UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
MESTRADO EM INFORMÁTICA

CAMILO ALVES CARVALHO

Modelagem Paralela em C+CUDA de Sistema Neural de Visão Estereoscópica

Vitória – ES
2009

CAMILO ALVES CARVALHO

Modelagem Paralela em C+CUDA de Sistema Neural de Visão Estereoscópica

Dissertação apresentada ao Mestrado de Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Dr. Alberto Ferreira De Souza.

Vitória – ES
2009

CAMILO ALVES CARVALHO

Modelagem Paralela em C+CUDA de Sistema Neural de Visão Estereoscópica

Dissertação apresentada ao Mestrado de Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Mestre em Informática.

Aprovada em 31 de agosto de 2009.

COMISSÃO EXAMINADORA

Prof. Dr. Aberto Ferreira de Souza
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Elias de Oliveira
Universidade Federal do Espírito Santo

Prof. Dr. Cláudio Luiz de Amorim
Universidade Federal do Rio de Janeiro

Vitória – ES
2009

Para minha família.

AGRADECIMENTOS

Agradeço aos meus pais, José e Zenilde, e à minha irmã Ana Amália, por representarem os pilares dos meus valores pessoais, e modelos nos quais me inspiro para continuar evoluindo enquanto indivíduo.

Agradeço ao professor Alberto Ferreira De Souza, por sua amizade e orientação, além da paciência e incentivo incansáveis, que tornaram possível o desenvolvimento deste trabalho.

Agradeço aos professores Elias de Oliveira e Claudio Amorim por terem gentilmente aceitado participar de minha avaliação, mesmo que convidados com pouca antecedência.

A todos os meus amigos do LCAD, dentre os quais Bruno Zanetti, Renderson Loriatto e Dijalma Fardim, além de Felipe Pedroni (o primeiro amigo que fiz na UFES) e, em especial, Lucas Veronese, cuja ajuda foi determinante para os sucessos alcançados.

Também aos meus amigos do Mestrado, dentre eles Vinícius Freitas, Vitor Campana, Bernard Pereira, Márcia Oliveira, Eduardo Monteiro e, em especial, Francisco Santiago, por suas amizades neste período em que estive em Vitória.

Aos professores e funcionários do Departamento de Informática da UFES, pelo bom trabalho e a boa convivência no decorrer do curso. Também à CAPES pela bolsa recebida no início de meu curso de mestrado.

Por fim, gostaria de agradecer aos vários amigos de casa, a Bahia, dentre eles Wilton Lacerda, Stenio Araújo, Fabyano Gomes, Bruno Boaventura e Joaquim Júnior, pela amizade e incentivo, e por sempre me cobrarem “terminar ligeiro e voltar logo pra casa”.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

As imagens projetadas em nossas retinas são bidimensionais; entretanto, a partir delas, o nosso cérebro é capaz de sintetizar uma representação 3D com a cor, forma e informações de profundidade sobre os objetos ao redor no ambiente. Para isso, após a escolha de um ponto no espaço 3D, os nossos olhos vergem em direção a este ponto e, ao mesmo tempo, o sistema visual é realimentado com informações sobre o posicionamento dos olhos, interpretando-as como a distância deste ponto ao observador. A percepção de profundidade ao redor do ponto de vergência é obtida utilizando-se a disparidade entre as imagens direita e esquerda, ou seja, a diferença entre as posições, nas retinas, das duas projeções de um determinado ponto no espaço 3D causada pela separação horizontal dos olhos. A maior parte do processamento da percepção da profundidade é feita no córtex visual, principalmente na área primária (V1) e temporal medial (MT). Neste trabalho, foi desenvolvida uma implementação em C+CUDA de um modelo, criado na UFES, da arquitetura neural dos córtices V1 e MT que usa como base modelos anteriores de células corticais e mapeamento log-polar. A implementação seqüencial deste modelo é capaz de construir uma representação tridimensional do mundo externo por meio de pares de imagens estereoscópicas obtidas a partir de um par de câmeras fronto-paralelas. Nossa implementação paralela em C+CUDA é quase 60 vezes mais rápida que a seqüencial e permite a reconstrução 3D em tempo real.

ABSTRACT

The images formed on our retinae are bidimensional; however, from them our brain is capable of synthesizing a 3D representation with color, shape and depth information about the objects in the surrounding environment. For that, after choosing a point in 3D space, our eyes verge to this point and, at the same time, the visual system is fed back with the eyes position information, interpreting it as the distance of this point to the observer. Depth perception around the vergence point is obtained using visual disparity, i.e., the difference between the positions in the retinae of the two projections of a given point in 3D space caused by the horizontal separation of the eyes. Most of the depth perception processing is done in the visual cortex, mainly in the primary (V1) and medial temporal (MT) areas. In this work, we developed a parallel implementation in C+CUDA of model, built at UFES, of the neural architecture of the V1 and MT cortices that uses as building blocks previous models of cortical cells and log-polar mapping. A sequential implementation of this model can create tridimensional representations of the external world using stereoscopic image pairs obtained from a pair of fronto-parallel cameras. Our C+CUDA parallel implementation is almost 60 times faster and allows real-time 3D reconstruction.

1. INTRODUÇÃO

Tal como câmeras, os olhos humanos captam as imagens do ambiente num formato bidimensional. De posse desta informação, o cérebro é capaz de construir uma representação tridimensional deste ambiente, inferindo a profundidade dos objetos. A distância do observador a um ponto específico é obtida por meio das informações fornecidas pelo sistema óculo-motor, responsável pela movimentação dos olhos. As diferenças em seu posicionamento, oriundas do processo de vergência, em que os olhos se direcionam para um ponto fixo no espaço tridimensional, são usadas para o cálculo da distância deste ponto ao observador. A noção de profundidade ao redor do ponto de vergência pode ser obtida calculando-se a disparidade, ou seja, a diferença de posição entre pontos correspondentes das imagens projetadas nas retinas, causada pela separação horizontal dos olhos na cabeça. No cérebro humano, o cômputo das disparidades é realizado no córtex visual, mais precisamente nas áreas V1 e MT.

Uma modelagem matemático-computacional de uma arquitetura neural representando as áreas corticais V1 e MT, baseada em filtros de Gabor e no mapeamento log-polar da retina para o córtex, foi proposta por Oliveira [OLI05]. Esta implementação computacional é capaz de criar, a partir de duas imagens bidimensionais, uma representação interna tridimensional do mundo externo. Para tanto, é construído um mapa das disparidades entre as imagens esquerda e direita e, a partir deste mapa, é possível calcular a posição de pontos dos objetos no espaço tridimensional. Desta forma, é possível verificar se a representação interna ao computador está condizente com as informações de forma e profundidade presentes na cena original.

Como parte do trabalho de Oliveira [OLI05] foi desenvolvida uma arquitetura de *hardware* e *software*, que contava com um robô equipado com câmeras ajustadas segundo um modelo semelhante ao sistema óptico humano. Uma vez alcançado este resultado, ou seja, viabilizada a obtenção de imagens tridimensionais a partir de imagens bidimensionais por meio do processo desenvolvido em trabalhos anteriores [OLI05, KOM02], um passo importante para se avançar nesta área de pesquisa é obter um desempenho o mais próximo possível do sistema biológico. Mas, para isso,

é necessário contornar alguns obstáculos técnicos, principalmente os relativos à capacidade de processamento das plataformas de *hardware*.

A construção de mecanismos artificiais que reproduzam funções desempenhadas pelo organismo humano demanda melhorias de desempenho (otimização) suficientes para que se tenha um tempo de resposta aceitável. Para tanto, estratégias de desenvolvimento podem ser adotadas para se incrementar este desempenho, principalmente levando-se em conta o atual cenário tecnológico da indústria de microprocessadores, que tem voltado seu investimento para o desenvolvimento de chips de múltiplos núcleos. Outra tendência atual da indústria é a de desenvolver chips de processamento gráfico (*Graphics Processing Units* – GPUs) cada vez mais poderosos, que hoje superam, em capacidade bruta de processamento, os próprios microprocessadores. Possuem um propósito mais específico, porém suas arquiteturas permitem também a execução de trechos de código de programas de propósito geral (*General Purpose Computation on GPUs* – GPGPU) [LUE04]. O desempenho destas GPUs é por vezes centenas de vezes maior que os alcançados por microprocessadores, ou *Central Processing Units* (CPUs) [LUE08].

Tendo em vista este novo paradigma de desenvolvimento de aplicações de alto desempenho, buscou-se, neste trabalho, utilizar a tecnologia de programação paralela de GPUs conhecida como *Compute Unified Device Architecture* (CUDA), disponibilizada pela NVidia em 2006 [HAL08]. CUDA é uma extensão da linguagem C e programas em C+CUDA permitem o acesso direto aos recursos das GPUs, possibilitando, assim, um novo patamar de desempenho de certas aplicações – desempenhos duas ou mesmo três ordens de grandeza superiores àqueles observados em CPUs tem sido obtidos com GPUs programadas em C+CUDA (www.nvidia.com/cudazone).

1.1. MOTIVAÇÃO

A disponibilidade de poder computacional oferecida pelas novas GPUs e as necessidades de pesquisa do Grupo de Pesquisa em Ciência da Cognição (GPCG) do Programa de Pós-Graduação em Informática (PPGI) da UFES associadas à visão artificial 3D foram as principais motivações para a realização deste trabalho. Outra motivação importante foi a oportunidade do desenvolvimento de uma aplicação

relevante de processamento de alto desempenho na área de ciência da cognição – o GPCG é correntemente abrigado, no PPGI, pelo Laboratório de Computação de Alto Desempenho (LCAD) e este trabalho, portanto, permitiria aumentar a sinergia entre as áreas de Ciência da Cognição e Computação de Alto Desempenho do PPGI.

O modelo biológico empregado no trabalho de Oliveira [OLI05] dá prosseguimento à linha de pesquisa do GPCG, que tem por objetivo entender o funcionamento do cérebro em suas atividades associadas à visão. A percepção de profundidade pode ser modelada por uma série de métodos (ver Seção 9.1), que inclusive revelam ter vantagens em relação ao tempo de execução, porém não possuem tal inspiração biológica.

Desenvolvida para ser executada de maneira seqüencial, a aplicação desenvolvida por Oliveira [OLI05] é eficaz, mas peca na eficiência, caso seja levada em conta à busca pelo desempenho próximo ao sistema biológico. Uma vez que as GPUs têm alcançado números superlativos comparados às CPUs, obter resultados significativos no desenvolvimento de estratégias de implementação paralela de código representa avanços nas metas das pesquisas em andamento. Este trabalho está inserido numa linha de pesquisa do PPGI cujo objetivo de longo prazo é implementar uma arquitetura neural artificial com capacidades semelhantes, ou até mesmo maiores, que a capacidade humana.

Alcançar resultados que permitam, por exemplo, uma execução em tempo-real de modelos da visão biológica, ou muito próximos do sistema biológico, tornaria possível a utilização destes modelos em aplicações como robótica, uma outra área de pesquisa desenvolvida pelo PPGI. Neste sentido, uma plataforma robótica, desenvolvida utilizando conceitos de robótica probabilística, está sendo desenvolvida, e suas capacidades de localização espacial, mapeamento e descoberta, podem ser ampliadas com os resultados deste trabalho, uma vez que a plataforma robótica desenvolvida já possui, acoplada em sua estrutura, câmeras esteroscópicas.

1.2. OBJETIVOS

O principal objetivo deste trabalho foi reimplementar um modelo matemático-computacional dos sistemas neurais humanos associados à percepção da profundidade desenvolvido por Oliveira [OLI05], empregando técnicas de

programação paralela, de maneira a se obter maior desempenho em termos de tempo.

Outro objetivo importante foi avançar nas linhas de pesquisa do PPGI associadas à simulação dos sentidos humanos por meio de modelos matemático-computacionais, mais especificamente, avançar na modelagem do sentido da visão, e avançar o estado da arte em implementação de programas paralelos em C+CUDA.

1.3. CONTRIBUIÇÕES

As principais contribuições deste trabalho foram:

- A reimplementação paralela em C+CUDA do modelo matemático-computacional dos sistemas neurais humanos associados à percepção da profundidade desenvolvidos por Oliveira [OLI05];
- O desenvolvimento de técnicas incrementais de desenvolvimento de aplicações explorando o poder de processamento de GPUs.

Neste trabalho de pesquisa foi reconstruída a aplicação desenvolvida por Oliveira [OLI05] em que se obtêm objetos tridimensionais a partir de imagens estéreo, emulando a arquitetura neural do córtex visual humano. Os resultados das execuções desta aplicação podem ser conferidos visualmente, mas o tempo de execução ainda era alto considerando o objetivo de se alcançar o desempenho do cérebro humano.

O estudo do gargalo associado à disponibilidade de recursos computacionais para o processamento necessário para esta reconstrução foi o alvo da pesquisa e, para contorná-lo, foram aplicadas técnicas de programação paralela em C+CUDA que permitiram explorar o poder de processamento de GPUs modernas. O domínio de tais técnicas amplia o leque de possibilidades de desenvolvimento de aplicações que demandam processamento de alto desempenho.

2. SISTEMA VISUAL HUMANO

Neste capítulo é descrito sumariamente o sistema visual humano. Ele apresenta conceitos e termos que são essenciais para compreender as contribuições deste trabalho. Seu conteúdo foi, fundamentalmente, extraído do trabalho de Oliveira [OLI05].

2.1. O OLHO

O globo ocular, com cerca de 25 milímetros de diâmetro, é o responsável pela captação da luz refletida pelos objetos. Anatomicamente, o globo ocular fica alojado em uma cavidade formada por vários ossos chamada órbita e é constituído por três túnicas (camadas): túnica fibrosa externa, túnica intermédia vascular pigmentada e túnica interna nervosa.

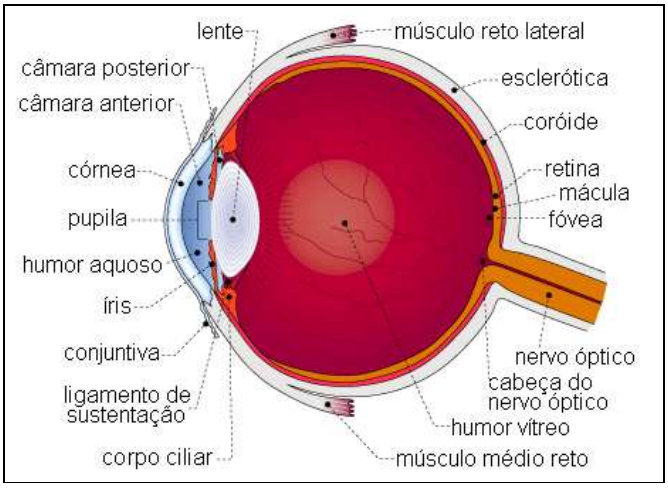

Figura 2-1 – Anatomia do olho humano. Corte medial horizontal do olho direito visto de cima. Figura retirada de http://www.escolavesper.com.br/olho_humano.htm

Na Figura 2-1 estão representadas todas as partes que formam as túnicas fibrosa externa, intermédia vascular pigmentada e a túnica interna nervosa. A túnica fibrosa externa ou esclerótica, também chamada de “branco do olho”, tem uma função protetora. É resistente, de tecido fibroso e elástico, e envolve externamente o olho (globo ocular). A maior parte da esclerótica é opaca e chama-se esclera. A ela

estão conectados os músculos extra-oculares que movem os globos oculares, dirigindo-os ao seu objetivo visual. A parte anterior da esclerótica chama-se córnea, que é transparente e atua como uma lente convergente.

A túnica intermédia vascular pigmentada ou úvea compreende a coróide, o corpo ciliar e a íris. A coróide está situada abaixo da esclerótica e é bastante pigmentada para absorver a luz que chega a retina, evitando sua reflexão dentro do olho. Ela é intensamente vascularizada e tem também como função nutrir a retina. A íris é uma estrutura muscular de cor variável (parte circular que dá cor aos olhos), é opaca e tem uma abertura central, chamada pupila, por onde a luz passa. O diâmetro da pupila varia, aproximadamente de 2mm a 8mm, de acordo com a intensidade luminosa do ambiente. Em ambientes claros a pupila se estreita, diminuindo a passagem de luz, evitando a saturação das células detectoras de luz da retina. No escuro a pupila se dilata, aumentando a passagem de luz e sua captação pela retina. A luz que passa pela pupila atinge imediatamente o cristalino, uma lente gelatinosa que focaliza os raios luminosos sobre a retina.

O corpo ciliar é uma estrutura formada por musculatura lisa e que envolve o cristalino (a lente do olho). Ele á capaz de mudar a forma do cristalino permitindo assim ajustar a visão para objetos próximos ou distantes. Este processo é conhecido como acomodação visual. A convergência correta do cristalino faz com que a imagem seja projetada nitidamente na retina. Se a imagem for maior ou menor que a necessária, fica fora de foco. Se o cristalino está ajustado para uma certa distância de um objeto, e este objeto se aproxima, a imagem perde a nitidez. Para recuperá-la, o corpo ciliar aumenta a convergência do cristalino, acomodando-o, diminuindo a distância focal. Caso o objeto se afaste, ocorre o processo inverso. Este processo está ilustrado na Figura 2-2.

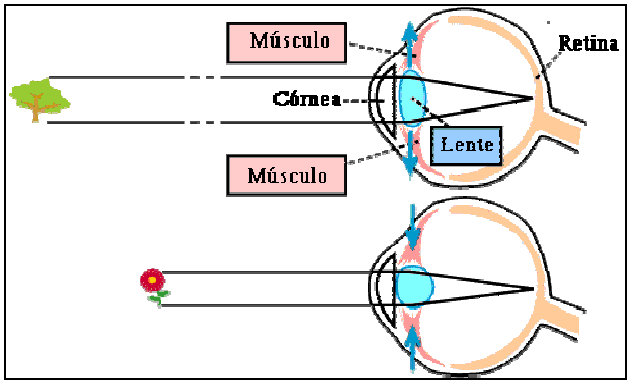

Figura 2-2 – Figura mostrando a acomodação visual do cristalino.

A túnica interna nervosa é a retina. É na retina que se formam as imagens visualizadas. A imagem projetada na retina é invertida, mas isto não causa nenhum problema já que o cérebro se adapta a isto desde o nascimento. Para que a imagem seja projetada na retina, a luz percorre o seguinte caminho: primeiramente a luz atinge a córnea, que conforme visto anteriormente é um tecido transparente, passa pela pupila, que é a abertura situada na íris que regula a intensidade de luz que entra no olho, atravessa o cristalino, que é a lente gelatinosa e que tem a função de focalizar a imagem na retina, atravessa um fluido viscoso chamado humor vítreo, que preenche a região entre o cristalino e a retina, e finalmente atinge a retina.

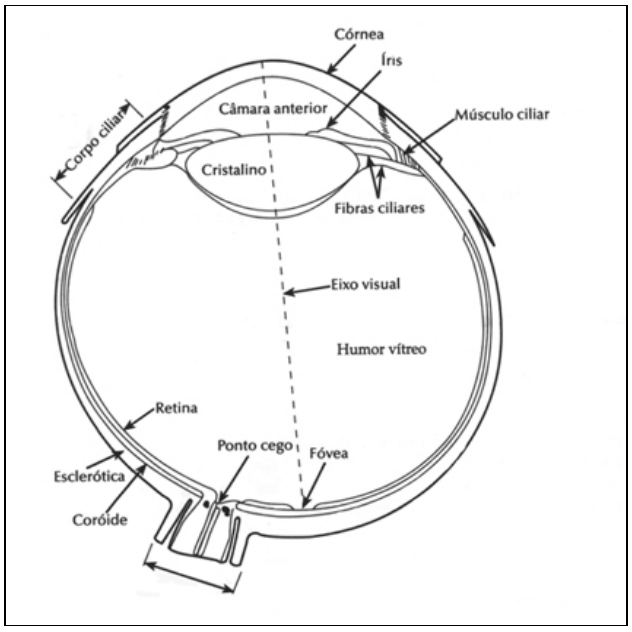

Figura 2-3 – Eixo visual. Figura retirada de http://www.on.br/glossario/alfabeto/o/olho_humano.html

A retina é composta por mais de 100 milhões de células fotossensíveis: cerca de 7 milhões de cones e entre 75 milhões e 150 milhões de bastonetes. Estas células, quando excitadas pela energia luminosa, estimulam as células nervosas adjacentes, gerando um impulso nervoso que é propagado pelo nervo óptico. A imagem fornecida pelos cones é mais nítida e rica em detalhes. Os cones são sensíveis às cores. Há três tipos de cones: um que se excita com luz vermelha, outro com luz verde e outro com luz azul. Os bastonetes não têm poder de resolução visual tão bom nem conseguem detectar cores, mas são mais sensíveis à luz. Em situações de pouca luminosidade a visão passa a depender exclusivamente dos bastonetes.

As imagens dos objetos visualizados diretamente são projetadas normalmente numa região da retina chamada *fovea centralis* ou simplesmente fóvea com cerca de 1,5 mm de diâmetro e que fica na direção da linha (eixo visual) que passa pela córnea, pupila e pelo centro do cristalino (Figura 2-3). Os cones são encontrados na retina central, em um raio de aproximadamente 10 graus a partir da fóvea. Os bastonetes estão localizados principalmente na retina periférica. O local da retina de onde sai o nervo óptico não possui cones nem bastonetes. Este local é chamado de ponto cego porque uma imagem que forme sobre este ponto, não é vista.

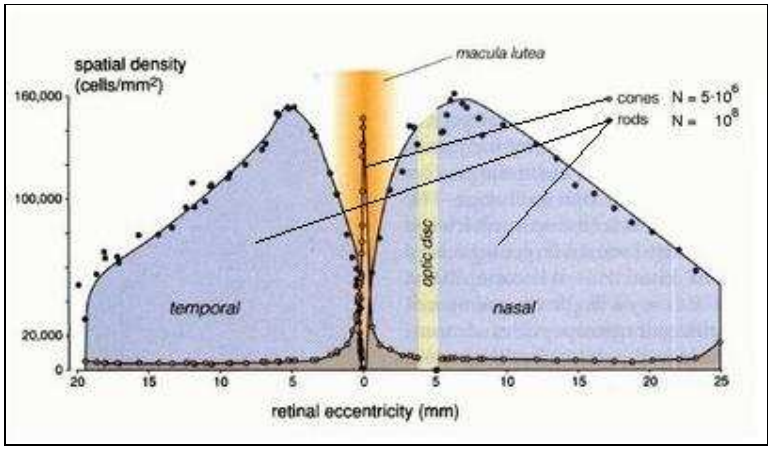

Figura 2-4 – Distribuição de cones e bastonetes (*rods*) na retina. A *macula lutea* (região da fóvea e vizinhanças) possui alta densidade de cones, enquanto que os bastonetes se concentram na periferia. O ponto cego fica na região do disco óptico (*optic disc*). Figura retirada de http://www.brainworks.uni-freiburg.de/group/wac

Os neurônios de saída da retina são as células ganglionares que projetam seus axônios através do nervo óptico, levando a informação visual para o cérebro. Cada célula ganglionar recebe informações de um conjunto de células fotorreceptoras vizinhas em uma área circunscrita na retina que é o seu **campo receptivo**. Duas características importantes podem ser percebidas nos campos receptivos das células ganglionares. Primeiro, são aproximadamente circulares. Segundo, são divididos em duas partes: um círculo central e um anel periférico.

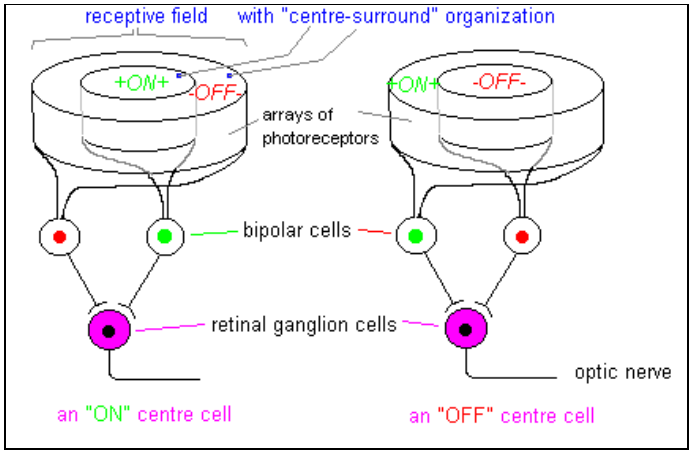

Figura 2-5 – Campo receptivo das células ganglionares. Figura retirada de http://www.cf.ac.uk/biosi/staff/jacob/teaching/sensory/vision.html

As células ganglionares respondem bem a uma iluminação diferencial entre o centro e a periferia dos seus campos receptivos. Assim, é possível identificar dois tipos de células ganglionares: *on center* e *off center*. Células ganglionares com campos receptivos *on center* ficam excitadas quando a luz estimula o centro e ficam inibidas quando a luz estimula o contorno do campo receptivo. Células *off center*, funcionam ao contrário, ficam excitadas quando a luz estimula a periferia e ficam inibidas quando a luz estimula o centro do campo receptivo. Os sinais visuais de intensidade luminosa (na verdade, de contraste), depois das transformações feitas na retina, são levados até o cérebro pelo nervo ótico.

2.2. FLUXO DE INFORMAÇÕES VISUAIS

Nesta seção será descrito o fluxo de informações visuais em dois estágios: primeiro, a informação visual saindo da retina e indo para o mesencéfalo e tálamo

(Figura 2-8), e depois, a informação saindo do tálamo para o córtex visual primário, conforme indicado na Figura 2-6. Para tanto, serão definidos alguns conceitos a seguir.

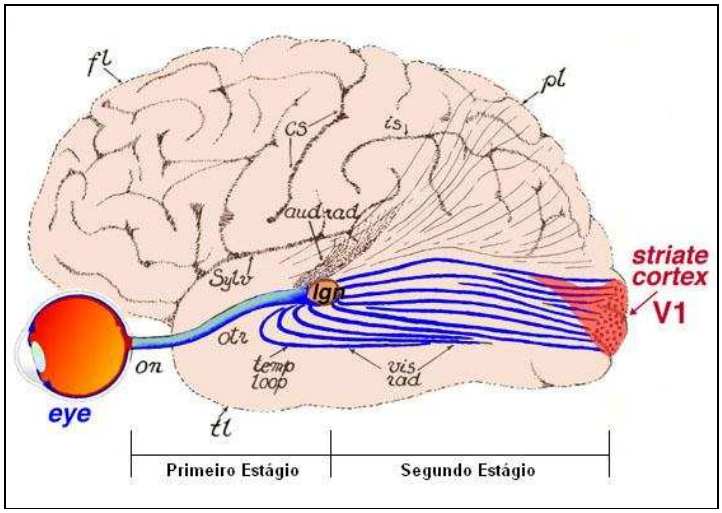

Figura 2-6 – Fluxo das informações visuais. Figura retirada de http://webvision.med.utah.edu/VisualCortex.html e alterada com iserção dos estágios.

A retina pode ser dividida em duas partes: hemirretina nasal e hemirretina temporal, cuja separação é uma linha imaginária que corta o olho de cima a baixo passando pela fóvea. Numa situação em que as fóveas de ambos os olhos estão fixas num ponto do espaço situado em linha reta com o nariz, é possível dividir o campo visual em *left hemifield* (campo visual esquerdo) à esquerda do ponto fixo no espaço e *right hemifield* (campo visual direito) à direita do ponto fixo no espaço. O *left hemifield* é projetado na hemirretina nasal do olho esquerdo e na hemirretina temporal do olho direito. O *right hemifield* é projetado na hemirretina nasal do olho direito e na hemirretina temporal do olho esquerdo, conforme mostrado na Figura 2-7.

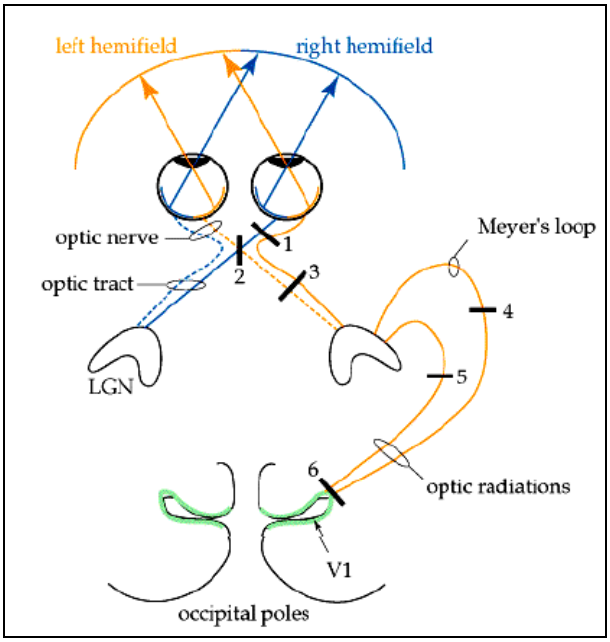

Figura 2-7 – Campo visual. 1) Nervo óptico, 2) Quiasma óptico, 3) Trato óptico. Figura retirada de http://thalamus.wustl.edu/course/basvis.html

O nervo óptico de cada olho projeta-se para o quiasma óptico que é região onde é feita a separação das fibras de cada olho, em tratos (feixes de axônios) ópticos, destinadas para um mesmo lado do cérebro. Os tratos ópticos se projetam cada uma para três áreas subcorticais simétricas (que existem nos dois lados de cérebro): região pretectal ou *pretectum*, o *superior culliculus* do mesencéfalo e o *lateral geniculate nucleus* (LGN) do tálamo conforme mostra a Figura 2-8.

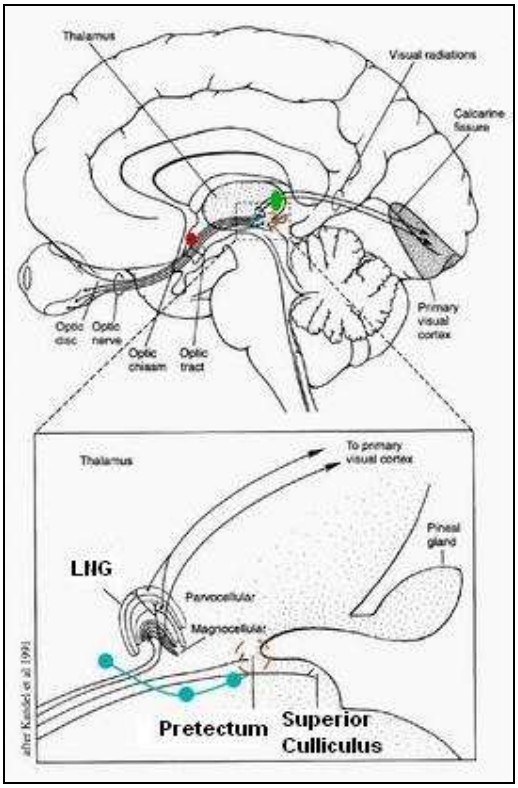

Figura 2-8 – Projeções da retina no mesencéfalo. Figura retirada de http://www.brainworks.uni-freiburg.de/group/wachtler/VisualSystem/

A região pretectal do mesencéfalo possui células que se projetam bilateralmente para os neurônios do sistema simpático/parassimpático que controla os reflexos pupilares, contraindo e dilatando a pupila de acordo com a quantidade de luz que incide nos olhos. O *superior culliculus* é uma estrutura de camadas alternantes cinzentas e brancas localizada no teto do mesencéfalo. As células das camadas superficiais projetam-se para uma vasta área do córtex cerebral, formando uma via indireta da retina para o córtex cerebral. As camadas superficiais também recebem sinais provenientes do córtex visual enquanto que as camadas mais profundas recebem projeções de várias outras áreas do córtex ligadas a outros sentidos. O *superior culliculus* possui um mapeamento visual além de responder a estímulos auditivos e somatossensórios.

Células das camadas mais profundas do *superior culliculus* respondem positivamente antes dos movimentos sacádicos dos olhos, no qual os olhos trocam rapidamente de um ponto de fixação para outro numa cena. Estas células formam um mapa de movimento sacádico ordenado com o mapa visual. Para controlar os

movimentos sacádicos, o *superior culliculus* recebe informações não só da retina, mas também do córtex cerebral.

O *lateral geniculate nucleus* (LGN) é o ponto de retransmissão das informações visuais provenientes da retina para o córtex visual. Cerca de 90% dos axônios da retina chegam até o LGN, que possui uma representação retinotópica da metade contralateral (lado oposto) do campo visual.

A razão entre uma área do LGN e uma área correspondente da retina que representa um grau do campo visual é chamada de fator de magnificação daquela área do LGN. A fóvea possui uma representação relativamente maior que a retina periférica no LGN, ou seja, as regiões do LGN que monitoram a fóvea possuem um maior fator de magnificação.

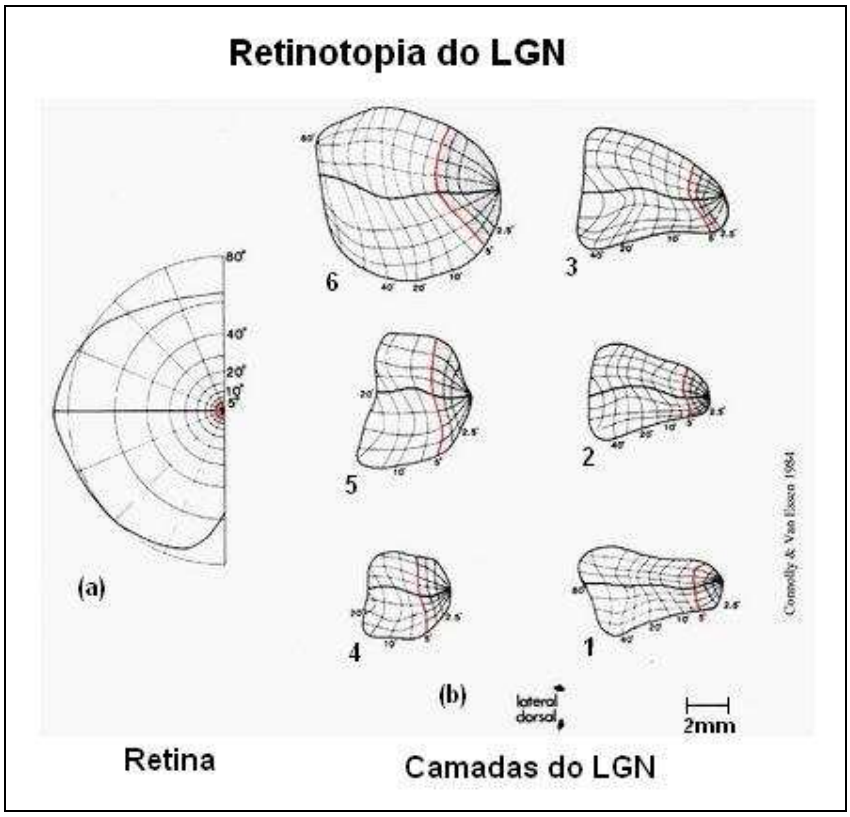

Figura 2-9 – Retinotopia do LGN. (a) Mapeamento da retina. (b) Mapeamento da retina nas camadas 1 à 6 do LGN. Figura retirada e adaptada de http://www.brainworks.uni-freiburg.de/group/wachtler/VisualSystem/.

Nos primatas, incluindo os humanos, o LGN é formado por seis camadas numeradas de 1 (ventral) a 6 (dorsal). As duas camadas mais ventrais (camadas 1 e 2) contém células relativamente grande que recebem conexões das células

ganglionares M da retina e são conhecidas como camadas magnocelulares enquanto que as outras 4 camadas dorsais (camadas 3, 4, 5 e 6) contém células que recebem conexões das células ganglionares P da retina e são conhecidas como camadas parvocelulares. A Figura 2-9 mostra de forma esquemática o mapeamento da retina nas diversas camadas do LGN. Nela é possível ver como o fator de magnificação varia da fóvea para a periferia no LGN.

Todas as camadas do LGN possuem células com campo receptivo *on center* e *off center*, sendo que cada camada recebe sinais somente de um olho. As fibras da hemirretina nasal contralateral (outro lado) são projetadas nas camadas 1, 4 e 6, enquanto que as fibras da hemirretina temporal ipsilateral (mesmo lado) são projetadas nas camadas 2, 3 e 5 conforme mostrado na Figura 2-10.

