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

MARCELO SCHIAVON PORTO

# Arquiteturas de Alto Desempenho e Baixo Custo em Hardware para a Estimação de Movimento em Vídeos Digitais

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Altamiro Amadeu Susin
Orientador

Prof. Dr. Sergio Bampi
Co-orientador

Porto Alegre, março de 2008.

**CIP – CATALOGAÇÃO NA PUBLICAÇÃO**

Porto, Marcelo Schiavon

Arquiteturas de Alto Desempenho e Baixo Custo em Hardware para a Estimação de Movimento em Vídeos Digitais / Marcelo Schiavon Porto – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

100 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Orientador: Altamiro Amadeu Susin; Co-orientador: Sergio Bampi.

1.Compressão de Vídeo. 2.Estimação de Movimento 3.Arquiteturas VLSI. I. Susin, Altamiro Amadeu; Bampi, Sergio. III. Título.

# AGRADECIMENTOS

Gostaria de agradecer, primeiramente, aos meus pais Roberto Garcia Porto e Ivone Schiavon Porto, por terem me educado e me proporcionado a oportunidade de estudar. Pelo apoio que sempre me deram ao longo da vida, e que não foi diferente durante esse período de mestrado. Agradeço pelos momentos alegres que passamos juntos, sempre que voltava para a casa deles, que me dava ânimo novo para retornar ao trabalho. Sinto muita falta da companhia diária de vocês, agora que estamos distantes, mas isso não diminui o carinho e a admiração que sinto pelos dois, amo vocês.

Queria agradecer também ao meu irmão, Matheus Porto, que além de irmão é um grande amigo. Mesmo distante, ele esteve sempre presente, quando ficávamos por horas conversando pela internet.

Outra pessoa que merece um agradecimento especial é minha tia Maria Helena Schiavon. Ela é minha segunda mãe e sempre esteve disposta para me ajudar nas horas difíceis, seja na hora de alugar um apartamento, seja quando o salário atrasa ou simplesmente com o carinho me dá toda vez que nos encontramos.

Gostaria de agradecer a minha namorada Júlia Islabão, que entrou na minha vida novamente ao longo deste período de mestrado. Agradeço por estar comigo, e suportar a distância e a ausência constante do nosso namoro.

Sou muito grato também aos professores Altamiro Susin e Sergio Bampi, meus orientadores. Muito obrigado por terem me aceitado como seu aluno de mestrado, foi um grande prazer trabalhar estes dois anos junto com professores tão capacitados e experientes. Espero que possamos trabalhar juntos por muito tempo, e espero também poder retribuir a toda a confiança que os senhores me deram.

Um agradecimento muito especial ao amigo, e orientador informal, Luciano Agostini. O Luciano tem me apoiado desde 2003, quando começamos a trabalhar juntos no GACI, eu como bolsista IC e ele como meu orientador e professor da graduação. Sem dúvida foi quem me inspirou a seguir em frente e entrar para o mestrado. Muito obrigado pelo tempo precioso que destinastes para me ajudar ao longo deste mestrado, seja com discussões técnicas, revisões de *papers*, propostas para trabalhos de disciplinas e até mesmo na revisão do texto desta dissertação. Espero que possamos continuar trabalhando juntos por muito tempo, e que algum dia eu consiga retribuir todo o apoio que sempre me deste.

Gostaria de agradecer também aos bolsistas IC do GACI, Leandro Zanetti, Fabiane Ridiess e Rafael Petry, que contribuíram de forma significativa para a conclusão deste trabalho. O apoio de vocês foi fundamental, espero que esse período de trabalho conjunto também tenha sido proveitoso para vocês, e que possamos continuar trabalhando juntos.

Aos meus amigos e colegas de laboratório na UFRGS, Dieison Deprá, Cláudio Diniz, Guilherme Freitas e Roger Porto, com quem convivi diariamente durante este período de mestrado, meu muito obrigado. Um agradecimento especial ao amigo e colega Bruno Zatt, com quem, nas conversas da "hora do cafezinho", tive diversas discussões de onde surgiram idéias que acabaram sendo desenvolvidas neste trabalho.

Devo agradecer também a minha prima Raquel Schiavon e seu noivo Felipe Knorr, assim como a Ieda Oliveira e Álvaro Oliveira pelas diversas caronas que me deram ao longo desses dois anos de mestrado. Essas viagens a Pelotas foram muito importantes, era lá que podia relaxar e voltar com disposição para o trabalho.

Por fim, gostaria de agradecer a todos os amigos e colegas que de alguma forma me apoiaram durante este período e que não foram nominalmente citados.

# SUMÁRIO

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ACC | Acumulador |
| ASIC | *Application-specific Integrated Circuit* |
| BRAM | *Block* RAM |
| BS | *Block Subsampling* |
| BE | Bloco Escolhido |
| Cb | *Chrominance blue* |
| CDL | Controle Dinâmico de Laços |
| CIF | *Common Intermediate Format* |
| CLB | *Configurable Logic Block* |
| Cr | *Chrominance red* |
| DCS | *Dual Cross Search* |
| DDR | *Double Data Rate* |
| DS | *Diamond Search* |
| DVD | *Digital Versatile Disk* |
| FPGA | *Field Programmable Gate Array* |
| FS | Full Search |
| FTS | *Flexible Triangle Search* |
| HDTV | *High Definition Digital Television* |
| HS | *Hexagon Search* |
| HSI | *Hue, Saturation, Intensity* |
| IEC | *International Electrotechnical Commission* |

| | |
|---|---|
| ISO | *International Organization for Standardization* |
| ITU-T | International Telecommunication Union - Telecommunication |
| LDSP | *Large Diamond Search Pattern* |
| 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* |
| ML | Memória Local |
| MPEG | *Moving Picture Experts Group* |
| MSE | *Minimum Square Error* |
| VM | Vetor de Movimento |
| OTS | *One at a Time Search* |
| PSNR | *Peank-to-Signal Noise Ratio* |
| PMVFAST | Predictive Motion Vector Field Adaptive Search Technique |
| QCIF | *Quarter Common Intermediate Format* |
| QHDTV | *Quad High Definition Digital Television* |
| QSDS | Quarter Sub-sampled Diamond Search |
| RAM | *Random Access Memory* |
| RD | Registrador de deslocamento |
| RGB | *Red, Green, Blue* |
| SAD | *Sum of Absolute Differences* |
| SBTVD | Sistema Brasileiro de Televisão Digital |
| SDS | Sub-sampled Diamond Search |

| | |
|---|---|
| SDSP | *Small Diamond Search Pattern* |
| SDTV | *Standart Definition Television* |
| T | TransformadaTSMC*Taiwan Semiconductor Manufacturing Company* |
| TIC | Tecnologia de Informação e Comunicação |
| TSS | *Three Step Search* |
| UFPel | Universidade Federal de Pelotas |
| UFRGS | Universidade Federal do Rio Grande do Sul |
| UP | Unidade de Processamento |
| VGA | *Video Graphics Adapter* |
| VHDL | VHSIC *Hardware Description Language* |
| VHSIC | *Very High Speed Integrated Circuit* |
| VQEG | *Video Quality Experts Group* |
| Y | *Luminance* |
| YCbCr | *Luminance, Chrominance Blue, Chrominance Red* |

# LISTA DE FIGURAS

# LISTA DE TABELAS

# RESUMO

A evolução das Tecnologias de Informação e Comunicação (TIC) favoreceu o crescimento do uso de variados meios na comunicação. Entre diversos meios, o vídeo em particular, necessita de uma grande banda para ser transmitido, ou de um grande espaço para ser armazenado. Uma análise dos diversos sinais de uma comunicação multimídia mostra, entretanto, que existe uma grande redundância de informação. Utilizando técnicas de compressão é possível reduzir de uma a duas ordens de grandeza a quantidade de informação veiculada, mantendo uma qualidade satisfatória. Uma das formas de compressão busca a relação de similaridade entre os quadros vizinhos de uma cena, identificando a redundância temporal existente entre as imagens. Essa técnica chama-se estimação de movimento, este processo é muito eficaz, mas o custo computacional é elevado, exigindo a implementação de algoritmos eficientes em hardware, para o caso de compressão em tempo real de vídeos de alta resolução.

Esta dissertação apresenta uma investigação sobre algoritmos de estimação de movimento visando implementações em hardware. Todos os algoritmos foram desenvolvidos primeiramente em linguagem C e submetidos a diversos testes para avaliação de desempenho e custo computacional. Os algoritmos foram aplicados a diversas amostras de vídeo utilizadas pela comunidade científica, para avaliação em aplicações reais. As avaliações demonstraram que os algoritmos rápidos conseguem realizar o processo de estimação de movimento de maneira eficiente, obtendo bons resultados em termos de qualidade de vetores, esforço computacional e desempenho. Com as análises dos resultados obtidos, o algoritmo Busca Diamante (*Diamond Search*) foi escolhido para ser implementado em hardware, com dois níveis diferentes de sub-amostragem de *pixel*: 2:1 e 4:1.

As arquiteturas para o algoritmo Busca Diamante, com sub-amostragem de *pixel* de 2:1 e 4:1, foram descritas em VHDL, sintetizadas para FPGAs Virtex-4 da Xilinx e também para *standard cells* na tecnologia TSMC 0,18µm. Os resultados mostram que as arquiteturas desenvolvidas possuem desempenho superior ao necessário para tratar vídeos HDTV 1080p em tempo real a 30 quadros por segundo. As arquiteturas desenvolvidas também apresentam um baixo consumo de recursos de hardware, após a síntese para FPGA e ASIC.

**Palavras-Chave:** Compressão de vídeo, estimação de movimento, arquiteturas VLSI.

**High Performance and Low Cost Hardware Architectures for Digital Videos Motion Estimation**

# ABSTRACT

The evolution of the communication and information technologies push the development of several communication media. These media, video in particular, need a large bandwidth to be transmitted, or a large digital storage capacity. Many multimedia signals show, however, a high information redundancy. By using compression techniques it is possible to reduce the amount of coded information by one or two orders of magnitude, keeping a satisfactory visual quality. One of these compression techniques searches the similarity between neighboring frames of a scene, identifying the temporal redundancy between them. This technique is called motion estimation, and it is a very efficient method for compression. However, the computational complexity of the motion estimation requires high performance algorithms in hardware, when used for real time compression of high resolution videos.

This dissertation presents a comprehensive investigation about motion estimation algorithms, targeting a hardware implementation. All the investigated algorithms were first developed in C language and submitted to many evaluation tests. The algorithms were applied to ten video samples used by the scientific community for the evaluation of real application. The evaluation showed that fast algorithms can carry out the motion estimation process efficiently, producing good results in vectors quality, computational effort and performance. With the results analyses, the Diamond Search algorithm was chosen to be hardware designed, with two different levels of pixel subsampling, 2:1 and 4:1.

The architectures for Diamond Search algorithm, with pixel subsampling of 2:1 and 4:1, were described in VHDL, synthesized to Xilinx Virtex-4 FPGAs and also to standard cells TSMC 0.18µm technology. The developed architectures have sufficient performance to process HDTV 1080p videos at 30 frames per second and demand small hardware resources consumption after synthesis to FPGA and ASIC.

**Keywords:** Video compression, motion estimation, VLSI design.

# 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 tornam-se praticamente impossíveis se considerarmos vídeos digitais sem compressão. Por exemplo, considerando vídeos com resolução de 720x480 *pixels* a 30 quadros por segundo (usado em televisão digital com definição normal – SDTV e em DVDs), utilizando 24 bits por pixel, a taxa necessária para a transmissão sem compressão seria próxima a 249 milhões de bits por segundo (249 Mbps). Para armazenar uma seqüência de curta duração, com 10 minutos, seriam necessários quase 19 bilhões de bytes (19GB). Para vídeos com resolução de 1920x1080 *pixels* a 30 quadros por segundo (usado em televisão digital com alta definição ou HDTV), com 24 bits por pixel, a taxa de transmissão sobe para 1,5 bilhões de bits por segundo (1,5 Gbps) e seriam necessários 112 bilhões de bytes (112 GB) para armazenar um vídeo com 10 minutos.

Os vídeos digitais utilizam uma grande quantidade de informações para serem representados, no entanto, em geral, os vídeos digitais apresentam uma grande quantidade de informações redundantes. Estas informações redundantes podem ser encontradas e eliminadas, reduzindo significativamente a quantidade de informação necessária para representar os vídeos digitais. O processo de compressão de vídeo utiliza um conjunto de técnicas que busca identificar e eliminar as redundâncias dos vídeos digitais. Diversos padrões de compressão de vídeo foram desenvolvidos, agregando várias técnicas de compressão que atacam diversos tipos de redundância presentes nos vídeos digitais. Com a utilização destes padrões de compressão, foi possível obter uma compressão significativa na quantidade de informações necessária para representar os vídeos digitais, tornando possível a manipulação de vídeos em diversos dispositivos como celulares, DVD *players*, televisão digital de alta definição (HDTV), entre outros.

Os padrões de compressão de vídeo são formados por um conjunto de algoritmos e técnicas que tentam reduzir o número de informações necessárias 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 as quais as informações que compõem o vídeo são submetidas, dentre elas as principais são: Estimação de movimento (ME), Compensação de movimento, Transformadas, Quantização e Codificação de entropia (BHASKARAN, 1999). Dentre as etapas dos compressores atuais, a mais complexa e que resulta nos melhores resultados de taxa de compressão é a estimação de movimento (ME – *Motion Estimation*). A ME é responsável por encontrar uma co-relação entre quadros, mapeando a redundância temporal entre os quadros vizinhos de uma cena. O processo de estimação é extremamente complexo, isto dificulta o desenvolvimento da estimação de movimento

em software para tratar vídeos de alta resolução em tempo real. Isto faz com que o desenvolvimento de arquiteturas de hardware dedicadas para este fim, seja indispensável quando se deseja trabalhar com vídeos de alta resolução 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. Este trabalho apresenta uma ampla investigação sobre algoritmos rápidos de estimação de movimento, visando uma implementação em hardware. Diversos algoritmos foram estudados e implementados em software para as análises. Várias métricas de comparação entre os algoritmos foram desenvolvidas, como qualidade dos vetores gerados, custo computacional, entre outras. Os algoritmos desenvolvidos em software foram aplicados a diversas amostras de vídeo, para que resultados médios para aplicações reais pudessem ser encontrados. Todas as análises foram desenvolvidas de maneira independente e sem a intenção de favorecer nenhum algoritmo. Esta avaliação dos algoritmos em software tem como objetivo ajudar o projetista na escolha do algoritmo ideal para a sua aplicação seja ela em software ou em hardware. Ao final deste estudo, o algoritmo *Diamond Search* (DS) foi o algoritmo que obteve o maior destaque dentre todos os algoritmos avaliados.

Esta dissertação também apresenta duas arquiteturas de hardware para a estimação de movimento. O algoritmo DS foi desenvolvido em hardware com duas versões diferentes de sub-amostragem de *pixel*, 2:1 e 4:1. Estas arquiteturas foram descritas em VHDL e sintetizadas para FPGAs da Xilinx. Também foram geradas versões em *standard cells* na tecnologia TSMC 0,18µm. As duas arquiteturas desenvolvidas trabalham com blocos de tamanho 16x16 e utilizam o SAD como critério de distorção. Os resultados de síntese mostram que as arquiteturas desenvolvidas podem aliar um baixo consumo de recursos de hardware com altas taxas de processamento. As duas arquiteturas desenvolvidas são capazes de tratar vídeos HDTV 1920x1080 em tempo real a 30 quadros por segundo. Os resultados obtidos com a síntese das arquiteturas são comparados com diversas soluções encontradas na literatura, tanto para implementações em FGPA, quanto para soluções em *standard cells*. Também é apresentada uma comparação, entre os resultados obtidos com as arquiteturas desenvolvidas neste trabalho e soluções desenvolvidas anteriormente pelo nosso grupo de pesquisa.

Este trabalho está inserido no esforço acadêmico para construir o Sistema Brasileiro de TV Digital (SBTVD), através da proposta da construção do codec H.264. Todas as soluções arquiteturais desenvolvidas neste trabalho possuem desempenho suficiente para serem integradas aos demais blocos do codificador. As soluções arquiteturais desenvolvidas possuem um baixo custo em hardware. Esta característica é muito importante na hora da integração de todos os blocos que compões o codificador.

A estrutura do texto desta dissertação está organizada da seguinte maneira. O capítulo dois desta dissertação apresenta alguns conceitos básicos de compressão de vídeo digital, apresentando conceitos e fornecendo informações que serão úteis no decorrer do texto. No capítulo três, são apresentados alguns conceitos da estimação de movimento, comentando o papel da ME no contexto da codificação de vídeo. Também são apresentados os algoritmos estudados com uma explicação detalhada das suas técnicas, bem como a descrição do critério de similaridade utilizado nas avaliações dos algoritmos. O capítulo quatro apresenta os resultados das implementações em C para os algoritmos de estimação de movimento estudados, juntamente com as análises comparativas visando à implementação em hardware. Os capítulos cinco e seis apresentam as arquiteturas de hardware desenvolvidas para os algoritmos SDS e QSDS

respectivamente. O capítulo sete apresenta uma comparação entre os resultados obtidos pala síntese das arquiteturas desenvolvidas neste trabalho e soluções encontradas na literatura. Por fim, o capítulo oito apresenta as conclusões e propostas de trabalhos futuros.

# 2 CONCEITOS DE COMPRESSÃO DE VÍDEO DIGITAL

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

## 2.1 Conceitos Básicos de vídeo Digital

Um vídeo digital é formado por uma seqüência de imagens, que por sua vez são formadas por pontos (*pixels*). Pontos pertencentes à mesma imagem e, espacialmente vizinhos, são chamados de pontos vizinhos. Imagens, também chamadas de quadros (*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.

Todo vídeo digital é formado por uma seqüência de quadros, que por sua vez, são divididos em blocos. Estes blocos podem ter tamanhos variados, mas comumente são utilizados tamanhos de 4x4, 8x8 ou 16x16 *pixels*. A Figura 1.1 ilustra uma seqüência de quadros de um vídeo digital, salientando a divisão dos blocos do quadro.

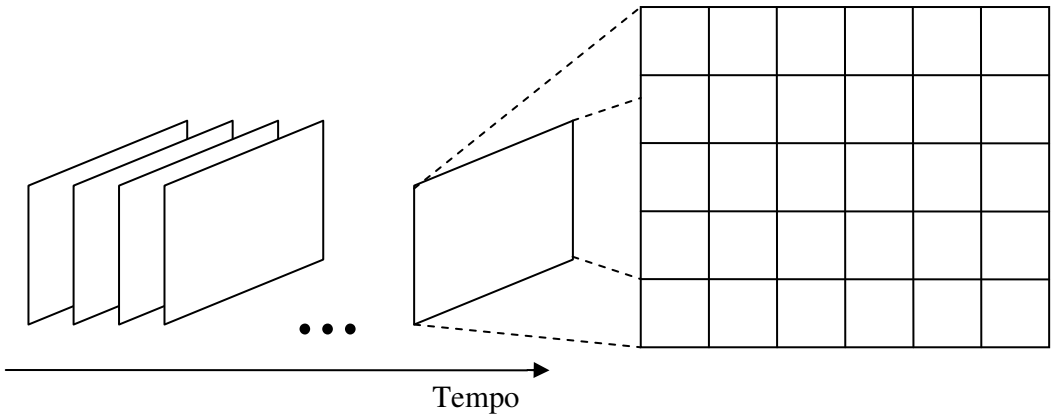



Figura 2.1: Seqüência de quadros em um vídeo digital

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. A amostragem temporal é realizada como uma série de quadros, estáticos, amostrados a certos intervalos de tempo, conforme ilustrado na Figura 2.1. 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 cor da amostra, ou o brilho e cor, 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. Para o caso de captura de imagens coloridas, cada componente de cor é separada e projetada em um sensor.

### 2.1.1 Amostragem espacial

A saída do sensor da câmera é um sinal de vídeo analógico, uma variação do sinal elétrico que representa uma imagem do vídeo. Amostrando o sinal em intervalos de tempo se produz uma imagem amostrada, ou quadro, que tem valores definidos em um conjunto de pontos de amostragem. A forma mais comum da imagem amostrada é um retângulo com os pontos de amostragem posicionados numa matriz quadrada ou retangular. A amostragem ocorre em cada intersecção das linhas da matriz e a imagem amostrada pode ser reconstruída representando cada amostra como um elemento de figura quadrado. A qualidade da imagem é influenciada pela quantidade de pontos de amostragem (*pixels*). Quanto mais espaçados são os pontos de amostragem menor a resolução da imagem amostrada, assim como quanto mais próximos são os pontos de amostragem maior é a resolução da imagem amostrada, e conseqüentemente melhor será o resultado visual da imagem.

### 2.1.2 Amostragem temporal

Cada quadro de um vídeo é capturado por um sistema eletrônico de sensores, em intervalos regulares de tempo. Quando esta seqüência de quadros é apresentada de forma contínua no tempo, obtém-se a sensação de movimento. Quanto mais alta é a freqüência em que os quadros são capturados (*frame rate*) mais suave será a sensação de movimento, no entanto, uma quantidade maior de amostras deve ser capturada e armazenada, aumentando a complexidade para o tratamento e armazenamento desses vídeos. O padrão de freqüência de amostragem das TVs fica entre 25 e 30 quadros por segundo. Taxas entre 10 e 20 quadros por segundo são utilizadas para comunicação de vídeos em canais de transmissão de baixa capacidade, taxas de 50 ou 60 quadros por segundo são utilizadas para vídeos de alta qualidade.

### 2.1.3 Quadros e Campos

Um sinal de vídeo pode ser amostrado como uma série de quadros progressivos (*frames – progressive sampling*) ou como uma seqüência de campos entrelaçados (*interlaced sampling*). Numa seqüência de vídeo entrelaçado, metade dos dados de um quadro (um campo) é amostrada a cada intervalo temporal de amostragem. A Figura 2.2 ilustra os campos das linhas pares e ímpares de um quadro, bem como uma seqüência de campos em um vídeo entrelaçado. Um campo é formado apenas por linhas pares ou por linhas ímpares de um quadro de uma seqüência inteira de quadros. Uma seqüência de vídeo entrelaçado contém uma seqüência de campos, cada um representando metade da informação de um quadro do vídeo. A vantagem desse tipo de amostragem é que é possível enviar dois campos no mesmo intervalo de tempo em que seria possível enviar um quadro de um vídeo progressivo, ou seja, com o mesmo *data rate*.

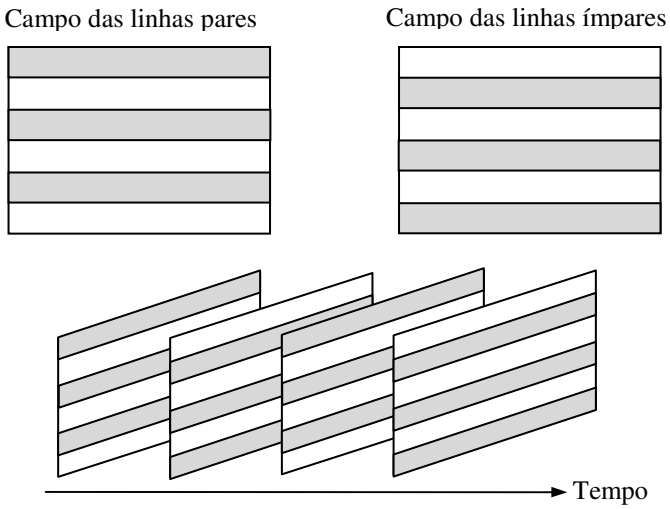



Figura 2.2: Campos e seqüência de um vídeo entrelaçado

## 2.2 Sub-amostragem de Cores

A representação digital de um vídeo colorido está associada à interpretação das cores pelo sistema visual humano. Existem muitas formas de se 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.

No espaço de cores YCbCr, as três componentes utilizadas são luminância (Y), que define a intensidade luminosa ou o brilho, crominância azul (Cb) e crominância vermelha (Cr) (MIANO, 1999). Os componentes R, G e B possuem um elevado grau de correlação, tornando difícil o processamento de cada uma das informações de cor de forma independente. Por isso, a compressão de vídeos é aplicada para 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 brilho. Deste modo, estas informações podem ser tratadas de forma diferenciada pelos compressores de imagens estáticas e vídeos.

O sistema visual humano é menos sensível às informações de cor contidas nas imagens (GONZALEZ, 2003). Então os padrões de compressão de imagens estáticas e vídeos podem explorar esta característica psicovisual 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 sub-amostragem 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 sub-amostragem. 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). Assim, a sub-amostragem não está sendo aplicada. No formato 4:2:2, para cada quatro informações de Y apenas duas informações de Cb e Cr são representadas, e para o formato 4:2:0, para cada quatro informações de Y apenas uma informação de Cb e uma de Cr são representadas.

A sub-amostragem 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, uma vez que cada componente de crominância possui exatamente um quarto das amostras presentes no componente de luminância, então 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 taxa de compressão de 50%, considerando apenas a sub-amostragem.

## 2.3 Redundância de Informação em Vídeos Digitais

A compressão é baseada na eliminação de dados redundantes existentes em vídeos e imagens. Um dado é considerado redundante quando seu valor não representa uma nova informação relevante para a representação da imagem. Basicamente, existem quatro tipos diferentes de redundâncias exploradas na compressão de vídeos: redundância espacial, redundância temporal, redundância psicovisual e redundância entrópica. Existem controvérsias entre os autores a respeito da definição dos tipos de redundância. A classificação apresentada neste trabalho, com quatro diferentes tipos de redundância, é baseada em (SHI, 1999) e (GONZALEZ, 2003).

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 conseqüência, com elevado grau de redundância de informação, gerando pouco acréscimo de informação visual a cada ponto em relação aos vizinhos.

As características do sistema visual humano fazem com que não consigamos captar alguns tipos de informações que podem estar presentes na imagem. Além disso, algumas informações da imagem, como o brilho, por exemplo, são mais importantes para o sistema visual humano do que outras, como as cores. Este tipo de redundância de informação é chamado de redundância psicovisual (GONZALEZ, 2003). Para explorar este tipo de redundância, parte da informação original da imagem é eliminada de forma irreversível pelo codificador. Existem dois tipos principais de processos utilizados para este fim. O primeiro, chamado de sub-amostragem é utilizado na entrada do vídeo e elimina parte das informações de cores. O segundo, conhecido por quantização, normalmente é aplicado no domínio das freqüências e elimina ou atenua as freqüências de menor percepção 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.

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.

A redundância temporal, também chamada de "redundância 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.

# 3 A ESTIMAÇÃO DE MOVIMENTO

Quando se analisa um trecho de um vídeo, podemos perceber que as imagens vizinhas são muito similares. As diferenças, quando existem, são geradas em sua maioria por movimentos da câmera ou de objetos pertencentes à cena. Esta característica dos vídeos produz uma grande redundância temporal, ou seja, se comparados dois quadros vizinhos pode-se perceber uma enorme quantidade de informações repetidas, causadas por regiões dentro do quadro que não são alteradas de um quadro para outro.

As Figuras. 3.1 e 3.2 representam, respectivamente, o primeiro e o segundo quadro de uma seqüência de vídeo. Pode-se observar que as diferenças entre os quadros 1 e 2 são praticamente imperceptíveis. Grande parte da imagem permanece estática e não sofre alterações de um quadro para outro. A Figura 3.3 ilustra as diferenças entre os dois quadros. Esta imagem foi gerada a partir da subtração *pixel* a *pixel* entre os quadros 1 e 2. Nas regiões onde a imagem não se altera o valor dos *pixels* também não se altera, o que resulta em um valor nulo após a subtração. Para as regiões que sofrem algum tipo de alteração, a diferença entre os *pixels* dos dois quadros resulta em um valor não nulo. Para facilitar a percepção das diferenças de movimento de um quadro para outro, o resultado da subtração foi somado a 128. Deste modo, os pontos em zero (pontos idênticos nos dois quadros) passaram a possuir o valor 128, que é o cinza, predominante na Figura 3.3.

O objetivo principal da estimação de movimento é identificar a redundância temporal entre imagens vizinhas e mapeá-las através de vetores de movimento. A partir de um ou mais quadros, denominados de quadros de referência, o quadro atual, que é o quadro para o qual se deseja realizar a estimação, é estimado usando vetores de movimento obtidos a partir do(s) quadro(s) de referência. Para cada bloco do quadro atual será gerado um vetor de movimento que corresponderá ao deslocamento deste bloco em relação ao quadro de referência.

Uma área de pesquisa será determinada no quadro de referência para cada bloco do quadro atual. A área de pesquisa pode conter o quadro inteiro, mas normalmente esta área é definida como uma fração do quadro para diminuir a complexidade computacional da estimação de movimento. Cada bloco da área de pesquisa é chamado de bloco candidato, pois qualquer um deles pode ser escolhido no processo de estimação. Um algoritmo de busca determina a maneira como esse bloco se desloca dentro desta área de pesquisa para que a melhor relação de similaridade (melhor *matching*) seja encontrada. A Figura 3.4 ilustra um quadro de referência e um quadro

atual. No quadro de referência está destacada a área de pesquisa e o bloco que está sendo pesquisado.

