4
Detecção de Formas: Multirresolução e Agrupamento de Pixels

Este capítulo apresenta algoritmos que oferecem suporte à camada de processamento do conteúdo visual denominada *Detecção de Formas*. Mais especificamente, temas relacionados ao tratamento de descontinuidades nas superfícies visíveis na imagem em relação ao ponto de vista do observador.

As formas detectadas ao se observar uma cena variam enormemente, de acordo com a escala em que a imagem da visualização da cena é produzida (For02). Este fato motiva a abertura dos temas abordados neste capítulo, relacionados a diferentes representações do conteúdo visual em GPU, pelo tratamento de representações em multirresolução a ser apresentado na Seção 4.1.

O termo segmentação cobre uma variedade de problemas e técnicas em visão computacional. Forsyth e Ponce apresentam em (For02) uma distinção em três categorias principais para os problemas relacionados ao conceito de segmentação, os quais são classificados como: tarefas de partição, agrupamento e encaixe.

De forma genérica, o encaixe acontece quando se deseja encontrar instâncias de características descritas por um modelo pré-existente, geométrico ou probabilístico, através do enquadramento de tal modelo ao conjunto de dados.

O particionamento acontece quando se deseja dividir um grande conjunto de dados em partes menores, no interior das quais existe uma coerência, atendendo a um critério lógico, definido por um modelo pré-estabelecido. O conceito de partição é explorado na Seção 4.2 para decompor o conjunto de pixels de uma imagem em um conjunto com número reduzido de elementos, utilizando a homogeneidade cromática entre pixels vizinhos como critério de coerência para o interior das regiões criadas.

O agrupamento é caracterizado pela formação de aglomerados de itens distintos em conjuntos que façam sentido ao contexto da aplicação. A Seção 4.6 explora este conceito para definir agrupamentos de pixels ou de regiões de pixels de características distintas na criação de aglomerados para segmentação

de um objeto.

Dando suporte às propostas apresentadas neste capítulo, estruturas de partição são apresentadas nas seções 4.3 e 4.4 e utilizadas como motivação para acelerar o algoritmo de agrupamento de pixels em objetos de uma cena natural apresentado na Seção 4.2.

Também neste capítulo, os algoritmos apresentados foram desenvolvidos de maneira pertinente à implementação paralela em GPU.

4.1 Multirresolução

Diversas tarefas de visão computacional estão relacionadas ao conteúdo de regiões de uma imagem, cujas informações não são obtidas pela análise local de valores de pixels isolados. A criação de algoritmos baseados em análises de multirresolução procura reverter características resultantes do processo de formação das imagens, no qual objetos maiores e mais próximos à câmera ocupam áreas maiores da imagem, podendo ser reconhecidos em versões da imagem com baixa resolução, enquanto objetos pequenos ou distantes da câmera ocupam poucos pixels, exigindo análises em resoluções altas (For02). Entre os algoritmos que se beneficiam da análise em multirresolução, destacam-se, entre outros, os destinados às tarefas de reamostragem, segmentação, detecção de arestas, compressão de imagens e análise de movimento.

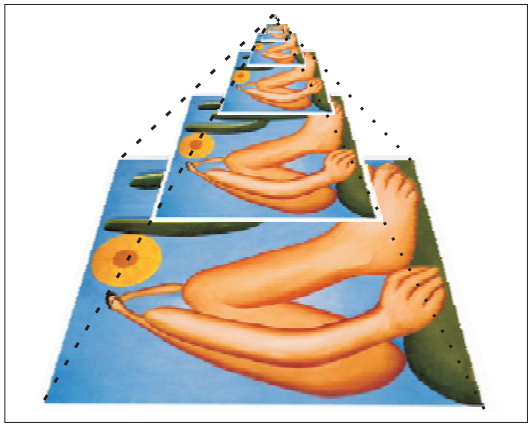


Figura 4.1: Pirâmides de Imagens

A construção mais popular entre as tarefas de visão computacional que utilizam informações em diferentes escalas é a de pirâmide de imagens (Ade84, Rod02, Mar06). As pirâmides descrevem o conteúdo da imagem em uma estrutura hierárquica, contendo a imagem em resolução original em sua base, versões em resoluções decrescentes em seus níveis intermediários, até representar a imagem por um único pixel em seu topo (Figura 4.1).

O algoritmo básico para a construção de pirâmide de imagens calcula o valor de cada pixel de um determinado nível de resolução a partir apenas dos dados com resolução mais alta do nível anterior. Os dados consultados são obtidos pela leitura de um número constante de pixels do respectivo nível anterior. Diferentes operações, tradicionalmente computadas em CPU, são utilizadas pelos algoritmos baseados em multirresolução no processo de reamostragem ou aglomeração entre amostras de níveis consecutivos da pirâmide.

A computação de pirâmides de imagem em hardware gráfico foi introduzida em sistemas gráficos 3D, pela técnica denominada "mipmapping". Esta técnica foi desenvolvida com o intuito de minimizar distorções ("aliasing") na aplicação de texturas com diferenças de amostragem entre a textura original e a área na qual deve ser aplicada, oferecendo suporte a interpolação linear e bilinear dos texels.

O aumento da programabilidade do hardware gráfico viabilizou a codificação das diferentes operações de aglomeração comumente utilizadas em algoritmos de construção de pirâmides de multirresolução. Entre as mais utilizadas que passaram a ser codificáveis em GPU destacam-se: as pirâmides gaussianas, construídas pela aplicação de filtros gaussianos como peso para as amostras usadas em cada aglomeração, as pirâmides laplacianas, construídas pela diferença entre duas gaussianas, as pirâmides de arestas (a de Sobel é seu exemplo mais característico) e pirâmides morfológicas. Além das operações clássicas, o atual suporte à programação das placas permite a codificação de funções arbitrárias para a aglomeração das amostras de níveis distintos, sendo a mais simples obtida pela média das amostras de níveis consecutivos.

Sobre as primeiras tentativas de computação de pirâmides em GPU programáveis, destaca-se ainda a necessidade inicialmente existente de transferir os dados de cada nível computado entre a memória gráfica e a da CPU. O custo de tais transferências era considerado como gargalo para esse tipo de computação. O suporte à renderização para textura eliminou a necessidade de fazer sucessivas cópias dos pixels renderizados da memória gráfica para a principal e de volta para a memória gráfica como textura (Str06). Esse recurso permite que os dados produzidos sejam reutilizados diretamente como textura em passadas seguintes, eliminando, portanto, tal gargalo.

Para a codificação em GPU, a computação de pirâmides de imagens pode ser vista como a aplicação do padrão de programação paralela de operação de redução, no qual os resultados intermediários são armazenados como níveis internos da pirâmide e a operação sobre as amostras para formação de um novo nível, codificada de acordo com o interesse da aplicação.

4.2 Partição de imagens

A visão humana decompõe uma imagem em partes correspondentes aos objetos visualizados. Em visão computacional tal decomposição se torna um problema de inferência, definido sobre os dados adquiridos, o qual procura determinar quais desses dados capturam informações sobre a visualização de quais objetos. Em grande parte das cenas naturais não é viável formular tal inferência a partir da análise isolada do valor de cada pixel, uma vez que as informações locais de cor e luminância, quando utilizadas isoladamente, tornam-se muitas vezes ambíguas para a identificação dos objetos.

Com o artifício de fornecer uma representação mais compacta, diversos algoritmos de segmentação de objetos utilizam um nível intermediário para o conteúdo visual criado pelo particionamento do conjunto inicial de pixels em regiões com propriedades similares. Tais algoritmos assumem que pixels pertencendo à mesma região possuem grande chance de pertencerem ao mesmo objeto (Gon01).

Nas seções seguintes abordamos o tema de segmentação de imagens como o processo em baixo nível de particionar o conjunto inicial de pixels em subconjuntos que apresentam propriedades fotométricas homogêneas. Nesta abordagem, o resultado esperado para o processo de segmentação é um conjunto de regiões que juntas cobrem inteiramente a imagem original.

Grande parte dos algoritmos de segmentação envolvem o uso de limiares associados aos seus parâmetros. A escolha propícia de tais limiares é uma tarefa extremamente complexa que influi diretamente na qualidade dos resultados de tais algoritmos. Abordagens para definição automática de limiares comumente utilizadas incluem a utilização de informações previamente conhecidas, tais como, intensidades, forma ou tamanho característicos dos objetos observados ou, ainda, sobre o número total de regiões a serem produzidas pelo algoritmo.

Existem três classes de estruturas utilizadas para representar a partição criada pela segmentação de imagens: os grafos de adjacência de regiões (RAG - "Region Adjacency Graph"), as árvores de imagens ("picture trees"), e as "supergrids". Em um RAG, as regiões segmentadas da imagem são representadas como nós de um grafo e as relações de adjacência entre regiões são representadas por arcos entre eles, indicando a existência de borda em comum entre as regiões por eles representadas. As árvores de imagens procuram enfatizar a inclusão de uma região dentro de outra região, fornecendo uma partição recursiva hierárquica de uma imagem em partes componentes. A partição é repetida até satisfazer condições de características constantes dentro de cada parte. As "supergrids", por sua vez, oferecem suporte à precisão de subpixel,

viabilizando a definição de bordas entre pixels. Nelas, cada pixel é rodeado por oito pontos virtuais (não-pixels) incluídos para o cálculo de bordas. Para isso, a partir de uma imagem de resolução $w \times h$, é criada uma grade denominada "supergrid"de tamanho $(2w+1)\times(2h+1)$. Iniciativas de algoritmos sequenciais de segmentação com representação de RAGs e "supergrids"são encontradas, respecitvamente, em (Tan77, Saa94, Lez03) e (Kot02).

4.3 Representação de imagens com Quadtrees

As quadtrees são uma estrutura de dados espacial baseadas no princípio de decomposição recursiva do espaço. São definidas de forma que seu nó raiz seja formado por um quadrado cobrindo o espaço total representado pela estrutura. Recursivamente, cada nó é subdividido em quatro nós filhos na forma de quadrados idênticos representando seus quadrantes (Zac03, Sam08), com exceção dos nós folhas, nos quais um critério pré-estabelecido de parada da subdivisão é atendido. As quadtrees são classificadas como uma decomposição regular do espaço, por criar subdivisões em partes iguais a cada nível.