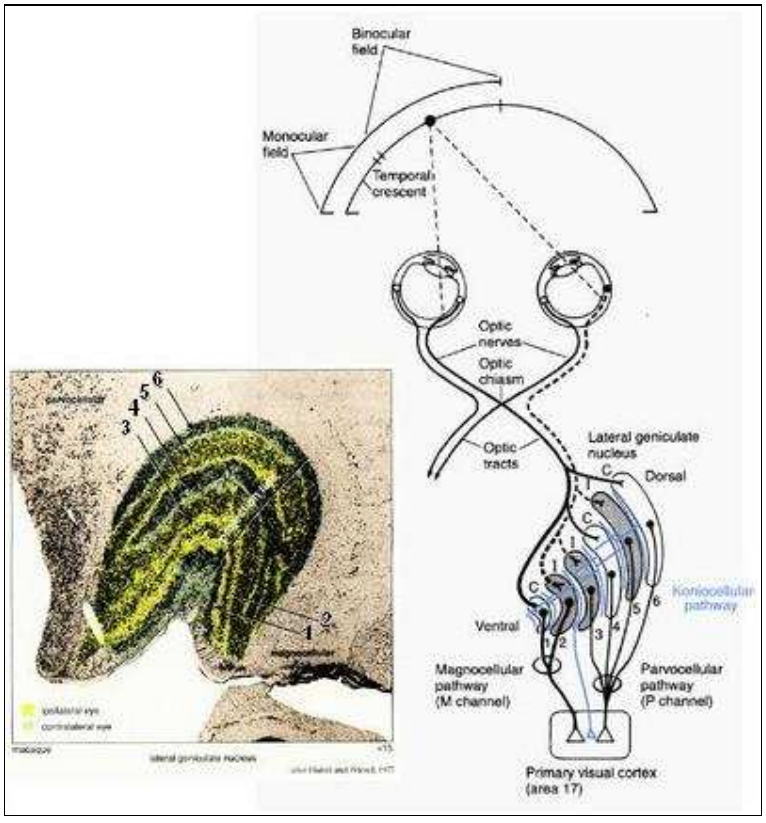

Figura 2-10 – LGN. Figura retirada de http://www.brainworks.uni-freiburg.de/group/wachtler/VisualSystem/

As células das camadas magnocelulares e parvocelulares do LGN projetam-se para o córtex visual formando duas vias independentes (vias M e P) que se estendem desde a retina até o córtex visual primário. A via P é essencial para a

visão de cores e sensível a estímulos de alta freqüência espacial e baixa freqüência temporal da imagem na retina. A via M é mais sensível a estímulos de baixa freqüência espacial e alta freqüência temporal.

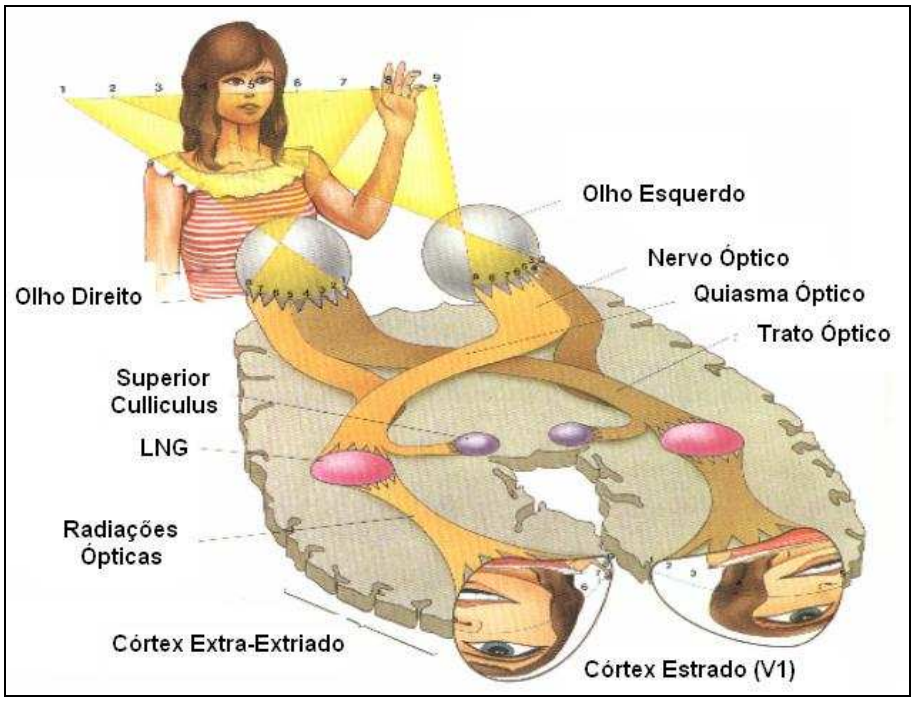

Figura 2-11 – Exemplo ilustrando o fluxo de informações visuais da retina ao córtex estriado. Figura retirada de http://www.brainworks.uni-freiburg.de/group/wachtler/VisualSystem/

2.3. ORGANIZAÇÃO DO CÓRTEX VISUAL

A informação visual é processada em diversas áreas corticais, sendo que cada uma delas contribui diferencialmente para o processamento da percepção de movimento, profundidade, forma e cor. Aqui serão descritas brevemente cinco áreas corticais visuais mais diretamente ligadas a este trabalho: V1, V2, V3, V4 e MT (também conhecida como V5). Na Figura 2-12-A é apresentada uma vista lateral de um hemisfério do cérebro de um macaco e na Figura 2-12-B é mostrado este hemisfério estendido de modo a formar um plano, onde são indicadas com uma tonalidade mais escura as áreas corticais visuais e são apontadas as áreas V1, V2, V3, V4 e MT. Como é possível observar na Figura 2-12-A, as áreas corticais visuais ocupam aproximadamente metade do córtex de um macaco.

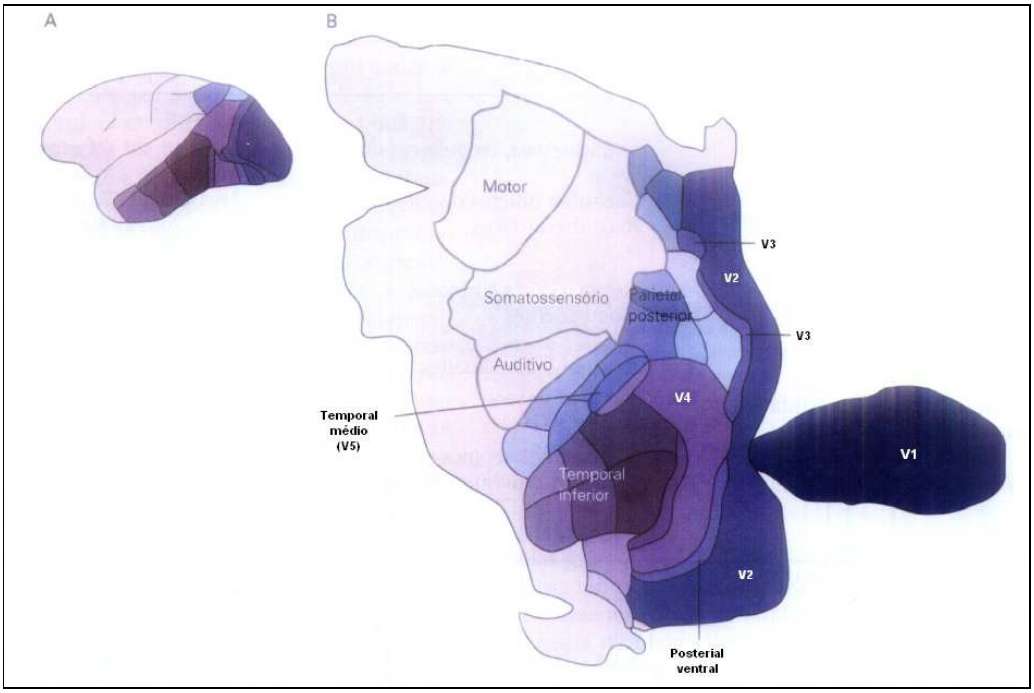

Figura 2-12 – Áreas corticais. Figura retirada de [KAN00].

2.3.1. V1

Quase toda informação visual vinda da retina entra no córtex via a área V1 que, devido a sua aparência estriada, também é conhecida como córtex estriado. As outras áreas são conhecidas como córtex extra-estriado. V1 também é chamada de córtex visual primário e de área 17 de Brodmann [KAN00]. Nos humanos o córtex visual primário possui cerca de 2mm de espessura e é dividido em 6 camadas numeradas de 1 à 6 (Figura 2-13). A camada 4 é a que recebe a maioria das projeções dos axônios do LGN e pode ser dividida em 4 sub camadas: 4A, 4B, 4Cα e 4Cβ. Os axônios de células das camadas parvocelulares (P) do LGN terminam principalmente na camada 4Cβ com algumas poucas projeções para 4A e 1, enquanto que os axônios de células das camadas magnocelulares (M) terminam principalmente na camada 4Cα.

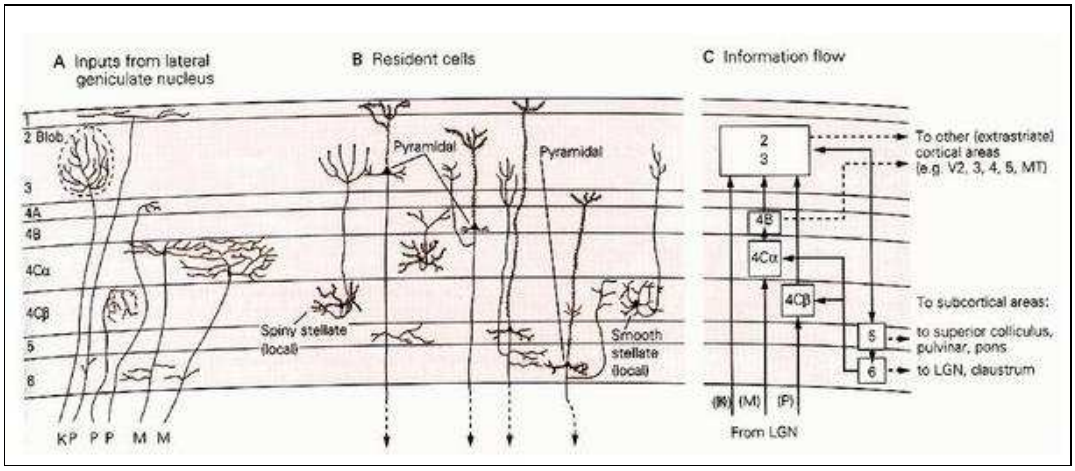

Figura 2-13 – Organização de V1. A) Os axônios dos neurônios P e M do LNG terminam na camada 4; B) Células de V1; C) Concepção do fluxo de informação em V1. Figura retirada de http://www.brainworks.uni-freiburg.de/group/wachtler/VisualSystem/

Através de estudos feitos em macacos verificou-se que o córtex estriado, assim como LGN, possui um mapa retinotópico, isto é, áreas do campo visual vizinhas da retina são também vizinhas em V1, do campo visual contralateral [TOT82]. O aspecto mais importante deste mapa é que cerca da metade das projeções da retina sobre o córtex visual primário são provenientes da fóvea e regiões circunvizinhas. Esta área apresenta a maior acuidade visual.

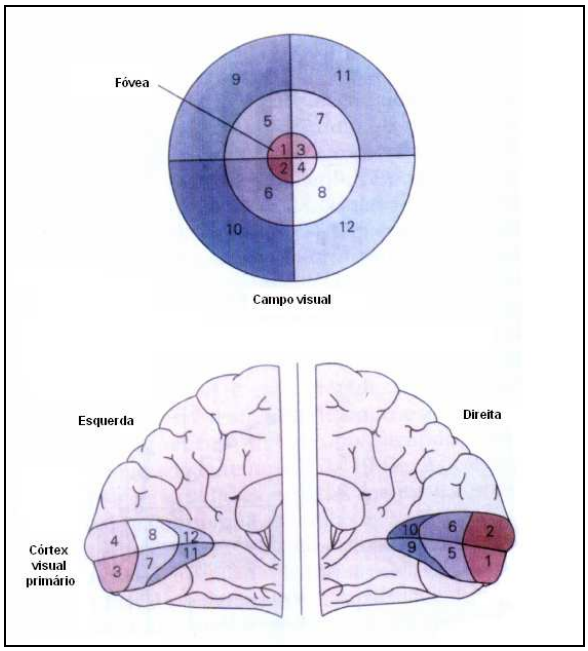

Figura 2-14 – Campo visual representado no córtex visual primário humano. Figura retirada de [KAN00].

O formato do campo receptivo das células de V1 é diferente do formato dos campos receptivos das células da retina e do LGN que são circulares. Em V1, os campos receptivos das células são alongados e, conseqüentemente, respondem melhor à estímulos alongados do que à estímulos pontuais. Hubel e Wiesel [HUB62] classificaram as células de V1 de acordo com a complexidade de sua resposta, dividindo-as em dois grupos chamados simples e complexos.

As células simples também possuem campo receptivo com regiões excitatórias e inibitórias, contudo estas regiões têm seu formato alongado. Estas células respondem melhor à estímulos na forma de barras com uma orientação específica. Uma célula simples que responde melhor a um estímulo vertical não responderá bem à um estímulo horizontal ou oblíquo, e vice-versa.

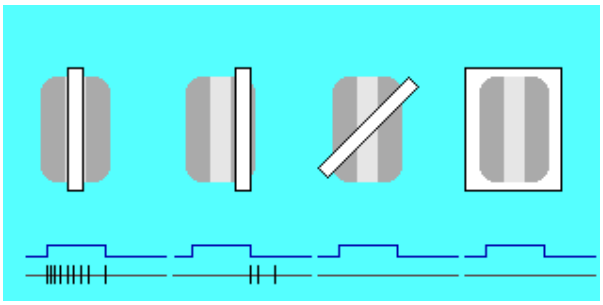

Figura 2-15 – Resposta de uma célula simples em função da projeção de um estímulo em forma de barra. Figura retirada de [MATa]

Na Figura 2-15 é apresentado de forma esquemática o comportamento de uma célula simples quando um estímulo em forma de barra é projetado em seu campo receptivo. Para produzir os resultados mostrados na Figura 2-15 o pesquisador, tipicamente, monitora a tensão no interior da célula através de um microeletrodo (na verdade, uma micropipeta que perfura a parede celular), ao mesmo tempo em que o animal cuja célula está sendo monitorada observa um estímulo. Na Figura 2-15, quatro estímulos na forma de barra são mostrados posicionados sobre uma representação do campo receptivo da célula; os três primeiros, da esquerda para a direita, são barras brancas estreitas e de mesma largura, e o quarto é uma barra branca larga que cobre todo o campo receptivo. O campo receptivo em questão possui orientação vertical, sendo que a parte do mesmo que excita a célula é central e as que inibem ficam nas laterais esquerda e

direita. Abaixo de cada conjunto estímulo-campo receptivo são mostrados dois gráficos. O primeiro, imediatamente abaixo de cada conjunto estímulo-campo receptivo, mostra o momento em que o estímulo está desligado ou ligado (trata-se do comportamento de um sinal elétrico ao longo do tempo que, no nível baixo, indica que o estímulo está inativo e, no nível alto, indica que o estímulo está ativo). O segundo representa o sinal capturado pelo microeletrodo conectado à célula.

Como o primeiro par de gráficos (Figura 2-15) (o mais a esquerda) mostra, a célula simples responde fortemente (emitindo vários pulsos pouco afastados no tempo, que é o modo como as células do córtex sinalizam sua ativação) quando o estímulo é ligado estando corretamente orientado e posicionado sobre a parte central do campo receptivo. A resposta da célula é mais vigorosa imediatamente após o acionamento do estimulo, o que mostra um aspecto temporal da resposta da célula. Na verdade, permanecendo o estímulo por muito tempo (de dezenas de segundos a alguns minutos), a resposta da célula desapareceria totalmente, por um processo conhecido como acomodação. Mas um estímulo constante por muito tempo não ocorre naturalmente, uma vez que movemos os olhos continuamente.

Como o segundo par de gráficos da Figura 2-15 mostra, quando o estímulo é posicionado sobre a parte do campo receptivo que inibe a célula ela não responde no momento em que o estímulo é ligado, embora responda, fracamente, imediatamente após o estímulo ser desligado (também uma evidência do aspecto temporal da resposta da célula). Nos outros casos a célula não responde.

As células complexas são mais numerosas em V1 do que as células simples e, assim como as células simples, respondem bem apenas para um estímulo com uma orientação específica. Porém, diferentemente das células simples, a resposta das células complexas não é seletiva à posição espacial do estímulo, ou seja, não varia com a posição do estímulo dentro do seu campo receptivo. Muitas células complexas são sensíveis ao sentido e direção do movimento do estímulo dentro do seu campo receptivo, respondendo somente quando este estímulo se move numa determinada direção e sentido.

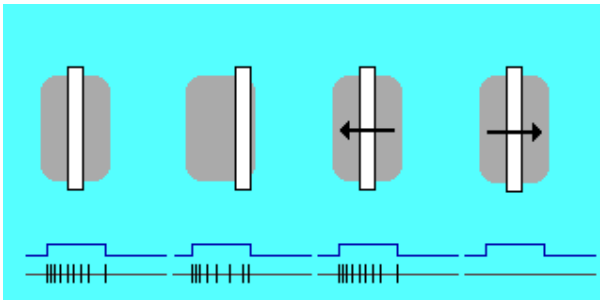

Figura 2-16 – Resposta de uma célula complexa em função da projeção de um estímulo em forma de barra. Figura retirada de [MATa]

Na Figura 2-16 é apresentado o comportamento de uma célula complexa quando um estímulo em forma de barra é projetado no seu campo receptivo. A célula complexa responde independente da posição do estímulo no campo receptivo, diferentemente da célula simples. As células complexas também são sensíveis à movimentação do estímulo dentro de seu campo receptivo. Caso o estímulo se mova no mesmo sentido em que o campo receptivo esteja sintonizado, a célula continua respondendo, caso contrário para de responder ao estímulo.

Hubel e Wiesel foram os primeiros a descobrir que as células de V1 são arranjadas e organizadas de uma forma precisa em relação à sensibilidade à orientação. Ao longo da superfície de V1, a sensibilidade à orientação varia gradualmente, mas permanece constante ao longo dos 2mm de córtex (de uma coluna do córtex). Hubel e Wiesel também descobriram que a resposta das células varia de acordo com o olho estimulado. Muitas células de V1 respondem de forma aproximadamente equivalente a estímulos provenientes de ambos os olhos, mas a maioria das células de V1 respondem preferencialmente à estímulos provenientes de um determinado olho. Esta característica, chamada de dominância ocular, é organizada no córtex visual primário de uma forma que não varia verticalmente (em colunas), mas alterna ao longo da superfície de V1.

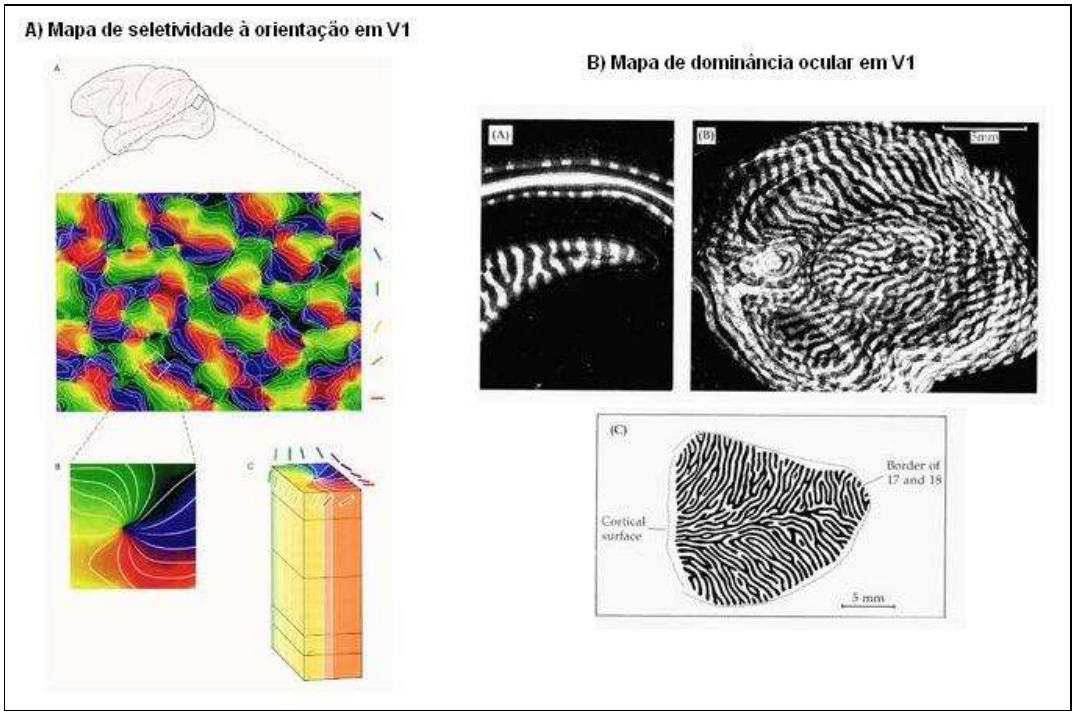

Figura 2-17 – A seletividade à orientação (a) e a dominância ocular (b), variam ao longo da superfície de V1, porém não se alteram numa mesma coluna. Figuras retiradas de http://www.brainworks.uni-freiburg.de/group/wachtler/VisualSystem/.

Um grupo de colunas que respondem à linhas (estímulos) com todas as orientações numa região particular do campo visual foi denominado de hipercoluna por Hubel e Wiesel. Essas hipercolunas aparecem repetidas regularmente e precisamente sobre a superfície de V1, ocupando cada uma cerca de $1mm^2$. Essa organização sugere uma modularização do córtex cerebral, na qual cada módulo de uma área do córtex processaria todas as variantes locais da informação visual tratada naquela área cortical. Assim, se uma determinada área processa orientações do estímulo visual, uma hipercoluna dela codifica todas as orientações da região do campo visual monitorada pela hipercoluna. O mesmo ocorrendo para áreas corticais responsáveis por processar. profundidade, movimento, etc.

2.3.2. V2

A área V2 possui uma extensa fronteira com a área V1. Esta área apresenta regiões chamadas de faixas grossas e faixas finas separadas por regiões chamadas de interfaixas (vide Figura 2-18). As faixas finas e interfaixas recebem projeções da

via parvocelular que vêm das camadas 2 e 3 da área V1 enquanto que as faixas grossas recebem projeções da via magnocelular que vêm das camadas 4B, 4Cα e 4Cβ. As faixas finas e as interfaixas se projetam para a área V4, enquanto que as faixas grossas se projetam para área temporal média (MT ou V5). Estes caminhos não são totalmente separados conforme descrito, pois existem conexões entre as faixas finas e grossas e também projeções da área V4 de volta para as faixas finas de V2. Também existem conexões das faixas grossas com a área V3.

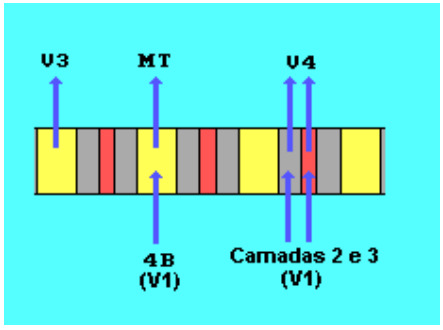

Figura 2-18 – Esquemático de V2. Figura retirada de [MATa]

As células de V2, assim como as células de V1, são sensíveis à orientação, cor e à profundidade dos estímulos, ou seja, estas células continuam a análise iniciada em V1. A resposta das células de V2 para contornos reais e ilusórios foram testados juntamente com as células de V1 em alguns experimentos. Um exemplo de percepção de contornos ilusórios pode ser visto na Figura 2-19. No desenho da esquerda, existe realmente um quadrado desenhado. No desenho do centro, apesar de não existir um quadrado desenhado, é possível facilmente “enxergar” um contorno ilusório de um quadrado, enquanto que no desenho da direita, apesar dos objetos serem os mesmos que no desenho do centro, não é possível “enxergar” o mesmo quadrado.

Muitas células de V2 responderam aos contornos ilusórios exatamente como responderam às bordas, enquanto que poucas células de V1 responderam aos mesmos contornos ilusórios da mesma forma como responderam às bordas [HEY84]. Essas observações sugerem que na área V2 é feito um processamento de contornos num nível acima do processamento que ocorre em V1, constituindo assim uma evidência da análise progressiva que ocorre no córtex visual.

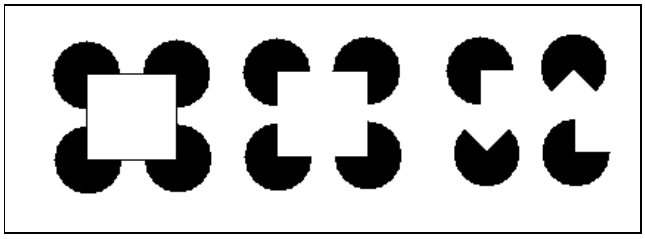

Figura 2-19 – Contorno ilusório. É possível "visualizar" um quadrado branco na figura do meio, apesar de não existir um quadrado desenhado explicitamente.

2.3.3. V3

Pouco se sabe sobre as propriedades funcionais dos neurônios da área extra-estriada V3. Esta área recebe informações das faixas grossas da área V2 e da camada 4B da área V1, e faz projeções tanto para a área temporal média (MT ou V5) quanto para a área V4. Grande parte das células de V3 são seletivas em relação à orientação e direção do estímulo visual, sendo que algumas células são seletivas a cores, o que sugere que em V3 ocorre uma interação entre o processamento de cor e movimento [GEG97].

2.3.4. V4

A área extraestriada V4 foi estudada em profundidade inicialmente por Semir Zeki [ZEK73]. Esta área recebe projeções principalmente das faixas finas e interfaixas de V2, provenientes do caminho parvocelular que vêm do LGN e retina, mas também recebe projeções das áreas V1 e V3. Inicialmente pensava-se que as células de V4 fossem exclusivamente dedicadas ao processamento de cores, porém estudos posteriores mostraram que as células de V4 são sensíveis a combinações de cores e formas. As células de V4 se projetam principalmente para o córtex temporal inferior, onde é feito o reconhecimento de faces e de outras formas complexas.

2.3.5. V5 ou MT

A área V5, também conhecida como área temporal média ou MT, recebe projeções da camada 4B do córtex visual primário (V1) e das faixas grossas de V2.

Estas conexões são provenientes do caminho magnocelular que parte das células M da retina, passa pelas camadas magnocelulares do LGN e chega no córtex MT. Assim como a área V1, MT possui um mapa retinotópico do campo visual contralateral, porém os campos receptivos das células de MT são bem maiores que os campos receptivos das células de V1.

O processamento de movimento começa de forma rudimentar em V1 atingindo formas bem mais abstratas em MT numa abstração sucessiva, ou seja em etapas. Anthony Movshon *et al*. [MOV85] testou a hipótese de que o movimento é processado em 2 etapas, registrando a resposta de células em V1 e MT para um padrão de linhas cruzadas em forma de xadrez em movimento. As células de V1 responderam ao movimento dos elementos isolados do padrão, que são as linhas, enquanto que as células em MT responderam ao movimento do padrão em forma de xadrez por completo.

A percepção de profundidade também é processada em MT. Embora células sensíveis à disparidade binocular sejam encontradas em várias áreas corticais, como V1, V2 e V3, as células de MT respondem melhor a estímulos em distâncias específicas do plano de fixação (mais próximo, ou mais distante do ponto de fixação). Esse processamento da disparidade binocular pode ser utilizado tanto para a percepção de profundidade quanto para o controle do movimento de vergência dos olhos.

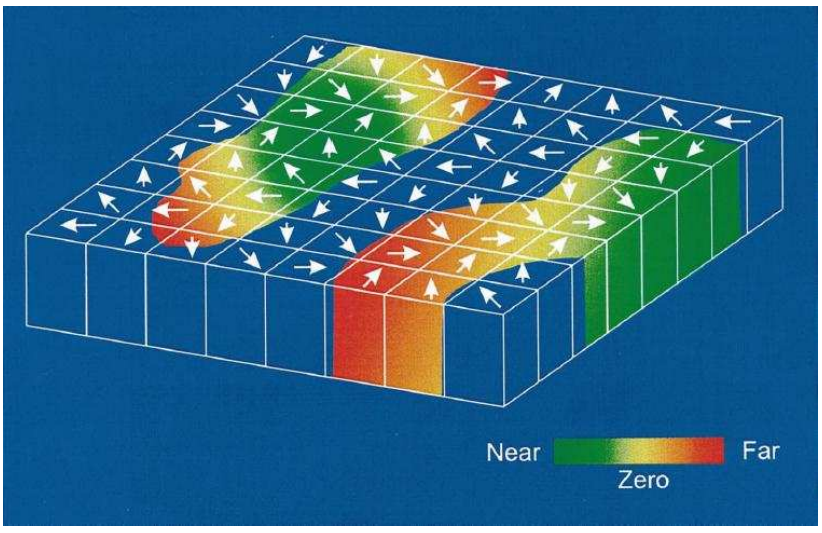

Figura 2-20 – Modelo esquemático da arquitetura funcional de MT. Figura retirada de [DEA99].

A Figura 2-20 apresenta um modelo esquemático da arquitetura funcional de MT que mostra que esta área processa tanto informações de movimento (direção) quanto de profundidade. Na figura, as setas indicam a direção preferencial dos neurônios numa coluna. Estas direções preferenciais variam suavemente através da superfície de MT. A percepção de disparidade é representada pela faixa colorida que varia de *near* (estimulo afastado do ponto de fixação, mas perto do observador), em verde, até *far* (estimulo afastado do ponto de fixação, mas longe do observador), em vermelho, passando pela disparidade zero (estímulo à mesma distância do observador do que o ponto de fixação), codificada em amarelo. As regiões do modelo que estão em azul são regiões da área MT que aparentemente têm pouca seletividade para disparidade.

2.4. VIAS PARALELAS

As informações visuais vindas da retina são conduzidas através de vias paralelas que se iniciam na retina, passam pelo LGN, chegam em V1, e depois continuam até os córtices parietal posterior (via dorsal) e temporal inferior (via ventral), como mostra a Figura 2-21. As células P da retina se projetam para as camadas parvocelulares (camadas 3, 4, 5 e 6) do LGN e seguem para o córtex visual primário, recebendo o nome de via P ou via parvocelular. A partir de V1, esta via se projeta para as faixas finas e interfaixas de V2, que depois seguem para a área V4, formando assim a via ventral que alcança o córtex temporal inferior (Figura 2-21). Os neurônios que fazem parte da via ventral são mais sensíveis em relação ao contorno das imagens, sua orientação e bordas. Um outro aspecto importante que é processado nesta via é a percepção de cores. Estas células possuem alta resolução espacial, baixa resolução temporal e alta sensibilidade a cores e bordas, o que proporciona a este sistema a capacidade de analisar “o quê” é visto. Lesões no lobo temporal inferior causam deficiências relacionadas ao reconhecimento de objetos complexos, inclusive o reconhecimento de faces.

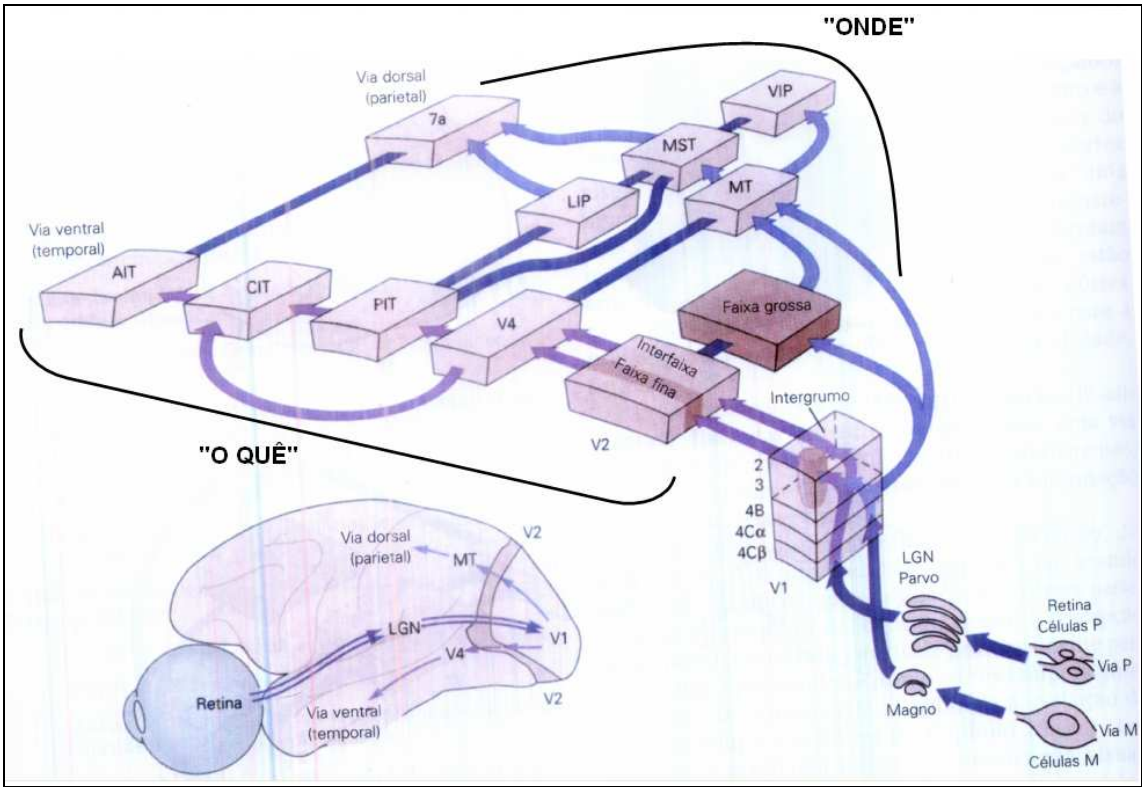

Figura 2-21 – As vias paralelas M e P projetam-se para o córtex visual passando pelo LGN. Figura retirada de [KAN00] e alterada com a inserção .

As células M da retina se projetam para as camadas magnocelulares (camadas 1 e 2) do LGN e também seguem para o córtex visual primário, recebendo o nome de via M ou via magnocelular. A via M se estende de V1 até as faixas grossas de V2, que depois se projetam para a área temporal média (MT) formando a via dorsal que se estende até o córtex parietal posterior (Figura 2-21). Conforme visto anteriormente, o MT (também chamado de V5) está relacionado ao processamento do movimento e profundidade. Os neurônios que formam este sistema são poucos sensíveis a cor e objetos parados, diferentemente dos neurônios associados à via ventral, mas possuem alta resolução temporal e sensibilidade a disparidade binocular, o que faz com que este sistema tenha capacidade de analisar "onde" estão os objetos vistos. Lesões na via dorsal causam deficiência na percepção de movimentos e nos movimentos dos olhos dirigidos a alvos em movimento (movimento de perseguição suave).

A via dorsal, responsável pela análise de "o quê" é visualizado, continua até terminar numa região do córtex pré-frontal especializada na memória de trabalho visual espacial enquanto que a via ventral, responsável pela análise de "onde" estão

os objetos visualizados, continua até terminar numa outra região também do córtex pré-frontal especializada na memória de trabalho de cognição. Esta análise mostra que o sistema visual está organizado em vias paralelas bem definidas, com uma organização seqüencial e hierárquica em cada uma delas.

2.5. SISTEMA ÓCULO-MOTOR

O ângulo total de visão humana possui arco de cerca de 200 graus. A melhor definição fica na fóvea, que tem pouco menos de 1 mm de diâmetro e representa cerca de 2 graus de arco no centro do campo de visão (aproximadamente o tamanho da unha do polegar à distância do braço estendido). Cerca de metade de V1 é devotada inteiramente às fóveas e esta concentração de recursos neurais, que vem da retina e persiste nas outras áreas corticais visuais além de V1, resulta em uma percepção visual muito melhor das imagens que estão sobre as fóveas. Por essa razão, a visão é um sistema bastante elaborado para a movimentação rápida e precisa dos olhos.

