PSNR indicam uma qualidade relativamente baixa na imagem (RICHARDSON, 2002).

$$PSNR_{dB} = 20 \cdot \log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right) \quad (1)$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(R_{i,j} - O_{i,j}\right)^2 \quad (2)$$

Mesmo o PSNR sendo a métrica de qualidade objetiva mais aceita pela comunidade científica, esta métrica não consegue identificar critérios subjetivos presentes em imagens e vídeos digitais. Um exemplo disto pode ser observado nas Figuras 2.3, 2.4 e 2.5. A Figura 2.3 apresenta a imagem original e as Figuras 2.3 e 2,4, apresentam uma versão modificada da imagem original. Na Figura 2.4, o fundo da imagem está embaçado e esta imagem possui um PSNR de 26,8dB. A Figura 2.4 apresenta uma versão da imagem original com a parte frontal do caminhão embaçada e esta imagem possui um PSNR de 28,4dB. Claramente, a Figura 2.5 possui uma qualidade visual inferior, no entanto, o PSNR dela é superior ao da Figura 2.4. Isto ocorre porque o PSNR considera o cálculo do MSE entre os pixels para definir o seu valor de qualidade. Na Figura 2.5, a região da parte frontal do caminhão possui uma grande diferença em relação à imagem original (Figura 2.3), no entanto, os demais pixels do quadro possuem uma diferença nula. Já a Figura 2.4, possui uma grande variação nos pixels do fundo da imagem e isto faz com que, na média, o MSE da Figura 2.5 seja menor do que o obtido pela Figura 2.4, mesmo que visualmente o resultado seja inferior. Observadores humanos darão uma importância maior à região frontal do caminhão, em detrimento à região do fundo da imagem. Este tipo de questão subjetiva pode não ser captada pelas métricas objetivas de qualidade como o PSNR.

Figura 2.3: Imagem Original

Figura 2.4: Imagem modificada com fundo embaçado, PSNR 26,8dB

Figura 2.5: Imagem modificada com a parte da frente do caminhão embaçada, PSNR 28,4dB

3 A ESTIMAÇÃO DE MOVIMENTO

As taxas de amostragem dos vídeos digitais variam entre 24 a 30 quadros por segundo, para garantir o efeito de movimento contínuo. Esta taxa tem crescido nos últimos anos, visando a melhoria da captação dos movimentos e a possibilidade de análise dos movimentos em câmera lenta (*slow motion*). Algumas câmeras atuais podem captar centenas de quadros por segundo (FASTEC, 2011). Neste contexto, ao analisar as imagens de um trecho de um vídeo digital, é possível perceber que as imagens vizinhas são muito similares. As diferenças são geradas, em sua maioria, por movimentos da câmera ou de objetos pertencentes à cena, exceto quando existem cortes de cena com troca de contexto visual. Esta característica dos vídeos digitais produz uma grande redundância temporal e ao comparar dois quadros vizinhos, é possível obter uma enorme quantidade de informações repetidas (redundantes), causadas por regiões da cena que não são alteradas em quadros subsequentes.

Para explicar melhor a importância da estimação de movimento na compressão de vídeos digitais, as Figuras 3.1 e 3.2 serão analisadas. Estas figuras representam, respectivamente, dois quadros temporalmente vizinhos de uma sequência de vídeo. É possível observar que as diferenças entre os quadros das Figuras 3.1 e 3.2 são praticamente imperceptíveis visualmente. Grande parte da imagem permanece estática e não sofre alterações de um quadro para outro. A Figura 3.3 foi gerada a partir da diferença (pixel a pixel) entre os quadros das Figuras 3.1 e 3.2 (codificação diferencial), ilustrando as diferenças existentes entre os dois quadros. Nas regiões onde a imagem não se altera, o valor dos pixels também não se altera, o que resulta em um resíduo (resultado da diferença entre duas imagens) de valor zero. Para as regiões que sofrem algum tipo de alteração, a diferença entre os pixels dos dois quadros resulta em um valor não nulo. Estas diferenças aparecem na Figura 3.3 como regiões claras, para diferenças positivas, e regiões escuras, para diferenças negativas. A imagem é ajustada para exibição somando-se o valor médio de um pixel a todos os pixels da imagem, para que os pixels iguais a zero tenham uma tonalidade de cinza médio. É possível observar que as diferenças ocorrem em toda a região de fundo, devido ao movimento da câmera. Além disso, existe movimentação de deslocamento do trator e também de partes móveis, como as rodas do trator. Estas diferenças são bastante significativas, sendo que nos pontos brancos, ou pretos, existem diferenças iguais, ou muito próximas, ao valor máximo.

Os resíduos gerados pela codificação diferencial podem ser drasticamente reduzidos com a estimação de movimento (EM). A Figura 3.4 ilustra a imagem resultante dos resíduos gerados entre os quadros das Figuras 3.1 e 3.2 após o processo de estimação de movimento. É possível perceber uma diminuição significativa na quantidade de ruído da imagem, na comparação com a imagem da Figura 3.3. Isto se deve ao processo de identificação de blocos similares realizado pela estimação de

movimento. Ao remontar o quadro, utilizando as informações da estimação de movimento, é possível reduzir as diferenças causadas pelos movimentos de um quadro para outro, reduzindo, assim o resíduo.

Figura 3.1: Quadro 1

Figura 3.2: Quadro 2

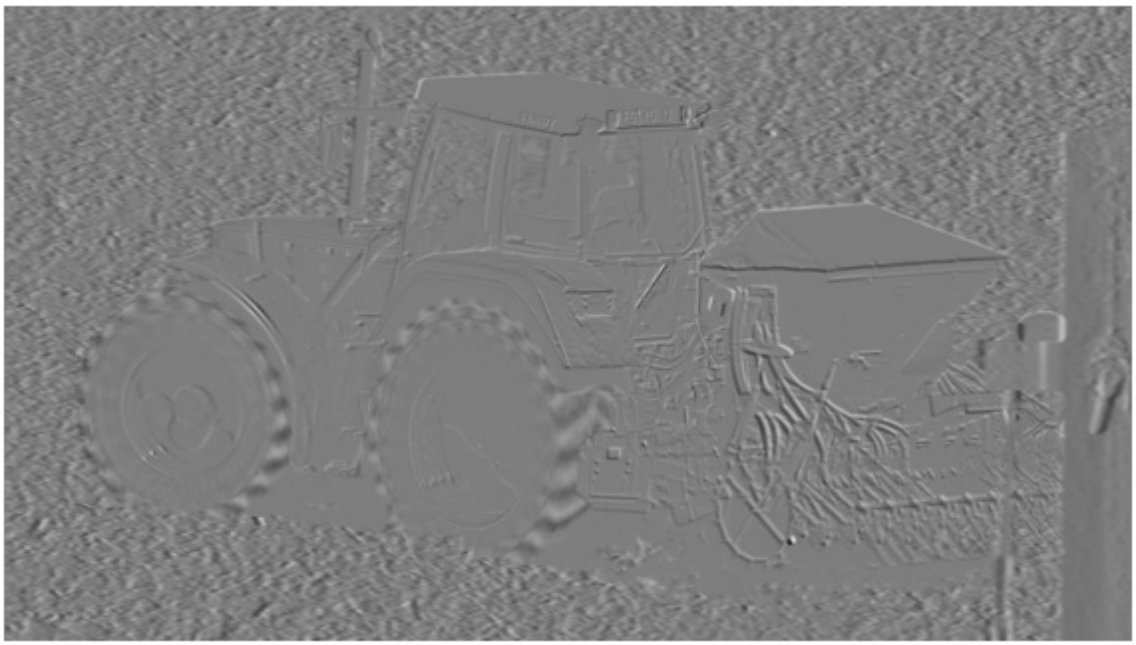


Figura 3.3: Resíduo sem estimação de movimento

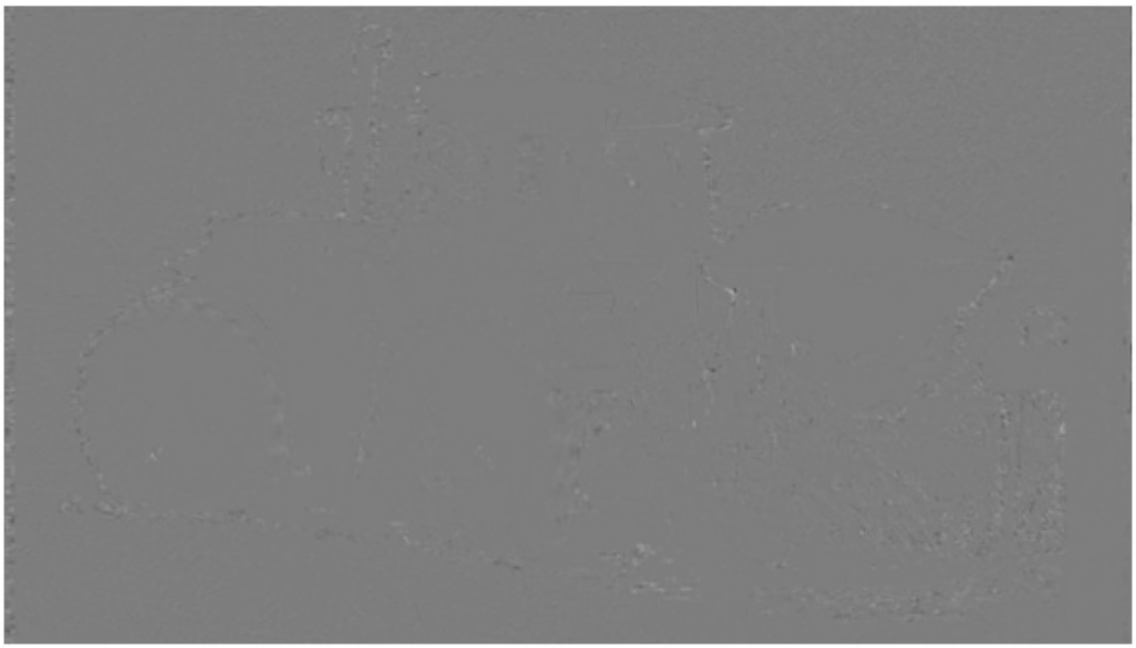


Figura 3.4: Resíduo após a estimação e compensação de movimento

A estimação de movimento visa identificar e reduzir a redundância temporal entre imagens vizinhas de uma cena. Para isso, a EM mapeia as informações redundantes através de vetores de movimento. Os vetores de movimento são vetores de duas dimensões, no formato (x, y), que indicam o deslocamento espacial de um bloco em um quadro, em relação a um quadro de referência. Desta forma, a EM realiza um casamento de padrões de um quadro em relação a outro, que não necessariamente indica a existência de movimento em uma determinada região.

O processo de estimação de movimento para um quadro (quadro atual) começa com a definição do quadro de referência. Mais de um quadro de referência pode ser usado no processo, como será melhor explicado nas próximas sessões do texto. Para cada bloco do quadro atual, será gerado um vetor de movimento, que corresponderá ao deslocamento deste bloco em relação ao quadro de referência. Os blocos do quadro atual serão comparados com os blocos do quadro de referência, para isso, algum critério de similaridade deverá ser utilizado, como o MSE (já explicado no capítulo anterior) ou a soma de diferenças absolutas (SAD – *Sum of Absolute Differences*), por exemplo (KUHN, 1999). O bloco que obtiver a menor diferença, de acordo com o critério similaridade adotado, será o escolhido para representar o bloco do quadro atual.

O SAD consiste na soma das diferenças absolutas entre as amostras do bloco atual e dos blocos do quadro de referência. Este critério é um dos mais utilizados, principalmente para o desenvolvimento em hardware, devido à sua simplicidade. A equação (3) define o cálculo do SAD, onde R são as amostras do bloco de referência e O são as amostras do bloco do quadro de referência.

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left| R_{i,j} - O_{i,j} \right| \tag{3}$$

O SAD pode ser desenvolvido em hardware utilizando apenas somas e subtrações, pois não exige nenhuma operação de divisão ou exponenciação, como o MSE. Mesmo sendo mais simples, o SAD oferece bons resultados para o processo de EM. A Figura 3.5 ilustra os elementos presentes no processo de estimação de movimento. A EM pode utilizar todo o quadro de referência como área de busca, para encontrar o bloco mais similar ao bloco do quadro atual. No entanto, geralmente, uma área de busca (menor que o quadro) é determinada no quadro de referência para cada bloco do quadro atual. Esta estratégia é utilizada para reduzir a complexidade computacional da estimação de movimento. Esta área deve abranger uma região ao redor da posição original do bloco, pois a possibilidade de encontrar blocos similares, próximos à posição original, é elevada (KUO, 2009), (LAI, 2010). Cada bloco da área de busca é chamado de bloco candidato, pois qualquer um deles pode ser escolhido no processo de estimação.

Para definir a escolha do bloco candidato é necessária a utilização de um algoritmo de busca. O algoritmo de busca determina a maneira como a busca será realizada dentro da área de busca. Existem diversos algoritmos de busca publicados na literatura, cada um deles com características distintas. A escolha do algoritmo de busca tem grande influência na qualidade dos vetores de movimento gerados. Neste texto, a qualidade dos vetores de movimento será medida pelo valor do resíduo gerado pelo bloco candidato escolhido. Alguns algoritmos têm compromisso apenas com a qualidade dos vetores, sem considerar a complexidade computacional necessária para a sua implementação. Outros algoritmos têm compromisso com a baixa complexidade, em detrimento da qualidade dos vetores gerados. Em geral, algoritmos com maior complexidade computacional resultam em melhores resultados de qualidade dos vetores de movimento gerados. No entanto, algoritmos de busca atuais visam aliar baixo custo em complexidade com alto desempenho em termos de qualidade dos vetores de movimento. Na seção 3.3 alguns exemplos de algoritmos de busca para a EM serão explicados mais detalhadamente.

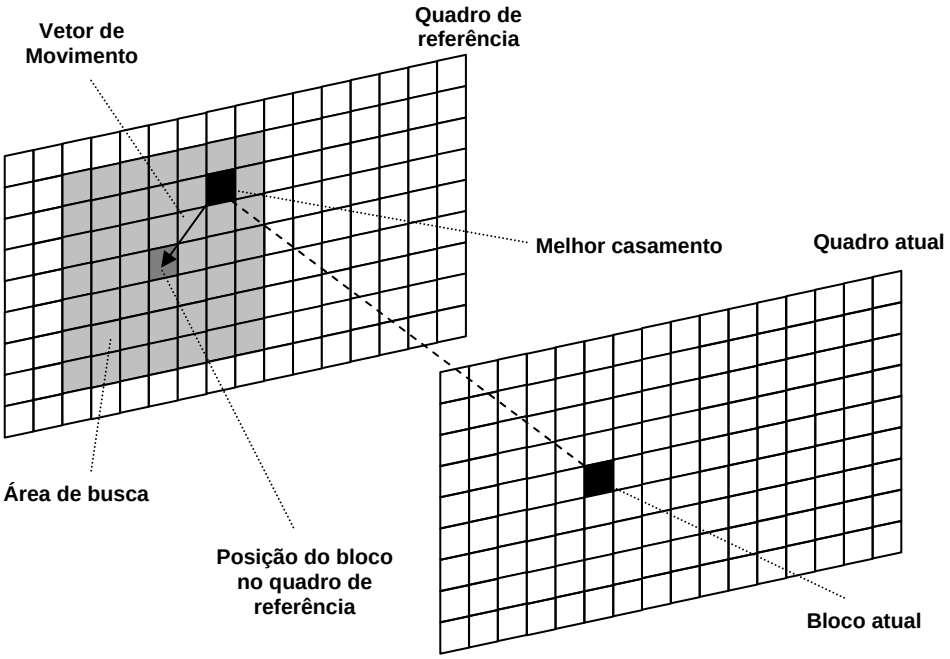


Figura 3.5: Elementos presentes no processo de estimação de movimento (PORTO, 2008d)

A estimação de movimento dos padrões de compressão de vídeos atuais trabalha sobre as informações de luminância do vídeo. As informações de crominância não são consideradas na EM. Os vetores são definidos para a luminância e a crominância reusa estes vetores. Esta estratégia é utilizada, pois as informações de crominância são menos relevantes para a composição do vídeo. Alguns algoritmos preveem também o uso da EM em informações de crominância (ZHIHANG, 1994), (ADIKARI, 2006), mas tais algoritmos não são foco deste trabalho. Na EM, um vetor de movimento é gerado para cada bloco de luminância do quadro atual. O processo de EM implica em uma grande complexidade computacional, no entanto, pode gerar uma redução significativa da quantidade de informação necessária para formar o vídeo, quando um bom casamento (bloco muito semelhante) é encontrado. Todas as informações contidas em um bloco (4x4 ou 8x8 ou 16x6 pixels, etc...) são substituídas por um vetor de duas dimensões, acompanhado de uma informação de resíduo. O resíduo é a diferença pixel a pixel do bloco do quadro atual e o bloco escolhido do quadro de referência. Um bom algoritmo de EM pode gerar resíduos de baixa amplitude, com valores tendendo a zero. O resíduo ainda pode ser muito comprimido através de outras técnicas de compressão, presentes nos codificadores de vídeo. O processo completo de compressão de vídeo em um codificador atual será apresentado no próximo capítulo deste texto.

3.1 Modelo de Codificador de Vídeos Digitais

A Figura 3.6 apresenta o diagrama em blocos de um codificador de vídeo genérico (AGOSTINI, 2007). Este modelo de codificador está de acordo com a grande maioria dos padrões de compressão de vídeo atuais. Os principais blocos que compõem o codificador são: Codificação inter-quadros, codificação intra-quadro, transformadas (T), quantização e codificação de entropia. Cada um destes blocos trabalha para reduzir um

determinado tipo de redundância existente nos vídeos digitais, sejam elas espaciais, temporais ou entrópicas.

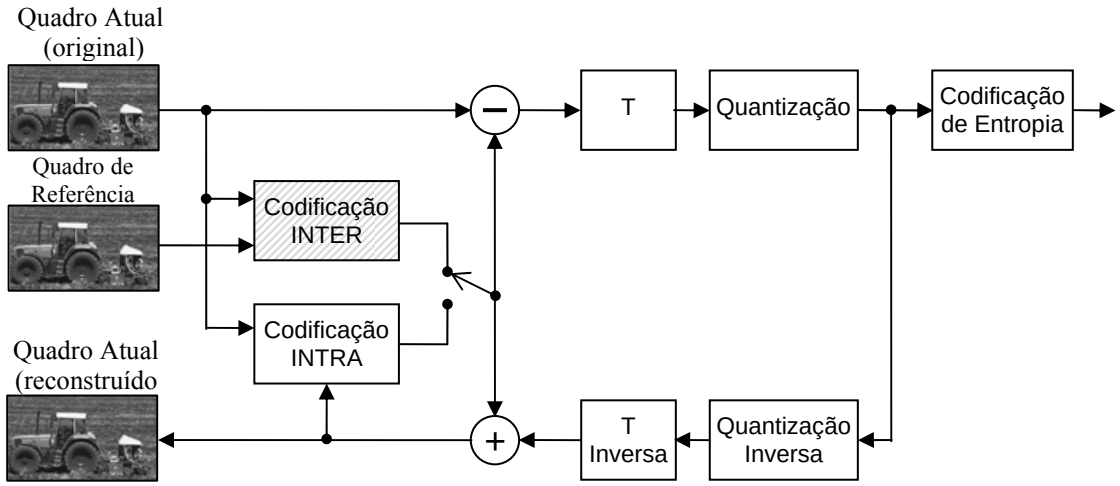


Figura 3.6: Modelo genérico de codificador de vídeo (AGOSTINI, 2007)

A EM está situada dentro do processo da codificação inter-quadros (*interframe*) e é responsável por identificar a redundância temporal entre os quadros vizinhos de uma cena. Esta é a etapa que apresenta a maior complexidade computacional de todo o codificador (PURI, 2004), no entanto, é na EM que se obtêm a maior parte dos ganhos em termos de taxa de compressão. Os vetores de movimento gerados pela EM são utilizados pela etapa de compensação de movimento – CM (*motion compensation*), que também está presente no bloco de codificação inter-quadros. A CM é responsável por remontar os quadros a partir dos vetores de movimento gerados pela EM e montar o quadro reconstruído.

O bloco de codificação intra-quadro (*intraframe*) é um dos responsáveis por reduzir as redundâncias espaciais. Diferentemente da EM, a codificação intra-quadro utiliza apenas as informação do quadro atual para realizar a predição. Vários algoritmos podem ser utilizados para este fim, como por exemplo, os algoritmos utilizados em compressão de imagens estáticas (SALOMON, 2006).

Dependendo das características do vídeo a ser codificado, uma das predições, inter-quadros ou intra-quadro, é aplicada a cada bloco do vídeo. O controle global do codificador define qual das predições será utilizada. A diferença residual, após a codificação inter-quadros ou intra-quadro, é obtida através de uma subtração dos valores do quadro atual e do quadro resultante destas predições (reconstruído). Esta diferença é chamada de resíduo. O resíduo resultante das predições (inter-quadros ou intra-quadro) deve ser somando ao quadro reconstruído, para que este quadro possa servir como referência para a codificação do próximo quadro na codificação inter-quadros. Quando a predição intra-quadro for escolhida, a reconstrução deve ser feita por blocos, pois o bloco recém predito poderá servir de referência para a predição do próximo bloco. Tanto a predição inter-quadros, quanto a predição intra-quadro, não inserem perdas no processo de codificação.

O resíduo é o resultado da subtração entre os quadros original e reconstruído (usando a predição inter-quadros ou intra-quadro). Valores baixos de resíduo indicam alta eficiência da predição, valores altos de resíduo, por sua vez, indicam baixa eficiência da predição. Valores baixos de resíduo facilitam as etapas seguintes do

processo de codificação. A etapa seguinte, após a predição inter-quadros ou intra-quadro, é a transformada (bloco T na Figura 3.6). O bloco T transforma as informações do domínio espacial para o domínio das frequências, sem perda de informação quando transformadas inteiras são utilizadas. Neste domínio, a quantização pode ser aplicada de maneira mais eficiente. A quantização das amostras transformadas é utilizada para eliminar as frequências menos relevantes ao sistema visual humano, reduzindo a quantidade de informação, com perdas visuais insignificantes. A quantização gera perdas, pois as informações eliminadas não poderão ser adicionadas novamente na decodificação do vídeo. Esta é a etapa do processo de codificação em que os codificadores sem perda e com perda se diferenciam (RICHARDSON, 2002).

A última etapa no processo de codificação é a codificação de entropia. Esta etapa é utilizada para reduzir a redundância entrópica, que está relacionada à forma como os dados são codificados. A codificação de entropia é mais uma técnica de compressão sem perdas. Diversos algoritmos de compressão sem perdas podem ser utilizados na codificação de entropia, como Huffman (SALOMON, 2008), Codificação Aritmética (SALOMON, 2006), Codificação Baseada em Dicionários (RODRIGUES, 2008), entre outros. O padrão H.264/AVC, por exemplo, utiliza os algoritmos Exp-Golomb, CAVLC e CABAC, sendo que a utilização de cada um deles depende do tipo de dado a ser processado (RICHARDSON, 2003).

O quadro atual reconstruído, presente na Figura 3.6, é gerado através das etapas de transformadas e de quantização inversa. Este processo é necessário pois as perdas geradas na quantização são irreversíveis. Desta forma, o codificador utiliza o quadro atual reconstruído como quadro de referência para a codificação inter-quadros ou intra-quadro, na codificação dos próximos blocos ou do próximo quadro. Em outras palavras, os codificadores atuais possuem, internamente, grande parte de um decodificador (sem decodificação de entropia). A última etapa do processo é a soma dos resíduos da predição, como mencionado anteriormente. Este processo é necessário para que as referências, usadas na compressão e na descompressão do vídeo, sejam idênticas.

3.2 Ferramentas de Codificação da EM em Padrões Atuais

Os padrões de compressão de vídeo atuais utilizam uma série de ferramentas que aumentam a qualidade dos vetores de movimento gerados. Estas ferramentas contribuem também para o aumento da complexidade da EM. No entanto, os padrões de compressão normatizam apenas o decodificador, deixando livres as configurações do codificador, que apenas deve gerar um *stream* de vídeo possível de ser utilizado pelo decodificador. O uso ou não destas ferramentas irá depender da finalidade do codificador e da necessidade da aplicação alvo por maior qualidade ou maior taxa de compressão.

Uma das ferramentas presentes em padrões atuais como o MPEG-2 e o H.264\AVC é o uso de múltiplos tamanhos de bloco. O padrão MPEG-2 permite que a estimação de movimento seja realizada para diferentes tamanhos de blocos, 16x16 e 16x8 (GHANBARI, 2003). O padrão H.264/AVC expandiu a possibilidade de particionamento de blocos, sendo que sete tamanhos de blocos podem ser utilizados (4x4, 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16) (RICHARDSON, 2003).

Outra ferramenta importante é a precisão fracionária de pixels. O padrão MPEG-2 prevê que os vetores de movimento podem ter precisão de até meio pixel (GHANBARI, 2003). Estas posições fracionárias devem ser geradas por interpolação, a partir dos

pixels inteiros. Mais uma vez, o padrão H.264/AVC avançou, possibilitando a precisão de um quarto de pixel. Neste caso, uma nova bateria de interpolações se faz necessária para a geração das posições de um quarto de pixel, geradas a partir das posições de meio pixel.

O uso de múltiplos quadros de referência para a geração dos vetores é outra ferramenta presente nos padrões atuais. O padrão H.264/AVC permite que a estimação de movimento utilize múltiplos quadros de referência, tanto quadros passados, quanto futuros na ordem de captura. Isso é possível porque o padrão também permite o processamento de quadros fora da ordem de captura e, deste modo, um quadro futuro pode ser processado antes e, então, pode ser usado como referência para a EM. Além disso, o vetor de movimento pode ser originado por dois blocos, oriundos de diferentes quadros, e este processo é chamado de bi-predição (RICHARDSON, 2003).

O padrão mais atual (ainda em desenvolvimento) é o HEVC (*High Efficient Video Coding*), também conhecido como H.265 (SULIVAN, 2010), (JCT-VT, 2011), (DONG, 2011). Este padrão ainda está na fase de aprovação de propostas no momento da conclusão deste texto, portanto a descrição do processo de codificação/decodificação ainda não está definida. O HEVC está propondo algumas inovações na EM, como o uso de blocos de até 64x64 pixels. Esta inovação, se adotada pelo padrão, pode ser extremamente relevante para codificação de vídeos de alta definição. A EM em vídeos de alta definição será o foco de discussão específica na seção 3.6.

O foco deste trabalho está no desenvolvimento de novos algoritmos para a estimação de movimento em vídeos de alta definição, bem como o desenvolvimento arquitetural da EM visando elevadas taxas de processamento para a codificação de vídeos em tempo real. Todas as soluções propostas neste trabalho são compatíveis com as técnicas discutidas nesta seção, sendo que o uso de cada uma delas (ou até mesmo todas juntas) irá depender da finalidade para a qual o codificador será desenvolvido.

3.3 Algoritmos de Busca na EM

Os algoritmos de busca (ou de pesquisa) para a EM definem como a busca pelo bloco mais semelhante (*best matching*) deve ocorrer dentro da área de busca no quadro de referência. O algoritmo de EM é um dos principais fatores que definem a complexidade computacional e a qualidade dos vetores de movimento gerados. Os algoritmos mais utilizados são chamados de BMAs (*Block Matching Algoriths*) e realizam a busca particionando o quadro em blocos de pixels. Os algoritmos de EM podem ser divididos em dois grandes grupos: algoritmos ótimos e subótimos (rápidos).

Algoritmos ótimos analisam todos os blocos candidatos dentro da área de busca para gerar o vetor de movimento. Desta forma, o vetor de movimento gerado corresponde ao bloco com a menor diferença em relação ao bloco atual, gerando, assim, um resíduo ótimo. Os algoritmos subótimos ou também chamados de algoritmos rápidos, utilizam heurísticas para a aceleração do processo de busca, diminuindo a quantidade de blocos candidatos utilizados e, consequentemente, reduzindo a complexidade computacional. Isto implica na geração de vetores de movimento subótimos que podem representar blocos candidatos que possuem resíduos maiores que o resíduo ótimo. Mesmo não encontrando o bloco com o resíduo ótimo, muitos algoritmos rápidos conseguem encontrar blocos candidatos com resíduos muito próximos ao valor ótimo. Desta forma, algoritmos rápidos podem obter bons resultados

em termos de qualidade, com uma redução significativa em termos de complexidade computacional.

Existem dezenas de algoritmos para EM publicados na literatura, alguns deles já foram implementados e avaliados por diversos padrões de compressão, além disso, são de conhecimento comum da comunidade científica. Dentre estes algoritmos, é possível citar: Busca completa (*Full Search* - FS) (BHASKARAN, 1999) e (LIN, 2005), *Three Step Search* (TSS) (JING, 2004), *Diamond Search* (DS) (KUHN, 1999), (ZHU, 2000) e (YI, 2005), *One at a Time Search* (OTS) (RICHARDSON, 2002), *Hexagon Based Search* (HS) (ZHU, 2002) e *Dual Cross Search* (DCS) (BANH, 2004) e (BANH, 2005), PMVFAST (LI, 2005) e (WONG, 2005). Além destes algoritmos, já conhecidos e consolidados na comunidade científica, muitos outros vem sendo propostos nos últimos anos. A seção 3.4 será dedicada a apresentação de alguns destes algoritmos, mostrando as inovações recentes em termos de desenvolvimento algorítmico para a EM.

Dentre todos os algoritmos citados acima, apenas o algoritmo *Full Search* (FS) pode ser considerado um algoritmo ótimo. Todos os demais algoritmos estão classificados como subótimos. Neste trabalho será apresentada uma breve explicação sobre o funcionamento do algoritmo FS, pois ele servirá como parâmetro para as avaliações de qualidade e complexidade de todos os demais algoritmos. O FS será usado como limite superior em termos de qualidade. Além disto, o algoritmo FS é largamente utilizado para soluções em hardware e o desenvolvimento de arquiteturas de hardware dedicadas para a EM é um dos focos deste trabalho.

A maior parte dos esforços de pesquisa em desenvolvimento algorítmico para a EM, estão focadas nos algoritmos rápidos. A maior parte dos algoritmos rápidos possui um padrão de busca. Este padrão é aplicado na área de busca e repetido até que uma dada condição de parada seja alcançada. O número de iterações do algoritmo (repetições do padrão de busca) pode variar consideravelmente entre um bloco e outro. Esta característica dificulta o desenvolvimento de soluções arquiteturais, pois é impossível determinar, a priori, quantos ciclos de *clock* serão necessários para a geração de um vetor de movimento. Além disto, os algoritmos baseados em padrões de busca possuem uma forte dependência de dados, sendo que a próxima iteração do algoritmo depende dos resultados gerados no passo atual. Esta característica também dificulta o desenvolvimento deste tipo de algoritmo em hardware, pois limita a exploração do paralelismo e dificulta o gerenciamento da memória local. Neste trabalho, apenas os algoritmos rápidos DS, TSS e OTS serão apresentados com maiores detalhes. Estas considerações serão importantes para a melhor compreensão dos algoritmos desenvolvidos e apresentados no capítulo 6.

3.3.1 *Full Search* (FS)

O algoritmo FS avalia todos os blocos candidatos de uma dada área de busca, logo, ele sempre encontrará o bloco candidato de menor resíduo, gerando um vetor de movimento ótimo (LIN, 2005). Cada bloco do quadro atual deverá ser deslocado amostra a amostra dentro da área de busca, em busca do melhor casamento. A forma como o bloco é deslocado pode variar, começando do canto superior esquerdo, até o canto inferior direito ou partindo do centro a área de busca em espiral, por exemplo (RICHARDSON, 2002). Quando todos os blocos candidatos da área de busca forem comparados e a menor diferença tenha sido encontrada, o vetor de movimento é gerado para este bloco, indicando a distância entre a posição original do bloco e o bloco

candidato de menor diferença. A Figura 3.7 ilustra o algoritmo FS percorrendo uma área de busca, iniciando no canto superior esquerdo.

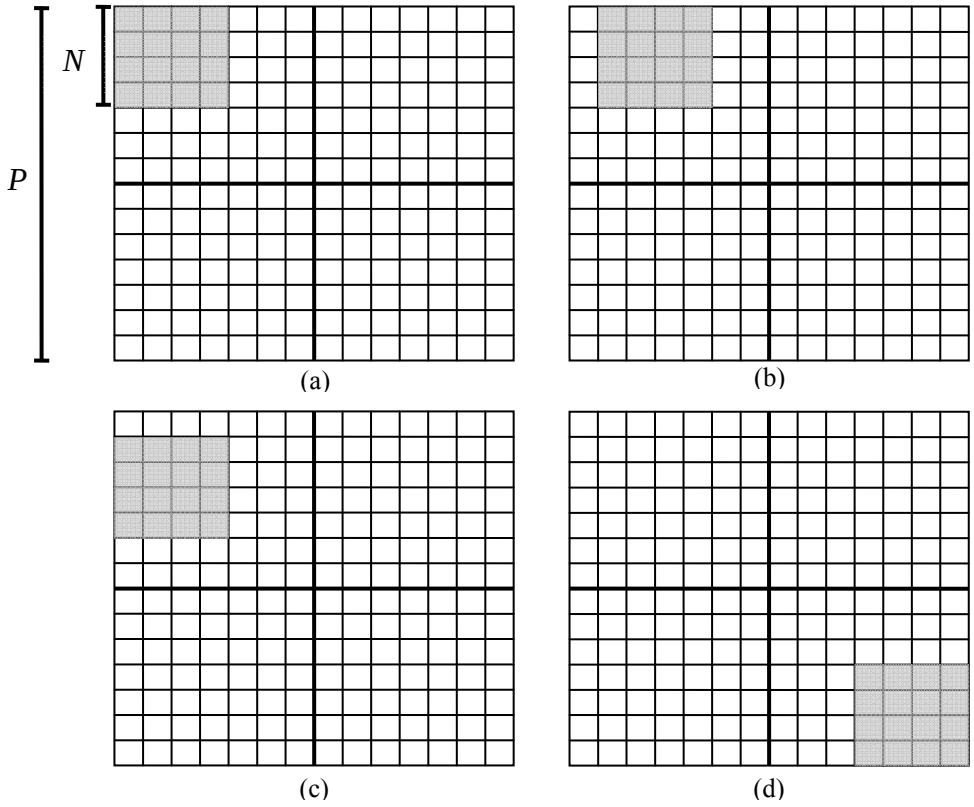


Figura 3.7: Algoritmo de Busca Completa (FS)

A área de busca apresentada na Figura 3.7 é de +/- 5 amostras ao redor de um bloco atual de 4x4 amostras (considerando a posição original do bloco no centro área de busca). Cada quadrado na figura representa uma amostra e as regiões em cinza representam o posicionamento do bloco do quadro atual. O FS compara o bloco atual com o primeiro bloco candidato (Figura 3.7 (a)), aplicando o critério de similaridade (SAD, por exemplo) a cada uma das amostras do bloco. Logo após, o algoritmo desloca o bloco atual uma amostra para a direita dentro da área de busca e calcula novamente a diferença (Figura 3.7 (b)). Este processo é repetido para todos os blocos candidatos da linha, sempre deslocando uma amostra para a direita. Depois, a busca continua com o deslocamento de uma amostra para baixo e o processo é repetido para todos os blocos candidatos da linha, começando pelo primeiro à esquerda (Figura 3.7 (c)). A busca termina quando todos os blocos candidatos de todas as linhas da área de busca forem avaliados.

Dada uma área de busca de tamanho *P*x*P* e blocos de tamanho *N*x*N*, o número de blocos candidatos presentes na área de busca será igual a (*P*-*N*+1)*(*P*-*N*+1). O critério de similaridade, por exemplo, o SAD, deverá ser aplicado a todos os blocos avaliados, logo, *N***N* comparações entre pixels serão necessários por bloco candidato. Uma área de busca deve ser criada para cada bloco do quadro atual. Considerando um quadro com um número *X* de blocos, um total de $X*N^2*(P-N+1)^2$ comparações serão necessárias para gerar os vetores de movimento dos X blocos do quadro atual. Como exemplo, considerando um quadro HDTV com resolução de 1920x1080, área de busca de 64x64 e blocos de tamanho 4x4 amostras, existem 129.600 blocos em um quadro e 3.721 blocos candidatos em uma área de busca. Para cada quadro, serão necessários

7.715.865.600 comparações para a geração dos vetores de movimento de todos os blocos deste quadro.

O algoritmo FS possui uma grande complexidade computacional. Como o FS compara todos os blocos da área de busca, a complexidade do FS está diretamente ligada ao tamanho desta área, sendo que o custo computacional do FS cresce proporcionalmente ao aumento da área. O aumento do tamanho da área de busca pode proporcionar melhorias consideráveis em relação à redução do resíduo, no entanto, estes ganhos em qualidade não são proporcionais ao aumento da área (PORTO, 2008). Esta complexidade computacional torna praticamente impossível à utilização do algoritmo FS em software para aplicações que trabalhem com vídeos de alta resolução em tempo real. Nestes casos, se faz necessário o desenvolvimento de soluções arquiteturais dedicadas. Apesar da enorme quantidade de cálculos, o algoritmo FS não apresenta dependência de dados, possibilitando a livre exploração do paralelismo. Arquiteturas de hardware podem explorar grandes níveis de paralelismo, com grande custo em área e potência, mas atendendo os requisitos de desempenho necessários. Este tipo de solução pode ser interessante quando a aplicação estiver focada em qualidade.

3.3.2 *Diamond Search* (DS)

O algoritmo *Diamond Search* (DS) é um dos algoritmos rápidos mais conhecidos e utilizados na literatura. O DS pode reduzir significativamente a complexidade computacional da EM, mantendo bons resultados de qualidade, em comparação ao algoritmo FS. Este algoritmo utiliza um padrão em forma de diamante para realizar a busca dentro da área de busca e um segundo padrão diamante para o refinamento final da escolha. A Figura 3.8 ilustra estes dois padrões de busca, chamados de *Large Diamond Search Pattern* (LDSP) e *Small Diamond Search Pattern* (SDSP) (ZHU, 2000). Na Figura 3.8, cada intersecção representa um bloco candidato e os círculos representam o posicionamento do bloco do quadro atual. Os círculos em branco (L) representam o padrão LDSP e os círculos em cinza (S) representam o padrão SDSP. O padrão LDSP forma um diamante de nove blocos candidatos ao redor do centro da área de busca. Este padrão é utilizado na primeira etapa da busca. O padrão SDSP é formado por quatro blocos candidatos ao redor da posição central do LDSP e é utilizado na etapa final da pesquisa, com o intuito de refinar o resultado obtido na etapa anterior (KUHN, 1999).

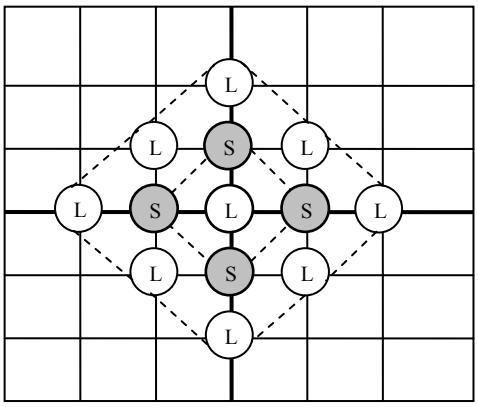


Figura 3.8: *Large Diamond* (LDSP) (L) e *Small Diamond* (SDSP) (S) (PORTO, 2008)

O algoritmo DS inicia a busca aplicando o padrão LDSP, com nove comparações, ao centro da área de busca. Esta é uma estratégia comum a grande maioria dos algoritmos rápidos, visando explorar a característica intrínseca dos vídeos digitais, que

tendem a apresentar bons resultados para blocos candidatos próximos a posição original (KUO, 2009) e (LAI, 2010). A primeira etapa do algoritmo DS acaba quando o menor resíduo (menor diferença) for encontrado no centro do LDSP. Se esta condição for satisfeita na primeira aplicação do LDSP, o algoritmo utiliza o padrão SDSP para refinar o resultado obtido. Isto implica na comparação de mais quatro blocos candidatos com uma amostra de distância para cima e para baixo e para ambos os lados. Após a aplicação do SDSP, o vetor de movimento é gerado para o bloco que resultar no menor resíduo. No entanto, esta condição em que o melhor resultado é encontrado no centro do LDSP logo na primeira iteração não tende a ser a mais provável. Isto ocorre apenas em cerca de 23% dos vetores de movimento (PORTO, 2008a).

Para cerca de 77% dos vetores(PORTO, 2008a), o algoritmo DS utiliza mais de uma aplicação do LDSP. Nestes casos, um novo LDSP é aplicado, tendo como novo centro a posição de menor resíduo da etapa anterior. Esta posição pode pertencer a uma aresta ou a um vértice do diamante, sendo que para cada uma destas situações existe um procedimento diferente. A busca por uma aresta e a busca por um vértice estão representadas na Figura 3.9 (a) e (b), respectivamente. Os círculos em branco (1) representam o primeiro LDSP e os círculos em cinza (2) representam o segundo.

Na busca por uma aresta, três novos blocos candidatos são comparados, formando um novo diamante ao redor do novo centro (posição de menor resíduo do primeiro LDSP). Quando o menor resíduo é encontrado em um vértice do diamante, mais cinco blocos candidatos devem ser comparados, formando um novo diamante ao redor do novo centro. Quando o menor resíduo não é encontrado no centro do novo LDSP, a etapa de busca será repetida, seja com uma busca por aresta ou por um vértice. Quando o menor erro for encontrado para o centro do LDSP, o padrão SDSP é aplicado.

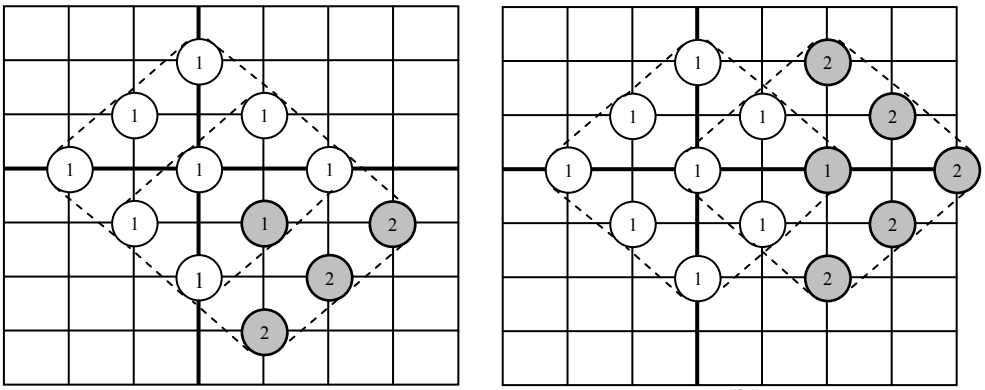


Figura 3.9: Aplicação do LDSP (a) por aresta e (b) por vértice (PORTO, 2008)

Não é possível determinar quantas repetições (iterações) do padrão LDSP serão necessárias até que o melhor resultado seja encontrado no centro. Em média, o algoritmo DS utiliza cerca de três iterações do padrão LDSP em sequências de vídeo SDTV (740x480 pixels) (PORTO, 2008), no entanto, este número oscila de acordo com as características do vídeo a ser processado. Esta característica dificulta o cálculo de desempenho do algoritmo DS, pois o número de iterações, e consequentemente o número de cálculos, está diretamente ligado às características do vídeo.

Considerando o melhor caso, quando o algoritmo gera o vetor sem nenhuma iteração, apenas 13 blocos candidatos serão avaliados para a geração de um vetor de movimento. Para um caso médio, onde três iterações são utilizadas e considerando uma

média de quatro blocos candidatos comparados por cada novo LDSP (média entre os três necessários na busca por aresta e os cinco utilizado na busca por vértice), apenas 25 blocos candidatos serão analisados. Utilizando o mesmo exemplo apresentado anteriormente para o algoritmo FS, o algoritmo DS utilizaria 26.956.800 comparações, no melhor caso, e 51.840.000, no caso médio. Comparado com o valor obtido para o FS, o algoritmo DS apresenta uma redução estimada no número de comparações que varia entre 149 (caso médio) e 286 (melhor caso) vezes.

3.3.3 *Three Step Search* (TSS)

O algoritmo *Three Step Search* é mais difundido com o nome de busca em três passos (3SS ou TSS), no entanto, ele pode ser estendido para um busca em *n* passos (JING, 2004). Neste texto será abordada apenas a versão com três passos, no entanto, a exploração em *n* passos segue o mesmo princípio. Diferentemente do algoritmo DS, o TSS possui um número fixo de comparações. Esta é uma vantagem do algoritmo TSS, que possui regularidade em termos de complexidade computacional, pois todos os vetores de movimento serão gerados com o mesmo número de comparações. A Figura 3.10 ilustra o algoritmo TSS para uma área de busca de +/- 7 amostras. Os blocos candidatos estão representados com círculos na Figura 3.10, sendo que os blocos candidatos de menor SAD, escolhidos a cada passo do algoritmo, estão destacados em cinza. O resultado final está destacado em preto.

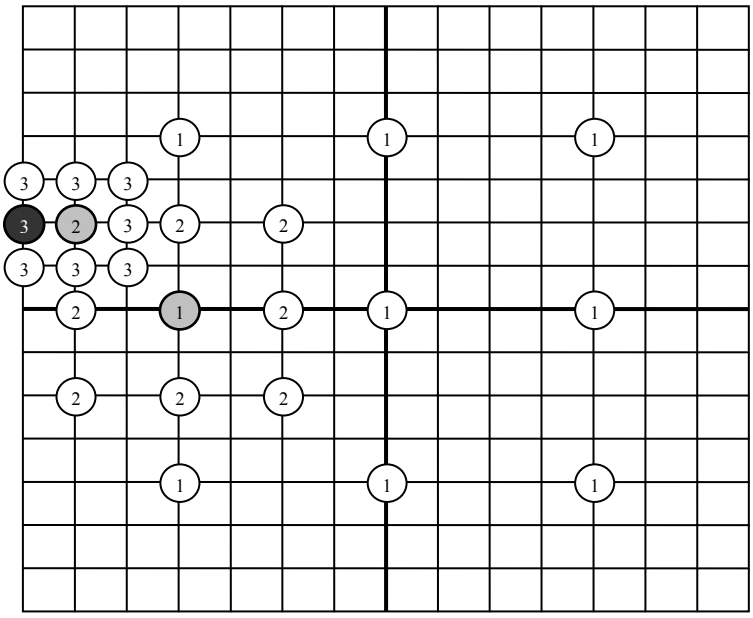


Figura 3.10: Algoritmo *Three Step Search* (PORTO, 2008)

No primeiro passo, o algoritmo compara nove blocos candidatos: o bloco central e mais oito blocos a uma distância *D* amostras ao redor do centro da área de busca (no exemplo $D = 4$). Os nove blocos candidatos são comparados e o que resultar no menor SAD será o novo centro da busca. O valor da distância é atualizado para $D = D/2$. No segundo passo, mais oito blocos candidatos com uma distância *D* ao redor do centro da busca (melhor resultado do passo anterior) são avaliados e a distância de *D* é novamente atualizada para $D = D/2$. Novamente, o bloco candidato de menor SAD será o centro da busca para o terceiro e último passo do algoritmo. Nesta etapa, mais oito blocos candidatos são comparados e o bloco que resultar no menor SAD será escolhido para a geração do vetor de movimento. Os três passos do algoritmo TSS resultam na comparação de 25 blocos candidatos (9+8+8) e este é o custo fixo do algoritmo TSS

para a geração de um vetor de movimento. Diferentemente do algoritmo DS, as características do vídeo não interferem no custo computacional do algoritmo TSS, pois não existem etapas iterativas neste algoritmo.

3.3.4 *One at a Time Search* (OTS)

O algoritmo OTS é um dos algoritmos rápidos mais simples e que apresenta um dos menores custos computacionais dentre todos os algoritmos da literatura. O OTS utiliza uma estratégia de encontrar um bloco candidato com menor resíduo em uma busca horizontal e, em seguida, um bloco candidato com menor resíduo em uma busca vertical (RICHARDSON, 2002). O custo computacional do algoritmo OTS é baixo, pois as iterações são realizadas de uma em uma amostra, logo, existe uma grande tendência a uma rápida convergência, tanto na busca vertical quanto na busca horizontal. O OTS inicia a busca calculando o bloco candidato da região central da área de busca, em seguida, os dois blocos candidatos vizinhos, um imediatamente à direita e outro imediatamente à esquerda, são comparados. Caso o valor do centro seja o menor, o algoritmo passa para o estágio de busca vertical, caso contrário, o novo centro da busca será o bloco candidato de menor SAD. O próximo bloco candidato, à direita ou à esquerda (de acordo com o bloco candidato escolhido no passo anterior) deve ser comparado e este processo deve ser repetido até que uma nova iteração não resulte em um resultado de SAD inferior ao obtido na iteração anterior. A etapa de busca vertical é muito semelhante, no entanto, a variação na busca ocorre para cima e para baixo. A Figura 3.11 ilustra uma busca do algoritmo OTS em uma área de busca de +/- 7 amostras.

No exemplo da Figura 3.11 o algoritmo OTS calcula o bloco candidato do centro e seus vizinhos à esquerda e a direita e o menor resultado é obtido para o bloco candidato à esquerda. Esta escolha resulta em mais quatro iterações, sempre avançando uma amostra para a esquerda, até que o menor resultado de SAD é encontrado para o bloco candidato do centro da busca horizontal. O próximo passo do algoritmo é iniciar a busca vertical, calculando os vizinhos superior e inferior, sendo que o vizinho superior obteve o menor SAD. Mais três iterações são realizadas e, então, o menor resultado de SAD é encontrado (bloco 8 na Figura 3.11).

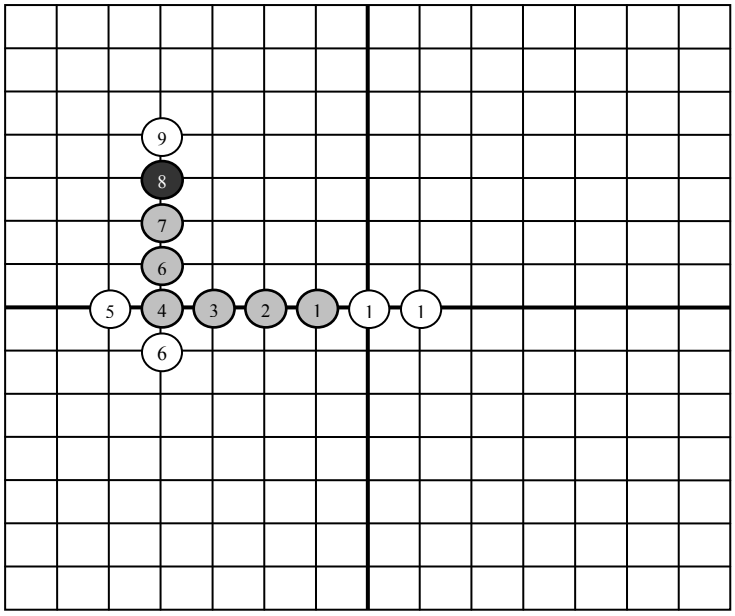


Figura 3.11: Algoritmo *One at a Time Search*

O algoritmo OTS não possui um número fixo de comparações, no entanto, o valor máximo de comparações para uma determinada área de busca pode ser calculado. Dada uma área de busca de +/- *P*, o valor máximo de comparações do algoritmo OTS será Max = 2**P*+3. Para o exemplo apresentado na Figura 3.11, o número máximo de comparações será igual a dezessete (2*7+3). No entanto, no exemplo ilustrado na Figura 3.11, apenas doze comparações são realizadas. No melhor caso, o algoritmo pode concluir a estimação em uma área de pesquisa calculando a diferença para apenas cinco posições.

3.4 Estado da Arte em Algoritmos de Busca para a EM

Os padrões de compressão de vídeo atuais introduziram diversas ferramentas que contribuem diretamente na qualidade dos vetores de movimento gerados pela EM, como apresentado na seção 3.2. O uso destas ferramentas pode refinar as escolhas feitas pelos algoritmos de busca, contribuindo assim com o ganho em qualidade. No entanto, caso o algoritmo de busca seja ineficiente, escolhendo blocos candidatos ruins dentro da área de busca, o uso destas ferramentas pode se tornar ineficiente, pois o refinamento de um resultado ruim pode continuar sendo ruim. O algoritmo de busca é um dos principais parâmetros da EM, tanto para a qualidade, quanto para a complexidade computacional. Este fator fez com que os esforços no desenvolvimento de novos algoritmos de EM tenham se mantido desde a sua criação, sendo um tema relevante na comunidade científica atualmente.

O trabalho de (PO, 2009) apresenta um novo algoritmo de EM chamando DGDS (*Directional Gradient Descent Search*). Este algoritmo combina dois algoritmos, o clássico OTS (*One at a Time Search*) e outro desenvolvido previamente BBGDS (*Block-Based Gradient Descent Search*) (LIU, 1996). O artigo apresenta uma segunda versão do algoritmo chamada FDGDS (*Fast Directional Gradient Descent Search*), que utiliza limiares (*thresholds*) durante a escolha dos blocos candidatos. O valor do limiar define um valor considerado aceitável para o resíduo, sendo que quando este valor é atingido, a busca termina, mesmo que a condição de parada do algoritmo não tenha sido satisfeita. Esta técnica é conhecida como antecipação de parada (*early termination)*, pois pode encerrar a busca. Esta versão pode ser consideravelmente mais rápida que a versão original, dependendo do valor de limiar utilizado. Quanto maior o valor do limiar, mais rápido será o algoritmo, no entanto, piores serão os resultados de qualidade. Todos os resultados apresentados neste artigo consideraram apenas sequências de teste na resolução CIF (352x288 pixels).

Outro novo algoritmo para a EM é apresentado por (TASDIZEN, 2009). Este algoritmo é chamado de DVSS (*Dynamically Variable Step Search*) e é derivado de um trabalho anterior do autor (TASDIZEN, 2009a), que por sua vez, é baseado no algoritmo NSS (N *Step Search*). No entanto, neste algoritmo o número de passos a serem executados, bem como a distância deles, é calculado através do valor do vetor de movimento encontrado para o seu bloco vizinho do lado esquerdo. Além disto, o algoritmo também utiliza limiares para acelerar o processo de busca. Os resultados, apresentados no artigo, mostram que o algoritmo proposto obtêm resultados de qualidade próximos aos obtidos pelo algoritmo FS, com uma redução significativa do número de blocos candidatos avaliados. Apesar do autor mencionar que apresenta algoritmos para a estimação de movimento em vídeos HD, todos os resultados são apresentados para vídeos de resolução CIF ou 720x576 pixels, ou ainda vídeos CIF (352x288) com *upscaling* para 720x576 pixels.

Em (CHENG, 2009) um algoritmo focado em vídeos de alta definição é apresentado. Este algoritmo explora a característica inerente aos vídeos de que blocos próximos tendem a apresentar pouca diferença entre si. Sendo assim, o algoritmo aplica uma busca completa dentro de uma pequena área (+-8 amostras) ao redor da área de busca. Além disso, o algoritmo também realiza uma busca em uma área consideravelmente maior (tipicamente +- 128 amostras) e, então, aplica um algoritmo rápido de busca com apenas 25 comparações. As duas etapas do algoritmo são paralelizáveis, tornando este algoritmo uma opção para o desenvolvimento em hardware. O artigo apresenta os resultados da integração deste algoritmo ao software de referência do padrão H.264/AVC. Os resultados foram gerados para sequências HD e apresentam pequenas perdas em qualidade, com ganhos significativos na redução do tempo de processamento. No entanto, a avaliação usando o software de referência não permite isolar os ganhos causados apenas pela EM, pois os resultados de qualidade são obtidos considerando todo o processo de compressão. Assim, os ganhos ou as perdas de qualidade podem ser potencializados por outros módulos do compressor e a avaliação dos ganhos ou perdas da EM fica prejudicada.

Outro algoritmo de EM voltado para vídeos de alta definição é apresentado em (CETIN, 2010) e (CETIN, 2011). Este algoritmo é chamado de *Adaptive True Motion Estimation* (ATME) e é uma evolução do algoritmo 3-D *Recursive Search* (HAAN, 1993). Este algoritmo é focado em baixo custo computacional e apresenta diferentes níveis de configuração com resultados de qualidade e custo computacional distintos (CETIN, 2011). Os resultados são apresentados para cinco sequências de teste HD 720p e três sequências HD 1080p para uma área de busca de [-64, +64] e blocos 16x16. Mesmo tendo foco em vídeos de alta definição, o algoritmo ATME apresenta perdas de qualidade na comparação dos resultados obtidos para sequências HD 720p e versões da mesma sequência de vídeo em HD 1080p. Isto demonstra que o aumento da resolução do vídeo afeta negativamente os resultados de qualidade. Os resultados apresentados em (CETIN, 2011) serão comparados com os resultados obtidos nesta tese na seção 5.5.