Como estrutura espacial de dados, quadtrees são usadas para armazenar dados de diferentes natureza, tais como, dados pontuais, curvas e dados de pixels de imagens, sendo estes últimos os de maior interesse desta tese. No contexto de representação de imagens, as quadtrees são consideradas um caso especial de árvores de imagens.

Uma imagem, como vetor bidimensional de pixels, pode ser representada por uma quadtree, tradicionalmente denominada quadtree de regiões ("region quadtrees"), construída pela subdivisão sucessiva do vetor de pixels da imagem em quatro quadrantes de tamanhos iguais, até que as regiões cobertas por tais quadrantes representem grupos homogêneos de dados por um critério de uniformidade previamente definido (Sam90). Embora tal formulação apresente subdivisões sucessivas em um processamento da raíz para as folhas ("top-down"), na construção de quadtrees de regiões, normalmente tal procedimento é realizado das folhas para a raíz ("bottom-up"), pelo agrupamento das folhas da quadtree, seguido pelo agrupamento dos níveis internos, até atingir a raiz.

O procedimento "bottom-up"descrito é semelhante à construção de uma pirâmide de imagens. A dualidade existente entre quadtrees e pirâmides de imagens foi inicialmente relatada por Knuth (Knu73) e Tanimoto e Pavlidis (Tan75). Tal dualidade estabelece que os pixels na base da pirâmide correspondam às folhas da quadtree, os pixels em imagens de menor resolução na pirâmide sejam associados aos nós internos, o topo da pirâmide à raiz da quadtree, enquanto os agrupamentos de pixels de um nível para formar os do nível

seguinte da pirâmide correspondem aos arcos da quadtree. Tal dualidade é explorada nas propostas de computação de quadtrees em GPU apresentadas a seguir.

4.3.1 Quadtrees de regiões em GPU

Ziegler et al. propõe em (Zie07) a construção de quadtrees em GPU utilizando um operador de redução que produz uma pirâmide de imagens denominada *QuadPyramid*. O núcleo de processamento do redutor proposto é responsável pela criação dos texels da *QuadPyramid* com um tratamento diferenciado, de acordo com os valores consultados pelo redutor no nível anterior da pirâmide, em três possibilidades distintas.

O primeiro tratamento lida com a possibilidade de os valores consultados representarem informações semelhantes. Neste caso, o fragmento corrente sendo gerado recebe a média de tais valores, além de um rótulo com valor -1 sinalizando o agrupamento realizado dos pixels similares do nível anterior da pirâmide.

O segundo tratamento lida com a decisão de não agrupar os pixels do nível anterior, ao identificar valores não similares. Neste caso, um fragmento com rótulo de valor 4 é produzido, representando o número de folhas filhas do nó corrente.

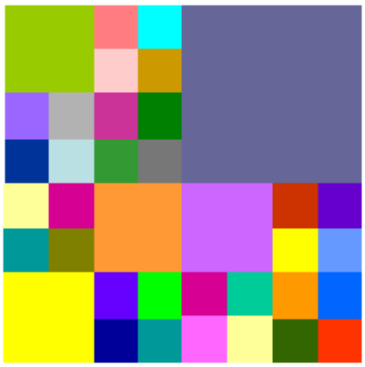
O terceiro tratamento lida com os casos em que pelo menos um dos pixels consultados já não representa agrupamento de regiões similares, ou seja, já deixaram de ser agrupados no processamento de níveis anteriores da pirâmide. Em tais casos, o fragmento é emitido com um rótulo equivalente à soma dos módulos dos rótulos dos pixels consultados, calculando, desta forma, o total do número de folhas cobertas pela região correspondente ao textel gerado.

Uma vez construída a *QuadPyramid*, a partir de sua análise, uma segunda etapa de processamento constrói uma lista ordenada das folhas da quadtree. A lista contém as informações de cada folha, descrevendo sua posição de origem no espaço da imagem e seu tamanho (ou nível na árvore). Para construir a lista em paralelo, são produzidos tantos fragmentos quantas forem as folhas da quadtree. Esse número pode ser lido na raiz da *QuadPyramid*. Cada processamento independente fica responsável por produzir o conteúdo do i-ésimo fragmento pela identificação na textura da *QuadPyramid* da i-ésima folha da lista. A busca por uma determinada folha na pirâmide simula um atravessamento ("traversal") da árvore da quadtree, inicializado no nó raiz e percorrendo os nós internos da árvore, até alcançar a folha correspondente. Para controlar tal atravessamento, o índice da folha sendo buscada é comparado com uma

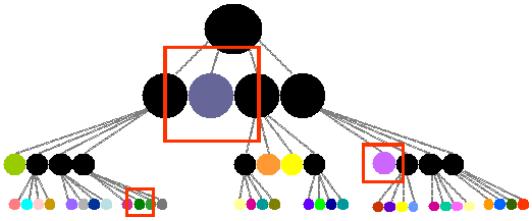
soma do número de folhas cobertas por cada uma das regiões percorridas, a qual pode ser atualizada pela leitura dos rótulos armazenados nos nós da *QuadPyramid*.

Nesta segunda fase, para construir uma lista contendo m folhas de uma quadtree cobrindo uma imagem de N pixels, tal algoritmo requer $O(m * \log(\sqrt{(N)}))$ acessos à textura para computar tais atravessamentos.

4.3.2 Representação das Quadtrees



4.2(a): Subdivisão espacial em Quadtree



4.2(b): Representação hierquica

4.2(c): Representação Linear

Figura 4.2: Quadtrees e suas representações

O modelo adotado pelo algoritmo desenvolvido por Ziegler et al. (Zie07) para representação de uma quadtree como uma lista contendo apenas suas folhas é conhecido tradicionalmente como representação linear ou representação sem ponteiros ("pointerless") (Fri02) (Figura 4.2 (c)). Nesse tipo de representação, um código de localização é associado a cada folha, permitindo que sejam feitas buscas e navegações na árvore por operações aritméticas entre pares de folhas, utilizando tais códigos como chaves.

O modelo mais popular entre as aplicações em CPU para representação de quadtrees é conhecido como representação hierárquica, na qual a descrição da quadtree é armazenada em uma estrutura de árvore composta por um nó raiz, nós internos e nós folhas (Zac03) (Figura 4.2 (b)). Cada nó possui ponteiros para os nós seus filhos e, opcionalmente, para o nó seu pai. Nessa representação as operações básicas, tais como navegação, localização de folhas que cobrem uma determinada posição do espaço, testes de vizinhança, operações de junção

ou quebra de folhas, entre outras, são solucionadas percorrendo a árvore utilizando tais ponteiros.

Portanto, para responder questões de vizinhança utilizando uma dessas formas de representação faz-se necessário a execução de testes aritméticos a ser aplicado ao conjunto de folhas ou atravessar a árvore percorrendo nós internos. Uma vez que os dados representados por uma quadtree são normalmente armazenados em seus nós folhas, diversos algoritmos são construídos utilizando operações de consultas a vizinhanças de folhas. Uma primeira solução para responder tais consultas de maneira eficiente pode ser formulada pela inclusão de referências em cada nó folha para cada uma das suas folhas vizinhas. Tal solução, entretanto, é extremamente ineficiente, tanto em relação ao espaço de armazenamento requerido, quanto a operações de manutenção para atualização da estrutura da quadtree. Para uma implementação em GPU, outra desvantagem de tal abordagem é ser desconhecido a priori o número de vizinhos de cada folha e, portanto, o número de referências adicionadas a cada folha. Nos casos extremos, esse número pode alcançar até, aproximadamente, o total de folhas da árvore.

Motivados pela necessidade de responder questões de vizinhança de maneira eficiente, formulamos na próxima seção um novo modelo de representação para quadtrees, que permite navegar por entre o sistema de vizinhança de suas folhas, sem a necessidade de percorrer ou armazenar os nós internos da árvore. Também é apresentado como esta nova estrutura pode ser construída em GPU.

4.4 QuadN4trees: sistema de referências entre folhas vizinhas

Em (Vas08a) (Anexo C) apresentamos um novo modelo de representação de quadtrees que recebeu o nome de *QuadN4tree*, com o intuito de destacar a alteração na representação dos seus nós folhas, pela inclusão de referências para quatro folhas vizinhas especialmente selecionadas (N do inglês "neighbouring").

O quarteto de referências é escolhido de forma a permitir uma navegação entre folhas vizinhas seguindo o contorno de cada uma das bordas de uma determinada folha, ou seja, permite visitar suas vizinhas em sequências estabelecidas em relação às suas bordas.

No algoritmo apresentado, padronizamos a adoção de quatro referências: uma para o norte e uma para o leste, tomadas de forma alinhada com o canto inferior esquerdo da folha ("left bottom corner-- *lbc*); uma para o sul e uma para o oeste, tomadas em alinhamento ao canto superior direito ("right top corner-- *rtc*) de cada folha (Figura 4.5 c.). A Figura 4.3 apresenta o modelo

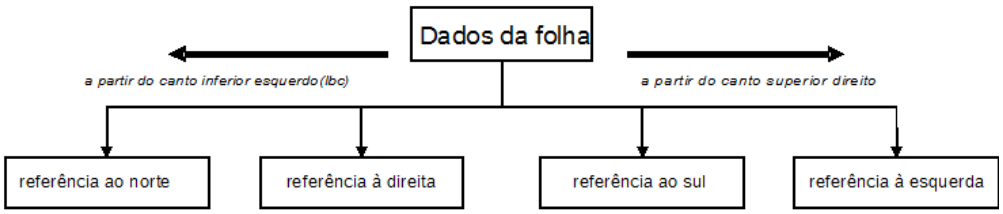
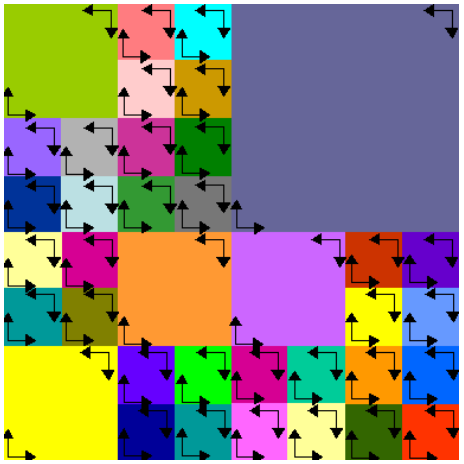
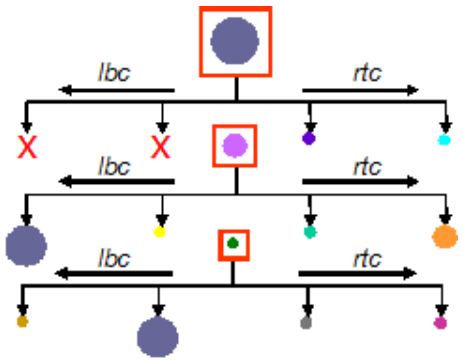


Figura 4.3: QuadN4: modelo de referências para folhas vizinhas



4.4(a): ilustração gráfica do modelo



4.4(b): representação do armazenamento

Figura 4.4: Construção da QuadN4tree para a Figura 4.2

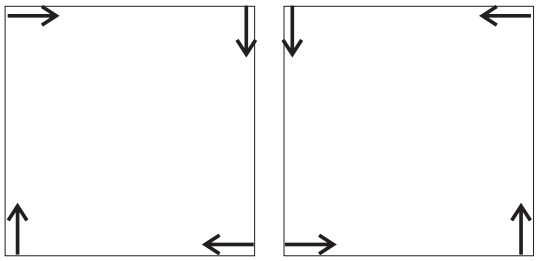
de representação de folhas no modelo QuadN4. A Figura 4.4 apresenta um exemplo de uma QuadN4tree construída para a subdivisão apresentada na Figura 4.2.