Os movimentos dos olhos se dão em 3 eixos de rotação (Figura 2-22): vertical (eixo X, movimento para baixo e para cima - *depression* e *elevation*), horizontal (eixo Y, movimento de lado para outro, *abduction* e *adduction*) e torsional (eixo Z, *extorsions* e *intorsions*)

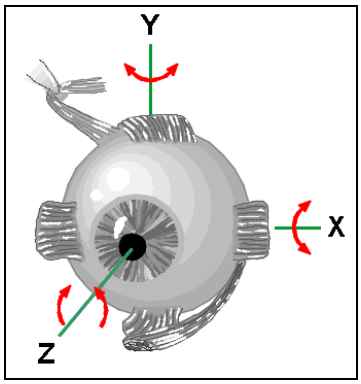

Figura 2-22 – Movimentos oculares. Figura retirada de http://www.auto.ucl.ac.be/EYELAB/Welcome.html com alterações para inclusão dos eixos X, Y e Z.

Cada um dos olhos possui 3 pares de músculos extra-oculares, que operam antagonicamente (Figura 2-23): *Medial Rectus (adduction)* e *Lateral Rectus*

(abduction); *Superior Rectus (elevation)* e *Inferior Rectus (depression)*; *Superior Oblique (extorsion)* e *Inferior Oblique (introsion)* [KAN00].

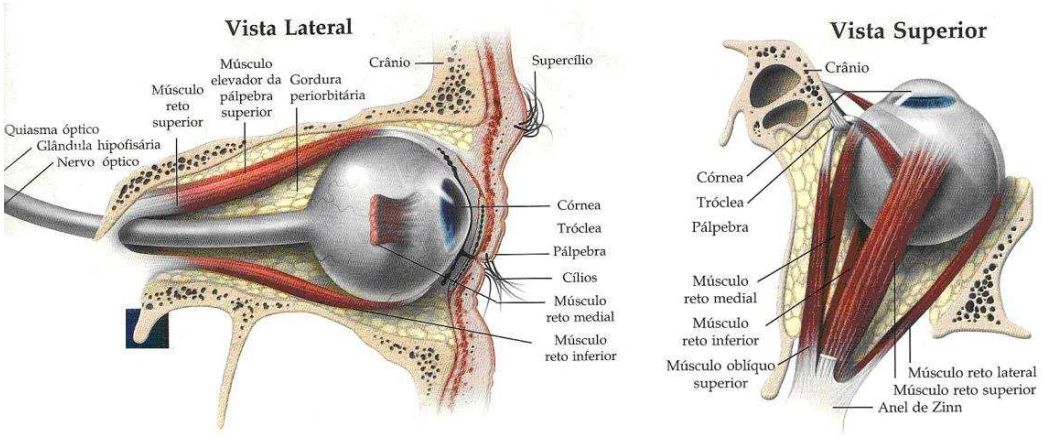

Figura 2-23 – Músculos oculares. A) Vista Lateral; B) Vista Superior. Figuras retiradas de http://www20.brinkster.com/tonho/olho/olhohumano.html

Olhar de forma exploratória em busca de um ponto de interesse requer mover os olhos rapidamente de modo que a imagem dos objetos seja projetada sobre nossas fóveas. Uma vez localizado o ponto de interesse, contudo, precisamos estabilizar sua imagem na retina, mesmo que a cabeça se movimente. O sistema óculo-motor tem, então, duas grandes funções:

1. Posicionar a imagem do ponto de interesse – o alvo – na parte da retina com maior acuidade, a fóvea;
2. Manter a imagem estacionária na fóvea, independente de movimentos do alvo ou da cabeça.

Por volta de 1902, Raymond Dodge descreveu 5 sistemas separados de controle da posição dos olhos [DOD03]. Estes 5 sistemas podem ser divididos em dois grupos, segundo as duas grandes funções descritas do sistema óculo-motor, como mostrado na Tabela 2-1.

Os primeiros 4 movimentos são conjugados, cada olho se move na mesma direção e na mesma quantidade. O último é desconjugado: os olhos se movem em direções diferentes e, muitas vezes, de diferentes quantidades.

Movimento	Função
Movimentos que estabilizam o olho quando a cabeça se move	
Vestíbulo-ocular	Mantém as imagens estáveis na retina durante rápidas rotações da cabeça
Optokinético	Mantém as imagens estáveis na retina durante rotação lenta e contínua da cabeça
Movimentos que mantém a fóvea no alvo	
Sacada	Trás novos pontos de interesse para a fóvea
Perseguição suave	Mantém a imagem de um alvo em movimento na fóvea
Vergência	Ajusta os olhos para que o mesmo ponto seja levado a ambas as fóveas

Tabela 2-1 – Movimentos Oculares.

Existem outros tipos de movimentos que têm como principal característica amplitudes muito pequenas. Estes movimentos são involuntários e ocorrem quando se está observando um objeto fixo. Estes movimentos são chamados de movimentos de *sustaining* e possuem como principal função manter o foco sobre o objeto que está sendo observado, e produzir variações constantes, mesmo que pequenas, da imagem na retina. Sem estas variações a imagem “desapareceria” devido à acomodação dos neurônios do sistema visual.

2.6. VISÃO BINOCULAR

Uma das principais funções do sistema visual é transformar as imagens bidimensionais projetadas na retina numa imagem tridimensional. Estudos indicam que esta transformação é baseada tanto em pistas monoculares para a profundidade como o tamanho familiar dos objetos (sabendo previamente o tamanho aproximado de um objeto, é possível julgar a distância deste objeto), oclusão (caso um objeto A esconda parcialmente um objeto B, assume-se que A está mais próximo que B), entre outras pistas, quanto em pistas estereoscópicas oriundas da disparidade binocular causada pela leve diferença das imagens projetadas nas retinas.

Quando os objetos observados estão a distâncias maiores do que cerca de 30 metros, as imagens projetadas nas retinas são praticamente idênticas, eliminando quase que totalmente a disparidade binocular, fazendo com que a percepção de profundidade seja baseada principalmente nas pistas monoculares. Entretanto, a percepção de profundidade a distâncias menores do que 30 metros é mediada preferencialmente pela *stereopsis* (originário do grego ‘*stereo*‘, que significa sólido, referenciando a imagem em 3D), que é a percepção de profundidade visual baseada

na disparidade binocular. A visão estereoscópica é possível porque os dois olhos estão separados horizontalmente, a uma distância média de 65mm em adultos, sendo que cada olho observa o mundo através de uma posição ligeiramente diferente. Desta forma, objetos a distâncias diferentes produzem imagens com afastamento relativo (disparidade) diferente nas retinas (Figura 2-24).

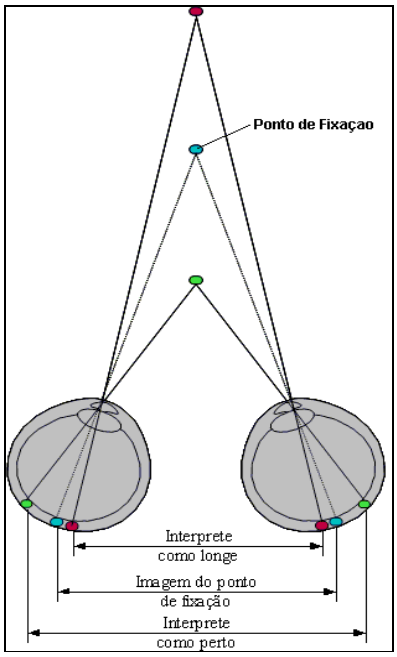

Figura 2-24 – Visão Binocular. Figura retirada de [KAN00] com alterações.

Ao olhar um ponto no espaço, a imagem deste ponto é projetada em pontos correspondentes na retina de cada olho, mais precisamente na fóvea de cada olho. Este ponto no espaço é chamado de ponto de fixação e o plano vertical de pontos no espaço onde se localiza o ponto de fixação é chamado de plano de fixação. Pontos no espaço situados antes do plano de fixação, ou seja, que estão mais próximos do que o ponto de fixação, são projetados em pontos na retina que possuem disparidade positiva (um ponto no olho esquerdo tem seu correspondente no olho direito mais a direita com relação à fóvea), enquanto aqueles mais distantes do que o ponto de fixação possuem uma disparidade negativa (um ponto no olho esquerdo tem seu correspondente no olho direito mais a esquerda). Pontos que estão a mesma distância, possuem disparidade zero (um ponto no olho esquerdo tem seu correspondente no olho direito na mesma posição com relação à fóvea).

3. MODELAGEM MATEMÁTICO-COMPUTACIONAL DO SISTEMA VISUAL HUMANO

A construção de uma imagem tridimensional interna ao computador do mundo externo a partir de duas imagens bidimensionais, que busque similaridade com o sistema biológico, requer uma modelagem das transformações na informação visual que ocorrem durante o caminho que ela faz, saindo da retina, passando pelo LGN até o córtex, e do processamento ocorrido no próprio córtex, em especial nas partes do córtex visual que estão envolvidas no processamento e percepção de profundidade, que são as áreas V1 e MT. Esta modelagem foi feita por Oliveira [OLI05] iniciando com uma transformação nas imagens direita e esquerda (imagens projetadas nas retinas direita e esquerda, respectivamente), que tem como objetivo reduzir a quantidade de informação a ser processada e fornecer mais atenção à informação visual situada no ponto focal, semelhante à transformação ocorrida na imagem nas retinas, LGN, e depois propagada para V1. Após esta transformação, é feita uma extração de características das imagens através de filtros que modelam o processamento feito pelas células de V1 e MT. Esta extração é feita numa arquitetura com níveis crescentes de abstração. Combinando as informações obtidas através destas características de cada imagem, são obtidas as informações necessárias para a reconstrução 3D. Foi implementada também uma inversa desta modelagem para permitir a visualização da representação tridimensional interna ao computador.

Como o capítulo anterior, este também é fundamentalmente baseado no trabalho de Oliveira [OLI05] e está divido em três partes nas quais serão explicadas detalhadamente como foi modelada cada parte do sistema visual humano, na mesma seqüência no qual é feito o processamento da disparidade binocular. Na primeira parte será abordada a transformação na imagem ocorrida entre a retina e o córtex V1, passando pelo LGN. Na segunda parte será abordada a modelagem das células do córtex V1 que realizam o processamento de percepção de profundidade através da disparidade binocular, e como se interagem. Na terceira parte será apresentada a modelagem da área MT, mostrando como esta área recebe as informações de V1 e as organiza de forma a permitir a extração das informações necessárias para a reconstrução tridimensional.

3.1. MODELAGEM DO MAPEAMENTO RETINOTÓPICO RETINA-V1 (MAPEAMENTO LOG-POLAR)

Num dado momento, apenas uma fração da informação disponível da cena visual que recai nas duas retinas pode ser processada. Esta fração é representada principalmente pelas fóveas que, por serem as regiões das retinas que possuem maior quantidade de fotorreceptores, possuem uma maior representação no córtex visual primário.

Através de estudos feitos em primatas, verificou-se como é feito este mapeamento da retina em V1. A Figura 3-1 mostra a representação de uma imagem contendo círculos concêntricos no córtex V1. Esta representação foi obtida por Tootell [TOO82] através da aplicação de 2-deoxyglucose radiativa em um olho e do controle da fixação da fóvea deste olho no centro da imagem. A imagem não era estática; os pontos brancos que a compõem se moviam ao longo do círculo para garantir que as respostas das células do sistema visual não se acomodassem. Após 45 minutos, o macaco foi sacrificado e a parte correspondente a V1 do hemisfério esquerdo do cérebro do macaco foi recortada, aplainada e posicionada sobre um filme fotográfico sensível à radiação que, quando revelado, mostrou a imagem representada na Figura 3-1.

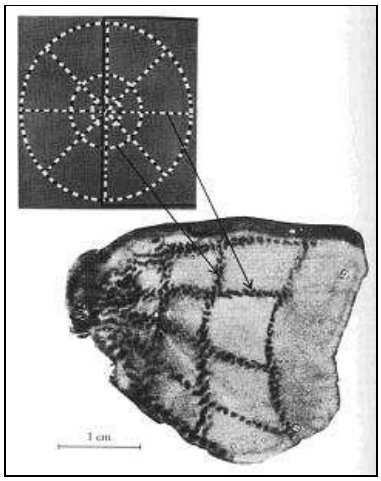

Figura 3-1 – Representação da imagem projetada na retina em V1 [TOO82].

A 2-deoxyglucose radiativa é confundida pelas células do sistema visual como sendo glicose, e é passada pelas sinapses das células mais ativadas de um estágio para outro do sistema visual, segundo o mapeamento da imagem na retina e seus

correspondentes em cada estágio. Este mapeamento, que ocorre na passagem das informações visuais no caminho entre a retina e V1, pode ser modelado matematicamente utilizando uma transformação log-polar da imagem projetada na retina para o córtex visual primário. Esta transformação realiza o mapeamento dos pontos do plano cartesiano (coordenadas x e y) para pontos no plano log-polar (coordenadas η e ξ) de acordo como mostrado na Figura 3-2.

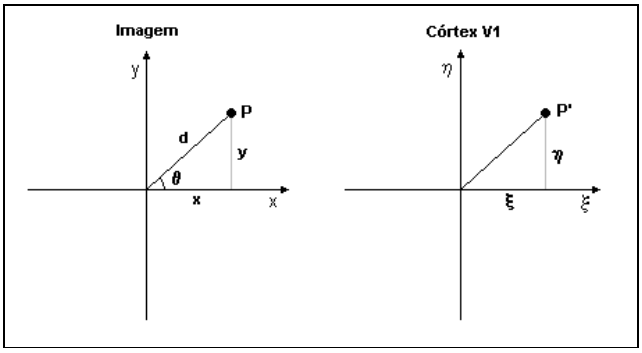

Figura 3-2 – Transformação log-polar.

O mapeamento log-polar é descrito como uma transformação **F:P→P'** na qual:

$$\xi \propto \log(d) = \log\left(\sqrt{x^2 + y^2}\right) \quad (1)$$

$$\eta \propto \theta = \arctan\left(\frac{y}{x}\right) \quad (2)$$

Uma característica importante desta transformação é que, devido ao comportamento logarítmico da variável ξ, a qual é diretamente proporcional ao logaritmo da distância d da origem ao ponto **P,** as regiões mais próximas ao centro das coordenadas cartesianas possuem uma maior representação, e portanto uma melhor definição, no plano log-polar, enquanto que as regiões mais afastadas do centro possuem uma menor representação, e portanto uma pior definição no plano log-polar, semelhante ao que ocorre no sistema visual humano. Este comportamento pode ser visto na Figura 3-3. A Figura 3-3-a mostra o plano x-y repartido em fatias e estas recortadas por círculos concêntricos cujos afastamentos (entre os círculos) segue uma função logarítmica da distância ao centro. A Figura 3-3-b mostra o plano

$\eta-\xi$, no qual cada retângulo representa um dos segmentos de fatia da Figura 3-3a, e deixa claro, nesta representação, como as regiões no centro do plano x-y são mais bem representadas no plano $\eta-\xi$.

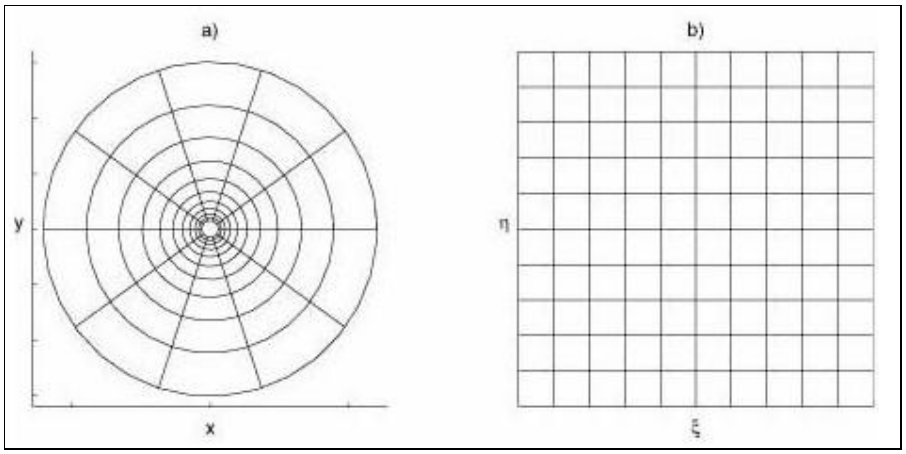

Figura 3-3 – Representação do comportamento logarítmico da transformação log-polar. Figura retirada de http://omni.isr.ist.utl.pt/~alex/Projects/TemplateTracking/logpolar.htm

Na Figura 3-4 é apresentado um exemplo da aplicação da transformação log-polar numa imagem. Uma imagem (Figura 3-4a) de 128x128 *pixels* sofre uma transformação log-polar com uma amostragem de 64x32 (η e ξ respectivamente) representada na Figura 3-4b. Realizando a transformação inversa (Figura 3-4c) é possível verificar facilmente a perda de definição à medida que se afasta do centro.

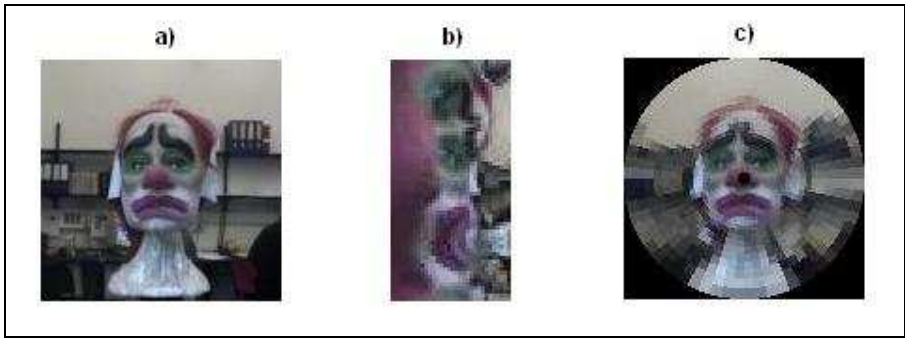

Figura 3-4 – Exemplo da aplicação da transformação log-polar. Figuras retiradas de http://omni.isr.ist.utl.pt/~alex/Projects/TemplateTracking/logpolar.htm

Oliveira [OLI05] modificou a forma de visualização da imagem no plano log-polar a fim de preservar a relação de vizinhança entre os pontos da imagem no

centro da transformação log-polar de acordo com as vizinhanças observadas no córtex relativas às imagens projetadas na fóvea. A Figura 3-5 mostra um exemplo da transformação log-polar utilizada neste caso. A cruz vermelha na Figura 3-5a (próxima ao centro da imagem, sobre o olho) representa o centro da transformação. Na Figura 3-5b, a metade esquerda representa a metade esquerda da imagem original, conforme seria projetada na retina e propagada para V1 se ponto o de atenção fosse a cruz vermelha, e a metade direita representa a metade direita da imagem original na mesma situação.

Figura 3-5 – Exemplo da transformação log-polar utilizada neste caso.

3.2. MODELAGEM DA ÁREA V1

Conforme descrito na seção 2.3.1, quase toda informação visual vinda dos olhos em direção ao córtex visual passa inicialmente pelo córtex visual primário (V1), fazendo com que sua modelagem seja necessária para praticamente qualquer sistema que queira reproduzir o comportamento da visão humana. No trabalho de Oliveira [OLI05], além de ter sido modelado o mapeamento retina-V1, foram modeladas também as células simples e células complexas de V1 (Seção 2.3.1). Para esta modelagem foram utilizadas funções Gabor [MOVa], conforme descrito a seguir.

3.2.1. Células Simples

A informação visual proveniente das células ganglionares da retina passa primeiramente pelo LGN antes de chegar a V1. Neste ponto, a informação vinda dos olhos ainda está segregada; entretanto, a partir de V1 surgem células que recebem informação de ambos os olhos e possuem seletividade para a disparidade binocular [BAR67, PET68]. As projeções do LGN chegam em V1 principalmente nas células simples que, conforme visto na seção 2.3.1, são seletivas à orientação do estímulo visual e possuem campo receptivo alongado com regiões excitatórias e inibitórias, sendo também seletivas a uma determinada freqüência espacial do estímulo.

Hubel e Wielsel [HUB62] sugeriram um padrão de interconexão entre o LGN e V1 que mostra como o campo receptivo das células simples binoculares no córtex V1 pode ser originado a partir do das células do LGN. Através deste padrão de interconexão (Figura 5-6), os campos receptivos de várias células do LGN são sobrepostos para formar o campo receptivo das células simples de V1 (Figura 2-15). Esta sobreposição pode ser tanto de células *on center* quanto *off center*, sendo que na Figura 5-6 é apresentado um esquema para células *on center*.

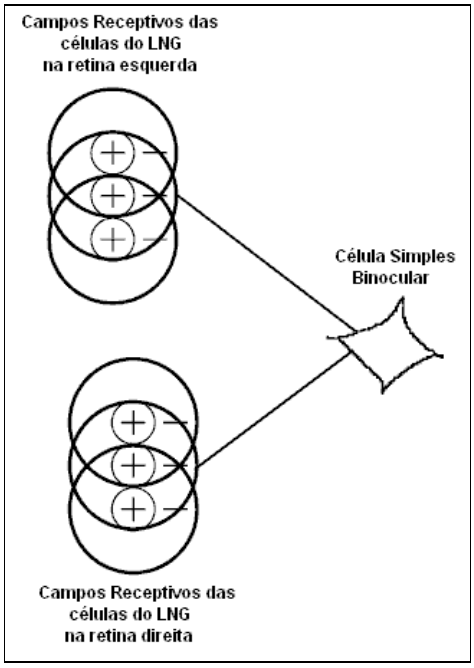

Figura 3-6 – Modelo de Hubel e Wiesel para o campo receptivo das células simples binoculares. Figura retirada de [QIA97b].

Jones e Palmer [JON87] analisaram o perfil do campo receptivo de células simples no córtex estriado em gatos e o descreveram como sendo uma função de Gabor que é o produto de uma cossenóide complexa denominada **portadora**, por uma função gaussiana bidimensional denominada **envoltória** [MOVa]. Jones e Palmer mostraram que a parte real da função de Gabor complexa se ajusta muito bem com os campos receptivos encontrados nas células simples no córtex estriado em gatos. Estudos posteriores mostraram que, semelhante ao campo receptivo das células simples do córtex visual de gatos, o perfil do campo receptivo das células simples dos macacos também podem ser descrito por uma função de Gabor.

A fórmula de uma função de Gabor complexa bidimensional $g(x,y)$, no domínio do espaço é:

$$g(x,y) = s(x,y) * w(x,y) \tag{3}$$

onde x e y são as coordenadas espaciais de um ponto na retina, $s(x,y)$ é a cossenóide complexa (portadora) e $w(x,y)$ é a função gaussiana bidimensional (envoltória). Considerando somente a parte real da função de Gabor, a portadora é:

$$s(x,y) = \cos\left(2\pi\left(u_0 x + v_0 y\right) + \phi\right) \tag{4}$$

onde o parâmetro Φ define a fase da portadora enquanto que os parâmetros u_0 e v_0 definem a freqüência espacial da portadora em coordenadas cartesianas. Esta freqüência espacial também pode ser expressa em coordenadas polares com uma magnitude F_0 e uma orientação θ_0:

$$\begin{aligned} F_0 &= \sqrt{{u_0}^2 + {v_0}^2} \\ \theta_0 &= \arctan\left(\frac{v_0}{u_0}\right) \end{aligned} \tag{5}$$

ou seja,

$$u_0 = F_0 \cos(\theta_0)$$
$$v_0 = F_0 \sin(\theta_0) \tag{6}$$

A envoltória da função de Gabor é modelada da seguinte forma:

$$w(x, y) = K \exp\left(-\pi\left(a^2 (x - x_0)_r^{\ 2} + b^2 (y - y_0)_r^{\ 2}\right)\right) \tag{7}$$

no qual o parâmetro K é um fator de escala para a amplitude da envoltória, x_0 e y_0 representam a coordenada do pico da gaussiana e, a e b são parâmetros de escala da envoltória que controlam a largura de banda ω, em oitavas, da portadora:

$$a = F_0 \omega$$
$$b = \frac{a}{aspect\_ratio} \tag{8}$$

no qual *aspect_ratio* é a relação entre a largura e a altura do campo receptivo. A Figura 3-7 mostra o gráfico de uma gaussiana bidimensional simétrica com *aspect_ratio* igual a 1.

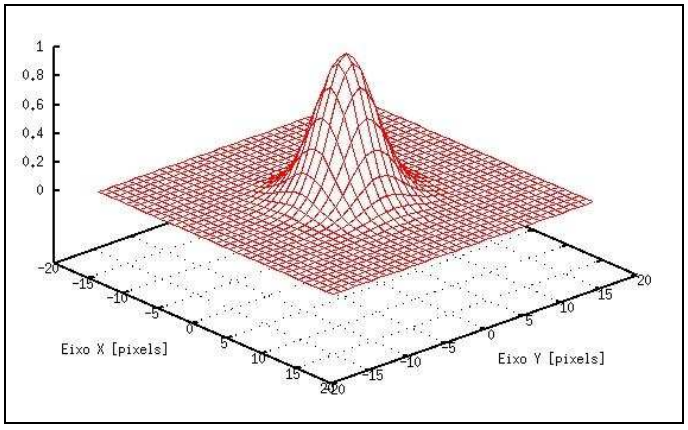

Figura 3-7 – Gaussiana bidimensional simétrica, ou seja, com *aspect_ratio* = 1.

O subscrito r na equação (7) indica a possibilidade de uma operação de rotação bidimensional do campo receptivo da célula simples. No caso desta rotação, os termos entre parêntesis com subscrito r na equação (7) devem ser substituídos por:

$$\begin{cases} (x-x_0)_r = (x-x_0)\cos(\theta_0)+(y-y_0)\sin(\theta_0) \\ (y-y_0)_r = -(x-x_0)\sin(\theta_0)+(y-y_0)\sin(\theta_0) \end{cases} \quad (9)$$

Dessa forma, combinando as equações (3), (4) e (7) a parte real da função de Gabor fica:

$$g(x,y) = K\exp\left(-\pi\left(a^2(x-x_0)_r^{\,2} + b^2(y-y_0)_r^{\,2}\right)\right) * \cos(2\pi(u_0 x + v_0 y) + \phi) \quad (10)$$

Na verdade, a equação (10) possui uma componente DC (um valor constante, não nulo, que é somado à todas as respostas da célula, independente da entrada) indesejável, uma vez que a célula não deve responder para estímulos constantes, independente de sua intensidade. Para resolver este problema, é incluída uma compensação da componente DC [MOVa], na portadora descrita anteriormente na equação (4):

$$s(x,y) = \cos(2\pi(u_0 x + v_0 y) + \phi) - \exp\left(-\pi\left(\frac{u_0^{\,2}}{a_0^{\,2}} + \frac{v_0^{\,2}}{b_0^{\,2}}\right)\right)\cos(\phi) \quad (11)$$

Assim, a forma final da função de Gabor que modela o campo receptivo de uma célula simples é:

$$g(x,y) = K\exp\left(-\pi\left(a^2(x-x_0)_r^{\,2} + b^2(y-y_0)_r^{\,2}\right)\right) \left[\cos(2\pi(u_0 x + v_0 y) + \phi) - \exp\left(-\pi\left(\frac{u_0^{\,2}}{a_0^{\,2}} + \frac{v_0^{\,2}}{b_0^{\,2}}\right)\right)\cos(\phi)\right] \quad (12)$$

A Figura 3-8 mostra o gráfico de funções de Gabor para diferentes valores de ϕ.

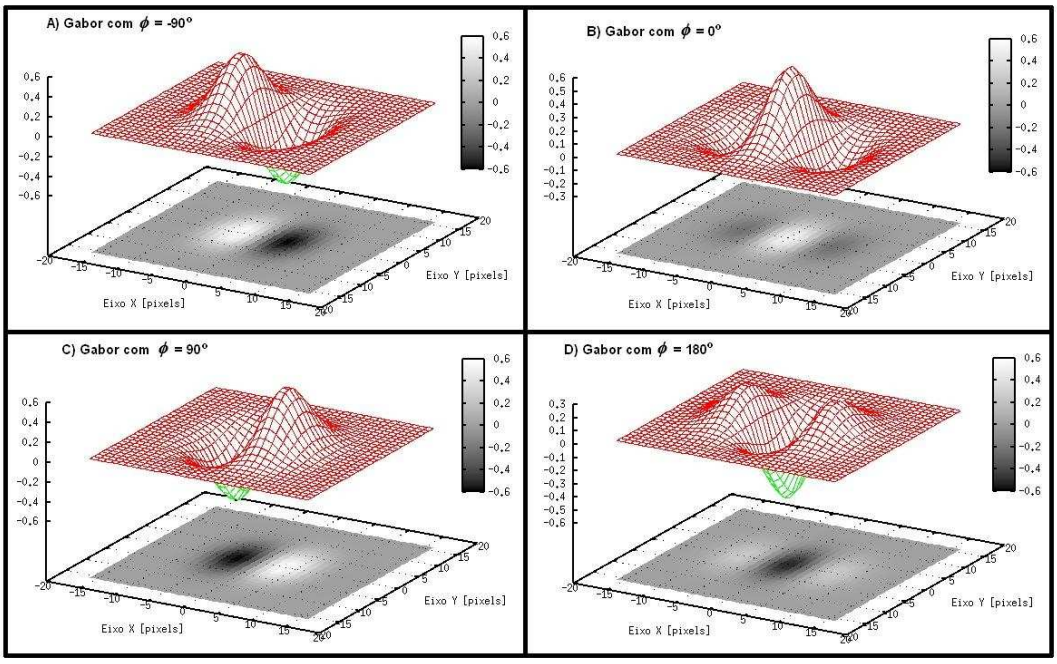

Figura 3-8 – Funções de Gabor para diferentes valores de fase. A) Φ = -90°. B) Φ = 0°. C) Φ = 90°. D) Φ = 180°. Todos os gráficos estão na mesma escala e a área coberta pela função de Gabor é de 33x33 unidades (*pixels*).

A resposta de uma célula simples, representada por $R_s(x,y)$ na equação (13), é aproximada pela convolução espacial de uma função de Gabor com o estímulo visual, ou seja, a convolução da equação (12) com o estímulo, representado por $Estimulo(x,y)$. Realizar esta convolução é o equivalente a extrair a soma das amplitudes das componentes de freqüência espacial que compõe o estimulo de acordo com o filtro de Gabor bidimensional, de freqüências u_0 e v_0 e largura de banda ω.

$$R_S = g(x,y) * Estimulo(x,y) \qquad (13)$$

Considerando o campo receptivo das células simples binoculares, como sendo a composição de dois campos receptivos conforme descrito na equação (12), um para o olho direito e outro para o olho esquerdo, existem basicamente três maneiras de se detectar disparidades binoculares com células simples binoculares: através da diferença da posição dos campos receptivos nos dois olhos, ou da diferença de fase dos campos receptivos nos dois olhos, ou de ambas. No modelo de diferença de posição (Figura 3-9) os campos receptivos em cada olho possuem a

mesma forma, porém não estão centrados na mesma posição relativa à fóvea nas duas retinas. No modelo de diferença de fase (Figura 3-9) os campos receptivos estão centrados em posições correspondentes nas duas retinas porém, devido à diferença de fase, possuem perfil diferente. Ambos os modelos podem ser combinados em um modelo híbrido, de posição e fase.

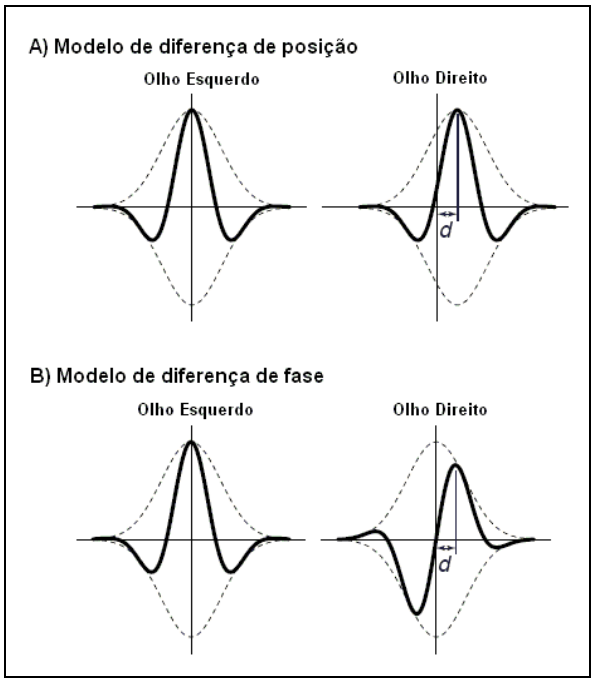

Figura 3-9 – Modelos de codificação de disparidade. A) No modelo de diferença de posição os campos receptivos são idênticos porém centrados em posições não correspondentes nas duas retinas. B) No modelo de diferença de fase os campos receptivos possuem formas diferentes porém estão centrados em posições correspondentes nas duas retinas. Figura adaptada de [DEA00].

Existe uma diferença no valor máximo de disparidade d que cada modelo pode codificar. No modelo de diferença de fases, a faixa de disparidades binoculares que é possível codificar é inversamente proporcional à freqüência espacial central da célula, já que a reposta da célula segue uma função periódica – a portadora da função Gabor. Assim, quanto maior a freqüência, menor a região coberta por um ciclo da função portadora da Gabor, e menor a disparidade que se consegue detectar. O modelo de diferença de posição não possui esta restrição.

Mapeando simultaneamente o campo receptivo dos olhos direito e esquerdo em pares de células simples de V1, Anzai *et al.* [ANZ99a] mostraram que existe tanto disparidade por diferença de posição quanto por diferença de fase, permitindo um modelo híbrido (fase e posição) de codificação da disparidade binocular.

No trabalho de Oliveira [OLI05] foram implementados dois tipos de células simples: células simples monoculares e células simples binoculares. As células simples monoculares foram implementadas utilizando um campo receptivo modelado por uma função de Gabor, de acordo com a equação (12), centrado num ponto da retina de um dos dois olhos (na verdade, câmeras). A saída desta célula simples monocular é a convolução da imagem centrada neste ponto com o campo receptivo, conforme equação (13).

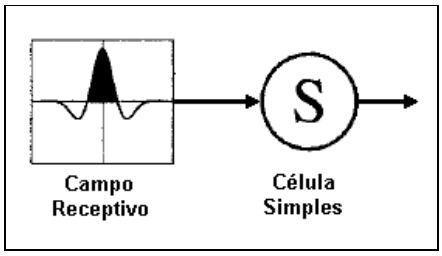

Figura 3-10 – Modelo de célula simples monocular implementada com o campo receptivo ajustado com fase $\Phi = 0$.

As células simples binoculares foram implementadas utilizando dois campos receptivos, cada um recebendo informações de um olho com diferença de fase de 180º entre eles. A resposta desta célula simples binocular é computada somando a contribuição de cada campo receptivo. Ou seja, a saída da célula simples binocular é computada como sendo a soma de duas células simples monoculares, uma de cada olho, com diferença de fase de 180º. O diagrama da Figura 3-11 mostra como implementamos as células simples binoculares.

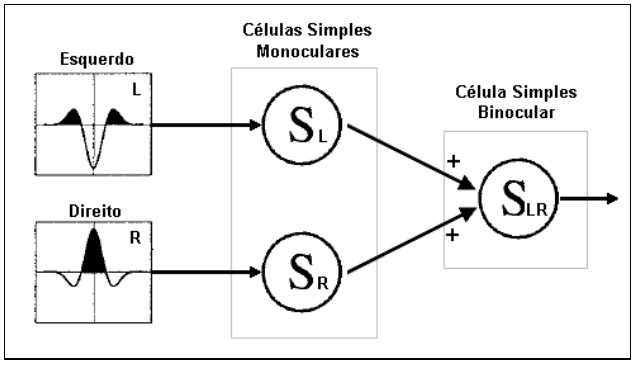

Figura 3-11 – Modelo de célula simples binocular implementada cuja resposta é a soma das respostas de duas células simples monoculares.

Na célula simples binocular implementada, quando um estímulo é projetado sobre seus campos receptivos em ambas as câmeras simultaneamente, a resposta da célula simples binocular é nula, pois a contribuição de um campo receptivo anula a do outro devido à defasagem de 180º na portadora dos dois campos receptivos. Esta situação é mostrada na Figura 3-12. Caso o estímulo não seja projetado em pontos correspondentes, as contribuições dos campos receptivos terão amplitudes diferentes, fazendo com que a célula simples binocular tenha uma resposta não nula.