Uma variação do algoritmo DS, chamada HMDS (*Hardware-oriented Modified Diamond Search*) é apresentada por (NDILI, 2010). Este algoritmo também utiliza dois padrões de busca, no entanto, o primeiro é uma variação do padrão LDSP com 17 comparações. O algoritmo também explora a região central, sendo que o primeiro padrão calcula os nove pontos ao redor do centro e depois expande a busca em forma de cruz. O melhor resultado desta etapa será o centro da busca na etapa seguinte, onde o padrão LDSP é aplicado. Apesar de o autor dar ênfase no fato do algoritmo ser orientado para o desenvolvimento em hardware, nenhuma proposta de solução arquitetural é apresentada. O autor salienta, apenas, que os padrões regulares facilitam a organização da memória. Os resultados de qualidade também foram obtidos através do software de referência do padrão H.264/AVC e mostram que o algoritmo apresenta pequenas perdas em relação ao algoritmo FS, com resultados similares a outros algoritmos rápidos de EM. No entanto, mais uma vez, apenas sequências de baixíssima resolução, como QCIF e CIF, foram utilizadas nos testes.

O algoritmo proposto em (KUO, 2009) explora o princípio de que, na média, os melhores vetores são encontrados próximos ao centro (*center–biased*). O algoritmo avalia cinco pontos imediatamente ao redor do centro da área de busca. Esta posição da busca é determinada a partir da predição do vetor do bloco anterior. Em uma segunda etapa, 11 diferentes padrões, comparando três blocos candidatos, podem ser utilizados. A utilização de cada um destes padrões irá depender da predição feita a partir do vetor

de movimento do bloco anterior. Os resultados de qualidade apresentados são muito similares aos obtidos por algoritmos rápidos clássicos, como o DS e o TSS, com uma redução significativa do número de blocos comparados. No entanto, apenas sequências QCIF e CIF foram utilizadas nas avaliações.

Novos algoritmos vêm sendo propostos focando em ferramentas específicas dos novos padrões de compressão de vídeo, como o H.264/AVC, por exemplo. O algoritmo apresentado em (BYUN, 2010) considera a estimação de movimento com foco na utilização de múltiplos quadros de referência. A estratégia é realizar uma busca completa (FS) dentro da área de busca dos quadros mais próximos (temporalmente). Para os demais quadros, a área de busca é determinada através de uma relação linear entre o tamanho dos vetores e a distância dos quadros de referência, em relação ao quadro atual. O autor destaca que esta solução não utiliza antecipação de parada, o que garante o tempo de execução fixo para todos os vetores, facilitando a implementação em hardware deste algoritmo. A grande maioria dos resultados é apresentada para sequências de resolução QCIF e CIF, apenas três sequências com resolução 720p foram avaliadas. As comparações não consideram algoritmos clássicos, como o DS, por exemplo. Além disto, os resultados de qualidade não foram contrastados com o algoritmo FS.

Outro algoritmo, apresentado em (AKRAM, 2010) também propõe uma estratégia para a estimação de movimento em múltiplos quadros de referência. Para o primeiro quadro de referência (o mais próximo temporalmente) um algoritmo de eliminação sucessiva (SEA – *Successive Elimination Algorithm*) (LI, 1995) é utilizado. Para os demais quadros de referência, a diferença simples (sem EM) em relação ao quadro atual é calculada. Analisando os valores de diferença é possível encontrar regiões com baixo resíduo onde os melhores blocos candidatos podem estar situados. Os resultados de qualidade apresentados são muito parecidos com os obtidos pelo algoritmo FS, quando sequências CIF são consideradas.

O algoritmo apresentado em (SARWER, 2009) foi desenvolvido focando no uso de todos os tamanhos de partição de blocos previstos no padrão H.264/AVC. Para acelerar o processo de busca, o algoritmo utiliza antecipação de parada. Além disto, a região da área de busca é dividida em setores, considerando a probabilidade de ocorrência. Desta forma, o algoritmo pode satisfazer a condição de parada (valor abaixo do limiar) mais rapidamente. A avaliação de probabilidades foi realizada com o software de referência do padrão H.264/AVC, baseada nas características estatísticas de custo de taxa-distorção (RD – *rate distortion*). Esta avaliação também foi usada para a determinação do valor do limiar utilizado na antecipação de parada. Mais uma vez, apenas sequências QCIF e CIF são consideradas na apresentação dos resultados.

Analisando os diversos algoritmos mencionados, é possível perceber que diversas estratégias são comuns a diversos deles. O uso de limiar, por exemplo, é uma estratégia bastante utilizada para implementar a antecipação de parada e acelerar o processo de busca. No entanto, a eficiência desta técnica, em termos de qualidade, está diretamente ligada ao valor do limiar utilizado. Uma grande bateria de simulações, considerando dezenas de vídeos com características diferentes de movimentação e conteúdo deve ser feita para que dados consistentes de resíduo sejam gerados e, então, um valor de limiar possa ser definido. Além disto, com uso da antecipação de parada é difícil determinar o tempo (ou número de ciclos) necessário para a geração de cada vetor. Esta característica dificulta a implementação destes algoritmos em hardware.

Outra estratégia muito utilizada é a combinação de mais de um algoritmo de busca. Alguns algoritmos combinam vantagens apresentadas por determinados algoritmos e, então, realizam a busca em duas ou mais etapas. Cada etapa se beneficia de uma determinada vantagem que um algoritmo já conhecido pode oferecer. Os algoritmos utilizados nestas combinações podem ser originais ou com alguma alteração no seu padrão de busca, dependendo da finalidade.

A predição de partida também é bastante explorada em algoritmos recentes. Esta estratégia pode contribuir para evitar a escolha de um mínimo local. Um mínimo local é um bloco candidato que apresenta um resíduo pequeno (em relação aos demais em uma dada região), no entanto, este resíduo não representa o mínimo global, que é ainda menor e pode estar situado em outra região. O tema mínimos locais será abordado novamente, com maiores detalhes, na seção 3.6. Algoritmos *center-biased* podem evitar mínimos locais ao considerarem o centro da busca como sendo a posição indicada pelo vetor de movimento do bloco anterior. A predição de partida pode contribuir também para a redução do número de iterações dos algoritmos, pois já indica a região onde a busca deve começar.

Vários algoritmos também consideram características que favoreçam a sua implementação em hardware. A EM é a etapa mais complexa dos codificadores de vídeo e o seu desempenho, para aplicações em tempo real, é crítico, principalmente considerando vídeos de alta definição. Neste cenário, o desenvolvimento de arquiteturas de hardware dedicadas é de estrema relevância. No entanto, a grande maioria dos algoritmos rápidos possui características que dificultam a sua implementação em hardware, como dependência de dados, irregularidade no acesso a memória e tempo de execução variável (e não previsível).

O grande problema, associado a grande maioria dos artigos encontrados na literatura, que apresentam algoritmos de EM, está na resolução das sequências de teste utilizadas para a geração dos resultados. As diferenças de qualidade entre os algoritmos de EM é praticamente nula para sequências de baixa resolução, como QCIF e CIF. Os resultados dos algoritmos de EM podem sofrer grandes alterações, quando aplicados a vídeos de alta definição. Desta forma, é difícil avaliar a contribuição efetiva destes novos algoritmos. Na seção 3.6 serão apresentados alguns resultados que confirmam estas afirmações. O cenário atual requer soluções eficientes voltadas para vídeos de alta definição, portanto, novos algoritmos de EM devem ser avaliados considerando este tipo de vídeo. Assim, é possível avaliar a relevância deste novo algoritmo em relação a outras propostas da literatura.

Neste trabalho, todas as propostas algorítmicas serão voltadas para vídeos de alta definição. Os algoritmos de EM desenvolvidos serão focados em solucionar problemas que são mais evidentes neste tipo de vídeo.

3.5 Estado da Arte em Arquiteturas para a EM

O desenvolvimento de arquiteturas dedicadas para a EM também é um tema que vem sendo explorado há bastante tempo pela comunidade científica. A crescente demanda por altas taxas de processamento, gerada pelo progressivo aumento da resolução dos vídeos digitais, fazem com que o desenvolvimento de arquiteturas dedicadas à EM tenham uma importância ainda maior. Além da complexidade, inerente ao se trabalhar com vídeos de alta resolução, as diversas ferramentas apresentadas pelos padrões de compressão recentes aumentam significativamente a complexidade da EM,

exigindo um desempenho ainda maior. Esta complexidade dificulta o desenvolvimento de aplicações em software para a compressão de vídeos de alta definição em tempo real (30 quadros por segundo).

A arquitetura apresentada em (PORTO, 2009) está focada em uma das ferramentas do padrão H.264/AVC, o uso de múltiplos tamanhos de bloco. Esta arquitetura trabalha sobre blocos de tamanho 4x4 guardando os resíduos de todos os blocos candidatos. Uma hierarquia de memórias é utilizada para a combinação dos resíduos, gerando os resultados para todos os demais tamanhos de bloco (4x8, 8x4, 8x8 e 16x16). Os resultados de síntese para *standard cells* 0.18µm mostram que esta arquitetura atinge uma frequência máxima de 296MHz e pode processar até 34 quadros HD 1080p por segundo. O número de *gates* utilizados é de 113K, para uma área de busca de 16x16 amostras. Apesar de alcançar o desempenho necessário para processar vídeos HD 1080p em tempo real, a área de busca suportada é muito pequena para vídeos desta resolução.

Outra solução arquitetural para o algoritmo FS é apresentada em (KAO, 2009). Esta arquitetura também possui suporte para múltiplos tamanho de bloco, além disto, suporta o uso de vetores de movimento fracionários. A frequência de operação para processar vídeos HD 1080p é de 150MHz, menor que a frequência apresentada em (PORTO, 2009). No entanto, esta diminuição da frequência de operação foi obtida com o paralelismo utilizado para os cálculos dos vários tamanhos de bloco. Três unidades de processamento são utilizadas em paralelo, a primeira para os blocos 8x8 e 4x4, a segunda para os blocos 8x4 e 4x8 e a terceira para os blocos 8x16, 16x8 e 16x16. Com isto, o número de *gates* utilizados na síntese para *standard cells* na tecnologia 0,18µm é de 321K. A área de busca suportada por esta arquitetura não foi mencionada no artigo.

Uma arquitetura de hardware para a EM utilizando um algoritmo rápido é apresentada em (TASDIZEN, 2009). O algoritmo utilizado também é proposto pelo autor, conforme apresentado na seção anterior. Esta arquitetura apresenta um *array* reconfigurável para o cálculo dos SADs em blocos de 16x16 amostras e suporta uma área de busca de 48x24 amostras. Os resultados de síntese foram obtidos com uma síntese para um FPGA. A arquitetura ocupa uma área de aproximadamente 9,1K CLBs (*Configurable Logic Block*) e atinge uma frequência máxima de operação de 130MHz. Com esta frequência, a arquitetura processa mais de 34 quadros HD 1080p por segundo, utilizando um limiar de 256. Esta arquitetura possui o mesmo desempenho da arquitetura apresentada em (PORTO, 2009), no entanto, com uma frequência de operação consideravelmente menor. Em relação à qualidade, a comparação é extremamente complexa. O uso de um algoritmo rápido já pressupõe alguma perda na qualidade, assim como o uso de um limiar, no entanto, a área de busca utilizada (consideravelmente maior) pode compensar parcialmente esta diferença.

A arquitetura apresentada em (WANG, 2009) explora mais intensamente a questão do baixo consumo de potência. Esta arquitetura utiliza um algoritmo rápido desenvolvido anteriormente por um dos autores chamado ABEM (*All Binary Motion Estimation*) (LUO, 2002). No entanto, os resultados de qualidade deste algoritmo não são apresentados. A arquitetura suporta uma área de busca de 48x48 amostras, e trabalha com dois tamanhos de blocos, 8x8 e 16x16 amostras. Esta arquitetura pode operar com uma potência de 0,89mW, operando a uma frequência de 1,94MHz, com síntese para *standard cells* na tecnologia de 0,18µm. A utilização dos recursos de hardware é de 62,2K *gates*. A potência apresentada é realmente baixa, no entanto, os resultados de qualidade não são apresentados e o desempenho da arquitetura suporta apenas a resolução CIF.

O trabalho publicado em (YIN, 2010) apresenta um arquitetura de hardware para a EM voltada para vídeos de alta definição. Esta arquitetura implementa uma variação do algoritmo MMEA (*Multi-resolution ME Algorithm*), que utiliza subamostragem de pixels para a redução do número de comparações (LIN, 2008). Com o uso da subamostragem, é possível reduzir o tamanho da arquitetura (número de transistores), mantendo o desempenho. Além disto, esta arquitetura apresenta uma estrutura de reaproveitamento da memória interna, reduzindo o número de acessos à memória externa. Esta arquitetura também implementa a maioria das ferramentas do padrão H.264/AVC, como múltiplos tamanhos de blocos e quadros de referência passados e futuros. Os resultados de síntese para a tecnologia 0,18µm mostram que a arquitetura pode processar vídeos HD 1080p a uma frequência de 200MHz, utilizando cerca de 260K *gates*.

Outra arquitetura de hardware focada em vídeos de alta definição é apresentada em (CETIN, 2011). Este trabalho apresenta uma arquitetura de hardware para o algoritmo ATME, com desempenho suficiente para processar vídeos HD 720p em tempo real, quando implementada em FPGA. Esta arquitetura trabalha com blocos de 16x16 pixels, e utiliza 256 unidades de processamento de SAD, uma para cada pixel. Desta forma, é possível atingir uma elevada taxa de processamento com uma baixa frequência de operação.

O uso de subamostragem, para a aceleração dos cálculos de SAD também é a estratégia utilizada na arquitetura apresentada em (LAI, 2010). Esta arquitetura utiliza um algoritmo do tipo *Top-Winners*, que é dividido em duas etapas. Na primeira etapa, uma busca em espiral é realizada, utilizando subamostragem de nível 8:1 (calcula um SAD a cada 8 amostras) e antecipação de parada. A busca é realizada em espiral para explorar a maior probabilidade de encontrar um bom resultado próximo ao centro, acelerando o processo de busca. Então, um número $\boldsymbol{N}$ de blocos candidatos (os que apresentam o menor SAD) são escolhidos. Na segunda etapa, os SADs para estes $\boldsymbol{N}$ blocos candidatos são calculados, sem subamostragem e, então, o bloco que obtiver o menor SAD será escolhido. O artigo apresenta resultados de qualidade considerando a variação do nível de subamostragem e o valor de $\boldsymbol{N}$. A arquitetura trabalha sobre blocos de 16x16 amostras, em uma área de busca de 48x48 amostras. Os resultados de síntese para a tecnologia 0,18µm mostram que a arquitetura utiliza uma pequena quantidade de recursos de hardware, cerca de 26K *gates*. A frequência máxima de operação é de 83,3MHz, o que é suficiente para processar vídeos 800x600 pixels em tempo real.

Um dos principais problemas no desenvolvimento de arquiteturas dedicadas para a EM é a adaptação dos algoritmos. Algoritmos regulares, como o FS, por exemplo, são facilmente desenvolvidos em hardware, pois o paralelismo pode ser explorado de maneira eficiente, atingindo elevadas taxas de processamento. No entanto, o custo em área (número de transistores) deste paralelismo, necessário para garantir as taxas de processamento exigidas por vídeos de alta definição em tempo real, é impraticável. Além disto, outras questões como o custo em memória (tamanho e tempo de acesso) e potência também são limitadores para o desenvolvimento arquitetural. A maior parte das soluções propostas para o algoritmo FS considera alguma técnica de redução de complexidade, como redução do tamanho da área de busca, subamostragem de pixels, antecipação de parada, entre outras.

Em geral, os algoritmos rápidos possuem características que dificultam a sua implementação em hardware, como irregularidade no acesso a memória, dependência de dados e número variável de ciclos para a geração dos vetores. No entanto, alguns

algoritmos rápidos vêm sendo desenvolvidos levando em consideração a sua implementação em hardware. Esta é uma tendência atual, que visa abordar o problema do aumento significativo da complexidade na compressão de vídeos de alta definição. A crescente demanda por aplicações de vídeo de alta definição, principalmente em dispositivos móveis, como celulares, exige o desenvolvimento de arquiteturas de alto desempenho, porém, de baixo custo tanto em área quanto em potência. Algoritmos rápidos voltados para o desenvolvimento arquitetural são a opção mais eficiente para este propósito.

Outra característica presente em diversas soluções arquiteturais recentes é a integração das ferramentas para o aumento de qualidade da EM presentes nos padrões de compressão atuais. Muitas arquiteturas consideram a estimação de movimento com diversos tamanhos de bloco, múltiplos quadros de referência e precisão fracionária de pixel, por exemplo. Estas ferramentas agregam qualidade aos vetores de movimento mas, no entanto, limitam o uso destas arquiteturas a determinado padrão de compressão, que dão suporte ao uso destas ferramentas.

O uso de técnicas de aceleração dos cálculos também vem sendo muito explorada atualmente. Técnicas como subamostragem de pixel e antecipação de parada, estão presentes em diversos artigos recentes. O uso de subamostragem de pixel é uma técnica antiga, e quando usada em baixos níveis (2:1, por exemplo), podem reduzir significativamente o número de comparações, com pequenas perdas em qualidade (PORTO, 2008). No entanto, com o aumento da resolução dos vídeos, a subamostragem em níveis maiores pode ser considerada, devido a grande quantidade de pixels utilizados para representar uma imagem.

O uso de um limiar para a escolha do bloco candidato, necessário para o processo de antecipação de parada, pode acelerar consideravelmente a busca mas, no entanto, implica em uma perda de qualidade. Um bom limiar depende de um bom valor médio de resíduo. Para isto, uma bateria de simulações, com diversas sequências de vídeo deve ser realizada. Assim, um valor médio de resíduo por bloco pode ser obtido e, então, um valor médio de limiar pode ser determinado. No entanto, o uso da antecipação de parada aumenta a incidência da escolha de mínimos locais, pois, em muitos casos, fará com que o algoritmo interrompa a busca antes de encontrar o bloco candidato de resíduo ótimo.

O próximo capítulo desta tese apresenta um estudo sobre a EM em vídeos de alta definição, bem como as principais diferenças associadas à EM na codificação de vídeo de baixa e alta definição.

4 A ESTIMAÇÃO DE MOVIMENTO EM VÍDEOS DE ALTA DEFINIÇÃO

Atualmente, a demanda por vídeo de alta resolução tem crescido drasticamente. Até mesmo dispositivos móveis, como celulares e câmeras digitais, por exemplo, podem trabalhar com vídeos de alta definição, também conhecidos como HD (*High Definition*). O aumento da definição dos vídeos digitais, na prática, é obtido com o aumento do número de pixels que compõem a imagem, gerando maior resolução. Um quadro de um vídeo HD 1080p, com resolução de 1920x1080 pixels, tem cerca de 82 vezes mais pixels que um quadro QCIF com resolução de 176x144 pixels, por exemplo. Mesmo com esta enorme diferença no número de pixels, que confere ao quadro HD 1080p uma definição de imagem muito superior, os dois quadros devem representar a mesma imagem. Com isto, a importância do valor de um pixel na composição da imagem diminui com o aumento da resolução do vídeo.

A Figura 4.1 apresenta parte de um quadro de uma sequência de teste HD 1080p, com uma região em destaque (bloco tracejado). Cerca de dez mil pixels são utilizados para representar apenas a região em destaque na imagem. Esta região mostra parte do para-lama do trator, a parte escura, entre o para-lama e a roda, bem como parte do pneu. Esta mesma sequência de teste foi re-escalada para diferentes resoluções, com menor definição. Todas as resoluções inferiores a 1920x1080 pixels mantiveram a mesma proporção (*aspect ratio*) do vídeo original, que é de 16:9. As resoluções utilizadas foram:

- 256x144 (144p) – Equivalente à resolução QCIF, mas em formato 16:9
- 480x272 (272p) – UMD – *Universal Media Disc*
- 854x480 (480p) – EDTV – *Enhanced-Definition Television*
- 1280x720 (720p) – HD 720p – *Half High Definition Television*
- 1920x1080 (1080p) – HD 1080p - *High Definition Television*

Um bloco de 16x16 pixels será analisado em cada uma das resoluções. A posição do primeiro pixel deste bloco (pixel superior esquerdo) é fixa para todas as resoluções. A Figura 4.2 (a) apresenta um bloco de 16x16 pixels da imagem da Figura 4.1, na resolução de 256x144 pixels. Este bloco, apesar de possuir uma quantidade muito inferior de pixels (256 apenas) está representando a mesma região em destaque no quadro da Figura 4.1, que apresenta o quadro em alta definição. Para a resolução UMD (Figura 4.2 (b)) o mesmo bloco de 16x16 pixels representa apenas parte de para-lama do trator e da região escura entre o para-lama e o pneu. Para as demais resoluções, o mesmo bloco de 16x16 pixels representa apenas uma região do para-lama do trator. Analisando a Figura 4.2 é possível perceber claramente que poucos pixels podem

representar grandes regiões do quadro, em vídeos de baixa resolução. Vídeos de alta definição utilizam uma quantidade muito grande de pixels para inserir detalhamento na imagem, no entanto, grande parte dos pixels possui valores semelhantes, como pode ser observado na Figura 4.2 (d) e (e).

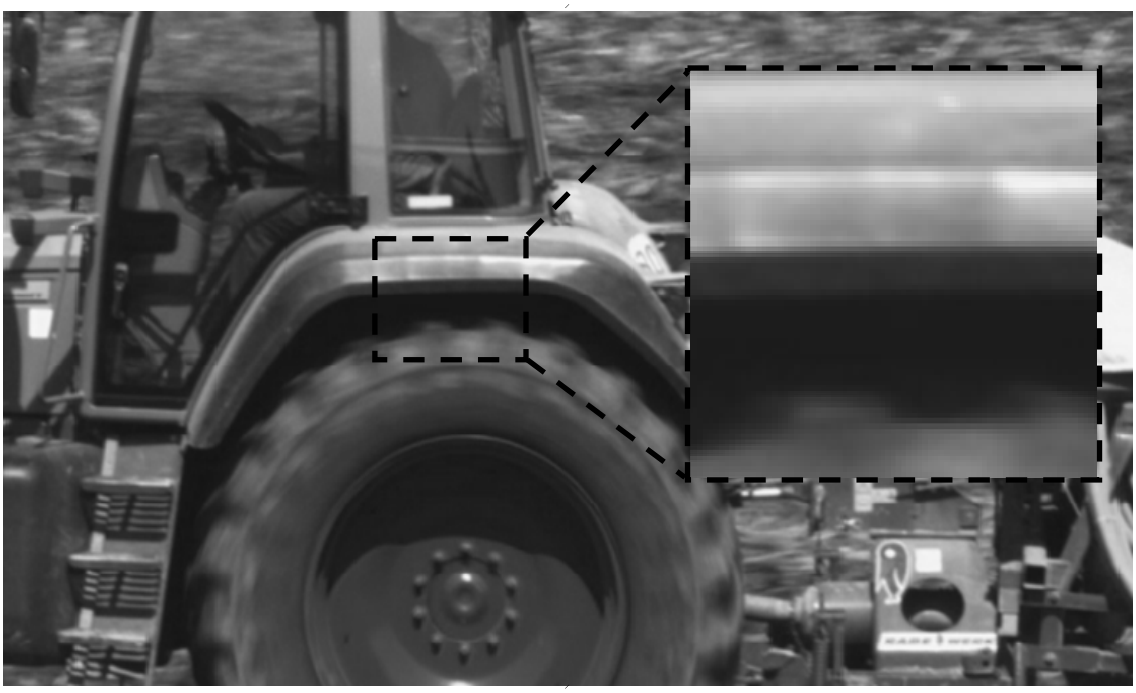
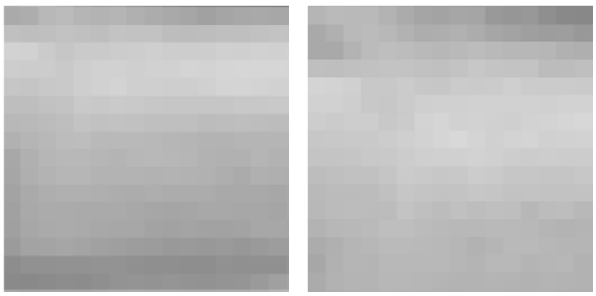


Figura 4.1: Pequena região em destaque em parte de uma imagem HD 1080p

(a) 256x144 (b) 480x272 (c) 854x480



(d) 1280x720 (e) 1920x1080

Figura 4.2: Bloco 16x16 pixels da mesma região da imagem em diferentes resoluções

O aumento da resolução do vídeo pode influenciar diretamente no resultado da EM. Vídeos de baixa resolução possuem uma quantidade menor de pixels para representar a mesma informação visual que vídeos de alta definição. Isto faz com que os algoritmos de EM tenham comportamentos diferentes quando aplicados ao mesmo vídeo em

diferentes resoluções. Para demonstrar esta característica, os algoritmos DS e FS foram aplicados a dez sequências de teste HD 1080p (XIPH.ORG, 2010), redimensionadas nas diferentes resoluções apresentadas anteriormente. A Figura 4.3 apresenta o primeiro quadro de cada uma das sequências de teste utilizadas. O algoritmo DS foi escolhido por ser um dos algoritmos rápidos mais conhecidos e utilizados na literatura e também por ser um dos algoritmos rápidos mais eficientes.

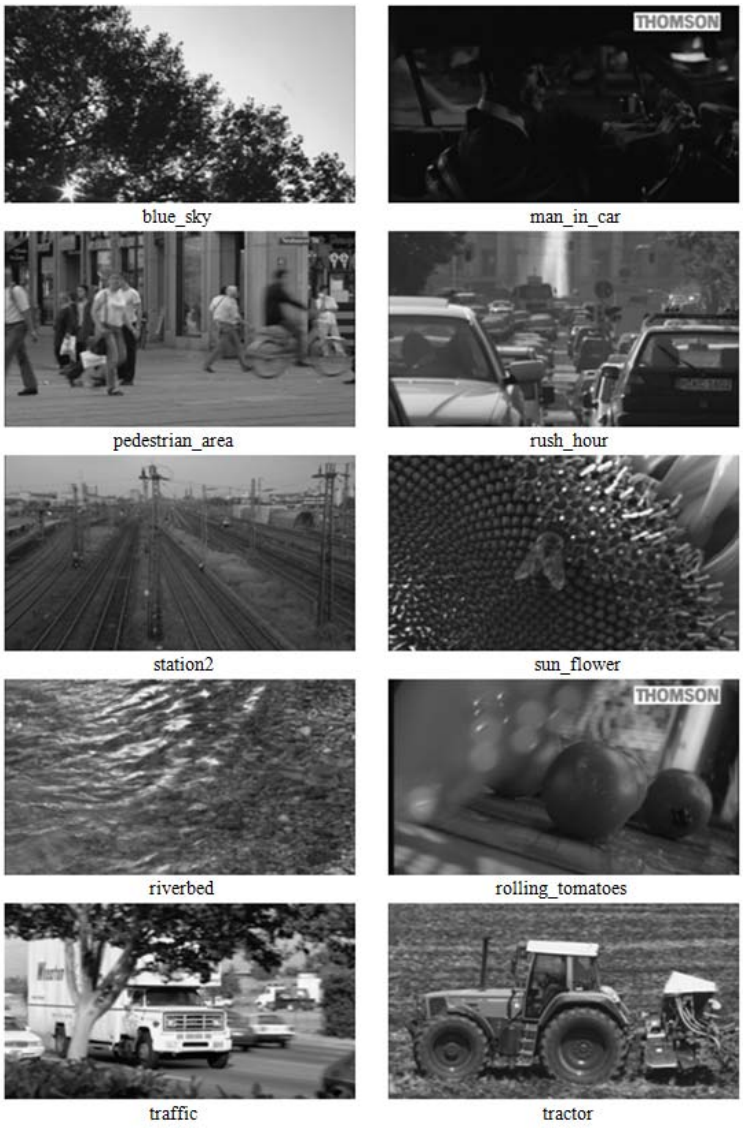


Figura 4.3: Primeiro quadro das sequências de teste HD 1080p utilizadas

A Figura 4.4 apresenta um gráfico com as diferenças de PSNR médio (para as 10 sequências de teste) para os algoritmos FS e DS, nas várias resoluções. A área de busca utilizada foi diferente para cada resolução do vídeo, sempre considerando a distância em pixels (*range*) a partir do bloco atual, localizado no centro. Para a resolução de 256x144 pixels, a área de busca utilizada foi de [-16, +16] e para a resolução UMD a área utilizada foi de [-24, + 24]. Na resolução EDTV, a área de busca utilizada foi de [-32, +32] pixels, para a resolução HD 720p a área de busca utilizada foi de [-64, +64] pixels e para a resolução 1080p, a área de busca utilizada foi de [-128, + 128].

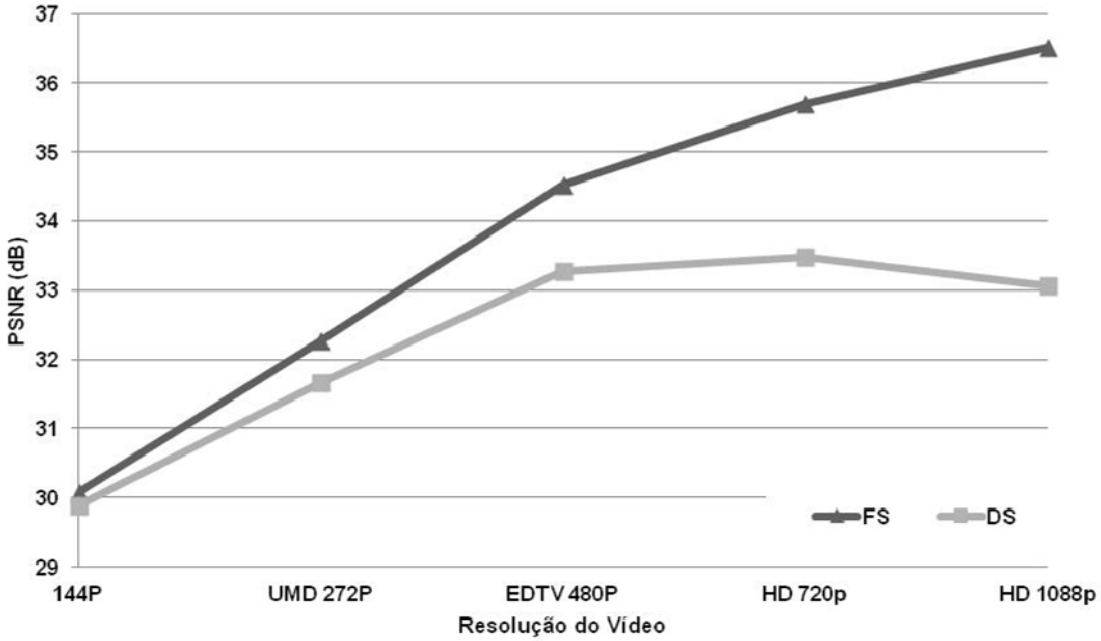


Figura 4.4: PSNR médio para o algoritmo FS e DS em diferentes resoluções

Analisando as curvas apresentadas na Figura 4.4 é possível perceber que, com o aumento da resolução do vídeo, a diferença de PSNR entre o algoritmo FS e DS aumenta consideravelmente. O algoritmo FS incrementa constantemente o PSNR com o aumento da resolução. Isto ocorre, pois a área de busca também aumenta, desta forma, o FS pode pesquisar uma região maior do quadro de referência, o que aumenta a possibilidade de encontrar blocos com menor resíduo. Para os algoritmos rápidos, como DS, o aumento da área de busca não proporciona um aumento considerável no PSNR (PORTO, 2007). Além disto, fica claro que a eficiência do algoritmo DS diminui de acordo com o aumento da resolução do vídeo. Isto ocorre devido à heurística utilizada para a aceleração da convergência, que se torna menos eficiente quando o algoritmo compara blocos que são muito semelhantes entre si. Desta forma o algoritmo tende a satisfazer a sua condição de parada muito cedo, aumentando a ocorrência da escolha de mínimos locais.

A Tabela 4.1 apresenta os resultados médios de PSNR e complexidade computacional, medida em número de blocos candidatos calculados (BCC), para os algoritmos FS e DS em diversas resoluções. Em vídeos de baixa resolução, como o 144p, a diferença de PSNR é inferior a 0,2dB. Para a resolução EDTV, esta diferença é superior a 1,3dB e para as resoluções HD 720p e HD 1080p, a diferença sobe para cerca de 2,2dB e 3,4dB, respectivamente.

Estes resultados demonstram que o algoritmo DS é muito eficiente em termos de qualidade para baixas resoluções, pois obtém um resultado de qualidade praticamente igual ao algoritmo FS, com uma redução de complexidade sempre superior a 72 vezes. Com o aumento da resolução, as perdas em qualidade começam a ser significativas e a grande relevância do algoritmo passa a ser apenas na redução da complexidade.

Isto demonstra que os resultados de qualidade dos algoritmos rápidos, obtidos para vídeos de baixa definição, não podem ser extrapolados para vídeos de alta definição. Com isso, os resultados apresentados por diversos algoritmos de EM publicados na literatura perdem relevância no cenário da compressão de vídeo de alta definição. O crescimento do número de mínimos locais, gerado pelo aumento da resolução dos

vídeos, deve ser considerado na geração dos resultados de qualidade dos algoritmos de EM.

Tabela 4.1: Resultados médios de PSNR e BCCs para os algoritmos FS e DS em diferentes resoluções

Resoluções	PSNR (dB)			BCCs (x10^9)		
	FS	DS	Diferença (dB)	FS	DS	Aumento (vezes)
256x144 (144p)	30,07	29,88	0,19	0,03	0,0004	72,5
480x272 (UMD)	32,27	31,67	0,6	0,23	0,002	117,5
864X480 (EDTV)	34,52	33,18	1,34	1,33	0,007	190
1280X720 (HDTV)	35,70	33,45	2,25	11,8	0,018	655,5
1920X1080 (HDTV)	36,50	33,05	3,45	106,95	0,048	2228,1

4.1 Análise dos Mínimos Locais

A perda de eficiência dos algoritmos rápidos em relação ao algoritmo FS com o aumento da resolução está diretamente ligada ao aumento das ocorrências de escolha de mínimos locais. Com o aumento da área de busca, muitas vezes o bloco candidato ótimo pode estar relativamente longe do centro da área. Além disso, vídeos de alta resolução possuem grandes regiões homogeneas, aumentando a quantidade de mínimos locais. A Figura 4.5 apresenta um gráfico em três dimensões (3D) com os valores de SAD (mapa de SAD) para uma área de busca de 128x128 pixels na resolução HD 1080p da sequência sun_flowers. Este gráfico apresenta a mapa de SADs para blocos de 16x16 separados por faixas de valor, representadas por diferentes cores, onde cada cor compreende uma faixa de valores de SAD. É possível perceber a ocorrência de picos (valores altos de resíduo) e vales (valores baixos de resíduo) em toda a área de busca. Algoritmos rápidos (iterativos) como o DS, por exemplo, são incapazes de transpor estes picos, pois ao encontrarem blocos candidatos com valores de resíduo maiores que os encontrados anteriormente, o critério de parada do algoritmo é satisfeito e a busca acaba. Algoritmos rápidos com número fixo de comparações, como o TSS, por exemplo, podem até transpor um pico dentro da área de busca, no entanto, como possuem um número fixo de iterações, acabam não refinando de maneira eficiente os melhores blocos ao redor da sua escolha. A visualização em 3D é interessante para demonstrar a existência dos picos e vales presentes em uma área de busca, no entanto, esta visualização dificulta a percepção das regiões homogêneas.

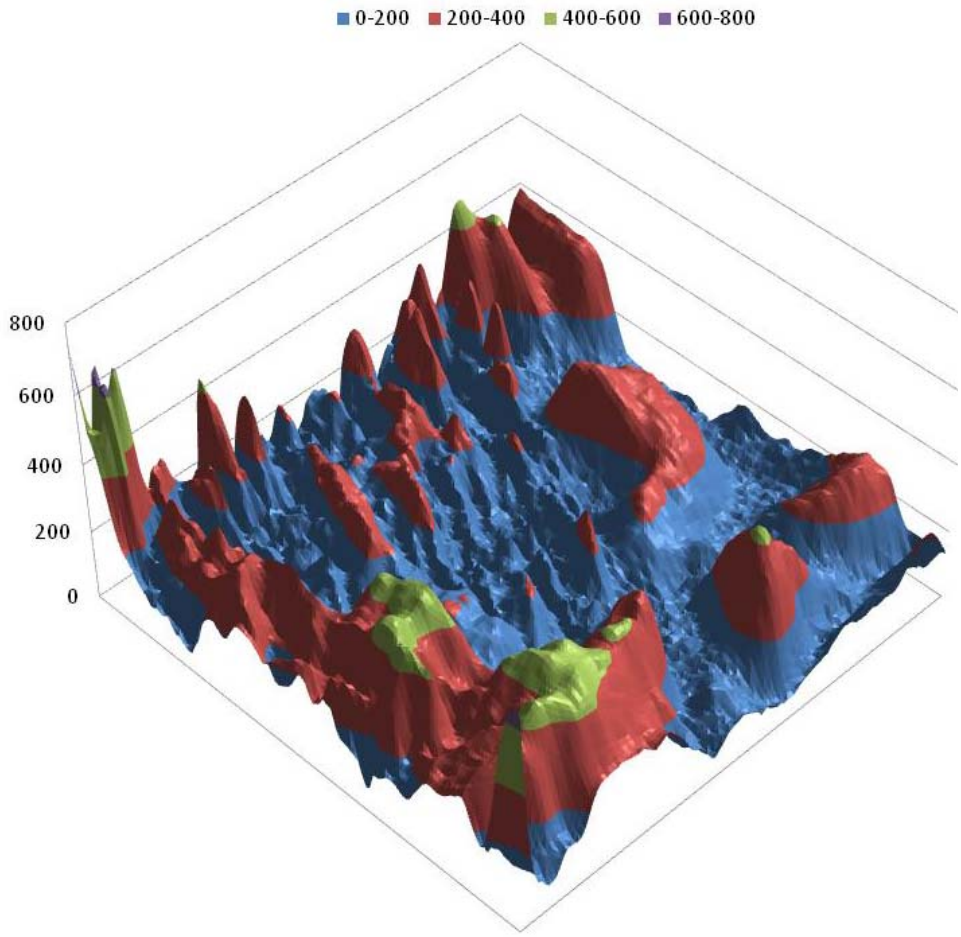
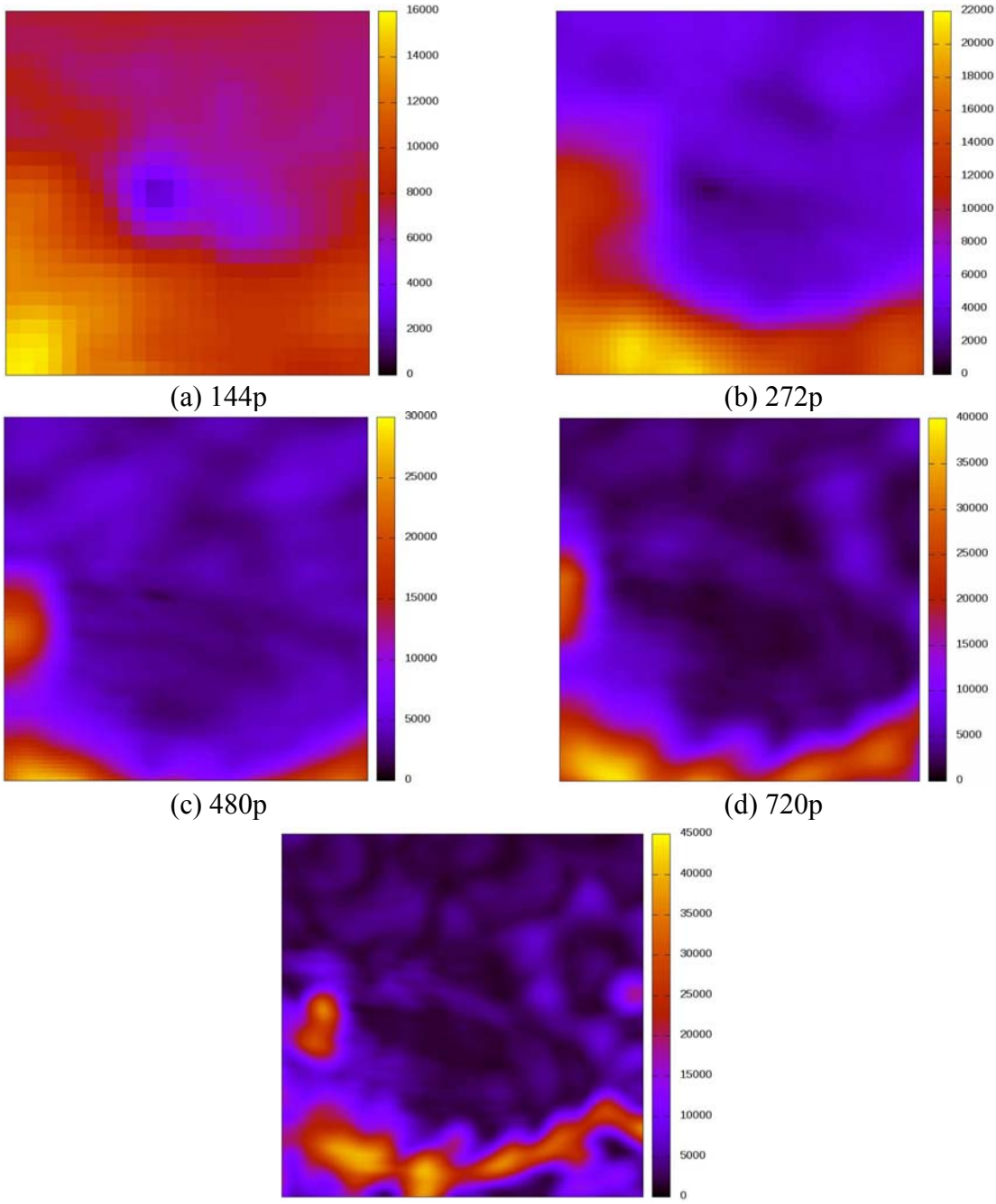


Figura 4.5: Resultados de SAD para uma área de busca de 128x128 pixels e blocos de 16x16 pixels (visualização 3D)

Os resultados apresentados na Figura 4.4 podem ser explicados com a análise dos mapas de SAD em cada resolução. A Figura 4.6 apresenta o mapa de SADs para áreas de pesquisa que representam a mesma região do quadro para o vídeo sun_flower nas cinco diferentes resoluções. A Figura 4.6(a), (b), (c), (d) e (e) apresentam os mapas de SAD para as resoluções 144p, 272p, 480p, 720p e 1080p, respectivamente. As imagens representam os valores de SAD em uma espécie de mapas de calor, onde baixos valores de resíduo são representados com tons de azul escuro, e valores altos são representados por tons de laranja claro.

Analisando a Figura 4.6(a) é possível perceber que no vídeo de resolução 144p o mapa de SAD apresenta bons resultados apenas próximo ao centro da área de busca. No exemplo específico desta imagem, existem apenas quatro blocos que visualmente podem conter os menores resultados de SAD. Na Figura 4.6(b), é possível perceber que novas regiões de baixos valores de SAD podem ser encontradas além da região central da área de busca. Esta é uma evidência do aumento do número de mínimos locais gerados pelo aumento da resolução do vídeo. Apesar do surgimento de novas regiões de mínimos locais, até a resolução 480p (Figura 4.6(c)) ainda é possível distinguir visualmente a região que apresenta o mínimo global. Esta região, com coloração mais próxima de azul escuro, está situada na mesma região da área de busca em todas as resoluções. No entanto, para as resoluções 720p e 1080p, o número de mínimos locais cresce de tal forma que é impossível determinar, visualmente, qual a região que apresenta o mínimo global.



(a) 144p (b) 272p

(c) 480p (d) 720p

(e) 1080p

Figura 4.6: Mapas de SAD para o vídeo sun_flower em diferentes resoluções

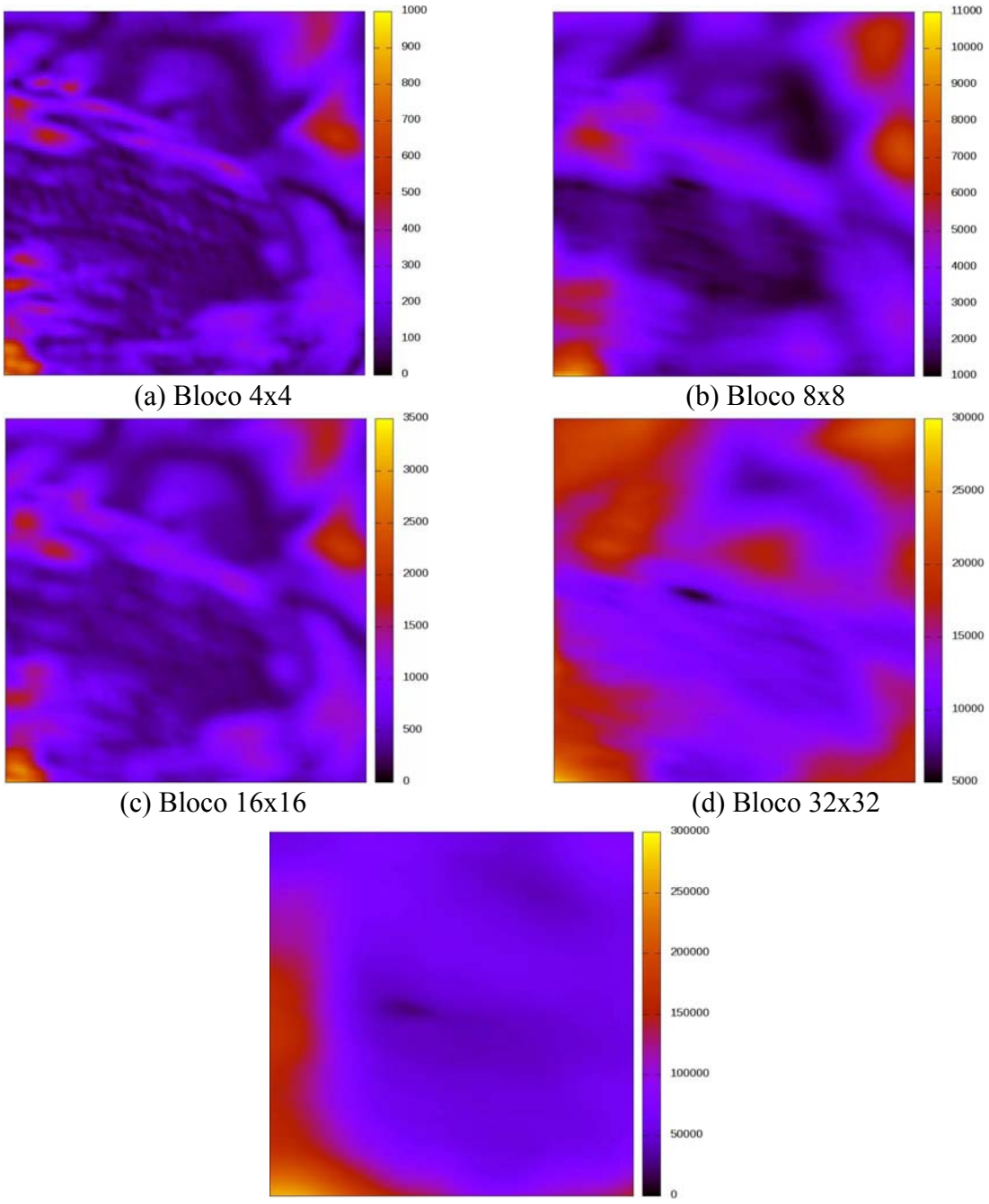
A análise das imagens apresentadas na Figura 4.6 explica visualmente os resultados de qualidade apresentados na Figura 4.4. Os algoritmos DS e FS escolhem blocos candidatos com SADs muito parecidos (ou até os mesmos) nos vídeos de baixa resolução. Como ilustrado na Figura 4.6(a), existem poucos blocos candidatos com valores baixos de SAD e mesmo que o algoritmo DS não escolha o bloco ótimo, ele escolherá um bloco subótimo com valor de SAD muito próximo ao valor ótimo. Para os vídeos de alta definição, como as resoluções 720p e 1080p, o número excessivo de mínimos locais prejudica consideravelmente o funcionamento do algoritmo DS. A heurística utilizada para a aceleração do processo de busca faz com que o algoritmo escolha mínimos locais quando pequenos picos de SAD são encontrados. Já o algoritmo FS, que explora todos os blocos candidatos da área de busca, sempre encontrará o bloco

candidato ótimo, não sendo afetado pelo aumento dos mínimos locais. Isto demonstra que estratégias para evitar a escolha de mínimos locais são extremamente importantes no desenvolvimento de algoritmos rápidos para a EM focados em vídeos de alta definição.

Outra constatação importante desta análise é que, na medida em que a definição do vídeo aumenta, o processo de escolha do melhor casamento da EM deixar de ser uma estimativa do movimento do bloco atual na área de busca e passa a ser apenas um casamento de padrões, onde o movimento dos objetos deixa de ser o ponto mais importante. Na Figura 4.6 (e) os vários blocos de azul escuro são os mínimos locais e algum deles é o mínimo global. Neste cenário, fica claro que os blocos de azul escuro não representam o movimento do bloco atual na área de referência, pois o bloco que está sendo codificado não poderia mover-se para tantos lugares ao mesmo tempo. Esse fato também é importante para explicar o porquê das heurísticas de algoritmos rápidos perderem eficiência com o aumento da definição dos vídeos. Por exemplo: o uso do vetor do bloco anterior para definir o início da busca pelo próximo vetor passa a ser uma heurística sem sentido. A priorização pela exploração de blocos colocalizados e seus vizinhos, também perde a sua eficiência, pois o bloco central pode não gerar o casamento ótimo, que pode ser encontrado em um bloco da área de busca que, inclusive, pertença a outro objeto da cena, distante do objeto ao qual faz parte o bloco atual.

O tamanho do bloco utilizado na busca da EM também influencia diretamente no resultado do mapa de SADs. O uso de blocos pequenos tende a favorecer a escolha de bons blocos candidatos, já que as diferenças entre os blocos tendem ser menores. A Figura 4.7 apresenta os mapas de SAD gerados em uma área de 128x128 amostras para o vídeo sun_flower na resolução 1080p. As Figuras 4.7(a), (b), (c), (d) e (e) apresentam os mapas de SAD para os blocos 4x4, 8x8, 16x16, 32x32 e 64x64, respectivamente.

Analisando os mapas de SAD apresentados na Figura 4.7 é possível perceber que a ocorrência de mínimos locais pode variar drasticamente para uma mesma área de busca de acordo com o tamanho do bloco utilizado. Para blocos de tamanho 4x4, existem muitos mínimos locais, no entanto, com o aumento do tamanho do bloco, os valores de resíduos crescem e as regiões de azul mais escuro nas figuras passam a ser mais escassas. Isto implica na redução nos mínimos locais presentes nesta área, pois poucas regiões da área de busca apresentam baixos valores de SAD. Esta característica faz com que os algoritmos rápidos sejam menos sensíveis a utilização de tamanhos de blocos maiores, enquanto o algoritmo FS pode apresentar perdas mais significativas de qualidade em relação aos resultados obtidos para blocos menores.



(a) Bloco 4x4 (b) Bloco 8x8

(c) Bloco 16x16 (d) Bloco 32x32

(e) Bloco 64x64

Figura 4.7: Mapas de SAD para o vídeo sun_flower com diferentes tamanhos de bloco

4.2 Análise do Uso de Vários Tamanhos de Bloco no H.264/AVC

O aumento da resolução do vídeo tende a gerar grandes regiões homogêneas nos quadros. Neste cenário, o uso de grandes blocos para a estimação de movimento passa a ser mais relevante. Padrões atuais, como o H.264/AVC, permitem o uso de diferentes tamanhos de bloco na EM, sendo que o uso de cada tamanho de bloco é controlado pela relação taxa *versus* distorção. Regiões homogêneas são codificadas por blocos maiores, e regiões com maior nível de detalhamento, ou movimentação, são codificadas utilizando blocos menores. No entanto, nenhum padrão de compressão de vídeo atual

permite a utilização de blocos maiores que 16x16. A utilização de grandes blocos (até 64x64) deve ser incluída no novo padrão de compressão HEVC, também conhecido como H.265, atualmente em desenvolvimento (JCT-VT, 2011).

Para avaliar a relevância de cada tamanho de bloco na estimação de movimento de vídeos de alta definição, um experimento utilizando o código de referência do padrão H.264/AVC (SUHRING, 2010) foi realizado. O código de referência foi configurado para utilizar apenas um quadro inicial do tipo I e todos os demais do tipo P (SUHRING, 2010) e foi aplicado aos 100 primeiros quadros de nove sequências de teste HD 1080p. O padrão H.264/AVC pode trabalhar com sete tamanhos diferentes de bloco e a informação do tamanho escolhido pelo codificador foi armazenada e os resultados médios foram obtidos. A Tabela 4.2 apresenta os resultados percentuais para cada uma das sequências de vídeo utilizadas, assim como o resultado médio. Os resultados estão desconsiderando a ocorrência de blocos do tipo SKIP. Os blocos do tipo SKIP são de grande relevância, mas não são processados pelas predições inter-quadros e intra-quadro, por isso fogem do escopo deste trabalho. Blocos SKIP ocorrem quando o codificador considera que o bloco é muito similar ao original e, então, nenhuma informação é enviada para representar o bloco (RICHARDSON, 2003). Neste experimento, blocos SKIP foram escolhidos 48,19% das vezes, na média.

Analisando os resultados da Tabela 4.2 é possível perceber que, em média, aproximadamente 60% dos vetores são referentes a blocos de tamanho 16x16 (maior tamanho permitido pelo padrão H.264/AVC). O percentual de blocos de tamanho 16x16 é menor em vídeos com maior variação de informação visual, como no caso da sequência “Riverbed”. Esta sequência apresenta o leito de um rio, coberto por pedras, e uma rasa faixa de água passando sobre as pedras, cobrindo todo o quadro e gerando variações de movimento em toda a cena. Sequências com menor movimentação, como “Station2”, por exemplo, possuem os maiores percentuais de blocos 16x16.

Tabela 4.2: Percentuais de escolhas de cada tamanho de bloco do padrão H.264/AVC

Vídeo	**16x16**	**16x8**	**8x16**	**8x8**	**8x4**	**4x8**	**4x4**
Station2	78,48	9,36	10,56	1,08	0,17	0,33	0,02
Sun Flower	68,34	11,16	14,58	4,18	0,75	0,91	0,07
Tractor	46,50	21,22	16,41	10,42	3,19	2,02	0,24
Traffic	55,24	18,38	15,81	7,07	1,80	1,53	0,16
Man in Car	74,80	12,03	11,66	1,14	0,16	0,21	0,01
Pedestrian	57,13	16,33	17,80	6,07	1,26	1,34	0,07
Riverbed	25,36	36,13	14,17	15,94	5,63	2,52	0,26
Rolling	74,89	12,11	11,13	1,40	0,24	0,22	0,01
Rush Hour	57,74	17,46	17,70	5,09	0,98	0,97	0,07
Média	**59,83**	**17,13**	**14,42**	**5,82**	**1,57**	**1,12**	**0,10**

Os tamanhos de bloco 16x8 e 8x16 possuem percentuais similares para todas as sequências, com resultados médios de uso de cerca de 17% e 14%, respectivamente. Os blocos de tamanho 8x8 apresentam um baixo percentual médio, inferior a 6%. A maior incidência de blocos 8x8 ocorre em vídeos com alta movimentação, ou maior variação

de textura, como o a sequência “Tractor”. Neste caso, mais de 10% dos blocos escolhidos foram deste tamanho.

As sub-partições do bloco 8x8, como 8x4, 4x8 e 4x4 apresentam percentuais muito baixos. O percentual de uso de blocos de tamanho 8x4 e 4x8 somados é inferior a 3%. O bloco de tamanho 4x4 é raramente utilizado em sequências de vídeo HD 1080p, com 0,1% de uso em média. Estes dados mostram que a ocorrência média de blocos iguais ou maiores que 8x8 é de mais de 97% (sem considerar blocos SKIP). Isto evidencia a maior relevância de blocos grandes em vídeo de alta definição.

O percentual de blocos do tipo SKIP é diretamente proporcional às características do vídeo. Quanto maior a movimentação e a quantidade de texturas da sequência, menor será o percentual de blocos SKIP. Em sequências com poucas texturas e baixa movimentação, o percentual de blocos do tipo SKIP pode chegar a quase 75% (sequência “Man in car”). Um grande índice de blocos SKIP também acarreta em um grande índice de blocos grandes, principalmente 16x16. A Figura 4.8 apresenta um gráfico ilustrando visualmente os resultados apresentados na Tabela 4.2, desconsiderando os blocos SKIP.

Estes resultados são interessantes, pois exploram as características de utilização de tamanhos de blocos em vídeos de alta definição utilizando um codificador de vídeo estado da arte. Os resultados demonstram que o uso de blocos pequenos é praticamente irrelevante e que o maior percentual de uso é obtido pelo maior bloco permitido pelo padrão H.264/AVC, ou seja, 16x16. Isto demonstra que a utilização de blocos maiores que 16x16 deve ser avaliada na estimação de movimento de vídeos de alta definição e é por isso que o padrão emergente HEVC está considerando a inserção de blocos com tamanho de até 64x64 amostras.