Figura 3.1: Quadro 1 do vídeo Telefone (VQEG, 2006)

Figura 3.2: Quadro 2 do vídeo Telefone (VQEG, 2006)

Figura 3.3: Resultado da subtração entre os Quadros 1 e 2

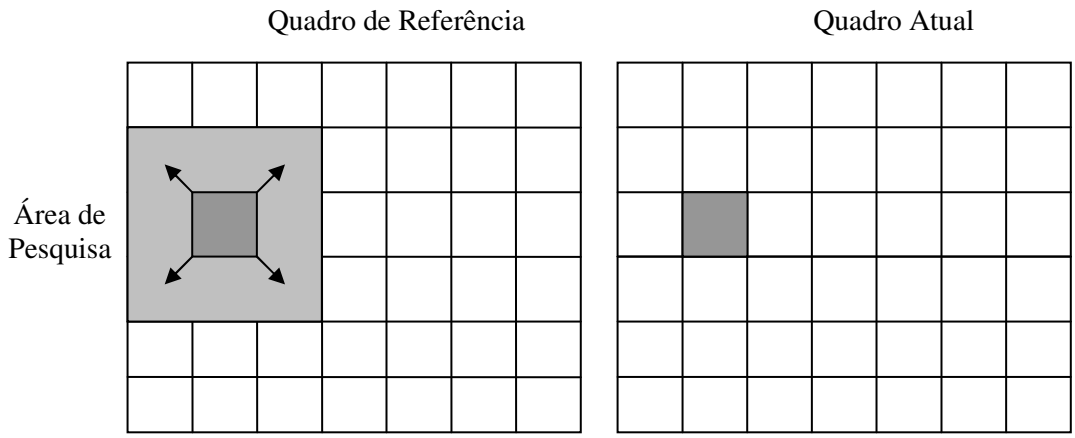



Figura 3.4: Determinação do vetor de movimento para um bloco

A informação do vetor de movimento é gerada para cada bloco de luminância do quadro atual, já que as informações de crominância são irrelevantes para a estimação de movimento. Esta operação implica em uma grande complexidade computacional, no entanto, quando encontrado um bom casamento, proporciona uma redução significativa da quantidade de informação necessária para formar o vídeo, pois todas as informações contidas em um bloco são substituídas por um vetor de duas dimensões e mais um resíduo. O resíduo é a diferença *pixel* a *pixel* do bloco do quadro atual e o bloco escolhido do quadro de referência. O resíduo, além de possuir baixa amplitude, tendendo a zero, ainda é muito comprimido através de outras técnicas de compressão, presentes nos codificadores de vídeo (RICHARDSON, 2002).

Alguns critérios devem ser cuidadosamente analisados para se obter o resultado esperado, tais como, o tamanho da área de pesquisa e dos blocos dos quadros. O tamanho da área de pesquisa pode influenciar diretamente na qualidade dos vetores gerados e também na complexidade computacional do processo de estimação. Quanto maior a área de pesquisa, maior será a chance de se encontrar um bloco que possua o melhor casamento com relação ao bloco que está sendo pesquisado. No entanto, a complexidade computacional também cresce, pois a quantidade de blocos candidatos também aumenta. O tamanho dos blocos deve ser suficientemente pequeno para que áreas menores da imagem sejam comparadas, gerando erros menores. No entanto, deve ser grande o suficiente para que não exista um número muito grande de blocos em um quadro, pois para cada bloco é necessário um vetor, que deve ser representado por um par de números inteiros. Estes vetores devem ser enviados junto com a informação comprimida e, se existe um número muito grande de vetores, o ganho obtido através dos erros menores é perdido em função da codificação do próprio vetor de movimento.

Diversos fatores devem ser observados para a implementação de um estimador de movimento. Fatores como a complexidade computacional e a qualidade e quantidade dos vetores são sempre contraditórios. Além disso, outro fator importante é a taxa de processamento, que para vídeos em tempo real deve estar entre 24 e 30fps (quadros por segundo). Considerando estes fatores e as necessidades específicas que se deseja, sejam elas qualidade ou taxa de processamento, é que devemos determinar as características do estimador. A complexidade computacional agregada ao processo de estimação torna praticamente impossível atender os requisitos de desempenho para tratar vídeos de alta resolução em tempo real com aplicações em *software*. Isto faz com que implementações em hardware de alto desempenho para a estimação de movimento sejam extremamente importantes

## 3.1 ME na Codificação de Vídeo

Para identificar o papel da ME na codificação de vídeo, um modelo genérico de codificador será apresentado, com explicações sucintas sobre cada bloco. A Figura 3.5 apresenta o diagrama em blocos do codificador de vídeo genérico adotado. Os principais blocos que compõem o codificador são: Codificação inter, codificação intra, transformadas (T), quantização e codificação de entropia. Cada um destes blocos aborda um determinado tipo de redundância existente nos vídeos digitais.

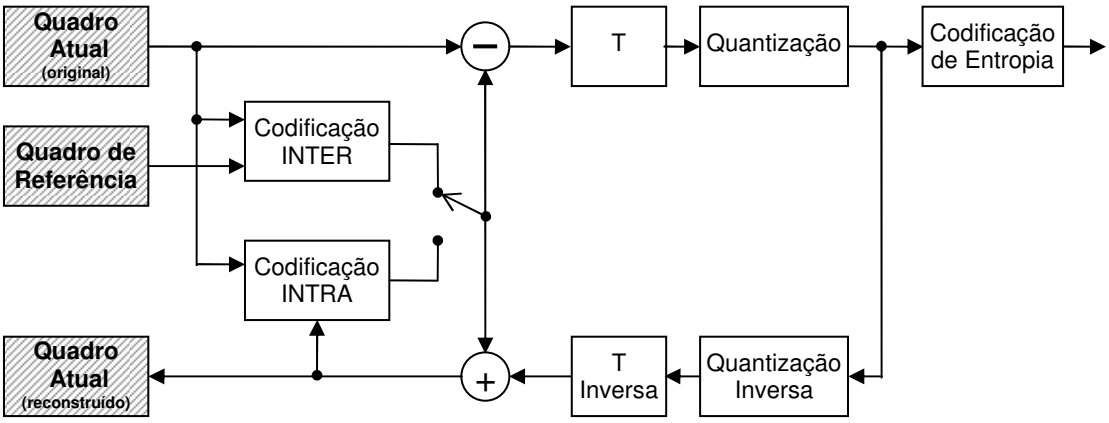



Figura 3.5: Modelo genérico de codificador de vídeo

A ME está situada dentro da Codificação Inter. Este bloco é responsável por reduzir a redundância temporal entre os quadros vizinhos de uma cena. Os vetores de movimento gerados pela ME são utilizados pela etapa de compensação de movimento (MC), que também está presente dentro do bloco de Codificação Inter. A MC é responsável por remontar os quadros a partir dos vetores de movimento gerados pela ME.

O bloco de Codificação Intraquadro é responsável por reduzir a redundância espacial, utilizando, para tanto, apenas a informação do quadro atual em processamento. Vários algoritmos podem ser utilizados para este fim, notadamente, os algoritmos utilizados em compressão de imagens estáticas podem ser utilizados na Codificação Intraquadro.

Apenas uma das codificações, Inter ou Intra, é aplicada a cada quadro do vídeo. Para isso, existe uma chave seletora que define qual das codificações deve ser utilizada. A diferença residual após a codificação intraquadro ou interquadro é obtida através de uma subtração dos valores do quadro atual e do quadro resultante destas codificações. Esta diferença é chamada de resíduo.

O resíduo, então, é enviado para os blocos responsáveis por reduzir a redundância psicovisual. A primeira operação nesta direção é a transformada (bloco T na Figura 3.5). O bloco T transforma a informação do domínio espacial para o domínio das freqüências. Neste domínio, a Quantização pode ser aplicada de maneira mais eficaz, eliminando as freqüências menos relevantes ao olho humano, reduzindo, assim, a redundância psicovisual. Por fim, a Codificação de Entropia reduz a redundância entrópica, que está relacionada à forma como os dados são codificados.

A informação de quadro atual reconstruído, presente na Figura 3.5, é gerada através das etapas de transformadas e de quantização inversa. Desta forma o codificador utiliza o quadro atual reconstruído como quadro de referência, para a codificação Intra, na

codificação do próximo quadro. Esta técnica é utilizada para evitar erros entre os processos de compressão e descompressão.

## 3.2 Algoritmos Investigados

Os algoritmos de busca para a estimação de movimento determinam a forma como o melhor casamento (*best matching*), para o bloco do quadro atual, será buscado dentro da área de pesquisa do quadro de referência. O algoritmo de busca tem influência direta na complexidade computacional da estimação, bem como na qualidade dos vetores gerados. Podem-se dividir os algoritmos de estimação em dois grandes grupos: algoritmos ótimos e sub-ótimos. Algoritmos ótimos analisam todas as posições dentro da área de pesquisa para gerar o vetor que represente a menor diferença entre as regiões pesquisadas. Algoritmos sub-ótimos utilizam determinados métodos de busca que eliminam alguns cálculos, diminuindo a complexidade e gerando vetores que podem não ser os que resultam no menor erro.

Neste trabalho vamos abordar diversos algoritmos de estimação como: busca completa (*Full Search* - FS) (BHASKARAN, 1999) e (LIN, 2005), *Three Step Search* (TSS) (JING, 2004), *Diamond Search* (DS) (KUHN, 1999) e (YI, 2005), *One at a Time Search* (OTS) (RICHARDSON, 2002), *Hexagon Based Search* (HS) (ZHU, 2002) e *Dual Cross Search* (DCS) (BANH, 2004). Para cada algoritmo também foram geradas versões com sub-amostragem de *pixel*, mais conhecido como *Pel Subsampling* (PS) (KUHN, 1999). Para o algoritmo FS foram geradas diversas versões combinando a sub-amostragem de *pixel* com sub-amostragem de blocos, chamada de *Block Subsampling* (BS) (KORAH, 2005).

De todos os algoritmos estudados, apenas o algoritmo *Full Search* pode ser considerado um algoritmo ótimo, pois ele avalia todas as posições possíveis da área de pesquisa para determinar o melhor resultado. Os demais algoritmos apresentados se enquadram na classe de algoritmos sub-ótimos, que utilizam determinadas heurísticas para reduzir o número de cálculos na obtenção do melhor casamento entre os blocos. Nesta seção serão detalhados os 15 algoritmos estudados, considerando as combinações de sub-amostragem de *pixel* e de blocos utilizadas. Também será mostrado o critério de similaridade SAD, que será usado na etapa de avaliação em software.

### 3.2.1 *Full Search* (FS)

O algoritmo FS procura a melhor relação entre as áreas de pesquisa comparando o bloco do quadro atual com todas as posições dentro da área de pesquisa do quadro de referência (LIN, 2005). O bloco é deslocado de um *pixel* dentro da área de pesquisa, começando do canto superior esquerdo, até o canto inferior direito, quando todos os blocos candidatos da área de pesquisa tenham sido comparados, e a menor diferença tenha sido registrada. Ao final, será gerado um vetor de movimento referente ao deslocamento do bloco para a região de melhor *matching*. A Figura 3.6 ilustra o processo de busca completa.

A Figura 3.6 ilustra uma região de pesquisa de +/- 7 *pixels* em torno do bloco corrente (supondo que a posição original do bloco seja a posição central da área de pesquisa). Cada ponto na figura representa um vetor de movimento, sendo que o vetor que resultar no melhor *matching* será o escolhido. O algoritmo aplica a função de similaridade para cada *pixel* dos blocos em comparação. Logo após, o algoritmo desloca

o bloco atual de um *pixel* dentro da área de pesquisa e re-calcula a diferença. Esta operação é repetida para todas as posições dentro da área de pesquisa.

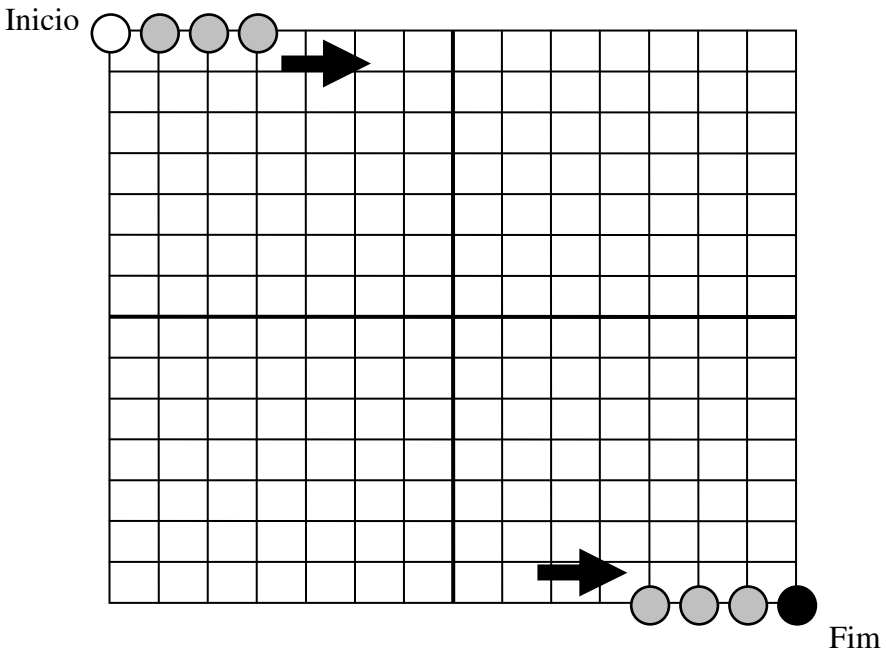



Figura 3.6: Busca Completa

Podemos perceber que o algoritmo FS possui uma grande complexidade computacional. Como o algoritmo calcula a diferença para todas as posições na área de pesquisa ele se torna diretamente dependente do tamanho da área de pesquisa. Para grandes áreas de pesquisa o desempenho do algoritmo cai consideravelmente. Com isso, torna-se praticamente impossível à utilização do algoritmo FS se a demanda for por aplicações de vídeo de alta resolução em tempo real. Aplicações em hardware podem acelerar o processo ao paralelizar as operações de cálculo do FS, possibilitando a utilização deste algoritmo quando o principal requisito seja a qualidade dos vetores gerados.

### 3.2.2 *Three Step Search* (TSS)

Este algoritmo é mais difundido com o nome de busca em três passos. No entanto, ele pode ser estendido para *n* passos (JING, 2004). O principal objetivo deste algoritmo é reduzir drasticamente o número de comparações, se comparado ao algoritmo FS. Para isso, um número finito de comparações é determinado e dividido nos três passos do algoritmo. A Figura 3.7 ilustra o algoritmo *Three Step Search* para uma área de pesquisa de +/- 7 *pixels*. Os pontos de melhor *matching* escolhidos a cada passo do algoritmo estão destacados em cinza.

Para uma área de pesquisa de +/- $(2^n - 1)$ *pixels*, o passo inicial S deve ser inicializado com $S = 2^{n-1}$. O primeiro passo consiste em posicionar a busca no centro da área de pesquisa e calcular o erro para esta posição. Em seguida, calcular mais oito valores +/- S *pixels* em torno da posição (0,0) (centro da área de pesquisa). Comparar os nove valores de erro obtidos e determinar como nova posição de origem a posição de menor erro.

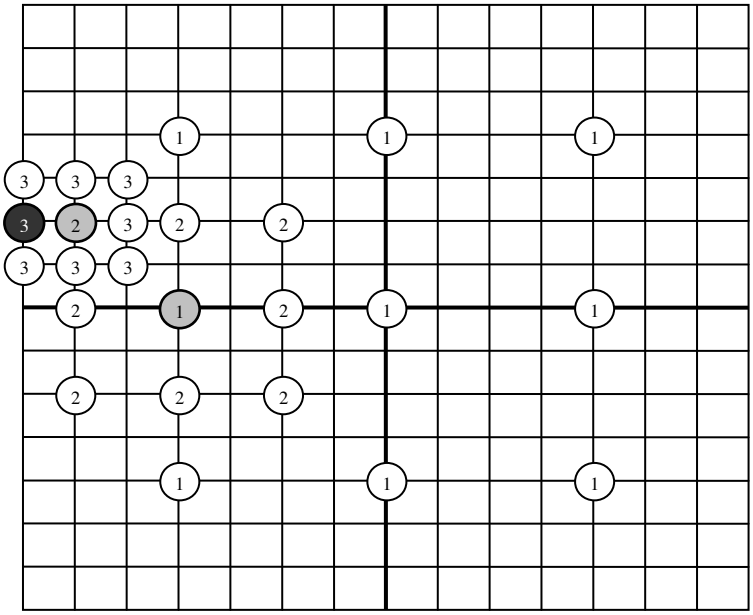



Figura 3.7: *Three Step Search*

No segundo passo a variável S é dividida por dois e novamente oito valores +/- S *pixel* em torno da origem são calculados. Desta vez, oito valores são comparados e, novamente, a origem será substituída pela posição com o menor erro.

No terceiro e último passo, mais uma vez a variável S é dividida por dois. Desta vez ela irá conter o valor um. Isto indica o último estágio de busca. Mais uma vez, os oito valores de erros são comparados e o vetor de movimento será gerado para a posição com o menor valor de erro.

Após os três passos do algoritmo, 25 posições são comparadas (9 + 8 + 8). Em geral 8n + 1 comparações são necessárias para uma busca em uma área de pesquisa de +/- (2n-1). Se utilizássemos o algoritmo de busca completa, para a mesma área de pesquisa teríamos um total de 225 (15 x 15) comparações.

### 3.2.3 One at a Time Search (OTS)

Este algoritmo é bastante simples e visa reduzir ainda mais o número de comparações. O algoritmo baseia-se na idéia de encontrar primeiro um mínimo horizontal e, em seguida, um mínimo vertical (RICHARDSON, 2002). O algoritmo começa calculando o erro para a posição central da área de pesquisa. Em seguida, mais dois valores, um imediatamente à direita e outro imediatamente à esquerda, são calculados. Caso o valor do centro seja o menor, o algoritmo passa para o estágio de busca vertical. Caso contrário, redefine-se o centro com a posição do menor erro e calcula-se o valor para o vizinho, seja ele à direita ou à esquerda (dependendo do valor escolhido no passo anterior). O estágio de cálculo dos valores verticais é muito semelhante, variando a busca para cima e para baixo. A Figura 3.8 ilustra uma busca de um algoritmo *One at a Time Search* em uma área de pesquisa de +/- 7 *pixels*.

O algoritmo começa realizando a busca horizontal, calculando três pontos. Para o exemplo da Figura 3.8, o melhor resultado foi obtido para o ponto à esquerda do centro. O algoritmo então desloca a busca um *pixel* à direita e calcula o novo valor. Este passo é repetido até o ponto número 5, onde o resultado do erro é maior que no ponto 4, então o

algoritmo encerra a busca horizontal e inicia a busca vertical. São calculados dois valores, um acima, e outro abaixo da posição de número 4.

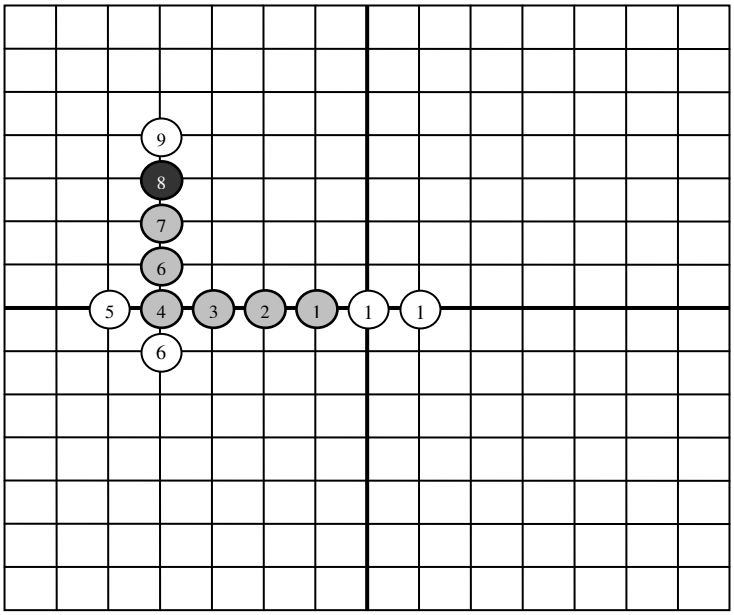



Figura 3.8: *One at* a *Time Search*

Na Figura 3.8, o menor erro é encontrado acima. O algoritmo segue deslocando um *pixel* acima até que a posição número 9 resulta em um erro maior que a posição 8. O algoritmo encerra a busca e gera o vetor para a posição 8.

Para este algoritmo, não é possível definir o número exato de operações, pois isso dependerá diretamente das informações contidas na área de pesquisa e no bloco. Estas informações determinarão o número de cálculos na horizontal e na vertical. No entanto, pode-se definir o número máximo de comparações que o algoritmo pode realizar. Este valor será igual a:

$$\text{Max} = 2*(P+2)$$

Onde $P$ é o tamanho da área de pesquisa.

Para o exemplo apresentado, o número máximo de comparações será igual a dezoito (2*(7+2)). No entanto, no exemplo ilustrado na Figura 3.8, apenas doze comparações são realizadas. No melhor caso, o algoritmo pode concluir a estimação em uma área de pesquisa calculando a diferença para apenas seis posições.

Pode-se perceber, pelo reduzido número de operações, que este algoritmo é bastante rápido, no entanto, está muito suscetível a encontrar mínimos locais, ou seja, determinar o vetor de movimento para regiões bem próximas ao início da pesquisa, podendo desviar o rumo da pesquisa numa direção contrária a região de menor erro.

### 3.2.4 Diamond Search (DS)

O algoritmo *Diamond Search* possui dois padrões diamante que são usados na etapa inicial e final do algoritmo. A Figura 3.9 ilustra os padrões *Large Diamond Search Pattern* (LDSP) e o padrão *Small Diamond Search Pattern* (SDSP). O padrão LDSP consiste em nove comparações e é utilizado na etapa inicial da pesquisa. Já o padrão SDSP consiste em quatro comparações e é utilizado na etapa final da pesquisa, com o intuito de refinar o resultado obtido na etapa anterior (KUHN, 1999).

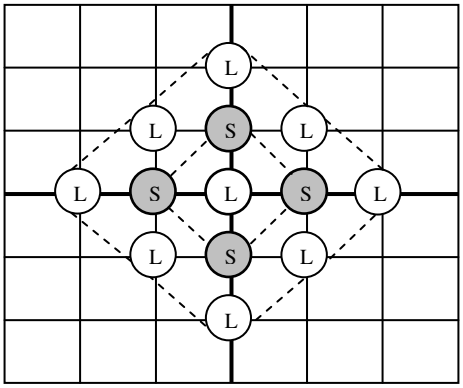



Figura 3.9: *Large Diamond* (LDSP) (L) e *Small Diamond* (SDSP) (S)

O algoritmo começa aplicando o padrão LDSP ao centro da área de pesquisa. Caso o valor de menor erro seja encontrado no centro, o algoritmo aplica o padrão SDSP para refinar o resultado obtido. Caso contrário, um novo LDSP é aplicado à posição de menor erro da etapa anterior. Esta posição pode pertencer a uma aresta ou a um vértice do diamante. As Figuras, 3.10 e 3.11 representam, respectivamente, a busca por uma aresta e a busca por um vértice.

No caso da busca por uma aresta, mais três valores são calculados para formar um novo diamante em torno da nova origem. Quando o novo centro é um vértice do diamante, mais cinco valores são calculados para formar o novo diamante em torno do centro. Caso o menor erro não seja encontrado no centro do novo diamante, a etapa de busca será repetida, seja ela por uma aresta ou por um vértice. Quando o menor erro for encontrado para o centro do diamante o padrão SDSP é aplicado. Então mais quatro valores imediatamente ao redor do centro serão calculados e a posição com o menor erro será a escolhida.

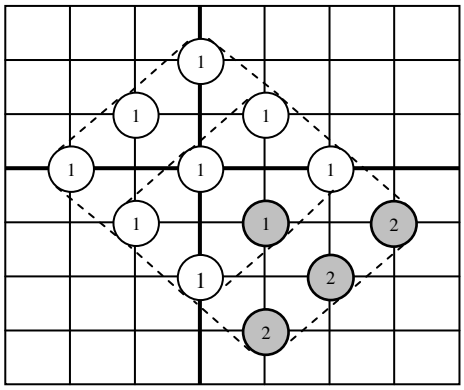



Figura 3.10: Busca por uma aresta

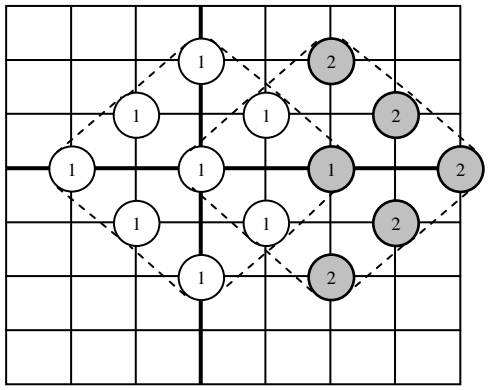



Figura 3.11: Busca por um vértice

Novamente não se pode determinar o número de operações do algoritmo. O algoritmo *Diamond Search* pode começar a sua busca em uma direção e desviar a pesquisa ao longo do processo, o que pode evitar que o algoritmo caia em um mínimo local.

### 3.2.5 Hexagon Search (HS)

O algoritmo *Hexagon Search* (ZHU, 2002) pode ser considerado uma evolução do algoritmo *Diamond Search*. Assim como no algoritmo *Diamond Search*, o *Hexagon* possui dois padrões hexágono, um para a etapa inicial *Large Hexagon Pattern*, com sete

cálculos, e outro para a etapa final, com 4 cálculos, *Small Hexagon Pattern*. Os dois padrões hexágono são ilustrados na Figura 3.12.

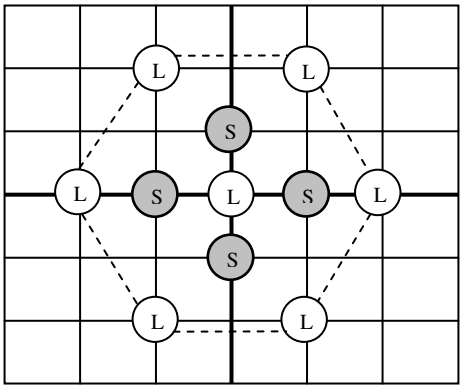



Figura 3.12: *Large Hexagon Pattern* (L) e *Small Hexagon Pattern* (S)

O algoritmo começa no centro da área de pesquisa aplicando o *Large Hexagon Pattern*. Caso o melhor *matching* seja encontrado no centro do hexágono, o algoritmo aplica o *Small Hexagon Pattern*. Caso o melhor *matching* seja encontrado em um dos vértices do hexágono, este vértice passa a ser o centro de outro hexágono, e mais três posições são calculadas. Este processo é repetido até que o melhor resultado seja obtido para o centro do hexágono. Então o *Small Hexagon Pattern* é aplicado a este centro e mais 4 pontos são avaliados. O melhor resultado dentre esses cinco pontos será o escolhido. A Figura 3.13 ilustra uma busca com o algoritmo *Hexagon*, onde o melhor resultado é obtido no terceiro passo do algoritmo.

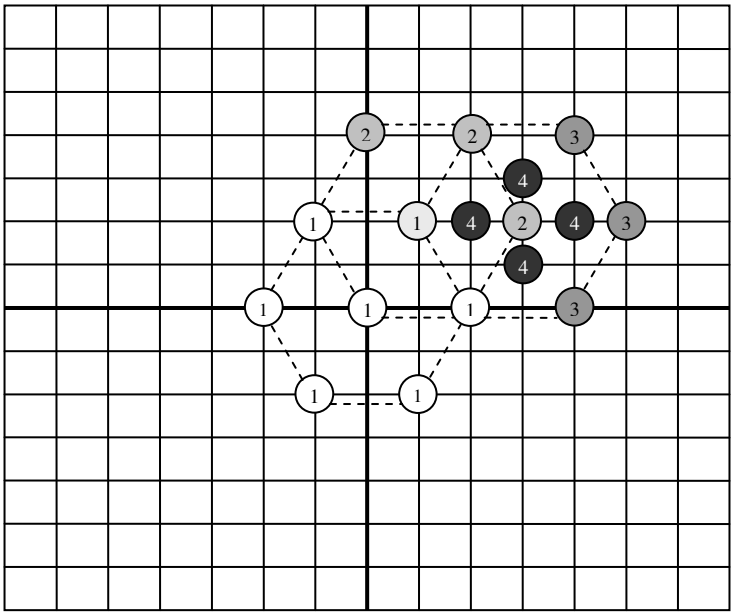



Figura 3.13: Exemplo de busca do algoritmo *Hexagon Search*

No exemplo da Figura 3.13, os passos do algoritmo estão numerados de um a quatro. No passo um, o padrão *Large Hexagon* é aplicado ao centro da área de pesquisa e sete pontos são calculados e comparados. No segundo e terceiro passos, mais três pontos são calculados até que o melhor resultado é encontrado no centro do hexágono. Então o padrão *Small Hexagon* é aplicado ao centro para refinar o resultado obtido no passo anterior. Nesta etapa, quarta etapa na Figura 3.13, mais quatro pontos são

calculados e comparados com o centro. O melhor resultado será obtido após a comparação destes cinco pontos.