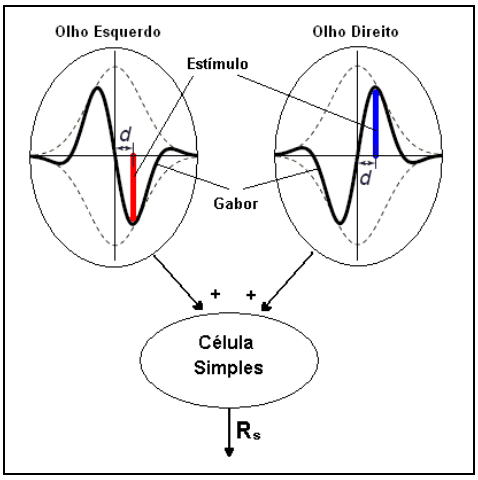

Figura 3-12 – Resposta da célula simples binocular implementada. O mesmo estímulo é projetado em posições correspondentes nos campos receptivos esquerdo e direito. Assim, a contribuição de um campo receptivo é anulada pela do outro, fazendo anulando a reposta R_S da célula simples.

O primeiro estágio de codificação da profundidade através da disparidade binocular é feito pelas células simples binoculares. Entretanto, devido à característica de seu campo receptivo, que possui claramente regiões excitatórias alternadas com regiões inibitórias, a resposta das células simples binoculares é influenciada pela posição do estímulo dentro de seu campo receptivo, ou seja, pela fase do estímulo. As células complexas, descritas a seguir, não são influenciadas pela fase do estímulo.

3.2.2. Células Complexas

Para codificar a profundidade dos objetos no campo visual, o sistema visual humano precisa resolver o problema da correspondência, ou seja, descobrir, para

cada ponto da imagem projetada na retina direita, qual é o ponto correspondente na imagem projetada na retina esquerda, e vice-versa. As células complexas computam uma aproximação da solução para o problema da correspondência [ANZ99b].

Analisando a resposta das células complexas, Hubel e Wiesel [HUB62] observaram que estas células não possuem regiões excitatórias e inibitórias bem definidas e que, de fato, as células complexas respondem a estímulos, independente da sua posição dentro do campo receptivo (vide Figura 2-16). Como uma possível explicação para o campo receptivo das células complexas, Hubel e Wiesel propuseram um modelo hierárquico no qual o campo receptivo de uma célula complexa seria formado por uma composição dos campos receptivos de células simples [HUB62], conforme mostrado na Figura 3-13.

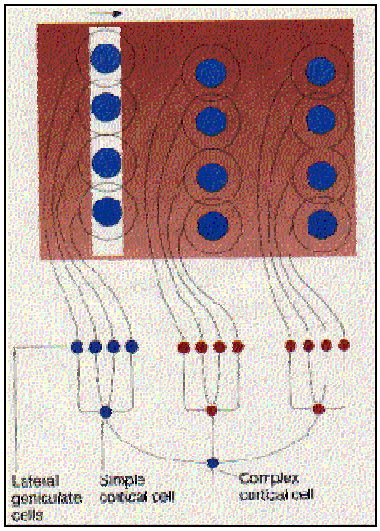

Figura 3-13 – Modelo Hubel e Wiesel de células complexas. O campo receptivo é composto de campos receptivos de células simples que por sua vez são compostos por campos receptivos de células do LGN. Figura retirada de http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/V/VisualProcessing.html

Vários estudos demonstraram que as respostas de células complexas não poderiam ser resultado de uma combinação linear das respostas de células simples [ANZ99b]. Contudo, as células complexas aparentam prover um nível seguinte de abstração para o processamento da *stereopsis* baseado na atividade de células simples, mesmo que não uma combinação linear da resposta destas.

Vários modelos de células complexas têm sido propostos utilizando uma combinação entre subunidades de campos receptivos de células simples. Uma modelagem bastante popular de células complexas é a denominada de **modelo de**

energia, que é construída com dois filtros passa banda em quadratura de fase (fases com diferença de 90º) cujas saídas são elevadas ao quadrado e então somadas [ADE85]. Ohzawa *et al.* [OHZ97] propôs um modelo de energia de segunda ordem para células complexas binoculares, o qual provê uma boa aproximação do comportamento exibido por células complexas do córtex estriado de gatos. Este modelo computa a soma do quadrado da saída de um par de células simples em quadratura. Cada célula simples soma a contribuição de seus dois campos receptivos, um na retina direita e outro na retina esquerda (Figura 3-14).

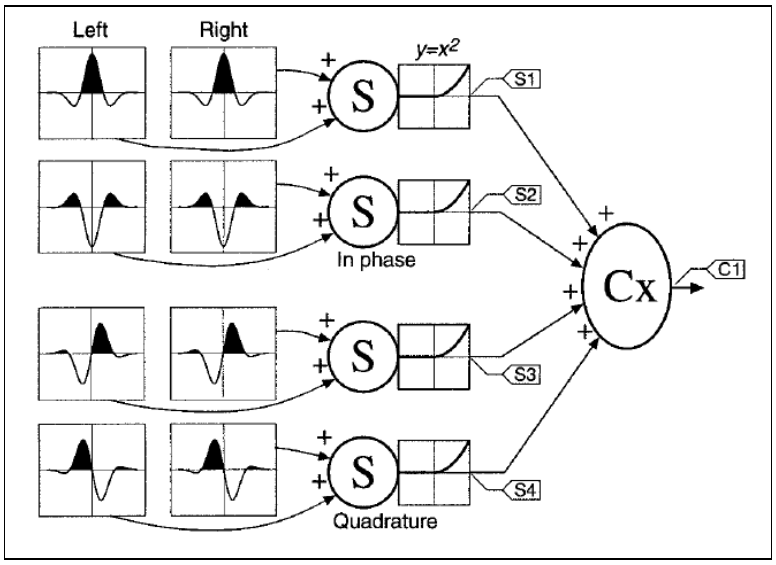

Figura 3-14 – Modelo de energia para célula complexa proposto por Ohzawa para detecção de disparidade binocular. Figura retirada de [OHZ97].

Oliveira [OLI05] propôs dois tipos de células complexas: células complexas monoculares e células complexas binoculares. Ambos os tipos recebem informações de duas células simples e computam a soma do quadrado da saída destas células. As células complexas monoculares são modeladas recebendo projeções de duas células simples também monoculares, com campos receptivos centrados num mesmo ponto da retina de um mesmo olho, porém em quadratura de fase, ou seja, com diferença de fase Φ = 90º. A saída de cada célula simples é elevada ao quadrado e somadas, formando a resposta da célula complexa monocular. A Figura 3-15 mostra de forma esquemática a arquitetura da célula complexa monocular implementada.

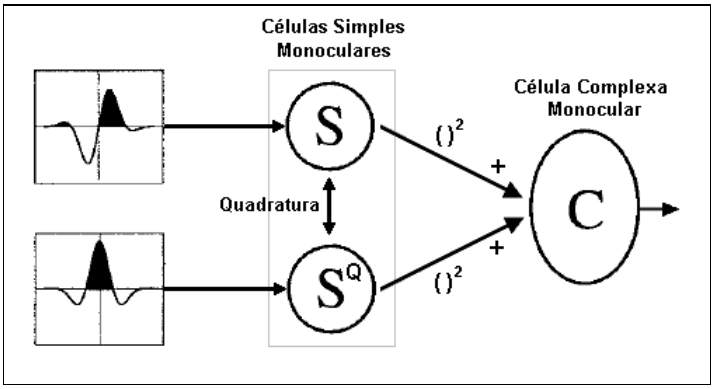

Figura 3-15 – Modelo de célula complexa monocular, formada pela composição de duas células simples em quadratura de fase.

O modelo de célula complexa binocular proposto por Oliveira [OLI05], tem como base o modelo proposto por Ohzawa [OHZ97] (Figura 3-14), porém no modelo de célula complexa proposto pelo Ohzawa, os campos receptivos direito e esquerdo de cada célula simples estão em fase (diferença de fase nula), enquanto que as células simples entre si (S1, S2, S3 e S4) possuem diferença de fase (Figura 3-14). No modelo proposto, os campos receptivos de uma mesma célula simples possuem uma diferença de fase de 180º como mostrado na Figura 3-11. Contudo, assim como no modelo de Ohzawa, neste modelo há uma diferença de fase de 90º entre as células simples (elas estão em quadratura). A Figura 3-16 mostra de forma esquemática o modelo implementado de célula complexa binocular.

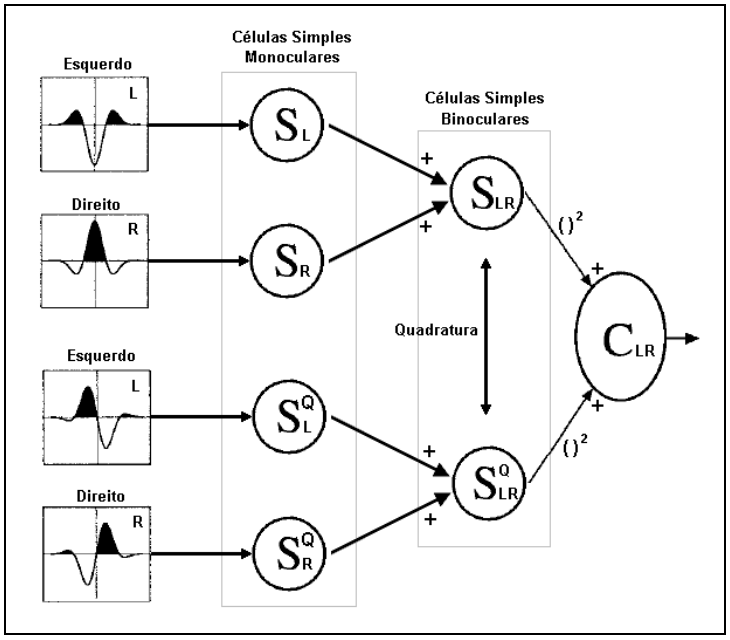

Figura 3-16 – Modelo implementado de célula complexa binocular. Este modelo consiste em subunidades de células simples binoculares em quadratura, que por sua vez são compostas de células simples monoculares.

Esta diferença de fase de 180º graus entre os campos receptivos das células simples binoculares que compõem a célula complexa binocular, faz com que o comportamento desta célula complexa seja do tipo *tuned inhibitory*. Células *tuned inhibitory* têm resposta mínima quando os estímulos nos seus campos receptivos no olho esquerdo e direito são iguais. O modelo proposto por Ohzawa (Figura 3-14) é do tipo *tuned excitatory*, no qual a célula complexa responde de forma máxima quando os estímulos nos seus campos receptivos no olho esquerdo e direito são iguais. A Figura 3-17 mostra o perfil da resposta de células *tuned inhibitory* e *tuned excitatory*.

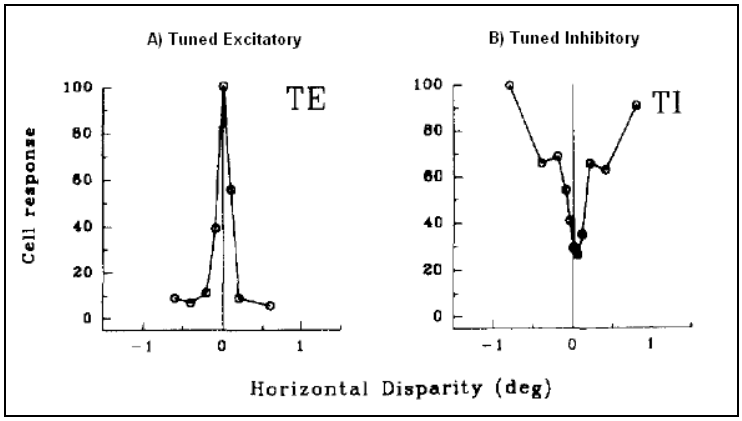

Figura 3-17 – Resposta de células do tipo *tuned excitatory* e *tuned inhibitory*. Figura adaptada de [GON98].

3.3. MODELAGEM DA ÁREA MT

A área do córtex conhecida como MT ou V5 recebe projeções principalmente da camada 4B de V1 e das faixas grossas de V2 (Seção 2.3.5), sendo que a área MT, além de processar informações sobre profundidade, também processa informações sobre movimento. No trabalho de Oliveira [OLI05] é abordado somente o processamento de profundidade e, por essa razão, a modelagem de MT foi especificamente baseada na continuidade do processamento da percepção da profundidade partindo do processamento já realizado em V1, pelas células simples e complexas. As respostas das células complexas de V1 são propagadas para a área MT, onde continuará o processamento da percepção de profundidade. Basicamente, em MT, esta informação de profundidade vinda de V1 sofrerá um refinamento e, em seguida, será agrupada por níveis superiores do córtex de modo a selecionar a melhor disparidade para cada ponto processado.

Na modelagem da área MT proposta, cada célula de MT recebe projeções de uma célula complexa binocular (C_{LR}) e de duas células complexas monoculares (C_L e C_R), com campos receptivos centrados nos mesmos pontos da célula complexa binocular. Desta forma, cada célula de MT é binocular e possui um campo receptivo formado pela combinação de uma célula complexa binocular e duas células simples monoculares, conforme mostrado na Figura 3-18.

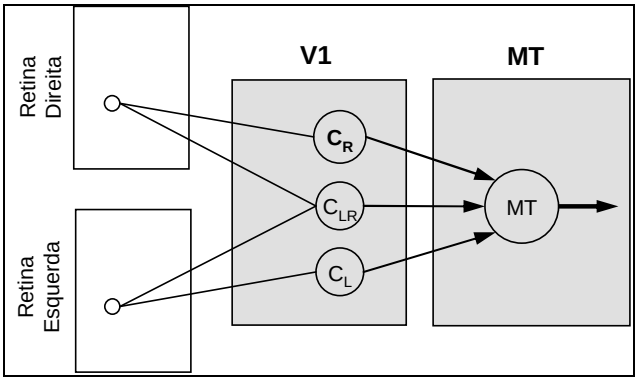

Figura 3-18 – Uma célula de MT implementada recebe projeções de células complexas de V1 com campos receptivos centrados nos mesmos pontos.

Neste modelo, uma célula de MT normaliza a saída de uma célula complexa binocular pela soma das saídas das células complexas monoculares. Uma constante k foi adicionada a esta normalização para controlar a seletividade da célula. Desta forma, a saída de uma célula de MT é dada pela equação (14).

$$MT = \frac{C_{LR}}{C_L + C_R + k} \tag{14}$$

De acordo com o modelo esquemático da arquitetura funcional de MT mostrado na Figura 2-20 (Seção 2.3.5), a percepção de profundidade é codificada em faixas que variam suave e continuamente entre disparidades *near* (perto) e disparidades *far* (longe) passando pela disparidade zero. A modelagem desta arquitetura foi feita utilizando várias camadas de células complexas, onde cada camada está ajustada para detectar uma determinada disparidade. Camadas sucessivas detectam disparidades sucessivas.

Neste modelo, uma camada ajustada para disparidade d em MT é modelada como um conjunto de células que recebe a projeção das repostas das células complexas de V1 que possuem seus campos receptivos ajustados para detectar esta disparidade. Para modelar uma camada ajustada para uma disparidade $d+1$, os campos receptivos que recebem informações sobre a imagem projetada na retina direita são centrados na mesma posição que os mesmos campos receptivos da camada d, enquanto que os campos receptivos que recebem informações sobre a imagem projetada na retina esquerda são centrados numa posição deslocada de

uma unidade (um *pixel*) para a direita, aumentando a disparidade binocular em uma unidade, conforme mostrado na Figura 3-19. Logo, no modelo proposto por Oliveira [OLI05] a disparidade é computada segundo a diferença de posição dos campos receptivos esquerdo e direito das células e não segundo a fase (vide Figura 3-9).

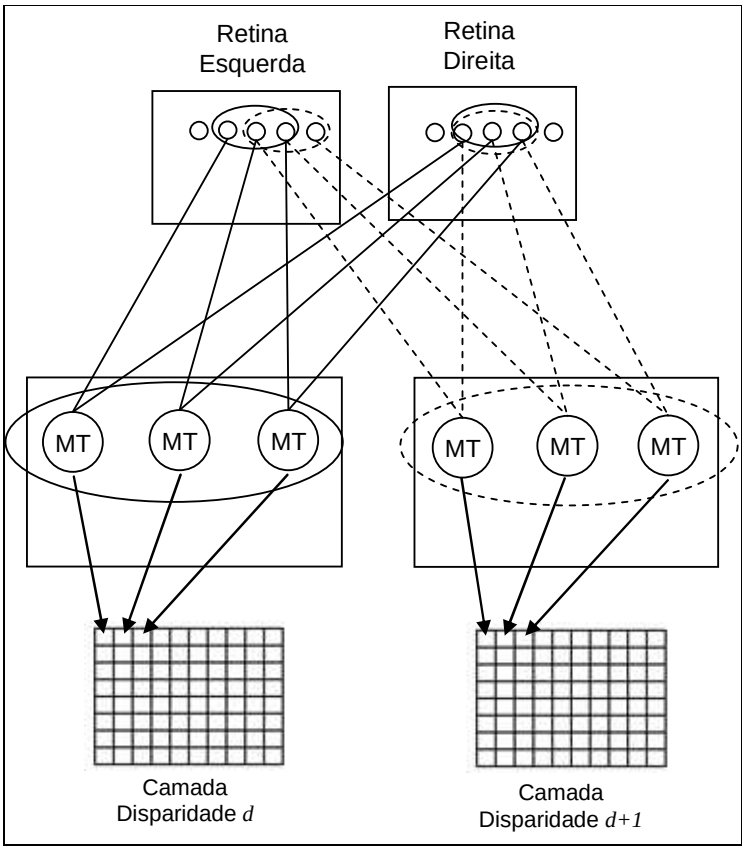

Figura 3-19 – Camadas com disparidade *d* e *d+1* em MT .

Várias camadas com disparidades sucessivas são agrupadas em ordem, conforme mostrado na Figura 3-20, de modo a modelar a arquitetura de MT de acordo com o modelo funcional biológico mostrado na Figura 2-20 (Seção 2.3.5). No cérebro, as células que codificam diferentes disparidades associadas à um mesmo ponto da imagem são vizinhas na superfície do córtex MT; neste modelo, apenas por conveniência, esta vizinhança ocorre na dimensão da profundidade da matriz tridimensional da Figura 3-20. No cérebro este arranjo não seria possível, devido à organização planar bidimensional do córtex.

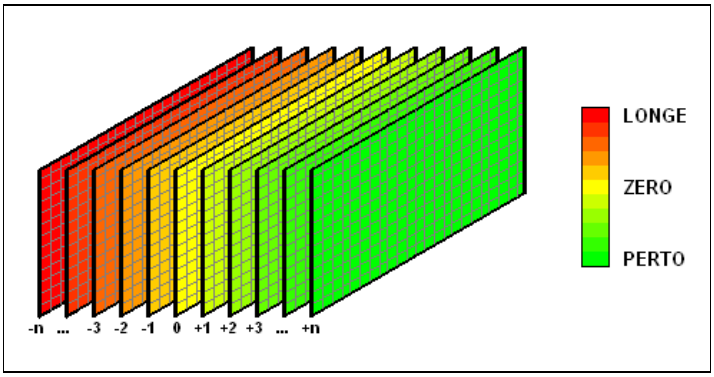

Figura 3-20 – Modelagem em camadas da arquitetura funcional de MT.

Em cada camada de MT, foi incluída uma forma de cooperação entre as células vizinhas de uma mesma camada, o que melhora significantemente a percepção de profundidade. Uma célula de MT influencia no resultado de outras células em sua vizinhança, numa mesma camada, e vice versa. Isto é baseado no fato que o tamanho do campo receptivo das células complexas é, em média, maior que os campos receptivos das células simples de mesma excentricidade [HUB62], e na modelagem feita por Oliveira [OLI05], as células de MT recebem projeções exclusivamente de células complexas de V1.

Foi proposto por Qian & Zhu [QIA97a] que esta influência entre as células vizinhas pode ser incorporada no modelo de uma célula complexa calculando a média dos pares de células simples em quadratura próximas umas das outras, sobrepondo seus campos receptivos. Isto pode ser modelado matematicamente aplicando uma convolução com uma função peso espacial, processo no qual é chamado de ***spatial pooling***.

Com esta cooperação entre as células vizinhas, a resposta final de uma célula complexa (R_c) é dada pela convolução espacial das respostas das células complexas ao seu redor (R_q) com uma função que dá pesos diferentes para esta cooperação em função da distância da célula vizinha à célula complexa em questão. A função peso (w) utilizada foi uma Gaussiana bidimensional assim como descrito na equação (7).

$$R_c = R_q * w \qquad (15)$$

3.4. ARQUITETURA DE OLIVEIRA

A arquitetura de MT proposta por Oliveira [OLI05] é apresentada na Figura 3-21. As células da área MT recebem projeções das células complexas de V1 (tanto células complexas binoculares quanto monoculares) que por sua vez recebem informações das células simples binoculares e monoculares respectivamente.

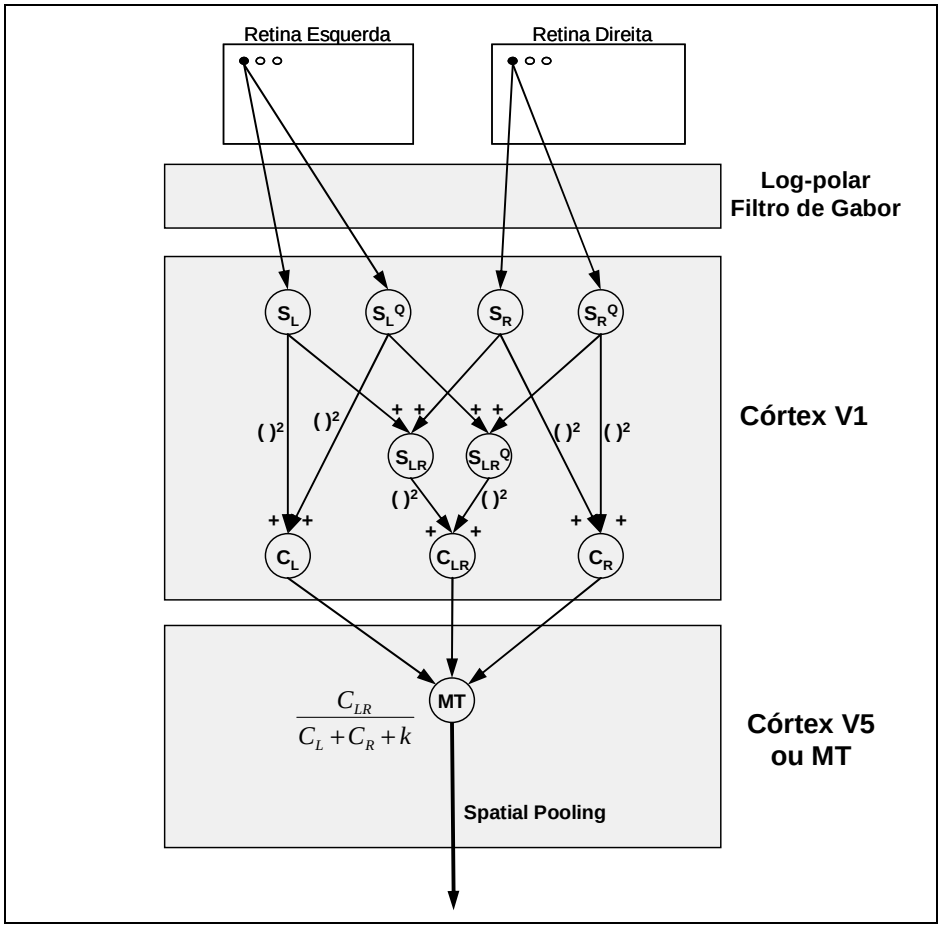

Figura 3-21 – Modelo representando a arquitetura de V1 e MT. Cada célula de MT recebe projeções das células complexas de V1.

Neste modelo, as imagens projetadas nas retinas são conduzidas até ao córtex V1 passando pelo LGN. Neste caminho, devido aos fatores de magnificação e ao tipo de mapeamento retinotópico existente no LGN (Figura 2-9 – Seção 2.2) e no próprio córtex (Figura 2-14 – Seção 2.3.1), as imagens sofrem uma transformação log-polar. Os filtros de Gabor modelam o campo receptivo das células simples em V1. As células simples monoculares S_L e S_L^Q, com campo receptivo na retina

esquerda, estão em quadratura de fase, assim como S_R e S_R^Q, na retina direita. As células simples binoculares S_{LR} e S_{LR}^Q recebem projeções das células simples monoculares em fase e em quadratura respectivamente e, portanto, também estão em quadratura de fase.

A célula complexa monocular C_L tem como entrada a saída das células simples monoculares S_L e S_L^Q, enquanto que C_R tem como entrada a saída das células simples monoculares S_R e S_R^Q. A célula complexa binocular C_{LR} recebe projeções das células simples monoculares S_{LR} e S_{LR}^Q. Uma célula em MT recebe projeções de três células complexas, sendo duas delas monoculares e uma binocular. Desta forma, cada célula em MT recebe informações provenientes de um processamento de um conjunto de células simples e complexas em V1 composto de:

1. Duas células simples monoculares em quadratura de fase com campo receptivo no olho direito;
2. Duas células simples monoculares em quadratura de fase com campo receptivo no olho esquerdo;
3. Duas células simples binoculares em quadratura compostas pelas células dos itens 1 e 2;
4. Uma célula complexa monocular, com campo receptivo no olho direito, formada pelas células do item 1;
5. Uma célula complexa monocular, com campo receptivo no olho esquerdo, formada pelas células do item 2;
6. Uma célula complexa binocular formada pelas células do item 3.

Considerando os itens citados como bloco de processamento em V1, a Figura 3-22 mostra um esquema da propagação da informação visual da retina até MT proposta no modelo utilizado por Oliveira [OLI05] para detecção de disparidade. Em camadas de MT com células sintonizadas em disparidades mais elevadas (perto), os campos receptivos provenientes do olho esquerdo estão centrados em posições deslocadas para a direita, já em camadas de MT com células sintonizadas em disparidades mais baixas (longe), estes campos receptivos estão centrados em posições deslocadas para a esquerda. Os campos receptivos provenientes do olho direito não são deslocados. A Figura 3-22 mostra este deslocamento dos campos

receptivos para camadas com disparidade d e $d+1$, juntamente com os blocos de processamento em V1.

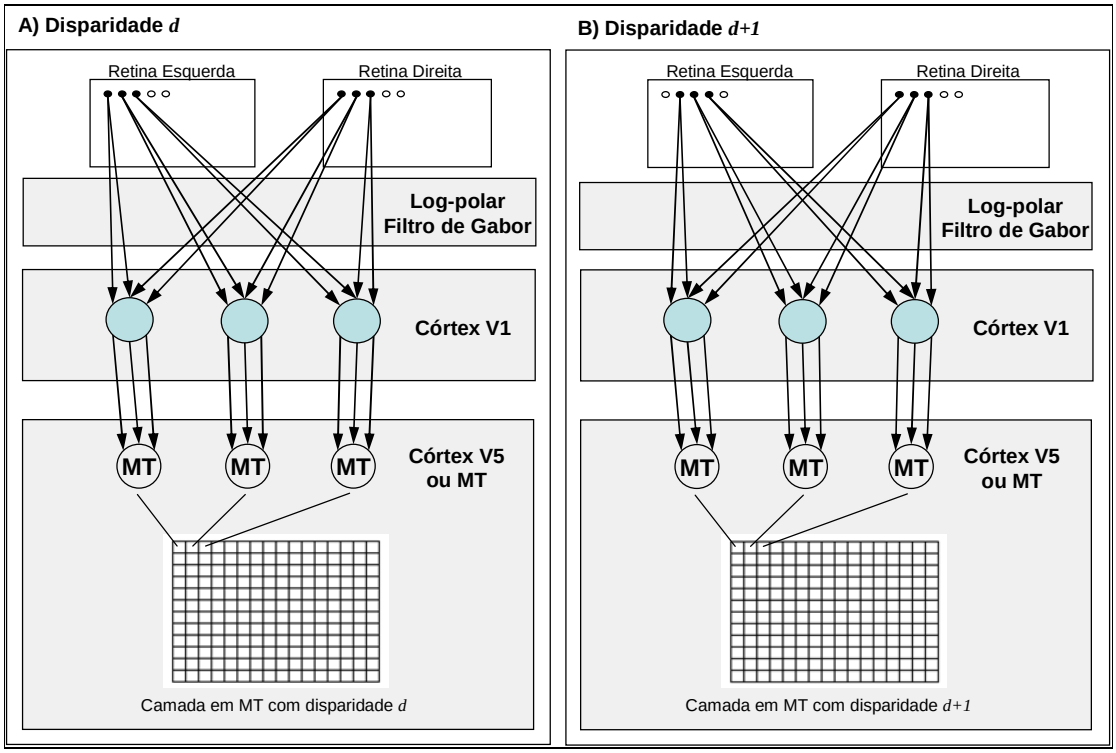

Figura 3-22 – Esquema da arquitetura proposta mostrando que cada célula de MT recebe projeções de um bloco de processamento em V1. A camada com disparidade *d+1* possui os campos receptivos provenientes da retina esquerda centrados em posições deslocadas de uma unidade (*pixel*) em relação à camada com disparidade *d*.

4. IMPLEMENTAÇÃO DO MODELO

Neste capítulo é descrito o processo realizado para computar a disparidade binocular de duas imagens, direita e esquerda, e reconstruir uma imagem tridimensional a partir destas informações. Será discutido o processo desde a captura das imagens por câmeras estéreo (duas câmeras afastadas de uma certa distância e apontadas para uma mesma direção, que imitam os olhos humanos), até o processamento das imagens, incluindo a detecção da disparidade binocular entre elas e a reconstrução tridimensional interna ao computador. Este processo consiste nos seguintes passos:

1. Capturar as imagens direita e esquerda
2. Escolher um ponto de atenção na imagem direita
3. Vergência
4. Calcular a disparidade entre as imagens direita e esquerda
5. Criar um mapa de disparidade binocular com a disparidade mais adequada para cada ponto no mapa
6. Reconstruir a imagem tridimensional a partir do item 5.

De todos os passos citados existe um, o passo 2, que é feito manualmente; os demais são realizados pelos computadores pertencentes à arquitetura do sistema desenvolvido. Os passos 3, 4 e 5 são realizados num mesmo laço (*loop*) de iterações. Como resultado do processamento deste laço é obtido um mapa final de disparidades entre as imagens direita e esquerda. Com este mapa de disparidades é possível realizar um mapeamento inverso, reconstruindo uma imagem tridimensional a partir das imagens bidimensionais capturadas.

Contudo, antes de prosseguir é necessário descrever como é calculada a coordenada de um ponto no espaço a partir da projeção deste ponto na retina direita e esquerda. Após esta descrição serão discutidos os 6 passos do processo de cálculo da disparidade binocular mencionados acima.

4.1. CÁLCULO DA COORDENADA ESPACIAL DE UM PONTO

A vergência constitui um dos movimentos oculares mais importantes e é imprescindível para a percepção de distância em relação aos objetos do mundo no campo de visão. Ela permite direcionar os olhos para um determinado ponto, ou seja, fixar as projeções do ponto em questão sobre as fóveas. Em paralelo, o sistema visual é realimentado com informações sobre o posicionamento dos olhos, fornecidas pelo sistema óculo-motor e as interpreta como distância. Desta forma, o observador é capaz de estimar a distância até um ponto em particular do mundo. O processo através do qual se realiza a vergência será abordado na seção 4.4. Nesta seção será discutida a formulação utilizada para calcular a posição de um ponto no espaço a partir das coordenadas deste ponto projetada nas retinas, que neste caso são as câmeras.

Dada a vergência num ponto qualquer, tem-se a projeção deste ponto nas imagens e, partir daí, localizá-lo no espaço é relativamente simples, a principal dificuldade para viabilizar tal processo é conseguir realizar uma vergência razoavelmente precisa no ponto em questão. A Figura 4-1 ilustra de forma sucinta a formulação desenvolvida para determinar as coordenadas de um ponto no mundo (espaço 3D).

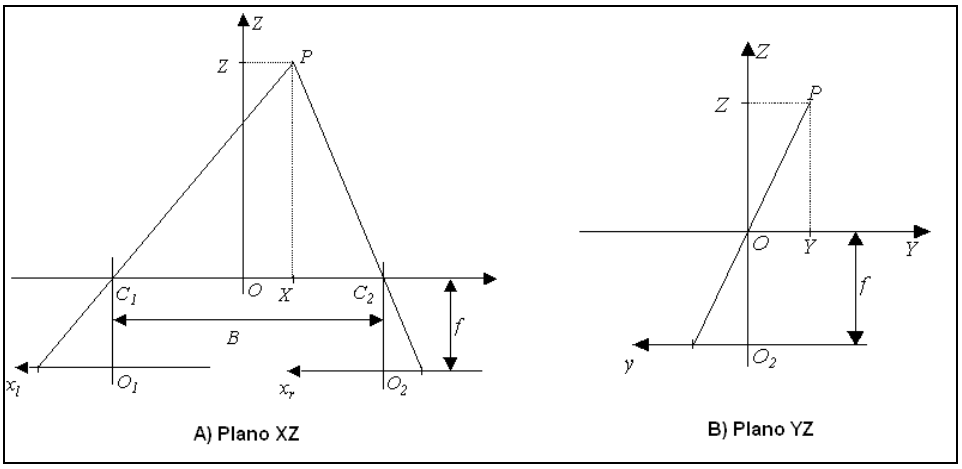

Figura 4-1 – Geometria do cálculo de um ponto no espaço a partir da projeção nas câmeras.

Seja f a distância focal das câmeras medida em *pixels* e B a separação entre elas em metros (Figura 4-1). Por semelhança de triângulos, a coordenada Z do ponto no mundo em relação ao referencial O (Figura 4-1-A) é dada por:

$$Z = \frac{f \cdot B}{x_l - x_r} \tag{16}$$

onde x_l e x_r são as abscissas das projeções do ponto nas imagens direita e esquerda, respectivamente.

De forma análoga, as coordenadas X e Y do ponto *P* do mundo são dadas por:

$$X = \frac{B}{2} \cdot \frac{(x_l + x_r)}{(x_l - x_r)} \tag{17}$$

$$Y = \frac{y \cdot B}{(x_l - x_r)} \tag{18}$$

no qual $y = y_l = y_r$.

4.2. CAPTURA DAS IMAGENS

Todo o processo de computar a disparidade binocular das imagens direita e esquerda, e reconstruir uma imagem tridimensional a partir destas informações é iniciado pela captura das imagens pelas câmeras que estão instaladas paralelamente em cima de um robô [OLI05], separadas a uma distância de 6,9 cm.

Para implementar o modelo do sistema visual humano descrito no (Capítulo 3), cada conjunto de células de um mesmo tipo foi agrupado numa mesma camada de neurônios, ou *neuron_layer*. Filtros foram utilizados para implementar o processamento que é feito na passagem da informação visual de uma camada para outra.

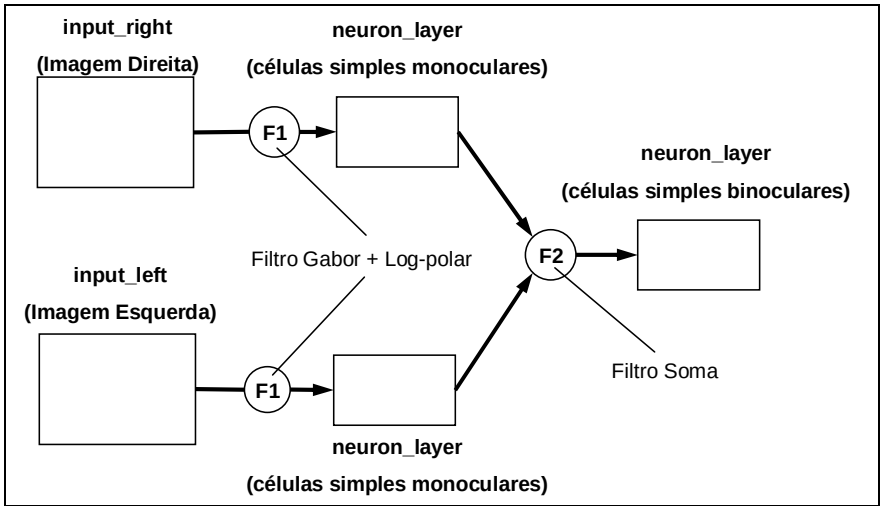

Figura 4-2 – Ilustração da implementação de uma camada de células simples binoculares, utilizando `input`, `neuron_layer` e filtros.