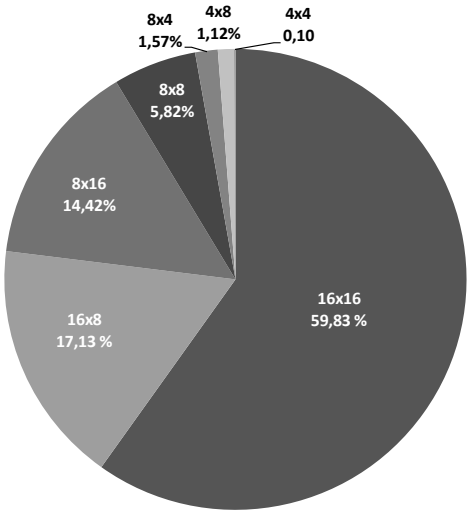


Figura 4.8: Resultados médios de tamanho de bloco para vídeos HD 1080p desconsiderando os blocos SKIP

4.3 Avaliação da EM com Tamanhos de Bloco Maiores

Para avaliar a utilização de grandes blocos na estimação de movimento, um novo experimento foi realizado, considerando apenas a EM de maneira isolada. Neste experimento, a EM foi avaliada para quatro tamanhos de bloco 8x8, 16x16, 32x32 e 64x64. Os tamanhos de bloco menores que 8x8 (permitidos no H.264/AVC) não foram considerados neste experimento, devido aos percentuais de uso inexpressivos para

vídeos HD apresentados na Tabela 4.2. Vale salientar que nenhum padrão de compressão atual permite a utilização de blocos maiores que 16x16. Os algoritmos FS e DS foram aplicados a todas as 10 sequências de teste HD 1080p apresentadas na Figura 4.3. Os algoritmos FS e DS foram escolhidos, pois o FS é utilizado como parâmetro de comparação de qualidade e complexidade e o algoritmo DS é um dos algoritmos rápidos mais conhecidos e utilizados. Os dois algoritmos foram aplicados aos 200 primeiros quadros das 10 sequências de teste, para uma área de busca de [-64, + 64]. A Tabela 4.3 apresenta os resultados médios de PSNR, percentual de redução do resíduo (PRR), número de blocos candidatos calculados (BCC), número de comparações realizadas (Comp.) e número de vetores por quadro. O PRR representa a redução do resíduo proporcionada pela EM em relação à codificação diferencial. A codificação diferencial calcula a diferença amostra a amostra entre os quadros vizinhos, logo, quanto maior o valor do PRR, mais eficiente é o algoritmo. O número de comparações representa a quantidade de comparações entre pixels, utilizada por cada algoritmo para o cálculo do SAD.

Analisando os resultados da Tabela 4.3 é possível perceber que os resultados de qualidade (PSNR e PRR) são afetados negativamente pelo aumento do tamanho do bloco. Este resultado era esperado, pois um bom casamento é mais difícil para blocos maiores, conforme já havia sido exposto na análise dos mínimos locais apresentado na Figura 4.7. A EM está sendo avaliada de maneira isolada, portanto, não estão sendo consideradas as demais etapas do codificador, como as transformadas, quantização e codificação de entropia, que podem ser beneficiadas por este aumento do tamanho do bloco. Os resultados de PSNR para o algoritmo FS têm uma queda grande na comparação entre os resultados dos blocos 8x8 e 64x64 (mais de 6,6dB). O algoritmo DS também apresenta uma grande diminuição do PSNR, no entanto, em uma escala menor (cerca de 2,1dB). É interessante perceber que para blocos maiores, o algoritmo DS passa a ter resultados mais próximos do FS. Estes resultados já eram esperados, de acordo com a análise dos mínimos locais apresentados na Figura 4.7.

Tabela 4.3: Resultados para a EM com FS e DS para diferentes tamanhos de bloco

Parâmetro	Algoritmo	Bloco 8x8	Bloco 16x16	Bloco 32x32	Bloco 64x64
PSNR (dB)	FS	38,56	36,03	33,94	31,88
	DS	33,34	33,06	32,46	31,22
PRR (%)	FS	79,6	72,59	65,55	57,73
	DS	64,25	61,65	58,71	53,98
Nº de BCCs ($x10^9$)	FS	106,2	26,7	6,7	1,7
	DS	0,172	0,048	0,012	0,003
Nº de Comp. ($x10^9$)	FS	6.845,1	6.845,1	6.845,1	6.845,1
	DS	11,01	12,33	13,28	13,64
Nº de Vetores		32640	8160	2040	510

Os resultados de complexidade computacional são ilustrados em número de blocos candidatos comparados (BCC) e também em número de comparações pixel a pixel. O número de BCCs calculados pelo algoritmo FS diminui proporcionalmente ao aumento do tamanho do bloco. Já os resultados de número de comparações são estáveis, pois o

número pixels em cada bloco cresce na mesma proporção em que o número de blocos processados diminui. O algoritmo DS apresenta um pequeno crescimento no número de comparações com o aumento do tamanho do bloco. O aumento do tamanho do bloco reduz o número de blocos no quadro e, consequentemente, o número de repetições da EM. O uso de blocos maiores também reduz o número de blocos candidatos comparados para o algoritmo DS, praticamente na mesma proporção, quando maior o bloco utilizado, menores serão os valores de BCCs.

No entanto, com o aumento do tamanho do bloco, o algoritmo DS necessita de um número maior de iterações para convergir para o resultado final. Para blocos pequenos, como 8x8, por exemplo, a tendência é que exista uma grande quantidade de mínimos locais, como ilustrado na Figura 4.7(b). Isto faz com que o algoritmo DS rapidamente convirja para o seu resultado final, utilizando um pequeno número de iterações. Para blocos 8x8, a média de iterações do algoritmo DS é de aproximadamente três iterações. Para blocos de tamanho 16x16, esta média sobe para mais de 3,7 iterações, e para blocos de 64x64 a média sobe para 4,4 iterações. Uma iteração do algoritmo DS pode representar uma busca por aresta (3 novos BCCs) ou um busca por vértice (5 novos BCCs). Além disso, para os blocos maiores, cada novo bloco candidato avaliado representa um número muito grande de comparações. Tomando como exemplo os blocos 8x8 e 64x64, o número de comparações necessárias para o cálculo do SAD em cada bloco é de 64 e 4096, respectivamente.

Apesar das perdas em qualidade, a utilização de grandes blocos gera uma enorme redução no número de vetores de movimento. A última linha da Tabela 4.3 apresenta o número de vetores de movimento por quadro para cada um dos tamanhos de bloco avaliados. A redução do número de vetores na comparação entre os blocos 8x8 e 64x64 chega a 64 vezes. No entanto, a comparação mais justa deve ser feita em relação ao bloco de tamanho 16x16. Os resultados da Tabela 4.2 mostram que um codificador estado da arte, como o H.264/AVC, escolhe blocos de tamanho 8x8 em apenas 5,8% dos casos, já os blocos de 16x16 são escolhidos em mais de 50%. O algoritmo FS com blocos 32x32 apresenta, em média, um PSNR cerca de 2,1dB inferior ao obtido com blocos 16x16, com uma redução de quatro vezes no número de vetores. Os resultados mais interessantes são obtidos com o algoritmo DS, onde a perda em PSNR foi de apenas 0,6dB, com a mesma redução no número de vetores gerados. Na comparação com o bloco de tamanho 64x64, o algoritmo DS perde apenas 1,8dB, com uma redução no número de vetores de 16 vezes.

Esta relação entre o valor do resíduo gerado e o número de vetores de movimento necessários para a obtenção deste resíduo pode favorecer o uso de grandes blocos quando o processo completo de compressão é considerado. Os resíduos são tratados pelas etapas posteriores à EM, como a transformada e a quantização. Na etapa da transformada, os resíduos são convertidos do domínio espacial para o domínio das frequências (RICHARDSON, 2003). Esta transformação faz com que os valores de baixa frequência (mais relevantes ao sistema visual humano) tenham valores mais significativos. Desta forma, a etapa de quantização pode eliminar as informações de altas frequências, reduzindo a quantidade de informação para representar o bloco.

As etapas de transformadas e quantização podem se beneficiar da utilização de grandes blocos de pixels, pois as transformadas irão concentrar as informações relevantes de uma grande região em apenas alguns coeficientes e a quantização irá transformar a grande matriz de coeficientes gerada pelas transformadas em uma matriz esparsa. Deste modo, a taxa de compressão será maximizada quando blocos maiores são

utilizados, pois a tendência é que existam mais coeficientes zero ao final do processo de quantização, o que potencializa a codificação de entropia. A redução do número de vetores de movimento pode gerar uma redução significativa no *bitstream* do vídeo codificado. Diferentemente das informações de resíduo, os vetores de movimento não podem ser descartados na etapa de quantização. As informações de vetores de movimento podem ser reduzidas apenas pela predição de vetores e pela codificação de entropia (RICHARDSON, 2003), que são técnicas de compressão sem perdas. Os vetores de movimento podem representar percentuais significativos do tamanho total do *bitstream*. A Figura 4.9 apresenta um gráfico com os resultados de percentual de *bitstream* utilizado exclusivamente para as informações de vetores de movimento no padrão H.264/AVC (ITU-T, 2011). Os resultados consideram diferentes sequências de teste CIF e avaliam o percentual do *bitstream* ocupado pelos vetores de movimento de acordo com a variação do parâmetro QP (*Quantization Parameter*). Quando maior o valor do parâmetro QP, maiores serão os cortes de informação realizados pela quantização.

Observando as curvas apresentadas na Figura 4.9, é possível perceber que o percentual cresce com o aumento do valor do QP. Para um valor de QP igual a 35, por exemplo, em média 35% do *bitstream* são utilizados apenas para as informações de vetores de movimento (ITU-T, 2011). Este percentual pode chegar próximo aos 50% para a sequência "Foreman" com o valor de QP igual a 40. Estes percentuais crescem com o aumento do QP, pois quanto maiores forem os valores de QP, maiores serão os cortes nos resíduos. Com uma quantidade menor de resíduos no *bitstream*, cresce a representatividade das informações de vetores de movimento.

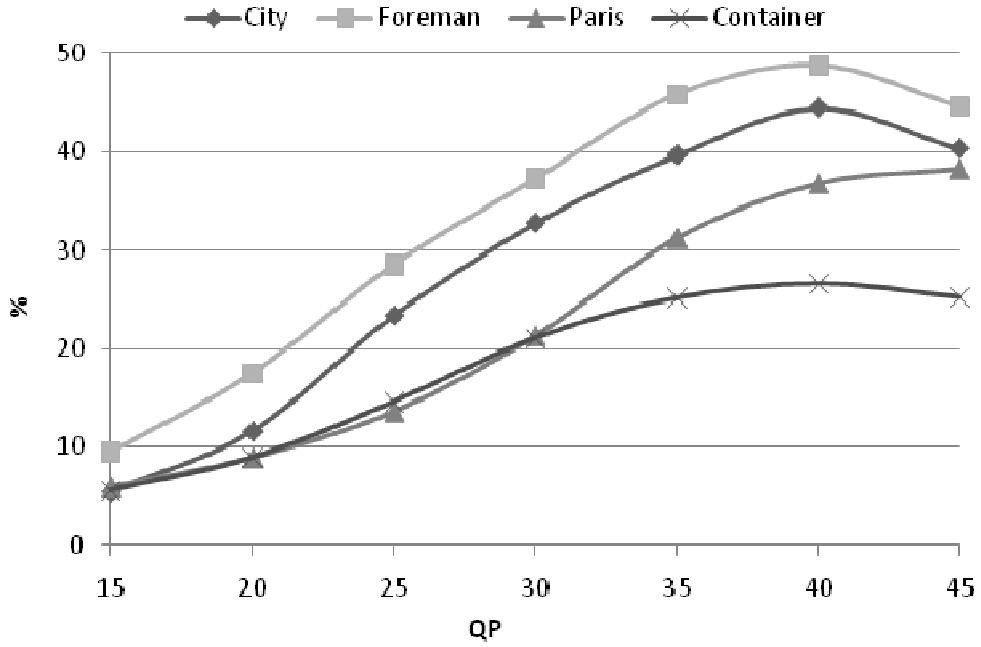


Figura 4.9: Percentual do *bitstream* dedicado aos vetores de movimento no H.264/AVC (ITU-T, 2011)

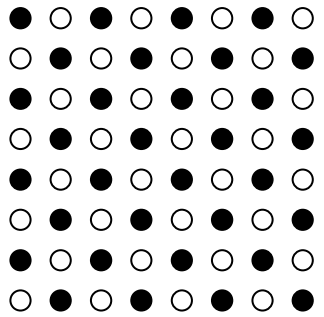
O uso de grandes blocos na EM pode contribuir significativamente para a redução do percentual correspondente a vetores de movimento do *bitstream* do vídeo codificado. A EM com blocos de tamanho 64x64, por exemplo, pode gerar uma redução no número de vetores de 16 (pior caso) a 256 vezes (melhor caso), na comparação com blocos 4x4 e 16x16, respectivamente. Considerando os resultados obtidos com o QP igual a 35, o

uso de blocos de 64x64 pode gerar uma redução no *bitstream* entre 32,9% (pior caso) e 34,9% (melhor caso).

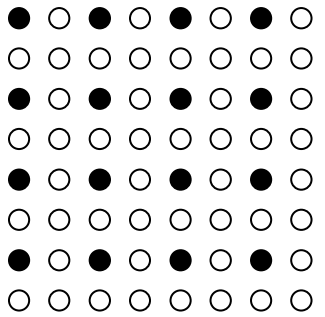
4.4 Avaliação do Uso de Subamostragem de Pixel

A utilização de grandes blocos na EM pode favorecer a utilização de subamostragem de pixel. A subamostragem de pixel é uma técnica largamente utilizada para a aceleração dos cálculos de SAD na EM (KUHN, 1999) e pode ser utilizada em diversos níveis. A Figura 4.10 apresenta quatro exemplos de subamostragem considerando um bloco de 8x8 pixels. O cálculo de SAD utilizará apenas aos pontos pretos do bloco e os pontos brancos serão simplesmente desconsiderados. As Figuras 4.10(a), (b), (c) e (d) apresentam a subamostragem de nível 2:1, 4:1, 8:1 e 16:1, respectivamente. Apenas metade dos pixels do bloco serão comparados para a subamostragem 2:1, um quarto para a subamostragem 4:1, um oitavo para a subamostragem 8:1 e um dezesseis avos para a subamostragem de 16:1.

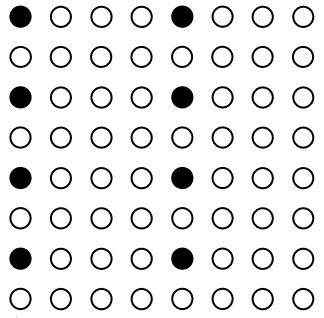
Trabalhos anteriores já mostraram que a utilização da subamostragem 2:1 é altamente recomendada para vídeos SDTV 720x480 (*Standard Definition Television*) com blocos de tamanho 16x16, visto que resulta em perdas pequenas de qualidade, com um ganho de 50% em número de comparações (PORTO, 2007). Com subamostragem de 4:1, as perdas em qualidade tornam-se mais significativas, no entanto, a redução no número de comparações ainda é interessante quando a restrição de desempenho é a prioridade (PORTO, 2008). As subamostragens de 8:1 e 16:1 não costumam ser utilizadas, mesmo para blocos de tamanho 16x16, pois as perdas em qualidade são muito significativas (KUHN, 1999).



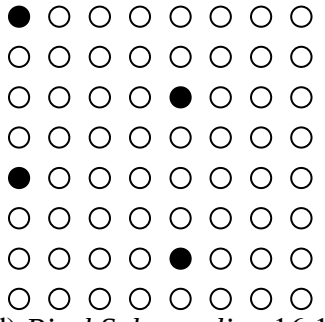
a) *Pixel Subsampling* 2:1



b) *Pixel Subsampling* 4:1



c) *Pixel Subsampling* 8:1



d) *Pixel Subsampling* 16:1

Figura 4.10: Subamostragem de pixel *(*PS - *Pixel Subsampling*) em diferentes níveis

Os resultados de custo computacional apresentados na Tabela 4.3 mostram que o aumento do tamanho do bloco reduz proporcionalmente o número de BCCs. No entanto, o número de comparações calculadas pelo algoritmo FS é estável e, para o

algoritmo DS, este número sobe com o crescimento do tamanho do bloco. O uso de subamostragem de pixel em níveis mais elevados para blocos maiores pode reduzir drasticamente o número de comparações calculadas, com pequenas perdas em qualidade. Isto pode valorizar ainda mais o uso de grandes blocos, visto que, além dos ganhos em redução do número de vetores de movimento, ganhos em redução do custo computacional também podem ser obtidos.

Para avaliar o impacto da utilização de elevados níveis de subamostragem em vídeos de alta definição, uma nova bateria de testes foi realizada. Mais uma vez, os algoritmos escolhidos foram o FS e DS. Os algoritmos foram aplicados ao mesmo conjunto de 10 sequências de teste HD 1080p citado anteriormente. Os algoritmos foram testados para todos os tamanhos de bloco apresentados na Tabela 4.3, com todos os níveis de subamostragem apresentados na Figura 4.10. A Figura 4.11 apresenta as curvas de PSNR para o algoritmo FS, com os diferentes tamanhos de bloco utilizados, mostrando as perdas introduzidas com o aumento da subamostragem. O aumento do nível da subamostragem afeta negativamente os resultados de PSNR, para todos os tamanhos de bloco utilizados. No entanto, os blocos maiores são menos afetados pelo aumento do nível da subamostragem de pixel.

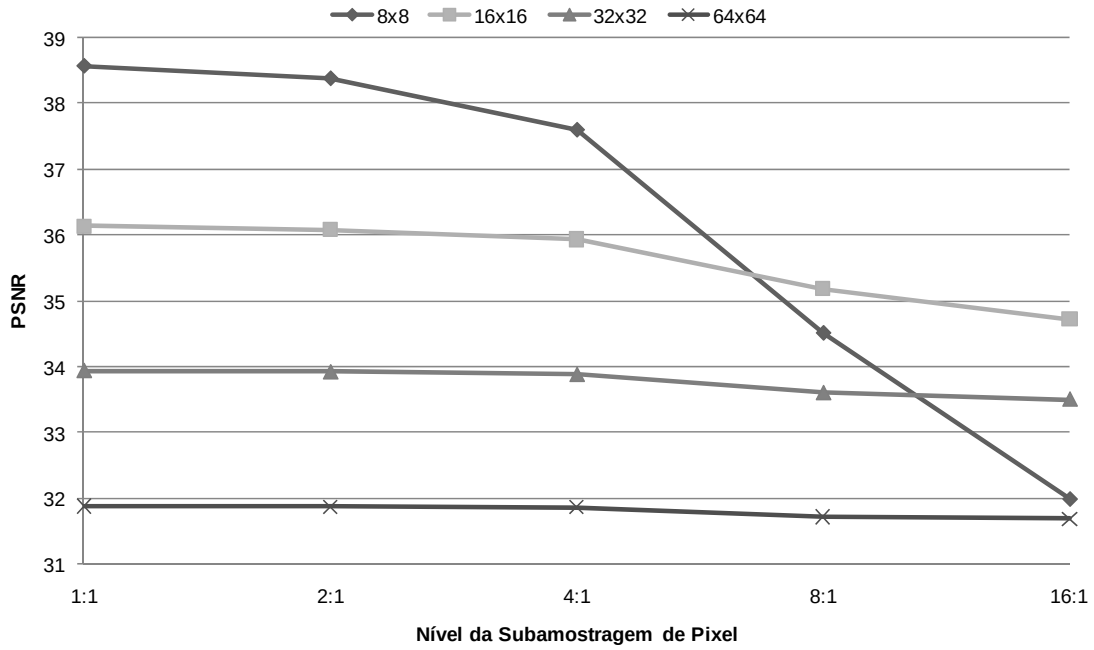


Figura 4.11: Variação do PSNR médio para o algoritmo FS com o aumento da subamostragem de pixel

Os resultados de PSNR para blocos de tamanho 8x8 começam a apresentar perdas significativas a partir da subamostragem de pixel 4:1, sendo que para a subamostragem de nível 8:1 e 16:1 as perdas se acentuam drasticamente. A mesma característica pode ser observada nos resultados para o bloco 16x16, sendo que as perdas significativas começam a partir da subamostragem 4:1. Já os blocos grandes (32x32 e 64x64), apresentam pequenas variações com o aumento da subamostragem de pixel. Até mesmo com a subamostragem de 16:1 as perdas verificadas em PSNR foram moderadas.

A Figura 4.12 apresenta as curvas de PSNR para o algoritmo DS, com os diferentes tamanhos de bloco utilizados. Os resultados do algoritmo DS variam menos, em relação ao algoritmo FS, com a alteração do tamanho do bloco. No entanto, as perdas

introduzidas com o aumento da subamostragem seguem a mesma tendência. Com isso, o PSNR para blocos 8x8 com subamostragem de pixel 4:1 já é inferior ao obtido com blocos 16x6, com o mesmo nível de subamostragem. O mesmo acontece com o resultado obtido para o bloco 32x32, com subamostragem 8:1, que tem PSNR superior ao obtido pelo bloco 16x6, com o mesmo nível de subamostragem.

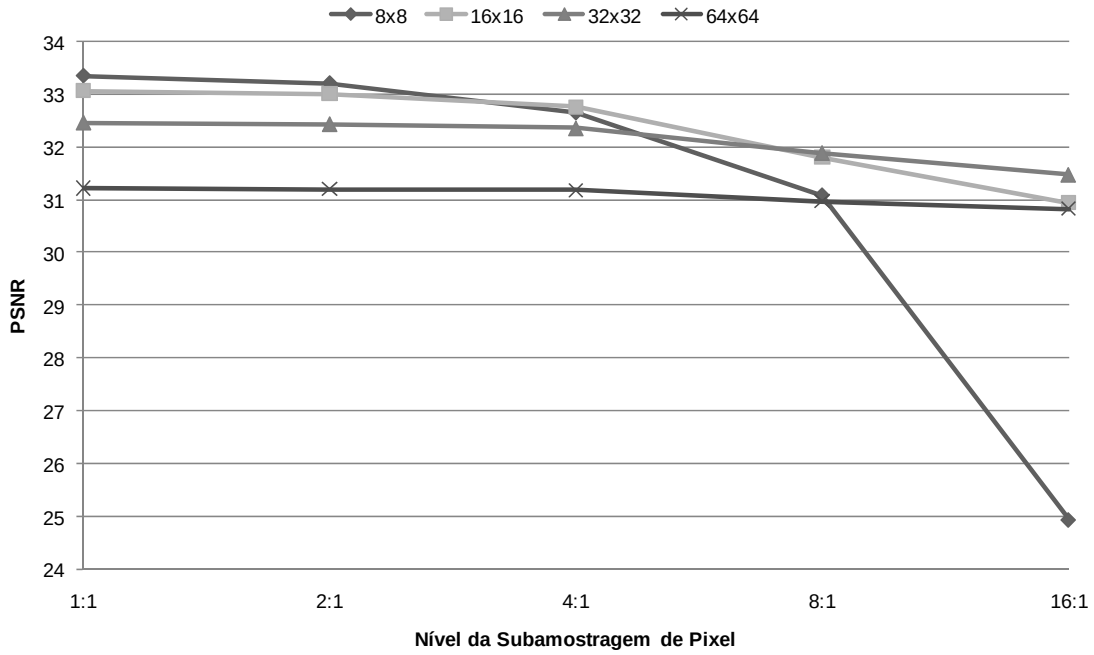


Figura 4.12: Variação do PSNR médio para o algoritmo DS com o aumento da subamostragem de pixel

A grande vantagem do uso da subamostragem de pixel é a redução do número de comparações. Para o algoritmo FS, esta redução é diretamente proporcional ao nível da subamostragem utilizada. Logo, os resultados de número de comparações apresentados na Tabela 4.3 são divididos por dois, quatro, oito e dezesseis com o uso da subamostragem de pixel 2:1, 4:1, 8:1 e 16:1, respectivamente. Esta redução é ainda maior para o algoritmo DS, pois o uso da subamostragem interfere nas escolhas do algoritmo. A Tabela 4.4 apresenta o número médio de iterações do algoritmo DS para todos os tamanhos de bloco utilizados e considerando a variação do nível da subamostragem de pixel. Os resultados mostraram que existe uma pequena redução no número de iterações do algoritmo DS com o aumento no nível de subamostragem. Isto faz com que a redução no número de comparações seja ainda maior do que aquela causada apenas pelo nível de subamostragem utilizada.

Tabela 4.4: Número médio de iterações do algoritmo DS com diferentes níveis de subamostragem

Bloco	Sem Sub	Sub 2:1	Sub 4:1	Sub 8:1	Sub 16:1
Bloco 8x8	3,04	2,96	2,74	2,31	1,97
Bloco 16x16	3,75	3,68	3,48	3,03	2,61
Bloco 32x32	4,23	4,18	4,03	3,65	3,28
Bloco 64x64	4,40	4,30	4,24	3,95	3,67

A diferença nos resultados médios de PSNR entre o algoritmo DS com blocos 64x64 sem subamostragem de pixel e o algoritmo DS com subamostragem de pixel 16:1 é de apenas 1dB, no entanto, a redução no número de comparações é superior a 6434 vezes.

O uso da subamostragem de pixel insere perdas na qualidade, no entanto, pode proporcionar ganhos expressivos na redução do custo computacional. A avaliação desta relação entre uma perda aceitável de qualidade e o ganho obtido em redução do custo computacional não é tarefa simples. Para tentar visualizar esta relação, a Tabela 4.5 apresenta os resultados de qualidade e custo computacional para o algoritmo DS, onde o nível de subamostragem escolhido é o maior possível desde que apresente uma perda média de PSNR inferior a 0,6dB.

Para bloco 8x8, o nível de subamostragem escolhido foi 2:1, onde a perda média de PSNR foi de 0,44dB e a redução do número de comparações foi de 2,04 vezes em relação à versão sem subamostragem. A subamostragem 4:1 insere uma perda média de PSNR de apenas 0,3dB para o algoritmo DS com blocos de 16x16. A redução do número de comparações foi de aproximadamente 4,1 vezes, acima do nível de subamostragem utilizado. Para blocos 32x32, o nível de subamostragem escolhido foi 8:1, com perdas médias de PSNR de 0,57dB e com uma redução do número de comparações de aproximadamente 8,7 vezes, em relação à versão sem subamostragem. Os resultados para o bloco 64x64 são os menos afetados pelo aumento da subamostragem, pois mesmo para o nível 16:1, as perdas médias em PSNR são inferiores a 0,4dB. A redução em números de comparações foi superior a 16,2 vezes, em relação à versão sem subamostragem de pixel. A última linha da Tabela 4.5 apresenta também o número de vetores de movimento necessários (por quadro) com a utilização de cada tamanho de bloco.

Tabela 4.5: Resultados para a EM com o algoritmo DS para diferentes tamanhos de bloco e diferentes níveis de subamostragem

Parâmetro	Bloco 8	Bloco 16	Bloco 32	Bloco 64
PSNR sem Subamostragem (dB)	33,34	33,06	32,46	31,22
Subamostragem	2:1	4:1	8:1	16:1
Variação PSNR (dB)	-0,44	-0,3	-0,57	-0,39
Nº de BCCs com sub ($x10^9$)	0,169	0,046	0,012	0,003
Redução de BCCs (vezes)	1,01	1,03	1,09	1,02
Nº de Comp. com sub ($x10^9$)	5,42	2,97	1,53	0,84
Redução de Comp. (vezes)	2,03	4,15	8,7	16,26
Nº de vetores por quadro	32640	8160	2040	510

Os resultados do uso da subamostragem de pixel na EM em vídeos de alta definição mostraram que esta é uma estratégia interessante, principalmente para blocos grandes. O uso da subamostragem de pixel em níveis baixos, como 2:1 e 4:1, pode ser usado até mesmo para blocos menores, no entanto, implica em perdas desprezíveis para blocos 32x32 e 64x64 (sempre inferiores a 0,1dB). No entanto, assumindo uma perda um pouco maior, os resultados de redução do custo computacional são extremamente significativos. A redução do número de comparações simplifica o cálculo do SAD, já que sua complexidade está diretamente ligada ao número de pixels comparados.

As avaliações apresentadas neste capítulo demonstraram que os resultados da EM sofrem alterações significativas com o aumento da resolução do vídeo. Os resultados comparativos, entre os algoritmos FS e DS, demonstraram que as diferenças de qualidade entre estes algoritmos crescem significativamente com o aumento da resolução dos vídeos. Com a análise dos mínimos locais foi possível evidenciar que este aumento é causado pelo crescimento no número de mínimos locais, que são mais evidentes em vídeos de alta definição. Esta análise demonstrou que o número de mínimos locais cresce significativamente com o aumento da resolução dos vídeos. No entanto, quando uma resolução específica é avaliada, o número de mínimos locais está diretamente ligado ao tamanho do bloco utilizado na EM, sendo que quanto menor o bloco, maior o número de mínimos locais.

O uso de vários tamanho de bloco na EM em vídeos HD 1080p também foi avaliado. Para isto, o software de referência do padrão H.264/AVC foi utilizado. Os resultados demonstraram que em praticamente 60% dos casos o maior bloco possível, o bloco de 16x16 pixels, é o escolhido. Isto demonstra a importância de blocos maiores para a EM em vídeos HD 1080p, e que esta relevância tende a aumentar para resoluções maiores. Pensando nisso, uma avaliação da EM em vídeos HD 1080p, com blocos de tamanho 32x32 e 64x64 pixels foi realizada. Os resultados de qualidade apresentam perdas significativas, no entanto, a avaliação isolada da EM não contempla os principais benefícios da utilização de blocos maiores na EM, como a redução no número de vetores de movimento, por exemplo. Estudos realizados com o padrão H.264/AVC demonstram que entre 35% a 50% do *bitstream* do vídeo codificado pode ser utilizado apenas para representar os vetores de movimento (ITU-T, 2011). O uso de blocos maiores pode beneficiar também outras etapas do codificador, como as transformadas e quantização e codificação de entropia.

O uso da subamostragem de pixel também foi avaliado, e se mostrou uma alternativa interessante também para vídeos de alta definição. Diferentes níveis de subamostragem foram aplicados a diferentes tamanhos de bloco. Os resultados demonstraram que blocos maiores suportam maiores níveis de subamostragem, apresentando menores perdas de qualidade.

Os resultados apresentados neste capítulo foram importantes para o desenvolvimento algorítmico para a EM em vídeos de alta definição, que será apresentado no próximo capítulo. Com a análise dos mínimos locais, ficou clara a necessidade do desenvolvimento de algoritmos que sejam mais eficientes em relação a esta característica dos vídeos de alta definição. O uso de blocos maiores que 16x16 também foi explorado nos novos algoritmos, assim como a utilização da subamostragem de pixel. Para cada tamanho de bloco foi utilizado um nível de subamostragem adequado, de acordo com os resultados obtidos neste capítulo.

5 ALGORITMOS DESENVOLVIDOS

Neste capítulo serão apresentados os resultados das investigações algorítmicas realizadas nesta tese. Todos os algoritmos foram desenvolvidos em linguagem C e aplicados às dez sequências de teste HD 1080p (1920x1080) já apresentadas. Os resultados algorítmicos apresentados neste capítulo são referentes às médias obtidas para estas 10 sequências.

Os resultados de qualidade foram avaliados pelos valores de PSNR obtidos e também pelo percentual de redução do resíduo (PRR). O custo computacional será mensurado pelo número de blocos candidatos comparados (BCC), bem como pelo número de operações de comparação executadas.

Os resultados obtidos com os algoritmos desenvolvidos nesta tese geraram algumas publicações originais, como (PORTO, 2011), (CRISTANI, 2011), (NOBLE, 2011) e (PORTO, 2012). As publicações geradas a partir dos resultados desta tese são apresentadas em detalhes no Apêndice B.

5.1 Os Algoritmos SDS-DIC e QSDS-DIC

Os primeiros algoritmos desenvolvidos especialmente focados para a implementação em hardware foram o *Sub-sampled Diamond Search with Dynamic Iteration Control* (SDS-DIC) e o *Quarter Sub-sampled Diamond Search with Dynamic Iteration Control* (QSDS-DIC) (PORTO, 2008). Estes algoritmos utilizam o algoritmo DS como padrão de busca, aliados a subamostragem de pixel de 2:1 no SDS-DIC e 4:1 no QSDS-DIC e com um controle dinâmico no número de iterações.

A implementação em hardware do algoritmo DS apresenta todos os complicadores inerentes ao desenvolvimento de algoritmos rápidos de EM em hardware. Um dos principais problemas é a impossibilidade de determinar quantas iterações serão necessárias para a geração de um vetor de movimento. O algoritmo DS pode usar *n* iterações do padrão LDSP até que a condição de parada seja atendida. O valor de *n* pode variar significativamente de acordo com o tamanho do bloco utilizado, conforme apresentado no capítulo 4. O número de iterações também pode variar drasticamente de acordo com as características do vídeo, onde vídeos com movimentação intensa tendem a gerar um número maior de iterações. Desta forma, é impossível determinar quantos ciclos de *clock* serão necessários para que a arquitetura entregue um vetor de movimento. Além disso, o cálculo do desempenho e também a integração da EM com os demais blocos do codificador, ficam extremamente complicados.

Para solucionar estes problemas, algum mecanismo de restrição do uso de iterações deve ser implementado. O método mais simples é inserir uma restrição fixa, limitando o número máximo de iterações para a geração de cada vetor de movimento. Desta forma, caso este número de iterações seja ultrapassado, a busca é encerrada e o vetor de

movimento é gerado para o bloco candidato de menor SAD. Evidentemente, a inserção desta restrição ao algoritmo irá acarretar em perdas de qualidade, já que, em alguns casos, a busca será interrompida antes que a condição de parada do algoritmo seja satisfeita. Quanto maior o valor de *n*, menores serão as ocorrências de interrupção antecipada da busca, no entanto, valores muito elevados de *n* podem prejudicar a taxa de processamento atingida pela arquitetura.

O controle dinâmico de laços (DIC – *Dynamic Iteration Control*) foi desenvolvido para inserir uma restrição dinâmica no número de iterações do algoritmo. Com o DIC, o algoritmo tem a possibilidade de utilizar um número variável de iterações. Isto possibilita que, em alguns casos, o algoritmo possa convergir para regiões mais distantes do centro em busca de blocos candidatos com menor SAD. O DIC trabalha com um histórico de número de iterações de 16 blocos atuais. O conjunto de 16 blocos atuais tem um total de 16**n* iterações para a geração dos seus vetores de movimento. O valor de *n* deve ser maior que a média de iterações (obtida por experimentação sobre as amostras de teste HD 1080p) do algoritmo DS. Toda vez que um vetor for gerado com um número de iterações menor que o valor máximo estipulado (*n*), as iterações restantes são acumuladas e disponibilizadas para os demais blocos. O DIC permite que se tenha um valor fixo de iterações para um conjunto de 16 blocos candidatos, mesmo que o número de iterações utilizadas por cada bloco (dentro deste grupo de 16) seja diferente. O fluxograma da Figura 5.1 apresenta o comportamento do DIC, utilizado nos algoritmos SDS-DIC e QSDS-DIC.

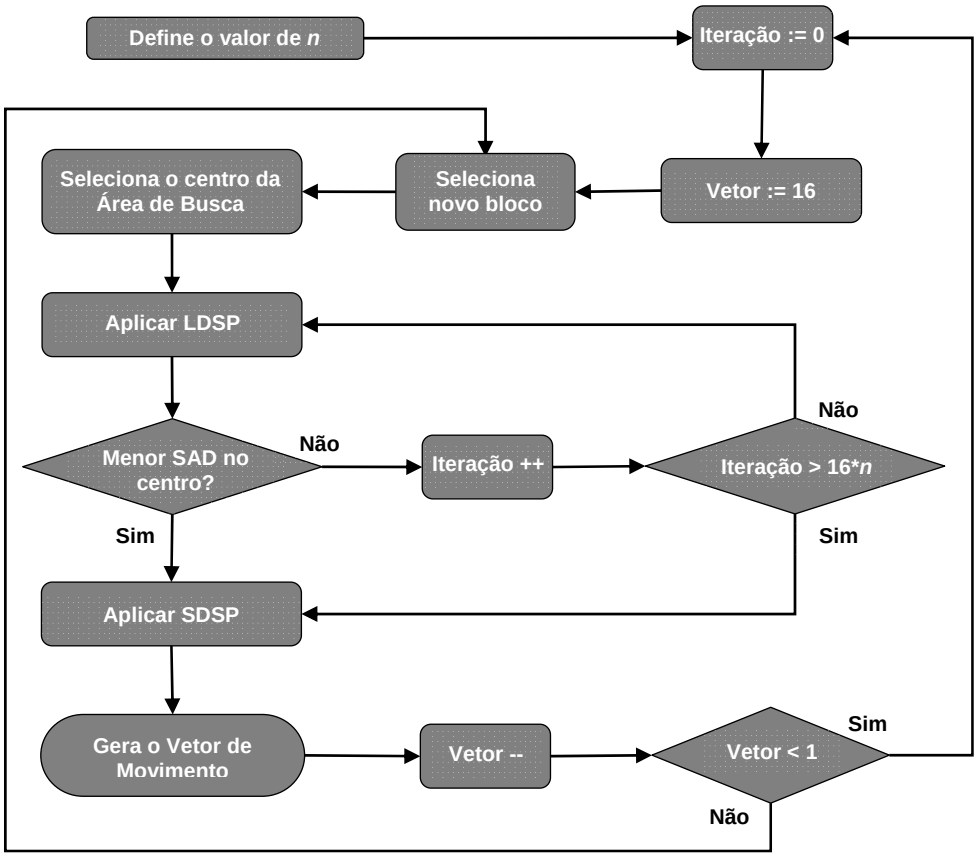


Figura 5.1: Fluxograma do algoritmo do DIC para os algoritmos SDS-DIC e QSDS-DIC

Os algoritmos SDS-DIC e QSDS-DIC utilizam subamostragem de pixel no nível 2:1 e 4:1, respectivamente. A subamostragem de pixel reduz a quantidade de memória

interna necessária para o armazenamento da área de busca e do bloco atual, além disso, reduz também a largura de banda necessária para a comunicação com a memória externa. O uso de subamostragem também possibilita a redução das unidades de processamento de SAD, reduzindo os custos em área e potência.

A Tabela 5.1 apresenta os resultados médios de qualidade (PSNR) e custo computacional (medido em número de comparações) para os algoritmos SDS-DIC e QSDS-DIC, para diferentes valores de *n*. Os resultados para o algoritmo DS com subamostragem de pixel, 2:1 (SDS) e 4:1 (QSDS), também são apresentados, bem como versões com restrição fixa de iterações. Os resultados da Tabela 5.1 foram gerados para blocos de tamanho 16x16. Este tamanho de bloco foi escolhido, pois é o mesmo tamanho da arquitetura desenvolvida para este algoritmo, que será apresentada na sessão 6.5.

Tabela 5.1: Resultados para os algoritmos SDS e QSDS com restrição fixa iterações e com DIC

Algoritmo	PSNR	PRR (%)	Comp. ($x10^9$)
SDS	33,0	61,38	6,1
SDS-DIC (n = 10)	32,8	54,65	6,04
SDS n = 10 fixo	32,53	53,11	5,86
QSDS	32,76	60,22	2,97
QSDS-DIC (n = 20)	32,75	54,02	2,97
QSDS n = 20 fixo	32,69	53,70	2,95

Os resultados de qualidade apresentados na Tabela 5.1 demonstram a eficiência do uso do DIC em relação à restrição fixa de laços. Para o algoritmo SDS, o uso do DIC resultou em uma perda média de apenas 0,2dB, acompanhado de uma pequena redução do número de comparações. A versão com restrição fixa de laços, apesar de utilizar o mesmo valor de iterações máximas por vetor (n = 10), apresentou uma redução de aproximadamente 0,5dB em relação ao SDS. Os resultados para o algoritmo QSDS não apresentaram variações muito significativas, sendo que a versão com DIC possui praticamente o mesmo PSNR obtido sem a restrição de laços. A versão com restrição fixa apresenta uma redução média no PSNR de 0,07dB. Mesmo apresentando ganhos médios pequenos, o uso do DIC pode gerar ganhos mais expressivos especialmente em sequências de maior movimentação, onde a probabilidade de ocorrência de restrições na busca é maior. Para sequências de baixa movimentação, os resultados obtidos com o DIC e com a restrição fixa são muito similares, pois, nestes casos, o valor médio de iterações do SDS e QSDS é muito inferior aos valores de *n* avaliados (10 e 20).

5.2 O Algoritmo MPDS

O primeiro algoritmo da classe multiponto desenvolvido foi o *Multi-Point Diamond Search* (MPDS). O MPDS utiliza como base o algoritmo DS, que é aplicado em cinco instâncias (núcleos de busca). Cada núcleo de busca executa o padrão do algoritmo DS em diferentes regiões da área de busca. Um núcleo é aplicado ao centro da área de busca, gerando o mesmo resultado que o algoritmo DS. Os quatro núcleos restantes são aplicados em quatro setores, que representam um quarto da área de busca cada. A Figura 5.2 ilustra a aplicação dos cinco núcleos de busca do algoritmo MPDS a uma

área de busca. A distância da aplicação dos núcleos dos setores é determinada pelo parâmetro *d*. O valor de *d* determina a distância, em pixels, do centro do núcleo de busca em relação aos eixos X e Y. Por exemplo, para *d* = 10, o núcleo do setor A será situado na posição (10, 10) da área de busca, (-10, 10) no setor B, (-10, -10) no setor C e (10, -10) no setor D. O MPDS executa os cinco núcleos de busca até que todos encontrem os seus melhores resultados. O resultado final do MPDS será o melhor resultado dentre os melhores resultados encontrados pelos cinco núcleos. No pior caso, o MPDS encontrará o resultado igual ao algoritmo DS. Isto ocorrerá quando nenhum resultado, entre os quatro setores, for melhor que o resultado do núcleo central.

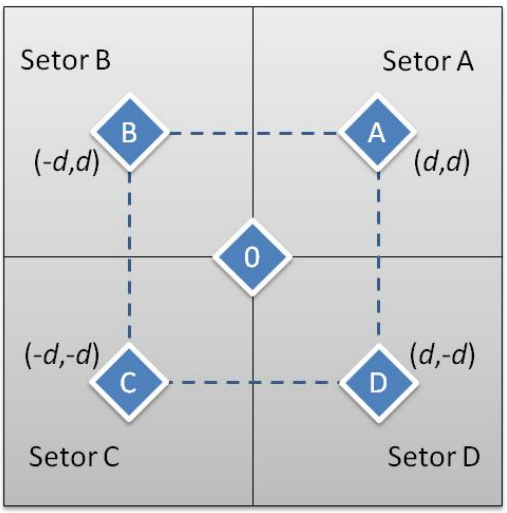


Figura 5.2: Núcleos de busca do algoritmo MPDS

O fluxograma da Figura 5.3 descreve o comportamento do algoritmo MPDS. Para facilitar a visualização, a aplicação do DS em cada setor é apresentada de nodo sequencial. Todos os núcleos do MPDS podem trabalhar no modo serial ou paralelo. Quando o modo paralelo é utilizado, o desempenho do algoritmo é praticamente igual ao do algoritmo DS, exceto pela etapa de comparação adicional entre os resultados das cinco instâncias. O MPDS prioriza os resultados obtidos pelo núcleo central, como critério de desempate. Esta estratégia é utilizada pois os vetores gerados pelo núcleo central tendem a ser menores, contribuindo para a redução do *bitstream* do vídeo codificado (RICHARDSON, 2003). A mesma estratégia é utilizada para o desempate entre os setores, onde o setor com menor vetor de movimento será o vencedor.

O valor do parâmetro *d* pode influenciar drasticamente no resultado do algoritmo MPDS. O valor ideal de *d* pode variar significativamente com as características do vídeo a ser codificado. Para vídeos com baixa movimentação, o MPDS pode obter bons resultados com baixos valores de *d*, pois bons blocos candidatos tendem a ser encontrados próximos ao centro da área de busca. Já em vídeos com alta movimentação, o MPDS pode ser beneficiado com um valor maior de *d*, pois facilitaria a busca de bons blocos candidatos localizados em regiões mais distantes do centro. A Figura 5.4 apresenta as curvas de variação do ganho PSNR para o algoritmo MPDS, considerando a variação do parâmetro *d*. As curvas da Figura 5.4 foram geradas considerando a variação do parâmetro *d* de zero a quarenta, incrementado de cinco em cinco unidades, para as dez sequências de teste HD 1080p apresentadas anteriormente, considerando blocos de tamanho 8x8.

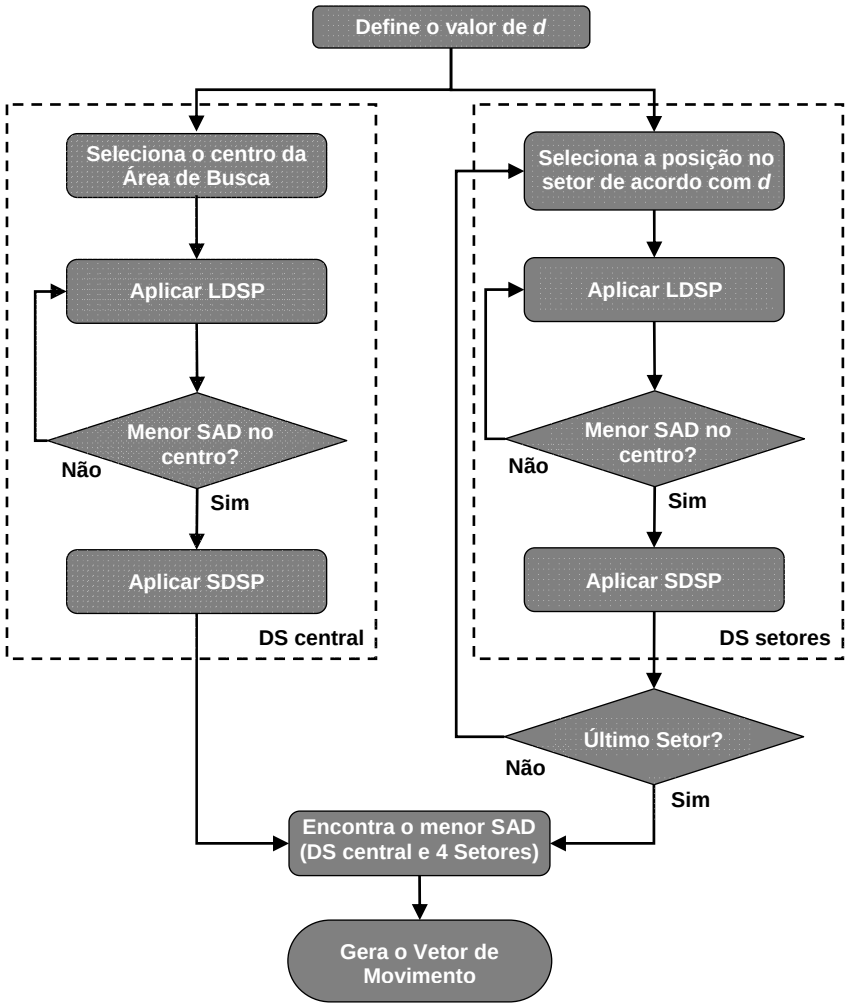


Figura 5.3: Fluxograma do algoritmo MPDS

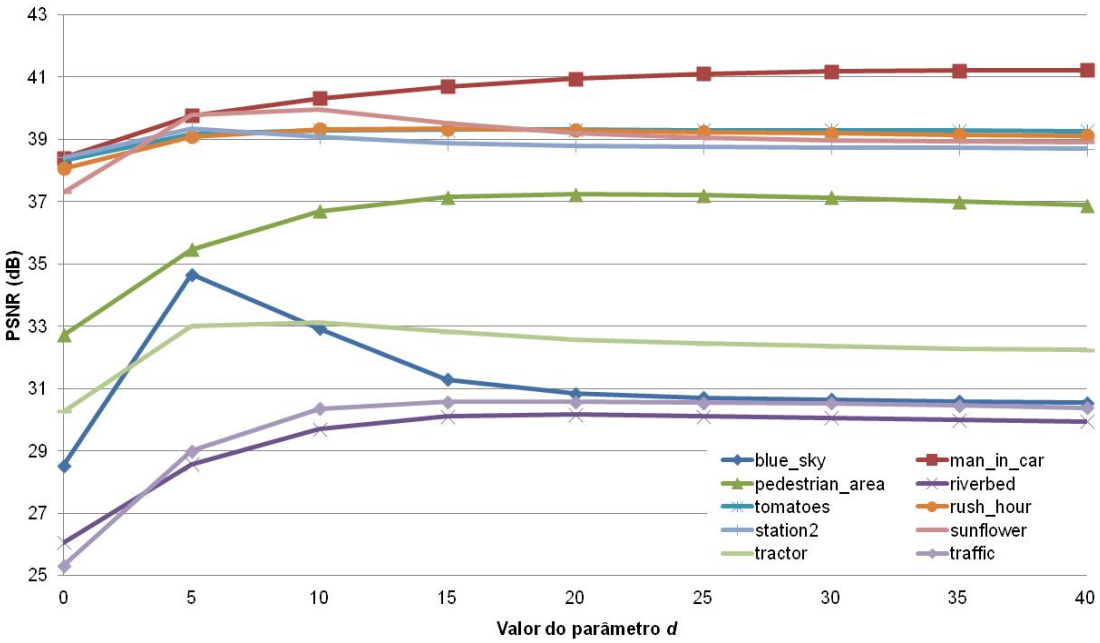


Figura 5.4: Variação do parâmetro *d* para o algoritmo MPDS com blocos 8x8

Analisando as curvas da Figura 5.4 é possível perceber que o resultado de PSNR obtido para um dado valor de *d* varia consideravelmente de acordo com a sequência codificada. Para sequências com baixa movimentação, como blue_sky, por exemplo, o melhor resultado de PSNR foi obtido para *d* = 5, com um ganho superior a 6dB em relação à distância zero (resultado do algoritmo DS). Para valores maiores de *d* os resultados de PSNR são inferiores, sendo que para *d*s maiores de 25 os valores de PSNR são praticamente estáveis. Para sequências com maior movimentação, como man_in_car, o MPDS apresenta ganhos constantes de PSNR com o aumento do *d*. O melhor resultado foi obtido para *d* = 40, com um ganho de 2,8dB, no entanto, os resultados para as distâncias 35, 40 e 25 apresentam resultados muito semelhantes.

Para determinar um valor para *d* que maximize o ganho médio de PSNR, o mesmo experimento foi realizado para os tamanhos de bloco 8x8, 16x16, 32x32 e 64x64. Os resultados médios para cada tamanho de bloco são apresentados na Tabela 5.2. Analisando os resultados apresentados na Tabela 5.2 é possível encontrar o valor de *d* onde o melhor resultado médio de qualidade é obtido, considerando as 10 sequências de teste utilizadas. O melhor valor de *d* varia de acordo com o tamanho do bloco utilizado. Para os blocos 8x8 e 16x16, o melhor resultado de PSNR foi encontrado para *d* = 10, enquanto que o bloco 32x32 obteve o maior ganho para *d* = 15 e para o bloco 64x64, o melhor PSNR foi obtido para *d* =20.

Tabela 5.2: Resultados médios de PSNR (dB) do algoritmo MPDS, com a variação do parâmetro *d*, em diferentes tamanhos de bloco

Distância	8x8	16x16	32x32	64x64
0	33,34	33,05	32,46	31,22
5	35,79	34,51	33,07	31,52
10	**36,07**	**34,76**	33,30	31,65
15	35,97	34,67	**33,36**	31,70
20	35,89	34,56	33,35	**31,72**
25	35,84	34,50	33,32	31,71
30	35,81	34,46	33,28	31,70
35	35,76	34,41	33,24	31,68
40	35,72	34,38	33,21	31,66

Os resultados da Tabela 5.2 apresentam uma variação do parâmetro *d* de cinco em cinco unidades. Para encontrar o valor exato de *d*, um refinamento foi executado variando a distância *d* de uma em uma unidade dentro da faixa que apresentou os melhores resultados. Para os blocos de tamanho 8x8 e 16x16, os melhores resultados foram obtidos com valores de *d* variando de 5 a 15, para blocos 32x32 de 10 a 20 e para os blocos 64x64 a melhor faixa de valores para *d* foi de 15 a 25. A Tabela 5.3 apresenta o valor de *d* que maximiza os resultados médios de qualidade do algoritmo MPDS em cada tamanho de bloco avaliado. Os resultados de ganho médio, em relação ao algoritmo DS também são apresentados. O ganho diminui com o aumento do tamanho do bloco, isto ocorre, pois para tamanhos maiores de bloco, o número de bons blocos candidatos diminui, conforme ilustrado na Figura 4.7, que apresenta os mapas de SAD para diferentes tamanhos de bloco em uma mesma área de busca. O algoritmo MPDS

para bloco 64x64 apresenta um ganho médio de apenas 0,5dB, em relação ao algoritmo DS. Mesmo considerando o enorme potencial de subamostragem, inerente a blocos de tamanho 64x64, este tamanho de bloco será desconsiderado dos resultados finais do algoritmo MPDS, pois os ganhos em qualidade são marginais.

Tabela 5.3: Melhor valor de *d* para o algoritmo MPDS em diferentes tamanhos de bloco

Tamanho de Bloco	**8x8**	**16x16**	**32x32**	**64x64**
Valor de *d*	8	10	15	21
Ganho Médio sobre o DS (dB)	2,75	1,71	0,9	0,51

Os resultados de qualidade e custo computacional do algoritmo MPDS (considerando os valores de *d* apresentados na Tabela 5.3) para diferentes tamanhos de bloco são apresentados na Tabela 5.4. Os resultados com subamostragem de pixel também são apresentados. Cada tamanho de bloco usará o nível de subamostragem mais adequado, de acordo com os resultados apresentados na Tabela 4.5. Os resultados de qualidade mostram que o algoritmo MPDS pode atingir um PSNR médio maior que 36dB para blocos 8x8, gerando uma redução do resíduo superior a 73%. Com o aumento do tamanho dos blocos, os resultados de PSNR apresentam perdas em torno de 1,4dB e, em relação ao PRR, as perdas são da ordem de 5%. Apesar da redução do número de blocos comparados, o uso de blocos maiores também aumenta o número de comparações necessárias.

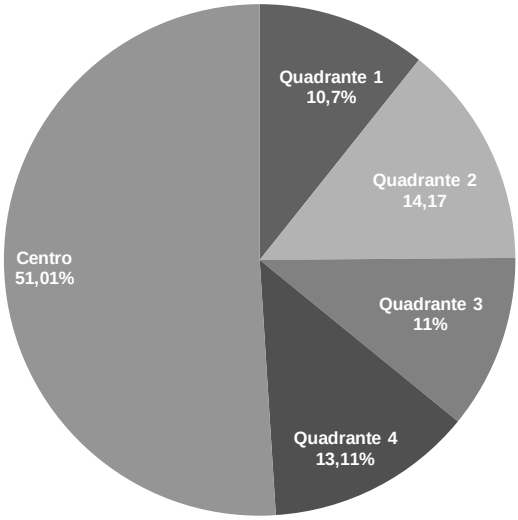
No entanto, além da redução do número de vetores de movimento, o uso de blocos maiores possibilita uma exploração maior da subamostragem. O uso da subamostragem de pixel 4:1 no algoritmo MPDS com blocos 16x16 gera uma perda média de PSNR de apenas 0,24dB e aproximadamente 1% no PRR, enquanto o custo computacional é reduzido em mais de 76%. Estes resultados demonstram que o uso da subamostragem de pixel também é uma estratégia interessante para o algoritmo MPDS, pois uma redução significativa no custo computacional pode ser obtida com perdas praticamente insignificantes nos resultados de qualidade.

