Pós-Graduação em Ciência da Computação

Lucas Fernando da Silva Cambuim

Um módulo de Hardware de Tempo Real de Correspondência Semi Global para um Sistema de Visão Estéreo

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

RECIFE

2017

Lucas Fernando da Silva Cambuim

Um módulo de Hardware de Tempo Real de Correspondência Semi Global para um Sistema de Visão Estéreo

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Edna Natividade da Silva Barros

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

C178m Cambuim, Lucas Fernando da Silva
Um módulo de hardware de tempo real de correspondência semi global para um sistema de visão estéreo / Lucas Fernando da Silva Cambuim. – 2017.
147 f.:il., fig., tab.

Orientadora: Edna Natividade da Silva Barros.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndices.

1. Engenharia da computação. 2. Correspondência estéreo. I. Barros, Edna Natividade da Silva (orientadora). II. Título.

621.39 CDD (23. ed.) UFPE- MEI 2017-231

Lucas Fernando da Silva Cambuim

Um módulo de Hardware de Tempo Real de Correspondência Semi Global para um Sistema de Visão Estéreo

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 31/07/2017

_____

Orientadora: Profa. Dra. Edna Natividade da Silva Barros

BANCA EXAMINADORA

_____

Prof. Dr. Manoel Eusebio de Lima
Centro de Informática / UFPE

_____

Prof. Dr. Elmar Uwe Kurt Melcher
Centro de Engenharia Elétrica e Informática / UFCG

_____

Prof. Dr. Marcus Vinicius Duarte dos Santos
Instituto Federal de Pernambuco – Campus Caruaru

Eu dedico esta dissertação à minha família, aos meus amigos e professores que deram apoio incondicional, força, incentivo e amizade sem igual. Sem eles nada disto seria possível.

AGRADECIMENTOS

Dedico este momento à minha família, sobretudo ao meu querido e eterno pai, Fernando Cambuim e à minha mãe, Sônia Cambuim. Eles estiveram comigo em toda minha caminhada e, mesmo com todas as dificuldades, me deram tudo que eu precisei: o pão de cada dia, os livros do colégio, o carinho, os brinquedos. Não há como esquecer dos muitos beijos e cheiros na minha cabeça que meu pai me dava. Também não poderei jamais esquecer a difícil decisão que minha mãe precisou tomar de largar o emprego para cuidar de mim e me apoiar nos estudos, principalmente quando eu ainda era bem pequeno. Eles, juntos, me deram o principal: o Amor! A minha maior inspiração! O que me move a ser um bom filho, um bom amigo, um bom cidadão, e um bom profissional. Se hoje estou aqui, me formando, é graças aos meus pais. Papai do Céu resolveu antecipar a ida do meu Pai, mas eu sei que, onde ele estiver, estará muito feliz por mim. Pai, eu te Amo! E nunca me esquecerei de você!

Agradeço, acima de tudo, a Deus! Por me conceder este momento, e porque sem Sua permissão, nada disso poderia acontecer. Deus sempre me amparou nos momentos complicados durante estes dois anos de mestrado. E se por algum momento estive preocupado, com medo, ansioso, me sentindo incapaz, Ele sempre me lembrava que eu sou mais forte do que eu penso. Agradeço aos anjos que Deus me enviou antes e durante esta minha caminhada na Universidade. Essas pessoas, com bonitos gestos, me apoiaram e acreditaram em mim. Entre esses anjos estão as minhas tias Fracinete Cambuim, Floripes Cambuim, Idalina Cambuim; e as minhas professoras Tatiana, Luzinete Kurtinaitis, e Edna Natividade. Quero ainda fazer um agradecimento especial ao meu irmão, Rodrigo Cambuim, que esteve sempre presente e me deu muita força para continuar firme. E por fim, meu muito obrigado aos meus amigos de faculdade, com os quais compartilhei bons momentos. Rimos, nos estressamos , nos ajudamos, tomamos muito cafés, e realizamos muitos trabalhos juntos. Aprendi com eles a importância de trabalhar em grupo, o que não só facilitou o término dos trabalhos, mas também, e principalmente, renderam bons frutos, como o companheirismo, a confiança, além de ter fortalecido concretamente nossa amizade. Se eu pudesse dar um único conselho, eu diria: estude e confie em Deus, que o resto Ele te acrescentará.

Obrigado a todos por acreditarem em mim!

"Os créditos pertencem ao homem que está na arena, com a face coberta de poeira, suor e sangue; que luta com bravura, erra e, seguidamente, tenta atingir o alvo. É aquele que conhece os grandes entusiasmos, as grandes devoções e se consome numa causa justa. É aquele que, no sucesso, melhor conhece o triunfo final dos grandes feitos e que, se fracassa, pelo menos falha ousadamente, de modo que o seu lugar jamais será entre as almas tímidas, que não conhecem nem a vitória, nem a derrota"

(Theodore Roosevelt)

RESUMO

Abordagens de correspondência estéreo que geram mapas de disparidade densos, precisos, robustos e em tempo real são bastante atraentes para muitas aplicações tais como reconstrução 3D e navegação autônoma. Entre as abordagens mais adotadas, a técnica de Correspondência Semi Global (SGM) permite obter mapas de disparidades de ótima qualidade devido a sua capacidade de otimização que propaga custos de similaridades menores a partir de vários caminhos unidimensionais independentes ao longo de toda a imagem. Além disso, esta técnica, combinada com métricas de similaridade local suporta de forma robusta aos vários desafios de correspondência estéreo tais como ruídos, baixa textura e oclusões. Contudo, o acesso irregular aos dados, a grande quantidade de operações computacionais e a necessidade de grandes espaços de armazenamento para resultados intermediários impõem desafios para implementação paralela da técnica SGM em plataformas FPGAs. Na busca por resolver tais desafios, este trabalho propõe uma arquitetura escalável em hardware baseado na combinação de várias técnicas tais como paralelismos em vários níveis tais como no nível de processamento de linhas de imagem, de disparidade, de caminhos de propagação e processamento em pipeline. Para a implementação da técnica SGM foi proposta a sua combinação com o filtro de gradiente Sobel como uma etapa de pré-processamento e diferenças absoluta (AD) como um método de similaridade local. Esta combinação mostrou robustez tanto para imagens de alta qualidade disponíveis no banco de imagens Middlebury (22.7% de pixels errados) como para imagens de baixa qualidade fornecidas a partir do sistema de câmeras construído no nosso grupo de pesquisa. Além disso, também foi desenvolvido a etapa de pós processamento em hardware, que permitiu detectar regiões ruidosas e regiões de oclusão. Todo este sistema de correspondência estéreo foi implementado, simulado e validado na plataforma FPGA Cyclone IV gerando mapas de imagens de disparidade em resolução HD (1024x768 pixels), com um intervalo de 128 níveis de disparidades e usando 4 direções de caminhos para o método SGM. Com essa configuração obteve-se uma frequência de operação de 100 MHz, fornecendo imagens em uma taxa de 127 frames por segundo (FPS), utilizando 70% de seu recurso em elementos lógicos para processamento (LUTs) e 63% de memória para armazenamento de dados intermediários. Além disso, esta abordagem de correspondência estéreo foi validada em um contexto real de um sistema estéreo completo. Para tal validação, foi utilizada a plataforma hardware/software DE2i-150 na qual foram implementadas as etapas de calibração e retificação em processador e a arquitetura proposta do SGM implementada em FPGA e ambos os processamentos se comunicando através do barramento PCI-Express usando o framework RIFFA 2.2. Este sistema de visão estéreo completo permitiu obter um ganho de desempenho de 21x em relação a abordagem SGM em processador oferecida pela biblioteca OpenCV e dissipando 2 Watts de potência.

Palavras-chaves: Correspondência Estéreo. Tempo Real. Alta Precisão.

ABSTRACT

Stereo matching approaches that generate dense, accurate, robust, and disparity maps in real time are quite appealing to many applications such as 3D reconstruction and autonomous navigation. Among the best approaches is the Semi Global Correspondence (SGM) technique. This technique, combined with local similarity metrics, robustly supports the various challenges present in the stereo camera system such as noise, low texture and occlusions. The great quality of the SGM technique is due to the fact that this algorithm performs an optimization throughout the image, propagating smaller costs from several independent one-dimensional paths through the image. However, irregular access to data, large amount of computational operations, and high bandwidth to store intermediate results poses challenges for parallel implementation of the SGM technique on FPGAs. In this way, in the seek for solving such challenges, this work proposes a scalable array architecture based on systolic array, fully pipeline. The architecture is based on a combination of multilevel parallelism such as image line processing (two-dimensional processing) and disparity. For the implementation of the SGM technique it was proposed to combine it with the gradient filter as a preprocessing step and absolute differences as a method of local similarity. This combination proved to be a robust approach for both high-quality images available in the Middlebury benchmark (22.7 % of wrong pixels) and for low-quality images provided from the camera system built by our research group. In addition, the hardware L/R check step was also developed, which allowed the detection of noisy and occluded regions. This whole stereo matching system was implemented, simulated and validated on the Cyclone IV FPGA platform, generating disparate image maps in HD resolution (1024x768 pixel), with a range of 128 disparity levels and using 4 path directions for the SGM method. With this configuration an operating frequency of 100 MHz was obtained, providing images at a rate of 127 frames per second, using 70% of its resource in logical elements for processing and 63% of memory for intermediate data storage. In addition, this stereo matching approach was validated in a real context of a complete stereo system in which a stereo camera system was built and the steps of calibration, rectification were implemented. For this validation, the DE2i-150 hardware/software platform was used with the calibration and rectification steps implemented in the processor and the proposed SGM architecture implemented in FPGA and both the platforms communicating itself through the PCI-Express bus using the RIFFA 2.2 framework. This complete stereo vision system has achieved a speedup of 21x over the SGM processor approach offered by the open source computer vision library (OpenCV) and with a power dissipation of 2 Watts.

Key-words: Stereo Matching. Real Time. High Precision.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AD Diferença Absoluta.

ASIC Circuito Integrado de Aplicação Específico.

BP Belief Propagation.

CPU Unidade Central de Processamento.

FPGA Field Programmable Gate Array.

FPS Frames Por Segundo.

GC Graph Cut.

GPU Unidade de Processamento Gráfico.

LUT LookUp Table.

OpenCV Open Source Computer Vision Library.

RIFFA A Reusable Integration Framework For FPGA Accelerators.

SGM Semi Global Matching.

SUMÁRIO

1 INTRODUÇÃO

Os sistemas embarcados têm sido usados eficientemente para interagir com o ambiente. Essa interação envolve sensores que oferecem informações do ambiente a fim de que o processador embarcado possa interpretar o que acontece ao seu redor e assim determinar as instruções que serão utilizadas por atuadores para executar alguma determinada ação (SANGIOVANNI-VINCENTELLI et al., 2013). Desta forma, vários tipos de sensores têm sido desenvolvidos para aumentar a capacidade dos sistemas embarcados de reproduzir fielmente o que está acontecendo no mundo.

Uma importante informação que pode ser obtida a partir do ambiente através de sensores é a informação de profundidade. A profundidade permite que o sistema embarcado tenha noção das distâncias para cada objeto no ambiente. Isto é fundamental para muitas aplicações tais como: reconstrução tridimensional, detecção e reconhecimento, navegação autônoma, etc. A maioria destas aplicações tipicamente exigem sensores que sejam capazes de gerar mapas densos de profundidade de maneira precisa, robusta e em tempo real. Por exemplo, sistemas inteligentes embarcados de detecção de obstáculos para automóveis necessitam de informação de profundidade para garantir que o sistema consiga identificar precisamente uma possível colisão e tomar uma decisão em tempo hábil para evitar um acidente (SIVARAMAN; TRIVEDI, 2013).

Sensores tradicionais capazes de fornecer informações de profundidade são do tipo ativos, tais como lasers, radares e sonares. Estes sensores detectam a distância de objetos medindo o tempo de transmissão e retorno de um sinal emitido pelos sensores e refletido pelos objetos. Estes equipamentos além de serem de alto custo são de baixa resolução, consomem muita energia e geralmente sofrem interferência de outros sensores ou do ambiente (SUN; BEBIS; MILLER, 2006). Outras abordagens promissoras empregam sistemas de câmeras estéreo. Câmeras estéreo são sensores óticos usualmente referidos como sensores passivos, porque eles adquirem dados de maneira não intrusiva. Tais sensores são ideais para extrair profundidade pois além de serem de baixo custo, fornecem uma grande quantidade de informação sobre o ambiente em altas taxas de captura de frames.

Usualmente, um sistema de visão estéreo possui duas câmeras que coletam imagens a partir de diferentes pontos de vista sobre o mesmo cenário. Assim, qualquer ponto 3D real está posicionado em coordenadas diferentes nas duas imagens. Através da técnica de Correspondência Estéreo (do inglês Stereo Matching) proposta por Scharstein e Szeliski (2002), os pixels correspondentes são buscados horizontalmente entre as duas imagens (as câmeras precisam estar calibradas e as imagens precisam estar retificadas (HARTLEY; ZISSERMAN, 2003a)) e a distância entre eles, também chamada de *disparidade*, é então calculada. O intervalo de busca que compreende os pixels candidatos é chamado de *intervalo de disparidades* ou *intervalo de disparidades* e a imagem gerada constituída de valores

de disparidades para cada pixel é chamada de *mapa de disparidade*. A profundidade real é obtida diretamente a partir do mapa de disparidade, calculando o inverso do valor de disparidade.

A qualidade da medição da profundidade depende diretamente da qualidade dos resultados obtidos da etapa de correspondência estéreo em encontrar os pixels correspondentes corretos. Varias abordagens de correspondência estéreo têm sido propostas buscando fornecer mapas de disparidade precisos. Entre estas abordagens destacamos a técnica de Correspondência Semi-Global (SGM) proposta por Hirschmuller (HIRSCHMULLER, 2008) que consiste em um dos melhores métodos avaliados através do benchmark Middlebury proposto por Scharstein (SCHARSTEIN; SZELISKI, 2002). Esta técnica combina abordagens de cálculo de similaridade local e abordagens de otimização global que permite ao SGM obter resultados robustos para vários desafios de correspondência estéreo, tais como regiões de baixa textura, oclusões, regiões de borda e diferenças radiométricas (HIRSCHMULLER; SCHARSTEIN, 2009). A alta qualidade da técnica SGM é devido ao fato desta abordagem realizar uma otimização ao longo de toda a imagem com o diferencial que a técnica SGM propaga custos menores a partir de caminhos unidimensionais independentes e combina-os para gerar mapas de disparidade de alta qualidade. Além disso, o SGM realiza a otimização global em uma única iteração em toda a imagem. Esta independência de caminhos é uma característica atrativa para implementações paralelas. Contudo a grande quantidade de operações computacionais necessárias torna as implementações da técnica SGM, em plataforma baseada em processador, bastante lentas, não sendo adequadas para uso em aplicações de tempo real. Por exemplo a implementação da técnica SGM em processador (GEHRIG; RABE, 2010) alcança uma taxa de 16 frames por segundo (FPS). Por outro lado as abordagens em (GPU) consegue atingir taxas de processamento superiores aos processadores. Por exemplo no trabalho propostos pelos autores Hernandez-Juarez et al. (2016) foi proposta uma implementação em GPU da técnica SGM que consegue processar mapas de disparidade a uma taxa de processamento de 48 FPS.

Contudo, arquiteturas baseadas em processador ou em GPU não são adequadas para aplicações embarcadas (por exemplo, carro alto dirigível e robôs autônomos (STEINGRUBE; GEHRIG; FRANKE, 2009)) pois consomem grande quantidade de energia (BAILEY, 2011). Por isso, soluções de hardware dedicado, baseados em (FPGA) ou em (ASIC) têm sido propostas devido ao ótimo desempenho alcançado e com baixo consumo de energia. Além disso, estas plataformas hardware conseguem explorar mais paralelismo do que as plataformas baseadas em processador e GPU e permitem acesso rápido aos dados suportando, assim, um desempenho em tempo real para o processamento do algoritmo SGM. Contudo o acesso irregular aos dados, uma grande quantidade de operações e a necessidade para armazenar grande quantidade de resultados intermediários impõem desafios para implementações em plataformas baseadas em FPGAs (BANZ et al., 2010). Estes problemas pioram com o aumento da resolução da imagem e um maior intervalo de disparidades, que

é uma tendência nas aplicações atuais (SINHA; SCHARSTEIN; SZELISKI, 2014). Por exemplo, no trabalho proposto pelos autores Gehrig, Eberli e Meyer (2009), foi desenvolvida uma implementação da técnica SGM em FPGA que processa mapas de disparidade a uma taxa de 25 FPS com resolução VGA e intervalo de 64 valores de disparidades. Além da baixa resolução e intervalo pequeno de disparidade a implementação não escalável ainda exige memórias externas com grandes espaços de armazenamento. No trabalho dos autores Honegger, Oleynikova e Pollefeys (2014) foi desenvolvido um hardware para processar a técnica SGM que consegue processar apenas um intervalo de 32 valores de disparidades.

Dessa forma, dado os desafios de implementar o algoritmo SGM, este trabalho propõe uma arquitetura para processamento da técnica SGM escalável para plataformas baseadas em FPGA baseada em array sistólico e com caminho de dados implementados em pipeline e totalmente parametrizável. A arquitetura inclui a combinação de paralelismos multinível tal como processamento de várias linhas de imagem, o processamento paralelo de todos os valores de disparidades e o processamento paralelo de todas funções de custo de caminho da técnica SGM. A abordagem serializada para o processamento de varias linhas permitiu ao módulo proposto do SGM atingir alto desempenho com pouco recurso de hardware. Para a implementação da técnica SGM foi proposto a sua combinação com o filtro de gradiente sobel como uma etapa de pré-processamento e diferenças absoluta (AD) como um método de similaridade local. Esta combinação mostrou-se uma abordagem robusta tanto para imagens de alta qualidade disponíveis no banco de imagens Middlebury (22.7% de pixels errados) como para imagens de baixa qualidade fornecido pelo sistema de câmeras construído para validação da implementação proposta. Para compor a arquitetura do SGM, é proposto um módulo em hardware para detectar e remover regiões de oclusão, importante fonte de erros em sistemas de correspondência estéreo. A abordagem SGM foi implementada em SystemVerilog, simulada e validada usando as ferramentas ModelSim e o Quartus 16.1 com a plataforma FPGA Cyclone IV gerando mapas de imagens de disparidade em resolução HD (1024x768 pixel), com intervalo de 128 níveis de disparidades e usando 4 direções de caminhos para o método SGM. Com isso obteve-se uma frequência de operação de 100 MHz, fornecendo imagens em uma taxa de 127 fps, utilizando 70% de seu recurso em elementos lógicos para processamento e 63% de memória para armazenamento de dados intermediários. Além disso, a frequência de operação é invariável com a mudança de parâmetros do sistema, devido a abordagem em pipeline, proposta neste trabalho.

A abordagem do SGM em hardware foi validada em um contexto real de um sistema estéreo completo no qual foi construído um sistema de câmeras estéreo e implementado as etapas de calibração e retificação. Para tal validação, foi utilizada a plataforma hardware/software DE2i-150 (TERASIC, 2013) com as etapas de calibração, retificação e pré-processamento implementadas no processador ATOM 2600 e a arquitetura proposta do SGM com a detecção de oclusão implementados no FPGA, Cyclone IV. Ambos os

processamentos se comunicam através do barramento PCI-Express usando o framework RIFFA 2.2 (JACOBSEN et al., 2015). Este sistema de visão estéreo completo permitiu obter frames de mapas de disparidade a uma taxa de 52 FPS com resolução de 640x480 e com intervalo de disparidades de 64 valores tendo um ganho de 21x em relação a abordagem SGM oferecida pela biblioteca OpenCV. Além disso este sistema dissipa 2 watts de potência, bastante inferior a potência dissipada pelas plataformas de processamento GPU e CPU atuais. Assim, a arquitetura proposta se mostra uma opção atrativa para aplicações atuais embarcadas em termos de taxa de processamento, resolução de profundidade, quantidade de recursos de processamento, precisão do mapa de disparidade e baixo consumo de energia.

O restante do trabalho é organizado da seguinte forma: a seção 2 faz uma revisão de todas as etapas de visão estéreo detalhando o algoritmo SGM e as principais dificuldades de implementar este algoritmo. Também neste capítulo é feito uma motivação para o uso das plataformas FPGA. A seção 3 descreve os trabalhos relacionados envolvendo abordagens do SGM em várias plataformas de processamento principalmente em hardware FPGA. A descrição da abordagem SGM e a estratégia de aceleração do algoritmo SGM estão na seção 4. A seção 5 apresenta a estratégia de validação da implementação da abordagem proposta do SGM. A seção 6 apresenta os resultados de desempenho, qualidade da implementação FPGA e comparativos com resultados de outras abordagens existentes do SGM em FPGA. Finalmente na seção 7 são descritas as conclusões obtidas neste trabalho e trabalhos futuros.

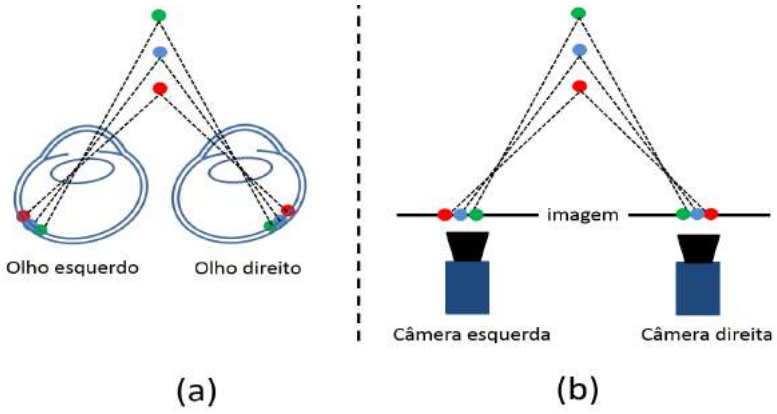
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta conceitos e definições necessários para entender todo o restante do trabalho. A seção 2.1 fornece a fundamentação para calcular profundidade a partir de um sistema de câmeras estéreo; a seção 2.2 aborda conceitos, desafios e abordagens existentes para obtenção de disparidade; a seção 2.3 detalha a técnica adotada neste trabalho, *correspondência semi global*, abordando as dificuldades de se implementar esta abordagem; e a seção 2.4 motiva o uso das plataformas FPGAs para implementação desta técnica. Se o leitor já é familiarizado tanto com a área de visão estéreo, como com as plataformas FPGAs, pode dirigir-se a seção 2.3.

2.1 Visão Geral dos Sistemas Estéreos

A visão dos seres humanos e a da maioria dos animais é baseada em um sistema de visão estéreo altamente sofisticado. Os dois olhos são separados por uma distância e portanto observa-se o ambiente ao redor a partir de perspectivas ligeiramente diferentes. Essa diferença de perspectiva permite ao cérebro extrair informação de profundidade através da seguinte lógica: quanto mais próximo o objeto esteja do ser humano, maior o seu deslocamento nas duas imagens geradas na retina. Assim, esta informação permite que o cérebro seja capaz de aprender a identificar a distância real de cada objeto na cena.

Figura 1 – Perspectivas diferente no sistema visual humano (a) e no sistema de câmeras estéreo (b)



Os sistemas de câmeras estéreo utiliza este mesmo princípio para determinar a profundidade dos objetos. Um ponto em uma cena no mundo real é projetado em pixels com posições relativamente diferentes nas duas imagens como mostrado na Figura 1(b), onde a função da Correspondência Estéreo é determinar estes pixels. Se for possível encontrar estes pixels correspondentes então é possível determinar a profundidade.

Para explicar o processo de obtenção de profundidade, o sistema estéreo é descrito em termos de *geometria epipolar*, como mostrado na Figura 2. Nessa geometria, as câmeras são aproximadas a um modelo de câmera *pinhole*, onde a abertura da câmera esquerda e direita é descrita como os pontos O_L e O_R, respectivamente. $P(x, y, z)$ é um ponto no espaço 3D e os dois pontos E_L e E_R representam os *epipolos*. Um epipolo é um ponto de intersecção da linha através dos centros óticos com o plano de imagem. Os pontos P, O_L e O_R formam um plano chamado de *plano epipolar*. A linha $P - O_L$ é visto pela câmera da esquerda como P_L porque este ponto está diretamente em linha com o centro da câmera O_L. No plano de imagem da direita a linha representada por $E_R - P_R$ é chamado de linha epipolar. Para cada ponto observado em uma imagem, o mesmo ponto deve ser observado numa linha epipolar correspondente na outra imagem. Isto é conhecido como *restrição epipolar* o qual reduz o problema de encontrar o pixel correspondente a uma busca ao longo de linhas epipolares conjugadas. A distância dos pontos correspondentes quando uma das duas imagens é projetada sobre a outra é chamada de *disparidade* e é calculada como $d = |x_L - x_R|$. A disparidade representa a diferença relativa na localização entre os pixels comuns (ou seja, a localização do pixel na imagem da direita relativa a localização deste pixel na imagem da esquerda) no par estéreo. A etapa responsável pela busca dos pixels correspondentes e obtenção do valor de disparidade é chamado de *Correspondência Estéreo*.

Figura 2 – Geometria Epipolar de um sistema de câmeras estéreo. (a) Vista 3D e (b) Vista 2D

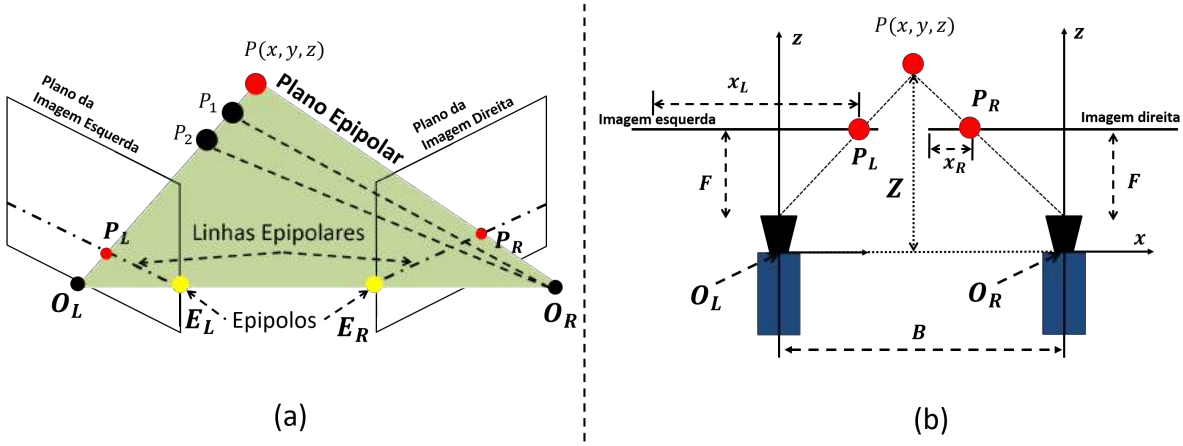


Imagem modificada a partir de Hadjitheophanous et al. (2010)

Dado que são conhecidos os pontos correspondentes $P_L = (x_L, y_L)$, $P_R = (x_R, y_R)$ e a geometria da câmera (ou seja, distância focal e distância entre centros óticos das câmeras), as coordenadas do ponto $P(x, y, z)$ no mundo tridimensional são calculadas através de *triangulação*. Por semelhança de triângulos dados pelos pontos (P, O_L, O_R) e (P, P_L, P_R) as coordenadas de P são obtidas através das Equações 2.1, 2.2 e 2.3. Nesta equação o valor B indica a distância entre os centros óticos O_L e O_R; o valor F é o

comprimento focal comum; e o valor Z é a distância entre o ponto P e a linha de base B, que representa a profundidade do objeto. Como pode ser constatado principalmente na Equação 2.3, a profundidade z é inversamente proporcional a disparidade.

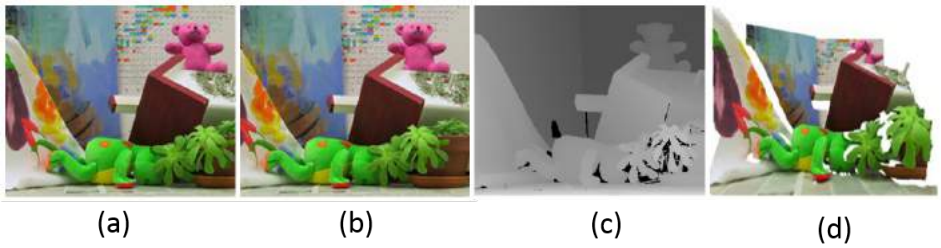
$$x = \frac{x_L \cdot B}{d} \tag{2.1}$$

$$y = \frac{y_L \cdot B}{d} \tag{2.2}$$

$$z = \frac{B \cdot F}{d} \tag{2.3}$$

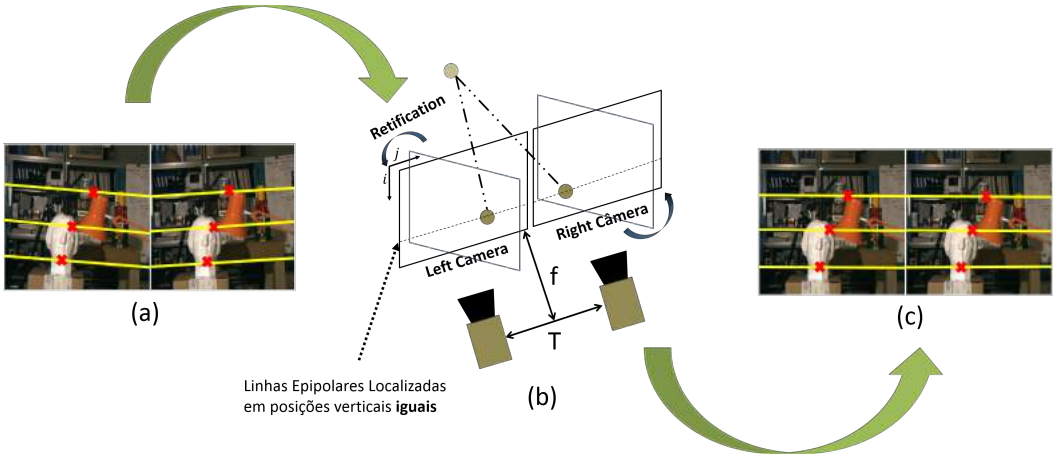
Se for encontrado o valor de disparidade para todos os pixels da imagem obtém-se o mapa de disparidade, como mostrado na Figura 3(c). Ao realizar o procedimento de triangulação em cada valor do mapa de disparidade obtém-se o mapa de profundidade, como mostrado na Figura 3(d), indicando a distância real de cada ponto para o sistema de câmeras. Quanto maior o valor de disparidade, ou seja, regiões mais clara no mapa de disparidade, mais próximo o objeto estará do sistema de câmeras estéreo.

Figura 3 – Imagem da câmera da esquerda (a) e da direita (b), mapa de disparidade (c) resultante e a projeção no espaço 3D (d)



Se for considerado que um conjunto de imagens estéreos vem a partir de câmeras *calibradas*, a primeira etapa em todo sistema estéreo é a *retificação* das imagens estéreos. A calibração extrai do sistema estéreo a geometria da câmera, que são os parâmetros intrínsecos (comprimento focal, centros óticos e distorção das lentes) e extrínsecos (posições relativas e orientações de cada câmera do sistema). Uma vez que os parâmetros intrínsecos e extrínsecos somente variam se a estrutura do sistema estéreo modificar, a calibração só precisa ser realizada uma única vez. O procedimento de retificação transforma, a partir dos parâmetros intrínsecos e extrínsecos, cada imagem em um plano de imagem comum, alinhando os pares de linhas epipolares conjugadas para um eixo de imagem comum (usualmente a horizontal) como mostrado na Figura 4. Sem a retificação a busca por pixels correspondentes envolveria um espaço de busca 2D no qual primeiro buscaria-se a linha epipolar correspondente e em seguida buscaria-se o pixel correspondente ao longo desta linha. Com a retificação a busca por pixels similares é reduzida a uma busca em uma dimensão ao longo da linha epipolar que corresponde a mesma localização da linha do pixel de referência na imagem da esquerda.

Figura 4 – Processo de retificação de imagens. (a) Imagens originais, (b) Processo de retificação e (c) Imagens retificadas



Vamos detalhar um pouco mais sobre o procedimento de calibração. A tarefa de calibração de câmera é determinar os parâmetros da transformação entre um objeto no espaço 3D e a imagem observada pela câmera a partir da informação visual (imagens). Seja $\mathbf{X} = (X, Y, Z, 1)^T$ a coordenada do ponto no mundo 3D. Então a coordenada 3D do mesmo ponto no frame da câmera $\mathbf{X}_{cam}$ é transformado como:

$$\mathbf{X}_{cam} = [\mathbf{R} \quad \mathbf{T}] \ \mathbf{X} \tag{2.4}$$

onde $\mathbf{R}$ é uma matriz de rotação 3x3 e $\mathbf{T}$ é uma matriz de translação 3x1. Agora, seja $\mathbf{x} = (x, y, 1)^T$ a coordenada na imagem deste ponto 3D, então o mapeamento do mundo 3D para o 2D torna-se:

$$\mathbf{x} = \mathbf{K} \ [\mathbf{R} \quad \mathbf{T}] \ \mathbf{X} \tag{2.5}$$

onde $\mathbf{K}$ é uma matriz 3x3 contendo os parâmetros intrínsecos da câmera como mostrado a seguir:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

onde f_x e f_y são os comprimentos focais da câmera no eixo x e y respectivamente. A tupla (c_x, c_y) é coordenada do ponto principal.

As matrizes $\mathbf{R}$ e $\mathbf{T}$ constituem os parâmetros extrínsecos da câmera. A matriz R de dimensão 3x3 constitui a matriz de rotação como mostrado na Equação 2.7 e a matriz T constitui a matriz de translação como mostrado na Equação 2.8.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.7}$$

$$\mathbf{T} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{2.8}$$

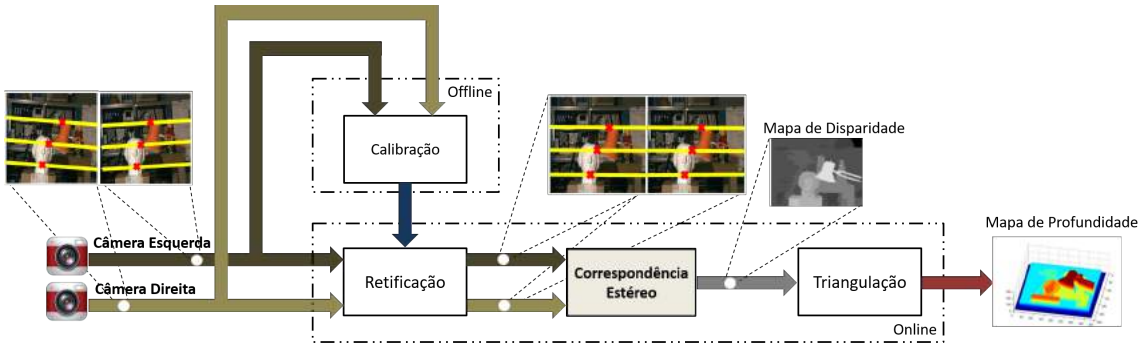
Agregando as duas matrizes obtém-se a matriz M como mostrado na Equação 2.9.

$$[\mathbf{R} \quad \mathbf{T}] = \mathbf{M} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.9}$$

Encontrar os parâmetros intrínsecos e extrínsecos que a priori são desconhecidos é tarefa da calibração de câmera. Não aprofundaremos na álgebra complexa envolvida na descoberta destes parâmetros, uma vez que este não é foco deste trabalho. Desta forma as tecnologias de calibração e retificação não serão mais discutidas nas seções e será considerado que todas as imagens estão perfeitamente retificadas, ou seja, linhas epipolares de ambas as imagens estão perfeitamente alinhadas. Um estudo detalhado sobre os conceitos de geometria epipolar, calibração e retificação pode ser encontrado em Hartley e Zisserman (2003b). A fim de validar a aplicação no contexto prático, são utilizadas funções disponíveis que são detalhadas no capítulo 5.

Todas as etapas descritas anteriormente (calibração, retificação, correspondência estéreo e triangulação) compõem o sistema estéreo mostrado na Figura 5. Dentre estas etapas, a *Correspondência Estéreo* é a mais crítica pois é ela quem determina diretamente a qualidade da profundidade real. A função principal da correspondência estéreo é encontrar os pixels correspondentes nas imagens estéreo e retornar a sua distância horizontal, chamada de disparidade.

Figura 5 – Arquitetura geral de um sistema estéreo



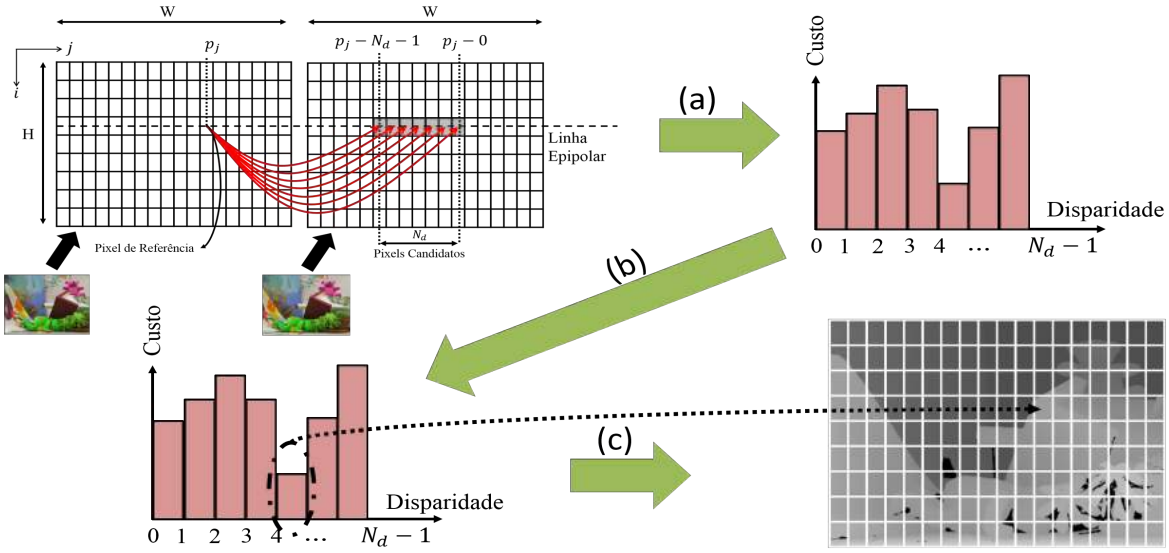
Uma vez que a busca por pixels similares é baseada em valores de intensidade da imagem, diversos obstáculos tais como ruídos, ambiguidade e variações de iluminação entre as imagens impõem desafios para encontrar precisamente os pares de pixels correspondentes. Diversas soluções têm sido desenvolvidas e avaliadas para minimizar os efeitos destes

obstáculos. A próxima seção aborda detalhadamente esta importante etapa do sistema de visão estéreo.

2.2 Visão Geral da Correspondência Estéreo

Como já esclarecido na seção 2.1, um ponto no espaço tridimensional é projetado em coordenadas diferentes nas imagens estéreo. A função principal da técnica de correspondência estéreo é encontrar estes pixels e retornar a sua distância horizontal também chamada de disparidade. A disparidade é uma medida de profundidade relativa no qual quanto maior for o valor da disparidade mais próximo o pixel está do sistema. Para cada pixel a partir da imagem de *referência* é realizada uma busca pelo pixel mais similar na imagem de *correspondência*. A maioria das técnicas de correspondência estéreo tem adotado como a imagem referência, a imagem da câmera da esquerda e a imagem de correspondência a imagem da câmera da direita, embora nada impede que possa ser utilizado a imagem da esquerda como imagem de correspondência e a imagem da direita como imagem de referência.

Figura 6 – Processo de busca pelo pixel mais similar. (a) Atribuição de custo de correspondência de pixel de referência para cada pixel candidato. (b) Busca pelo pixel candidato de menor custo. (c) Retorno do valor de disparidade do pixel de menor custo (mais similar)



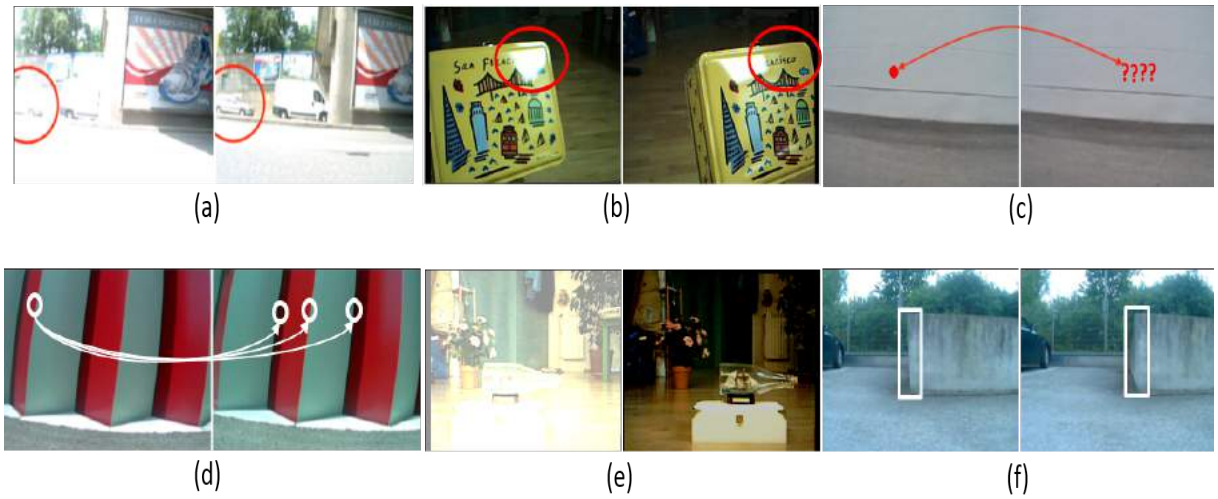
Uma vez que as imagens estão retificadas (o processo de retificação foi explicado na seção 2.1) então a busca pelo pixel mais similar na imagem de correspondência se concentra em uma busca ao longo da mesma linha do pixel de referência, como mostrado na Figura 6. A busca ao longo da linha fica limitada por um intervalo chamado de *intervalo de disparidade* com tamanho denotado de N_d e cada pixel dentro deste intervalo é chamado de *pixel candidato*. Cada pixel candidato é localizado variando o valor de d com relação a

posição do pixel de referência (i, j), ou seja, cada pixel candidato é localizado na posição $(i, j - d)$. Variando d a partir de 0 até $N_d - 1$ obtém-se todos os pixels candidatos.

Para realizar a busca pelo pixel mais similar é necessário definir primeiramente uma *função de custo* que avalia a similaridade do pixel referência com o pixel candidato. Uma vez definida esta função então calcula-se o custo de similaridade de todos os pixels candidatos em relação ao pixel referência e em seguida realiza-se a busca pelo pixel mais similar (geralmente o pixel mais similiar é aquele que tem o menor valor de custo). Ao encontrar o pixel mais similar então retorna-se para posição (p_i, p_j) da imagem de disparidade que está sendo determinada o valor de disparidade d deste pixel similar em relação ao pixel de referência. Definindo os custos de todos os pixels referência em relação a todos os seus pixels candidatos e em seguida fazendo a busca pelo pixel mais similar para todos os pixels referência obtém-se o mapa de disparidade *denso* em escala de cinza.

Assim, para que o cálculo de disparidade consiga determinar corretamente o valor de disparidade é necessário que a função de custo defina coerentemente os valores de custo que associa cada pixel de referência com cada pixel candidato. Num caso mais crítico, espera-se que pelo menos o pixel correspondente (o mais similar) tenha o menor valor de custo em relação a todos os outros pixels candidatos. Contudo, uma vez que a definição do custo baseia-se em valores de intensidade de pixel fornecido a partir de câmeras, diversos obstáculos reais exemplificados na Figura 7 impõem dificuldades para definir estes custos corretamente.

Figura 7 – Exemplo de situações que dificultam encontrar corretamente o pixel correspondente. (a) Variação de iluminação entre os frames, (b) Reflexo da luz em posições diferentes, (c) Regiões de baixa textura, (d) Estrutura repetida, (e) Objetos transparentes. (f) Oclusões

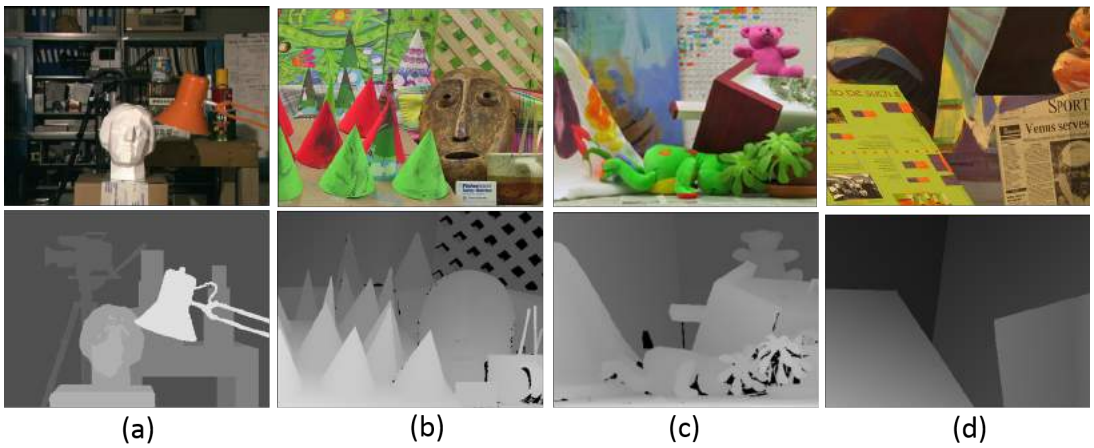


Diferença de iluminação ocorre quando imagens estéreo diferem entre si quanto a luminosidade em certas partes da imagem. Isto acontece devido a diferença de pontos de vista onde em um ponto de vista a luz incidiu diferente do outro ponto de vista como

evidenciado por Hirschmüller (HIRSCHMULLER; SCHARSTEIN, 2009). Como mostrado na Figura 7 (a) e (b), regiões nas quais incidiu mais luz em uma imagem do que na outra faz com o que o pixel mais similar entre os candidatos tenha valor de intensidade bastante diferente do pixel de referência dificultando encontrá-lo. **Distorções radiométricas** são diferenças na configuração do brilho, do gamma e da abertura entre as câmeras, que faz com que o ponto real 3D seja representado nas duas imagens por valores de intensidade bastante diferentes, podendo conduzir a erros na busca dos pixels correspondentes. Na Figura 7 (e) tem-se um exemplo de distorção radiométrica na qual a imagem da esquerda possui mais brilho que a imagem da direita. **Regiões de Baixa Textura e Estruturas Repetidas** como mostrados nas Figuras 7 (c) e (d) dificultam a tarefa de decidir sobre o pixel mais similar pois vários pixels candidatos diferentes têm o mesmo valor de intensidade e logicamente terão os mesmo custos de correspondência. **Oclusões** ocorre quando o ponto correspondente está fora da parte visível da imagem ou ocultado por um objeto mais próximo a câmera como mostrado na 7 (f). Regiões de oclusão são fonte de erros no processo de correspondência. **Imagens mal Alinhadas** no qual linhas epipolares não ficaram perfeitamente alinhadas verticalmente, gerados a partir do processo de calibração e retificação, podem comprometer todo o processo de correspondência estéreo se a abordagem adotada não for capaz de lidar com imagens desalinhadas.

A fim de lidar com todas estas dificuldades, diversos algoritmos de correspondência estéreo têm sido propostos, bem como benchmarks de imagens estéreo para avaliação. Os autores Scharstein e Szeliski (2002) definiram um benchmark chamado de Middlebury que fornece um conjunto de pares de imagens estéreo de alta qualidade, como mostrado na Figura 8, simulando todas as possíveis situações reais que dificultam o processo de correspondência estéreo, tal como aquelas descritas na Figura 7.

Figura 8 – Exemplos a partir do benchmark Middlebury de imagens estéreo de avaliação (apenas é mostrado a imagem esquerda) e seus respectivos mapas de disparidades de referência (a) Tsukuba (b) Cones (c) Teddy (d) Venus



(SCHARSTEIN; SZELISKI, 2002)

Para cada par de imagem estéreo deste benchmark tem-se o mapa de disparidade de referência que permite avaliar a qualidade dos resultados obtidos para uma abordagem proposta. Para avaliar a qualidade de cada abordagem os autores Scharstein e Szeliski (2002) definiram um conjunto de métricas. Entre estas as mais utilizadas são:

1. **Erro Médio Quadrático**. É o erro entre o mapa de disparidade $d_C(i,j)$ fornecido pela abordagem proposta e o mapa disparidade de referência $d_T(i,j)$. Esta medida está formalmente descrita pela Equação 2.10,onde N é o número total de pixels, W é a largura da imagem e H é a altura da imagem.

$$E = \left(\frac{1}{N} \sum_{i=0}^{i=H-1} \sum_{j=0}^{j=W-1} |d_C(i,j) - d_T(i,j)|^2 \right)^{1/2} \tag{2.10}$$

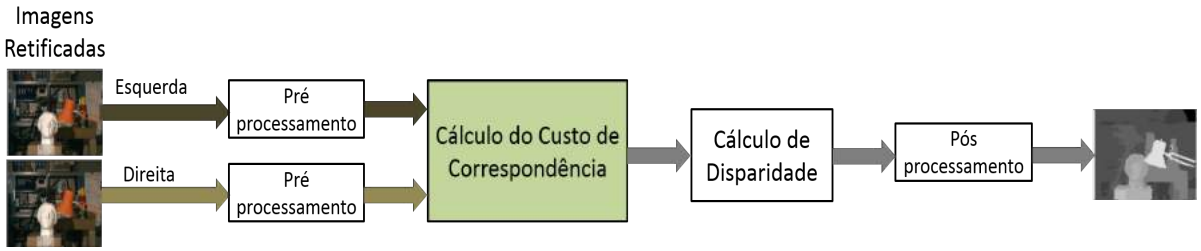
2. **Porcentagem de pixels de correspondência errados**. Esta medida descrita formalmente pela Equação 2.11 determina a porcentagem de pixels cujo valor de disparidade calculado diferenciou acima de um limiar σ_d do valor de disparidade correto. O parâmetro σ_d é chamado de tolerância do erro de disparidade.

$$E = \frac{1}{N} \sum_{i=0}^{i=H-1} \sum_{j=0}^{i=W-1} (|d_C(i,j) - d_T(i,j)| > \sigma_d) \tag{2.11}$$

Os algoritmos de correspondência estéreo podem ser classificados como algoritmos *esparsos* os quais efetuam a correspondência apenas para alguns pontos que são mais relevantes na imagem ou algoritmos *densos*, nos quais efetuam a correspondência para todos os pontos da imagem. Para diversas aplicações é mais interessante a utilização de abordagens que geram mapas densos pois proporcionam maior informação de profundidade envolvendo todos os possíveis objetos contidos no cenário.

Entre os algoritmos considerados densos, uma grande quantidade de abordagens têm sido propostas. De acordo com a estrutura elaborada pelos autores Scharstein e Szeliski (2002) a maioria dos algoritmos existentes realizam as etapas descritas na Figura 9.

Figura 9 – Estrutura adotada pelos algoritmos de correspondência estéreo

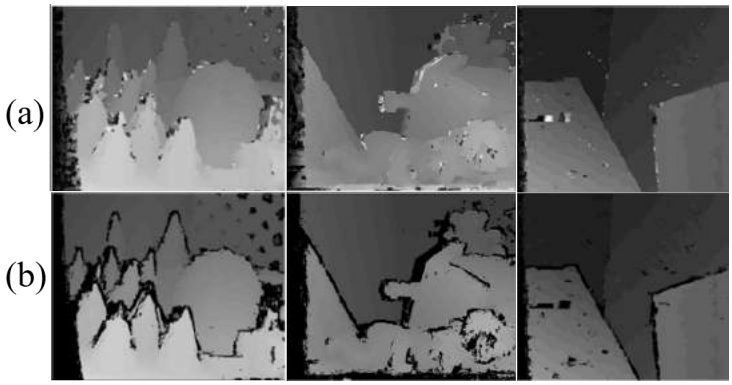


A etapa de *pré-processamento* é adotada para melhorar a qualidade dos dados contidos nas imagens fazendo com que o processo de busca consiga encontrar com mais precisão os

pixels correspondentes. Geralmente, funções de filtros são utilizadas para aliviar os ruídos provocados pelos sensores de câmera e distorções radiométricas. Exemplo de funções filtros comumente utilizados são filtros bilateral, filtros passa baixa tais como filtro de média, filtro laplaciano, laplaciano da gaussiana e filtros passa alta tal como filtro de sobel. Outras abordagens, além de filtros, também são adotadas tais como equalização de histogramas e conversão de cores. Particularmente, qualquer função que consiga tornar os dados de imagem mais descriminativos são apreciáveis nesta etapa.