Cada imagem capturada possui resolução de 320x240 *pixels* e 24 bits de cores. Antes de serem processadas, as imagens são convertidas de coloridas para monocromáticas, com escala de cinza de 8 *bits*/*pixel*. Esta conversão é feita para diminuir o volume de informação a ser transmitido e processado. Esta conversão não gera problemas, pois a percepção de cores não é essencial para o processamento, já que cada um destes ocorre numa região diferente do córtex visual. Enquanto o processamento de percepção de profundidade ocorre principalmente em MT (Seção 2.3.5), o processamento de cores ocorre principalmente em V4 (Seção 2.3.4).

4.3. ESCOLHA DO PONTO DE ATENÇÃO

É necessário escolher um ponto para qual será destinada a atenção visual, assim como ocorre na visão humana. Para escolher um ponto de atenção (ponto para onde o cérebro humano direcionará os olhos) na imagem projetada nas retinas, o sistema visual humano realiza o movimento de sacada, que é o movimento rápido dos olhos, com cerca de 900º/s, na direção do ponto de atenção (Tabela 2-1).

Conforme discutido na seção 2.2, o *superior culliculus* é a região do cérebro responsável pelo controle dos movimentos de sacadas dos olhos. Entretanto, este não é o foco deste modelo e, portanto, a escolha dos pontos de atenção é feita manualmente pelo usuário, utilizando o mouse para clicar no ponto desejado da imagem de entrada.

4.4. VERGÊNCIA, DISPARIDADE BINOCULAR E MAPA DE DISPARIDADES

Realizar a vergência significa encontrar nas duas imagens o mesmo ponto para o qual está voltada a atenção. Neste modelo, quando é escolhido um ponto na imagem direita, realizar a vergência significa achar o ponto correspondente a este na imagem esquerda. Calcular a disparidade binocular implica em armazenar em cada camada de MT (específica desta implementação), a medida das diferenças entre as coordenadas dos pontos correspondentes no olho direito e no esquerdo. Construir um mapa de disparidades, significa encontrar, para cada ponto ao redor do ponto de vergência numa imagem, o ponto correspondente na outra imagem, ou seja, para cada ponto na imagem direita, achar o ponto correspondente na imagem esquerda. Tanto para resolver vergência, quanto para construir o mapa de disparidades, é preciso resolver o problema da correspondência (Seção 3.2.2).

Após a escolha do ponto de atenção é necessário vergir os olhos para este ponto, ou seja, movimentar os olhos de forma que o ponto de atenção selecionado seja projetado na fóvea de cada retina (Seção 2.5). A partir do mesmo processo no qual é feita a vergência, obtém-se as informações necessárias para calcular um mapa de disparidade binocular, que é a representação interna e será utilizado posteriormente como base para a reconstrução tridimensional. Portanto, como estes dois processos são realizados em conjunto, serão discutidos nesta mesma seção.

Imediatamente após a escolha manual de um ponto da imagem direita, automaticamente o foco do olho esquerdo é movimento para a posição de mesmas coordenadas x e y na imagem esquerda. Desta forma, os olhos estão “olhando para o infinito”, na direção do ponto escolhido na imagem direita. A partir deste ponto é feita uma varredura no olho esquerdo, a partir deste ponto inicial, movendo para a direita o foco do olho esquerdo, como mostrado na Figura 4-3.

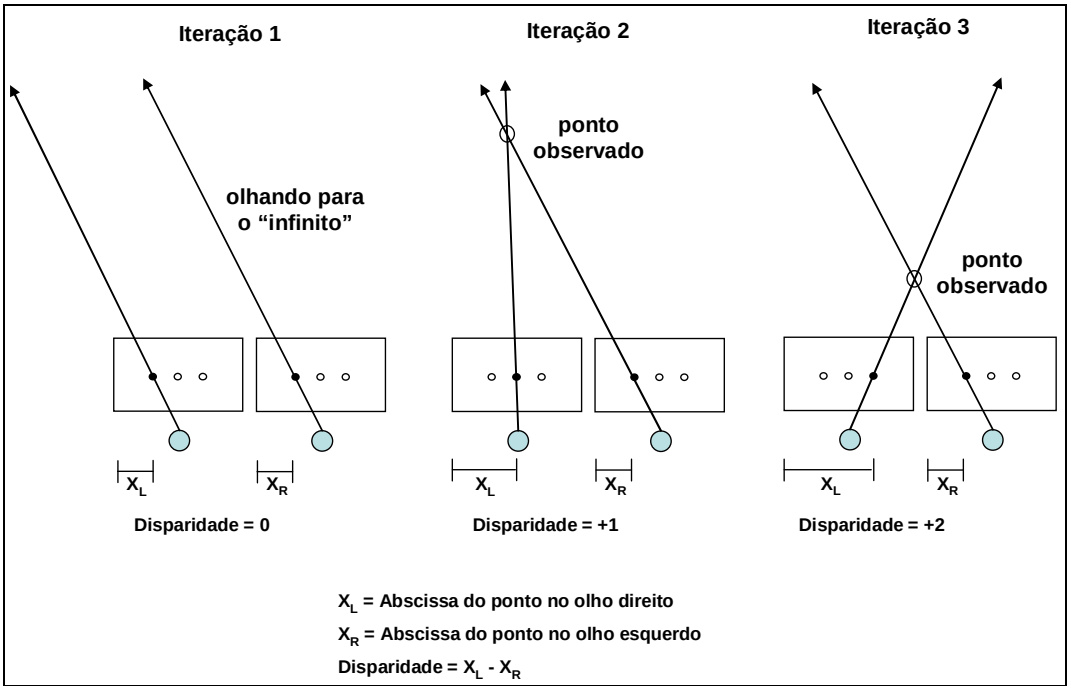

Figura 4-3 – A cada iteração o centro da focalização do olho esquerdo é deslocado um *pixel* para a direita, enquanto o olho direito fica parado. Desta forma a cada iteração são calculadas informações sobre uma disparidade diferente, ou seja, cada vez focalizando (vergindo os olhos) mais perto.

Em cada iteração é construída uma camada em MT calculando a resposta de cada célula desta camada utilizando a arquitetura proposta na seção 3.4. Se na iteração n foram calculadas as respostas das células da camada d em MT, na iteração $n+1$, serão calculadas as respostas das células da camada $d+1$, e assim sucessivamente (Figura 3-22). Após o cálculo de todas as iterações, e consequentemente, todas as camadas de MT, é obtida a estrutura de MT de acordo com a Figura 3-20. Note que esta forma seqüencial de computar as camadas de MT é resultado de sua implementação computacional, mas o modelo é totalmente paralelo.

A partir da informação contida na área cortical MT implementada é feita a vergência e construído um mapa de disparidade binocular que será a base para a reconstrução tridimensional. Cada camada da MT implementada é uma matriz com o resultado de um conjunto de células com mesma disparidade, isto é, para uma célula numa camada com uma disparidade d, o centro do campo receptivo na imagem direita é definido pelas coordenadas (x_R, y_R), e o centro do campo receptivo na imagem esquerda é dado pelas coordenadas (x_L, y_L), para uma célula nas mesmas condições, porém na camada $d+1$, as coordenadas dos centros dos campos

receptivos nas imagens direita e esquerda são dadas pelas coordenadas (x_R, y_R) e (x_L+1, y_L) respectivamente. Na Figura 3-22 é mostrado este deslocamento, com 3 células para cada camada.

Na condição de vergência o ponto visualizado no espaço é projetado na fóvea de cada retina. Como este ponto está projetado em posições correspondentes em cada retina, a disparidade é nula. As células de MT modeladas neste caso são *tuned inhibitory*, cujas respostas são mínimas quando um estímulo está sendo projetado em posições equivalentes nos seus campos receptivos direito e esquerdo e, portanto, não há disparidade entre os campos receptivos direito e esquerdo. Dessa forma, para calcular a coordenada de vergência, é calculado o somatório das respostas das células de cada camada em MT, e é selecionada a coordenada (x,y) da camada em MT que possui o menor somatório, ou seja, a camada que possui a melhor resposta para disparidade nula (Figura 4-4 e Figura 4-5).

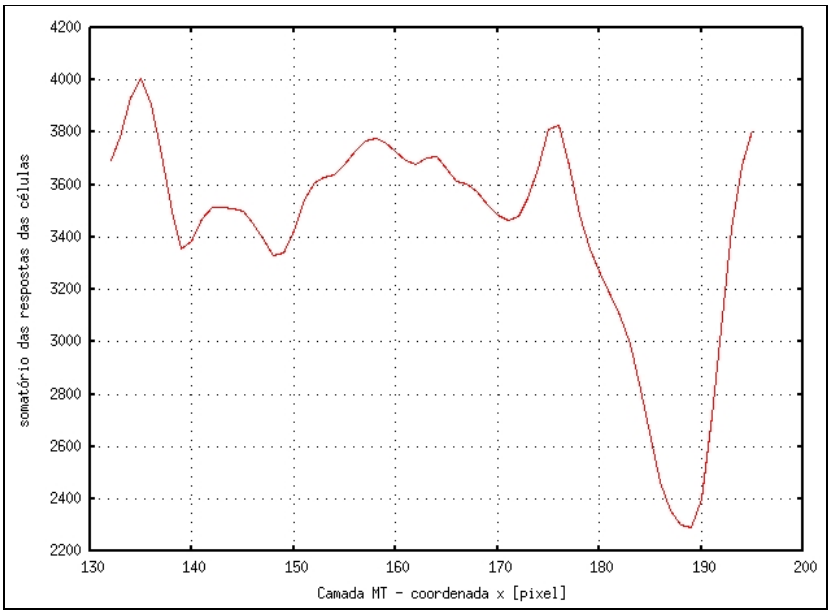

Figura 4-4 – Gráfico mostrando a resposta total de cada camada de MT versus a coordenada (na verdade, a disparidade) associada a cada camada para as imagens de entrada da Figura 4-5 (a disparidade zero é a da coordenada x = 132) . Neste exemplo o ponto de vergência está na coordenada x = 189, ou seja, das 64 camadas de MT modeladas (132 à 195), a camada relativa a coordenada 189 teve o menor somatório da resposta total das células, indicando o ponto de vergência.

Figura 4-5 – Imagens (320x240) direita e esquerda de uma mesma cena. A cruz vermelha indica a posição para onde o olho está olhando, ou seja, o ponto da imagem projetado diretamente na fóvea. (A) Ao escolher um ponto na imagem direita, coordenada [132,135], o olho esquerdo se move para a posição correspondente, coordenada [132,135], fazendo com que o observador esteja "olhando para o infinito". (B) Após o processo de vergência os dois olhos estão orientados de forma que o ponto escolhido esteja sendo projetado diretamente na fóvea de cada retina. O olho direito continua na coordenada [132,135], enquanto que o olho esquerdo moveu para a coordenada [189,135] que é o ponto correspondente ao ponto focalizado pelo olho direito [132,135].

Para construir o mapa de disparidade binocular entre as imagens, o procedimento é semelhante ao da vergência e, além disto, são utilizadas as mesmas informações que foram utilizadas para calcular o ponto de vergência. Na vergência é preciso encontrar a coordenada (o plano de disparidade *d* desta modelagem MT) no qual toda a imagem possui a menor disparidade (com destaque para a fóvea, devido ao mapeamento log-polar), portanto é selecionada a camada com o menor valor total (menor somatório das respostas das células), enquanto que para construir o mapa de disparidades é preciso encontrar, ponto a ponto, a menor disparidade (Figura 4-6).

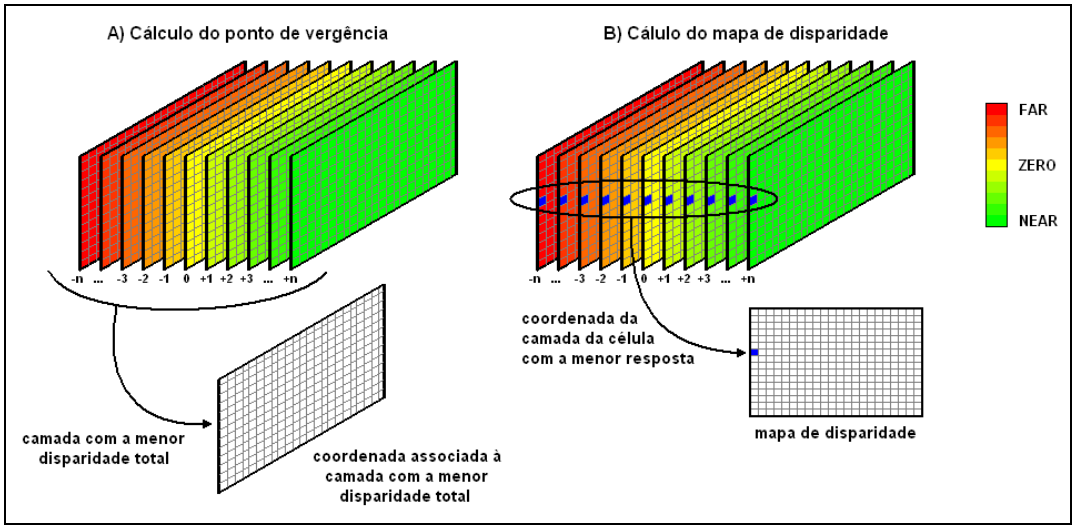

Figura 4-6 – No cálculo da vergência é selecionada a coordenada da camada com a menor disparidade total (a). Na construção do mapa de disparidades, para cada conjunto de células correspondentes entra as camadas, é selecionada a disparidade da célula com a menor resposta (b).

Em cada camada ocorre uma cooperação entre as células, ou seja, a resposta de uma célula é influenciada positivamente pelas respostas das células vizinhas (ver *spatial pooling*, Seção 3.3, página 62), enquanto que entre as camadas ocorre uma competição entre as células. Uma ilustração de como ocorre esta interação entre as células, tanto de camadas iguais quanto de camadas diferentes pode ser vista na Figura 4-7.

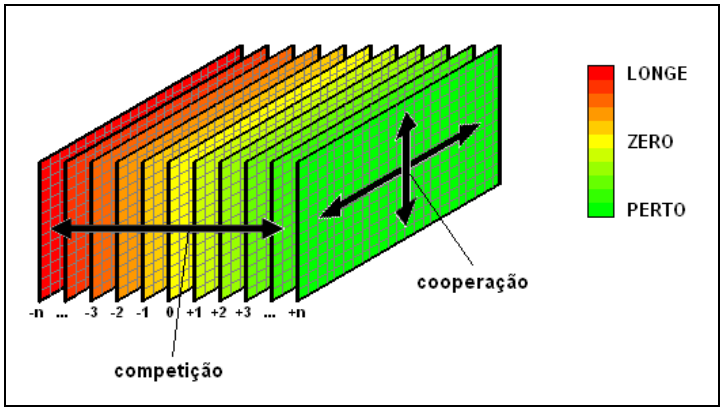

Figura 4-7 – Maneira com interagem as células de MT. Em cada camada, as respostas das células influenciam nas repostas de suas vizinhas (cooperação entre as células). Entre as camadas há uma competição para selecionar tanto cada célula individualmente com a menor disparidade (mapa de disparidade), quanto uma camada inteira com a menor disparidade (vergência).

Após cada iteração do loop que calcula cada camada desta modelagem MT, o mapa de disparidades (Figura 4-2) é atualizado. Os valores colocados no mapa a cada iteração são as coordenadas x (abscissas), na imagem, da célula em MT com menor valor de saída, ou seja, há uma competição entre as células. Assim, após o computo de todas as camadas de MT, o mapa de disparidades fica somente com valores positivos (abscissas das coordenadas de cada camada na qual estava a célula com a menor resposta). Esta é a disparidade absoluta entre as imagens. Contudo, a reconstrução tridimensional através da disparidade binocular se baseia na disparidade com relação ao ponto de vergência. Para se obter a disparidade com relação ao ponto de vergência é subtraído de cada ponto no mapa de disparidade o valor da abscissa da coordenada do ponto de vergência, obtendo-se, assim, um mapa de disparidades relativo ao ponto de vergência. Na Figura 4-8 é mostrado o mapa de disparidades antes e depois da compensação da vergência.

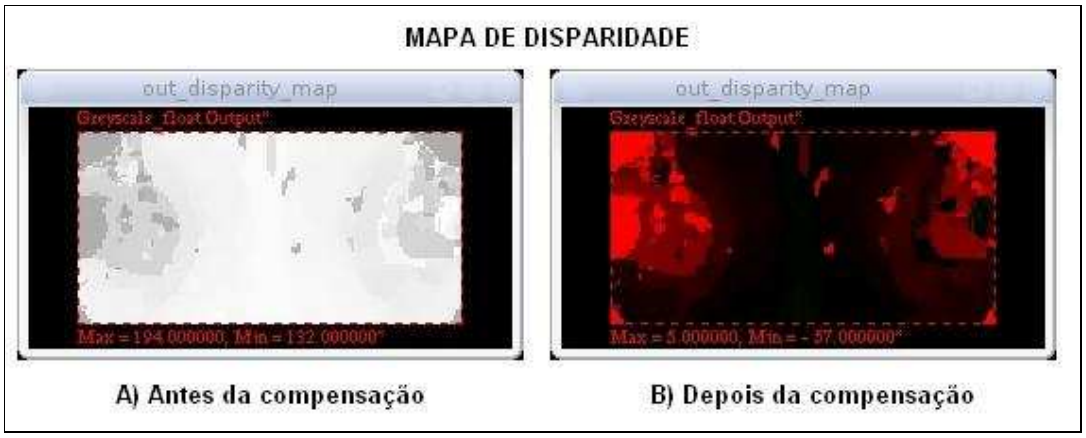

Figura 4-8 – Figura mostrando a compensação feita no mapa de disparidade após a vergência. (A) Antes da vergência, o mapa possui as disparidades associadas as coordenadas de cada uma das 64 camadas de MT, que neste caso variam de 132 à 194. (B) A vergência (neste caso) foi escolhida pela camada de coordenada 189, portanto, todo o mapa foi compensado, subtraindo a coordenada de vergência, onde a disparidade deve ser igual a zero, tendo então disparidades variando de -57 à +5.

4.5. RECONSTRUÇÃO DA IMAGEM TRIDIMENSIONAL

A informação tridimensional da imagem do mundo externo, projetada nas retinas direita e esquerda, está codificada no mapa de disparidade, após todo o processamento descrito na seção anterior. O mapa de disparidades contém a

disparidade entre os campos receptivos nas câmeras esquerda e direita de cada célula vencedora de MT.

Na Figura 4-9-A é mostrado como são projetados no espaço 3 pontos consecutivos com a mesma disparidade e, na Figura 4-9-B, é mostrado como um mesmo ponto na imagem direita pode ser projetado em diferentes pontos do espaço dependendo de sua disparidade.

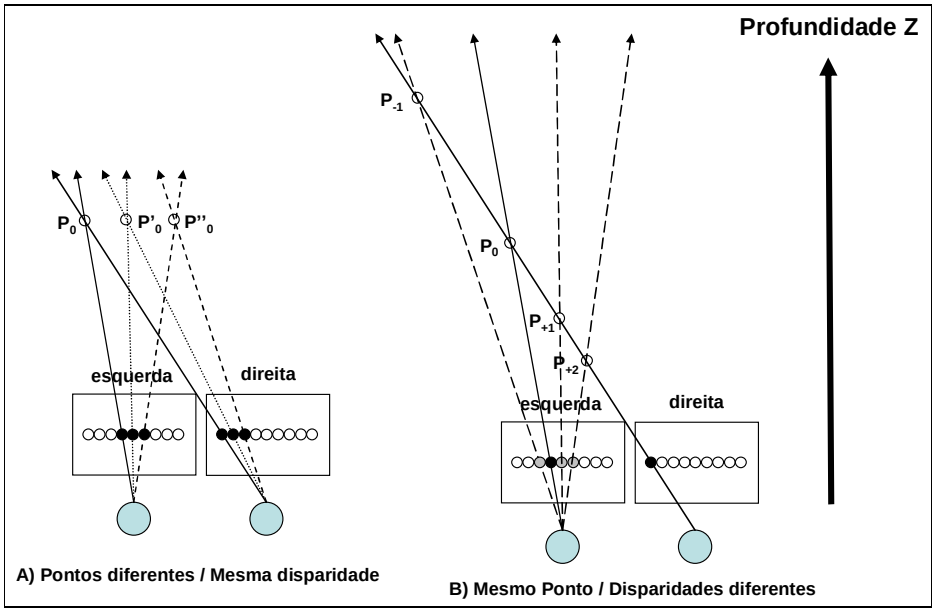

Figura 4-9 – Projeções de pontos diferentes no espaço mas com a mesma disparidade (A). Projeção do mesmo ponto no espaço considerando diferentes disparidades (B).

A reconstrução tridimensional, baseada nas imagens capturadas pelas duas câmeras, interna ao computador é baseada no mapa de disparidades, no qual toda a informação visual, desde as células simples em V1, sofreu uma transformação log-polar. Entretanto, o mapa de disparidade contém a disparidade nas retinas. Portanto, se pode calcular os pontos no espaço tridimensional externo a partir do mapa de disparidades é preciso seguir os seguintes passos para cada ponto do mapa:

1. Realizar o mapeamento log-polar inverso do córtex (mapa) para as retinas esquerda e direita.
2. Adicionar a disparidade associada a este ponto, deslocando horizontalmente o ponto na retina esquerda.
3. Calcular a coordenada espacial deste ponto.

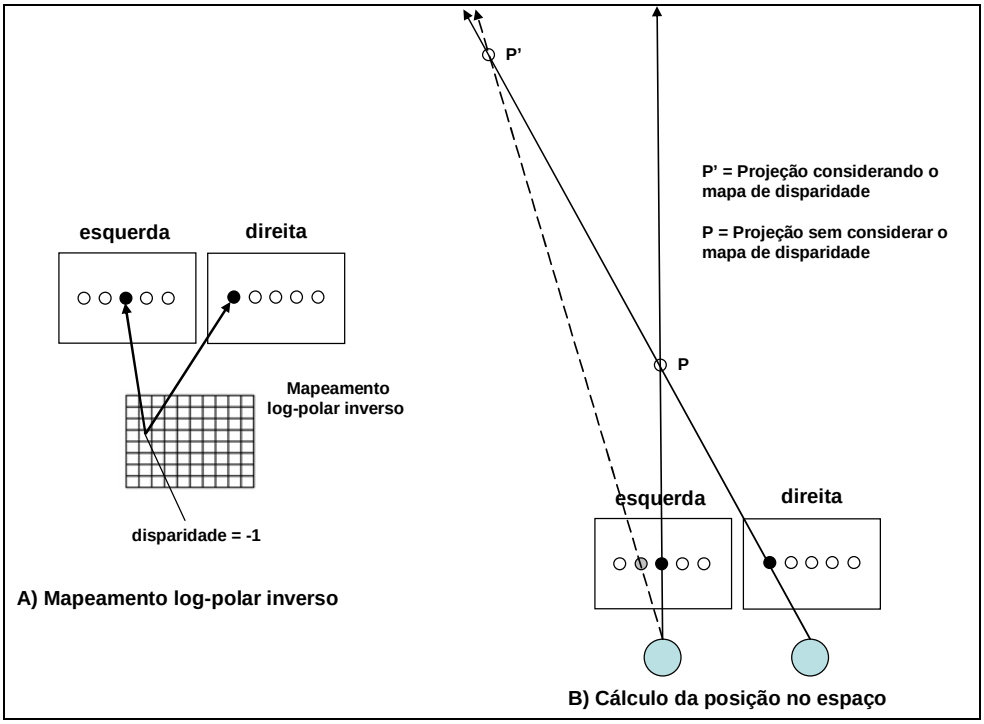

Figura 4-10 – Mapeamento inverso de um ponto no córtex para as retinas direita e esquerda (A). Calculando a posição deste ponto no espaço, é obtido P. O ponto P' é a projeção considerando uma disparidade = -1, no qual determina que a projeção deste ponto na retina esquerda deve ser deslocada um *pixel* para a esquerda.

Na Figura 4-10-A, mostra o exemplo de mapeamento log-polar inverso (Córtex → Retina) de um ponto qualquer no mapa de disparidade para as retinas (na verdade, câmeras) sem levar em consideração a disparidade, mas apenas a vergência, isto é, tratando o ponto como tendo disparidade zero, inicialmente. Com a projeção nas retinas, é possível calcular a posição deste ponto no espaço usando a formulação discutida na seção 4.1, obtendo-se assim o ponto P (Figura 4-10-B). Entretanto, de acordo com o mapa de disparidades, a disparidade deste ponto é -1 (Figura 4-10-A). Isto quer dizer que, na verdade, a projeção deste ponto na retina esquerda está deslocada -1 *pixel* na horizontal (1 *pixel* para a esquerda). A projeção na retina direita não é alterada. Desta forma, recalculando a posição do ponto no espaço, é obtido o ponto P' (Figura 4-10-B).

4.6. CINCO AMOSTRAS

Numa situação ideal, a disparidade de um ponto no espaço tridimensional corresponde ao ponto de mínimo global desta função. Contudo, num ambiente real, escolher a disparidade associada à célula de MT menos responsiva para um ponto no mapa nem sempre vai corresponder a determinar a disparidade correta para este ponto. Isto pode ser explicado pela diferença de brilho, contraste e foco entre as imagens esquerda e direita, que vão afetar as respostas das células da arquitetura de camadas neurais que culmina em MT, afetando as respostas das células em MT. Vários fatores podem interferir nas relações entre o brilho, contraste e foco de regiões correspondentes na imagem esquerda e direita, como, por exemplo:

- As câmeras utilizadas para captura das imagens, apesar de serem exatamente do mesmo modelo e estarem configuradas da mesma forma podem ter uma sensibilidade diferente à luz.
- Variação na iluminação ambiente
- Diferença na quantidade luz que incide em cada câmera

Na maioria dos casos observados com este problema, a disparidade correta, apesar de não estar associada ao mínimo global, está associada a um mínimo local, com amplitude similar ao mínimo global. Assim, foi utilizada uma heurística baseada no modelo ***the winner takes it all***, no qual a resposta de um ponto no mapa de disparidades tem que estar de acordo com as respostas dos pontos vizinhos.

Esta heurística consiste em, para cada ponto do mapa de disparidade, armazenar um conjunto ($CONJ$) de pares com as informações de intensidade da resposta para uma disparidade ($I(d)$) e esta disparidade associada (d) dos cinco menores mínimos da função, podendo assumir como disparidade correta para este ponto, qualquer uma das disparidades de um destes cinco mínimos.

$$CONJ = \{(d_i, I(d_i))\}_{1 \le i \le 5} \tag{19}$$

Inicialmente, a disparidade de um ponto no mapa de disparidades (d_p) é selecionada como sendo igual à disparidade do menor dentre os cinco mínimos armazenados, ou seja, o mínimo global.

$$d_p = d_i \left| I(d_i) = \min_{1 \le k \le 5} \{ I(d_k) \} \right. \quad (20)$$

Em seguida são feitas algumas iterações de forma a fazer com que cada ponto do mapa de disparidades fique em conformidade com a média dos 8 pontos vizinhos. Uma discrepância muito acentuada entre esses valores indicaria algum tipo de ruído no resultado.

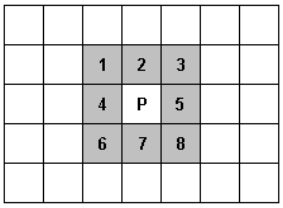

Figura 4-11 – Os oito pontos vizinhos de um ponto P específico.

Em cada iteração, para cada ponto no mapa de disparidades, é calculada a média (I_M) da intensidade da resposta das oito células vizinhas ao ponto, e escolhida uma nova disparidade, dentre as cinco previamente armazenadas que minimize a distância para esta média. Essa medida evita que a presença de mínimos globais fruto de ruido influencie na eleição da disparidade correta.

$$d_p = d_i \left| I(d_i) = \min_{1 \le k \le 5} \{ | I(d_k) - I_M | \} \right. \quad (21)$$

Estas iterações são executadas para todo o mapa até atingir a convergência, no qual nenhum ponto muda o valor de uma iteração para outra, ou atingir um número máximo de iterações predefinidas. Na Figura 4-12 é mostrada um mapa de disparidades antes deste processamento e após todas as iterações. É possível verificar visualmente a melhora no mapa de disparidades, após as iterações, no qual várias regiões ruidosas foram eliminadas. Esta heurística equivale a iterações possivelmente existentes na área MT biológica.

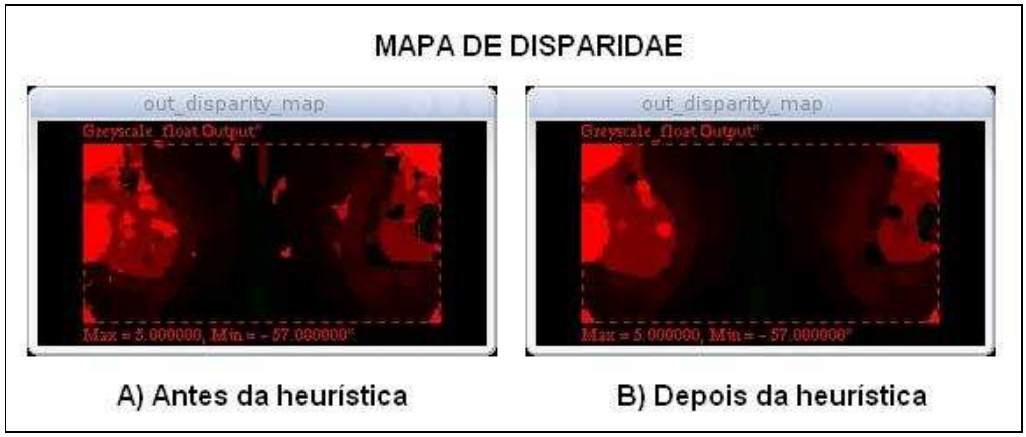

Figura 4-12 – Mapa de disparidade antes e depois de utilizar uma heurística baseada em *'The winner takes it all'* para melhorar a conformidade do mapa de disparidades. É nítida a eliminação de algumas inconformidades do mapa após a utilização da heurística.

4.7. ESTRUTURA DE ARMAZENAGEM DA RECONSTRUÇAO TRIDIMENSIONAL

É preciso armazenar toda a reconstrução tridimensional numa estrutura de dados na qual seja possível utilizar todas as informações para projetar os pontos no espaço e permitir que essa possa ser utilizada, em trabalhos futuros, como base para a navegação de um robô. Quanto mais simples esta estrutura, menor será o esforço computacional para percorrê-la e utilizá-la para a navegação de um robô.

Apesar da reconstrução ser tridimensional, foi utilizada uma estrutura (**estrutura TMap**) bidimensional para armazená-la. Isto porque, quando se olha para um objeto, obtém-se uma informação tridimensional sobre a superfície visível deste objeto, não se podendo afirmar, a princípio, nada acerca do que está por trás desta superfície. Desta forma, a estrutura TMap foi modelada como uma matriz bidimensional de pontos que representa a superfície visível da sua janela de visão, no qual cada ponto possui os seguintes atributos:

- Intensidade luminosa
- Distância
- Ângulo horizontal (α)
- Ângulo vertical (β)

Cada ponto na estrutura bidimensional TMap representa um ponto no espaço a uma direção fixa a partir de um ponto central imaginário entre as duas câmeras do robô. A posição do ponto no espaço associado a cada ponto da estrutura é codificada em coordenadas polares, e representada por α e β e distância. A distância é obtida a partir da disparidade e geometria das câmeras, e os ângulos α e β são dados pelas Equações (22), nas quais n_H e n_V são os índices horizontal e vertical em TMap, H e V são as dimensões horizontal e vertical da estrutura TMap, e FOV_H e FOV_V são os ângulos que representam a abertura dos campos de visão horizontal e vertical das câmeras respectivamente (Figura 4-3).

$$\begin{cases} \alpha = -\dfrac{FOV_H}{2} + \dfrac{n_H . FOV_H}{H} \\ \beta = -\dfrac{FOV_V}{2} + \dfrac{n_V . FOV_V}{V} \end{cases} \tag{22}$$

Nesta estrutura é possível armazenar qualquer superfície visível que esteja no campo de visão, bastando que a distância de cada ponto seja ajustada de forma adequada para coincidir com a superfície. No caso em que todos os pontos estão à mesma distância do centro das câmeras, é obtida uma superfície que representa uma casca esférica. Quanto maior a distância de um ponto representado no TMap, menor a sua resolução, assim como ocorre na visão humana.

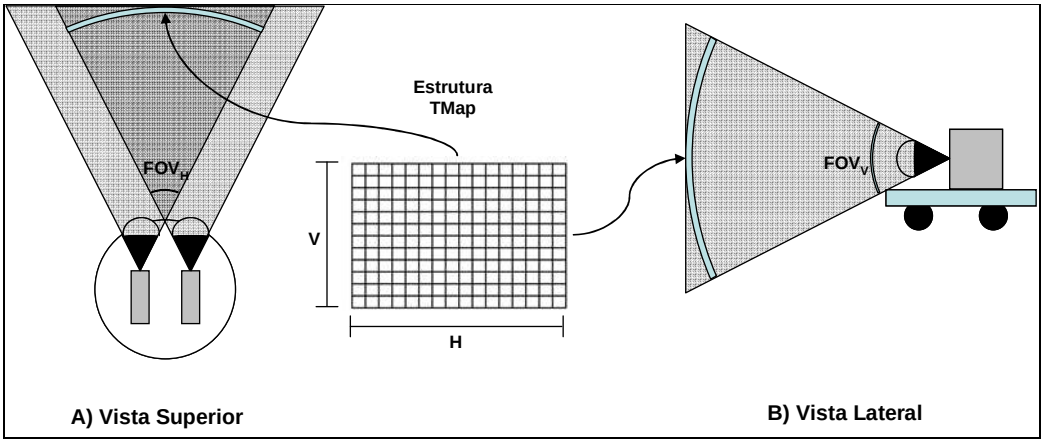

Figura 4-13 – Representação de como a estrutura TMap é utilizada.

A Figura 4-14 mostra um par de imagens estéreo capturas pelo robô, o mapa de disparidades computado e a reconstrução tridimensional vista de três ângulos.

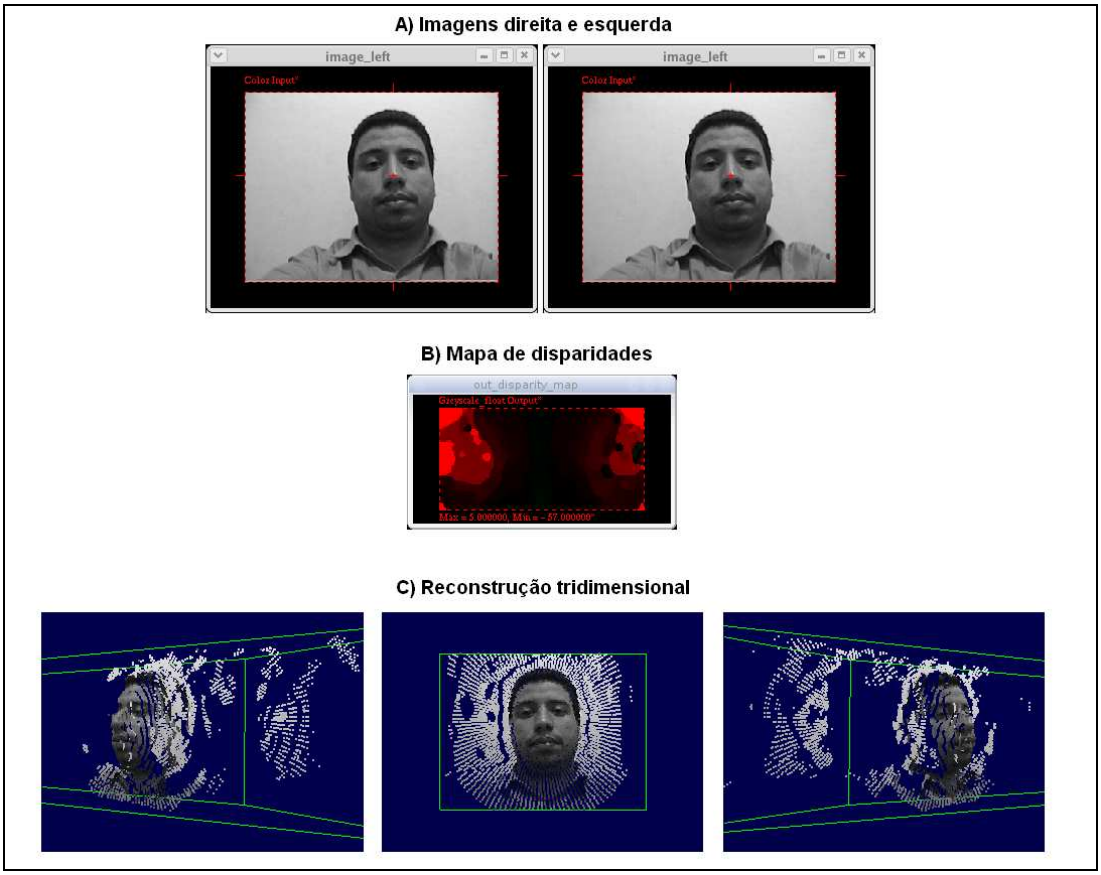

Figura 4-14 – Reconstrução 3D a partir de um par de imagens estereoscópicas.