Tabela 5.4: Resultados de qualidade e custo computacional do algoritmo MPDS

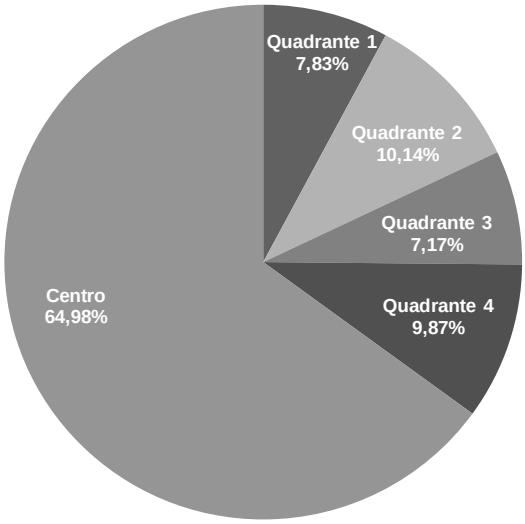
Bloco/Sub	**PSNR (dB)**	**PRR (%)**	**BCCs ($x10^9$)**	**Comp. ($x10^9$)**
8x8	36,09	73,62	1	64,17
8x8/2:1	35,94	73,04	0,98	31,48
16x16	34,75	68,32	0,31	78,72
16x16/4:1	34,51	67,29	0,29	18,56
32x32	33,35	63,2	0,1	101,233
32x32/8:1	32,91	61,7	0,08	10,79

Os ganhos em PSNR apresentados pelo algoritmo MPDS em relação ao algoritmo DS são obtidos com os blocos candidatos escolhidos nos setores. A Figura 5.5 apresenta gráficos que ilustram o percentual de escolha dos melhores blocos candidatos no centro e em cada um dos setores, para os diferentes tamanhos de bloco avaliados. Analisando

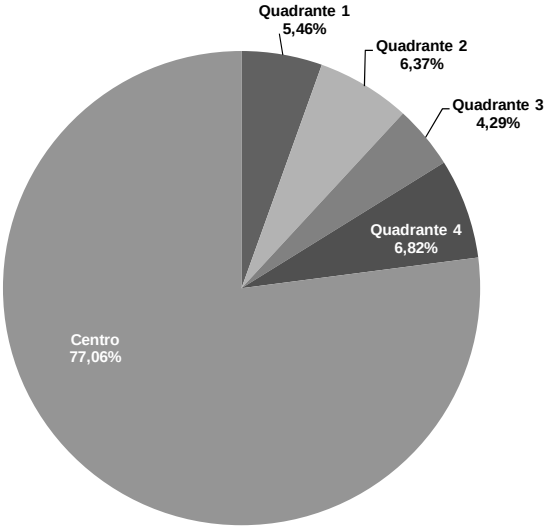
os gráficos da Figura 5.5, é possível entender melhor porque os ganhos do algoritmo MPDS, em relação ao algoritmo DS, diminuem com o aumento do tamanho do bloco. Para blocos de tamanho 8x8, o percentual de blocos candidatos escolhidos no centro e nos setores é praticamente igual. Isto significa que, em praticamente 50% dos casos, um bloco candidato com SAD melhor do que o obtido pelo algoritmo DS foi escolhido. Para blocos de tamanho 16x16, o percentual de escolha no centro é de praticamente 65%, logo, em apenas 35% dos casos um bloco candidato de menor resíduo é encontrado em um dos setores. Com o uso de blocos 32x32, o percentual de escolhas no centro é ainda maior, sendo que apenas 23% dos vetores são gerados para blocos candidatos de um dos setores.



Bloco 8x8



Bloco 16x16



Bloco 32x32

Figura 5.5: Percentual de escolha em cada setor do algoritmo MPDS

Estes resultados podem ser explicados através dos mapas de SAD apresentados na Figura 4.7, pois com aumento do tamanho dos blocos, diminuem as ocorrências de mínimos locais. Desta forma, o algoritmo MPDS encontra menos opções de bons blocos candidatos longe do centro da área de busca e seus resultados passam a se aproximar dos resultados obtidos pelo algoritmo DS. Os impactos do uso da subamostragem sobre

o percentual de escolhas no centro e nos setores também foi avaliado, no entanto, as diferenças foram muito pequenas. Para blocos 8x8 e 16x16, o percentual de escolha nos setores subiu cerca de 0,3% e 1,5%, respectivamente. Para blocos 32x32 este acréscimo foi de aproximadamente 3%.

5.3 O Algoritmo DMPDS

Os resultados de qualidade do algoritmo MPDS estão diretamente ligados ao valor utilizado para o parâmetro *d*, como ilustrado na Figura 5.4. Além disso, existe uma variação grande entre o valor ótimo obtido para cada sequência de teste. Sequências com baixa movimentação apresentam os melhores resultados com valores pequenos de *d*, já sequências de maior movimentação, são beneficiadas por valores mais altos. No entanto, o algoritmo MPDS utiliza um valor fixo para o parâmetro *d*, que corresponde ao valor que maximiza o resultado médio de PSNR para todas as sequências avaliadas.

A Tabela 5.5 apresenta os valores ótimos de *d* para o algoritmo MPDS, com blocos 8x8, 16x16 e 32x32 em cada sequência de teste utilizada. Os valores ótimos de *d* variam consideravelmente em todos os tamanhos de bloco utilizados. A maior variação ocorre entre as sequências station2 e man_in_car, onde o valor ótimo de *d* varia de cinco para 40 para todos os tamanhos de bloco. Também existe variação entre os tamanhos de bloco utilizados, sendo que o valor ótimo de *d* tende a aumentar com o aumento do tamanho do bloco. Esta tendência pode ser observada através dos resultados médios de *d*.

Tabela 5.5: Valores ótimos de *d* para o algoritmo MPDS em cada sequência

Vídeo	8x8	16x16	32x32
station2	5	5	5
blue_sky	5	6	11
sun_flower	8	12	17
tractor	8	6	7
traffic	16	13	14
man_in_car	40	40	40
pedestrian	20	21	26
riverbed	19	23	32
rolling	16	20	27
rush_hour	12	15	24
Média	**14,9**	**16,1**	**20,3**

O algoritmo *Dynamic Multi-Point Diamond Search* (DMPDS) utiliza o mesmo processo de busca apresentado na Figura 5.3 para o algoritmo MPDS, no entanto, o parâmetro *d* é controlado dinamicamente, de acordo com a característica do vídeo processado. Neste algoritmo, o valor do parâmetro *d* é ajustado de acordo com os resultados de qualidade obtidos. Desta forma, é possível que o algoritmo se adapte dinamicamente ao vídeo que está sendo processado, variando o valor do parâmetro *d* de forma que se aproxime ao máximo aos valores apresentados na Tabela 5.5. A Figura 5.6 apresenta um diagrama que ilustra o mecanismo de controle dinâmico do valor do

parâmetro *d* no algoritmo DMPDS. O DMPDS trabalha com o parâmetro *d* e um Δ. Para o primeiro quadro, será aplicado o algoritmo MPDS com um valor inicial de *d*, o segundo quadro com *d*1 = *d* - Δ e o terceiro com *d2* = *d* + Δ. O valor de *d* que resultar no maior PSNR será escolhido para a codificação do próximo quadro e o valor de Δ será atualizado para Δ = Δ/2. Este processo será repetido até que o valor de Δ seja igual a um e, então, a oscilação no valor de *d* será de uma unidade para mais ou para menos.

O algoritmo DMPDS possui ainda um mecanismo de aceleração de convergência. Quando a variação de *d* for de um pixel para mais ou para menos, caso o melhor valor de PSNR seja encontrado com o aumento do valor de *d* e isto ocorra duas vezes seguidas, o valor de Δ será multiplicado por dois. O mesmo ocorrer caso o melhor PSNR seja encontrado para uma redução do valor de *d*. No caso desta situação se repetir, o valor de Δ será novamente multiplicado por dois, e assim sucessivamente. Desta forma, é possível identificar uma tendência de variação da característica do vídeo e acelerar a convergência para o valor ideal de *d* para as novas características do vídeo.

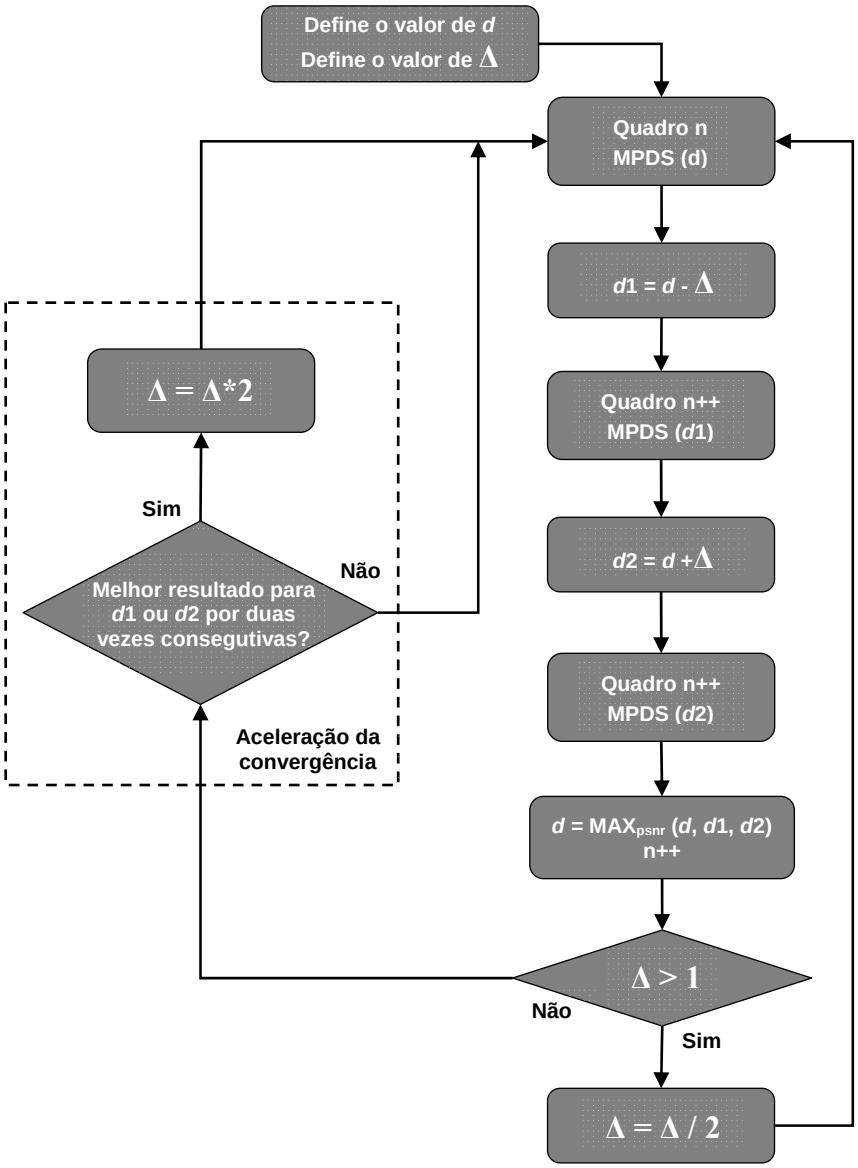


Figura 5.6: Controle dinâmico do parâmetro *d* no algoritmo DMPDS

O algoritmo DMPDS foi aplicado às dez sequências de testes HD 1080p utilizando blocos de tamanho 8x8, 16x16 e 32x32. O parâmetro *d* foi inicializado com valores diferentes para cada tamanho de bloco, tomando por base os resultados apresentados para o algoritmo MPDS, apresentados na Tabela 5.3. O valor do parâmetro Δ também varia de acordo com o tamanho do bloco utilizado, os valores de Δ utilizados foram 4, 5 e 8, para os blocos 8x8, 16x16 e 32x32, respectivamente. Os valores de *d* e Δ utilizados para cada tamanho de bloco (apresentados na Tabela 5.5) visam possibilitar ao algoritmo uma rápida convergência para o valor ótimo de *d* em cada sequência. Para avaliar a eficiência do controle dinâmico do algoritmo DMPDS, os resultados obtidos para o valor ótimo de *d* foram considerados. Estes resultados foram chamados de *Best Multi-Point Diamond Search* (BMPDS) e representam o limite superior do algoritmo MPDS, onde cada sequência é executada para o seu valor ótimo de *d*. A Figura 5.7 apresenta os resultados de PSNR para o algoritmo DMPDS com blocos 8x8 para as 10 sequências de teste HD 1080p. As curvas de PSNR para o algoritmo DS, MPDS, DMPDS e BMPDS também são apresentadas na Figura 5.7.

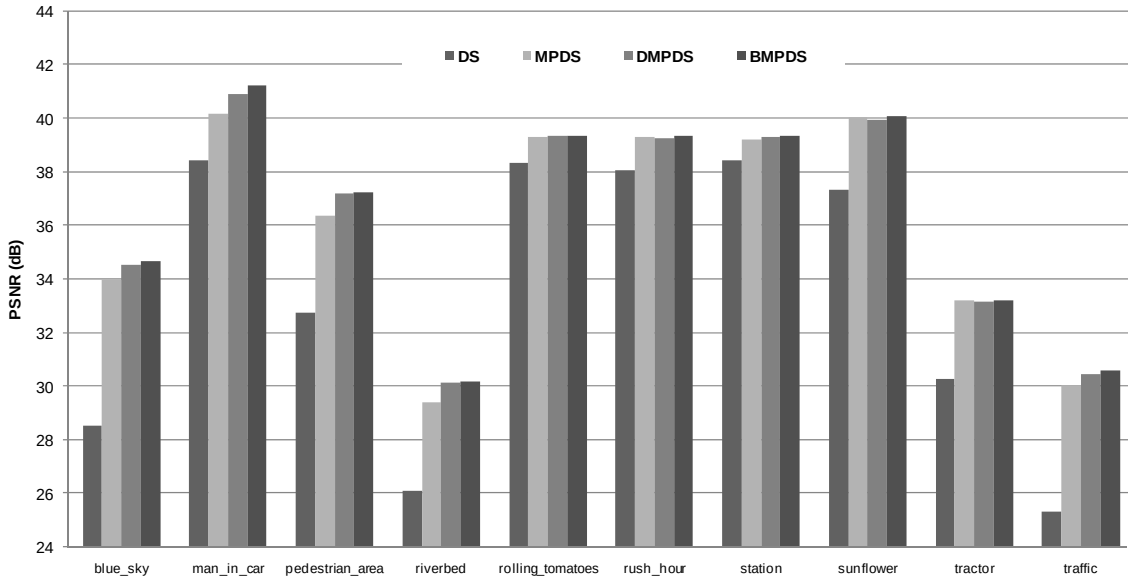


Figura 5.7: Curvas de ganho PSNR dos algoritmos DMPDS, MPDS, BMPDS e DS com blocos 8x8

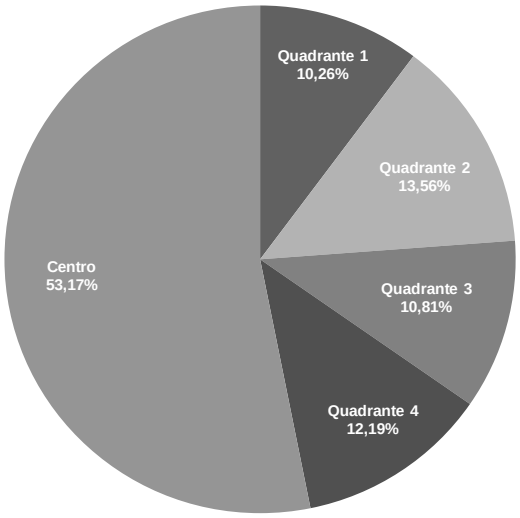
Os resultados de PSNR apresentados na Figura 5.7 demonstram que o algoritmo DMPDS apresenta ganhos em relação ao algoritmo MPDS em praticamente todas as sequências. O ganho médio do algoritmo DMPDS foi superior a 0,3dB em relação ao algoritmo MPDS, no entanto, para algumas sequências, como riverbed e pedestrian_area, os ganhos foram superiores a 0,7dB e 0,8dB, respectivamente. O algoritmo DMPDS obteve resultados muito similares aos obtidos pelo BMPDS, sendo que os resultados médios são apenas 0,15dB inferiores. Em relação ao algoritmo DS, o DMPDS apresenta um ganho médio superior a 3dB, no entanto, para a sequência blue_sky o ganho foi superior a 6dB e para as sequências pedestrian_area e riverbed, os ganhos foram superiores a 4dB.

A Tabela 5.6 apresenta os resultados de qualidade e custo computacional para o algoritmo DMPDS com os três tamanhos de bloco utilizados. Os resultados para o uso de subamostragem de pixel também são apresentados. Os resultados de qualidade (PSNR e PRR) do algoritmo DMPDS apresentam ganhos em relação aos obtidos pelo algoritmo MPDS em todos os tamanhos de bloco. As perdas inseridas com o aumento

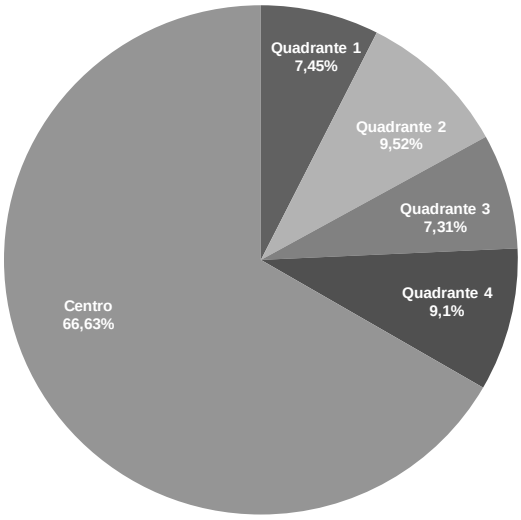
do tamanho dos blocos também são similares, cerca de 1,4dB no PSNR e 5% no PRR. O custo computacional do algoritmo DMPDS é ligeiramente superior em todos os casos. Os resultados com o uso de subamostragem de pixel também foram interessantes. Para blocos 16x16, por exemplo, as perdas foram inferiores a 0,2dB em PSNR e 1% no PRR, com uma redução do número de comparações superior a 76%.

Tabela 5.6: Resultados de qualidade e custo computacional do algoritmo DMPDS

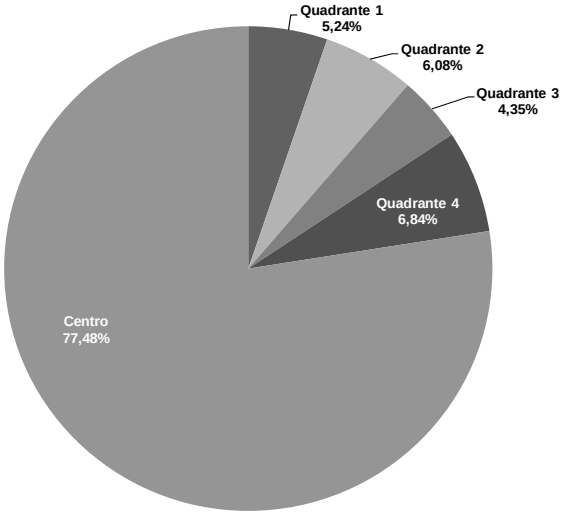
Bloco/Sub	PSNR (dB)	PRR (%)	BCCs (x10^6)	Comp. (x10^9)
8x8	36,41	74,27	1	65,42
8x8/2:1	36,26	73,69	1	32,16
16x16	34,87	68,7	0,32	81,55
16x16/4:1	34,65	67,76	0,3	19,29
32x32	33,38	63,35	0,1	103,27
32x32/8:1	32,96	61,88	0,08	10,88



Bloco 8x8



Bloco 16x16



Bloco 32x32

Figura 5.8: Percentual de escolha em cada setor do algoritmo DMPDS

Os percentuais de escolha em cada setor para o algoritmo DMPDS estão ilustrados na Figura 5.8 e são muito semelhantes aos obtidos pelo algoritmo MPDS, apresentados na Figura 5.5. O percentual de escolhas no centro aumentou para todos os tamanhos de bloco avaliados, inclusive para os resultados com subamostragem que não estão ilustrados na Figura 5.8. Isto demonstra que os blocos candidatos encontrado nos setores (1 a 4) são mais relevantes que os escolhidos pelo algoritmo MPDS, visto que mesmo com um percentual menor de uso, estes blocos resultam em melhores resultados de qualidade.

5.4 Os Algoritmos Aleatórios

A grande maioria dos algoritmos rápidos utiliza um padrão de busca, que deve iterar buscando a convergência para uma região de menor SAD dentro da área de busca. No entanto, esta estratégia é muito suscetível à escolha de mínimos locais, pois os critérios de parada sempre consideram que uma nova iteração deve gerar um resultado melhor que o obtido no passo anterior. Desta forma, é praticamente impossível ultrapassar um pico de SAD que separe um mínimo local do mínimo global. A primeira classe de algoritmos desenvolvida buscando robustez contra a escolha de mínimos locais foi à estratégia multiponto, a segunda classe é a dos algoritmos aleatórios.

Os algoritmos aleatórios exploram a aleatoriedade como forma de evitar a escolha de mínimos locais. A escolha aleatória de blocos candidatos dentro de uma área de busca possibilita ao algoritmo vencer eventuais picos de SADs, que seriam barreiras intransponíveis apenas com a utilização de refinamento iterativo. Dentro desta classe, diversos algoritmos foram desenvolvidos, desde a versão mais simples, que explora apenas a aleatoriedade, até soluções mais sofisticadas, que combinam características aleatórias e de refinamento. Todos os algoritmos aleatórios utilizam um tamanho fixo de área de busca. Esta estratégia é utilizada, pois serve como uma das métricas de comparação dos resultados entre os algoritmos. Uma característica interessante dos algoritmos aleatórios é que eles podem apresentar resultados diferentes para a execução do mesmo vídeo, pois pelo menos parte dos blocos candidatos dentro da área de busca é escolhida de forma aleatória.

Diversos algoritmos com busca aleatória foram desenvolvidos e todos possuem uma etapa aleatória, onde N blocos candidatos são sorteados e avaliados. No entanto, a simples exploração aleatória de blocos candidatos não resultou em bons resultados de qualidade. Novas versões foram desenvolvidas, agregando novas características, como a avaliação da região central da área de busca, refinamento fixo e iterativo sobre o melhor bloco candidato, entre outras. Os resultados da avaliação destes algoritmos demonstraram um ganho incremental proporcionado por cada nova característica agregada ao algoritmo. Nesta tese, apenas as quatro versões que apresentaram os melhores resultados de qualidade serão apresentadas.

5.4.1 Algoritmo *Random Search Plus* (RSP)

Uma característica importante na EM é que os blocos candidatos ao redor do centro da área de busca tendem a gerar bons resultados de resíduo. Isto ocorre, pois dado um bloco atual, o bloco candidato do quadro de referência situado na mesma posição espacial, tende a representar a mesma região do quadro. Grande parte dos algoritmos rápidos explora, primeiramente, a região central da área de busca visando explorar a localidade do bloco atual e podendo, assim, gerar uma convergência mais rápida. A exploração da localidade do bloco do quadro atual pode proporcionar a escolha de bons

blocos candidatos, principalmente em sequências de vídeos com baixa movimentação. Neste tipo de sequência de vídeo, grande parte dos mínimos globais é encontrada próxima à região central da área de busca, aumentando a relevância dos blocos candidatos desta região.

Outra característica presente na grande maioria dos algoritmos rápidos de EM é a iteração de um padrão de busca, que deve ser repetida até que uma dada condição seja satisfeita. Esta característica faz com que os algoritmos possam convergir para região que apresente menores SAD dentro da área de busca, através de *n* iterações do padrão de busca. Esta abordagem, utilizada de maneira isolada, apresenta grande suscetibilidade à escolha de mínimos locais, como mencionado anteriormente.

Além da exploração aleatória de blocos candidatos, o algoritmo RSP realiza uma avaliação da região central da área de busca e também um refinamento final interativo sobre o melhor bloco da etapa aleatória. O algoritmo RSP explora o bloco candidato central, bem como os quatro vizinhos imediatamente ao redor (superior, inferior, esquerdo e direito). Caso algum dos vizinhos apresente o menor SAD, a busca continua com uma nova iteração. Este bloco será o novo centro e os seus três vizinhos serão avaliados. O refinamento termina quando o menor SAD é encontrado no centro. Após o refinamento iterativo na região central (ou paralelamente a ele), o algoritmo RSP sorteia *N* blocos candidatos da área de busca. Os *N* blocos são comparados e o bloco candidato de menor SAD é encontrado. Por fim, o algoritmo RSP aplica um refinamento final iterativo (idêntico ao aplicado na região central da área de pesquisa), tendo o bloco candidato de menor SAD da etapa aleatória como centro.

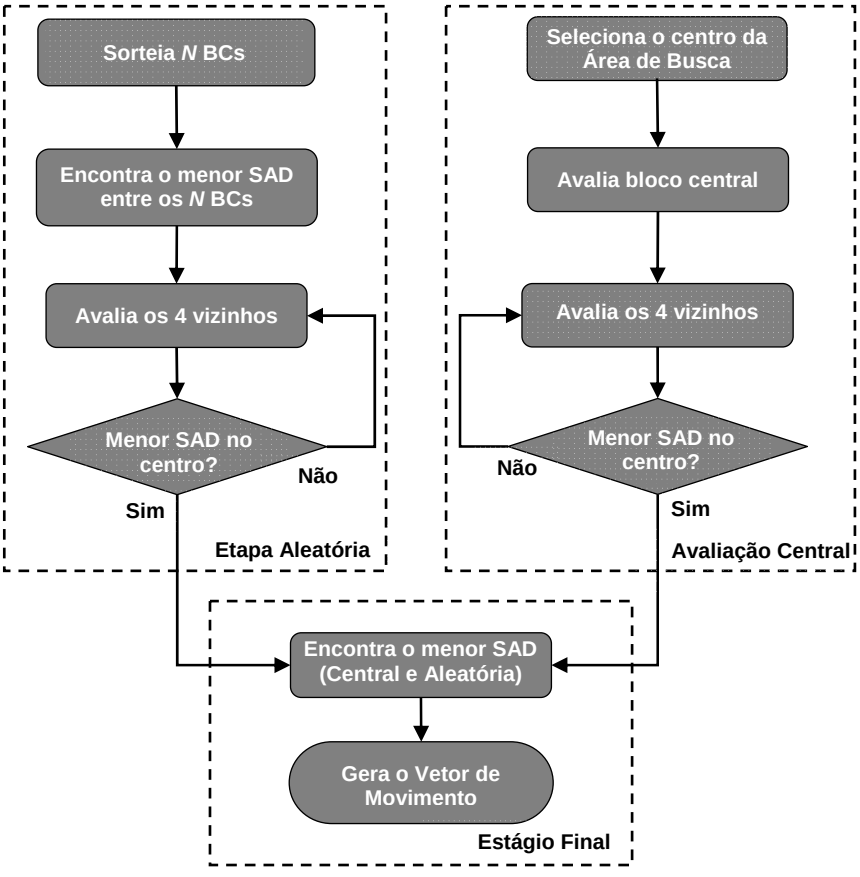


Figura 5.9: Fluxograma do algoritmo RSP

Com o uso da aleatoriedade o algoritmo RSP possui maior robustez para evitar a escolha de mínimos locais, desta forma, o refinamento iterativo pode contribuir para uma pequena convergência a uma região espacialmente próxima, mas que apresente menores resultados de SAD. A Figura 5.9 apresenta o fluxograma do algoritmo RSP. Este fluxograma apresenta o comportamento do algoritmo RSP de modo paralelo, para facilitar a compreensão, onde a avaliação da região central e a etapa aleatória são executadas em paralelo.

A Figura 5.10 apresenta um exemplo de busca do algoritmo RSP para N = 10. Na primeira etapa, o algoritmo RSP avalia o bloco central e seus quatro vizinhos (blocos em cinza escuro com bordas tracejadas). Neste exemplo, o melhor resultado da etapa de avaliação da região central da área de busca foi encontrado no centro (bloco com o número 1). Logo, apenas cinco blocos foram avaliados e nenhuma iteração foi realizada. Na segunda etapa, o algoritmo RSP sorteia os N (N = 10) blocos candidatos (blocos também em cinza escuro). O melhor bloco candidato, dentre os 10 sorteados, é encontrado (bloco com o número 2). O RSP aplica o refinamento final sobre o melhor bloco da etapa aleatória, avaliando seus quatro vizinhos imediatos (blocos em cinza claro). O menor SAD é encontrado no vizinho a direita (bloco com o número 3), uma nova iteração é aplicada, avaliando os três vizinhos deste bloco (blocos com linhas diagonais). Mais uma vez, o melhor resultado é encontrado para o vizinho a direita e uma nova iteração é realizada (blocos quadriculados). Desta vez, o menor SAD é encontrado no centro e a busca termina. Por fim, o algoritmo RSP compara o melhor bloco candidato da primeira e da segunda etapa (bloco 1 e bloco 4), o bloco de menor SAD será escolhido e um vetor de movimento será gerado para ele.

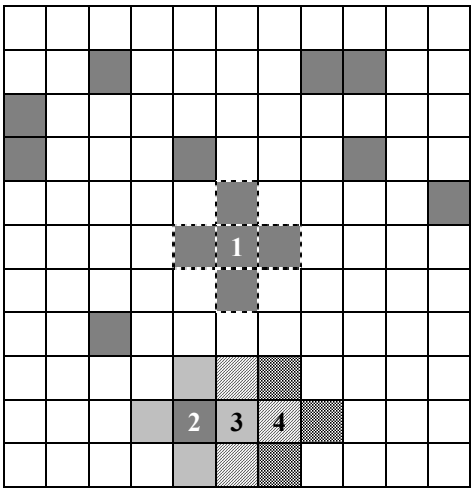


Figura 5.10: Exemplo de busca do algoritmo RSP para N = 10

O algoritmo RSP não possui custo computacional fixo. O número de blocos candidatos calculados e, consequentemente, o número total de comparações, é dependente do número de iterações realizadas nas etapas de refinamento da região central e do melhor bloco da etapa aleatória. Este número de iterações pode variar de acordo com o vídeo a ser codificado. É importante salientar que as duas principais etapas do algoritmo RSP (avaliação da região central e etapa aleatória) podem ser executadas em paralelo, pois não existem dependências de dados entre elas. Desta forma, é possível um ganho significativo de desempenho em aplicações multi tarefas, ou implementações em hardware, que explorem o paralelismo deste algoritmo.

5.4.2 Algoritmo *Iterative Random Search* (IRS)

O método utilizado para o sorteio dos *N* blocos candidatos nos algoritmos aleatórios não privilegia nenhum bloco candidato, ou seja, a probabilidade de escolha de um determinado bloco é a mesma para todos os blocos candidatos da área de busca. Desta forma, não é possível garantir uma distribuição adequada de blocos candidatos avaliados em todas as regiões da área de busca. É possível que grande parte dos *N* blocos candidatos sorteados (ou mesmo todos) estejam na mesma região da área de busca. Desta forma, nenhum bloco candidato seria avaliado em grande parte da área de busca.

Para garantir uma exploração mais eficiente da área de busca, o algoritmo *Iterative Random Search* (IRS) divide a área de busca em setores antes da realização dos sorteios. A área de busca é dividida em quatro setores (1, 2, 3 e 4), como ilustrado na Figura 5.11. O IRS executa *N*/4 sorteios em cada setor e, desta forma, garante uma melhor exploração da área de busca, pois cada setor (que representa ¼ da área de busca) avaliará ¼ dos blocos candidatos sorteados.

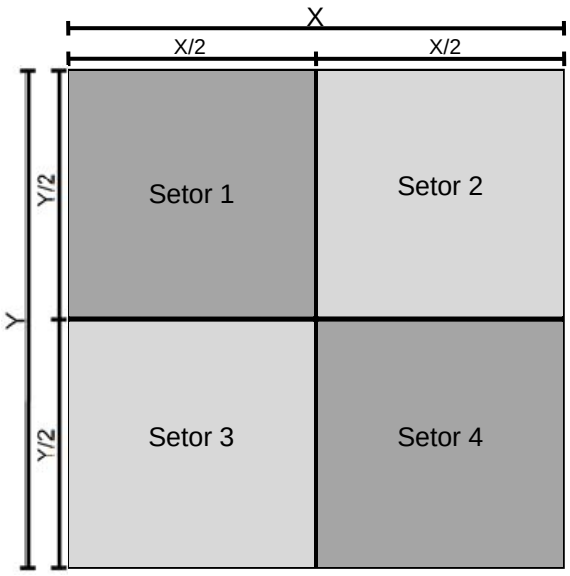


Figura 5.11: Divisão da área de busca em setores

Além disso, o algoritmo IRS dispara um processo iterativo em cada setor, além do processo iterativo central. Deste modo, cinco processos de refinamento iterativo são realizados, de maneira similar ao que acontecia no algoritmo MPDS, mas no IRS as posições iniciais dos refinamentos iterativos de cada setor são escolhidas após uma avaliação aleatória de N/4 blocos em cada setor.

O algoritmo IRS compara os ***N***/4 blocos candidatos sorteados em cada setor e executa um refinamento final iterativo para o bloco candidato de menor SAD, encontrado em cada um dos quatro setores. Além disto, o IRS também realiza a avaliação da região central da área de pesquisa, da mesma forma que o algoritmo RSP. O resultado final do algoritmo IRS é obtido após a comparação entre os melhores resultados obtidos pelos quatro setores e a avaliação da região central da área de pesquisa. A Figura 5.12 apresenta o fluxograma do algoritmo IRS. O fluxograma da Figura 5.12 apresenta as duas principais etapas do algoritmo IRS (avaliação da região central e etapa aleatória) sendo executadas em paralelo. No entanto, a avaliação dos setores é apresentada em modo sequencial, para facilitar a visualização do fluxograma.

É importante salientar que a exploração dos setores pode ser executada em paralelo, pois não existe dependência de dados entre elas, assim como não existem dependências de dados entre a etapa de avaliação central e aleatória. Esta característica do algoritmo IRS é relevante para aplicações multi tarefa em software e também para implementação em hardware.

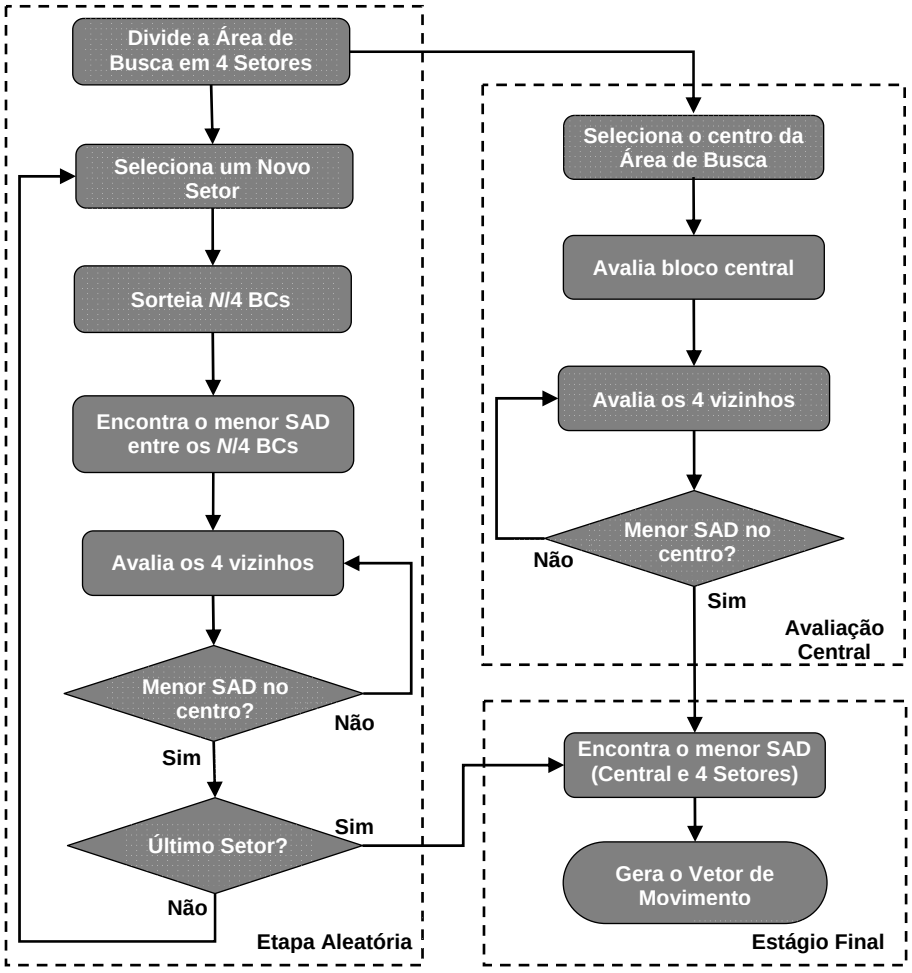


Figura 5.12: Fluxograma do algoritmo IRS

A Figura 5.13 apresenta um exemplo de busca do algoritmo IRS, neste caso, para $N = 16$. Na primeira etapa, o algoritmo IRS avalia o bloco candidato central e os quatro vizinhos imediatamente ao redor (destacados em cinza escuro com bordas tracejadas). O melhor resultado é encontrado para o bloco central (bloco 1), neste caso, nenhuma iteração é realizada e a avaliação da região central da área de pesquisa é encerrada. Na segunda etapa, o algoritmo IRS sorteia e avalia $N/4$ blocos candidatos em cada um dos setores da área de pesquisa (também destacados em cinza escuro). O melhor bloco candidato é encontrado em cada setor (blocos com o número 2) e, então, o algoritmo IRS inicia o refinamento final iterativo, tendo os melhores blocos em cada setor como centro do refinamento. O refinamento final iterativo é aplicado (blocos em cinza claro), mas apenas no setor 2 (de acordo com a Figura 5.13) um dos blocos vizinhos (bloco com número 3) apresenta menor SAD que o bloco central. Neste caso, uma iteração é realizada e mais três blocos candidatos são avaliados (blocos com riscos diagonais), no entanto, o melhor resultado ainda é o bloco central e a busca é encerrada. Por fim, o algoritmo IRS encontra o menor SAD entre o bloco escolhido na avaliação da região central da área de pesquisa e os melhores blocos obtidos em cada setor.

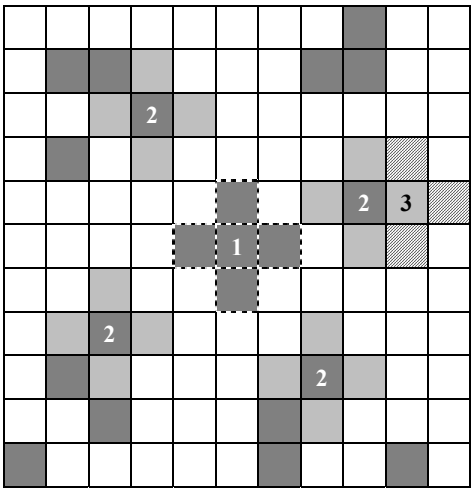


Figura 5.13 – Exemplo de busca do algoritmo IRS

O algoritmo IRS também apresenta custo computacional variável, dependendo da sequência de vídeo processada. As duas etapas do IRS apresentam refinamento final iterativo, e o número de iterações é dependente dos resultados obtidos para os blocos candidatos avaliados.

5.4.3 Algoritmo *Multiple Iterative Random Search* (MIRS)

O algoritmo IRS explora a região central da área de busca com um refinamento iterativo, desta forma é possível explorar a localidade do bloco atual. Isto garante a avaliação dos blocos centrais da área de busca, já que os blocos sorteados na etapa aleatória podem não contemplar esta região da área de busca. A exploração da localidade do bloco atual é importante principalmente para vídeos que apresentam baixa movimentação, tanto de objetos quanto de câmera. Neste cenário, bons blocos candidatos tendem a ser encontrados próximos ao centro da área de busca.

Para garantir uma melhor exploração da localidade do bloco atual, o algoritmo *Multiple Iterative Random Search* (MIRS) aplica o algoritmo DS ao centro da área de busca. Desta forma é possível avaliar esta região de modo mais abrangente, com a aplicação dos dois padrões de busca do algoritmo DS, o *Large Diamond Search Pattern* (LDSP) e o *Small Diamond Search Pattern* (SDSP). Com o algoritmo DS aplicado ao centro, é possível convergir para áreas próximas ao centro, porém mais afastadas em relação às áreas alcançadas pela exploração central do algoritmo IRS. Isto ocorre, pois as iterações do padrão LDSP do algoritmo DS acontecem com saltos de dois pixels, enquanto a iteração do algoritmo IRS é realizada pixel a pixel.

O algoritmo MIRS também utiliza a divisão da área de busca em quatro setores, conforme apresentado na Figura 5.11. Além da exploração da região central da área de busca com a aplicação do algoritmo DS, o algoritmo MIRS também sorteia $N/4$ blocos candidatos em cada setor e aplica um refinamento final iterativo ao melhor bloco candidato em cada setor, dentre os $N/4$ blocos candidatos avaliados. O fluxograma do algoritmo MIRS é apresentado na Figura 5.14. É importante destacar que a etapa aleatória do algoritmo MIRS também é apresentada de modo sequencial no fluxograma da Figura 5.14, no entanto, a exploração aleatória dos setores pode ser executada em paralelo, pois não existe dependência de dados entre elas.

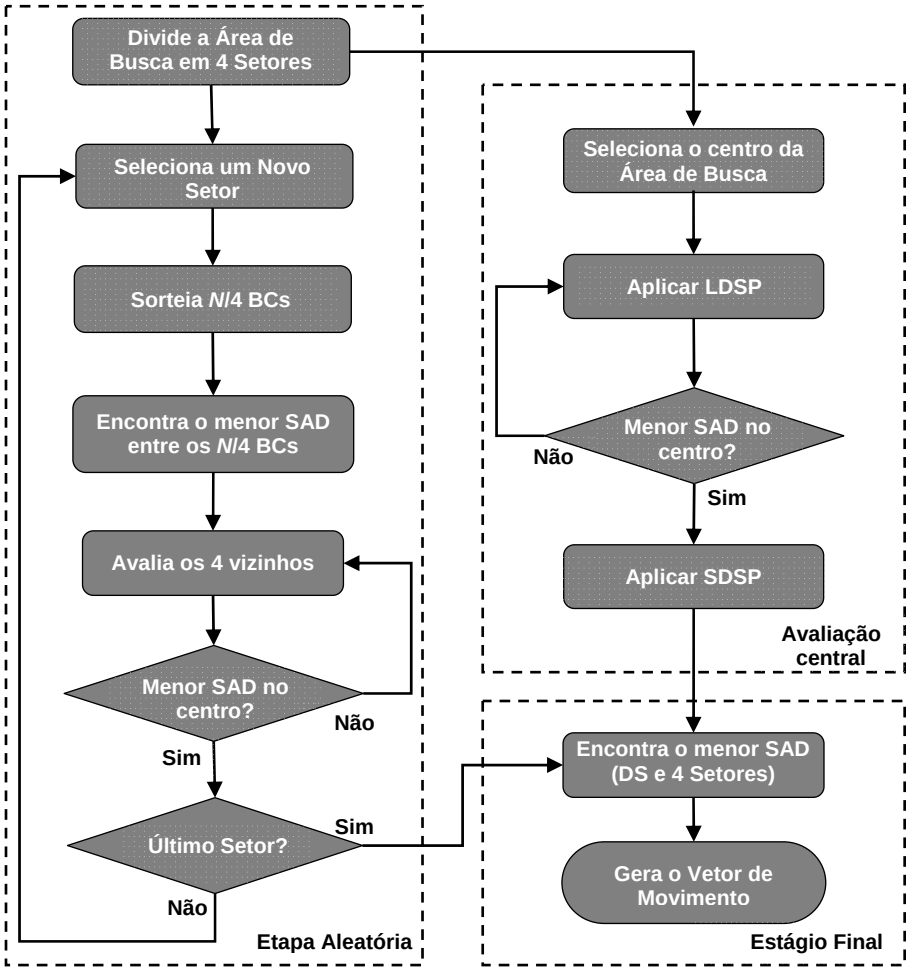


Figura 5.14: Fluxograma do algoritmo MIRS

O desempenho do algoritmo MIRS, em termos de tempo de processamento, pode ser equivalente ao obtido pelo algoritmo DS, visto que esta é a única etapa que apresenta dependência de dados. O custo computacional da etapa de exploração aleatória é diretamente proporcional ao valor de *N* utilizado, no entanto, o cálculo dos *N* blocos candidatos pode ser amplamente paralelizado, reduzindo o tempo de processamento. Além disso, as avaliações iterativas dos setores podem ser executadas em paralelo a execução do algoritmo DS.

O custo computacional do algoritmo MIRS também é variável, devido ao número de iterações do algoritmo DS e também das etapas de refinamento final da exploração aleatória em cada setor, que dependem das sequências de vídeo utilizadas como entrada.

5.4.4 Algoritmo *Random Diamond Search* (RDS)

O algoritmo *Random Diamond Search* (RDS) utiliza os conceitos de exploração de blocos candidatos aleatórios e também a divisão da área de busca em setores, conforme apresentados para os algoritmos IRS e MIRS. No entanto, o algoritmo RDS utiliza um refinamento mais abrangente, com a aplicação do algoritmo DS também aos melhores blocos candidatos obtidos em cada setor na etapa aleatória.

O fluxograma do algoritmo RDS é apresentado na Figura 5.15, onde é possível perceber claramente a possibilidade de exploração do paralelismo entre a exploração da região central e dos setores. No entanto, mais uma vez a possibilidade de paralelismo da

exploração dos setores não é destacada na Figura 5.15, privilegiando a facilidade de visualização do fluxograma.

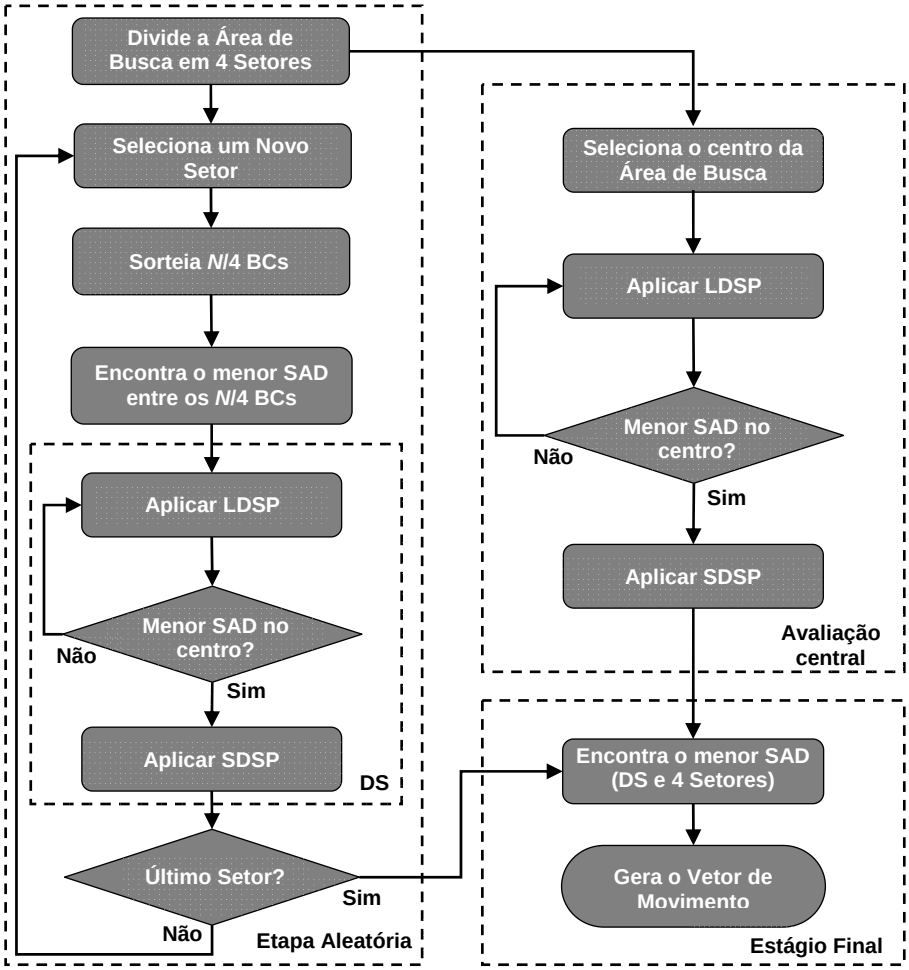


Figura 5.15: Fluxograma do algoritmo RDS

O algoritmo RDS apresenta o maior custo computacional dentre todos os algoritmos aleatórios apresentados, devido à aplicação do algoritmo DS como refinamento na etapa de exploração aleatória. O algoritmo DS avalia 13 blocos candidatos (nove do LDSP e quatro do SDSP) no melhor caso e este é um número muito superior aos quatro blocos candidatos avaliados no refinamento final dos algoritmos RSP, IRS e MIRS.

5.4.5 Exploração do Valor de *N* e do Tamanho da Área de Pesquisa

Todos os algoritmos aleatórios desenvolvidos neste trabalho possuem uma característica em comum, o sorteio de *N* blocos candidatos dentro da área de busca. É possível estimar que, quanto maior o valor de *N*, melhores serão os resultados de qualidade obtidos, uma vez que um número maior de blocos candidatos serão avaliados. Quanto maior o valor de *N*, maior será a probabilidade do algoritmo aleatório encontrar o bloco candidato ótimo, aproximando o seu resultado de qualidade ao obtido pelo algoritmo FS. No entanto, o custo computacional dos algoritmos aleatórios cresce proporcionalmente ao aumento do valor de *N* e para valores muito elevados de *N*, o custo computacional dos algoritmos aleatórios pode se aproximar ao obtido pelo algoritmo FS.

Outro fator importante para a definição dos resultados de qualidade e custo computacional dos algoritmos de ME (não apenas os aleatórios) é o tamanho da área de

busca. Em geral, áreas de busca maiores tendem a melhorar os resultados de qualidade, no entanto, tendem a aumentar o custo computacional. Para os algoritmos aleatórios, esta definição do tamanho da área de busca deve levar também em consideração o valor de *N*. O aumento da área de busca possibilita um incremento significativo no número de blocos candidatos disponíveis, diminuindo o percentual do número de blocos candidatos avaliados na etapa aleatória em relação ao número total de blocos candidatos. Com isto, o aumento do tamanho da área de busca pode não beneficiar os resultados de qualidade dos algoritmos aleatórios sem um aumento significativo do valor de *N* e, consequentemente, aumentando o custo computacional do algoritmo.

Para determinar o valor ideal de *N* na relação entre qualidade e custo computacional para cada algoritmo aleatório, uma bateria de avaliações foi realizada considerando a variação do valor de *N* utilizado e também a variação do tamanho da área de busca. A Figura 5.16 apresenta os gráficos com as curvas de ganho PSNR para os algoritmos (a) RSP, (b) IRS, (c) MIRS e (d) RDS. Cada curva dos gráficos da Figura 5.16 representa os resultados de PSNR para um dado valor de *N*, considerando a variação do tamanho da área de busca. Os valores de *N* utilizados nas avaliações foram: *N* = 4, 8, 12, 16, 24 e 32.

Os algoritmos foram avaliados para os seis valores de *N* apresentados acima, em todas as seguintes áreas de busca:

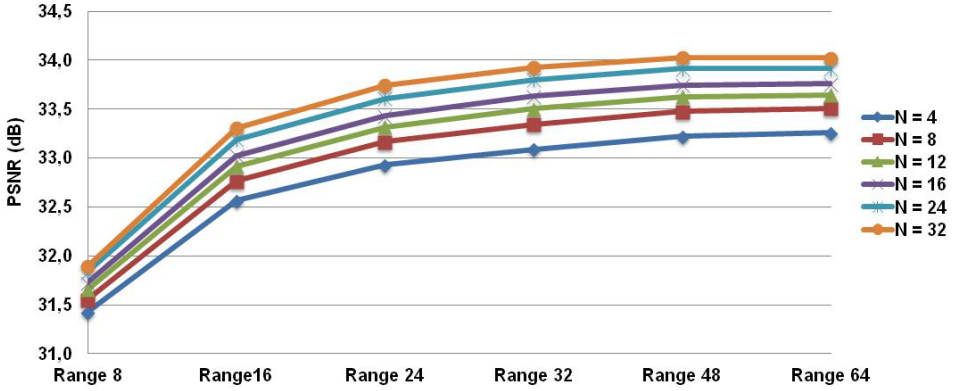
- *Range* [-8, +8] - Área de busca total de 32x32 pixels
- *Range* [-16, +16] - Área de busca total de 48x48 pixels
- *Range* [-24, +24] - Área de busca total de 64x64 pixels
- *Range* [-32, +32] - Área de busca total de 80x80 pixels
- *Range* [-48, +48] - Área de busca total de 112x112 pixels
- *Range* [-64, +64] - Área de busca total de 144x144 pixels

As curvas da Figura 5.16 representam o resultado médio de PSNR obtido para as 10 sequências de teste HD 1080p apresentadas anteriormente. O tamanho de bloco utilizado foi de 16x16 pixels.

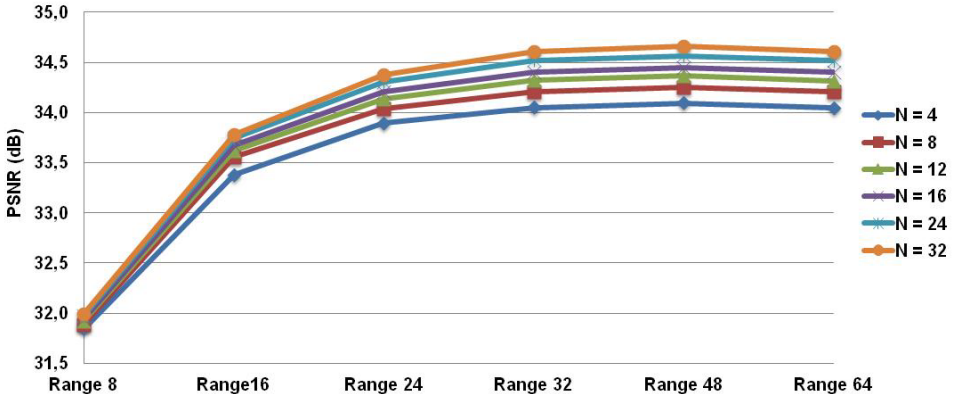
O comportamento das curvas apresentadas na Figura 5.16 é bastante similar para todos os algoritmos, sendo que o resultado de PSNR cresce com o aumento da área de busca. Este comportamento ocorre até a área de *range* [-48, +48]. Para a área de [-64, +64] os algoritmos apresentaram resultados médios ligeiramente inferiores. A única exceção foi o algoritmo RSP, que obteve seus melhores resultados para a área de [-64, +64] (exceto para *N* = 32), com ganhos muito pequenos de cerca de 0,02dB.

O uso de valores de *N* maiores resulta em ganhos nos resultados de PSNR obtidos, conforme já era esperado. É possível observar através da Figura 5.16 que independentemente do tamanho da área de busca utilizada, quanto maior o valor de *N*, maiores são os resultados de PSNR obtidos.

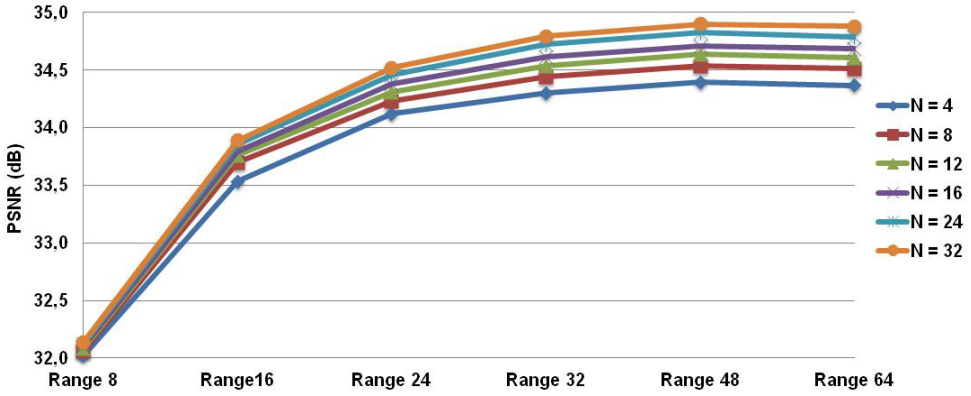
Os algoritmos aleatórios foram desenvolvidos de modo incremental, sendo que cada novo algoritmo apresenta pequenas melhorias no processo de busca em relação à versão anterior. Analisando a Figura 5.16 é possível perceber que estas melhorias resultam em pequenos ganhos incrementais entre os algoritmos RSP, IRS, MIRS e RDS, sendo que o algoritmo RDS apresenta os melhores resultados de qualidade para um mesmo valor de *N* e de área de busca.



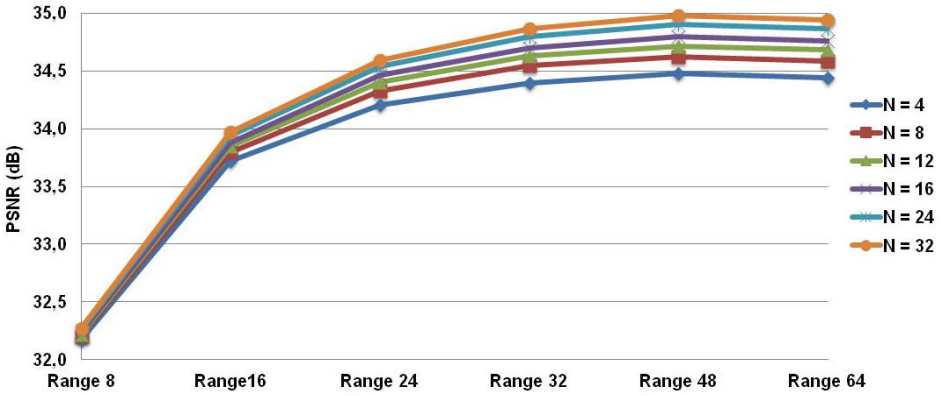
(a) RSP



(b) IRS



(c) MIRS



(d) RDS

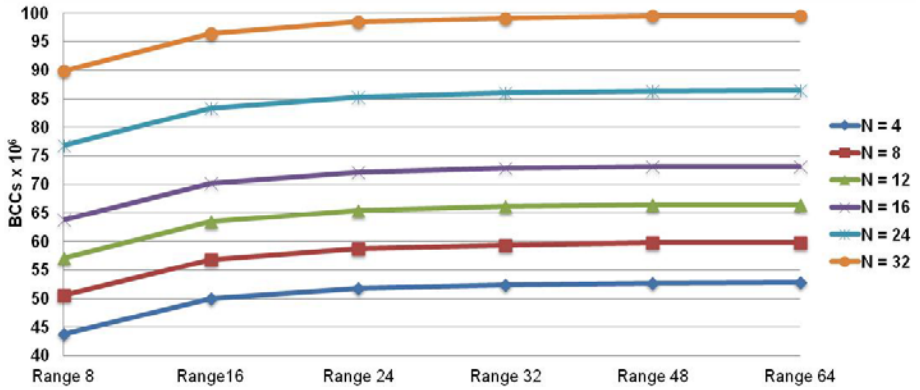
Figura 5.16: Avaliação de qualidade para a variação do valor de *N* e tamanho da área de busca para os algoritmos RSP(a), IRS(b), MIRS(c) e RDS(d).