A etapa de *pós-processamento* refina o mapa de disparidade. Métodos nesta etapa são utilizados para suprimir ruídos, suavizar os dados (geralmente utiliza-se filtro de média e interpolação), detectar regiões de oclusão, ou seja, regiões que são apenas visualizadas em uma imagem, e detectar resultados inconsistentes. Para detectar regiões de oclusão geralmente as abordagens criam dois mapas de disparidades no qual o sentido da busca pelos pixels correspondentes são diferentes para cada um destes mapas. Um mapa, denotado por D_L, é definido calculando-se o custo de correspondência no sentido da esquerda para direita, onde, a imagem de referência refere-se a imagem da esquerda e imagem de correspondência refere-se a imagem da direita. O outro mapa, denotado por D_R, é definido calculando-se o custo de correspondência no sentido da direita para a esquerda, onde, agora, a imagem de referência refere-se a imagem da direita e imagem de correspondência refere-se a imagem da esquerda.

Figura 10 – Exemplos de resultados com detecção de oclusão. (a) são mapas de disparidade sem detecção e (b) são mapas de disparidade com detecção



Estes dois mapas, D_L e D_R, possuem perspectivas diferentes, e nas regiões de oclusão, os resultados de disparidade de ambos os mapas geralmente diferem bastante. Assim para cada posição, é verificado se a diferença entre as disparidades de ambos os mapas ultrapassa um certo limiar fixo. Se sim, então esta posição é rotulada como pixel de oclusão, senão a disparidade a partir do mapa $D_L(i, j)$ é então retornada. Geralmente as regiões de oclusão estão localizadas próximas as bordas dos objetos como mostrado na Figura 10 (b) dos quais foram rotulados com valor 0 de intensidade. Esta Figura mostra exemplos de resultados de mapas de disparidade sem detecção de oclusão (a) e com as regiões de oclusão detectadas (b).

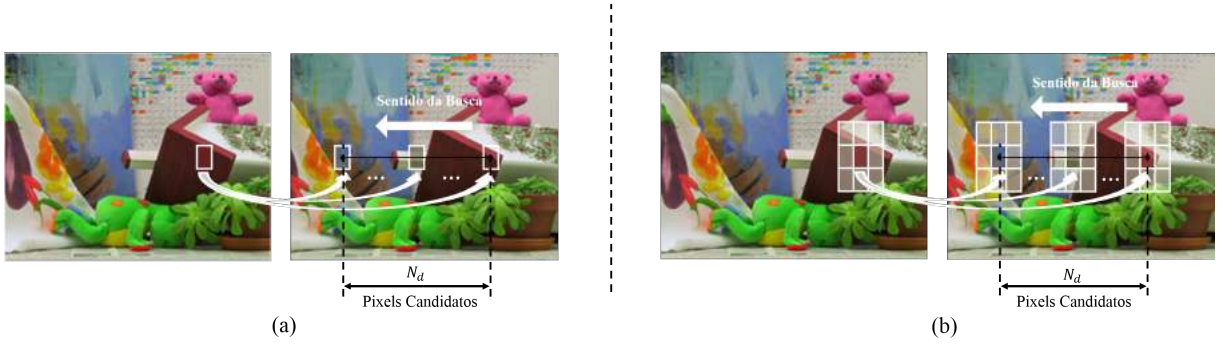
Tabela 1 – Abordagens Locais. R é o raio da janela, I_L, I_R são valores de intensidade do pixel na imagem da esquerda e direita respectivamente, $\overline{I}$ é a média de valores de intensidade dentro da janela, n é o número de pixel dentro da janela

Pontual	Diferença Absoluta (AD)	$C(i,j,d) = \lvert I_L(i,j) - I_R(i,j-d)\rvert$
Pontual	Diferença Quadrática (SD)	$C(i,j,d) = (I_L(i,j) - I_R(i,j-d))^2$
Baseado em região suporte	Soma de Diferenças Absoluta (SAD)	$C(i,j,d) = \sum_{i_w=i-W}^{i_w=i+R} \sum_{j_w=j-W}^{j_w=j+R} \lvert I_L(i_w,j_w) - I_R(i_w,j_w-d)\rvert$
Baseado em região suporte	Soma de Diferença Quadrática (SSD)	$C(i,j,d) = \sum_{i_w=i-W}^{i_w=i+R} \sum_{j_w=j-R}^{j_w=j+W} (I_L(i_w,j_w) - I_R(i_w,j_w-d))^2$
Baseado em região suporte	Correlação Cruzada Normalizada (NCC)	$C(i,j,d) = \frac{1}{n-1} \sum_{i_w=i-W}^{i_w=i+R} \sum_{j_w=j-W}^{j_w=j+R} ((I_L(i_w,j_w) - \overline{I_L})(I_R(i_w,j_w-d) - \overline{I_R}))^2$

A etapa de *cálculo de custo de correspondência* define o custo de similaridade de todos os pixels de referência, a partir da imagem de referência, em relação a todos os seus pixels candidatos, a partir da imagem da correspondência. A etapa de *cálculo de disparidade*, por sua vez, procura o pixel candidato que seja mais similar, baseado no seu valor de custo, e retorna a sua disparidade.

A etapa de *cálculo de custo de correspondência* é a mais importante de todo o sistema estéreo, uma vez que esta etapa determina diretamente a qualidade do mapa de disparidade gerado. Diversas abordagens para esta etapa têm sido desenvolvidas. As abordagens buscam lidar com todas as dificuldades anteriormente mostradas na Figura 7 com o objetivo de oferecer mapas de disparidades em alta qualidade. Os autores Scharstein e Szeliski (2002) classificam as abordagens existentes em duas grandes classes: as abordagens *locais* e abordagens *globais*.

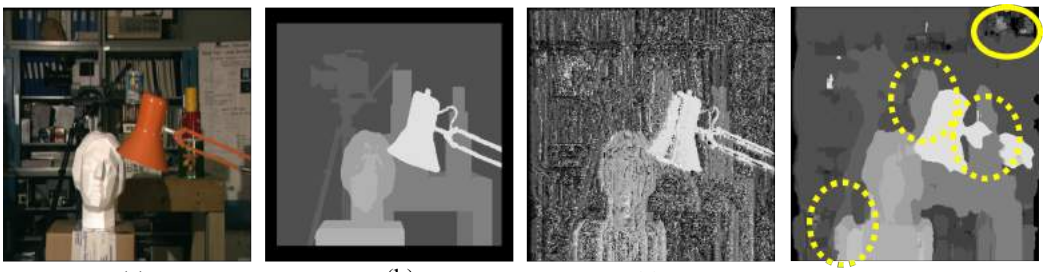
Figura 11 – Cálculo do custo de correspondência local. (a) Cálculo pontual (b) Cálculo baseado em regiões suporte



As abordagens locais ou baseadas em área realizam o cálculo do custo de correspondência baseado unicamente no valor de intensidade pontual ou sobre uma janela de suporte finita como mostrado respectivamente nas Figuras 11(a) e (b). Exemplos de funções pontuais e funções baseados em área são mostrados na Tabela 1.

As abordagens pontuais em geral definem mapas de disparidade bastante ruidosos, no sentido de que regiões de mesma profundidade real que deveriam ter o mesmo valor de disparidade, na verdade possuem valores valores bastante diferentes nos mapas de disparidade obtidos pela abordagem pontual. Isto acontece porque a determinação do custo envolve apenas o valor de intensidade do pixel de referência e o pixel candidato. Utilizar apenas uma medida pontual pode não ser suficiente, pois em uma situação prática, devido a ruído dos sensores e diferenças de iluminação, as imagens não possuem o mesmo valor de intensidade para a mesma região da cena. Isto faz com que pixels mais similares tenham grandes valores de custo de correspondência, o que não deveria acontecer, gerando assim, valores de disparidade diferentes dos pixels vizinhos que têm a mesma profundidade, como mostrado na Figura 12 (c). A Figura 12 (a) é uma das imagens estéreo retificada de entrada obtido a partir do benchmark Middlebury e a 12 (b) é o mapa de disparidade correto a partir desta imagem.

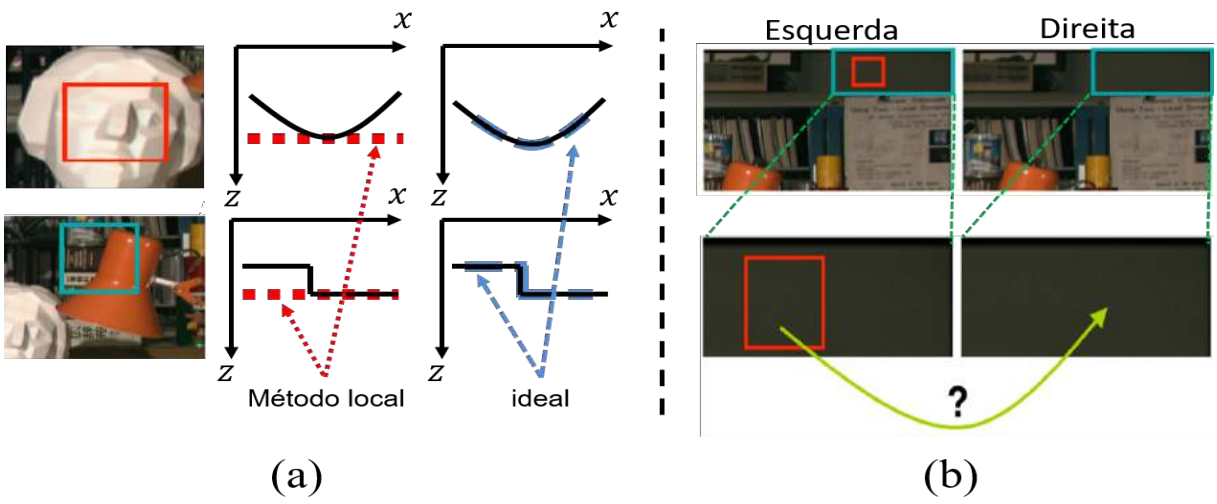
Figura 12 – Cálculo de custo de correspondência local. (a) Imagem Original (b) Mapa de disparidade correta (c) Mapa de disparidade obtido por correspondência local *pontual* (d) Mapa de disparidade obtido por correspondência local *baseado em área*



Este ruído pode ser reduzido quando o cálculo de custo é definido levando-se em conta os valores de intensidade dos pixels de vizinhança que estão dentro de uma região de suporte como mostrado na Figura 11 (b). Contudo, o aumento da região suporte gera erros de disparidade em regiões de descontinuidade de profundidade, tal como bordas e declives acentuados. Isto é devido a violação de profundidade constante dentro da região suporte, como evidenciado em Scharstein e Szeliski (2002) e mostrado na Figura 13 (a). Em outras palavras, para que abordagens locais baseadas em área obtenham resultados ótimos é necessário que dentro da janela suporte a profundidade seja constante, situação que é difícil de ser garantida em cenários com objetos com muitas profundidades diferentes. Como consequência, a medida em que a área de suporte aumenta as estruturas finas presentes em cenário real tenderão a ser engrossadas, bem como as bordas ficarão mais

arredondadas no resultado do mapa de disparidade, como mostrado pelos círculos pontilhados na Figura 12 (d). Soluções têm sido desenvolvidas para definir dinamicamente a forma dessa região suporte de maneira a garantir que dentro desta região a profundidade real seja constante, assim obtendo resultados melhores do que o mapa de disparidade produzido por abordagens com o formato fixo da região de suporte. Outras soluções têm buscado definir pesos dinamicamente para diferentes formas de regiões suporte de maneira a reforçar as formas mais adequadas para cada parte da imagem. Contudo, mesmo as melhores abordagem locais ainda não são capazes de lidar eficientemente com outras situações práticas tais como, regiões de baixa textura, como mostrado no canto superior direito da Figura 12 (d) e melhor descrito na Figura 13 (b), regiões repetitivas e regiões de diferenças de iluminação.

Figura 13 – Problemas enfrentados pelas abordagens locais baseados em área: (a) descontinuidade de profundidade e (b) baixa textura. A linha preta no gráfico indica a profundidade ideal e as linhas pontilhadas são os resultados calculados.



A fim de superar as dificuldades evidenciadas pelas abordagens locais, as abordagens globais têm sido adotadas. Nestas abordagens o problema de definição de disparidade é formulado como um problema de minimização de energia representado na Equação 2.12. Esta função define através de um valor numérico a qualidade do mapa de disparidade *inteiro*, de modo que a cada iteração este mapa vai sendo modificado, até que se obtenha um mapa com o menor valor de energia. Esta função combina dois termos: o termo chamado de energia de dados E_{data} e o termo chamado de energia de suavidade $E_{suavidade}$. O termo E_{data} é um valor numérico que mede quão bem a atribuição de disparidade se ajusta ao par de imagens estéreo e o termo $E_{suavidade}$ penaliza soluções de disparidade que não são suaves (parte-se da suposição que a disparidade de uma superfície não muda de valor). Na Equação 2.12 o termo D é uma solução de mapa de disparidade de dimensões H x W e o termo λ é um parâmetro de entrada que define o grau de suavização do mapa de disparidade resultante. Quanto maior for o valor de λ menos suave é a imagem.

$$E(D) = \sum_{i=0}^{i=H-1} \sum_{j=0}^{i=W-1} \Big[E_{data}(D,i,j) + \lambda E_{suavidade}(D,i,j) \Big] \tag{2.12}$$

onde

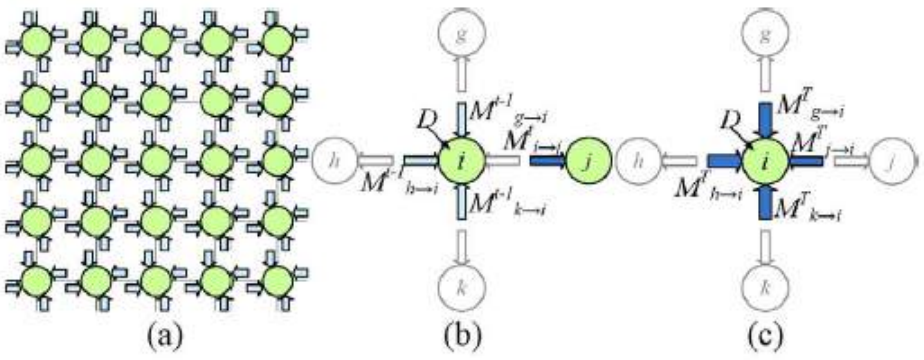
$$E_{data}(D,i,j) = |I_L(i,j) - I_R(i,j - D(i,j)|) \tag{2.13}$$

e

$$E_{suavidade}(D,i,j) = \sum_{k_i,k_j \in \{-1,0,1\}} |D(i,j) - D(i+k_i, j+k_j)| \tag{2.14}$$

Dado as imagens de entrada de dimensão H e W e um intervalo de disparidade N_d, existe um total de $(N_d)^{W \cdot H}$ combinações de disparidades possíveis. Encontrar uma solução de energia mínima é claramente um problema exponencial e portanto não é factível. Dessa forma, soluções de otimização tem sido propostas, tais como *Belief Propagation* (BP), *Graph Cut* (GC). Estes métodos geralmente modelam o problema de atribuição de disparidade através de grafos, onde cada pixel é um nó conectado aos seus vizinhos. Assim, o problema de otimização se resume a definir o rótulo (valor de disparidade) para cada nó (pixel) que leve a solução geral a um valor de energia mínimo com um número pequeno de iterações. O modo como se lida com o grafo e atualiza-se cada nó varia de abordagem para abordagem, mas geralmente, as abordagens dependem da informação dos nós vizinhos em iterações anteriores, para a atualização do nó atual.

Figura 14 – Ilustração do BP. (a) Matriz de Nós. (b) Passagem de mensagem. (c) Cálculo de Crença



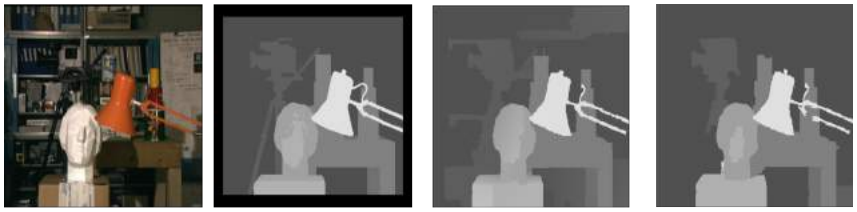
(TSENG; CHANG, 2010)

Por exemplo, o método BP minimiza a função de custo global através de um processo iterativo chamado de passagem de mensagem como mostrado na Figura 14 (a). Um nó i (que na verdade são os pixels) recebe mensagens de iterações anteriores a partir dos nós vizinhos com respeito a probabilidade de um determinado rótulo (que na verdade é a

disparidade), define um novo valor de probabilidade, e repassa este valor para o próximo nó, como mostrado na Figura 14 (b). A definição da nova probabilidade envolve técnicas de estimação bayesiana de probabilidade a posteriori, que leva em consideração funções de energia de suavidade, $E_{suavidade}$ e também de função de energia dos dados E_{data}. Esta passagem é feita ao longo de toda imagem. Depois que todas as mensagens são enviadas, cada nó atualiza a sua crença sobre cada rótulo, de acordo com a crença dos nós vizinhos, como mostrado na Figura 14 (c). Um rótulo com valor mais alto de crença significa que o nó (que na verdade é o pixel) acredita bastante que este rótulo seja o correto. Este processo de envio de mensagens precisa ser feito em várias iterações, para que os valores de crença convirja para valores ótimos.

As abordagens BP e GC buscam reduzir a função global obedecendo a restrição de suavidade em duas dimensões. Dessa forma, os esquemas de atualizações de nós possuem dependência bidimensional. A forte dependência não favorece implementações paralelas. Além disso, estes métodos necessitam de iteração para atingir o melhor mapa de disparidade que seria aquele com menor valor de energia. Iterações exigem grande quantidade de processamento, grandes quantidade de armazenamento para manter dados de cada iteração e exigem grandes larguras de banda para armazenar e carregar os dados.

Figura 15 – Cálculo de custo de correspondência global. (a) Imagem Original (b) Mapa de disparidade correta (c) Mapa de disparidade obtido através da técnica Corte de Grafo (GC) (d) Mapa de disparidade obtido através da técnica Propagação de Crença (BP)



Desta forma, embora o uso de benchmarks mostre uma melhoria significativa na qualidade do mapa de disparidade através de métodos globais, como exemplificados nas Figuras 15 (c) e (d), estes métodos não são atrativos para aplicações de tempo real, sobretudo devido a necessidade de iteração e a forte dependência de dados destes métodos (HALLER et al., 2010). Além disso, estas abordagens não são factíveis de serem implantadas em plataformas de baixo recurso computacional como nas plataformas FPGAs que possuem quantidade de memória e poder computacional limitadas. Como será explicado na seção 2.4, as plataformas FPGAs são atrativas para aplicações embarcadas pois possuem baixo consumo de energia e permite obter altas taxas de processamento da aplicação. Assim, as abordagens locais têm sido bastante requisitadas em tais plataformas, devido a sua simplicidade algorítmica e regularidade nas operações, mesmo pagando um custo com a baixa qualidade do mapa de disparidade.

2.3 Correspondência Estéreo Semi Global (SGM)

2.3.1 Teoria de Otimização Semi Global

Como elucidado na seção anterior, os métodos locais são métodos que definem o custo de correspondência baseado em um janela de tamanho finito, enquanto que os métodos globais definem custo de correspondência através da propagação dos custos de cada pixel ao longo de toda a imagem. As abordagens locais têm um baixo custo computacional, mas não são precisos em uma variedade de situações, tais como regiões de baixa textura, oclusões, regiões de borda e diferenças radiométricas. As abordagens globais, por outro lado, oferecem resultados satisfatórios para a maioria das situações onde a abordagem local falha, contudo, exigem um alto custo computacional, precisam de grande capacidade de armazenamento e tem uma forte dependência bidimensional entre os custos de pixels vizinhos processados. Entre estas duas grandes classes de métodos estão os métodos *semi globais*. Estes métodos focam na combinação de conceitos de métodos locais e métodos globais para obter mapas de disparidades precisos e com menor custo computacional.

O objetivo principal desta classe de algoritmos é atingir uma qualidade próximo ao obtido com os algoritmos globais com o desempenho próximo ao obtido pelos algoritmos locais. Isso é possível pois a abordagem semi-global realiza uma otimização aproximando o resultado de custos de correspondência local para um resultado de custos correspondência global em uma única iteração, diferentemente das abordagens globais que exigem várias iterações. Desta forma, a escolha do método local influencia diretamente na qualidade do mapa de disparidade resultante, pois ela é a solução a ser otimizada. Assim, é necessário escolher uma abordagem local que tente lidar ao máximo com os desafios evidenciados na seção 2.2. para que assim a otimização consiga refinar regiões que não foram possíveis de serem tratadas com a solução local.

$$E(D) = \sum_{\mathbf{p}} \Big[C(\mathbf{p}, D(\mathbf{p})) + \sum_{\mathbf{q} \in N_{\mathbf{p}}} P_1 T(|D(\mathbf{p}) - D(\mathbf{q})| = 1) + \sum_{\mathbf{q} \in N_{\mathbf{p}}} P_2 T(|D(\mathbf{p}) - D(\mathbf{q})| > 1) \Big] \tag{2.15}$$

A função global a ser otimizada pelos algoritmos semi-globais combina o *custo de correspondência local* e *restrições de suavidade global*, como mostrado na Equação 2.15. O primeiro termo é a soma dos custos de correspondência de todos os pixels $\mathbf{p} = (p_i, p_j)$ a partir dos resultados de disparidades da matriz D. A matriz D refere-se ao mapa de disparidade resultante obtido a partir da função de custo de correspondência local $C(\mathbf{p}, d)$ e através da busca pela disparidade de menor custo. O segundo termo adiciona uma constante P_1 para todos os pixels $\mathbf{q} = (q_i, q_j)$ na vizinhança $N_{\mathbf{p}}$ do pixel $\mathbf{p}$, para o qual a mudança de disparidade varia pouco em relação ao pixel central $\mathbf{p}$ (ou seja, a diferença entre o pixel $\mathbf{p}$ e um vizinho $\mathbf{q}$ é de 1 unidade). O terceiro termo adiciona uma penalidade

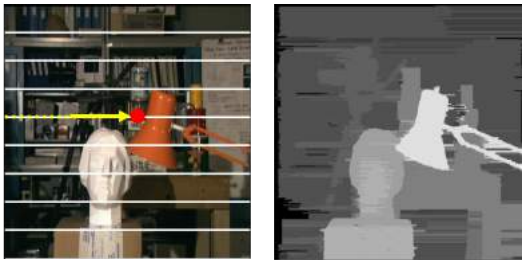
P_2, maior que P_1, para todas as mudanças grandes de disparidade na vizinhança $N_{\mathbf{p}}$ do pixel $\mathbf{p}$, ou seja, a diferença entre o pixel $\mathbf{p}$ e um vizinho $\mathbf{q}$ é maior que 1 unidade. A função T retorna 1 se a condição for verdadeira caso contrário retorna 0.

Esta função global é ligeiramente diferente da função de custo global definida na seção 2.2, usada pela maioria dos algoritmos globais. Ao usar uma penalidade menor P_1 para mudanças pequenas de disparidade na vizinhança, a função permite que a otimização semi global se adapte para superfícies suavemente curvadas e inclinadas, e ao usar penalidade maior P_2 para mudanças de disparidades maiores na vizinhança, a função permite que a otimização semi global preserve as grandes descontinuidades de profundidade como descrito em Boykov, Veksler e Zabih (2001). Esta característica para preservar ao mesmo tempo descontinuidades e suavizar regiões planas é um dos diferenciais das abordagens semi globais.

Ao contrário dos métodos globais convencionais que realizam a otimização da função de energia global obedecendo às restrições de suavidade em duas dimensões, as abordagens semi-globais realizam a otimização global obedecendo às restrições de suavidade em apenas *uma dimensão*. O resultado é uma redução na complexidade computacional do problema de otimização para tempo polinomial como evidenciado por Meerbergen et al. (2002). Similarmente ao algoritmo global Belief Propagation (este algoritmo foi descrito na seção 2.2), os algoritmos semi-globais podem ser descritos como passagem de mensagem. Contudo, ao contrário do algoritmo BP, a passagem de mensagem depende apenas do vizinho em apenas uma dimensão.

O algoritmo semi-global pioneiro é a abordagem por *programação dinâmica*. Esta abordagem realiza a otimização em uma dimensão para cada linha da imagem como mostrado na Figura 16 (a). Neste tipo de abordagem o sentido da otimização acompanha apenas as linhas epipolares da imagem. Esta abordagem facilmente sofre do efeito de estriamento horizontal como mostrado na Figura 16 (b), devido justamente ao não relacionamento de otimizações de várias linhas individuais da imagem a fim de que as restrições de suavidade bidimensionais sejam atendidas.

Figura 16 – Resultado do método de programação dinâmica

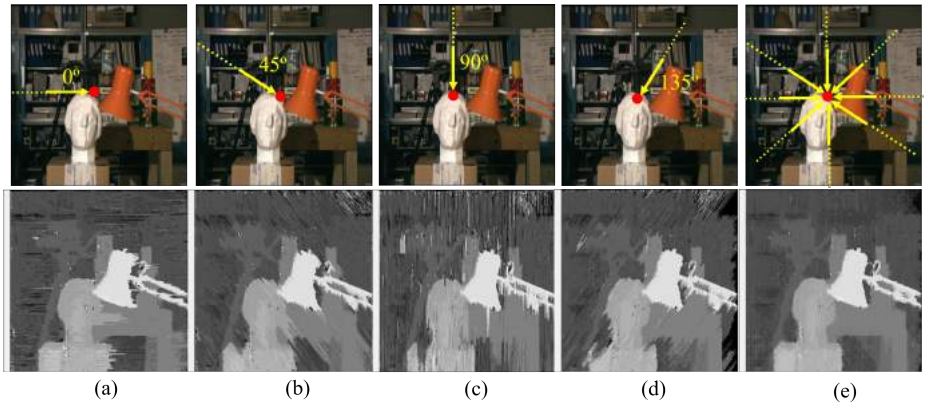


(a) (b)

Para resolver os problemas de estriamento do método de programação dinâmica, a técnica semi global proposta pelo autor Hirschmüller (HIRSCHMULLER, 2008) (SGM)

aproxima uma otimização bidimensional *agregando* vários funções de otimizações independentes, onde cada otimização compreende um caminho unidimensional de ângulos diferentes, como mostrados nas Figuras 17(a), (b), (c) e (d). A agregação dos diferentes caminhos localizados em diferentes ângulos permite que a otimização semi global remova o estriamento presente nos algoritmos de programação dinâmica, como mostrado na Figura 17(e). Esta aproximação por agregação tem permitido a obtenção de resultados com precisão similares aos métodos globais bidimensionais ou até superiores enquanto mantém o tempo de execução reduzido em relação a estes métodos.

Figura 17 – Resultado da otimização por caminho e a agregação de todas as otimizações do método semi global



Uma vez que o tópico de pesquisa deste trabalho é baseado na técnica Semi Global (SGM) proposta por Hirschmüller (HIRSCHMULLER, 2008), toda vez que o termo semi global ou SGM aparecer este está relacionado com a técnica proposta por este autor.

2.3.2 Computação do SGM em imagens estéreo

A minimização da função de custo mostrada na Equação 2.15 é realizada pela abordagem SGM através da *propagação* de custos ao longo de caminhos simétricos unidimensionais e posterior *agregação* destes custos, como mostrado na Figura 19. Todos os caminhos se encontram em todos os pixels da imagem.

Formalmente, cada custo de caminho $L_{\mathbf{r}^t}$ na direção $\mathbf{r}^t = (r_i^t, r_j^t)$, na disparidade d e para o pixel $\mathbf{p} = (p_i, p_j)$ é definido recursivamente da seguinte forma:

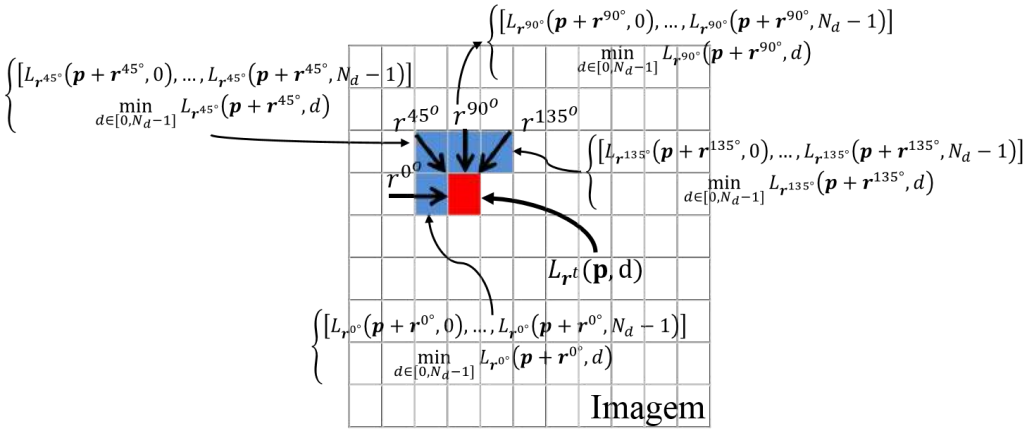
$$\begin{aligned} L_{\mathbf{r}^t}(\mathbf{p}, d) = C(\mathbf{p}, d) + \min[L_{\mathbf{r}^t}(\mathbf{p} + \mathbf{r}^t, d), L_{\mathbf{r}^t}(\mathbf{p} + \mathbf{r}^t, d-1) + P_1, \\ L_{r^t}(\mathbf{p} + \mathbf{r}^t, d+1) + P_1, \min_{i \in [0, N_d - 1]} L_{\mathbf{r}^t}(\mathbf{p} + \mathbf{r}^t, i) + P_2] - \min_{k \in [0, N_d - 1]} L_{\mathbf{r}^t}(\mathbf{p} + \mathbf{r}^t, k) \end{aligned} \tag{2.16}$$

onde $C(\mathbf{p}, d)$ é uma função de custo de correspondência inicial obtida com o uso de algoritmos locais como já explicado na seção 2.2. Os termos $L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, d)$, $L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, d-1)$, $L_{r^t}(\mathbf{p} + \mathbf{r}^t, d+1)$ são custos de caminhos de pixels anteriores que já foram calculados. O segundo termo adiciona o menor custo de caminho do pixel anterior $\mathbf{p} + \mathbf{r}^t$ incluindo

a penalidade apropriada para mudanças de disparidades. Para custos de caminhos anteriores cujas disparidades variam de uma unidade em relação a disparidade atual d (ou seja, $|\delta d| = 1$ unidade) adiciona-se a penalidade P_1 e para custos de caminhos anteriores cujas disparidades variam de mais de uma unidade (ou seja, $|\delta d| > 1$ unidade) adiciona-se a penalidade P_2 maior que P_1. O termo $\min_{i \in [0,N_d-1]} L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, i)$ é exatamente o custo de caminho do pixel anterior, cuja disparidade é maior que 1. A penalidade P_1 é empiricamente constante e P_2 pode ser constante ou dinamicamente modificado através do conteúdo da imagem, garantindo que sempre a condição $P_1 < P_2$ seja estabelecida. O uso das duas penalidades faz com que a otimização lide ao mesmo tempo com superfícies suaves e inclinadas (devido a P_1) e preserve descontinuidades de profundidade na cena (devido a P_2). O termo de subtração no final da equação evita que o custo de caminho aumente permanentemente ao longo do caminho, podendo conduzi-lo a valores infinitamente altos como observado por Hirschmüller (HIRSCHMULLER, 2008). Com este termo o limite superior do valor do custo de caminho é dado como $L \leq C_{max} + P_2$, onde C_{max} é o maior valor que a função de custo local $C(\mathbf{p}, d)$ pode ter.

Como pode ser observado, a partir da Equação 2.16, os termos $L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, d)$, $L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, d-1)$, $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d+1)$ e $\min_{i \in [0,N_d-1]} L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, i)$ são custos de caminhos em relação a posição do pixel anterior $\mathbf{p}+\mathbf{r}^t$, que são utilizados na computação do custo de caminho do pixel atual $L_{\mathbf{r}^t}(\mathbf{p}, d)$, como mostrado na Figura 18. A posição do pixel anterior é determinado pelo ângulo da reta t no qual a propagação do custo está sendo realizada.

Figura 18 – Uso dos custos de caminho de pixels anteriores para direções diferentes na computação do custo de caminho do pixel atual



Assim, para cada caminho de ângulo t, $\mathbf{r}$ assume diferentes valores, como por exemplo, $\mathbf{r}^{0°} = (0,-1)$, $\mathbf{r}^{45°} = (-1,-1)$, $\mathbf{r}^{90°} = (-1,0)$, $\mathbf{r}^{135°} = (-1,1)$, $\mathbf{r}^{180°} = (0,1)$, $\mathbf{r}^{225°} = (1,1)$, $\mathbf{r}^{270°} = (0,1)$, $\mathbf{r}^{315°} = (1,-1)$. Para facilitar o entendimento, as Equações 2.17, 2.18, 2.19 e 2.20 são as funções de custo de caminho respectivamente para as direções $\mathbf{r}^{0°}$, $\mathbf{r}^{45°}$, $\mathbf{r}^{90°}$ e $\mathbf{r}^{135°}$.

$$\begin{aligned}\underline{L_{\mathbf{r}^{0^\circ}}(i,j,d)} = C(i,j,d) + \min[&L_{\mathbf{r}^{0^\circ}}(i,j-1,d),\\ &L_{\mathbf{r}^{0^\circ}}(i,j-1,d-1)+P_1, L_{\mathbf{r}^{0^\circ}}(i,j-1,d+1)+P_1,\\ &\min_{i\in[0,N_d-1]} L_{\mathbf{r}^{0^\circ}}(i,j-1,i)+P_2] - \min_{k\in[0,N_d-1]} L_{\mathbf{r}^{0^\circ}}(i,j-1,k)\end{aligned} \tag{2.17}$$

$$\begin{aligned}\underline{L_{\mathbf{r}^{45^\circ}}(i,j,d)} = C(i,j,d) + \min[&L_{\mathbf{r}^{45^\circ}}(i-1,j-1,d),\\ &L_{\mathbf{r}^{45^\circ}}(i-1,j-1,d-1)+P_1, L_{\mathbf{r}^{45^\circ}}(i-1,j-1,d+1)+P_1,\\ &\min_{i\in[0,N_d-1]} L_{\mathbf{r}^{45^\circ}}(i-1,j-1,i)+P_2] - \min_{k\in[0,N_d-1]} L_{\mathbf{r}^{45^\circ}}(i-1,j-1,k)\end{aligned} \tag{2.18}$$

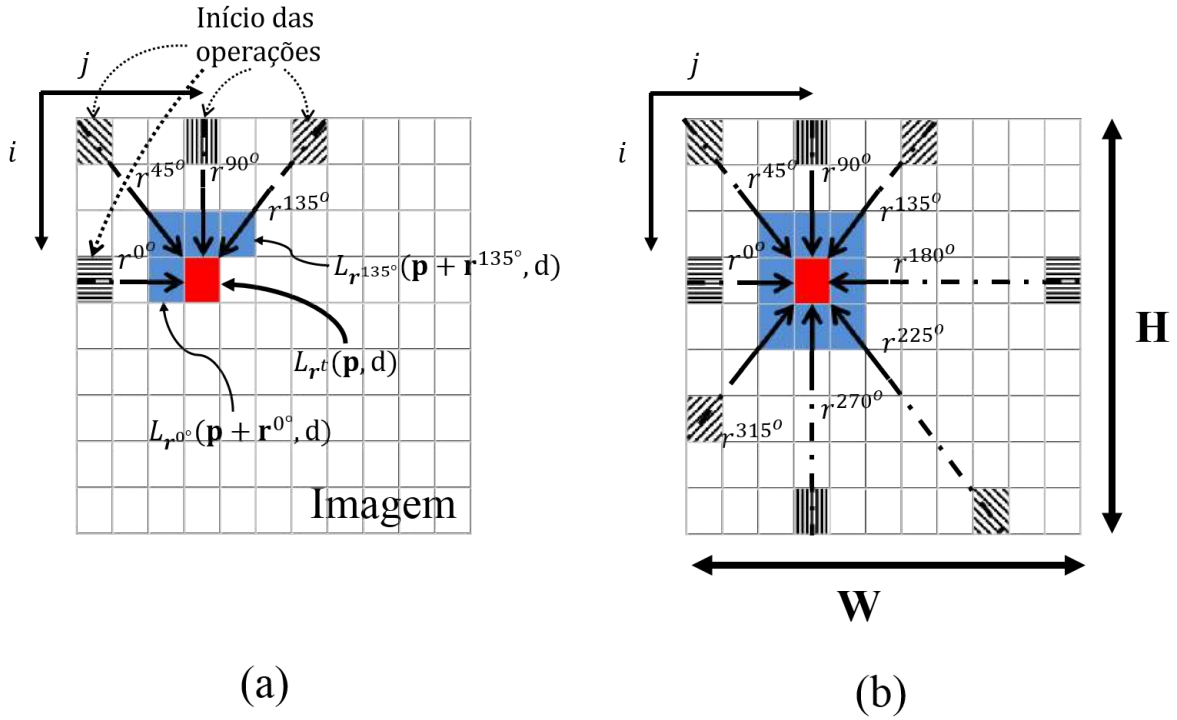
$$\begin{aligned}\underline{L_{\mathbf{r}^{90^\circ}}(i,j,d)} = C(i,j,d) + \min[&L_{\mathbf{r}^{90^\circ}}(i-1,j,d),\\ &L_{\mathbf{r}^{90^\circ}}(i-1,j,d-1)+P_1, L_{\mathbf{r}^{90^\circ}}(i-1,j,d+1)+P_1,\\ &\min_{i\in[0,N_d-1]} L_{\mathbf{r}^{90^\circ}}(i-1,j,i)+P_2] - \min_{k\in[0,N_d-1]} L_{\mathbf{r}^{90^\circ}}(i-1,j,k)\end{aligned} \tag{2.19}$$

$$\begin{aligned}\underline{L_{\mathbf{r}^{135^\circ}}(i,j,d)} = C(i,j,d) + \min[&L_{\mathbf{r}^{135^\circ}}(i-1,j+1,d),\\ &L_{\mathbf{r}^{135^\circ}}(i-1,j+1,d-1)+P_1, L_{\mathbf{r}^{135^\circ}}(i-1,j+1,d+1)+P_1,\\ &\min_{i\in[0,N_d-1]} L_{\mathbf{r}^{135^\circ}}(i-1,j+1,i)+P_2] - \min_{k\in[0,N_d-1]} L_{\mathbf{r}^{135^\circ}}(i-1,j+1,k)\end{aligned} \tag{2.20}$$

É importante observar que a propagação necessita de um ponto de partida. O ponto de partida são os pixels de bordas como mostrado pelos pixels listrados na Figura 19. Nestas posições de borda o cálculo do custo de caminho necessita do custo de caminho anterior que reside fora da imagem, ou seja, não tem definição deste custo. Por exemplo, para calcular o custo de caminho $L_{\mathbf{r}^{0^\circ}}(0,0,0)$ precisa-se do custo de caminho anterior $L_{\mathbf{r}^{0^\circ}}(0,0-1,0)$ que está fora do espaço de imagem. Em casos como este o autor Hirschmüller (HIRSCHMULLER, 2008) define que o resultado é apenas o custo inicial $C(\mathbf{p},d)$, ou seja, $L_{\mathbf{r}^t}(\mathbf{p},d) = C(\mathbf{p},d)$. No caso do exemplo anterior então $L_{\mathbf{r}^{0^\circ}}(0,0,0) = C(0,0,0)$.

Situações similares ao que acontece quando a posição do custo anterior está fora dos limites da imagem, também acontece quando a variável solicitada é de uma disparidade negativa ou o valor da disparidade é maior que $N_d - 1$ (valor máximo de disparidade) ou quando o valor de intensidade solicitado é de um pixel $I_R(i, j-d)$ cuja a posição está fora da imagem. Por exemplo, para calcular o custo de caminho $L_{\mathbf{r}^{0^\circ}}(0,1,0)$ precisa-se do custo de caminho $L_{\mathbf{r}^{0^\circ}}(0,1-1,0-1)$ no qual a disparidade tem valor negativo. Estas variáveis são consideradas inexistentes e precisam ser detectadas e removidas durante o cálculo do custo de caminho atual de modo a manter a precisão da Equação 2.16. Uma solução é definir para estas variáveis inexistentes valores infinitamente altos, pois valores de custo bastante grandes terão pequenas chances de serem escolhidos nas funções de mínimo da Equação 2.16, e assim nunca participarão de fato da cálculo desta equação.

Figura 19 – Sentido de propagação de cada caminho na contribuição do custo de correspondência de um determinado pixel e a quantidade de caminhos utilizados. (a) Quatro caminhos de propagação e (b) oito caminhos de propagação



O custo de correspondência final otimizado ao longo de toda imagem é alcançado através da *agregação* de todos os custos de caminhos $L_{\mathbf{r}^t}(\mathbf{p}, d)$ a partir do conjunto de direções S_t consideradas para um pixel ao longo de N_d valores de disparidades, como mostrado na Equação 2.21. Geralmente a agregação é realizada através do somatório. A quantidade de caminhos considerada nos trabalhos relacionados estão entre 4 e 8 como mostrado respectivamente nas Figuras 19 (a) e (b). Quanto mais caminhos de propagação, melhor a qualidade do mapa de disparidade pois ele agregará mais custos de caminhos de direções diferentes no cálculo do custo final.

$$S(\mathbf{p}, d) = \sum_{t \in \{S_t\}} L_{\mathbf{r}^t}(\mathbf{p}, d) \tag{2.21}$$

Uma vez definido os custos de caminho, o valor agregado destes custos são computados. A próxima etapa, *computação de disparidade* (como já explicado na seção 2.2), define o mapa de disparidade resultante D_L através da abordagem winner-takes-all (WTA), como mostrado na Equação 2.22. Basicamente esta abordagem retorna a disparidade cujo o valor de custo foi o menor entre os N_d custos finais $S(\mathbf{p}, d)$.

$$D_L(\mathbf{p}) = d|_{\min_{d \in [0, N_d - 1]}} S(\mathbf{p}, d) \tag{2.22}$$

A detecção de regiões de oclusão usando a abordagem SGM acontece primeiramente definindo um segundo mapa de disparidade denotado por D_R. Este mapa possui uma

perspectiva ligeiramente diferente do mapa de disparidade D_L (como já dito na seção 2.2), onde nas regiões de oclusão os dados de disparidade de ambos os mapas geralmente diferem bastante. Assim, para cada posição no mapa D_L, é verificado se seu valor de disparidade difere bastante do valor de disparidade no mapa D_R medido através de um limiar fixo, como mostrado na Equação 2.23. Se sim, então esta disparidade é rotulada como ocludido ou inválido, senão a disparidade do mapa D_L é então retornada para compor o mapa final D. Para cada pixel ocludido que foi detectado é atribuído um valor fixo no mapa de disparidade D. Esta abordagem de detecção de oclusão, além de servir para encontrar regiões de oclusão permite encontrar disparidades que foram mal correspondidas, servindo assim como um medidor de qualidade do resultado do algoritmo SGM.

$$D(\mathbf{p}) = \begin{cases} D_L(\mathbf{p}), & \text{if } |D_L(\mathbf{p}) - D_R(\mathbf{p})| \leq \sigma \\ D_{inv}, & \text{Caso Contrário} \end{cases} \tag{2.23}$$

A abordagem clássica define o mapa D_R realizando novamente, a partir do zero, todo o processo de cálculo de custo (pré processamento, cálculo inicial, cálculo do custo de caminho, agregação do custo de caminho e definição de disparidade), mudando apenas a ordem de busca dos pixels correspondentes. Ou seja, a imagem da esquerda que antes era a imagem de referência na definição do mapa D_L se torna a imagem de correspondência e a imagem da direita que antes era a imagem de correspondência na definição do mapa D_L se torna a imagem de referência. Nessa inversão é importante lembrar que a posição de cada pixel candidato é $(p_i, p_j + d)$ ao contrário da primeira abordagem onde a localização do pixel candidato é $(p_i, p_j - d)$. Esta abordagem é a mais convencional e a que permite encontrar corretamente mais pixels ocludidos. Contudo, replicar todo o processo talvez não seja factível em plataformas de hardware com poucos recursos lógicos de processamento e de armazenamento, além de ser necessário implementar lógicas para parear os dois resultados.

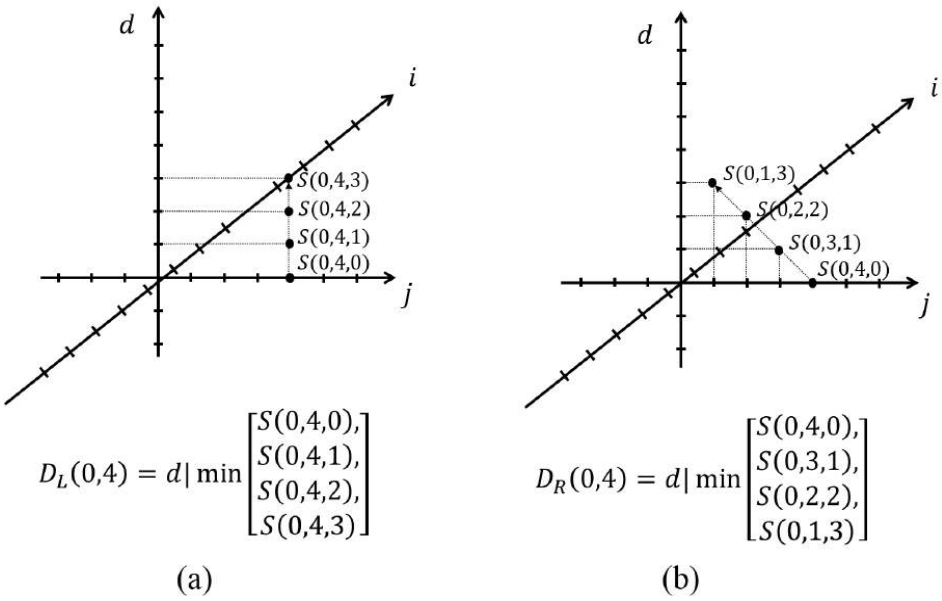
Assim, uma segunda abordagem, utilizada na maioria das abordagens em hardware para computação do SGM, gera o mapa o D_R a partir diretamente do resultado da matriz tridimensional S da Equação 2.21, como mostrado na Equação 2.24.

$$D_R(\mathbf{p}) = D_R(p_i, p_j) = d | \min_{d \in [0, N_d - 1]} S(p_i, p_j - d, d) \tag{2.24}$$

Esta equação é similar à Equação 2.22 com a diferença de que, para determinar $D_R(p_i, p_j)$, é realizada a busca pelo valor de disparidade de menor custo com relação a valores de S cujas posições variam tanto no eixo das disparidades d como no eixo das colunas j. Isto é devido a posição da coluna p_j, referente ao pixel que se queira calcular a sua disparidade, ser subtraído da disparidade d para determinar a posição no eixo da coluna do custo a ser buscado, ou seja, $p_j - d$, como pode ser observado na Equação 2.24. Para melhorar o entendimento, para calcular $D_L(p_i, p_j)$ a busca da disparidade de custo mínimo é realizada em relação aos valores $S(p_i, p_j, 0), S(p_i, p_j, 1), ..., S(p_i, p_j, N_d - 1)$, enquanto que para calcular $D_R(p_i, p_j)$, a busca da disparidade de custo mínimo é realizada

em relação aos valores $S(p_i, p_j - 0, 0), S(p_i, p_j - 1, 1), ..., S(p_i, p_j - N_d - 1, N_d - 1)$. Na Figura 20(a) e (b) são mostrados exemplos da determinação da disparidade para $D_L(0, 4)$ e $D_R(0, 4)$ respectivamente com um intervalo de disparidade N_d igual a quatro. Observe que para calcular D_L, a busca acontece variando-se apenas a posição das disparidades, e mantendo-se as posições $(p_i, p_j) = (0, 4)$ constantes. Para calcular D_R a busca acontece variando-se as posições de disparidade, d, e de coluna, $p_j - d$ e mantendo-se constante a posição de linha p_i. Na literatura esta abordagem para detecção de oclusão é chamada de *checagem LR rápida*.

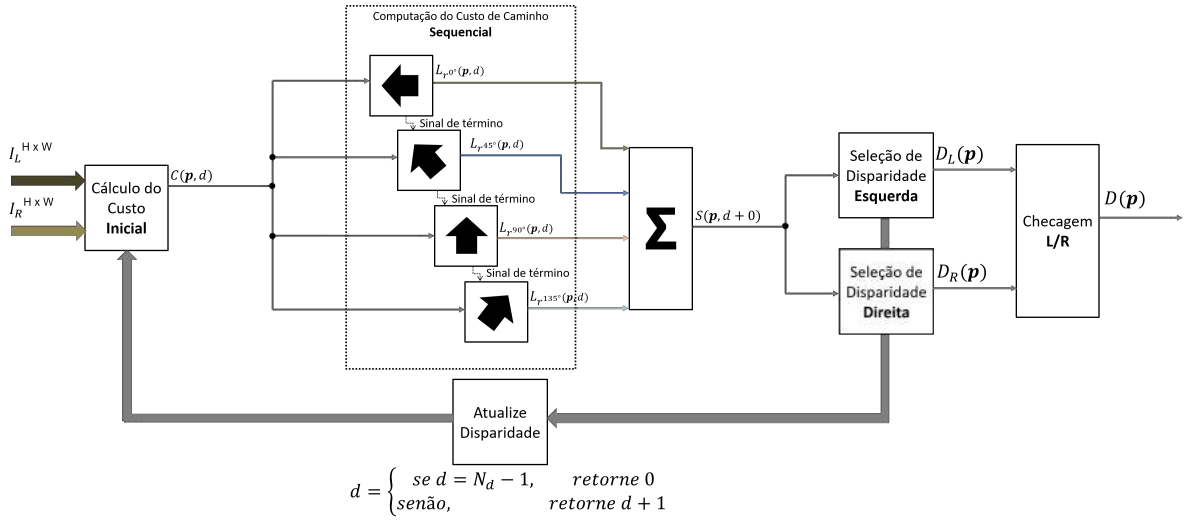
Figura 20 – Sentido da busca pela disparidade de menor custo na geração dos mapas de disparidades (a) D_L e (b) D_R



As etapas para o cálculo de todas as equações descritas anteriormente são ilustradas no diagrama de etapas mostrado na Figura 21. Geralmente o fluxo de processamento começa computando-se o custo inicial $C(\mathbf{p}, d)$, o qual é adicionado ao custo de caminho $L_{\mathbf{r}^t}(\mathbf{p}, d)$ para todos os caminhos $\mathbf{r}^t$ paralelamente. O resultado do custo de caminho para cada pixel $\mathbf{p}$ e disparidade d são agregados através do somatório para gerar $S(\mathbf{p}, d)$. A partir de S são definidos os dois mapas de disparidades $D_L(\mathbf{p})$ e $D_R(\mathbf{p})$ paralelamente. Estes dois mapas são utilizados na etapa de detecção de oclusão, chamado de *checagem L/R* para computar como resposta final o mapa de disparidade $D(\mathbf{p})$.

A abordagem de implementação mais simples e mais ingênua do SGM é mostrada através do pseudocódigo 1, considerando apenas as direções $\mathbf{r}^{0^\circ}$ e $\mathbf{r}^{45^\circ}$. Esta abordagem é mostrada como proposta didática. Neste pseudocódigo é possível perceber o processo iterativo do algoritmo SGM onde, para cada iteração d, todas as etapas descritas na Figura 21 são realizadas sequencialmente. Também pode-se perceber que o fluxo de acesso aos pixels da imagem começa no topo superior esquerdo devido as condições de bordas descritos na Figura 19.

Figura 21 – Etapas envolvidas na abordagem SGM



Vamos detalhar mais um pouco o pseudocódigo 1 a fim de esclarecer todo o processo de computação do algoritmo SGM. Primeiro, são alocados espaços em memória para as matrizes tridimensionais de custo de caminho $\mathbf{r}^{0°}$, $\mathbf{r}^{45°}$ custo final S (linhas 3, 4 e 5 do pseudocódigo 1 respectivamente), bem como, a matriz bidimensional final de disparidade D (linha 2 do pseudocódigo 1). A adição de dois espaços para a disparidade é feito para tratar das condições de disparidade $d = -1$ e $d = N_d$ que passam do espaço permitido de disparidade que é entre 0 e $N_d - 1$. Então, quando for buscada a posição $d = -1$ na matriz L, na verdade é a posição $d = 0$ que está sendo acessada, e quando for buscada posição de N_d na matriz L na verdade é a posição $N_d + 1$ que está sendo acessada.

Três declarações de loop aninhadas são realizadas, no qual para cada posição de linha, todas as colunas são percorridas e para cada posição de coluna todas as disparidades são percorridas. Para cada posição de coluna as variáveis de custo final *best_cost_D_L* e *best_cost_D_R* são definidas como valores infinitos (ou na prática, o maior valor representável pela variável no computador) para que a cada iteração de disparidade um novo custo de D_L e D_R possam substituir o valor armazenado nestas variáveis. Dentro do loop de disparidades, primeiro é definido o custo inicial *C_i_j_d*. Se $j - d$ é negativo, isto indica que o valor não é válido para indexar uma posição na imagem. Neste caso, é atribuído um valor de infinito (ou na prática, o maior valor representável pela variável no computador). Caso $j - d$ não seja negativo então é atribuído o valor de diferença absoluta da intensidade do pixel $I_L(i, j)$ da imagem da esquerda com a intensidade do pixel da imagem da direita $I_R(i, j - d)$.