4.8. COMPENSAÇÃO DO EFEITO DA DISCRETIZAÇÃO

A utilização de imagens matriciais em todo este processo leva a um problema de discretização na representação da imagem tridimensional, que afeta fortemente o eixo Z (profundidade) devido a própria discretização da imagem. Este efeito pode ser visto na Figura 4-9 e na Figura 4-10, no qual apenas um *pixel* de diferença afasta ou aproxima a projeção do ponto no espaço consideravelmente. Isto é agravado pelo fato do mapa de disparidades obtido possuir somente valores inteiros, uma vez que a disparidade está expressa nele em *pixels*. O impacto da discretização é tão mais expressivo quanto maior for a distância do ponto no mundo.

Para reduzir este efeito e produzir representações tridimensionais mais realistas foi utilizada novamente a técnica de *spatial pooling* mostrado na seção 3.3.

A Figura 4-15 mostra o resultado da suavização no mapa de disparidade obtida aplicando o *spatial pooling* após sua construção.

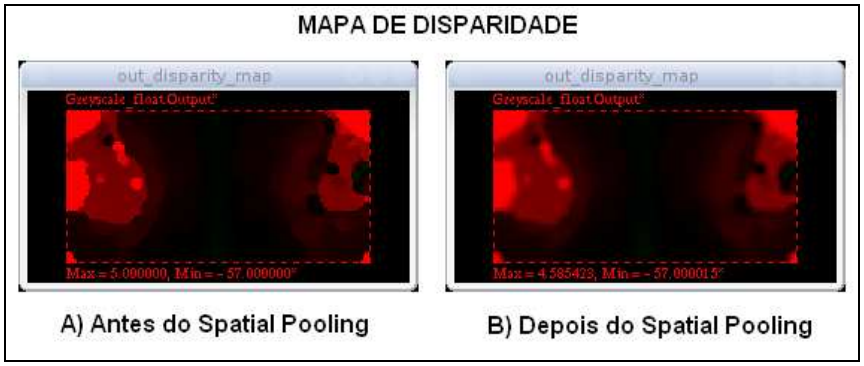

Figura 4-15 – Resultado da utilização do *spatial pooling* após a construção do mapa de disparidades. As transições entre as disparidades que eram abrutas (apenas disparidades inteiras), são suavizadas pelo *spatial pooling*, semelhante ao efeito de um filtro passa baixa.

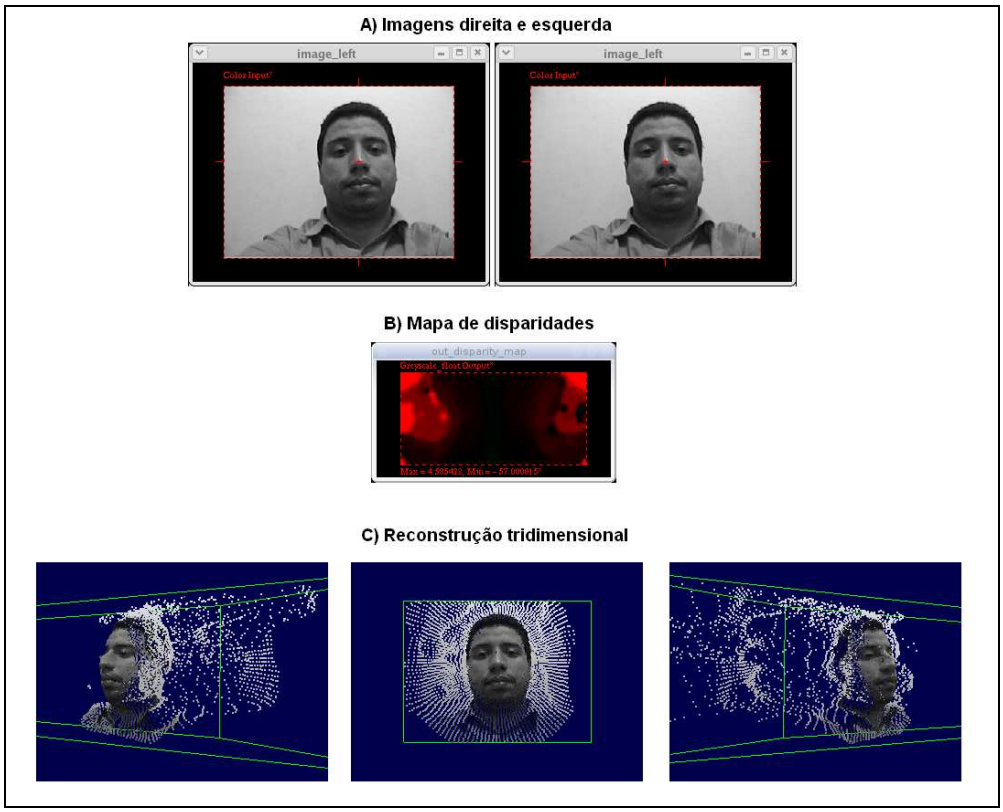

Figura 4-16 – Reconstrução tridimensional a partir de um par de imagens estereoscópicas. A técnica *spatial pooling* suaviza o mapa de disparidades de melhora significativamente a reconstrução tridimensional.

5. PROGRAMAÇÃO PARALELA EM C+CUDA

Com a evolução e o resultante grande poder computacional das *Graphics Processor Units* (GPUs), elas passaram a ser usadas para vários propósitos. Em 2006, a NVIDIA lançou a arquitetura CUDA (*Compute Unified Device Architecture*). A partir daí, o uso das GPUs para processamento não gráfico ganhou força na comunidade científica, sendo publicados muitos trabalhos sobre sua aplicação em diversas áreas. Em alguns casos, o desempenho chega a ser 100 vezes maior que o de *Central Processor Units* (CPUs).

Além do grande desempenho, GPUs ocupam pouco espaço, são fáceis de instalar e configurar, e são mais baratas se comparadas a um *cluster*, por exemplo. Uma GPU GeForce GTX 280, que atualmente é a topo de linha da NVIDIA, possui desempenho máximo teórico de 933 GFlop/s [NVI08b], enquanto que o *cluster* antigo do Laboratório de Computação de Alto Desempenho (LCAD), com 64 computadores Athlon 1800, chegava a apenas 195,8 GFlop/s. A grande limitação das GPUs em comparação com um *cluster* é a quantidade de memória. A GTX 280 possui 1GB, enquanto que o cluster antigo do LCAD possuía 16 GB [LCA09, NVI08b].

A NVIDIA (principal fabricante de GPUs) criou CUDA para facilitar a programação e o uso das placas de vídeo na execução de trechos de código de programas de propósito geral. Muito embora GPUs sejam diferentes de CPUs no que diz respeito à arquitetura e à forma de programar, elas já vêm sendo usadas em aplicações não gráficas há algum tempo [FUN04].

5.1. HISTÓRICO

Em 1987, a IBM lançou no mercado o primeiro *chip* gráfico chamado de *Video Graphics Array* (VGA). Ele era conhecido como *dumb frame buffer*, pois as atualizações de todos os *pixels* eram feitas pela CPU [RAN03].

Antes das primeiras gerações de GPUs, as empresas Silicon Graphics (SGI) e Evans & Sutherland desenvolviam *hardwares* gráficos especializados mais inteligentes. Essas companhias desenvolveram vários conceitos, tais como transformação de vértices e mapeamento de textura. Entretanto, os *hardwares* gráficos não tiveram muita aceitação, devido a seu custo elevado [RAN03].

Por volta de 1998, vieram as primeiras GPUs modernas, incluindo a NVIDIA TNT2, ATI Rage e as 3dfx Voodoo3. Estas GPUs eram capazes de pré-transformar os triângulos em que são discretizadas imagens gráficas e aplicar uma ou duas texturas a estes triângulos. Elas também eram implementadas com as características do padrão DirectX 6, da Microsoft[1]. No entanto, estas GPUs não eram capazes de transformar vértices de objetos 3D; assim, estas transformações precisavam ser feitas pela CPU [RAN03].

Entre 1999 e 2000, a NVIDIA lançou as GPUs GeForce 256 e GeForce 2, a ATI a Radeon 8500, e a S3 a Savage3D. Estas GPUs eram capazes de realizar a transformação de vértices 3D e manipulação de luz, tirando estas tarefas da CPU. A transformação rápida dos vértices foi uma das principais capacidades que diferenciou as GPUs desta geração das anteriores. Por outro lado, muito embora essas GPUs pudessem ser configuradas, elas ainda não eram verdadeiramente programáveis [RAN03].

Em 2001, a NVIDIA apresentou a Geforce 3 e a ATI a Radeon 9500. Esta geração passou a proporcionar a programação de operações sobre vértices em vez de simplesmente oferecer mais configurabilidade. Em 2003, a NVIDIA lança sua quinta geração de GPUs, a Geforce FX, e a ATI a Radeon 9700. Esta geração passou a ser programável no nível de *pixel* e vértices. O lançamento da linguagem Cg nessa mesma época facilitou essa programação, uma vez que ela é bastante parecida com a linguagem de programação C. A partir daí, alguns pesquisadores passam a usá-las para processamentos não gráficos. Contudo, programar trechos de código de programas para propósito geral utilizando Cg se mostrou muito complexo, já que os programas só podiam ser feitos dentro do contexto das *Application Programming Interfaces* (APIs) de OpenGL e DirectX [RAN03].

Em 2006, a NVIDIA lançou no mercado as novas GPUs, série 8, com *shader* unificado (vários passos do *pipeline* da GPU, como, por exemplo, *vertex shader* e *pixel shader*), que receberam a denominação de *stream processors* [NVI06]. Nesta mesma época, a NVIDIA também lançou a extensão CUDA da linguagem C para a programação de GPUs CUDA *enabled*. Com tanto poder computacional, e agora com uma linguagem de programação mais simplificada, as GPUs passaram a ser

[1] http://msdn.microsoft.com/pt-br/directx/default(en-us).aspx

usadas não só para processamento gráfico, mas também para computação de trechos de código de programas para propósito geral [HAL08].

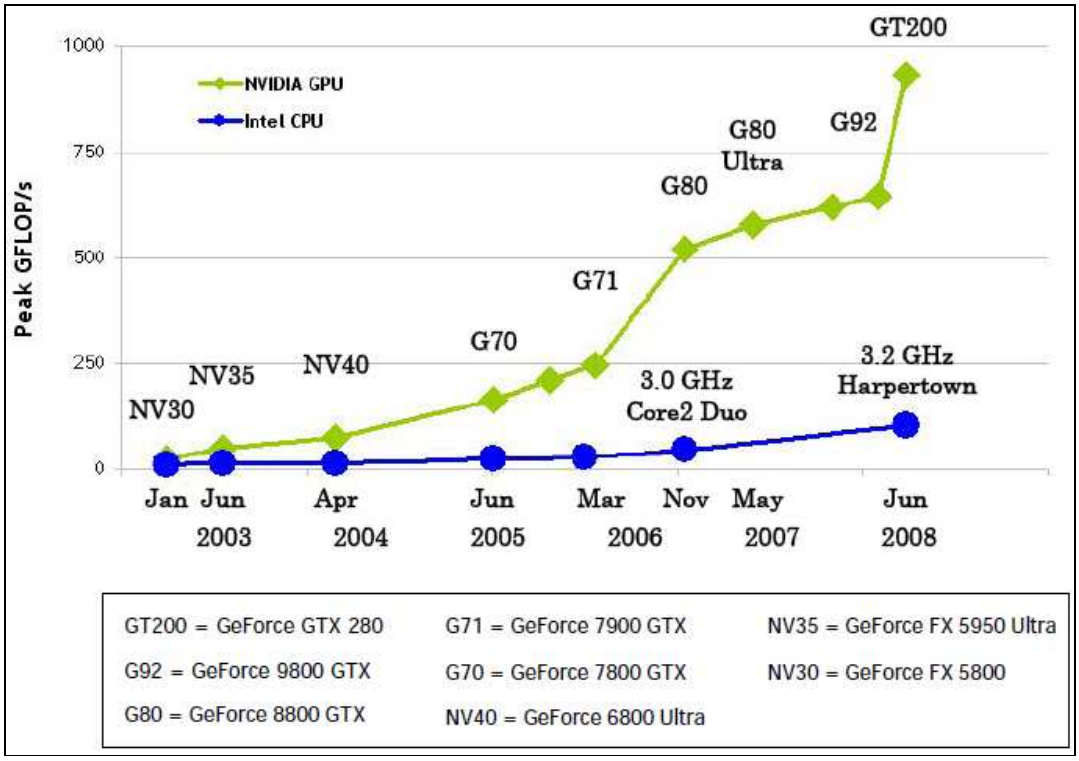

Figura 5-1 – Evolução das GPUs com relação às CPUs [NVI08b].

Em poucos anos, o poder computacional bruto das GPUs superou em muito o das CPUs, como mostra a Figura 5-1. Com vários *cores* e uma grande largura de banda de memória (Figura 5-2), elas se tornaram poderosos recursos tanto para processamento gráfico quanto para o de trechos de código de programas de propósito geral [NVI07]. A Figura 5-2 mostra como a taxa de transferência dos dados pelo barramento de memória das GPUs cresceu; este crescimento foi necessário, pois as GPUs têm *caches* pequenas.

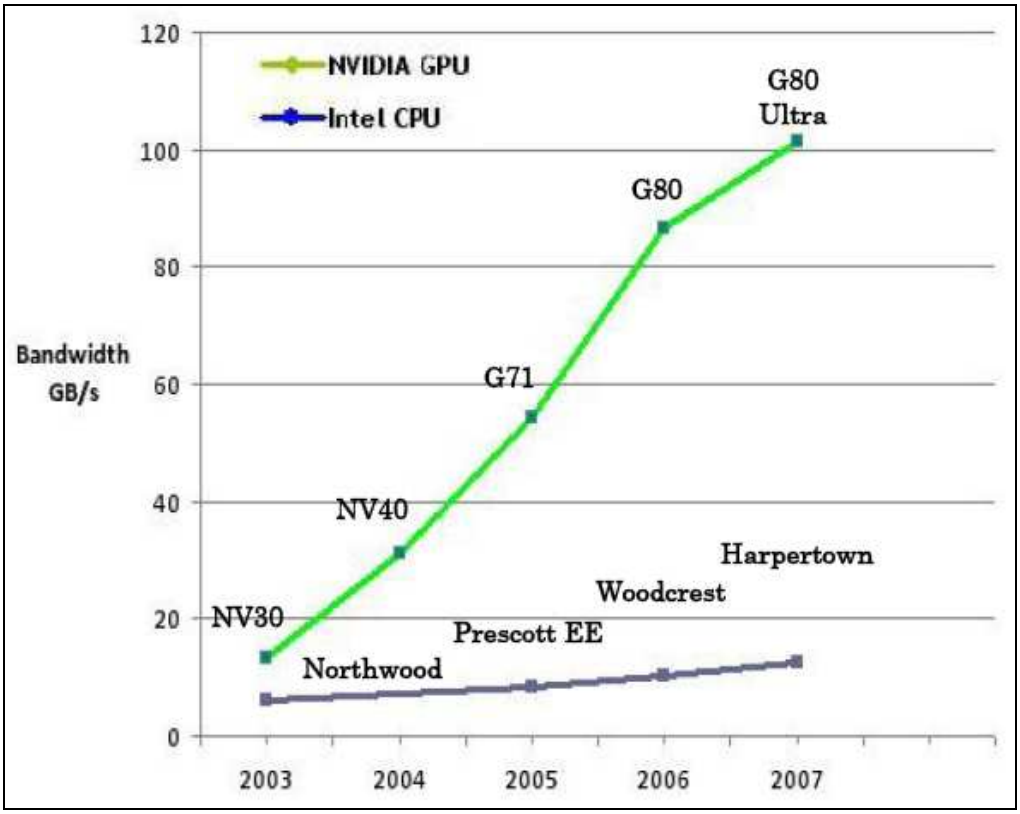

Figura 5-2 – Evolução da banda de memória [NVI08b].

5.2. ARQUITETURA

As Geforce séries 8 e 9, e as da série GTX 200 usam vários *Stream Processors* (SPs) para executar operações inteiras e de ponto flutuante. Os SPs são muito eficientes, pois executam operações sobre um *stream* de entrada, enquanto produzem um *stream* de saída que pode ser usado por outros SPs. Os SPs são agrupados em *Stream Multiprocessors* (SMs) que, por sua vez, são agrupados em *Texture/Processor Clusters* (TPCs) para prover um grande poder de processamento paralelo. A Figura 5-3 mostra um TPC composto por 2 SMs com 8 SPs cada [NVI06].

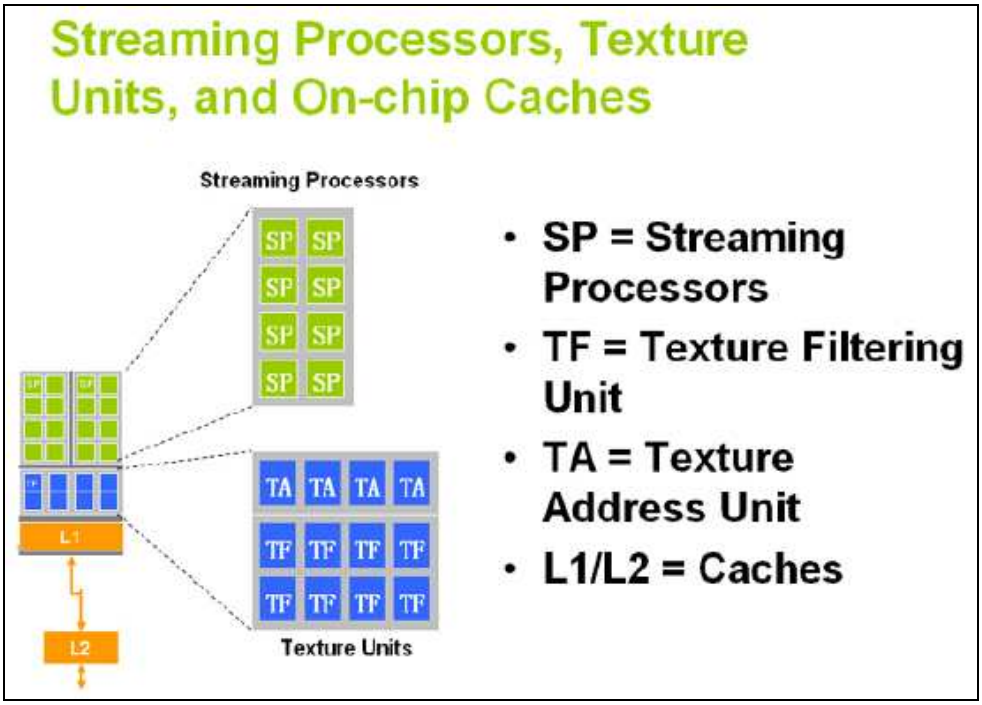

Figura 5-3 – Arquitetura de um *Stream Multi-Processor* [NVI06].

Os TPCs possuem hardware capaz de decodificar e enviar instruções para seus SPs em grande velocidade. Operações similares são executadas em diferentes elementos de um *data stream*. A memória, tipicamente usada para armazenar a saída de um SP, pode ser lida como entrada por outros SPs. Tudo isso é controlado por instruções *Single Instruction/Multiple Thread* (SIMT) que controlam agrupamentos de SPs de maneira eficiente [NVI06].

A placa gráfica GTX 280, que foi usada neste trabalho, possui GPU com 10 TPCs, cada um com 3 SMs, totalizando 30 SMs (Figura 5-4). Cada SM possui 8 SPs, totalizando 240 SPs, que operam com um clock de 1.4 GHz. A memória principal da placa é uma GDDR3 de 1GB, clock de 2.322 MHz, barramento de 512 bits e a taxa de transmissão de 142 GB/s.

Milhares de *threads* podem ser despachadas para execução dentro de uma GPU CUDA. O hardware CUDA mantém os SPs ocupados usando seu escalonador (*Thread Scheduler*, ver Figura 5-4), que despacha estas *threads* para execução em paralelo [NVI06].

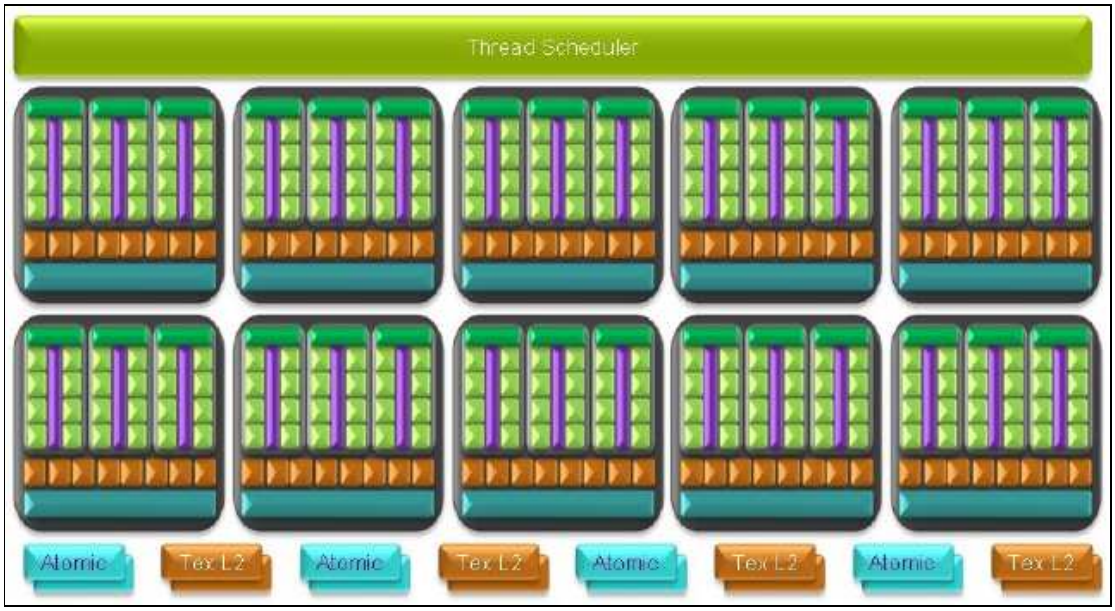

Figura 5-4 – Arquitetura da GeForce GTX 280 [NVI08a].

Uma GPU GTX 280 tem três áreas de armazenamento, além da memória principal (*global memory*), quais sejam: a *constant memory* (memória de constantes) de 65536 bytes; a *shared memory* (memória compartilhada), que é compartilhada pelas *threads* de um bloco de *threads*, onde cada bloco pode alocar até 16384 bytes; e os *registers*, que compreendem 16384 registradores de 32 bits [NVI08a, NVI08b].

5.3. PROGRAMAÇÃO EM C+CUDA

A programação em C+CUDA é viabilizada por uma pequena extensão da linguagem C e por uma nova biblioteca C. A Figura 5-5 apresenta os níveis de abstração de um programa C+CUDA.

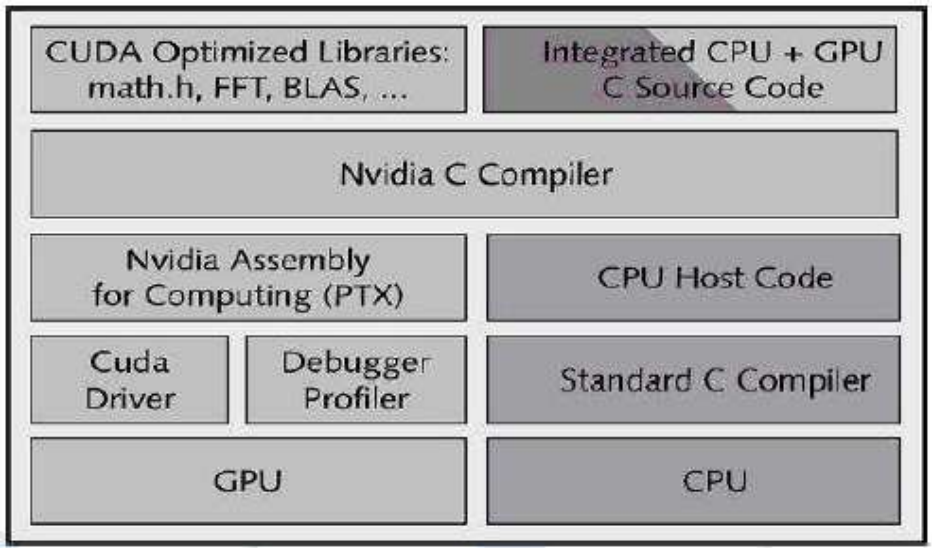

Figura 5-5 – Arquitetura da linguagem C+CUDA [HAL08].

Como mostra a Figura 5-5, no nível mais alto, um programa fonte em C+CUDA é especificado em C mais comandos específicos para GPU, e pode incluir chamadas a funções das bibliotecas otimizadas CUDA da Nvidia. Um fonte em C+CUDA deve ser compilado pelo *NVIDIA C Compiler*, que gera código *assembly* para o *assembler* da NVIDIA e código em C para ser compilado por um compilador C padrão (gcc, por exemplo). O código *assembly* traduzido para código de máquina de GPU é levado à GPU pelo driver da placa de vídeo CUDA *enabled* sob demanda do código de máquina de CPU produzido pelo compilador C.

A GPU (*device*) é vista pela CPU (*host*) como um co-processador capaz de executar um número muito grande de *threads* em paralelo [NVI07]. Tanto o *host* como o *device* mantém memória (*Dynamic Random Access Memory* - DRAM) própria, chamadas de *host memory* e *device memory*. Os dados podem ser copiados de forma otimizada de uma DRAM para a outra através de chamadas à biblioteca CUDA [NVI07].

5.4. DIFERENÇAS ENTRE C+CUDA E C PADRÃO

Em um programa C+CUDA, funções usuais da linguagem C, que são chamadas e executadas pela CPU, são chamadas de funções *host*. Além das funções *host*, podem existir dois novos tipos de função em programas C+CUDA: funções *kernel* e funções *device*. Funções *kernel* são executadas pelo *device* e

invocadas pelo *host*. Para especificar que uma função é deste tipo é necessário utilizar o qualificador "__**global**__", que deve ser inserido antes do tipo de retorno da função, que deve ser sempre *void*. Funções *device* são chamadas e executadas somente pelo *device*. Seu qualificador é "__**device**__", que também deve ser colocado antes do tipo de retorno da função. Neste tipo de função é permitido o retorno de qualquer tipo [NVI08a].

Um programa em C+CUDA especifica uma hierarquia de *threads*. Esta hierarquia é formada por um *grid*, que pode ter até duas dimensões de blocos de *threads*, sendo que os blocos podem ter até três dimensões de *threads*, conforme mostrado na Figura 5-6. A função "__**syncthreads()**" funciona como uma barreira e permite fazer o sincronismo de *threads* de um mesmo bloco [NVI08a].

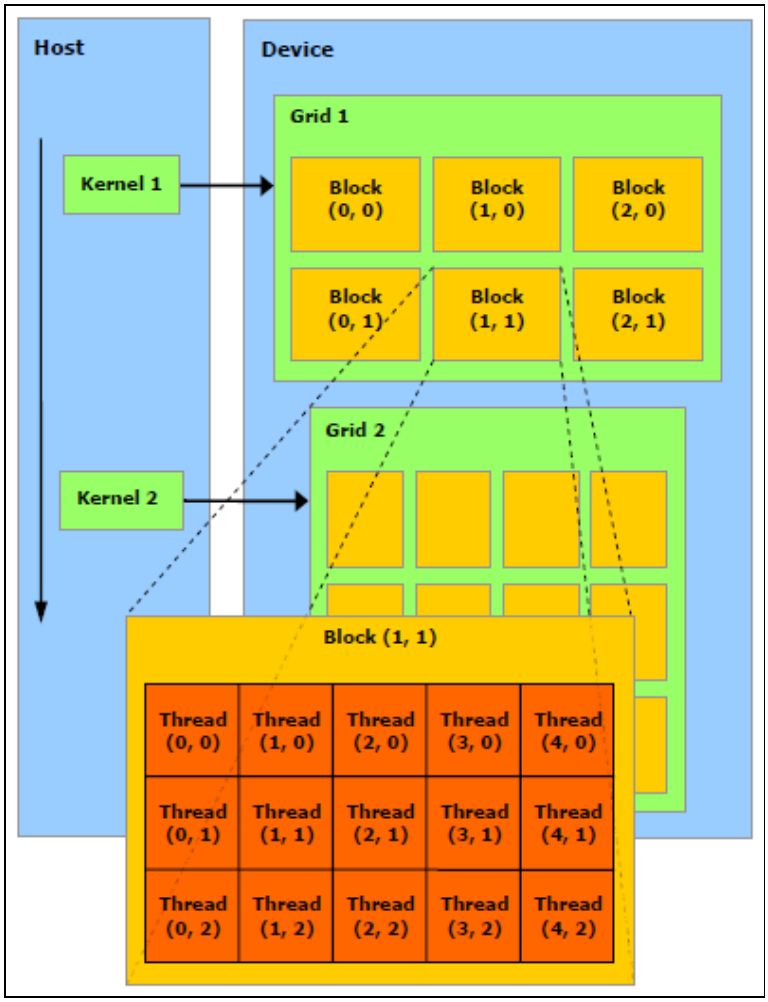

Figura 5-6 – Hierarquia de *threads* em C+CUDA [NVI07].

C+CUDA oferece ao programador autonomia para usar os três tipos de memória presentes na GPU por meio de qualificadores do tipo das variáveis empregadas. O qualificador __**device**__ especifica que uma variável deve ser alocada na *global memory*, o qualificador __**constant**__ especifica que uma variável deve ser alocada na *constant memory*, e o qualificador __**shared**__ especifica que uma variável deve ser alocada na *shared memory*. Uma variável automática definida em uma função *device* geralmente reside em um registrador (estas variáveis podem ser movidas para outras regiões de memória se não houver registradores suficientes).

Os registradores são de acesso exclusivo de cada *thread*. A *shared memory* é compartilhada por todas as *threads* de um bloco e, apesar de pequena, é muito importante, pois possui baixa latência de acesso. A *constant memory* também possui baixa latência e pode ser acessada por todas as *threads* de um *grid*, mas apenas para leitura – apenas o *host* pode escrever nesta memória. A *global memory* é o local onde o *host* publica os dados que serão processados. Ela é compartilhada por todas as *threads* de um *grid*, mas possui alta latência (400 a 600 ciclos de *clock* para uma leitura). Existe outra memória específica para aplicações gráficas, a *texture memory* (memória de textura), que é uma memória somente de leitura [NVI08a].

A Figura 5-7 mostra um trecho simples de código seqüencial em C e uma versão paralela do mesmo em CUDA. Na figura, __**global**__ indica que a função `saxpy_paralelo` é um *kernel* e deve ser executada na GPU. Em C+CUDA, *kernels* são invocados por meio de chamadas de função estendidas com o formato:

```
kernel <<<dimGrid, dimBlock>>> (parâmetros);
```

onde *dimGrid* e *dimBlock* são vetores de três elementos do tipo *dim3* (parte da API de CUDA) que especificam as dimensões dos *grids* e *blocks*, respectivamente. Dimensões não especificadas são consideradas 1 [NVI08a].

Computação serial de *y* ← *ax* + *y*, *y* e *x* vetores e *a* escalar

```
void saxpy_serial (int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; i++)
        y[i] = a * x[i] + y[i];
}

// Invoca o kernel serial SAXPY
saxpy_serial (n, 2.0, x, y);
```

Computação paralela de *y* ← *ax* + *y* em C + CUDA

```
__global__
void saxpy_paralelo (int n, float a, float *x, float *y)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) y[i] = a * x[i] + y[i];
}

// Invoca o kernel paraelo SAXPY com 256 threads por bloco
int nblocks = (n + 255) / 256;
saxpy_paralelo <<<nblocks, 256>>> (n, 2.0, x, y);
```

Figura 5-7 – Exemplo de código em C+CUDA.

No exemplo da Figura 5-7, o comando:

```
saxpy_paralelo <<<nblocks, 256>>> (n, 2.0, x, y)
```

invoca uma *grid* unidimensional com *nblocks blocks* unidimensionais, cada um com 256 *threads* (por simplicidade, funções da API para alocação de memória e transferência de dados da CPU para a GPU e vice-versa não são mostradas). Cada uma destas *threads* computa um elemento do vetor resultado. Note que os valores de *blockIdx.x*, *blockDim.x*, *threadIdx.x*, que identificam unicamente cada *thread*, são globais ao *kernel* e computados automaticamente durante a execução a partir dos parâmetros usados ao invocar o *kernel*. Note, também, que o código da Figura 5-7 funciona para qualquer *n*, limitado apenas pelo *hardware* da GPU.

GPUs hoje disponíveis possuem dezenas de SMs e cada um pode executar *blocks* com até 512 *threads*, sendo o número de *blocks* por *grid* limitado apenas pela memória disponibilizada à GPU. Tipicamente, centenas de *thread blocks* são possíveis, o que resulta em dezenas de milhares de *threads* por *kernel*. O custo de criar e chavear entre estas *threads* é extremamente baixo – poucos ciclos de relógio.

A placa gráfica GTX 280 empregada neste trabalho permite criar 512 *threads* por bloco, que podem ser distribuídas em três dimensões x, y e z (a dimensão z pode somente endereçar até 64) – quando são declaradas 16 *threads* em x e 16 em y, na verdade serão criados 256 *threads*. Pode ser criado um grande número de blocos também em 2 dimensões – a GTX 280 pode suportar até 30.720 *threads on the fly*. Ela pode processar dados em precisão simples e dupla, mas o desempenho máximo em precisão dupla é de 80 GFlop/s, enquanto que o em precisão simples é igual à 933 GFlop/s [NVI08b].

O grande número de threads por SM, dezenas de SMs por GPU e o baixo custo do chaveamento entre threads (o que oculta a latência de memória) viabilizam o modelo de computação massivamente paralelo e de alto desempenho de C+CUDA.

6. IMPLEMENTAÇÃO DO MODELO EM C+CUDA

Neste capítulo será descrito como se deu a investigação e posterior implementação paralela do modelo matemático-computacional das áreas corticais associadas a percepção de profundidade proposto por Oliveira [OLI05]. Para tanto, alguns conceitos básicos serão definidos a princípio.