Considerando apenas os resultados de qualidade, o valor ideal de *N* para todos os algoritmos seria *N* = 32, e a área de busca seria de [-64, +64] para o algoritmo RSP e [-48, +48] para os demais. No entanto, é necessária a avaliação do custo computacional associado a cada valor de *N* em cada área de busca, para a definição do valor ideal de *N* em cada algoritmo que represente a melhor relação qualidade versus custo computacional.

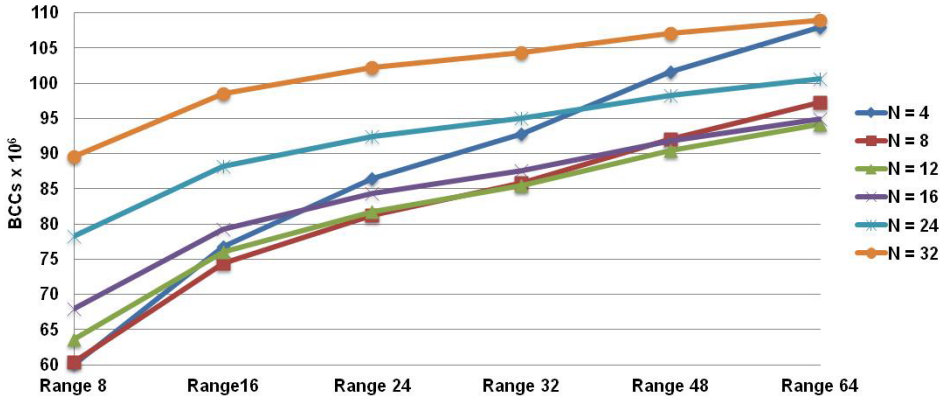
A Figura 5.17 apresenta os resultados de custo computacional para os algoritmos (a) RSP, (b) IRS, (c) MIRS e (d) RDS, mensurados em número de blocos candidatos comparados (BCC), para a variação do valor de *N* e do tamanho da área de busca. Os valores de *N* e das áreas de busca são os mesmos apresentados anteriormente para a avaliação de qualidade. Ao contrário do apresentado na Figura 5.16, as curvas da Figura 5.17(a), (b), (c) e (d) possuem características diferentes. O algoritmo RSP (Figura 5.17(a)) apresenta curvas regulares, com um leve crescimento no número de BCCs com o aumento do tamanho da área de pesquisa. O número de BCCs também está diretamente relacionado ao valor de *N* utilizado, onde valores maiores de *N* resultam em números maiores de BCCs. Isto ocorre, pois o algoritmo RSP aplica o refinamento final iterativo apenas ao melhor bloco candidato, escolhido na etapa aleatória. Além disto, o refinamento final do algoritmo RSP avalia apenas três novos blocos candidatos a cada iteração, desta forma, o valor de *N* utilizado determina a maior parte do número de BCCs.

As curvas para os algoritmos IRS e MIRS são semelhantes, no entanto, para o algoritmo IRS o número de BCCs varia de $60x10^6$ a aproximadamente $110x10^6$, enquanto que o algoritmo MIRS apresenta custos maiores, com resultados oscilando entre 90 $x10^6$ e $145x10^6$. Esta diferença ocorre devido ao método de exploração da região central utilizado pelo algoritmo MIRS, que implementa o algoritmo DS ao invés do SDSP utilizado pelo algoritmo IRS. O custo computacional dos algoritmos IRS e MIRS não está diretamente relacionado com o valor de *N* utilizado, ou seja, valores menores de *N* não significam necessariamente custos menores.

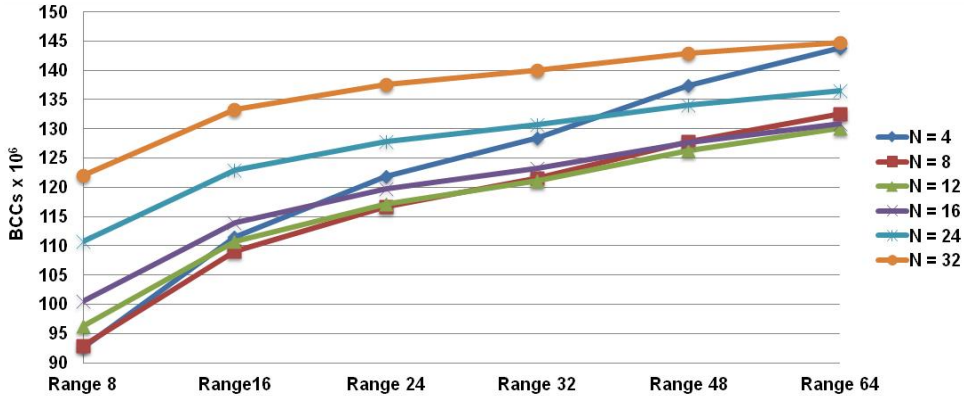
O menor número de BCCs encontrado para os algoritmos IRS e MIRS é obtido para *N* = 4, com área de busca [-8, +8]. No entanto, para a área de busca [-16, +16] o número de BCCs para *N* = 4 já é superior ao apresentado para *N* = 8 e *N* = 12. O crescimento no número de BCCs para *N* = 4 com o aumento do tamanho da área de busca é superior ao apresentado por valores maiores de *N*. Para a área de busca de [-48, +48] o número de BCCs para *N* = 4 já é superior a todos os demais valores, exceto *N* = 32, sendo que para a área de busca de [-64, +64] os resultados são praticamente iguais. Isto ocorre devido ao processo de refinamento final iterativo dos algoritmos IRS e MIRS, realizado para o melhor bloco candidato encontrado em cada setor da área de busca. Valores maiores de *N* aumentam as probabilidades da escolha de bons blocos candidatos, desta forma, o número de iterações tende a ser menor. Esta característica se torna mais evidente para áreas de busca maiores, onde valores baixos de *N* possuem uma representatividade muito baixa do universo total de blocos candidatos, aumentando significativamente o número de iterações necessárias para a obtenção de um bom bloco candidato e, consequentemente, aumentando o custo computacional dos algoritmos. Analisando a área de busca de [-48, +48], onde os melhores resultados de qualidade foram obtidos para estes dois algoritmos, o menor custo computacional foi obtido para *N* = 12. O resultado obtido para *N* = 16 foi muito próximo, apenas 1,4% superior para o algoritmo IRS e 1% superior para o algoritmo MIRS.



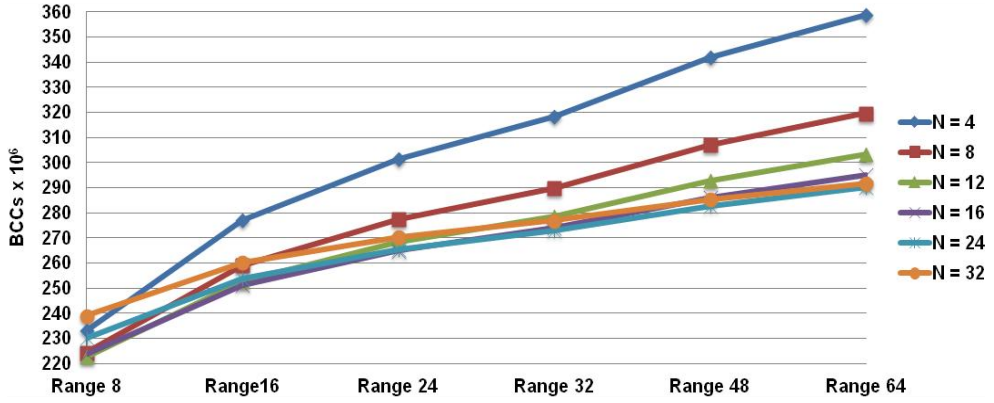
(a) RSP



(b) IRS



(c) MIRS



(c) RDS

Figura 5.17: Avaliação de custo computacional para a variação do valor de *N* e tamanho da área de busca para os algoritmos RSP(a), IRS(b), MIRS(c) e RDS(d).

O algoritmo RDS apresenta o maior custo computacional dentre todos os algoritmos aleatórios desenvolvidos, com número de BCCs variando entre $223x10^6$ e $359x10^6$, aproximadamente. Este resultado já era esperado devido ao processo de refinamento final que aplica o algoritmo DS ao melhor bloco candidato obtida na etapa aleatória de cada setor. Devido ao custo mais elevado deste refinamento, os maiores valores de BCCs são obtidos para os menores valores de *N*. O custo computacional do algoritmo RDS para *N* = 4 é o mais elevado para todas as áreas de busca avaliadas, exceto para [-8, +8], ainda assim, o custo nesta área é o segundo maior. Devido ao alto custo associado ao processo de refinamento final, o algoritmo RDS obtêm seus melhores resultados de custo computacional para os maiores valores de *N*. Considerando novamente a área de busca de [-48, +48], o menor número de BCCs foi obtido para *N* = 24 e, em segundo lugar, para *N* = 32, com um custo apenas 0,9% superior.

Analisando os resultados de qualidade e custo computacional obtidos com a variação do valor de *N* e do tamanho da área de pesquisa é possível determinar os valores que correspondem a melhor relação qualidade versus custo computacional para os algoritmos aleatórios. Os melhores resultados de qualidade foram apresentados para a área de busca de [-48, +48], com exceção do algoritmo RSP. No entanto, os ganhos de qualidade apresentados com a área de busca de [-64, +64] são insignificantes e vêm acompanhados de uma pequena elevação no custo computacional. Desta forma, a área de busca de [-48, +48] será utilizada para a geração dos resultados finais dos algoritmos aleatórios neste trabalho.

A Tabela 5.7 apresenta a evolução dos resultados de qualidade e de número de BCCs para o algoritmo RSP com o aumento do valor de *N*. O custo computacional do algoritmo RSP para a área de busca de [-48, +48] está diretamente ligado ao valor de *N* utilizado, sendo que tanto os resultados de qualidade quanto o número de BCCs crescem com o aumento do valor de *N*. O aumento no número de BCCs é da ordem de 13% para *N* = 8, 12 e 16, respectivamente, no entanto, a partir de *N* = 24 este aumento passa para cerca de 25%. O algoritmo RSP com *N* = 16 apresenta um ganho de 0,52dB em relação a *N* = 4, com um aumento no número de BCCs de aproximadamente 38,7%. Para *N* = 24, o ganho é de 0,17dB em relação ao resultado para *N* = 16, com um aumento no número de BCCs de 25,1%. Desta forma, o valor de *N* que representa a melhor relação qualidade versus custo computacional para o algoritmo RSP é *N* = 16.

Tabela 5.7: Resultados de qualidade e custo computacional para o algoritmo RSP com a variação do valor de *N* para a área de busca de [-48, +48]

Valor de *N*	**PSNR (dB)**	**Nº BCCs ($x10^6$)**
4	33,26	52,80
8	+0,25	+13,2%
12	+0,40	+26%
16	+0,52	+38,7%
24	+0,69	+63,8%
32	+0,81	+88,7%

Os algoritmos IRS e MIRS apresentam comportamentos semelhantes, tanto em qualidade quando em custo computacional, para a variação do valor de *N*. A Tabela 5.18 apresenta a evolução dos resultados de qualidade e custo computacional para os algoritmos IRS e MIRS com a variação do valor de *N*, para a área de busca de [-48, +48]. Para ambos os algoritmos o menor custo computacional é obtido para *N* = 12, no entanto, para *N* = 16 o terceiro melhor resultado de qualidade é obtido, com o segundo menor custo computacional. Os resultados de qualidade obtidos para *N* = 24 são inferiores apenas aos obtidos por *N* = 32, no entanto, o custo computacional ainda é inferior ao obtido por *N* = 4. O aumento no custo computacional apresentado para *N* = 16 em relação ao resultado obtido para *N* = 12 é muito pequeno para os dois algoritmos. O incremento no custo é levemente maior na comparação entre *N* = 16 e *N* =24, no entanto, ambos os algoritmos atingem ganhos de 0,12dB. A definição do valor ideal de *N* para os algoritmos IRS e MIRS é uma tarefa complicada, mas é possível afirmar que os melhores resultados são obtidos para *N* =16 e *N* = 24. Os resultados obtidos para *N* = 16 apresentam pequena vantagem, pois apresentam o custo computacional mais próximo ao valor mínimo, ainda assim, apresentam ganhos de 0,35dB e 0,31dB para os algoritmos IRS e MIRS, respectivamente.

Tabela 5.8: Resultados de qualidade e custo computacional para os algoritmos IRS e MIRS com a variação do valor de *N* para a área de busca de [-48, +48]

Valor de *N*	**IRS**		**MIRS**	
	PSNR (dB)	**Nº BCCs ($x10^6$)**	**PSNR (dB)**	**Nº BCCs ($x10^6$)**
4	34,09	101,63	34,40	137,45
8	+0,16	-9,4%	+0,14	-7%
12	+0,27	-10,9%	+0,24	-8,1%
16	+0,35	-9,6%	+0,31	-7,1%
24	+0,47	-3,3%	+0,43	-2,4%
32	+0,57	+5,3%	+0,50	4%

A Tabela 5.9 apresenta os resultados de qualidade e custo computacional para o algoritmo RDS com a variação do valor de *N*. O algoritmo RDS apresenta uma redução mais significativa do custo computacional com o aumento do valor de *N*, ao mesmo tempo em que apresenta ganhos regulares em qualidade. O maior custo computacional é obtido para *N* = 4 e o menor é obtido para *N* = 24. O segundo melhor resultado, tanto de qualidade quanto de custo computacional, é obtido para *N* = 24. No entanto, o custo computacional para *N* = 32 é apenas 0,9% superior ao obtido para *N* = 24, com um incremento de 0,08dB no PSNR. Desta forma, o melhor valor de *N* para o algoritmo RDS é 32, pois é possível atingir o melhor resultado de qualidade e ainda o segundo menor custo computacional.

A análise da variação do tamanho da área de busca e do valor de *N* para os algoritmos aleatórios demonstrou que a melhor relação qualidade versus custo computacional é obtida na área de busca de [-48, +48]. Analisando a variação do valor de *N* nesta área de busca, os algoritmos RSP, IRS e MIRS obtiveram seus melhores resultados para *N* =16, enquanto que para o algoritmo RDS os melhores resultados

foram obtidos para $N = 32$. Estas serão as configurações utilizadas para a avaliação dos algoritmos aleatórios na próxima seção.

Tabela 5.9: Resultados de qualidade e custo computacional para o algoritmo RDS com a variação do valor de *N* para a área de busca de [-48, +48]

Valor de *N*	PSNR (dB)	Nº BCCs (x10^6)
4	34,48	341,96
8	+0,14	-10,2%
12	+0,24	-14,4%
16	+0,32	-16,3%
24	+0,42	-17,3%
32	+0,50	-16,6%

5.4.6 Resultados para os Algoritmos Aleatórios

Os resultados de qualidade e custo computacional para todos os algoritmos aleatórios serão apresentados em conjunto. Desta forma, é possível avaliar mais facilmente a evolução dos resultados de qualidade e também o acréscimo no custo computacional de cada nova versão. Assim como os resultados apresentados para os demais algoritmos, os algoritmos aleatórios foram aplicados as 10 sequências de teste HD 1080p já apresentadas. Este capítulo apresenta apenas os resultados para os algoritmos aleatórios, sendo que os resultados comparativos para todos os algoritmos desenvolvidos, bem como para algoritmos da literatura, são apresentado no capítulo 5.6.

Uma característica atípica dos algoritmos aleatórios é a possibilidade da obtenção de resultados diferentes quando executados sobre o mesmo vídeo. Isto pode ocorrer devido à etapa aleatória, que sorteia *N* blocos candidatos da área de busca. Esta etapa está presente em todos os algoritmos aleatórios desenvolvidos. Para medir esta variação, o desvio padrão (FONSECA, 1996) foi calculado para a etapa aleatória dos algoritmos.

O desvio padrão da etapa aleatória foi calculado considerando 10 execuções do algoritmo para cada sequencia de teste, considerando blocos de tamanho 8x8, 16x16 e 32x32, considerando o valor de $N = 16$. Nesta avaliação os *N* blocos candidatos podem ser sorteados livremente na área de busca, sem a divisão por setores. Isto aumenta a aleatoriedade da busca e, consequentemente, a variabilidade dos resultados. A Tabela 5.10 apresenta os resultados desta avaliação. As diferenças apresentadas foram praticamente insignificantes, com um pequeno aumento para blocos maiores. Em nenhum dos casos, esta diferença média foi superior a 0,08dB. Estes resultados demonstram que o desvio médio na execução da etapa aleatória é insignificante para todos os tamanhos de bloco avaliados. Este pequeno desvio pode ainda ser atenuado nas demais etapas dos algoritmos aleatórios. Desta forma, a variação nos resultados gerada pela aleatoriedade dos blocos candidatos sorteados não será considerada nos resultados finais dos algoritmos RSP, IRS, MIRS e RDS.

Tabela 5.10: Resultados médios e de desvio padrão (DP) de PSNR para a etapa aleatória

Vídeo	8x8		16x16		32x32	
	PSNR (dB)	DP	PSNR (dB)	DP	PSNR (dB)	DP
blue _sky	29,84	0,048	28,21	0,062	27,05	0,085
man_in_car	22,53	0,048	20,35	0,183	18,96	0,157
pedestrian	28,92	0,084	26,6	0,095	25,05	0,095
riverber	26,38	0,017	24,67	0,009	22,69	0,016
rolling_tomatoes	24,77	0,080	22,71	0,106	21,09	0,095
rush_hour	30,35	0,058	28,77	0,069	27,36	0,050
station2	28,99	0,039	27,11	0,039	25,55	0,041
sun_flower	26,04	0,149	23,82	0,197	22,0	0,255
tractor	31,22	0,068	29,64	0,106	28,11	0,082
traffic	30,77	0,058	28,81	0,100	27,3	0,113
Média	**27,98**	0,065	**26,07**	0,097	**24,52**	0,099

Todos os algoritmos foram avaliados para blocos de tamanho 8x8, 16x6 e 32x32 e utilizam a área de busca de [-48, +48] pixels. O valor de N utilizado foi 16 para os algoritmos RSP, IRS e MIRS, para o algoritmo RDS foi utilizado $N = 32$. A Figura 5.18 apresenta os resultados de PSNR para os algoritmos RSP, IRS, MIRS e RDS, geradas a partir dos resultados obtidos para as 10 sequências de teste HD 1080p para blocos de tamanho 16x16 pixels. De modo geral, o comportamento dos algoritmos aleatórios é similar ao de outros algoritmos de EM, sendo que vídeos com maior movimentação, ou maior quantidade de texturas, como riverbed e traffic, por exemplo, apresentam os menores resultados de qualidade para todos os algoritmos. Os melhores resultados são obtidos para sequências de baixa movimentação, como man_in_car, por exemplo. Os algoritmos RSP, IRS, MIRS e RDS apresentam ganhos incrementais para todos os vídeos avaliados, no entanto, existe uma variação significativa nos ganhos obtidos em cada vídeo.

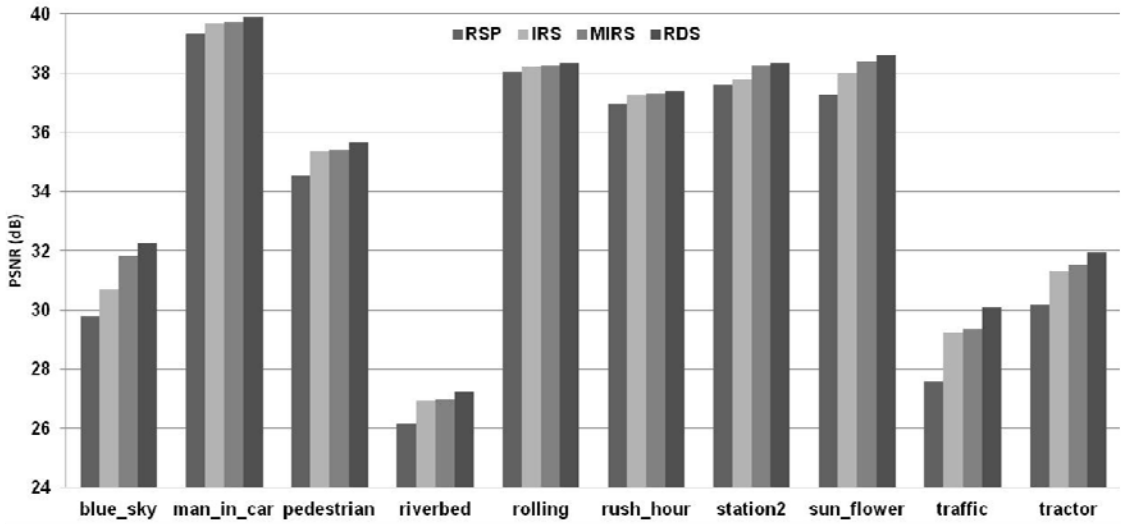


Figura 5.18: PSNR para os algoritmos RSP, IRS, MIRS e RDS com blocos 16x16

O algoritmo IRS apresenta um ganho médio de aproximadamente 0,7dB em relação ao algoritmo RSP, que apresenta um resultado médio de PSNR de 33,74dB. No entanto, o ganho do algoritmo IRS chega a 1,65dB para a sequência de vídeo traffic, que apresenta grande movimentação. Para outras sequências de maior movimentação, como blue_sky, pedestrian, riverbed e tractor os ganhos obtidos foram em torno de 1dB. Os menores ganhos (aproximadamente 0,2dB) foram obtidos para as sequências de menor movimentação, como rolling e station2. Este ganho mais significativo em vídeos de maior movimentação pode ser explicado devido à exploração mais eficiente da busca aleatória do algoritmo IRS. A divisão da área de busca em setores garante uma exploração mais homogenea de toda área de busca, garantindo que $N/4$ blocos candidatos sorteados em cada setor. Além disto, o refinamento iterativo aplicado ao melhor bloco de cada setor possibilita a convergência para regiões mais remotas do centro da área de busca e em diferentes direções, gerando melhores resultados especialmente para vídeos com maior movimentação, onde os blocos candidatos ótimos tendem a ser encontrados longe do centro da área de busca. Os algoritmos RSP e IRS possuem a mesma etapa de exploração da região central da área de busca, com isto, os seus resultados para vídeos de baixa movimentação, onde os vetores ótimos tendem a ser encontrados perto do centro da área de busca, são similares.

O algoritmo MIRS realiza um processo mais eficiente de exploração da região central da área de busca, com a aplicação do algoritmo DS. Com isso, o algoritmo MIRS obtém um ganho médio de 0,26dB em relação ao algoritmo IRS. Este ganho é obtido principalmente para sequências de baixa movimentação, como station2 e sun_flower, com ganhos de 0,46dB e 0,4dB, respectivamente. No entanto, o maior ganho foi obtido para a sequência blue_sky (aproximadamente 1,13dB), onde existe grande movimentação, porém, o movimento de câmera é lento e existe pouca variação de texturas. Nestas condições, a aplicação do algoritmo DS ao centro a área de busca pode atingir melhores blocos candidatos próximos ao centro, mas não tão próximos para serem encontrados pelo simples refinamento iterativo dos algoritmos RSP e IRS.

O algoritmo RDS apresenta um ganho médio de 0,27dB em relação ao algoritmo MIRS. O processo de refinamento final da etapa aleatória, aplicado a cada setor da área de busca que utiliza o algoritmo DS, possibilita ganhos mais significativos para sequências de maior movimentação. O maior ganho do algoritmo RDS em relação ao algoritmo MIRS (0,71dB) foi obtido para a sequência traffic. Para sequências com baixa movimentação, os ganhos são discretos, sendo o menor deles obtido para a sequência station2, com apenas 0,08dB.

Os resultados de qualidade dos algoritmos aleatórios apresentam ganhos médios incrementais e sem grande variação, exceto para determinadas sequências. No entanto, os resultados de custo computacional apresentam grandes variações. A Figura 5.19 apresenta os resultados de custo computacional, mensurados em número de BCCs, para os algoritmos RSP, IRS, MIRS e RDS para a área de busca [-48, +48] e blocos de 16x16 pixels. Os resultados de custo computacional não estão diretamente ligados aos resultados de qualidade obtidos, ou seja, os maiores resultados de custo nem sempre são encontrados para as sequências onde o maior ganho em qualidade foi obtido.

O algoritmo RSP apresenta os menores resultados de custo computacional, como já era esperado, com um resultado médio de $73{,}16 \times 10^6$ BCCs e com valores variando entre $53{,}12 \times 10^6$ para a sequência rolling e $100{,}8 \times 10^6$ para a sequência traffic. O algoritmo IRS apresenta um acréscimo médio de 26% sobre o custo computacional do algoritmo RSP. No entanto, o maior acréscimo no custo computacional foi obtido para a sequência

rush_hour (58,9%), onde o IRS obteve um dos menores ganhos de qualidade sobre o algoritmo RSP, sendo apenas 0,29dB superior. O acréscimo no custo computacional para a sequência traffic, onde o IRS obteve o maior ganho de qualidade em relação ao algoritmo RSP, é de apenas 10%. Isto ocorre, pois a etapa aleatória do algoritmo IRS avalia um número maior de blocos candidatos devido ao refinamento iterativo aplicado a todos os setores. Desta forma, é possível obter maiores ganhos em qualidade para vídeos de alta movimentação, no entanto, para vídeos de baixa movimentação, o bloco candidato escolhido é obtido geralmente pela etapa de avaliação da região central da área de busca, tendo em vista que para este tipo de vídeo a maior parte dos blocos candidatos escolhidos pelos algoritmos estão próximos ao centro da área de busca. Nestes casos, muitos blocos candidatos são comparados nos setores, porém, poucos deles são escolhidos como melhores candidatos, aumentando significativamente o custo computacional sem ganhos significativos me qualidade.

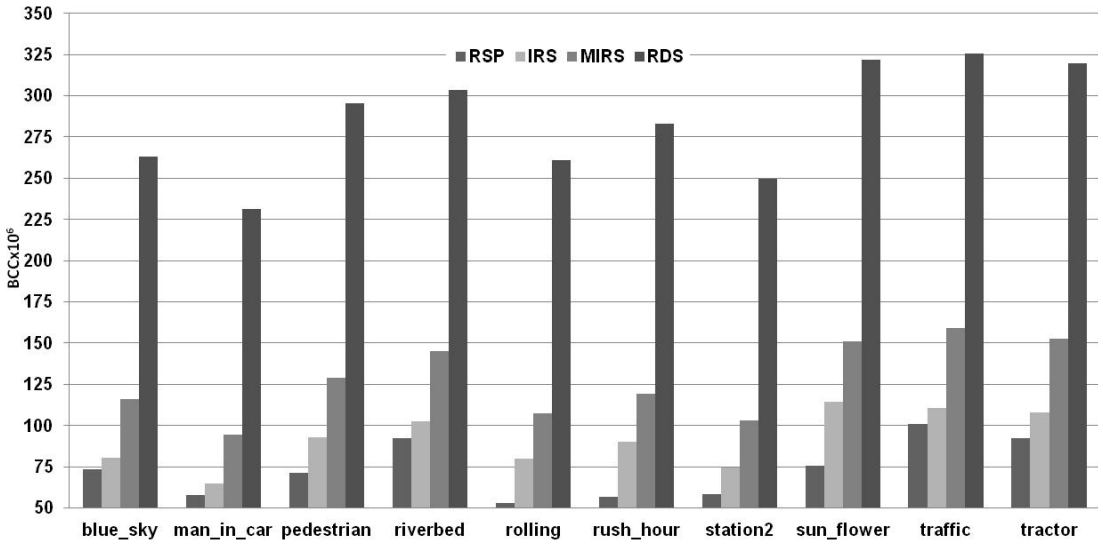


Figura 5.19: Número de BCCs para os algoritmos RSP, IRS, MIRS e RDS com blocos 16x16

Os resultados de número de BCCs utilizados pelo algoritmo MIRS são, em média, 39% superiores aos obtidos pelo algoritmo IRS. Este acréscimo não possui grandes oscilações em todas as 10 sequências avaliadas. A aplicação do algoritmo DS ao centro da área de busca causa um acréscimo mínimo de 32% no custo computacional do algoritmo MIRS, quando comparado ao algoritmo IRS. Dependendo da sequência de vídeo, o número de iterações do algoritmo DS varia, podendo gerar um aumento de 45%. Mais uma vez, os resultados de custo computacional não estão diretamente ligados aos ganhos de qualidade obtidos. O maior acréscimo de custo computacional apresentado pelo algoritmo MIRS em relação ao algoritmo IRS foi obtido para a sequência man_in_car (45,3%), onde o ganho de qualidade foi de apenas 0,05dB. O menor acréscimo no custo computacional foi obtido para a sequência sun_flower (31,9%), onde o algoritmo MIRS obteve um dos maiores ganhos de qualidade em relação ao algoritmo IRS.

O algoritmo RDS apresenta o maior custo computacional dentre todos os algoritmos aleatórios, com um acréscimo médio de 123% no custo computacional em relação ao algoritmo MIRS. O menor acréscimo no custo computacional foi obtido para a sequência traffic (105%), onde o RDS obteve o maior ganho em relação ao algoritmo

MIRS. É importante salientar que o algoritmo RDS foi avaliado utilizando N = 32, o dobro do valor utilizado para o algoritmo MIRS, pois o custo computacional para o algoritmo RDS nesta configuração é inferior ao obtido para N =16, além de apresentar ganhos em qualidade.

A Tabela 5.11 apresenta os resultados completos para as avaliações dos algoritmos aleatórios. Os resultados médios de qualidade e custo computacional para todos os algoritmos aleatórios são apresentados para os três tamanhos de bloco utilizados. Os resultados de qualidade são apresentados através do ganho PSNR e do percentual de redução do resíduo (PRR). Os resultados de custo computacional são apresentados em número de blocos candidatos comparados (BCC) e também número de comparações (Comp.) Estes resultados foram gerados para a área de busca de [-48, +48] pixels, com o valor de N ótimo definido na seção anterior para cada algoritmo. Os resultados médios de qualidade demonstram uma evolução muito similar para todos os tamanhos de bloco.

Tabela 5.11: Resultados de qualidade e custo computacional para os algoritmos aleatórios em diferentes tamanhos de bloco

Bloco	Parâmetro	RSP	IRS	MIRS	RDS
8x8	**PSNR (dB)**	34,8	35,78	36,12	36,55
	PRR (%)	62,96	65,98	66,98	68,23
	Nº de BCCs (x10^9)	0,26	0,29	0,43	1
	Nº de Comp. (x10^9)	16,83	18,94	27,97	63,75
16x16	**PSNR (dB)**	33,74	34,45	34,72	34,98
	PRR (%)	58,14	60,46	61,21	62,04
	Nº de BCCs (x10^6)	0,07	0,09	0,13	0,28
	Nº de Comp. (x10^9)	18,73	23,5	33,1	73,03
32x32	**PSNR (dB)**	32,82	33,23	33,35	33,48
	PRR (%)	53,92	55,47	55,92	56,37
	Nº de BCCs (x10^6)	0,02	0,03	0,04	0,08
	Nº de Comp. (x10^9)	20	29,3	39,33	84,69

Todos os algoritmos apresentam perdas de qualidade com o aumento do tamanho dos blocos. Estas perdas foram da ordem de 1dB com o aumento do bloco de 8x8 para 16x16 e de 16x16 para 32x32, para o algoritmo RSP. As perdas são maiores para os algoritmos mais sofisticados, chegando a 1,5dB para o algoritmo RDS. O ganho apresentado entre os algoritmos também diminui com o aumento do tamanho do bloco. Para blocos 8x8, o ganho do algoritmo IRS sobre o algoritmo RSP é de

aproximadamente 1dB, no entanto, para blocos 16x16 este ganho cai para 0,71dB, chegando a 0,41 para bloco 32x32. Esta característica se repete (em proporções diferentes) na comparação entre os ganhos do algoritmo MIRS sobre o algoritmo IRS e os ganhos do algoritmo RDS sobre o algoritmo MIRS.

Os resultados de PRR para os algoritmos aleatórios possuem características similares aos obtidos com o PSNR com a variação do tamanho de bloco, no entanto, os ganhos em PSNR obtidos são mais expressivos. O algoritmo IRS obteve um aumento no PRR de aproximadamente 3% em relação ao algoritmo RSP, enquanto que o ganho em PSNR foi da ordem de 1dB. Considerando a escala logarítmica do PSNR, o ganho em qualidade torna-se mais significativo.

Os resultados de custo computacional entre os algoritmos aleatórios variam consideravelmente entre si e também com a variação do tamanho de bloco. Com o aumento do tamanho do bloco, o número de BCCs diminui, no entanto, o número de comparações aumenta. Este aumento no número de comparações ocorre devido às etapas de refinamento iterativo dos algoritmos aleatórios, que tendem a aumentar o número de iterações quando operam com blocos de tamanho maior. Com isto, o número de BCCs diminui, porém, não na mesma proporção do aumento no número de pixels dos blocos. Como discutido no capítulo 4 (Figura 4.7), quando o tamanho do bloco é aumentado, o número de bons blocos candidatos diminui, desta forma, os algoritmos tendem a aumentar o número de iterações em busca de melhores blocos candidatos. No entanto, o incremento no número de BCCs entre os algoritmos aleatórios cresce com o aumento do tamanho do bloco utilizado. O acréscimo no número de BCCs do algoritmo IRS sobre o algoritmo RSP é de 12,5% para blocos 8x8, 26% para blocos 16x16 e 46,5% para blocos 32x32. Esta característica se repete para os demais algoritmos. Mais uma vez, o aumento no número de BCCs entre os algoritmos aleatórios não é proporcional aos ganhos de qualidade obtidos, que são menores para os blocos maiores.

Os algoritmos aleatórios também foram avaliados com o uso da subamostragem de pixel. A Tabela 5.12 apresenta os resultados de qualidade e custo computacional para os algoritmos aleatórios com subamostragem. O nível de subamostragem de pixel utilizado para cada tamanho de bloco foi o mesmo definido no capítulo 4, sendo 2:1 para blocos 8x8, 4:1 para blocos 16x16 e 8:1 para blocos 32x32.

Os resultados apresentados na Tabela 5.12 demonstram que os algoritmos aleatórios apresentam perdas em qualidade similares aos demais algoritmos avaliados neste trabalho quando utilizando subamostragem de pixel. As perdas em qualidade dos algoritmos RSP e IRS com o uso de subamostragem de pixel são da ordem de 0,25dB para blocos 8x8 e aumentam para aproximadamente 0,4dB para blocos 16x16 e 0,6dB para blocos 32x32. Para os algoritmos MIRS e RDS as perdas são ligeiramente menores. As perdas de qualidade do algoritmo RDS, por exemplo, para blocos 8x8 com subamostragem de pixel são da ordem de 0,15dB, para blocos 16x16 as perdas são de aproximadamente 0,23dB e de 0,37dB para blocos 32x32. Isto ocorre devido ao grande número de BCCs que é muito maior nestes algoritmos. Desta forma, estes algoritmos são capazes de reduzir as perdas causadas pelo uso da subamostragem de pixel através da análise de um número maior de blocos candidatos.

Tabela 5.12: Resultados de qualidade e custo computacional para os algoritmos aleatórios com subamostragem de pixel

Bloco/sub	Parâmetro	RSP	IRS	MIRS	RDS
8x8/2:1	PSNR (dB)	34,56	35,52	35,93	36,4
	PRR (%)	61,5	64,57	65,85	67,28
	Nº de BCCs ($x10^9$)	0,25	0,27	0,41	0,98
	Nº de Comp. ($x10^9$)	8,02	8,71	13,21	31,4
16x16/4:1	PSNR (dB)	33,35	34,05	34,41	34,75
	PRR (%)	55,97	58,39	59,54	60,68
	Nº de BCCs ($x10^6$)	0,07	0,08	0,11	0,27
	Nº de Comp. ($x10^9$)	4,3	4,95	7,35	17,43
32x32/8:1	PSNR (dB)	32,22	32,65	32,92	33,1
	PRR (%)	51,19	53,07	54,12	54,93
	Nº de BCCs ($x10^6$)	0,02	0,02	0,03	0,07
	Nº de Comp. ($x10^9$)	2,2	2,69	3,91	9,48

Os dados mais importantes apresentados na Tabela 5.12 são os resultados de custo computacional. O número de BCCs sofre pequena alteração com o uso de subamostragem, apresentando pequena redução. Isto demonstra que o uso da subamostragem interfere pouco no comportamento nos algoritmos aleatórios, no entanto, contribui para uma pequena redução no número de iterações das etapas de refinamento iterativo. As etapas de refinamento iterativo são as únicas responsáveis por estas diferenças, uma vez que a etapa aleatória avalia os mesmos *N* blocos candidatos, com ou sem subamostragem de pixel. A grande diferença está no número de comparações, que cai significativamente com o uso da subamostragem de pixel. A taxa de redução no número de comparações é sempre superior ao fator de subamostragem utilizado e apresenta algumas variações significativas. Com o uso da subamostragem de pixel de 2:1, a redução do número de comparações foi da ordem de 2,1 vezes para todos os algoritmos. Para a subamostragem de 4:1, todos os algoritmos apresentaram uma redução no número de comparações superior a 4 vezes, sendo que os algoritmos IRS e MIRS apresentaram taxas de redução superiores a 4,5 vezes. Os resultados obtidos com a subamostragem de pixel de 8:1 apresentam os maiores ganhos de redução. Para todos os algoritmos, a redução do número de comparações é sempre superior a 9 vezes, sendo que para os algoritmos IRS e MIRS esta redução é superior a 10 vezes.

5.5 Avaliação dos Algoritmos Desenvolvidos para Vídeos Maiores que HD 1080p

O desenvolvimento algorítmico deste trabalho está focado na EM para vídeos de alta definição, mais especificamente a resolução HD 1080p, ou *Full* HD. Os resultados apresentados no capítulo 4 demonstram que as perdas em qualidade dos algoritmos rápidos de EM em relação ao algoritmo ótimo FS, crescem significativamente com o aumento da resolução do vídeo processado (Figura 4.4, algoritmo FS *versus* DS). Os algoritmos desenvolvidos neste trabalho visam reduzir estas perdas, aproximando os resultados de qualidade aos obtidos pelo algoritmo FS quando aplicados a vídeos HD 1080p. No entanto, algumas resoluções maiores que HD 1080p já podem ser encontradas, como a *Wide Quad eXtended Graphics Array* (WQXGA - 2560x1600 pixels) e a *Quad Full High Definition* (QFHD - 3840x2160 pixels), por exemplo. Seguindo a tendência dos resultados apresentados na Figura 4.4, as perdas em qualidade dos algoritmos rápidos para resoluções maiores que HD 1080p podem ser ainda maiores.

Para analisar a eficiência dos algoritmos desenvolvidos neste trabalho para vídeos de alta definição maiores que HD 1080p, uma nova bateria de testes foi realizada. Desta vez, apenas os resultados de qualidade serão apresentados, uma vez que os resultados de custo computacional não sofreram alterações significativas. Para esta avaliação, duas sequências de teste WQXGA foram utilizadas: PeopleOnStreet e Traffic (TNT, 2011). Vale destacar que esta sequência Traffic, apesar de homônima, não é igual à sequência Traffic apresentada na Figura 4.3. A Tabela 5.13 apresenta os resultados obtidos para os algoritmos desenvolvidos com blocos 16x16 e a Tabela 5.14 apresenta os resultados para blocos de tamanho 32x32.

Todos os algoritmos desenvolvidos apresentam resultados superiores aos obtidos pelo algoritmo DS. Os algoritmos MPDS e DMPDS apresentaram a menor diferença média de qualidade em relação ao algoritmo FS para blocos 16x16. O ganho dos algoritmos multiponto sobre os algoritmos aleatórios foi obtido para a sequência Traffic que apresenta menor movimentação que a sequência PeopleOnStreet. Para blocos 32x32, o algoritmo RDS obtém o melhor resultado médio dentre os algoritmos rápidos, mesmo apresentando resultado inferior para a sequência Traffic na comparação com o algoritmo DMPDS.

Tabela 5.13: Resultados comparativos de PSNR para os algoritmos em vídeos WQXGA para blocos 16x16

Algoritmos	**PSNR (dB) blocos 16x16**		
	PeopleOnStreet	**Traffic**	**Média**
FS	28,92	36,28	32,6
MPDS	28,54	36,02	32,28
DMPDS	28,50	36,05	32,28
RDS N=64	28,65	35,84	32,24
MIRS N=64	28,60	35,77	32,18
IRS N=64	28,49	35,46	31,97
DS	27,87	34,76	31,31

Tabela 5.14: Resultados comparativos de PSNR para os algoritmos em vídeos WQXGA para blocos 32x32

Algoritmos	PSNR (dB) blocos 32x32		
	PeopleOnStreet	Traffic	Média
FS	25,59	34,95	30,27
RDS N=64	25,55	34,76	30,16
DMPDS	25,49	34,81	30,15
MPDS	25,52	34,71	30,12
MIRS N=64	25,53	34,69	30,11
IRS N=64	25,51	34,50	30,01
DS	25,30	34,00	29,65

O resultado mais importante desta avaliação para vídeos com resolução superior a HD 1080p é demonstrar que os algoritmos desenvolvidos podem atingir ganhos maiores com o aumento da resolução dos vídeos. Para isto, as duas sequências de teste WQXGA foram reescaladas para HD 1080p e utilizadas para avaliar os algoritmos desenvolvidos. A tabela 5.15 apresenta os resultados médios de ganho de PSNR obtidos em relação ao algoritmo DS para as sequências de teste WQXGA e reescaladas para HD 1080p, considerando blocos de tamanho 16x16 e 32x32. Os resultados da Tabela 5.15 demonstram que os ganhos médios obtidos pelos algoritmos desenvolvidos crescem com o aumento da resolução. Isto demonstra que os algoritmos desenvolvidos nesta tese são eficientes para a codificação de vídeo HD 1080p e também podem ser eficientes para resoluções maiores.

Tabela 5.15: Ganho dos algoritmos (dB) sobre o DS em HD 1080p e WQXGA para blocos 16x16 e 32x32

Algoritmo	Bloco 16x16		Bloco 32x32	
	HD 1080p	WQXGA	HD 1080p	WQXGA
FS	0,61	1,28	0,41	0,61
MPDS	0,50	0,97	0,31	0,50
RDS N=64	0,49	0,96	0,30	0,49
DMPDS	0,46	0,93	0,28	0,46
MIRS N=64	0,46	0,87	0,25	0,46
IRS N=64	0,35	0,66	0,17	0,35

A Figura 5.20 apresenta os resultados médios de ganho PSNR (dB) sobre o algoritmo DS nas resoluções HD 1080p e WQXGA. Analisando a Figura 5.20 é possível perceber visualmente que os ganhos crescem para todos os algoritmos na resolução WQXGA, para os dois tamanhos de bloco avaliados.

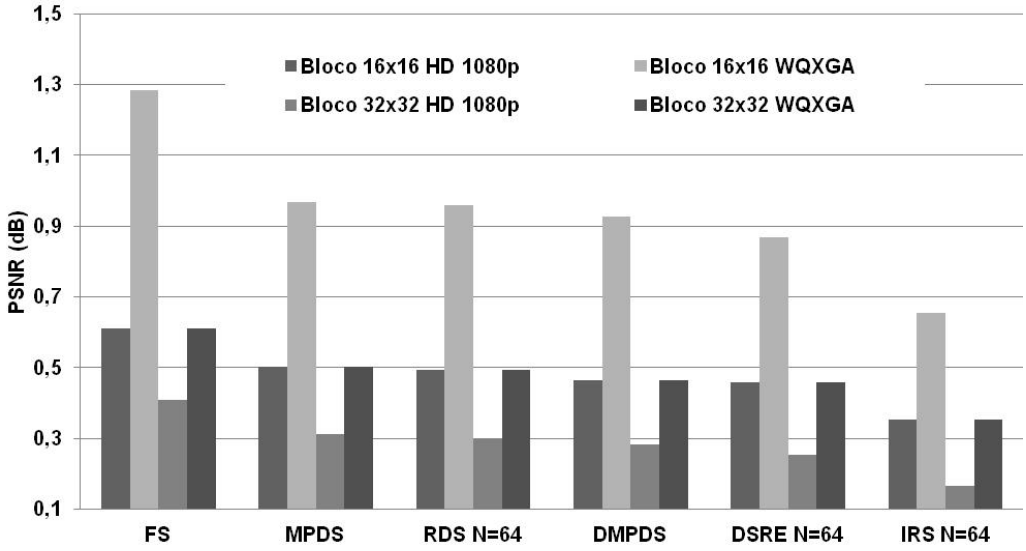


Figura 5.20: Ganho dos algoritmos sobre o DS em HD 1080p e WQXGA para blocos 16x16 e 32x32

5.6 Discussão e Comparações Sobre o Desenvolvimento Algorítmico

Os algoritmos rápidos desenvolvidos para a EM em vídeos de alta definição visam reduzir a ocorrência da escolha de mínimos locais, problema que ocorre com grande frequência nos algoritmos rápidos e que se torna ainda maior para vídeos de alta definição. Os algoritmos rápidos desenvolvidos neste trabalho utilizam abordagens diferentes para a redução da escolha de mínimos locais e podem ser divididos em duas classes: algoritmos multiponto e algoritmos aleatórios. Neste capítulo, os resultados obtidos para os algoritmos desenvolvidos serão comparados entre si e também em relação a outros algoritmos de EM importantes da literatura. Todos os algoritmos foram avaliados para uma área de busca de [-48, +48], utilizando tamanho de bloco 8x8, 16x16 e 32x32 pixels. O uso de subamostragem de pixel também foi avaliado nos níveis 2:1, 4:1 e 8:1 para os blocos de tamanho 8x8, 16x16 e 32x32, respectivamente. No total, 78 combinações diferentes de algoritmos, tamanho de bloco e nível de subamostragem foram avaliadas. No entanto, devido ao grande volume de dados, apenas os resultados mais significativos para blocos 16x16 e 32x32 estão apresentados no corpo deste texto, sendo que os resultados completos estão apresentados no Apêndice A desta tese. A escolha do tamanho de bloco 16x16 se justifica pelos resultados apresentados no capítulo 4, quanto ao uso de cada tamanho de bloco em vídeos HD 1080p pelo codificador H.264/AVC. Os resultados para o tamanho de bloco 32x32 também serão discutidos, pois a tendência do uso de blocos maiores está presente nas propostas do novo padrão de compressão de vídeo, o HEVC, ainda em desenvolvimento até o momento da escrita desta tese.

Todos os algoritmos avaliados possuem foco em qualidade, ou seja, não existem restrições de custo computacional. No entanto, os resultados de qualidade dos algoritmos aleatórios dependem do valor de N utilizado. Desta forma, a comparação dos resultados de qualidade obtidos para os algoritmos aleatórios com os valores de $\boldsymbol{N}$ definidos no capítulo 5.4.5 podem ser injustos, pois estes valores foram escolhidos considerando a melhor relação qualidade versus custo computacional. Pensando nisto, os algoritmos aleatórios também foram avaliados para $N = 64$, para a geração de novos resultados, desta vez, focados em máxima qualidade.

A Tabela 5.16 apresenta os resultados de qualidade (PSNR) e custo computacional (número de comparações) para os algoritmos, DS, UMH (YI, 2005), MPDS, DMPDS, RSP, IRS, MIRS e RDS para uma área de busca de [-48, +48] com blocos de 16x16 pixels e também para subamostragem de pixel 4:1. Os resultados apresentados na Tabela 5.16 estão ordenados pelos resultados de PSNR (de maior a menor), juntamente com os resultados de diferença (**Δ**) obtidos em relação ao algoritmo FS. O custo computacional está apresentado em número de operações de comparação entre pixels, para evidenciar os ganhos obtidos com a utilização da subamostragem de pixel, uma vez que o número de BCCs se mantém praticamente estável. Além disto, a Tabela 5.16 também apresenta os resultados de redução do custo computacional (em número de vezes) em relação ao custo computacional do algoritmo FS.

Tabela 5.16: Resultados comparativos de qualidade e custo computacional para os algoritmos de EM com tamanho de bloco 16x16

Algoritmos/sub	PSNR (dB)	Δ PSNR (dB)	Comp.x10^9	Δ Comp.x10^9 (vezes)
FS	35,89	-	3753,62	-
FS/4:1	35,7	-0,19	938,41	-4,0
RDS N=64	35,18	-0,71	80,85	-46,4
MIRS N=64	35,11	-0,78	47,9	-78,4
RDS N=64/4:1	34,95	-0,94	19,51	-192,4
DMPDS	34,87	-1,02	81,55	-46,0
MIRS N=64/4:1	34,85	-1,04	11,36	-330,4
MPDS	34,75	-1,14	78,72	-47,7
DMPDS/4:1	34,65	-1,24	19,29	-194,6
IRS N=64/4:1	34,56	-1,33	8,95	-419,4
UMH	34,42	-1,47	79,86	-47,0
MIRS N=16/4:1	34,41	-1,48	7,35	-510,7
UMH/4:1	34,35	-1,54	19,95	-188,2
RSP N=16/4:1	33,35	-2,54	4,3	-872,9
DS	33,02	-2,87	12,3	-305,2
DS/4:1	32,74	-3,15	2,96	-1268,1

Os algoritmos RDS e MIRS com N = 64 apresentam os melhores resultados de qualidade entre todos os algoritmos rápidos avaliados, com perdas de aproximadamente 0,75dB em relação ao algoritmo FS. Ainda assim, a redução no custo computacional foi alta, cerca de 46 vezes para o algoritmo RDS com N=64 e 78 vezes para o algoritmo MIRS com N=64. O algoritmo DMPDS também apresenta bons resultados de qualidade, com perdas em torno de 1dB em relação ao algoritmo FS e redução do custo computacional similar ao obtido pelo algoritmo RDS. O uso da subamostragem de pixel pode beneficiar os algoritmos aleatórios mais complexos como o RDS e MIRS de modo mais significativo. Estes algoritmos são menos suscetíveis às perdas causadas pelo uso da subamostragem de pixel, desta forma, os resultados de qualidade obtidos se mantêm

entre os melhores, ao mesmo tempo em que os resultados de custo computacional se aproximam dos menores dentre todos os algoritmos rápidos. O algoritmo RDS com N = 64 e subamostragem de pixel de 4:1 (N=64/4:1) apresenta um dos melhores resultados de qualidade, com uma redução do custo computacional superior a 192 vezes em relação ao FS. Os resultados de custo computacional do algoritmo RDS N=64/4:1 são muito próximos dos resultados obtidos pelo algoritmo RSP com N = 16, no entanto, apresentam um ganho de aproximadamente 1,21dB em qualidade.

Para uma análise mais clara dos resultados, a Figura 5.21 apresenta um gráfico que relaciona os resultados de PSNR e número de comparações para parte dos algoritmos avaliados com blocos 16x16. O algoritmo FS foi excluído deste gráfico, pois o número de comparações é entre 46 e 1268 vezes superior aos resultados apresentados pelos algoritmos rápidos avaliados. Isto dificulta a visualização dos resultados quando contrastados com os resultados obtidos pelos algoritmos rápidos. O gráfico da Figura 5.21 apresenta no eixo horizontal o número de comparações ($x10^9$) executadas pelos algoritmos, e no eixo vertical os resultados de 1/PSNR (o inverso do PSNR em dB) obtidos pelos algoritmos. Desta forma, quanto melhor a qualidade da EM, medida em termos de PSNR, mais próximo de zero (no eixo Y) estará a metrica 1/PSNR. Com isso, é possível identificar uma região (abaixo da curva da Figura 5.21) onde os melhores resultados na relação PSNR versus comparações podem ser obtidos.

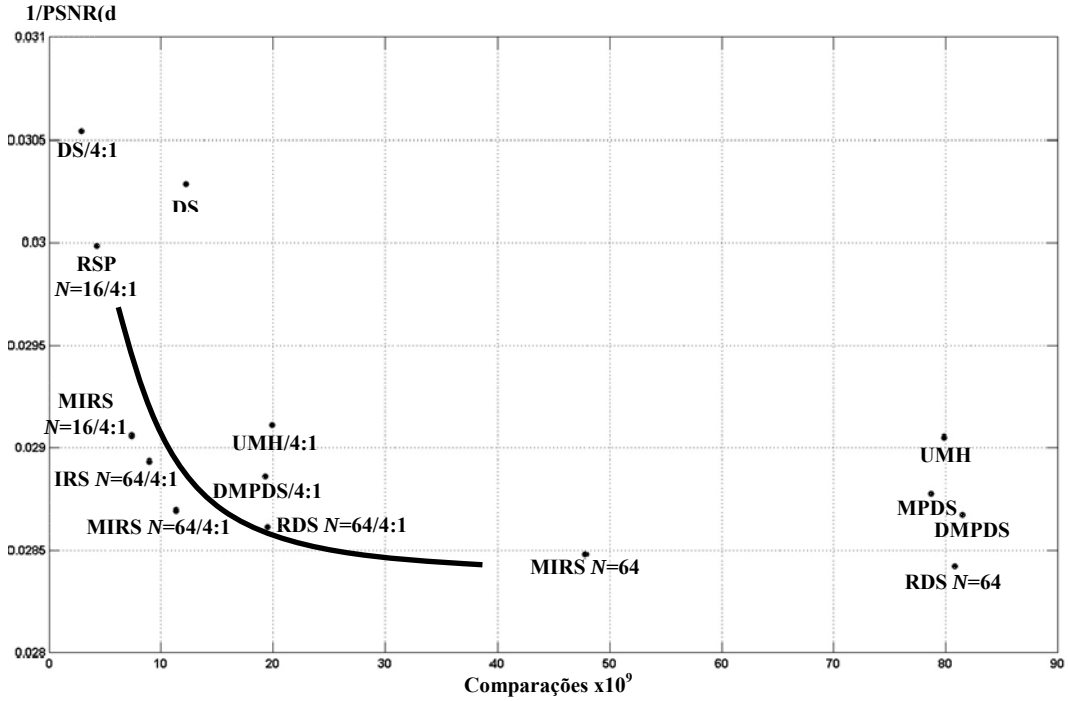


Figura 5.21: Relação 1/PSNR versus o número de comparações para blocos 16x16

Analisando a Figura 5.21 é possível perceber as grandes variações nos resultados de dos algoritmos avaliados. Resultados de qualidade muito próximos podem gerar resultados muito discrepantes de número de comparações. Os algoritmos RSD e MIRS com N = 64 atingem resultados médios de PSNR superiores a 35dB, no entanto, o MIRS apresenta um custo computacional 41% inferior. A Figura 5.21 apresenta ainda seis algoritmos com resultados de PSNR entre 34,5dB e 35dB. Dentre eles, destaque para os algoritmos MIRS e IRS com N = 64 e subamostragem de pixel 4:1 (N=64/4:1), que obtiveram os menores resultados de número de comparações. O algoritmo RDS

N=64/4:1 apresenta um pequeno ganho em qualidade em relação ao algoritmo DMPDS/4:1, com um número de comparações muito similar. Os algoritmos MPDS e DMPDS sem subamostragem apresentam bons resultados de qualidade, no entanto, o número de comparações destes algoritmos está entre os maiores.

O menor custo computacional foi obtido para o algoritmo DS com subamostragem de pixel 4:1, no entanto, acompanhado do pior resultado de PSNR dentre todos os algoritmos avaliados. Mesmo a versão sem subamostragem do algoritmo DS apresenta um PSNR inferior ao pior resultado de qualidade dentre os algoritmos aleatórios, que foi obtido pelo algoritmo RSP N=16/4:1. Além disso, o RSP N=16/4:1 apresenta um resultado de custo computacional próximo ao obtido pelo algoritmo DS com subamostragem de pixel, com um ganho de aproximadamente 0,6dB.

Os algoritmos que apresentam a melhor relação entre o PSNR obtido e o número de comparações utilizadas estão representados na região abaixo (ou próximos) da curva da Figura 5.21. Os algoritmos MIRS *N*=16/4:1, *N*=64/4:1 e IRS *N*=64/4:1 apresentam a melhor relação PSNR versus número de comparações dentre todos os algoritmos avaliados. O algoritmo RDS *N*=64/4:1 também apresenta um bom resultado, estando muito próximo da curva que indica a região de melhor resultado.

A Tabela 5.17 apresenta os resultados de qualidade e custo computacional para os algoritmos de EM com blocos de tamanho 32x32.