A próxima etapa é calcular o custo de caminho para $\mathbf{r}^{0°}$ (entre as linhas 18 e 21 do pseudocódigo 1) e $\mathbf{r}^{45°}$ (entre as linhas 23 e 26 do pseudocódigo 1). No caso do custo de caminho $\mathbf{r}^{0°}$, primeiro é verificado se o valor da posição de coluna do pixel anterior $j - d$, cujo custo de caminho está sendo solicitado, é negativo, valor que não é válido para indexar uma posição na imagem. Se sim, então é atribuído para a variável $L_{\mathbf{r}^{0°}}(i, j, d)$

apenas o custo inicial $C\_i\_j\_d$. Se não, é calculado de fato a função de custo de caminho, acessando os valores de custo de caminho de todas as disparidades do pixel anterior na posição $(i, j-1)$ e adicionando as penalidades constantes P_1 e P_2 corretamente. Este mesmo procedimento é feito para calcular $\mathbf{r}^{45^\circ}$. No entanto, neste caso, faz-se a verificação das condições de borda para $i-1$ e para $j-1$, que é justamente a posição do pixel anterior que está na linha e coluna anterior da imagem respectivamente.

Depois de calculados os custos de caminho $L_{\mathbf{r}^{0^\circ}}(i,j,d)$ e $L_{\mathbf{r}^{45^\circ}}(i,j,d)$ para uma determinada disparidade d, estes custos são somados para determinar o custo final otimizado $S(i,j,d)$ (linha 28 do pseudocódigo).

Em seguida, o custo final $S(i,j,d)$ é verificado se é menor que o melhor custo atual $best\_cost\_D\_L$ que será usado para definir a disparidade $D_L(i,j)$ (entre as linhas 29 e 32 do pseudocódigo 1). Se sim, $best\_cost\_D\_L$ recebe este valor de custo de $S(i,j,d)$ e o $best\_disp\_D\_L$ recebe a disparidade d da iteração atual.

Algorithm 1 Algoritmo SGM

1: **procedure** SGM($I_L, I_R, H, W, N_d, P_1, P_2, \sigma$)
2: $D \leftarrow allocate\_matrix(H, W)$
3: $L_{\mathbf{r}^{0^\circ}} \leftarrow allocate\_matrix(H, W, N_d + 2)$
4: $L_{\mathbf{r}^{45^\circ}} \leftarrow allocate\_matrix(H, W, N_d + 2)$
5: $S \leftarrow allocate\_matrix(H, W, N_d)$
6: **for** $i = 0; i < 0; i = i + 1$ **do**
7: **for** $j = 0; j < 0; j = j + 1$ **do**
8: $best\_disp\_D_L \leftarrow 0$
9: $best\_cost\_D_L \leftarrow \infty$
10: $best\_disp\_D_R \leftarrow 0$
11: $best\_cost\_D_R \leftarrow \infty$
12: **for** $d = 0; d < N_d - 1; d = d + 1$ **do**
13: **if** $j - d \geq 0$ **then** ▷ Verificação de Limites da Imagem
14: $C\_i\_j\_d \leftarrow |I_L(i,j) - I_R(i,j-d)|$
15: **else**
16: $C\_i\_j\_d \leftarrow \infty$
17: **end if**
18: **if** $j - 1 < 0$ **then** ▷ Caso base de $L_{\mathbf{r}^{0^\circ}}(i,j,d)$
19: $\underline{L_{\mathbf{r}^{0^\circ}}(i,j,d)} \leftarrow C\_i\_j\_d$
20: **else**
21: $\underline{L_{\mathbf{r}^{0^\circ}}(i,j,d)} \leftarrow C\_i\_j\_d + \min[L_{\mathbf{r}^{0^\circ}}(i,j-1,d),$
$L_{\mathbf{r}^{0^\circ}}(i,j-1,d-1) + P_1, L_{\mathbf{r}^{0^\circ}}(i,j-1,d+1) + P_1,$
$\min_{i\in[0,N_d-1]} L_{\mathbf{r}^{0^\circ}}(i,j-1,i) + P_2] - \min_{k\in[0,N_d-1]} L_{\mathbf{r}^{0^\circ}}(i,j-1,k)$
22: **end if**
23: **if** $i - 1 < 0$ OR $j - 1 < 0$ **then** ▷ Caso base de $L_{\mathbf{r}^{45^\circ}}(i,j,d)$

24: $\underline{L_{\mathbf{r}^{0°}}(i,j,d)} \leftarrow C\_i\_j\_d$
25: **else**
26: $\underline{L_{\mathbf{r}^{45°}}(i,j,d)} \leftarrow C\_i\_j\_d + \min[L_{\mathbf{r}^{45°}}(i-1,j-1,d),$
$L_{\mathbf{r}^{45°}}(i-1,j-1,d-1)+P_1, L_{\mathbf{r}^{45°}}(i-1,j-1,d+1)+P_1,$
$\min_{i\in[0,N_d-1]} L_{\mathbf{r}^{45°}}(i-1,j-1,i)+P_2]$
$-\min_{k\in[0,N_d-1]} L_{\mathbf{r}^{45°}}(i-1,j-1,k)$
27: **end if**
28: $\underline{S(i,j,d)} \leftarrow L_{\mathbf{r}^{0°}}(i,j,d) + L_{\mathbf{r}^{45°}}(i,j,d)$
29: **if** $S(i,j,d) < best\_cost\_D_L$ **then**
30: $\underline{best\_disp\_D_L} \leftarrow d$
31: $\underline{best\_cost\_D_L} \leftarrow S(i,j,d)$
32: **end if**
33: **if** $j-d<0$ AND $S(i,j-d,d) < best\_cost\_D_R$ **then**
34: $\underline{best\_disp\_D_R} \leftarrow d$
35: $\underline{best\_cost\_D_R} \leftarrow S(i,j-d,d)$
36: **end if**
37: **end for**
38: **if** $|best\_disp\_D_L - best\_disp\_D_R| > \sigma$ **then**
39: $\underline{D(i,j)} \leftarrow 255$ ▷ codificação para oclusão
40: **else**
41: $\underline{D(i,j)} \leftarrow best\_disp\_D_L$
42: **end if**
43: **end for**
44: **end for**
45: **return** D
46: **end procedure**

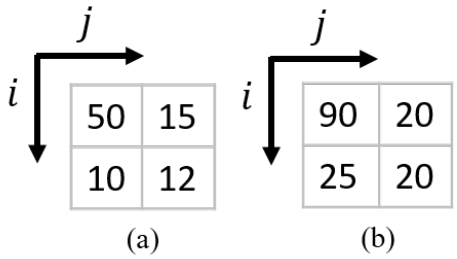
Em seguida, o custo final $S(i, j-d, d)$ é verificado se é menor que o melhor custo atual *best_cost_D_R* que será usado para definir a disparidade $D_R(i,j)$ (entre as linhas 33 e 36 do pseudocódigo 1). Se sim, *best_cost_D_R* recebe este valor de custo de $S(i,j,d)$ e o *best_disp_D_R* recebe a disparidade d da iteração atual. É importante observar que é necessário verificar se $j-d$ é maior que 0, ou seja, está dentro dos limites da imagem.

Ao terminar toda a iteração de disparidade, os resultados armazenados nas variáveis *best_disp_D_L* e *best_disp_D_R* são de fato os resultados das disparidades finais $D_L(i,j)$ e $D_R(i,j)$ para o pixel de posição (i,j). Estes são então comparados (linha 38 do pseudocódigo 1). Se a diferença absoluta entre eles for maior que um limiar σ fixo fornecido como entrada da função, então é atribuída a $D(i,j)$ uma codificação que representa que a posição (i,j) é de oclusão. No pseudocódigo foi atribuído o valor 255, mas pode ser qualquer valor fora dos valores destinados para representar a disparidade. Se a diferença for menor então $D(i,j)$ recebe o resultado de *best_disp_D_L*.

2.3.3 Exemplificando o algoritmo SGM

A fim de melhorar o entendimento sobre o SGM, é dado um exemplo da computação deste método sobre duas imagens hipotéticas que considera-se que sejam estéreo e retificadas como mostrado na Figura 22. Os dados das duas imagens são valores aleatórios e portanto não têm interpretação real do cenário.

Figura 22 – Imagem de teste para a computação do SGM. (a) Imagem da esquerda (b) Imagem da direita



As operações seguem o fluxo mostrado no pseudocódigo 1, no qual primeiro calcula-se para cada disparidade o custo inicial, depois os custos de caminho para todos caminhos considerados, em seguida os custos de caminhos são agregados através do somatório e por fim é verificado a disparidade que tem o menor custo para definir D_L. O fluxo do ponto de vista do acesso aos pixels e comutação de disparidade é realizado da seguinte forma: para cada linha, acessados de cima para baixo são computados todas as colunas da esquerda para a direita e para cada coluna são computados todas as disparidades começando da menor disparidade para a maior. É mostrado no final de cada iteração d o estado atual dos custos de caminho $L_{\mathbf{r}^t}$, do valor agregado S e da disparidade D_L. Por simplicidade é adotado para o cálculo do custo inicial o método de diferença absoluta $C(p_i, p_j, d) = |I_L(p_i, p_j) - I_R(p_i, p_j - d)|$. Considere o intervalo de disparidade $N_d = 3$, os caminhos $\mathbf{r}^{0^\circ}$ e $\mathbf{r}^{45^\circ}$ para o cálculo do custo de caminho $L_{\mathbf{r}^t}$ e as penalidades $P_1 = 24$ e $P_2 = 96$.

Para computar o custo agregado do pixel $p(0,0)$ na disparidade $d = 0$, ou seja, $S(0,0,0)$ tem-se:

$$S(0,0,0) = L_{\mathbf{r}^{0^\circ}}(0,0,0) + L_{\mathbf{r}^{45^\circ}}(0,0,0) \tag{2.25}$$

no qual $L_{\mathbf{r}^{0^\circ}}(0,0,0)$ e $L_{\mathbf{r}^{45^\circ}}(0,0,0)$ são:

$$\begin{aligned} L_{\mathbf{r}^{0^\circ}}(0,0,0) = |I_L(0,0) - I_R(0,0-0)| + \min[&L_{\mathbf{r}^{0^\circ}}(0,0-1,0), \\ &L_{\mathbf{r}^{0^\circ}}(0,0-1,0-1) + 24, L_{\mathbf{r}^{0^\circ}}(0,0-1,1) + 24, \\ \min_{i\in[0,3-1]} &L_{\mathbf{r}^{0^\circ}}(0,0-1,i) + 96] - \min_{k\in[0,3-1]} L_{\mathbf{r}^{0^\circ}}(0,0-1,k) \end{aligned} \tag{2.26}$$

$$\begin{aligned} L_{\mathbf{r}^{45^\circ}}(0,0,0) = |I_L(0,0) - I_R(0,0-0)| + \min[L_{\mathbf{r}^{45^\circ}}(0-1,0-1,0), \\ L_{\mathbf{r}^{45^\circ}}(0-1,0-1,0-1) + 24, L_{\mathbf{r}^{45^\circ}}(0-1,0-1,1) + 24, \\ \min_{i\in[0,3-1]} L_{\mathbf{r}^{45^\circ}}(0-1,0-1,i) + 96] - \min_{k\in[0,3-1]} L_{\mathbf{r}^{45^\circ}}(0-1,0-1,k) \end{aligned} \tag{2.27}$$

Como pode ser observado os custos de caminho anterior $L_{\mathbf{r}^{0^\circ}}(0,0-1,0)$ e $L_{\mathbf{r}^{45^\circ}}(0-1,0-1,0)$ estão fora do espaço de imagem. Isto acontece nas bordas da imagem. Estes casos são considerados inválidos e não podem participar do cálculo do custo de caminho atual como já mencionado anteriormente. Nos casos de borda o custo de caminho no qual o custo de caminho anterior está fora da imagem pode resumir-se ao valor de $C(\mathbf{p}, d)$. Outra situação inválida é quando o valor de disparidade é negativo ou é maior que $N_d - 1$, ou a intensidade de um pixel cuja localização está fora da imagem. Estas variáveis precisam ser detectadas e removidas durante o cálculo do custo de caminho atual.

Assim, aplicando os valores de custo de caminho e intensidade do pixel para as Equações 2.26 e 2.27, e verificando as condições de invalidez, obtém-se:

$$L_{\mathbf{r}^{0^\circ}}(0,0,0) = |50 - 90| = \mathbf{40} \tag{2.28}$$

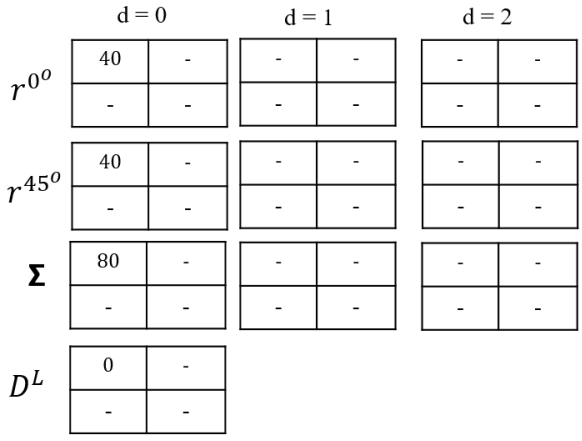
$$L_{\mathbf{r}^{45^\circ}}(0,0,0) = |50 - 90| = \mathbf{40} \tag{2.29}$$

Assim aplicando o resultado das Equações 2.28 e 2.29 na Equação 2.25 obtém-se:

$$S(0,0,0) = 40 + 40 = 80 \tag{2.30}$$

Uma vez que S(0,0,0) é único valor calculado até agora, então a disparidade atual é 0.

Figura 23 – Estado atual do custo de caminho, do resultado agregado e disparidade computada para o pixel $p(0,0)$ e disparidade 0



O estado atual dos custos de caminhos são mostrados na Figura 23. É importante observar que os dados de custos de caminho precisam ser armazenados para serem acessados posteriormente pelos cálculos de custos de caminho a serem realizados pelos pixels adiante.

Repetindo as mesmas etapas para computar $S(0,0,1)$ tem-se:

$$S(0,0,1) = L_{\mathbf{r}^{0^\circ}}(0,0,1) + L_{\mathbf{r}^{45^\circ}}(0,0,1) \tag{2.31}$$

no qual $L_{\mathbf{r}^{0^\circ}}(0,0,1)$ e $L_{\mathbf{r}^{45^\circ}}(0,0,1)$ são respectivamente:

$$\begin{aligned} L_{\mathbf{r}^{0^\circ}}(0,0,1) = |I_L(0,0) - I_R(0,0-1)| + \min[L_{\mathbf{r}^{0^\circ}}(0,0-1,1), \\ L_{\mathbf{r}^{0^\circ}}(0,0-1,1-1) + 24, L_{\mathbf{r}^{0^\circ}}(0,0-1,1+1) + 24, \\ \min_{i\in[0,3-1]} L_{\mathbf{r}^{0^\circ}}(0,0-1,i) + 96] - \min_{k\in[0,3-1]} L_{\mathbf{r}^{0^\circ}}(0,0-1,k) \end{aligned} \tag{2.32}$$

$$\begin{aligned} L_{\mathbf{r}^{45^\circ}}(0,0,1) = |I_L(0,0) - I_R(0,0-1)| + \min[L_{\mathbf{r}^{45^\circ}}(0-1,0-1,1), \\ L_{\mathbf{r}^{45^\circ}}(0-1,0-1,1-1) + 24, L_{\mathbf{r}^{45^\circ}}(0-1,0-1,1+1) + 24, \\ \min_{i\in[0,3-1]} L_{\mathbf{r}^{45^\circ}}(0-1,0-1,i) + 96] - \min_{k\in[0,3-1]} L_{\mathbf{r}^{45^\circ}}(0-1,0-1,k) \end{aligned} \tag{2.33}$$

A variável $I_R(0,0-1)$ é inválida. Então neste caso como não se tem um custo inicial válido, define-se como resultado dos custos de caminho valores infinitamente grandes. Ao se definir valores de custo grande para custos de caminhos inválidos, faz com que estes custos não interfiram no cálculo de disparidade final o que torna as operações mais simples. Isto pode ser verificado diretamente na Equação 2.16 onde valores grandes de custo têm chance pequena de serem escolhidos pelas funções de mínimo, e assim nunca participarão das operações. No exemplo, para estas variáveis inválidas é adotado o valor 255, valor máximo permitido na representação de 8 bits da variável.

Assim, aplicando as condições nas Equações 2.32 e 2.33 obtém-se:

$$L_{\mathbf{r}^{0^\circ}}(0,0,1) = 255 \tag{2.34}$$

$$L_{\mathbf{r}^{45^\circ}}(0,0,1) = 255 \tag{2.35}$$

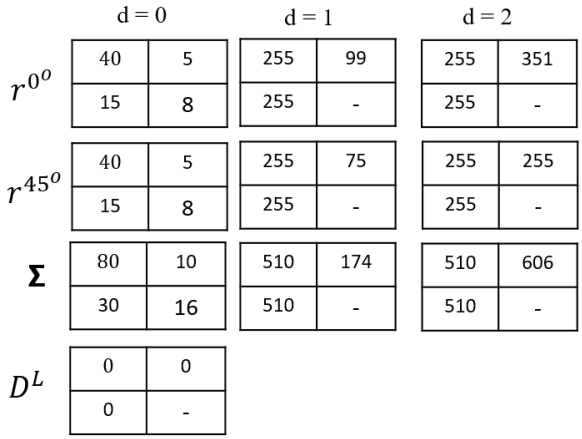
Usando o resultado da Equações 2.34 e 2.35 na Equação 2.31 obtém-se:

$$S(0,0,1) = 255 + 255 = 510 \tag{2.36}$$

O valor de $S(0,0,1)$ é maior do que o anterior $S(0,0,0)$. Assim o valor de disparidade permanece inalterado.

O estado atual dos resultados de custo de caminho e agregação depois de se calcular $S(0,0,0)$, $S(0,0,1)$, $S(0,0,2)$, $S(0,1,0)$, $S(0,1,1)$, $S(0,1,2)$, $S(1,0,0)$, $S(1,0,1)$, $S(1,0,2)$, $S(1,1,0)$ é mostrado na Figura 24.

Figura 24 – Estado atual do custo de caminho, resultado agregado e disparidade depois de computado os custos de caminho do pixel $p(1,1)$ para disparidade 0



Vamos considerar um caso de computação de custo de caminho que não entra em nenhuma condição de borda. Então para computar $S(1,1,1)$ tem-se:

$$S(1,1,1) = L_{\mathbf{r}^{0^\circ}}(1,1,1) + L_{\mathbf{r}^{45^\circ}}(1,1,1) \tag{2.37}$$

no qual $L_{\mathbf{r}^{0^\circ}}(1,1,1)$ e $L_{\mathbf{r}^{45^\circ}}(1,1,1)$ são:

$$\begin{aligned} L_{\mathbf{r}^{0^\circ}}(1,1,1) = |I_L(1,1) - I_R(1,1-1)| + \min[&L_{\mathbf{r}^{0^\circ}}(1,1-1,1), \\ &L_{\mathbf{r}^{0^\circ}}(1,1-1,1-1) + 24, L_{\mathbf{r}^{0^\circ}}(1,1-1,1+1) + 24, \\ &\min_{i\in[0,3-1]} L_{\mathbf{r}^{0^\circ}}(1,1-1,i) + 96] - \min_{k\in[0,3-1]} L_{\mathbf{r}^{0^\circ}}(1,1-1,k) \end{aligned} \tag{2.38}$$

$$\begin{aligned} L_{\mathbf{r}^{45^\circ}}(1,1,1) = |I_L(1,1) - I_R(1,1-1)| + \min[&L_{\mathbf{r}^{45^\circ}}(1-1,1-1,1), \\ &L_{\mathbf{r}^{45^\circ}}(1-1,1-1,1-1) + 24, L_{\mathbf{r}^{45^\circ}}(1-1,1-1,1+1) + 24, \\ &\min_{i\in[0,3-1]} L_{\mathbf{r}^{45^\circ}}(1-1,1-1,i) + 96] - \min_{k\in[0,3-1]} L_{\mathbf{r}^{45^\circ}}(1-1,1-1,k) \end{aligned} \tag{2.39}$$

Assim, aplicando os valores de custo de caminho e intensidade do pixel para as Equações 2.38 e 2.39 obtém-se:

$$L_{\mathbf{r}^{0^\circ}}(1,1,1) = |12-25| + \min[255, 15+24, 255+24, 15+96] - 15 = 37 \tag{2.40}$$

$$L_{\mathbf{r}^{45^\circ}}(1,1,1) = |12-25| = \min[255, 40+24, 255+24, 40+96] - 40 = 37 \tag{2.41}$$

Assim, aplicando o resultado das Equações 2.40 e 2.41 na Equação 2.37 obtém-se:

$$S(1,1,1) = 37 + 37 = 74 \tag{2.42}$$

O valor de $S(1,1,1)$ é maior do que o anterior $S(1,1,0)$. Assim, o valor de disparidade permanece inalterado.

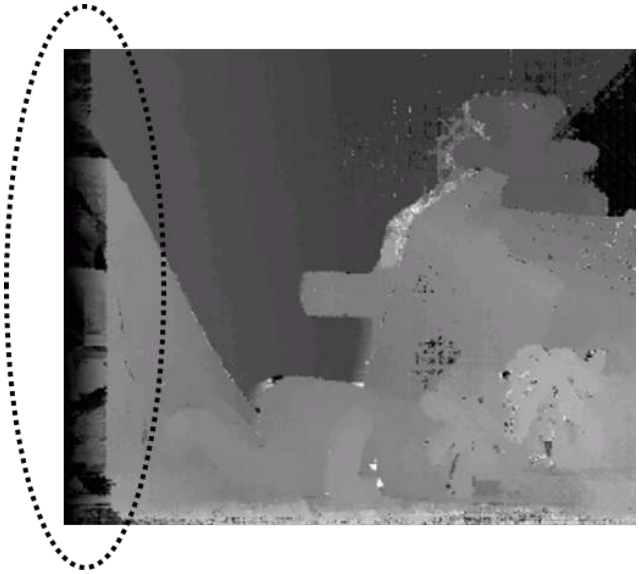
Figura 25 – O resultado final de custo de caminho, agregação e disparidade D_L

	d = 0		d = 1		d = 2	
r^{0^o}	40	5	255	99	255	351
	15	8	255	37	255	351
r^{45^o}	40	5	255	75	255	255
	15	8	255	37	255	351
Σ	80	10	510	174	510	606
	30	16	510	74	510	702
D^L	0	0				
	0	0				

O estado atual depois de computado todos os custos de caminho e agregação é mostrado na Figura 25.

Por coincidência os menores custos agregados para todos os pixels estão localizados na disparidade 0. Esta situação acontece porque próximo da borda esquerda da imagem as maiores disparidades ultrapassam os limites da imagem (como já é esperado). Intuitivamente, em outras palavras, isto acontece porque estas regiões próximas da borda esquerda são vistas apenas por uma das câmeras e portanto a disparidade não pode ser analisadas para todos os valores de disparidades, somente aqueles menores que ainda não ultrapassaram os limites da imagem.

Figura 26 – Exemplo de mapa de disparidade destacando as regiões escurecidas próximo a borda esquerda.



Uma vez que são definidos valores bastante grandes para as disparidades que ultrapassaram os limites da imagem, estas não serão escolhidas durante o cálculo da disparidade que realiza uma função de mínimo dos custos de todas as disparidades. Um exemplo do

mapa de disparidade, destacando esta região próximo a borda esquerda é mostrada na Figura 26.

2.3.4 Complexidade, dependência de dados e armazenamento do SGM

Como pode ser observado no pseudocódigo 1, para decidir sobre o valor de disparidade final de um dado pixel $\mathbf{p}$ é necessário calcular para cada disparidade d os custos de caminho $L_{r^t}(\mathbf{p}, d)$ para todas direções $\mathbf{r}^t$ consideradas e em seguida calcular o valor agregado destes custos de caminho para compor $S(\mathbf{p}, d)$. Dado que a imagem tem largura W e comprimento H, o algoritmo precisa calcular N_d valores de disparidades e N_t custos de caminhos. Então a complexidade para operar toda a imagem é O(W x H x N_t x N_d). Os trabalhos atuais geralmente têm buscado intervalos de disparidade maior ou igual a 128 valores ($N_d > 128$) com uma quantidade mínima de 4 custos de caminhos (geralmente são para as direções de $\mathbf{r}^{0^\circ}$, $\mathbf{r}^{45^\circ}$, $\mathbf{r}^{90^\circ}$, $\mathbf{r}^{135^\circ}$) e em resolução VGA (W = 640 e H = 480) ou superiores. Dependendo da plataforma e da abordagem de implementação adotada, esses requisitos podem comprometer o desempenho do algoritmo SGM. Por exemplo, implementação do SGM em CPU para imagens VGA com 128 níveis de disparidades alcança taxa de frame de 16 fps o que é um desempenho ruim para aplicações de tempo real Gehrig e Rabe (2010).

Além da alta complexidade, o SGM possui dependências que dificultam a sua implementação de maneira paralela. Embora a função de custo de caminho $L_{r^t}(\mathbf{p}, d)$ não possua a dependência entre valores de disparidades diferentes, esta função tem dependência de custos de caminhos entre pixels vizinhos. Como pode ser visto na equação 2.16, o cálculo do custo de caminho $L_{r^t}(\mathbf{p}, d)$ para um dado pixel necessita da informação de custo de caminho a partir do pixel vizinho $\mathbf{p} + \mathbf{r}^t$, ou seja, o custo de caminho atual para cada disparidade d precisa dos custos de caminho $L_{r^t}(\mathbf{p} + \mathbf{r}^t, d)$, $L_{r^t}(\mathbf{p} + \mathbf{r}^t, d + 1)$, $L_{r^t}(\mathbf{p} + \mathbf{r}^t, d - 1)$) bem como do mínimo dos custos de caminhos para todas as disparidades ($\min_{k \in [0, N_d - 1]} L_{\mathbf{r}^t}(\mathbf{p} + \mathbf{r}^t, k)$). Para definir este último termo é necessário que o custo de caminho do pixel vizinho tenha sido definido para todo o intervalo N_d de disparidades. Como será visto nos capítulos 3 e 4, esta dependência é o principal problema que dificulta desenvolver implementações do SGM em hardware, explorando-se paralelismo.

Outra questão importante levantada é sobre a grande quantidade de armazenamento necessária para manter os dados de custo de caminho de pixels anteriores e a grande quantidade de acessos necessários para calcular o custo acumulado para cada pixel. A escolha do recurso de armazenamento pode impactar diretamente no desempenho e no consumo de potência de todo o sistema. Adicionalmente os autores Pantilie e Nedevschi (2012) evidenciam a dificuldade de implementar sistemas completos de correspondência estéreo em altas resoluções e com grande intervalo de disparidades em plataformas baseadas em FPGAs, uma vez que a quantidade de recursos computacionais nestas plataformas é um fator limitante.

2.4 FPGAs para aplicações de tempo real e embarcadas

Além da alta qualidade, diversas aplicações exigem sistemas de visão estéreo que processem mapas de disparidade em tempo real. Por exemplo em aplicações navegação autônoma, para evitar a colisão de um carro com algum obstáculo, a abordagem de visão estéreo precisa fornecer informações de profundidade de forma precisa e rápida o suficiente para que o restante do sistema seja capaz de detectar o obstáculo e acionar os freios ou mudar a direção em tempo suficiente para evitar a colisão Kowsari, Beauchemin e Cho (2011).

Para as aplicações embarcadas, além da qualidade e do desempenho, questões tais como consumo de potência e tamanho são levadas em consideração ao analisar uma abordagem de visão estéreo. Em linhas gerais, aplicações embarcadas são sistemas que funcionam através de baterias e são geralmente pequenos. A usabilidade de uma aplicação embarcada depende diretamente da duração da bateria e do tamanho de cada componente. Além disso, o baixo custo de fabricação do sistema é um aspecto preferível em tais aplicações. O espaço de armazenamento de dados necessários e a quantidade de lógicas de processamento refletem diretamente no consumo de energia, no custo de fabricação e também no tamanho da abordagem de visão estéreo. Dessa forma, é preferível abordagens que exijam cada vez menos espaços de armazenamento e lógicas de processamento.

Alcançar resultados ótimos em termos de velocidade, precisão, consumo de potência e tamanho depende da natureza do algoritmo, da estratégia de processamento, e também da plataforma de hardware para processamento. Como observado nas seções 2.3 e 2.2, os algoritmos de correspondência estéreo de alta qualidade, tal como o semi global (SGM), demandam uma grande quantidade de operações massivas para computar as disparidades. Assim, processar tais algoritmos em uma configuração aceitável de resolução e intervalo de disparidade (por exemplo, resolução 640x480 e 128 valores) certamente não alcançará desempenho aceitável (por exemplo, próximo dos 30 fps) utilizando processamento sequencial. Dessa forma, uma maneira bastante promissora para alcançar desempenho de tempo real seria explorar o paralelismo do algoritmo. Existem quatro tipos de abordagens que permitem lidar com paralelismo: as abordagens baseadas em processadores de propósito geral (CPUs), as abordagens baseados em unidades de processamento gráfico (GPUs) e abordagens baseados em hardware tais com os ASICs e os FPGAs.

As CPUs, embora seja uma arquitetura de processamento inerentemente sequencial, com os avanços das tecnologias de arquitetura de processadores, esta passou a oferecer algumas estratégias de obtenção de paralelismo tais como instruções vetoriais que operam em um espaço de dados de 128 bits, os chamados instruções SIMD (Single instruction, multiple data), e os vários núcleos (cores) de processamento paralelo (tipicamente 8 núcleos). Geralmente os custos no cálculo de correspondência estéreo são representados por um vetor de 16 bits. Desta forma, o máximo que pode ser extraído usando CPU sem mudança na técnica original do SGM é o processamento de no máximo 8 custos em pa-

ralelo usando instruções SIMD e o também o processamento de cada direção do SGM de maneira paralela distribuída cuidadosamente em cada threads (ou núcleos). Contudo, esta quantidade de paralelismo ainda é insuficiente para a quantidade de operações exigidas pela abordagem semi global. As CPUs são processadores de propósito geral que oferecem um tipo de programação em forma execução de instruções. As escolhas das instruções determina a funcionalidade desejada. Através de linguagens de código amigáveis, o desenvolvimento e correção de uma determinada funcionalidade neste tipo de plataforma é rápido.

Ao contrário de processadores de propósito geral (CPUs) que oferecem uma quantidade limitada de paralelismo os processadores GPUs oferecem uma capacidade maior de extração de paralelismo dos algoritmos devido ao uma grande quantidade de núcleos (mais de 1000 núcleos) de processamento disponíveis para computação independente dos dados. Assim, é possível processar uma quantidade bem maior de disparidades para correspondência estéreo em paralelo e desta forma, aumentar drasticamente o ganho de desempenho em relação a implementações em CPU. Contudo, as GPUs são processadores que trabalham em altas frequências e como consequência consomem altas potências (por exemplo, GPU consome mais de 275 Watts de potência) não sendo adequadas para aplicações embarcadas (BAILEY, 2011). Além disso, devido a dependência de dados da abordagem de correspondência semi global proposta neste trabalho as implementações em GPUs não conseguem aproveitar bem o paralelismo existente devido a grande necessidade de troca de informação entre os núcleos de processamento que impõem atrasos de comunicação.

Os Circuitos Integrados de Aplicação Específica (ASICs) são abordagens baseados em hardware onde a o desenvolvimento de uma funcionalidade é traduzido em um conjunto de lógicas primitivas que formarão um circuito integrado (CI). Os ASICs permitem o desenvolvimento de sistemas totalmente customizados de acordo com a aplicação possibilitando desenvolver produtos de alta tecnologia com baixo custo e consumo reduzido de energia. Para isso, é necessário aperfeiçoar e otimizar a lógica de acordo com a sua função. O consumo de energia também é reduzido devido a esta otimização pois o processador pode operar em uma menor frequência, já que possui alto desempenho. Contudo, dois problemas residem no desenvolvimento de abordagens com ASICs. O primeiro problema é a inflexibilidade para mudança de funcionalidade. Uma vez que o circuito foi configurado, o circuito integrado se torna difícil ou até impossível de ser reconfigurado caso a suaa funcionalidade precise ser modificada. O segundo problema é o alto custo de fabricação e desenvolvimento para produzir um circuito integrado com a funcionalidade desejada além do grande tempo para desenvolver tal circuito. O custo e o tempo de desenvolvimento alto inviabiliza projetos que precisam de um retorno rápido do produto para o mercado.

Por outro lado, os Field Programmable Gate Array (FPGA) combinam as características dos ASICs com a flexibilidade do desenvolvimento de software das CPUs no qual sua funcionalidade pode ser reprogramada ou reconfigurada a qualquer momento. Os FPGAs

são dispositivos em forma de chip programável que permite desenvolver um circuito de hardware totalmente customizado para a aplicação desejada tal como os ASICs. A grande vantagem no desenvolvimento de projeto usando FPGA é a obtenção de um protótipo em fases iniciais do projeto. Isto é porque estes dispositivos, juntamente com as ferramentas de síntese e linguagem de alto nível amigável, oferecem uma estratégia de verificação de implementação que combinam ao mesmo tempo a facilidade de simulação e a obtenção do realismo do protótipo Courtoy (1998). Uma vez que o custo de desenvolvimento dos circuitos integrados específicos de aplicação (ASICs) ainda permanece alto, a utilização dos FPGAs têm sido largamente adotado tanto na comunidade acadêmica quanto na indústria. Com a crescente evolução tecnológica destes dispositivos, com uma grande quantia de unidades lógicas em um único circuito integrado, torna-se possível criar dispositivos altamente complexos em um único chip de FPGAs. O uso da computação configurável através de FPGAs possibilita constante melhoria da técnica de processamento de uma determinada função, uma vez que a programação do chip FPGA é descrito através de uma linguagem de descrição de hardware (HDL), sem que sejam necessárias alterações físicas do próprio hardware, resultando em sistemas eficientes, compactos, de baixo consumo de energia e de custo reduzido.

A seguir, será explicado com um pouco mais detalhes a estrutura interna de uma FPGA e como esta plataforma processa uma determinada funcionalidade. Os conceitos servirão como base para a explicação da abordagem proposta neste trabalho.

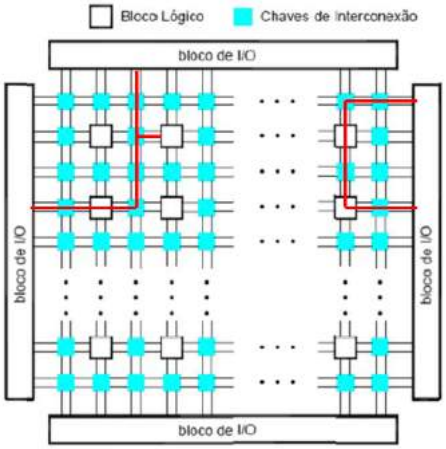
2.4.1 Estrutura Interna de um FPGA

A estrutura geral de uma FPGA é apresentada na Figura 27, apresentando três tipos de recursos: blocos I/O para conectar os pinos de entrada e saída do dispositivo, blocos lógicos (ou *slices*, em inglês) dispostos num arranjo de matriz e um conjunto de chaves de interconexão organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas dos blocos lógicos.

O Array of Logical Blocks e o Configurable Logical Block (CLB) são os nomes dados aos blocos lógicos que compõem uma FPGA do fabricante Altera e o fabricante Xilinx, respectivamente. Os blocos lógicos, de uma forma geral, são constituídos por uma unidade lógica combinacional programável e um conjunto de somadores, registradores, flip-flops e latches.

Cada fabricante determina a organização interna do seu bloco lógico e seu comportamento, mas na maioria das vezes a estrutura interna dos blocos lógicos pode ser representada através da Figura 28. Existem diferentes formas de implementar a unidade lógica combinacional. O método mais comum consiste em usar look-up tables (LUTs), que são células de memórias conectadas a um multiplexador que seleciona uma de tais células como a saída da unidade lógica combinacional.

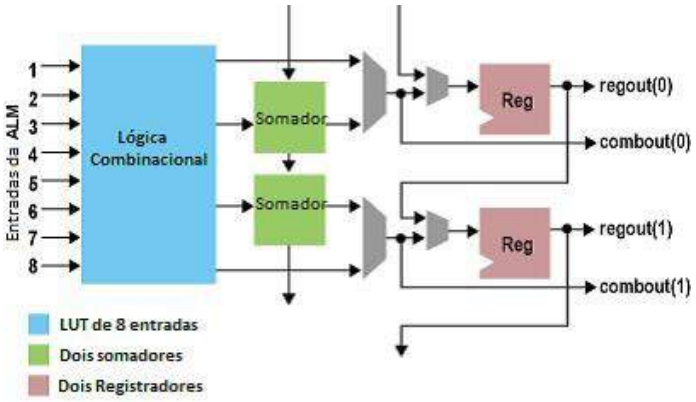
Figura 27 – Estrutura interna de uma FPGA



(BROWN; VRANESIC, 2000)

Na LUT ficam armazenadas possíveis respostas de partes de um sistema a um conjunto de estímulos. O sinal com tais estímulos é utilizado como o seletor do multiplexador que fica localizado na saída da unidade lógica combinacional. Assim, de acordo com o estímulo, é selecionada a resposta correta a ser encaminhada para a saída da unidade lógica combinacional.

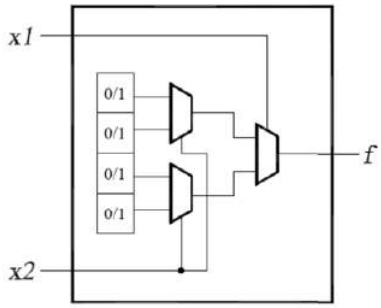
Figura 28 – Organização interna de um bloco lógico de FPGA



(DUTRA, 2010)

A Figura 29 apresenta a estrutura de uma pequena LUT com duas entradas, x_1 e x_2, e uma saída f, a qual é capaz de implementar qualquer função de duas variáveis. Devida à tabela verdade de duas variáveis possuir quatro combinações possíveis, esta LUT possui quatro células de armazenamento. As variáveis de entrada x_1 e x_2 são usadas com seletores de entrada dos três multiplexadores, que dependendo dos valores destas variáveis selecionam o conteúdo de umas das quatro células como saída da LUT. As FPGAs disponíveis no mercado geralmente possuem LUTs de quatro a oito entradas.

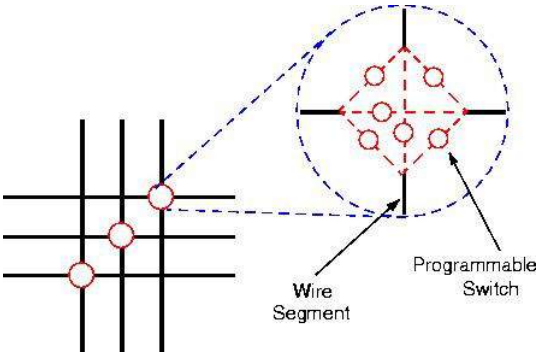
Figura 29 – Estrutura de uma LUT de duas entradas



(BROWN; VRANESIC, 2000)

Quando um circuito lógico é implementado em um FPGA, os blocos lógicos são programados para realizarem as funções necessárias e as chaves de interconexão são programadas para definir o canal de comunicação entre estes blocos lógicos. A interconexão entre os blocos lógicos permite que sejam construídos circuitos complexos e circuitos paralelos em FPGA e tal interconexão é provida pelos elementos de roteamento. Estes elementos são distribuídos por todo FPGA e são programados, na maioria das vezes, por ferramentas de software, que são capazes de encontrar os melhores caminhos e conexões a serem ativados para que os recursos disponíveis no FPGA sejam utilizados da forma mais otimizada possível.

Figura 30 – Estrutura de uma matriz de chaveadores (*Switch Matrix*)



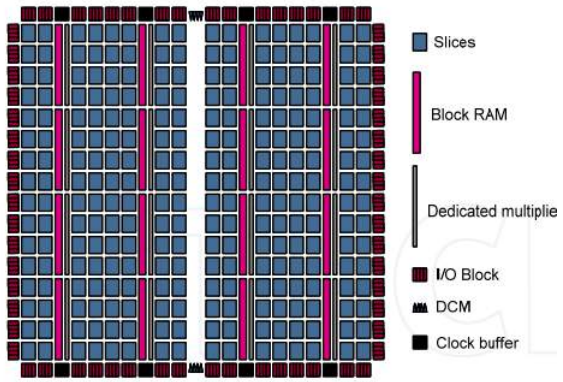
(ROCHA, 2010)

A programação dos elementos de roteamento é feita através da configuração de uma estrutura conhecida como matriz de chaveadores (ou do inglês *switch matrix*, que é apresentada na Figura 30. Com o uso de tal estrutura, para ativar uma conexão entre dois elementos lógicos basta ativar um ou mais caminhos que levam sinais de um elemento para o outro.

Os FPGAs possuem blocos de memória embarcados (BRAM) que ficam conectadas aos blocos lógicos e que podem ser configurados para funcionar como SRAM ou ROM como mostrado na Figura 31. Tais unidades de memória servem para prover o armazenamento de

dados no FPGA. Isso é importante, pois, como é descrito no capítulo 4, o armazenamento interno dos resultados intermediários do processamento realizado no FPGA elimina a necessidade de que tais resultados sejam escritos e lidos constantemente da memória externa que é conectada ao FPGA. Assim, todo o dado necessário para o processamento é lido apenas uma vez e o resultado e escrito apenas uma vez na memória externa conectada ao FPGA, o que provoca uma redução da necessidade de largura de banda de comunicação entre FPGA e memória.

Figura 31 – Ilustração da memória (BRAM) na estrutura de um FPGA genérico



(MAGDALENO; RODRÍGUEZ, 2012)

Os blocos de entrada e saída implementam a comunicação do FPGA com elementos externos. Através dos blocos de entrada e saída é possível enviar sinais para serem processados dentro do FPGA e receber as respostas geradas por tal processamento. Além das estruturas básicas discutidas anteriormente, os FPGAs atuais possuem ainda em sua estrutura um conjunto maior de tipos de elementos para aplicações especiais, como, por exemplo: unidades DSP, para processamento rápido de sinais digitais; unidades de memória, para armazenamento interno de dados DUTRA (2010); canais de comunicação de alta velocidade para a troca de dados entre FPGAs; e núcleos de CPU, que podem ser utilizados para executar aplicações no dispositivo reconfigurável. Todos estes são fatores que têm contribuído para que este tipo de tecnologia se torne cada vez mais uma opção atraente para o desenvolvimento de projetos que seguem a linha da computação reconfigurável.

2.5 Conclusão

Embora a abordagem SGM ofereça resultados bastante precisos esta abordagem é bastante complexa, possui uma grande dependência de dados e necessita de grandes espaços de armazenamentos. Desta forma para que esta abordagem possa ser utilizada em aplicações de tempo real e sobretudo implementadas em plataformas de baixo recurso como nos FPGAs a implementação deste algoritmo precisa resolver todos os desafios que a implementação do SGM impõe de maneira eficiente.

3 TRABALHOS RELACIONADOS

Muitas aplicações do mundo real que utiliza visão estéreo, tais como navegação autônoma, reconstrução 3D, segmentação de vídeo e rastreamento exigem, sistemas de correspondência estéreo que ofereçam mapas de disparidades em tempo real de alta qualidade processando imagens em altas resoluções e grandes intervalos de disparidades. Uma meta bastante utilizada em diversos trabalhos é a taxa de processamento de mapas de disparidade de no mínimo 30 frames por segundo (fps) em uma resolução mínima de 640x480 (VGA) com um espaço de disparidade de no mínimo 128 valores (BANZ et al., 2010). Isto ocorre devido às limitações atuais de tecnologia de câmeras e sistemas de processamento não processarem frames além desta da taxa. Contudo, esta tendência está mudando e já é possível termos câmeras que processam em resoluções muito mais altas (exemplo: 1920x1200) e a uma taxa muito mais alta que 30 fps. Assim, as aplicações têm exigido sistemas de correspondência estéreo cada vez mais rápidos e mais precisos que computam intervalos de disparidades acima de 128 valores e em resoluções acima de VGA. Por exemplo, os pares de imagem estéreo do banco de imagens Middlebury de 2005 de tamanho completo (SCHARSTEIN; SZELISKI, 2002) têm em média resoluções de 1,4 megapixels (MP) e intervalo mínimo de de disparidades de 200 pixels. Além disso, mapas de disparidades com maiores resoluções e com mais detalhes permitem melhorar a qualidade da reconstrução 3D (KIM, 2015).

A técnica Semi Global (SGM) tem se mostrado capaz de atender a todas estas demandas e tem sido a mais adotada por muitas aplicações embarcadas. Contudo, como elucidado na seção 2.3, algumas questões importantes impõem desafios para implementações da técnica SGM em diversas plataformas, principalmente em hardware, tal como alta complexidade, dependência de dados e grande quantidade de armazenamento de dados finais e intermediários. Desta forma, tentando lidar com todas estas questões, várias arquiteturas para sistemas estéreo baseados na técnica SGM têm sido propostas.

No trabalho de Gehrig (GEHRIG; RABE, 2010), foi implementada a técnica de SGM em um processador de propósito geral. Para reduzir a largura de banda de memória e a complexidade para computar a disparidade, os autores usam subamostragem de profundidade e de imagem. A subamostragem de profundidade garante uma incerteza abaixo de 1 metro de profundidade. Além disso, os autores empregam instruções SIMD (single instruction, multiple data) para computar 16 disparidades de uma só vez durante a etapa de acumulação. Eles usaram também a linguagem de OpenMP (DAGUM; MENON, 1998) para especificar os algoritmos que calculam todos os custos de caminho de direções diferentes de maneira independente e em paralelo. Desta maneira, para que o algoritmo mencionado atenda aos requisitos de tempo real, as imagens são subamostradas reduzindo-se pela metade e para um quarto da resolução original. Para calcular o custo inicial o algoritmo

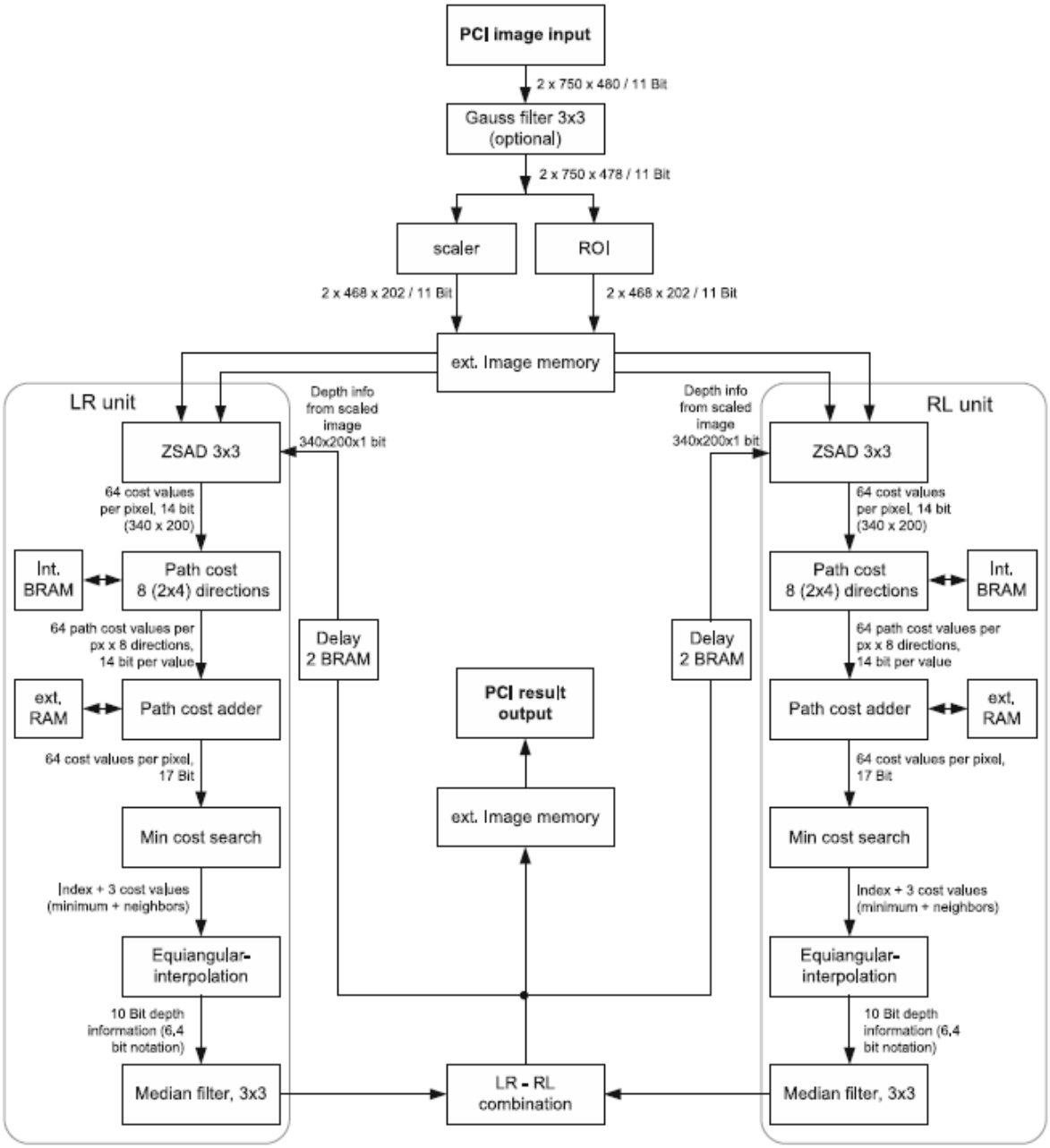
usa a transformada census, que é computada de forma paralelizada linha por linha com OpenMP. Para imagens de 640 x 320 e um espaço de disparidade de 128, a implementação roda em 14 fps em um processador Intel Core i7 de 3.3 Ghz. Buscando contornar a grande complexidade da abordagem SGM, os autores em (SPANGENBERG et al., 2014) propuseram reduzir o espaço de busca computando as disparidades da seguinte forma: entre os valores de disparidade 0 até 63 o algoritmo computa todos os valores de disparidades e entre os valores de disparidade 64 e 128 o algoritmo computa apenas as disparidades pares. Esta abordagem certamente reduz a qualidade do mapa de disparidade, mas os autores argumentam que dependendo da aplicação pode ser suficiente reduzir a qualidade amostrando disparidades maiores com um grid mais esparso. Além disso, os autores usam operações SIMD, computando custos de caminho para 8 disparidades paralelamente. O sistema com um processador Core i7 com quatro núcleos de 2.6 Ghz cada é capaz de computar mapas de disparidades em resolução de 640x480 com um espaço de disparidade 128 em um taxa maior que 16 FPS, o que é ainda abaixo do que se espera para aplicações de tempo real. Estes desempenhos em plataforma baseada em processadores de propósito geral ainda estão abaixo do que é esperado para aplicações de tempo real (acima de 30 fps) sobretudo porque a quantidade de paralelismo que se pode obter com processadores de propósito geral é limitada.

As abordagens baseadas em hardware conseguem atingir desempenhos iguais ou superiores que as abordagens baseadas em processadores de propósito geral com frequência e consumo de potência bem menores e com bem menos recursos de processamento e armazenamento, sobretudo por causa da capacidade que este tipo de plataforma tem de extrair uma quantidade maior de paralelismo. Estas características são particularmente interessantes para aplicações embarcadas que exigem baixo consumo de energia, tal como robô autônomo que é alimentado por baterias (MURRAY; LITTLE, 2000). A seguir são detalhados algumas abordagens para implementação da técnica SGM baseadas em hardware reconfigurável (FPGA), tecnologia usada neste trabalho.

No trabalho proposto em (GEHRIG; EBERLI; MEYER, 2009), os autores desenvolveram uma arquitetura para implementar o algoritmo SGM baseada em uma plataforma FPGA com desempenho de tempo real. Os autores calculam 8 custos de caminho em dois passos, onde em cada passo são calculados 4 caminhos simultaneamente. Os resultados intermediários são armazenados em uma memória externa durante a computação dos 4 primeiros custos de caminhos e em seguida estes resultados são lidos de volta para posterior acumulação com o resultado da computação dos 4 últimos custos de caminhos. Para reduzir o custo computacional mantendo uma qualidade aceitável do mapa de disparidade, os autores computam o método SGM em uma imagem subamostrada. Os autores trabalharam com intervalos de disparidades de 64 valores na imagem subamostrada. Uma vez que a resolução de imagem foi reduzida pela metade, as 64 disparidades calculadas correspondem aproximadamente a 128 disparidades na imagem original. Embora essa subamostragem

permita redução na quantidade de recursos, as aplicações de hoje têm exigido intervalos de disparidades e resolução de imagem cada vez maiores devido a melhoria contínua na qualidade das câmeras. Além disso, à medida em que a distância entre os eixos focais aumentam, o sistema exige processamento de um espaço de disparidade maior para poder encontrar os pixels correspondentes corretos.