Observamos que existem outras combinações de referências que atendem às mesmas propriedades que o padrão adotado, ao permitir a navegação por folhas vizinhas, independentemente das possíveis combinações de tamanho de tais folhas. As variações ilustradas na Figura 4.5 sugerem outras combinações de referências que atendem às propriedades do modelo QuadN4 ao escolher os pares de referências cobrindo as direções vertical e horizontal em sentidos opostos e tomados a partir de cantos opostos de cada folha.

4.4.1 Regras para navegação por vizinhos de uma borda

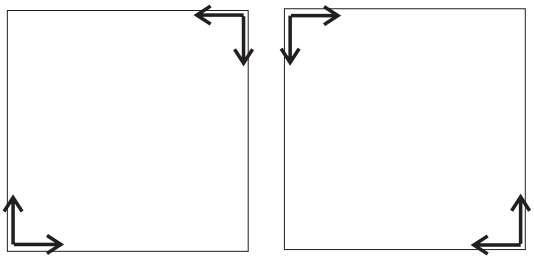
O quarteto de referências adotado permite percorrer as folhas vizinhas de uma determinada folha a partir do conjunto de regras estabelecido a seguir. As regras descrevem uma navegação sem a utilização de nós internos da árvore, mas apenas das referências do modelo QuadN4.

Considerando uma folha A, para recuperar os vizinhos de A em sua borda norte, inicialmente são lidas as dimensões de A para serem usadas no



4.5(a): sentido anti-horário

4.5(b): sentido anti-horário



4.5(c): padrão adotado com sentido misto

4.5(d): outro de sentido misto

Figura 4.5: Diferentes possibilidades para tomada de referências atendendo às propriedades do modelo QuadN4

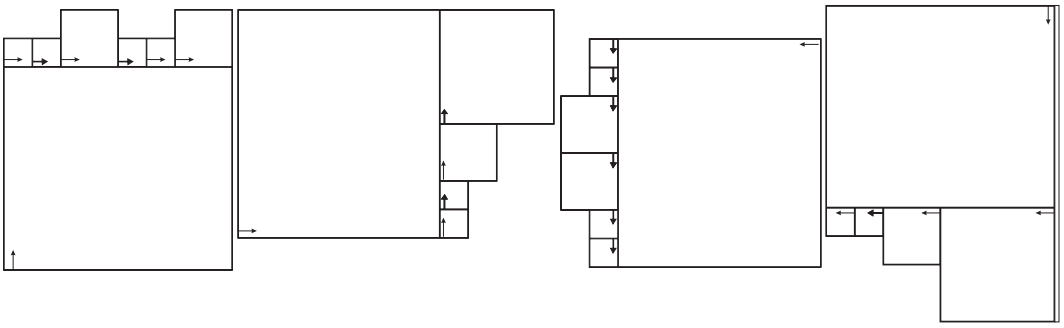


Figura 4.6: Navegação pelas Bordas Norte, Leste, Oeste e Sul

controle da busca. A navegação para a primeira vizinha B é feita utilizando-se a referência para o norte, armazenada na representação de A no modelo QuadN4. Sucessivamente, para cada folha vizinha B, se a largura de B não ultrapassar o limite horizontal de A (definido pela posição horizontal do canto de A e seu tamanho), então a busca por novos vizinhos de A continua seguindo a referência de B para o leste, encontrando uma nova vizinha. A Figura 4.6 ilustra o procedimento de navegação.

Descrições similares podem ser formuladas para recuperar as folhas vizinhas pelo contorno da borda ao leste da folha. Inicialmente são obtidas as dimensões de A para o controle da busca que se inicia. Uma primeira vizinha B é encontrada seguindo a referência em A para o leste. Interativamente, para cada folha vizinha nova encontrada B, sua posição e extensão vertical são

comparadas com a borda de A. Enquanto as dimensões de tal vizinha não ultrapassarem às de A, novos vizinhos são obtidos seguindo a referência de B para o norte.

Da mesma maneira, para recuperar as folhas vizinhas na borda a oeste de A, é utilizada, inicialmente, sua referência para o oeste e a interação acontece seguindo as referências para o sul de cada folha nova vizinha encontrada. Neste caso, o limite de A testado é sua extremidade inferior, uma vez que a navegação está sendo feita de cima para baixo da borda.

De forma análoga, para recuperar as folhas vizinhas da borda ao sul de A, inicialmente é utilizada sua referência para o sul e segue-se pelas vizinhas utilizando suas referências para o oeste até ultrapassar a posição do canto esquerdo de A.

Na prática, como as imagens representadas pela QuadN4tree são representadas por um sinal discreto, os testes de limites das folhas são feitos em função do número de pixels e posição de origem da folha no espaço da imagem. Os testes podem ser alterados para que a navegação visite, também, as folhas sem borda em comum com a folha processada, mas vizinhas em relação a um de seus cantos. Esta adaptação é feita alterando o teste de parada da navegação para ultrapassar o número de pixels da folha original mais um.

4.4.2 Propriedades do Modelo QuadN4

A escolha das quatro referências por folha não é feita de forma arbitrária, mas sim com o intuito de cobrir o sistema de vizinhança existente entre as folhas de uma quadtree. Nesta seção mostramos porque as referências escolhidas no modelo QuadN4 atendem a tal requerimento.

O caso trivial no tratamento de vizinhança acontece em um par de folhas vizinhas que possuem o mesmo tamanho. A descrição do modelo QuadN4 assegura que tais folhas necessariamente possuem referências de uma para a outra e vice-versa, uma vez que as regras de subdivisão espacial de quadtrees impõem um mesmo intervalo para a borda na qual tais folhas fazem vizinhança. Mais especificamente, usando as referências do modelo QuadN4, as folhas de mesmo tamanho vizinhas horizontalmente referenciam uma a outra com, respectivamente, as referências para o leste e para o oeste, e as vizinhas verticalmente com as referências para o norte e para o sul.

Supondo um par de folhas vizinhas com tamanhos diferentes, as propriedades do modelo QuadN4, juntamente com a subdivisão espacial da quadtree, asseguram que a folha menor L_s necessariamente possui uma referência para a maior L_l, já que a borda da menor está, necessariamente e completamente

contida na borda da maior. Portanto, é possível navegar da folha menor para a maior seguindo a referência do modelo QuadN4 na folha L_s que aponta para a folha L_l.

Não é possível afirmar que em um par de folhas vizinhas de tamanhos distintos a folha maior possua, no modelo QuadN4, uma referência direta para a menor. Mostramos, então, que é possível navegar da folha maior para a menor, ainda que a maior não a referencie diretamente. Supondo que a folha menor não é alinhada com nenhuma das quatro referências, então, existe outra folha L_{s1} alinhada com a referência para borda de L_l, na qual L_s está situada. Pode-se assumir que L_{s1} também é menor que L_l ou a suposição inicial de que L_l e L_s sejam vizinhas seria contrariada. O modelo QuadN4 permite seguir junto à borda de L_l, na qual ambas L_s e L_{s1} estão situadas, inicialmente usando a referência de L_{s1} para alcançar outras vizinhas na mesma borda. Por recorrência, tal indução continua reafirmando as suposições de que cada folha encontrada é menor que L_l e possui uma referência para contornar a borda, seguindo a navegação de folha a folha, até atingir L_s.

A propriedade de permitir o contorno por cada uma das bordas de uma folha, não importando seu tamanho, garante que a QuadN4tree cobre o sistema de vizinhança das folhas no espaço 2D representado pela quadtree.

As propriedades expostas permitem a construção de um teste direto de vizinhança entre duas folhas quaisquer. É suficiente analisar se, pelo menos uma das duas folhas, possui uma referência do modelo QuadN4 direta para a outra.

4.4.3 Análise Assintótica da Representação QuadN4tree

As regras de navegação apresentadas podem ser diretamente aplicadas para converter da representação de vizinhança da QuadN4 para uma representação completa que armazena referências a todos os vizinhos de uma folha. O custo do processo de conversão por folha é associado a uma operação de leitura de referência por vizinho.

Supondo uma busca pelos n vizinhos de uma folha qualquer, a representação seguindo o modelo QuadN4 faz tal levantamento ao custo de $O(n)$ leituras de referências. Para realizar a mesma tarefa, a representação hierárquica realiza $O(n * d)$ operações, onde d é a altura da árvore e expressa o custo proporcional a n vezes a busca de uma única vizinha. Na representação linear, para recuperar as vizinhas de uma folha, a existência de borda em comum com cada folha da árvore precisa ser aritmeticamente testada, resultando em um

total de operações da ordem do número de folhas na árvore f, $O(f)$. [1].

Em outro cenário, avaliamos o custo de busca de um vizinho específico no modelo QuadN4. Os piores casos, em tal cenário, acontecem quando o vizinho procurado é o último da sequência de uma determinada borda, segundo o modelo QuadN4. Casos extremos acontecem quando a busca se iniciou em uma folha com o maior tamanho representado na quadtree e todos os vizinhos nesta borda são do nível mais baixo da quadtree.