Podemos perceber que o algoritmo *Hexagon* pode reduzir o número de comparações se comparado ao algoritmo *Diamond Search*. O *Hexagon* pode sair na primeira etapa, aplicando os padrões *Large* e Small *Hexagon*, com 11 comparações, contra 13 do algoritmo *Diamond*. A cada iteração do padrão *Large*, o algoritmo calcula três novos pontos, ao contrário do algoritmo *Diamond* que pode calcular três ou cinco pontos a cada iteração.

### 3.2.6 Dual Cross Search (DCS)

Este algoritmo visa reduzir ainda mais o número de comparações realizadas pelos algoritmos *Diamond* e *Hexagon Search*. Este algoritmo também possui dois padrões de pesquisa, o 2x2 *Cross Search Pattern* e o 4x4 *Cross Search Pattern* (BANH, 2004). O algoritmo começa aplicando o padrão 2x2 ao centro da área de pesquisa. São calculados o centro e mais quatro pontos, imediatamente ao redor, como ilustra a Figura 3.14. Os resultados são comparados e se o melhor resultado é encontrado no centro a busca acaba, caso contrário, o padrão 4x4 é aplicado, sendo seu centro o melhor resultado obtido no passo anterior. O padrão *Cross* 4x4 calcula mais três pontos a cada iteração e compara os novos valores com o centro do *Cross*. Este laço será repetido até que o melhor resultado seja encontrado no centro do *Cross*.

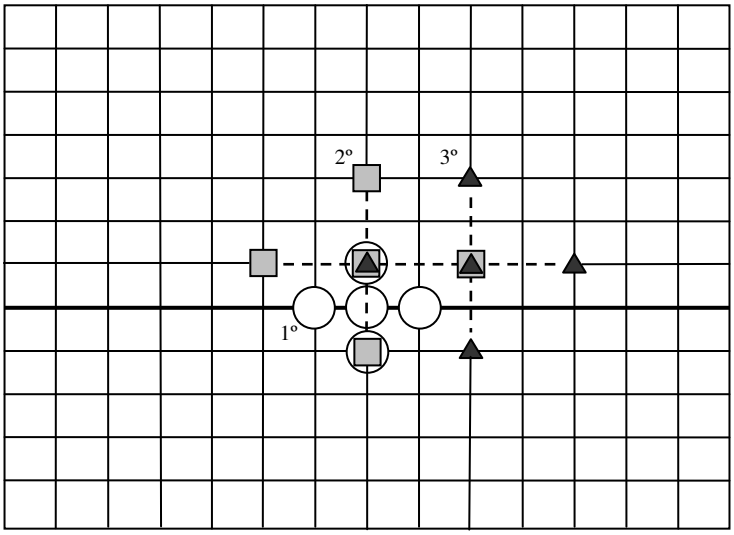



Figura 3.14: Exemplo de busca do algoritmo *Dual Cross Search*

A Figura 3.14 ilustra uma busca com o algoritmo DCS com três iterações. No primeiro passo o padrão *Cross* 2x2 é aplicado ao centro da área de pesquisa. No segundo e terceiros passos, o padrão *Cross* 4x4 é aplicado. A pesquisa termina quando o melhor resultado é obtido para o centro da terceira etapa.

Para todos os casos onde o melhor resultado não é encontrado no *Cross* 2x2, um refinamento final é aplicado ao resultado obtido no *Cross* 4x4. A Figura 3.15 ilustra as quatro possibilidades de refinamento final do algoritmo DCS.

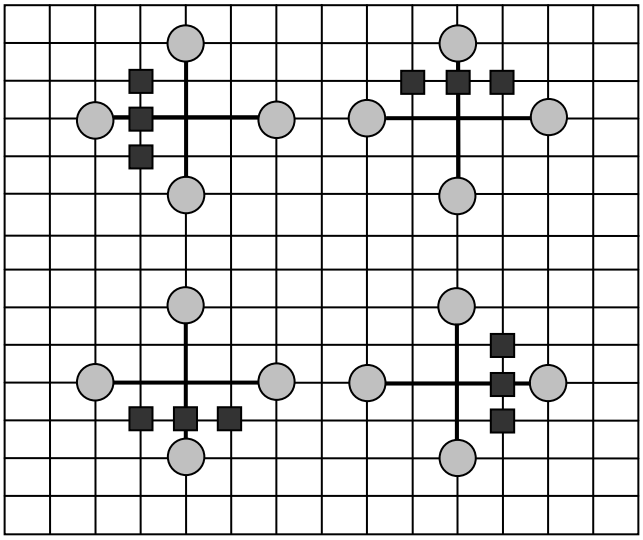


Figura 3.15: Refinamento final do algoritmo *Dual Cross Search*

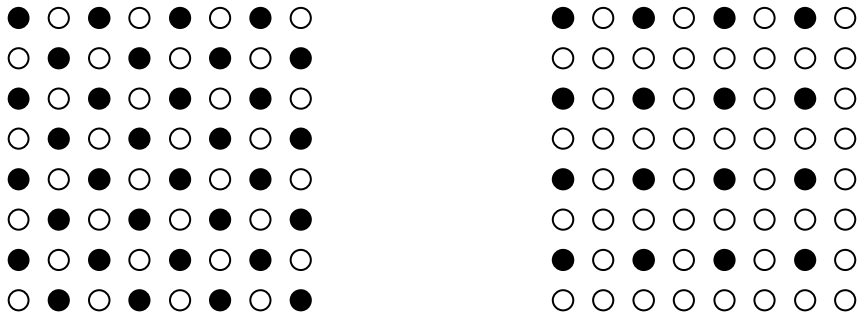
Depois de encontrar o melhor resultado com a aplicação do padrão *Cross* 4x4, o algoritmo tenta refinar o resultado encontrado calculando mais três pontos intermediários entre o centro e o ponto de melhor resultado do passo anterior. Este algoritmo pode reduzir o número de comparações se comparado ao algoritmo *Hexagon*, pois a busca pode ser interrompida na primeira etapa, com quatro comparações, ao invés das 11 comparações (mínimo) necessárias para concluir uma busca com o algoritmo *Hexagon*. O DCS também calcula três novas posições a cada iteração no *Cross* 4x4, assim como o algoritmo *Hexagon*, no entanto, a etapa de refinamento do DCS calcula apenas mais três pontos, contra quatro do *Hexagon*.

### 3.2.7 Pel Subsampling (PS)

A técnica de *Pel Subsampling* (também chamada de *Pel Decimation*) visa acelerar o processo de estimação através da sub-amostragem de *pixels* (KUHN, 1999). A cada bloco candidato comparado, o algoritmo calcula a diferença para alguns *pixels* do bloco e simplesmente descarta outros. Isto diminui o tempo de processamento e a complexidade computacional do algoritmo. A técnica de *Pel Subsampling* pode ser aplicada em qualquer proporção, como 2:1, 4:1, 8:1, por exemplo. A Figura 3.16 ilustra a técnica para as proporções 2:1(a) e 4:1(b) para um bloco de 8x8 *pixels*. Apenas as posições em preto são calculadas, as posições em branco são desconsideradas.

Utilizando PS pode-se reduzir significativamente o número de operações do algoritmo. Neste caso, para um bloco de 8x8 *pixels*, a cada comparação, o algoritmo deve realizar 64 operações (8x8). Para o exemplo da Figura 3.16(a) o algoritmo deve realizar 32 operações (4x8) e para o exemplo da Figura 3.16(b) o algoritmo deve realizar apenas 16 operações (4x4).

Esta técnica reduz a complexidade computacional e o tempo de operação e pode ser aplicada a qualquer algoritmo de busca. No entanto, os vetores resultantes podem não ser os vetores ótimos. Ao desconsiderar alguns elementos do bloco, o algoritmo pode conduzir a escolha de uma região que não gera o resultado ótimo. Os resultados tendem a piorar proporcionalmente ao aumento da sub-amostragem.



a) *Pel Subsampling* 2:1 b) *Pel Subsampling* 4:1

Figura 3.16: Técnica de *Pel Subsampling*

### 3.2.8 *Block Subsampling* (BS)

A técnica de *Block Subsampling* também visa à aceleração do processo de busca. Da mesma forma que a técnica de PS, realiza sub-amostragem de *pixel*, o *Block Subsampling* utiliza a sub-amostragem de blocos. Neste caso, não são comparados todos os blocos candidatos da área de pesquisa, os blocos são sub-amostrados, sendo que a diferença é calculada para alguns e outros são simplesmente descartados. Assim como o *Pel Subsampling*, o *Block Subsampling* pode ser aplicado com os mais diversos níveis de sub-amostragem, como 2:1, 4:1, 8:1 entre outros. As Figuras 3.17 e 3.18 ilustram duas áreas de pesquisa de 17x17 *pixel*. A primeira apresenta todos os blocos candidatos existentes, e a segunda, apresenta o resultado para uma sub-amostragem de bloco na ordem de 4:1.

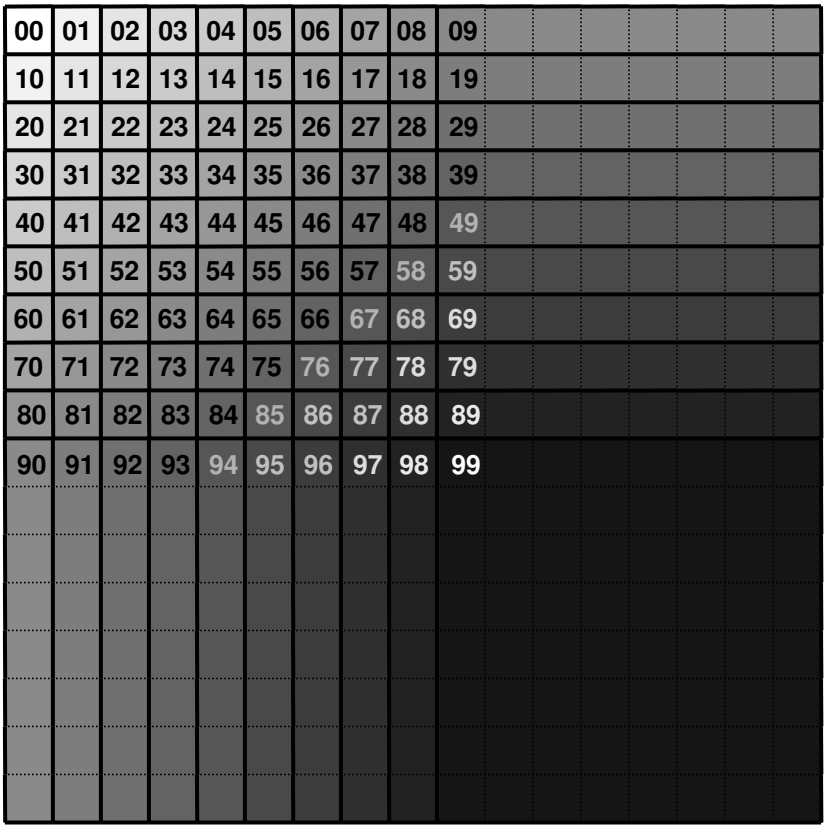



Figura 3.17: Área de pesquisa sem sub-amostragem de bloco

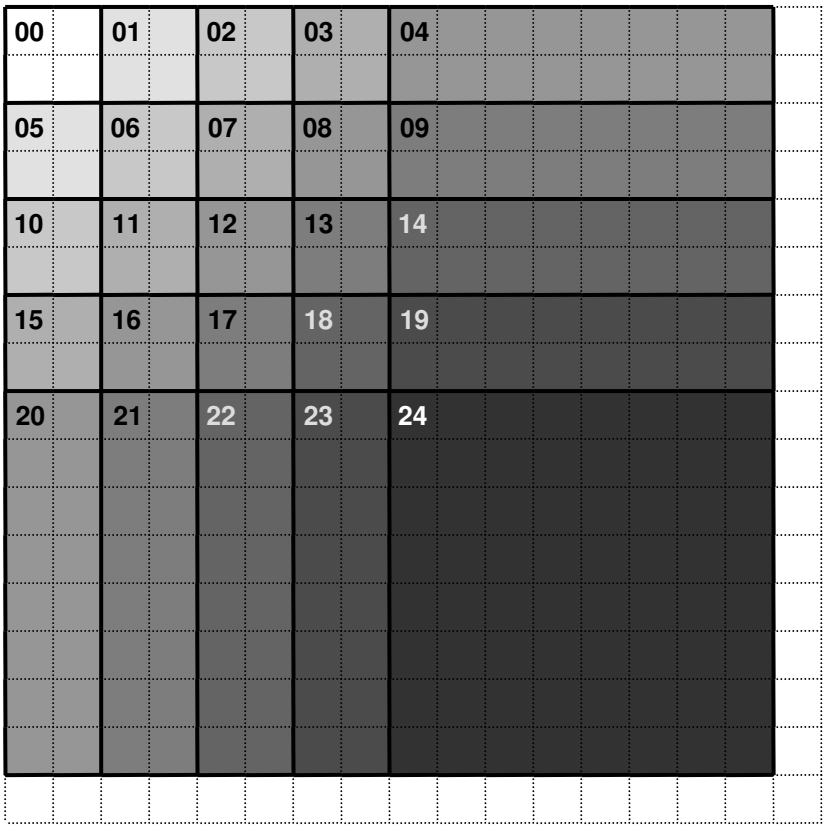



Figura 3.18: Área de pesquisa com sub-amostragem de bloco de 4:1

Para uma área de pesquisa de 17x17 *pixels*, considerando blocos de 8x8, temos 100 blocos candidatos. Considerando uma sub-amostragem de blocos de 4:1, o número de blocos candidatos é reduzido para 25, como pode ser visto na Figura 3.18. O algoritmo *Block Subsampling* também pode ser aplicado em conjunto com a técnica *Pel Subsampling*. Desta forma, aplica-se uma sub-amostragem ao nível de *pixel* aos blocos que não foram excluídos no processo de sub-amostragem dos blocos. Isto acelera ainda mais o processo, no entanto, aumenta a probabilidade do bloco escolhido não ser o resultado ótimo. A técnica BS só pode ser aplicada ao algoritmo FS, pois para os demais algoritmos não conseguimos determinar antecipadamente quais blocos serão avaliados. Os algoritmos rápidos já realizam algo parecido com a sub-amostragem de bloco devido as suas heurísticas de aceleração, que eliminam diversos blocos candidatos do processo de busca.

## 3.3 Critério de Similaridade

O critério de similaridade (critério de distorção) é a maneira como as diferenças entre as regiões comparadas são avaliadas. Os critérios podem considerar desde a simples diferença aritmética entre as regiões até cálculos que tentam fazer uma relação com o resultado visual da comparação (RICHARDSON, 2002). Dentre os critérios mais utilizados podemos citar o menor erro quadrático MSE (*Minimum Square Error*), o menor erro absoluto MAE (*Minimum Absolute Error*) (RICHARDSON, 2002) e a soma das diferenças absolutas SAD (*Sum of Absolute Differences*) (RICHARDSON, 2002).

O SAD é o critério de similaridade mais comumente utilizado para os algoritmos de estimação de movimento, devido a sua simplicidade. O SAD calcula a distorção entre as

regiões comparadas como sendo o somatório das diferenças absolutas, para cada ponto do bloco atual e do bloco candidato da área de pesquisa (KUHN, 1999). A função para o cálculo do SAD é dada por (1):

$$SAD_{x,y} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|R_{i,j} - P_{i+x,j+y}\right| \quad (1)$$

Onde:

- $SAD_{x,y}$ representa o SAD para a posição (x,y).
- *R* representa o ponto da área de referência.
- *P* representa o ponto da área de pesquisa.
- *N* é o tamanho do bloco.

O primeiro passo, para calcular o SAD para o bloco de posição (x,y), é calcular o módulo da diferença entre as amostras do bloco de referência e do bloco da área de pesquisa. Esse cálculo é repetido para todas as amostras do bloco de tamanho *N*x*N*, sendo que os resultados intermediários são acumulados, e o resultado final desta acumulação é o valor resultante de SAD para o bloco candidato da área de pesquisa.

O resultado do SAD é sempre positivo, pois a acumulação é feita através do somatório dos módulos das diferenças (ZANDONAI, 2002). Como são necessárias apenas subtrações e acumulações este critério é largamente utilizado em aplicações em hardware. Daqui para frente neste trabalho, toda vez que for mencionado critério de similaridade ou resultado de erro para qualquer algoritmo, estaremos sempre nos referindo ao resultado de SAD. O SAD será o critério adotado para todas as avaliações dos algoritmos apresentados neste trabalho.

# 4 AVALIAÇÃO DOS ALGORITMOS EM SOFTWARE

Nesta seção serão apresentados os resultados das avaliações em software realizadas com os algoritmos investigados. Diversas análises foram desenvolvidas visando classificar os algoritmos. As avaliações foram realizadas de maneira totalmente imparcial, ou seja, sem a intenção de favorecer nenhum algoritmo e sim de identificar as qualidades e defeitos de cada algoritmo. Alguns trabalhos foram encontrados na literatura como (LIU, 2004) e (BANH, 2005) que apresentam avaliações em software para algoritmos de ME. No entanto, as avaliações apresentadas nestes trabalhos não são muito abrangentes, e os autores estão sempre comparando os algoritmos da literatura com os algoritmos que estão propondo. Desta forma, os resultados apresentados não imparciais, pois os autores tendem a mostrar os resultados que mais favorecem aos algoritmos que estão propondo. Outro fator importante é a quantidade de algoritmos e de amostras de vídeo utilizadas nas comparações, que em geral são poucas nos trabalhos encontrados na literatura. Além disso, os resultados apresentados nestes trabalhos apenas consideram amostras de vídeo de baixa resolução, como QCIF e CIF, por exemplo.

As avaliações em software desenvolvidas neste trabalho utilizam várias métricas de comparação, e avalia diversos algoritmos com diversas combinações de sub-amostragem. Todos os algoritmos foram descritos em linguagem C, e rodando em uma plataforma x86 com processador Intel Pentium 4, 3.4 GHz com 1GB de memória RAM. Para a avaliação dos algoritmos foram utilizados os 100 primeiros quadros de 10 amostras de vídeo SDTV 720x480 não comprimidas (VQEG, 2006). As amostras são bastante variadas, contendo desde vídeos com pouco movimento de objetos e também de câmera, até vídeos com muito movimento de objetos e também de câmera. Com a aplicação dos algoritmos de estimação para 10 amostras de vídeos, com variados níveis de movimentação, podemos avaliar o desempenho médio dos algoritmos em aplicações reais. Para todos os algoritmos o tamanho do bloco foi de 16x16 pixels e o critério de similaridade utilizado foi o SAD. Alguns dos resultados apresentados nesta seção foram publicados em (PORTO, 2007).

A Figura 4.1 ilustra os primeiros quadros de cada um dos 10 vídeos utilizados nas análises. Os nomes apresentados para cada amostra foram sugestões nossas simplesmente para facilitar as considerações posteriores sobre os resultados obtidos para cada vídeo.

As avaliações apresentadas nesta seção vão desde a busca por uma área de pesquisa ótima para a estimação, até uma análise aprofundada de qualidade e custo computacional dos algoritmos para várias áreas de busca. Além destes testes, foram desenvolvidos outros métodos de análise para os algoritmos rápidos que não são baseados no algoritmo *Full Search*. Estes algoritmos não possuem um número finito de

operações para determinar o vetor de movimento, por isso, foram desenvolvidas análises que tentam estimar o número de iterações médio para cada algoritmo. Para facilitar as análises, vamos considerar a partir daqui, algoritmos rápidos como sendo apenas os algoritmos que não são baseados no algoritmo FS, ou seja, algoritmos rápidos serão os que utilizam alguma heurística para a redução do número de comparações, e não sub-amostragem, como os algoritmos rápidos baseados em FS.

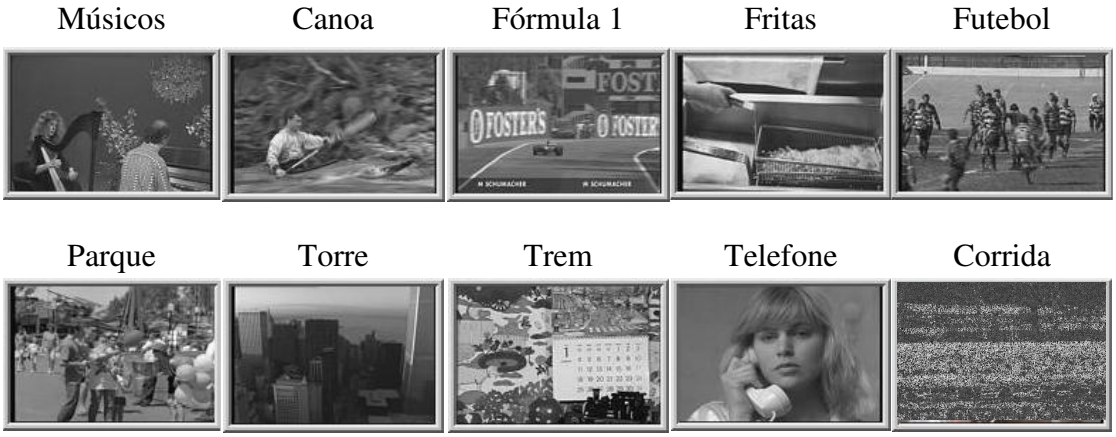



Figura 4.1: Primeiro quadro das amostras de vídeo utilizadas

Nesta seção serão apresentados os resultados para os 15 algoritmos desenvolvidos neste trabalho. Os algoritmos *Full Search* (FS), *Diamond Search* (DS), *Three Step Search* (TSS), *One at a Time Search* (OT), *Hexagon Search* (HS) e *Dual Cross Search* (DCS), foram desenvolvidos em versões com e sem sub-amostragem. A sub-amostragem de *pixel* (PS - *Pel Subsampling*) foi aplicada a todos os algoritmos na proporção de 2:1, e na proporção de 4:1 apenas para o algoritmo FS. Para o algoritmo FS também foram geradas versões com sub-amostragem de bloco (BS – *Block Subsampling*) de 2:1 e 4:1, aliada a sub-amostragem de pixel também de 2:1 e 4:1.

## 4.1 Análises das Áreas de Busca

Diversos fatores influenciam na qualidade dos vetores gerados no processo de estimação. Dentre eles, os principais são o algoritmo de busca e o tamanho da área de pesquisa. Para avaliar o impacto do aumento na área de pesquisa na qualidade dos vetores gerados, realizamos alguns experimentos utilizando o algoritmo FS. Primeiramente determinamos que a área de pesquisa considerada para a estimação será o quadro inteiro. Desta forma todos os vetores encontrados serão os melhores vetores para esse quadro. Tendo em vista que o algoritmo FS sempre gera vetores ótimos, considerando uma dada área de pesquisa. Ao considerarmos o quadro inteiro como área de pesquisa o algoritmo FS irá encontrar, seja onde for dentro do quadro, o melhor vetor possível. Realizando a busca em todo o quadro é possível estimar onde o algoritmo costuma achar os vetores, e desta forma, definir uma área de pesquisa ótima para o processo de estimação.

Para definir onde existem as maiores ocorrências de vetores ótimos, os vetores foram divididos em 11 faixas de tamanho de suas componentes. As 10 primeiras com intervalo de 32, e a última, para todos os vetores com componentes maiores que 320. A estimação foi realizada para os 100 primeiros quadros de cinco das 10 amostras apresentadas. Nesta análise não foram utilizadas as 10 amostras de vídeo, pois este

processo é computacionalmente muito custoso, sendo que para cada amostra foi necessário cerca de 25 horas de processamento. Dentre as amostras avaliadas, quatro possuem uma grande quantidade de movimentos, o que implica em uma maior probabilidade de vetores grandes, já a última amostra é de um vídeo com pouco movimento, onde a tendência é de que os vetores sejam menores. As cinco amostras foram criteriosamente escolhidas para que um resultado médio fosse obtido, sendo que quatro amostras possuem grande quantidade de movimentação e uma apresenta um vídeo com pouca movimentação. A Figura 4.2 ilustra um gráfico com a curva percentual de tamanho de vetores para cada uma das amostras de vídeo avaliadas.

Analisando as curvas do gráfico da Figura 4.2 percebe-se que a maior parte dos vetores ótimos encontra-se nas primeiras faixas. As curvas mostram que ao analisar os quatro vídeos com maior movimento, "Canoa", "F1", "Fritas" e "Futebol", existe uma oscilação de 60% a 80% dos vetores ótimos na faixa de tamanho de componente do vetor de até 32, já para o vídeo "Telefone", que possui menor quantidade de movimento, cerca de 95% dos vetores ótimos estão dentro desta faixa. As curvas ainda apresentam um crescimento considerável, para todas as amostras, até a faixa de tamanho de 96. Após, todas as curvas começam a saturar, apresentando um crescimento mínimo no percentual de vetores, sendo que para a amostra "Telefone", o crescimento é de menos de 1% a partir da faixa 96. Na média, mais de 85% dos vetores ótimos podem ser encontrados em uma área de pesquisa com componentes vetores de tamanho até 96, para a média obtida com as cinco amostras avaliadas. Considerando vetores com componentes de tamanho até 96, temos uma área de pesquisa de +-96 *pixels* em torno do bloco. Como estamos utilizando blocos de tamanho 16x16, temos uma área de 96+16+96, o que representa uma área de pesquisa de 208x208 *pixels*. Esta forma de calcular a área de pesquisa serve para todas as demais faixas.

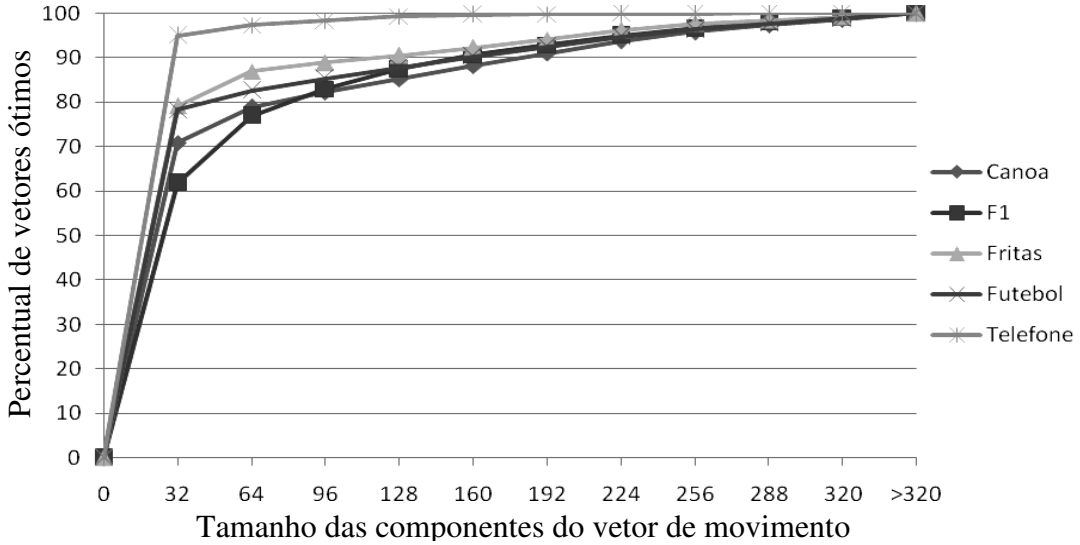



Figura 4.2: Percentual de vetores ótimos por faixa de tamanho das componentes do vetor

Avaliar o percentual de vetores ótimos a cada faixa de tamanho de vetores é um dado interessante para a escolha de uma determinada área de pesquisa, no entanto, é possível também avaliar quanto o aumento do tamanho da área de pesquisa interfere na qualidade dos vetores gerados no processo de estimação. Para analisar a qualidade do processo de estimação, será utilizado como parâmetro o PSNR (*Peak-to-Sinal Noise*

*Ratio*) (Richardson 2003). O PSNR é considerado o parâmetro de avaliação mais utilizado para comparar a qualidade de vídeos comprimidos e descomprimidos e pode ser calculado utilizando a fórmula apresentada em (2). O valor de MAX representa o maior valor possível para o *pixel*. Para todos os testes desenvolvidos neste trabalho o valor de MAX será de 255, pois todas as amostras são de oito bits. O PSNR é medido em uma escala logarítmica, utilizando como base o critério MSE (*Mean Squared Error*), entre o quadro original e o estimado, relativo ao valor máximo de representação do *pixel*. A fórmula apresentada em (3) apresenta o cálculo do MSE para um bloco.

$$PSNR = 20 * log\left(\frac{MAX}{\sqrt{MSE}}\right) \tag{2}$$

$$MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(R_{(i,j)} - P_{(i,j)}\right)^2 \tag{3}$$

A qualidade pode ser diretamente ligada ao valor do PSNR obtido, valores altos indicam alta qualidade, valores baixos de PSNR podem indicar baixa qualidade. No entanto, importante salientar que o PSNR possui uma série de limitações como critério de qualidade, sendo que em alguns casos, uma imagem que possui uma qualidade visual superior pode ter um valor de PSNR inferior à outra com um resultado visual pior (RICHARDSON, 2003). Isto ocorre porque o PSNR considera apenas o valor do MSE para cada *pixel* da imagem, e desta forma, não necessariamente avalia os critérios subjetivos de qualidade (RICHARDSON, 2003). A Figura 4.3 ilustra o gráfico com as curvas de ganho PSNR para cada um dos vídeos avaliados.