Figura 32 – Arquitetura para computação do SGM proposta por Gehrig

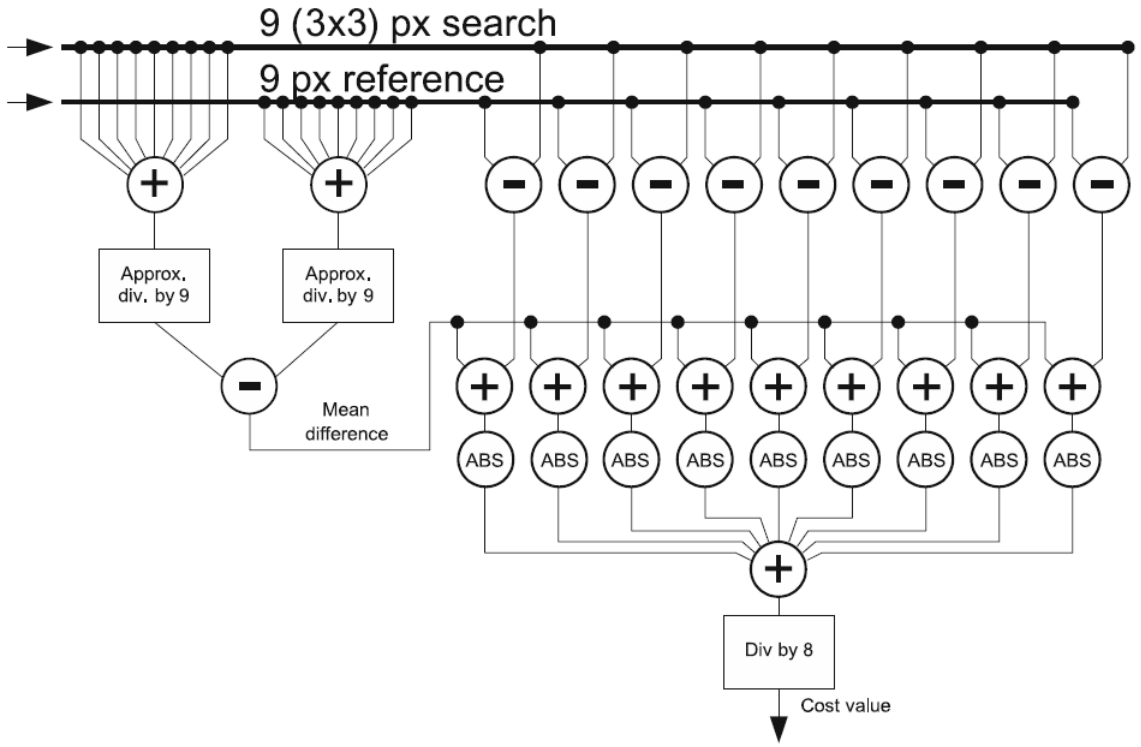


(GEHRIG; EBERLI; MEYER, 2009)

Os autores empregam neste trabalho, para o cálculo do custo inicial, a abordagem local baseada em área chamada de *soma de diferenças absolutas centrada na média* (ZSAD). As abordagens baseadas em área para o cálculo do custo inicial do método SGM podem

definir custos iniciais errados em regiões da cena que apresentam estruturas finas e com grandes descontinuidades de profundidades como detalhado na seção 2.2. Além disso, o emprego de abordagens baseadas em área aumenta a quantidade de recurso necessário em hardware, devido às diversas operações de agregação como soma, divisão e unidades de armazenamentos temporários como mostrado na Figura 33.

Figura 33 – Arquitetura para a abordagem local ZSAD da etapa de custo inicial do SGM proposta por Gehrig



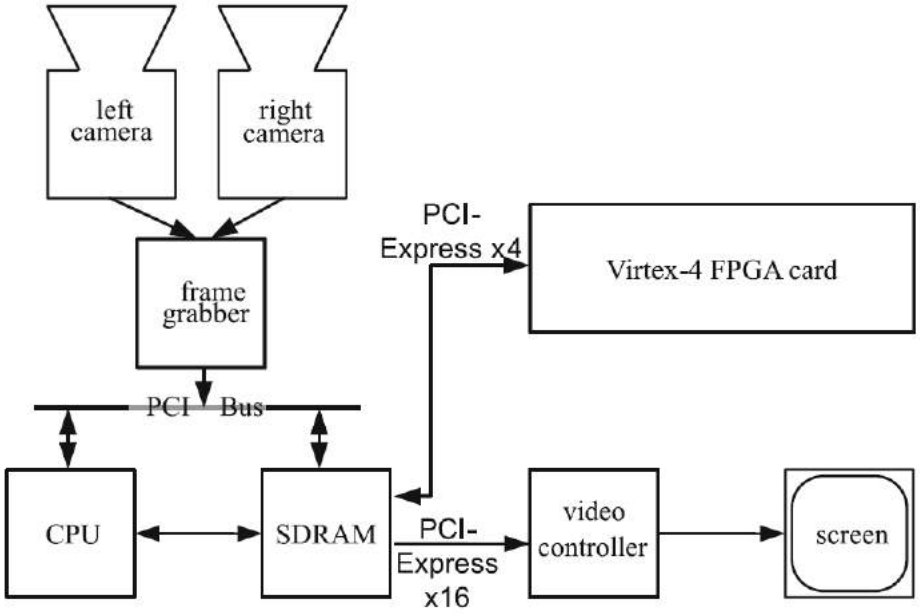
(GEHRIG; EBERLI; MEYER, 2009)

O sistema de visão estéreo completo incluindo a técnica SGM é avaliado numa plataforma experimental composto de uma FPGA Virtex-4 de alto desempenho e um processador de propósito geral. Nesta plataforma, os autores implementam no processador a etapa de retificação e na FPGA implementam todas as etapas da técnica SGM mostradas na Figura 32. Na etapa de detecção de oclusões os autores replicaram a unidade SGM, onde uma unidade, chamado de *LR unit* gera o mapa de disparidade no qual a imagem de referência é a imagem da esquerda e a imagem de correspondência é a imagem da direita e a outra unidade, chamado de *RL unit*, gera o mapa de disparidade no qual a imagem de referência é a imagem da direita e a imagem de correspondência é a imagem da esquerda. O resultado dos dois mapas são utilizados na etapa de *LR-RL combination* para definir o mapa de disparidade final rotulando as regiões de oclusão. Esta abordagem baseada em replicação da unidade SGM para detecção de oclusão duplica a quantidade de recursos necessários para a implementação da técnica SGM podendo tornar o sistema não

implementável em plataformas baseadas em FPGA de baixo custo. Os autores empregam filtros gaussianos para a etapa de pré-processamento para reduzir os ruídos das câmeras. Filtros gaussianos podem tornar o mapa de disparidade final bastante borrado à medida que o tamanho da janela aumenta.

Na plataforma experimental os autores utilizam três memórias externas DDR2, uma para armazenar a imagem de entrada e o resultado do mapa de disparidade e as outras duas para armazenar os custos de caminho intermediários de cada unidade de processamento estéreo. A adoção de memórias externas ao sistema além de aumentar o consumo de energia pode reduzir o desempenho do sistema.

Figura 34 – Plataforma de prototipação proposta por Gehrig



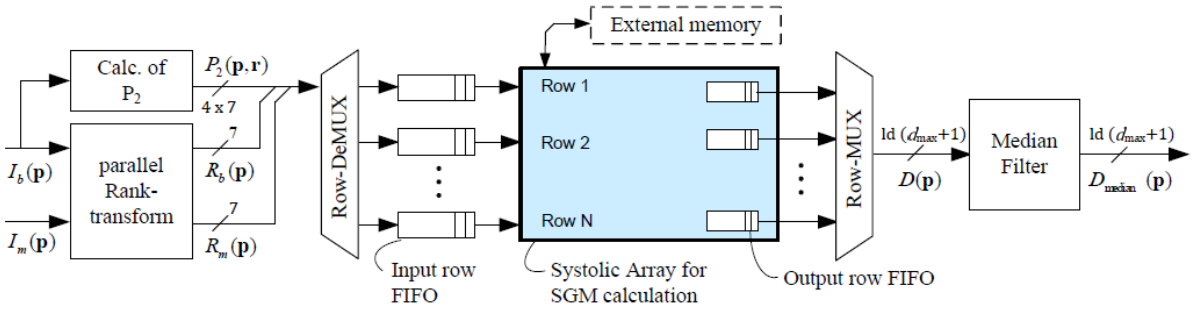
(GEHRIG; EBERLI; MEYER, 2009)

Na plataforma experimental os autores reportaram uma taxa de 27 frames por segundo (fps) para imagens de 340 x 200 (reduzido a partir de 680 x 400) com uma velocidade de 200 MHz, um espaço de disparidade de 64 valores e consumindo quase 60000 elementos lógicos. Os autores reportaram também a qualidade da abordagem proposta avaliando-a com o banco de imagem estéreo do benchmark Middlebury e usando a métrica de porcentagem de pixels errados onde os autores consideram como erro quando a diferença da disparidade entre a referência e o obtido pela aboradgem é maior do que 1 unidade. Desta forma, os autores encontram uma taxa de 5.86 % de pixels errados para a imagem estéreo Tsukuba, 3.85 % para Venus, 13.28 % para Teddy e 9.54 % para Cones.

No trabalho proposto por Banz (BANZ et al., 2010), foi desenvolvida uma arquitetura para a implementação da técnica SGM mostrada na Figura 35, na qual para a etapa de cálculo de custo inicial, foi implementado o algoritmo local transformada de rank, na etapa de detecção de oclusão foi implementado a detecção rápida mostrada na seção 2.3

e foi implementado um módulo de atualização dinâmica do parâmetro P_2 em função da variação de intensidade da imagem.

Figura 35 – Arquitetura do SGM proposta por Banz

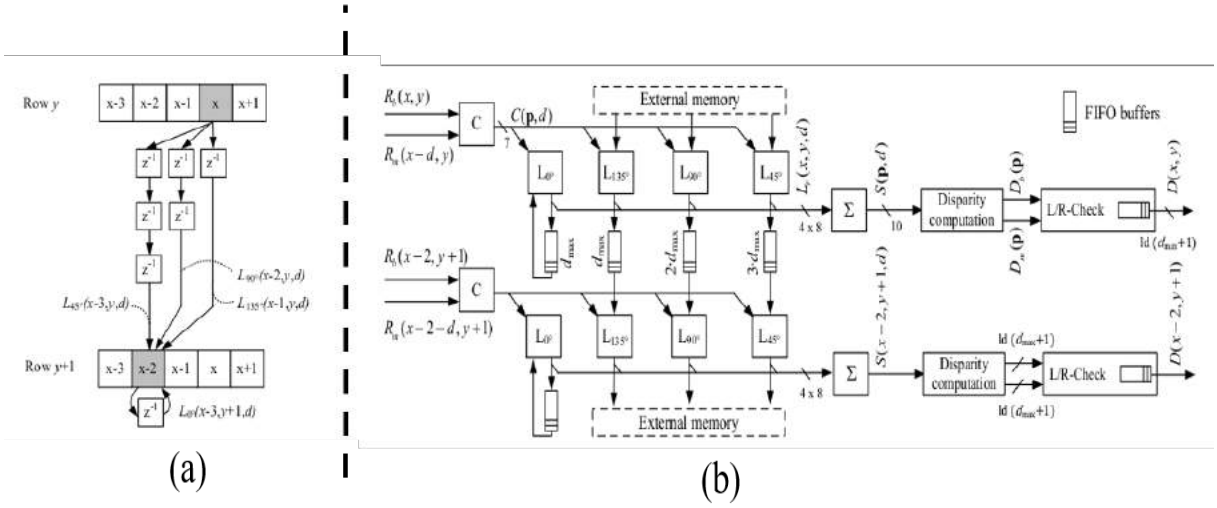


(BANZ et al., 2010)

A implementação da etapa de cálculo de custo e agregação em hardware é baseado em array-sistólico que processa em um único ciclo múltiplos custos de caminhos de várias linhas da imagem para um dado valor de disparidade. Este processamento é realizado através de operações de atraso de custo de caminho entre as linhas. Isto foi feito para garantir o acesso alinhado aos custos de caminho de linhas anteriores durante o cálculo do custo de caminho da linha atual como mostrado nas Figuras 36 (a) e (b). A Figura 36 (a) mostra o conceito de paralelização. Os pixels são processados da esquerda para a direita ao longo da linha da imagem. Depois de processar o pixel $p_{-1} = [x-1, y]$ da linha superior, todos os custos de caminho sobre d de todas as direções estão disponíveis nos buffers de custo de caminho (z^{-1}). Os custos de caminho são atrasados, de acordo com suas direções de caminho de 90^o e 45^o por um e dois processamentos adicionais, respectivamente. Depois disso, os custos de caminho de $L_{45°}(x-3, y, d)$, $L_{90°}(x-2, y, d)$, $L_{135°}(x-1, y, d)$ estão disponíveis na saída dos buffers de custo de caminho. Estes são exatamente os custos de caminho necessitado para o cálculo síncrono e paralelo de todos os custos de caminho de todas as orientações para o $p_2 = [x-2, y+1]$. Cálculo síncrono permite soma direta de custos de caminho em um pipeline que retorna o custo agregado de S. Portanto todos as direções para os pixels $p_1 = [x, y]$ e $p_2 = [x-2, y+1]$ são calculados em paralelo em uma etapa de processamento único. Este conceito é generalizado para um número arbitrário de linhas. Um atraso adicional de dois pixels é introduzido para cada nova linha. Imagens são separadas em fatias da imagem (ou slices no inglês) de N linhas paralelas a fim de processar a imagem inteira. Custos de caminho da última linha de uma fatia precisa ser armazenado e tornado disponível para a primeira linha da próxima fatia. A Figura 36 (b) mostra o diagrama de blocos da abordagem proposta por Banz (BANZ et al., 2010). O processamento de um pixel $\mathbf{p}$ é realizado sequencialmente sobre todas as disparidades deste pixel. O primeiro elemento de processamento (C-PEs) calcula o custo de correspondência $C(\mathbf{p}, d)$. Cada um dos seguintes PEs (L-PEs) calcula

os custos de caminho L_r ao longo de de cada caminho. Os resultados são buferizados em um buffer de custo de caminho apropriado. Todos os L-PEs são completamente idênticos e as orientações são unicamente definidos através de atrasos introduzidos pelos buffers de custo de caminho. Os custos de caminho são somados com S e em seguida processado através de PEs de computação de disparidade (D-PEs). D-PEs localiza o mínimo, ou seja, a disparidade correta, para os mapas de disparidade D_b e D_m que são os D_L e D_R na nomenclatura deste trabalho.

Figura 36 – Abordagem de paralelismo por nível de linha do SGM. (a) Ajuste dos custos de caminhos anteriores através de atrasos (b) Estrutura para processamento de duas linhas



(BANZ et al., 2010)

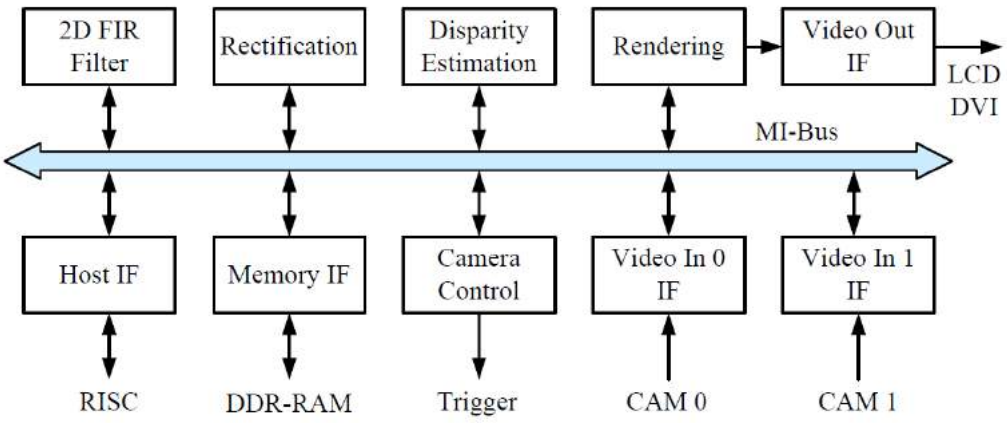
Este processamento em nível de linha compensa o atraso devido a dependência entre pixels vizinhos na mesma linha. Assim, enquanto o próximo pixel da mesma linha espera até que o anterior seja finalizado, a abordagem pode computar os pixels das linhas seguintes uma vez que os atrasos permitiram resolver as dependências de dados entre linhas. O uso desta estrutura permite uma redução no tempo de processamento da imagem inteira em um fator de número de linhas computadas em paralelo. Contudo, esta estrutura não suporta paralelismo em nível de disparidade, ou seja, a arquitetura só processa um valor de disparidade a cada ciclo. Se a quantidade de linhas não for igual ao espaço de disparidade, o fluxo de streaming de dados de entrada precisaria ser parado para dar tempo da operação ao longo da disparidade terminar. Assim, o módulo de otimização semi-global seria o gargalo do sistema inteiro necessitando armazenar toda imagem de entrada.

Uma vez que as imagens são enviadas para o módulo que implementa a técnica SGM linha por linha começando no canto superior esquerdo, os custos de caminho cujas direções são de cima para baixo são usados a fim de não aumentar a latência. Dessa forma, os autores utilizaram os quatro custos de caminhos para o SGM cuja as direções têm ângulos

de $t = 0°$, $t = 45°$, $t = 90°$, $t = 135°$. Dessa forma, o uso destas direções permite que o processamento do custo de caminho e agregação siga o mesmo fluxo de entrada dos pixels que são fornecidos pela câmera. Se fossem utilizadas as direções $t = 180°$, $t = 225°$, $t = 270°$, $t = 315°$ para compor as 8 direções propostas pelo algoritmo original do SGM (HIRSCHMULLER, 2008), o sistema precisaria esperar a chegada de todos os pixels da imagem inteira para poder iniciar o processamento dos custos destas direções. Assim a etapa de agregação precisaria esperar até que os custos de todas as direções tenham sido definidos para poder computar o somatório. Como consequência, o sistema precisaria de espaço para armazenar os pixels anteriores que não foram ainda processados e necessitaria de operações adicionais para agregar todos os custos e atrasaria ainda mais o término do cálculo do mapa de disparidade. Além disso, os autores evidenciam que a redução da quantidade de custo de caminho de 8 para 4 direções marginalmente piora a qualidade do mapa de disparidade e para sistemas de tempo real e em plataformas de baixo recurso o ganho significante em velocidade de processamento compensa por esta pequena perda de qualidade.

Além da etapa de estimação de disparidade com a técnica SGM, os autores implementaram todas as outras etapas do sistema de visão estéreo em FPGA tais como a captura de imagem, pré-processamento para remoção de ruídos, retificação e renderização para visualização do mapa de disparidade. Todas estas etapas foram implementadas como elementos de processamento (PE) independentes conectados através de um barramento compondo uma arquitetura totalmente em hardware. Um PE é ou uma interface (IF) para um dispositivo externo ou realiza uma tarefa de processamento de imagem específico. Um processador RISC é conectado via interface de barramento Host e pode configurar registradores, iniciar e terminar os PEs. Através do alto nível de programabilidade do processador RISC, um alto grau de flexibilidade para o controle do sistema e funções de sincronismo é alcançado.

Figura 37 – Arquitetura modular do sistema estéreo completo totalmente em FPGA



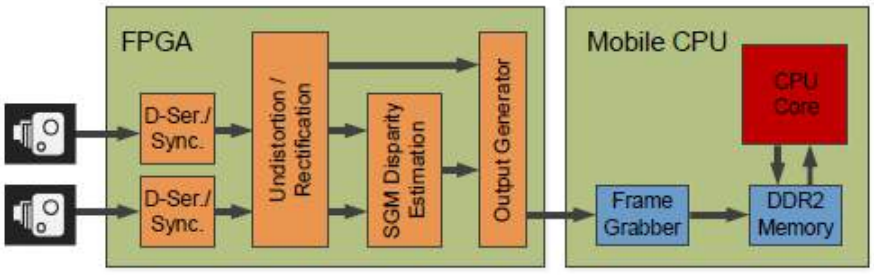
(BANZ et al., 2010)

A prototipação deste sistema completo foi realizada na FPGA Xilinx Virtex-5 gerando mapas de disparidades em resolução VGA (640 x 480 pixel) com um espaço de disparidade de 128 pixels em uma taxa de 30 fps e frequência de 39 MHz. Este sistema inteiro com 10 linhas paralelas, consumiu um total de 68.427 LUTs, 2346 LUTRAM, 183 BRAMs de 18Kb e 50 fatias de DSP.

Os autores reportaram também a taxa de processamento apenas do módulo de SGM variando o número de linhas paralelas onde os autores encontraram com um espaço de disparidade 128 valores para a melhor configuração com 30 linhas paralelas uma taxa de 103 fps e para pior configuração com 10 linhas paralelas uma taxa de 37 fps. Com um espaço de disparidade de 64 valores para a melhor configuração com 30 linhas paralelas o sistema obteve uma taxa de 167 fps e para a pior configuração uma taxa de 66 fps. Para melhor configuração, o módulo SGM usa 45.614 Luts, 7.456 LutRAM and 120 18Kb-BRAMs e para pior configuração 17.743 Luts, 2.336 LutRAM and 40 18Kb-BRAMs. Neste tipo de sistema o número de recurso não varia com o aumento espaço de disparidade, mas o tempo para processamento da imagem inteira aumenta.

Os autores reportaram também a qualidade da abordagem proposta no benchmark Middlebury usando a métrica de porcentagem de pixels errados onde os autores consideram erro quando a diferença da disparidade entre a referência e o obtido pela abordagem é maior do que 1 unidade). Dessa forma, os autores encontram uma taxa 6.8% de pixels errados para para a imagem estéreo Tsukuba, 4.1% para Venus, 13.3% para Teddy e 9.5% para Cones.

Figura 38 – Abordagem hardware/software proposto por Honegger para acelerar o algoritmo SGM



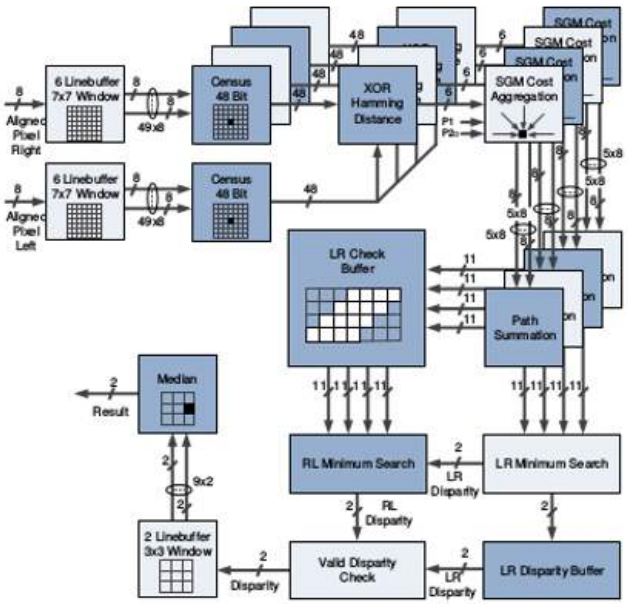
(HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014)

No trabalho proposto por Honegger (HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014), os autores desenvolveram um sistema de visão estéreo que compreende a combinação de FPGA com processador de propósito geral para aplicações móveis para superar as limitações computacionais e de latência na computação do mapa de disparidade da técnica SGM. Como mostrado na Figura 38, no lado do FPGA os autores implementam a etapa de retificação e o cálculo de custo de correspondência semi global (SGM) e no lado do

processador de propósito geral os autores implementam funções simples tal como coletor de frame.

Os autores propuseram uma implementação da técnica SGM em FPGA com paralelismo em nível de disparidade operando fatias de quatro valores de disparidade ao mesmo tempo, como mostrados na Figuras 39 e 40. Este tipo de paralelismo permite uma redução no tempo total para processar a disparidade em um fator da profundidade do paralelismo. Contudo, em virtude da forte dependência de dados com os pixels vizinhos o início do cálculo do caminho do próximo pixel precisaria ser atrasado uma certa quantidade de tempo até que o cálculo do custo de caminho do pixel vizinho anterior para todos os valores de disparidades tenha sido finalizado. Este atraso compromete o fluxo de entrada dos pixels fornecidos pela câmera que precisaria ser interrompido para acompanhar o processamento. Para compensar esta desvantagem, os autores aumentaram em 10 vezes a frequência do módulo SGM (250 MHz) em relação a frequência da câmera (25 MHz) a fim de garantir o término do custo de caminho para todas as disparidades antes que o próximo pixel seja liberado a partir da câmera. Para aplicações que demandam um pequeno intervalo de disparidade este aumento de frequência pode ser satisfatório, mas para aplicações de alta precisão que exigem grande intervalo de disparidades (acima de 128 níveis), o sistema necessitará de frequências mais altas, sendo portanto não adequado para implementações baseadas em plataformas FPGAs.

Figura 39 – Arquitetura para a implementação do SGM em FPGA: organização geral compreendendo todos passos do SGM

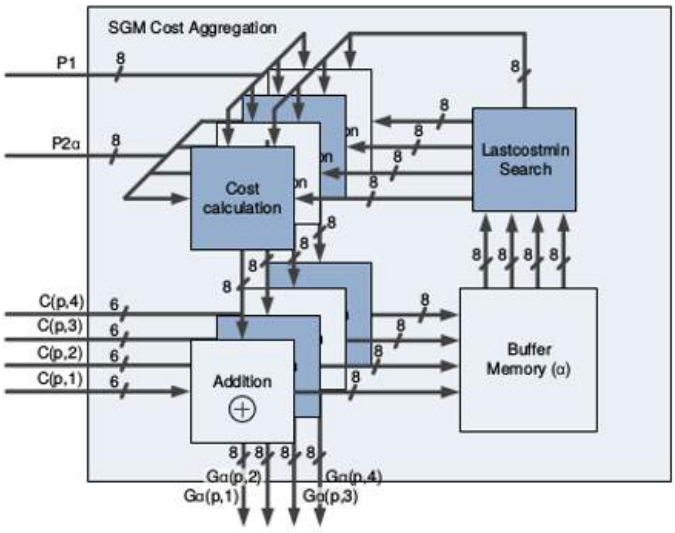


(HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014)

Neste trabalho, os autores implementaram como custo inicial a transformada census. A transformada census é o método local baseado em área e os autores utilizam uma janela de

suporte de tamanho 7x7. Os autores argumentam que a máscara census é menos sensível a mudanças de iluminação do que outros algoritmos locais e que o ganho interno e funções de exposição dos sensores de imagem podem ser definidos independentemente para cada câmera. Contudo, para se ter uma qualidade aceitável em aplicações práticas é necessário uma janela de suporte grande o que pode exigir mais hardware e além disso janelas grandes podem gerar erros em regiões no cenário que possuem estruturas finas e descontinuidades de profundidade como explicado na seção 2.2.

Figura 40 – Arquitetura para a implementação do SGM em FPGA: módulo de custo de caminho



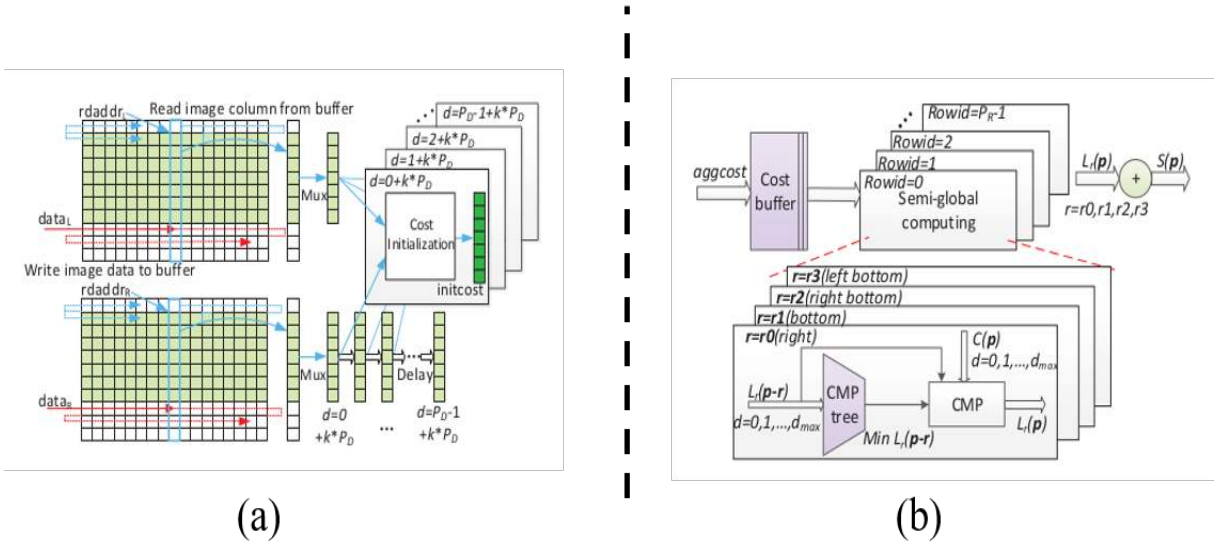
(HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014)

No trabalho proposto por Wang (WANG et al., 2013), os autores desenvolveram uma arquitetura para implementar a técnica SGM baseada tanto paralelismo a nível de disparidade como paralelismo a nível de linhas. A arquitetura processa simultaneamente um conjunto de pixels em linhas subsequentes na mesma coluna da imagem e para cada pixel é calculado paralelamente uma certa quantidade de valores de disparidade como mostrado na Figura 41(a). Neste tipo de arquitetura, o ganho de desempenho são dependentes da quantidade de disparidades e linhas processadas paralelamente.

Uma vez que não existe dependência em relação a função de custos de caminho de diferentes direções, os autores propõem o processamento paralelo destes custos através de módulos paralelos similares onde cada um processa um custo de caminho em uma dada direção como mostrado na Figura 41(b). Posteriormente estes custos de caminho são agregados para compor o custo final. Os autores adotaram os quatro custos de caminhos cuja as direções têm ângulos $t = 0°$, $t = 45°$, $t = 90°$, $t = 135°$. Para implementar o paralelismo em nível de linhas, os custos otimizados das linhas acimas são usados como entradas para as linhas atuais para as direções adotadas.

Nesta abordagem a quantidade de disparidades calculadas por ciclo não é igual ao espaço total de disparidade. Dessa forma o módulo que implementa a técnica SGM precisaria ficar dedicado, processando apenas um pixel de uma única linha ao longo de toda a disparidade o que levaria uma certa quantidade de ciclos para terminar. Assim com um único módulo SGM o processamento do custo de caminho do pixel da próxima linha não poderia ser inicializado até que o cálculo do custo de caminho do pixel da linha anterior tenha sido finalizado. Desta forma, para conseguir o paralelismo em nível de linha, os autores teriam que replicar este módulo SGM para computar o pixel de cada linha da imagem. Isto demandaria uma grande quantidade de recurso no FPGA. Além disso, devido ao número de linhas paralelas ser menor que a quantidade de ciclos para processar o custo de caminho de um pixel, o sistema todo precisa travar a entrada de novos pixels até que o módulo tenha terminado o processamento do custo de caminho deste conjunto de pixels, comprometendo o fluxo de entrada dos pixels vindo da câmera. Além disso o sistema precisaria armazenar os novos pixels que ainda não foram processados, exigindo assim grandes espaços de armazenamento, além de ser necessário descartar frames para poder terminar o processamento.

Figura 41 – Arquitetura proposta por Wang para a computação da técnica SGM. (a) Paralelismo em nível de linha e disparidade. (b) Módulos para a computação paralela do custo de caminho para várias direções

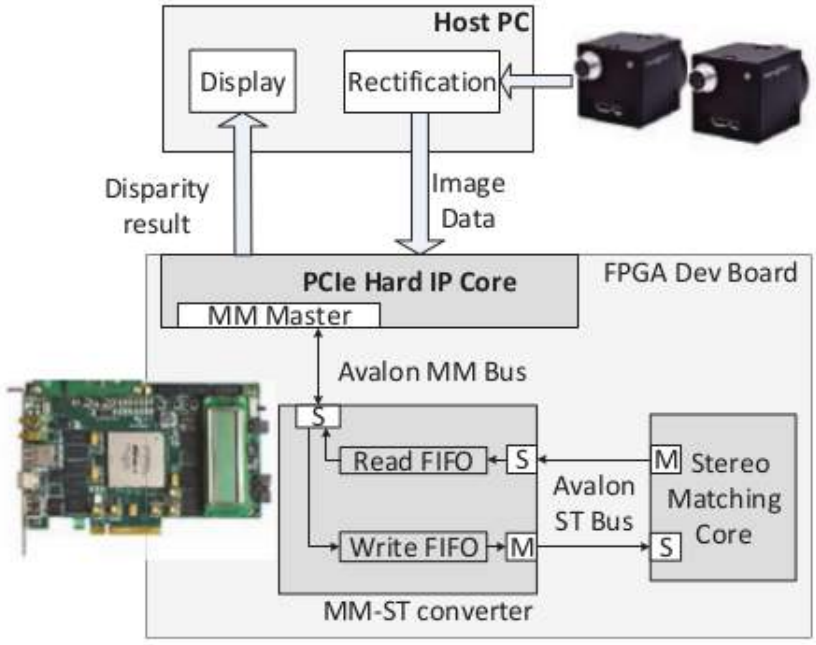


(WANG et al., 2013)

Além da arquitetura, os autores adotaram para a etapa de custo inicial um algoritmo local baseado na combinação entre diferença absoluta e transformada census. A transformada census é um algoritmo local baseado em área e, como já dito na seção 2.2, os algoritmos baseados em área falham em regiões de estruturas finais e em regiões de descontinuidade de profundidade. Para reduzir essa falha, os autores propõem uma arquitetura

em hardware da técnica de janelas dinâmicas proposto por (ZHANG; LU; LAFRUIT, 2009) para a transformada census.

Figura 42 – Arquitetura usada para validar a abordagem proposta por Wang



(WANG et al., 2015)

Os autores reportaram vários resultados de consumo de hardware e desempenho em fps para várias resoluções e tamanhos de intervalo de disparidades diferentes modificando a quantidade de linhas paralelas e a quantidade de disparidades operadas em paralelo. Para a resolução de 640x480 com disparidade 64 a melhor configuração obtém o desempenho de 244.1 fps, com um gasto de recurso em FPGA de 139.009 LUTs, 79.314 registradores e aproximadamente 5 mega bytes de memória RAM. Para a resolução de 1024x768 com espaço de disparidade 96 a melhor configuração obtém o desempenho de 63.58 fps, com um gasto de recurso em FPGA de 161.999 LUTs, 92.345 registradores e aproximadamente 10 megabytes de memória RAM.

Os autores reportaram também a qualidade da abordagem proposta no benchmark Middlebury usando a métrica de porcentagem de pixels errados onde os autores consideram erro quando a diferença da disparidade entre a referência e o obtido pela abordagem é maior do que 1 unidade). Dessa forma, os autores encontraram uma taxa 2.95 % de pixels errados para para a imagem estéreo Tsukuba, 1.43 % para Venus, 13.8 % para Teddy e 11.1 % para Cones.

Para validar esta implementação em um sistema de visão estéreo completo, os autores do trabalho proposto por (WANG et al., 2015) utilizam uma arquitetura composta de um processador de propósito geral e uma FPGA como mostrada na Figura 42. O software

executando no processador de propósito geral obtém os dados a partir de câmeras estéreo e envia estes para o módulo SGM implementado no FPGA através da interface PCI-Express. A retificação é também processada em software uma vez que processadores atuais são capazes de processar esta etapa em tempo real. O mapa de disparidade final é lido de volta a partir do FPGA e mostrado na interface executando no processador de propósito geral. No lado do FPGA é necessário introduzir os módulos de comunicação com o barramento PCI-Express o que exige uma quantidade de recurso em hardware.

Na Tabela 2, nós comparamos todos os trabalhos relacionados com relação a alguns aspectos que são críticos para o desenvolvimento da técnica SGM em hardware. Este tipo de comparação permite visualizar os pontos que precisam ser melhorados para se desenvolver arquiteturas otimizadas para o processamento do algoritmo SGM em termos de recursos de processamento e armazenamento, desempenho e qualidade do mapa de disparidade.

O uso de janelas de processamento de custo local grandes, tipicamente com tamanho de janela maiores que 3x3, como o que acontece nos trabalhos de (BANZ et al., 2010) e (HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014), certamente exige grande quantidade de recursos de armazenamento e processamento. A replicação do módulo SGM para o processamento da etapa de detecção de oclusão, como o que acontece no trabalho de (GEHRIG; EBERLI; MEYER, 2009) dobra a quantidade de recursos de processamento e armazenamento necessários para computar o mapa de disparidade final. O processamento de direções de propagação contrárias ao fluxo de entrada dos pixels, como o que acontece no trabalho de (GEHRIG; EBERLI; MEYER, 2009), aumenta o tempo de processamento do mapa de disparidade, além de exigir grandes espaços de armazenamentos de dados intermediários.

Todos estas desvantagens aliadas a necessidade de processamento de grandes resoluções e intervalos de disparidades fizeram com que os autores buscassem estratégias para lidar com a baixa vazão de processamento da técnica SGM. Nos trabalhos de (BANZ et al., 2010) e (HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014) os autores propuseram aumentar a frequência de operação em múltiplos da frequência da câmera. Já no trabalho de (GEHRIG; EBERLI; MEYER, 2009), os autores propuseram o uso de grandes memórias DDR2 para armazenar todos os pixels e custos intermediários que não foram ainda processados. Ambas as estratégias não são escaláveis para grandes intervalos de disparidades e resoluções de mapa de disparidades. Por outro lado, adoção do paralelismo de linha, permite aumentar a vazão de processamento, contudo nos trabalhos de (BANZ et al., 2010) e (WANG et al., 2013), foi observado que a quantidade de recursos de processamento é proporcional ao número de linhas que serão processadas em paralelo. Isto faz com que não seja possível definir um número de linhas suficientes para garantir altas vazões de processamento da técnica SGM, tendo que aumentar a frequência de operação ou aumentar a quantidade de armazenamento.

Tabela 2 – Comparativo dos trabalhos relacionados sob alguns critérios

Característica	Referência			
	(GEHRIG; EBERLI; MEYER, 2009)	(BANZ et al., 2010)	(HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014)	(WANG et al., 2013)
Tamanho de janela	3x3 (ZSAD)	9x9 (RANK)	7x7 (CENSUS)	5x5 (CENSUS)
Número de linhas de propagação	8	4	4	4
Detecção de oclusão	Duplica o módulo SGM	**Não** duplica o módulo SGM	**Não** duplica o módulo SGM	**Não** duplica o módulo SGM
Frequência mínima de operação (MHz)	Câmera	Múltiplo da câmera: Câmera: 21 SGM: 109	Múltiplo da câmera: Câmera: 25 SGM: 250	Câmera
Memória RAM externa	3 DDR 2	Não adotado	Não adotado	Não adotado
Paralelismo de linha	Não adotado	Adotado: LUT **varia** com o número de linhas de processamento em paralelo	Não adotado	Adotado: LUT **varia** com o número de linhas de processamento em paralelo

Como já foi mencionado, as aplicações atuais têm exigido sistemas de correspondência estéreo que sejam capazes de processar em tempo real mapas de disparidades precisos com resoluções e intervalos de disparidades cada vez mais altos (por exemplo, resoluções acima de 640x480 e intervalo de disparidades acima de 128 valores). Quanto maior estes parâmetros mais informações de profundidade serão possíveis de serem obtidas. Contudo, tem sido difícil, atingir tal feito com as abordagens propostas pelos trabalhos relacionados.

Além disso, tem-se observado nestes trabalhos que as abordagens usam geralmente todo o recurso disponível da FPGA para a computação do módulo de correspondência estéreo. No entanto, hoje em dia, espera-se que mais módulos sejam integrados em um sistema embarcado para realização de alguma determinada aplicação e todos estes módulos irão competir por recursos de processamento e armazenamento da FPGA. Portanto, é desafiador desenvolver arquiteturas mais compactas para processamento da técnica SGM. Dessa forma, este trabalho também foca em otimizações para o módulo de correspon-

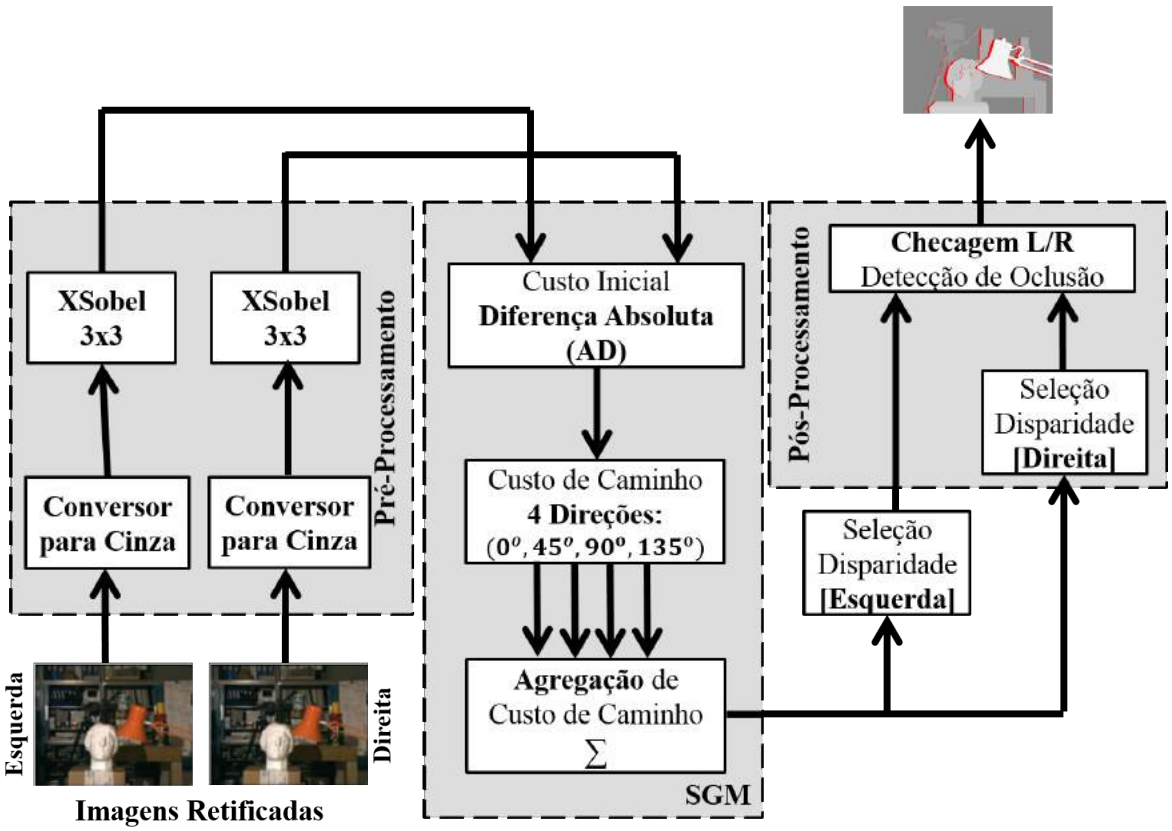
dência estéreo semi global a fim de reduzir a utilização dos recursos de processamento e armazenamento.

Assim, este trabalho propõe uma abordagem para o algoritmo SGM que atinge altas taxas de frames, utilizando menos recursos de hardware com relação aos trabalhos anteriores e ainda consegue atingir alta qualidade do mapa de disparidade que pode ser utilizada para a maioria das aplicações de visão estéreo.

4 PROPOSTA DE ARQUITETURA PARA O SGM

Uma visão geral da arquitetura proposta neste trabalho para a técnica de correspondência estéreo semi-global (SGM) é mostrada na Figura 43. Uma vez que diversas abordagens existem para cada etapa do sistema, então primeiramente são detalhadas as escolhas das técnicas adotadas para cada uma destas etapas. Em seguida é descrita a arquitetura desenvolvida para acelerar as etapas de custo inicial e a otimização da técnica SGM que demanda maior custo de processamento do sistema.

Figura 43 – Arquitetura geral da correspondência semi global proposta



4.1 Técnicas adotadas para as etapas de correspondência

Nesta seção serão dadas as motivações e os detalhes das abordagens adotadas em cada etapa que compõem a arquitetura geral do SGM, mostrado na Figura 43. Em seguida, será detalhado a técnica de paralelismo proposta neste trabalho, e, por fim, será detalhada a arquitetura em hardware que dá suporte a técnica de paralelismo, esmiuçando todos os módulos envolvidos na computação da técnica SGM.

4.1.1 Pré processamento

A etapa de *pré-processamento*, como já foi mencionado na seção 2.2, é geralmente realizada para tornar as informações na imagem mais discriminativas e aumentar, assim, a precisão e robustez do algoritmo de correspondência. Uma das maneiras comumente utilizada é através de emprego de filtros. Os filtros reduzem os ruídos dos sensores de câmeras, sendo os mais usados os filtros de média, filtros de mediana e filtros de laplaciano da gaussiana. Estes filtros conseguem reduzir os efeitos de ruído das câmeras mas acabam borrando as imagens de entrada e por consequência fazem com que o sistema de correspondência gere mapas de disparidade borrados.

Por outro lado, o filtros baseados em gradiente tal como o de *Sobel* (GONZALEZ; WOODS, 2006) não borram o resultado e eliminam diferenças de offset entre as imagens estéreo devido ao uso do operador de derivada. Para evitar linhas horizontais que podem dificultar a busca pelos pixels correspondentes, o filtro de Sobel adotado neste trabalho, chamado de XSobel, computa o gradiente apenas na direção horizontal da imagem (Hirschmuller e Gehrig (2009)). A função de kernel é mostrada na Equação 4.1. Uma vez que a resposta deste filtro pode compreender valores negativos, os resultados são normalizados para valores positivos entre o intervalo de 0 a 255.

Juntamente com o filtro é empregada a etapa de conversão para escala de cinza. A redução de 3 canais RGB para 1 canal permite a redução da quantidade de operações necessárias, favoráveis a implementação em hardware. Além disso, foi possível observar por experimentos que a redução de canais não comprometeu a qualidade do mapa de disparidade. No trabalho de (BENEDETTI et al., 2012) os autores fazem um estudo dos diversos métodos de conversão para escala de cinza e afirma que dependendo do tipo de conversão a ser realizado, é possível obter qualidades de mapas de disparidades superiores aos mapas obtidos usando técnicas que envolvem o uso do espaço de cor RGB.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.1}$$

4.1.2 Custo inicial

Para a etapa de computação do *custo inicial* foi adotado o método local de diferença absoluta (AD) como mostrado na Equação 4.2. Devido a sua simplicidade algorítmica este método tem sido bastante adotado por muitas abordagens de correspondência estéreo em plataformas de hardware. Além disso, a combinação do filtro XSobel com técnica AD e com SGM tem permitido obter resultados bastante satisfatórios em termos de qualidade de mapa de disparidade. Esta combinação tem resultado em mapas com qualidade comparados as obtidas com abordagem Census, que é uma das melhores abordagens locais conhecida na literatura Hirschmuller e Scharstein (2009). Contudo a abordagem Census

exige janelas grandes para obter ótimos resultados enquanto que a XSobel só precisa de uma janela de tamanho 3x3. A combinação XSobel-AD-SGM consegue lidar satisfatoriamente com os desafios de correspondência estéreo tais como distorções radiométricas, e até com pequenos erros de calibração, como evidenciados no trabalho de Hirschmuller e Gehrig (2009). Nos resultados deste trabalho poderá ser observada a robustez desta combinação, tanto em imagens reais de baixa qualidade obtidos a partir do sistema de câmeras quanto em imagens de alta qualidade disponíveis no benchmark Middlebury (SCHARSTEIN; SZELISKI, 2002). A equação de custo inicial com diferença absoluta (AD) é descrita a seguir.

$$C(i,j,d) = |I_L(i,j) - I_R(i,j-d)| \tag{4.2}$$

Basicamente esta equação realiza uma diferença absoluta entre o valor de intensidade do pixel na imagem da esquerda localizada na posição (i,j) com o pixel na imagem da direita localizada na posição $(i,j-d)$. Esta equação já foi detalhada na seção 2.3.

4.1.3 As direções do custo de caminho

A etapa de *custo de caminho* como detalhado na seção 2.3 otimiza o custo inicial através de propagações dos custos de caminho $L_{\mathbf{r}^t}(\mathbf{p},d)$ como mostrado na Equação 4.3 ao longo de várias direções $\mathbf{r}^t$ ($L_{\mathbf{r}^{0^\circ}}(\mathbf{p},d)$, $L_{\mathbf{r}^{45^\circ}}(\mathbf{p},d)$,..., $L_{\mathbf{r}^{315^\circ}}(\mathbf{p},d)$) unidimensionais independentes em toda a imagem. A etapa de *agregação* que soma as contribuições dos custos em todas as direções adotadas resultando S_t, define o custo final otimizado como mostrado na Equação 4.4.

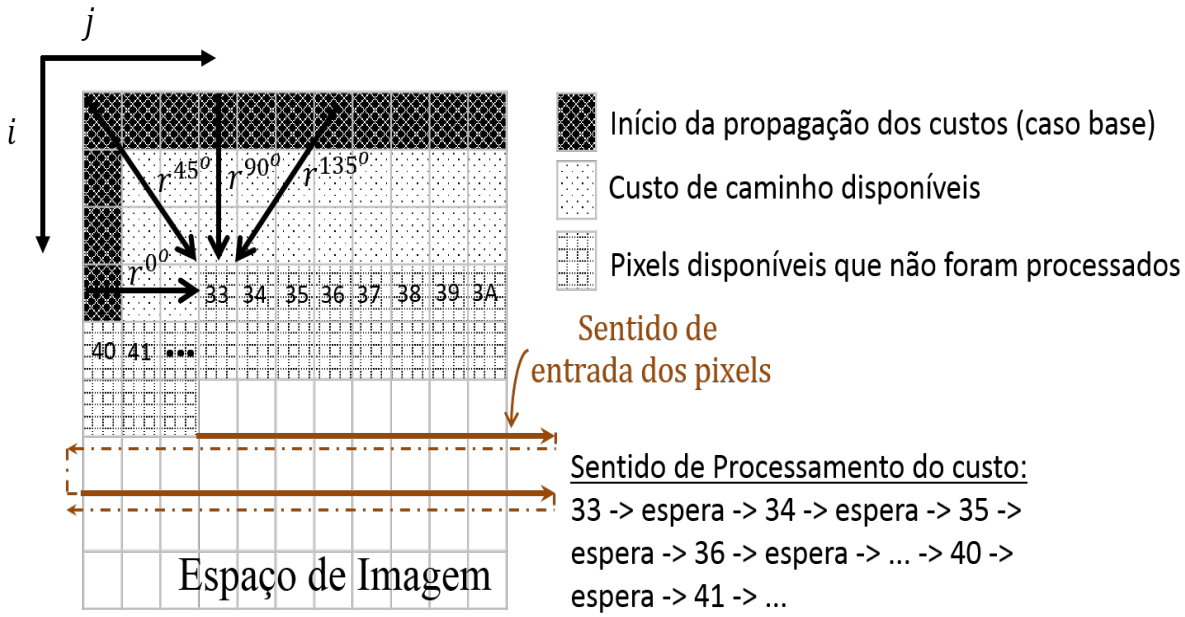
$$\begin{aligned} L_{\mathbf{r}^t}(\mathbf{p},d) = C(\mathbf{p},d) + \min[L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t,d), L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t,d-1)+P_1, \\ L_{r^t}(\mathbf{p}+\mathbf{r}^t,d+1)+P_1, \min_{i\in[0,N_d-1]} L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t,i)+P_2] - \min_{k\in[0,N_d-1]} L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t,k) \end{aligned} \tag{4.3}$$

$$S(\mathbf{p},d) = \sum_{t\in\{S_t\}} L_{\mathbf{r}^t}(\mathbf{p},d) \tag{4.4}$$

No trabalho proposto por Hirschmuller (HIRSCHMULLER, 2008) é apresentada uma implementação da técnica SGM original tendo 8 direções para o custo de caminho. Este trabalho demonstrou que quanto mais direções são consideradas, melhor é a qualidade do mapa de disparidade, pois isto agrega mais informações da imagem contidas em direções diferentes. Contudo, a maioria das abordagens implementadas em hardware reconfigurável (FPGA), que buscam responder em tempo real, têm adotado os 4 caminhos dados por: $\mathbf{r}^{0^\circ}$, $\mathbf{r}^{45^\circ}$, $\mathbf{r}^{90^\circ}$, $\mathbf{r}^{135^\circ}$. Estes caminhos favorecem o fluxo de entrada dos pixels fornecidos a partir de câmeras. Em outras palavras, a escolha destas direções para o cálculo do custo de caminho é devido ao processo de varredura da imagem iniciar na borda superior esquerda que é onde também se inicia o cálculo do custo de caminho para estas direções,

como mostrado na Figura 44. É possível observar que, seguindo o fluxo de processamento mostrado na Figura 44, se já existir pixels disponíveis, o processamento do custo de caminho já pode ser realizado, pois a condição de iniciar a propagação já foi satisfeito pelos primeiros pixels de borda.

Figura 44 – Sentido de entrada dos pixels e o sentido de processamento dos custos de caminho para as direções adotadas



Se fossem utilizados as direções contrárias tais como $\mathbf{r}^{135°}$, $\mathbf{r}^{180°}$, $\mathbf{r}^{225°}$, $\mathbf{r}^{270°}$ e $\mathbf{r}^{315°}$ para compor as 8 direções propostas na versão original da técnica SGM, o processamento do custo para tais direções só poderia começar quando o último pixel da imagem estivesse disponível. Isto é devido ao fato de que o processamento começa a partir do último pixel no canto inferior direito da imagem e processa no sentido contrário de baixo para cima até o primeiro pixel. Dessa forma, seria necessário armazenar toda a imagem pois, os pixels do restante superior da imagem não foram ainda processados a partir destes caminhos. Assim, para resoluções grandes da imagem, o sistema precisaria de grande espaço de armazenamento para guardar toda a imagem, o que pode não ser possível em plataformas baseadas em FPGA que possui pouco recurso de armazenamento. Além disso, seria necessário armazenar os custos de caminho das direções que já foram computadas para poder posteriormente acumular com o resultado do custo de caminho destas direções contrárias. Além do problema de ter que armazenar toda a imagem e os custos intermediários, a espera pelo último pixel para que o processamento do custo seja inicializado para as direções contrárias, atrasaria todo o processamento e poderia levar o sistema a ter que descartar frames enquanto o processamento do frame anterior ainda estivesse sendo realizado, comprometendo o desempenho do sistema.