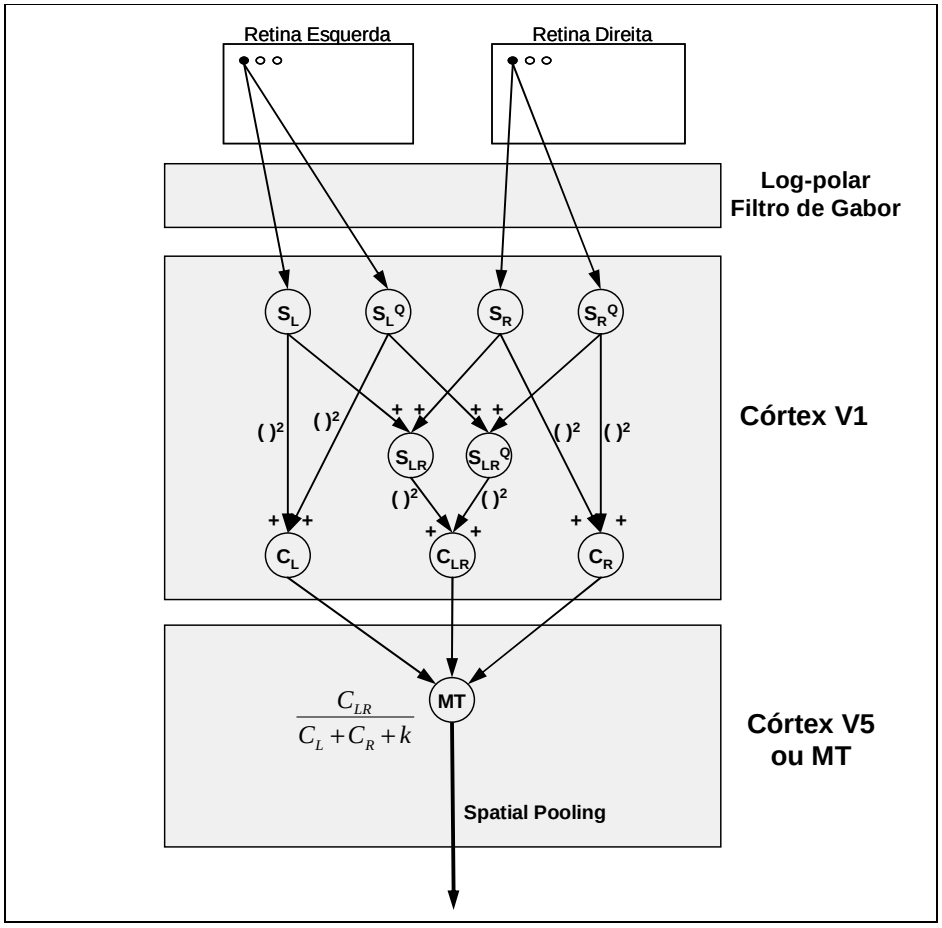

Figura 6-1 – Modelo dos córtices associados à percepção de profundidade.

A Figura 6-1 é idêntica à Figura 3-21, e foi replicada aqui para benefício do leitor. Cada uma das células do modelo mostrado na Figura 6-1 foi implementada, seqüencial e paralelamente, como uma camada bidimensional de neurônios onde cada um possui valores de entrada e um valor de saída computado por uma função em C (versão seqüencial) ou C+CUDA (versão paralela). Nós denominamos estas

funções de *filtro*. Cada um destes filtros reproduz o comportamento de uma camada bidimensional de neurônios de um determinado tipo.

As características de cada filtro guiaram a forma como foi elaborada a estratégia de paralelização de código utilizada neste trabalho. Cada um destes filtros foi objeto de tentativas cada vez mais elaboradas de paralelização, na busca por resultados que refletissem em ganhos de desempenho.

Cada um dos filtros desenvolvidos representa parte da modelagem do sistema visual feita por Oliveira [OLI05]. Todas as partes desta modelagem foram apresentadas no Capítulo 3 e, portanto, serão apenas mencionadas superficialmente neste capítulo.

6.1. PRÉ-PROCESSAMENTO

Em qualquer aplicação que seja alvo de investigação para aprimoramento do desempenho de execução, os primeiros passos do processo dizem respeito aos usos dos recursos de entrada/saída do sistema. Escritas em arquivos ou mesmo nos *prompts* de comando, *plots* gráficos na tela, entre outras coisas, podem responder por uma boa parte do tempo de processamento de uma aplicação. Desta forma, a primeira providência neste trabalho foi remover todos os *displays* gráficos que permitiam a visualização das mudanças ocorridas nas camadas neurais em todas as etapas de processamento.

Outra providência tomada foi a inserção de um contador de tempo para avaliar o desempenho geral da execução. Este contador de tempo foi incorporado à função `set_vergence` da aplicação (ver código disponível em http://www.lcad.inf.ufes.br/svn/MAE/examples/robot_cuda/robot_user_funcions/robot_user_functions.c), que coordena o ciclo de interações dos filtros no processo de reconstrução 3D.

6.2. FILTROS CORRESPONDENTES ÀS ÁREAS V1 E MT

Os primeiros filtros a serem abordados foram escolhidos segundo o critério de simplicidade de implementação, podendo facilmente ser reconhecidos. Foram eles `robot_sum_filter`, `robot_complex_cell` e `robot_mt_cell`. Destes, os dois

primeiros representam funções desempenhadas por células presentes na camada V1 do córtex visual, e o terceiro pelas células da camada MT.

6.2.1. `cuda_sum_filter`, `cuda_complex_cell` e `cuda_mt_cell`

O filtro `robot_sum_filter` modela as células simples binoculares (S_{LR} e S_{LR}^{Q}, Figura 6-1) a partir de células simples monoculares (S_L e S_L^{Q}, e S_R e S_R^{Q}, respectivamente) por meio da soma de seus resultados. Na implementação atual, cada uma destas células é representada por um neurônio na respectiva camada neural, sendo as dimensões da mesma iguais a 128 linhas e 64 colunas[2], ou seja, uma camada de 8192 neurônios que precisam fazer 8192 somas a cada reconstrução 3D. A estrutura de dados que representa os valores dos neurônios da camada é um vetor de números ponto-flutuante (precisão simples, i.e., do tipo *float*). Desta maneira, a cada atualização dos filtros que implementam o modelo, o filtro `robot_sum_filter` soma dois vetores de *floats*. A Figura 6-2 apresenta o *kernel* CUDA que implementa o filtro.

```
__global__ void
cuda_add (float *d_C, float *d_A, float *d_B, int num_neurons)
{
      int idx = blockIdx.x * blockDim.x + threadIdx.x;
      if ( idx < num_neurons )
            d_C[idx] = d_A[idx] + d_B[idx];
}
```

Figura 6-2 – Kernel `cuda_sum_filter`.

Na Figura 6-2, **__global__** indica que a função `cuda_add` é um *kernel* e deve ser executada na GPU. Ela computa a soma dos vetores `d_A` e `d_B` e escreve o resultado em `d_C`. Na verdade, a função `cuda_add` é chamada uma vez para cada elemento de `d_A`, `d_B` e `d_C`. Cada uma destas chamadas é uma *thread* que executa em um SP da GPU. Para identificar que elemento dos vetores cada SP deve manipular, a variável `idx` é utilizada. Ela é computada a partir de `blockIdx.x`, `blockDim.x` e `threadIdx.x`. Muito embora pareçam variáveis comuns,

[2] Estes parâmetros podem ser facilmente modificados e são informados aqui apenas para que o leitor tenha idéia do volume de operações envolvido.

`blockIdx.x`, `blockDim.x` e `threadIdx.x` são computados automaticamente a partir dos parâmetros de chamada do *kernel* durante o despacho das instruções SIMT para os SPs da GPU (ver Seção 5.2). A Figura 6-3 mostra como `blockIdx.x`, `blockDim.x` e `threadIdx.x` são empregadas para identificar um dos elementos dos vetores d_A, d_B e d_C.

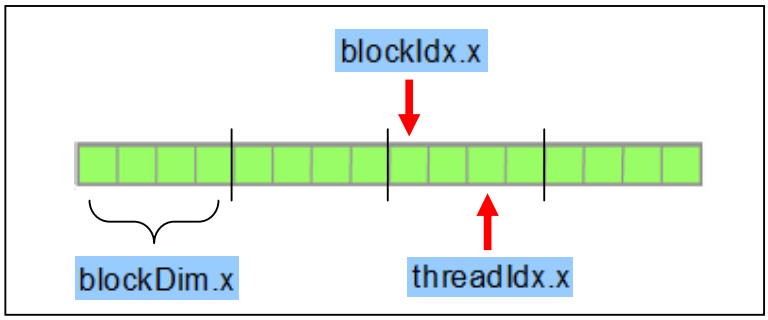

Figura 6-3 – Uso de `blockIdx.x`, `blockDim.x` e `threadIdx.x` para especificar um elemento de um vetor.

Como a Figura 6-3 mostra, `blockDim.x` é a dimensão x de cada bloco de *threads* invocado para executar `cuda_add`[3]; `blockIdx.x` é o número do bloco, ou seja, um índice para o início de um dos blocos; e `threadIdx.x` é o número de uma *thread* de um dos blocos, i.e., um índice para um dos elementos de um dos blocos.

Threads podem ser invocadas para executar o *kernel* `cuda_add` por meio das linhas de código abaixo:

```
dim3 dimBlock (BLOCKSIZE);
dim3 dimGrid ((num_neurons/dimBlock.x) + (!(num_neurons%dimBlock.x)?0:1));

cuda_add <<<dimGrid, dimBlock>>> (d_C, d_A, d_B, num_neurons);
```

onde: a primeira linha define a quantidade de *threads* de um bloco (`dimBlock`) por meio da constante `BLOCKSIZE`[4], que no nosso caso é igual a 64[5]; a segunda a quantidade de *grids*, que é calculada a partir do tamanho da estrutura de dados enviada para a GPU (`num_neurons`, que é igual a 8192) e do valor de `dimBlock` (que é igual a 64, ou seja, o número de *grids* de blocos é igual a 8192 / 64 = 128); e

[3] Se os blocos fossem bi- ou tri-dimensionais, as variáveis `blockDim.y` e `blockDim.z` poderiam ser utilizadas no código para acessar o tamanho das dimensões y e z dos blocos, respectivamente.
[4] A sintaxe desta linha é própria de código C+CUDA.
[5] Os blocos possuem, então, 64 *threads*, e `dimBlock` apenas uma dimensão (x), mas pode possuir até três (y e z).

a terceira linha invoca 128 *grids*, de 64 blocos de *threads* cada (um total de 8192 *threads*), para executar `cuda_add`.

De maneira muito semelhante foi implementado outro filtro da camada V1 abordado nesta etapa, o filtro `cuda_complex_cell`. Ele modela as células complexas monoculares C_L e C_R (Figura 6-1, pág. 98) a partir das células simples monoculares S_L e S_L^Q, e S_R e S_R^Q, respectivamente. Este mesmo filtro foi utilizado para modelar as células complexas binoculares C_{LR}, uma vez que ela possui a mesma funcionalidade das células C_L e C_R, possuindo apenas entradas diferentes, as células simples binoculares S_{LR} e S_{LR}^Q (ver Figura 6-1). A modelagem matemática do filtro `cuda_complex_cell` é definida como a soma dos quadrados dos neurônios de entrada, e foi implementada em CUDA conforme mostra a Figura 6-4.

```
__global__ void
cuda_mult_add (float *d_C, float *d_A, float *d_B, int num_neurons)
{
       int idx = blockIdx.x * blockDim.x + threadIdx.x;
       if ( idx < num_neurons)
              d_C[idx] = (d_A[idx] * d_A[idx]) + (d_B[idx] * d_B[idx]);
}
```

Figura 6-4 – Kernel `cuda_complex_cell`.

Os procedimentos necessários para invocar o filtro `cuda_complex_cell` são equivalentes ao do filtro anterior (`cuda_sum_filter`), inclusive as definições de *grids* e *blocks*.

Por fim, apresentamos o filtro `cuda_mt_cell`, que modela o comportamento de células não mais da camada V1 do córtex, mas sim da camada MT. Este filtro computa a saída das células MT (Figura 6-1, pág. 98) a partir das células complexas resultantes do processamento anterior, C_L, C_{LR} e C_R. Mesmo não estando presente na mesma esfera de modelagem dos filtros anteriores, `cuda_mt_cell` foi abordado neste momento em função da semelhança de sua implementação em CUDA com as dos filtros anteriores. Este filtro realiza a divisão do valor das células complexas binoculares pela soma dos valores das células complexas monoculares e de um parâmetro de seletividade. Sua implementação em CUDA é apresentada na Figura 6-5.

```
__global__ void
cuda_div (float *d_D, float *d_A, float *d_B, float *d_C, int num_neurons,
            float k_param)
{
       int idx = blockIdx.x * blockDim.x + threadIdx.x;
       if ( idx < num_neurons)
              d_D[idx] = d_C[idx] / (d_A[idx] + d_B[idx] + k_param);
}
```

Figura 6-5 – Kernel `cuda_mt_cell`.

Os procedimentos necessários para invocar este filtro são equivalentes aos dos filtros anteriores (`robot_sum_filter` e `cuda_complex_cell`), inclusive nas definições de *grids* e *blocks*.

6.2.2. `cuda_bigfilter`

Claramente semelhantes, os três filtros apresentados na seção anterior poderiam ser unidos num único filtro. Todos possuem vetores de entrada e saída de mesma dimensão e realizam operações aritméticas simples. Por essas razões, implementamos o `cuda_bigfilter`, que mescla os filtros `cuda_sum_filter`, `cuda_complex_cell` e `cuda_mt_cell`.

Uma vez que o `cuda_bigfilter` desempenhará as funções dos filtros anteriores somadas, ele terá como valores de entrada as células simples monoculares – S_L e S_L^Q, S_R e S_R^Q – entradas do `cuda_sum_filter`, e como valores de saída as células MT (saída do `cuda_mt_cell`). O código CUDA da Figura 6-6 implementa o `cuda_bigfilter` por meio do *kernel* `cuda_bigfilter`.

```
__global__ void
cuda_bigfilter (float *mt, float *s_r, float *s_r_q, float *s_l, float *s_l_q,
                    int num_neurons, float k_param)
{
       float s_lr, s_lr_q, c_l, c_r, c_lr;

       int idx = blockIdx.x * blockDim.x + threadIdx.x;
       if ( idx < num_neurons )
       {
              s_lr   = s_l[idx] + s_r[idx];
              s_lr_q = s_l_q[idx] + s_r_q[idx];

              c_l    = (s_l[idx] * s_l[idx]) + (s_l_q[idx] * s_l_q[idx]);
              c_r    = (s_r[idx] * s_r[idx]) + (s_r_q[idx] * s_r_q[idx]);
              c_lr   = (s_lr * s_lr) + (s_lr_q * s_lr_q);

              mt[idx] = c_lr / (c_l + c_r + k_param);
       }
}
```

Figura 6-6 – Kernel `cuda_bigfilter`.

O *kernel* `cuda_bigfilter` é invocado por meio da linha de código abaixo:

```
       cuda_bigfilter <<< dimGrid , dimBlock >>> (d_mt, d_s_r, d_s_r_q, d_s_l,
d_s_l_q, num_neurons, k_param);
```

onde `d_mt` representa o vetor de saída, que receberá o resultado das operações realizadas sobre `d_s_r`, `d_s_r_q`, `d_s_l` e `d_s_l_q`, vetores de entrada. O `k_param` já foi mencionado anteriormente (seletividade da célula MT), assim como a dimensão dos vetores (`num_neurons`).

A elaboração deste `cuda_bigfilter` atendeu a dois objetivos: primeiro, simplificar a estrutura do código; segundo, reduzir a quantidade de cópias de dados entre a memória principal e a memória da GPU. Um dos principais gargalos de desempenho quando se desenvolve soluções usando C+CUDA é justamente a constante transferência entre os barramentos. Com este novo filtro, algumas dessas cópias foram eliminadas, o que é um avanço no objetivo do ganho de desempenho.

6.3. FILTROS DE ENTRADA E SAÍDA DAS ÁREAS CORTICAIS MODELADAS

Os dois filtros restantes do modelo são mais complexos que os anteriores e, por essa razão, foram implementados após os anteriores e são apresentados aqui também nesta ordem (após os anteriores). Eles implementam as células de entrada de V1, ou seja, as células simples monoculares S_L, S_L^Q, S_R e S_R^Q (Figura 6-1, pág. 98), e o *spatial pooling* das células de MT (ver Seção 3.3, página 62), e foram denominados `cuda_biological_gabor_filter` e `cuda_gaussian_filter`.

6.3.1. `cuda_biological_gabor_filter`

Conforme descrito nas seções 3.1 e 3.2, a informação visual é inserida no sistema no formato de uma imagem de dimensões 320 por 240 *pixels*, em escala de cinza. No mapeamento retinotópico retina-V1, a imagem sofre uma transformação log-polar tal como a ocorrida no caminho neural da retina ao córtex visual, mais especificamente V1. Nesta transformação, parte da informação visual se perde, uma vez que a resolução do que é captado diminui na medida em que se afasta do foco de atenção. Essa transformação se refletirá inclusive na estrutura de dados, que passará de um vetor de dimensão 76800 (320*240) para um vetor de dimensão 8192 (128*64), adotadas como padrão nas camadas neurais da aplicação.

O `cuda_biological_gabor_filter` implementa a as células S_L, S_L^Q, S_R e S_R^Q e seu código em CUDA é apresentado na Figura 6-7 que, basicamente, invoca a função `cuda_bidimentional_convolution` para realizar o processamento propriamente dito.

```
__global__ void
cuda_biological_gabor (float *s_m, int *xi, int *yi, int wi, int hi,
                           int num_neurons, int rf_num_points, int *image_vector,
                           float global_factor, float c_delta_area, float teste)
{
       for (int i = blockIdx.x; i < num_neurons; i += gridDim.x)
       {
              s_m[i] = c_delta_area * cuda_bidimentional_convolution (rf_num_points,
                                                  image_vector, xi[i], yi[i], wi, hi,
                                                  global_factor, teste) / 6.0;
       }
}
```

Figura 6-7 – Kernel `cuda_biological_gabor_filter`.

.

O *kernel* `cuda_biological_gabor_filter` é invocado por meio da linha de código abaixo:

```
cuda_biological_gabor <<<dimGrid, dimBlock>>> (d_s_m, d_xi, d_yi, wi,
                   hi, num_neurons, rf_num_points, d_image_vector,
                   global_factor, c_delta_area, teste);
```

onde `d_s_m` será o vetor de saída do filtro, os vetores `d_xi` e `d_yi` possuem as coordenadas centrais na imagem de entrada dos campos receptivos das células simples monoculares (S_L, S_L^Q, S_R e S_R^Q); `wi` e `hi` as dimensões da imagem de entrada, `num_neurons` a dimensão da camada neural de saída; `rf_num_points` o número de pontos dos campos receptivos; `d_image_vector` o vetor com os pixels da imagem de entrada; e `global_factor`, `c_delta_area` e `teste` constantes necessárias ao cômputo da saída das células monoculares. As definições de *grids* e *blocks* também são semelhantes às já citadas anteriormente.

Além da transformação log-polar, também convoluções espaciais são aplicadas sobre os pontos da imagem de maneira a fazer com que as representações dos neurônios receptores, e seus respectivos campos receptivos, possam assim colaborar para a construção das células simples monoculares a partir as informações visuais advindas das imagens estéreo.

O *kernel* aciona a função *device*, onde a convolução é executada.

```
__device__ float
cuda_bidimentional_convolution (int rf_num_points, int *image_vector, int x_center,
                                int y_center, int w, int h, float p_global_factor,
                                float p_teste)
{
      int x_current, y_current, i, pixel;
      float intensity, red, green, blue;
      __shared__ float accumulator[BLOCKSIZE];

      accumulator[threadIdx.x] = 0.0;
      for (i = threadIdx.x; i < rf_num_points; i += blockDim.x)
      {
            x_current = x_center + d_rf_points_x[i];
            y_current = y_center + d_rf_points_y[i];

            if ((x_current < 0) || (x_current >= w) ||
                  (y_current < 0) || (y_current >= h))
                  continue;

            pixel = image_vector[y_current * w + x_current];
            red   = (float) RED   (pixel);
            green = (float) GREEN (pixel);
            blue  = (float) BLUE  (pixel);

            intensity = p_teste + (p_global_factor * (red + green + blue)) / 3.0;
            accumulator[threadIdx.x] +=
                  d_rf_points_w[i] * d_rf_points_g[i] * intensity;
      }
      sum_tree_like_reduction (accumulator, BLOCKSIZE);
      __syncthreads();

      return (accumulator[0]);
}
```

Figura 6-8 – Função *device* do *kernel* `cuda_biological_gabor_filter`.

As células simples monoculares são construídas, e os valores de seus neurônios são armazenados no vetor correspondente, sendo alimentada pela função de convolução espacial acionada pelo filtro. Ao final da computação sobre o bloco de dados associado à coordenada de foco na imagem, os valores são reduzidos a um só através da função `sum_tree_like_reduction`, que tem como propósito somar todos os elementos de um vetor [NIK08].

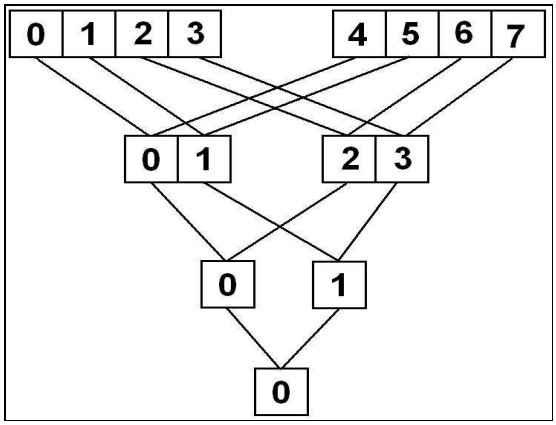

Figura 6-9 – Exemplo da soma em árvore.

Suponha que o vetor cujos elementos serão somados tenha dimensão 8. Para somar seus elementos, os mesmos são divididos em dois grupos (ver Figura 6-9). São somados os elementos 0 e 4 e o resultado é colocado em 0; 1 e 5 e o resultado é colocado em 1; 2 e 6 e o resultado é colocado em 2; e por fim, são somados os elementos 3 e 7 e o resultado é colocado em 3. Em seguida, o vetor é reduzido para tamanho 4 e novamente é dividido em dois. Assim, são somados os elementos 0 e 2 e o resultado é colocado em 0; e 1 e 3 e o resultado é colocado em 1. Finalmente, os elementos onde foram colocados os resultados das últimas somas são divididos novamente em dois grupos e são somados os elementos 0 e 1, e o resultado é colocado em 0. O resultado da soma de todos os elementos do vetor fica no elemento 0 ao final da execução do algoritmo. A Figura 6-10 mostra o código paralelo em C+CUDA do algoritmo.

```
__device__ void
sum_tree_like_reduction (float *vector, int size)
{
	int j, k;
	for (k = size / 2; k > 0; k >>=1)
	{
		__syncthreads();
		for (j = threadIdx.x; j < k; j += blockDim.x)
			vector[j] += v[k + j] ;
	}
}
```

Figura 6-10 – Código `sum_tree_like_reduction`.

Os parâmetros de entrada do algoritmo são: o vetor que desejamos somar, `vector`; e o tamanho deste vetor, `vector_size`. É importante lembrar que o tamanho do vetor deve ser potência de dois. No final do cômputo da soma, a *thread* 0 será a única que terá o valor da soma. As iterações do *loop* (`for`) mais externo fazem a divisão do vetor que desejamos somar, e as iterações do *loop* (`for`) mais interno são responsáveis por fazer a soma em paralelo.

O `cuda_biological_gabor_filter` representa o maior peso de processamento de toda a aplicação, uma vez que sua natureza já é menos simples, e ainda é executado quatro vezes a cada interação, visto que são produzidas quatro células simples monoculares a partir das duas imagens. Sua paralelização representa um grande ganho de desempenho na aplicação. Semelhante ao `cuda_biological_gabor_filter` é o `cuda_gaussian_filter`.

6.3.2. `cuda_gaussian_filter`

```
__global__ void
cuda_gaussian (float *mt_gaussian, float *mt, int num_points, int wo, int ho,
                    int wi, int hi)
{
       for (int i = blockIdx.x; i < wo; i += gridDim.x)
       {
              for (int j = blockIdx.y; j < ho; j += gridDim.y)
              {
                     mt_gaussian[j * wo + i] = cuda_apply_gaussian_kernel (mt,
                                                   num_points, i, j, wo, ho);
              }
       }
}
```

Figura 6-11 – Kernel `cuda_gaussian_filter`.

Também de uma convolução espacial, mas com o objetivo de aproveitar a relação de cooperação dos resultados dos neurônios vizinhos numa mesma camada para melhorar a precisão da construção do *mapa de disparidades*, estrutura com a qual é montada a representação tridimensional interna do objeto do mundo real captado pelo sistema visual.

```
__device__ float
cuda_apply_gaussian_kernel (float *mt, int num_points, int x0, int y0, int wi,
                                    int hi)
{
       int i, j, k, xr, yr;
       float fltWeight;
       __shared__ float fltWeightSum[BLOCKSIZE];
       __shared__ float fltResult[BLOCKSIZE];
       fltWeightSum[threadIdx.x] = 0.0f;
       fltResult[threadIdx.x] = 0.0f;

       for (i = threadIdx.x; i < num_points; i += blockDim.x)
       {
              xr = x0 + d_rf_points_x[i];
              yr = y0 + d_rf_points_y[i];

              if ((xr < 0) || (xr >= wi) || (yr < 0) || (yr >= hi)) continue;

              fltWeightSum[threadIdx.x] += fltWeight = d_rf_points_g[i];
              fltResult[threadIdx.x] += fltWeight * mt[yr * wi + xr];
       }
       for (k = BLOCKSIZE / 2; k > 0; k >>=1)
       {
              __syncthreads();
              for (j = threadIdx.x; j < k; j += blockDim.x)
              {
                     fltWeightSum[j] += fltWeightSum[k + j];
                     fltResult[j] += fltResult[k + j];
              }
       }
       __syncthreads();
       return ((fltWeightSum[0] != .0f) ? fltResult[0] / fltWeightSum[0] : .0f);
}
```

Figura 6-12 – Função *device* do *kernel* `cuda_gaussian`.

Neste ponto do trabalho, todos os principais trechos de código dos filtros estão paralelizados e executando na GPU por meio do CUDA. Agrupando essas chamadas em uma mesma função principal representa um novo filtro, completo, que transforma toda a informação visual desde sua captação pelas câmeras até a montagem da representação interna tridimensional com informações referentes à profundidade.

6.4. REFINAMENTOS

Os *kernels* de todos os filtros foram paralelizados, as chamadas de entrada/saída da aplicação removidas, restava apenas a definição de um novo filtro que comportasse as chamadas CUDA. Além de simplificar a legibilidade do código, o novo filtro serviria também para uma última providência no sentido de contornar os gargalos comuns em implementações que mesclam trechos seqüenciais e paralelizados: as cópias de dados.

Na impementação do filtro que foi chamado `v1_mt`, uma vez que representa as funções desempenhadas por estas mesmas áreas corticais do sistema visual, providenciou-se que as chamadas a funções CUDA fossem agrupadas num único trecho de código, e através de uma estrutura de dados elaborada para tal, as cópias de dados entre os barramentos da CPU e da GPU se resumiram à imagem da CPU para a GPU no início, e a camada neural `mt_gaussian` da GPU para a CPU no final. A partir desta última camada neural, é possível a reconstrução tridimensional do objeto. Também foi possível diminuir a quantidade de vezes que o filtro `cuda_biological_gabor_filter` deveria ser executado por ciclo, levando em conta que seria necessário apenas uma vez sobre a imagem direita, pois na vergência apenas as coordenadas da imagem esquerda variam.

6.4.1. `cuda_map_v1_to_image`

Neste momento também foi observado que, mesmo com os principais *kernels* dos filtros já em paralelo, havia trechos do código executados seqüencialmente que representavam um percentual grande no tempo total da aplicação. O principal deles se mostrou o mapeamento retinotópico retina-V1, efetuado pela função `map_v1_to_image`. Detectado este gargalo, sua paralelização em C+CUDA acrescentou mais *speedup* à aplicação comparada à original.

```
__global__ void
map_v1 (int *xi, int *yi, int wi, int hi, int w, int h, int x_center, int y_center,
             float correction, float log_factor, int shift)
{
       float CUDA_LOG_POLAR_SCALE_FACTOR = 1.0;
       float CUDA_LOG_POLAR_THETA_CORRECTION = 0.0;
       float d, theta, exp_val, x;
       int i, adjust, u = 0, v = 0;

       for (v = blockIdx.x; v < h; v+=gridDim.x)
       {
              for (u = threadIdx.x; u < w; u+=blockDim.x)
              {
                     i = v * w + u;
                     if (u < w/2)
                     {
                            adjust = (w-1)/2 - u;
                            x = ((float) adjust / (float) (w/2)) * log_factor;
                            exp_val = (float) (wi/2) * (exp (log (log_factor) *
                                          (x - log_factor) / log_factor) –
                                          (1.0/log_factor)) *
                                          (log_factor / (log_factor - 1.0));
                            d = CUDA_LOG_POLAR_SCALE_FACTOR * exp_val;
                            theta = pi * ((h * (3.0 / 2.0) - (v * correction)) / h) +
                                          CUDA_LOG_POLAR_THETA_CORRECTION;
                     }
                     else
                     {
                            adjust = u - w/2;
                            x = ((float) adjust / (float) (w/2)) * log_factor;
                            exp_val = (float) (wi/2) * (exp (log (log_factor) *
                                          (x - log_factor) / log_factor) –
                                          (1.0/log_factor)) *
                                          (log_factor / (log_factor - 1.0));
                            d = CUDA_LOG_POLAR_SCALE_FACTOR * exp_val;
                            theta = pi * ((h * (3.0 / 2.0) + (v * correction)) / h)
                                          CUDA_LOG_POLAR_THETA_CORRECTION;
                     }
                     xi[i] = (int) (d * cos(theta) + 0.5) + x_center + shift;
                     yi[i] = (int) (d * sin(theta) + 0.5) + y_center;
              }
       }
}
```

Figura 6-13 – Kernel `cuda_map_v1_to_image`.

Ao fim da etapa de codificação, resumiu-se a seqüência de 5 filtros seqüenciais para apenas 1 paralelo, e diminuiu-se a quantidade de cópias de dados para apenas 2 – imagem da CPU/GPU e camada neural final da GPU/CPU. Os *displays* na tela sobraram apenas cinco – imagem esquerda, imagem direita, reconstrução tridimensional, reconstrução em vista superior e console de comandos. O tempo é medido por meio do *prompt*, numericamente convertido em segundos, e afere a execução a partir do comando de vergência (tecla V) até sua conclusão. Tantos os comandos para selecionar o foco, executar a vergência e reconstruir o modelo do objeto permaneceram manuais.

7. METODOLOGIA

Neste capítulo são descritos a plataforma computacional utilizada no desenvolvimento deste trabalho, em quesitos de *hardware* e *software*, além das métricas empregadas nas análises de desempenho.

7.1. PLATAFORMA COMPUTACIONAL

A plataforma computacional sobre a qual o trabalho foi realizado, e onde os experimentos de tempo foram aferidas conta com um processador AMD Athlon(tm) 64 X2 Dual Core Processor 5400+ 2.8GHz, com 512KB de *cache* L2 por *core* e 4GB de DRAM DDR2 de 800 MHz. O sistema operacional empregado foi o Linux Fedora 10 32 bits e o compilador de C o *gcc* 4.3.2.

A placa de vídeo utilizada foi uma NVIDIA GeForce GTX 285 com 1GB de DRAM GDDR3. A versão do compilador CUDA foi o *nvcc* 2.1. A GPU da GTX 285 possui vários *Stream Processors* (SPs) para executar operações inteiras e de ponto flutuante. Os SPs são agrupados em *Stream Multiprocessors* (SMs) que, por sua vez, são agrupados em *Thread Processing Clusters* (TPCs) [NVI08b].

A GTX 285 possui 10 TPCs de 3 SMs, cada um com 8 SPs, totalizando 240 SPs, e opera com um *clock* de 1,48 Ghz. A memória principal de 1GB possui *clock* de 2.484 MHz, barramento de 512 bits e a taxa de transferência de 159 GB/s. Também permite criar 512 *threads* por bloco e é capaz de manter o estado de 32K *threads* simultaneamente (uma *grid* de 32K *threads*). Ela pode processar dados em precisão simples e dupla, mas o desempenho máximo em precisão dupla é de 80 GFlop/s, enquanto que em precisão simples é igual à 933 GFlop/s [NVI08b].

7.2. MÉTRICA EMPREGADA

Uma vez que se busca ganho de desempenho na paralelização da aplicação, a métrica de avaliação adotada foi o *speedup*, que é o resultado da divisão do tempo da média de tempo da execução do código sobre o qual se está trabalhando pela média do tempo de execução do código original. Esta métrica é quantificada em

vezes, ou seja, quantas vezes o código mais recente executa mais rápido que o código original.

Para se alcançar as médias de tempo de execução, foram aferidos os tempos de 10 execuções de cada versão do código, inclusive o original. A cada alteração significativa, estipula-se uma nova versão e efetua-se a medição. A soma dos tempos dividida pela quantidade de amostras possibilita a média.

7.3. DESCRIÇÃO DOS RESULTADOS EXPERIMENTAIS

Os resultados obtidos experimentalmente serão apresentados na forma de tabelas de dados, com valores de dez amostras de tempos de execução. A precisão dos resultados obedecerá ao critério de desvio padrão, podendo variar em seu número de casas decimais em decorrência do mesmo, uma vez que algumas chamadas de *kernels* CUDA são executadas em décimos de segundo (10^{-1} segundos), sendo que outras em centésimos de segundo (10^{-2} segundos), e suas variações podem ser na ordem de algumas centenas de microsegundos ($100x10^{-6}$ segundos).

O método empregado para a obtenção dos tempos de execução com a precisão mais adequada foi a função *gettimeofday()* (*time.h*). Uma vez que as chamadas aos *kernels* CUDA alcançaram frações de segundo na casa dos microsegundos, outras funções como *clock()* (*time.h*) e *getticks()* (*sys/time.h*) não se mostraram adequadas, pois sua precisão não foi suficiente (décimos e milésimos de segundo, respectivamente) para aferir os resultados.

8. EXPERIMENTOS

Como descrito no Capítulo 6, uma série de etapas foi percorrida ao longo deste trabalho, visando alcançar o melhor desempenho possível da aplicação de reconstrução tridimensional a partir de imagens estéreo. Estas etapas não foram estipuladas seguindo algum roteiro pré-estabelecido, somente seguiu um caminho natural descoberto durante o processo de investigação das possibilidades de paralelização do código. A estas etapas deu-se o nome de Passos, e cada um deles será explicado no decorrer deste capítulo, assim como o reflexo de suas modificações no desempenho, seja favoravelmente ou desfavoravelmente, como aconteceu em algumas ocasiões. Para aferir os tempos de execução, foram tomadas 10 amostras, de onde foram extraídas médias, para assim ser calculado o *speedup* em relação à execução do código original. A unidade de tempo é o segundo, e o *speedup* em quantas vezes mais rápido o código se tornou.

8.1. PASSOS 1 E 2

As primeiras investidas na reestruturação do código foram ainda no escopo da linguagem C. Inicialmente, foram removidas as chamadas de sistema que efetuavam algum tipo de entrada/saída, tais como escritas em arquivos, impressões em tela etc., tarefas que sempre contribuem para o aumento do tempo de execução das aplicações. No **Passo 1**, os *displays* de visualização menos necessários foram removidos, assim como os relatórios exibidos no *prompt*, e as escritas em arquivos de *log* (relatórios eletrônicos). Alcançou-se assim uma média de 13,7 segundos, que comparado à média de 16,9 segundos do **Original** representou um *speedup* de 1,23 vezes. Os valores referentes às amostras estão dispostos na Tabela 8-1

	Original	Passo 1	Passo 2
	16,91150	13,74675	13,47288
	16,68240	13,60346	13,47886
	17,07930	13,58221	13,45368
	16,74035	13,83114	13,42912
	17,02239	14,01479	13,45926
	16,88516	13,59995	13,45330
	16,86578	13,59937	13,44625
	16,87360	13,77217	13,44475
	16,79567	13,60485	13,42742
	16,95008	13,57504	13,42404
Média	**16,9**	**13,7**	**13,44**
Desvio Padrão	**0,1**	**0,1**	**0,01**
Speedup		**1,23**	**1,25**

Tabela 8-1 – Tempos de execução Original, Passo 1 e Passo 2.

