LOOP SNAKES - SNAKES COM CONTROLE TOPOLÓGICO OTIMIZADO

Saulo Pereira Ribeiro

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Antonio Alberto Fernandes de Oliveira, D.Sc.

Prof. Ricardo Cordeiro de Farias, Ph.D.

Prof. Gilson Giraldi, D.Sc.

Prof. Luiz Henrique de Figueiredo, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 2005

PEREIRA RIBEIRO, SAULO

Loop Snakes - Snakes com Controle Topológico Otimizado [Rio de Janeiro] 2005

XIV, 78 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Segmentação de imagens. 2 - Contornos ativos. 3 - Snakes topologicamente adaptáveis.

I. COPPE/UFRJ II. Título (série)

*A minha família,*
*pelo amor, carinho e dedicação.*

# Agradecimentos

Sei que essas palavras dirão apenas uma parte do que realmente gostaria de expressar. Quero primeiramente agradecer ao meu orientador, o prof. Antônio Oliveira pela paciência, confiança e por ter me dado a oportunidade de conhecer mais profundamente a área de processamento de imagens. Lógico que a confiança transmitida por ele sempre foi sentida por todos nós, afinal quém é que nos salva nos apertos, tirando as dúvidas que não encontramos nos livros e nem na internet?

Quero agradecer ao prof. Claudio Esperança pelos puxões de orelha, pelo direção dada em relação a pesquisa, pelas inesquecíveis aulas de geometria computacional!!!

Ao prof. Paulo Roma pelo grande apoio, ou seja manter nossas máquinas, e redes funcionando, a fim de que possamos nos dedicar inteiramente a pesquisa.

Ao prof. Gilson Giraldi pelo incentivo e grande apoio, tendo a oportunidade de dar continuidade a sua tese de doutorado. Ao prof. Luiz Henrique por ter aceito o convite e pelo tempo dedicado a esse trabalho.

Ao prof. Ricardo Farias, amigão entusiasta e cheio de idéias novas.

Ao prof. Edilberto Strauss, pelo compartilhamento de seu conhecimento na área de imagens médicas.

Bom não posso esquecer do turma do LCG: Alvaro, Disney, Cesar, Ricardo, Alopes, Okamoto, Fabio, Morante, Meg, Mara, Aruquia, Marcelo, Barrére, Vitor, Aline, Daniel, Karl, Yalmar, Leandro, Roque, Caique, André, Henrique, Max, Leonardo. A todos vocês que de certa forma colaboraram para a realização deste trabalho, mas também pelo carinho, amizade e lutas lado a lado. Acho que ninguém irá esquecer as aulas de geometria computacional.

Aos amigos feitos: Erik, Gina, Kleber, Mariela, Manoel, Michel, Marisa, Felipe, Alberto, Ana, Ítalo, Genilce, André, Vilany.

Ao PESC, professores e pessoal técnico-administrativo. Sempre a postos a nos ajudar.

A CAPES pelo apoio com a bolsa, sendo de grande importância para a realização deste trabalho.

A minha famíla:Alberto, Célia e Nádia. Pelo apoio, amor, participação e por estarem sempre comigo em todos os momentos.

A minha grande amiga Neide pelo apoio, amor e paciência.

A Deus, sobretudo, por sempre me mostrar que é possível ir sempre além.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc)

LOOP SNAKES - SNAKES COM CONTROLE TOPOLÓGICO OTIMIZADO

Saulo Pereira Ribeiro

Fevereiro/2005

Orientador: Antonio Alberto Fernandes de Oliveira

Programa: Engenharia de Sistemas e Computação

Snakes topologicamente adaptáveis, ou simplesmente T-snakes, são uma ferramenta padrão para identificar automaticamente múltiplos segmentos em uma imagem. Este trabalho introduz uma nova abordagem para se controlar a topologia de uma T-snake. Ele foca, especialmente, os loops formados pela chamada curva projetada, a qual é obtida a cada estágio da evolução da snake. A idéia é fazer com que esta curva seja a imagem de um mapeamento linear por partes de uma classe adequada. Então, com a ajuda de uma estrutura adicional — a Loop-Tree — é possível decidir em tempo O(1) se a região delimitada por um desses loops já foi explorada ou não pela snake. Isto torna possível construir um algoritmo ótimo para implementar o processo de evolução de uma T-snake, cuja performance é demonstrada, também, no trabalho, por meio de estatísticas e de uma série de exemplos.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

LOOP SNAKES - SNAKES WITH ENHANCED TOPOLOGY CONTROL

Saulo Pereira Ribeiro

February/2005

Advisor: Antonio Alberto Fernandes de Oliveira

Department: Computing and Systems Engineering

Topologically adaptable snakes, or simply T-snakes, are a standard tool for automatically identifying multiple segments in an image. This work introduces a novel approach for controlling the topology of a T-snake. It focuses on the loops formed by the so-called projected curve which is obtained at every stage of the snake evolution. The idea is to make that curve the image of a piecewise linear mapping of an adequate class. Then, with the help of an additional structure—the Loop-Tree—it is possible to decide in O(1) time whether the region enclosed by each loop has already been explored by the snake. This makes it possible to construct an enhanced algorithm for evolving T-snakes whose performance is assessed by means of statistics and examples.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

O uso dos **Modelos de Contornos Ativos**, usualmente chamado de **Snakes**, tornou-se uma técnica padrão para segmentar uma imagem[23]. Eles têm a apreciável propriedade de produzir curvas poligonais fechadas em qualquer iteração do processo enquanto que no enfoque clássico[23], essas curvas são produzidas somente como o resultado final, depois de uma seqüência de diferentes operações. Em comparação com outros esquemas dinâmicos também tentados para identificar contornos, eles têm a vantagem de considerar somente a interação entre snaxels cujos índices, na seqüência que define a snake, são próximos. Não existe a necessidade de se verificar explicitamente se existem outros snaxels numa vizinhança de cada um deles como em enfoques bem conhecidos que usam um sistema de partículas para achar os contornos. A interação entre snaxels serve tanto para suavizar as curvas obtidas em relação às que seriam produzidas se fosse considerado somente o campo externo, como também para manter a snake se movendo onde esse campo é nulo.

Centenas de trabalhos sobre o modelo de snakes comuns foram produzidos desde sua introdução em [23]. O modelo de snakes já foi adaptado ao contexto de diversas aplicações específicas como tracking[5], detecção de faces e reconhecimento de objetos e já foi usado em combinação com técnicas de diferentes áreas como: morfologia matemática, estimação estatística, splines e até mesmo redes neurais. Várias alternativas já foram também propostas para contornar as dificuldades típicas do modelo tais como a dificuldade de se escolher parâmetros adequados, ou evitar auto-interseções ou ainda, o fato de uma snake poder ficar retida num mínimo local do funcional de energia utilizado. O capítulo 2 deste trabalho é dedicado à apresentação e considerações sobre o modelo original de snakes.

Uma das mais óbvias limitações das snakes comuns, no entanto, é o fato de que para

cada contorno que precisa ser encontrado, uma snake diferente tem de ser inicializada pelo usuário. Não é incomum ter mais do que um contorno alvo e nesse caso o usuário deve criar uma snake para cada um deles. Mais ainda, em algumas aplicações, todos os segmentos que têm certas propriedades devem ser achados, ou simplesmente contados, e o número deles é tão grande que torna este enfoque manual totalmente inapropriado. Exemplos: identificar estruturas celulares de um dado tipo em imagens microscópicas, todos os vasos sangüíneos em um angiograma ou os componentes na imagem de um circuito eletrônico.

Essa limitação foi superada com a introdução em [21], das **T-snakes**, forma resumida de *snakes topologicamente adaptáveis*. Essas snakes têm a habilidade de mudar sua topologia, seja por subdivisão ou por agregação, permitindo que cada contorno do segmento possa ser aproximado por exatamente uma snake exatamente ou por uma snake interna e outra externa [13]. Em circunstâncias ideais as T-snakes funcionam de acordo com um dos seguintes esquemas:

**A)** Inicialmente toda a imagem é envolvida por uma snake fechada. Durante o processo essa snake é continuamente contraída e eventualmente dividida em partes. Cada uma dessas partes é considerada uma nova snake e submetida ao mesmo processo. Quando todas as snakes geradas se tiverem ajustado a um contorno ou se tornado muito pequenas o processo pára;

**B)** De uma maneira dual [12], uma série de snakes muito pequenas - *as snakes sementes* - são aleatoriamente espalhadas pela imagem. Elas são, então, continuamente expandidas e quando duas delas colidem elas se fundem em uma única snake. Um contorno que tenha snakes sementes em seu interior será aproximado por uma T-snake interior a ele. Snakes sementes localizadas no background dão origem a snakes que aproximam os contornos pelo lado de fora e também, mediante sucessivas fusões, a uma snake longa que acaba se ajustando a borda da imagem;

**C)** Snakes que se contraem e se expandem são empregadas simultaneamente de modo que cada contorno é aproximado por uma snake que se contrai vindo do exterior e uma que se expande proveniente de seu interior. Cada uma delas é usada como um limitante para a evolução da outra. Esta aproximação dupla confere uma maior credibilidade à detecção do contorno;

A estrutura de uma T-snake [21] é a de uma snake comum mais uma estrutura $2D$ — *a Estrutura Auxiliar — EA* — que contém a informação requerida por um dos processos descritos anteriormente, que não está disponível de uma forma imediata, numa lista contendo as coordeandas dos snaxels. Objetivamente: a informação na *EA* permite relacionar diretamente snaxels próximos em $2D$, mas longe um do outro se tivermos que caminhar ao longo do contorno da snake. Essa informação diz respeito a como fazer o processo de evolução da snake ter as três seguintes propriedades:

i. **monotonicidade** - a evolução de uma snake deve ser monótona — ou sempre se contraindo ou sempre se expandindo — para assegurar a convergência do processo;

ii. **completude** - a única razão para que nenhuma snake explore uma região da imagem é a existência de uma barreira de campo externo que as impeça de penetrar aa região. Assim, sob condições ideais uma snake que se contrai deve explorar todo o background e uma que se expande dentro de um segmento, todo o segmento;

iii. **simplicidade** - todas as snakes geradas durante o processo devem ser linhas poligonais simples.