Ao utilizar os custos de caminho para as quatro direções ($\mathbf{r}^{0°}$, $\mathbf{r}^{45°}$, $\mathbf{r}^{90°}$, $\mathbf{r}^{135°}$), o processamento dos custos de caminho para todas estas direções poderão ser realizados e os custos já poderão ser acumulados e liberados como resultado, a medida que os pixels de entrada estejam disponíveis. Dessa forma, o sistema não precisará armazenar a imagem toda, nem armazenar o custo acumulado e, assim, será possível desenvolver arquiteturas que garantam a perda de nenhum frame como é o caso da arquitetura proposta neste trabalho.

Além disso, a redução da quantidade de caminhos de propagação de custos de 8 para 4 direções marginalmente piora a qualidade do mapa de disparidade, como evidenciado no trabalho proposta por Banz (BANZ et al., 2010). Assim, para sistemas de tempo real executando em plataformas de baixo poder computacional, tal como em plataformas FPGAs, a pouca perda de qualidade do mapa de disparidade é justificada pelo ganho significante em velocidade de processamento e a redução significante da quantidade de recursos utilizados nas plataformas reconfiguráveis.

4.1.4 Pós processamento

Na etapa de *pós processamento* é feita a detecção de regiões de oclusão usando a abordagem de detecção rápida detalhada na seção 2.2. Esta abordagem é bastante utilizada em plataformas de hardware reconfigurável (FPGAs) pois exige bem menos recursos de hardware em relação a abordagem original, que exige replicação do módulo SGM para detecção de oclusão. Esta abordagem de detecção rápida utiliza o resultado da etapa de agregação da técnica SGM para gerar um segundo mapa de disparidades, como mostrado na Equação 4.5, suavemente diferente do mapa original D_L, mas que diverge em seus valores de disparidade em regiões de oclusão.

$$D_R(\mathbf{p}) = D_R(p_i, p_j) = d | \min_{d \in [0, N_d - 1]} S(p_i, p_j - d, d) \tag{4.5}$$

Assim, para cada posição $\mathbf{p}$ no mapa D_L, é verificado se seu valor de disparidade difere de um certo limiar do valor de disparidade na mesma posição no mapa D_R, como mostrado na Equação 4.6. Se sim, a disparidade na posição $\mathbf{p}$ é rotulada como ocludida, senão a disparidade do mapa D_L é então retornada para compor o mapa final D.

$$D(\mathbf{p}) = \begin{cases} D_L(\mathbf{p}), & \text{if } |D_L(\mathbf{p}) - D_R(\mathbf{p})| \leq \sigma \\ D_{inv}, & \text{Caso Contrário} \end{cases} \tag{4.6}$$

4.2 Acelerando o processamento do método SGM

Nesta seção serão detalhadas as dependências de dados do algoritmo SGM, os tipos de paralelismos existentes e as deficiências destes paralelismos. Em seguida, é proposta uma arquitetura para processar uma nova abordagem de paralelismo para o SGM.

4.2.1 Dependência de dados e tipos de paralelismos existentes

Como foi esclarecido na seção 2.3, para decidir sobre o valor de disparidade final de um dado pixel $\mathbf{p}$ é necessário calcular para cada disparidade d os custos de caminho $L_{r^t}(\mathbf{p}, d)$ para todas direções $\mathbf{r}^t$ consideradas, como mostrado na Equação 4.3 e, em seguida, calcular o valor agregado destes custos de caminho para compor o custo final, $S(\mathbf{p}, d)$, como mostrado na Equação 4.4. Dado que a imagem tem largura igual a W e tem o comprimento igual a H, o tamanho do intervalo de disparidades é dado por N_d e a quantidade de custos de caminhos é dada por N_t, a complexidade para operar toda a imagem é da ordem de O $(W \cdot H \cdot N_t \cdot N_d)$.

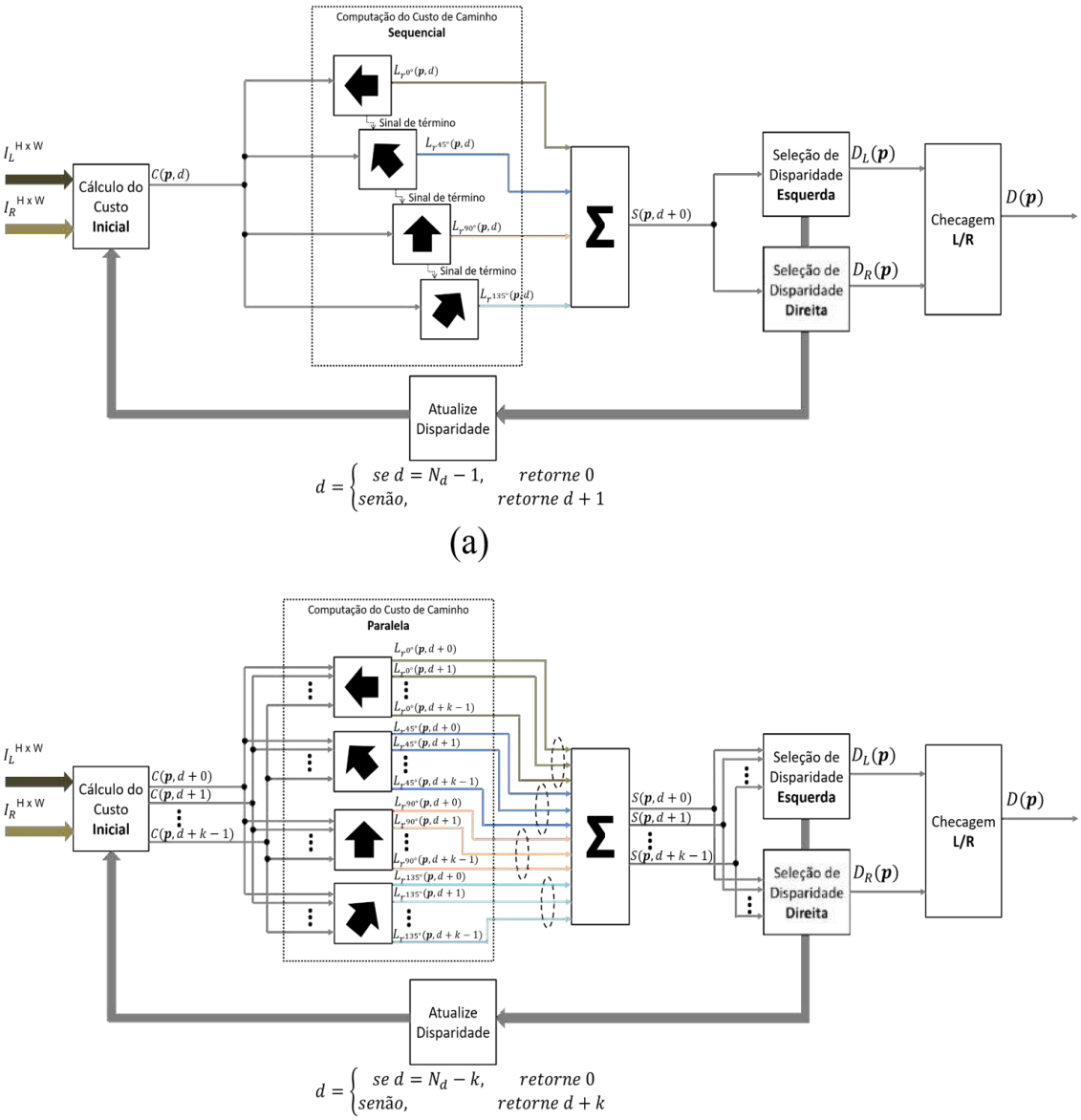
As aplicações atuais geralmente têm exigido espaços de disparidade maior ou igual a 128 valores ($N_d \geq 128$), com uma quantidade mínima de 4 custos de caminhos (geralmente são para as direções de $\mathbf{r}^{0^\circ}$, $\mathbf{r}^{45^\circ}$, $\mathbf{r}^{90^\circ}$, $\mathbf{r}^{135^\circ}$) e em resolução VGA (W = 640 e H = 480) ou superiores. Dependendo da plataforma e da abordagem de implementação adotada, esses requisitos podem comprometer o desempenho do algoritmo que implementa a técnica SGM. Por exemplo, implementações mais recentes do método SGM em plataformas baseadas em processadores para imagens de resolução VGA com 128 níveis de disparidades possuem taxas de processamento de 16 FPS (GEHRIG; RABE, 2010). Esta taxa está abaixo do mínimo exigido pelas aplicações de tempo real (BANZ et al., 2010). Dessa forma, são exigidas novas abordagens para acelerar o processamento da técnica SGM.

Uma maneira de se conseguir a melhoraria de desempenho é através de paralelismo; isto é, computando todas as funções de custo de caminho e também computando para cada função um conjunto de valores de disparidade de uma única vez. A Figura 45 mostra a versão original da técnica SGM sem paralelismo e a versão que explora paralelismo no cálculo da disparidade e das funções de custo de caminho. Fatias de k disparidades são computadas paralelamente, a cada iteração e para cada função de custo de caminho.

O primeiro tipo de paralelismo é conseguido em hardware através de módulos de custo de caminho que calculam ao mesmo tempo o custo de caminho para cada uma das direções. Estes módulos geralmente são bastante similares, no qual para cada direção as condições de borda são diferentes e podem ser definidas como parâmetros de entrada do módulo. O segundo paralelismo é conseguido dividindo o intervalo de disparidades em um conjunto de fatias no qual dentro de cada fatia os custos para todas as disparidades são processados paralelamente. Este dois tipos de paralelismo permitem uma redução no tempo total de processamento de um determinado pixel em um fator da quantidade de fatias, bem como, da quantidade de módulos independentes para o cálculo de custo de caminho que foram paralelizadas. As implementações existentes em plataformas baseadas em processadores de propósito geral só conseguem definir o tamanho da fatia de no máximo 16 disparidades através de instruções vetoriais SIMD (mais detalhes em Gehrig e Rabe (2010)). Em plataformas baseadas em FPGAs, o tamanho da fatia pode ser muito maior do que 16 devido a capacidade que este tipo de plataforma tem de instanciar cente-

nas de adicionadores, deslocadores e comparadores para calcular o custo de caminho para cada disparidade. A limitação da quantidade de fatias depende do tamanho da FPGA adotada e da arquitetura desenvolvida.

Figura 45 – Arquiteturas de implementação da técnica SGM. (a) Versão original sem qualquer tipo de paralelismo do SGM e (b) é a versão do SGM com paralelismo em disparidade e em custos de caminho diferentes



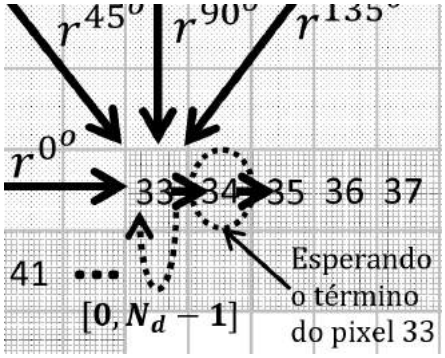
(a)

(b)

Embora a função de custo de caminho não possua a dependência do mesmo pixel para diferentes disparidades, esta função tem a dependência em relação ao custo de caminho de pixels vizinhos que impõem dificuldades para obter implementações eficientes desta função de custo caminho. Como pode ser visto na equação 4.3, para que o cálculo do custo de caminho para um dado pixel $L_{r^t}(\mathbf{p}, d)$ possa ser realizado é necessário que o custo de caminho do pixel vizinho anterior $\mathbf{p} + \mathbf{r}^t$, tenha sido determinado para todas as

N_d disparidades. Isto é devido ao fato do custo de caminho atual para cada disparidade d precisar dos custos de caminho do pixel anterior $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d)$, $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d+1)$ e $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d-1)$, bem como, do resultado do mínimo dos custos de caminhos para todas as disparidades $\min_{k\in[0,N_d-1]} L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, k)$. Este último termo força a necessidade de se ter todos os custos calculados para um dado pixel antes de se iniciar o cálculo para o próximo pixel. Em termos práticos, se o fluxo de processamento dos custos de caminho seguir o mesmo fluxo de entrega dos pixels, ou seja, processando todas as colunas da esquerda para direita antes de prosseguir para a próxima linha abaixo como mostrado na Figura 44, o caso de dependência mais crítico é com relação ao cálculo do custo de caminho na direção $\mathbf{r}^{0^\circ}$. Uma vez que o processamento do custo de caminho para computar todas as disparidades leva um certo tempo, o próximo pixel teria que esperar até que o pixel anterior na mesma linha tenha terminado seu processamento. Por exemplo, o processamento do custo de caminho do pixel rotulado como 34 não poderia começar o processamento antes que o cálculo do pixel anterior de rótulo 33 tenha finalizado o cálculo do custo de caminho na direção $\mathbf{r}^{0^\circ}$, pois o pixel 34 depende deste resultado como mostrado na Figura 46. No caso, para as outras direções $\mathbf{r}^{45^\circ}$, $\mathbf{r}^{90^\circ}$, $\mathbf{r}^{135^\circ}$ não existiria este problema pois os resultados do custo de caminho estaria na linha anterior que já teriam sido computados antes do início do processamento do pixel que precisará destes resultados na próxima linha.

Figura 46 – Atraso do processamento dos pixels devido a dependência do custo de caminho na direção $\mathbf{r}^{0^\circ}$

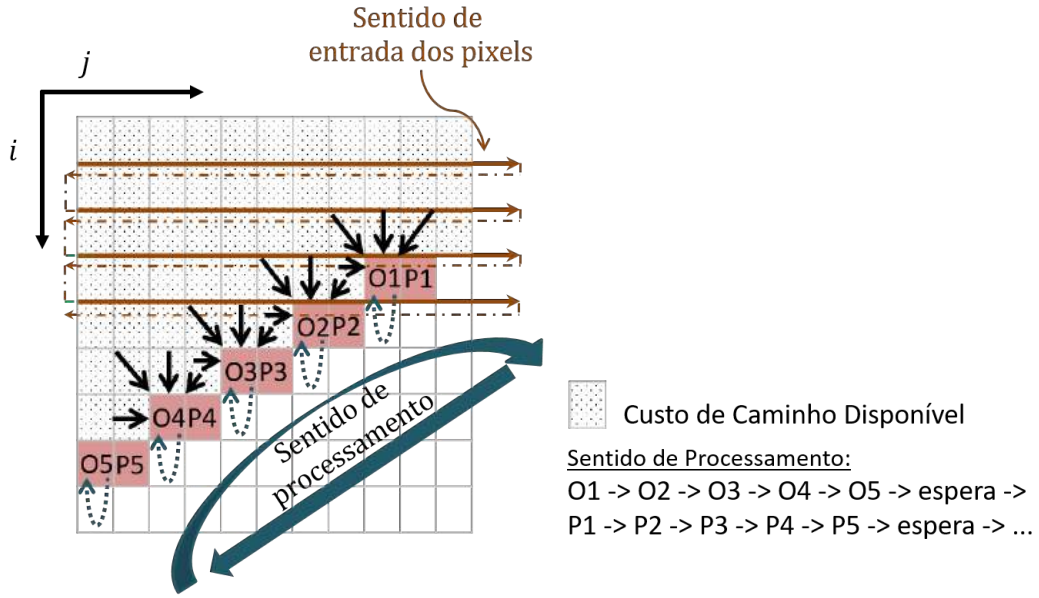


Esta espera faz com que a vazão de processamento do custo de caminho seja menor do que a vazão de entrega dos pixels de entrada, o que não é desejado, e assim o tempo de processamento de um mapa de disparidade inteiro seria comprometido. Além disso, a espera faz com que o sistema precise armazenar os pixels que ainda não foram processados, que geralmente totaliza quase a imagem inteira. Quando se trata de plataforma FPGA esta quantidade de espaço é um fator crítico e a medida que são exigidos resoluções maiores esta quantidade de espaço se torna infactível para tal plataforma.

Para compensar os ciclos de espera devido a dependência entre pixels vizinhos para a direção $\mathbf{r}^{0^\circ}$ é possível processar o custo de caminho de pixels das linhas seguintes. Este tipo de processamento é possível porque a vazão do processamento do custo de caminho de um determinado pixel é menor que vazão dos pixels de entrada e por isso existirão

mais pixels disponíveis a partir das próximas linhas aguardando para serem processados. Assim, enquanto o próximo pixel da mesma linha espera até que o anterior seja finalizado, o sistema pode computar os pixels das linhas seguintes como mostrado na Figura 47. Nesta figura é possível observar o processamento começando do pixel rotulado por O1, passando por O2 até O5 e depois retornando para o pixel P1 para terminar em P5 e assim por diante. Neste fluxo de processamento, é importante garantir a disponibilidade dos custos de caminho a partir da linha anterior durante o cálculo do custo de caminho dos pixels das linhas seguintes. Para isso é necessário introduzir distancias entre pixels de linhas adjacentes de maneira que os custos já estejam prontos na hora do processamento de um determinado pixel, fazendo com que o fluxo de processamento seja realizado na diagonal. Por exemplo, na Figura 47 os pixels O1 e O2 estão separados de dois pixels de distância e que por causa dessa distância, quando for calcular os custos de caminho para o pixel O2, os custos vizinhos já vão estar disponíveis.

Figura 47 – Sentido de processamento dos custos de caminho por linha da imagem



No trabalho desenvolvido por Wang (WANG et al., 2013), os autores propuseram uma abordagem que introduz o paralelismo de linha e também o paralelismo de disparidade. Nesta abordagem duas questões críticas podem ser observadas. A primeira questão é que a quantidade de disparidades computadas em paralelo não é igual ao intervalo total de disparidades. Eles processam no máximo 16 disparidades em paralelo. Como consequência, o módulo SGM, que compreende o módulo de custo inicial e quatro módulos para computar os custos de caminho, fica dedicado processando as múltiplas disparidades de apenas um único pixel. Assim, para conseguir o paralelismo em nível de linha, os autores precisam replicar o módulo SGM para computar o custo de caminho dos pixels de cada linha da imagem. Isto demanda uma grande quantidade de recursos no FPGA. A segunda questão é que o número de linhas computadas paralelamente é menor que a quantidade

de ciclos para processar o custo de caminho de um determinado pixel. Assim, o tempo de processamento de uma diagonal não seria grande o suficiente para garantir que quando for calcular o primeiro pixel da próxima diagonal, os custos de caminho dos pixels vizinhos já estejam disponíveis. Desta forma, a vazão de processamento do sistema seria menor do que a taxa de entrada dos pixels, o que obrigaria o sistema a ter que armazenar os novos pixels que ainda não foram processados, exigindo assim grandes espaços para armazenamento. Além disso, ainda seria necessário descartar frames constantemente para poder terminar o processamento atual do mapa de disparidade devido a taxa de processamento do sistema não ser igual a taxa de entrada dos pixel vindo da câmera.

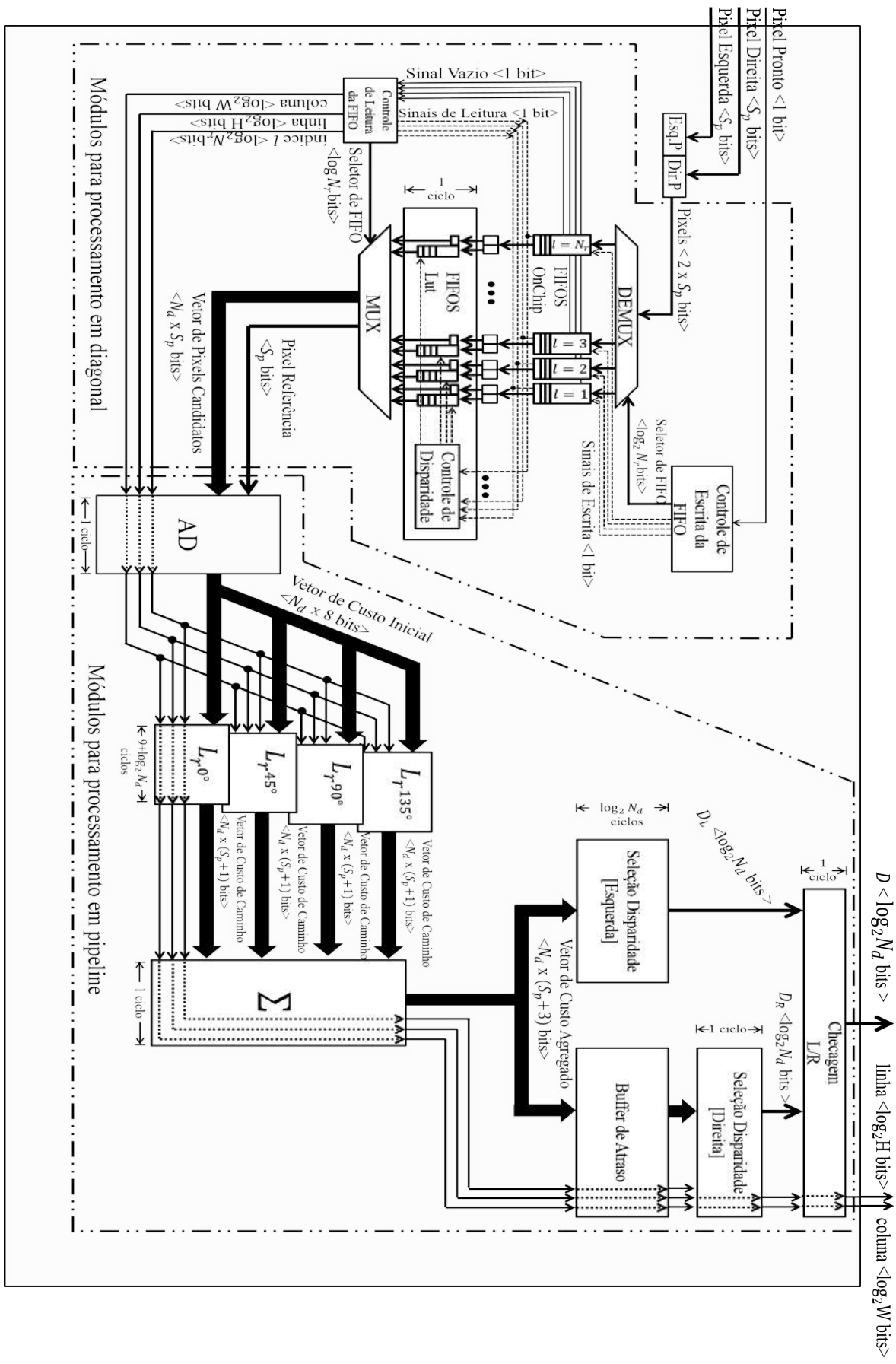
4.2.2 Arquitetura Proposta

Afim de contornar as desvantagens de alto custo de hardware da abordagem desenvolvida por Wang (WANG et al., 2013), este trabalho propõe uma arquitetura que introduz a técnica de *serialização* para processar os custos de caminho de todas as N_r linhas usando um único módulo SGM. Além disso, para dar suporte à serialização, a arquitetura proposta neste trabalho realiza o processamento através de vários estágios independentes para processar o algoritmo de correspondência semi global e o processamento paralelo de todas as disparidades e de todas as diferentes funções de custo de caminho.

A arquitetura proposta para o método SGM é mostrada na Figura 48. De forma simplificada, o módulo despacha pixels de várias linhas ciclo após ciclo que serão processados por vários estágios em pipeline responsáveis pela implementação da técnica SGM. Em cada estágio operações simples e sem aninhamento são implementadas buscando atingir a eficiência máxima de frequência da operação na plataforma baseada em FPGA. Uma vantagem da arquitetura proposta, como será mostrado nos resultados, é que a frequência de operação se torna invariável com mudanças dos parâmetros do sistema devido à organização da arquitetura em pipeline.

Neste trabalho a quantidade de linhas de processamento é sempre igual a quantidade de ciclos de relógio necessários para calcular um pixel ao longo de todos os estágios do pipeline. Assim, a cada ciclo um pixel de uma linha diferente estará sempre sendo processado pelo módulo. Como consequência, o módulo SGM processa a uma taxa de um pixel de disparidade por ciclo de relógio, a mesma taxa de entrada dos pixels (isto será reiterado na seção 6). Essa taxa de processamento é uma das principais contribuições deste trabalho. Além do alto desempenho de processamento, o sistema necessita de espaços de armazenamento de tamanho fixo e reduzido para o armazenamento dos pixels não processados e dos valores de custos de caminho intermediários. Uma outra vantagem da arquitetura proposta é que ela é totalmente parametrizável em termos de resolução de imagem, intervalo de disparidades a ser computado, permitindo assim que esta arquitetura possa ser configurada de acordo com os requisitos da aplicação em termos de precisão do mapa de disparidade.

Figura 48 – Arquitetura proposta para a técnica de correspondência estéreo semi global



Do ponto de vista mais externo, o módulo armazena um conjunto de pixels das duas imagens contidos em N_r linhas da imagem através de estruturas do tipo FIFO (First-In-First-Out). Cada FIFO é responsável por uma determinada linha da imagem. O pixel da imagem da esquerda e o pixel da imagem da direta são concatenados antes de serem armazenados na FIFO e quando forem acessados pelo módulo *Controle de disparidade*, são separados. A cada ciclo um conjunto de N_d pixels candidatos e o pixel de referência a partir destas FIFOs são despachados para serem processados estágio por estágio passando pelos módulos que implementam o custo inicial (AD), os custos de caminho ($\mathbf{r}^{0^\circ}$,$\mathbf{r}^{45^\circ}$, $\mathbf{r}^{90^\circ}$, $\mathbf{r}^{135^\circ}$), agregação, seleção de disparidade (Esquerda e Direita) e de detecção de oclusão (checagem LR). O módulo que calcula a diferença absoluta (AD) recebe um vetor de N_d de pixels candidatos e um pixel de referência onde o pixel tem largura de S_p bits de largura e computa os N_d valores da diferença absoluta entre o pixel referência para todos os pixels candidatos ao mesmo tempo em um único ciclo de relógio. Este módulo devolve um vetor de N_d custos iniciais onde cada custo é representado por um vetor de S_p bits de largura. Este vetor é utilizado nos quatro módulos de custo de caminho que calculam o custo de caminho em vários estágios de processamento, onde cada estágio é um ciclo de relógio, e no final devolve o vetor de N_d custos de caminho onde cada custo tem S_p+1 bits de largura. Os quatro vetores de custo de caminho são então somados para cada valor de disparidade no módulo de agregação, que devolve um vetor de N_d custos otimizados onde, cada custo é representado por um vetor de $S_p + 3$ bits bits de largura. Este vetor é utilizado pelos módulos de *Seleção Disparidade* para obter as disparidades D_L e D_R de menor custo. Estes dois valores de $log_2 N_d$ bits são usados pelo módulo *Checagem LR* que devolve o resultado final. Para pixels ocludidos é retornado o valor 0. Neste trabalho, a largura de bits S_p do pixel tem valor 8, mas este parâmetro pode ser modificado de acordo com a configuração das câmeras. Embora não esteja representado na Figura 48, cada módulo sinaliza o término de sua operação e envia seu resultado para serem calculados pelo próximo módulo e os sinais de relógio e reset existem em todos os módulos. O módulo *Buffer de Atraso* é usado para alinhar o resultado dos módulos de seleção de disparidade esquerda e direita para o processamento pelo módulo de checagem LR.

Dois tipos de FIFOs são empregados para computar as N_r linhas: uma FIFO Onchip e uma FIFO LUT. A FIFO Onchip armazena os pixels recebidos de cada linha que ainda não foram processados. O módulo *Controlador de Escrita da FIFO* define em qual FIFO Onchip o pixel será armazenado através da seguinte lógica: a FIFO de índice l armazena os pixels a partir das linhas $l + i_s \cdot N_r$, onde i_s é o índice do conjunto. Por exemplo, as linhas que serão armazenadas na FIFO de índice $l = 1$, são $(1 + 1 \cdot N_r), (1 + 2 \cdot N_r), (1 + 3 \cdot N_r)$ e assim por diante. A FIFO Onchip não usa os elementos lógicos do FPGA e sim suas memórias internas que são destinadas a armazenamento e acesso rápido dos dados (BAILEY, 2011). As memórias internas permitem que grandes quantidades de

pixels que ainda não foram processados sejam armazenados e ainda assim, tenham espaços para armazenamento dos custos de caminho, como será visto adiante. Isto evita que os elementos lógicos do FPGA que já são relativamente escassos sejam utilizados para armazenamento e sim exclusivamente para processamento do custo de caminho. Porém, este tipo de estrutura de armazenamento só permite ler um pixel por vez, o que não permite extrair o paralelismo de disparidade que é necessário para a abordagem proposta.

Dessa forma, para garantir o processamento paralelo dos N_d valores de disparidades, *FIFOs LUT*, são usadas com tamanho de N_d pixels para cada linha do conjunto de N_r linhas. Estas FIFOs permitem acesso direto a todos os N_d pixels candidatos ao mesmo tempo e também do pixel referência. Na medida em que um novo pixel chega na sua respectiva FIFO LUT, a partir da FIFO Onchip, o módulo *Controle de Disparidade* descarta o último pixel desta FIFO através da operação de deslocamento a esquerda (do inglês shift-left), insere o novo pixel na sua primeira posição e despacha os N_d pixels candidatos e o de referência, para que o custo através do algoritmo SGM, seja calculado começando pelo módulo de custo inicial AD. Este processo leva um ciclo de relógio. Quando uma linha da imagem é finalizada, os dados da FIFO LUT desta linha são zerados, para que possa se processar uma nova linha. Desta forma, o fornecimento dos dados dos pixels de referência e dos N_d candidatos ao mesmo tempo, a partir destas FIFOs, estarão sempre bem organizados para que possam ser processados de uma única vez, em cada módulo que implementa o algoritmo SGM.

Além do processamento paralelo de todas as N_d disparidades, duas outras importantes abordagens foram desenvolvidas para dar suporte à serialização: o fluxo de processamento em diagonal e o processamento em pipeline dos custos de caminho. A seguir são explicados com mais detalhes estas duas abordagens.

4.2.2.1 Fluxo de processamento de disparidade em diagonal

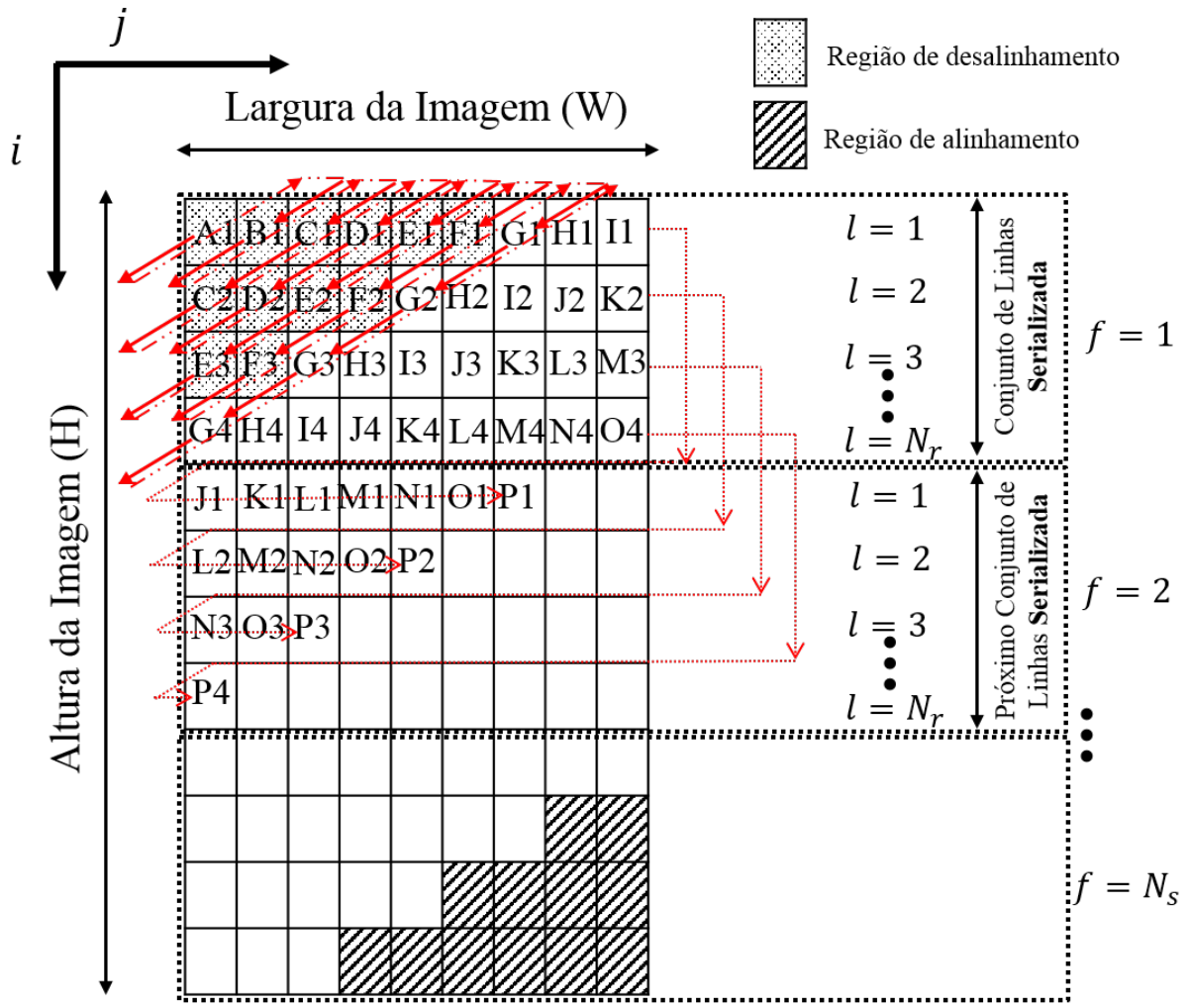
O processamento do custo em um pixel de uma determinada linha da imagem acontece quando uma das FIFOS Onchip despacha um pixel para a FIFO LUT, e a partir desta FIFO os dados são ajustados e enviados para o módulo AD.

O despacho dos pixels de cada linha a partir da FIFO Onchip para a FIFO LUT é controlado pelo módulo *Controlador de Leitura da FIFO* e a ordem de despacho segue o fluxo mostrado na Figura 49.

De acordo com este fluxo, o módulo divide a imagem em N_s conjuntos de linhas dado por $N_s = H/N_r$, onde H é altura da imagem e N_r é o número de linhas que serão processadas serialmente). Em cada conjunto de linhas o processamento acontece serialmente em diagonal, e em cada posição N_d, valores de disparidades são processados. Como pode ser observado, o despacho dos pixels das linhas seguintes, dentro do mesmo conjunto f de linhas a serem processadas, são separados por dois pixels horizontais em relação ao pixel da linha anterior. Por exemplo, entre o processamento de C1 e C2, os pixels A1 e B1

já foram processados. Isto é feito para resolver a dependência de custo, disponibilizando com antecedência os custos de caminhos quando o sistema precisar calcular os custos das linhas seguintes.

Figura 49 – Ordem de processamento dos custos de caminho dos pixels na imagem

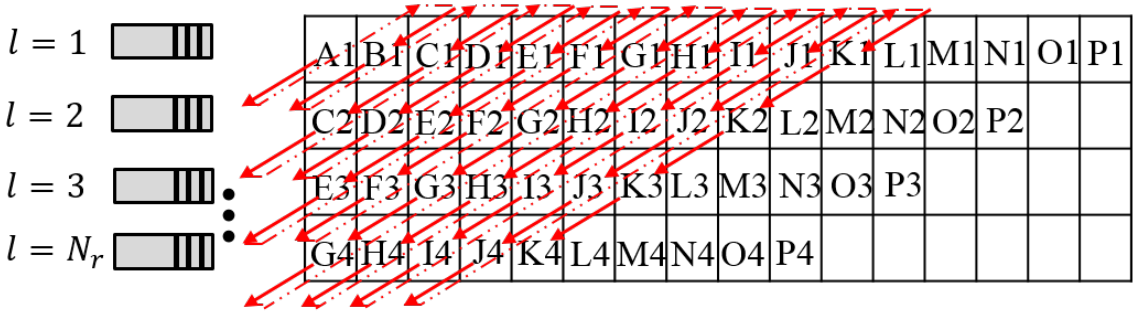


Este espaçamento entre linhas dá a conotação em diagonal ao fluxo de processamento dos custos de caminho. Assim, o módulo *Controlador de Leitura da FIFO* primeiro despacha três pixels da primeira linha (A1, B1 e C1) para depois despachar o primeiro pixel da segunda linha C2, Em seguida, despacha três pixels da segunda linha para poder despachar o primeiro da terceira linha E3 e assim por diante. Este processo é realizado para todas as linhas dentro do mesmo conjunto de linhas e é chamado de etapa de *desalinhamento dos pixels*. Nesta etapa este módulo, espera o tempo necessário para resolver as dependências quando for processar o próximo pixel da mesma linha da imagem. Este tempo é, na verdade, a quantidade de ciclos de relógio para processar e armazenar um valor do custo no módulo de custo de caminho. Por exemplo, o despacho do pixel B1 só pode acontecer depois que o processamento de A1 já tenha sido finalizado e o resultado

armazenado. Em relação aos pixels cujas dependências já tenham sido resolvidas, estes já podem ser despachados para processamento. Por exemplo, ao despachar o pixel C1, no próximo ciclo, o pixel C2 já pode ser despachado pois os pixels A1 e B1 já foram calculados. No entanto, D1 não pode ser despachado porque o pixel C1 ainda não foi finalizado.

Depois de estabelecidos os espaçamentos entre todas as linhas, então o módulo *Controlador de Leitura da FIFO* despacha para processamento do algoritmo SGM os pixels de todas as linhas em diagonal ao longo de toda a imagem, passando por todos os conjuntos $f = 1, 2, ...N_s$. Na Figura 49 o processamento normal acontece a partir da diagonal G1 -> G2 -> G3 -> G4. Ao longo de toda a imagem, quando uma determinada linha do conjunto é finalizada, o processamento da próxima diagonal começa na próxima linha do próximo conjunto e em seguida volta para o conjunto anterior para finalizar o restante das linhas. Por exemplo ao terminar a diagonal I1 -> I2 -> I3 -> I4 a próxima diagonal será J1 -> J2 -> J3 -> J4, com J1 localizado no próximo conjunto de linhas $f = 2$, mas J2, J3 e J4 ainda são do conjunto de linhas anterior, $f = 1$, como pode ser visto na Figura 49. De maneira fluida, o processamento passa de um conjunto de linhas para o outro. Essa mudança de conjunto é transparente para o módulo *Controlador de Leitura da FIFO* como mostrado na Figura 49 pois os pixels das linhas referente aos próximos conjuntos já estão disponíveis nas FIFOS Onchip correspondentes. Por exemplo, na sequência de acesso J1 -> J2 -> J3 -> J4, J1 já está na primeira FIFO embora este pixel seja do próximo conjunto de linhas da imagem.

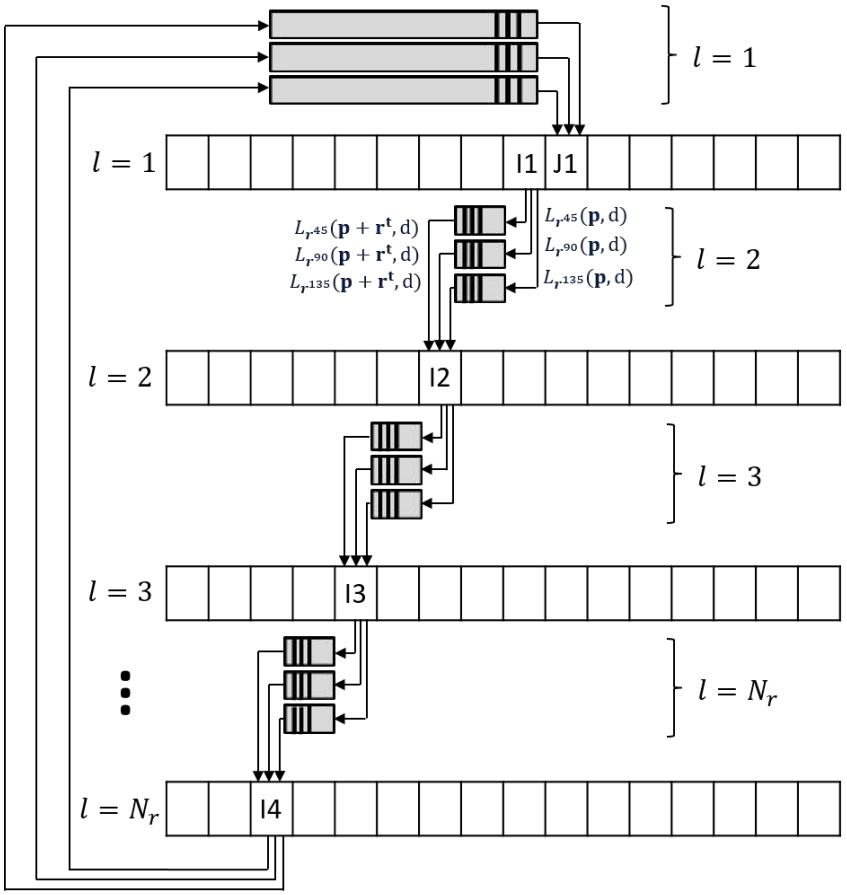
Figura 50 – Ordem de processamento dos custos de caminho dos pixels na imagem do ponto de vista do módulo



É importante reiterar que, tão logo os pixels a partir das câmeras já estejam disponíveis, o processamento do custo de caminho já pode ser realizado. O módulo que verifica esta disponibilidade é o *Controle de Leitura da FIFO*, que só despacha o pixel a partir da FIFO Onchip para FIFO Lut se existir algum pixel disponível na FIFO Onchip, caso contrário, este módulo espera até que o pixel esteja disponível. A disponibilidade é verificada através da flag de vazio da FIFO Onchip, que quando existe algum pixel armazenado na FIFO, esta flag tem valor zero. Por exemplo, quando o módulo precisar processar o pixel

C2 certamente este pixel ainda não estará disponível pois a câmera ainda está despachando os pixels da primeira linha. Assim o módulo espera até que este pixel seja inserido na FIFO Onchip e a flag fique zero para poder despachá-lo.

Figura 51 – Comunicação entre pixels de linhas diferentes na computação do custo de caminho de cada linha



Outro ponto importante, que será também abordado na arquitetura do módulo de custo de caminho, é a comunicação entre os pixels de linhas vizinhas para a computação do custo de caminho. Na arquitetura proposta para computação do custo de caminho, esta comunicação é realizada por meio de FIFOs que armazenam os custos de caminho de cada direção separadamente e que serão utilizados para computação do custo de caminho da linha seguinte como mostrado na Figura 51. Ao garantir as condições para armazenamento do custo de caminho na FIFO que leva em consideração as condições de borda de cada direção, os custos de caminho estarão na ordem correta para acesso na linha seguinte.

Dois pontos importantes sobre as FIFOs para comunicação entre linhas que precisam ser observados. O primeiro ponto é a necessidade de armazenar o custo de caminho de todos os pixels da última linha de um dado conjunto para o início do processamento do próximo conjunto de linhas seriais. Isto é devido ao processamento do próximo conjunto acontecer apenas depois que o processamento do conjunto anterior tenha sido finalizado. O segundo ponto é sobre o tamanho da FIFO para armazenamento do custo entre linhas.

Como o processamento de todas as linhas acontece com o espaçamento de três pixels entre linhas, os resultados de custo que foram armazenados na FIFO serão consumidos em no máximo depois de três pixels de processamento, o que exige espaços para armazenar o resultado de três custos de caminho nas FIFOs intermediárias. Questões de tamanho de FIFO serão detalhados na seção seguinte.

4.2.2.2 Arquitetura em pipeline do Módulo SGM

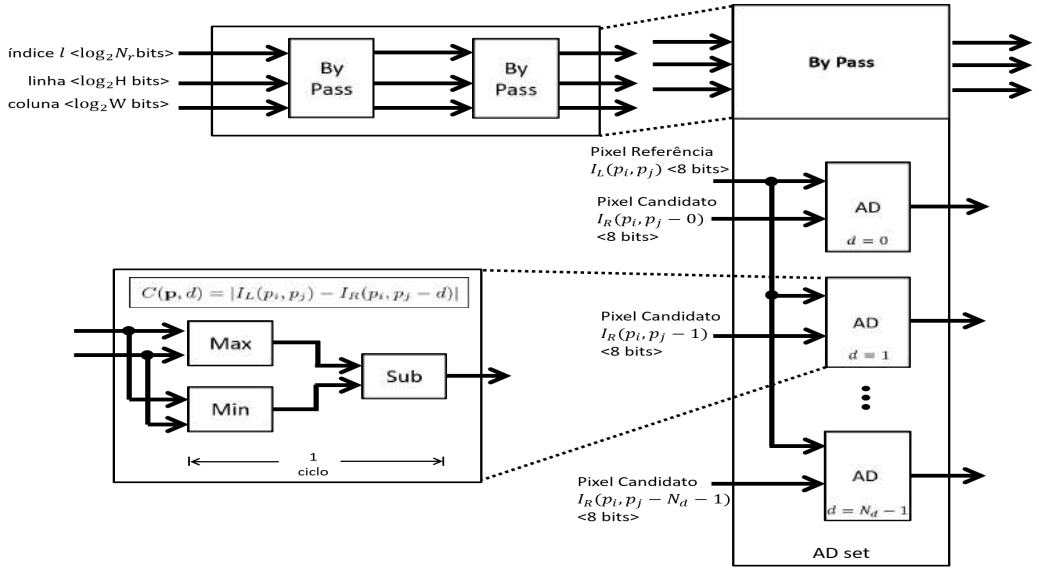
Após os pixels serem despachados a partir das FIFOs LUT, estes são utilizados para a computação da técnica SGM (Equação 4.3) através de vários módulos implementados como estágios independentes. Cada estágio processa todos as N_d disparidades ao mesmo tempo e passa o seu resultado para o próximo estágio.

Os módulos envolvidos na computação da técnica SGM são aqueles utilizados para o cálculo de AD, os quatro módulos de custo de caminho, o módulo de agregação, o módulo de seleção de disparidade e o módulo de checagem LR, como mostrado na Figura 48. Cada módulo sinaliza o seu término e o seu resultado é passado para o próximo módulo para ser processado. A seguir são detalhadas as arquiteturas dos módulos mencionados.

4.2.2.2.1 Módulo de Diferença Absoluta

O módulo de diferença absoluta (AD) recebe um vetor de N_d pixels candidatos e um pixel de referência com largura de 8 bits. Após este recebimento, são computados os N_d valores de diferença absoluta entre o pixel referência e todos os pixels candidatos ao mesmo tempo em um único ciclo de relógio, como mostrado na Figura 52. Este módulo devolve um vetor de N_d custos iniciais, onde cada custo é representado por 8 bits.

Figura 52 – Módulo de computação do custo inicial através de diferenças absoluta



4.2.2.2.2 Módulo de custo de caminho

Entre estes módulos, o mais complexo e o que demandou maior esforço de otimização é o módulo de custo de caminho. Este módulo processa o custo de caminho através de sete módulos que trabalham em pipeline como pode ser visto na Figura 53: módulo *Leitura* do custo anterior, *Deslocamento a esquerda e a direita*, *Adição de Penalidades* P_1 *e* P_2, *Computação do mínimo da Equação*, *Computação Final*, *Computação do mínimo do resultado* e por fim, *Escrita* do custo de caminho atual. Este módulo de custo de caminho é replicado quatro vezes para computar os custos de caminho das quatro direções, no qual para cada módulo o tamanho das FIFOs intermediárias e também as condições de leitura e escrita nestas FIFOs são diferentes. Todos os outros módulos para as demais direções são similares.

No caso do tamanho das FIFOs, FIFOs de tamanhos diferentes são usadas para armazenar custos entre linhas e o custo entre conjuntos de linhas seriais. O armazenamento de dados entre linhas permite a comunicação entre linhas e o armazenamento entre conjuntos permite a comunicação entre conjunto de linhas seriais. No caso da comunicação entre conjuntos, a FIFO que será lida pela primeira linha $l = 1$ de um dado conjunto de linhas de índice $f = a$ armazena uma linha inteira da imagem que é o resultado do custo de caminho e o mínimo da última linha $l = N_r$ do conjunto de linhas seriais anterior $f = a - 1$. No caso do armazenamento entre linhas, as FIFOs são menores e armazenam para o custo de caminho $t = 0^o$ e $t = 135^o$ o resultado de apenas um pixel, para $t = 90^o$, o resultado de dois pixels a frente e para $t = 45^o$, o resultado de três pixels. Este tamanho de FIFOs são determinados através do parâmetro *Direcão* que é a direção de propagação da função de custo de caminho $L_{r^t}(\mathbf{p}, d)$, ou seja, $\mathbf{r}^{0^\circ}$, $\mathbf{r}^{45^\circ}$, $\mathbf{r}^{90^\circ}$ ou $\mathbf{r}^{135^\circ}$.

No caso da condição de leitura, o módulo *Leitura (Read)* sinaliza quando deve-se ler o custo de caminho e o mínimo do pixel anterior que está no topo de uma determinada FIFO de índice l. Importante lembrar que ao solicitar uma leitura na FIFO, o dado que está no topo é descartado. A condição de leitura no módulo de *Leitura (Read)* leva em conta as condições de borda de cada direção do custo de caminho. Como mostrado na Figura 54, para a direção $\mathbf{r}^{0^\circ}$, o módulo sinaliza a leitura se a posição da coluna for diferente de zero; para a direção $\mathbf{r}^{45^\circ}$, o módulo sinaliza a leitura se as posições de linha e coluna forem diferentes de zero; para a direção $\mathbf{r}^{90^\circ}$, o módulo sinaliza a leitura se a posição da linha for diferente de zero; e para a direção $\mathbf{r}^{135^\circ}$, o módulo sinaliza a leitura se as posições de linha e coluna forem respectivamente diferente de zero e diferente da última posição. Este módulo também determina em qual FIFO devem ser lidos estes dados. Para todas estas direções a FIFO de índice l_f que vai ser lida é a mesma do índice l fornecido na entrada do módulo, como pode ser visto na Figura 53, ou seja, $l_f = l_r = l$. O índice l_r indica o índice da FIFO que é solicitado para leitura a partir do módulo *Leitura (Read)*.