Tabela 5.17: Resultados comparativos de qualidade e custo computacional para os algoritmos de EM com tamanho de bloco 32x32

Algoritmos/sub	PSNR (dB)	Δ PSNR (dB)	Comp.x10^9	Δ Comp.x10^9 (vezes)
FS	33,76	-	3695,34	-
FS/8:1	33,44	-0,32	461,92	-8,0
RDS N=64	33,57	-0,19	90,72	-40,7
MIRS N=64	33,55	-0,21	52,46	-70,4
IRS N=64	33,46	-0,3	42,41	-87,1
DMPDS	33,38	-0,38	103,27	-35,8
MPDS	33,35	-0,41	101,23	-36,5
MIRS N=16	33,35	-0,41	39,33	-94,0
RDS N=64/8:1	33,22	-0,54	10,44	-354,0
MIRS N=64/8:1	33,17	-0,59	5,9	-626,3
UMH	33,07	-0,69	79,91	-46,2
IRS N=64/8:1	33,01	-0,75	4,67	-791,3
DMPDS/8:1	32,96	-0,8	10,88	-339,6
UMH/8:1	32,8	-0,96	9,98	-370,3
RSP N=64/8:1	32,69	-1,07	4,72	-782,9
IRS N=16/8:1	32,65	-1,11	2,69	-1373,7
DS	32,41	-1,35	13,2	-280,0
DS/8:1	31,88	-1,88	1,52	-2431,1

Os resultados de qualidade apresentam perdas, em relação aos obtidos para blocos 16x16, para todos os algoritmos. No entanto, as diferenças em relação ao algoritmo FS diminuem significativamente. Os resultados de custo computacional sofrem poucas alterações, em geral, apresentando pequenas reduções. Alterações mais significativas no número de comparações foram obtidas apenas para os algoritmos com subamostragem de pixel, pois para blocos 32x32 a subamostragem de pixel utilizada foi de 8:1.

Os algoritmos RDS e MIRS com N = 64 obtiveram resultados de PSNR com perdas de aproximadamente 0,2dB em relação ao algoritmo FS. Estes resultados são praticamente iguais aos obtidos para o algoritmo FS com subamostragem de pixel 8:1, no entanto, com redução no número de operações de aproximadamente 5 vezes para o algoritmo RDS e 9 vezes para o algoritmo MIRS. Mais uma vez, o menor custo computacional foi obtido para o algoritmo DS com subamostragem de pixel, desta vez, reduzindo em aproximadamente 2431 vezes o número de operações em relação ao algoritmo FS, no entanto, também apresenta a maior perda em qualidade (próximo a 2dB).

A Figura 5.22 apresenta um gráfico relacionando os resultados de PSNR e número de comparações para alguns dos algoritmos avaliados. Os melhores resultados de relação PSNR versus número de comparações foram obtidos pelos algoritmos aleatórios IRS, MIRS e RDS com $N = 64$ e subamostragem de pixel 8:1. O algoritmo DMPDS também aprsentou um bom resultado, ficando muito próximo a curva da Figura 5.22. Os algoritmos IRS com $N = 64$ e MIRS, $N = 16$ e $N = 64$ também obtiveram bons resultados. Os algoritmos multiponto, MPDS e DMPDS, assim como o RDS $N = 64$, atingem resultados de PSNR elevados, no entanto, também apresentam valores elevados de número de comparações, fazendo com que seus resultados no gráfico estejam fora da região de melhores valores.

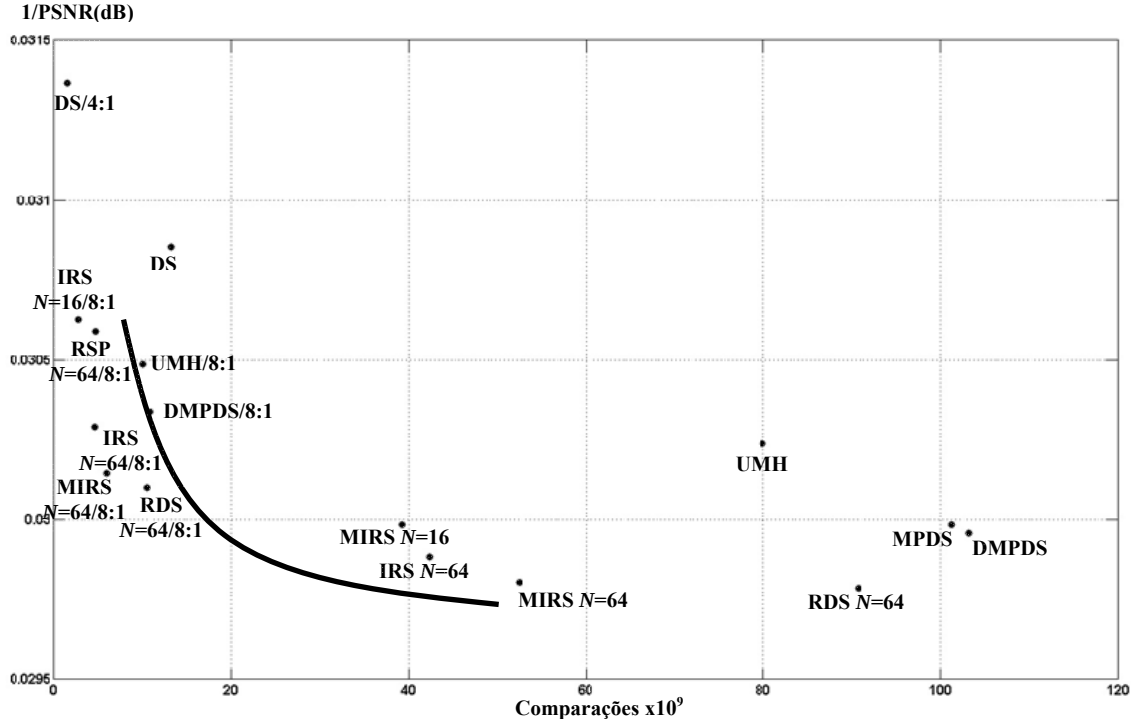


Figura 5.22: Relação 1/PSNR versus o número de comparações para blocos 32x32

Todos os algoritmos desenvolvidos nesta tese, sejam multipontos ou aleatórios, apresentam resultados de qualidade superiores aos obtidos pelo algoritmo DS, quando aplicados a sequências de teste HD 1080p. Os melhores resultados de qualidade são obtidos para as versões que apresentam os maiores custos computacionais. Nestes casos, existe um grande acréscimo no número de comparações em relação ao algoritmo DS. No entanto, em alguns casos é possível obter ganhos tanto em qualidade quanto em redução do custo computacional. Desta forma, os algoritmos desenvolvidos nesta tese podem ser usados em todo tipo de aplicação, desde alta qualidade, sem preocupação com o custo computacional, até aplicações que necessitem de alta qualidade, porém, sem extrapolar o número de comparações.

O uso de subamostragem de pixel se mostrou mais uma vez uma estratégia muito interessante, pois possibilita que mesmo os algoritmos mais complexos, como RDS e MPDS, apresentem resultados de custo computacional comparáveis aos obtidos por outros algoritmos rápidos.

Os algoritmos aleatórios e multiponto também foram comparados com o algoritmo ATME, apresentado em (CETIN, 2011). Os resultados apresentados em (CETIN, 2011) para o algoritmo ATME foram gerados para três sequências de teste HD 1080p: Crowd_run, In_to_tree e Park_joy (VQEG, 2011). Para uma comparação justa, os algoritmos desenvolvidos nesta tese também foram aplicados as estas três sequências. Os resultados médios de qualidade e custo computacional estão apresentados na Tabela 5.18. Como o algoritmo ATME tem foco em baixo custo computacional, a Tabela 5.18 apresenta apenas os resultados de menor custo computacional para os algoritmos desenvolvidos. Foram selecionados os melhores resultados de qualidade apresentados em (CETIN, 2011) para o algoritmo ATME, uma vez que diversas configurações do algoritmo são apresentadas com diferentes resultados de qualidade e custo computacional. Os resultados do algoritmo ATME foram gerados para blocos 16x16 em uma área de busca de [-64, +64]. Os algoritmos aleatórios utilizaram uma área de busca de [-48, +48] e os valores de *N* definidos no capítulo 5.4.5.

Tabela 5.18: Resultados comparativos de qualidade e custo computacional em relação ao algoritmo ATME

Algoritmo	PSNR (dB)	BCCs x 10^6	Comparações x 10^9
MIRS N=16/4:1	30,73	42,54	2,72
MPDS 4:1	30,72	114,74	7,34
IRS N=16/4:1	30,45	28,91	1,85
RSP N=16/4:1	30,28	26,06	1,67
ATME	29,68	8,2	2,1

Todos os algoritmos desenvolvidos apresentam resultados de qualidade superiores para todas as sequências de vídeo e, consequentemente, melhores resultados médios. Dentre os algoritmos apresentados na Tabela 5.18, o algoritmo MIRS apresentou os melhores resultados de qualidade para estas sequências. O algoritmo MPDS obteve um resultado ligeiramente superior ao algoritmo DMPDS, praticamente imperceptível após o truncamento para apresentação na tabela. Este resultado não era esperado, mas pode ser explicado devido às características das sequências, que possuem grande movimentação e são compostas de apenas 150 quadros. Desta forma, o algoritmo

DMPDS não consegue convergir para o valor ótimo de *d* em tempo de gerar resultados mais eficientes do que os gerados pelo algoritmo MPDS.

O algoritmo MIRS apresenta um acréscimo no número de comparações de aproximadamente 30%, com um ganho médio superior a 1dB em relação ao algoritmo ATME. Com o uso da subamostragem de pixel 4:1, os algoritmos IRS e RSP apresentam um número de comparações inferior ao apresentado pelo algoritmo ATME, além de obterem ganhos de qualidade de 0,6dB e 0,77dB, respectivamente.

Os algoritmos desenvolvidos nesta tese apresentam os resultados de qualidade mais próximos do algoritmo FS dentre todos os demais algoritmos avaliados. Todos os algoritmos desenvolvidos apresentam uma redução significativa no custo computacional, na comparação com o algoritmo FS. No entanto, alguns destes algoritmos apresentam um custo computacional elevado, em relação a outros algoritmos rápidos. Com o uso da subamostragem, grande parte dos algoritmos desenvolvidos apresenta resultados de qualidade ainda superiores aos obtidos pelo algoritmo DS, por exemplo, e na maioria dos casos, com um custo computacional inferior.

Estes resultados demonstram que algoritmos iterativos como o DS, por exemplo, não apresentam acréscimo significativo no custo computacional com o aumento da resolução dos vídeos, no entanto, não conseguem manter bons resultados de qualidade nestas condições. Os algoritmos muitiponto e aleatórios exploram diferentes regiões da área de busca simultaneamente, o que aumenta o custo computacional, no entanto, garante ganhos significativos de qualidade em relação a algoritmos rápidos convencionais. Os algoritmos aleatórios possuem diferentes heurísticas de exploração da região central da área de busca, e também de refinamento da exploração aleatória, gerando diferentes resultados de qualidade e também uma grande variação de custo computacional.

6 ARQUITETURAS DE HARDWARE DESENVOLVIDAS

Neste capítulo estão apresentados os resultados do desenvolvimento arquitetural para a EM realizado neste trabalho. As arquiteturas desenvolvidas foram descritas em VHDL e validadas utilizando o software ModelSim (MODELSIM, 2010). Os resultados de síntese foram obtidos para FPGAs (Altera e Xilinx) e também para ASIC com síntese para *standard cells* na tecnologia TSMC 0,18µm e 90nm. Oito arquiteturas foram desenvolvidas, que implementam quatro algoritmos diferentes e apresentam diferentes configurações, com resultados distintos de qualidade, desempenho e uso de recursos de hardware e memória.

Os resultados obtidos nesta tese com o desenvolvimento de arquiteturas dedicadas de EM geraram algumas publicações originais, como: (PORTO, 2008b), (PORTO,2008c), (PORTO, 2009), (ROSA, 2009), (SILVA, 2009), (PORTO, 2010), (PORTO, 2011a), (SANCHEZ, 2011a), (SANCHEZ, 2011b), (CRISTANI, 2011a), (PORTO, 2011b) e (PORTO, 2012b). Mais detalhes sobre os artigos publicados com os resultados obtidos no desenvolvimento arquitetural estão apresentados no Apêndice B desta tese.

6.1 Arquiteturas dos algoritmos SDS-DIC e QSDS-DIC

As arquiteturas desenvolvidas para os algoritmos SDS-DIC (*Sub-sampled Diamond Search with Dynamic Iteration Control*) e QSDS-DIC (*Quarter Sub-sampled Diamond Search with Dynamic Iteration Control*) são muito semelhantes e utilizam o mesmo *template* arquitetural. Ambas operam sobre blocos de 16x16 amostras e utilizam o SAD como critério de similaridade. No entanto, o nível de subamostragem utilizado em cada arquitetura é diferente, o que interfere no tamanho das memórias internas, no controle e no desempenho das arquiteturas. Todas as figuras que ilustram a arquitetura são idênticas, portanto, apenas a arquitetura do SDS-DIC será considerada nas explicações.

O diagrama de blocos da arquitetura do SDS-DIC é apresentado na Figura 6.1. Esta arquitetura considera o nível máximo de paralelismo do algoritmo DS, onde um LDSP, com nove blocos candidatos pode ser processado em paralelo. O módulo da memória local (ML) armazena todos os dados dos blocos candidatos do LDSP e, também, todos os blocos candidatos para a próxima iteração do algoritmo. Um conjunto de memórias de bloco candidato (MBC) é utilizado para armazenar individualmente os blocos candidatos do LDSP e uma memória é dedicada para armazenar o bloco do quadro atual (MBA). Uma memória especial é destinada apenas para o armazenamento dos blocos candidatos do refinamento final, ou SDSP (MBCS). As unidades de processamento (UP0 a UP8) calculam o SAD para cada linha dos blocos candidatos e um conjunto de somadores e acumuladores gera o valor final de SAD. Quando todas as linhas dos

blocos forem processadas, os resultados de SAD são enviados para o comparador, que identifica o menor SAD. A informação do bloco candidato escolhido (BE) é enviada para o controle da arquitetura, que define o próximo passo.

Existem três possibilidades para o próximo passo da arquitetura, busca por aresta, busca por vértice e refinamento final (SDSP). Um valor é atribuído para cada bloco candidato armazenado na MBC, de acordo com sua posição dentro do LDSP. A Figura 6.2 ilustra o LDSP com todos os seus blocos candidatos numerados. Cada UP trabalha sobre um bloco candidato do LDSP, sendo que a UP0 calcula o SAD para o bloco candidato 0, a UP1 calcula o SAD para o bloco candidato 1 e assim sucessivamente. Quando o bloco 4 obtiver o menor SAD, a unidade de controle dispara o processo de refinamento final, com o SDSP. Os blocos candidatos para o SDSP estão armazenados na MBCS e podem ser acessados imediatamente, sem nenhum ciclo de latência de memória.

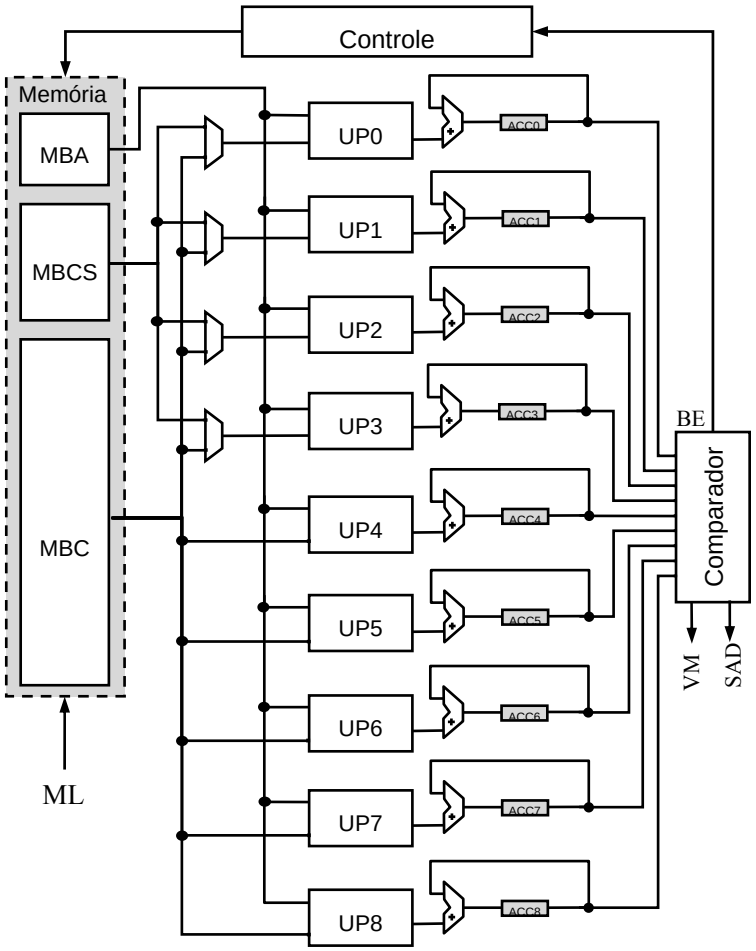


Figura 6.1: Diagrama em blocos da arquitetura do SDS-DIC

Quando os blocos 0, 3, 5 ou 8 resultarem no menor SAD, o controle da arquitetura dispara a busca por vértice, onde mais cinco blocos candidatos serão comparados. Caso o bloco 1, 2, 6 ou 7 possua o menor SAD, o controle inicia uma busca por aresta e mais três blocos candidatos são comparados. Tanto na busca por aresta, quanto na busca por vértice, existe uma latência de memória para o carregamento dos novos blocos candidatos. O número de repetições de busca por vértice ou aresta pode variar

consideravelmente, de acordo com o vídeo a ser codificado, no entanto, o DIC é responsável por restringir dinamicamente estas iterações.

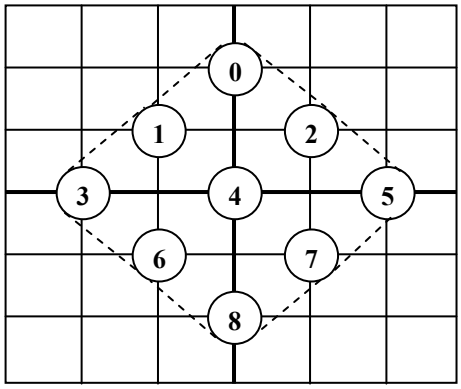


Figura 6.2: LDSP com os blocos candidatos numerados

As UPs foram desenvolvidas utilizando uma árvore de somadores em um pipeline de quatro estágios. Cada UP recebe uma linha inteira do bloco atual e uma linha do bloco candidato. Com o uso da subamostragem de pixel, cada linha possui oito amostras. Dezesseis acumulações são necessárias para a geração do resultado de SAD de um bloco candidato, uma para cada linha de um bloco 16x8 (bloco 16x6 com subamostragem de pixel 2:1). A Figura 6.3 ilustra a arquitetura da UP utilizada na arquitetura do SDS-DIC.

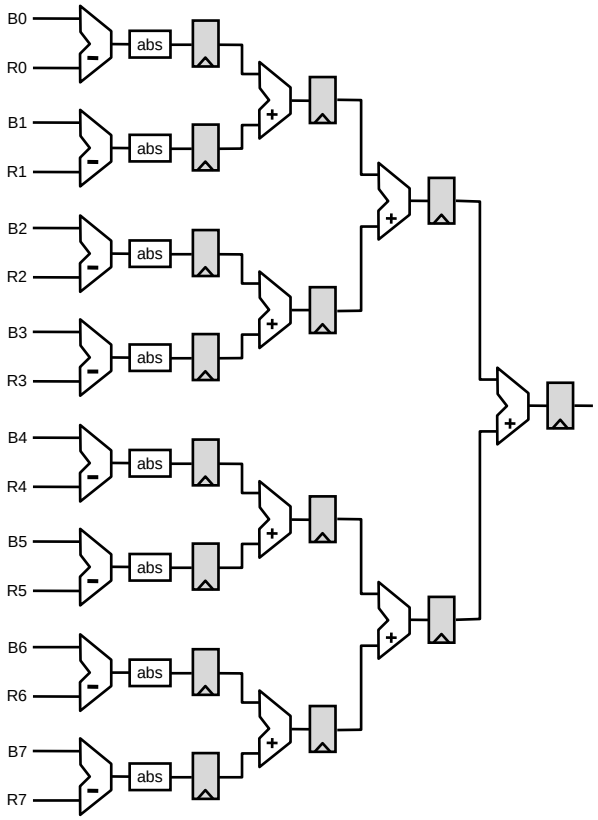


Figura 6.3: Arquitetura da UP utilizada na arquitetura do SDS-DIC

A memória interna da arquitetura do SDS-DIC está organizada em 15 memórias distintas, como apresentado na Figura 6.4. A ML armazena a região onde estão localizados os nove blocos candidatos do LDSP, juntamente com todos os blocos

candidatos possíveis para a próxima etapa do algoritmo. Desta forma, os dados para o próximo passo do algoritmo estão sempre disponíveis na memória interna, evitando o acesso à memória externa. Quando o controle da arquitetura decide continuar a busca, a ML é recarregada. Este processo ocorre simultaneamente à leitura e, desta forma, a ML é atualizada paralelamente ao processamento do SAD para os blocos candidatos. A ML é composta por 24 palavras de 128 bits cada. Outras 14 pequenas memórias são utilizadas para armazenar os nove blocos candidatos do LDSP, mais os quatro blocos do SDSP e o bloco do quadro atual. Os blocos candidatos do LDSP são armazenados nas MBCs e o bloco atual é armazenado na MBA. Os quatro blocos candidatos do SDSP são armazenados nas MBCSs. Estas 14 memórias são compostas de 16 palavras de 64 bits (oito amostras de oito bits) cada e armazenam um bloco subamostrado de 16x8 amostras.

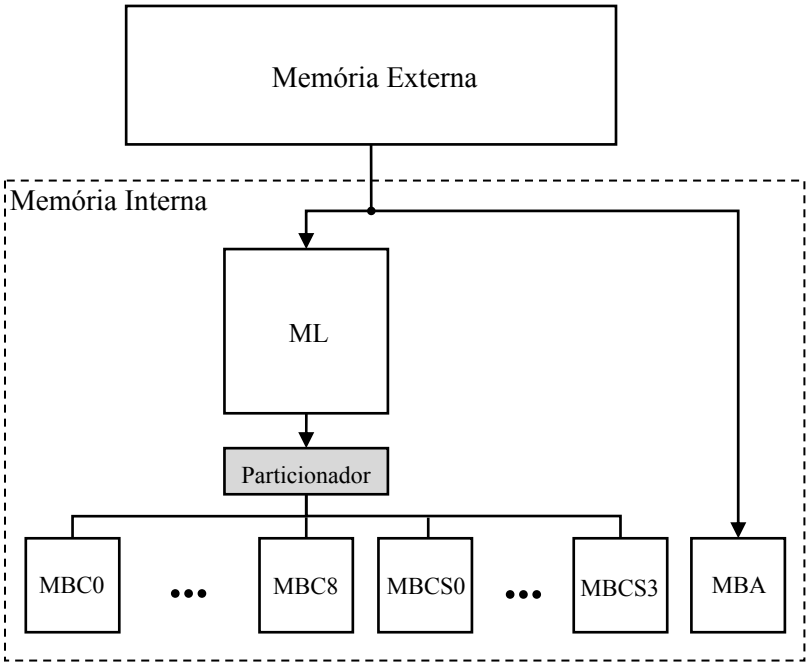


Figura 6.4: Organização das memórias internas

Quando o controle da arquitetura define que o próximo passo será um busca por vértice ou busca por aresta, a ML precisa ser recarregada, de forma que contenha novamente todos os próximos blocos candidatos para o novo LDSP. Neste caso, 10 ciclos de latência são necessários, para que uma linha seja escrita em cada uma das nove MBCs e, então, o cálculo dos SADs possa ter início. Caso o controle da arquitetura decida pela aplicação do SDSP, os quatro blocos candidatos já estão armazenados nas MBCSs e o cálculo dos SAD inicia sem nenhum ciclo de latência de memória.

A memória da arquitetura do SDS é otimizada, pois a área de busca é carregada de acordo com a necessidade do algoritmo. Para a ML são usadas 24 palavras de 128 bits cada, em um total de 3072 bits de memória. Cada memória de bloco (MBC, MBCS ou MBA) utiliza 16 linhas de 64 bits cada, totalizando 1024 bits. No total, a arquitetura do SDS-DIC utiliza 17.408 bits de memória interna.

O DIC é implementado como um controle externo à arquitetura e o custo adicional em hardware é extremamente baixo. A Figura 6.5 apresenta a forma de implementação do DIC na arquitetura. A informação de número de iterações utilizadas pela arquitetura é enviada para um registrador de deslocamento (RD) de 16 posições. Um somador

recebe os valores de número de iterações utilizadas para os últimos 16 vetores e faz a soma do número total de laços utilizados. A arquitetura do SDS-DIC trabalha com *N*=10 e, desta forma, todo bloco candidato que utilizar menos de 10 iterações disponibilizará iterações adicionais para outros blocos candidatos. O número de iterações disponíveis para o próximo bloco candidato será o resultado entre a subtração do valor máximo de iterações (*N**16) e o total acumulado até o bloco anterior. O DIC não interfere no desempenho da arquitetura e permite o sincronismo da EM com os demais blocos do codificador, pois existe um número fixo de ciclos para cada conjunto de 16 vetores de movimento.

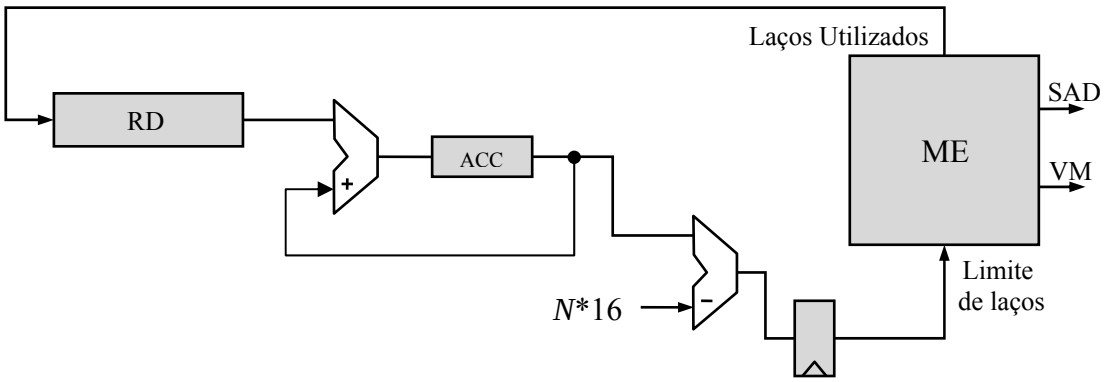


Figura 6.5: Estrutura do controle dinâmico de laços (DIC)

A arquitetura do SDS-DIC utiliza 36 ciclos de *clock* para preencher todas as memórias e iniciar o processo de cálculo de SADs. As UPs possuem uma latência de quatro ciclos e mais 15 ciclos são necessários para processar as 16 linhas dos blocos candidatos. O comparador necessita de mais cinco ciclos para encontrar o menor SAD. No total, 60 ciclos são utilizados para calcular o primeiro LDSP. O SDSP utiliza mais 28 ciclos para calcular os SADs dos quatro blocos candidatos e comparar os seus resultados com o menor SAD do passo anterior. Desta forma, no melhor caso, a arquitetura do SDS-DIC necessita de 88 ciclos de *clock* para determinar o vetor de movimento para um bloco.

Para os casos onde o LDSP é repetido, tanto para uma busca por vértice quanto para uma busca por aresta, 10 ciclos de *clock* são necessários para escrita nas MBCs. Os mesmo 24 ciclos são utilizados para as UPs e o comparador. Desta forma, 34 ciclos são necessários para cada iteração do LDSP. A arquitetura do SDS-DIC pode variar dinamicamente o número de iterações do LDSP, sendo que no pior caso, o valor máximo será restrito pelo DIC com *N**16 para um conjunto de 16 vetores de movimento. No pior caso, a arquitetura poderá utilizar até 428 ciclos de *clock*. Um caso médio pode ser considerado, levando em conta o valor médio de iterações do algoritmo SDS (aproximadamente 6) em vídeos de alta definição para blocos 16x16. Neste caso, 292 ciclos de *clock* são utilizados para a geração de um vetor de movimento.

Cada iteração do algoritmo SDS-DIC avança duas amostras na área de busca, seja para esquerda, direita, para cima ou para baixo. Para a aplicação do primeiro LSDP é necessário uma área de busca de 20x20 amostras. Logo, considerando *N*=10, a arquitetura do SDS-DIC pode utilizar uma área de busca máxima de 660x660 amostras. Com uma restrição fixa (sem DIC), com o valor máximo de 10 iterações por vetor, a área de pesquisa máxima seria de 100x100 amostras.

6.1.1 Resultados de Síntese da Arquitetura do SDS-DIC

A Tabela 6.1 apresenta os resultados de síntese para os principais blocos da arquitetura do SDS-DIC. A arquitetura do algoritmo SDS-DIC foi descrita em VHDL e sintetizada para FPGA. A ferramenta ISE 8.1i (XIILINX, 2010a) foi utilizada na etapa de descrição da arquitetura e a ferramenta ModelSim 6.1 (MODELSIM, 2010) foi utilizada para a etapa de simulação e validação da arquitetura. O código VHDL foi sintetizado para o FPGA XC4VLX15 da família Virtex-4 da Xilinx (XILINX, 2007).

A arquitetura do SDS-DIC utiliza uma pequena quantidade de recursos de hardware, sendo apenas 3890 LUTs, de um total de 15K disponíveis neste dispositivo. A utilização das memórias do dispositivo também é pequena, sendo apenas 32 BRAMs. A frequência máxima de operação atingida foi elevada, o que garante uma taxa de processamento de 53 quadros HD 1080p por segundo, no pior caso. Esta taxa de processamento é superior à necessária para processar vídeos desta resolução em tempo real.

Tabela 6.1: Resultados de síntese para a arquitetura do SDS-DIC

Bloco	LUTs	Slices	Slices FF	BRAMs	Frequência (MHz)
UP	227	137	146	-	466,76
Comparador	509	331	350	-	232,88
Memória	519	276	250	32	308,88
Controle	127	70	50	-	366,66
SDS-DIC	3890	2190	2379	32	184,19

Dispositivo: Virtex-4 XC4VLX15

6.1.2 Arquitetura do algoritmo QSDS-DIC

Como já foi mencionado anteriormente, a arquitetura do QSDS-DIC é muito similar a apresentada para o SDS-DIC. O *template* arquitetural é o mesmo, assim como a organização das memórias e arquitetura das UPs. A principal diferença está no nível de subamostragem de pixel utilizado, que no QSDS-DIC é de 4:1. O aumento do nível de subamostragem reduz o tamanho das memórias internas e, também, o número de ciclos de *clock* necessários para a geração de um vetor, pois o número de pixels para o cálculo de SAD é reduzido pela metade. No entanto, a arquitetura do QSDS-DIC utiliza $N = 20$, o que possibilita o dobro de iterações em relação à arquitetura do SDS-DIC.

A organização das memórias e a comunicação com o controle são iguais aos apresentados para a arquitetura do SDS-DIC. A diferença está no tamanho, pois com a subamostragem de 4:1, os blocos de 16x16 são reduzidos a blocos 8x8. A ML da arquitetura do QSDS-DIC utiliza 16 linhas de 128 bits cada, o que resulta em 2048 bits de memória. As memórias dos blocos candidatos (MBC e MBCS), assim como a memória do bloco atual (MBA) possuem oito linhas de 64 bits cada, o que resulta em 512 bits de memória para cada memória de bloco. No total, as 15 memórias internas da arquitetura do QSDS-DIC utilizam apenas 9216 bits de memória. A arquitetura do QSDS-DIC reduz em mais de 43% a quantidade de memória necessária, na comparação com a arquitetura do SDS-DIC.

A arquitetura do QSDS-DIC necessita de um número menor de ciclos de *clock*, comparado a arquitetura SDS-DIC. Com a subamostragem de pixel 4:1, a arquitetura do QSDS-DIC utiliza apenas 26 ciclos de *clock* para preencher todas as memórias. As UPs continuam com uma latência de quatro ciclos e mais sete são necessários para processar as oito linhas dos blocos candidatos. Mais cinco ciclos são utilizados para o comparador e, desta forma, a arquitetura do QSDS-DIC utiliza 42 ciclos de *clock* para calcular o primeiro LDSP. O SDSP utiliza mais 20 ciclos para calcular os SADs dos quatro blocos candidatos e comparar os seus resultados com o menor SAD do passo anterior. Assim, no melhor caso, a arquitetura do QSDS-DIC utiliza 62 ciclos de *clock* para gerar vetor de movimento para um bloco.

A busca por aresta ou por vértice exige a mesma latência de 10 ciclos para a escrita nas MBCs. Isto ocorre, pois o número de memórias de bloco permanece o mesmo da arquitetura do SDS-DIC. Desta forma, a arquitetura do QSDS-DIC utiliza 26 ciclos de *clock* para cada LDSP. No pior caso, a arquitetura do QSDS-DIC pode utilizar 582 ciclos de *clock*. No caso médio, para cinco iterações, 192 ciclos de *clock* são necessários para a geração de um vetor de movimento.

As iterações do algoritmo QSDS-DIC também avançam duas amostras e o primeiro LSDP utiliza a mesma área de busca de 20x20 amostras. Logo, para N=20, a arquitetura do QSDS-DIC pode utilizar uma área de busca máxima de 1300x1300 amostras. Com uma restrição fixa (sem DIC), com o valor máximo de 20 iterações por vetor, a área de pesquisa máxima seria de 200x200 amostras.

6.1.3 Resultados de Síntese da Arquitetura do QSDS-DIC

A arquitetura do algoritmo QSDS-DIC também foi descrita em VHDL utilizando a ferramenta ISE 8.1i. O código VHDL foi sintetizado para o mesmo FPGA Virtex-4 FPGA XC4VLX15 da família da Xilinx. A arquitetura do QSDS foi simulada e validada utilizando a ferramenta ModelSim 6.1. A Tabela 6.2 apresenta os resultados de síntese para os principais módulos da arquitetura do QSDS-DIC.

Tabela 6.2: Resultados de síntese para a arquitetura do QSDS-DIC

Módulo	LUTs	Slices	Slices FF	BRAMs	Frequência (MHz)
UP	227	137	146	-	466,76
Comparador	365	240	254	-	250,23
Memória	486	258	216	32	308,88
Controle	125	69	50	-	412,34
QSDS-DIC	3610	2007	2086	32	213,37

Dispositivo: Virtex-4 XC4VLX15

Os resultados de síntese para a arquitetura do QSDS-DIC são semelhantes aos resultados obtidos para a arquitetura do SDS-DIC, no entanto, a utilização de LUTs, *Slices* e *Slices* FF são menores. Esta diminuição na utilização dos recursos de hardware se deve principalmente à redução to tamanho dos acumuladores, que precisam armazenar um número menor de linhas do bloco. Os blocos de controle e o comparador também foram simplificados e contribuíram na redução dos recursos de hardware utilizados. Apesar da redução do tamanho das memórias utilizadas, o número de BRAM

permanece o mesmo. Isto ocorre devido às características de implementação das BRAMs do dispositivo, que possuem 256 palavras de 64 bits. A arquitetura do QSDS-DIC utiliza memórias com menos palavras, mas com o mesmo tamanho de palavra utilizado pela arquitetura do SDS-DIC. Assim, a mesma quantidade de BRAMs precisa ser empregada, mesmo que estejam sendo subutilizadas.

As simplificações dos blocos de controle e memórias proporcionaram um aumento na frequência de operação da arquitetura do QSDS-DIC, que pode operar a 213,37 MHz. No entanto, mesmo com o aumento na frequência de operação e a menor quantidade de ciclos necessários para gerar um vetor, o desempenho da arquitetura do QSDS-DIC no pior caso, é inferior ao obtido com a arquitetura do SDS-DIC. Isto ocorre, pois o QSDS-DIC possibilita um valor máximo de iterações duas vezes maior que a arquitetura do SDS-DIC. Mesmo assim, a arquitetura do QSDS-DIC pode processar até 45 quadros HD 1080p por segundo, desempenho superior ao necessário para processar vídeos desta resolução em tempo real.

6.2 Arquiteturas do SDS-DIC e QSDS-DIC com Somadores Compressores

As unidades de processamento (UP) do cálculo do SAD, presentes nas arquiteturas de EM, são formadas por uma árvore de somadores e subtratores em um *pipeline*. As UPs são replicadas diversas vezes dentro de uma arquitetura de EM para proporcionar a paralelização dos cálculos de SAD e, consequentemente, acelerar o processo de geração dos vetores de movimento. Esta característica pode ser evidenciada principalmente em arquiteturas de alto desempenho, que trabalham com vídeos de alta definição em tempo real. Esta estrutura arquitetural das UPs favorece a aplicação de somadores compressores, que podem acelerar o processo ao realizarem a soma de vários operandos no mesmo ciclo de *clock*.

Os circuitos somadores compressores podem realizar a soma de vários operandos simultaneamente. As estruturas dos somadores compressores 4-2, que permite realizar a soma de quatro operandos simultaneamente e compressores 8-2, que realizam a soma de oito operandos simultaneamente, foram utilizadas no desenvolvimento de UPs rápidas para as arquiteturas do SDS-DIC e QSDS-DIC. A estrutura de soma de múltiplos operandos, chamada 4-2 *Carry-Save Module*, que possibilitava a realização de somas simultâneas de quatro operandos, foi apresentada inicialmente em (WEINBERGER, 1981). Esta estrutura foi posteriormente aperfeiçoada em (OKLOBDZIJA, 1996). Esta estrutura contém uma combinação de células de somadores completos em conexão truncada na qual possibilita uma rápida compressão dos produtos parciais. O conceito da soma de múltiplos operandos em paralelo, usando o compressor 4-2, pode ser estendido para a soma de oito operandos simultaneamente. Os somadores compressores 8-1 utilizados neste trabalho foram obtidos com o uso de somadores compressores 4-1 estruturados de modo hierárquico.

6.2.1 Arquitetura da UP 4-2 e UP 8-2

Duas novas versões de UPs foram desenvolvidas utilizando somadores compressores, a versão que utiliza somadores compressores 4-1 foi chamade da UP 4-2, e a versão com somadores compressores 8-1 foi chamda de UP 8-2. Assim como na versão original da UP, a UP 4-2 também pode processar oito amostras paralelamente e a principal diferença está nos somadores utilizados. A Figura 6.6 apresenta a arquitetura da UP 4-2. São utilizados dois módulos de somadores compressores 4-2 para receber os

resultados parciais armazenados nos registradores após a barreira de subtratores. As entradas para os dois módulos de compressores 4-2 representam o módulo das subtrações realizadas em cada par de operandos de entrada da UP 4-2. O resultado final na UP 4-2 é calculado através da soma dos resultados produzidos pelas saídas de cada um dos módulos compressores 4-2. A realização de quatro somas em paralelo pelos módulos compressores 4-2 permite a eliminação de uma barreira de somadores, como pode ser visto comparando as Figuras 6.3 e 6.6. Desta forma, a arquitetura da UP 4-2 reduz em um ciclo de *clock* a latência da UP original.

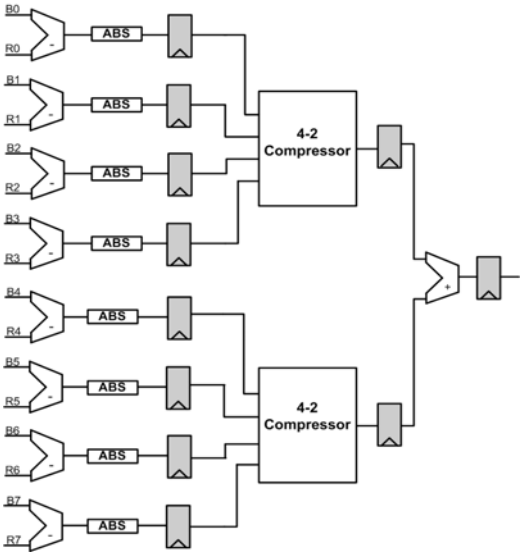


Figura 6.6: Arquitetura da UP 4-2

A redução da latência da UP pode ser ainda maior, com a utilização de somadores compressores 8-2 (UP 8-2). A Figura 6.7 apresenta a arquitetura da UP 8-2. Esta arquitetura possibilita a soma de oito amostras simultaneamente através da utilização do compressor 8-2. Desta forma, duas barreias de somadores podem ser eliminadas, reduzindo em dois ciclos de *clock* a latência da UP original. Como pode ser visto na Figura 6.7, após a primeira barreira de registradores, apenas um módulo do compressor 8-2 é necessário para calcular o resultado final do SAD.

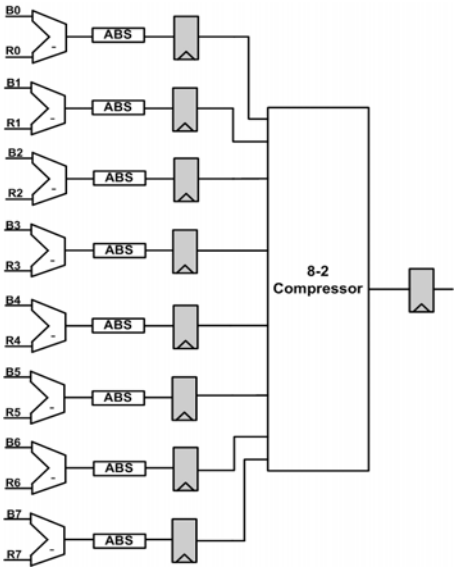


Figura 6.7: Arquitetura da UP 8-2

6.2.2 Resultados de Síntese com o uso de Somadores Compressores

As arquiteturas das UP 4-2 e UP 8-2 foram descritas em VHDL e integradas às arquiteturas do SDS-DIC e QSDS-DIC. Todas as arquiteturas foram sintetizadas para *standard cells* na tecnologia 0.18µm. A ferramenta *Encounter* RTL *Compiler* da Cadence (CADENCE, 2010) foi utilizada nesta etapa do processo. Estas versões das arquiteturas do SDS-DIC e QSDS-DIC não foram sintetizadas para FPGA, pois as vantagens do uso dos somadores compressores só podem ser observadas com a síntese ASIC. Nesta síntese, a frequência de operação utilizada foi a mínima necessária para atingir a taxa de processamento de 30 quadros HD 1080p por segundo, considerando o pior caso em cada arquitetura. Esta estratégia foi utilizada para otimizar os resultados de uso de recursos de hardware e consumo de potência.

A Tabela 6.3 apresenta os resultados de síntese para as arquiteturas do SDS-DIC e QSDS-DIC original e para as versões com UP 4-2 e UP 8-2. Os resultados de utilização de recursos de hardware são apresentados em número de transistores. A frequência de operação para processar 30 quadros HD 1080p, bem como os resultados de consumo de potência, também são apresentados para cada arquitetura.

O uso da UP 4-2 e da UP 8-2 possibilita uma pequena redução do número de transistores, sendo que a versão com a UP 8-2 é a que apresenta o menor número de transistores, tanto para a arquitetura do SDS-DIC quando para o QSDS-DIC. Como já era esperado, o uso das UPs com somadores compressores pode atingir a taxa de processamento para 30 quadros HD 1080p por segundo com uma frequência de operação mais baixa. Isto ocorre devido à redução de um ciclo de *clock* proporcionada pela UP 4-2, e de dois ciclos pela UP 8-2, em relação à UP original. A arquitetura do SDS-DIC com UP 8-2 pode processar vídeos HD 1080p em tempo real com uma frequência de operação inferior a 99MHz. As arquiteturas do QSDS-DIC operam com frequências mais altas, devido ao DIC, que permite o dobro de iterações para cada vetor de movimento. Os resultados de potência mostram que todas as versões das arquiteturas do SDS-DIC e QSDS-DIC podem processar vídeos HD 1080p em tempo real com um consumo de potência de aproximadamente 40mW. A arquitetura do QSDS-DIC original apresentou o menor consumo de potência, cerca de 32,9mW.

Tabela 6.3: Resultados de síntese das arquiteturas do SDS-DIC e QSDS-DIC com somadores compressores

Arquitetura	Transistores (k)	Frequência (MHz)	Potência (mW)
SDS-DIC	187	104	39,39
SDS-DIC com UP-42	170	101,0	41,10
SDS-DIC com UP-82	164	98,1	40,02
QSDS-DIC	176	141,1	32,92
QSDS-DIC com UP-42	156	136,0	37,24
QSDS-DIC com UP-82	148	130,7	35,66

6.3 Arquitetura do SDS-DIC para Múltiplos Quadros de Referência

Outra versão da arquitetura do SDS-DIC foi desenvolvida implementando uma das ferramentas presentes no padrão H.264/AVC, o uso de múltiplos quadros de referência. Esta arquitetura foi denominada *Multiple Reference Sub-sampled Diamond Search with Dynamic Iteration Control* (MRSDS-DIC) e utiliza três quadros passados como quadros de referência para a geração dos vetores de movimento. A Figura 6.8 ilustra o processo de EM com três quadros passados de referência.

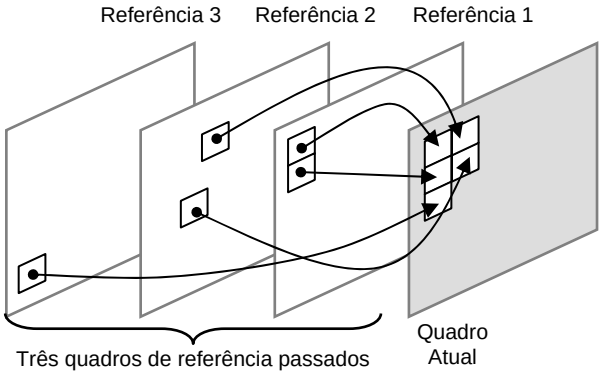


Figura 6.8: Estimação de movimento com três quadros (passados) de referência

A arquitetura desenvolvida para o algoritmo SDS-DIC com múltiplos quadros de referência (MRSDS-DIC) está ilustrada na Figura 6.9. As principais diferenças da MRSDS-DIC em relação à arquitetura original do SDS-DIC estão nas memórias internas da arquitetura. Duas novas memórias internas, idênticas às apresentadas na arquitetura original, foram adicionadas para o armazenamento das áreas de busca dos quadros de referência adicionais (Referência 2 e Referência 3 na Figura 6.8). Um novo conjunto de acumuladores e um novo comparador foram adicionados para o armazenamento e a escolha do melhor resultado de SAD entre os três quadros de referência. Uma nova linha de multiplexadores foi introduzida para selecionar qual das memórias enviará seus dados às UPs. Com a inserção de hardware e memórias adicionais, o controle da arquitetura foi completamente modificado.

As unidades de processamento são compartilhadas entre as três memórias dos quadros de referência. Ou seja, a busca deve ser finalizada para o primeiro quadro para ter início no segundo quadro de referência. Desta forma, o número de ciclos de *clock* necessários para a geração de um vetor de movimento é três vezes maior do que o apresentado no capítulo 6.5, para a arquitetura original. Esta estratégia foi adotada para manter a baixa utilização de recursos de hardware, no entanto, a exploração do paralelismo máximo, com um *datapath* de UPs para cada memória, pode ser uma alternativa para alto desempenho.

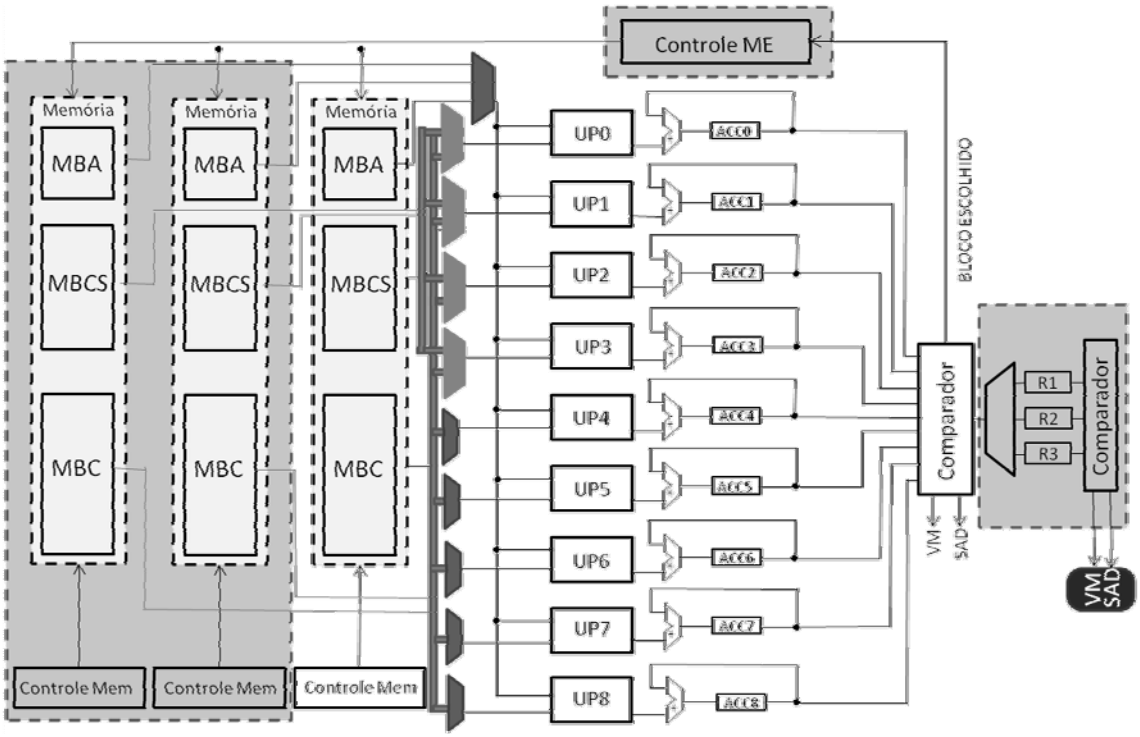


Figura 6.9: Diagrama em blocos da arquitetura do MRSDS-DIC

6.3.1 Resultados de Síntese da Arquitetura do MRSDS-DIC

A arquitetura do MRSDS-DIC foi descrita em VHDL utilizando a ferramenta ISE 8.1i da XILINX. A ferramenta ModelSim 6.1 da Mentor Graphics foi utilizada para simulação e validação da arquitetura. A Tabela 6.4 apresenta os resultados de síntese da arquitetura desenvolvida para o FPGA Virtex4 XC4VLX200 da XILINX.

Tabela 6.4: Resultados de síntese da arquitetura do MRSDS-DIC

Parâmetros	Resultados de síntese
Flip-Flops	4597
LUTs	7312
BRAM	96
Slices	4154
CLBs	1038
Frequência	159,3MHz
Quadros HD 1080p (caso médio)	39
Quadros HD 1080p (pior caso)	16

A arquitetura desenvolvida utilizou um baixo número de recursos de hardware do dispositivo, apenas 7,3K LUTs. A utilização de BRAM foi exatamente três vezes maior do que a arquitetura original, devido à inserção das novas memórias para os quadros de referência. A frequência máxima de operação da arquitetura do MRSDS-DIC foi significativa, permitindo uma taxa de processamento de 39 quadros HD 1080p por segundo no caso médio e 16 quadros no pior caso. A utilização das mesmas unidades de processamento para todas as área de pesquisa reduz o desempenho da arquitetura do

MRSDS-DIC em mais de três vezes em relação a arquitetura original. Uma solução explorando o paralelismo das unidades de processamento pode aumentar o desempenho da arquitetura, tornando possível a codificação de vídeo HD 1080p em tempo real no pior caso de execução.

6.4 Arquitetura Dedicada para o Algoritmo MPDS

O algoritmo MPDS apresenta uma característica importante para o seu desenvolvimento em hardware: a possibilidade de exploração do paralelismo. Os cinco núcleos de busca do algoritmo MPDS, que executam o algoritmo DS, podem operar em paralelo, e desta forma, o desempenho da arquitetura do algoritmo MPDS pode ser praticamente igual ao de uma arquitetura para algoritmos DS. No entanto, a implementação em hardware do algoritmo DS necessita de alguma restrição quanto o número de iterações, para garantir o desempenho desejado e também facilitar o processo de integração com outros módulos que compõe o codificador de vídeo.

Para avaliar as perdas em qualidade causadas pela inserção da restrição de iterações nos algoritmos DS e MPDS, uma nova bateria de testes foi realizada, utilizando as 10 sequências de tese HD 1080p apresentadas anteriormente. A Tabela 6.5 apresenta os resultados médios de qualidade (PSNR) e custo computacional (BCCs e comparações) para os algoritmos DS e MPDS em diferentes configurações para blocos de 16x16 pixels. Os algoritmos DS e MPDS foram avaliados com restrição de cinco e 11 iterações, juntamente com o uso de subamostragem de pixel de 4:1. A restrição de cinco iterações permite que o algoritmo repita no máximo cinco vezes o padrão de busca e, mesmo que o critério de parada não seja satisfeito, o refinamento final será aplicado.

As versões com 11 iterações apresentam pequenas perdas em qualidade, cerca de 0,3dB para ambos algoritmos, e o custo computacional apresenta uma ligeira redução. Com a restrição de cinco iterações, as perdas em qualidade são mais significativas, aproximadamente 1,2dB para o algoritmo MPDS. Mesmo assim, o resultado de qualidade obtido pelo algoritmo MPDS é superior ao obtido pelo algoritmo DS, sem o uso de restrição de iterações. A redução do custo computacional é mais significativa com a restrição de cinco iterações, cerca de 21% no número de comparações para o algoritmo MPDS.

Tabela 6.5: Resultados de qualidade e custo computacional para os algoritmos DS e MPDS com restrição de iterações

Algoritmos	PSNR (dB)	BCCs ($x10^6$)	Comp. ($x10^9$)
MPDS/4:1	34,51	290	18,56
DS/4:1	32,76	46,41	2,97
MPDS/4:1 – 11 iterações	34,19	276,91	17,72
DS/4:1 – 11 iterações	32,43	45,1	2,89
MPDS/4:1 – 5 iterações	33,28	227,96	14,59
DS/4:1 – 5 iterações	31,42	41,3	2,64

A arquitetura para o algoritmo MPDS foi desenvolvida a partir de um núcleo de hardware que implementa o algoritmo DS. Duas versões da arquitetura do MPDS foram desenvolvidas, uma com restrição de cinco iterações (versão 1) e outra com restrição de

11 iterações (versão 2). As duas arquiteturas são muito similares, apresentando diferenças apenas no módulo de controle. O número de iterações da arquitetura tem influência direta nos resultados de qualidade, como apresentado na Tabela 6.5, além disso, também causa impacto no desempenho, pois um número maior de ciclos de *clock* é necessário para executar mais iterações. A versão da arquitetura do MPDS com cinco iterações visa o maior desempenho, enquanto a versão com 11 iterações visa a maior qualidade.

O diagrama de blocos da arquitetura desenvolvida para o algoritmo DS está apresentada na Figura 6.10. Uma memória interna (Memória de Referência) de 9,2Kbits (34x34 amostras) armazena a região do quadro de referência que contêm todos os blocos candidatos para uma execução do primeiro LDSP, as cinco iterações e também todos os blocos candidatos para o refinamento final. Um conjunto de 14 memórias internas com quatro linhas de 128bits, totalizando 512bits (8x8 pixels) cada, armazena os nove blocos do LDSP (MEM1 a MEM9), o bloco do quadro atual e os quatro blocos do SDSP (MEMa a MEMd). Estas memórias estão organizadas em quatro linhas de 16 amostras cada (16x4), esta organização é utilizada para fornecer duas linhas do bloco candidato por ciclo de *clock* para as UPs. No total, a arquitetura do DS utiliza 16,4Kbits de memória interna, 9,2Kbits para a memória de referência e 7,2Kbits para as memórias dos blocos candidato e de referência. Um conjunto de multiplexadores entrega os dados corretos para as unidades de processamento (UP1 a UP9), caso o controle decida por executar um LDSP ou um SDSP. Esta arquitetura pode operar com o máximo paralelismo do algoritmo DS, ou seja, pode calcular os nove blocos candidatos do LDSP em paralelo.