A estrutura adicional descrita acima, consiste de uma matriz cujos elementos estão associados aos vértices [21], arestas [4] ou células [3], de uma malha quadrada cobrindo o domínio da imagem.

Como a motivação original deste enfoque que será introduzido aqui foi contornar algumas dificuldades bem conhecidas do enfoque original das T-snakes dado em McInerney e Tezopoulos [21], esse enfoque será descrito em detalhe no capítulo 3.

Em linhas gerais a abordagem original considera que as T-snakes de uma iteração $k$ se movem continuamente até coincidir com a chamada **Curva Fisicamente Transformada** — $TC_k$ — dessa iteração. Essa curva é a determinada pelas novas posições dos snaxels após terem sido deslocados pela ação do sistema de forças que atua sobre eles em suas posições na T-snake inicial da iteração. Assumindo-se esse movimento contínuo as T-snakes varrem — ou queimam ou visitam, que nesse contexto são usados como sinônimos — em cada iteração, toda uma faixa de pontos. Chame de **Conjunto Visitado** ao conjunto de pontos varridos dessa maneira em todas as iterações até a atual. Para assegurar que a evolução da T-snake terá as três propriedades indicadas acima basta tomar como uma T-snake da iteração seguinte $(k + 1)$ cada componente conexa do contorno do Conjunto Visitado.

Obter essas componentes a cada iteração, entretanto, pode ser computacionalmente custoso se os snaxels puderem ser submetidos a um movimento genérico. É mais simples manter uma representação do Conjunto Visitado, a partir da qual, uma aproximação de sua borda pode ser obtida com facilidade. É essa representação que se encontra armazenada na estrutura auxiliar usada no trabalho de McInerney [21]. Explicitamente, ela contém uma amostragem da **função característica** do conjunto visitado $X_v$ realizada nos vértices da malha ($M$) utilizada. O método identifica as arestas da malha cujos vértices tem valores de **(**$X_v$**)** diferentes e posiciona um snaxel da T-snake da próxima iteração em um ponto de cada uma dessas arestas. Se a evolução da snake é monótona esses pontos podem ser escolhidos entre os vértices da chamada **Curva Projetada** — $PC_k$ — que é a poligonal definida pelas intersecções de $TC_k$ com as arestas da malha.

A abordagem proposta aqui enfoca, especificamente, os loops formados pela curva projetada. Como nessa metodologia, cada snake a ser desenvolvida no próximo estágio — $k+1$ — está relacionada a um desses loops, elas são chamadas de **Loop-Snakes** [3]. A curva projetada de um estágio $k$ é considerada como a imagem de uma versão dilatada da snake $S_k$, através de um mapeamento linear por partes, $\Gamma_k$.

Ao se definir $\Gamma_k$ numa versão dilatada de $S_k$ e não na própria snake, torna-se mais fácil satisfazer algumas condições locais que são necessárias para tornar a estratégia computacionalmente atrativa. No capítulo 4, a seção 4.1 descreve essas condições e a seção 4.2 é dedicada a mostrar quão simples e direto é fazer com que essas condições valham embora explicar porque isso é verdade demande um racicínio mais elaborado.

Alguns loops de $PC_k$, chamados de **abertos** ou **inexplorados**, delimitam regiões que ainda serão exploradas pela snake enquanto que outros, os **fechados** ou **explorados**, delimitam regiões que já foram duplamente exploradas. O atributo de ser aberto ou fechado será referido neste trabalho como o rótulo do loop. Os resultados do capítulo 5 demonstram que é possível determinar o rótulo de um novo loop simplesmente verificando o rótulo dos loops adjacentes a ele que foram achados anteriormente. Mesmo quando não existem loops adjacentes anteriores, eles podem ser rotulados em tempo $O(1)$.

Os procedimentos empregados pela metodologia são bastante simples até que a curva $PC_k$ retorna para uma célula $C$ já visitada. O capítulo 6 mostra como ela procede nesse último caso. Existem duas possibilidades para uma célula $C$ que é re-visitada por $PC_k$:

**i.** $C$ se torna uma **célula dupla**, isto é, uma que contém um vértice de $PC_k$ em cada uma de suas arestas;

**ii.** Uma mudança topológica envolvendo as arestas de $PC_k$ contidas em $C$ deve ser realizada.

Essas mudanças topológicas são executadas em duas situações diferentes:

**(A)** $PC_k$ tem uma auto-intersecção ou *Knot* em $C$ que se mantém ainda que $PC_k$ seja substituido por outra curva cortando as mesmas arestas da malha;

**(B)** $PC_k$ revisita uma aresta $e$ de $C$.

Considera-se que um novo loop é formado no caso *B* porque pontos numa mesma aresta da malha são assumidas serem indistingüíveis. Esta é uma suposição razoável se o tamanho de uma aresta da malha for a própria precisão requerida na determinação dos contornos. A concentração de todos os pontos de $PC_k$ que estão sobre uma aresta num só também faz com que uma faixa estreita, cujo contorno, contido em $PC_k$, cruza duas vezes uma seqüência de arestas da malha seja transformada em uma linha que pode então ser um *whisker* de um loop ou uma ligação entre dois loops, situação que é conhecida como um gargalo. Todos esses casos devem ser identificados e tratados adequadamente. A questão é que a situação que gera este processamento mais elaborado, isto é, o fato de $PC_k$ revisitar uma célula, ocorre com uma freqüência muito pequena em relação ao enorme número de snaxels gerados, como é mostrado pelas estatísticas no capítulo 7. Estes números atestam a adequação da estratégia "lazy" empregada aqui: fazer o mínimo necessário quando processar um snaxel comum e transferir toda a complicação para o último momento, isto é, para quando um loop está em vias de ser formado. Como o número de loops é, usualmente, centenas de vezes menos do que o número de snaxels, o processo resulta mais rápido.

As estatíticas obtidas foram extraídas de inúmeros exemplos de aplicação de um sistema que implementa a metodologia das Loop-snakes a problemas de segmentação em imagens de diferentes tipos. Os exemplos mais representativos e a interface usada por esse sistema são também apresentados no capítulo 7.

O capítulo 8 é dedicado as conclusões e trabalhos futuros. Parte dela se refere à possibilidade de estender para 3D o enfoque das Loop-snakes o que não parece ser algo natural, já que ele usa a ordem em que os vértices são alcançados, quando se percorre uma curva. Um ponto importante a destacar é que para limitar o seu escopo — dado que mesmo com essa restrição já se tem uma quantidade bastante palpável de resultados– este trabalho trata somente do caso onde as snakes estão sempre se contraindo. Entretanto,

estender o método para o caso em que as snakes estão se expandindo também, não é absolutamente uma dificuldade insuperável.

Além disso devemos inidicar que as demonstrações dos lemas apresentados não foram incluídas por se considerar que isso ultrapassa em muito o contexto de uma tese de mestrado.

As principais contribuições deste trabalho podem ser vistas em detalhes na seção 3.6 na qual se explica as razões que geraram as características deste novo enfoque, que reduz o custo computacinal de cada iteração do método da T-snake. O objetivo de redução do custo é motivado pelo fato de que milhares ou milhões de snaxels podem ser gerados durante a evolução completa de uma T-snake. Assim objetivando-se fazer pequenas economias, porém elementares, obtidas através de número grande de vezes pode-se obter uma implementação de T-snake com um melhor desempenho computacional. Desta forma se estabeleceu um conjunto de objetivos a serem cumpridos a fim de se reduzir este custo:

**1)** Não verificar se um ponto já foi visitado.

**2)** Não usar triangulações.

**3)** Nenhum histórico deve ser necessário.

**4)** Percorrer as curvas de um determinado estágio uma única vez.

**5)** Determinar cada nova snake tão logo quanto possível.

O cumprimento destes objetivos foi obtido através da implementação do método de Loop Snakes e pode-se ver nas duas tabelas apresentadas no capítulo 7 os resultados que atestam a funcionalidade deste enfoque.

# Capítulo 2

# Fundamentos dos Modelos Deformáveis

Os fundamentos matemáticos dos modelos deformáveis representam a confluência da geometria, física e teoria de aproximação. A geometria serve para representar a forma do objeto, a física impõe restrições sobre como a forma pode variar no espaço e no tempo, e a teoria de aproximação ótima provê uma generalização para os mecanismos para ajustar os modelos aos dados medidos.