Figura 53 – Arquitetura em pipeline do módulo de custo de caminho do algoritmo SGM

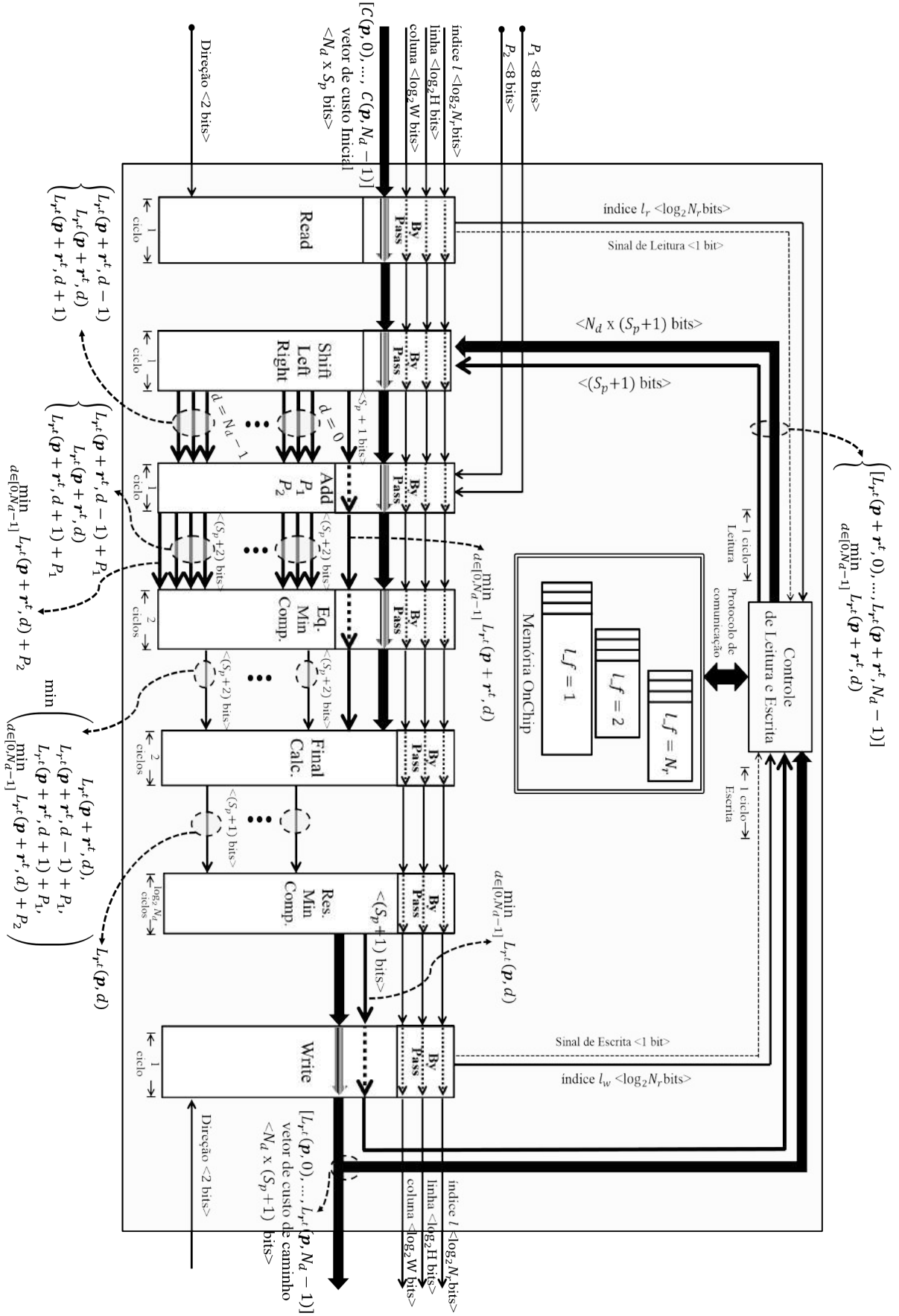


Figura 54 – Lógica para sinalizar a leitura na FIFO para diferentes direções de custo de caminho: (a) $\mathbf{r}^{0^o}$, (b) $\mathbf{r}^{45^o}$, (c) $\mathbf{r}^{90^o}$ e (d) $\mathbf{r}^{135^o}$

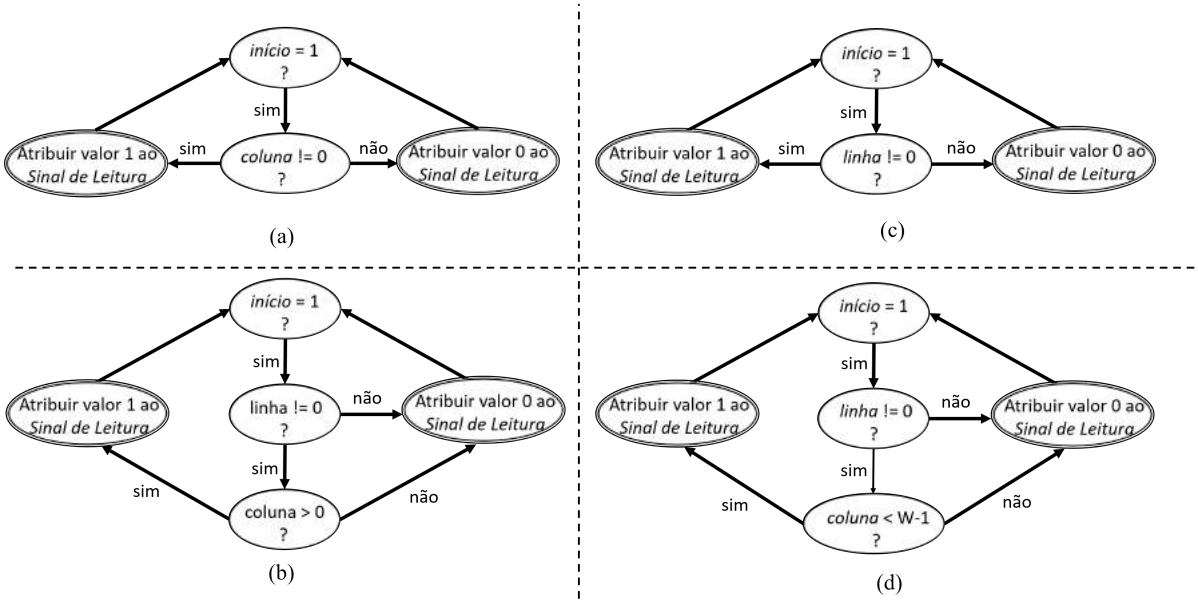
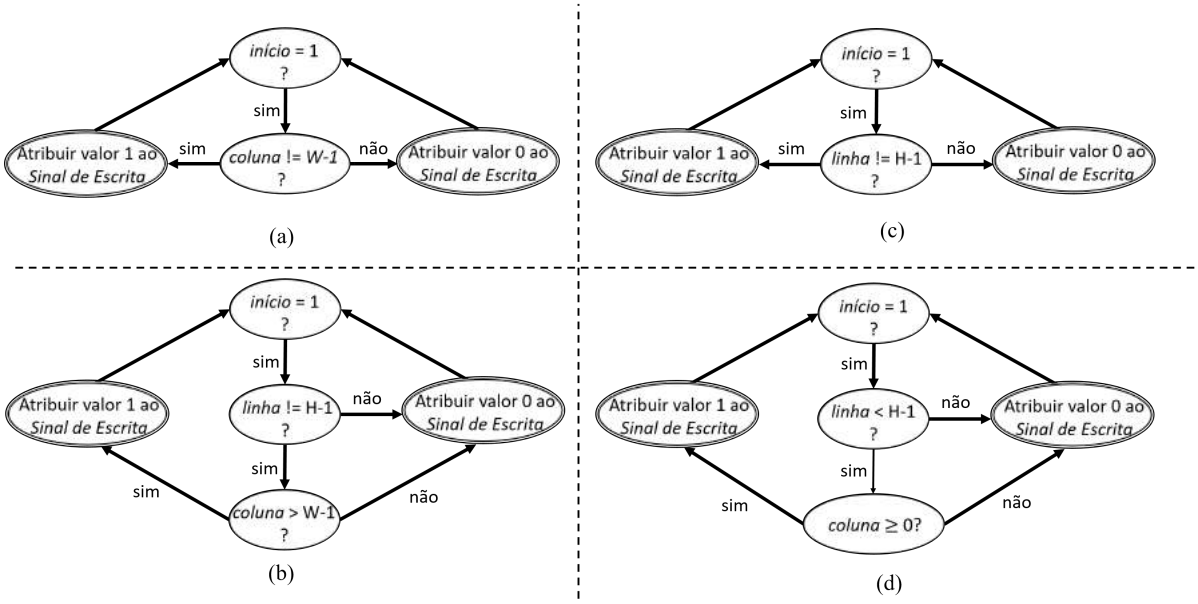


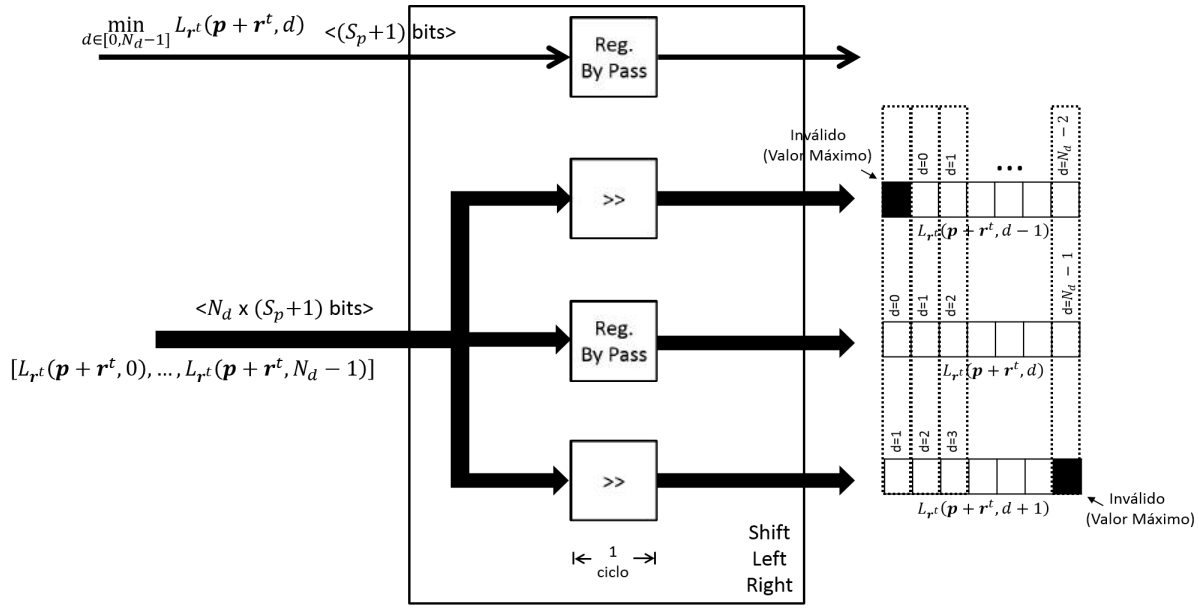
Figura 55 – Lógica para sinalizar a escrita na FIFO para diferentes direções de custo de caminho: (a) $\mathbf{r}^{0^o}$, (b) $\mathbf{r}^{45^o}$, (c) $\mathbf{r}^{90^o}$ e (d) $\mathbf{r}^{135^o}$



No caso da condição de escrita, o módulo *Escrita (Write)* sinaliza quando deve-se escrever o custo de caminho e o mínimo do pixel atual em uma determinada FIFO de índice l. É importante lembrar que ao solicitar uma escrita na FIFO o novo dado será armazenado no final da FIFO. A condição de escrita a partir do módulo de *Escrita (Write)* leva em conta se estes dados serão lidos posteriormente, senão, estes não são armazenados. Como mostrado na Figura 55, para a direção $\mathbf{r}^{0^o}$, o módulo sinaliza a escrita se a posição da coluna for diferente da última posição da coluna na imagem; para a direção $\mathbf{r}^{45^o}$, o módulo sinaliza a escrita se as posições de linha e coluna forem respectivamente diferentes

da última posição da linha e da última posição da coluna na imagem; para a direção $\mathbf{r}^{90^o}$, o módulo sinaliza a escrita se a posição da linha for diferente da última posição da linha na imagem; e para a direção $\mathbf{r}^{135^o}$, o módulo sinaliza a escrita se as posições de linha e coluna requisitada para escrita forem respectivamente diferentes da última linha e diferente da primeira coluna na imagem.

Figura 56 – Operação de deslocamento para obter as componentes $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d-1)$ e $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d+1)$



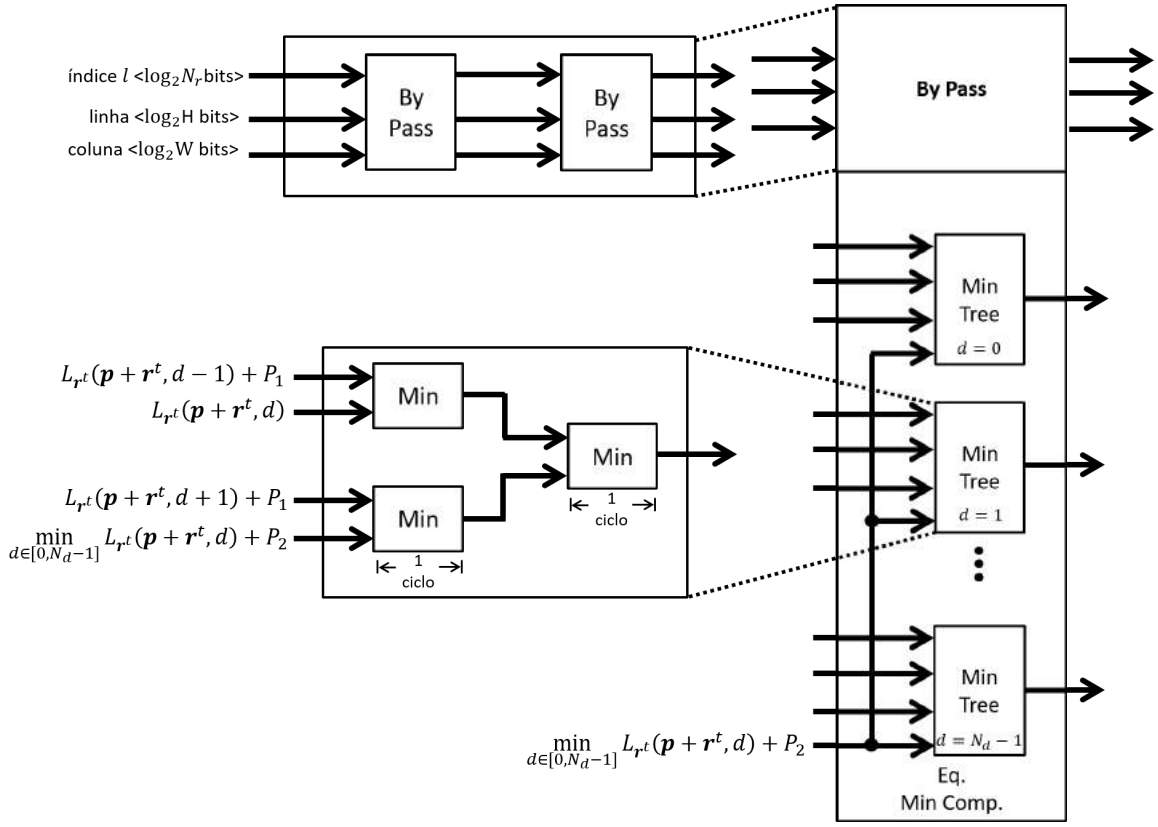
O módulo *Escrita (Write)* também determina em qual FIFO devem ser escritos os resultados. Para as direções $\mathbf{r}^{45^o}$, $\mathbf{r}^{90^o}$, $\mathbf{r}^{135^o}$ a FIFO de índice l_f que será escrita é sempre a próxima em relação ao índice l fornecido na entrada do módulo, ou seja, se l for diferente de N_r então $l_f = l_w = l+1$, senão $l_f = l_w = 1$. Para a direção $\mathbf{r}^{0^o}$ a FIFO que vai ser escrita é a mesma do índice l fornecido na entrada do módulo, ou seja, $l_f = l_w = l$. O índice l_w indica o índice da FIFO que é solicitado para escrita a partir do módulo *Escrita (Write)*.

O módulo de *Controle de Leitura e Escrita* gerencia um conjunto de N_r FIFOs através de um único bloco de memória RAM da FPGA (Onchip). Uma faixa de endereço é alocado para cada FIFO e dentro de cada faixa o módulo mantém os endereços da posição de leitura e escrita de cada FIFO. Quando é solicitada uma escrita em uma determinada FIFO, o dado é escrito na posição de escrita desta FIFO e a posição é incrementada de maneira circular , ou seja, se a posição chegar no final da faixa, ele vai para o início. A leitura segue o mesmo princípio, quando é solicitada uma leitura em uma determinada FIFO, o dado que está na posição de leitura desta FIFO libera o dado e esta posição é incrementada de maneira circular. O uso de um único bloco permite que o espaço de memória da FPGA seja melhor aproveitado. Se fosse alocado separadamente um bloco

de memória para cada FIFO de cada linha da imagem iria-se ter desperdícios de espaço, uma vez que a quantidade de dados intermediários é muito menor que o tamanho de um bloco de memória e, uma vez alocado um bloco, o espaço não usado não é realocado.

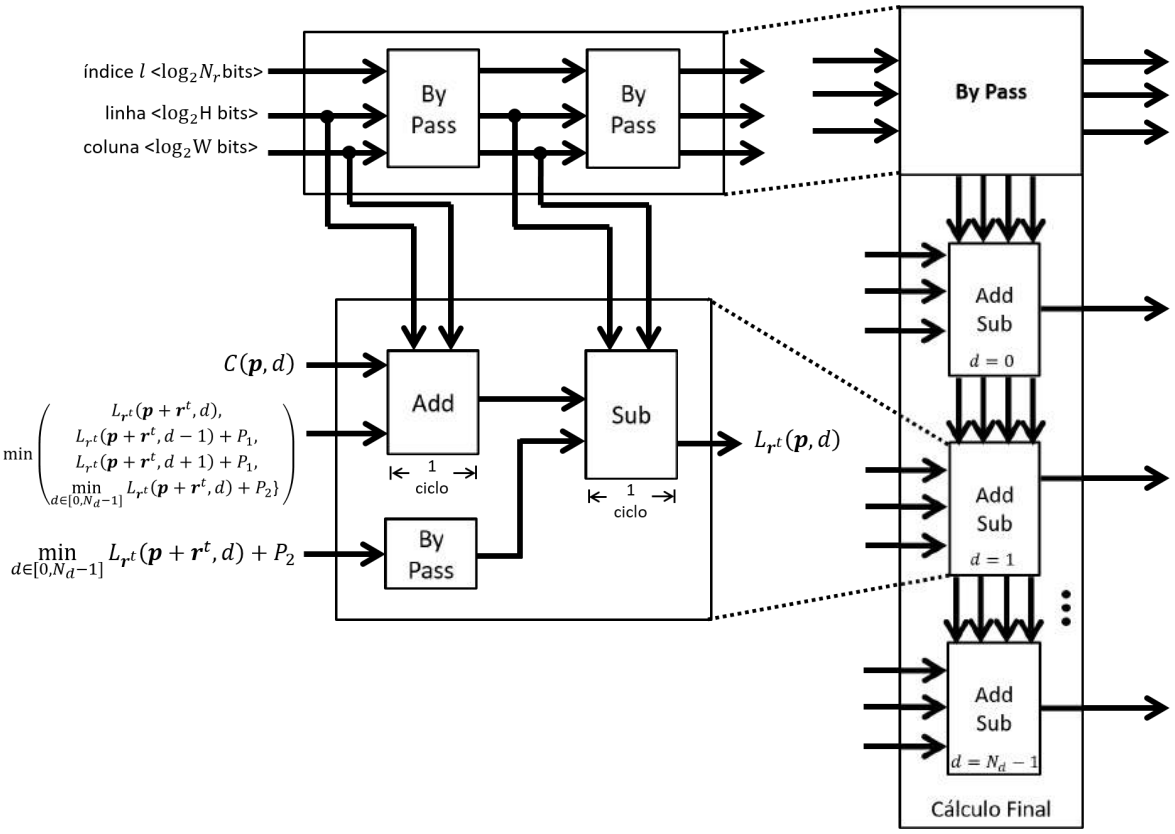
Após solicitada a leitura do vetor de custos de caminho do pixel anterior na direção $\mathbf{r}^t$ e o seu custo de caminho mínimo, estes dados estarão disponíveis para o módulo *Deslocamento Esquerda e Direita (Shift Left Right)* no próximo ciclo de relógio. Este módulo define dois vetores de N_d valores a partir do vetor de custos de caminho de entrada para representar os custos $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d-1)$ e $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d+1)$. Estes novos vetores vêm a partir de operações de deslocamento para a direita e para a esquerda como mostrado na Figura 56. Dessa forma, para cada posição de disparidade tem-se os três termos necessários para a computação do algoritmo SGM: $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d)$, $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d-1)$ e $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d+1)$.

Figura 57 – Conjunto de árvore de comparadores para obter o valor de mínimo para cada disparidade



Estes vetores juntamente com o mínimo são enviados para o módulo *Adição de Penalidades* que adiciona as penalidades P_1 para $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d-1)$ e $L_{r^t}(\mathbf{p}+\mathbf{r}^t, d+1)$ e P_2 para o custo de caminho mínimo $\min_{k\in[0,N_d-1]} L_{\mathbf{r}^t}(\mathbf{p}+\mathbf{r}^t, k)$ do pixel anterior. Para assegurar que a adição de penalidade não ultrapasse o máximo valor representável pelo vetor de bits (em outras palavras, para evitar overflow) é adicionado mais um bit na largura de representação do custo na resposta deste módulo.

Figura 58 – Conjunto de somadores e subtratores para calcular o custo de caminho final para cada disparidade



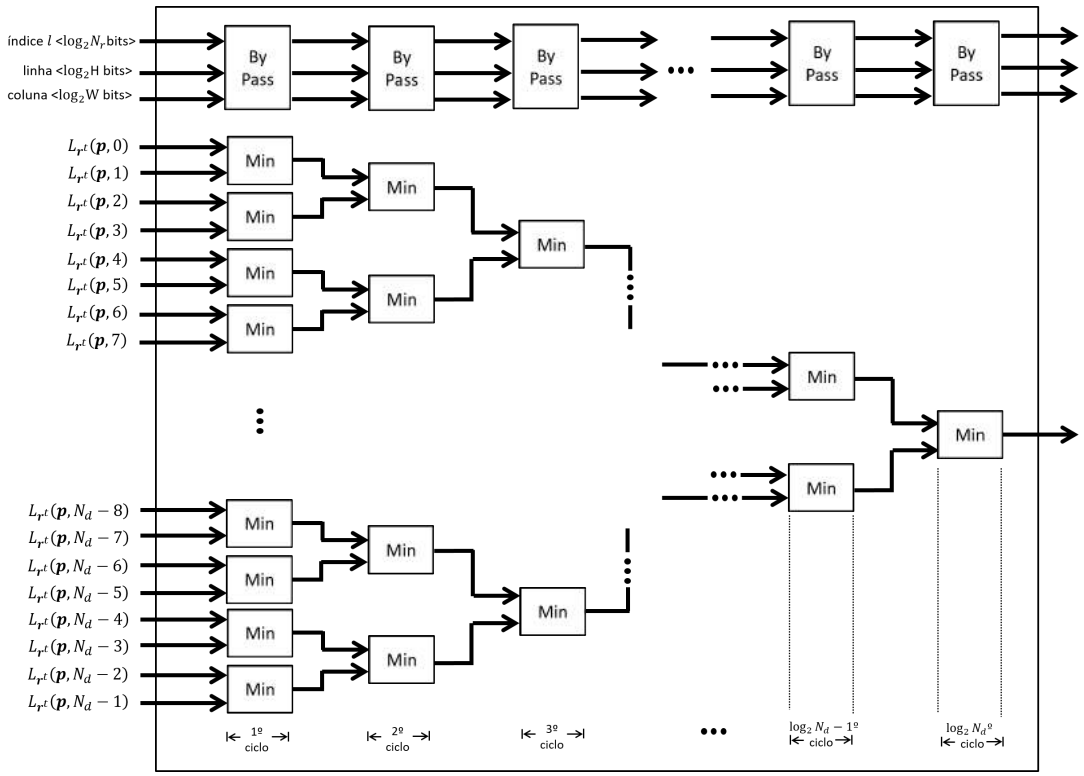
Depois do módulo de *Adição de Penalidades* os quatro valores que compõem o termo de mínimo da Equação 4.3 são passados para o módulo *comparador de mínimo*. Este módulo obtém o mínimo de cada disparidade através de uma árvore de comparadores de dois estágios como mostrado na Figura 57. Para cada disparidade existe uma árvore de comparadores e todas as árvores calculam o mínimo de cada disparidade paralelamente.

Assim, os valores de custo inicial, e os valores de mínimo da Equação 4.3 já estão disponíveis no módulo *Cálculo Final* que calculará o resultado final do custo de caminho $L_{r^t}(\mathbf{p}, d)$ para todas as disparidades. Este módulo primeiramente soma o custo inicial com o termo de mínimo obtido do módulo *comparador de mínimo* e em seguida subtrai o resultado dessa soma do valor de custo de caminho mínimo que foi obtido a partir do módulo de *leitura*, como mostrado na Figura 58. O módulo *Cálculo Final* também verifica, através das condições de leitura mostrado na Figura 54, se as operações de adição e subtração devem ser de fato realizadas ou simplesmente devem repassar o valor do custo inicial.

O resultado do custo de caminho final para todas as N_d disparidades do módulo *Cálculo Final* é passado para o módulo *comparador de mínimo do resultado* que retorna o valor do custo mínimo entre estes N_d valores. A comparação de todos estes valores é realizada por meio de uma árvore de comparadores como mostrado na Figura 59. Em cada estágio, são comparados todos os valores do estágio anterior dois a dois. O resultado

de todas as comparações são passados para o próximo estágio. Esta estrutura em árvore permite obter a máxima eficiência em frequência de operação, uma vez que a cada ciclo envolve operações simples de comparações envolvendo apenas dois valores. Além disso, este tipo de estrutura tem latência de $log_2 N_d$ ciclos de relógios para encontrar o custo de caminho mínimo e exige que a quantidade N_d de valores a serem comparados seja uma potência de 2. Quantidade de valores que seja potência de 2 permite construir árvores genéricas, que para computar outras quantidades, apenas adiciona-se e remove-se mais um estágio de computação de mínimo. Além disso, quantidades em potência de 2 permite definir árvores com a melhor eficiência em quantidade de operações por ciclo de relógio, ou seja, em cada ciclo todos os valores serão sempre processados. Se for necessário computar outras quantidade de valores que não sejam potência de 2 é necessário definir um tipo de árvore irregular que não computará todos os valores ao mesmo tempo em cada estágio.

Figura 59 – Árvore de comparadores para obter o mínimo entre N_d valores de custo

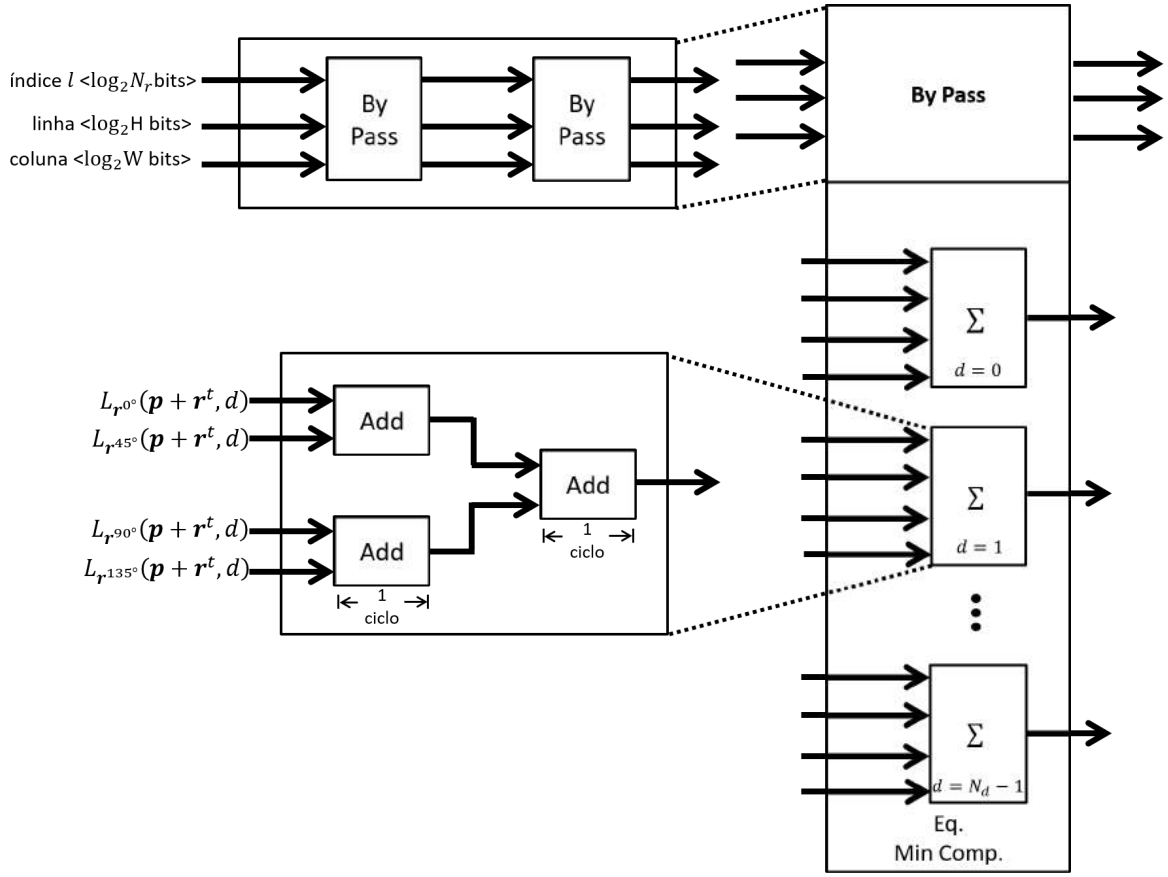


O resultado do vetor de custos de caminho final e o mínimo são passados para o módulo *Escrita* que define em qual FIFO serão escritos estes dados de acordo com as condições mostradas na Figura 55. O vetor de custo de caminho a partir deste módulo é de fato a saída do módulo inteiro de custo de caminho, juntamente com o índice l, posição de linha e coluna do pixel que foi requisitado para ser calculado. Estas informações, além do custo de caminho, serão utilizados pelos módulos a frente. A posição de linha e coluna é utilizado entre outras funções para montar a imagem completa do mapa de disparidade, uma vez que os dados não são fornecidos em um padrão regular.

4.2.2.2.3 Módulo de Agregação

Os custos de caminho dos quatro módulos são então somados através do módulo *Agregação* para compor o custo final otimizado. Para cada disparidade são somados quatro valores. Estes quatro valores são somados através de três somas dividido em dois estágios em pipeline, no qual, no primeiro estágio são somados em paralelo os valores $L_{\mathbf{r}^{0°}}(\mathbf{p}, d)$ e $L_{\mathbf{r}^{45°}}(\mathbf{p}, d)$ e os valores $L_{\mathbf{r}^{90°}}(\mathbf{p}, d)$ e $L_{\mathbf{r}^{135°}}(\mathbf{p}, d)$. Os resultados destas duas somas são novamente somados no próximo estágio como mostrado na Figura 60 resultando em $S(\mathbf{p}, d)$. Esta estrutura é análoga a uma árvore de somadores de dois estágios. Este processamento é realizado para todas as disparidades em paralelo e no final o módulo devolve um vetor de N_d custos otimizados, onde cada custo é representado por um vetor de $S_p + 3$ bits de largura.

Figura 60 – Módulo de agregação dos custos de caminho



4.2.2.2.4 Módulo de Seleção de Disparidade [Esquerda]

O vetor $S(\mathbf{p}, d)$ com os N_d custos otimizados é enviado para o módulo denominado *seleção disparidade [esquerda]* que busca o menor custo e retorna a disparidade $D_L(\mathbf{p})$ associada com esse custo. O valor de disparidade tem largura de representação de $log_2 N_d$

bits. Este módulo utiliza a mesma estrutura de comparação utilizada pelo módulo *computação de mínimo do resultado* mostrado na Figura 59, com a diferença de que, além do menor custo, o índice de disparidade associado ao menor custo é propagado até a saída do módulo.

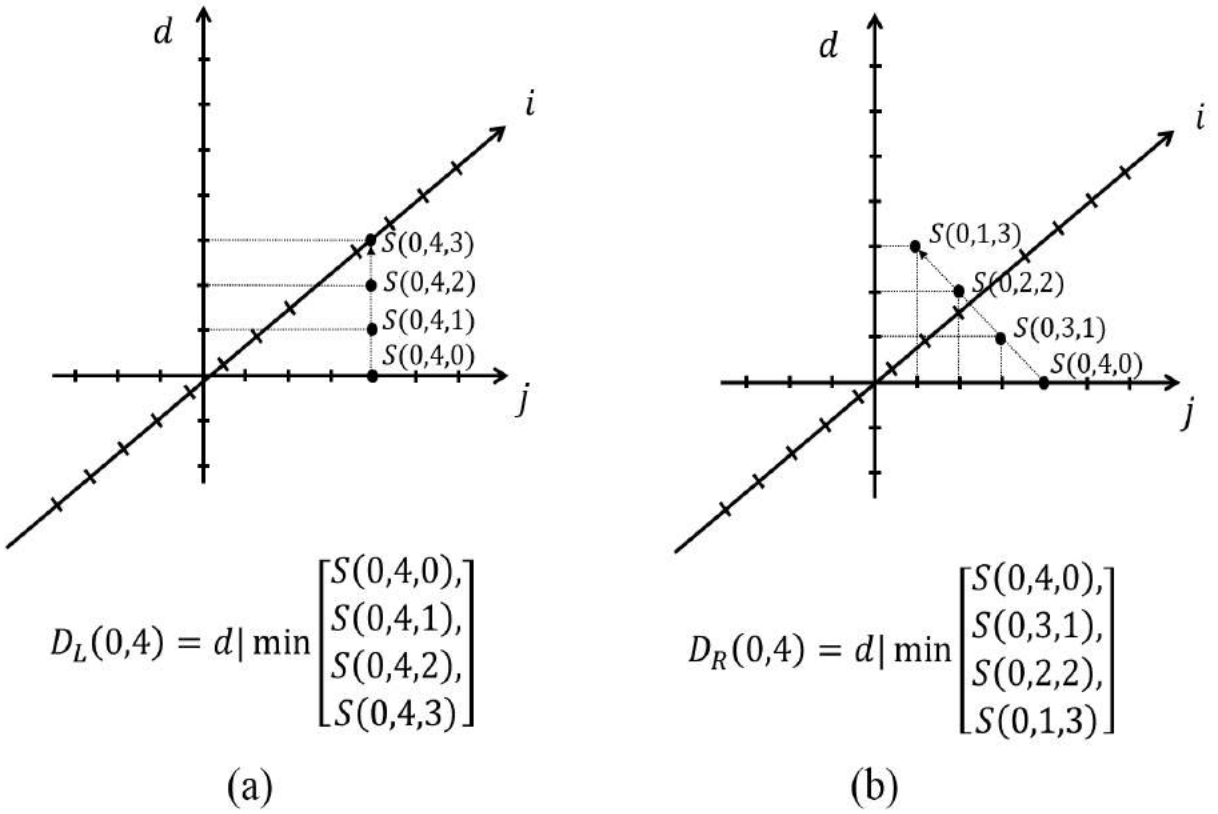
4.2.2.2.5 Módulo de Seleção de Disparidade [Direita]

Este módulo também recebe o vetor $S(\mathbf{p}, d)$ com os N_d custos otimizados e define uma disparidade $D_R(p_i, p_j)$ de custo mínimo. A busca que este módulo realiza é diferente da busca que é realizada pelo módulo *Seleção de Disparidade [Esquerda]*. Esta diferença é melhor entendida através da Figura 61 que foi extraída da Seção 2.3. A Figura 61 exemplifica o processo de determinação da disparidade $D_L(0,4)$ e $D_R(0,4)$, com um intervalo de disparidades N_d igual a quatro. Observe que para calcular D_L a busca acontece variando apenas a posição das disparidades e mantendo as posições $(p_i, p_j) = (0,4)$ constantes, enquanto que para calcular D_R a busca acontece variando as posições de disparidade d, e de coluna $p_j - d$, mantendo constante a posição de linha p_i. Assim, generalizando, para calcular $D_L(p_i, p_j)$ é buscado o valor mínimo entre os valores $S(p_i, p_j, 0), S(p_i, p_j, 1), ..., S(p_i, p_j, N_d - 1)$, enquanto que, para calcular $D_R(p_i, p_j)$ é buscado o valor mínimo entre os valores $S(p_i, p_j - 0, 0), S(p_i, p_j - 1, 1), ..., S(p_i, p_j - N_d - 1, N_d - 1)$. Esta abordagem para determinação de D_R para detecção de oclusão é chamada de *checagem LR rápida*. É possível observar na computação do vetor $D_R(p_i, p_j)$ que existe a necessidade de se conhecer os custos de pixels anteriores para computar a disparidade associada do pixel (p_i, p_j). Esta é a grande dificuldade de definir uma arquitetura que execute este tipo de operação de forma eficiente, pois os custos de pixels anteriores já foram processados e descartados pelo pipeline do módulo. É importante ter em mente que o processamento de cada pixel na arquitetura proposta é realizado para todos os N_d disparidades em um único ciclo de relógio, ou seja, no próximo ciclo o módulo já estará processando o próximo pixel e assim por diante.

A ideia inovadora que permitiu desenvolver este módulo de forma eficiente foi considerar o uso antecipado dos custos dos pixels anteriores no cálculo do mínimo dos pixels que seria feito mais a frente. Por exemplo, para o pixel de posição $(0,1)$ são necessários os custos $S(0,0,1)$ e $S(0,1,0)$, no qual $S(0,0,1)$ já foi fornecido durante o cálculo do pixel anterior $(0,0)$. Outro exemplo, para o pixel de posição $(0,2)$ são necessários os custos $S(0,0,2)$, $S(0,1,1)$ e $S(0,2,0)$, no qual o custo $S(0,0,2)$ já foi fornecido na computação de dois pixels (ou dois ciclos) anteriores e $S(0,1,1)$ já foi fornecido na computação de um pixel atrás. Mais um outro exemplo, para o pixel de posição $(0,3)$ são necessários os custos $S(0,0,3)$, $S(0,1,2)$, $S(0,2,1)$ e $S(0,3,0)$, no qual o custo $S(0,0,3)$ já foi fornecido na computação de três pixels anteriores e $S(0,1,2)$ já foi fornecido na computação de dois pixels atrás e $S(0,2,1)$ já foi fornecido na computação de um pixel anterior. Outra importante observação é que na medida em que a disparidade aumenta, mais valores de

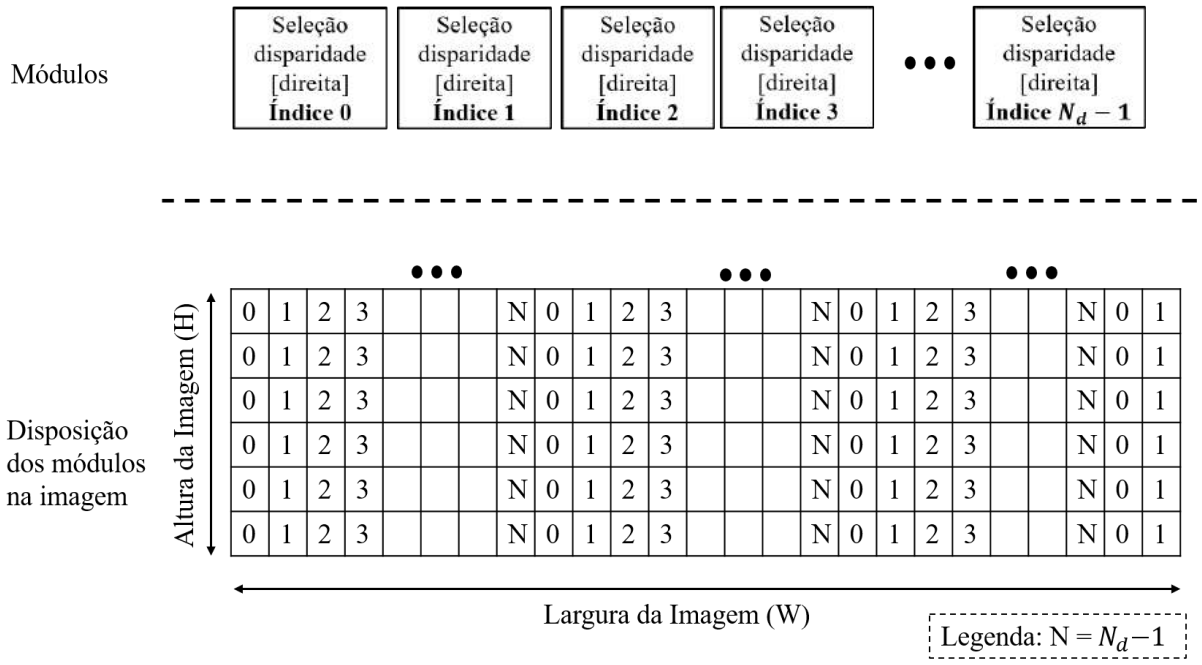
custos são necessários em relação a pixels mais distantes do pixel que está sendo calculado. No último exemplo, para a buscar o custo de disparidade 2, o custo associado a esta disparidade está no pixel (0,1); para a buscar o custo de disparidade 3 do pixel (0,3), o custo associado a esta disparidade está no pixel (0,0) e assim por diante.

Figura 61 – Sentido da busca pela disparidade de menor custo na geração dos mapas de disparidades (a) D_L e (b) D_R



Dessa forma, foram incluídos N_d módulos chamados de *Unidade de Disparidade da Direita*, os quais são responsáveis por lidar com os N_d pixels de maneira independente. Estes módulos têm acesso, ao mesmo tempo, a todos os N_d custos finais dado por $S(p_i, p_j, d)$ de cada pixel (p_i, p_j) na medida em que estes são despachados a partir do módulo de *agregação*. Para cada módulo é atribuído um índice diferente dentro do intervalo de $[0, N_d - 1]$. Este índice determina quais os pixels que cada módulo ficará responsável por determinar a disparidade de custo mínimo. Um módulo de índice i calculará os pixels cuja posição na horizontal são i, $i + N_d$, $i + 2N_d$ e assim por diante sem ultrapassar o limite da largura da imagem, como mostrado na Figura 62 (na figura o termo N é igual N_d). Por exemplo, para um intervalo de 6 valores de disparidades, a unidade de índice 1, mostrado na Figura 62, processa os pixels de todas as linhas cuja posição na coluna j da imagem é $j = (1 + 0 \cdot 5) = 1$, $j = (1 + 1 \cdot 5) = 1 + 5$, $j = (1 + 2 \cdot 5) = 1 + 10$, $(1 + 3 \cdot 5) = 1 + 15$ e assim por diante.

Figura 62 – Conjunto de N_d módulos e quais os pixels cada módulo ficou responsável por calcular o mínimo



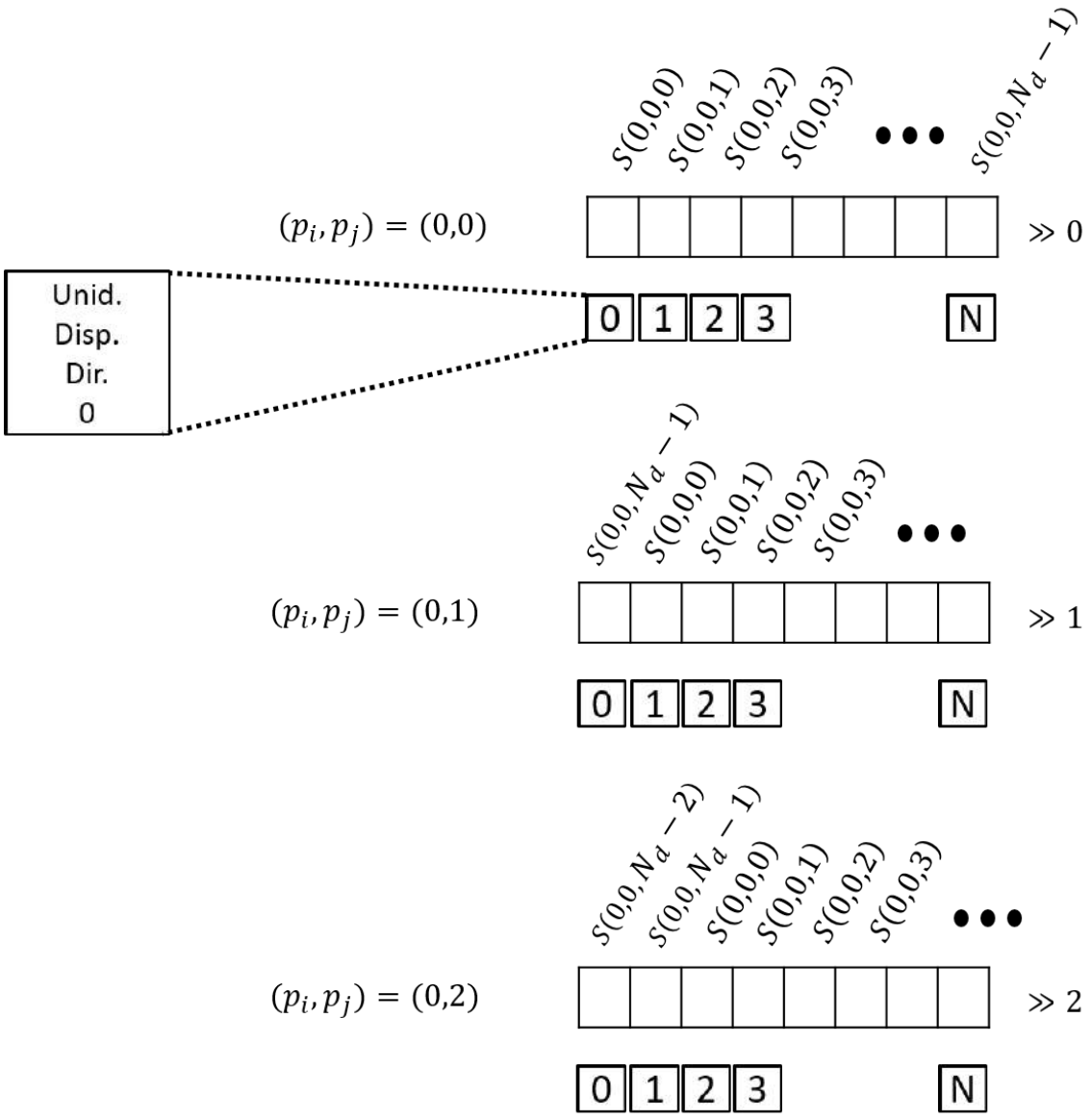
Além de determinar os pixels que cada módulo ficará responsável, o índice também determina as posições no vetor de custo, de acordo com a distância horizontal do pixel no qual este módulo está atualmente calculando o mínimo, e o pixel atual, cujo custo foi despachado pelo módulo de agregação. Assim, o módulo, cujo pixel de posição j_d, que está sendo calculado, acessará a posição $|j_d - j|$ do vetor de custo do pixel de posição horizontal j que veio a partir do módulo de agregação.

O custo que foi acessado no vetor na posição $|j_d - j|$ é verificado se é menor do que o custo atual armazenado no módulo. Se sim, o custo armazenado é atualizado com o custo que foi acessado e a disparidade atual recebe a nova disparidade $|j_d - j|$. Quando $j_d = j$ então o cálculo para o pixel j_d foi finalizado, o resultado do mínimo foi despachado e o módulo responsável foi realocado para calcular o mínimo do próximo pixel que no caso seria $j_d + N_d$.

Para um melhor entendimento vamos detalhar este processo através de um exemplo. Quando o módulo de agregação libera o vetor de custo final do pixel $(i, j) = (0, 0)$, os módulos, cujos pixels estão sendo calculados estão na posição horizontal 0, 1, 2, 3,...N_d−1 respectivamente, acessarão os custos $S(0, 0, 0)$, $S(0, 0, 1)$, $S(0, 0, 2)$, $S(0, 0, 3)$,...,$S(0, 0, N_d - 1)$. Uma vez que o módulo de índice 0 cujo pixel está na posição $j_d = 0$ é igual a do pixel fornecido $j = 0$, então o pixel j_d foi finalizado e o módulo de índice 0 é realocado para computar o mínimo do pixel $0 + N_d$ onde os valores de mínimo temporários foram definidos para infinito. Continuando o exemplo, quando o módulo de agregação libera o custo do próximo pixel $(i, j) = (0, 1)$ os módulos de índice 0, 1, 2, 3,...$N_d - 1$, respectivamente,

acessam os custos $S(0, 1, N_d - 1)$, $S(0, 1, 0)$, $S(0, 1, 1)$, $S(0, 1, 2)$,...,$S(0, 1, N_d - 2)$. Uma vez que o módulo de índice 1, cujo pixel está na posição $j_d = 1$, é igual a do pixel fornecido $j = 1$, então o pixel j_d foi finalizado e o módulo de índice 1 é realocado para computar o mínimo do pixel $1 + N_d$ onde os valores de mínimo temporários foram definidos para infinito. Continuando com mais outro exemplo, quando o módulo de agregação libera o custo do próximo pixel $(i, j) = (0, 2)$, os módulos de índice 0, 1, 2, 3,...$N_d - 1$ respectivamente, acessam os custos $S(0, 2, N_d - 2)$, $S(0, 2, 0)$, $S(0, 2, 1)$, $S(0, 2, 2)$,...,$S(0, 2, N_d - 3)$. Uma vez que o módulo de índice 2 cujo pixel está na posição $j_d = 2$, é igual a do pixel fornecido $j = 2$ então o pixel j_d foi finalizado e o módulo de índice 2 é realocado para computar o mínimo do pixel $2 + N_d$ onde os valores de mínimo temporários foram definidos para infinito.

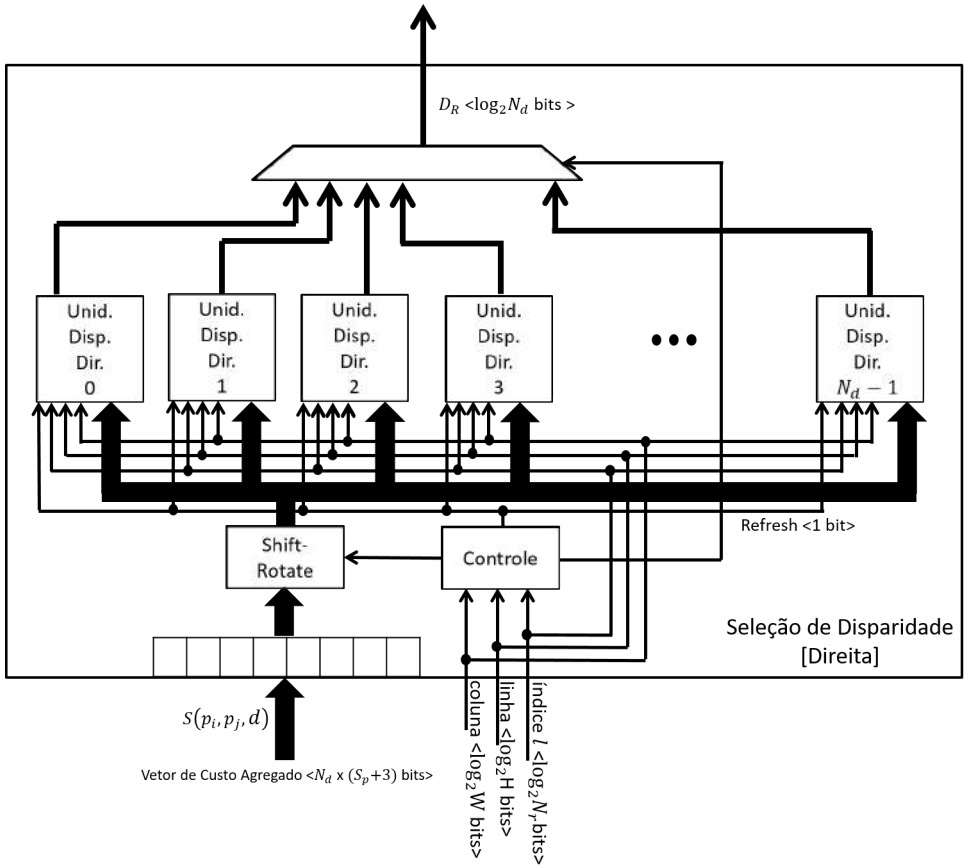
Figura 63 – Ilustração através de exemplos do processo de rotação para alocar as posições corretas do custo em cada módulo



É importante observar, através destes exemplos, que os índices de acesso ao vetor de entrada $S(p_i, p_j, d)$, para cada módulo, vão sendo decrementados à medida em que vão chegando os custos S dos próximos pixels. A implementação realizada desta forma, isto é, os índices para todos os módulos, consome muito espaço de processamento no FPGA devido à grande quantidade de operações de subtração e também de multiplexadores para escolher a posição no vetor como foi verificado em experimentos. Uma outra forma de implementar é manter os índices fixos de cada módulo e deslocar apenas o vetor de entrada de maneira circular. A Figura 63 mostra a aplicação de rotação nos três exemplos do parágrafo anterior.

Outro ponto importante: quando termina-se o processamento do mínimo de uma linha inteira, todos os módulos são reiniciados para poder começar a computar a próxima linha. Quem verifica e envia o sinal de término, ou *refresh* é o módulo de *Controle* como mostrado na Figura 64. Além disso, o módulo de controle define qual das unidades enviará o seu resultado de mínimo para a saída do módulo de *Seleção de Disparidade [Direita].*

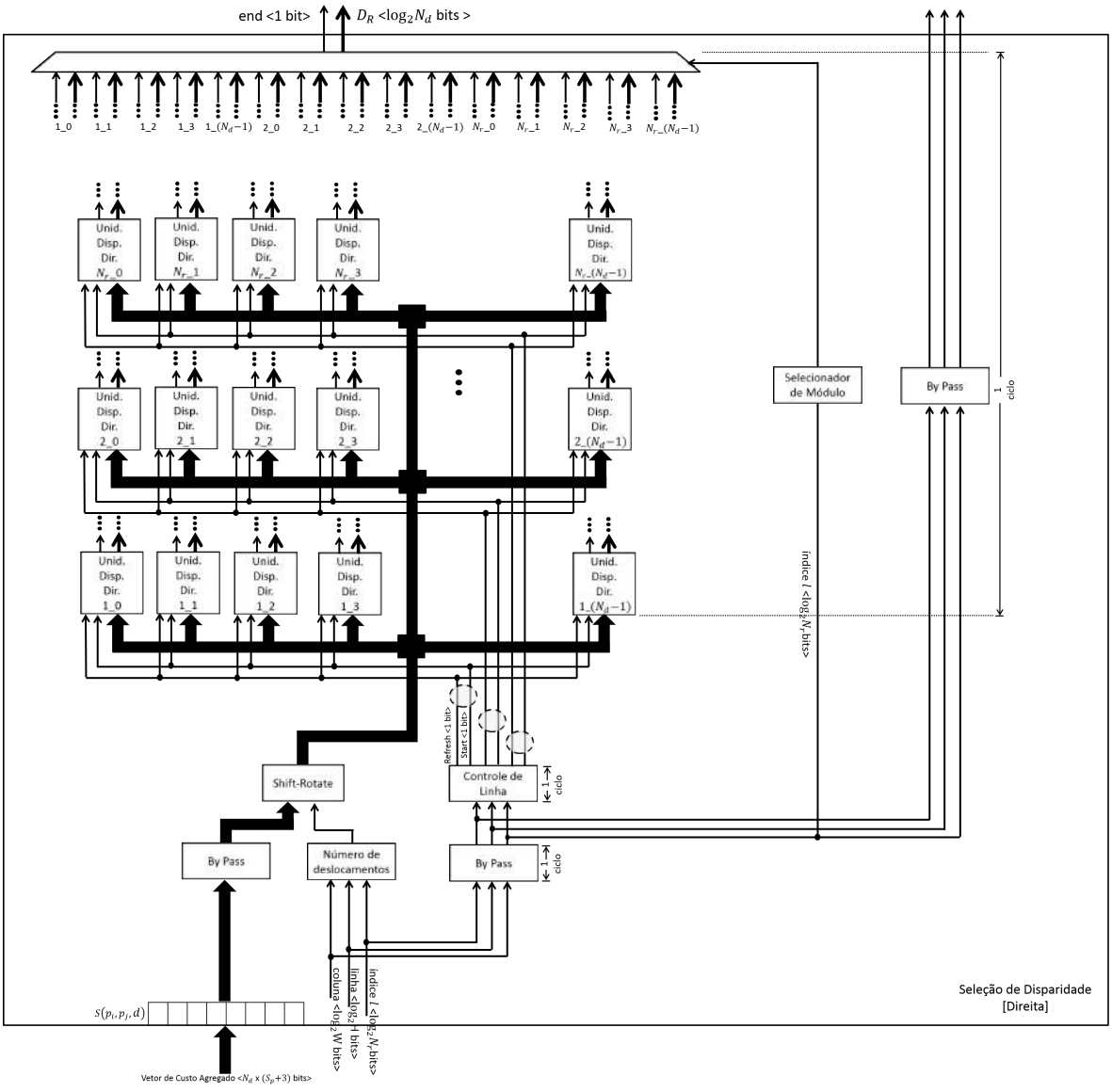
Figura 64 – Arquitetura para o processamento do mapa de disparidade D_R sem pipeline e sem processamento de linha



Para lidar com o processamento de várias linhas e com o processamento em pipeline como proposto neste trabalho, as N_d *unidades de disparidade direita* são replicadas N_r vezes para computar as outras linhas da imagem, seguindo a mesma lógica da arquitetura

anterior, como mostrado na Figura 65. Esta é a arquitetura final que foi concebida e implementada para a computação do mapa disparidade D_R neste trabalho. O módulo *Número de deslocamentos* define a quantidade de deslocamentos que deve ser realizado para cada linha. Este módulo armazena e incrementa a quantidade de deslocamentos de cada linha a medida que a linha é requisitada através do índice l de entrada.