Para avaliar este cenário, supomos uma quadtree cobrindo um espaço $2^n \times 2^n$, cujas menores folhas representam áreas de 1×1 e que a maior folha representada possui $2^{(n-1)} \times 2^{(n-1)}$ de área. Assim, a maior sequência percorrida é de $2^{(n-1)} - 1$ folhas, antes de se encontrar a última folha da sequência. Portanto, para este cenário é preferível o uso da representação hierárquica, na qual qualquer vizinho é encontrado percorrendo a árvore até um ancestral em comum e descendo até encontrar o vizinho desejado (Sam90).

O caso extremo da representação hierárquica acontece quando ambos os vizinhos são folhas do menor tamanho representado na árvore, portanto, armazenadas na maior profundidade e seu único ancestral em comum é a raiz, fazendo com que a busca percorra um total de duas vezes a altura ou profundidade da árvore. Neste cenário a QuadN4tree alcança o vizinho pela leitura de uma única referência, garantida pela propriedade de que folhas com o mesmo tamanho se referenciam uma a outra.

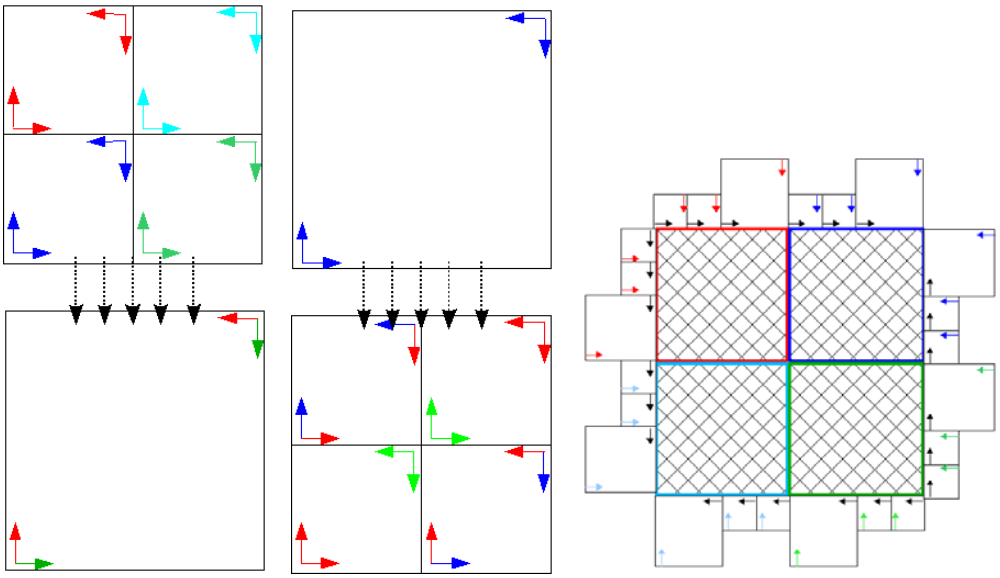
De maneira geral, a performance da busca por um vizinho específico na QuadN4tree melhora quanto mais equilibrados forem os tamanhos das folhas vizinhas ou, de forma equivalente, quanto mais próximos forem seus níveis na árvore.

4.4.4 Operações sobre a estrutura

As duas operações básicas de alteração da estrutura de quadtree são: o agrupamento de quatro folhas de um mesmo nível e mesmo nó pai em uma folha nova e a operação de quebra de uma folha para gerar quatro folhas filhas (Sam90). Esta seção apresenta o procedimento de atualização das referências do modelo QuadN4tree.

Ao criar uma nova folha, a partir da união de quatro folhas filhas de um mesmo nó interno da árvore, as referências da nova folha criada podem ser lidas diretamente das referências armazenadas nas folhas originais. As referências para o norte e para oeste da folha nova podem ser copiadas da folha mais ao

[1] O tempo de consulta na representação linear pode ser reduzido, como por exemplo, pela abordagem estatística apresentada por (Cas08)



4.7(a): Referências após agrupamento

4.7(b): Referências após quebra

4.7(c): Referências externas afetadas

Figura 4.7: Atualização de referências

norte e à esquerda entre as quatro agrupadas. Já as referências para o sul e para o leste podem ser lidas diretamente da folha inferior direita agrupada (as referências copiadas são ilustradas no lado esquerdo da Figura 4.7).

Ao quebrar uma folha em quatro novas folhas menores, manter o modelo QuadN4 equivale a criar 16 referências. Oito das referências novas no interior das folhas criadas apontam para as próprias folhas novas, quatro podem ser copiadas da folha original e as outras quatro precisam ter seus alvos buscados na vizinhança da área afetada. Os alvos são encontrados comparando os limites das novas folhas com os intervalos das bordas das folhas na região externa afetada (Figura 4.7 a direita).

As duas formas de operações na estrutura da quadtree demandam um procedimento local de atualização das referências no modelo QuadN4, afetando apenas as referências na vizinhança da(s) folha(s) atualizada(s), considerada neste texto como *área operacional* (área hachurada na Figura 4.7). O alinhamento imposto pelas regras de subdivisão espacial de uma quadtree garante que o conjunto de folhas a serem atualizadas possa ser obtido circulando a borda da *área operacional*. As mesmas regras de navegação ao redor de uma folha são aplicadas, mas alterando seus testes para as dimensões da *área operacional*.

4.5

QuadN4trees em GPU

Em (Vas08a) sugerimos que o modelo de referências QuadN4 pode ser incorporado às representações tradicionais linear e hierárquica por, respectivamente, inclusão de campos de referência na lista de folhas e inclusão de mais quatro apontadores na representação dos registros de folhas. Nesta seção descrevemos a representação mais desafiadora para a QuadN4tree, relativa a sua construção em paralelo na GPU.

Inicialmente, a quadtree é construída pelo procedimento proposto por Ziegler et al. (Zie07) (apresentado na Seção 4.3.1) e seus elementos organizados em uma lista de folhas. Para incorporar as referências do modelo QuadN4 a esta representação da quadtree em GPU, é preciso referenciar o quarteto apropriado de vizinhas em cada folha. Para executar esta tarefa, fazemos uma abstração de outra representação da quadtree em GPU, proposta em (Vas07) (Anexo D), na qual a quadtree é representada por uma estrutura de pirâmide de imagens. Ao contrário da representação em lista de folhas, a representação em pirâmide é propícia às consultas sobre localização e vizinhança na árvore, mas possui a desvantagem de requerer maior espaço de armazenamento. Por esta desvantagem, a representação da quadtree em GPU na estrutura de pirâmide de imagens é utilizada apenas como etapa intermediária para a construção da QuadN4tree, viabilizando consultas que fazem a identificação das referências do modelo QuadN4, enquanto a representação em lista é usada para armazenar efetivamente a descrição das folhas.

A seguir apresentamos como a representação da quadtree como pirâmide é construída na GPU (Subseção 4.5.1), como tal representação pode ser usada para responder consultas de localização e vizinhança (Subseção 4.5.2), como utilizar um processamento inverso para construir tal pirâmide partindo da lista de folhas (Subseção 4.5.3) e, finalmente, como obter a representação da QuadN4tree (Subseção 4.5.4).

4.5.1 Quadtrees como Pirâmide de Imagens em GPU

Em (Vas07), propomos a construção de uma quadtree em GPU produzindo uma estrutura distribuída em níveis hierárquicos. O algoritmo paralelo descrito opera em duas etapas de processamento.

A primeira etapa produz uma pirâmide de imagens representando uma árvore quaternária cheia. Cada elemento produzido na pirâmide representa um nó v da árvore cheia (Figura 4.8). Inicialmente, a base da pirâmide é criada com as dimensões da imagem que se deseja representar e cada folha da árvore recebe o valor de um pixel. Os nós de níveis intermediários da árvore são

construídos contendo avaliações sobre a similaridade ou homogeneidade das regiões por eles cobertas na imagem (no sentido que determinadas propriedades se mantêm em toda a sua extensão), além de valores representativos de tais regiões (normalmente é usada a média dos valores de seus quatro nós filhos).

A operação para construção da pirâmide cheia e avaliação da homogeneidade dos pixels nas regiões cobertas por seus nós é feita por um operador de redução bidimensional. O operador associa cada fragmento criado a um rótulo binário sinalizando a existência ou não de homogeneidade na região correspondente ao fragmento. Caso a análise de um fragmento decida não ser homogênea a região da imagem por ele coberta, todos os fragmentos produzidos nas passadas seguintes representado nós ancestrais do nó corrente também serão rotulados como não-homogêneos.

Uma segunda etapa do algoritmo é responsável por transformar a pirâmide de texturas produzida, representando uma árvore quaternária cheia, em uma representação de quadtree regional na qual cada nó folha atende ao critério de representar o agrupamento máximo de pixels semelhantes encontrado na análise da imagem obedecendo a subdivisão espacial imposta pela quadtree. Em uma única passada, esta segunda etapa identifica entre o conjunto de nós da árvore cheia quais nós atendem ao critério exposto para as folhas da quadtree.

Ao avaliar os resultados do procedimento da primeira etapa, a verificação de que o rótulo de um fragmento indica uma região não homogênea já é suficiente para que o nó correspondente a tal fragmento seja considerado um nó interno da quadtree. Por outro lado, o rótulo de homogeneidade não indica que um fragmento represente necessariamente uma folha da quadtree, já que pode ter sido reagrupado nos níveis mais altos na pirâmide, em processamentos subsequentes, para formação de folhas maiores.

A descrição do processamento paralelo efetuado para esta análise se inicia pela criação de um fluxo de elementos em disposição de pirâmide com tantos elementos quantos forem os da pirâmide de árvore cheia. Cada elemento deste fluxo é processado em paralelo e de forma independente. Este processamento avalia o elemento na pirâmide na mesma posição do elemento corrente, verificando seu rótulo. Quando o elemento consultado na pirâmide original recebeu o rótulo de não-homogeneidade na árvore cheia, o elemento corrente é associado a um nó interno da quadtree. Quando a consulta indica o rótulo de homogeneidade, a pirâmide representando a árvore cheia precisa ser consultada novamente, agora nas posições correspondendo ao nó pai do nó corrente. Nos casos em que também o nó pai recebeu o rótulo de homogeneidade, o nó corrente foi reagrupado em uma folha de tamanho maior na quadtree e o

fragmento corrente emite um sinal de vazio pois não representa um nó para a quadtree. Já nos casos em que o pai possui rótulo de não-homogeneidade, pode-se concluir que o nó corrente não foi agrupado aos seus vizinhos e portanto representa uma folha para a quadtree.