As curvas mostram que existe uma variação muito pequena no ganho PSNR para todas as amostras de vídeo utilizadas, sendo que para a amostra "Telefone", que possui pouca movimentação, o crescimento no ganho é imperceptível. As demais amostras apresentam um leve crescimento no ganho entre as faixas de 32, 64 e 96. A partir da faixa de 96, o crescimento fica muito pequeno e passa a ser praticamente imperceptível nas curvas do gráfico.

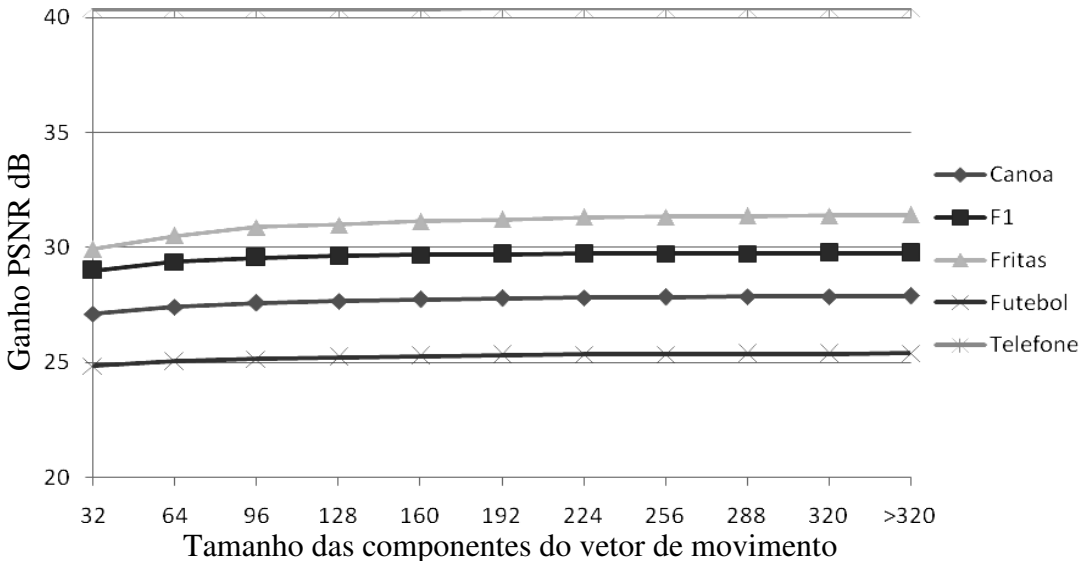



Figura 4.3: Curvas de ganho PSNR para todas as amostras avaliadas

Estes resultados mostram que o ganho do processo de estimação praticamente não é afetado pelas perdas em relação à escolha de vetores não ótimos. Isto significa que o vetor encontrado em uma área de pesquisa menor, apesar de não ser o ótimo, gera um resíduo muito pequeno, bem próximo ao gerado pelo vetor ótimo, fazendo com que o ganho seja praticamente o mesmo. Isto pode ser observado na Figura 4.4, que ilustra as curvas de erro para cada uma das amostras de vídeo avaliadas.

O valor de erro é a soma de todos os SADs resultantes do processo de estimação, e quanto menor o valor do erro, melhor será considerado o processo de estimação. Os valores apresentados no gráfico da Figura 4.4 para o erro estão na escala de $10^6$. As curvas de erro seguem a mesma tendência das curvas de ganho PSNR, apresentadas no gráfico da Figura 4.3, no entanto, o valor do erro absoluto tende a diminuir com o aumento da área de pesquisa. Pode-se perceber que existe uma diminuição significativa no erro entre as faixas de 32, 64 e 96. A partir da faixa de 96, a diminuição do erro passa a ser muito pequena, e sua variação em relação ao aumento da faixa de vetores passa a ser imperceptível.

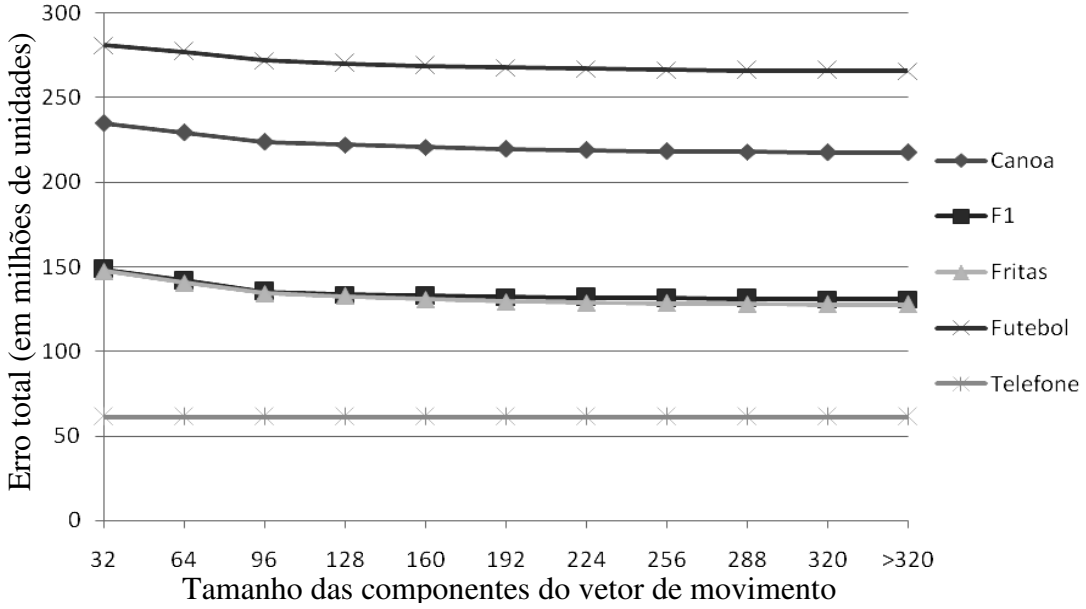



Figura 4.4: Curvas de diminuição do erro para todas as amostras avaliadas

Considerando o método para calcular a área de pesquisa, apresentado anteriormente, pode-se calcular o número de blocos candidatos e conseqüentemente o número de cálculos de SAD necessários para cada uma das áreas de pesquisa. Tomando como exemplo a área de pesquisa de 80x80, que abrange os vetores com componentes de tamanho até 32, temos 80-16+1=65 blocos candidatos por linha, em um total de 65 linhas, totalizando 4225 blocos candidatos por área de pesquisa. Cada bloco possui 16x16 amostras, o que resulta em 256 cálculos de SAD por bloco avaliado. Desta forma, 1081600 cálculos de SAD devem ser realizados para avaliar toda a área de pesquisa de 80x80 *pixels*. Considerando vídeos SDTV de 740x480 *pixels*, com blocos de 16x16, temos 1350 blocos por quadro. Como uma área de pesquisa deve ser criada para cada bloco, temos que 1350x1081600 = 1460160000 cálculos de SAD são necessários para estimar um quadro com uma área de pesquisa de 80x80 *pixels*.

A Figura 4.5 apresenta as curvas de crescimento do número de operações e diminuição do erro, para cada uma das faixas de tamanho. O número de operações está

representado em bilhões de operações. Os valores da curva de erro representam a média dos valores de erro obtidos para as cinco amostras avaliadas. Os resultados foram multiplicados por 100 para que pudessem ser observados mais claramente no gráfico. O crescimento da curva de operações é mais acentuado até a faixa de 224. Isto ocorre, pois para as faixas de vetores maiores que 224, não é possível aumentar as duas dimensões da área de pesquisa na mesma proporção. Para a faixa de vetores com componentes de 256, por exemplo, a área de pesquisa resultante é de 528x528, no entanto, este tamanho extrapola o tamanho vertical do quadro, que é de 480. Desta forma a área de pesquisa efetiva para a faixa de 256 é de 528x480 *pixels*. Esta restrição se repete para as faixas de 288, 320 e também para a faixa >320, para a qual a área de pesquisa é o quadro inteiro.

A curva do erro médio da Figura 4.5 mostra a mesma tendência das curvas de erro apresentadas na Figura 4.4, sendo que o erro cai entre as faixas de 32, 64 e 96 e a partir disso a curva fica praticamente estável. O mais interessante na Figura 4.5 é avaliar o número de operações necessárias, que cresce muito para cada faixa, em contraste a diminuição do erro, que é praticamente imperceptível. O número de operações passa de cerca de 12,8 Goperações, para a faixa de 96, para cerca de 116,5 Goperações para a faixa >320. Este aumento de aproximadamente nove vezes no número de operações resulta em um ganho em diminuição de erro de apenas 2,9%. Estes resultados mostram que não é uma estratégia interessante ampliar a área de pesquisa para tamanhos maiores que 208x208 para vídeos SDTV de 740x480, pois a relação entre diminuição do erro e número de operações necessárias passa a ser muito ruim.

Devemos ter em mente que os resultados apresentados nesta seção sevem como base para determinar um tamanho de área de pesquisa ótima para o processo de estimação de movimento. O algoritmo utilizado foi o FS, que sempre encontra o vetor ótimo em uma dada área de pesquisa, desta forma, o seu resultado de erro absoluto sempre diminuirá com o aumento da área de busca, mesmo que com ganhos praticamente insignificantes. Estes resultados de erro absoluto gerado e ganho PSNR obtido podem ter comportamentos diferentes quando considerarmos algoritmos rápidos na estimação.

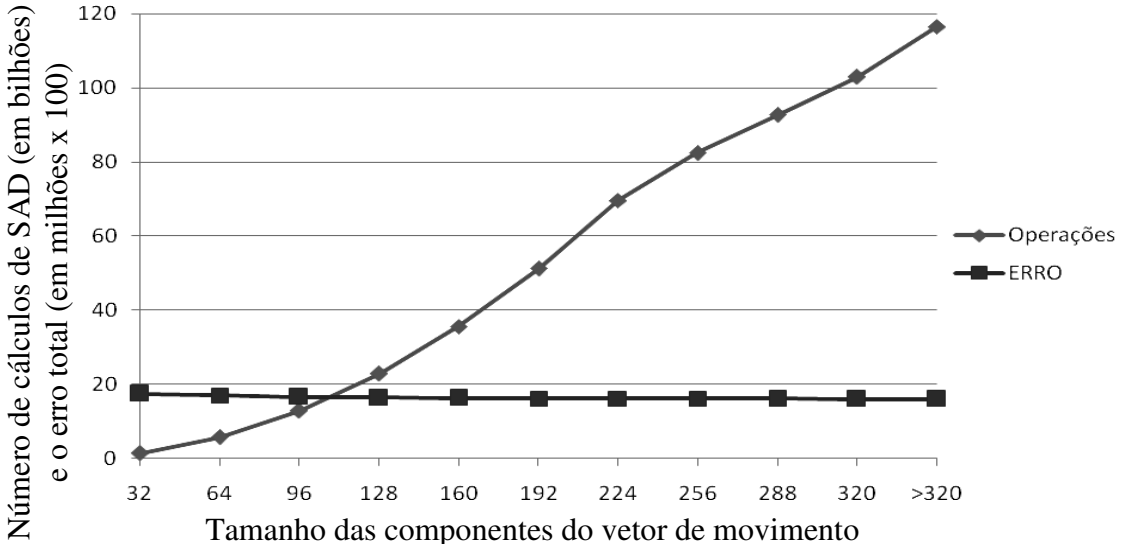



Figura 4.5: Número de operações versus diminuição do erro absoluto

Nas próximas seções serão apresentados os resultados obtidos em termos de qualidade e custo computacional para as quatro áreas de pesquisa escolhidas 46x46,

80x80, 144x144 e 208x208 *pixels*. Estas áreas de pesquisa cobrem as faixas de vetores com componentes de tamanhos 15, 32, 64 e 96 respectivamente. A área de 46x46 não aparece nas avaliações de tamanho de áreas de pesquisa apresentadas acima, no entanto, devido aos bons resultados apresentados por pequenas áreas de pesquisa, resolvemos aprofundar as avaliações para uma área menor. Os resultados apresentados são referentes às médias obtidas para cada algoritmo com a simulação para os 10 vídeos apresentados anteriormente.

## 4.2 Resultados de Qualidade

Para avaliar a qualidade dos vetores gerados por cada algoritmo de estimação investigado serão utilizados dois critérios, a diminuição do erro em relação ao erro absoluto e o ganho PSNR. Nesta seção serão apresentados os resultados médios obtidos para cada algoritmo com as 10 amostras de vídeo utilizadas.

### 4.2.1 Diminuição do erro absoluto

O primeiro critério que analisaremos para avaliar a qualidade do processo de estimação será a diminuição do erro em relação ao erro absoluto. O erro absoluto é o somatório das diferenças *pixel* a *pixel* entre os quadros visinhos. A Tabela 4.1 apresenta os resultados médios percentuais de diminuição do erro para todos os algoritmos estudados nas quatro áreas de pesquisa avaliadas.

Tabela 4.1: Diminuição percentual do erro absoluto

| Algoritmos | Áreas de Pesquisa | | | | Diferença FS (%) |
|---|---|---|---|---|---|
| | 46x46 | 80x80 | 144x144 | 208x208 | |
| FS | 51,80 | 55,70 | 56,69 | 57,04 | - |
| FS + PS 2:1 | 51,52 | 55,40 | 56,38 | 56,73 | -0,30 |
| FS + PS 4:1 | 47,40 | 51,17 | 52,00 | 52,26 | -4,60 |
| FS + BS 2:1 + PS 2:1 | 41,91 | 48,86 | 52,15 | 52,51 | -6,45 |
| DS | 47,15 | 48,97 | 49,15 | 49,17 | -6,70 |
| DS + PS 2:1 | 46,74 | 48,52 | 48,69 | 48,70 | -7,14 |
| HEX | 45,72 | 47,38 | 47,55 | 47,57 | -8,25 |
| HS + PS 2:1 | 45,32 | 46,95 | 47,10 | 47,12 | -8,69 |
| OTS | 42,87 | 44,50 | 44,62 | 44,64 | -11,15 |
| OTS + PS 2:1 | 41,84 | 43,33 | 43,42 | 43,43 | -12,30 |
| DCS | 41,82 | 43,19 | 43,31 | 43,33 | -12,40 |
| FS + BS 4:1 + PS 4:1 | 27,06 | 45,67 | 46,51 | 46,81 | -13,80 |
| TSS + PS 2:1 | 40,12 | 40,54 | 33,04 | 40,13 | -16,85 |
| TSS | 37,00 | 38,88 | 33,48 | 31,33 | -20,14 |
| DCS + PS 2:1 | 30,19 | 30,39 | 30,51 | 30,52 | -24,91 |

Os resultados mostram que o algoritmo FS possui uma redução do erro absoluto sempre superior a 51% para todas as áreas de pesquisa. Sendo que para a área de 208x208 a diminuição do erro chega aos 57,4%. Pode-se perceber também que a diminuição do erro entre as áreas de 46x46 e 80x80 chega a quase 4%, entre as demais áreas esta diferença é de menos de 1%. O algoritmo FS com PS 2:1 obteve a segunda colocação em termos de redução de erro, obtendo uma redução do erro apenas 0,3% inferior comparado ao resultado do FS. A versão do algoritmo FS com PS 4:1 também apresenta uma boa diminuição do erro, no entanto, seus resultados ficam em média 4,6% abaixo do algoritmo FS para as quatro áreas de pesquisa. O algoritmo FS com BS 2:1 e PS 2:1 apresenta sub-amostragem de bloco e de *pixel* de 2:1, isto representa o mesmo nível de sub-amostragem adotado pela versão FS com PS 4:1, só que implementada de maneira diferente. Isto faz com que seus resultados sejam um pouco inferiores aos do FS com PS 4:1. A diferença é maior para as áreas menores, e para as maiores áreas os resultados são praticamente iguais. Seu resultado de erro é um pouco inferior aos melhores algoritmos rápidos para a área de 46x46. Apenas a partir da área de 144x144 que os seus resultados são melhores que o do algoritmo DS, por exemplo. O algoritmo FS com BS 4:1 e PS 4:1 apresenta os piores resultados dentre os algoritmos derivados do FS. Seus resultados figuram entre os piores resultados dos algoritmos rápidos.

Dentre os algoritmos rápidos podemos destacar os resultados dos algoritmos DS e HS. O algoritmo DS obteve resultados de redução do erro sempre superiores a 47,15%. Entre as áreas de 46x46 e 80x80 o percentual sobe 1,82%, já entre as áreas de 80x80 e 208x208, a diminuição do erro aumenta apenas 0.2%. Esta característica se mantém para praticamente todos os algoritmos rápidos. Isto mostra que os algoritmos rápidos são muito suscetíveis a cair em mínimos locais, não aproveitando de maneira eficiente a expansão da área de pesquisa. Os algoritmos OTS e DCS obtiveram resultados intermediários, com reduções no erro variando de 41,82% a 44,64%. Os piores resultados foram obtidos pelos algoritmos TSS e a versão do algoritmo DCS com PS 2:1. Os algoritmos DS, HS e OTS sofreram um impacto muito pequeno com a aplicação da sub-amostragem de *pixel* em 2:1. Os algoritmos DS e HS com PS 2:1 apresentam resultados de redução do erro apenas 0,44% inferiores as suas versões sem sub-amostragem.

Analisando os dados da Tabela 4.1 podemos perceber que o algoritmo FS, e suas versões com sub-amostragem de *pixel* e de blocos, conseguem aproveitar melhor a expansão da área de pesquisa, buscando vetores ótimos em pontos mais distantes, e conseqüentemente melhorando seus resultados de erro. Já os algoritmos rápidos possuem uma melhora muito discreta em relação ao aumento da área de pesquisa. Em casos extremos, como o do algoritmo TSS, o resultado chega ser pior, devido às características do algoritmo. Os resultados da Tabela 4.1 mostram que a partir de uma área de pesquisa com 80x80 os ganhos em relação à diminuição do erro são mínimos, tanto para o algoritmo FS quanto para os algoritmos rápidos.

### 4.2.2 Ganho PSNR

O segundo critério para avaliar a qualidade dos vetores gerados por cada algoritmo de estimação será o ganho PSNR. Os resultados de ganho PSNR em dB, para cada um dos algoritmos, em todas as áreas de pesquisa avaliadas, estão apresentados na Tabela 4.2. Mais uma vez o algoritmo FS possui o melhor resultado. As versões do algoritmo FS com PS 2:1 e 4:1 obtiveram os melhores resultados dentre os demais algoritmos,

sendo que para o FS com PS 2:1 a diferença média do PSNR entre as quatro áreas de pesquisa foi de apenas -0,061dB. Em geral os algoritmos mantiveram a mesma tendência dos resultados apresentados na Tabela 4.1. A versão com PS 2:1 para os algoritmos rápidos geraram em média uma perda de apenas 0,1dB no ganho PSNR.

A única mudança significativa entre os dados das Tabelas 4.1 e 4.2 são com relação aos algoritmos que usam sub-amostragem de bloco. Ao contrário dos resultados de redução do erro absoluto, o algoritmo FS com BS e PS 2:1 obteve um resultado muito ruim. Este algoritmo foi melhor que todos os algoritmos rápidos nos dados apresentados na Tabela 4.1, no entanto, com relação ao ganho PSNR ele obteve o pior resultado, sendo melhor apenas do que a versão do algoritmo FS com BS e PS 2:1. Isto indica que a eliminação de blocos inteiros, durante a busca dos vetores, pode não influenciar muito na quantidade de erro gerado, no entanto, influenciar mais significativamente na qualidade final do vídeo estimado.

Tabela 4.2: Ganho PSNR dos algoritmos em dB

| **Algoritmos** | **Áreas de Pesquisa** | | | | **Diferença FS (dB)** |
|---|---|---|---|---|---|
| | **46x46** | **80x80** | **144x144** | **208x208** | |
| FS | 28,114 | 28,674 | 28,905 | 28,992 | - |
| FS + PS 2:1 | 28,012 | 28,625 | 28,858 | 28,945 | -0,061 |
| FS + PS 4:1 | 26,887 | 27,398 | 27,552 | 27,599 | -1,312 |
| DS | 27,093 | 27,264 | 27,283 | 27,285 | -1,440 |
| DS + PS 2:1 | 27,021 | 27,188 | 27,205 | 27,207 | -1,541 |
| HS | 26,813 | 26,964 | 26,981 | 26,982 | -1,736 |
| HS + PS 2:1 | 26,748 | 26,895 | 26,910 | 26,911 | -1,805 |
| OTS | 26,339 | 26,485 | 26,497 | 26,499 | -2,216 |
| OTS + PS 2:1 | 26,185 | 26,314 | 26,324 | 26,324 | -2,384 |
| DCS | 26,196 | 26,309 | 26,320 | 26,321 | -2,385 |
| DCS + PS 2:1 | 26,096 | 26,204 | 26,216 | 26,216 | -2,488 |
| TSS + PS 2:1 | 25,827 | 25,644 | 25,827 | 25,827 | -2,890 |
| TSS | 25,536 | 25,580 | 25,015 | 24,744 | -3,452 |
| FS + BS 2:1 + PS 2:1 | 23,094 | 24,772 | 24,998 | 25,082 | -4,185 |
| FS + BS 4:1 + PS 4:1 | 17,922 | 20,624 | 20,763 | 20,813 | -8,641 |

Dentre os algoritmos rápidos devemos destacar o bom resultado do algoritmo DS que obteve o melhor resultado dentre os algoritmos rápidos. Os algoritmos HS, OTS e DCS também obtiveram bons resultados. O pior desempenho ficou com o algoritmo TSS, que em nenhuma das áreas de busca superou os 25,8dB. Os resultados dos algoritmos rápidos para erro apresentados para os na Tabela 1 se refletem nos seus resultados de ganho PSNR na Tabela 2. Existe um crescimento no ganho entre as áreas de 46x46 e 80x80, no entanto entre as demais áreas o crescimento é praticamente nulo, sendo que entre as áreas de 144x144 e 208x208 o crescimento oscila entre 0,01dB e

0,02dB para os algoritmos rápidos. Esta característica vale para todos os algoritmos rápidos exceto o algoritmo TSS.

## 4.3 Custo Computacional

Para medir o custo computacional de cada algoritmo de estimação, vamos analisar quantos cálculos de SAD cada algoritmo utiliza para realizar o processo de estimação. A Tabela 4.3 apresenta os resultados médios de número de cálculos de SAD para todos os algoritmos em todas as áreas de pesquisa avaliadas. O algoritmo FS, e suas versões com sub-amostragem, possuem número fixo de cálculos de SAD, independentemente do vídeo utilizado. Os algoritmos rápidos possuem número variado de cálculos de SAD, portanto, na Tabela 4.3, são apresentados os resultados médios obtidos com a estimação para as 10 amostras de vídeo utilizadas. Os resultados apresentados na Tabela 4.3 estão representados em Goperações ($10^9$ operações de SAD).

Os resultados mostram que o algoritmo FS, como já era esperado, é o algoritmo com o maior custo computacional, ou seja, ele é o algoritmo que necessita do maior número de operações para realizar a estimação em todas as áreas de pesquisa. O FS necessita de cerca de 33,21 bilhões de operações para estimar 100 quadros com uma área de busca de 46x46, e cerca de 1287,32 bilhões de operações para os mesmos 100 quadros com uma área de pesquisa de 208x208. Isto representa um crescimento de aproximadamente 39 vezes no número de cálculos de SAD.

Tabela 4.3: Número de Operações de SAD dos algoritmos em Goperações

| **Algoritmos** | **Áreas de Pesquisa** | | | |
|---|---|---|---|---|
| | **46x46** | **80x80** | **144x144** | **208x208** |
| OTS + PS 2:1 | 0,16 | 0,17 | 0,18 | 0,18 |
| DCS + PS 2:1 | 0,25 | 0,26 | 0,26 | 0,26 |
| HS + PS 2:1 | 0,30 | 0,31 | 0,31 | 0,31 |
| OTS | 0,34 | 0,36 | 0,37 | 0,44 |
| DS + PS 2:1 | 0,41 | 0,43 | 0,43 | 0,43 |
| TSS + PS 2:1 | 0,47 | 0,47 | 0,47 | 0,47 |
| DCS | 0,50 | 0,52 | 0,52 | 0,52 |
| HS | 0,60 | 0,63 | 0,63 | 0,63 |
| DS | 0,82 | 0,86 | 0,87 | 0,87 |
| TSS | 0,93 | 0,93 | 0,93 | 0,93 |
| FS + BS 4:1 + PS 4:1 | 2,08 | 9,13 | 35,94 | 80,46 |
| FS + BS 2:1 + PS 2:1 | 8,30 | 36,50 | 143,78 | 321,83 |
| FS + PS 4:1 | 8,30 | 36,50 | 143,78 | 321,83 |
| FS + PS 2:1 | 16,61 | 73,01 | 287,66 | 643,66 |
| FS | 33,21 | 146,02 | 575,11 | 1.287,33 |

Os algoritmos baseados no FS, com sub-amostragem de pixel e de blocos, também possuem um elevado número de cálculos de SAD, diminuindo proporcionalmente ao aumento da sub-amostragem utilizada. No entanto, o número de operações cresce na mesma proporção do FS com o aumento da área de pesquisa.

Os resultados de número de cálculos de SAD devem ser contrastados com os resultados de qualidade, apresentados na seção anterior. Em geral, algoritmos que necessitam de um número maior de cálculos de SAD apresentam melhores resultados de qualidade, mas isto não é uma regra entre os algoritmos sub-ótimos (todos exceto o FS). Os algoritmos FS com PS 4:1 e FS com BS e PS 2:1 possuem o mesmo número de operações, no entanto, os resultados de qualidade do algoritmo PS 4:1 são melhores, tanto para a redução do erro absoluto quanto em relação ao ganho PSNR. Os algoritmos rápidos também não seguem esta regra, o algoritmo TSS possui o maior número de operações dentre os rápidos e ao mesmo tempo os piores resultados em termos de qualidade. Já os algoritmos DS, HS e OTS mantêm esta relação, sendo que o algoritmo DS possui os melhores resultados de qualidade e também o maior número de cálculos de SAD.

Alguns algoritmos merecem destaque especial, como por exemplo, as versões dos algoritmos DS e HS com PS 2:1. Estes algoritmos conseguem se manter nas primeiras posições tanto em relação à qualidade dos seus vetores quanto em relação ao seu custo computacional. O algoritmo DS com PS 2:1, por exemplo, possui resultados de qualidade superiores aos algoritmos HS, OTS e DCS mesmo utilizando um número menor de cálculos de SAD. Devemos ressaltar que os algoritmos rápidos podem diminuir o número de operações em até 7151 vezes, se compararmos os extremos da Tabela 4.3 para a área de 208x208, e mesmo assim possuem bons resultados em termos de erro gerado e ganho PSNR.

Uma característica importante dos algoritmos rápidos é o baixo aumento no custo computacional com o aumento da área de pesquisa. Podemos perceber que em todos os casos, a diferença de número de cálculos de SAD utilizados entre as áreas de 46x46 e 208x208 é muito pequena. Isto mostra que, em geral, os algoritmos rápidos não costumam buscar seus vetores em posições muito distantes do início da busca.

## 4.4 Percentual de Vetores ótimos

Quando avaliamos a qualidade dos vetores gerados por cada algoritmo estamos considerando apenas os seus resultados de SAD acumulado e ganho PSNR. No entanto, mesmo os resultados de SAD sendo muito próximos ao valor ótimo, os vetores de movimento gerados podem ser completamente diferentes. O cálculo do SAD considera apenas a diferença absoluta entre a amostra da área de pesquisa e a amostra do bloco que está sendo pesquisado. Isto pode fazer com que os algoritmos de pesquisa encontrem um valor de SAD próximo do valor ótimo, mesmo desviando a busca em relação à região onde se encontra o vetor de movimento ótimo.

Para avaliar quais dos algoritmos são mais eficazes na escolha dos vetores, realizamos uma comparação entre os percentuais de acerto de vetores ótimos dos algoritmos em relação aos vetores gerados pelos algoritmos FS, que sempre gera os vetores ótimos para uma dada área de pesquisa. A Tabela 4.4 apresenta os resultados médios de percentual de vetores ótimos encontrados para os algoritmos sub-ótimos em todas as áreas de pesquisa.

Mais uma vez o algoritmo FS com PS 2:1 obteve os melhores resultados dentre os algoritmos rápidos acertando mais cerca de 75% dos vetores ótimos na média para as quatro áreas de pesquisa avaliadas. A versão do algoritmo FS com PS 4:1 não manteve os bons resultados obtidos nos testes de qualidade, o que mostra que em muitos casos ele encontra um bom resultado de SAD só que para vetores não ótimos. As versões do algoritmo FS com sub-amostragem de blocos obtiveram resultados ainda piores. Isto pode ser explicado pelo fato do algoritmo desconsiderar a metade dos blocos candidatos da área de pesquisa antes mesmo de iniciar o processo de busca. Desta forma, o bloco candidato que gera o vetor ótimo pode ser eliminado da busca antes mesmo de ser comparado.