A geometria do modelo deformável geralmente pode ser aproximada empregando-se representações geométricas que envolvem muitos graus de liberdade, tais como as splines. O modelo se mantém tratável, entretanto, devido ao fato de que suas variáveis não poderem evoluir independentemente, mas sim governadas através de princípios físicos que dão um comportamento que sob o ponto de vista geométrico pode ser descrito de forma intuitiva. O nome "modelos deformáveis" deriva do uso da teoria da elasticidade a um nível físico, geralmente dentro do contexto de dinâmica Lagrangeana. Do ponto de vista físico, para computar a sua evolução, os modelos deformáveis são corpos elásticos que respondem naturalmente às forças e restrições aplicadas. Tipicamente, funções representando um tipo de energia (acumulada em conseqüência da deformação) e que dependem de variáveis de natureza geométrica, são associadas aos modelos deformáveis. A energia cresce monotonicamente na medida que a forma do modelo se distancia de uma especificação natural ou "estado de repouso" e geralmente sua expressão inclui termos que estabelecem um certo padrão de suavidade ou simetria do modelo. No contexto Lagrangeano, a deformação da energia é expressa pela existência de forças elásticas internas ao modelo. Tendo em vista expressar em termos físicos a teoria de otimização clássica funções representando uma energia potencial externa são definidas em termos dos dados de interesse ao qual o modelo deve se ajustar. Estas energias potenciais fazem surgir as forças externas as quais deformam o modelo de tal forma que este ajuste maximamente

aos dados. Curvas, superfícies, e modelos sólidos deformáveis ganharam popularidade depois que eles foram propostos para o uso em visão computacional [9] e na computação gráfica [29] em meados dos anos 80. Na visão computacional, modelos de curvas e superfícies deformáveis foram propostos como soluções para os problemas inversos mal formulados tais como detecção de arestas e reconstrução de superfície. Terzopoulos introduziu a teoria de modelos deformáveis contínuos (multidimensionais) cuja evolução dinâmica é determinada por uma energia de deformação os quais são constituídos por curvas splines generalizadas (continuidade controlada) [28]. A spline de continuidade controlada pode formalmente ser considerada como regularização [30] de problemas mal formulados, reescrevendo-os como problemas de minimização de um funcional bem definido. O modelo deformável que mais atraiu a atenção é popularmente conhecido como "snakes" [28]. Snakes são contornos deformáveis planares que são úteis em várias tarefas de análise de uma imagem. Elas geralmente são usadas para aproximar as bordas dos segmentos de uma imagem, baseadas no fato de que essas bordas são pedaços contínuos ou suaves por partes. Na sua forma básica, a formulação matemática de snakes vem da teoria de otimização expressa empregando funcionais de energia [31].

## 2.1 Introdução aos contornos ativos

Desde a sua introdução por Kass, Witkin e Terzopoulos em 1988 [23], os Modelos de Contornos Ativos (snakes) vêm se estabelecendo como um importante recurso para a solução de problemas em processamento de imagens digitais, particularmente para segmentação de imagens e rastreamento de objetos em seqüências de vídeo.

Segmentar uma imagem significa separá-la em regiões com propriedades comuns (cor, textura, etc.) que, idealmente, correspondem ao fundo da cena, a objetos ou a partes conexas deles. A segmentação é um passo fundamental para a identificação das características relevantes em uma imagem e sua análise (figura 2.1).

Os Modelos de Contornos Ativos, na sua formulação original fazem uso de informações sobre um trecho localizado de um contorno, podendo incorporar também informações obtidas a priori a partir de uma imagem como um todo ou sobre os segmentos procurados. O uso de Modelos Ativos para a extração das bordas dos objetos de uma cena, são em geral aplicados conjuntamente com técnicas de filtragem usadas na detecção de pontos de bordas. Estes métodos se iniciam em uma configuração mais ou menos arbitrária, um contorno inicial, que evolui até contornar o objeto de interesse (figura 2.2).

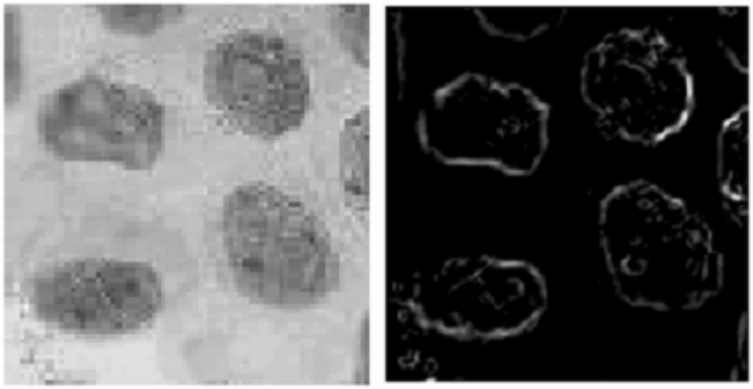

Figura 2.1: Exemplo de segmentação.

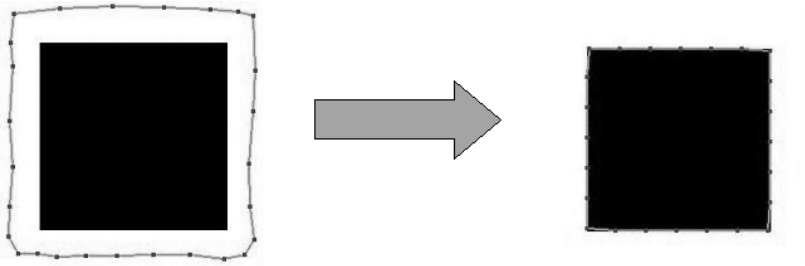

Figura 2.2: Curva paramétrica fechada.

Este comportamento dinâmico faz com que sejam também conhecidos como “modelos deformáveis”. A evolução dos modelos deformáveis a partir da curva inicial é controlada pela ação de “forças internas” (intrínsecas à geometria da curva) e externas (derivadas da imagem) atingindo o equilíbrio sobre a fronteira do objeto [2].

## 2.2 Snakes

As snakes são um tipo de modelo deformável no qual uma curva poligonal ou mais geralmente, spline, é deformada pela ação de forças internas e externas. Essa deformação provoca um deslocamento da Snake em direção à borda de interesse [23] , [5]. O objetivo do processo é fazer com que a energia associada a Snake atinja um posição de equilíbrio apenas quando o modelo aproximar a borda de um segmento desejado. No contexto, das imagens em nível de cinza uma borda é caracterizada pela expressão

$$\|\nabla I^2\| > T, \tag{2.1}$$

onde $T$ é um limiar previamente estabelecido usando-se, por exemplo, um histograma da intensidade do campo gradiente na imagem.

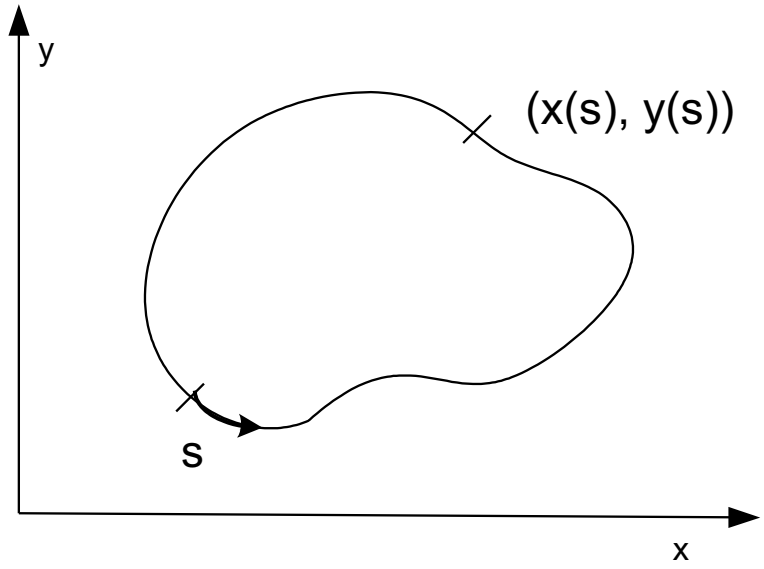


Figura 2.3: Curva paramétrica fechada.

### 2.2.1 Modelo Original de Snake

A snake é a forma mais simples dos contornos ativos. Sua energia será composta por um termo diretamente relacionado à imagem que guarda as características de interesse da mesma, além de um termo de regularização ou de suavização[30], [27].Este último, além de garantir estabilidade numérica irá guardar características geométricas supostas a priori para a solução.

O primeiro passo para a definição precisa da energia é definir o espaço de curvas utilizado[7]. Seja então $Sc$ o espaço de curvas parametrizadas por um parâmetro $s$, $c(s) = (x(s), y(s))$, tal que:

$$Sc = \left\{ c : [0,1] \to D \subset \mathbb{R}^2;\ c \in C^4 \right\},$$

onde $D$ é um domínio de interesse, podendo ser todo o domínio de definição da imagem ou apenas uma parte deste.