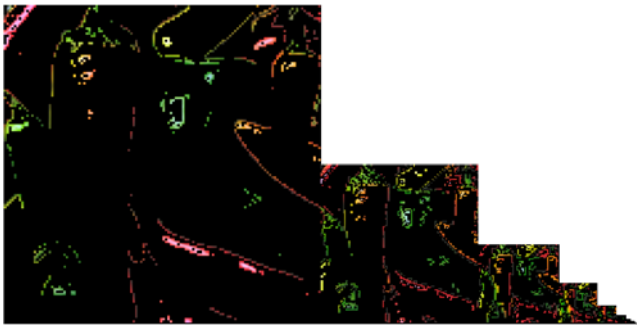


Figura 4.8: Máscara de Folhas

O fluxo processado ao ter seus elementos marcados com uma sinalização de *preenchidos*, para representar folhas, ou de *vazios*, nos demais casos, forma uma máscara binária chamada *textura de máscara de folhas* ou simplesmente *máscara de folhas* (Figura 4.8). Conforme descrito, a segunda etapa constrói esta máscara contendo apenas informações sobre os nós folhas dispostos de forma hierárquica, em um processamento utilizando no máximo dois acessos à textura por elemento da pirâmide.

4.5.2 Consultas à máscara de folhas

Uma das operações básicas sobre a estrutura de uma quadtree é a localização de qual folha cobre uma determinada posição no espaço 2D representado. Esta operação é necessária para a construção da QuadN4tree, uma vez que as referências são tomadas em relação a posições do espaço 2D, já que a descrição do modelo escolhe as referências por suas posições coladas às bordas e alinhadas às referências.

O objetivo da representação hierárquica para a quadtree é criar uma pirâmide que se comporta como uma máscara de folhas, contendo informação nas posições correspondente às folhas da árvore e contendo um valor associado à inexistência de informação nas demais, viabilizando o processamento de consultas à quadtree em GPU.

Propomos a formulação de consultas por uma função de mapeamento que utiliza a *textura de máscara de folhas* como uma máscara binária. Vale relembrar que no modelo de subdivisão de quadtrees cada ponto do espaço é coberto por uma e somente uma folha. Uma vez que a *máscara de folhas* preserva o nível das folhas e sua disposição espacial, é utilizada como uma

máscara que ativa na função de mapeamento o nível no qual um ponto do espaço 2D foi agrupado em uma folha. O mapeamento entre uma posição (x, y) do espaço 2D e sua folha representativa na quadtree é obtido por:

$$LeafData(x, y) = \sum_{i=1}^{N} \alpha_i * (level_i, dx_i(x), dy_i(y)) \quad (4\text{-}1)$$

na qual N é o número de níveis da quadtree, α_i é uma variável binária indicando se i é ou não o nível da folha cobrindo (x, y) cujo valor é lido da *máscara de folhas*, e $dx_i(x)$, $dy_i(x)$ são as coordenadas da posição inferior esquerda do canto da folha no nível i que cobre o ponto (x, y) (obtidas pela quantização das coordenadas de x e y pelo tamanho apropriado de uma folha em tal nível).

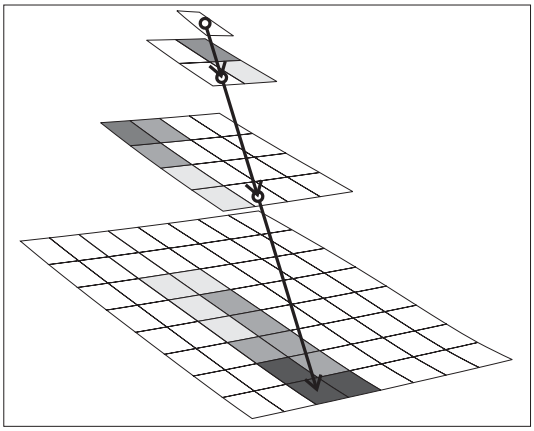
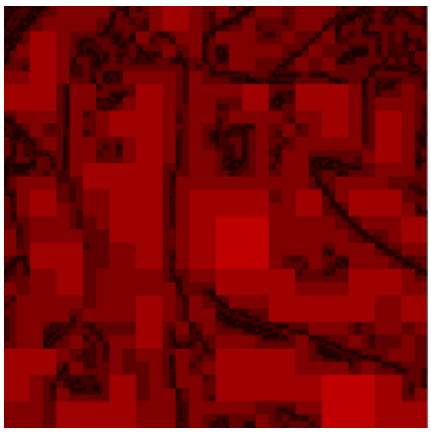


Figura 4.9: Raio atravessando a pirâmide partindo da raiz da árvore

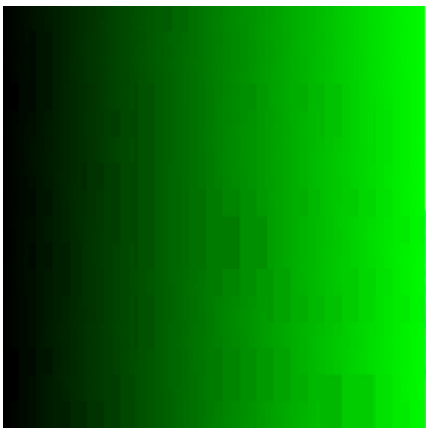
Em uma interpretação semântica da expressão 4-1, é possível imaginar as camadas da máscara de folhas dispostas efetivamente como uma pirâmide 3D e a consulta pela folha que representa uma posição 2D do espaço representado pode ser interpretada como o traçado de um raio ligando a raíz da árvore até a posição na base da pirâmide equivalente a posição procurada no espaço 2D. Tal raio deve interceptar apenas um elemento preenchido da máscara de folhas, o qual responde à consulta (Figura 4.9).

Como exemplo de aplicação do mapeamento, a Figura 4.10 é formada pelo mapeamento de cada uma das posições de seus pixels para a descrição das folhas que a eles representa na quadtree produzida pelo processamento da Figura 4.8.

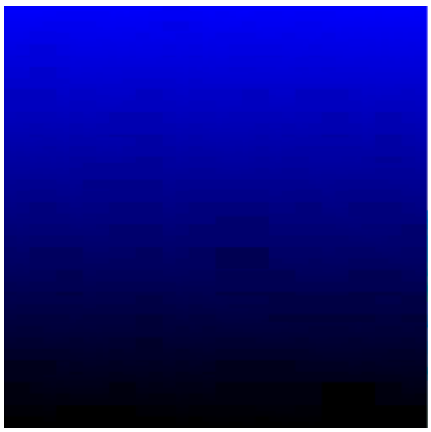
Em consultas de vizinhança, para identificar as características descritivas de folhas vizinhas, armazenadas como dados na máscara de folhas, basta traçar um raio para posições no espaço 2D na faixa de pixels em volta da borda de uma folha.



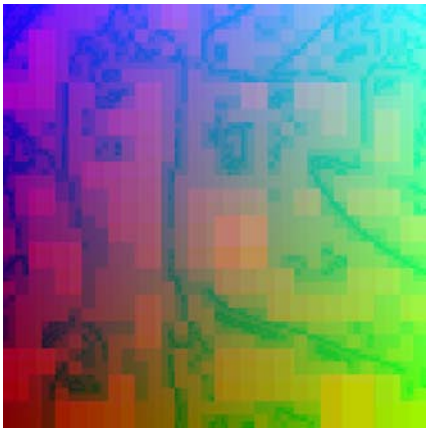
4.10(a): Nível da Folha(em vermelho)



4.10(b): Posição Horizontal do Canto (em verde)



4.10(c): Posição Vertical do Canto (em azul)



4.10(d): Mapa Composto

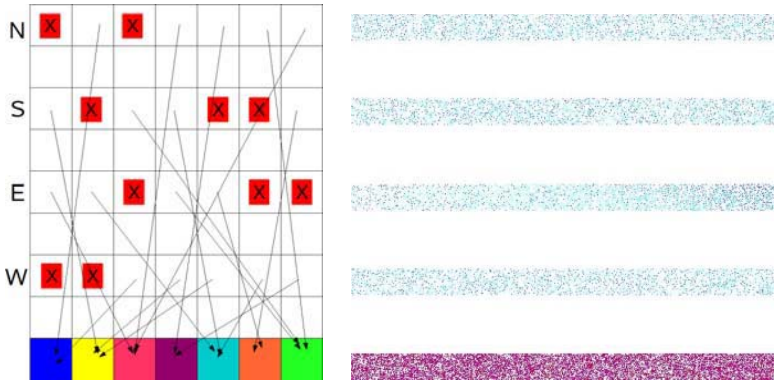
Figura 4.10: Consulta de Características das folhas

4.5.3 Da Lista de Folhas para a Pirâmide em GPU

Em sua versão original, a estrutura hierárquica apresentada como máscara de folhas é construída contendo os dados das folhas da quadtree. Para a obtenção das referências da QuadN4tree produzimos uma máscara partindo da lista de folhas produzida por (Zie07). Inicialmente um fluxo de vértices com tantos elementos quantas forem as folhas na lista é criado. Em um "vertex shader"cada processamento lê as informações sobre o nível e a posição de uma folha armazenadas na lista de folhas e posiciona o vertex em processamento sobre a posição correspondente na estrutura hierárquica, que tem seus níveis dispostos sequencialmente em um espaço 2D. O vértice que inicialmente é relacionado a uma posição na lista passa esta posição como um de seus atributos (por exemplo, sua cor) e é reposicionado para a estrutura hierárquica. Em um fragment shader, o atributo do fragmento, representando a posição da

folha na lista é repassado como conteúdo do fragmento. Para distinção entre essa pirâmide e a anteriormente apresentada, denominamos esta construção de *pirâmide de referências*.

4.5.4 Representação da QuadN4tree em GPU



4.11(a): Diagrama ilustrativo

4.11(b): Armazenamento em Textura (listas distanciadas para uma melhor visualização)

Figura 4.11: Armazenando a QuadN4tree na GPU

Após construir a lista de folhas pela abordagem de (Zie07) e a *máscara de folhas* pelo reposicionamento das folhas no espaço da hierarquia a partir de tal lista, quatro novas listas são criadas, cada uma armazenando uma das referências do modelo QuadN4 (Figura 4.11). As referências são encontradas aplicando uma consulta à *pirâmide de referências* para as quatro posições das vizinhas do modelos. Na pirâmide de referências uma consulta retorna a posição na lista de folhas, armazenada como sendo o dado de cada folha nela representada.