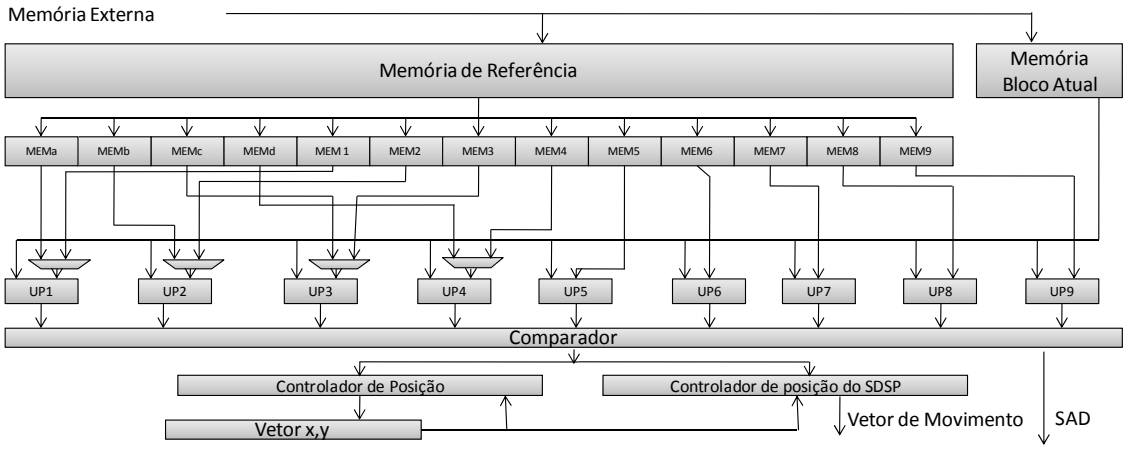


Figura 6.10: Diagrama em blocos da arquitetura do algoritmo DS

A arquitetura do DS necessita de 34 ciclos para preencher a Memória de Referência, no entanto, para reduzir a latência de acesso à memória externa, a leitura dos dados não é regular. Os primeiros dados lidos da memória externa são aqueles necessários para o primeiro LDSP, desta forma, as unidades de processamento podem começar a operar, ao mesmo tempo em que a memória de Referência é preenchida com os demais dados necessários para as próximas iterações (cinco na versão 1 e 11 na versão 2).

Quando a Memória de Referência estiver preenchida com todos os dados necessários para o primeiro LDSP, as 13 memórias locais (MEM1 a MEM 9 e MEMa a MEMb) conterão os blocos candidatos para o LDSP e SDSP. Quando o algoritmo decidir executar o SDSP, os blocos candidatos já estarão disponíveis nas memórias

MEMa a MEMd. Desta forma, não haverá latência de acesso à memória para a execução do SDSP.

Nove unidades de processamento (UP) são utilizadas, uma para cada bloco candidato de um LDSP, desta forma, os nove blocos candidatos do LDSP podem ser calculados em paralelo. Para reduzir o custo de hardware, quatro destas UPs são também utilizadas para o cálculo do SDSP. As UPs foram desenvolvidas com uma árvore de somadores em um pipeline de cinco estágios. Cada UP pode processar duas linhas do bloco de 16x16 subamostrado (8x8 pixels) por ciclo de *clock*, logo, quatro acumulações são necessárias para gerar o resultado final de SAD para um bloco candidato.

O comparador envia a informação de posição do bloco de menor SAD para o Controlador de Posição (Figura 6.10), que é o responsável por atualizar a nova posição deste bloco. Este passo é necessário para definir se o bloco escolhido estava no centro, em uma aresta ou em um vértice do LDSP. Caso o bloco de menor SAD seja encontrado no centro (UP5), o SDSP é aplicado, caso contrário, uma nova iteração será executada. Para as novas iterações, não existe latência de acesso à memória, pois a memória de Referência já contêm todos os dados necessários para as cinco iterações. A Memória de Referência e as UPs necessitam de 28 (34 ciclos para o primeiro vetor devido à latência de memória) ciclos de *clock* para a geração dos SADs para o primeiro LDSP. Mais 14 ciclos são necessários para o SDSP, logo, no melhor caso, a arquitetura do DS pode gerar um vetor de movimento em 42 (48 para o primeiro vetor) ciclos de *clock*. Cada nova iteração utiliza 27 ciclos de *clock*. Considerando a restrição de cinco iterações, 183 ciclos de *clock* são necessários para a geração do primeiro vetor movimento, e 169 ciclos para os demais vetores, no pior caso.

A arquitetura do algoritmo MPDS utiliza cinco instâncias da arquitetura apresentada na Figura 6.10, uma para a região central e mais uma para cada setor. A Figura 6.11 ilustra o diagrama de blocos da arquitetura desenvolvida para o algoritmo MPDS. Cada instância da arquitetura do algoritmo DS foi chamada de Núcleo, sendo que o Núcleo C, representa o núcleo de busca central, e os Núcleos 1 a 4 representam os núcleos de busca dos setores 1 a 4. A hierarquia da arquitetura do MPDS é relativamente simples, pois cada núcleo trabalha de maneira independente, e assim que suas buscas encerram, cada núcleo escreve o seu resultado final no registrador de entrada do comparador que, então, define qual dos núcleos de busca resultou no menor resultado de SAD. O comparador na Figura 6.11 é responsável por encontrar o melhor bloco candidato dentre os resultados obtidos pelos cinco núcleos de busca.

A arquitetura do MPDS começa seu processamento preenchendo a memória de referência do Núcleo C. Assim que a memória de referência do Núcleo C estiver cheia, a arquitetura inicia a leitura dos dados para os demais Núcleos (1, 2, 3 e 4), enquanto inicia o processamento do primeiro LDSP para o núcleo C. Este processo é repetido para os demais núcleos, até que a memória de referência do Núcleo 4 esteja preenchida.

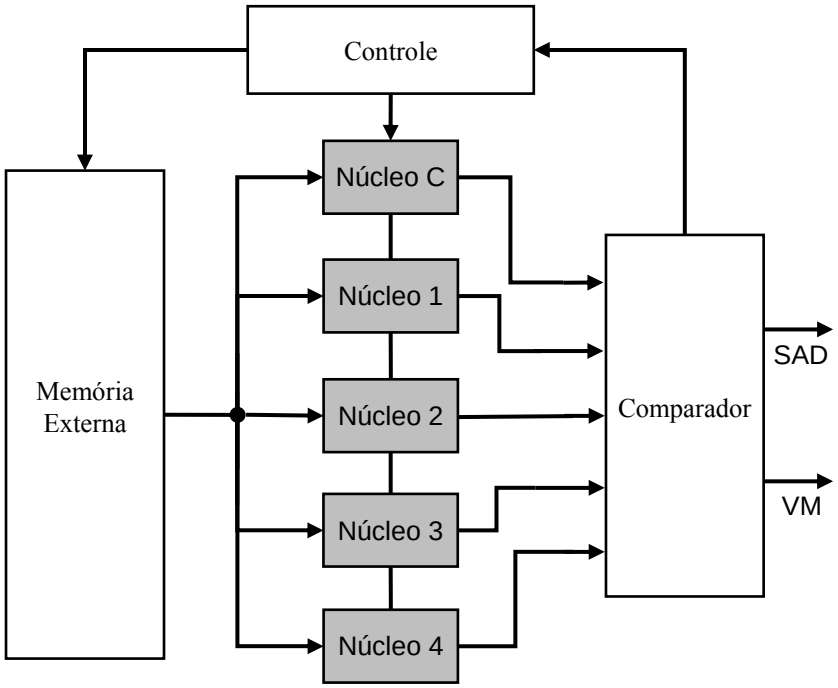


Figura 6.11: Arquitetura do algoritmo MPDS

A primeira versão da arquitetura do MPDS pode armazenar todos os dados necessários para as suas cinco iterações em suas memórias internas (memórias de referência dos núcleos DS). Assim que a arquitetura terminar o preenchimento da memória de referência do Núcleo 4, o Núcleo C inicia novamente o processo de leitura dos dados para a geração do próximo vetor de movimento. A arquitetura do MPDS possui uma latência de memória de 170 ciclos para preencher as memórias de referência dos cinco núcleos. Mais 135 ciclos são necessários para que o Núcleo 4 termine o processamento de seus blocos candidatos, 14 para o cálculo do SDSP, três ciclos para o comparador encontrar o melhor resultado entre os cinco núcleos e mais um ciclo é necessário para atualizar os resultados na saída da arquitetura. Para a geração do primeiro vetor de movimento são necessários 323 ciclos, no entanto, os demais vetores de movimento necessitam de apenas 170 ciclos. Isto é possível, pois quando o primeiro vetor de movimento é gerado, os Núcleos C, 1, 2 e 3 já estão gerando os resultados para o próximo bloco do quadro atual. A Figura 6.12 apresenta o diagrama de ciclos da arquitetura do MPDS, onde o primeiro vetor de movimento é gerado no ciclo 323, e o segundo vetor é gerado no ciclo 493.

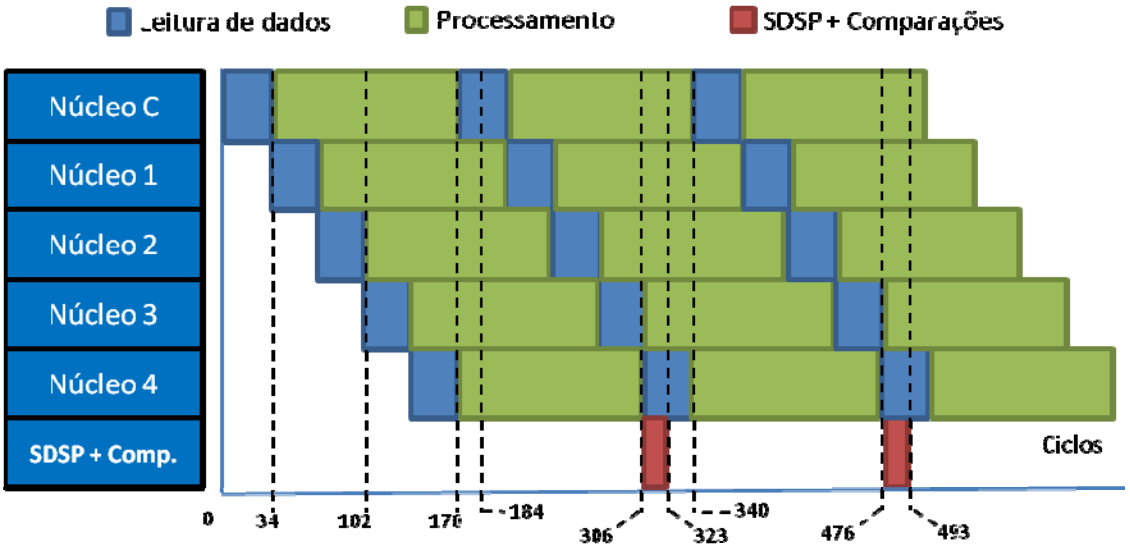


Figura 6.12: Diagrama de ciclos da arquitetura do MPDS

A segunda versão da arquitetura do MPDS precisa atualizar as memórias de referência de todos os núcleos para executar as 11 iterações. Após o término dos cálculos, cada núcleo inicia um novo processo de leitura de dados, para a execução de mais seis iterações. Seis iterações são possíveis na segunda etapa devido ao fato do primeiro LDSP não ser considerado como iteração, desta forma, a primeira carga da memória contêm os dados para o primeiro LDSP e mais cinco iterações, e a segunda carga contêm os dados para seis novas iterações. Devido à necessidade de atualização das memórias de referência, a segunda versão da arquitetura do MPDS utiliza 340 ciclos para a geração de um vetor de movimento. A latência para a geração do primeiro vetor de movimento é de 493 ciclos. O diagrama de ciclos da Figura 6.12 também é utilizado para a segunda versão da arquitetura do MPDS, sendo que apenas o primeiro vetor de movimento é representado no ciclo 493, o segundo vetor será gerado no ciclo 833.

As duas versões desenvolvidas para o algoritmo MPDS são capazes de manter os seus núcleos de busca operando a maior parte do tempo, com pequenas interrupções para leitura de dados. Devido à organização das memórias internas, e da leitura não linear dos dados, poucos ciclos são necessários para disponibilizar os dados às unidades de processamento. Estas arquiteturas podem explorar de maneira eficiente os recursos de hardware utilizados, atingindo melhores resultados de desempenho e consumo de energia.

6.4.1 Resultados de Síntese da Arquitetura do MPDS

A arquitetura do algoritmo MPDS foi descrita em VHDL e sintetizada para o dispositivo EP4S40G2F40I2 da família Stratix 4 da Altera (ALTERA, 2010) e também para ASIC com síntese para *standard cells* na tecnologia 90nm da TSMC, utilizando a ferramenta Synopsys (SYNOPSYS, 2011). As duas versões da arquitetura do MPDS descritas anteriormente foram sintetizadas, sendo que a primeira versão, com restrição de cinco iterações, foi chamada de "V. 1", e a segunda versão, com 11 iterações, foi chamada de "V.2". Os resultados de síntese em FPGA para as duas versões da arquitetura do MPDS são apresentados na Tabela 6.6.

As duas arquiteturas apresentam resultados muito similares quanto à utilização recursos de hardware do FPGA (ALUTs e registradores), além de apresentarem o mesmo número de bits de memória interna. As arquiteturas do MPDS utilizam cinco

instâncias da arquitetura do DS, que utilizam 16,4Kbits de memória cada, totalizando 82Kbits de memória interna. No entanto, divido a simplificações realizadas pela ferramenta de síntese para FPGA, e também devido à conversão de algumas memórias para registradores, os resultados de uso de memória apresentados na Tabela 6.6 são de apenas 46,2Kbits.

A principal diferença está no desempenho apresentado por cada versão da arquitetura. A primeira versão da arquitetura do MPDS necessita de metade do número de ciclos utilizados pela segunda versão, além disso, pode operar com uma frequência mais elevada. Isto faz com que o desempenho da arquitetura do MPDS V.1 seja mais de duas vezes superior ao obtido pela arquitetura do MPDS V. 2. Mesmo assim, as duas versões da arquitetura apresentam taxa de processamento superior ao necessário para codificar vídeos HD 1080p em tempo real a 30 quadros por segundo, sendo que a MPDS V.1 pode processar até mesmo vídeos QFHD a mais de 30 quadros por segundo.

É importante salientar que além dos resultados de síntese, as duas versões da arquitetura do algoritmo MPDS apresentam diferenças significativas de qualidade. O algoritmo MPDS com restrição e 11 iterações apresenta um ganho médio de 0,91dB, em relação a restrição de cinco iterações, como apresentado na Tabela 6.5. Desta forma, as versões desenvolvidas para a arquitetura do algoritmo MPDS podem ser utilizadas para atingir diferentes objetivos. A primeira versão atinge o máximo desempenho, podendo ser utilizada para aplicações em tempo real até mesmo para vídeos QFHD, assumindo maiores perdas em qualidade. A segunda versão atinge os melhores resultados de qualidade, com uma grande redução no desempenho que, no entanto, ainda é suficiente para processar vídeos HD 1080p em tempo real.

Tabela 6.6: Resultados de síntese da arquitetura do MPDS para FPGA

Parâmetros	**MPDS V. 1**	**MPDS V.2**
Frequência (MHz)	199,2	185,2
ALUTs	34554	34578
Registradores	44447	44476
Bits de memória	46240	46240
Ciclos por bloco	170	340
Quadros HD 1080p por segundo	145	67
Quadros QFHD por segundo	36	17

Os resultados da síntese ASIC foram gerados para a mínima frequência necessária para processar 30 quadros HD 1080p por segundo, sendo que para a arquitetura do MPDS V.1 também foi realizada uma síntese considerando a frequência mínima para processar 30 quadros QFHD por segundo. Esta estratégia foi utilizada para otimizar os resultados de consumo de potência para uma resolução de vídeo específica. A arquitetura do MPDS V.1 pode atingir a taxa de processamento de 30 quadros por segundo nas resoluções HD 1080p e QFHD com frequências de 41,3MHz e 165,2MHz, respectivamente. Para a arquitetura do MPDS V.2, a frequência mínima para processar 30 quadros HD 1080p por segundo é de 82.6MHz. Mais uma vez, os resultados de uso de recursos de hardware para as duas arquiteturas são muito similares, a diferença mais significativa está no consumo de potência. Devido à baixa frequência de operação necessária, e da eficiência energética da tecnologia de 90nm, a arquitetura do MPDS V.

1 pode processar vídeos HD 1080p em tempo real com um consumo de potência de apenas 4,5mW. Para a arquitetura do MPDS V.2 o consumo de potencia é praticamente o dobro, devido à duplicação da frequência de operação. Estas frequências de operação são relativamente baixas para células na tecnologia 90nm, logo, os resultados de número de *gates* (equivalentes a NAND de duas entradas) utilizadas é o mesmo para as sínteses da arquitetura MPDS V.1 para HD 1080p e QFHD. Isto indica que as mesmas células foram utilizadas para implementar as duas versões da arquitetura. No entanto, o aumento de quatro vezes na frequência de operação ocasiona um aumento de mesma ordem na potência consumida, para a síntese focada em QFHD.

Tabela 6.7: Resultados de síntese da arquitetura do MPDS para TSMC 90nm

Parâmetros	MPDS V. 1 HD 1080p	MPDS V. 1 QFHD	MPDS V.2
Frequência (MHz)	41,3	165,2	82,6
Área (*gates*)	50028	50028	50239
Bits de memória	82080	82080	82080
Potência (mW)	4,5	18,04	9
Ciclos por bloco	170	170	340
Quadros HD 1080p por segundo	30	-	30
Quadros QFHD por segundo	-	30	-

6.4.2 Arquitetura para o algoritmo DMPDS

Uma arquitetura para o algoritmo DMPDS também foi implementada, usando como base a arquitetura do MPDS apresentada. Esta arquitetura opera sobre o mesmo tamanho de bloco (16x16) e com o mesmo nível de subamostragem de pixel (4:1) utilizados na arquitetura original. Apenas a versão com restrição de cinco iterações foi implementada para o algoritmo DMPDS. As duas arquiteturas apresentam o mesmo *template* arquitetural, com diferenças apenas na unidade de controle e a inserção de uma unidade que define dinamicamente o valor de *d* (parâmetro de distância dos núcleos de busca multiponto). Esta arquitetura para o algoritmo DMPDS foi sintetizada para o mesmo dispositivo FPGA utilizado na síntese da arquitetura do MPDS, apresentada na Tabela 6.6. A Tabela 6.8 apresenta os resultados de síntese da arquitetura do algoritmo DMPDS.

Tabela 6.8: Resultados de síntese da arquitetura do algoritmo DMPDS

Parâmetros	DMPDS
Frequência (MHz)	187,58
ALUTs	34500
Registradores	44500
Bits de memória	46240
Ciclos por bloco	170
Quadros HD 1080p por segundo	136
Quadros QFHD por segundo	34

As alterações na arquitetura causam uma ligeira redução na frequência máxima de operação, o que implicou em uma pequena redução no desempenho. No entanto, o algoritmo DMPDS com restrição de cinco iterações, operando sobre blocos 16x16 e subamostragem de pixel de 4:1, apresenta um ganho médio de PSNR de 0,39dB sobre o algoritmo MPDS com a mesma configuração. Além disso, os resultados de desempenho da arquitetura do algoritmo DMPDS são suficientes para a codificação de vídeos HD 1080p, e até mesmo vídeos QFHD, em tempo real a 30 quadros por segundo.

6.5 Arquitetura Dedicada para o Algoritmo *Hardware Oriented - Multiple Iterative Random Search* (HO-MIRS)

A etapa aleatória dos algoritmos de EM desenvolvidos pode ser facilmente implementada em hardware, no entanto, as etapas de refinamento iterativo apresentam os mesmos problemas de implementação em hardware presentes na maior parte dos algoritmos rápidos: dependência de dados e irregularidade no acesso a memória. Pensando na maior eficiência da arquitetura, uma versão orientada a hardware (HO - *Hardware Oriented*) foi desenvolvida para o algoritmo MIRS. Nesta versão, a etapa de refinamento final iterativo, aplicada ao melhor bloco candidato escolhido em cada setor, foi eliminada. Desta forma, o aceso a memória externa se torna mais regular, e a possibilidade de paralelismo aumenta, pois não existe dependência de dados em toda a etapa aleatória. A Figura 6.13 apresenta o fluxograma do algoritmo HO-MIRS, que difere do algoritmo MIRS original pela ausência do refinamento final iterativo aplicado aos melhores blocos candidatos encontrados em cada setor.

Com a remoção do refinamento final iterativo, o algoritmo HO-MIRS naturalmente apresenta perdas de qualidade em relação ao algoritmo MIRS original. No entanto, com a eliminação da dependência de dados do refinamento final iterativo, é possível aumentar o paralelismo da etapa aleatória, e desta forma, utilizar valores maiores de N, mantendo o desempenho e melhorando os resultados de qualidade. O algoritmo HO-MIRS também necessita de uma restrição no número de iterações do algoritmo DS (aplicado ao centro da área de busca). Esta restrição também implica em perdas de qualidade, no entanto, é essencial para garantir um número de ciclos máximo para a arquitetura, garantindo a taxa de processamento e facilitando a sua integração com os demais blocos do codificador de vídeo.

A Tabela 6.9 apresenta os resultados médios (para as 10 sequências de teste HD 1080p) de qualidade e custo computacional para os algoritmos MIRS e HO-MIRS. Os resultados apresentados foram gerados para uma área de busca de [-48, +48] e utilizam blocos de tamanho 16x16 com subamostragem de pixel de 4:1. Para o algoritmo MIRS foi utilizado N = 16, e para o algoritmo HO-MIRS, N = 96. O algoritmo HO-MIRS também utiliza restrição de cinco iterações para o algoritmo DS aplicado ao centro da área de busca.

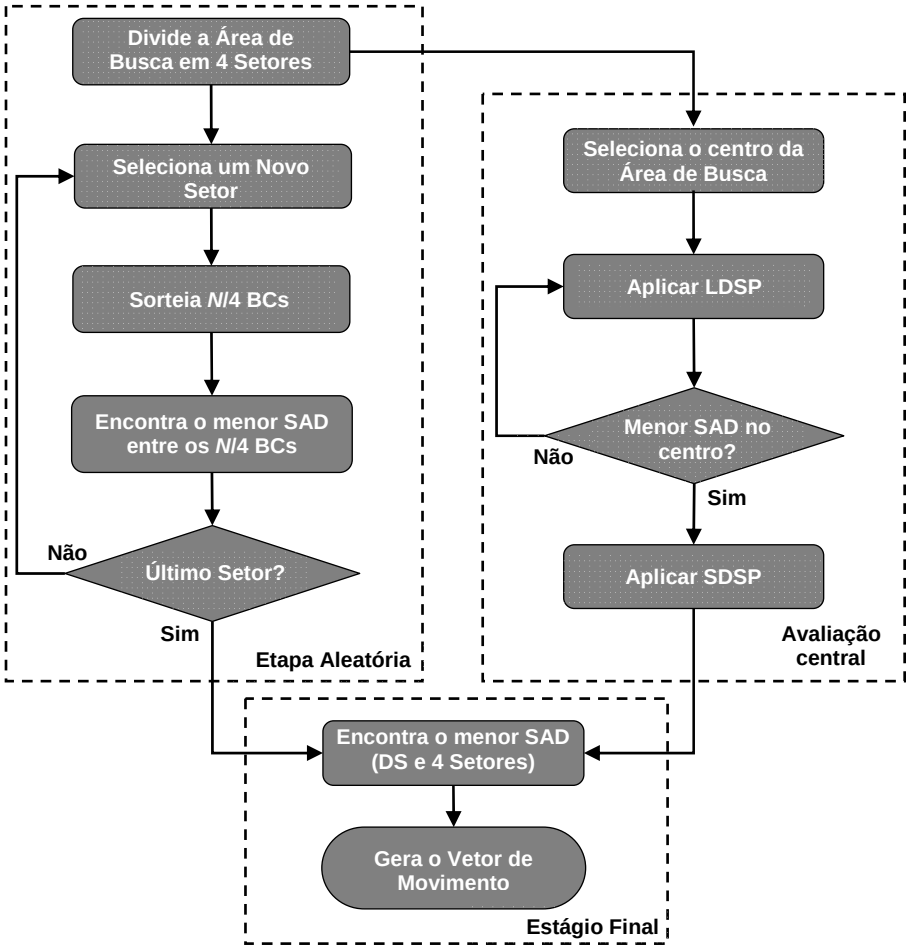


Figura 6.13: Fluxograma do algoritmo HO-MIRS

Tabela 6.9: Resultados médios de qualidade e custo computacional para os algoritmos MIRS e HO-MIRS

Parâmetros	MIRS	HO-MIRS
PSNR (dB)	34,41	34,03
PRR (%)	59,5	65,09
BCCs x 10^6	114,86	193,54

Apesar do aumento significativo no valor de N utilizado, com a restrição do número de iterações do DS e remoção do refinamento final iterativo, o algoritmo HO-MIRS apresenta uma perda de PSNR de 0,38dB. Esta perda é apenas um terço da perda apresentada pelo algoritmo MPDS (Tabela 6.5) com o uso da restrição de cinco iterações. Mesmo com perdas em PSNR, os resultados de redução de resíduo do algoritmo HO-MIRS são superiores. Isto demonstra que o aumento do valor de N pode proporcionar a escolha de blocos candidatos de menor SAD, no entanto, estes blocos não necessariamente apresentam menor MSE (métrica utilizada no cálculo do PSNR). O algoritmo HO-MIRS apresenta um grande acréscimo no número de BCCs, no entanto, e etapa aleatória do algoritmo HO-MIRS possibilita a livre exploração do paralelismo entre os N blocos candidatos sorteados. Esta característica é muito importante para o desenvolvimento em hardware, e possibilita a obtenção de melhores resultados de desempenho, mesmo calculando um número muito maior de BCCs.

O diagrama de blocos da arquitetura de hardware desenvolvida para o algoritmo HO-MIRS é apresentado na Figura 6.14. A arquitetura está dividida em dois grandes blocos: Núcleo DS e Núcleo Aleatório. Estes dois núcleos compartilham o acesso a uma única memória de referência interna de 55x55 bytes. Esta memória contêm todos os blocos candidatos necessários para as cinco iterações do algoritmo DS, além de todos os blocos candidatos da etapa aleatória do algoritmo HO-MIRS. Esta memória armazena a área de busca subamostrada (4:1), correspondente a uma área de busca de [-47, +47]. Esta área é um pixel menor, em cada direção do *range*, em relação à área de busca escolhida para as avaliações em software. Esta pequena redução foi necessária para garantir o sincronismo de acesso à memória de referência pelos Núcleos DS e Aleatório. busca. Após a leitura dos dados os dois núcleos podem processar seus blocos candidatos de modo independente, e em paralelo. O Núcleo DS tem acesso privilegiado à memória de referência, pois a sua execução é o gargalo de processamento do algoritmo HO-MIRS.

O Núcleo DS é composto por 13 memórias de bloco candidato (Mem 1 a Mem 9 e Mem A a Mem D) de 8x8 bytes para o armazenamento dos nove blocos candidatos do LDSP e os quatro blocos candidatos do SDSP, totalizando 6,7Kbits de memória. Nove UPs são utilizadas para que todos os blocos candidatos do LDSP possam ser calculados em paralelo. Um comparador de cinco estágios é utilizado para encontrar o melhor bloco candidato. Esta estrutura é muito similar à utilizada no Núcleo DS da arquitetura do MPDS, apresentada na seção 6.4, no entanto, este Núcleo DS não possui memória de referência interna e suas UPs processam apenas uma linha de cada bloco candidato por ciclo.

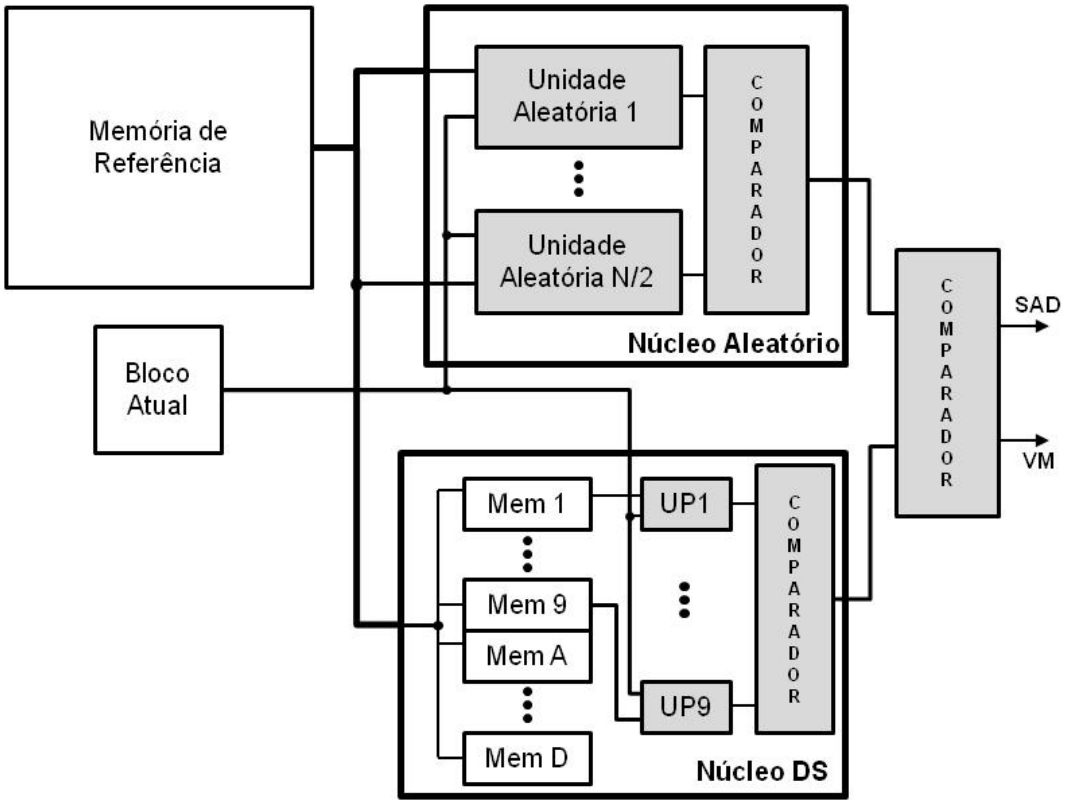


Figura 6.14: Diagrama de blocos da arquitetura do algoritmo HO-MIRS

O Núcleo Aleatório é composto por 48 Unidades Aleatórias (UA) e um comparador de sete estágios, responsável por comparar os resultados dos 48 blocos candidatos dos

dois setores superiores da área de busca. Cada UA contêm uma memória de bloco candidato (BC) de 8x8 bytes, uma UP (igual à utilizada no Núcleo DS) e um *Linear Feedback Shift Register* (LFSR) (BEKER, 1983) de 32 bits cada. Os LSFRs são utilizados para a geração dos números pseudo-aleatórios para o sorteio dos blocos candidatos. Dos 32 bits, apenas os últimos 10 são utilizados, cinco para a posição de X, e outros cinco para a posição de Y do bloco candidato sorteado. Esta estratégia foi utilizada para garantir uma aleatoriedade mais efetiva aos números gerados, já que o LFSR sempre gera os números em uma mesma sequência, e a frequência de repetição desta sequência é proporcional ao número de bit utilizados. A Figura 6.15 apresenta o diagrama de bloco simplificado da UA utilizada no Núcleo Aleatório.

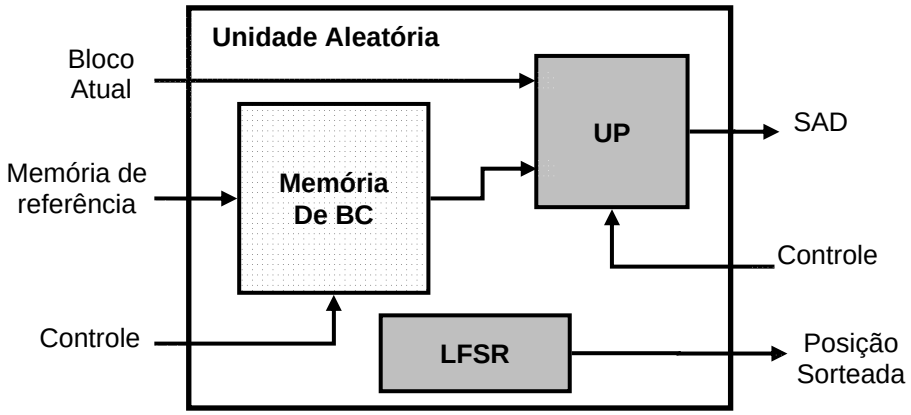


Figura 6.15: Diagrama de blocos da arquitetura da Unidade Aleatória

A arquitetura do algoritmo HO-MIRS começa o seu processamento preenchendo os dados da memória de referência. Para otimizar o processamento do Núcleo DS, a memória de referência começa a ser preenchida pela região central, disponibilizando todos os dados necessários para o primeiro LDSP. Após a primeira linha ser escrita na memória de referência, o Núcleo DS inicia o processo de leitura, preenchendo as suas memórias de blocos candidatos (Mem). Assim que as memórias de bloco estiverem cheias, as UPs do Núcleo DS iniciam o processamento dos SADs dos blocos candidatos. Em paralelo, os LFSRs da unidade aleatória geram os números para os 48 blocos candidatos aleatórios, correspondentes aos dois setores superiores da área de busca. As UPs utilizam um pipeline de quatro estágios e processam uma linha do bloco candidato a cada ciclo, logo, mais sete acumulações são necessárias para a geração do SAD de um bloco candidato, totalizando 11 ciclos.

Para tornar o acesso à memória de referência do Núcleo Aleatório mais regular, a memória é lida sequencialmente a partir da primeira linha, até a última linha que compõe os blocos candidatos dos setores superiores da área de busca (32 linhas no total). Uma unidade de controle dentro da unidade aleatória define se a linha lida contém algum bloco candidato dentre os 48 sorteados pelos LFSRs, e define os bits corretos que devem ser armazenados nas memórias de bloco das URs. O mesmo processo é executado para os dois setores inferiores, onde as 32 linhas da memória de referência necessárias para compor os blocos candidatos são lidas sequencialmente. O número total de linhas lida (64) é superior ao número de linhas da memória de referência, pois existem linhas comuns para blocos candidatos dos setores superiores e inferiores da área de busca. Esta estratégia é utilizada, pois o acesso à memória de referência pode ser muito irregular, dependendo das posições dos blocos candidatos sorteados. Sem a leitura completa da memória, uma etapa de processamento dos blocos

candidatos sorteados seria necessária, para identificar quais linhas da memória deveriam ser lidas, e em quais delas poderia haver mais de um bloco sorteado. Além disso, o número de acesso à memória não seria fixo, dificultando o compartilhamento da memória de referência com o Núcleo DS.

O acesso à memória de referência não ocorre em intervalos regulares entre os núcleos e, como mencionado anteriormente, privilegia o acesso ao Núcleo DS, que é o gargalo de desempenho da arquitetura do HO-MIRS. A Figura 6.16 apresenta o diagrama de ciclos de *clock* da arquitetura do HO-MIRS, apresentando os tempos em que cada núcleo está acessando a memória de referência, processando os SADs ou parado. O Núcleo DS inicia o processo de leitura da memória de referência, e após o início dos cálculos de SAD dos blocos candidatos do primeiro LDSP, o Núcleo Aleatório inicia o processo de leitura da memória de referência, a partir da primeira linha. Este processo de leitura é interrompido assim que o Núcleo DS termina os cálculos do primeiro LDSR, e inicia novamente a leitura dos dados necessários para a primeira iteração. Este processo se repete, e o Núcleo Aleatório utiliza os ciclos de processamento do Núcleo DS para acessar a memória de referências e preencher a memórias de bloco das suas URs.

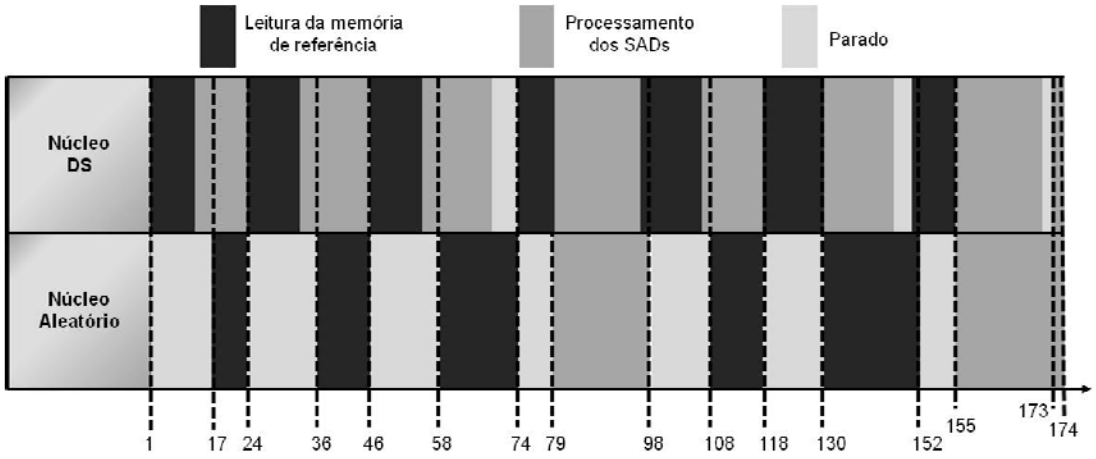


Figura 6.16: Diagrama de ciclos para a arquitetura do algoritmo HO-MIRS

O Núcleo Aleatório possui apenas duas etapas de processamento de SAD, uma para os 48 blocos candidatos dos setores superiores, e outra para os 48 blocos candidatos dos setores inferiores da área de busca. Estas duas etapas foram sincronizadas para serem executadas em paralelo com a terceira iteração do Núcleo DS, no ciclo 79, e com a execução do SDSP, no ciclo 155. Desta forma é possível compartilhar o acesso à memória do bloco atual entre os dois núcleos. O Núcleo DS necessita de 17 ciclos para cada iteração (UP e comparadores), já o Núcleo Aleatório necessita de 19 ciclos, devido ao comparador de sete estágios, responsável por encontrar o menor SAD entre os 48 blocos candidatos sorteados. O último comparador encontra o menor bloco candidato entre os três resultados intermediários, obtidos pelo Núcleo DS e pela primeira e segunda execução do Núcleo Aleatório. Este comparador realiza a comparação dos três resultados em um ciclo, e o resultado final da arquitetura do HO-MIRS é obtido em 174 ciclos.

6.5.1 Resultados de Síntese da Arquitetura do HO-MIRS

A arquitetura do algoritmo HO-MIRS foi descrita em VHDL e sintetizada para o dispositivo EP4S40G2F40I2 da família Stratix 4 da Altera e também para *standard cells* na tecnologia 90nm da TSMC, utilizando a ferramenta Synopsys. Os resultados de

síntese para FPGA, apresentados na Tabela 6.10, mostram que a arquitetura do HO-MIRS possui desempenho suficiente para processar vídeos HD 1080p, e até mesmo vídeos QFHD, em tempo real a mais de 30 quadros por segundo. A arquitetura apresenta baixo consumo de recursos de hardware de dispositivo, apenas 18,5K ALUTs e 13,9K registradores são utilizados. A arquitetura do HO-MIRS necessita de um total de 55,9Kbits de memória interna, dos quais 31,7K para a memória de referência e 24,2Kbits para memórias de bloco (bloco atual e blocos candidatos dos núcleos DS e Aleatório), de 512bits cada. No entanto, as otimizações da ferramenta de síntese, e a conversão de memórias para registradores, reduziram a utilização de bits de memória para 37Kbits.

Tabela 6.10: Resultados de síntese da arquitetura do HO-MIRS para FPGA e ASIC

Parâmetros	**FPGA**	**ASIC**	
	HO-MIRS	**HO-MIRS HD 1080p**	**HO-MIRS QFHD**
Frequência (MHz)	210,48	42,28	169,13
ALUTs/*Gates*	18520	84327	84330
Registradores	13950	-	-
Bits de memória	36968	55944	55944
Potência (mW)	-	16,27	62,26
Ciclos por bloco	174	174	174
Quadros HD 1080p por segundo	149	30	-
Quadros QFHD por segundo	37	-	30

A frequência máxima de operação atingida pela arquitetura do HO-MIRS, na síntese em FPGA, é superior a mínima frequência necessária para processar 30 quadros por segundo nas resoluções HD e QFHD. As sínteses em *standard cells*, na tecnologia 90nm da TSMC, foram direcionadas a estas frequências mínimas, 42,28MHz para vídeos HD 1080p e 169,13MHz para vídeos QFHD. Estas frequências são facilmente atingidas pelas células das bibliotecas de 90nm, logo, os resultados de número de *gates* na síntese em *standard cells* são muito similares para as duas frequências. A grande diferença está na potência consumida, que na síntese focada em HD 1080p, obteve um resultado de apenas 16,27mW, enquanto que a versão com desempenho para QFHD apresenta um resultado cerca de quatro vezes mais (proporcional ao aumento de frequência de operação).

6.6 Avaliação e Comparações dos Resultados do Desenvolvimento Arquitetural

Nesta seção, os resultados obtidos com o desenvolvimento arquitetural para a EM serão comparados entre si, e também, em relação a trabalhos relacionados da literatura. Devido a grande dificuldade de uma comparação completa entre os resultados apresentados nesta tese, e resultados apresentados em artigos científicos, estas comparações serão divididas em dois subcapítulos. O primeiro subcapítulo (6.6.1) apresentará uma comparação mais detalhada entre todas as arquiteturas desenvolvidas, comparando não só os resultados de síntese (FPGA e ASIC), mas também os resultados

de qualidade obtidos. No subcapítulo 6.6.2, as arquiteturas desenvolvidas serão comparadas com soluções publicadas na literatura recente, desta vez, de modo menos abrangente, adaptando as comparações aos resultados apresentados nos artigos.

6.6.1 Resultados Comparativos para as Arquiteturas Desenvolvidas

Os principais resultados para as arquiteturas desenvolvidas são apresentados na Tabela 6.11. Os resultados de síntese apresentados são para a síntese FPGA. Os resultados da arquitetura do QSDS-DIC não são apresentados, pois são muito similares aos apresentados pela arquitetura do SDS-DIC. A arquitetura do MRSDS-DIC também não é apresentada nesta tabela, pois não possui desempenho para processar vídeos HD 1080p a 30 quadros por segundo. Todas as arquiteturas da Tabela 6.12 apresentam taxa de processamento suficiente para processar mais de 30 quadros HD 1080p por segundo. O desempenho das arquiteturas é suficiente também para processar 30 quadros QFHD por segundo, exceto para a arquitetura MPDS V.2. No entanto, esta arquitetura apresenta os melhores resultados de qualidade dentre todas as arquiteturas desenvolvidas nesta tese.

Os resultados de utilização de recursos de hardware entre as arquiteturas MPDS e DMPDS são muito similares, e os maiores dentre as arquiteturas desenvolvidas. A arquitetura do HO-MIRS utiliza apenas cerca de 54% do número de ALUTs utilizadas pela arquitetura do MPDS V. 1, já a arquitetura do SDS-DIC utiliza apenas 11%, no entanto, é importante salientar que os resultados do SDS-DIC foram gerados para um FPGA da família Virtex-4.

Tabela 6.11: Resultados comparativos das arquiteturas desenvolvidas em FPGA

Critério	**Arquiteturas**				
	SDS-DIC	**MPDS V. 1**	**MPDS V.2**	**DMPDS**	**HO-MIRS**
PSNR (dB)	32,8	33,28	34,19	33,67	34,03
PRR (%)	54,65	60,93	66	56,19	65,09
FPGA	Virtex-4	Stratix 4	Stratix 4	Stratix 4	Stratix 4
LUTs/ALUTs	3890	34554	34578	34500	18520
Bits de Memória	17408	46240	46240	46240	36968
Frequência (MHz)	184,19	199,2	185,2	187,58	210,48
Ciclos por Vetor	428	170	340	170	174
Quadros HDTV 1080p por segundo	120	145	67	136	149
Quadros QFHD por segundo	30	36	17	34	37

A arquitetura SDS-DIC apresenta os menores resultados de utilização de recursos de hardware do FPGA, no entanto, também apresenta os piores resultados de qualidade. A arquitetura do MPDS V.2 apresenta os melhores resultados de qualidade, e também o maior custo em número de ALUTs utilizadas. A arquitetura do HO-MIRS apresenta a

melhor relação entre uso de recursos de hardware e qualidade obtida. Tanto os resultados de qualidade, quanto os resultados de numero de ALUTs, da arquitetura do HO-MIRS são os segundos melhores dentre todas as arquiteturas desenvolvidas. A qualidade da arquitetura do algoritmo HO-MIRS é similar à obtida pela arquitetura do algoritmo MPDS V.2, no entanto, seu desempenho (e qualidade) é superior ao obtido pelas arquiteturas MPDS V. 1 e DMPDS, com menor uso de ALUTs em relação a elas.

A comparação de resultados de síntese para arquiteturas de EM não é uma tarefa trivial, vários parâmetros precisam ser analisados e relacionados com resultados de qualidade, além de outros critérios menos objetivos, como a facilidade de integração com outros módulos do codificador, por exemplo. O gráfico apresentado na Figura 6.17 apresenta os resultados de custo em hardware (LUTS/ALUTs) contrastados com os resultados de qualidade (PSNR) obtidos por cada arquitetura desenvolvida. A arquitetura do SDS-DIC apresenta a menor relação LUT utilizada por dB de PSNR obtido. Apesar da variação dos resultados de qualidade, as arquiteturas do MPDS e DMPDS apresentam uma relação de ALUT utilizada por dB de PSNR ganho muito similar, com pequena vantagem para a arquitetura MPDS V.2. Isto ocorre, pois os resultados de número de ALUTs utilizadas apresenta pequena oscilação, no entanto, a arquitetura do MPDS V.2 apresenta um aumento mais significativo na qualidade. A arquitetura do HO-MIRS apresenta um resultado intermediário, pois utiliza cerca de metade das ALUTs das versões MPDS e DMPDS, com resultados de qualidade apenas inferiores a versão MPDS V.2. Com isso, a arquitetura do HO-MIRS apresenta a melhor relação ALUT por dB ganho em PSNR.

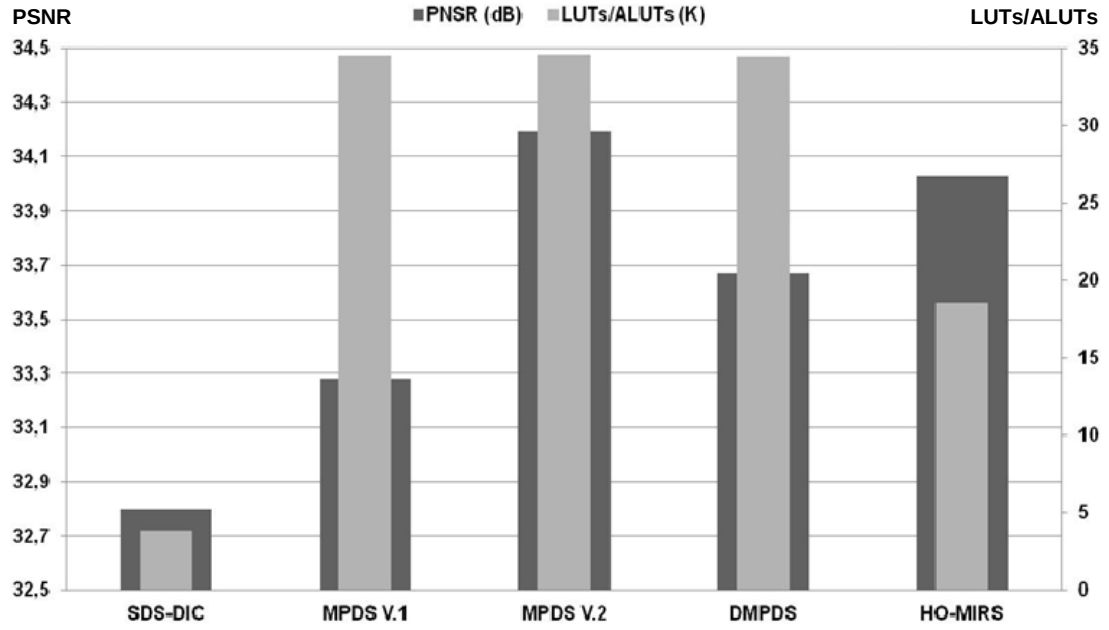


Figura 6.17: Número de LUTs/ALUTs das arquiteturas desenvolvidas e respectivos resultados de qualidade

A análise dos resultados da Tabela 6.11, e da Figura 6.17, mostram que as arquiteturas desenvolvidas nesta tese podem ser utilizadas em diferentes aplicações, com diferentes propósitos. A arquitetura do SDS-DIC (e também QSDS-DIC) é a melhor opção para aplicações com foco em menor consumo de hardware. A arquitetura do MPDS V. 2 é a melhor solução para aplicações com foco em qualidade, podendo ser usada para aplicações de tempo real em vídeos HD 1080p. A arquitetura do HO-MIRS é

a melhor opção para aplicações de tempo real em vídeos de altíssima qualidade, pois apresenta desempenho suficiente para trabalhar com vídeos QFHD. As arquiteturas MPDS V. 1 e DMPDS apresentam resultados intermediários de qualidade, desempenho e uso de recursos de hardware, servindo para aplicações de tempo real em vídeos HD 1080p e também QFHD.

A Tabela 6.12 apresenta os resultados das arquiteturas para ASIC com as sínteses para *standard cells* nas tecnologias 0,18µm e 90nm da TSMC. Todas as arquiteturas foram sintetizadas para a frequência mínima necessária pra processar 30 quadros HD 1080p por segundo. As arquiteturas MPDS V. 1 e HO-MIRS também foram sintetizadas para a frequência mínima para processar 30 quadros QFHD por segundo. As arquiteturas do SDS-DIC e QSDS-DIC apresentadas correspondem às versões originais, sem a utilização de somadores compressores, e foram sintetizadas para a tecnologia 0,18µm. As demais arquiteturas utilizam a tecnologia 90nm. A arquitetura DMPDS não é apresentada, pois não foi sintetizada para *standard cells*.

Os resultados da síntese ASIC apresentam características diferentes das observadas nos resultados de síntese para FPGA. As arquiteturas SDS-DIC e QSDS-DIC apresentam o menor número de *gates*, no entanto, com resultados próximos aos obtidos pelas demais arquiteturas. Devido a maior frequência de operação necessária, e também a tecnologia de 0,18µm, estas arquiteturas apresentam os maiores resultados de consumo de potência. As arquiteturas do MPDS V. 1 e HO-MIRS utilizam frequências de operação bastante similares, no entanto, a arquitetura do HO-MIRS apresenta um número de *gates* aproximadamente 68% maior. Isto implica em um consumo de potência aproximadamente 3,6 vezes superior. Este resultado de utilização de recursos de hardware é inverso ao apresentado na síntese FPGA, onde a arquitetura HO-MIRS apresentava menor utilização de ALUTs. A Arquitetura do MPDS V.2 apresenta um pequeno acréscimo no número de *gates* utilizadas, em relação ao MPDS V. 1. A frequência de operação é duas vezes maior, o que resulta no dobro do consumo de potência. No entanto, é importante salientar que a arquitetura do MPDS V.2 apresenta um ganho médio de PSNR de aproximadamente 0,9dB, em relação ao MPDS V. 1.

Tabela 6.12: Resultados comparativos das arquiteturas desenvolvidas em ASIC

Arquiteturas	**Frequência (MHz)**	***Gates* (k)**	**Bits de memória**	**Potência (mW)**
Resultados para HD 1080p (30fps)				
SDS-DIC	104	46,7	17408	39,39
QSDS-DIC	141,1	44	9216	32,92
MPDS V. 1	41,3	50	82080	4,5
MPDS V.2	82,6	50,2	82080	9
HO-MIRS	42,3	84,3	55944	16,27
Resultados para QFHD (30fps)				
MPDS V.1	165,2	50	82080	18,04
HO-MIRS	169,1	84,3	55944	62,26

As arquiteturas do MPDS V. 1 e HO-MIRS apresentam resultados similares de frequência de operação para a síntese focada em QFHD. No entanto, devido à diferença

significativa no uso de recursos de hardware, o resultado de consumo de potência apresentado pela arquitetura do HO-MIRS é 3,4 vezes superior ao obtido pela arquitetura MPDS V. 1.

A relação custo de hardware versus qualidade obtida é diferente para a síntese ASIC das arquiteturas desenvolvidas. A Figura 6.18 apresenta um gráfico relacionando o custo em hardware (número de *gates*) e o PSBR obtido para a síntese ASIC das arquiteturas desenvolvidas. As arquiteturas SDS-DIC e QSDS-DIC apresentam os menores resultados de número de *gates* em relação ao PSNR obtido, no entanto, seus resultados são muito próximos aos obtidos pelas arquiteturas MPDS V.1 e V.2. Isto ocorre, pois apesar de um pequeno aumento no número de *gates* utilizadas, as arquiteturas do MPDS apresentam ganhos significativos de qualidade. A arquitetura do HO-MIRS, que apresentou a melhor relação ALUTs utilizadas por dB obtido em PSNR, não apresenta o melhor resultado na síntese ASIC, devido ao maior número de *gates* utilizadas

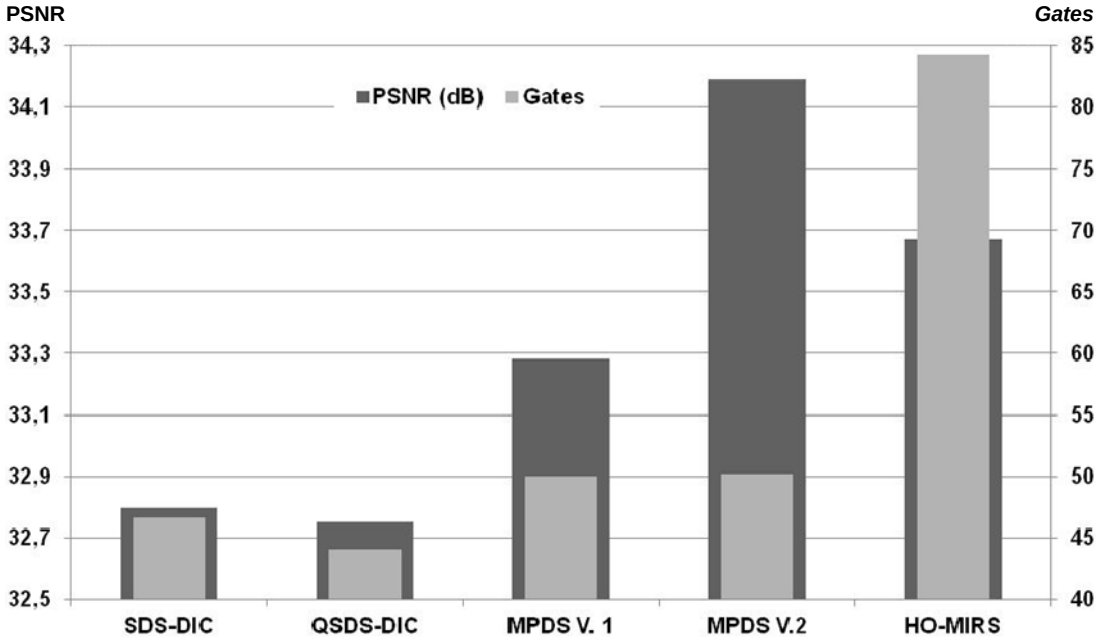


Figura 6.18: Resultados de qualidade e número de *gates* utilizados pelas arquiteturas desenvolvidas

6.6.2 Resultados Comparativos com Trabalhos Relacionados

Os resultados de síntese obtidos para as arquiteturas desenvolvidas também foram comparados com trabalhos relacionados. A Tabela 6.13 apresenta os resultados de síntese de trabalhos relacionados publicados na literatura, bem como os resultados de parte das arquiteturas desenvolvidas nesta tese. A comparação com trabalhos relacionados não é uma tarefa fácil, devido a grande variação de dispositivos (FPGAs) e tecnologias utilizadas nas sínteses. Além disso, não existe um padrão quanto a forma de apresentação dos resultados entre os artigos, e muitos deles omitem, ou não deixam claros, alguns de seus resultados ou parâmetros. As arquiteturas desenvolvidas foram comparadas com soluções voltadas para FPGAs e ASIC, no entanto, a maior parte dos trabalhos publicados na literatura apresenta seus resultados para sínteses ASIC.

Tabela 6.13: Resultados comparativos das arquiteturas desenvolvidas

Arq.	Tecno.	Freq. (MHz)	Ciclos p/ Bloco	Área	Mem. bits (K)	Pot. (mW)	HD 1080p (fps)
(TASDIZEN, 2009)	Virtex 5	130	467	18,2 KLUTs	0,51	-	34
(CETIN, 2011)	-	63	104	33 KLUTs	-	-	74
SDS-DIC	**Virtex 4**	**184**	**190**	**3,9 KLUTs**	**17,4**	**-**	**120**
MPDS V. 1	**Stratix 4**	**199**	**170**	**34,5 KALUTs**	**46,2**	**-**	**145**
MPDS V. 2	**Stratix 4**	**185**	**340**	**34,6 KALUTs**	**46,2**	**-**	**67**
DMPDS	**Stratix 4**	**187**	**170**	**34,5 KALUTs**	**46,2**	**-**	**136**
HO-MIRS	**Stratix 4**	**210**	**174**	**18,5 KALUTs**	**33,9**	**-**	**149**
(KAO, 2009)	0,18µm	154	631	321 K*Gates*	9,72	374	30
VANNE (2009)	130nm	200	437	14 K*Gates*	2,5	59	56
(LAI, 2010)	0,18µm	83,3	1282	26 K*Gates*	28,7	60,8	8
(YIN, 2010)	0,18µm	200	872	260 K*Gates*	11,3	-	28
QSDS-DIC	**0,18µm**	**141**	**140**	**44 K*Gates***	**9,2**	**32,9**	**30**
MPDS V. 1	**90nm**	**41,3**	**170**	**50 K*Gates***	**82**	**4,5**	**30**
MPDS V. 2	**90nm**	**82,6**	**340**	**50 K*Gates***	**82**	**9**	**30**
HO-MIRS	**90nm**	**42,3**	**174**	**84,3 K*Gates***	**55,9**	**16,3**	**30**

A arquitetura apresentada em (TASDIZEN, 2009) implementa o algoritmo rápido *Dynamic Variable Step Search* (DVSS), esta arquitetura utiliza o mesmo tamanho de bloco utilizados em todas as arquiteturas desenvolvidas nesta tese, e apresenta seus resultados de síntese para FPGA. Esta arquitetura pode processar mais de 34 frames HD 1080p por segundo a uma frequência de operação de 130MHz, obtida na síntese para um FPGA da família Virtex-5 da Altera. Esta arquitetura apresenta um número de ciclos por bloco maior do que o utilizado por todas as versões das arquiteturas desenvolvidas, sendo que seu desempenho também é inferior. Quanto à utilização dos recursos de hardware, a arquitetura apresentada em (TASDIZEN, 2009) utiliza um número de LUTs similar ao número de ALUTs apresentado pela arquitetura do HO-MIRS. Este número de LUTs é praticamente metade do número utilizado pelas arquiteturas do MPDS e DMPDS, no entanto, é cerca de 4,6 vezes maio do que o apresentado pela arquitetura do

SDS-DIC. O melhor resultado desta arquitetura é quanto ao tamanho da memória interna, que é de apenas 0,51Kbits. Esta memória é a menor dentre todas as arquiteturas comparadas, e pode explicar o maior número de ciclos por vetor apresentado, devido à necessidade de recarga da memória interna.