Uma snake é um contorno que pode ser descrito por uma função $c : [0,1] \to \mathbb{R}^2$ com certas condições sob seus extremos, se forem requeridas pela situação. Geometricamente, uma snake pode ser vista no plano como uma curva paramétrica fechada $c(s) = (x(s), y(s))$,onde $s \in [0,1]$ é o comprimento do arco normalizado ao longo do contorno(domínio paramétrico), e x, y são as funções coordenadas de $c$ (figura 2.3, 2.4, 2.5).

O contorno é colocado sobre a imagem $f : \mathbb{R}^2 \to \mathbb{R}$, e esta se move em direção a uma

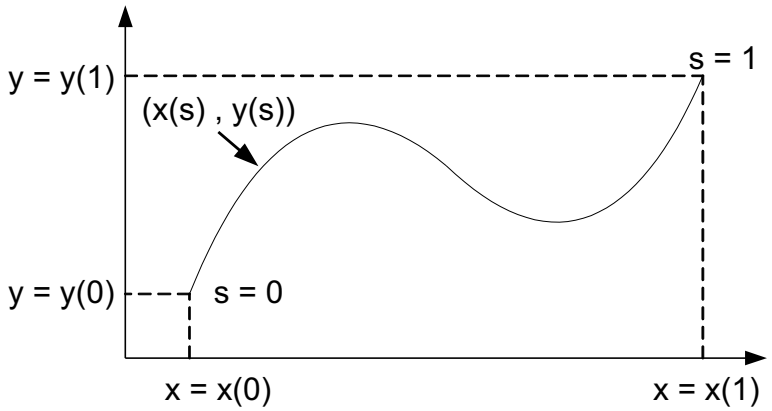


Figura 2.4: Snake: curva aberta.

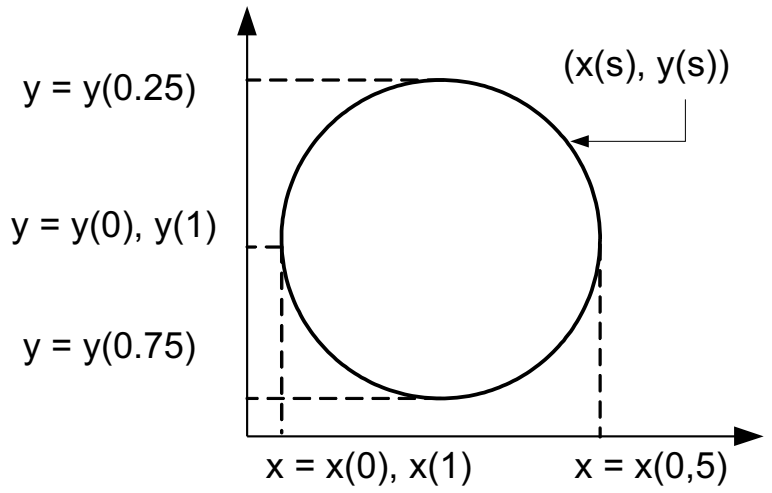


Figura 2.5: Snake: curva fechada.

posição e forma “ótimas” através da minimização de sua própria energia.

Basicamente, este método originalmente proposto por Kass at al. [23], parte da idéia de formular o problema da extração de bordas via a otimização de um funcional montado a partir das características de interesse (bordas, no caso) [15].

Em Kass at al. [23] o funcional de energia da snake tem a forma geral dada por:

$$E(c) = E_{ext}(c) + E_{int}(c) + E_{vínculo},$$

onde $E_{ext}$ é a *energia externa* da snake, e refere-se às características extraídas da própria imagem e que servirá para caracterizar a ocorrência de bordas de interesse durante o processo dinâmico de movimentação da Snake

$$E_{ext}(c) = \int_0^1 P(c(s))\, ds, \tag{2.2}$$

$E_{int}$ é o temo de regularização, que quantifica a deformação de um contorno elástico, sendo denominado *energia interna*, e dado por:

$$E_{int}(c) = \int_0^1 \left(\omega_1 \|c'(s)\|^2 + \omega_2 \|c''(s)\|^2\right) ds, \tag{2.3}$$

onde $\omega_1$ e $\omega_2$ são parâmetros internos denominados parâmetros de elasticidade e rigidez, respectivamente, e $c'(s) = dc/ds$ e $c''(s) = d^2c/ds^2$.

O potencial externo $P$ vai gerar um campo de forças externo à curva que deverá “puxar” a mesma em direção às bordas. Outras possibilidades para a energia interna podem ser encontradas em [27] onde encontramos uma discussão criteriosa para o problema da regularização em visão computacional.

O termo $E_{vínculo}$ corresponde a forças de vínculo que podem ser definidas a priori ou através de interação com o usuário. Não trataremos deste termo nesta discussão.

Assim, a energia da snake toma a forma:

$$E(c) = \int_0^1 \left(\omega_1 \|c'(s)\|^2 + \omega_2 \|c''(s)\|^2\right) ds + \int_0^1 P(c(s))\, ds. \tag{2.4}$$

Este funcional corresponde à energia potencial de uma corda fina imersa em um campo de forças externo definido pelo potencial $P$ (uma variação do modelo de placa fina usado para superfícies [5]). Vejamos alguns detalhes deste modelo iniciando pelos parâmetros do mesmo [13].

O efeito dos parâmetros internos $\omega_1$ e $\omega_2$ na energia acima pode ser explicitado considerando-se as expressões para as derivadas $c'(s)$ e $c''(s)$ em (2.4). Seja então o comprimento de arco $\alpha$ da curva $c$. Então, as derivadas em (2.4) tomam a forma:

$$\frac{dc}{ds} = \frac{dc}{d\alpha}\frac{d\alpha}{ds}, \tag{2.5}$$

$$\frac{d^2c}{ds^2} = K\overrightarrow{N}\left(\frac{d\alpha}{ds}\right)^2 + \overrightarrow{T}\frac{d^2\alpha}{ds^2}, \tag{2.6}$$

onde $\overrightarrow{T}$ e $\overrightarrow{N}$ denotam o vetor tangente unitário e o vetor normal unitário, respectivamente, e $K$ é a curvatura. Assim, os parâmetros $\omega_1$ e $\omega_2$ em (2.4) multiplicam, respectivamente, os seguintes termos:

$$\left\|\frac{dc}{ds}\right\|^2 = \left(\frac{d\alpha}{ds}\right)^2, \tag{2.7}$$

$$\left\|\frac{d^2c}{ds^2}\right\|^2 = \left\|K\overrightarrow{N}\left(\frac{d\alpha}{ds}\right)^2 + \overrightarrow{T}\frac{d^2\alpha}{ds^2}\right\|^2 \tag{2.8}$$

Vemos desta forma que o termo que multiplica $\omega_1$ varia durante a deformação somente se o comprimento de arco da curva se alterar. Portanto, este termo está diretamente vinculado à elasticidade da curva (*corda*), o que justifica a denominação *parâmetro de elasticidade* para $\omega_1$.

O termo (2.8) inclui a curvatura e o vetor normal e, portanto, está diretamente vinculado à forma da curva. Quanto maior o fator $\omega_2$, mais dependente é a energia da corda de sua forma, o que implica em maior resistência à deformação ou mais precisamente, em maior rigidez da corda. Em síntese: o coeficiente $\omega_1$ pondera uma energia associada a uma compressão ou a um estiramento da snake enquanto que $\omega_2$ pondera uma energia associada à sua curvatura. Em Giraldi [13] temos um detalhamento maior sobre as equações de Euler-Lagrange.

### 2.2.2 Limitações do Modelo Original

Em [13] temos um estudo detalhado sobre as limitiações do modelo original conforme descrito abaixo.

Apesar de suas vantagens para extração de contornos, o modelo original de snakes apresenta várias limitações discutidas em [20, 14, 11].

Primeiramente, o funcional de energia (2.2.1) é em geral não convexo e assim, os métodos de otimização baseados em informações locais não oferecem garantia de que a solução encontrada seja o mínimo global [14]. Este problema implica que a solução é sensível à inicialização do método.

Ao problema da não-convexidade da energia acrescenta-se as dificuldades com a componente normal das forças internas. Esta força tem o efeito de suavizar a solução final mas tende a contrair a curva inicial (suposta fechada) fazendo com que a mesma colapse em um ponto caso as forças de campo não sejam suficientes para equilibrá-la. Tal comportamento traz dificuldades para a escolha dos parâmetros do modelo.

Esse comportamento da força normal interna está diretamente relacionado com a não invariância da energia (2.4) por transformações de escala [14, 16]. Isto fica explicito na equação (2.7) a qual mostra que a energia da snake varia com a mudança do comprimento de arco. Assim, por exemplo, a energia interna de uma circunferência $((s \rightarrow r\,(\cos(s), \mathrm{sen}(s))))$ diminui quando seu raio diminui.

Tal comportamento pode não ser desejável quando possuímos informações a priori sobre a geometria dos objetos de interesse. Por exemplo, quando sabemos que os objetos de uma cena são deformações de um protótipo, não é em geral interessante que escalas menores tenham prioridade. Em resumo, a energia interna da snake deve ser modificada para admitir invariância por outras transformações além da rotação e translação.