Portanto, o mapeamento entre uma coordenada (x, y) e sua posição na lista de folhas é expressa com a seguinte soma:

$$Referencia(x, y) = \sum_{i=1}^{N} \alpha_i * (LL_x(x, y, i), LL_y(x, y, i)) \qquad (4\text{-}2)$$

na qual (x, y) são coordenadas no espaço 2D representado pela quadtree, N e α_i são as mesmas variáveis da equação 4-1 e $LL_x(x, y, i)$, $LL_y(x, y, i)$ retornam respectivamente a posição no interior da lista de folhas da folha consultada que cobre o ponto (x, y) no nível i.

Seguindo a conotação semântica anteriormente apresentada, supondo conhecida a posição (x, y) e o tamanho t (em pixels) da folha a partir da qual se deseja obter as vizinhas, as referências do quarteto de folhas vizinhas QuadN4 podem ser obtidas pelo traçado de quatro raios sobre a pirâmide de

referências para as seguintes posições do espaço 2D: $(x, y + t)$, $(x + t, y)$, $(x - 1, y + t - 1)$ e $(x + t - 1, y - 1)$.

O algoritmo descrito para a construção da QuadN4tree é extremamente eficiente por ter todas as suas etapas descritas como processamentos independentes sobre fluxos de elementos, portanto sendo computados inteiramente em paralelo no hardware gráfico. As etapas tiveram seus tempos de processamento aferidos (em milisegundos) utilizando uma placa gráfica nVidia GeForce 8800 GTX, conforme apresentados na tabela 4.1. Os tempos apresentados confirmam a viabilidade de utilização da QuadN4tree em aplicações de tempo real.

Tabela 4.1: Tempo de processamento (em milisegundos)

Resolução do espaço representado:	$2^9 \times 2^9$	$2^{10} \times 2^{10}$	$2^{11} \times 2^{11}$
Pirâmide de Images:	0.14	0.29	0.37
Lista de Folhas:	0.47	1.02	1.37
Máscara de Referências:	0.28	1.00	1.29
Lista de Vizinhança (QuadN4):	0.22	0.72	1.17

4.6 Agrupamento para detecção de objetos

Uma face, um carro ou um objeto qualquer são identificados pela visão humana pelo agrupamento de regiões com características distintas na imagem. Diversas abordagens em visão computacional para detectar determinados objetos dispostos em cenários naturais utilizam modelos baseados na discriminação ou distinção entre imagens ("discriminability") (Roh97), definida como a diferença entre pares de imagens.

Entre os diversos modelos que adotam a diferença entre pares de imagens para resolver problemas de segmentação de objetos em imagens naturais, os algoritmos baseados na minimização de energia via corte em grafo ("Graph Cuts") se destacam pela qualidade dos resultados produzidos e por serem conhecidos métodos para sua resolução em tempo polinomial ao número de elementos avaliados.

Além da segmentação, outras tarefas de visão computacional e processamento de imagens podem ser naturalmente expressas como problemas de otimização de uma função de energia que descreve suas propriedades. Mais especificamente, a técnica de minimização de energia via corte em grafo fornece uma maneira de repensar em tais tarefas como uma busca pela melhor classificação dos seus elementos de entrada, os quais normalmente representam os pixels, entre um conjunto pré-definido de rótulos ("labels"), representando

os possíveis elementos de saída. Uma vez repensadas tais tarefas como problemas de classificação, o maior desafio é definir apropriadamente uma função de energia que modele coerentemente as propriedades específicas da aplicação, associando benefícios (ou custos) relativos à classificação por cada rótulo. A formulação apropriada de um grafo expressando o problema de classificação é apresentada na Subseção 4.6.1.

Alguns exemplos de outras aplicações de visão computacional, modeladas como um problema de classificação e que utilizam o corte de grafo para encontrar sua solução ótima, incluem: coloração (Yun06) e restauração de imagens (Boy98, Raj05), composição de fotos (Aga04, Wil05), produção de texturas (Kwa03), pareamento em estéreo (Kol01, Wei05, Ble07), análise de movimento (Ble06, Wan07) e aquisição de modelos 3D (Fleck).

Dando continuidade aos temas abordados por esta tese, nesta seção apresentamos a segmentação de imagens em objeto/fundo formulada como uma classificação binária dos pixels. O método de corte de grafo pode ser interpretado como um algoritmo de aglomeração ("clustering") que trabalha no espaço de características da imagem para produzir agrupamentos ("clusters") espacialmente coerentes. A coerência espacial é garantida pela inclusão no modelo de uma penalidade imposta à associação de pixels vizinhos a rótulos distintos. No grafo, tais penalidades são incluídas como pesos para as arestas entre pixels vizinhos.

A abordagem mais comum para a segmentação objeto/fundo em cenários naturais (Rot04, Yin04, Wan05) processa uma imagem de entrada a partir de indicações brutas iniciais de pixels, tanto do objeto quanto do fundo, fornecidas pelo usuário da aplicação. Tais indicações são usadas para compor as restrições iniciais do processo de minimização, calibrando a função de energia. O corte em um grafo apropriado é então aplicado para encontrar automaticamente a segmentação global ótima para o restante da imagem.

A formulação adotada nesta seção para segmentação objeto/fundo em imagens naturais via corte de grafo foi proposta por Sá et al. (Sa06) e modela a função de energia, com base na discriminação de um par de imagens, obtido pela técnica conhecida como *iluminação ativa* (Subseção 4.6.2). Essa técnica consiste no uso de uma fonte de luz adicional na captura da cena, a qual é posicionada de frente para os objetos que se deseja segmentar, de forma a iluminá-los com maior intensidade que o fundo. O par de imagens é capturado variando a intensidade de tal fonte de luz, de forma que a discriminação entre as imagens ofereça um indício automático para a diferenciação entre as regiões de objeto e de fundo, dispensando a intervenção manual de um usuário.

Existem três linhas de iniciativas distintas para acelerar a computação

de otimizações via corte de grafos, aqui caracterizadas, respectivamente: pela diminuição do número de elementos sobre o qual a otimização é aplicada; pela utilização do hardware gráfico com o intuito de acelerar a computação dos termos da função de energia; pela computação em paralelo do corte em grafo.

A computação em paralelo do corte em grafo é a mais recente entre as três iniciativas. Em (Hus07, Vin08), são apresentadas duas iniciativas que utilizam os recursos de programação genérica da linguagem CUDA para resolver problemas solucionáveis pelo algoritmo conhecido na teoria de grafos como algoritmo de empurrar e reclassificar("push-relabel algorithm"), fornecendo uma implementação do algoritmo de corte de grafo para modelos com dois rótulos.

Em uma iniciativa anterior, Yin et al.(Yin04) propuseram a diminuição do tamanho do grafo como medida de aceleração em uma abordagem sequencial. Com este objetivo, utilizam regiões uniformes da imagem como sendo o conjunto de elementos a serem classificados pela otimização, ao invés do regular uso do conjunto de pixels. Para obter as regiões, a imagem original é segmentada usando um algoritmo clássico de processamento de imagens, conhecido como algoritmo de "watershed", batizado em função de uma interpretação semântica na qual o algoritmo simula a elevação de águas em bacias hidrográficas (sua descrição pode ser encontrada em (Gon01)).

Em (Vas07), adotamos duas linhas de aceleração, ambas processadas em GPU, como formas de pré-processamento à construção do grafo. A Subseção 4.6.3 explora a computação dos termos da formulação geral da função de energia para minimização via corte de grafo, sob o ponto de vista da computação paralela, e a Seção 4.6.4 ilustra como os termos da aplicação de segmentação por luz ativa podem ser computados em GPU. Em uma segunda linha de aceleração, propomos a diminuição do tamanho do grafo pela reformulação do corte em grafo, para que seus nós representem as folhas de uma quadtree formada agrupando pixels semelhantes na imagem, conforme apresentado na Seção 4.7.

A motivação para particionar o conjunto de pixels da imagem em folhas de uma quadtree antes da construção do grafo é baseada no fato de esta estrutura fornecer um sistema de vizinhança amigável à recuperação de características como área e fronteiras entre folhas vizinhas, além de ser possível construir a quadtree (e a QuadN4tree) em GPU, conforme apresentado anteriormente.

4.6.1

Conceitos básicos de minimização de energia via Corte de Grafo

Em uma formulação geral, o objetivo de aplicar algoritmos de corte mínimo em grafos é encontrar uma função de classificação L, que associa a cada variável $p \in P$ de um conjunto de entrada a um rótulo $L_p \in L$, de forma que L minimize uma função de energia correspondente ao grafo (Boy04).

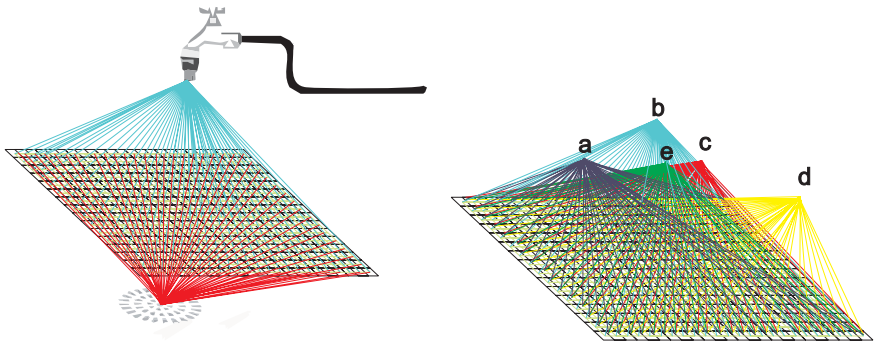
Normalmente, para as tarefas de visão computacional, o conjunto de variáveis é definido como o conjunto de pixels da imagem analisada na aplicação. Por este motivo e para facilitar a compreensão da aplicação da técnica de corte de grafo no contexto desta tese, as formalizações apresentadas nesta subseção consideram como sendo os pixels os elementos que se deseja classificar.