Também na Tabela 8-1 encontram-se os valores referentes ao **Passo 2**, que consistiu na primeira alteração no código dos filtros da aplicação. Devido à sua semelhança estrutural, o `sum_filter`, `complex_cell` e `mt_cell` foram unificados em um único filtro, o `bigfilter`. Esta alteração, ainda no escopo do C seqüencial, serviu para demonstrar que já é possível ganhar algum desempenho reduzindo as cópias de dados, mesmo que ainda no barramento da CPU. A média de execução deste segundo passo foi de 13,44 segundos, um *speedup* de 1,25 vezes em relação ao **Original**.

8.2. PASSOS 3 E 4

Partindo em direção ao código paralelo em C+CUDA, os mesmos três filtros menos complexos foram os primeiros a serem recodificados. Individualmente executados na GPU no **Passo 3** (agora como `cuda_sum_filter`, `cuda_complex_cell` e `cuda_mt_cell`) alcançaram uma média de 15,31 segundos, e unidos no agora `cuda_bigfilter` no **Passo 4** alcançaram uma média de 14,9 segundos, promovendo assim um *speedup* de, respectivamente, 1,10 e 1,13 vezes em relação ao **Original**. As amostras de execução estão na Tabela 8-2 a seguir.

	Original	Passo 3	Passo 4
	16,91150	15,33123	14,92397
	16,68240	15,29954	14,87629
	17,07930	15,28252	14,83655
	16,74035	15,32300	15,11515
	17,02239	15,38681	14,86103
	16,88516	15,30655	15,18136
	16,86578	15,29222	14,87375
	16,87360	15,29376	14,95861
	16,79567	15,30359	15,11897
	16,95008	15,30456	14,86683
Média	**16,9**	**15,31**	**14,9**
Desvio Padrão	**0,1**	**0,02**	**0,1**
Speedup		**1,10**	**1,13**

Tabela 8-2 – Tempos de execução Original, Passo 3 e Passo 4.

Percebam que no caso anterior houve um decréscimo no *speedup* em relação ao **Passo 2**, que fazia o mesmo porém sequencialmente. A resposta a esta questão já foi mencionada anteriormente: cópias de dados entre os barramentos da CPU e GPU.

8.3. PASSOS 5, 6 E 7

O próximo passo já representou o maior impacto nos tempos de execução deste trabalho. O `biological_gabor_filter`, como já descrito anteriormente, representa o maior peso de processamento da aplicação, seja por ser mais complexo que os primeiros filtros trabalhados, mas também por ser executado quatro vezes a cada ciclo, duas para cada imagem estéreo. A paralelização do principal *kernel* do filtro em C+CUDA no **Passo 5** provocou uma redução da média de execução da aplicação para 2,14 segundos e um *speedup* de 7,89 vezes em comparação ao **Original**. Nos **Passos 6** e **7**, uma tentativa de interagir os filtros já paralelizados. Assim, os tempos de execução dos filtros paralelizados `cuda_biological_gabor_filter`, `cuda_sum_filter`, `cuda_complex_cell` e `cuda_mt_cell` na GPU tiveram como média 2,44 segundos com *speedup* 6,92 vezes. A redução do desempenho comparado ao **Passo 5** (2,44 segundos contra 2,14 segundos) tem como razão o mesmo motivo já citado anteriormente – cópias de dados. Há uma pequena melhoria no **Passo 7**, onde o `cuda_biological_gabor_filter` e o `cuda_bigfilter` são executados na

GPU, porque o `bigfilter` economiza algumas cópias de dados entre os barramentos, mas ainda sim inferior ao **Passo 5**. Os valores de média e *speedup* são 2,18 segundos e 7,75 vezes, mostrados na Tabela 8-3.

	Original	Passo 5	Passo 6	Passo 7
	16,91150	2,14453	2,43225	2,18363
	16,68240	2,12545	2,42646	2,16374
	17,07930	2,12407	2,42789	2,16491
	16,74035	2,12563	2,43358	2,16005
	17,02239	2,14248	2,44063	2,23037
	16,88516	2,12674	2,42990	2,16355
	16,86578	2,12461	2,48083	2,17097
	16,87360	2,16212	2,45803	2,19221
	16,79567	2,15489	2,42411	2,20028
	16,95008	2,17658	2,45460	2,16344
Média	**16,9**	**2,14**	**2,44**	**2,18**
Desvio Padrão	**0,1**	**0,02**	**0,02**	**0,02**
Speedup		**7,89**	**6,92**	**7,75**

Tabela 8-3 – Tempos de execução Original, Passo 5, Passo 6 e Passo 7.

8.4. PASSOS 8 E 9

O último dos filtros a ser paralelizado em C+CUDA foi o `gaussian_filter`, que se tornou `cuda_gaussian_filter`. O ganho de desempenho assinalado por este filtro também foi significativo, uma vez que se trata também de um filtro menos simples, mesmo que sendo executado somente uma vez por ciclo. Desta maneira, no **Passo 8**, a média de tempo de execução da aplicação caiu para 1,89 segundos, e o *speedup* subiu para 8,96 vezes comparado ao **Original**.

Uma vez que agora todos os filtros que originalmente executavam sequencialmente agora possuem seus *kernels* recodificados em C+CUDA, porém ainda comprometendo o desempenho em função de sucessivas cópias de dados entre os barramentos, a próxima etapa do trabalho seguiu no sentido de produzir um único filtro agrupando todos os anteriores. Assim, no **Passo 9** foi elaborado o `v1_mt`, que recebeu este nome referente às áreas corticais modeladas por ele. Neste filtro, todas as chamadas aos *kernels* ocorrem de maneira encadeada, e através de uma estrutura de dados elaborada para tanto, as cópias de dados foram contornadas, permitindo que toda a informação fosse carregada de uma única vez na GPU e sofresse seu processamento apenas neste escopo de barramento, o que fez o desempenho dobrar em relação ao passo anterior. A média foi de 0,939

segundo, com *speedup* de 17,98 vezes em relação ao **Original**. Na Tabela 8-4 a descrição das amostras.

	Original	Passo 8	Passo 9
	16,91150	1,88779	0,94000
	16,68240	1,86547	0,93621
	17,07930	1,86644	0,93416
	16,74035	1,90582	0,93479
	17,02239	1,91254	0,95362
	16,88516	1,86699	0,95094
	16,86578	1,90377	0,93490
	16,87360	1,86886	0,93179
	16,79567	1,88341	0,93234
	16,95008	1,87917	0,93836
Média	**16,9**	**1,89**	**0,939**
Desvio Padrão	**0,1**	**0,02**	**0,008**
Speedup		**8,96**	**17,98**

Tabela 8-4 – Tempos de execução Original, Passo 8 e Passo 9.

8.5. PASSOS 10 E 11

A proposta inicial, de trabalhar sobre a paralelização dos filtros, até que se alcançasse uma estrutura que permitisse evitar ao máximo as cópias de dados, foi *a priori* efetuada. Dando continuidade à investigação, outros gargalos de processamento seqüencial foram procurados a partir deste ponto. Além dos filtros, o trecho de código que é executado pela aplicação é aquele que comanda o mecanismo de vergência ocular, passo fundamental para a obtenção das disparidades binoculares, e por conseqüência, o *mapa de disparidades* que é usado para a reconstrução do modelo tridimensional. Uma análise do código permitiu verificar que vários testes com os dados das camadas neurais, e mesmo algumas camadas neurais extras, implementadas na aplicação original, levando em conta o modelo biológico em sua completude, mostraram-se pouco necessários no contexto geral, além de estarem comprometendo o desempenho. Desativar estes trechos no **Passo 10** provocou um ganho na média de execução, para 0,652 segundo e *speedup* de 25,91 vezes.

O processo investigativo continuou a vasculhar o código, e naturalmente outras passadas sobre o que já foi implementado foram feitas. Desta vez, mais aprimoramentos no código foram descobertos. No `v1_mt`, onde os *kernels* dos filtros originais são chamados para execução na GPU, foi possível verificar que o mais

pesado dos *kernels*, o `cuda_biological_gabor_filter`, estava sendo executado uma quantidade excessiva de vezes. Uma vez que se definiu na aplicação que a vergência é efetuada pelo olho esquerdo, enquanto o direito permanece estabilizado no mesmo ponto focal, este *kernel* só precisa ser executado uma única vez para a imagem direita, enquanto que para a imagem esquerda é necessário em todos os ciclos. Assim, no **Passo 11**, a eliminação de execuções desnecessárias de *kernels* promoveu um decréscimo na média do tempo de execução para 0,4305 segundo, com *speedup* de 39,20 vezes comparado ao **Original**. A Tabela 8-5 mostra os valores dos dois últimos passos descritos.

	Original	**Passo 10**	**Passo 11**
	16,91150	0,65294	0,43031
	16,68240	0,65330	0,43071
	17,07930	0,65347	0,43084
	16,74035	0,65399	0,43100
	17,02239	0,64932	0,43112
	16,88516	0,65125	0,43056
	16,86578	0,65034	0,43019
	16,87360	0,64935	0,43027
	16,79567	0,65151	0,42978
	16,95008	0,64911	0,43055
Média	**16,9**	**0,652**	**0,4305**
Desvio Padrão	**0,1**	**0,002**	**0,0004**
Speedup		**25,91**	**39,20**

Tabela 8-5 – Tempos de execução Original, Passo 10 e Passo 11.

8.6. PASSO 12

A cada momento em que prosseguiu a investigação, foram encontrados aprimoramentos no código que já foi paralelizado, e o trabalho continuou neste escopo – aprimoramentos. À procura de gargalos seqüenciais restantes, verificou-se que o número mínimo de testes, alocações de dados e estruturas já havia sido alcançado, porém um último *kernel* ainda se fazia presente, e havia sido negligenciado. Esta função, `map_v1_to_image`, responsável por mapear as coordenadas da imagem para a área cortical V1, estava sendo responsável por uma quantidade de processamento que poderia representar um grande acréscimo de desempenho. Sua recodificação para CUDA, agora com nome `cuda_map_v1_to_image`, ação efetuada no **Passo 12**, representou uma redução

na média de tempo de execução para 0,2942 segundo, com *speedup* de 57,38 vezes comparado ao **Original**. A Tabela 8-6 mostra os valores.

	Original	**Passo 12**
	16,91150	0,29414
	16,68240	0,29443
	17,07930	0,29426
	16,74035	0,29417
	17,02239	0,29419
	16,88516	0,29398
	16,86578	0,29427
	16,87360	0,29436
	16,79567	0,29412
	16,95008	0,29409
Média	**16,9**	**0,2942**
Desvio Padrão	**0,1**	**0,0001**
Speedup		**57,38**

Tabela 8-6 – Tempos de execução Original e Passo 12.

8.7. TEMPOS DE EXECUÇÃO E *SPEEDUP*

Ao fim do **Passo 12**, todo o código aparentemente paralelizável havia sido paralelizado. Agora todas as chamadas a possíveis *kernels* estavam codificadas em C+CUDA, e sendo efetuadas na GPU. Esta estrutura também permitiu que as cópias de dados fossem em sua maioria excluídas, evitando gargalos indesejáveis. O mínimo de código seqüencial estava sendo executado, e as chamadas para recursos de entrada/saída também foram reduzidas ao máximo. Aprimoramentos no código tanto seqüencial como paralelo permitiram alcançar, numa escala quase sempre ascendente, *speedups* significativos em relação aos tempos de execução seqüenciais originais, reforçando os promissores potenciais apresentados pela tecnologia CUDA para GPUs.

A seguir, na Tabela 8-7 são apresentados os valores referentes aos *speedups* aferidos no decorrer deste trabalho. A Figura 8-1 mostra os tempos de execução, e a Figura 8-2 o *speedup* obtido em cada passo na forma gráfica.

Passo	SPEEDUP
1	1,23
2	1,25
3	1,10
4	1,13
5	7,89
6	6,92
7	7,75
8	8,96
9	17,98
10	25,91
11	39,20
12	57,38

Tabela 8-7 – *Speedups* de cada Passo relativo à execução Original.

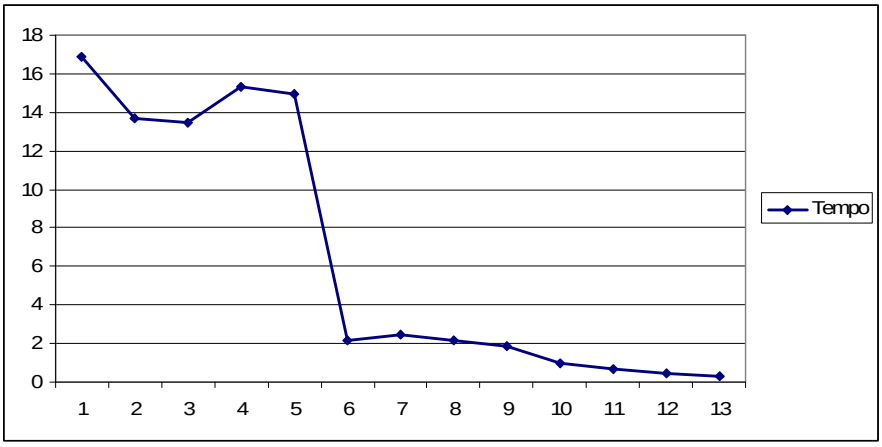

Figura 8-1 – Tempos de execução desde o Original.

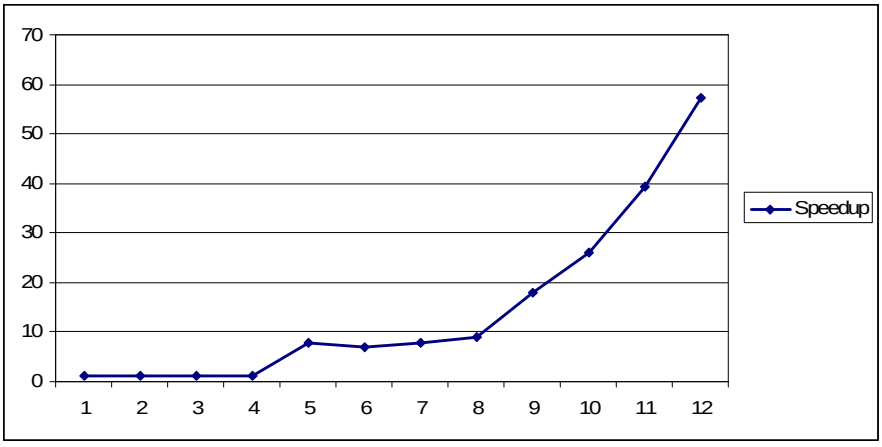

Figura 8-2 – *Speedup* total alcançado em todos os Passos.

8.8. GANHO DE DESEMPENHO SEQUENCIAL

Levando-se em conta que tanto alterações de código para C+CUDA quanto no escopo do C podem produzir um acréscimo no desempenho da aplicação, as mesmas modificações que foram feitas para a migração dos *kernels* para CUDA foram aplicadas para o código seqüencial. As remoções de testes, chamadas para entrada/saída e alocações de estruturas foram preservadas, e os trechos de código em CUDA foram adaptados para C padrão. Ao se aferir os tempos de execução desta nova aplicação seqüencial, uma boa constatação: a média foi bastante reduzida, se alcançou um bom *speedup* em relação ao original.

Desta maneira, na Tabela 8-8 mostram-se os tempos do **Original**, do novo **Seqüencial** e do **CUDA**, com suas médias e *speedups* relativos entre si. O novo **Seqüencial** alcançou uma média de 4,5172 segundos e um *speedup* de 3,7370 vezes comparado ao **Original**, enquanto o **CUDA** referente ao **Passo 12** citado anteriormente alcançou um *speedup* de 15,3542 vezes comparado ao novo **Seqüencial**.

	ORIGINAL	SEQUENCIAL	CUDA
	16,91150	4,50757	0,29414
	16,68240	4,50347	0,29443
	17,07930	4,51545	0,29426
	16,74035	4,52773	0,29417
	17,02239	4,51965	0,29419
	16,88516	4,50694	0,29398
	16,86578	4,50698	0,29427
	16,87360	4,52767	0,29436
	16,79567	4,52868	0,29412
	16,95008	4,52795	0,29409
Média	**16,9**	**4,51**	**0,2942**
Desvio Padrão	**0,1**	**0,01**	**0,0001**
Speedup		**3,74**	**15,35**

Tabela 8-8 – Tempos de execução Original, Seqüencial otimizado e CUDA.

8.9. TEMPOS DE EXECUÇÃO DAS CHAMADAS AOS KERNELS CUDA

A paralelização dos *kernels* dos filtros para C+CUDA propiciaram uma melhoria no tempo de execução, alcançando uma média de cerca de 0,29 segundos. Individualmente, cada *kernel* teve o seu tempo de chamada avaliado, para que sua contribuição no contexto geral da execução pudesse ser quantificada. Abaixo, os

valores obtidos em dez amostras, para cada filtro evocado em separado e, por último, agrupados.

	MAP_V1	BIOLOGICAL	BIGFILTER	GAUSSIAN	AGRUPADOS
	0,00165	0,07678	0,00086	0,08745	0,16903
	0,00163	0,07701	0,00086	0,08747	0,16902
	0,00163	0,07694	0,00088	0,08741	0,16900
	0,00168	0,07706	0,00086	0,08749	0,16872
	0,00164	0,07707	0,00091	0,08743	0,16926
	0,00165	0,07712	0,00086	0,08746	0,16882
	0,00170	0,07709	0,00087	0,08742	0,16863
	0,00170	0,07690	0,00086	0,08751	0,16867
	0,00168	0,07679	0,00087	0,08747	0,16857
	0,00165	0,07697	0,00087	0,08745	0,16863
Média	**0,00166**	**0,0770**	**0,00087**	**0,08746**	**0,1689**
Desvio Padrão	**0,00003**	**0,0001**	**0,00002**	**0,00003**	**0,0002**

Tabela 8-9 – Tempo de execução dos *kernels* em C+CUDA.

A soma dos tempos de execução dos *kernels* em CUDA, individualmente aferidos, foi de 0,1670 segundos. Já o tempo de execução de todos os *kernels* aferidos em bloco foi de 0,1689 segundos. Esta diferença é causada por três chamadas a *kernel* realizadas uma única vez no início da execução da aplicação (uma `cuda_map_v1_to_image` e duas `cuda_biological_gabor_filter`, aplicadas à imagem direita), que não são contabilizadas quando somamos os tempos individuais de cada *kernel.*

Considerando que a média de tempo de execução da aplicação foi de 0,2942 segundos, e as chamadas a *kernels* CUDA somaram 0,1689 segundos, tem-se então que o tempo gasto no trecho seqüencial de código ficou em torno de 0,1253 segundos.

9. DISCUSSÃO

Neste capítulo são discutidos trabalhos correlatos na área de visão esteroscópica e uma análise crítica deste trabalho de pesquisa.

9.1. TRABALHOS CORRELATOS

Vários outros modelos de percepção de profundidade existem e alguns deles inclusive já foram implementados em C+CUDA. Gibson e Marques [GIB08] desenvolveram um trabalho de paralelização sobre um modelo proposto por Hirschmüller [HIR05], chamado *Semi-Global Matching* (SGM). A média do tempo de execução alcançada por Gibson e Marques foi de 0,17 segundos [GIB08]. Muito embora o desempenho em termos de tempo alcançado por Gibson e Marques seja superior ao nosso, o modelo de visão estéreo empregado pelos mesmos não é inspirado na biologia (por exemplo, não é log-polar).

Rosenberg *et al*. [ROS06] realizaram trabalho semelhante ao de Gibson e Marques, e também usaram CUDA. A média de execução alcançada por Rosenberg *et al*. foi 0,13 segundos [ROS06]. Outro trabalho, desenvolvido por Choi, *et al.* [CHO08] sobre a proposta de Rusinkiewicz e Levoy [RUS01] também usou C+CUDA. O modelo de percepção de profundidade foi o *Iterative Projection Point* (IPP). A média de tempo de execução alcançado por Choi, *et al.* foi 0,21 segundos [CHO08]. Em todos estes trabalho também não foram empregados modelos biologicamente inspirados.

9.2. DESEMPENHO ALCANÇADO

Conforme descrito na Seção 7.1, o desempenho máximo da placa GTX 285 é 933 GFlop/s. Examinando o código fonte desenvolvido, é possível estimar o número de operações de ponto flutuante que ele demanda. A Tabela 9-1 mostra, para cada trecho de código em C+CUDA das Seções 6.2, 6.3 e 6.4, o número de operações por iteração, o número de iterações, o número de chamadas de *kernel*, o total de

Flop por reconstrução 3D e o total de GFlop/s. Para o cálculo do total de GFlop/s (10^9 Flop), foram usados os tempos por *kernel* da Tabela 8-9.

Figura	Operações	Iterações	Chamadas do *kernel*	Flop	GFlop/s
Figura 6-6	14	8192	64	7340032	0,023
Figura 6-7	2882	8192	130	3175710720	18,705
Figura 6-11	631	8192	64	330825728	1,941
Figura 6-13	22	8192	65	11714560	0,064

Tabela 9-1 – Desempenho dos *kernels* em C+CUDA, em GFlop/s.

Como a Tabela 9-1 mostra, o desempenho máximo alcançado foi 3,53 GFlop. Contudo, o desempenho obtido na aplicação como um todo (uma reconstrução 3D), em termos de GFlop/s, foi 20,8 GFlop/s. Isso mostra que ainda há espaço para ajuste de parâmetros e/ou melhoria na implementação. Mostra também que é difícil alcançar o desempenho máximo da GPU empregada tendo em vista suas limitações, principalmente relacionadas à hierarquia de memória.

9.3. ANÁLISE CRÍTICA DESTE TRABALHO DE PESQUISA

Uma possível falha de nosso trabalho é a não implementação de um mapeamento planar da imagem para o mapa de disparidades – implementamos apenas o mapeamento log-polar. A implementação de um mapeamento planar permitiria a comparação direta dos desempenhos em termos de tempo e de qualidade dos mapas de disparidade com o de outras implementações de visão estereoscópica em CUDA, como as de Gibson e Marques [GIB08], Rosenberg *et al*. [ROS06] e Choi, *et al.* [CHO08].

Outro aspecto que poderia ser melhorado em nosso trabalho seria a conversão da imagem de entrada para níveis de cinza antes do processamento em CUDA, ao invés de fazer esta conversão em CUDA, como mostrado na Figura 6-8, página 107. Acreditamos que isso poderia melhorar ainda mais o desempenho.

Seria também importante examinar detidamente os parâmetros de chamada dos *kernels*, pois blocos maiores ou menores do que os que utilizamos podem resultar em melhor desempenho. Outro fator que pode influenciar no desempenho e que não foi investigado é a utilização de *kernels* 2D, ao invés de 1D.

10. CONCLUSÃO

Neste capítulo, apresentamos um sumário do trabalho de pesquisa desenvolvido e, em seguida, discutimos os resultados e conclusões obtidas com o mesmo. Por fim, sugerimos novas linhas de trabalho que podem ser desenvolvidas a partir deste.

10.1. SUMÁRIO

Neste trabalho, descrevemos a modelagem matemático-computacional das áreas V1 e MT do córtex visual humano proposta por Oliveira [OLI05] que, a partir de duas imagens bidimensionais (imagens estéreo) de uma cena do mundo real, produz uma representação tridimensional interna ao computador com informações sobre as profundidades (distâncias ao observador) e intensidades luminosas observadas na cena original. Apresentamos em seguida uma versão paralela do modelo de visão esterescópica desenvolvido por Oliveira [OLI05]. A nova implementação foi proposta com o objetivo de reduzir o tempo de execução da versão original de modo a viabilizar aplicações de tempo real. Para tanto, técnicas de programação paralela em C+CUDA (*Compute Unified Device Architecture*) foram empregadas.

A estrutura da aplicação desenvolvida por Oliveira baseou-se na utilização de filtros, cada um atuando como uma etapa das transformações sofridas pela informação visual desde que esta é captada pelas câmeras (equivalente aos olhos nos sistema biológico) até o momento em que se apresenta na forma de um *mapa de disparidades*. O mapa de disparidades é uma estrutura de dados que permite o cálculo das profundidades dos pontos no mundo real com relação às câmeras.

10.2. RESULTADOS E CONCLUSÕES

Foram avaliados os tempos de execução das reconstruções tridimensionais, que antes eram efetuadas sequencialmente, agora codificadas em C+CUDA. Os resultados foram bastante satisfatórios, viabilizando o emprego da tecnologia desenvolvida em sistemas de tempo real. No entanto, a qualidade da representação

interna não pode ser diretamente comparada com a do modelo biológico, característica herdada da aplicação original, uma vez que não há aparato para se aferir isto no cérebro humano.

A métrica de medição dos ganhos de desempenho em termos de tempo alcançados com nosso trabalho foi o de *speedup*, ou seja, quantas vezes mais rápido é o código em CUDA. Análises individuais das respostas das células envolvidas no processo também foram deixadas de lado momentaneamente, uma vez que os filtros foram desconstruídos e modelados em outro formato de código. Porém, os resultados visuais obtidos são equivalentes aos da aplicação original, uma vez que os cálculos dos *kernels* foram mantidos exatamente como suas versões originais, localizadas nos filtros seqüenciais, tendo sido removidos apenas (i) os testes preliminares de dados, na maioria das vezes redundantes devido à estrutura encadeada dos filtros, e (ii) as próprias cópias de dados entre as estruturas de armazenamento.

O potencial do *hardware*, mesmo realizando uma quantidade massiva de processamento de dados – visto o ganho de desempenho da aplicação –, foi uma pequena fração do todo disponível. Conforme foi descrito na Seção 7.1, o desempenho bruto da placa GTX 285 é 933 GFlop/s, enquanto que o alcançado neste trabalho foi de, aproximadamente, 20,8 GFlop/s (Seção 9.2).

A corretude da reconstrução tridimensional proporcionada pelos sistemas gráficos empregados neste trabalho é idêntica à corretude da aplicação seqüencial original, que possui taxa de erro na ordem de 0,2% em aplicações práticas, conforme comprovado por meio de experimentos realizados por De Souza e Machado em [DES07].

10.3. TRABALHOS FUTUROS

Os resultados satisfatórios obtidos até o momento motivam continuar as pesquisas para alcançar o objetivo final de dotar um sistema robótico da capacidade de criar representações internas do mundo externo que permitam que esse sistema possa navegar e interagir com o meio externo através desta representação. O obstáculo do tempo de processamento de cada reconstrução já chegou a um patamar que permite uma execução do processo em tempo real.

Mais experimentos, que variem as configurações de tamanhos de blocos, podem mostrar ser possível alcançar ainda maior desempenho. Tamanhos adequados para cada *kernel*, de acordo com os tamanhos das estruturas de dados que serão manipuladas pela GPU, podem otimizar a execução. Assim, seria interessante um maior estudo sobre a arquitetura do *hardware*, além das estruturas de programação nativas, como a hierarquia de memória e áreas específicas da GPU, bem como várias funções implementadas em *hardware*, tais como seno e cosseno, que poderiam agilizar o processo de execução de determinados trechos de código.

Placas gráficas com mais de uma GPU, ou mesmo interligadas entre si, não foram testadas. Outros modelos de GPU no mercado também não puderam ser examinados por falta de recursos e tempo. Portar o código para Brook+ (equivalente CUDA da ATI – vide http://graphics.stanford.edu/projects/brookgpu/) ou mesmo OpenCL (projeto para unificar as plataformas em relação ao código – vide http://www.khronos.org/news/press/releases/khronos_launches_heterogeneous_computing_initiative/) são também avenidas de pesquisa futura interessantes.

Por fim, embarcar a tecnologia em uma plataforma robótica funcional, dotada de câmeras estéreo e um *hardware* que pudesse executar o código CUDA poderia confirmar a viabilidade da percepção de profundidade inspirada na biologia como mecanismo para ajudar robôs a se locomoverem e interagirem com o mundo real.

11. REFERÊNCIAS BIBLIOGRÁFICAS

[ADE85] ADELSON, Edward H.; BERGEN, James R. **Spatiotemporal Energy Models for the Perception of Motion.** J. Opt. Soc. Am. A 2: 284–299, 1985.

[ANZ99a] ANZAI, A.; OHZAWA, I.; FREEMAN, Ralph D. **Neural Mechanisms for Processing Binocular Information I. Simple Cells.** The Journal of Neurophysiol. 82(2): 891-908, 1999.

[ANZ99b] ANZAI, A. OHZAWA, I., FREEMAN, Ralph D. **Neural Mechanisms for Processing Binocular Information II. Complex Cells.** The Journal of Neurophysiol. 82(2): 909-924, 1999.

[BAR67] BARLOW, H.B. et al. **The Neural Mechanism of Binocular Depth Discrimination.** J. Physiol. 193: 327-342, 1967.

[CHO08] CHOI, S; PARK, S; KIM, J.; PARK, Y. **Multi-view Range Image Registration using CUDA.** 23rd Int. Tech. Conf. on Circuits/Systems Computers and Communications – ITC-CSCC, p. 536-538, 2008.

[DEA99] DeANGELIS, Gregory C.; NEWSOME, Willian T. **Organization of Disparity-selective Neurons in Macaque Area MT.** The Journal of Neuronscience 19(4): 1398-1415, 1999.

[DEA00] DeANGELIS, Gregory C. **Seeing in Three Dimensions: the Neurophysiology of Stereopsis.** Trends Cogn Sci 4: 80–90, 2000.

[DES07] DE SOUZA, A. F.; MACHADO, F. **Visão artificial na medição e controle da produção e fluxo de materiais - do manuseio de granel até as linhas de produção.** XXVI Seminário de Logística, 2007, Vitória. Anais XXVI Seminário de Logística, 2007 (em CD).

[DOD03] DODGE, R. **Five Types of Eye Movement in the Horizontal Meridian Plane of the Field of Regard.** Am. J. Physiol. 8: 307-329, 1903.

[GEG97] GEGENFURTNER, Karl R. **Functional Properties of Neurons in Macaque Area V3**. The Journal of Neurophysiol. 77: 1906-1923, 1997.

[GON98] GONZALEZ, F.; PEREZ, R. **Neural Mechanisms Underlying Stereoscopic Vision**. Progress in Neurobiology. 55: 191-224, 1998.

[FUN04] FUNG, J.; MANN, S. **Using Multiple Graphics Cards as a General Purpose Parallel Computer Applications to Computer Vision**. In: Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04), p. 805-808, 2004.

[HAL08] HALFHILL, T. R. **Parallel Processing with CUDA: Nvidia's High-Performance Computing Platform Uses Massive Multithreading. Microprocessor**, Microprocessor Report, January 2008. Disponível em: http://www.nvidia.com/docs/IO/55972/220401_Reprint.pdf.

[GIB08] GIBSON, J.; MARQUES, O. **Stereo Depth With a Unified Architecture GPU**, IEEE Computer Vision and Pattern Recognition Workshops, 2008. DOI: 10.1109/CVPRW.2008.4563092.

[HEY84] VON der Heydt R.; PETHERHANS E.; BAUMGARTNER G. **Illusory Contours and Cortical Neuron Responses**. Science, 224:1260-1262, 1984.

[HIR05] HIRSCHMÜLLER, H. **Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information**. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Vol. 2, 807-814, 2005.

[HUB62] HUBEL, D. H.; WEISEL, T. N. **Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex**. J. Physiol. 160: 106-154, 1962.

[JON87] JONES, J. P.; PALMER, L. A. **An Evaluation of the Two-dimensional Gabor Filter Model of Simple Receptive Fields in Cat Striate Cortex.** J. Neurophysiol 58:1233-1258, 1987.

[KAN00] KANDEL, Eric R.; SCHWARTZ, James H.; JESSELL Thomas M. **Principles of Neural Science**. 4th Ed. Prentice-Hall International, Inc., 2000.

[KOM02] KOMATI, Karin Satie; DE SOUZA, Alberto Ferreira. **Vergence Control in a Binocular Vision System using Weightless Neural Networks**. In: Proceedings of the 4th International Symposium on Robotics and Automation. Los Alamitos: IEEE, 2002.

[LCA09] LCAD. **Laboratório de Computação de Alto Desempenho**. http://www.lcad.inf.ufes.br/index.php?option=com_content&task=view&id=14&Itemid=65, último acesso em 10/08/2009.

[LUE04] LUEBKE, D.; HARRIS, M.; KRÜGER, J.; PURCELL, T.; GOVINDARAJU, N.; BUCK, I.; WOOLLEY, C.; LEFOHN, A. **GPGPU: General Purpose Computation on Graphics Hardware**. International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2004, Course Notes, Article No. 33, 2004.

[LUE08] LUEBKE, D. **GPU Computing: The Democratization of Parallel Computing**, 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08), Course Notes, 2008. Disponível em http://www.gpgpu.org/asplos2008/ASPLOS08-1-intro-overview.pdf.

[MATa] MATHER, George. **The Visual Cortex**. [on line] Disponível em http://www.lifesci.sussex.ac.uk/home/George_Mather/Linked%20Pages/Physiol/Cortex.html, último acesso 01/08/2009.

[MOVa] MOVELLAN, Javier R. **Tutorial on Gabor Filters.** [on line] Disponível em http://mplab.ucsd.edu/tutorials/pdfs/gabor.pdf, último acesso 01/08/2009.

[MOV85] MOVSHON, J. A.; ADELSON, E. H.; GIZZI, M. S.; NEWSOME, W. T. **The Analysis of Moving Visual Patterns**. In C. Chagas, R. Gattass, C. Gross (eds.), Pattern Recognition Mechanisms. New York, Springer, 117-151, 1985.

[NIK08] NICKOLLS, J.; BUCK, I.;GARLAND, M.; SKADRON, K. **Scalable Parallel Programming With CUDA**. ACM Queue Vol. 6, Issue 2, p. 40-53, 2008.

[NVI06] NVIDIA. **Technical Brief: NVIDIA GeForce 8800 GPU Architecture Overview**. [on line] Disponível em http://www.nvidia.com/cuda, 2006.

[NVI07] NVIDIA. **NVIDIA CUDA: Compute Unified Device Architecture - Programming Guide 1.0**. [on line] Disponível em http://www.nvidia.com/cuda, 2007.

[NVI08a] NVIDIA. **NVIDIA CUDA: Compute Unified Device Architecture - Programming Guide 2.0**. [on line] Disponível em http://www.nvidia.com/cuda, 2008a.

[NVI08b] NVIDIA. **Technical Brief: NVIDIA GeForce GTX 200 Architectural Overview**. [on line] Disponível em http://www.nvidia.com/cuda, 2008b.

[OHZ97] OHZAWA, I.; DeANGELIS, Gregory C.; FREEMAN, Ralph D. **Encoding of Binocular Disparity by Complex Cells in the Cat's Visual Cortex**. The Journal of Neurophysiology, 77: 2879-2909, 1997.

[OLI05] OLIVEIRA, H. **Uma Modelagem Computacional de Áreas Corticais do Sistema Visual Humano Associadas à Percepção de Profundidade**. Dissertação de Mestrado. Programa de Pós-Graduação em Informática, UFES, 2005.

[PET68] PETTIGREW, J.D. et al. **Binocular Interaction on Single Units in Cat Striate Cortex: Simultaneous Stimulation by Single Moving Slit with Receptive Fields in Correspondence**. Exp. Brain res. 6: 391-410.

[QIA97a] QIAN, Ning; ZHU, Yudong. **Physological Computation of Binocular Disparity**. Vision Res. 37: 1811–1827, 1997.

[QIA97b] QIAN, Ning. **Binocular Disparity and the Perception of Depth**. Neuron 18: 359-368, 1997.

[RAN03] RANDIMA, F.;MARK, K. **The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics**, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[ROS06] ROSENBERG, D. I.; DAVIDSON, P. L.; MULLER, C. M. R.; HAN, J. Y. **Real-time Stereo Vision Using Semi-Global Matching on Programmable Graphics Hardware**. ACM SIGGRAPH Sketches, Article No. 89, 2006.

[RUS01] RUSINKIEWICZ, S.; LEVOY, M. **Efficient Variants of the ICP Algorithm.** Proceedings of the International Conference on 3D Digital Imaging and Modeling (3DIM). p. 145–152, 2001.

[TOO82] TOOTELL, R. B.; SILVERMAN, M. S.; SWITKES, E.; VALOIS, R. L. **Deoxyglucose Analysis of Retinotopic Organization in Primate Striate Cortex**. Science, 218: 902-904, Nov 26, 1982.

[ZEK73] ZEKI, S. M. **Colour Coding of the Rhesus Monkey Prestriate Cortex**. Brain Research, 422-427, 1973.