Uma outra limitação inerente ao modelo da seção é a sua dificuldade de tratar com mudanças da topologia da curva inicial, isto é, aplicações de operações de splits sobre a curva inicial [20]. A menos que acrescentemos outros recursos ao modelo, a evolução da curva baseada na minimização do funcional (2.7) impõe que a variedade correspondente á curva permaneça conexa durante a sua evolução; isto é, a curva não pode sofrer subdivisões (*splits*).

Neste trabalho estamos particularmente interessados em soluções para os problemas dos mínimos locais e limitações topológicas do modelo original.

# Capítulo 3

# T-Snakes: Snakes Topologicamente Adaptáveis

## 3.1 Introdução

Apesar das vantagens, estabelecidas no capítulo anterior, os modelos clássicos de snakes paramétricas têm várias limitações que restringem sua utilidade em um amplo conjunto de problemas em análise de imagens e que podem tornar impossível automatizá-las completamente mantendo-se, ao contrário, a necessidade de que o usuário interfira no processo [10]. Um defeito, que se torna impeditivo em certas aplicações é sua inflexibilidade topológica. Neste capítulo será descrito a classe de modelos de contorno deformáveis conhecida como Snakes topologicamente adaptáveis ou T-snakes. Os modelos dessa classe incorporam uma decomposição do domínio da imagem em células afins(DICA) [18, 1], criando um framework matemático que significativamente estende a capacidade do modelo padrão de snake. A decomposição da imagem em células afins divide o domínio da imagem numa coleção de polígonos convexos que se repetem periodicamente. Snakes paramétricas convencionais são imersas em uma *DICA* e se aplica um procedimento de reparametrização para evitar que vários snaxels sucessivos se localizem na mesma célula. O resultado é uma estrutura capaz de adaptar-se a geometrias e topologias complexas (as T-snakes). Deve-se observar que a *DICA* capacita as T-snakes a manter as características tradicionais dos modelos paramétricos, tais como interação com o usuário simples e o fato do deslocamento dos snaxels ser determinado por funções de energia ou de força, enquanto supera várias limitações desse modelo. Ela também fornece um mecanismo conveniente para a incorporação de complicadas restrições geométricas(por exemplo: fazer com que duas snakes não se cruzem) e topológicas(por exemplo: fazer com que definam uma poligonal simples). Estas propriedades combinadas produzem uma ferramenta ge-

ral e efetiva para a extração e análise de estruturas presentes em uma imagem totalmente automatizado.

## 3.2 Fundamentos

A segmentação de imagens (o particionamento de pontos da imagem em subconjuntos correspondentes a estruturas significativas) é um primeiro estágio essencial da maioria das tarefas de análise de imagens, incluindo a análise das estruturas representadas, sua visualização (preferencialmente individualizada), registro em relação a outras imagens delas, rotulação dos pixels usados para representá-las, e rastreamento do movimento efetuado por elas numa seqüência de frames. Estas tarefas normalmente requerem que as estruturas imageadas sejam reduzidas a uma forma de representação compacta. A segmentação, manual de imagens como era feita, especialmente no contexto de imagens médicas há alguns anos podia ser extremamente trabalhosa e demorada. Assim procedimentos semi-automáticos e, mais idealmente, técnicas completamente automatizadas passaram a ser uma meta altamente desejável. Deve-se atentar para o fato de que a automatização não só alivia o especialista de um trabalho tedioso, mas também contribui para uma maior eficiência, precisão, e talvez acima de tudo, capacidade de reproduzir as segmentações. Porém, como interpretações automatizadas errôneas são normalmente inaceitáveis em contextos como o de imagens médicas, qualquer técnica de segmentação para ser bem sucedida deve incluir a possibilidade, através de uma interface intuitiva, de que o perito interfira no processo, seja para guiar a sua evolução ou simplesmente editar os resultados obtidos. Dentro desse quadro pode-se compreender a opção por se procurar estender a capacidade do modelo padrão de snakes, aumentando seu grau de automatização, enquanto se mantêm seus tradicionais pontos fortes como a interação fácil com o usuário. Outra propriedade importante a ser mantida é a que se refere à parte física do modelo, conservando-se a habilidade de escolher diferentes funções representando energias e forças para determinar a evolução do mesmo.

As limitações do modelo padrão de snakes [20] podem ser de três tipos: posicional, geométrica e topológica conforme o indicado a seguir:

- **snakes paramétricas** padrão foram projetadas como modelos interativos e eles normalmente devem ser trazidos para perto dos bordas do objeto designado para que possam aproximar adequadamente essas bordas. Assim, para tornar as snakes mais automáticas, devem ser adicionados mecanismos que superem esse problema de

sensibilidade à inicialização;

- **a parametrização geométrica** fixa de uma snake padrão (onde por exemplo, o número de snaxels se mantém fixo) junto com as restrições determinadas pela energia de deformação interna (normalmente vinculada ao comprimento do arco e a curvatura) podem limitar sua flexibilidade e impedir a snake de se adaptar a formas tubulares longas ou com ramificações significativas ou ainda com protusões;

- **a parametrização fixa** de uma snake padrão também a faz incapaz de transformações topológicas, e em vista disso a topologia do objeto de interesse deve ser conhecida com antecedência. Na verdade, uma snake diferente deve ser inicializada para cada componente conexa da borda do objeto a ser encontrado.

Vários pesquisadores tentaram superar algumas destas limitações adicionando maior funcionalidade à formulações dos modelos paramétricos, como a introdução de forças que inflam a snake [8, 9], o uso de mecanismos automáticos para a subdivisão da snake [17] e a utilização no processo de detecção de bordas de informações relativas a região que se quer segmentar [25]. Porém, mesmo com o uso desses recursos, os modelos paramétricos utilizados continuaram incapazes de se adaptar a topologia dos objetos procurados e os resultados de segmentação permaneciam muito dependentes da posição inicial. Assim quando se lida com imagens que contém objetos de formas complexas ou objetos cujo contorno está dentro de outros objetos, mesmo com os aprimoramentos no sentido de melhorar a automação ainda pode ser necessária uma interação extensa com o usuário. De forma interessante, em contraste com o caso 2D, há modelos de superfícies deformáveis paramétricas 3D que são capazes de se adaptar automaticamente à topologia do objeto [19]. Ao contrário das T-superfícies que são automaticamente reparametrizadas pelo framework DICA, o processo de reparametrização destes modelos é tipicamente baseado em regras de subdivisão e não na geometria local do objeto alvo. O mecanismo de refinamento de triângulos pode, entretanto, criar sensibilidade à posição inicial, o que adversamente afeta a reprodutibilidade da segmentação. Para introduzir flexibilidade topológica, vários pesquisadores desenvolveram snakes implícitas adotando a técnica de evolução que emprega os level-sets de Osher e Sethian [24] para o problema de segmentação de imagem [6]. Nestes modelos os contornos que evoluem como “frentes” que definem uma curva de nível de alguma superfície imersa no espaço tridimensional projetadas sobre o domínio da imagem. A característica principal deste enfoque é que mudanças de topologias são controladas naturalmente, dado que o lugar geométrico dos pontos em

um mesmo nível simplesmente não precisa ser conexo. A superfície imersa no espaço 3D permanece como o gráfico de uma função simples até mesmo quando as superfícies de nível(level set) mudam de topologia. Concebida como sendo uma dessas curvas de nível, a T-snake ainda é projetada sobre as arestas da DICA. Essa projeção significa a substituição dos vértices originais pelas intersecções com as arestas da malha. A projeção serve para eliminar os segmentos dela que estejam inteiramente contidos em uma única célula e, em particular, as intersecções entre eles. As T-snakes geradas dessa maneira, mantém as características desejáveis das snakes convencionais, como facilidade de interação com o usuário e incorporar restrições baseadas nas energias ou no sistema de forças utilizadas. Assim, elas podem aproximar objetos com topologias complexas ou diversos objetos imersos no background, como também automaticamente fundir-se com outras T-snakes introduzidas pelo usuário. As T-snakes podem ser feitas menos dependentes da posição inicial do que as snakes comuns: usualmente, para diferentes situações iniciais, as soluções finais encontradas são equivalentes no sentido de cortarem as mesmas células.

## 3.3 Formulação das T-Snakes

Tal como uma snake comum uma T-snake [20, 21] é um contorno poligonal fechado cujos vértices são sensíveis a ação de forças físicas determinadas por energias (energia externa, campo externo) que dependem tanto dos dados da imagem como do próprio traçado da poligonal (energia interna). As forças determinadas pela energia interna agem no sentido de tornar suave o traçado da snake e os usuários podem interagir com o modelo, por exemplo, através de forças expressas por $f(x) = Kd(x, p)$, onde $d$ é a distância entre um ponto ($x$) qualquer e uma dada localização $p$ onde se quer que a snake passe. Essas forças cuja definição emulam o modelo físico de resistência de uma mola são por isso chamadas, “spring forces”. Por razões computacionais óbvias, se aplicam normalmente apenas numa vizinhança de localização $p$. Forças de expansão podem também ser usadas para empurrar o modelo em direção às extremidades da imagem até que sejam compensadas pelo campo externo determinado pelos dados de imagem. A deformação do modelo é gorvernada por discretizações das equações de Lagrange para o movimento. Diferentemente das snakes tradiconais, o número de vértices e o de arestas de uma T-snake não permanece constante durante sua evolução.