Tabela 4.4: Percentual de acerto de vetores ótimos

| **Algoritmos** | **Áreas de Pesquisa** | | | |
|---|---|---|---|---|
| | **46x46** | **80x80** | **144x144** | **208x208** |
| FS + PS 2:1 | 77,16 | 75,85 | 75,22 | 74,96 |
| DS | 69,97 | 67,63 | 65,96 | 65,22 |
| DS + PS 2:1 | 59,6 | 58,02 | 56,92 | 56,41 |
| HS | 57,42 | 56,3 | 55,14 | 54,61 |
| OTS | 55,1 | 53,5 | 52,58 | 52,17 |
| FS + PS 4:1 | 52,64 | 51,74 | 51,02 | 50,68 |
| HS + PS 2:1 | 50,93 | 50,11 | 49,29 | 48,9 |
| DCS | 49,34 | 47,64 | 46,69 | 46,27 |
| FS + BS 2:1+ PS 2:1 | 40,92 | 50,12 | 49,25 | 49,02 |
| OTS + PS 2:1 | 48,6 | 47,24 | 46,56 | 46,26 |
| DCS + PS 2:1 | 43,63 | 42,33 | 41,62 | 41,31 |
| TSS | 51,82 | 39,4 | 30,32 | 22,96 |
| FS + BS 4:1+ PS 4:1 | 13,04 | 31,73 | 30,87 | 30,61 |
| TSS + PS 2:1 | 45,13 | 32,56 | 27,23 | 30,01 |

Dentre os algoritmos rápidos devemos destacar o excelente resultado do algoritmo DS, que obteve um percentual de acerto superior a 67% na média para as quatro áreas de pesquisa. A versão do algoritmo DS com PS 2:1 também obteve um bom resultado, sendo mais eficiente que os demais algoritmos acertando mais de 57% dos vetores ótimos em média. Os algoritmos HS e OTS também obtiveram bons resultados, sempre melhores que os algoritmos FS com PS 4:1 e FS com BS em todas as áreas de pesquisa.

O percentual de acerto dos vetores cai com o aumento da área de pesquisa para praticamente todos os algoritmos. Isto ocorre porque o algoritmo FS consegue explorar melhor as vantagens do aumento da área de busca, enquanto os demais algoritmos, acabam escolhendo vetores sub-ótimos mais próximos do início da busca.

## 4.5 Avaliação das Iterações dos Algoritmos Rápidos

Os algoritmos rápidos possuem uma série de características que visam acelerar o processo de estimação. Como o objetivo deste trabalho é analisar os algoritmos de estimação de movimento visando uma implementação em hardware, estas características dos algoritmos rápidos devem ser melhor avaliadas. Em geral, os algoritmos rápidos realizam a busca em dois passos distintos. O primeiro avalia o centro da região de pesquisa, juntamente com alguns pontos ao redor, caso o melhor resultado seja obtido para o centro a pesquisa é encerrada. Caso contrário, o segundo passo é aplicado, onde algum determinado padrão de busca é repetido até que uma condição de parada seja satisfeita.

Realizamos uma análise sobre o percentual de término da busca dos algoritmos rápidos no primeiro e segundo passos. Os resultados mostraram, que em média, os algoritmos terminam a sua busca no primeiro passo em 22% dos casos. Isto indica que geralmente 78% das buscas terminam no segundo passo dos algoritmos, onde em geral existe uma repetição de um padrão de busca, que pode ser repetido *n* vezes. O número médio de iterações de cada algoritmo é um dado importante a ser avaliado quando consideramos uma implementação em hardware. Este dado pode ajudar a determinar o nível de paralelismo ideal para desenvolver o *datapath* para segundo passo do algoritmo em uma arquitetura de hardware. Uma exploração correta do paralelismo no segundo passo dos algoritmos rápidos pode influenciar diretamente no desempenho da arquitetura. A Tabela 4.5 apresenta o número de iterações no segundo passo para todos os algoritmos rápidos em todas as áreas de pesquisa. O algoritmo TSS não aparece, pois possui número fixo de iterações igual a três.

Tabela 4.5: Média de número de iterações para os algoritmos rápidos

| **Algoritmos** | **Áreas de Pesquisa** | | | |
|---|---|---|---|---|
| | **46x46** | **80x80** | **144x144** | **208x208** |
| DCS + PS 2:1 | 2,62 | 2,85 | 2,88 | 2,89 |
| DCS | 2,65 | 2,90 | 2,94 | 2,94 |
| HS + PS 2:1 | 2,62 | 2,92 | 2,97 | 2,97 |
| HS | 2,67 | 2,99 | 3,04 | 3,04 |
| DS + PS 2:1 | 2,79 | 3,12 | 3,17 | 3,17 |
| DS | 2,83 | 3,18 | 3,23 | 3,24 |
| OTS + PS 2:1 | 4,64 | 5,06 | 5,17 | 5,19 |
| OTS | 4,83 | 5,57 | 5,76 | 6,22 |

Cada valor na Tabela 4.5 representa a média de iterações de cada algoritmo no seu segundo passo, para as 10 amostras de vídeo utilizadas. Para todos os algoritmos, exceto o algoritmo TSS, o número de iterações tente a aumentar consideravelmente para vídeos com maior movimentação de imagem. Os algoritmos DCS e HS calculam três novos blocos candidatos a cada iteração. Devido a isto, seus resultados médios de iterações são bastante similares. O algoritmo DS pode calcular três ou cinco novos blocos a cada iteração, devido a isto, seus resultados são um pouco menores do que os algoritmos

DCS e HS. Já o algoritmo OTS, calcula apenas um novo bloco candidato a cada passo da iteração, devido a isto, ele possui a maior média de iterações dentre todos os algoritmos investigados.

Mais uma vez os resultados não mudam significativamente com o aumento da área de pesquisa. Dentre os três algoritmos mais similares, apenas o DS teve um aumento mais significativo elevando a sua média de iterações de 2,83, para a área de 46x46, para 3,90 para a área de 208x208.

A aplicação de PS 2:1 gerou resultados similares para todos os algoritmos. As versões com sub-amostragem apresentam uma diminuição no número médio de iterações para todos os algoritmos. Isto ocorre, pois ao utilizar a sub-amostragem os algoritmos acabam escolhendo blocos candidatos diferentes. Desta forma, o algoritmo pode mudar o rumo da busca para uma direção diferente, comparado a direção do algoritmo sem sub-amostragem. Em geral, esse caminho escolhido pelos algoritmos com sub-amostragem tende a ser menor, por isso menos iterações são utilizadas na segunda etapa do algoritmo.

### 4.5.1 Pior caso no número de Iterações

Para determinar o desempenho de uma arquitetura de hardware para um algoritmo rápido, também é importante determinar o pior caso no número de iterações para o segundo passo dos algoritmos. A informação de pior caso médio, aliada a informação de nível de paralelismo dessa etapa na arquitetura, pode determinar o desempenho médio da arquitetura de estimação. A Tabela 4.6 apresenta o pior caso em número de iterações para o segundo passo dos algoritmos rápidos para todas as áreas de pesquisa. Mais uma vez o algoritmo TSS possui um número fixo para o pior caso igual a três e não aparece nos resultados da Tabela 4.6.

Tabela 4.6: Pior caso em termos de número de iterações

| **Algoritmos** | **Áreas de Pesquisa** | | | |
|---|---|---|---|---|
| | **46x46** | **80x80** | **144x144** | **208x208** |
| DS | 17,2 | 27,3 | 34,1 | 38,9 |
| DS + PS 2:1 | 15,5 | 27,0 | 35,0 | 39,2 |
| HS | 15,1 | 22,5 | 30,2 | 33,1 |
| HS + PS 2:1 | 14,5 | 22,4 | 29,8 | 35,4 |
| OTS | 22,9 | 40,8 | 66,4 | 83,7 |
| OTS + PS 2:1 | 22,2 | 39,9 | 57,6 | 70,1 |
| DCS | 15,9 | 27,1 | 32,4 | 34,9 |
| DCS + PS 2:1 | 15,5 | 25,8 | 33,2 | 35,0 |

Diferentemente de todos os resultados apresentados anteriormente, os algoritmos possuem variações consideráveis nos piores casos de iterações entre as áreas de pesquisa. Para todos os casos o valor máximo de iteração aumenta em mais de duas vezes se compararmos à menor e a maior área de pesquisa. Isto significa que em alguns casos, os algoritmos rápidos direcionam a sua busca para pontos distantes na área de pesquisa, realizando um grande número de iterações. No entanto, isto não ocorre com

muita freqüência, pois estes valores médios de pior caso não afetam significativamente no número médio de iterações dos algoritmos, apresentados na Tabela 4.5.

Ao contrário do que foi apresentado na Tabela 4.5, os resultados da Tabela 4.6 mostram que a sub-amostragem aumenta o número máximo de iterações usadas para todos os algoritmos. Este é um dado curioso, pois ao mesmo tempo em que diminui a média de laços utilizados, a sub-amostragem aumenta os valores máximos de iterações. Estas ocorrências de grande número de iterações, apesar de maiores do que as obtidas com as versões sem sub-amostragem, não são muito freqüentes, caso contrário as médias e número de iterações seriam maiores.

## 4.6 Discussão dos Resultados

Com as discussões apresentadas nesta seção, buscamos avaliar os algoritmos de estimação de movimento sobre diversos critérios. A questão do desempenho, quando pensamos em aplicações em hardware, insere outro critério na comparação dos resultados, pois a inserção de paralelismo acarreta em um acréscimo no custo em área para a arquitetura do estimador. Comparando os resultados de número de operações dos algoritmos, percebe-se que para todas as versões dos algoritmos FS com PS e BS este valor é muito superior aos valores obtidos pelos algoritmos rápidos. Isto mostra que as arquiteturas para estes algoritmos necessitam de um número muito maior de operadores para realizar a estimação, caso estejamos considerando a paralelização das operações para aumentar o desempenho. Isto implica em um grande consumo de recursos de hardware para arquiteturas que utilizam os algoritmos FS e suas versões com sub-amostragem.

As avaliações em diversas áreas de pesquisa mostraram que não é uma estratégia interessante ampliar o tamanho da área para vetores com componentes maiores que 32. A partir disto o custo computacional passa a ser muito elevado em vista dos ganhos em qualidade que são praticamente nulos. Vale ressaltar que o estudo sobre tamanhos de área de busca vale para vídeos SDTV 740x480 com blocos de 16x16 *pixels*, no entanto, estes tamanhos de área de pesquisa servem como base para determinar o tamanho ótimo da área de pesquisa para outras resoluções de vídeo. Para vídeos HDTV, por exemplo, onde as resoluções são bem maiores, a área de pesquisa ideal também será maior. Isto aumenta ainda mais as vantagens dos algoritmos rápidos, pois o seu custo computacional, em número de cálculos de SAD, não é diretamente proporcional ao aumento da área de busca, assim como o algoritmo FS e suas versões com sub-amostragem.

Considerando a qualidade dos vetores gerados, podemos perceber que a sub-amostragem de blocos causa perdas maiores no erro e no ganho PSNR em relação à sub-amostragem de *pixel*. Isto fica claro quando comparamos os resultados do algoritmo FS com PS 4:1 e FS com BS e PS 2:1, que possuem o mesmo número de cálculos de SAD. Os resultados de qualidade do algoritmo FS com PS 4:1 são melhores em todos os critérios de qualidade avaliados. Os resultados para os algoritmos rápidos são considerados satisfatórios. Os algoritmos rápidos apresentaram a melhor relação entre diminuição do erro, ganho PSNR, percentual de acerto de vetores ótimos e o custo computacional envolvido.

A aplicação de PS 2:1 se mostrou uma excelente estratégia para todos os algoritmos. Para o algoritmo FS é possível diminuir o número de cálculos de SAD em 50%, com uma perda de qualidade muito pequena. Estas perdas são de apenas 0,3% na

diminuição do erro absoluto e de -0,061dB no ganho PSNR. Para os algoritmos rápidos os resultados também foram muito bons. Para todos os algoritmos a redução no número e cálculos de SAD foi sempre no mínimo igual a 50%, sendo que para alguns algoritmos a redução foi superior a 50%. As perdas em qualidade apresentadas foram mínimas, sendo que os algoritmos DS e HS a diferença na redução do erro absoluto foi de apenas 0,44%, e no ganho PSNR de cerca 0,1dB.

Depois de avaliar os resultados de todos os algoritmos, comparando qualidade, custo computacional e características específicas de cada algoritmo, o algoritmo DS foi escolhido para ser desenvolvido em hardware. O algoritmo DS obteve os melhores resultados de qualidade entre todos os algoritmos rápidos investigados, aliado a um baixíssimo custo computacional se comparado ao algoritmo FS e suas versões com sub-amostragem. Na próxima seção vamos apresentar uma avaliação mais aprofundada do algoritmo DS.

## 4.7 Refinamento da Avaliação do Algoritmo DS

Depois de avaliar diversos algoritmos de estimação, considerando vários critérios de qualidade e custo computacional, o algoritmo DS foi o escolhido para ser desenvolvido em hardware. O algoritmo DS apresentou a melhor relação entre custo computacional e qualidade dos vetores gerados dentre todos os algoritmos avaliados.

Vamos agora realizar mais alguns testes, apenas com o algoritmo DS, para definir os parâmetros finais para a implementação em hardware. Para estes testes será considerada a área de pesquisa como sendo o quadro inteiro. Deste modo poderemos extrair os resultados máximos do algoritmo. Vamos comparar três diferentes versões do algoritmo, sem sub-amostragem, com PS 2:1 e com PS 4:1. A Tabela 4.7 apresenta os resultados completos para as três versões do algoritmo DS, considerando os resultados médios obtidos para as 10 amostras de vídeo utilizadas.

Tabela 4.7: Resultados completos para o DS, DS + PS 2:1 e DS + PS 4:1

| Critério | DS | DS + PS 2:1 | DS + PS 4:1 |
|---|---|---|---|
| Redução do erro (%) | 49,17 | 48,71 | 43,67 |
| PSNR (dB) | 27,285 | 27,207 | 26,033 |
| Número de SADs ($10^9$) | 0,87 | 0,43 | 0,21 |
| Número de Iterações (médio) | 3,24 | 3,17 | 2,96 |
| Pior caso de Iterações | 45,8 | 45,3 | 40,6 |

Os resultados do DS e DS + PS 2:1, apresentados na Tabela 4.7, são praticamente iguais aos resultados obtidos por estes algoritmos na área de 208x208, apresentados nas seções anteriores. A diferença mais significativa é no pior caso em número de iterações, que cresce consideravelmente. Isto mostra que em alguns casos o algoritmo busca seus vetores em pontos remotos dentro do quadro.

Comparando os resultados de qualidade da Tabela 4.7, podemos perceber que a sub-amostragem de *pixel* de 4:1 gera uma degradação considerável. O algoritmo DS com PS 4:1 apresenta um resultado de redução do erro 5,5% inferior ao DS, enquanto que para o DS com PS 2:1 a diferença é de apenas 0,46%. Com relação ao ganho PSNR o

algoritmo DS com PS 4:1 apresenta uma perda de 1,252 dB, enquanto o algoritmo DS com PS 2:1 gera um resultado apenas 0,078 dB inferior ao algoritmo DS.

Em relação ao custo computacional as versões com sub-amostragem levam vantagem. Usando PS 2:1 o número de cálculos de SAD utilizados cai em mais de duas vezes, comparado ao número de SADs utilizado na versão sem sub-amostragem. Para a versão com PS 4:1 o número de SADs cai em mais de quatro vezes. Os resultados de número de iterações também diminuem, sendo que para a versão DS com PS 4:1 a média de iterações é menor que três. Os resultados de pior caso também seguem esta tendência, e diminuem com o aumento da sub-amostragem.

Analisando todos os dados da Tabela 4.7 concluímos que o algoritmo DS com PS 2:1 possui os melhores resultados na relação qualidade versus custo computacional. Com o PS 2:1 é possível reduzir o número de SADs em mais de duas vezes, e com perdas mínimas de qualidade em relação ao DS sem sub-amostragem. A versão do algoritmo DS com PS 4:1 pode ser uma opção de baixíssimo custo computacional, no entanto, com perdas mais significativas de qualidade.

Depois da análise de todos os resultados apresentados, definimos que a arquitetura de hardware será desenvolvida para duas versões do algoritmo DS, com PS 2:1 e 4:1. Para facilitar o entendimento do texto, vamos chamar o algoritmo DS com PS 2:1 de SDS (*Sub-sampled Diamond Search*) e o algoritmo DS com PS 4:1 de QSDS (*Quarter Sub-sampled Diamond Search*). Daqui para frente, a sigla SDS será utilizada para representar o algoritmo DS com PS 2:1, e para representar o algoritmo DS com PS 4:1 será usada a sigla QSDS.

# 5 A ARQUITETURA DO SDS (*SUB-SAMPLED DIAMOND SEARCH*)

Após a etapa de avaliação algorítmica, que culminou na escolha dos algoritmos SDS e QSDS para a implementação em hardware, deu-se início ao desenvolvimento arquitetural para a ME. A primeira arquitetura desenvolvida foi para o algoritmo SDS. Os principais objetivos a serem alcançados com as arquiteturas de ME desenvolvidas neste trabalho são desempenho e economia de recursos de hardware, isso sem comprometer o resultado de qualidade dos vetores de movimento gerados. Os resultados de qualidade apresentados pelos algoritmos SDS e QSDS nas avaliações em software comprovam a sua eficiência, no entanto, o desenvolvimento destes algoritmos em hardware apresenta diversos desafios.

Durante as avaliações em software, o desempenho e o custo computacional dos algoritmos foi medido através do número médio de cálculos de SAD obtido. Essa métrica é muito relevante, pois com a execução seqüencial das instruções o desempenho e o custo computacional são proporcionais ao número de cálculos de SAD necessários. Para soluções em hardware o desempenho não depende apenas do número de cálculos de SAD, pois vários cálculos de SAD podem ser executados em paralelo, o que pode acelerar consideravelmente o processo de estimação. No entanto, a capacidade de paralelismo está diretamente ligada às características do algoritmo. O algoritmo FS, incluindo suas versões com sub-amostragem, possibilita a utilização do paralelismo em diversos níveis, inclusive o paralelismo máximo, onde todos os cálculos de SAD podem ser executados ao mesmo tempo. Para os algoritmos rápidos, como o SDS e QSDS, por exemplo, a capacidade de paralelismo é limitada, pois existe uma dependência de dados entre as etapas do algoritmo. Outro fator que influencia no desempenho é a freqüência de operação, que pode ser ajustada, dependendo da finalidade da aplicação da arquitetura.

Outro critério importante no desenvolvimento da arquitetura é o consumo de recursos de hardware. Arquiteturas para o algoritmo FS necessitam de um nível de paralelismo alto para atingir um desempenho elevado, isso implica em um grande custo em recursos de área. As soluções arquiteturais para algoritmos rápidos, mesmo com baixo nível de paralelismo, podem atingir altas taxas de processamento, devido ao baixo número de cálculos de SAD utilizados. Assim, as arquiteturas para algoritmos rápidos, como o SDS e QSDS, podem obter alto desempenho com baixo custo em consumo de recursos de hardware.

A arquitetura desenvolvida para o algoritmo SDS opera sobre blocos de tamanho 16x16 e utiliza o SAD como critério de distorção. A arquitetura do SDS possui o nível máximo de paralelismo permitido pelo algoritmo, sendo que um LDSP completo pode ser calculado em paralelo. O diagrama em blocos da arquitetura desenvolvida para o algoritmo SDS está ilustrado na Figura 5.1.

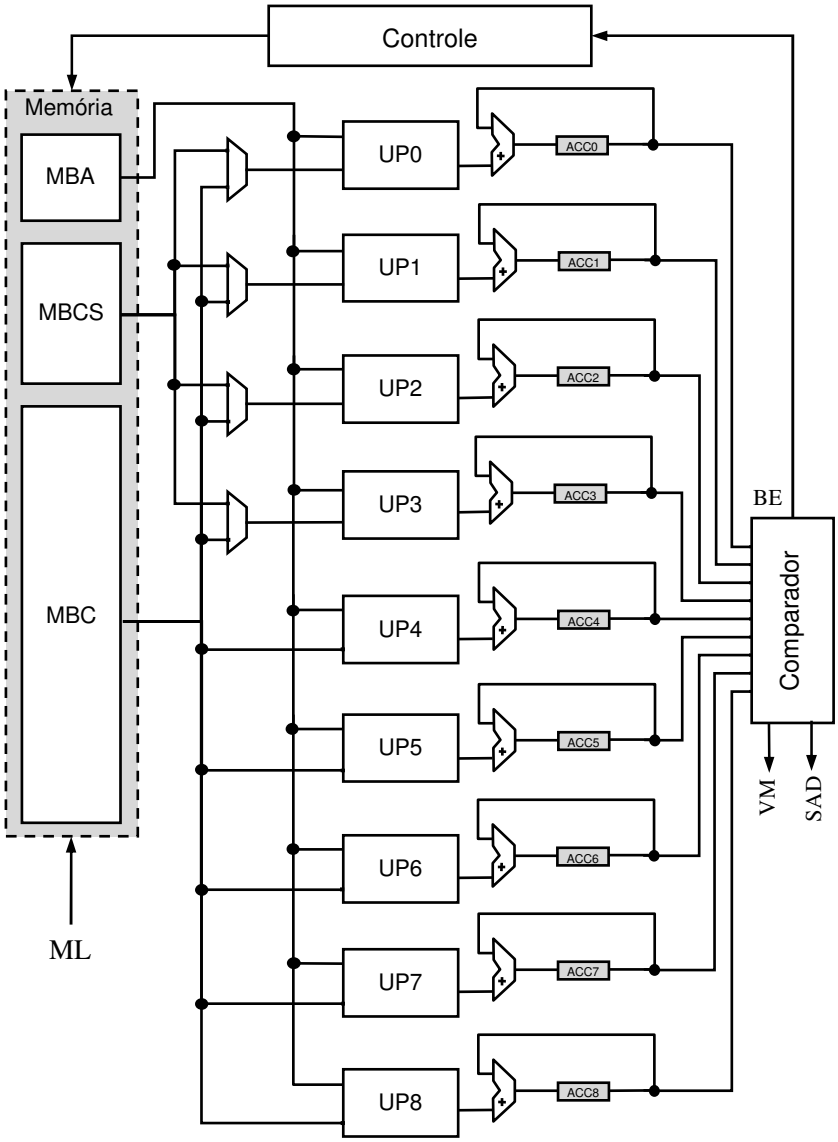



Figura 5.1: Diagrama de blocos da arquitetura do estimador de movimento

A arquitetura possui um módulo de memória, que reúne um conjunto de memórias internas para armazenar os blocos candidatos e o bloco do quadro atual, para o qual a estimação está sendo realizada. São utilizadas nove unidades de processamento (UP), onde cada UP é responsável por calcular o SAD para um bloco candidato. A Figura 5.1 também apresenta o módulo de controle e o comparador. O comparador recebe o resultado de SAD calculado pelas nove UPs e encontra o menor resultado.

Cada UP pode processar oito amostras em paralelo, isso significa que uma linha inteira do bloco de 16x16 sub-amostrado (o bloco sub-amostrado possui 16 linhas de oito amostras) é processada em paralelo. A Figura 5.2 ilustra a arquitetura de uma UP

de oito amostras. Este bloco consiste em uma árvore de somadores em *pipeline*. Primeiramente as oito amostras do bloco atual e do bloco candidato são subtraídas, após, o módulo da diferença entre as amostras é somado e o resultado final da acumulação das diferenças, para todas as amostras da linha, é armazenado no registrador de saída. A UP tem uma latência de quatro ciclos e, com o *pipeline* cheio, ela pode calcular uma linha de bloco candidato a cada ciclo de relógio. Um conjunto somador/acumulador é utilizado nas saídas das UPs para armazenar os resultados intermediários de cada bloco candidato. Ao todo, 16 acumulações são necessárias para gerar o resultado final do SAD para um bloco candidato, uma acumulação para cada linha do bloco. As UPs recebem os dados das memórias de bloco candidato (MBC) e da memória do bloco atual (MBA). Cada UP recebe uma palavra da MBC e uma palavra da MBA e calcula a diferença entre as oito amostras presentes em cada uma destas palavras.

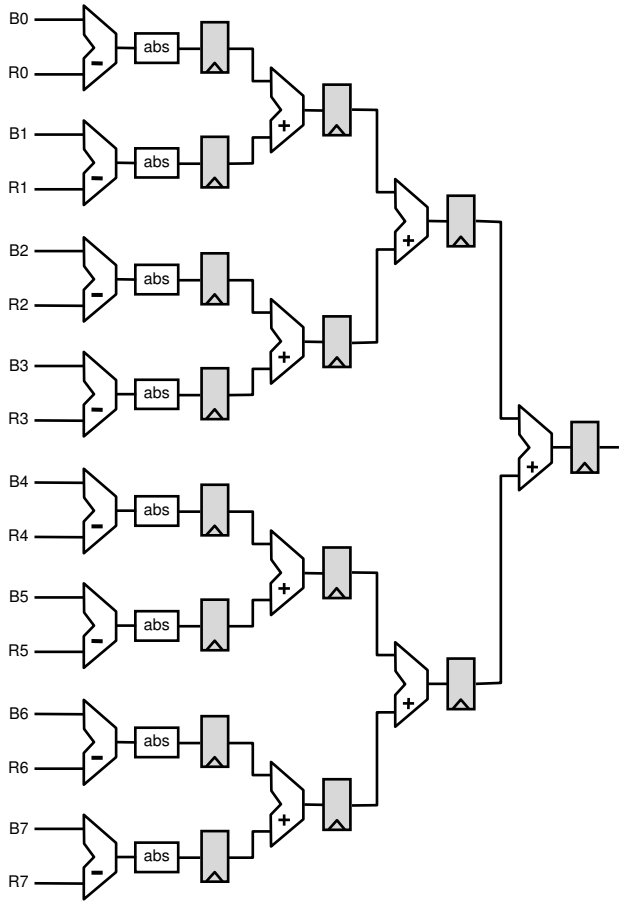



Figura 5.2: Diagrama de blocos da arquitetura da unidade de processamento (UP)

A arquitetura do algoritmo SDS possui quatro modos de operação: o primeiro passo, o segundo passo, que pode ser dividido em busca por aresta e busca por um vértice, e o refinamento final (SDSP). Para todos estes casos, a arquitetura consegue calcular todos os blocos candidatos em paralelo. No primeiro passo, os nove blocos candidatos do LSDP são avaliados e as nove UPs são utilizadas em paralelo. No segundo passo, é possível calcular três ou cinco blocos candidatos. No caso de uma busca por aresta, três novos blocos candidatos são calculados. Quando a busca é por um vértice, cinco novos blocos candidatos são avaliados. Para o SDSP, mais quatro blocos candidatos são calculados. Neste caso as UPs recebem os dados de memórias específicas, que

armazenam os blocos do SDSP. Estas memórias são chamadas de MBCS (Memória de Bloco Candidato do SDSP) e um conjunto de multiplexadores seleciona estas entradas para as quatro primeiras UP de acordo com os sinais de controle.

Os resultados finais de SAD para cada bloco candidato são armazenados em acumuladores (ACC0 a ACC8 na Figura 5.1). Estes resultados são enviados ao comparador. O comparador analisa os valores dos nove acumuladores em paralelo e encontra o menor SAD dentre os blocos candidatos. A Figura 5.3 apresenta a arquitetura interna do comparador utilizado.