Figura 65 – Arquitetura final para o processamento do mapa de disparidade D_R com pipeline e com processamento de linha



O módulo de *Controle de linha* define a linha que será ativada baseado também no índice l. Este módulo também define a reinicialização através do *refresh* dos módulos de uma determinada linha, quando esta linha tiver sido finalizada. Isto é verificado através da posição da coluna atual e do índice l fornecido na entrada do módulo. Cada *Unidade de disparidade direita* possui um contador interno que é inicializado a cada processamento de linha com o valor do índice atribuído a esta unidade, que vai sendo decrementado até

atingir o valor 0. Quando atingir o valor 0, esta unidade ativa o bit de final, entrega o resultado da disparidade e o seu contador é reinicializado com o valor $N_d - 1$. Enquanto a flag de *refresh* não tiver sido ativada, este módulo reinicializa o contador interno sempre com $N_d - 1$. Quando a flag for ativada, o contador é reinicializado com o valor do índice atribuído ao módulo.

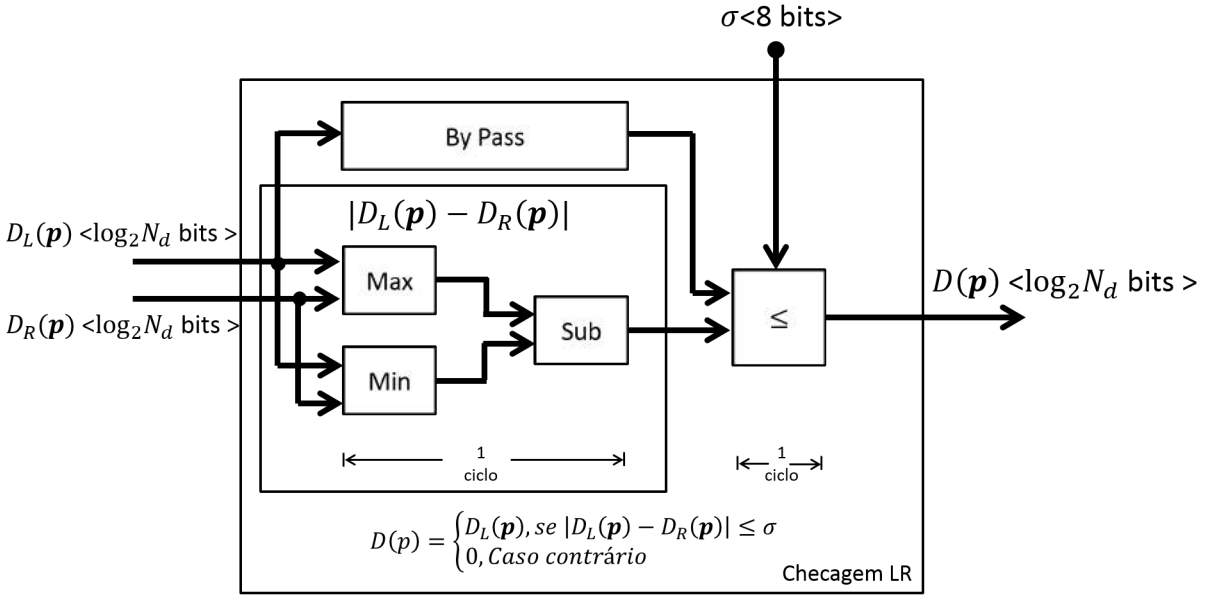
Este módulo de computação de disparidade da direita leva 3 ciclos para computar um valor de disparidade. Mas, uma vez que o pipeline tenha sido preenchido, a taxa de processamento passa a ser de 1 ciclo de relógio por resposta de disparidade.

A implementação deste módulo foi uma das importantes contribuições deste trabalho, devido à sua complexidade e também porque os trabalhos anteriores não têm abordado esta etapa do processamento estéreo que é a detecção de oclusão.

4.2.2.2.6 Checagem LR

Este módulo recebe os dois valores de disparidades de um determinado pixel $\mathbf{p}$, $D_L(\mathbf{p})$ e $D_R(\mathbf{p})$ e calcula o valor final da disparidade $D(\mathbf{p})$ em dois estágios, como mostrado na Figura 66. No primeiro estágio o módulo calcula a diferença absoluta entre os dois valores e no segundo estágio o módulo compara com um limiar. Se for menor ou igual a um parâmetro de limiar, a saída se torna o próprio valor $D_L(\mathbf{p})$, caso contrário, a saída é zerada para indicar que o resultado foi inválido.

Figura 66 – Arquitetura do módulo de Checagem LR



4.2.2.3 Análise de desempenho e uso de recursos da arquitetura proposta

O número de ciclos, dado por P_c, necessários para o processamento do custo de caminho de um pixel para todas as N_d disparidades, atravessando todas as etapas do pipeline mostrado na Figura 53 é dado pela Equação 4.7. A constante de valor 9 desta equação indica a quantidade total de ciclos passando por todos os módulos de pipeline sem considerar o módulo de computação do mínimo, como mostrado na Figura 59. Uma vez que o módulo de computação de mínimo emprega árvores de comparadores, a quantidade de ciclos para computar o mínimo é igual a $log_2 N_d$.

$$P_c = (9 + log_2 N_d)\ [Ciclos] \tag{4.7}$$

A condição necessária para se atingir a máxima eficiência da arquitetura proposta é que o número de linhas N_r processadas serialmente seja maior ou igual a P_c. Essa condição garante que o módulo de custo de caminho estará operando na mesma taxa de entrada dos pixels, ou seja, ciclo a ciclo o sistema sempre processa um novo pixel e devolve um novo resultado de custo de caminho. Esta condição também garante que as FIFOS Onchip de entrada terão tamanho fixo e bem menor do que a quantidade de pixels da imagem. Uma vez que cada FIFO armazena dois pixels, cada um de 1 byte, (um pixel da imagem da esquerda e um da imagem da direita), a estimativa do tamanho necessário de cada FIFO é dado pela Equação 4.8, no qual W é a largura da imagem.

$$S_r = (2 \cdot W) \cdot 16\ [Bits] \tag{4.8}$$

Se N_r for menor que P_c, o tempo de processamento de todos os pixels da diagonal anterior será menor do que o tempo para computar o custo de caminho do primeiro pixel da próxima diagonal. Em outras palavras, a arquitetura precisará esperar o término do cálculo do custo de caminho do primeiro pixel da diagonal anterior para poder computar o pixel da próxima diagonal. Essa espera fará com que o sistema trave o fluxo de entrada de pixel e necessite de FIFOs OnChip para armazenar praticamente a imagem inteira, além de precisar descartar frames periodicamente, o que não é interessante para aplicações de auto desempenho e em plataformas de baixo recurso de armazenamento como nos FPGAs. Embora, não seja aconselhado, a arquitetura lida também com esta situação, de N_r ser menor que P_c. Na arquitetura mostrada na Figura 48, o módulo *Controle de Leitura da FIFO* garante esta quantidade de ciclos de espera entre processamento de diagonais. Sendo assim, o tamanho das FIFOs são configuradas para armazenar o tamanho de uma imagem inteira.

A comunicação das funções de custo de caminho entre as linhas vizinhas é feita através de FIFOS Onchip como mostrado nas Figuras 49b e 53. Cada FIFO armazena para um dado pixel o resultado de seu custo de caminho para todas as disparidades e o mínimo de todos estes custos. Este conjunto de valores intermediários são consumidos durante o

cálculo do custo de caminho do pixel da linha seguinte. FIFOS de tamanhos diferentes são usadas para cada linha do conjunto que está sendo processado. A FIFO que será lida para o processamento dos pixels da primeira linha $l = 1$ de um determinado com conjunto $f = x$ de linhas seriais, armazena o resultado do custo de caminho e mínimo de todos os pixels da última linha $l = N_r$ do conjunto de linhas anterior $f = x - 1$. As FIFOs utilizadas para comunicação entre linhas são menores e armazenam para o custo de caminho $t = 0^o$ e $t = 135^o$ o resultado de apenas um pixel; para $t = 90^o$, o resultado de dois pixels; e para $t = 45^o$, o resultado de três pixels.

Dado que o custo de caminho para uma dada disparidade tem largura de 9 bits, ou seja, um bit a mais em relação ao tamanho do pixel, então um conjunto P_s que compreende os custos de caminhos para todas as disparidades e o seu mínimo, tem o tamanho dado como segue:

$$P_s = (N_d \cdot 9) + 9 \ [Bits] \tag{4.9}$$

Assim, a quantidade de espaço total em memória Onchip para armazenar os pixels de entrada de cada linha, os custos de caminhos intermediários e o mínimo é estimado como segue:

$$\begin{aligned} W_r = (N_r \cdot S_r) + \\ (P_s + 2 \cdot P_s + 3 \cdot P_s) \cdot (N_r - 1) + \\ (3 \cdot W \cdot P_s) \ [Bits] \end{aligned} \tag{4.10}$$

Para definir a quantidade de ciclos total para processar uma imagem é preciso observar três momentos distintos do cálculo do custo de caminho ao longo da imagem. O primeiro momento é quando os pixels precisam ser desalinhados para que o sistema possa processar os pixels de maneira serial. O segundo momento é quando a imagem está desalinhada e o processamento serial está sendo realizado através de todas as N_r linhas. O terceiro momento é quando os pixels precisam ser novamente alinhados para que o sistema possa processar os custos de caminho dos últimos pixels de cada linha. Tomando em consideração estes três momentos, o número de ciclos total C_t estimado para computar a imagem inteira é dado como segue:

$$\begin{aligned} C_t = (2 \cdot (N_r - 1) \cdot P_c) \ + \\ (W \cdot H) \ + \\ (2 \cdot (N_r - 1) \cdot P_c) \ [Ciclos] \end{aligned} \tag{4.11}$$

Onde o primeiro termo compreende a etapa de desalinhamento, o segundo termo compreende o processamento serial e o terceiro termo compreende o alinhamento.

Com a frequência de operação em H_z obtido na plataforma FPGA, o tempo total em segundos pode ser definido usando a Equação 4.11. Por exemplo, considere que imagem a ser processada é de resolução HD (ou seja, W = 1024 e H = 768), intervalo de disparidades

$N_d = 128$, e o processamento envolve $N_r = 16$ linhas seriais. Então, $C_t = (2 \cdot (16-1) \cdot 16) + (1024 \cdot 768) + (2 \cdot (16-1) \cdot 16) = 787.392$ ciclos. A partir deste resultado e com uma frequência de operação do sistema de 100 MHz e obtém-se uma taxa de 127 fps. Com uma frequência de 200 MHz obtêm-se uma taxa de 254 fps. Ainda utilizando o exemplo anterior a quantidade de espaço necessário na memória OnChip é de aproximadamente 4 megabytes. Se fosse utilizado uma resolução de VGA esta quantidade seria 2.6 megabytes e se fosse utilizado resolução QVGA esta quantidade seria aproximadamente 1.4 megabytes.

Os resultados obtidos a partir da Equação 4.10 que estima o tamanho de armazenamento das FIFOs são aproximadamente iguais aos resultados obtidos através de ferramenta de síntese para plataforma FPGA. Importante mencionar que só foi possível obter esta estimativa por causa da alta vazão de processamento desta arquitetura que garante que a quantidade de dados de imagem nunca ultrapassará o tamanho de espaço definido corretamente para as FIFOs. Como consequência e contribuição deste trabalho é possível comportar todo a estrutura de armazenamento dentro do FPGA, não sendo necessário o uso de memórias externas como acontece nos trabalhos de Gehrig (GEHRIG; EBERLI; MEYER, 2009), Banz (BANZ et al., 2010) e Honegger (HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014).

4.3 Conclusão

Neste capítulo foram dadas as motivações para as escolhas das abordagens que compuseram cada etapa do sistema de correspondência estéreo semi global (SGM). Foram detalhados, para cada abordagem, seus respectivos algoritmos. Foi mostrada a importância do filtro de Sobel na pré-processamento para a melhoria da qualidade do mapa de disparidade. Exclusivamente, para o algoritmo SGM, foram discutidos, com mais detalhes, as questões críticas que impendem obter implementações de alto desempenho deste algoritmo. Com base nestas questões, uma arquitetura otimizada foi proposta que combina diversos tipos de abordagens tais como paralelismo em nível de disparidades, paralelismo em nível de processamento de caminhos de propagação, processamento em pipeline do algoritmo SGM e processamento em diagonal de um conjunto de linhas. A combinação destas abordagens permitiu obter uma vazão de processamento de um pixel de disparidade a cada ciclo de relógio. Esta alta vazão de processamento permitiu grande redução da quantidade de informações que precisam ser armazenadas para processamento. Além disso, foi proposta uma arquitetura otimizada para a etapa de processamento de detecção de oclusão. Esta etapa é muito importante pois permite encontrar e remover correspondências de disparidade errada. A arquitetura desenvolvida para esta etapa dá suporte a mesma vazão de processamento da arquitetura proposta do SGM.

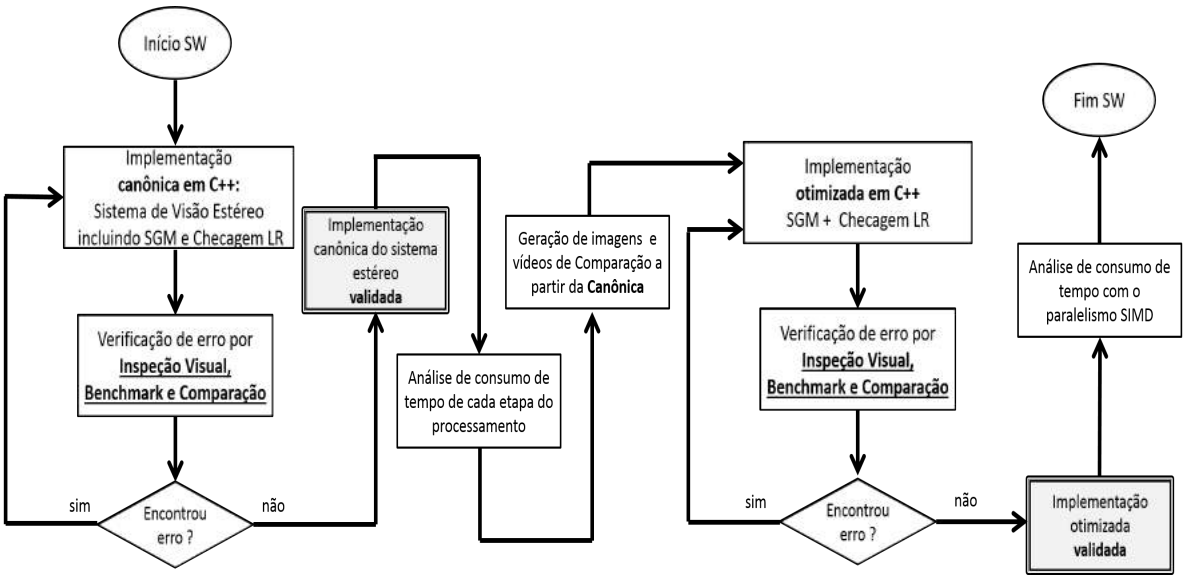
5 VALIDAÇÃO

O fluxo de validação da arquitetura proposta para acelerar o algoritmo de correspondência semi global (SGM) inclui três importantes etapas: a etapa de *software*, etapa de *hardware* e etapa de *software e hardware*. Estas três etapas foram seguidas ao longo de todo este trabalho de mestrado.

5.1 Etapa de Software

Em um primeiro momento, o desenvolvimento seguiu o fluxo mostrado na Figura 67. Todas as etapas do sistema de visão estéreo tais como calibração, retificação, pré processamento, correspondência semi global, computação de disparidade e detecção de oclusão foram implementadas em linguagem de alto nível C++ e integradas. Com exceção da calibração e retificação todas as outras etapas foram implementadas de forma canônica, sem nenhum recurso de otimização de desempenho.

Figura 67 – Fluxo de desenvolvimento e validação da abordagem proposta de visão estéreo em software



A implementação foi validada primeiramente por inspeção visual verificando a coerência do mapa de disparidade gerado pela implementação com o mundo real tridimensional. Para isso foi necessário construir um sistema de câmeras estéreo como mostrado na Figura 68(a). A câmera utilizada foi a Microsfoft Webcam VX 2000 de 1.3MP[1] que custa menos do que 30 dólares. Um outro sistema de câmeras estéreo que também foi utilizado, embora com menos frequência, foi a Webcam Minoru 3D[2] como mostrado na Figura 68(b). A van-

[1] especificação pode ser encontrada em https://www.microsoft.com/accessories/pt-br/d/lifecam-vx-2000

[2] especificação pode ser encontrada em http://www.minoru3d.com/

tagem deste último sistema estéreo é sua praticidade uma vez que a sua estrutura estéreo rígida e já definida evita uma perda de tempo tentando obter os parâmetros precisos de calibração deste sistema. Contudo a qualidade das câmeras são inferiores em relação ao primeiro sistema estéreo construído e também o seu valor comercial para adquiri-lo é alto em relação ao primeiro sistema. Para se ter ideia o valor comercial desta câmera estéreo é de 500 reais. Dessa forma, foi importante gastar um certo tempo tentando construir e calibrar o sistema estéreo a fim de mostrar que é possível gastar menos que 50 reais e obter mapas de disparidades precisos.

Figura 68 – Sistemas de câmeras estéreo utilizados neste trabalho para validar a implementação: (a) Sistema construído (b) Sistema já pronto

(a) (b)

A partir deste sistema de câmera estéreo construído, foram coletadas diversas imagens estéreo de cenários diferentes e também vídeos estéreo em ambientes indoor e outdoor mostrado na Figura 69. Para dar mais suporte a inspeção visual, estes dados coletados foram executados pela abordagem semi global oferecida a partir da biblioteca de visão computacional de código aberto OpenCV (BRADSKI, 2000) e também de uma outra abordagem mais atual proposto por Geiger (GEIGER; ROSER; URTASUN, 2011). Os resultados de mapa de disparidade de ambas as abordagens foram comparados com a implementação canônica desenvolvida neste trabalho. Esta comparação permitiu encontrar regiões onde a implementação canônica falhava e as outras abordagens não falhavam e assim buscar soluções para resolver estas questões. Foi a partir destas imagens de baixo custo que o estudo deste trabalho chegou a solução de usar a derivada de sobel como etapa de pré processamento pois removeu ruídos proveniente deste tipo de câmera.

inda na validação, foi utilizado um framework[3] que permitiu avaliar a abordagem proposta em um banco de imagens estéreo de alta qualidade através de várias métricas tais como erro médio e a porcentagem de pixels errados (estas métricas já foram detalhadas na seção 2.2). O banco de imagem que este framework utiliza é o Middlebury e foi proposto em Scharstein e Szeliski (2002). Este framework disponibiliza imagens de diversos tipos

[3] código pode ser encontrado em http://vision.middlebury.edu/stereo/submit3/

de situações que dificultam o processo de correspondência estéreo. Uma vez que este framework tem sido adotado em diversos trabalhos, foi possível fazer uma comparação real e precisa da qualidade da implementação com outras abordagens existentes na literatura.

Figura 69 – Exemplos de imagens e vídeos coletados para validação: (a) imagens (b) alguns frames

(a) (b)

Após validada a implementação canônica, um estudo de desempenho foi realizado em cada função desta implementação, onde se verificou, como já se era esperado, que as etapas de computação do custo inicial, dos custos de caminho, agregação e seleção de disparidade estavam consumindo a maior parte do processamento e comprometendo o desempenho da resposta de todo o sistema estéreo. Desta forma, foi proposta uma segunda versão utilizando paralelismo SIMD no qual todas estas etapas processaram 8 disparidades paralelamente através de instruções vetoriais de 128 bits. Também foi feita uma implementação com paralelismo em nível de linha, onde cada linha era processada por uma thread independente. Como consequência, a introdução do paralelismo SIMD proporcionou uma redução no tempo de processamento de aproximadamente 7 vezes e o uso de thread permitiu uma redução de duas vezes no tempo de processamento em relação a versão canônica. O objetivo principal destas implementações otimizadas foi aumentar o entendimento da técnica SGM e assim permitir entender melhor os diversos paralelismos que poderiam ser extraídos deste algoritmo.

As etapas de calibração e retificação foram implementadas através de chamadas de funções prontas otimizadas oferecidas pela biblioteca OpenCV. A seguir é explicado o processo de calibração e retificação e como as funções de OpenCV foram utilizadas para realizar tais processos.

5.1.1 Calibração e Retificação

Como já mencionado na seção 2.1, o objetivo principal da calibração é encontrar os parâmetros intrínsecos e extrínsecos do sistema de câmeras. Estes parâmetros permitem obter uma função de transformação entre um objeto no espaço 3D e a imagem 2D observada

pela câmera. Por sua vez, esta função de transformação permite extrair distâncias reais em unidades de comprimento a partir de imagens 2D. A etapa de retificação transforma, a partir dos parâmetros intrínsecos e extrínsecos, cada imagem estéreo em um plano de imagem comum, alinhando as linhas horizontais das duas imagens. Este alinhamento é extremamente necessário para o processo de correspondência estéreo.

Figura 70 – Imagens de diferentes posições do tabuleiro de xadrez para calibração

O método adotado pelo OpenCV para calibração necessita de um conjunto de pares de pontos que correspondem aos mesmos pontos no cenário real 3D. Um procedimento padrão para obter estes pontos é através do uso do tabuleiro de xadrez. São tomadas várias imagens de um tabuleiro de xadrez com um tamanho de quadrado fixo e a partir destas imagens são extraídos os pontos de canto em cada imagem. Estes pontos de canto na imagem correspondem a algum ponto 3D do mundo real (que é fácil de calcular, uma vez que o tabuleiro de xadrez possui uma geometria muito bem definida). Dessa forma, foram armazenadas estas correspondências ponto a ponto e, a partir dessas informações, foi executada a função de calibração do OpenCV que usa um algoritmo não linear, resultando nos parâmetros de calibração. É recomendado que sejam utilizados pelo menos 30 imagens do tabuleiro de xadrez em todas as orientações possíveis do tabuleiro de xadrez para obter bons resultados de calibração como mostrado na Figura 70. Estas imagens foram tomadas a partir do sistema de câmeras construído neste trabalho como mostrado na Figura 68 (a) do qual está se buscando seus parâmetros de calibração para posterior validação.

O código base utilizado neste trabalho para retificação e calibração utilizando métodos do OpenCV se encontra no apêndice A. Ao longo deste trabalho este código foi modificado para calcular a correspondência estéreo semi global tanto em hardware como em software

e também para carregar os parâmetros de calibração diretamente de arquivo externo, uma vez que o sistema de câmera permaneceu inalterado. Basicamente, este código fica esperando a detecção de bordas de um tabuleiro de xadrez nas duas câmeras através da função *findChessboardCorners*, como mostrado na Figura 71. Nesta Figura tem-se alguns exemplos de detecção e localização de pontos de quina a partir das imagens da Figura 70.

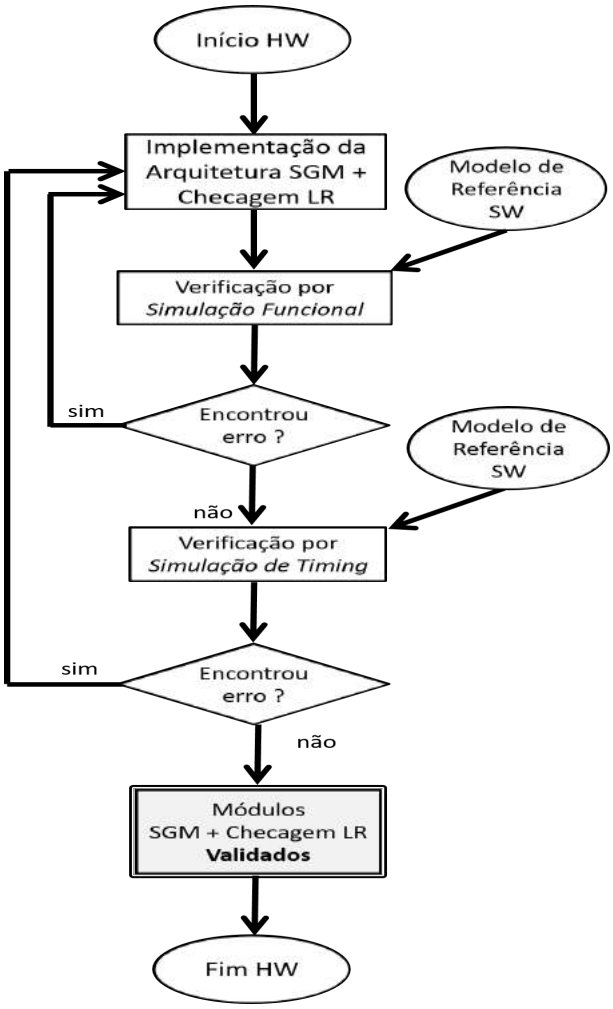
Figura 71 – Exemplo de detecção e localização de pontos de quina no tabuleiro de xadrez que serão usados no processo de calibração

Quando as bordas são detectadas, as suas localizações são armazenadas em estruturas de vetor e o contador de imagens é incrementado. Este processo se repete até que o contador seja maior ou igual a quantidade de imagens solicitadas como parâmetro de entrada para a calibração. Quando o contador for maior que este parâmetro, então os conjuntos de pontos armazenados são passados para a função de calibração, *stereoCalibrate*, que define os parâmetros intrínsecos e extrínsecos dados por *CM1*, *CM2*, *D1*, *D2*, *R*, *T*, *E* e *F*. Estes parâmetros são utilizados como entrada da função *stereoRectify* que define as matrizes de rotação e projeção *R1*, *R2*, *P1*, *P2* bem como a matriz de mapeamento de disparidade para profundidade *Q*. Estes parâmetros são usados na função *initUndistortRectifyMap* que define a função de transformação para retificação e correção de distorções na forma de mapas com base em todos estes parâmetros anteriores. Estes mapas são constantes uma vez que o sistema de câmeras não é modificado e são utilizados como entrada para a função *remap* que transforma os novos frames e para garantir o alinhamento horizontal. Com estas imagens alinhadas, o algoritmo de correspondência estéreo pode ser executado.

5.2 Etapa de Hardware

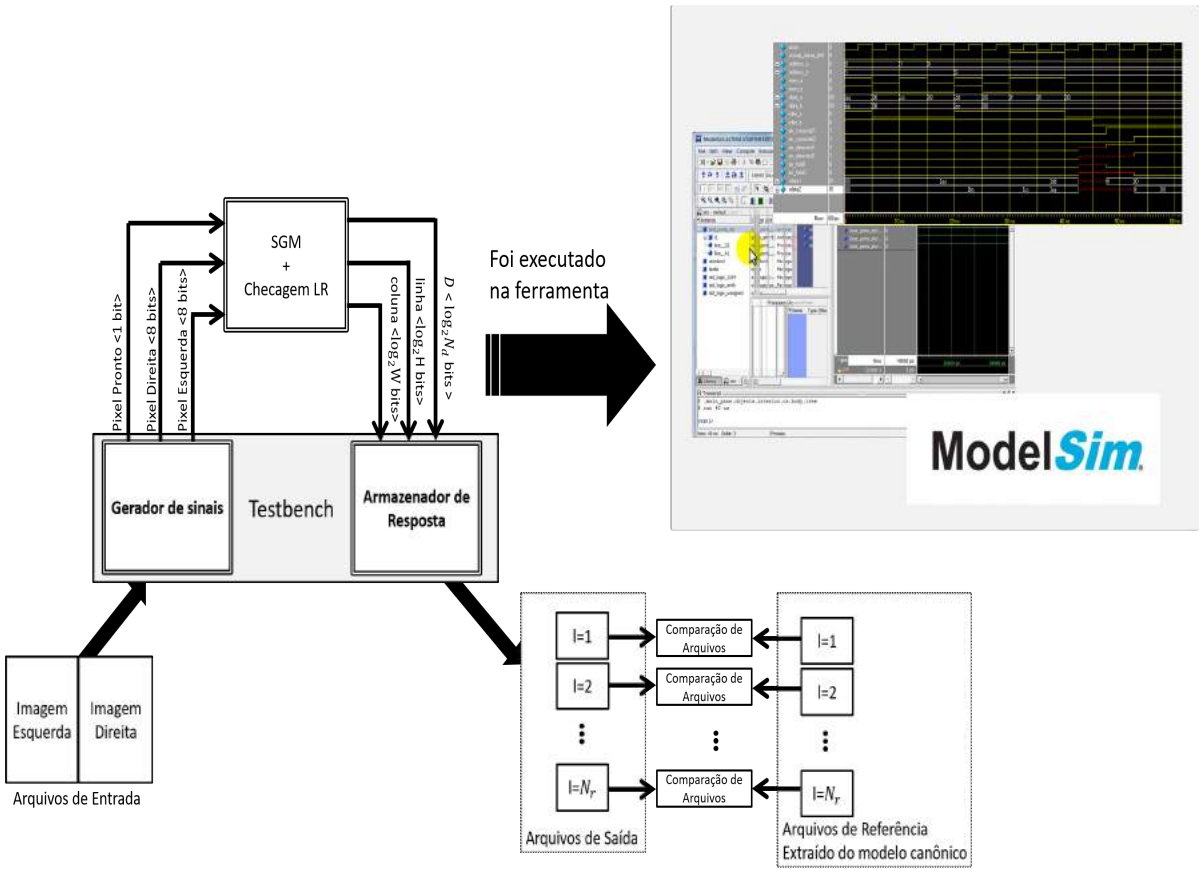
A implementação e validação da arquitetura proposta em hardware de correspondência estéreo semi global seguiu o fluxo mostrado na Figura 72. A implementação de todos os módulos de hardware foi realizada em linguagem de descrição de hardware, SystemVerilog (SPEAR, 2008).

Figura 72 – Fluxo de desenvolvimento e validação da abordagem proposta para acelerar o SGM com detecção de oclusão



Para validar a implementação primeiramente foi construído um ambiente de teste como mostrado na Figura 73 que avalia não apenas o valor da disparidade de resposta mas também a sua posição de linha e coluna. Este ambiente de teste, *testbench*, recebe um conjunto de imagens estéreo já pré-processados em forma de arquivo e envia para a entrada do módulo todos os pixels deste arquivo em forma de sinais de estímulo sequencialmente seguindo o fluxo da esquerda para direita e de cima para baixo, semelhante ao processo de varredura e produção de pixels das câmeras reais. A fim de cobrir diversos tipos de cenários, o código de testbench define um certo grau de aleatoriedade no intervalo de tempo de envio de cada pixel. Isto permitiu encontrar erros cruciais na implementação tal como intervalos de tempo entre pixels, que leva o módulo a perder o fluxo correto de processamento. Outro cenário que o testbench cobre é o envio de vários frames um após o outro, simulando uma situação real, em que o módulo implementado está conectado a um sistema de câmeras enviando continuamente diversos frames sem interrupção. Dessa forma, foi possível encontrar na implementação, erros entre frames que não eram possíveis de serem encontrados se fossem utilizados frames isolados para avaliação.

Figura 73 – Ambiente de teste utilizado para validar a implementação da arquitetura proposta

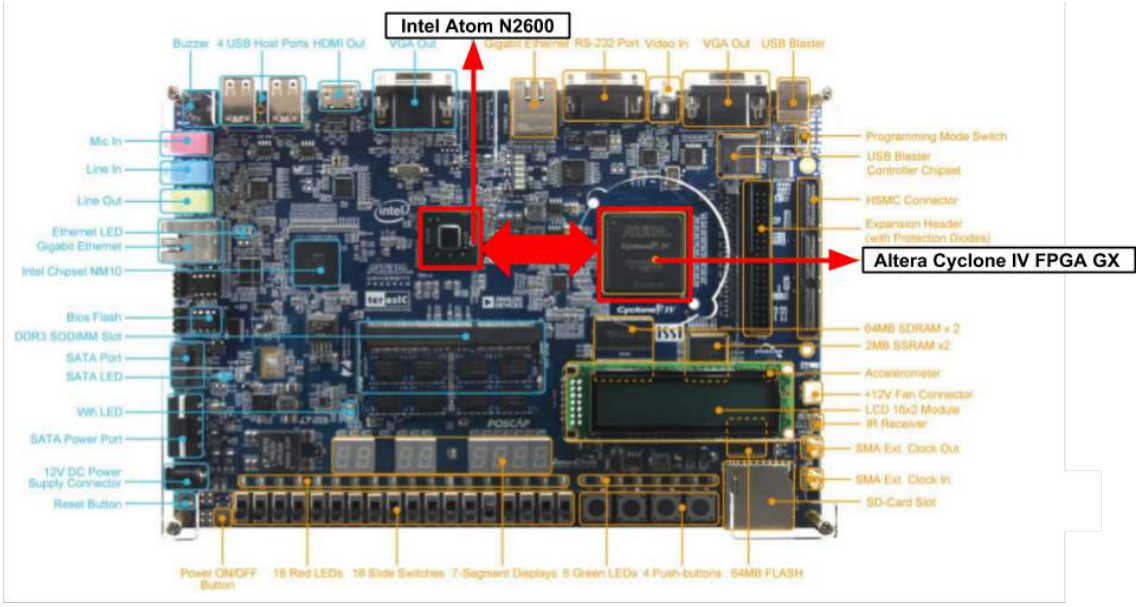


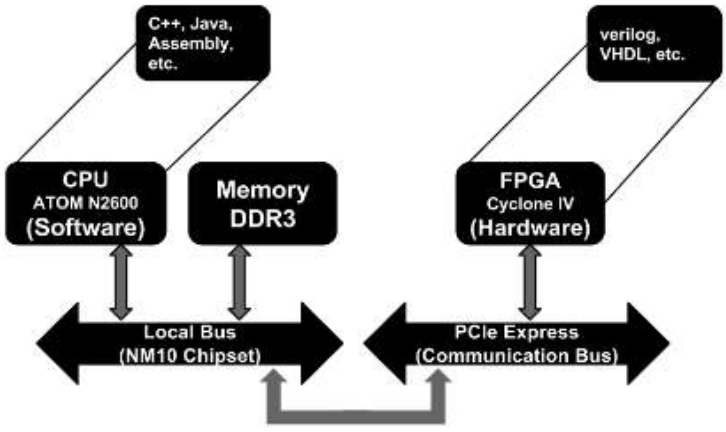
Os resultados do módulo implementado foram armazenados em N_r arquivos pelo testbench de acordo com a sua posição de linha na imagem. O arquivo de índice l armazena as linhas da imagem cuja as posições são $(l+0\cdot N_r)$, $(l+1\cdot N_r)$, $(l+2\cdot N_r)$,...,$(l+(H/N_r)\cdot N_r)$. Esta divisão é realizada para avaliar o processamento do conjunto de linhas seriais da abordagem proposta uma vez que os resultados não são fornecidos regularmente uma linha após a outra. Esta divisão permite que os dados sejam armazenados de maneira organizada nos arquivos. Para cada resultado de disparidade é registrado também a sua posição de coluna. Os dados de cada arquivo de saída são comparados por meio de um comparador de arquivos, com os arquivos gerados pelo modelo canônico escrito em C++. Se todos os arquivos de saída estiverem iguais com os arquivos do modelo canônico então a implementação está validada. Dessa forma, foi possível avaliar todos os parâmetros de saída do módulo implementado e também avaliar diversos cenários do mundo real. A ferramenta utilizada para avaliar, debugar e validar a implementação com o testbench foi o ModelSim (GRAPHICS, 2007). O ModelSim oferece uma maneira de gerar sinais de entrada a partir de linguagem de alto nível tal como SystemVerilog e visualizar todos os sinais internos e externos de todos os módulos implementados através de formas de onda. A ferramenta permite também que os sinais de entrada possam vir de arquivos de textos e os sinais de saída também possam ser exportados para arquivos de texto. Assim, com arquivos de

texto é possível comparar o resultado do módulo SGM implementado com o resultado do seu modelo canônico em C++.

Através desta ferramenta a implementação foi avaliada primeiramente por meio de simulação funcional, na qual o módulo é executado e avaliado apenas através da lógica escrita em SystemVerilog. Dessa forma, diversos erros puramente de lógica funcional puderam ser rapidamente detectados e corrigidos neste tipo de simulação.

Figura 74 – Diagrama de blocos da Placa DE2i-150 destacando a comunicação entre a CPU e o FPGA





Por outro lado, simulação funcional não leva em consideração atrasos de propagação dos sinais reais da plataforma FPGA. Uma simulação de comportamento que leva em consideração além da lógica funcional todos os atrasos de propagação de sinal permite uma simulação mais próxima do que acontece quando o módulo implementado é prototipado na plataforma FPGA, revelando vários erros de funcionalidades que não são capturados

em simulação funcional. Este tipo de simulação é chamado de simulação de tempo. Uma funcionalidade implementada pode funcionar em simulação funcional, mas que, devido a atrasos de propagação, pode não funcionar corretamente em simulação de tempo. Assim, é necessário detectar todas estas funcionalidades críticas, reimplementá-las, validar em simulação funcional e por fim em simulação de tempo. O tempo de atraso é definido em função da plataforma FPGA adotada. Assim, para a simulação de tempo é necessário que o código seja primeiramente sintetizado na ferramenta Quartus utilizando uma determinada plataforma de FPGA. Ao longo deste trabalho, foi utilizado a versão 16.1 Standart Edition do Quartus. A síntese gera dois arquivos com extensão .svo e .sdo onde o primeiro é o código do módulo em um nível mais primitivo, somente com portas lógicas, e o segundo define o atraso de propagação de cada porta lógica de acordo com a plataforma FPGA. Ao longo deste trabalho a plataforma FPGA adotada como referência foi a Cyclone IV[4] (EP4CGX150DF31C7). As razões para usar esta plataforma são várias. A primeira é que esta FPGA oferece um compromisso entre taxa de processamento e consumo de potência[5]. A sua pouca quantidade de recursos computacionais e tecnologias avançadas de fabricação permite atingir consumo de energia menor do que as plataformas comerciais tais como Stratix e Arria. A segunda razão é o baixo custo comercial para comprar esta FPGA em relação as outras plataformas comerciais. Esta plataforma é uma das mais baratas e assim faz com que o custo do projeto seja reduzido. A terceira razão é que esta FPGA é integrada em uma plataforma acadêmica, DE2i-150, que dispunha de um processador de baixo custo, ATOM N2600, conectado a esta FPGA através de um barramento PCI-Express como mostrado na Figura 74.

Esta arquitetura híbrida, processador mais FPGA, permite que o sistema completo de visão estéreo seja implementado parte em um processador, codificada em linguagem de alto nível, tal como C++, e interligada, através de uma função de comunicação com o barramento, com o módulo em hardware de correspondência estéreo semi global. Dessa forma, o tempo de desenvolvimento do projeto inteiro se torna mais curto e o sistema obtém um desempenho de processamento que, mesmo com os atrasos de comunicação, é aceitável para aplicações de tempo real, ou seja, que exige taxa de processamento acima de 30 FPS. Por curiosidade, se o desempenho não fosse atendido, seria por conta do gargalo na comunicação entre as duas plataformas, mas neste trabalho este problema não existiu, devido a uso do protocolo RIFFA (JACOBSEN et al., 2015) que será explicado na Seção 5.3. Além disso, o conjunto formado pelo processador, FPGA e barramento são plataformas de baixo custo e otimizadas para consumir baixa potência. Esta etapa de integração processador mais FPGA para desenvolvimento do sistema estéreo é detalhada na seção seguinte.

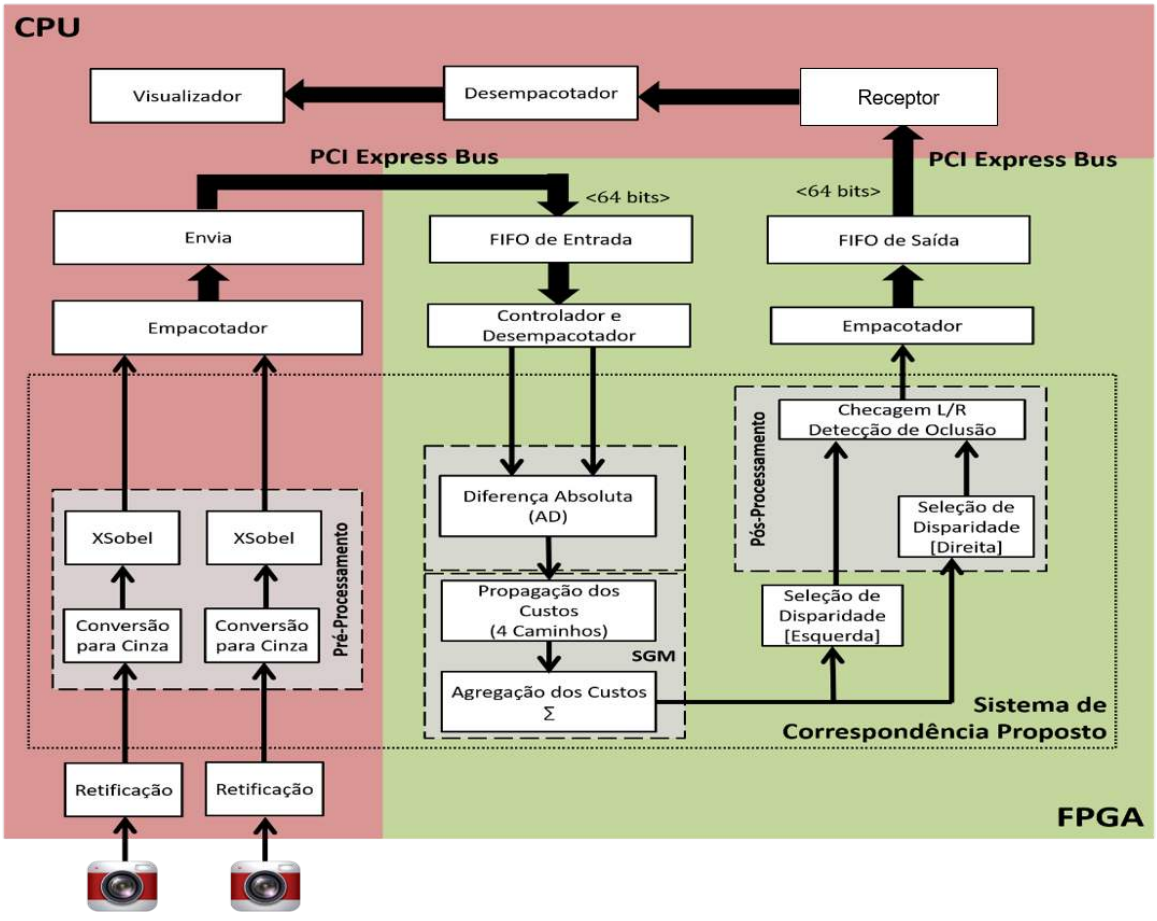
[4] especificação pode ser encontrada em https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html

[5] informações mais detalhadas em https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/features/cyiv-power.html

5.3 Etapa de Software e Hardware

Uma vez que a implementação em hardware do módulo SGM foi validada, a próxima etapa foi a integração deste módulo com as outras etapas de visão estéreo que foram desenvolvidas em software como mostrado na Figura 75. Como já dito na seção anterior a plataforma base adotada neste trabalho para validação foi a DE2i-150. A principal questão foi fazer com que as funções em software e o módulo em hardware se comunicassem através do barramento PCI-Express. O ponto importante, neste contexto, é identificar a maneira como é utilizado o barramento que impacta diretamente no desempenho de todo o sistema. Neste sentido o framework RIFFA (JACOBSEN et al., 2015) consegue atingir a máxima eficiência do barramento disponível. Este framework oferece do lado do processador chamadas de funções em alto nível para envio e recebimento de dados pelo barramento (drivers) e, do lado hardware, oferece uma interface que permite que estes dados cheguem corretamente até o módulo em hardware e que os resultados do processamento a partir do módulo possam ser enviados para a CPU através do barramento.

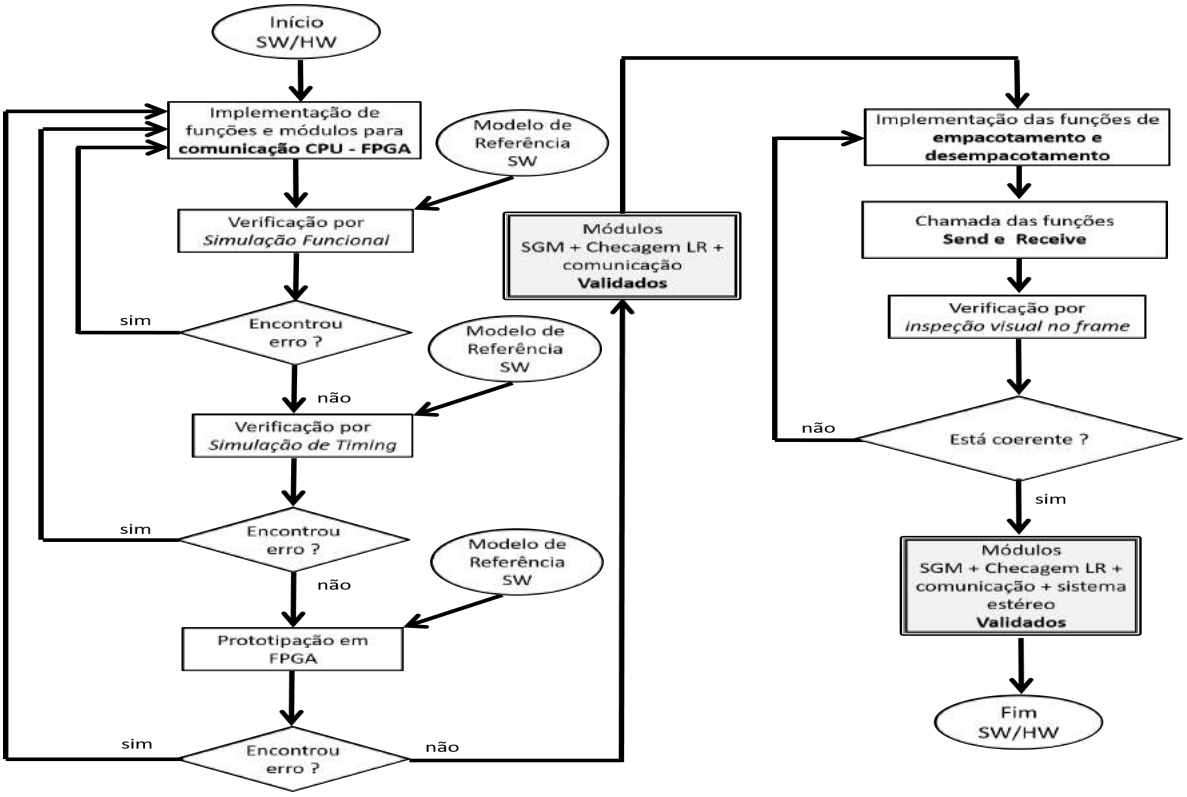
Figura 75 – Sistema de visão estéreo completo implementados na plataforma CPU mais FPGA



Uma vez que o barramento da plataforma DE2i-150 só permite um tipo de transação por vez, é necessário primeiramente que as duas imagens estéreos sejam completamente

enviadas através da função *send* do RIFFA para depois poder chamar a função *receive* para coletar o mapa de disparidade resultante. Dessa forma, foi necessário, inserir estruturas de FIFO antes do módulo SGM para armazenar a imagem, e na saída deste módulo para armazenar o mapa de disparidade. Também foi implementado um módulo para controlar a leitura da FIFO de entrada, verificando se existem dados na FIFO e, caso verdadeiro, envia-os para o módulo SGM processar.

Figura 76 – Fluxo de desenvolvimento e validação da abordagem proposta para o sistema estéreo integrando funções de software e módulos de hardware



A largura do barramento para plataforma DE2i-150 é de 64 bits usando o framework de comunicação RIFFA. Assim, para que a largura seja totalmente ocupada, no lado do processador, as duas imagens estéreo são empacotadas e armazenadas em um vetor de inteiros, onde cada inteiro armazena dois pares de pixels (cada par é um pixel da imagem da esquerda e um pixel da imagem da direita). Quando este vetor é enviado para a FPGA os dados passam a ser representados como 64 bits, que são na verdade dois inteiros de 32 bits concatenados. Cada dado de 64 bits é armazenado diretamente na FIFO a medida que estes estão disponíveis através do protocolo de comunicação do RIFFA. Quando o controlador verifica que tem dados na FIFO de entrada a serem processados, este consome um dado, desempacota-o em 4 pares de pixels e envia cada par para o módulo SGM ciclo após ciclo. Quando o módulo libera a resposta da disparidade, este dado é armazenado no *empacotador* desenvolvido em hardware até que o espaço de 64 bits seja ocupado. É importante lembrar que os dados de saída são compostos pelo valor da disparidade bem

como da posição de linha e coluna. Por exemplo, no caso, para resolução de 320x240 e espaço de disparidade de 64 valores, são necessários 8 bits para representar a disparidade, 9 bits para representar a largura e 8 bits para representar a altura, totalizando uma largura de 25 bits para representar uma saída do módulo. Assim, neste caso, o módulo de empacotamento consegue empacotar duas respostas e os envia para a FIFO de saída. A FIFO de saída está ligada diretamente na interface de comunicação do protocolo RIFFA. Assim, quando o processador solicitar a função *receive*, o barramento vai ler diretamente a partir desta FIFO. A função de *desempacotamento* em software pega cada inteiro recebido e o separa em três campos: disparidade, linha e coluna. Os dados de localização de linha e coluna são utilizados pela função de desempacotamento para escrever os resultados de disparidade nas posições corretas, numa matriz de imagem bidimensional. O trecho de código mostrado no apêndice B detalha o envio e o recebimento dos dados, bem como, cada etapa do sistema de visão estéreo.

Assim, devido a necessidade de implementar módulos de hardware adicionais para a comunicação, todo o conjunto em hardware precisou ser revalidado, tanto em simulação funcional, como em simulação temporal, como mostrado na Figura 76. A diferença para a etapa de hardware, que só validou o módulo SGM, é que os dados são lidos do arquivo de entrada, empacotados e enviados para a FIFO de entrada, simulando o barramento, e os resultados são apresentados a partir da FIFO de saída, desempacotados e enviados para os arquivos de saída.

6 RESULTADOS

Como detalhado na seção 5, o módulo SGM proposto neste trabalho foi validado por simulação funcional, simulação temporal e também por prototipação em sistema de visão estéreo prático. Para ambas as etapas de validação, a plataforma DE2i-150 foi utilizada como suporte que disponibiliza o processador, ATOM2600, integrado com uma FPGA, Cyclone IV (EP4CGX150DF31C7), através do barramento PCI-Express. O framework de comunicação utilizado neste trabalho foi o RIFFA 2.2.2. Para as etapas de simulação, foi utilizado o banco de imagens de alta qualidade Middlebury (SCHARSTEIN; SZELISKI, 2002) e também um banco de imagens reais de baixa qualidade coletadas a partir do sistema estéreo construído neste trabalho. Os resultados de mapa de disparidade, a partir do módulo em hardware, foi comparado com o resultado do modelo canônico implementado em C++ do algoritmo SGM.