O domínio da imagem é decomposto em uma malha de células discretas. A cada movimento o modelo é reparametrizado obtendo-se um novo conjunto de vértices que são os

pontos de intersecção dele com as arestas da malha. Também se guarda dados da região já varrida pelo modelo através da "ativação" de quaisquer vértices da malha já visitados pela T-snake durante seu movimento. Além disso, após um certo número de iterações pode-se refinar a malha através de uma técnica de subdivisão simples. O modelo é relativamente insensível a sua posição inicial e consegue evoluir de maneira equilibrada na presença de formas e topologias complexas. É interessante observar que ele provê tanto uma representação da borda, assim como uma da região interior de um objeto, dada pela coleção de células interior a essa borda. Além combina elementos como o particionamento do espaço e a parametrização intrínsica induzida pela malha e como a flexibilidade topológica que uma formulação implícita possui com a facilidade de descrição da borda de um modelo paramétrico. O movimento de uma T-snake é análogo ao movimento de uma frente de propagação. Porém, existem duas fases distintas para o movimento. Na primeira fase, uma T-snake se comporta como uma snake paramétrica padrão e evolui de acordo com a dinâmica Lagrangeana. Por isso seu deslocamento nessa fase é chamado de *deslocamento físico de uma T-snake* e a curva resultante desse deslocamento é chamada de **curva fisicamente transformada** — abreviadamente: $TC$. Durante a segunda fase ou fase de reparametrização, os vértices de $TC$ são substituídos pelas intersecções dela com as arestas da malha, o que é chamado de *projeção sobre a malha*. Por isso a curva definida por essas intersecções é intitulada a **curva projetada** — de forma abreviada: $PC$. Além disso, a ativação ou não dos vértices da grade é usada para localizar o interior do modelo de contorno fechado, criando uma divisão espacial semelhante a aquela determinada por uma curva de nível de uma função implícita. É esta fase provê adaptabilidade topológica. Ela também confere mais elasticidade a evolução na medida de que os snaxels vizinhos que tenham eventualmente se distanciado são substituidos por seqüências de snaxels mais próximos. Na verdade, o deslocamento de cada snaxel é usualmente limitado pelo tamanho das arestas da grade por razões que serão descritos no capítulo 4. A conversão para a representação do modelo de snakes paramétricas tradicionais simplesmente é uma questão de desabilitar a grade a qualquer momento durante o processo de evolução.

## 3.4 Relação com a Frente de Onda

O movimento de uma T-snake é análogo ao movimento de um frente de onda, como em uma propagação de uma chama [20]. Existem várias técnicas numéricas tradicionais que são usadas para resolver as equações de movimento deste problema. O primeiro conjunto

destas técnicas, discutido em [26], parametriza a frente de movimento e discretiza a parametrização em um conjunto de partículas marcadoras ou nós. A direção normal à frente (front), a curvatura e o esticamento são aproximados por derivadas discretas computadas nos nós marcadores. O movimento dos nós marcadores é então governado pela aproximação de equações de movimento. A formulação Lagrangeana comprovadamente sofre de problemas de instabilidade e as mudanças topológicas na linha de frente são difíceis de se detectar dessa maneira. O segundo conjunto destas técnicas, conhecido como técnicas de "volume do fluido" [19], localizam o movimento da região interior no lugar do borda. Estes algoritmos discretizam a região interior sobrepondo uma grade de células ao domínio e atribuindo a cada célula uma "fração do volume" correspondente a quantia do fluído interior corrente na célula. A frente se move pela construção de aproximações poligonais locais da frente em cada célula, baseada nas frações de volume dos vizinhos. Esta técnica Euleriana é estável e as mudanças topológicas são facilmente controladas. Entretanto pode-se tornar difícil calcular propriedades da frente, tais como, curvatura e normais, a partir da representação grosseira da borda usada sem recorrer a grades de resolução muito finas. Uma terceira técnica é o enfoque de curvas de níveis (level set) de Osher e Sethian [24]. Esta técnica modela a frente como curvas de nível de uma superfície imersa num espaço tridimensional que estão em evolução. O movimento dessa superfície é descrito por um equação de Hamilton-Jacob com um lado direito parabólico. Uma formulação de Euler provê estabilidade e as mudanças topológicas são controladas naturalmente no espaço de dimensão superior. Uma T-snake é um modelo híbrido que contém aspectos de todas as três técnicas e procura combinar suas qualidades. Entre estágios de reparametrização, uma T-snake se comporta como uma snake clássica e evolui de acordo com a dinâmica Lagrangeana. Esta fase da formulação de Lagrange permite que qualquer dados derivados ou forças definidas pelo usuário guiem a snake. Durante a fase de reparametrização, a snake é reparametrizada em termos de uma grade simplicial e os pontos fixos da grade são usados para localizar o interior do modelo de contorno fechado, criando um espaço de particionamento similar a aquele de uma função implícita. Esta fase da formulação Euleriana provê estabilidade, parametrização intrínseca e flexibilidade topológica.

## 3.5 Descrição do Modelo

### 3.5.1 T-Snakes

O método das T-Snakes [20, 21], que passaremos a apresentar agora aborda as questões topológicas num outro contexto. Os elementos usados por este método são de três diferentes naturezas: (1) uma triangulação do domínio de interesse, (2) um modelo discreto da curva deformável, (3) uma função característica.

#### 3.5.1.1 Decomposição da Imagem por Célula Afim

A idéia intuitiva de uma decomposição espacial é subdividir espaço em uma coleção de subconjuntos conexos com interiores disjuntos. Usualmente se utilizam células k-dimensionais, isto é, conjuntos homeomorfos, a um disco fechado de dimensão $k$. A borda de cada célula é constituída da união finita de células de menor dimensão e o conjunto de células em que se subdivide o espaço é conhecido como "complexo celular" [18]. Complexos celulares mais comuns são aqueles em que cada célula pode ser obtida aplicando-se a uma outra uma translação ou transformação afim.

Dado um conjunto fechado $D \subset \mathbb{R}^n$, existem dois dois tipos principais de métodos para decomposição do domínio em células: não-simpliciais e simpliciais.

A maioria dos métodos **não-simpliciais** empregam um tesselagem retangular do espaço. Estes métodos são rápidos e fáceis de implementar mas eles não podem ser usados para representar as bordas de um objeto definido de forma implícita — como é o caso das T-snakes, mas não das Loop-Snakes — não ambígüa sem o uso de métodos adicionais para resolver esta questão.

Métodos que usam a **decomposição simplicial** do domínio não apresentam ambigüidades para a geração de aproximações poligonais de curvas ou superfícies de nível de uma função obtidas por interpolação linear a partir dos valores da função nos vértices dos símplices empregados [1].

Na decomposição simplicial do espaço, também conhecida como *triangulação*, o espaço é particionado em células formadas pelos mais simples objetos geométricos em sua dimensão, isto é, triângulos em 2D e tetraedros em 3D. Se todas as células são idênticas, definidas apenas pelo posicionamento e orientação, o esforço computacional necessário para lidar com essa estrutura é menor, possibilitando a criação de algoritmos eficientes e de fácil manuseio. O tipo mais simples de triangulação do espaço Euclideano $\mathbb{R}^n$ com essa propriedade é o de Coxeter_Freudenthal. Esta triangulação é construída dividindo

o espaço por um *grid* quadrático uniforme, sendo a triangulação obtida através da subdivisão de cada um dos quadrados que compõem o *grid* em simplices definidos pelas suas diagonais.

Dado um conjunto de $k+1$ de pontos $\{v_0, v_1, v_2, \ldots, v_k\} \in \mathbb{R}^n$, o mesmo é dito *afim independente* se os vetores $\{v_1 - v_0, v_2 - v_1, v_3 - v_2, \ldots, v_k - v_0\}$ são linearmente independentes no $\mathbb{R}^n$.

O fecho convexo

$$\sigma = [v_0, v_1, v_2, \ldots, v_k] = \left\{ v \in \mathbb{R}^n; v = \sum_{i=0}^{k} \alpha_i v_i \text{ tal que } \alpha_i \geq 0, \sum_{i=0}^{k} \alpha_i = 1 \right\}, \tag{3.1}$$

com vértices dados por $k+1$ pontos *afim independentes* $\{v_0, v_1, v_2, \ldots, v_k\}$ é chamado um *k-simplexo*. Os coeficientes $\alpha_i$ geralmente são chamados de coordenadas baricêntricas de $v$.

Principalmente em função de se trabalhar precisamente apenas $\mathbb{R}^2$, considera-se desnecessário definir triangulação e uma série de conceitos como simplices, facetas e adjacência de triângulos. Esclarece-se apenas o significado da terminologia que tem um uso menos definido. Assim, dado uma triângulo, o procedimento usado para passar a um adjacente a este é denominado *pivoteamento*.

Mais especificamente: Seja $\sigma = [v_1, v_2, v_3]$ um triângulo de uma triangulação $T$ de $\mathbb{R}^2$ e $\left[v_k, v_j\right]$ a aresta de $\sigma$ oposta ao vértice $v_i$. Existe um único nó $\tilde{v}_i$ diferente de $v_i$ e tal que $\tilde{\sigma} = \left[v_j, \sigma_j, \tilde{v}_i\right]$ é um triangulo de $T$ . A passagem de $\sigma$ para $\tilde{\sigma}$ é chamada *pivoteamento*. Dizemos que através da face $\tau$, o vértice $v_i$ de $\sigma$ é *pivoteado* no vértice $\tilde{v}_i$, e que o simplexo $\sigma$ é *pivoteado* no simplexo $\tilde{\sigma}$ .