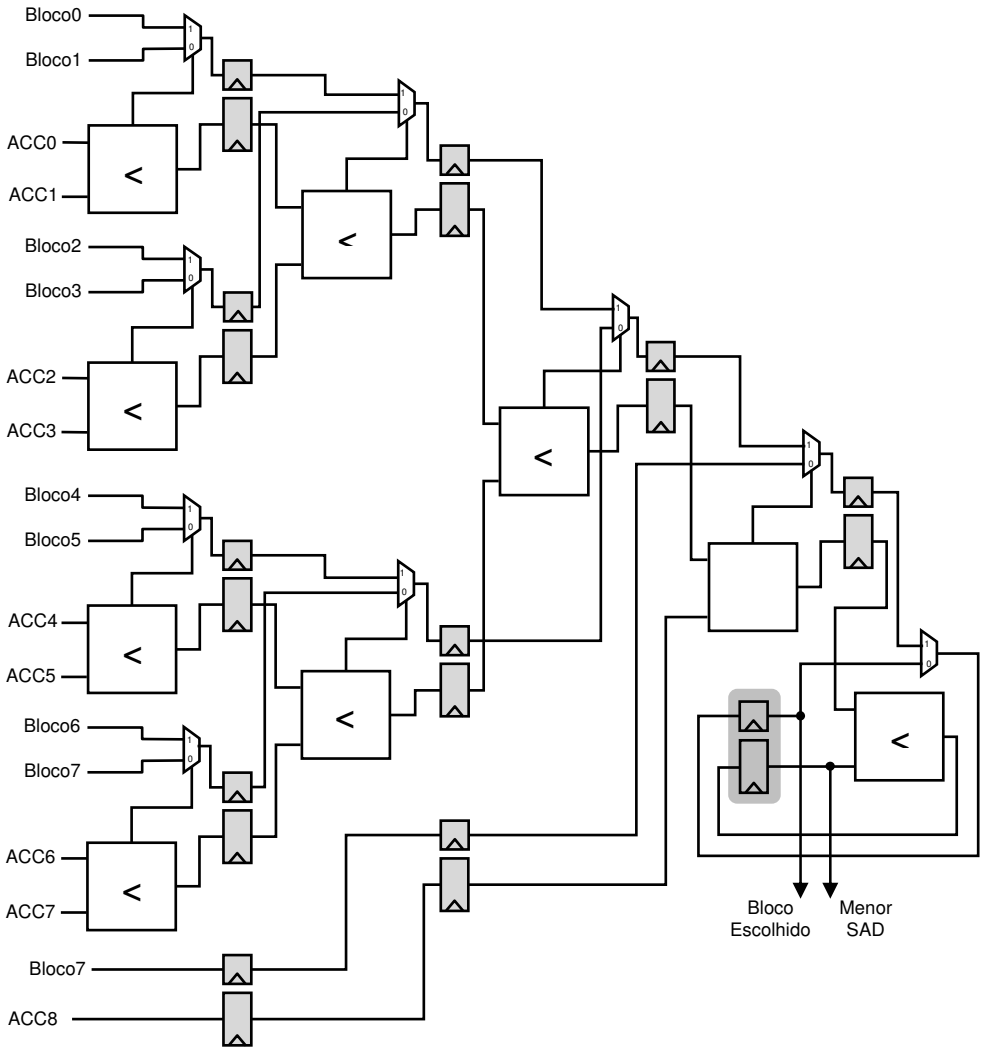



Figura 5.3: Diagrama de blocos da arquitetura do comparador

O comparador está estruturado em um *pipeline* de cinco estágios. O último estágio do comparador compara o menor resultado de SAD dentre as nove entradas e o menor resultado obtido no passo anterior. Desta forma, o registrador final do comparador armazena o menor resultado de SAD para todo o processo de estimação. O comparador também identifica qual foi o bloco com o menor SAD, e armazena esta informação no registrador final.

Cada bloco candidato possui um número, de acordo com sua posição dentro do LDSP. A Figura 5.4 ilustra o LDSP com todos os seus blocos candidatos numerados. Cada UP trabalha sempre sobre o mesmo 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. O comparador identifica o menor SAD e o bloco correspondente e envia a informação de bloco escolhido para a unidade de controle. A unidade de controle analisa qual bloco candidato foi escolhido e então decide o próximo passo do algoritmo.

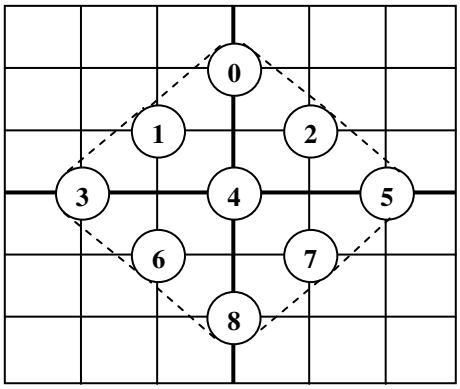



Figura 5.4: LDSP com os blocos candidatos numerados

Caso o bloco 4 seja escolhido, a unidade de controle identifica que o menor resultado de SAD foi encontrado no centro do LDSP. Neste caso, a unidade de controle dispara o processo de refinamento final, com o SDSP. Para o SDSP mais quatro blocos candidatos são calculados e seus resultados de SAD são enviados ao comparador. O menor SAD é encontrado e a informação do bloco escolhido é enviada ao controle. A unidade de controle recebe a informação de bloco escolhido e gera o vetor de movimento correspondente ao bloco.

Quando o bloco escolhido no primeiro passo do algoritmo for 0, 3, 5 ou 8, a unidade de controle começa a segunda etapa do algoritmo com uma busca por vértice. Neste caso, mais cinco blocos candidatos são avaliados e seus resultados de SAD são comparados com o menor SAD do passo anterior. Se o bloco escolhido no primeiro passo for 1, 2, 6 ou 7, a unidade de controle começa a segunda etapa do algoritmo com uma busca por aresta. Na busca por aresta mais três blocos candidatos são avaliados e comparados com o menor SAD do passo anterior. O segundo passo do algoritmo pode ser repetido *n* vezes, até que o menor resultado de SAD seja encontrado para o bloco candidato do centro do LDSP, então o SDSP é aplicado.

## 5.1 Organização da Memória

A memória interna do estimador está organizada em 15 memórias distintas, como apresentado na Figura 5.5. A memória local (ML) acessa a memória externa, onde está armazenado o vídeo sem compressão, e 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, quando a unidade de controle decide quais blocos devem ser avaliados no próximo passo do algoritmo a ML já possui estes dados armazenados. Toda vez que o controle da arquitetura decide continuar a busca, a ML é recarregada, isto ocorre simultaneamente à leitura, assim nenhum ciclo é gasto com latência na escrita recarga da ML. A ML é composta por 24 palavras de 128 bits cada. Outras 14 pequenas memórias são utilizadas para armazenar os blocos candidatos e o bloco 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 sub-amostrado de 16x8 amostras.

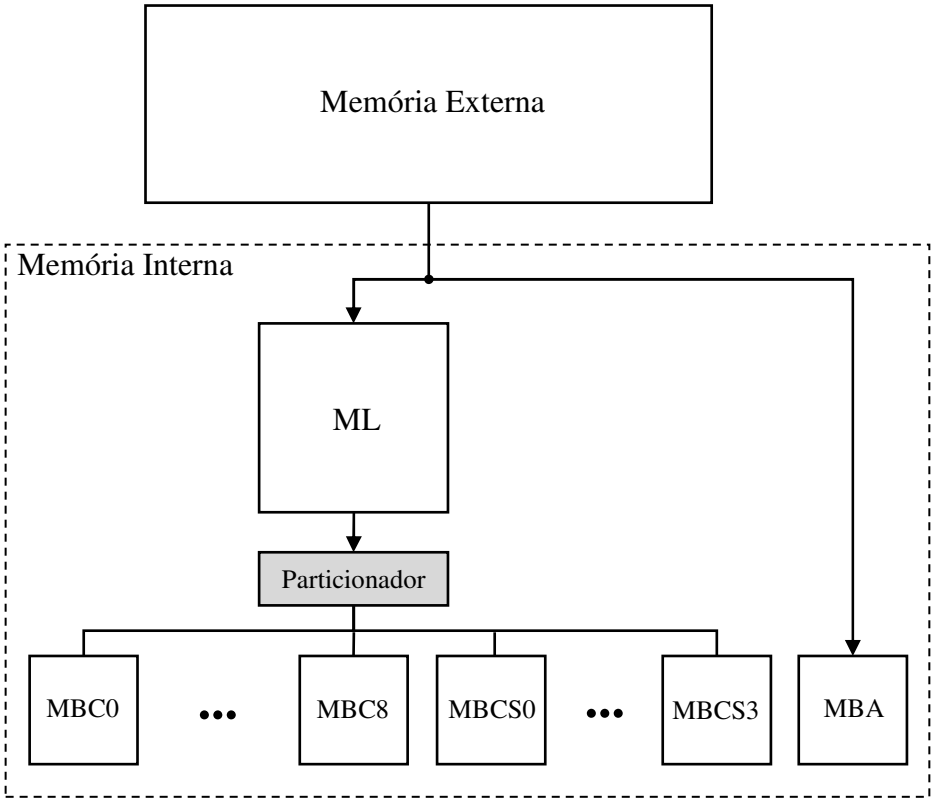



Figura 5.5: Organização das memórias do estimador

A memória ML é lida linha por linha e os dados são armazenados nas MBCs. Nove MBCs são utilizadas para armazenar os blocos do LDSP. Quatro MBCSs são utilizadas para armazenar os blocos do SDSP e estes blocos são armazenados sempre que a ML é lida, deste modo, quando a unidade de controle decide iniciar o SDSP, os blocos candidatos já estão disponíveis nas MBCSs. Esta solução aumenta o desempenho da arquitetura e reduz a latência de acesso às memórias dos blocos candidatos do SDSP. Quando o modo SDSP é ativado, a unidade de controle seleciona os multiplexadores na Figura 5.1 e as UPs recebem os dados das memórias MBCSs. Desta forma, nenhum ciclo de latência no acesso a memória é gasto para começar o SDSP.

Cada linha da ML possui 128 bits, no entanto, a unidade de particionamento (Particionador na Figura 5.5) seleciona apenas 64 bits desta palavra para cada MBC e MBCS. Uma linha da ML pode conter informações de mais de um bloco candidato, devido a isto, o Particionador identifica os 64 bits que correspondem a uma linha de cada bloco candidato. Quando a primeira linha da ML é lida, o Particionador seleciona os 64 bits da palavra original (com 128 bits) que correspondem a primeira linha do bloco candidato 0, que será armazenada na MBC0. Quando a segunda linha é lida, o Particionador seleciona os 64 bits que correspondem a segunda linha do bloco candidato 0, que será armazenada na segunda linha da MBC0, e também os 64 bits que correspondem a primeira linha do bloco candidato 1, que será armazenada na MBC1. Este processo termina quando toda a ML é lida e todas as MBCs forem preenchidas, incluindo as quatro MBCSs.

Uma unidade de controle local foi desenvolvida para controlar as operações de escrita e leitura nas memórias internas do estimador. A unidade de controle da Figura 5.1 não é responsável pelo controle de acesso às memórias da arquitetura, ela apenas indica o início das operações e informa o bloco escolhido, a cada passo do algoritmo, para que o controle local identifique os dados corretos a serem carregados nas memórias. Quando o *datapath* conclui o cálculo dos SADs dos blocos candidatos, a unidade de controle informa ao controle da memória se a busca deve continuar ou se o SDSP deve ser aplicado. Caso a busca continue, seja por uma busca por vértice ou por uma busca por aresta, as MBCs são armazenadas com os novos blocos candidatos. A ML é recarregada, de forma que contenha novamente todos os próximos blocos candidatos possíveis para o novo LDSP. Neste caso 10 ciclos de latência são necessários, para que uma linha seja escrita em cada uma das nove MBCs. Caso o SDSP seja aplicado, os dados referentes aos quatro blocos candidatos já estão armazenados nas MBCSs e o cálculo dos SAD inicia sem nenhum ciclo de latência.

Mesmo usando 15 memórias diferentes, a arquitetura do SDS utiliza uma pequena quantidade de memória interna. A área de pesquisa é carregada de acordo com a necessidade do algoritmo e, desta forma, nenhum dado irrelevante para o algoritmo é carregado. Para a ML são usadas 24 palavras de 128 bits cada, num total de 3072 bits de memória. Cada memória de bloco (MBC, MBCS ou MBA) usa apenas 16 linhas de 64 bits cada, totalizando 1024 bits. Somando todos os recursos de memória utilizados pela arquitetura do estimador chegamos a um total de 17408 bits. Este valor pode ser considerado muito pequeno se compararmos a outras arquiteturas de ME desenvolvidas no grupo de pesquisa (PORTO, 2006) e (AGOSTINI, 2007).

## 5.2 Avaliação do Desempenho

O algoritmo SDS termina sua busca quando o menor SAD é encontrado para o bloco candidato do centro do LDSP. Esta condição pode ser satisfeita no primeiro passo do algoritmo, ou após *n* aplicações do LDSP. Quando o menor SAD é encontrado no centro do LDSP, nenhuma busca por vértice ou por aresta é realizada. Este é o melhor caso para uma busca do algoritmo SDS, quando apenas 13 blocos candidatos são avaliados: nove blocos do LDSP e mais quatro blocos do SDSP.

A arquitetura do SDS utiliza 36 ciclos de relógio para preencher todas as memórias e iniciar o processo de cálculo de SADs. As UPs possuem uma latência de quatro ciclos, e mais 15 são necessários para processar as 16 linhas dos blocos candidatos. O comparador necessita de mais cinco ciclos para encontrar o menor SAD dentre os resultados dos nove blocos candidatos. Assim, 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 necessita de 88 ciclos de relógio 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 relógio 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 repetição do LDSP. A arquitetura do SDS pode repetir o LSDP *n* vezes e para cada uma das repetições mais 34 ciclos serão necessários. Assim, para $n = 5$, por exemplo, 170 ciclos são utilizados para calcular as cinco repetições do LDSP. Ainda são necessários mais 60 ciclos para o primeiro LSDP e mais 28 para o SDSP, no

total, 258 ciclos de relógio são necessários para determinar o vetor de movimento quando cinco repetições do LDSP são usadas no segundo passo do algoritmo. Em função desta característica, é interessante definir algum tipo de restrição no número de repetições do LDSP, para garantir que a arquitetura não utilize um número muito grande de ciclos de relógio para gerar os vetores de movimento. Sem esta restrição, fica impossível garantir o sincronismo da arquitetura da ME, sendo necessária a implementação de algum protocolo para sincronizar a arquitetura com os demais módulos de um codificador de vídeo.

## 5.3 Controle Dinâmico de Laços

O algoritmo SDS pode utilizar *n* repetições do LDSP no segundo passo do algoritmo. As repetições serão chamadas de laços nesta dissertação. No entanto, é interessante que a arquitetura do SDS restrinja este número de laços, para que não utilize um número de ciclos de relógio muito elevado para gerar o vetor de movimento. Isto acabaria diminuindo o desempenho, e conseqüentemente, a arquitetura não atingiria as taxas de processamento necessárias para trabalhar com vídeos de alta resolução em tempo real. Além disso, o problema do sincronismo da arquitetura, com outros módulos do codificador, também é relevante e deve ser considerado. Para evitar estes problemas, é interessante e recomendável a aplicação de algum mecanismo de controle de laços na arquitetura. Este controle afeta o funcionamento do algoritmo, que em alguns casos terá a sua busca interrompida pela restrição no uso de laços. Esta interrupção antecipada na busca pode inserir perdas na qualidade do processo de estimação.

Para limitar o número de laços utilizado pela arquitetura, sem causar perdas significativas na qualidade dos vetores de movimento, foi desenvolvido um sistema dinâmico de controle de laços (CDL). O CDL permite que a arquitetura utilize uma quantidade variada de laços, dependendo da necessidade do algoritmo. Um histórico é armazenado com o número de ciclos utilizados pelos últimos 16 vetores, sendo que cada vetor tem um valor máximo de número de laços. Toda vez que o vetor for gerado com um número de laços inferior ao valor máximo que foi estipulado, os laços restantes são acumulados e os próximos vetores poderão utilizar um número maior de laços. Para avaliar o impacto, na qualidade dos vetores, com a inserção do CDL, uma nova bateria de testes em software foi realizada. A Tabela 5.1 apresenta os dados comparativos entre os algoritmos SDS, com e sem controle de laços. Foram gerados diversas versões do CDL, com valores máximos de laços por vetor deferentes. Também é apresentado o resultado gerado para o algoritmo SDS considerando uma restrição fixa de laços, também para valores variados.

A Tabela 5.1 apresenta os resultados médios de redução do erro absoluto e de ganho PSNR para o algoritmo SDS, considerando os valores médios para as 10 seqüências de vídeos apresentadas no capítulo 4, e utilizando o quadro inteiro como área de pesquisa. É possível perceber que os resultados de qualidade melhoram significativamente com o aumento do número máximo de laços na restrição fixa. O resultado de redução do erro absoluto cresce mais de 4% comparando as versões com restrição fixa de cinco e de vinte laços. Os resultados obtidos com o CDL são ainda melhores. Mesmo considerando um valor máximo de 10 laços por vetor, com o CDL é possível obter um resultado de qualidade superior ao obtido com a restrição fixa para 20 laços. A versão do SDS com CDL, para um valor máximo de 10 laços, obtém uma redução do erro apenas 0,13% inferior a versão do algoritmo sem nenhuma restrição de laços. O resultado de ganho PSNR também sofre uma pequena perda, de apenas 0,012dB.

Tabela 5.1: Resultados médios de qualidade para o algoritmo SDS, SDS com controle dinâmico de laços e SDS com controle fixo de laços

| Algoritmo | Redução do erro (%) | PSNR (dB) |
|---|---|---|
| SDS | 48,71 | 27,207 |
| SDS + CDL 20 laços | 48,66 | 27,203 |
| SDS + CDL 10 laços | 48,58 | 27,195 |
| SDS + restrição de 20 laços | 48,50 | 27,188 |
| SDS + restrição de 10 laços | 47,43 | 27,083 |
| SDS + CDL 5 laços | 44,65 | 26,795 |
| SDS + restrição de 5 laços | 44,41 | 26,750 |

Este bom desempenho acontece, pois a média de número de laços utilizados pelo algoritmo é baixa, apenas 3,12 laços por vetor (Tabela 4.5), assim, considerando 10 laços como restrição, em média sobram mais de seis laços a cada vetor gerado. Com o sistema de controle dinâmico é possível armazenar estes laços excedentes e utilizá-los quando o algoritmo precisar realizar uma busca mais longa. O CDL permite que o algoritmo possa, em alguns casos, utilizar um número muito grande de laços. Com um histórico de número de laços utilizado pelos últimos 16 vetores, sendo que cada vetor pode utilizar até 10 laços, tem-se um total de 160 laços que podem ser utilizados por cada grupo de 16 vetores. Considerando que a cada laço o algoritmo SDS avança duas amostras na área de pesquisa e que para o primeiro LSDP é utilizada uma área de pesquisa de 20x20 amostras, obtém-se uma área de pesquisa máxima de 660x660 amostras para a arquitetura do SDS com o CDL. Se for considerada uma restrição fixa, com o valor máximo de 20 laços para cada vetor, a área de pesquisa será de apenas 100x100 amostras.

Os ganhos apresentados nos resultados médios com a utilização do CDL são significativos quando comparados aos resultados obtidos com a restrição fixa de laços. No entanto, este ganho pode ser ainda maior, se forem considerados apenas os resultados obtidos para vídeos com grande movimentação. Vídeos com grande movimentação possuem uma tendência natural à ocorrência de vetores maiores e isto faz com que o algoritmo necessite de um número maior de laços para encontrar o melhor casamento. Nestes casos, a restrição fixa insere uma perda maior, pois não dá ao algoritmo a possibilidade de buscar vetores em posições mais distantes. A Tabela 5.2 apresenta os resultados obtidos para o vídeo de maior movimentação dentre as 10 amostras de vídeo utilizadas.

Os resultados mostram que a diferença de ganho obtida com o aumento no número de laços é maior do que as apresentadas para o caso médio. A diferença entre os resultados de diminuição do erro, para controle fixo com cinco e vinte laços, foi de mais de 15%. Essa diferença é muito superior aos 4% apresentados para o caso médio. A diferença entre as versões com CDL e com controle fixo também é maior. Considerando as versões para 10 laços, com a utilização do CDL é possível obter um ganho superior a 5% na redução do erro, em relação à versão com restrição fixa para esta amostra de vídeo. Para este mesmo caso, o ganho médio em redução do erro obtido com o CDL é

de apenas 1,15% em relação à versão com controle fixo de 10 laços. Apenas a versão do CDL com cinco iterações obteve um resultado de redução do erro inferior ao apresentado pela versão com restrição fixa. Isto ocorre, pois o controle dinâmico permite que os primeiros vetores, dos 16 armazenados no histórico, utilizem muitos laços. Assim, acabam sobrando poucos laços para os últimos vetores do histórico, ocorrendo um número maior de interrupções na busca. Mesmo assim, os resultados de ganho PSNR obtidos com o CDL são superiores aos apresentados pela versão com controle fixo.

Tabela 5.2: Resultados de qualidade do algoritmo SDS com e sem controle de laços para o vídeo de maior movimentação

| Algoritmo | Redução do erro (%) | PSNR (dB) |
|---|---|---|
| SDS | 58,00 | 26,761 |
| SDS + CDL de 20 laços | 57,16 | 26,663 |
| SDS + CDL de 10 laços | 56,63 | 26,611 |
| SDS + restrição de 20 laços | 56,23 | 26,576 |
| SDS + restrição de 10 laços | 51,30 | 26,108 |
| SDS + CDL de 5 laços | 39,76 | 25,293 |
| SDS + restrição de 5 laços | 41,13 | 25,176 |

A estrutura utilizada para implementar o CDL na arquitetura do SDS está ilustrada na Figura 5.6. O mecanismo é bastante simples e utiliza uma pequena quantidade de hardware para determinar o número de laços disponível para a arquitetura do estimador. A informação de número de laços utilizado pela arquitetura é enviada para um registrador de deslocamento (RD) de 16 posições. Um somador recebe os valores de número de laços utilizados para os últimos 16 vetores e, com o auxílio de um acumulador, faz a soma do número total de laços utilizados. Foi determinado um valor máximo de 10 laços para cada vetor. Desta forma, toda vez que forem utilizados menos do que 10 laços para determinar o vetor, os laços restantes são acumulados e, para os demais vetores, mais de 10 laços poderão ser utilizados em suas buscas. O valor acumulado de número de laços para os últimos 16 vetores é subtraído de 160, que é o valor máximo de laços que podem ser utilizados para 16 vetores (16 x 10 = 160). O valor resultante da subtração é o número de laços disponível para o cálculo do próximo vetor.

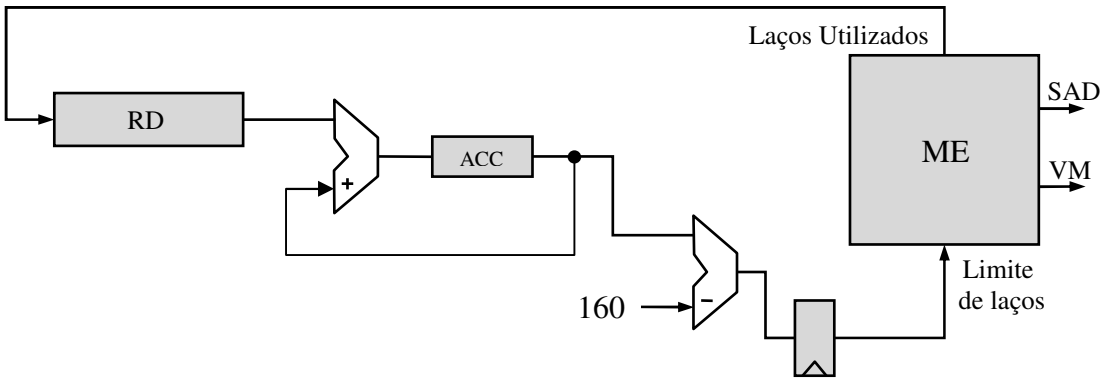



Figura 5.6: Controle de laços dinâmico

A inserção do CDL necessita de um pequeno acréscimo de hardware e não afeta o desempenho da arquitetura. O cálculo do número de laços disponível para o próximo vetor é realizado em 19 ciclos. A arquitetura do SDS analisa a restrição de laços após a realização do primeiro laço, o que acontece apenas depois de 94 ciclos, 60 do primeiro LDSP, e mais 34 para o primeiro laço. O CDL também garante o sincronismo da arquitetura do estimador com os demais blocos do compressor. Considerando que para o CDL disponibiliza um total de 160 laços, para cada conjunto de 16 vetores, e que para cada laço são gastos 34 ciclos de relógio, no máximo serão gastos 6848 ciclos para processar um conjunto de 16 vetores de movimento. Desta forma, é possível armazenar os resultados obtidos para cada conjunto de 16 vetores e garantir o sincronismo da ME com os demais blocos do compressor de vídeo.

## 5.4 Resultados de Síntese

A arquitetura do algoritmo SDS apresentada foi descrita em VHDL utilizando a ferramenta ISE 8.1i (XIILINX, 2007ª). O código VHDL foi sintetizado para o FPGA XC4VLX15 da família Virtex-4 da Xilinx (XILINX, 2007). A ferramenta ModelSim 6.1 (MODELSIM, 2007) foi utilizada para a etapa de simulação e validação da arquitetura. A Tabela 5.3 apresenta os resultados de síntese para os principais módulos da arquitetura do SDS.

Tabela 5.3: Resultados de síntese para a arquitetura do SDS na ferramenta ISE

| Módulo | LUTs | Slices | Slices FF | BRAMs | Freqüência (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 com CDL | 3890 | 2190 | 2379 | 32 | 184,19 |

Dispositivo: Virtex-4 XC4VLX15

Pode-se perceber que a arquitetura do SDS utiliza uma pequena quantidade de recursos de hardware do dispositivo. A arquitetura completa do SDS utiliza apenas 3890 LUTs e 2190 *Slices*. A utilização das memórias do dispositivo também é pequena, sendo apenas 32 BRAMs. A Tabela 5.2 também mostra que todos os principais blocos da arquitetura podem operar com freqüências de operação elevadas, sendo que a arquitetura completa atinge 184,19MHz.

## 5.5 Resultados de Desempenho

Considerando o resultado de freqüência de operação, apresentado na Tabela 5.2, e os resultados de número de ciclos de relógio necessários para gerar o vetor, apresentados anteriormente, é possível determinar a capacidade máxima de processamento da arquitetura do SDS.

Os resultados de desempenho para a arquitetura serão apresentados em três diferentes condições: melhor caso, caso médio e pior caso. No melhor caso, a

arquitetura utiliza apenas 88 ciclos de relógio para gerar o vetor de movimento. No caso médio, onde são considerados cerca de três laços como média de número iterações, são utilizados 190 ciclos. Para o pior caso, onde o CDL é considerado com um valor máximo de 10 laços para cada vetor, a arquitetura poderá utilizar até 428 ciclos de relógio. A tabela 5.4 apresenta o resultado de desempenho da arquitetura do SDS, em quadros por segundo, para diversos padrões de resolução de vídeo.

Tabela 5.4: Resultados de desempenho, em número de quadros por segundo, para a arquitetura do SDS em diversos padrões de vídeo

| **Padrão** | **Quadros por segundo** | | |
|---|---|---|---|
| | **Melhor caso** | **Caso médio** | **Pior caso** |
| QCIF (176 x 144) | 21142 | 9792 | 4347 |
| CIF (352 x 268) | 5286 | 2448 | 1087 |
| VGA (640 x 480) | 1744 | 808 | 359 |
| SDTV (720 x 480) | 1550 | 718 | 319 |
| HDTV (1280x720) | 436 | 202 | 90 |
| HDTV (1920 x 1080) | 258 | 120 | 53 |
| QHDTV (3840x2048) | 68 | 32 | 14 |

Avaliando os resultados apresentados na Tabela 5.4 pode-se perceber as elevadas taxas de processamento da arquitetura SDS, para todas as resoluções de vídeo apresentadas. A arquitetura foi especialmente projetada para atingir tempo real (30 quadros por segundo) para a resolução HDTV 1920x1080 (1080p). Esta é a resolução adotada como padrão pelo SBTVD (Sistema brasileiro de TV digital) (SBTVD, 2006). No pior caso, a arquitetura do SDS pode processar até 53 quadros por segundo neste formato. A Tabela 5.4 também mostra que a arquitetura do SDS pode operar em tempo real para todos os padrões de vídeo avaliados, exceto para a resolução QHDTV 3480x2048, onde no pior caso, o desempenho foi inferior a 30 quadros por segundo.

O resultado obtido para o pior caso garante que a arquitetura alcance o desempenho mínimo necessário para atingir tempo real de processamento em todos os padrões de vídeo avaliados. No entanto, o resultado obtido para o caso médio é o dado mais relevante para avaliar o desempenho da arquitetura do SDS. Para o caso médio, o desempenho da arquitetura é muito superior ao necessário para atingir tempo real, para todos os padrões. Considerando o padrão HDTV 1080p, a arquitetura de SDS pode processar cerca de 120 quadros por segundo. Esta taxa de processamento é quatro vezes maior do que a necessária para processar esta resolução de vídeo em tempo real. Considerando o caso médio, a arquitetura do SDS é capaz de processar até mesmo vídeos na resolução QHDTV em tempo real, pois atingiu uma taxa de processamento de 32 quadros por segundo nesta resolução. Os resultados obtidos para o melhor caso valem apenas para determinar a taxa de processamento máxima da arquitetura. Esta condição depende exclusivamente dos dados de entrada da arquitetura, e não se pode considerar esta taxa de processamento como resultado absoluto de desempenho.

Pode-se notar que, devido à alta freqüência de operação obtida pela arquitetura do SDS, seus resultados de desempenho estão muito acima do mínimo necessário para

processar os vídeos em tempo real para praticamente todas as resoluções. A Tabela 5.5 apresenta a freqüência mínima necessária para atingir a taxa de processamento para tempo real nas diversas resoluções avaliadas. A segunda coluna da Tabela 5.5 apresenta os resultados de freqüência de operação mínima para a arquitetura do SDS considerando o número de ciclos de relógio utilizados no pior caso. A terceira coluna utiliza o caso médio para determinar a freqüência mínima de operação.

Os resultados da Tabela 5.5 mostram a eficiência da arquitetura do algoritmo SDS, que pode operar com freqüências de operação muito baixas para vídeos com as menores resoluções. Considerando o pior caso, a arquitetura do SDS necessita de apenas 1,27 MHz para processar vídeos QCIF em tempo real. Mesmo para resoluções altas, como HDTV 1280x720 (720p), a freqüência de operação necessária não é elevada, ficando abaixo dos 62 MHz.