O número de possíveis valores assumidos pelas variáveis da função de energia é finito – motivo pelo qual essa técnica de minimização é considerada uma otimização discreta – e modelado como o conjunto de rótulos L, cada rótulo representando um possível valor de saída.

A forma geral da função de energia a ser otimizada é apresentada como (Boy04):

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{p,q \in N} V_{p,q}(L_p, L_q), \tag{4-3}$$

na qual $N \subset P \times P$ representa o sistema de vizinhança entre pixels para as aplicações sobre imagens, $D_p(L_p)$ é uma função que mede o custo de associar o rótulo L_p ao pixel p, enquanto $V_{p,q}$ mede o custo de associar os rótulos $\{L_p, L_q\}$ aos pixels adjacentes p e q, incluído na função para impor suavidade espacial.



4.12(a): Grafo s-t (fonte e sumidouro)

4.12(b): Grafo com múltiplos terminais

Figura 4.12: Grafos para o problema de classificação

Para minimizar a equação 4-3 pelo método de corte de grafo, um grafo é construído contendo um nó para representar cada pixel da imagem e nós adicionais, chamados nós terminais, um para representar cada rótulo do conjunto de classificações possíveis (exemplos na Figura 4.12). Dois tipos de arestas são criadas no grafo: as "n-links"e as "t-links". As "n-links"são

arestas incluídas no grafo para conectar nós representando pares de pixels vizinhos. Portanto, as "n-links"traduzem o sistema de vizinhança da imagem na estrutura do grafo. As "t-links"são arestas criadas para conectar nós representando pixels a cada um dos nós terminais, incluindo na estrutura construída uma simbolização das possibilidades iniciais de classificação de cada pixel.

Todas as arestas do grafo recebem um peso (ou custo) relacionado a um termo da função de energia. O peso de uma "t-link"corresponde à penalidade imposta à associação de um pixel ao rótulo correspondente à aresta. Tal custo expressa o termo da análise local do dado D_p na equação 4-3. O peso de uma "n-link"corresponde à penalidade relacionada à existência de descontinuidade de classificação entre os pixels vizinhos conectados por tal aresta. Tal custo expressa o termo de suavização entre pixels $V_{p,q}$ da equação 4-3.

Uma vez criado um grafo obedecendo tal descrição, seu corte mínimo expressa um mínimo da função de energia (equação 4-3), proporcionando uma rotulação ótima para os nós do grafo, ou seja, para os elementos que se deseja classificar (Boy04).

4.6.2 Segmentação por Iluminação Ativa

Nesta seção apresentamos a segmentação usando iluminação ativa, na qual uma fonte de luz de intensidade modulável e de posição fixa é usada para obtenção de um par de fotos do objeto que se deseja segmentar, conforme proposto em (Sa06).

A classificação dos pixels entre os rótulos de objeto ou de fundo é modelada por uma função de energia cujos parâmetros são calibrados pela análise de um indício inicial da segmentação. O indício inicial é obtido avaliando a diferença entre as duas imagens para produzir a semente da segmentação dos pixels entre objeto e fundo. Tal segmentação inicial permite ajustar os pesos de atribuição de cada rótulo, avaliando a distribuição das características nos dois grupos iniciais.

A função objetivo proposta em (Sa06) emprega como informações para caracterizar a segmentação a diferença de luminância entre as duas imagens de entrada e o histograma cromático da região inicialmente considerada objeto. O comportamento da diferença de luminância nos pixels de fundo, $p_B(p)$, é expresso assumindo uma distribuição Gaussiana com desvio padrão σ_L:

$$\mathrm{p}_B(p) = \frac{1}{\sqrt{2\pi}\sigma_L}\exp\left(\frac{-|L_{I_2}(p) - L_{I_1}(p)|^2}{2\sigma_L^2}\right), \tag{4-4}$$

A partir do cálculo desta distribuição (Figura 4.13 b), a semente da

segmentação é definida limiarizando a distribuição com um limiar t, produzindo uma classificação inicial de pixels como objeto por $O = \{p \mid \mathrm{p}_B(p) < t\}$.

Assim como na proposta de (Rot04), o histograma cromático dos pixels da área inicialmente considerada como objeto na semente de segmentação é usado para estimar a distribuição real do objeto. O histograma é levantado pela análise de frequências em apenas uma das imagens do par. Normalmente, a imagem escolhida é a adquirida com menor intensidade da fonte de luz ativa, por apresentar menores distorções cromáticas. Portanto, a distribuição esperada para o objeto é modelada como:

$$\mathrm{p}_O(p) = \frac{n_k}{n_O} \tag{4-5}$$

na qual n_k é o número de pixels associados à entrada k do histograma e n_O é o número total de pixels inicialmente considerado objeto.

Uma vez modeladas as distribuições esperadas para os pixels do objeto e do fundo, supondo a associação do valor um para o rótulo do objeto e zero para o fundo, o termo de análise de dados, ou termo regional, é modelado como:

$$R(x_p) = \begin{cases} -\log(\mathrm{p}_O(p)), & \text{if } x_p \text{ is 1} \\ -\log(\mathrm{p}_B(p)), & \text{if } x_p \text{ is 0} \end{cases} \tag{4-6}$$

Já a probabilidade de que pixels vizinhos recebam rótulos distintos é expressa em função de sua similaridade cromática como:

$$\mathrm{B}(p,q) = 1 \; - \; \exp\left(\frac{-(||Lab(p) - Lab(q)||)^2}{2\sigma_C^2}\right), \tag{4-7}$$

na qual $Lab(p)$ representa o valor do pixel no espaço Lab de representação de cor, e σ_C o desvio padrão esperado para a diferença de cor tomada em norma L2. A partir desta distribuição, para garantir a suavidade na clasificação de pixels vizinhos p, q, a penalidade imposta à sua classificação em rótulos x_p e x_q distintos é descrita como $-|x_p - x_q| \log \mathrm{B}(p,q)$.

Portanto, a função objetivo para descrever o problema de segmentação objeto/fundo a ser minimizada pela técnica de corte de grafo é expressa como:

$$E(\mathbf{X}) = \sum_{p \,\in\, I_1} R(x_p) - \sum_{p,q \,\in\, I_1} |x_p - x_q| \cdot \log \mathrm{B}(p,q), \tag{4-8}$$

4.6.3 Computação paralela dos termos da Função de Energia

Conforme apresentado na Seção 4.6.1, a construção do grafo representando uma determinada aplicação associa pesos a cada uma de suas arestas, representando os termos da função de energia. Por esse motivo, é preciso computar o valor do custo das arestas antes da construção efetiva da estrutura.

Sob o ponto de vista da computação paralela (conforme apresentado no Capítulo 2), tais computações da função de energia em pré-processamento à criação do grafo apresentam as características ideais para serem processadas em GPU, por serem aritmeticamente intensas e aplicadas a grandes volumes de dados, compreendendo todos os pixels e todos os pares de vizinhos da imagem.

Fundamentando a viabilidade de computar a função de energia em GPU, observamos que a distinção entre as arestas "n-links"e as "t-links"permite pensarmos na computação da função de energia em termos de fluxos distintos de elementos independentes. Por causa de tal distinção, é possível isolar as computações dos termos da função de energia relativos à esses dois tipos de arestas.

A independência da computação sobre elementos de um fluxo se dá em relação ao elemento representando o nó do grafo, ou seja, no contexto adotado, o fluxo de elementos é formado pelos pixels da imagem que podem ter suas diferentes características avaliadas de forma independente e paralela a outros pixels da imagem.

Uma outra separação possível se dá em relação aos rótulos da aplicação, que podem ter sua avaliação isolada em diferentes etapas de processamento.

4.6.4 Paralelizando o cálculo da Função de Energia da Segmentação por Luz Ativa

Em (Vas07) os termos da segmentação objeto/fundo por luz ativa são processados na GPU, exemplificando os conceitos de aceleração propostos para o pré-processamento dos termos da função de energia em minimizações via corte de grafo.

A partir dos recursos da programação de "shaders", foram propostas computações eficientes para acelerar o cálculo tanto do termo regional, quanto do termo de suavização.

A função de energia adotada analisa os valores cromáticos dos pixels em um sistema cromático diferente do sistema no qual a imagem é obtida. A conversão entre os espaços cromáticos RGB e CIE Lab é feita por uma adaptação do padrão de programação paralela de mapeamento, de forma a gerar um fluxo contendo tal conversão, uma vez que cada pixel da imagem de entrada com valor RGB é processado de maneira isolada para produzir um pixel na mesma posição espacial na imagem de saída, com valor CIE Lab. O fluxo resultante armazenando os valores para os canais a e b, a serem utilizados na função de energia, é ilustrado na Figura 4.13(a) com seus valores armazenados, respectivamente, nos canais R e G dessa figura.

4.13(a): Canais a e b do espaço de cores *Lab CIE*

4.13(b): probabilidade de fundo

4.13(c): Canais RGBA armazenando respectivamente *a, b*, semente de segmentação e probabilidade de fundo

Figura 4.13: Textura armazenando parcelas da função de energia.

O algoritmo proposto distribui as computações que avaliam rótulos distintos em diferentes passadas, computando a probabilidade de fundo (equação 4-4) como uma etapa em separado. Esta etapa de processamento avalia a distribuição da diferença de luminância no par de imagens de entrada, gerando um fluxo de saída contendo tal avaliação (ilustrada na Figura 4.13 b). A computação desta etapa também é realizada como uma variação do padrão de mapeamento, isto porque são consultadas as luminâncias dos pixels nas duas imagens de entrada em uma determinada posição espacial para produzir um pixel na mesma posição na imagem de saída.

Na equação (4-4) os valores de $1/\sqrt{2\pi}\sigma_L$ e $1/(2\sigma_L^2)$ podem ser identificados como valores que permanecem constantes para um valor de σ_L fixo. Para otimizar a computação da probabilidade de fundo, tais valores são pré-computados e passados como constantes ao processamento, evitando repetir desnecessariamente sua computação a cada elemento.

Para produzir a semente de segmentação, em uma etapa seguinte, a

imagem produzida na análise do fundo é processada por outra adaptação do padrão de mapeamento, codificando um processo de limiarização que produz um novo fluxo representando a semente da segmentação.

Outra etapa de processamento paralelo é responsável pela avaliação do rótulo de objeto pela computação da função de distribuição do objeto (equação 4-5). Seu resultado expressa o histograma dos canais a e b dos pixels considerados objetos pela semente de segmentação em uma das imagens de entrada convertida ao espaço Lab.