#### 3.5.1.2 Aproximação Simplicial da Snake

Se o traçado da snake fosse a curva de nível zero de uma função, então os conjuntos de pontos interiores e exteriores a ela ficariam inteiramente caracterizados pelo sinal de sua função (*Propriedade P1*).

Suponha ainda que a intersecção do traçado da snake com cada triângulo, seja uma única curva simples aberta com extremidades em arestas diferentes (*Propriedade P2*). Nesse caso um elemento de transição (face ou aresta), ou seja, um elemento que é cortado pela snake, ficará caracterizado por ter vértices onde a função tem sinais diferentes. Tri-

ângulos de arestas interiores ou exteriores à curva ficam caracterizados pelo fato de todos os seus vértices terem um dado sinal 3.1.

Assumidas essas duas propriedades, então uma aproximação da snake com um erro menor pode ser obtida da seguinte maneira:

- **1)** Avalia-se o sinal da função nos vértices da malha triangular.
- **2)** Determina-se a seqüência de arestas da triangulação que ligam vértices onde os sinais são diferentes.
- **3)** Escolhe-se um ponto em cada uma dessas arestas e com eles se define uma aproximação poligonal de snake ligando-se dois quaisquer desses pontos que estejam em arestas de um mesmo triângulo.

O problema é que mesmo que a snake obtida numa dada iteração satisfaça as duas propriedades necessárias, elas podem obviamente ser perdidas depois que ela é deslocada pela ação das "forças físicas" consideradas. Se a **curva fisicamente transformada** da iteração, tiver um traçado genérico várias dificuldades podem surgir:

- **1)** Auto-interseções precisam ser eliminadas dado que elas impedem que a curva seja a borda de um segmento como se deseja obter ao final do processo. Além disso, pode ficar extremamente difícil definir uma função tal que $TC$ seja uma de suas curvas de nível.
- **2)** $TC$ pode ser uma curva simples mas cortar um mesmo triângulo segundo diversos segmentos de curva. Nesse caso pode não ser possível obter uma aproximação de $TC$ satisfazendo a propriedade $P2$ e com erro menor que $d$ sem sacrificar a convergência do processo de evolução da snake. Para garantir a monotonicidade do processo de evolução temos que caracterizar a aproximação de uma forma mais fraca: o que se pode exigir é que ela se situe na faixa entre as bordas da abertura e do fechamento do conjunto delimitado por $TC$ obtidos usando-se um elemento estruturante de raio $d$.

#### 3.5.1.3 Modelo de Snake

A T-Snake é uma forma discreta da clássica snake descrita no capítulo anterior. É definida por um conjunto de $N$ pontos (snaxels) cujas posições $\{v_i = (x_i, y_i), i = 0, ..., N-1\}$, são conectadas em seqüência formando uma curva fechada.

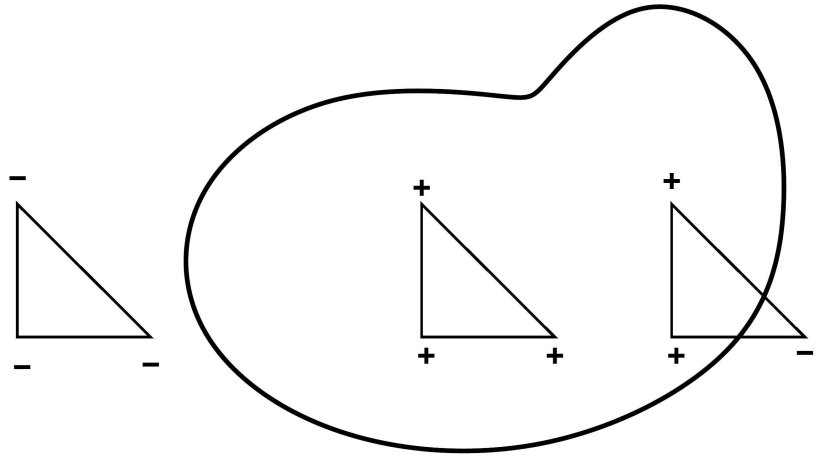


Figura 3.1: Classificação do simplexo.

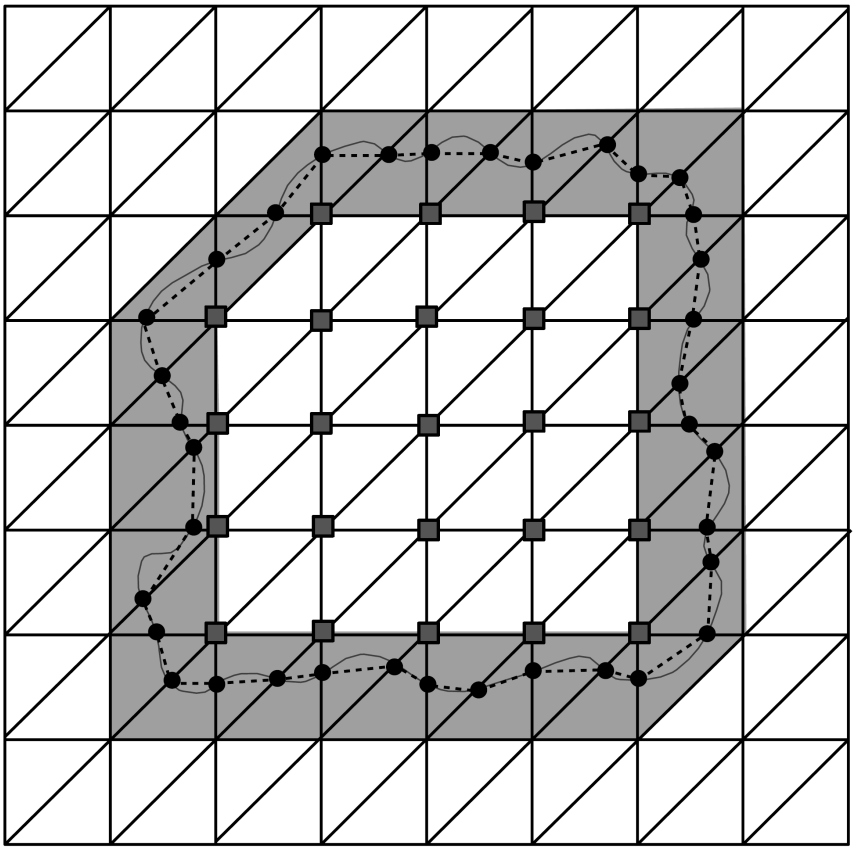

Figura 3.2: Aproximação simplicial (limha pontilhada) do contorno de um objeto (linha sólida) usando a triangulação Freundenthal. Os nós modelos (pontos de intersecção) estão marcados e as bordas do triângulo sombreadas.

Os pontos da snake são supostamente ligados por molas as quais são definidas por um parâmetro de elasticidade $a_i$ e um comprimento natural $l_i$ , isto é, a snake resiste a expansão ou compressão somente quando a distância entre snaxels $\|r(t)\| = \|v_{i+1} - v_i\|$ é maior ou menor que $l_i$, respectivamente.

Assim, dada a deformação $e_i = \|r_i(t)\| - l_i$, nós definimos a força de tensão correspondente pela expressão:

$$\alpha_i = a_i e_i r_i(t) - a_{i-1} e_{i-1} r_{i-1}(t). \tag{3.2}$$

Uma vez que o conjunto de snaxels e molas não permanece constante, o comprimento de repouso das molas no instante $t$ é definido em termos dos comprimentos das molas no instante anterior $t - \Delta t$. Isto fornece ao modelo o comportamento de um material com viscoelasticidade não nula [20].

Em adição à força (3.2) é conveniente definir uma força de rigidez para minimizar as curvaturas locais da snake garantindo suavidade para a solução. Esta força é definida da seguinte forma:

$$\beta_i = b_i \left( v_i - \frac{1}{2} (v_{i-1} + v_{i+1}) \right), \tag{3.3}$$

a qual atua no sentido de minimizar a distância entre um ponto $v_i$ e o centróide do segmento definido pelos seus vizinhos.

O modelo tem também uma força normal:

$$F_i = \text{sig}\,(v_i)\, k n_i, \tag{3.4}$$

onde $\text{sig}\,(v_i) = -1$ se $I\,(v_i) < T$ e $\text{sig}\,(v_i) = 1$ caso contrário ($T$ é um limiar para a intensidade de imagem previamente determinado).

Na expressão 3.4, $n_i$ é a normal ao vértice $v_i$ e $k$ é fator que define a intensidade da força. Esta é uma força tipo balloon , sendo usada para puxar a snake em direção às bordas dos objetos procurando evitar que a snake pare em regiões onde o campo externo é nulo.

Outra possibilidade para definir o sinal da força normal é usar estatísticas da imagem para definir a função $\text{sig}\,(v_i)$:

$$\text{sig}\,(v_i) = \begin{cases} 1 & \text{se } |I\,(v_i) - \mu| \leq \sigma, \\ 0 & \text{caso contrário.} \end{cases}$$

onde $\mu$ e $\sigma$ são a média e variância da imagem, respectivamente.