Tabela 5.5: Freqüência mínima para atingir tempo real a 30 quadros por segundo, em diversos padrões, para a arquitetura do SDS

| **Padrão** | **Freqüência mínima pior caso (MHz)** | **Freqüência mínima caso médio (MHz)** |
|---|---|---|
| QCIF (176 x 144) | 1,27 | 0,56 |
| CIF (352 x 268) | 5,08 | 2,26 |
| VGA (640 x 480) | 15,41 | 6,84 |
| SDTV (720 x 480) | 17,33 | 7,69 |
| HDTV (1280x720) | 61,63 | 27,36 |
| HDTV (1920 x 1080) | 104,00 | 46,17 |
| QHDTV (3840x2048) | 394,44 | 175,10 |

Quando o pior caso for considerado na avaliação do desempenho, se estará subestimando o desempenho da arquitetura, tendo em vista que, na média, a arquitetura utiliza um número bem menor de ciclos para gerar os vetores do que o considerado no pior caso. Considerando o número de ciclos de relógio utilizados no caso médio, os valores de freqüência mínima de operação são ainda menores. Neste caso, a arquitetura do SDS necessita de apenas 27,36 MHz para processar vídeos HDTV 720p em tempo real, e de aproximadamente 46,17 MHz para processar vídeos HDTV 1080p.

Estas avaliações das freqüências mínimas de operação da arquitetura do SDS, para várias resoluções de vídeo, mostram que, dependendo da resolução do vídeo que se deseja tratar, a arquitetura pode ser sintetizada para FPGAs mais lentos e, até mesmo, para tecnologias mais ultrapassadas. A arquitetura do SDS pode operar com freqüências menores do que as apresentadas na Tabela 5.3 para praticamente todas as resoluções de vídeo, exceto QHDTV. Isto também pode contribuir para a diminuição do consumo de energia. Esta é uma importante característica quando é considerada a aplicação desta arquitetura para dispositivos móveis, como celulares, por exemplo, que necessitam de arquiteturas com baixo consumo e que, em geral, tratam apenas vídeos de baixa resolução como QCIF e CIF.

# 6 A ARQUITETURA DO QSDS (*QUARTER SUB-SAMPLED DIAMOND SEARCH*)

A arquitetura para o algoritmo QSDS é totalmente baseada na arquitetura desenvolvida para o algoritmo SDS. O *template* arquitetural é o mesmo apresentado na Figura 5.1. O tamanho de bloco utilizado continua sendo de 16x16 amostras e o critério de distorção também é o SAD. Os blocos que compões a arquitetura, como as UPs (Figura 5.2) e o comparador (Figura 5.3) são exatamente os mesmo utilizados para a arquitetura do SDS. As diferenças estão no módulo de controle e no módulo de memória, e serão detalhadas nesta seção.

O funcionamento da arquitetura do QSDS é muito semelhante ao da arquitetura do SDS, no entanto, com a sub-amostragem de *pixel* de 4:1, o bloco sub-amostrado passa a ter apenas oito linhas com oito amostras cada. Desta forma, as UPs continuam processando uma linha do bloco candidato em paralelo, só que apenas oito iterações são necessárias para processar todos os SADs do bloco candidato.

A arquitetura do QSDS também possui os mesmo quatro modos de operação da arquitetura do SDS, sendo que o mesmo número de blocos candidatos é avaliado em cada etapa do algoritmo. A unidade de controle também é muito semelhante, no entanto, ela é mais simples, pois menos iterações nas UPs, e conseqüentemente menos acessos a memória, devem ser gerenciados.

## 6.1 Organização da memória

A arquitetura do QSDS possui 15 memórias distintas e segue a mesma estrutura da arquitetura do SDS apresentada na Figura 5.5. A diferença está no tamanho de cada uma delas, que para a QSDS é menor. A ML da arquitetura do QSDS possui 16 palavras de 128 bits, ao contrário da arquitetura do SDS que possui 24 palavras. As memórias MBC, MBCS e MBA possuem apenas oito palavras de 64 bits. Esta redução no tamanho das memórias é obtida com a sub-amostragem de *pixel* de 4:1, que reduz o tamanho do bloco original pela metade nas duas dimensões. Deste modo, cada bloco de 16x16 se transforma em um bloco de apenas 8x8 amostras.

O funcionamento da memória é o mesmo apresentado na seção 5.1 para a arquitetura do SDS. Como o tamanho da linha se mantém, o bloco Particionador também é o mesmo. A unidade de controle local das memórias é muito semelhante, no entanto, ela também é mais simples, pois precisa controlar um número menor de operações de leituras e escritas.

Com o aumento da sub-amostragem, e a conseqüente redução do tamanho das memórias, o total de bits de memória utilizados pela arquitetura do QSDS é menor do que o utilizado pela arquitetura do SDS. Para a ML são utilizadas 16 linhas de 128 bits cada, o que resulta em 2048 bits de memória. Para cada MBC, MBCS ou MBA são utilizadas oito linhas de 64 bits cada, o que resulta em 512 bits de memória para cada memória de bloco. No total, para as 15 memórias utilizadas, a arquitetura do QSDS utiliza apenas 9216 bits de memória. A arquitetura do QSDS reduz em mais de 43% a quantidade de memória necessária, na comparação com a arquitetura do SDS.

## 6.2 Avaliação do Desempenho

A arquitetura do algoritmo QSDS necessita de um número de ciclos de relógio menor, comparado ao necessário para a arquitetura SDS. Devido ao aumento da sub-amostragem e a conseqüente redução do tamanho das memórias, a arquitetura do QSDS utiliza apenas 26 ciclos de relógio para preencher todas as memórias e iniciar o processo de cálculo de SADs. 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 utiliza 42 ciclos de relógio 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 utiliza 62 ciclos de relógio para determinar o vetor de movimento para um bloco.

Os mesmos 10 ciclos de latência, para escrever uma linha nas MBCs, são necessários para a arquitetura do QSDS. Isto ocorre, pois o número de memórias de bloco permanece o mesmo da arquitetura do SDS. Desta forma, a arquitetura do QSDS utiliza 26 ciclos de relógio para cada LDSP. Considerando o mesmo exemplo apresentado para a arquitetura do SDS, com um número de repetições do LDSP igual a cinco, temos um total de 192 ciclos de relógio utilizados pela arquitetura do QSDS.

A redução do tamanho das memórias e do número de iterações utilizados pelas UPs resulta em um aumento do desempenho da arquitetura, que precisa de um número menor de ciclos de relógio para gerar os vetores, se comparada à arquitetura do SDS. Cada LDSP da arquitetura do QSDS utiliza 26 ciclos, este número é menor do que os 34 ciclos utilizados pela arquitetura do SDS. Desta forma, quanto mais longa for à busca (maior número de repetições do LDSP), maior será a vantagem de desempenho obtida pela arquitetura do QSDS sobre a arquitetura do SDS.

## 6.3 Controle Dinâmico de Laços

Mais uma bateria de testes em software foi desenvolvida para avaliar o impacto do controle de laços, desta vez para o algoritmo QSDS. Os resultados médios de redução do erro absoluto e ganho PSNR, para o algoritmo QSDS original e com controle de laços estão apresentados na Tabela 6.1. Foram geradas diversas versões do algoritmo QSDS com CDL e também com restrição fixa. Os resultados apresentados são os valores médios obtidos para as 10 seqüências de vídeo utilizadas, e considerando o quadro inteiro como área de pesquisa.

Tabela 6.1: Resultados médios de qualidade para o algoritmo QSDS, QSDS com controle dinâmico de laços e QSDS com controle fixo de laços

| Algoritmo | Redução do erro (%) | PSNR (dB) |
|---|---|---|
| QSDS | 43,67 | 26,033 |
| QSDS + CDL de 20 laços | 43,64 | 26,030 |
| QSDS + restrição de 20 laços | 43,51 | 26,020 |
| QSDS + CDL de 10 laços | 43,06 | 25,987 |
| QSDS + restrição de 10 laços | 42,63 | 25,945 |
| QSDS + CDL de 5 laços | 40,45 | 25,752 |
| QSDS + restrição de 5 laços | 40,00 | 25,694 |

O algoritmo QSDS também apresenta um ganho considerável com o aumento no número de laços da restrição. A diferença entre os resultados de redução do erro para as versões com restrição fixa de cinco e 20 laços é de 3,66%. Com o CDL, os resultados de redução do erro e ganho PSNR são melhores, considerando o mesmo número de laços na restrição. No entanto, ao contrário do que foi apresentado para o algoritmo SDS, os resultados obtidos pelo CDL de 10 laços não são melhores do que os obtidos pela restrição fixa de 20 laços. Devido a isto, o CDL para o algoritmo QSDS foi desenvolvido para 20 laços. As perdas obtidas para o QSDS com CDL de 20 laços são praticamente desprezíveis. Em relação à diminuição do erro, essas perdas foram de apenas 0,03%, para o ganho PSNR a perda foi de apenas 0,003dB.

O CDL desenvolvido para a arquitetura do QSDS possui o mesmo modelo arquitetural apresentado na Figura 5.6. A única diferença está no número máximo de laços permitido para cada vetor, que para a arquitetura do QSDS foi determinado em 20 laços. Este aumento foi possível, pois o QSDS utiliza menos ciclos para gerar cada vetor, assim, estes ciclos poupados podem ser empregados para um maior número de laços, quando o algoritmo necessitar, melhorando a qualidade final do processo de estimação sem afetar o desempenho final da arquitetura.

Os resultados da Tabela 4.7 mostram que o algoritmo QSDS utiliza em média apenas 2,96 laços por vetor, desta forma, considerando um valor máximo de 20 laços, em média sobram mais de 17 laços a cada vetor. Considerando um histórico de 16 vetores, temos que o valor máximo de laços disponível é de 320. Com este número de laços é possível atingir uma área de pesquisa máxima de 1300x1300 *pixels* para a arquitetura do QSDS. Se considerarmos uma restrição fixa de 20 laços por vetor a área máxima de pesquisa será de apenas 100x100 *pixels*.

Podemos perceber que o controle dinâmico de laços praticamente não insere perdas na qualidade dos vetores gerados pela arquitetura do QSDS. Devido ao aumento no número de laços médio para cada vetor, a diferença entre os resultados de qualidade para o QSDS e para as versões com controle de laços são ainda menores, se comparados às diferenças para o SDS. O controle de laços para a arquitetura do QSDS também não interfere em nada nos resultados de desempenho, e pode garantir o sincronismo da arquitetura com os demais blocos do compressor de vídeo. Considerando o histórico do CDL com 16 vetores, a arquitetura do QSDS utiliza um número máximo de ciclos igual a 9312, para processar um conjunto de 16 vetores.

## 6.4 Resultados de Síntese

A arquitetura do algoritmo QSDS 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 validade utilizando a ferramenta ModelSim 6.1. A Tabela 6.2 apresenta os resultados de síntese para os principais módulos da arquitetura do QSDS.

Os resultados de síntese apresentados na Tabela 6.2 são muito parecidos com os resultados obtidos para a arquitetura do SDS, no entanto, os resultados de utilização de LUTs, *Slices* e *Slices* FF são menores que os apresentados na Tabela 5.2, para o algoritmo SDS. Esta diminuição na utilização dos recursos de hardware se deve principalmente a redução to tamanho dos acumuladores, que precisam armazenar apenas sete iterações. Os blocos de controle das memórias, bem como a unidade 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 utilizado pela arquitetura do SDS. Isto acontece devido às características de implementação das BRAMs do dispositivo, que possuem 256 palavras de 64 bits. A arquitetura do QSDS utiliza memórias com menos palavras, mas com o mesmo tamanho de palavra utilizado pela arquitetura do SDS. Assim, a mesma quantidade de BRAMs precisa ser empregada, mesmo que estejam sendo subutilizadas.

Tabela 6.2: Resultados de síntese para a arquitetura do QSDS na ferramenta ISE

| Módulo | LUTs | Slices | Slices FF | BRAMs | Freqüência (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 com CDL | 3610 | 2007 | 2086 | 32 | 213,37 |

Dispositivo: Virtex-4 XC4VLX15

Todos os módulos da arquitetura do QSDS podem operar com altas freqüências de operação. As simplificações dos blocos de controle e memórias proporcionaram um aumento na freqüência de operação da arquitetura do QSDS, que pode operar a 213,37 MHz. Com este aumento na freqüência de operação, aliado a menor quantidade de ciclos necessários para gerar um vetor, a arquitetura do QSDS pode obter taxas de processamento superiores às obtidas pela arquitetura do SDS.

## 6.5 Resultados de Desempenho

A capacidade máxima de processamento da arquitetura do QSDS também será apresentada para três condições: melhor caso, caso médio e pior caso. Considerando os dados de freqüência de operação, e de número de ciclos de relógio para cada condição, podemos determinar a taxa máxima de processamento da arquitetura do QSDS em número de quadros por segundo. Para o melhor caso são gastos 62 ciclos de relógio para

gerar um vetor de movimento. Para o caso médio, onde foram considerados três laços como número médio de iterações, são utilizados 140 ciclos para gerar um vetor. No pior caso, onde o valor máximo de laços considerado é de 20, a arquitetura do QSDS utiliza 582 ciclos de relógio para gerar o vetor de movimento. A Tabela 6.3 apresenta o resultado de desempenho da arquitetura do QSDS, em quadros por segundo, para diversos padrões de resolução de vídeo.

Os resultados da Tabela 6.3 mostram que a arquitetura do QSDS pode atingir altas taxas de processamento, para todos os padrões de vídeo avaliados. Mais uma vez, a arquitetura foi projetada para atingir tempo real a 30 quadros por segundo para a resolução HDTV 1080p. No pior caso, a arquitetura do QSDS pode processar 45 quadros por segundo neste formato. A arquitetura do QSDS atinge taxas de processamento superiores a 30 quadros por segundo, no pior caso, para todas as resoluções de vídeo, exceto QHDTV. Isto garante que a arquitetura do QSDS pode operar em tempo real para todos os padrões de vídeo menores que QHDTV. Os resultados de taxa de processamento para o pior caso são menores dos que os apresentados para o SDS, devido ao aumento no número de laços do CDL de 10 para 20 laços. Os resultados para o caso médio são ainda mais favoráveis, sendo que para a resolução HDTV 1080p, a arquitetura do QSDS pode processar até 188 quadros por segundo. Esta taxa de processamento é mais de seis vezes superior à taxa necessária para processar esta resolução de vídeo em tempo real. Considerando o caso médio, a arquitetura do QSDS pode processar até 50 quadros QHDTV por segundo, este desempenho é superior ao necessário para atingir tempo real de processamento em vídeos desta resolução.

Tabela 6.3: Resultados de desempenho, em número de quadros por segundo, para a arquitetura do QSDS em diversos padrões de vídeo

| **Padrão** | **Quadros por segundo** | | |
|---|---|---|---|
| | **Melhor caso** | **Caso médio** | **Pior caso** |
| QCIF (176 x 144) | 34762 | 15395 | 3703 |
| CIF (352 x 268) | 8691 | 3849 | 926 |
| VGA (640 x 480) | 2868 | 1270 | 306 |
| SDTV (720 x 480) | 2549 | 1129 | 272 |
| HDTV (1280x720) | 717 | 318 | 76 |
| HDTV (1920 x 1080) | 425 | 188 | 45 |
| QHDTV (3840x2048) | 112 | 50 | 12 |

A Tabela 6.4 apresenta a freqüência mínima necessária para a arquitetura do QSDS atingir a taxa de processamento para tempo real nos diversos padrões avaliados. A segunda coluna da Tabela 6.4 apresenta os resultados de freqüência de operação mínima para a arquitetura do QSDS, considerando o número de ciclos de relógio utilizados no pior caso. A terceira coluna utiliza o caso médio para determinar a freqüência de operação da arquitetura do QSDS.

Os resultados da Tabela 6.4 mostram que a arquitetura do QSDS pode operar com freqüências de operação muito baixas para os padrões de vídeo de menor resolução. Considerando o pior caso, a arquitetura do QSDS necessita de apenas 1,73MHz para

processar vídeos QCIF em tempo real. Para resoluções mais altas, como SDTV 720x480, a freqüência de operação mínima no pior caso é de 23,57 MHz. Estes valores de freqüência mínima no pior caso para a arquitetura do QSDS são maiores dos que as apresentadas na Tabela 5.4, para a arquitetura do SDS. Mais uma vez isso ocorre devido ao aumento no número de laços do CDL, utilizado na arquitetura do QSDS. Desta forma, a freqüência necessária para a arquitetura do QSDS processar 30 quadros por segundo, no pior caso, é maior do que a freqüência necessária para a arquitetura do SDS.

Tabela 6.4: Freqüência mínima para atingir tempo real a 30 quadros por segundo, em diversos padrões, para a arquitetura do QSDS

| **Padrão** | **Freqüência mínima pior caso (MHz)** | **Freqüência mínima caso médio (MHz)** |
|---|---|---|
| QCIF (176 x 144) | 1,73 | 0,41 |
| CIF (352 x 268) | 6,91 | 1,66 |
| VGA (640 x 480) | 20,95 | 5,04 |
| SDTV (720 x 480) | 23,57 | 5,67 |
| HDTV (1280x720) | 83,81 | 20,16 |
| HDTV (1920 x 1080) | 141,43 | 34,02 |
| QHDTV (3840x2048) | 536,37 | 129,02 |

As freqüências mínimas da arquitetura do QSDS, para o caso médio, são menores que a as obtidas para o SDS. Neste caso, são considerados três laços, como número médio de laços, para as duas arquiteturas. Assim, como a arquitetura do QSDS utiliza um número menor de ciclos de relógio a cada laço, as freqüências mínimas necessárias para cada padrão de vídeo são menores que as necessárias para a arquitetura do SDS. Para processar vídeos HDTV 720p a arquitetura do QSDS necessita de apenas 20,16 MHz, para processar vídeos HDTV 1080p, a freqüência de operação mínima é de apenas 34,02 MHz.

# 7 COMPARAÇÕES ENTRE TRABALHOS RELACIONADOS

Neste capítulo será apresentada uma comparação entre diversas características de várias arquiteturas de ME. As comparações são divididas em duas etapas, na primeira, as arquiteturas desenvolvidas neste trabalho (SDS e QSDS), são comparadas com arquiteturas desenvolvidas anteriormente pelo nosso grupo de pesquisa, na UFRGS e na UFPel. Na segunda etapa, as arquiteturas do SDS e QSDS são comparadas com arquiteturas publicadas na literatura, que não foram desenvolvidas pelo nosso grupo de pesquisa.

A comparação com os trabalhos desenvolvidos no grupo será bem detalhada, pois temos condições de gerar todos os resultados necessários para as comparações, como: tamanho de área de pesquisa utilizado, redução do erro absoluto, ganho PSNR, recursos de hardware utilizados, desempenho etc. Nesta etapa, serão comparados os resultados obtidos pelas arquiteturas apresentadas neste trabalho com mais três arquiteturas desenvolvidas anteriormente.

A comparação com trabalhos relacionados, publicados na literatura, será menos detalhada, devido à dificuldade de se extrair certos resultados dos artigos publicados. Alguns artigos não apresentam, por exemplo, o tamanho da área de pesquisa utilizada ou qual o FPGA foi utilizado para a síntese da arquitetura. Muitos trabalhos apresentam somente resultados de implementação da arquitetura em *standard cells*. Devido a isto, foram geradas versões *standard cells* das arquiteturas do SDS e QSDS para tornar possíveis as comparações.

## 7.1 Comparações com Trabalhos Anteriores

Algumas arquiteturas de ME já foram desenvolvidas anteriormente no nosso grupo de pesquisa. No entanto, todas elas foram desenvolvidas para o algoritmo FS, seja com ou sem sub-amostragem. Todas as arquiteturas trabalham com blocos de tamanho 16x16 e utilizam o SAD como critério de distorção. A primeira arquitetura desenvolvida foi para o algoritmo FS sem sub-amostragem (AGOSTINI, 2007), com uma área de pesquisa de 32x32 amostras. No intuito de reduzir a quantidade de recursos de hardware utilizados pela arquitetura do FS, foi desenvolvida uma segunda solução, para o algoritmo FS com sub-amostragem de *pixel* de 4:1 (FS + PS 4:1) (PORTO, 2007). Uma terceira arquitetura foi desenvolvida para o algoritmo FS com sub-amostragem de *pixel* e de bloco de 4:1 (FS + BS 4:1 + PS 4:1) (AGOSTINI, 2007).

A Tabela 7.1 apresenta os resultados obtidos para as três arquiteturas desenvolvidas anteriormente e também para as arquiteturas do SDS e QSDS. São apresentados os resultados de diversos critérios de avaliação como, qualidade, desempenho e custo da implementação das arquiteturas em hardware.

Tabela 7.1: Comparação entre as arquiteturas FS, FS + PS 4:1, FS + BS 4:1 + PS 4:1, SDS e QSDS

| **Critério** | **Arquitetura** | | | | |
|---|---|---|---|---|---|
| | **FS** | **FS + PS 4:1** | **FS + BS 4:1+ PS 4:1** | **SDS** | **QSDS** |
| Área de pesquisa (amostras) | 32x32 | 64x64 | 64x64 | 660x660 | 1300x1300 |
| Redução do ERRO (%) | 46,81 | 50,20 | 44,65 | 48,58 | 43,64 |
| PSNR (dB) | 27,395 | 27,244 | 20,472 | 27,195 | 26,030 |
| LUTs | 37561 | 31063 | 30492 | 3890 | 3610 |
| FPGA | Virtex2P | Virtex2P | Virtex2P | Virtex-4 | Virtex-4 |
| Bits de Memória | 10240 | 33280 | 8704 | 17408 | 9216 |
| Freqüência (MHz) | 172,10 | 123,42 | 123,05 | 184,19 | 213,37 |
| Ciclos p/ Vetor | 333 | 2615 | 653 | 190 | 140 |
| Quadros p/ segundo (HDTV 1080p) | 63 | 5 | 23 | 120 | 188 |

Os tamanhos das áreas de pesquisa variam muito entre as cinco arquiteturas de ME apresentadas, sendo que a arquitetura do algoritmo FS possui a menor área, 32x32 amostras. A maior área de pesquisa apresentada é a da arquitetura do algoritmo QSDS. Mesmo assim, a arquitetura do algoritmo FS apresenta a maior utilização de LUTs dentre todas as soluções apresentadas, cerca de 37,5K LUTs. As arquiteturas do QSDS e SDS apresentam o menor resultado de utilização de LUTs, com cerca de 3,6K e 3,8K respectivamente. A redução no consumo de LUTs para as arquiteturas do SDS e QSDS é de mais de nove vezes, na comparação com a arquitetura do FS. Em relação à arquitetura do FS com PS 4:1, que utiliza uma área de pesquisa de 64x64 amostras e consome cerca de 31K LUTs, a diferença no consumo de LUTs é de aproximadamente oito vezes.

A arquitetura para o algoritmo FS com PS 4:1 é solução que utiliza a maior quantidade de bits de memória, aproximadamente 33,2Kbits. Isto ocorre, pois sua área de busca é quatro vezes maior do que a utilizada pela arquitetura do FS, e a sub-amostragem de pixel reduz apenas a quantidade de memória utilizada pelo bloco atual. Desta forma, toda a área de pesquisa de 64x64 amostras deve ser carregada na memória. A solução com BS e PS 4:1 reduz em aproximadamente quatro vezes a quantidade de memória utilizada pela arquitetura que utiliza apenas PS 4:1. Isso ocorre, pois os blocos descartados com a sub-amostragem de blocos não precisam ser armazenados na memória. A arquitetura do SDS, apesar de tratar de uma área de busca de 660x660, utiliza menos memória que a solução para o FS com PS 4:1. A solução para o QSDS,

mesmo atuando sobre uma área de pesquisa de 1300x1300, utiliza apenas 9,2Kbits de memória, e só perde para a arquitetura do FS com BS e PS 4:1 em economia de recursos de memória.

Os resultados de redução do erro absoluto variam entre 43,64%, para a arquitetura do QSDS, e 50,20% para a solução com FS e PS 4:1. A arquitetura do algoritmo FS não possui o melhor resultado de diminuição do erro absoluto, basicamente devido a sua pequena área de pesquisa. A arquitetura do FS com PS 4:1 utiliza uma área quatro vezes maior, o que torna possível gerar resultados de diminuição de erro melhores do que os gerados pela solução com o FS. O Segundo melhor resultado em termos de redução do erro foi obtido pela arquitetura do SDS, que reduz o erro em cerca de 48,58%. A arquitetura do FS com BS e PS 4:1, apesar de também trabalhar com uma área de pesquisa quatro vezes maior do que a utilizada pelo algoritmo FS, possui o segundo pior resultado de diminuição do erro devido à sub-amostragem de blocos em 4:1.

Apesar de não obter o melhor resultado em relação à diminuição do erro absoluto, a arquitetura do FS apresenta o maior ganho PSNR. Isto ocorre devido aos diferentes critérios utilizados para calcular o erro (SAD) e o PSNR (MSE). A arquitetura do FS apresenta um ganho PSNR de 27,395dB, enquanto a solução com PS 4:1 apresenta o segundo melhor resultado, com uma diferença de 0,151dB. A arquitetura do SDS também obteve um bom resultado de ganho PSNR, ficando apenas 0,2 dB abaixo da solução, para o FS. O pior resultado foi obtido pela arquitetura do FS com BS e PS 4:1, isto mostra que a sub-amostragem de bloco afeta mais o resultado de ganho PSNR do que os resultados de redução do erro absoluto.

Todas as arquiteturas apresentam altas freqüências de operação, variando de 123,05MHz para a arquitetura do FS com BS e PS 4:1, até 213,37MHz na solução para o QSDS. O número de ciclos utilizados para gerar cada vetor também varia muito entre as soluções arquiteturais apresentadas. Dentre as soluções desenvolvidas anteriormente, a arquitetura do algoritmo FS utiliza a menor quantidade de ciclos de relógio para gerar cada vetor, apenas 333 ciclos. A arquitetura para o FS com PS 4:1 utiliza 2615 ciclos para gerar cada vetor, este aumento significativo no número de ciclos acontece devido à área de pesquisa ser quatro vezes maior do que a utilizada na arquitetura do FS, e também devido à redução do paralelismo das unidades de processamento (UP), o que aumentou o número de iterações necessárias. A arquitetura para o FS com BS e PS 4:1 reduz em quatro vezes o número de ciclos utilizado pela solução para o FS com PS 4:1 devido à sub-amostragem de blocos de 4:1. As arquiteturas do SDS e QSDS são as soluções que utilizam a menor quantidade de ciclos de relógio por vetor de movimento. Os valores apresentados na Tabela 7.1 são os valores médios de número de ciclos para as duas arquiteturas.

Com os resultados de freqüência de operação e número de ciclos necessários por vetor de movimento, podemos determinar o número de quadros HDTV 1080p que cada arquitetura pode processar por segundo. Dentre as soluções anteriores, apenas a arquitetura do FS foi desenvolvida para trabalhar com vídeos HDTV, e por isso apenas ela pode tratar esta resolução de vídeo em tempo real. As arquiteturas para o FS com PS 4:1 e para o FS com BS e PS 4:1 foram desenvolvidas para tratar vídeos SDTV, portanto não suportam vídeos HDTV em tempo real. A arquitetura do FS com PS 4:1 pode processar apenas cinco quadros HDTV 1080p por segundo, enquanto a solução para o FS com BS e PS 4:1 pode processar até 23 quadros por segundo. As arquiteturas do SDS e QSDS foram desenvolvidas para tratar vídeos de alta resolução, e podem processar respectivamente 120 e 188 quadros HDTV 1080p por segundo. Mais uma vez

estes são os resultados para o caso médio, sendo que para o pior caso, as duas arquiteturas são capazes de processar mais de 30 quadros por segundo, atendendo os requisitos de operação em tempo real.

O gráfico ilustrado na Figura 7.1 apresenta os principais resultados da Tabela 7.1. Analisando o gráfico podemos relacionar melhor os resultados de qualidade, consumo de recursos de hardware e desempenho, pra cada uma das arquiteturas apresentadas. Os resultados de ganho PSNR são muito similares para as arquiteturas do FS, FS com PS 4:1 e SDS, apenas as soluções para o FS com BS e PS 4:1 e para o QSDS apresentam resultado um pouco inferiores. Para os resultados de utilização de LUTs, podemos perceber que as arquiteturas do SDS e QSDS utilizam uma quantidade significativamente menor do que as demais arquiteturas. Mesmo assim, os resultados de desempenho são muito superiores, como pode ser visto no gráfico da Figura 7.1.