Com esta finalidade, propomos adaptar a proposta de Scheuermann e Hensley (Sch07) para obter o histograma bidimensional desejado. Em sua formulação original, o levantamento de um histograma 1D é criado distribuindo um conjunto de primitivas pontuais, uma sobre cada posição de pixel na imagem. A renderização desse conjunto de primitivas utiliza um "vertex shader"que analisa o conteúdo da imagem em sua posição original, reposicionando o ponto para a posição correspondente no histograma. Este algoritmo conta com o recurso de "blending"do hardware gráfico para acumular os pontos direcionados à mesma posição de saída no histograma. A adaptação sugerida em (Vas07) altera o posicionamento proposto ao "vertex shader"para produzir um histograma 2D, com suas dimensões representando, respectivamente, as frequências nos canais a e b avaliadas.

Portanto, os diferentes termos e rótulos são avaliados de maneira independente e paralela sobre o fluxo de pixels, produzindo diferentes resultados intermediários para a função de energia proposta.

4.7 Quadtrees e Cortes em Grafos

Ao formular tarefas de visão computacional como problemas de minimização de energia, diferentes características dos pixels da imagem podem ser avaliadas para modelar as propriedades da tarefa, tais como luminância, cor, gradiente, entre outras, assim como, diferentes métricas sobre tais características. Entretanto, qualquer que sejam as características ou métricas adotadas na função de energia, é comum que imagens naturais possuam áreas de pixels apresentando valores similares em relação a elas.

Nesta Seção apresentamos a abordagem proposta em (Vas07), na qual assumimos que o critério de similaridade entre os pixels varia de acordo com a aplicação. Para qualquer que seja o critério adotado, assume-se também que os pixels considerados semelhantes devam receber o mesmo rótulo ao fim do processo de minimização de energia. Esta suposição é usada para sustentar o agrupamento espacialmente coerente de pixels similares em áreas uniformes,

com o intuito de diminuir o tamanho do conjunto de elementos sobre o qual a minimização é formulada, influenciando de forma direta na diminuição do tamanho do grafo sobre o qual o algoritmo de corte mínimo é aplicado.

Outra questão levantada em (Vas07), é o fato de que, se por um lado, o agrupamento de pixels reduz o tamanho do grafo, por outro lado, pode vir a gerar uma estrutura espacial cujas relações topológicas, especialmente as de adjacência entre seus elementos, demandem para serem recuperadas operações mais custosas que o habitual sistema de vizinhança entre pixels, quer seja definido com grau de conectividade 4 ou 8. Esta observação ressalta que, ao agrupar os pixels em regiões, a estrutura criada por tal partição pode vir a exigir um aumento de complexidade no procedimento de criação do grafo, vindo a reduzir a vantagem concebida pela diminuição de seu tamanho.

Com base nas observações expostas, em (Vas07), propomos a diminuição do conjunto de elementos a serem classificados pela otimização, a qual é obtida representando a imagem analisada em uma estrutura de quadtree regional. As variáveis da otimização passam a representar as folhas da quadtree, obtidas como regiões atendendo a um critério de similaridade e, ao mesmo tempo, às regras de subdivisão espacial da quadtree.

Diversas vantagens motivam a adoção da estrutura da quadtree pela proposta, destacando o fato de a quadtree fornecer um sistema regular de vizinhança entre suas folhas e serem conhecidos algoritmos para recuperar esse sistema. Colaborando em sua defesa, argumentamos que é possível construir eficientemente tal estrutura em GPU e ainda utilizar a variação proposta denominada QuadN4tree para oferecer um suporte à recuperação de folhas vizinhas de forma ótima, ao custo de uma leitura de referência por folha vizinha (conforme apresentado da Seção 4.3 à Seção 4.5).

A seguir descrevemos como o grafo para minimização de energia é construído para representar as folhas da quadtree em seus nós. A minimização do grafo criado com esta otimização foi denominada QuadCut na publicação (Vas07), incluída no anexo D desta tese.

4.7.1
QuadCut: Função de Energia sobre as folhas da Quadtree

Na formulação proposta como QuadCut o grafo da otimização é criado com seus nós representando as folhas da quadtree a serem classificadas entre os rótulos pré-definidos. Arestas são incluídas no grafo para o QuadCut, de forma que cada folha seja conectada a cada rótulo por uma aresta "t-link"e que cada folha seja conectada às folhas suas vizinhas por arestas "n-link"(Figura 4.14).

Considerando que os pixels são agrupados por um critério de similari-

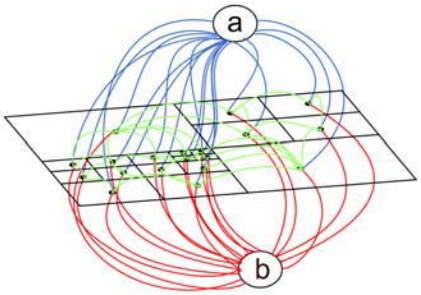


Figura 4.14: Grafo sobre as folhas de uma quadtree

dade, uma folha representando um agrupamento recebe um valor associado, descrevendo a característica de homogeneidade dos pixels e são consultados na avaliação da função de energia.

Revendo a formulação original da classificação por corte de grafos, o objetivo no processo de QuadCut é encontrar uma classificação L, que associa um rótulo $L_t \in L$ para cada folha $t \in T$ da quadtree de forma a minimizar uma função de energia adotada.

Supomos em um modelo geral que o conjunto de rótulos para o processamento de QuadCut seja o mesmo adotado em uma formulação sobre os pixels da imagem. Entretanto, faz-se necessária a reformulação do formato geral da função de energia apresentada na equação 4-3 com o objetivo de equilibrar a métrica para a topologia das folhas de uma quadtree. Uma tentativa inicial para a forma geral da função de energia tendo as folhas da quadtree como variáveis da classificação é expressa como:

$$E(L) = \sum_{t \in T} \alpha * D_t(L_t) + \sum_{t,u \in N} \beta * V_{t,u}(L_t, L_u), \tag{4-9}$$

na qual $N \subset T \times T$ representa o sistema de vizinhança das folhas de uma quadtree; $D_t(L_t)$, a função que avalia o custo da associação do rótulo L_t a folha t; $V_{t,u}$ avalia o custo de associar os rótulos $\{L_t, L_u\}$ às folhas adjacentes t e u e α e β são pesos inseridos para evitar a distorção da função de energia em relação à modelagem sobre pixels.

Para fundamentar o equilíbrio da reformulação proposta, recorremos a duas propriedades principais derivadas das regras de subdivisão espacial da quadtree. Uma primeira propriedade utilizada estabelece que a área e extensão da borda de uma folha, em número de pixels, podem ser computadas pela altura da folha na árvore. A segunda propriedade recorrida estabelece que considerando duas folhas vizinhas quaisquer, a inclusão da borda da folha menor no intervalo definido pela folha maior é garantida e se forem vizinhas situadas em um mesmo nível da árvore é assegurado que possuem uma borda em comum.

A formulação geral para a função de energia a ser aplicada às folhas de uma quadtree parte de tais propriedades para analisar o comportamento da expansão dos dois somatórios da equação 4-3. Ao expandir o somatório relativo ao termo de avaliação dos dados é esperado que cada parcela $D_p(p)$ correspondente a pixels similares apresentem valores semelhantes, portanto, em cada folha são agrupadas tantas parcelas do termo de avaliação dos dados, quantos forem os pixels agrupados pela folha.

Por outro lado, ao observar a expansão do termo de suavização $V_{p,q}(L_p, L_q)$, apenas os termos sobre pixels espalhados na borda da folha devem ser repassados à formulação baseada nas folhas da quadtree. Os pixels no interior da folha não fazem fronteira com pixels possíveis de receber classificações distintas, uma vez que a folha receberá classificação única. Por esse motivo, a expansão do segundo termo da formulação geral é repassada à formulação baseada na quadtree como o conjunto de parcelas de penalidades para pixels nas bordas de cada folha.

O número de pixels no interior de uma folha t pode ser obtido por $(2^{nivelFolha(t)})^2$, enquanto o número de pixels na borda entre duas folhas vizinhas t e u, como $2^{\min(nivelFolha(t),nivel(u))}$. Portanto, podemos reescrever a equação 4-9 como:

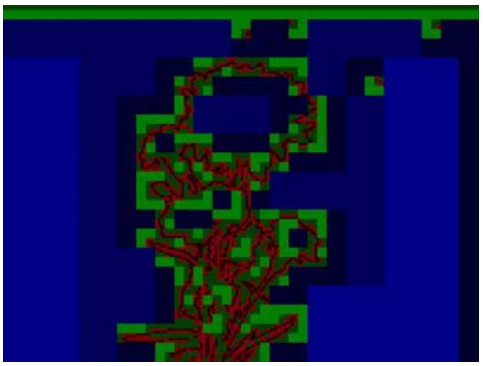
$$E(L) = \sum_{t \in T} (2^{nivel(t)})^2 * D_t(L_t) + \sum_{t,u \in N} 2^{\min(nivel(t),nivel(u))} * V_{t,u}(L_t, L_u) \quad (4\text{-}10)$$

assumindo o peso α para o termo regional como a área da folha e o peso β para o termo de suavização, como o tamanho em pixels da fronteira em comum entre duas folhas vizinhas.

Para exemplificar a aplicação do QuadCut, a Figura 4.7.1 apresenta o resultado da partição da imagem em uma Quadtree de regiões, na qual cada folha recebe uma coloração para destacar seu nível na árvore e o resultado final obtido pela otimização de segmentação objeto/fundo por luz ativa.

A partir da formulação geral proposta na equação 4-10, além da aplicação para segmentação por luz ativa ilustrada em (Vas07), a otimização proposta como QuadCut pode ser trivialmente aplicada a outras abordagens de segmentação, tais como as propostas em (Rot04, Yin04, Wan05) e, ainda, a algoritmos de coloração e restauração de imagens propostos respectivamente em (Yun06) e (Boy98, Raj05).

4.15(a): Imagem Original



4.15(b): Subdivisão em Quadtree de regiões

4.15(c): Resultado da segmentação objeto/fundo

4.15(d): Resultado da Composição de imagem

Figura 4.15: QuadCut: resultados mantém detalhes finos ao mesmo tempo que a otimização opera em conjuto reduzido de variáveis