Tabela 3 – Utilização de recursos e taxa de processamento sob diferentes configurações na plataforma FPGA Cyclone IV (EP4CGX150DF31C7)

Configuração			Utilização de Recurso			Desempenho	
Resolução	N_d	N_r	LUTs	Reg.	RAM (Bits)	Freq. (MHz)	FPS
320x240 (QVGA)	64	15	52.014	33.349	754.260	100	1220
320x240 (QVGA)	128	16	100.972	64.645	1.424.895	100	1220
640x480 (VGA)	64	15	52.254	33.549	1.459.140	100	324
640x480 (VGA)	128	16	101.010	64.853	2.640.810	100	324
1024x768 (HD)	64	15	52.280	33.578	2.305.140	100	127
1024x768 (HD)	128	16	101.091	64.884	4.162.602	100	127

A partir dos resultados de síntese em FPGA do módulo SGM sem a etapa de checagem LR e das Equações 4.11 e 4.10 da seção 4.2.2.3, as estimativas de taxas de frame de por segundo (FPS) e recursos utilizados, foram obtidas para diferentes resoluções de imagem e intervalos de disparidades N_d, como mostrado na Tabela 3. Como já detalhado na seção 4.2.2.3, a fim de obter eficiência máxima na taxa de processamento, o parâmetro de quantidade de linhas seriais, N_r, precisa ser maior ou igual a quantidade de ciclos P_c para processar um custo de caminho. Se N_r for menor que P_c o módulo *Controle de Leitura da FIFO* insere ciclos de atraso afim de garantir o tempo necessário para que o processamento de um custo de caminho seja finalizado antes do início do processamento do custo de caminho do próximo pixel à frente. Além disso, devido ao valor N_r ser menor que P_c, o sistema define tamanho de FIFOs para alocar praticamente todos os pixels da imagem, o que não é desejado. Desta forma, os resultados foram obtidos com a quantidade de linhas de processamento N_r igual a quantidade de ciclos P_c para processar um custo de

caminho, garantindo a eficiência máxima da arquitetura que é um resultado de disparidade por ciclo de relógio.

Como pode ser visto na tabela 3, o aumento da resolução impacta diretamente no aumento do recurso de memoria onchip e na redução da taxa de processamento de frames completas, enquanto que o número de LUTs e registradores praticamente não são alterados. O aumento de recurso de memória por causa da resolução é devido à necessidade de armazenamento de linhas mais largas de pixels e os custos de caminhos. A diminuição da taxa de frames por segundo (FPS) é devido ao aumento da quantidade de pixels para processar a imagem inteira. Pode ser observado também que o intervalo de disparidade N_d é diretamente proporcional a quantidade de hardware necessário no FPGA. Ao dobrar o valor de N_d praticamente também levou a uma duplicidade do número de LUTs e registradores. Isto é devido a necessidade ao paralelismo total em nível de disparidade proposto neste trabalho que exige mais operadores para computar todos os N_d valores de disparidade paralelamente.

É importante ressaltar que a análise de taxa de processamento mostrado na Tabela 3 leva em conta que o processamento de todos os frames tenha passado pelas etapas de desalinhamento e alinhamento dos pixels para realizar o processamento das linhas seriais. Em um contexto em que é necessário enviar um frame para o módulo de processamento e esperar que todo o mapa de disparidade seja obtido para poder enviar o próximo frame esta análise de tempo é correta, uma vez que as etapas de desalinhamento e alinhamento serão sempre executadas para cada frame. Contudo, se os dados de pixels da imagem vêm continuamente e diretamente a partir das câmeras sem interrupção, a etapa de desalinhamento só precisa ser realizada uma única vez no início de um frame. Para novos frames os dados já estarão desalinhados. Essa forma de processar trabalha diretamente com a mesma taxa de fornecimento dos pixels das câmeras, atingindo assim a máxima eficiência. Neste caso, os tempos de processamento da Tabela 3 certamente serão menores devido justamente a não necessidade de se realizar o desalinhamento e alinhamento em cada frame. Isto pode ser comprovado calculando-se o desempenho através da Equação 4.11 desconsiderando estas duas etapas.

Outra característica importante que pode ser observada na tabela 3 é que a frequência máxima de operação se manteve inalterada em torno dos 100 MHz para qualquer configuração de intervalo de disparidades e resolução. Isto é devido ao fato do módulo SGM ser implementado totalmente como estágios de processamento em pipeline, com operações simples sendo executadas ciclo após ciclo, sem aninhamento de operações. Isto é uma característica importante pois permite que a taxa de processamento de frames dependa apenas da resolução da imagem, permitindo, assim, que se possa calcular previamente a taxa de processamento utilizando a Equação 4.11.

Na Tabela 4, nós comparamos a abordagem proposta com as abordagens a partir dos trabalhos relacionados com relação a alguns aspectos que são críticos obter implementa-

ções de alta qualidade, com baixa quantidade de recursos em hardware da técnica SGM. O uso de janelas de processamento de custo local grandes, tipicamente com tamanho de janela maiores que 3x3, como o que acontece nos trabalhos de (BANZ et al., 2010) e (HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014), certamente exige grande quantidade de recursos de armazenamento e processamento. Dependendo da técnica adotada para calcular o custo local da técnica SGM, é necessário grandes janelas para obter mapas de disparidades precisos. Como será fala mais adiante, a abordagem proposta garante que usando um tamanho pequeno de janela, 3x3, é possível obter mapas de disparidades melhores que as abordagens propostas a partir dos trabalhos relacionados. A replicação do módulo SGM para o processamento da etapa de detecção de oclusão, como o que acontece no trabalho de (GEHRIG; EBERLI; MEYER, 2009) dobra a quantidade de recursos de processamento e armazenamento necessários para computar o mapa de disparidade final. Na abordagem proposta, não há a necessidade de replicação do módulo SGM para calcular a detecção de oclusão. O processamento de direções de propagação contrárias ao fluxo de entrada dos pixels, como o que acontece no trabalho de (GEHRIG; EBERLI; MEYER, 2009), aumenta o tempo de processamento do mapa de disparidade, além de exigir grandes espaços de armazenamentos de dados intermediários. No trabalho proposto, a arquitetura não processa caminhos contrários ao fluxo padrão de varredura dos pixels.

Tabela 4 – Comparativo dos trabalhos relacionados sob alguns critérios

Característica	Referência				**Abordagem proposta**
	Gehrig, Eberli e Meyer (2009)	Banz et al. (2010)	Honegger, Oleynikova e Pollefeys (2014)	Wang et al. (2013)	Cambuim, e Barros (2017)
Tamanho de janela	3x3 (ZSAD)	9x9 (RANK)	7x7 (CENSUS)	5x5 (CENSUS)	3x3 (XSobel+AD)
Número de linhas de propagação	8	4	4	4	4
Detecção de oclusão	Duplica o módulo SGM	**Não** duplica o módulo SGM	**Não** duplica o módulo SGM	**Não** duplica o módulo SGM	**Não** duplica o módulo SGM
Frequência mínima de operação (MHz)	Câmera	Múltiplo da câmera: Câmera: 21 SGM: 109	Múltiplo da câmera: Câmera: 25 SGM: 250	Câmera	Câmera
Memória RAM externa	3 DDR 2	Não adotado	Não adotado	Não adotado	Não adotado
Paralelismo de linha	Não adotado	Adotado: LUT **varia** com o número de linhas de processamento em paralelo	Não adotado	Adotado: LUT **varia** com o número de linhas de processamento em paralelo	Adotado: LUT **não varia** com o número de linhas de processamento em paralelo

Para resolver a questão crítica da baixa vazão de processamento da técnica SGM que são agravadas com a necessidade de processar mapas de disparidades de grandes reso-

luções e intervalos de disparidades, os autores criaram várias estratégias. Nos trabalhos de (BANZ et al., 2010) e (HONEGGER; OLEYNIKOVA; POLLEFEYS, 2014) os autores propuseram aumentar a frequência de operação em múltiplos da frequência da câmera. Já no trabalho de (GEHRIG; EBERLI; MEYER, 2009), os autores propuseram o uso de grandes memórias DDR2 para armazenar todos os pixels e custos intermediários que não foram ainda processados. Ambas as estratégias não são escaláveis para grandes intervalos de disparidades e resoluções de mapa de disparidades. Por outro lado, a adoção do paralelismo de linha, permite aumentar a vazão de processamento, contudo nos trabalhos de (BANZ et al., 2010) e (WANG et al., 2013), foi observado que a quantidade de recursos de processamento é proporcional ao número de linhas que serão processadas em paralelo. Isto faz com que não seja possível definir um número de linhas suficientes para garantir altas vazões de processamento da técnica SGM, tendo que aumentar a frequência de operação ou aumentar a quantidade de armazenamento. Já, na arquitetura proposta, a quantidade de recursos de processamento não varia com o número de linhas que se queira processar serialmente. Assim, foi possível determinar a quantidade necessária de linhas para obter a vazão máxima de processamento que é de uma disparidade por ciclo de relógio, sem aumentar a quantidade de recursos necessários.

Algumas comparações foram realizadas entre a abordagem proposta e as propostas nos trabalhos relacionados que implementam a técnica SGM. A Tabela 5 mostra os resultados de frequência de operação e taxa de processamento em frames por segundo (FPS) para algumas configurações de resolução e intervalos de disparidades. Observa-se que para a mesma resolução e o mesmo intervalo de disparidade, a abordagem proposta processa mapas de disparidades em taxas mais altas do que as outras abordagens. Esta superioridade conseguida é devido a alta vazão de processamento que é de um resultado de disparidade por ciclo de relógio que foi alcançada pela arquitetura proposta. Essa alta vazão é o resultado do uso das abordagens de paralelismo total em nível de disparidade, em nível de caminhos de propagação de custos e em nível de processamento de linhas seriais.

Tabela 5 – Comparativo com os trabalhos relacionados em termos de taxa de processamento. n_i significa *não informado*

		Referência				**Abordagem proposta**		
		Banz et al. (2010)		Wang et al. (2013)		Cambuim, e Barros (2017)		
Configuração	Resolução da imagem	640x480	640x480	640x480	1024x768	640x480	640x480	1024x768
	Intervalo de disparidade	64	128	64	96	64	128	128
Resultados	Frequência de operação (MHz)	133	133	n_i	n_i	100	100	100
	Taxa de processamento	105	59	244.1	63.58	324	324	127

A Tabela 6 mostra comparativos da quantidade de utilização de recursos de armazenamento e processamento da abordagem proposta e das outras abordagens propostas

nos trabalhos relacionados para implementação da técnica SGM em FPGA. É possível verificar que a quantidade de recursos de processamento (LUTs) da abordagem proposta é menor que abordagem proposto pelos trabalhos relacionados. O baixo uso de recursos é devido à serialização proposta neste trabalho. Outro diferencial dessa abordagem é a não utilização de memórias externas. O uso de memórias externas além de reduzir o desempenho aumenta o consumo de energia. Mesmo sem utilizar memórias externas, a arquitetura proposta precisa armazenar bem menos dados intermediários que arquitetura propostos nos trabalhos relacionados. A baixa necessidade de armazenamento de pixels e custos intermediários é uma consequência da alta vazão de processamento que reduz a quantidade de dados que precisam esperar para serem processados. Essa comparação não é justa pois cada trabalho utilizou uma plataforma FPGA diferente. Contudo, uma vez que que tecnologicamente as plataformas utilizados por outros autores são superiores a plataforma adotada neste trabalho, a comparação feita na Tabela 6 ajuda a ter uma perspectiva sobre a utilização de recursos nestas outras plataformas. É esperado que a utilização de recurso seja menor nestas plataformas.

Tabela 6 – Comparativo com os trabalhos relacionados em termos de recursos de processamento e armazenamento. n_i significa *não informado* e n_u significa *não utilizado*

		Referência			**Abordagem proposta**
		Gehrig, Eberli e Meyer (2009)	Banz et al. (2010)	Wang et al. (2013)	Cambuim, e Barros (2017)
Configuração	Resolução da imagem	320x240	640x480	640x480	640x480
	Intervalo de disparidade	64	64	64	64
Resultados de síntese	LUTs	60.000	51.858	139.009	52.254
	Registrador	n_i	7.456	79.314	33.549
	RAM externa	3 DDR 2	n_u	n_u	n_u
	RAM interna (bits)	2.457.60	1.802.24	5.202.77	1.459.140

Além das questões de desempenho e recursos, a qualidade do mapa de disparidade da arquitetura proposta para o cálculo do SGM foi avaliada usando o benchmark Middlebury e usando imagens de cenários do mundo real coletados a partir do sistema de câmeras construído para validação deste trabalho. Com o benchmark Middlebury, o módulo foi avaliado com imagens isoladas Tsukuba, Vênus, Cones, Teddy com os parâmetros P_1 e P_2 do SGM definidos como 10 e 40, respectivamente. Estas imagens isoladas são geralmente utilizadas como avaliação por diversos trabalhos relacionados inclusive aqueles referentes a hardware para SGM. O resultado da Tabela 7 mostra o comparativo entre trabalhos relacionados que propuseram implementação em hardware da técnica SGM e a implementação o proposta neste trabalho. Pode-se observar que, com um tamanho menor de janela

de processamento, a qualidade do mapa de disparidade gerado pela abordagem proposta é ligeiramente melhor que as outras abordagens em hardware.

Tabela 7 – Comparação em termos de qualidade das abordagens em hardware do SGM

Referência	Tamanho de janela	Qualidade			
		% Pixels errados thresh = 1.0			
		Tsukuba	Venus	Cones	Teddy
Gehrig, Eberli e Meyer (2009)	3x3 (ZSAD)	5.86	3.85	9.54	13.28
Banz et al. (2010)	9x9 (RANK)	6.8	4.1	9.5	13.3
Wang et al. (2013)	5x5 (CENSUS)	2.95	1.43	11.1	13.8
Abordagem proposta	3x3 (XSobel+AD)	4.1	2.5	8.9	10.3

A abordagem proposta também foi avaliada com o conjunto de imagens a partir da terceira versão do benchmark Middlebury proposto por Scharstein (SCHARSTEIN et al., 2014). Este benchmark possui três tipos de configurações: completo (com resolução até 3000 x 2000 e intervalo de disparidades de até 800), metade (com resolução até 1500 x 1000 e intervalo de disparidade de até 400) e quarto (com resolução até 750 x 500 e intervalo de disparidade de até 200). Este trabalho adotou a configuração de metade, a configuração mais utilizada nas avaliações dos trabalhos mais recentes. Esta configuração permite ter uma gama mais ampla de abordagens e técnicas para comparação que não necessariamente são implementados em hardware. Para este banco de dados, os valores dos parâmetros P_1 e P_2 do SGM são 10 e 40, respectivamente. Assim, a porcentagem média de pixels ruins (SCHARSTEIN; SZELISKI, 2002) para algumas imagens do banco de dados de treinamento com o limite (ou threshold) de 1.0 é 22.7% como mostrado na tabela 8.

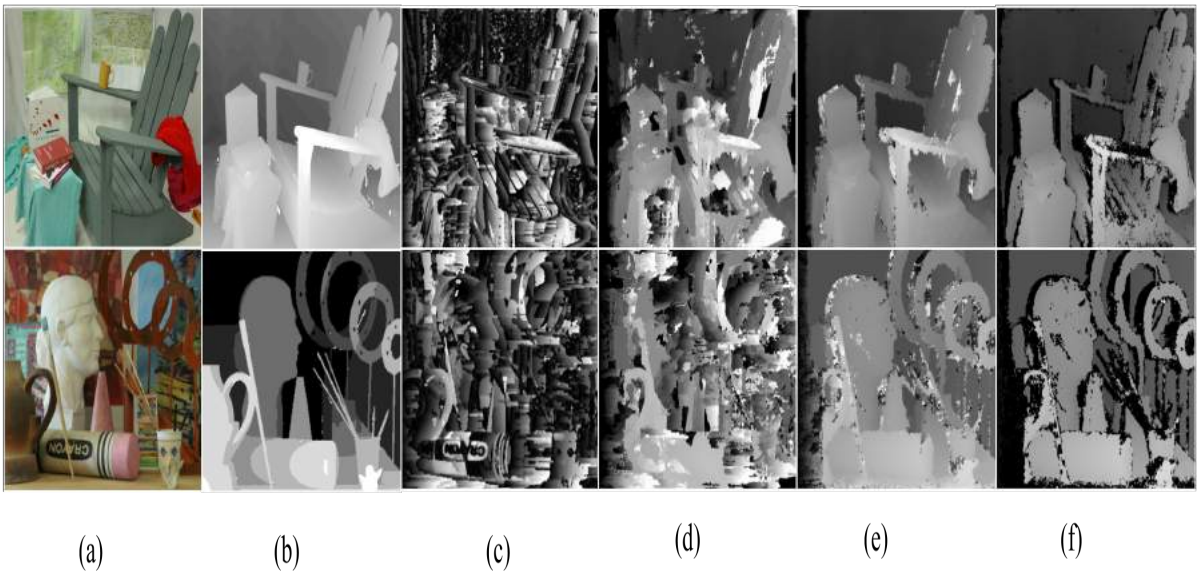
Tabela 8 – Percentagens de pixel ruim com um limiar de 1 pixel em áreas não ocludidas

Imagem	Referência						**Abordagem proposta**	
	Park e Lee (2016)	Kim e Kim (2016)	Zhang et al. (2015)	Barron e Poole (2016)	Kowalczuk, Psota e Perez (2013)	Stentoumis et al. (2014)	**sem pre processamen.**	**com pre processamen.**
Adirondack	11,0	14,3	20,6	12,2	24,9	31,0	69,59	23,57
ArtL	7,66	9,3	14,0	12,6	14,1	16,9	80,71	19,58
Motorcycle	8,27	9,82	20,6	11,0	24,2	27,9	44,79	17,89
Piano	21,9	22,4	27,0	23,4	24,6	27,8	60,51	26,12
Pipes	8,62	9,80	23,4	12,0	19,4	23,1	39,93	17,68
Playtable	21,0	22,4	34,1	26,7	62,0	45,6	61,01	45,19
Recycle	17,3	17,8	25,9	16,6	25,2	28,7	59,33	21,62
Teddy	5,81	6,16	10,3	7,06	8,67	10,8	21,97	10,30
Média de pixels errados	12,7	14,1	24,5	15,2	25,3	26,5	54,9	22,7

Com este resultado, a abordagem proposta foi comparada com algumas implementações de correspondência de estéreo disponíveis de estado da arte como mostrado na Tabela 8. A qualidade do sistema está entre as melhores abordagens atualmente disponíveis. As

abordagens mais precisas tais como aquelas propostas por Park (PARK; LEE, 2016) e Kim (KIM; KIM, 2016) são métodos que empregam técnicas de redes neurais convolucionais (LECUN; BENGIO, 1998) que não são atualmente adequadas para implementações em tempo real e em plataformas de baixo recurso computacional, como FPGAs.

Figura 77 – Mapa de disparidade obtido a partir do sistema proposto para as imagens Adirondack e ArtL da terceira versão do Middlebury: (a) imagem original. (b) mapa de disparidade de referência (c) mapa de disparidade usando a abordagem AD. (d) mapa de disparidade usando abordagem AD e SGM. (e) mapa de disparidade usando sobel, AD e SGM. (f) mapa de disparidade usando pré-processamento, AD, SGM e pós-processamento.



Para validar o efeito da otimização semi-global e da etapa de pré-processamento, foi obtido o resultado do mapa de disparidade para abordagem puramente local com a aplicação da técnica de diferença absoluta AD, com a combinação do AD e SGM e, finalmente, com a combinação do filtro de sobel, AD e SGM como mostrado na Figura 77. Pode ser observado que a otimização semi-global melhora os resultados de disparidade para o mapa produzido a partir de AD, mas ainda falha em regiões de baixas texturas devido à baixa robustez do AD para discriminar estas regiões. Com a adição da etapa de pré-processamento, os resultados em regiões de baixa textura melhoraram porque o filtro de gradiente consegue discriminar melhor nestas áreas. Essa melhoria em regiões de baixa é necessária tanto para imagens de alta qualidade como para baixa qualidade.

O sistema inteiro proposto (pré-processamento, AD, SGM, pós-processamento) foi testado em alguns cenários fornecidos pelo sistema de câmera estéreo construído neste trabalho. O mapa de disparidade para algumas imagens é mostrado na Figura 78 com P_1 e P_2 definidos como sendo 20 e 60, respectivamente. Os resultados mostram a robustez de todo o sistema, com câmeras de baixa resolução e em cenários com iluminação diferente e baixa textura.

A fim de demonstrar com mais clareza o ganho que a abordagem proposta oferece, comparamos o tempo de processamento do mapa de disparidade com o método SGM oferecido pela biblioteca OpenCV e o com o módulo implementado neste trabalho. O método SGM do OpenCV possui várias otimizações e uma delas é o processamento paralelo de 8 disparidades através de instruções vetoriais SIMD. A comparação foi realizada utilizando a plataforma DE2i-150 em uma resolução de 640x480 com o intervalo de disparidades de 64 valores. A comparação é feita medindo apenas o tempo da função que computa o mapa de disparidade pelo OpenCV e depois medindo o tempo de toda a abordagem hardware/software proposta neste trabalho para o SGM com detecção de oclusão como mostrado no trecho de código do apêndice B. Como resultado foi observado que o SGM com o OpenCV oferece mapa de disparidade a uma taxa de 2.5 frames por segundo enquanto que a abordagem proposta oferece a uma taxa de 52 frames por segundo, ou seja, um ganho de desempenho de 21 vezes.

Figura 78 – Resultados de mapa de disparidade do sistema completo proposto (pré-processamento, AD, SGM, pós-processamento) a partir de imagens reais

A implementação proposta também foi avaliada em termos de dissipação de potência, como mostrado na Tabela 9. A medição foi realizada através da ferramenta PowerPlay[1] disponibilizada pela ferramenta Quartus. Observa-se que a dissipação de potência da arquitetura proposta foi menor (2 Watts) que a dissipação das arquiteturas propostas nos trabalhos relacionados. Embora a ferramenta não garanta a estimativa de dissipação de potência de maneira precisa, o resultado obtido permite ter uma noção que a dissipação da arquitetura proposta está na mesma ordem de grandeza que a dissipação das arquiteturas propostas nos trabalhos relacionados.

[1] Manual pode ser encontrado em http://www.altera.com/support/devices/estimator/pow-powerplay.html

Tabela 9 – Comparação em termos de dissipação de potência das abordagens em hardware do SGM. n_i significa *não informado*

	Referência				**Abordagem proposta**
	Gehrig, Eberli e Meyer (2009)	Banz et al. (2010)	Honegger, Oleynikova e Pollefeys (2014)	Wang et al. (2013)	Cambuim, e Barros (2017)
Potência dissipada (Watts)	5W	n_i	3W	n_i	2W

Todos estes resultados de alto desempenho processamento, baixo recurso de hardware e alta qualidade dos resultados do mapa de disparidade permitiu uma publicação da abordagem proposta neste trabalho mestrado no 30º Simpósio em Circuitos Integrados e Projeto de Sistemas (SBCCI 2017) (CAMBUIM; ; BARROS, 2017).

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho focou na implementação de um hardware eficiente de alta qualidade e alta resolução do algoritmo de correspondência semi-global (SGM). Dado a necessidade por alto desempenho e pouca quantidade de armazenamento, esta abordagem propõe uma combinação de técnicas tais como serialização para o processamento de um conjunto de linhas da imagem, do paralelismo total das disparidades, paralelismo das funções de propagação de custo e computação em pipeline.

A abordagem de serialização fez com que o número de LUTs necessário para arquitetura se tornasse invariável com o número de linhas da imagem que se queira processar em paralelo. O paralelismo total de disparidades e o paralelismo das funções de propagação de custo fez com que a taxa de processamento não fosse degradada com o aumento do intervalo de disparidade. A abordagem em pipeline fez com que a frequência de operação se tornasse invariável para qualquer configuração da arquitetura. Com todas estas vantagens, a arquitetura proposta conseguiu processar em altas taxas de frames (127 FPS) mapas de disparidades em altas resoluções (1024 x 768), processando grande intervalo de disparidades (128 valores). A arquitetura proposta ocupou pouca quantidade de recursos de processamento de FPGA (100.000 LUTs na plataforma Cyclone IV) e de armazenamento (aproximadamente 4 megabytes) sem utilizar memória externa. Assim, levando em consideração as taxas de frames e o pouco uso de recursos FPGA, a abordagem proposta superou os melhores abordagens em hardware existentes do SGM.

Esta abordagem foi avaliada em um sistema de visão estéreo completo usando a plataforma hardware/software, DE2i-150. No processador ATOM2600 foram implementados as etapas de calibração, retificação e funções de comunicação com o FPGA. No lado do FPGA, além da implementação do SGM também foi implementada a etapa de detecção de oclusão, uma importante etapa de visão estéreo que permitiu encontrar e remover regiões que são fontes de possíveis erros de disparidade. Todo este sistema dissipou 2 Watts de potência o que torna esta implementação atrativa para sistemas embarcados. Este sistema completo permitiu um ganho de desempenho de 21 vezes em relação a implementação em processador oferecida pela biblioteca OpenCV.

Com relação à precisão, a abordagem do SGM proposta foi avaliado através do benchmark Middlebury, no qual esta abordagem gerou mapas de disparidades com qualidades comparáveis às melhores abordagens presentes na literatura. A introdução do filtro XSobel, como etapa de pré-processamento mais a diferença absoluta, como etapa de custo inicial, permitiu que abordagem obtivesse mapas de disparidades com qualidade superior as abordagens em hardware propostas por outros estudos.

A avaliação também foi realizada com imagens reais tiradas do sistema de câmera estéreo construído neste trabalho no qual a abordagem mostrou ser bastante robusta em

imagens de baixa qualidade e em regiões de baixa textura. A adoção da etapa de pré-processamento permitiu melhorias significativas na qualidade do mapa de disparidade para ambas as bases de imagem tanto o Middlebury como as imagens reais.

Como trabalhos futuros podemos sugerir que a técnica SGM seja melhorada em termos de qualidade do mapa de disparidade. Na etapa do cálculo de correspondência local, métodos mais robustos podem ser adotados baseados em janelas adaptativas ou técnicas que combinam métodos locais como evidenciado no trabalho de Mei (MEI et al., 2011). As melhorias também podem ser feitas no módulo SGM, definindo modos de penalidade adaptativas como evidenciado no trabalho de Hirschmuller (HIRSCHMULLER, 2008), ou definindo pesos dinâmicos para cada direção do custo do caminho, conforme proposto por Spangenberg (SPANGENBERG; LANGNER; ROJAS, 2013). Para todas estas melhorias adicionais é necessário verificar se o aumento no número recursos de hardware é justificado pelo ganho em precisão do mapa de disparidade.

REFERÊNCIAS

BAILEY, D. G. *Design for Embedded Image Processing on FPGAs*. 1st. ed. [S.l.]: Wiley Publishing, 2011. ISBN 0470828498, 9780470828496.

BANZ, C.; HESSELBARTH, S.; FLATT, H.; BLUME, H.; PIRSCH, P. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation. In: IEEE. *Embedded Computer Systems (SAMOS), 2010 International Conference on*. [S.l.], 2010. p. 93–101.

BARRON, J. T.; POOLE, B. The fast bilateral solver. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 617–632.

BENEDETTI, L.; CORSINI, M.; CIGNONI, P.; CALLIERI, M.; SCOPIGNO, R. Color to gray conversions in the context of stereo matching algorithms. *Machine Vision and Applications*, v. 23, n. 2, p. 327–348, Mar 2012. ISSN 1432-1769. Disponível em: <https://doi.org/10.1007/s00138-010-0304-x>.

BOYKOV, Y.; VEKSLER, O.; ZABIH, R. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, Washington, DC, USA, v. 23, n. 11, p. 1222–1239, nov. 2001. ISSN 0162-8828. Disponível em: <http://dx.doi.org/10.1109/34.969114>.

BRADSKI, G. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, Miller Freeman Inc., v. 25, n. 11, p. 120–123, 2000.

BROWN, S.; VRANESIC, Z. *Fundamentals of Digital Logic with VHDL Design*. [S.l.]: McGraw-Hill Higher Education, 2000.

CAMBUIM, L. F. S.; ; BARROS, E. N. S. Hardware module for low-resource and real-time stereo vision engine using semi-global matching approach. In: *2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI)*. [S.l.: s.n.], 2017. p. 1–6.

COURTOY, M. Rapid system prototyping for real-time design validation. In: IEEE. *Rapid System Prototyping, 1998. Proceedings. 1998 Ninth International Workshop on*. [S.l.], 1998. p. 108–112.

DAGUM, L.; MENON, R. Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 5, n. 1, p. 46–55, jan. 1998. ISSN 1070-9924. Disponível em: <http://dx.doi.org/10.1109/99.660313>.

DUTRA, B. H. T. C. Desenvolvimento de uma plataforma com uma arquitetura escalável para multiplicação de matrizes densas em sistemas reconfiguráveis de alto desempenho. In: *Dissertação de Mestrado*. [S.l.: s.n.], 2010.

GEHRIG, S. K.; EBERLI, F.; MEYER, T. A real-time low-power stereo vision engine using semi-global matching. In: SPRINGER. *International Conference on Computer Vision Systems*. [S.l.], 2009. p. 134–143.

GEHRIG, S. K.; RABE, C. Real-time semi-global matching on the cpu. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. [S.l.: s.n.], 2010. p. 85–92. ISSN 2160-7508.

GEIGER, A.; ROSER, M.; URTASUN, R. Efficient large-scale stereo matching. *Computer Vision–ACCV 2010*, Springer, p. 25–38, 2011.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.

GRAPHICS, M. *ModelSim*. 2007.

HADJITHEOPHANOUS, S.; TTOFIS, C.; GEORGHIADES, A. S.; THEOCHARIDES, T. Towards hardware stereoscopic 3d reconstruction: A real-time fpga computation of the disparity map. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010. (DATE '10), p. 1743–1748. ISBN 978-3-9810801-6-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1870926.1871347>.

HALLER, I.; PANTILIE, C.; ONIGA, F.; NEDEVSCHI, S. Real-time semi-global dense stereo solution with improved sub-pixel accuracy. In: IEEE. *Intelligent Vehicles Symposium (IV), 2010 IEEE*. [S.l.], 2010. p. 369–376.

HARTLEY, R.; ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. 2. ed. New York, NY, USA: Cambridge University Press, 2003. ISBN 0521540518.

HARTLEY, R.; ZISSERMAN, A. *Multiple view geometry in computer vision*. [S.l.]: Cambridge university press, 2003.

HERNANDEZ-JUAREZ, D.; CHACÓN, A.; ESPINOSA, A.; VÁZQUEZ, D.; MOURE, J. C.; LÓPEZ, A. M. Embedded real-time stereo estimation via semi-global matching on the gpu. *Procedia Computer Science*, Elsevier, v. 80, p. 143–153, 2016.

HIRSCHMULLER, H. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 30, n. 2, p. 328–341, Feb 2008. ISSN 0162-8828.

HIRSCHMULLER, H.; GEHRIG, S. Stereo matching in the presence of sub-pixel calibration errors. In: IEEE. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. [S.l.], 2009. p. 437–444.

HIRSCHMULLER, H.; SCHARSTEIN, D. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 31, n. 9, p. 1582–1599, Sept 2009. ISSN 0162-8828.

HONEGGER, D.; OLEYNIKOVA, H.; POLLEFEYS, M. Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu. In: IEEE. *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. [S.l.], 2014. p. 4930–4935.

JACOBSEN, M.; RICHMOND, D.; HOGAINS, M.; KASTNER, R. Riffa 2.1: A reusable integration framework for fpga accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, ACM, v. 8, n. 4, p. 22, 2015.

KIM, C. *3D Reconstruction and Rendering from High Resolution Light Fields*. Tese (Doutorado) — ETH Zurich, 2015.

KIM, K.-R.; KIM, C.-S. Adaptive smoothness constraints for efficient stereo matching using texture and edge information. In: IEEE. *Image Processing (ICIP), 2016 IEEE International Conference on*. [S.l.], 2016. p. 3429–3433.

KOWALCZUK, J.; PSOTA, E. T.; PEREZ, L. C. Real-time stereo matching on cuda using an iterative refinement method for adaptive support-weight correspondences. *IEEE transactions on circuits and systems for video technology*, IEEE, v. 23, n. 1, p. 94–104, 2013.

KOWSARI, T.; BEAUCHEMIN, S. S.; CHO, J. Real-time vehicle detection and tracking using stereo vision and multi-view adaboost. In: IEEE. *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. [S.l.], 2011. p. 1255–1260.

LECUN, Y.; BENGIO, Y. The handbook of brain theory and neural networks. In: ARBIB, M. A. (Ed.). Cambridge, MA, USA: MIT Press, 1998. cap. Convolutional Networks for Images, Speech, and Time Series, p. 255–258. ISBN 0-262-51102-9. Disponível em: <http://dl.acm.org/citation.cfm?id=303568.303704>.

MAGDALENO, E.; RODRÍGUEZ, M. Acceleration of computation speed for wavefront phase recovery using programmable logic. In: *Topics in Adaptive Optics*. [S.l.]: InTech, 2012.

MEERBERGEN, G. V.; VERGAUWEN, M.; POLLEFEYS, M.; GOOL, L. V. A hierarchical symmetric stereo algorithm using dynamic programming. *Int. J. Comput. Vision*, Kluwer Academic Publishers, Hingham, MA, USA, v. 47, n. 1-3, p. 275–285, abr. 2002. ISSN 0920-5691. Disponível em: <http://dx.doi.org/10.1023/A:1014562312225>.

MEI, X.; SUN, X.; ZHOU, M.; JIAO, S.; WANG, H.; ZHANG, X. On building an accurate stereo matching system on graphics hardware. In: IEEE. *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. [S.l.], 2011. p. 467–474.

MURRAY, D.; LITTLE, J. J. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, v. 8, n. 2, p. 161–171, Apr 2000. ISSN 1573-7527. Disponível em: <http://dx.doi.org/10.1023/A:1008987612352>.

PANTILIE, C. D.; NEDEVSCHI, S. Sort-sgm: Subpixel optimized real-time semiglobal matching for intelligent vehicles. *IEEE Transactions on Vehicular Technology*, v. 61, n. 3, p. 1032–1042, March 2012. ISSN 0018-9545.

PARK, H.; LEE, K. M. Look wider to match image patches with convolutional neural networks. *IEEE Signal Processing Letters*, IEEE, 2016.

ROCHA, R. C. F. Desenvolvimento de uma plataforma reconfigurável para modelagem 2d, em sísmica, utilizando fpga. In: *Dissertação de Mestrado*. [S.l.: s.n.], 2010.

SANGIOVANNI-VINCENTELLI, A.; ZENG, H.; NATALE, M. D.; MARWEDEL, P. *Embedded Systems Development: From Functional Models to Implementations*. [S.l.]: Springer Publishing Company, Incorporated, 2013. ISBN 1461438780, 9781461438786.

SCHARSTEIN, D.; HIRSCHMÜLLER, H.; KITAJIMA, Y.; KRATHWOHL, G.; NEŠIĆ, N.; WANG, X.; WESTLING, P. High-resolution stereo datasets with subpixel-accurate ground truth. In: SPRINGER. *German Conference on Pattern Recognition*. [S.l.], 2014. p. 31–42.

SCHARSTEIN, D.; SZELISKI, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, v. 47, n. 1, p. 7–42, 2002. ISSN 1573-1405. Disponível em: <http://dx.doi.org/10.1023/A:1014573219977>.

SINHA, S. N.; SCHARSTEIN, D.; SZELISKI, R. Efficient high-resolution stereo matching using local plane sweeps. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 1582–1589.

SIVARAMAN, S.; TRIVEDI, M. M. A review of recent developments in vision-based vehicle detection. In: *in IEEE Conf. Intell. Veh. Symp.* [S.l.: s.n.], 2013.

SPANGENBERG, R.; LANGNER, T.; ADFELDT, S.; ROJAS, R. Large scale semi-global matching on the cpu. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. [S.l.: s.n.], 2014. p. 195–201. ISSN 1931-0587.

SPANGENBERG, R.; LANGNER, T.; ROJAS, R. Weighted semi-global matching and center-symmetric census transform for robust driver assistance. In: SPRINGER. *International Conference on Computer Analysis of Images and Patterns*. [S.l.], 2013. p. 34–41.

SPEAR, C. *SystemVerilog for verification: a guide to learning the testbench language features*. [S.l.]: Springer Science & Business Media, 2008.

STEINGRUBE, P.; GEHRIG, S. K.; FRANKE, U. Performance evaluation of stereo algorithms for automotive applications. In: *Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems*. Berlin, Heidelberg: Springer-Verlag, 2009. (ICVS '09), p. 285–294. ISBN 978-3-642-04666-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-04667-4_29>.

STENTOUMIS, C.; GRAMMATIKOPOULOS, L.; KALISPERAKIS, I.; KARRAS, G. On accurate dense stereo-matching using a local adaptive multi-cost approach. *ISPRS Journal of Photogrammetry and Remote Sensing*, Elsevier, v. 91, p. 29–49, 2014.

SUN, Z.; BEBIS, G.; MILLER, R. On-road vehicle detection: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 28, n. 5, p. 694–711, May 2006. ISSN 0162-8828.

TERASIC. *DE2i-150 FPGA System User Manual*. 2013.

TSENG, Y.-C.; CHANG, T.-S. Architecture design of belief propagation for real-time disparity estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, v. 20, n. 11, p. 1555–1564, 2010.

WANG, W.; YAN, J.; XU, N.; WANG, Y.; HSU, F. H. Real-time high-quality stereo vision system in fpga. In: *2013 International Conference on Field-Programmable Technology (FPT)*. [S.l.: s.n.], 2013. p. 358–361.

WANG, W.; YAN, J.; XU, N.; WANG, Y.; HSU, F. H. Real-time high-quality stereo vision system in fpga. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 25, n. 10, p. 1696–1708, Oct 2015. ISSN 1051-8215.

ZHANG, C.; LI, Z.; CHENG, Y.; CAI, R.; CHAO, H.; RUI, Y. Meshstereo: A global stereo model with mesh alignment regularization for view interpolation. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2015. p. 2057–2065.

ZHANG, K.; LU, J.; LAFRUIT, G. Cross-based local stereo matching using orthogonal integral images. *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, v. 19, n. 7, p. 1073–1079, 2009.

APÊNDICE A – CÓDIGO DE RETIFICAÇÃO E CALIBRAÇÃO

```
#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include "opencv2/contrib/contrib.hpp"
#include <stdio.h>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
    int numBoards = atoi(argv[1]); //quantidade de imagens a ser consideradas. Aconselha-se que numBoards seja maior que 30
    int board_w = atoi(argv[2]);   //quantidade de bordas na horizontal do xadrez. Geralmente este parâmetro é 9
    int board_h = atoi(argv[3]);   //quantidade de bordas na vertical do xadrez.   Geralmente este parâmetro é 6

    Size board_sz = Size(board_w, board_h);
    int board_n = board_w*board_h;

    vector<vector<Point3f> > object_points;
    vector<vector<Point2f> > imagePoints1, imagePoints2;
    vector<Point2f> corners1, corners2;

    vector<Point3f> obj;
    for (int j=0; j<board_n; j++)
    {
        obj.push_back(Point3f(j/board_w, j%board_w, 0.0f));
    }

    Mat img1, img2, gray1, gray2;
    VideoCapture cap1 = VideoCapture(1);
    VideoCapture cap2 = VideoCapture(2);

    int success = 0, k = 0;
    bool found1 = false, found2 = false;

    while (success < numBoards) //obtendo todas as bordas de um conjunto de numBoards imagens de xadrez.
    {
        cap1 >> img1;
        cap2 >> img2;

        cvtColor(img1, gray1, CV_BGR2GRAY);
        cvtColor(img2, gray2, CV_BGR2GRAY);

        //funcao que detecta todas as bordas do xadrex da imagem 1
        found1 = findChessboardCorners(img1, board_sz, corners1, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
```

```
47          //funcao que detecta todas as bordas do xadrex da imagem 2
48          found2 = findChessboardCorners(img2, board_sz, corners2, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
49
50          //se encontrar todas as bordas desenhe na imagem todas elas
51          if (found1)
52          {
53              cornerSubPix(gray1, corners1, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
54              drawChessboardCorners(gray1, board_sz, corners1, found1);
55          }
56
57          //se encontrar todas as bordas desenhe na imagem todas elas
58          if (found2)
59          {
60              cornerSubPix(gray2, corners2, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
61              drawChessboardCorners(gray2, board_sz, corners2, found2);
62          }
63
64          imshow("image1", gray1);
65          imshow("image2", gray2);
66
67          k = waitKey(10);
68
69          //se for encontrado todas as bordas nas duas imagens então
70          //armazene no vetor de bordas, caso contrário não faça nada
71          if (found1 && found2)
72          {
73              k = waitKey(0);
74          }
75          if (k == 27)
76          {
77              break;
78          }
79          if (k == ' ' && found1 !=0 && found2 != 0)
80          {
81              imagePoints1.push_back(corners1); //armazene os pontos da imagem da esquerda
82              imagePoints2.push_back(corners2); //armazene os pontos da imagem da direita
83              object_points.push_back(obj);
84              printf ("Corners stored\n");
85              success++;
86
87              if (success >= numBoards)
88              {
89                  break;
90              }
91          }
92      }
```

```
93
94      destroyAllWindows();
95      printf("Starting Calibration\n");
96      Mat CM1 = Mat(3, 3, CV_64FC1);
97      Mat CM2 = Mat(3, 3, CV_64FC1);
98      Mat D1, D2;
99      Mat R, T, E, F;
100
101     //a partir dos pontos armazenados realize a calibração.
102     //A calibração vai definir os parametros CM1, CM2, D1, D2, R, T, E e F.
103     stereoCalibrate(object_points, imagePoints1, imagePoints2,
104                     CM1, D1, CM2, D2, img1.size(), R, T, E, F,
105                     cvTermCriteria(CV_TERMCRIT_ITER+CV_TERMCRIT_EPS, 100, 1e-5),
106                     CV_CALIB_SAME_FOCAL_LENGTH | CV_CALIB_ZERO_TANGENT_DIST);
107
108     FileStorage fs1("mystereocalib.yml", FileStorage::WRITE);
109     fs1 << "CM1" << CM1;
110     fs1 << "CM2" << CM2;
111     fs1 << "D1" << D1;
112     fs1 << "D2" << D2;
113     fs1 << "R" << R;
114     fs1 << "T" << T;
115     fs1 << "E" << E;
116     fs1 << "F" << F;
117
118     printf("Done Calibration\n");
119
120     printf("Starting Rectification\n");
121
122     Mat R1, R2, P1, P2, Q;
123
124     //a partir dos resultados da calibração execute a retifição.
125     //A retificação define os parâmetros de transformação R1, R2, P1, P2 e Q.
126     stereoRectify(CM1, D1, CM2, D2, img1.size(), R, T, R1, R2, P1, P2, Q);
127     fs1 << "R1" << R1;
128     fs1 << "R2" << R2;
129     fs1 << "P1" << P1;
130     fs1 << "P2" << P2;
131     fs1 << "Q" << Q;
132
133     printf("Done Rectification\n");
134
135     printf("Applying Undistort\n");
136
137     Mat map1x, map1y, map2x, map2y;
138     Mat imgU1, imgU2;
```

```
135     printf("Applying Undistort\n");
136
137     Mat map1x, map1y, map2x, map2y;
138     Mat imgU1, imgU2;
139
140     //Com os parâmetros de retificação é definido as funções de transformação e correção das imagens map1x, map1y, map2x, map2y.
141     //A partir destes parâmetros, os novos frames são corrigidos com a função remap.
142     initUndistortRectifyMap(CM1, D1, R1, P1, img1.size(), CV_32FC1, map1x, map1y);
143     initUndistortRectifyMap(CM2, D2, R2, P2, img2.size(), CV_32FC1, map2x, map2y);
144
145     printf("Undistort complete\n");
146
147     while(1)
148     {
149         cap1 >> img1;
150         cap2 >> img2;
151
152         remap(img1, imgU1, map1x, map1y, INTER_LINEAR, BORDER_CONSTANT, Scalar()); //função de alinhamento das novas imagens
153         remap(img2, imgU2, map2x, map2y, INTER_LINEAR, BORDER_CONSTANT, Scalar()); //função de alinhamento das novas imagens
154
155         imshow("image1", imgU1);
156         imshow("image2", imgU2);
157
158         ////////////////////////////////////////////////////////////////////////////////
159         //                                                                            //
160         //região destinado a implementação do código de                               //
161         //correspondência semiglobal estéreo                                          //
162         //utilizando as imagens alinhadas imgU1 e imgU2                               //
163         //                                                                            //
164         ////////////////////////////////////////////////////////////////////////////////
165
166         k = waitKey(5);
167
168         if(k==27)
169         {
170             break;
171         }
172     }
173
174     cap1.release();
175     cap2.release();
176
177     return(0);
178 }
```

APÊNDICE B – CÓDIGO DE COMUNICAÇÃO CPU E FPGA

```
225         //...
226         //...
227         //trecho principal do código para proposta didática.
228         //Trecho anterior de Código é usado para inicialização de variáveis e barramento
229         //e não são necessários para compreensão da abordagem proposta
230
231         cout << "Loading Calibration Data" << endl;
232
233         //carregando os parametros de calibração e retificação
234         FileStorage fs1("cameraartesanal.yml", FileStorage::READ);
235
236         fs1["CM1"] >> CM1;
237         fs1["CM2"] >> CM2;
238         fs1["D1"] >> D1;
239         fs1["D2"] >> D2;
240         fs1["R"] >> R;
241         fs1["T"] >> T;
242         fs1["E"] >> E;
243         fs1["F"] >> F;
244         fs1["roi1"] >> roi1;
245         fs1["roi2"] >> roi2;
246
247         printf("Loading Done\n");
248
249         printf("Loading Rectification Data\n");
250
251         fs1["R1"] >> R1;
252         fs1["R2"] >> R2;
253         fs1["P1"] >> P1;
254         fs1["P2"] >> P2;
255         fs1["Q"] >> Q;
256
257         printf("Loading Done\n");
258
259         fs1.release();
```

```
260
261         printf("Applying Undistort\n");
262
263         cv::Size imageSizeRect;
264         imageSizeRect.height = height;
265         imageSizeRect.width = width;
266
267         initUndistortRectifyMap(CM1, D1, R1, P1, imageSizeRect, CV_32FC1, map1x, map1y);
268         initUndistortRectifyMap(CM2, D2, R2, P2, imageSizeRect, CV_32FC1, map2x, map2y);
269
270         printf("Undistort complete\n");
271
272         //inicializacao do objeto OpenCV para o processamento SGM
273         //com a definicao dos seus valores de parametros
274         //o parametro mais importante é o espaco de disparidade sgbm.numberOfDisparities
275         StereoSGBM sgbm;
276         sgbm.preFilterCap = 127;
277         sgbm.SADWindowSize = 1;
278         sgbm.minDisparity = 0;
279         sgbm.numberOfDisparities = 64;
280         sgbm.P1 = 8 * 5 * sgbm.SADWindowSize * sgbm.SADWindowSize;
281         sgbm.P2 = 120 * sgbm.SADWindowSize * sgbm.SADWindowSize;
282         sgbm.uniquenessRatio = -1;
283         sgbm.speckleWindowSize = -1;
284         sgbm.speckleRange = -1;
285         sgbm.disp12MaxDiff = 255;
286         sgbm.fullDP = false;
287
288         while (1)
289         {
290
291             cap1 >> img1;
292             cap2 >> img2;
293
294             if (img1.empty() || img2.empty())
295                 break;
```

```
297         //retificando as imagens
298         remap(img1, imgU1, map1x, map1y, INTER_LINEAR, BORDER_CONSTANT, Scalar());
299         remap(img2, imgU2, map2x, map2y, INTER_LINEAR, BORDER_CONSTANT, Scalar()); //retificando as imagens
300
301         imgU1 = imgU1(roi1);
302         imgU2 = imgU2(roi2);
303
304         //transformando em escala de cinza
305         cvtColor(imgU1, g1, CV_BGR2GRAY);
306         cvtColor(imgU2, g2, CV_BGR2GRAY);
307
308         //ajustando a resolucao. A resolução original pode ser reduzido em um fator de escala definido como parametro
309         resize(g1,downg1,imageSize);
310         resize(g2,downg2,imageSize);
311
312         GetSystemTime(&start);
313
314         if(metodo == 0){
315             //inicio da abordagem com o opencv ////////////////////////////////////////////////////////////////////////////
316             sgbm(downg1, downg2, result);                                                                          /////////
317             result.convertTo(result, CV_8U, 1.0 / 8);                                                              /////////
318             //fim da abordagem com o opencv ///////////////////////////////////////////////////////////////////////////////
319         }
320         else {
321             //inicio da abordagem proposta ////////////////////////////////////////////////////////////////////////////
322             //aplicando o filtro de sobel-x                                                                        /////////
323             Sobel(downg1, g1_grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT );                              /////////
324             Sobel(downg2, g2_grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT );                              /////////
325             convertScaleAbs( g1_grad_x, g1_abs_grad_x );                                                           /////////
326             convertScaleAbs( g2_grad_x, g2_abs_grad_x );                                                           /////////
327                                                                                                                    /////////
328             //funcao de empacotamento.                                                                             /////////
329             //Converte do tipo Mat "g2_abs_grad_x" e "g1_abs_grad_x"                                               /////////
330             //para um vetor de inteiros "sendBuffer" com os dados já empacotados                                   /////////
331             convertFromTwoMatToUInt(g2_abs_grad_x, g1_abs_grad_x, sendBuffer, imageSize.height, imageSize.width); /////////
332                                                                                                                    /////////
```

```
333             //envia os dados para o barramento                                                    /////////
334             //numWords indica quantos palavras de 32 bits serão enviadas                          /////////
335             //neste caso numWords = (altura x largura)                                            /////////
336             sent = fpga_send(fpga, chnl, sendBuffer, numWords, 0, 1, 25000);                      /////////
337                                                                                                   /////////
338             //verifica se o envio ocorreu normalmente                                             /////////
339             if (sent != 0) {                                                                      /////////
340                                                                                                   /////////
341                 //se sim, recebe os dados a partir do barramento                                  /////////
342                 recvd = fpga_recv(fpga, chnl, recvBuffer, numWords, 25000);                       /////////
343             }                                                                                     /////////
344                                                                                                   /////////
345             //funcao de desempacotamento.                                                         /////////
346             //Converte o vetor de inteiros "recvBuffer" para o tipo Mat "result" com os dados já empacotados  /////////
347             convertFromTwoUIntToMat(recvBuffer, result, numWords, imageSize.width, pixelIndexInRow);  /////////
348             result.convertTo(result, CV_8U, 4);                                                   /////////
349             //fim da abordagem proposta ////////////////////////////////////////////////////////////////////////////////
350         }
351
352         GetSystemTime(&end);
353
354
355         //o restante do código é para visualizar a resposta
356         //e calcular o tempo fps para visualizar o mapa de disparidade
357         resize(result,result2,imageSizeRect);
358
359         msec = (end.wMilliseconds - start.wMilliseconds) + (end.wSecond - start.wSecond) * 1000 + 60 * 1000 * (end.wMinute -
360         fps = 1.0 / (msec / 1000);
361
362         croppedImage = result2(myROI);
363
364         sprintf(label, "FPS:%f %fms", fps, msec);
365         putText(croppedImage,String(label),Point(20,20),FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255,255,255),2);
```

```
367             imshow("result", result);
368             imshow("result2", result2);
369             imshow("result3", croppedImage);
370 
371             imshow("left", img1);
372             imshow("right", img2);
373 
374             waitKey(1);
375         }
376     }
377 
378     return 0;
379 }
380 
381 void convertFromTwoMatToUInt(Mat &dataIn1, Mat &dataIn2, unsigned int *dataOut, int height, int width){
382     int i, j;
383     int counter = 0;
384 
385     for(i = 0; i < height; i++){
386         for(j = 0; j < width; j+=2){
387 
388             dataOut[counter] = (((unsigned int)(dataIn1.at<uchar>(i,j+1)) << 24)&0xFF000000) +
389                                (((unsigned int)(dataIn2.at<uchar>(i,j+1) << 16))&0x00FF0000) +
390                                (((unsigned int)(dataIn1.at<uchar>(i,j) << 8))&0x0000FF00) +
391                                ((unsigned int)(dataIn2.at<uchar>(i,j))&0x000000FF);
392             counter++;
393         }
394     }
395 }
396 
397 void convertFromTwoUIntToMat(unsigned int *dataI, Mat &dataOut1,
398                              int totalNumberOfPixel, int width, unsigned int *pixelIndexInRow){
399     int i;
400     int rowIndex1;
401     int disparityValue1;
402     int rowIndex2;
403     int disparityValue2;
404     int counter = 0;
405     for(i = 0; i < totalNumberOfPixel; i+=2){
406         rowIndex1 = dataI[counter] & 0x000000FF;
407         disparityValue1 = ((dataI[counter] >> 8) & 0x0000003F) ;
408 
409         rowIndex2 = (dataI[counter] >> 16) & 0x000000FF;
410         disparityValue2 = ((dataI[counter] >> (8 + 16)) & 0x0000003F) ;
411 
412         dataOut1.at<uchar>(rowIndex1,pixelIndexInRow[rowIndex1]) = disparityValue1;
413         dataOut1.at<uchar>(rowIndex2,pixelIndexInRow[rowIndex2]) = disparityValue2;
414 
415         pixelIndexInRow[rowIndex1] = (pixelIndexInRow[rowIndex1]+1)%width;
416         pixelIndexInRow[rowIndex2] = (pixelIndexInRow[rowIndex2]+1)%width;
417 
418         counter++;
419     }
420 }
```