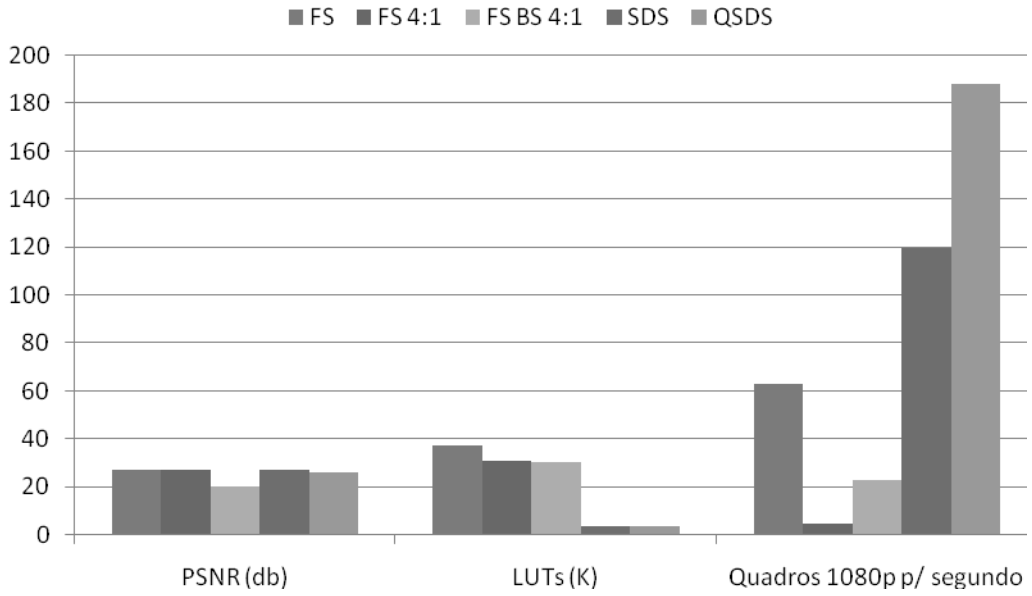



Figura 7.1: Resultados de qualidade, custo em hardware e desempenho para as arquiteturas avaliadas

A Figura 7.2 apresenta um gráfico que ilustra a quantidade de LUTs, utilizadas por cada solução arquitetural, para cada dB obtido no ganho PSNR. Este gráfico tenta demonstrar qual das arquiteturas apresenta a melhor relação entre consumo de recursos de hardware e qualidade dos vetores de movimento gerados, sendo que quanto maiores os valores apresentados no gráfico, menos eficiente é a arquitetura nesta relação.

O gráfico da Figura 7.2 mostra que as arquiteturas desenvolvidas para o algoritmo FS sejam com ou sem sub-amostragem, apresentam os piores resultados nesta relação consumo de recursos versus qualidade dos vetores. A pior relação é apresentada pela arquitetura do FS com BS e PS 4:1, que possui o menor ganho PSNR dentre todas as soluções e a utilização de LUTs é muito parecida com a arquitetura do FS com PS 4:1. A arquitetura do FS, mesmo possuindo o melhor resultado de ganho PSNR, não apresenta uma boa relação. Isto ocorre devido à grande utilização de recursos de hardware, a maior dentre as soluções apresentadas.

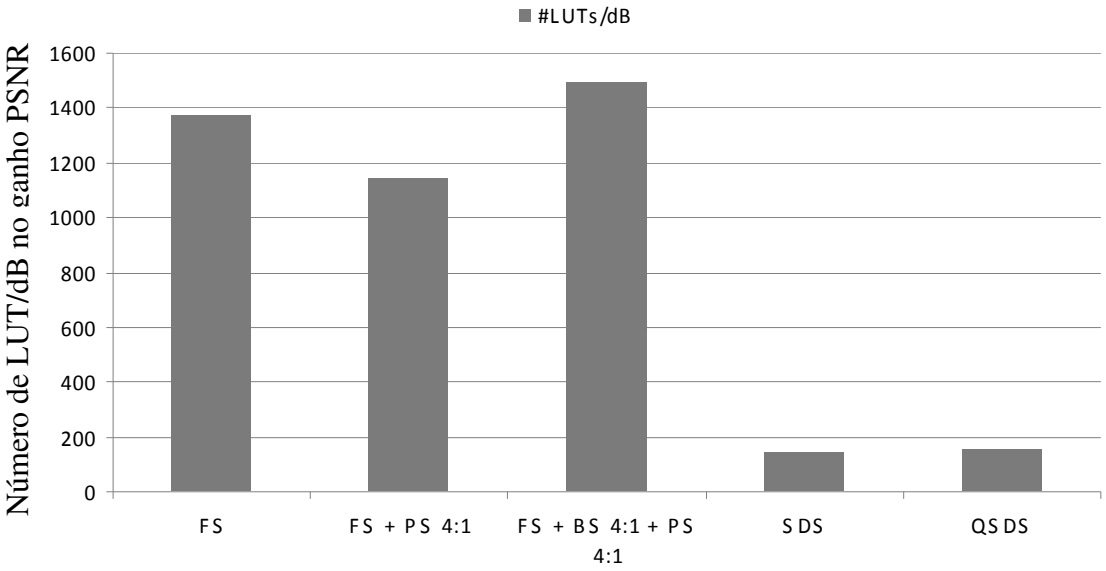



Figura 7.2: Relação de utilização de LUT por dB ganho no PSNR para cada arquitetura de ME

As arquiteturas do SDS e QSDS apresentam as melhores relações entre utilização de recursos de hardware e qualidade dos vetores gerados. Seus resultados são até 10 vezes melhores do que os obtidos pelas arquiteturas baseadas no algoritmo FS. O melhor resultado foi obtido pela arquitetura do SDS, pois seu resultado de ganho PSNR é superior ao obtido na solução para o QSDS, e o número de LUTs utilizadas pelas duas arquiteturas é muito semelhante.

## 7.2 Comparações com Trabalhos Relacionados da Literatura

Nesta seção serão comparados os resultados obtidos palas arquiteturas desenvolvidas neste trabalho com soluções apresentadas na literatura. Não foram encontrados muitos trabalhos com arquitetura de hardware para algoritmos rápidos, por isso serão apresentadas comparações com arquiteturas distintas. As comparações serão divididas em duas partes, na primeira vamos comparar as soluções desenvolvidas neste trabalho com outras também desenvolvidas para FPGAs. Na segunda parte, vamos apresentar os resultados das implementações em s*tandard cells* das arquiteturas do SDS e QSDS, e compará-las com alguns trabalhos publicados na literatura com desenvolvimento em *standard cells*.

### 7.2.1 Comparação com Soluções em FPGA

Os resultados obtidos com a implementação em FPGA para as arquiteturas do SDS e QSDS serão comparados com soluções para o algoritmo FS, com e sem sub-amostragem, e para soluções com algoritmos rápidos. As arquiteturas apresentadas em (MOHAMMADZADEH, 2005), (LOUKIL, 2004) e (ROMA, 2003) foram desenvolvidas para o algoritmo FS sem sub-amostragem. A solução apresentada em (LARKIN, 2006) utiliza o algoritmo FS com sub-amostragem de *pixel* de 4:1 aliada a uma técnica de terminação rápida, onde um valor de SAD é estipulado como *threshold* e toda vez que o valor de SAD encontrado é inferior a este valor pré-determinado a busca é interrompida. Uma solução em FPGA foi encontrada para o algoritmo rápido FTS (*Flexible Triangle Search*) (REHAN, 2006). A Tabela 7.2 apresenta os resultados

comparativos entre estas arquiteturas. São apresentados os resultados de utilização de recursos, freqüência máxima de operação, desempenho em número de quadros HDTV 1080p por segundo, bem como a família do FPGA utilizado em cada solução.

Tabela 7.2: Comparação entre as arquiteturas SDS, QSDS e soluções encontradas na literatura para implementações em FPGA

| Solução | FPGA | CLBs | Freqüência (MHz) | HDTV Quadros/s |
|---|---|---|---|---|
| Mohammadzadeh (2005) | Spartan 2 | 939 | 109,8 | 1,6 |
| Loukil (2004) | Stratix | 1654 | 103,8 | 3 |
| Roma (2003) | Virtex E | 29430 | 76,1 | 12,4 |
| Larkin (2006) | Virtex 2 | - | 120,0 | *24 |
| Rehan (2006) | Spartan 3 | 6142 | 74,0 | 45 |
| SDS | Virtex 4 | 507 | 184,1 | *120 |
| QSDS | Virtex 4 | 501 | 213,3 | *188 |

*Desempenho médio

As soluções apresentadas em (LOUKIL, 2004), (ROMA, 2003) e (LARKIN, 2006) utilizam uma área de pesquisa de 32x32 amostras, já as arquiteturas apresentadas em (MOHAMMADZADEH, 2005) e (REHAN, 2006) não informam o tamanho da área de pesquisa utilizada. A unidade de medida de consumo de recursos de hardware utilizada para a comparação é o número de CLBs (*Configurable Logic Block*) utilizadas. A ferramenta ISE 8.1i não apresenta esta informação de maneira clara, no entanto, podemos estimar a utilização das CLBs através da informação de número de *slices* utilizados. Cada CLB possui quatro *slices*, assim, os valores de número de CLBs utilizados para as arquiteturas do SDS e QSDS são estimados com sendo a quarta parte do valor de *slices* consumidos, apresentados nas Tabelas 5.2 e 6.2. As arquiteturas do SDS e QSDS apresentam o menor consumo de recursos de hardware dentre as soluções apresentadas, mesmo operando sobre área de pesquisa com tamanhos muito superiores.

As freqüências máximas de operação das arquiteturas desenvolvidas neste trabalho são superiores a todas as demais arquiteturas apresentadas, assim como os resultados de desempenho. Dentre as soluções encontradas na literatura, apenas a arquitetura desenvolvida por Rehan (2006) pode processar vídeos HDTV 1080p em tempo real, pois pode processar 45 quadros por segundo desta resolução de vídeo. O resultado de desempenho apresentado em (LARKIN, 2006), assim como os resultados do SDS e QSDS são referentes ao desempenho médio das arquiteturas.

Os resultados mostram a eficiência das arquiteturas apresentadas neste trabalho, que podem aliar um baixo consumo de recursos de hardware com altas taxas de processamento. A Figura 7.3 apresenta um gráfico relacionando os resultados de desempenho e consumo de recursos de hardware para as arquiteturas em FPGA avaliadas. A solução apresentada em (LARKIN, 2006) não aparece no gráfico, pois não apresenta o seu resultado de utilização de CLBs.

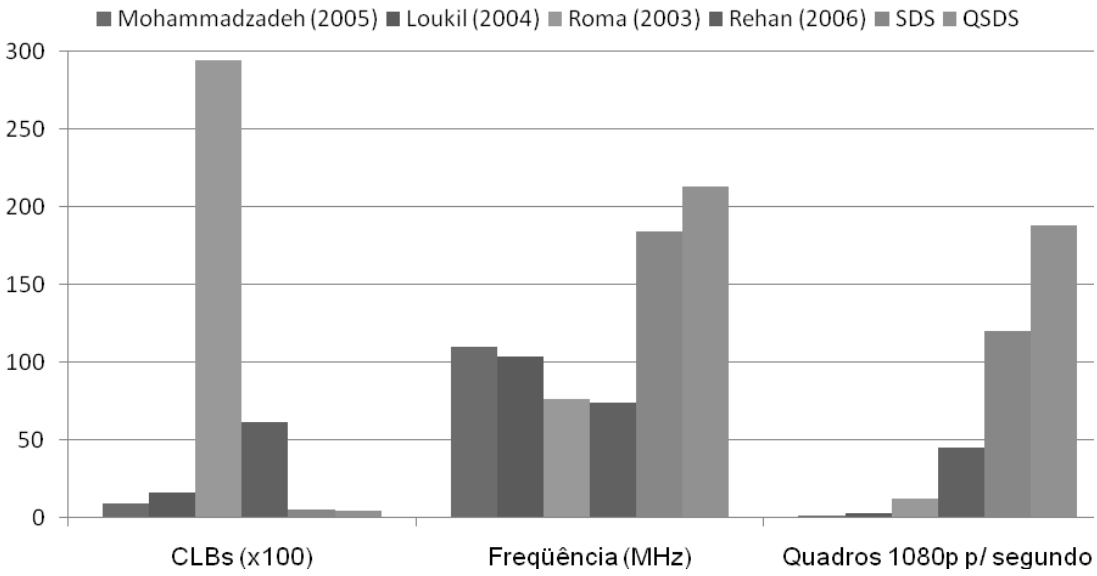



Figura 7.3: Resultados de custo em hardware e desempenho para as arquiteturas implementadas em FPGA avaliadas

Estes resultados estão fortemente ligados ao algoritmo utilizado para o desenvolvimento da arquitetura. Os algoritmos SDS e QSDS podem aliar bons resultados de qualidade dos vetores com alto desempenho e baixo custo computacional. Desta forma, as arquiteturas desenvolvidas para estes algoritmos são mais eficazes na utilização dos recursos de hardware, e podem atingir um desempenho superior a várias outras implementações arquiteturais.

### 7.2.2 Comparação com Resultados em *Standard Cells*

Muitos trabalhos da literatura apresentam os resultados do desenvolvimento de suas arquiteturas em *standard cells*. Para uma comparação mais justa com esses trabalhos, as arquiteturas do SDS e QSDS também foram implementadas em *standard cells*. A ferramenta Leonardo Spectrum (LEONARDOSPECTRUM, 2007) foi utilizada para a síntese da arquitetura na tecnologia TSMC 0,18 µm. A Tabela 7.3 apresenta os resultados da implementação *standard cells* para as arquiteturas SDS e QSDS, bem como as soluções encontradas na literatura.

Tabela 7.3: Comparação entre as arquiteturas SDS, QSDS e soluções encontradas na literatura para implementações em *Standard Cells*

| Solução | Tecnologia | Gates (K) | Freqüência (MHz) | HDTV Quadros/s |
|---|---|---|---|---|
| Larkin (2006) | 90nm | 10,1 | 250 | *50 |
| Li (2005) | 0,18µm | 17,5 | 200 | 23 |
| Gaedke (2007) | 90nm | 1200 | 334 | 25 |
| He (2007) | 0,25µm | 1020 | 100 | 46 |
| SDS | 0,18µm | 30,7 | 259 | *168 |
| QSDS | 0,18µm | 28,8 | 272 | *239 |

* Desempenho médio

A arquitetura apresentada em (LI, 2005) utiliza o algoritmo rápido PMVFAST, para uma área de pesquisa de 32x32 amostras. As arquiteturas apresentadas em (GAEDKE, 2007) e (HE, 2007) utilizam o algoritmo FS e não apresentam a informação de tamanho de área de pesquisa utilizada. Estas duas arquiteturas foram desenvolvidas para tratar vídeos de alta resolução em tempo real, e por utilizarem o algoritmo FS, são as soluções que utilizam a maior quantidade de *gates*. A número de g*ates* utilizadas por estas soluções é mais de 100 vezes maior do que o utilizado pela arquitetura apresentada em (LARKIN, 2006). As arquiteturas do SDS e QSDS utilizam 30,7K e 28,8K *gates* respectivamente, este consumo de recursos de hardware é maior do que os apresentados em (LARKIN, 2006) e (LI, 2005), no entanto, as soluções apresentadas para o SDS e QSDS operam sobre uma área de pesquisa aproximadamente 40 e 60 vezes maior, respectivamente. O consumo de recursos das arquiteturas do SDS e QSDS é cerca de 40 vezes menor que a solução apresentada em (GAEDKE, 2007), e aproximadamente 43 vezes menor que a solução apresentada em (HE, 2007).

Todas as arquiteturas podem operar com altas freqüências de operação, sendo que a arquitetura apresentada em (GAEDKE, 2007) possui a maior freqüência dentre as soluções avaliadas. Um dos fatores que contribui para essa elevada freqüência de operação é a tecnologia utilizada para a implementação da arquitetura (90nm), que é mais atual do que a utilizada pelas arquiteturas do SDS e QSDS (0,18µm). Mesmo não alcançando a maior freqüência de operação, as arquiteturas do SDS e QSDS obtêm os melhores resultados de desempenho. A arquitetura do SDS pode processar até 168 quadros HDTV 1080p por segundo no caso médio, e a solução para o QSDS pode processar até 239. Esta taxa de processamento é muito superior à apresentada pelas demais arquiteturas.

Analisando os resultados apresentados na Tabela 7.3 podemos perceber que as arquiteturas desenvolvidas para o SDS e QSDS possuem a melhor relação entre consumo de recursos de hardware de taxa de processamento. As soluções para o SDS e QSDS utilizam um número maior de *gates* que as arquiteturas apresentadas em (LARKIN, 2006) e (LI, 2005), no entanto, o desempenho alcançado é até 4,7 vezes maior do que o obtido por (LARKIN, 2006), que é o maior dentre as demais soluções. Isso pode ser percebido mais facilmente observando o gráfico apresentado na Figura 7.4, onde são apresentados os resultados de uso de recursos de hardware e de desempenho para as arquiteturas implementadas em *standard cells*.

O gráfico da Figura 7.4 apresenta os resultados de consumo de recursos de hardware, freqüência de operação e desempenho, para as arquiteturas desenvolvidas em *standard cells*. Analisando o gráfico é possível perceber que as arquiteturas do SDS e QSDS possuem os maiores resultados de desempenho, mesmo estando entre as arquiteturas de menor utilização de recursos de hardware. As soluções para o FS, apresentadas em (GAEDKE, 2007) e (HE, 2007), são as arquiteturas que apresentam o maior consumo de g*ates,* no entanto, mesmo obtendo altas freqüências de operação, seus resultados de desempenho estão entre os menores. As soluções para algoritmos rápidos (LARKIN, 2006) e (LI, 2005), possuem taxas de processamento similares com os menores resultados de utilização de recursos de hardware. Comparando apenas as arquiteturas desenvolvidas para algoritmos rápidos, as soluções para o SDS e QSDS possuem a melhor relação entre utilização de recursos e taxa de processamento obtido.

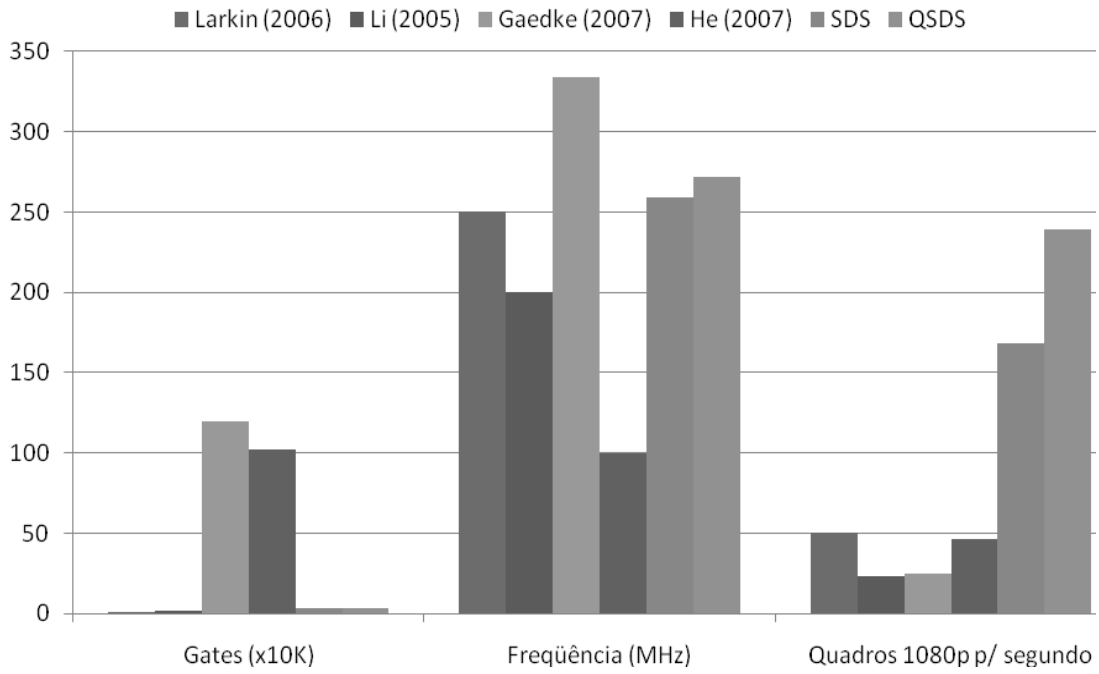



Figura 7.4: Resultados de custo em hardware e desempenho para as arquiteturas implementadas em *standard cells* avaliadas

# 8 CONCLUSÕES

Esta dissertação apresentou soluções arquiteturais de baixo custo em hardware e alto desempenho para a estimação de movimento de vídeos de alta resolução. Uma ampla avaliação sobre algoritmos de estimação de movimento foi desenvolvida, que serviu como base para escolher o algoritmo *Diamond Search* para as implementações em hardware. Uma pequena introdução aos conceitos de compressão de vídeo também foi apresentada.

O trabalho apresentado nesta dissertação foi dividido em duas grandes etapas: avaliação dos algoritmos de estimação de movimento e desenvolvimento arquitetural. Na primeira etapa, uma ampla pesquisa foi realizada sobre os principais algoritmos de estimação de movimento existentes na literatura. Os algoritmos estudados foram desenvolvidos em software e submetidos a diversos testes. Foram usadas 10 seqüências de vídeos como entrada para avaliar cada algoritmo. Os resultados médios de qualidade e custo computacional de cada algoritmo, para as 10 amostras de vídeo, foram apresentados. Uma avaliação destes resultados foi realizada, considerando as vantagens e desvantagens de cada algoritmo. O algoritmo *Diamond Search* apresentou os melhores resultados na relação custo computacional versus qualidade dos vetores de movimento gerados. Para refinar ainda mais a avaliação do algoritmo *Diamond Search*, uma nova bateria de testes foi desenvolvida, desta vez considerando outras versões do algoritmo com diferentes níveis de sub-amostragem de *pixel*. As versões com sub-amostragem de *pixel* de 2:1 e 4:1 foram as escolhidas para serem implementadas em hardware.

Na segunda etapa do trabalho foram desenvolvidas duas arquiteturas de hardware para os algoritmos SDS e QSDS. As arquiteturas foram descritas em VHDL e sintetizadas para FPGAs da Xilinx. Os resultados de síntese para as duas arquiteturas foram apresentados e mostraram que as arquiteturas desenvolvidas obtiveram um baixo custo em utilização de recursos de hardware aliado a um alto desempenho e bons resultados de qualidade dos vetores gerados. As arquiteturas do SDS e QSDS utilizam apenas cerca de 3,8K e 3,6K LUTs respectivamente, e no pior caso de operação, possuem desempenho suficiente para processar vídeos HDTV 1080p em tempo real a 30 quadros por segundo. Considerando o caso médio de operação, a arquitetura do SDS pode processar até 120 quadros HDTV 1080p, já a arquitetura do QSDS pode processar até 188 quadros por segundo nesta mesma resolução. A arquitetura do SDS apresenta um resultado de qualidade melhor do que o obtido pela arquitetura do QSDS. Assim, a solução para o SDS é mais apropriada para aplicações que necessitam de maior qualidade de estimação, e a solução para o QSDS é mais indicada para aplicações onde

seja necessária uma taxa de processamento maior, com menor utilização de recursos de hardware.

Também foi apresentado um capítulo de comparações, entre os resultados obtidos com as arquiteturas desenvolvidas e soluções encontradas na literatura. Primeiramente as arquiteturas do SDS e QSDS foram comparadas com soluções desenvolvidas anteriormente dentro do nosso grupo de pesquisa. Nesta comparação foi possível perceber a eficiência dos algoritmos SDS e QSDS, que implementados em hardware obtiveram a melhor relação entre utilização dos recursos de hardware, qualidade dos vetores gerados e taxa de processamento obtido. As soluções apresentadas neste trabalho obtiveram os menores resultados de utilização de recursos de hardware e também os maiores resultados de desempenho dentre todas as soluções anteriormente desenvolvidas no grupo de pesquisa.

As arquiteturas desenvolvidas também foram comparadas com soluções encontradas na literatura, tanto para implementações em FPGA quanto para *standard cells*. Mais uma vez as arquiteturas do SDS e QSDS obtiveram os maiores resultados de desempenho, e os menores resultados de utilização de recursos de hardware. As arquiteturas do SDS e QSDS também foram sintetizadas na tecnologia TSMC 0,18µm, para possibilitar uma comparação com outras soluções em *standard cells* encontradas na literatura. Os resultados de desempenho das arquiteturas do SDS e QSDS também foram os maiores dentre as soluções em *standard cells*. As soluções apresentadas neste trabalho também apresentaram os melhores resultados na relação consumo de recursos de hardware versus desempenho.

Como trabalhos futuros, pretendemos agregar diversas funcionalidades existentes nos padrões de compressão de vídeo atuais, as arquiteturas desenvolvidas neste trabalho. O padrão H.264, por exemplo, permite a utilização da estimação de movimento com diversos tamanhos de bloco, assim como a utilização de precisão de meio e quarto de pixel. Estas funcionalidades inserem uma grande complexidade computacional ao processo de estimação de movimento, o que implica em aumento da utilização de recursos de hardware e também em perda de desempenho. As arquiteturas desenvolvidas apresentam baixo custo em hardware, e desempenho superior ao necessário para processar vídeos HDTV em tempo real. Desta forma acreditamos que será possível inserir novas funcionalidades as arquiteturas de ME com uma utilização de hardware aceitável, e mantendo o desempenho para tratar vídeos HDTV em tempo real.

Outra proposta de trabalho futuro é desenvolver um novo algoritmo de estimação de movimento totalmente voltado para o desenvolvimento em hardware. Este algoritmo deve obter desempenho similar aos obtidos pelos algoritmos rápidos, no entanto, agregando características que facilitem a implementação em hardware, como regularidade no acesso a memória, possibilidade de paralelismos e ausência de dependência de dados.

# REFERÊNCIAS


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.

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.

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

BRASIL – Ministério das Comunicações. **Sistema brasileiro de TV Digital**. Brasília, 2005. Disponível em: <http://sbtvd.cpqd.com.br/>. Acesso em: mar. 2006.

CHIN, H. et al. A bandwidth efficient subsampling-based block matching architecture for motion estimation. In: IEEE ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. D/7-D/8.

GAEDKE, K. et al. Architecture and VLSI Implementation of a programmable HD Real-Time Motion Estimator. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2007. **Proceedings…** [S.l.]: IEEE, 2007. p. 1609-1612.

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

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

HE, W.; MAO, Z. An Improved Frame-Level Pipelined Architecture for High Resolution Video Motion Estimation. In: IEEE INTERNATIONAL SYMPOSIUM ON

CIRCUITS AND SYSTEMS, ISCAS, 2007. **Proceedings…** [S.l.]: IEEE, 2007. p. 1381-1384

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.

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, June 2004.

KORAH, R. et al. Motion Estimation with Candidate Block and Pixel Subsampling Algorithm. In: IEEE INTERNATIONAL WORKSHOP ON IMAGING SYSTEMS AND TECHNIQUES, IST, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 130-133.

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

LARKIN, D.; MURESAN, V.; O'CONNOR, N. A Low Complexity Hardware Architecture for Motion Estimation. In IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2006. **Proceedings…** [S.l.]: IEEE, 2006. p. 2677-2688.

LEONARDOSPECTRUM. Disponível em: <www.mentor.com/products/fpga_pld/synthesis/leonardo_spectrum/index.cfm>. Acesso em: out. 2007.

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

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

LIU, Y.; ORAINTARA, S. Complexity Comparison of Fast Block-Matching Motion Estimation Algorithms. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ICASSP, 2004. **Proceedings…** [S.l.]: IEEE, 2004. p. 341-344.

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.

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: jul. 2007.

MOHAMMADZADEH, M.; ESHGHI, M.; AZDFAR, M. An Optimized Systolic Array Architecture for Full-Search Block Matching Algorithm and its-Implementation on FPGA chips. In: IEEE NORTH-EAST WORKSHOP CIRCUITS AND SYSTEMS, NEWCAS, 2006. **Proceedings...** [S.l.]: IEEE, 2006. p.174-177.

REHAN, M. et al. An FPGA Implementation of the Flexible Triangle Search Algorithm for Block Based Motion Estimation. In IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2006. **Proceedings…** [S.l.]: IEEE, 2006. p. 521-523.

PORTO, M.; AGOSTINI, L.; BAMPI, S. Investigação de Algoritmos e Desenvolvimento Arquitetural para a Estimação de Movimento em Compressão de Vídeo Digital. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 2006. **Proceedings…** Chile: Universidad de Santiago de Chile, 2006. 1 CD-ROM.

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

PORTO, M. et al. High Throughput Hardware Architecture for Motion Estimation with 4:1 Pel Subsampling Targeting Digital Television Applications. In: PACIFIC-RIM SYMPOSIUM ON IMAGE AND VIDEO TECNOLOGY, PSIVT, 2007. **Proceedings…** [S.l.]: IEEE, 2007a.

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

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

ROMA, N.; DIAS, T.; SOUSA, L. **Customisable Core-Based Architectures for Real-Time Motion Estimation on FPGAs**. Berlin: Springer-Verlag, 2003. p. 745-754. (Lecture Notes in Computer Science, v. 2778).

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

VASSILIADIS, S. et al. The Sum-Absolute-Difference Motion Estimation Accelerator. In: IEEE EUROMICRO CONFERENCE. **Proceedings...** [S.l.]: IEEE, 1998. v. 2, p. 559-566.

VQEG. The Video Quality Experts Group Web Site. Disponível em: <www.its.bldrdoc.gov/vqeg/>. Acesso em: nov. 2006.

XILINX INC. **Xilinx**: The Programmable Logic Company. Disponível em: <www.xilinx.com>. Acesso em: jun. 2007.

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: jul 2007a.

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.

ZANDONAI, D. **Uma Arquitetura de Hardware para Estimação de Movimento aplicada à Compressão de Vídeo Digital**. Porto Alegre, 2003. 104f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

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.