Outra arquitetura de EM para alta definição sintetizada para FPGA é apresentada em (CETIN, 2011). Esta arquitetura implementa o algoritmo algoritmo *Adaptive True Motion Estimation* (ATME), e foi sintetizada para um FPGA com tecnologia de 90nm, no entanto, não foi especificado o fabricante ou família deste FPGA. Esta arquitetura necessita de um número de ciclos por bloco menor do que o utilizado por todas as arquiteturas desenvolvidas. Desta forma, é possível atingir o mesmo desempenho com menores frequências de operação. No entanto, os resultados de síntese mostram que a frequência máxima atingida pela arquitetura é de 63MHz, o que resulta na taxa de processamento de 74 quadros HD 1080p por segundo. Este resultado garante o desempenho para processar vídeos desta resolução em tempo real, no entanto, é inferior ao apresentado por todas as arquiteturas desenvolvidas. O número de LUTs apresentado é similar aos obtidos pelas arquiteturas do MPDS e DMPDS (maiores dentre todas as arquiteturas desenvolvidas), e os resultados de qualidade são inferiores aos obtidos com os algoritmos desenvolvidos nesta tese, como apresentados no subcapítulo 5.5.

Em (KAO, 2009) é apresentada uma arquitetura de EM com desempenho para processar 30 quadros HD 1080p por segundo, quando sintetizada para ASIC com tecnologia de 0,18μm. Devido à eficiência dos algoritmos, as arquiteturas desenvolvidas apresentam um número de ciclos, e resultados de utilização de *gates*, consideravelmente inferior. Comparada com a arquitetura do QSDS-DIC, que foi sintetizada para a mesma tecnologia, a arquitetura apresentada em (KAO, 2009) apresenta resultados similares de frequência de operação e tamanho de memória interna. No entanto, o número de *gates* utilizadas é 7,3 vezes superior, o que resulta em um consumo de potência aproximadamente 11,3 superior. Esta arquitetura também apresenta maior número de *gates* e maior consumo de potência do que as arquiteturas do MPDS e HO-MIRS. As arquiteturas MPDS V. 1 e HO-MIRS necessitam de uma frequência de operação aproximadamente quatro vezes menor para atingir a mesma taxa de processamento. Esta característica tem influência direta nos resultados de consumo de potência, que são consideravelmente maiores para (KAO, 2009). No entanto, é importante salientar que as arquiteturas MPDS e HO-MIRS utilizam uma tecnologia de síntese mais eficiente (90nm).

O trabalho publicado em (VANNE, 2010) apresenta uma arquitetura que implementa três algoritmos de ME, *Hexagon Based-Search* (HEXBS), *Block Based Gradient Descent Search* (BBGDS) and *Three Step Search* (TSS). Esta arquitetura utiliza 390, 437 e 680 ciclos por bloco para cada algoritmo, respectivamente. A Tabela 6.13 apresenta apenas os resultados médios, para 437 ciclos, sendo que em todas as configurações a arquitetura pode processar mais de 30 quadros HD 1080p por segundo. Este número de ciclos por bloco é superior ao necessário por todas as arquiteturas desenvolvidas, bem como a frequência de operação utilizada. O número de *gates* utilizado é inferior ao apresentado por todas as arquiteturas desenvolvidas, ainda assim, os resultados de consumo de potência são superiores aos obtidos para arquitetura desenvolvidas, até mesmo para a arquitetura do QSDS-DIC, que foi sintetizada para uma tecnologia mais ultrapassada.

A arquitetura apresentada em (LAI, 2010) implementa o algoritmo *Top-Winners Search* e apresenta seus resultados de síntese para ASIC em 0,18μm. Esta arquitetura

tem foco em baixa utilização de hardware e, e devido a isto, apresenta o maior número de ciclos por bloco dentre todas as arquiteturas comparadas. Devido ao foco me baixo custo em área, esta arquitetura não possui desempenho para processar vídeos HD 1080p em tempo real.

O trabalho publicado em (YIN, 2010) apresenta uma arquitetura para o algoritmo *Multi-Resolution* ME *Algorithm* (MMEA). Esta arquitetura utiliza uma quantidade de *gates* significativamente maior do que a utilizada pelas arquiteturas desenvolvidas. Parte deste aumento de deve ao fato desta arquitetura utilizar a busca em vários tamanhos de bloco, e utilizar dois frames de referência. Esta arquitetura necessita de uma frequência de operação muito elevada para atingir uma taxa de processamento similar às taxas atingidas pelas arquiteturas desenvolvidas. O resultado de consumo de potência não é apresentado, mas é possível estimar que seja significativamente maior, devido a maior quantidade de hardware e frequência de operação utilizada.

É importante salientar as arquiteturas desenvolvidas são as únicas que apresentam resultados de desempenho para processar vídeos QFHD em tempo real a 30 quadros por segundo (Tabelas 5.11 e 5.12). Todas as arquiteturas desenvolvidas apresentam desempenho suficiente para processar vídeos QFHD quando sintetizadas para FPGA (exceto a MPDS V. 2). As arquiteturas MPDS V. 1 e HO-MIRS também foram sintetizadas para ASIC com a frequência necessária para processar 30 quadros QFHD por segundo. As arquiteturas desenvolvidas também apresentam os melhores resultados de consumo de potência, dentre todos os trabalhos comparados. Além dos melhores resultados de desempenho e consumo de potência, as arquiteturas desenvolvidas apresentam resultados de utilização de recursos de hardware similares aos trabalhos relacionados. Isto demonstra a eficiência dos algoritmos e das arquiteturas de hardware dedicadas desenvolvidos.

7 CONCLUSÕES

Esta tese apresentou o desenvolvimento de algoritmos rápidos e arquiteturas de hardware dedicadas para a estimação de movimento (EM) em vídeos de alta definição. Este trabalho pode ser dividido em duas grandes etapas: desenvolvimento de algoritmos rápidos de EM focados em vídeos de alta definição e desenvolvimento de arquiteturas de hardware dedicadas aos algoritmos desenvolvidos. Um breve capítulo de conceitos de codificação de vídeo também foi apresentado, bem como uma revisão bibliográfica com o estado da arte em termos de algoritmos e arquiteturas de hardware para a estimação de movimento.

A avaliação bibliográfica demonstrou que a maioria dos artigos científicos apresenta resultados para a estimação de movimento em vídeos de baixa definição, como QCIF e CIF. No capítulo 4, uma avaliação da estimação de movimento em vídeos de alta definição foi apresentada, demonstrando que com o aumento da resolução, os algoritmos rápidos tendem a apresentar perdas significativas em relação ao algoritmo ótimo. Esta diferença ocorre, principalmente, devido ao crescimento significativo do número de mínimos locais, que ocorre devido ao aumento da resolução dos vídeos. Estes dados reforçam a necessidade do desenvolvimento de novos algoritmos rápidos de EM focados em vídeos de altíssima definição, especialmente superiores a HD1080p. Estes algoritmos devem ser mais resistentes à escolha de mínimos locais, gerando resultados de qualidade mais próximos ao resultado ótimo.

A partir do estudo realizado sobre a estimação de movimento em vídeos de alta definição, novos algoritmos de EM focados em alta qualidade para vídeos de elevada definição foram desenvolvidos e avaliados. Os primeiros resultados foram obtidos para otimizações do algoritmo DS, otimizações estas que foram orientadas para uma efetiva implementação em hardware. Após esta investigação inicial, novos algoritmos de EM foram desenvolvidos nesta tese. Estes algoritmos podem ser divididos em duas classes: algoritmos multiponto e algoritmos da classe aleatória. Estas classes foram denominadas de acordo com a abordagem utilizada para a redução da probabilidade de escolha de mínimos locais.

A investigação sobre otimizações do algoritmo DS, focadas na implementação em hardware, resultou no desenvolvimento nesta tese de dois novos algoritmos, o *Sub-sampled Diamond Search with Dynamic Iteration Control* (SDS-DIC) e o *Quarter Sub-sampled Diamond Search with Dynamic Iteration Control* (QSDS-DIC). Estes algoritmos apresentam um controle dinâmico de iterações (DIC), que restringe o número de iterações do algoritmo DS, facilitando sua implementação em hardware. Com o controle dinâmico de iterações é possível disponibilizar um grande número de iterações ao algoritmo, quando necessário, utilizando as iterações não utilizadas para vetores anteriores. Desta forma, as perdas em qualidade são muito pequenas, e o desempenho do algoritmo se mantém elevado.

Os algoritmos multiponto utilizam a estratégia de realizar a busca em diferentes pontos da área de busca. Nesta classe de algoritmos, foram desenvolvidos os algoritmos *Multi-Point Diamond Search* (MPDS) e *Dinamic Multi-Point Diamond Search* DMPDS. O algoritmo MPDS utiliza cinco núcleos de busca, um aplicado a região central da área de busca e outros quatro divididos entre os quatro setores. A distância, em relação ao centro, dos núcleos de busca dos setores é dada por um parâmetro *d*, que define a distância em pixels em relação aos eixos X e Y. O valor do parâmetro *d* tem influência direta no resultado de qualidade obtido pelo algoritmo MPDS, e um estudo sobre a variação do valor de *d* foi realizado, para determinar o melhor valor médio para cada tamanho de bloco utilizado.

O algoritmo DMPDS apresenta uma evolução do algoritmo MPDS, onde o valor do parâmetro *d* pode ser dinamicamente controlado. No algoritmo DMPDS os resultados de qualidade obtidos a cada quadro são armazenados e comparados e, dependendo das características do vídeo processado, o valor de *d* pode aumentar ou diminuir. Em geral, para sequências de vídeo com pouca movimentação (câmera e objetos) os melhores resultados de qualidade são obtidos para valores menores de *d*, enquanto que maiores valores de *d* são a melhor opção para vídeos de alta movimentação. Com o ajuste dinâmico, o algoritmo DMPDS pode obter melhores resultados de qualidade com um pequeno acréscimo no custo computacional.

A segunda classe de algoritmos desenvolvida foi a de natureza aleatória. Os algoritmos desta classe utilizam a aleatoriedade no processo de busca como estratégia para diminuir a incidência da escolha de mínimos locais. Diversos algoritmos foram desenvolvidos nesta classe e os melhores deles foram apresentados e avaliados nesta tese. Todos os algoritmos aleatórios desenvolvidos possuem uma etapa em comum: o sorteio e a avaliação de *N* blocos candidatos aleatórios dentro da área de pesquisa. Além da etapa de exploração aleatória, cada algoritmo apresenta uma forma diferente de exploração da região central da área de busca. Esta etapa também está presente em todos os algoritmos aleatórios desenvolvidos, pois garante bons resultados para vídeos de baixa movimentação. Os algoritmos aleatórios desenvolvidos também apresentam diferentes abordagens de refinamento final, aplicado sobre os resultados obtidos com a etapa aleatória.

Os resultados de qualidade e custo computacional dos algoritmos aleatórios estão diretamente relacionados ao valor de *N* utilizado, bem como relacionados ao tamanho da área de busca. Uma análise da variação do valor de *N* e também do tamanho da área de busca foi realizada para os algoritmos aleatórios desenvolvidos nesta tese. Este estudo demonstrou que quanto maior o valor de *N*, melhores serão os resultados de qualidade obtidos, no entanto, o número de blocos candidatos comparados não possui uma relação direta com o valor de *N*. Alguns algoritmos apresentam um número maior de blocos candidatos comparados, para valores baixos de *N,* e reduzem este número com o aumento do valor de *N*. Este comportamento ocorre devido à quantidade de iterações da etapa de refinamento final iterativo, que tende a ser maior para valores menores de *N*.

Os algoritmos desenvolvidos foram avaliados para 10 sequências de teste HD 1080p. As avaliações consideraram a EM de modo isolado, sendo que os resultados de qualidade foram obtidos através da comparação entre os quadros originais e os quadros remontados a partir dos vetores de movimento gerados. As avaliações de qualidade consideraram os resultados médios de PSNR e Percentual de Redução do Resíduo (PRR) obtido. O custo computacional dos algoritmos desenvolvidos foi mensurado em

número de Blocos Candidatos Comparados (BCC) e também com o número de comparações, que varia de acordo com o número de pixels do bloco avaliado. Os algoritmos foram avaliados para diferentes tamanhos de bloco (8x8, 16x16 e 32x32) e também com diferentes níveis de subamostragem de pixel (2:1, 4:1 e 8:1). Os resultados destas avaliações mostraram que os algoritmos desenvolvidos apresentam ganhos de qualidade significativos em relação aos algoritmos rápidos conhecidos. Desta forma, foi possível reduzir as perdas em relação ao resultado ótimo e ainda assim manter ganhos significativos em termos da redução do custo computacional.

Os algoritmos desenvolvidos também foram avaliados para sequências com resolução superior a HD 1080p. Os resultados obtidos para as avaliações em sequências de teste WQXGA (2560x1600 pixels) demonstram que a eficiência dos algoritmos desenvolvidos se mantém e que os ganhos obtidos sobre algoritmos rápidos convencionais são ainda maiores dos que os obtidos em HD 1080p. Estes resultados confirmam os dados apresentados no capítulo 4, onde foi demonstrado que algoritmos rápidos convencionais tendem a aumentar suas perdas em qualidade com o crescimento da resolução dos vídeos. No entanto, os algoritmos desenvolvidos nesta tese conseguiram obter ganhos similares aos obtidos pelo algoritmo FS com o aumento da resolução, demonstrando sua eficiência na estimação de movimento em vídeos de alta definição.

Esta tese também apresentou o desenvolvimento de arquiteturas de hardware dedicadas para os algoritmos de EM desenvolvidos. As arquiteturas foram desenvolvidas com foco em alto desempenho, sendo o requisito mínimo o processamento de 30 quadros HD 1080p por segundo, e baixo custo em hardware. As arquiteturas foram descritas em VHDL e sintetizadas para FPGAs e *standard cells*. Foram utilizados FPGAs das famílias Virtex-4 da Xilinx e da Stratix-5 Altera e as bibliotecas de *standard cells* nas tecnologias CMOS 0,18μm e 90nm da TSMC para a síntese da versão ASIC do sistema digital que implementa o mesmo algoritmo.

Devido ao foco deste trabalho no alto desempenho e no baixo custo em hardware, apenas os algoritmos tratados e/ou desenvolvidos pelo autor que apresentaram os melhores resultados foram implementados em hardware. No total, cinco algoritmos foram implementados em hardware: SDS-DIC, QSDS-DIC, MPDS, DMPDS e HO-MIRS. Diversas variações e versões destas arquiteturas também foram desenvolvidas, algumas priorizando a qualidade da EM e outras com maior foco na redução dos recursos de hardware e na obtenção de maior taxa de processamento.

A eficiência dos algoritmos propiciou o desenvolvimento de arquiteturas com baixa frequência de operação e baixa utilização de hardware em relação a outros trabalhos apresentados na literatura. Isto também influenciou na eficiência energética das arquiteturas desenvolvidas, que apresentam resultados baixos de potência para a síntese ASIC. O desempenho apresentado por algumas das arquiteturas na síntese para FPGAs é suficiente, inclusive, para o processamento de vídeos QFHD (3840x2160 pixels) com mais de 30 quadros por segundo. Duas arquiteturas também foram sintetizadas para *standard cells* com foco no processamento de 30 quadros por segundo nesta resolução. Até a apresentação deste trabalho, não há na literatura corrente outro que apresente uma arquitetura de EM com desempenho suficiente para vídeos QFHD em tempo real, como a aqui proposta.

Os resultados apresentados nesta tese demonstraram a necessidade do desenvolvimento de algoritmos de EM orientados especialmente aos vídeos de alta

definição. Os algoritmos desenvolvidos apresentaram ganhos de qualidade em relação aos algoritmos rápidos convencionais, diminuindo as perdas em relação ao algoritmo ótimo quando aplicados a vídeos de alta definição. Além disso, os algoritmos desenvolvidos apresentam uma melhor relação entre qualidade e custo computacional. As arquiteturas de hardware dedicadas desenvolvidas para os novos algoritmos apresentam desempenho suficiente para processar vídeos HD 1080p, e até mesmo vídeos QFHD em tempo real. Além disso, apresentam baixa utilização de recursos de hardware e baixa potência.

Existem diversas possibilidades de trabalhos futuros associados a esta tese, tanto no desenvolvimento algorítmico, quando no desenvolvimento arquitetural. Novos algoritmos, derivados das soluções multiponto e aleatória apresentadas nesta tese, já vêm sendo desenvolvidos. Uma das primeiras soluções visa reduzir o custo computacional dos algoritmos aleatórios, através da avaliação rápida dos setores da área de busca antes da exploração completa. Desta forma, é possível determinar o setor da área de busca onde existe a maior probabilidade da ocorrência de bons blocos candidatos. Neste setor, a busca é refinada, enquanto que, nos demais setores, a busca é interrompida.

Além do desenvolvimento algorítmico, outro trabalho futuro é avaliar as novas ferramentas desenvolvidas para a EM no padrão HEVC e aplicar estas ferramentas aos algoritmos desenvolvidos. Uma das novas propostas do HEVC é a utilização de blocos maiores na EM, como 32x32 pixels, por exemplo. Este tamanho de bloco já foi inclusive avaliado nesta tese. Os algoritmos desenvolvidos poderiam ser inseridos no código de referência do HEVC (hoje em desenvolvimento), para a obtenção de resultados mais completos de qualidade, com um compromisso aceitável de custo computacional e *bit rate* obtido no processo de codificação inter-quadros.

O desenvolvimento arquitetural também possibilita uma sequência de trabalhos futuros, pois uma nova e eficiente arquitetura pode ser desenvolvida para cada novo algoritmo de EM. Além disso, existe espaço de projeto para a exploração de refinamentos das arquiteturas de EM já desenvolvidas. As versões apresentadas são as primeiras soluções desenvolvidas, no entanto, todas apresentam resultados de desempenho superiores ao mínimo necessário para processar vídeos HD 1080p em tempo real. Com isto, otimizações podem ser feitas para a redução do custo do hardware e para a otimização da memória interna, aumentando a eficiência energética e a relação custo do hardware *versus* qualidade obtida.

REFERÊNCIAS

ADIKARI, A. Sequential motion estimation using luminance and chrominance information for distributed video coding of Wyner-Ziv frames. IET Electronics Letters, [S.l.], v. 42, n. 7, p. 389-399, Apr. 2006.

AGOSTINI, L. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas a Compressão de Vídeo Segundo o Padrão H.264/AVC**. 2007. 172f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ALTERA. FPGA CPLD and ASIC from Altera. **Altera Web Site**. Disponivel em: <www.altera.com>. Acesso em: Dez. 2010.

AKRAM, M.; IZQUIERDO, E. Fast Multifrme Motion Estimation for Surveillance Videos. In: ICIP 2010 – IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING. **Proceedings…** Hong Kong: IEEE, 2010, p. 753-756.

ASHENDEN, P. **The Student's Guide to VHDL**. San Francisco: Morgan Kaufmann, 1998.

BANH, X.; TAN, Y. Adaptive dual-cross Search algorithm for Block-matching motion estimation. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 50, n. 2, p.766-775, May 2004.

BANH, X.; TAN, Y. Efficient video motion estimation using dual-cross search algorithms. In: IEEE INTERNATIONAL SYMPOSIUM CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 5485-5488.

BEKER, H. **Cipher Systems: The Protection of Communications**. John Wiley & Sons Inc, 1983.

BHASKARAN, V.; KONSTANTINIDES, K. **Image and Video Compression Standards**: Algorithms and Architectures. 2nd ed. Boston: Kluwer Academic Publishers, 1999.

BYUN, J.; CHOI, J.; KIM, J. A Fast Multi-Reference Frame Motion Estimation Algorithm. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 56, n. 3, p. 1911-1917, Ago. 2010.

CADENCE. Encouter RTL Compiler. **Cadence Web Site**. Disponível em: <http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx>. Acesso em Nov. 2010.

CETIN, M.; HAMZAOGLY, I. An Adaptive True Motion Estimation Algorithm for Frame Rate Conversion of High Definition Video. In: International Conference on Pattern Recognition. **Proceedings…** Istambul**:** 2010, p. 4109-4112.

CETIN, M.; HAMZAOGLY, I. An Adaptive True Motion Estimation Algorithm for Frame Rate Conversion of High Definition Video and Its Hardware Implementations. **IEEE Transactions on Consumer Electronics**, [S.l], v. 57, n. 2, Mai. 2011.

CHENG, Y.; CHEN, Z.; CHANG, P. An H.264 Spatio-temporal Hierarchical Fast Motion Estimation Algorithm for High-Definition Video. In: ISCAS 2009 - IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** Taipei: IEEE, 2009, p. 880-883.

CRISTANI, C.; DALL'OGLIO, P.; PORTO, M.; AGOSTINI, L.; BAMPI, S. A Fast Random Based Motion Estimation Search Algorithm. In: SForum 2011 - 11th MICROELECTRONICS STUDENTS FORUM. **Proceedings…** João Pessoa: 2011.

CRISTANI, DALL'OGLIO, P.; PORTO, M.; AGOSTINI, L.; BAMPI, S. Hardware Design for the New Galaxy Random Search Motion Estimation Algorithm Focusing in HD Videos. In: WCAS 2011 - 1st WORKSHOP ON CIRCUITS AND SYSTEMS (WCAS). **Proceedings…** João Pessoa: 2011a.

DONG, J. LIU, Y. **H265.net Witness the development of H.265**. Disponível em: <http://www.h265.net/>. Acesso em: Jan. 2011.

FASTEC, Image LP: High-Seed Digital Cameras for Every Need. **FASTEC** web site. Disponível em: <http://www.fastecimaging.com/>. Acesso em: Jan, 2011.

FONSECA, J.; MARTINS, G. **Curso de Estatística**. 6ª ed. São Paulo: Atlas S.A., 1996.

GONZALEZ, R.; WOODS, R. **Processamento de Imagens Digitais**. São Paulo: Edgard Blücher, 2003.

GHANBARI, M. **Standard Codecs:** Image Compression to Advanced Video Coding. United Kingdom: The Institution of Electrical Engineers, 2003.

HAAN, G.; et al. True motion estimation with 3-D recursive search block matching. **IEEE Transaction on Circuits and Systems for Video Technology**, [S.l.], v. 3, n. 5, p. 368-379, Out. 1993.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 14496-2 - MPEG-4 Part 2 (01/1999)**: coding of audio visual objects – part 2: visual. [S.l.], 1999.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264 (03/05)**: advanced video coding for generic audiovisual services. [S.l.], 2005.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T** Documento VCEG-AC06. Disponível em : <wftp3.itu.int/av-arch/video-site/0607_Kla/VCEG-AC01.doc>. Acesso em: Jan. 2011.

JING, X.; CHAU, L. An efficient three-step search algorithm for Block motion estimation. **IEEE Transactions on Multimedia**, [S.l.], v. 6, n. 3, p. 435-438, Jun. 2004.

JOINT COLLABORATIVE TEAM ON VIDEO CODING - JCT-VT: Documentos das reuniões do grupo. Disponível em: <http://wftp3.itu.int/av-arch/jctvc-site/>. Acesso em: Jan. 2011.

KAO, C.; WU, C.; LIN, Y. A High-Performance Three-Engine Architecture for H.264/AVC Fractional Motion Estimation. **IEEE Transactions on Very Large Scale Integration Systems**, [S.l], v. 18, n. 4, p. 662-666, Jun. 2010.

KERNIGHAN, B.; RITCHIE, D. **C: a Linguagem de Programação Padrão Ansi**. Rio de Janeiro: Campus, 1999.

KUHN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

KUO, C.; et al. A Novel Prediction-Based Directional Asymmetric Search Algorithm for Fast Block-Matching Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 19, n. 6, p. 893-899, Jun. 2009.

LAI, Y.; CHEN, L.; HUANG, S. Hybrid Parallel Motion Estimation Architecture Based on Fast Top-Winners Search Algorithm. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 56, n. 3, p. 1837-1842, Ago. 2010.

LI, W.; SALARI, E. Successive elimination algorithm for motion estimation," **IEEE Transactions on Image Processing**, [S.l.], v. 4, p. 105-107, Jan. 1995.

LI, T.; et al. A Novel Configurable Motion Estimation Architecture for High-Efficiency MPEG-4/H.264 Encoding. In: ASPDAC 2005 - IEEE ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE. **Proceedings…** [S.l.]: IEEE, 2005. p. 1264-1267.

LIN, C.; LEOU, J. An Adaptative Fast Full Search Motion Estimation Algorithm for H.264. In: ISCAS 2005 - IEEE INTERNATIONAL SYMPOSIUM CIRCUITS AND SYSTEMS. **Proceedings…** [S.l.]: IEEE, 2005. p. 1493-1496.

LIN, Y.; et al. A Hardware-Efficient H.264/AVC Motion-Estimation Design for High-Definition Video. **IEEE Transactions on Circuits and Systems I: Regular Papers**, [S.l.], v. 55, n. 6, p. 1526-1535, Jul. 2008.

LIU, L.; FEIG, E. A block-based gradient descent search algorithm for block motion estimation in video coding. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 6, n. 4, p. 419–422, Ago. 1996.

LOUKIL, H. et al. Hardware Implementation of Block Matching Algorithm with FPGA Technology. In: INTERNATIONAL CONFERENCE ON MICROELECTRONICS, ICM, 2004. **Proceedings…** [S.l.:s.n.], 2004. p. 542-546.

LUO, J.; et al.. A Novel All-Binary Motion Estimation (ABME) with Optimized Hardware Architectures. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 12, n. 8, p. 700–712, Ago. 2002.

MIANO, J. Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP. Reading, MA: Addison Wesley, 1999.

MODELSIM – **A Comprehensive Simulation and Debug Environment for Complex ASIC and FPGA Designs**. Disponível em: <www.model.com>. Acesso em: Dez. 2010.

NDILI, O.; OGUNFUNMI, T. Hardware-Oriented Modified Diamond for Motion Estimation in H.246/AVC. In: ICIP 2010 – IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING. **Proceedings…** Hong Kong: IEEE, 2010, p. 749-752.

NOBLE, D.; et al. Two Novel Algorithms for High Quality Motion Estimation in High Definition Video Sequences. In: SIBGRAPI 2011 - CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES. **Proceedings…** Alagoas: 2011.

OKLOBDZIJA, V.; VILLEGER, D.; LIU, S. A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers and Algorithmic Approach. **IEEE Transaction on Computers**, [S.l.], v. .45, n. 3, 1996, p. 294-306.

PO, L.; et al. Novel Directional Gradient Descent Searches for Fast Block Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 19, n. 8, p. 1189-1195, Ago. 2009.

PORTO, M. et al. Investigation of Motion Estimation Algorithms Targeting High Resolution Digital Video Compression. In: WebMedia 2007 - ACM BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND WEB, 2007. **Proceeding…** Gramado: ACM, 2007, p. 111-118.

PORTO, Marcelo. Arquiteturas de Alto Desempenho e Baixo Custo em Hardware para a Estimação de Movimento em Vídeos Digitais. Porto Alegre, 2008. 100f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

PORTO, Marcelo. Avaliação do Algoritmo Diamond Search para Estimação de Movimento com Múltiplos quadros de Referência. Porto Alegre, 2008a. 48f. Trabalho Individual – Instituto de Informática, UFRGS, Porto Alegre.

PORTO, M.; AGOSTINI, L.; BAMPI, S.; SUSIN, A. A High Throughput and Low Cost Diamond Search Architecture for HDTV Motion Estimation. In: ICME 2008 - IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA & EXPO. **Proceedings…** Hannover: IEEE, 2008b, p. 1033-1036.

PORTO, M.; AGOSTINI, L.; SUSIN, A.; BAMPI, S. Architectural Design for the New QSDS with Dynamic Iteration Control Motion Estimation Algorithm Targeting HDTV. In: SBCCI 2008 - 21st SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN. **Proceedings…** Gramado: ACM, 2008c, p. 216-221.

PORTO, Roger Desenvolvimento Arquitetural para Estimação de Movimento de Blocos de Tamanhos Variáveis Segundo o Padrão H.264/AVC de Compressão de Vídeo Digital. Porto Alegre, 2008d. 96f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

PORTO, M.; ALTERMANN, J.; COSTA, E.; BAMPI, S. Power Efficient Architecture for Motion Estimation Using the QSDS-DIC Algorithm. In: ICECS 2009 - IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS, AND SYSTEMS. **Proceedings…** Hammamet: IEEE, 2009, v. 1, p. 331-334.

PORTO, M.; SILVA, A.; ALMEIDA, S.; COSTA, E.; BAMPI, S.. Motion Estimation Architecture Using Efficient Adder-Compressors for HDTV Video Coding. **Journal of Integrated Circuits and Systems**, [S.l], v. 5, n. 1, p. 78-88, Set. 2010.

PORTO, M.; NOBLE, D; AGOSTINI, L; BAMPI, S.. Two Fast Multi-Point Search Algorithms for High Quality Motion Estimation in High Resolution Videos. **International Journal of Information Technology, Communications and Convergence**, 2011, artigo aceito para publicação.

PORTO, M.; et al. A Real Time and Power Efficient HDTV Motion Estimation Architecture Using Adder-Compressors. In: LASCAS 2011 – 2nd LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** Bogotá: IEEE, 2011a.

PORTO, M.; et al. An Efficient ME Architecture for High Definition Videos Using the New MPDS Algorithm. In: SBCCI 2011 – 24th SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN. **Proceedings…** João Pessoa: ACM, 2011b, p. 119-124.

PORTO, M.; et al. Iterative Random Search: A New Local Minima Resistant Algorithm for Motion Estimation in High Definition Videos. **Multimedia Tools and Applications**, 2012, artigo aceito para publicação.

PURI, A.; et al. Video Coding Using the H.264/MPEG-4 AVC Compression Standard. **Elsevier Signal Processing: Image Communication**. [S.l.], n. 19, p.793–849, 2004.

RICHARDSON, I. **Video Codec Design :** Developing Image and Video Compression Systems. Chichester: John Wiley and Sons, 2002.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression :** Video Coding for Next-Generation Multimedia. Chichester: John Wiley and Sons, 2003.

RODRIGUES, N.; et al. On Dictionary Adaptation for Recurrent Pattern Image Coding. **IEEE Transactions o Image Processing**, [S. l.], v. 17, n. 9, p. 1640-1653, Set. 2008.

ROSA, L.; et al. Arquitetura de Alto Desempenho para o Algoritmo de Estimação de Movimento SDS-DIC com Múltiplos Quadros de Referência. In: XV WORKSHOP IBERCHIP. **Proceedings…** Buenos Aires: 2009, v. 2, p. 567-572.

SALOMON, D. **Data Compression**: The Complete Reference. 4th ed. New York: Springer, 2006.

SALOMON, D. **A Concise Introduction to Data Compression**. New York: Springer, 2008.

SANCHEZ, G.; NOBLE, D.; PORTO, M.; AGOSTINI, L.; BAMPI, Sl. High Efficient Motion Estimation Architecture with Integrated Motion Compensation and FME Support. In: LASCAS 2011 – 2nd LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** Bogotá: IEEE, 2011, p. 1-4.

SANCHEZ, G.; NOBLE, D.; PORTO, M.; AGOSTINI, L.; BAMPI, S. A Real Time HDTV Motion Estimation Architecture for the New MPDS Algorithm. In: EUROCON 2011 – INTERNATIONAL CONFERENCE ON COMPUTER AS A TOOL. **Proceedings…** Lisboa: IEEE, 2011a, p. 1-4.

SARWER, M.; WU, Q. Adaptive Variable Block-Size Early Motion Estimation Termination Algorithm for H.264/AVC Video Coding Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 19, n. 8, p. 1196-1201, Ago. 2009.

SHI, Y.; SUN, H. **Image and Video Compression for Multimedia Engineering:** Fundamentals, Algorithms and Standards. Boca Raton: CRC Press, 1999.

SILVA, A.; PORTO, M.; ALMEIDA, S.; COSTA, E.; BAMPI, S. High Performance Motion Estimation Architecture Using Efficient Adder-Compressors. In: SBCCI 2009 - 22st SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. **Proceedings…** Natal: ACM, 2009.

SUHRING, Karsten. **H.264/AVC Reference Software**. In: Fraunhofer Heinrich-Hertz-Institute. Disponível em: <http://iphome.hhi.de/suehring/tml/download/>. Acesso em: Dez. 2010.

SULLIVAN, G.; OHM, J. Meeting report of the first meeting of the Joint Collaborative Team on Video Coding (JCT-VC), Joint Collaborative Team on Video Coding of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 (JCTVC-A200), Dresden, 2010.

SYNOPSYS. **Synopsys Tools.** Disponível em: <http://www.synopsys.com/Tools/>. Acesso em: Abr. 2011.

TASDIZEN, O.; et al. Dynamically Variable Step Search Motion Estimation Algorithm and a Dynamically Reconfigurable Hardware for Its Implementation. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 55, n. 3, p. 1645-1653, Ago. 2009.

TASDIZEN, O.; et al. A high performance reconfigurable motion estimation hardware architecture. In: DATE 2009 - IEEE DESIGN, AUTOMATION & TEST IN EUROPE. **Proceedings…** Nice: IEEE, 2009a, p. 882-885.

TNT - **Institut für Informationsverarbeitung**. Disponível em <ftp://ftp.tnt.uni-hannover.de/testsequences>. Acesso em: Nov. 2011.

VANNE, J.; et al. A Configurable Motion Estimation Architecture for Block-Matching Algorithms. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 19, n. 4, p. 446-476, Abr. 2009.

VQEG. Video Quality Experts Group Benchmark Videos. Disponível em: http://www.its.bldrdoc.gov/vqeg/. Acesso em: Dez. 2011.

WANG, S.; TAI, S. CHIANG, T. A Low-Power and Bandwidth-Efficient Motion Estimation IP Core Design Using Binary Search. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 19, n. 5, p. 760-765, Mai. 2009

WEINBERGER, A. 4-2 Carry-Save Adder Module. **IBM Technical Disclosure Bulletin.** [S.l.], 1981.

WONG, H.; et al. Enhanced Predictive Motion Vector Field Adaptive Search Technique (E-PMVFAST)-Based on Future MV Prediction. In: ICME 2005 - IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO. **Proceedings…** Amsterdam: IEEE, 2005.

XILINX INC. **Xilinx**: The Programmable Logic Company. Disponível em: <www.xilinx.com>. Acesso em: Nov. 2010.

XILINX INC. **Xilinx ISE 8.1i Software Manuals and Help**. [S.l.], 2006. Disponível em: <www.xilinx.com/support/sw_manuals/xilinx82/index.htm>. Acesso em: Nov 2010a.

XIPH.ORG. **Xiph.org Test Media**. Disponível em: <media.xiph.org/video/derf/>. Acesso em: Dez. 2010.

YI, X.; LING, N. Rapid Block-matching motion estimation using modified diamond search algorithm. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 5489 – 5492.

YI, X.; ZHANG, J.; LING, N. "Improved and simplified fast motion estimation for JM," JVT-P021, July 2005a.

YIN, H.; et al. A Hardware-Efficient Multi-Resolution Block Matching Algorithm and Its VLSI Architecture for High Definition MPEG-Like Video Encoders. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 20, n. 9, p. 1242–1254, Set. 2010.

ZHIHANG, Z.; et al. A novel method of motion estimation using luminance and chrominance blocks. In: IEEE INTERNATIONAL SYMPOSIUM ON SPEECH, IMAGE PROCESSING AND NEURAL NETWORKS, ISSIPNN, 1994. **Proceedings...** [S.l.]: IEEE, 1994. p. 361-364.

ZHU, C.; LIN, X.; CHAU, L. Hexagon-based Search pattern for fast Block motion estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v. 12, n. 5, p. 349 – 355, May 2002

ZHU, S.; MA, K. A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation. **IEEE Transactions on Image Processing**, [S.l.], v. 9, n. 2, p. 287-290, Fev. 2000.]

APÊNDICE A - RESULTADOS COMPLETOS DA AVALIAÇÃO DOS ALGORITMOS DE EM DESENVOLVIDOS

Este apêndice apresenta os resultados completos da avaliação dos algoritmos de ME, apresentados de modo resumido no capítulo 5.5. A Figura A.1 e a Tabela A.1 apresentam os resultados para a avaliação dos algoritmos com blocos de tamanho 8x8. A Figura A.2 e a Tabela A.2 apresentam os resultados para a avaliação dos algoritmos com blocos de tamanho 16x16. A Figura A.3 e a Tabela A.3 apresentam os resultados para a avaliação dos algoritmos com blocos de tamanho 32x32. Os gráficos das Figuras A.1, A.2 e A.3 omitem os resultados do algoritmo FS para melhor visualização dos demais resultados.

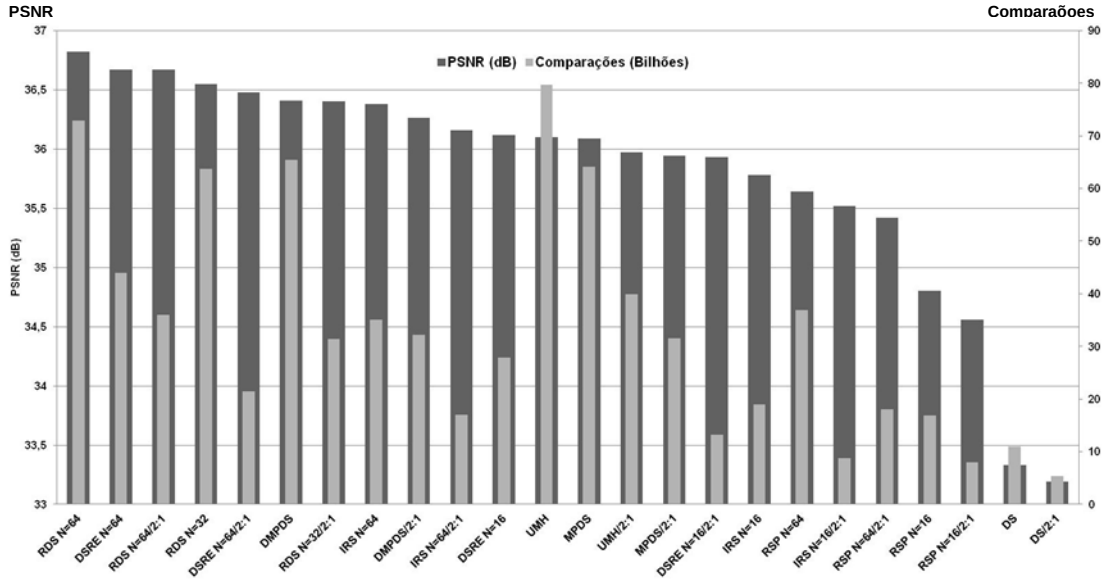


Figura A.1: Resultados de PSNR e número de comparações para os algoritmos de EM avaliados com blocos 8x8

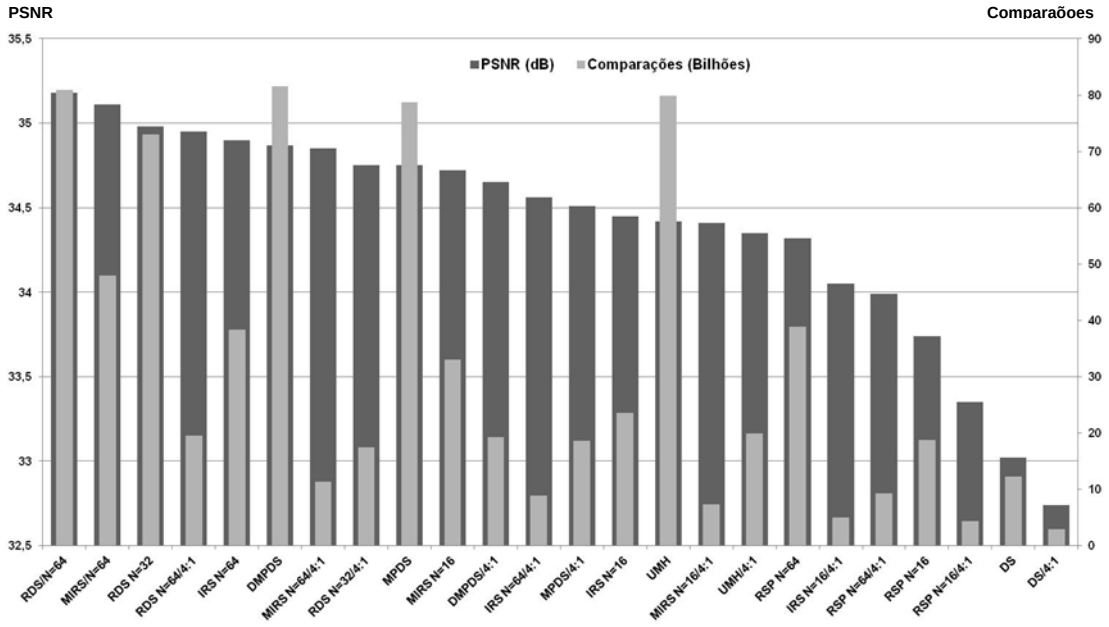


Figura A.2: Resultados de PSNR e número de comparações para os algoritmos de EM avaliados com blocos 16x16

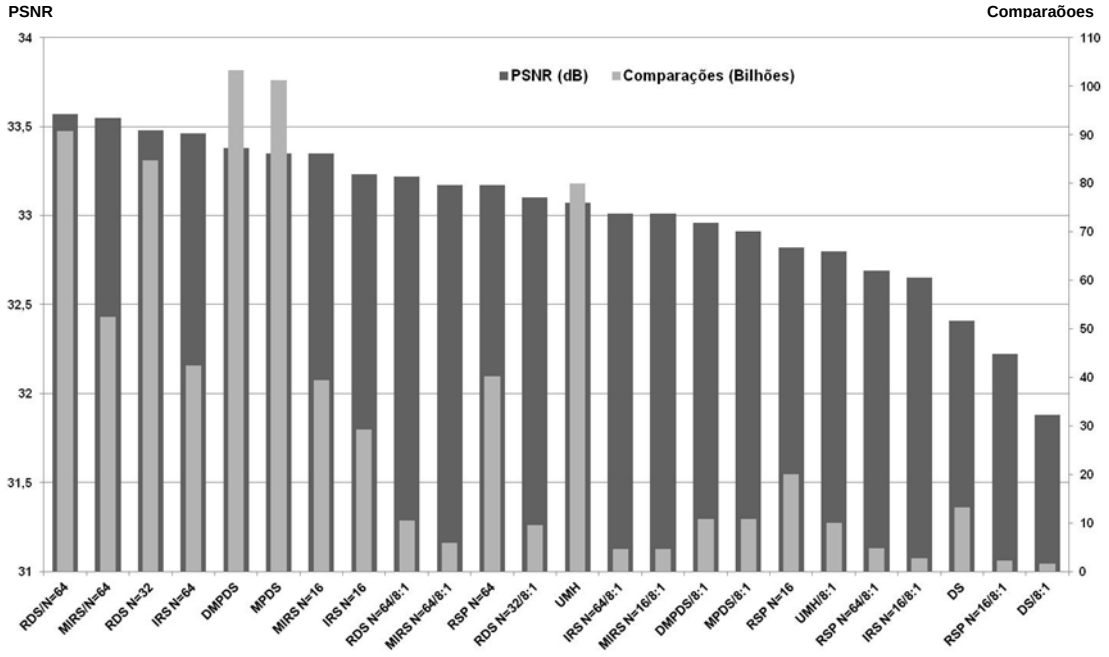


Figura A.3: Resultados de PSNR e número de comparações para os algoritmos de EM avaliados com blocos 32x32

Tabela A.1: Resultados comparativos de qualidade e custo computacional para os algoritmos de EM com tamanho de bloco 8x8

Algoritmos/sub	PSNR (dB)	Δ PSNR (dB)	Comp.x109	Δ Comp.x109 (vezes)
FS	38,25	-	3775,6	-
FS 2:1	38,16	0,09	1887,8	2,0
RDS N=64	36,82	1,43	72,89	51,8
DSRE N=64	36,67	1,58	43,94	85,9
RDS N=64/2:1	36,67	1,58	36,03	104,8
RDS N=32	36,55	1,7	63,75	59,2
DSRE N=64/2:1	36,48	1,77	21,48	175,8
DMPDS	36,41	1,84	65,42	57,7
RDS N=32/2:1	36,4	1,85	31,4	120,2
IRS N=64	36,38	1,87	35,02	107,8
DMPDS/2:1	36,26	1,99	32,16	117,4
IRS N=64/2:1	36,16	2,09	16,98	222,4
DSRE N=16	36,12	2,13	27,87	135,5
UMH	36,1	2,15	79,74	47,3
MPDS	36,09	2,16	64,17	58,8
UMH/2:1	35,97	2,28	39,86	94,7
MPDS/2:1	35,94	2,31	31,48	119,9
DSRE N=16/2:1	35,93	2,32	13,21	285,8
IRS N=16	35,78	2,47	18,94	199,3
RSP N=64	35,64	2,61	36,95	102,2
IRS N=16/2:1	35,52	2,73	8,71	433,5
RSP N=64/2:1	35,42	2,83	18,07	208,9
RSP N=16	34,8	3,45	16,83	224,3
RSP N=16/2:1	34,56	3,69	8,02	470,8
DS	33,33	4,92	11	343,2
DS/2:1	33,19	5,06	5,42	696,6

Tabela A.2: Resultados comparativos de qualidade e custo computacional para os algoritmos de EM com tamanho de bloco 16x16

Algoritmos/sub	PSNR (dB)	Δ PSNR (dB)	Comp.x10^9	Δ Comp.x10^9 (vezes)
FS	35,89	-	3753,62	-
FS/4:1	35,7	0,19	938,41	4,0
RDS/N=64	35,18	0,71	80,85	46,4
MIRS/N=64	35,11	0,78	47,9	78,4
RDS N=32	34,98	0,91	73,03	51,4
RDS N=64/4:1	34,95	0,94	19,51	192,4
IRS N=64	34,9	0,99	38,31	98,0
DMPDS	34,87	1,02	81,55	46,0
MIRS N=64/4:1	34,85	1,04	11,36	330,4
RDS N=32/4:1	34,75	1,14	17,43	215,4
MPDS	34,75	1,14	78,72	47,7
MIRS N=16	34,72	1,17	33,01	113,7
DMPDS/4:1	34,65	1,24	19,29	194,6
IRS N=64/4:1	34,56	1,33	8,95	419,4
MPDS/4:1	34,51	1,38	18,56	202,2
IRS N=16	34,45	1,44	23,5	159,7
UMH	34,42	1,47	79,86	47,0
MIRS N=16/4:1	34,41	1,48	7,35	510,7
UMH/4:1	34,35	1,54	19,95	188,2
RSP N=64	34,32	1,57	38,88	96,5
IRS N=16/4:1	34,05	1,84	4,95	758,3
RSP N=64/4:1	33,99	1,9	9,34	401,9
RSP N=16	33,74	2,15	18,73	200,4
RSP N=16/4:1	33,35	2,54	4,3	872,9
DS	33,02	2,87	12,3	305,2
DS/4:1	32,74	3,15	2,96	1268,1

Tabela A.3: Resultados comparativos de qualidade e custo computacional para os algoritmos de EM com tamanho de bloco 32x32

Algoritmos/sub	PSNR (dB)	Δ PSNR (dB)	Comp.x10^9	Δ Comp.x10^9 (vezes)
FS	33,76	-	3695,34	-
FS/8:1	33,44	0,32	461,92	8,0
RDS/N=64	33,57	0,19	90,72	40,7
MIRS/N=64	33,55	0,21	52,46	70,4
RDS N=32	33,48	0,28	84,69	43,6
IRS N=64	33,46	0,3	42,41	87,1
DMPDS	33,38	0,38	103,27	35,8
MPDS	33,35	0,41	101,23	36,5
MIRS N=16	33,35	0,41	39,33	94,0
IRS N=16	33,23	0,53	29,27	126,3
RDS N=64/8:1	33,22	0,54	10,44	354,0
MIRS N=64/8:1	33,17	0,59	5,9	626,3
RSP N=64	33,17	0,59	40,19	91,9
RDS N=32/8:1	33,1	0,66	9,48	389,8
UMH	33,07	0,69	79,91	46,2
IRS N=64/8:1	33,01	0,75	4,67	791,3
MIRS N=16/8:1	33,01	0,75	4,67	791,3
DMPDS/8:1	32,96	0,8	10,88	339,6
MPDS/8:1	32,91	0,85	10,79	342,5
RSP N=16	32,82	0,94	20,04	184,4
UMH/8:1	32,8	0,96	9,98	370,3
RSP N=64/8:1	32,69	1,07	4,72	782,9
IRS N=16/8:1	32,65	1,11	2,69	1373,7
DS	32,41	1,35	13,2	280,0
RSP N=16/8:1	32,22	1,54	2,2	1679,7
DS/8:1	31,88	1,88	1,52	2431,1

A Tabela A.4 apresenta os resultados de qualidade comparativos dos algoritmos desenvolvidos em relação ao algoritmo ATME, para as três sequências de teste apresentadas em (CETIN, 2011), que foram sumarizados no capítulo 5.5. A Tabela A. 4 apresenta os resultados em ordem decrescente em relação ao PSBR nédio obtido. A Tabela A.5 apresenta os resultados de número de comparações dos algoritmos desenvolvidos em relação ao algoritmo ATME, para as três sequências de teste apresentadas em (CETIN, 2011), que foram sumarizados no capítulo 5.5. A Tabela A. 5 apresenta os resultados em ordem crescente em relação ao número de comparações utilizadas por cada algoritmo.

Tabela A.4: Resultados comparativos de qualidade entre os algoritmos de EM desenvolvidos e o ATME

Algoritmo	PSNR (dB)			
	Crowd_run	In_to_tree	Park_joy	Média
MPDS	29,15	36,07	27,58	30,93
DMPDS	29,14	36,06	27,58	30,93
RDS	29,11	36,1	27,92	31,04
MIRS	29,06	36,06	27,75	30,96
MPDS N=16/4:1	29,04	35,76	27,36	30,72
DMPDS N=16/4:1	29,04	35,73	27,3	30,69
RDS N=16/4:1	29	35,78	27,77	30,85
IRS	28,95	36	27,56	30,84
MIRS N=16/4:1	28,94	35,75	27,5	30,73
RSP	28,79	35,93	27,16	30,63
IRS N=16/4:1	28,76	35,64	26,95	30,45
RSP N=16/4:1	28,6	35,6	26,65	30,28
ATME	25,92	34,62	28,51	29,68

Tabela A.5: Resultados comparativos de número de comparações entre os algoritmos de EM desenvolvidos e o ATME

Algoritmo	Comparações x 10^6			
	Crowd_run	In_to_tree	Park_joy	Média
RSP N=16/4:1	1,58	1,51	1,91	1,67
IRS N=16/4:1	1,98	1,67	1,9	1,85
ATME	2,14	2,04	2,11	2,10
MIRS N=16/4:1	2,76	2,43	2,98	2,72
RSP	6,4	6,18	8,7	7,09
MPDS N=16/4:1	7,78	6,41	7,84	7,34
RDS N=16/4:1	7,58	6,9	7,72	7,40
DMPDS N=16/4:1	7,63	6,76	7,93	7,44
IRS	9,57	8,04	9,64	9,08
MIRS	12,66	11,08	13,98	12,57
RDS	31,62	28,9	32,7	31,07
MPDS	33,35	27,84	34,71	31,97
DMPDS	32,64	28,97	35,13	32,25

APÊNDICE B – PUBLICAÇÕES ORIGINAIS OBTIDAS COM OS RESULTADOS APRESENTADOS NESTA TESE

Este apêndice lista as principais publicações realizadas pelo autor com os resultados dos desenvolvimentos algorítmicos e arquiteturais apresentados nesta tese.

Os resultados dos desenvolvimentos algorítmicos apresentados nesta tese foram tratados nas seguintes publicações:

- (PORTO, NOBLE, AGOSTINI, BAMPI) - Two Fast Multi-Point Search Algorithms for High Quality Motion Estimation in High Resolution Videos. International Journal of Information Technology, Communications and Convergence (IJITCC, 2011).
- (CRISTANI, DALL'OGLIO, PORTO, AGOSTINI, BAMPI) - A Fast Random Based Motion Estimation Search Algorithm. Microelectronics Students Forum (SForum, 2011). Este artigo recebeu o prêmio Best Paper Award do evento.
- (NOBLE, PORTO, AGOSTINI, ARAÚJO, LAMB) - Two Novel Algorithms for High Quality Motion Estimation in High Definition Video Sequences. Conference on Graphics, Patterns and Images (SIBGRAPI, 2011).
- (PORTO, CRISTANI, DALL'OGLIO, GRELLERT, MATTOS, BAMPI, AGOSTINI) - Iterative Random Search: A New Local Minima Resistant Algorithm for Motion Estimation in High Definition Videos. Multimedia Tools and Applications (MTAP, 2012). Artigo aceito para publicação.

Os resultados dos desenvolvimentos arquiteturais apresentados nesta tese estão incluídos nas seguintes publicações:

- (PORTO, AGOSTINI, BAMPI, SUSIN) - A High Throughput and Low Cost Diamond Search Architecture for HDTV Motion Estimation. IEEE International Conference on Multimedia & Expo (ICME, 2008).
- (PORTO, AGOSTINI, SUSIN, BAMPI) - Architectural Design for the New QSDS with Dynamic Iteration Control Motion Estimation Algorithm Targeting HDTV. Symposium on Integrated Circuits and System Design (SBCCI, 2008).
- (PORTO, ALTERMANN, COSTA, BAMPI) - Power Efficient Architecture for Motion Estimation Using the QSDS-DIC Algorithm. IEEE International Conference on Electronics, Circuits, and Systems (ICECS, 2009).
- (ROSA, PORTO, MATOS, SUSIN, BAMPI) - Arquitetura de Alto Desempenho para o Algoritmo de Estimação de Movimento SDS-DIC com Múltiplos Quadros de Referência. (Workshop Iberchip, 2009).

- (SILVA, PORTO, ALMEIDA, COSTA, BAMPI) - High Performance Motion Estimation Architecture Using Efficient Adder-Compressors. Symposium on Integrated Circuits and Systems Design (SBCCI, 2009).
- (PORTO, SILVA, ALMEIDA, COSTA, BAMPI) - Motion Estimation Architecture Using Efficient Adder-Compressors for HDTV Video Coding. Journal of Integrated Circuits and Systems (JICS, 2010).
- (PORTO, ALTERMANN, COSTA, AGOSTINI, BAMPI) - A Real Time and Power Efficient HDTV Motion Estimation Architecture Using Adder-Compressors. Latin American Symposium on Circuits and Systems (LASCAS, 2011).
- (SANCHEZ, NOBLE, PORTO, AGOSTINI) - A Real Time HDTV Motion Estimation Architecture for the New MPDS Algorithm. International Conference on Computer as a Tool (EUROCON, 2011).
- (SANCHEZ, NOBLE, PORTO, AGOSTINI) - High Efficient Motion Estimation Architecture with Integrated Motion Compensation and FME Support. Latin American Symposium on Circuits and Systems (LASCAS, 2011).
- (CRISTANI, DALL'OGLIO, PORTO, AGOSTINI, BAMPI) - Hardware Design for the New Galaxy Random Search Motion Estimation Algorithm Focusing in HD Videos. Workshop on Circuits and Systems (WCAS, 2011).
- (PORTO, SANCHEZ, NOBLE, AGOSTINI, BAMPI) - An Efficient ME Architecture for High Definition Videos Using the New MPDS Algorithm. Symposium on Integrated Circuits and Systems Design (SBCCI, 2011).
- (SANCHEZ, PORTO, AGOSTINI, BAMPI) - Real Time QFHD Motion Estimation Architecture for DMPDS Algorithm. Southern Programmable Logic Conference (SPL, 2012). Artigo aceito para publicação.