As forças dadas (3.2)-(3.4) são forças internas.

As forças externas são definidas em função das características de interesse na imagem, no caso, bordas dos objetos da cena. Uma possibilidade é definir o campo externo através do gradiente do potencial usual:

$$P = -\|\nabla I\|^2 , \tag{3.5}$$

ou de uma normalização do tipo daquela usada no Balloon:

$$f = \begin{cases} 0 & \text{se } \nabla P < T_F, \\ \lambda \frac{\nabla P}{|\nabla P|} & \text{se } \nabla P > T_F, \end{cases}$$

onde $T_F$ é um limiar previamente estabelecido e $\lambda$ é um fator de escala para a força.

A equação de evolução para a T-Snake é finalmente dada por:

$$v_i^{(t+\Delta t)} = v_i^t + \frac{\Delta t}{\gamma} \left(\alpha_i^t + \beta_i^t + F_i^t + f_i^t\right), \tag{3.6}$$

Durante a evolução da T-Snake alguns nós da malha tornam-se interiores à mesma (supondo expansão). Estes nós são denominados *nós queimados*, em analogia ao método dos conjuntos de níveis da seção , o qual inspirou o presente método [20].

Para evitar os problemas conhecidos para evolução de curvas na direção normal (desenvolvimento singularidades) o método das T-Snakes adota uma condição de *entropia* do tipo da usada no método dos conjuntos de níveis [20]: *uma vez que um nó é queimado, ele permanecerá queimado*.

Esta condição tem também a finalidade de permitir a definição de um critério de parada eficiente (daí o nome entropia). Neste sentido, primeiramente define-se uma *temperatura* para cada snaxel a qual é o número de deformações que o triângulo em que o elemento de modelo correspondente se encontra permaneceu como um triângulo de borda.

Uma T-Snake é considerada em equilíbrio quando a temperatura de todos os seus snaxels ultrapassar um limiar denominado *ponto de congelamento* (*freezing point*). Este limiar é estabelecido empiricamente.

A condição de entropia acima torna esta definição mais eficiente como critério de parada, pois do contrário uma snake poderia oscilar, expandindo e contraindo, e assim o número de interações necessárias para atingir o equilíbrio poderia ser muito alto. No entanto, esta condição limita o movimento da snake o que pode trazer dificuldades como veremos a seguir.

Uma vez que a T-Snake atingiu o equilíbrio, a malha correspondente à divisão simplicial pode ser descartada e a snake evoluir como uma snake discreta segundo a equação (3.6)

**Reparametrização do Modelo**

Durante cada evolução da curva (equação 3.6), a T-snake se movimenta de sua posição original para uma nova posição. No início de cada evolução, os snaxels estão localizados nas arestas da grade triangular previamente definido para a evolução da T-snake. Ao final de cada passo, os snaxels estarão localizados, em geral, "fora" das arestas dos triângulos da grade. É necessário restabelecer a correspondência entre o modelo e a grade, computando-se os novos snaxels da T-snake deformada através de um algoritmo de reparametrização.

Inicialmente, é feita uma busca local e testes de intersecção para cada elemento do modelo. Isto é, para cada segmento que conecta dois nós, será computado uma *bounding box* composta por esse segmento e sua nova posição. Para cada uma das arestas contidas nesta *bounding box*, é feito um teste de intersecção em relação ao elemento do modelo. Se um ponto de intersecção é encontrado, ele é guardado e poderá tornar-se um *snaxel* no modelo atualizado (Figura 3.3).

#### 3.5.1.4 Reparametrização Iterativa

Durante os *N* passos da equação 3.6 (referida como passo de deformação) a T-snake se move a partir de sua posição atual para uma nova posição. No começo do passo de deformação, os nós estão definidos em termos das arestas dos triângulos de borda do grid (triangulação). No fim do passo de deformação, os nós já se moveram para fora das arestas dos triângulos do grid (figura 3.3a). Então se restabelece a correspondência do modelo com o grid fazendo a computação de uma nova aproximação simplicial da T-snake deformada. Esta nova aproximação simplicial é computada usando um algoritmo de reparametrização de duas fases.

Na fase I, se executa uma busca local e testes de intersecção para cada elemento do modelo (par de snaxels consecutivos). Isto é, para cada elemento conectando dois nós, se computa a bound box do elemento e sua nova posição. Usando esta bound box, se determina quais arestas dos triângulos do grid potencialmente podem cortar o elemento do modelo. Para cada uma das arestas dentro da bounding box, um teste de intersecção é executado com cada elemento do modelo. Se um ponto de intersecção é encontrado, este é gravado e pode se tornar um nó do modelo atualizado(figura 3.3b,c). Se um ponto

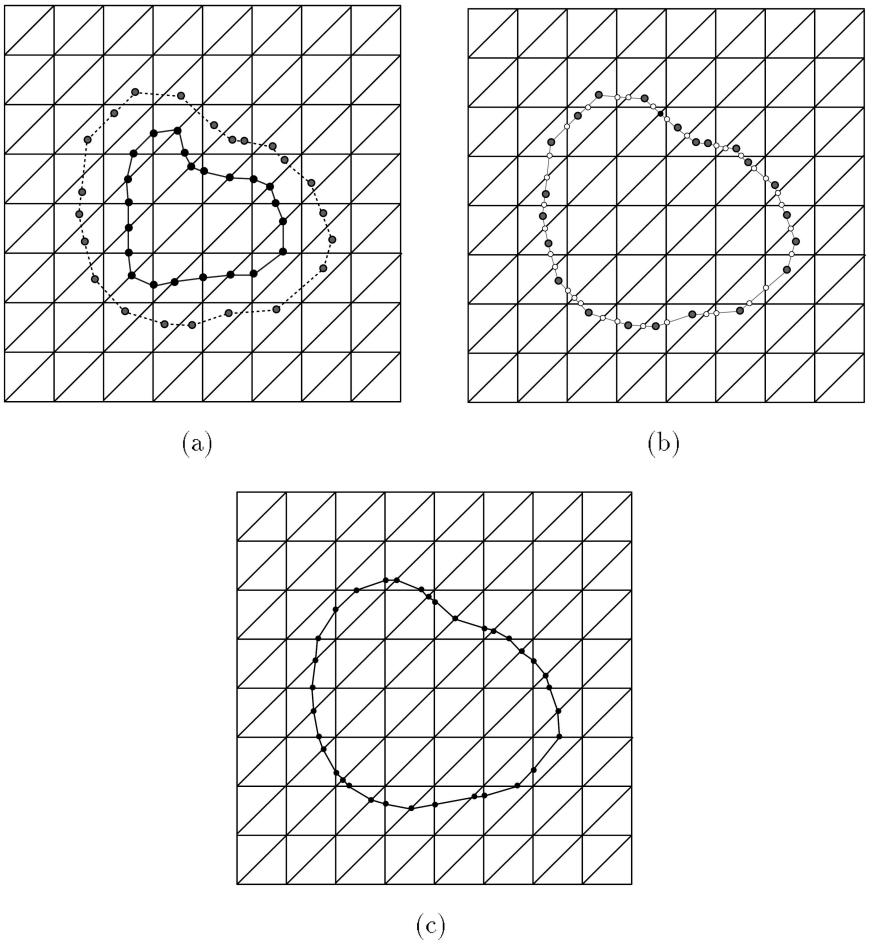


Figura 3.3: Fase 1 da reparametrização do modelo de T-snake: (a) A T-snake se expande e muda de posição durante o passo de deformação, (b) os novos nós são computados, (c) novos snaxels são criados.

de intersecção para esta aresta já existe, pode-se usar tanto o último ponto de intersecção como o ponto mais próximo do vértice externo ao grid desta aresta. Esta fase do processo de reparametrização é simples e eficiente e inerentemente paralela; cada elemento do modelo pode ser processado independentemente.

Durante o passo de deformação, uma T-snake passa sobre um conjunto de vértices de triângulos do grid (figura 3.4a). Usando a analogia da onda de propagação, especificamente uma chama de propagação, estes vértices do grid já foram "queimados". Se é capaz de determinar e localizar a região interior da T-snake através da identificação e gravação destes vértices queimados do grid durante o passo de deformação. Além disso, estes vértices internos do grid definem sem ambigüidade as bordas do modelo; eles são usados para localizar continuamente os triângulos de borda do grid ao longo da evolução do modelo e conseqüentemente determinam o conjunto de novos nós do modelo usados para formar a T-snake da borda. Através da manutenção da representação de uma região interior como também uma representação da borda da T-snake, se é capaz de construir uma função característica [20, 21] de um objeto $O$. Esta é uma função linear por partes definida por:

$$\chi : D \subset \mathbb{R}^2 \rightarrow \{0, 1\}, \tag{3.7}$$

onde $\chi(p) = 1$ se $p \in O$ e $\chi(p) = 0$, caso contrário, onde $p$ é um nó da malha e $D$ o domínio de interesse.

Determinar o conjunto de vértices do grid (nós da malha) que foram queimados durante o passo de deformação, usa um algoritmo de classificação(figura 3.5) simples, robusto e original. Cada elemento do modelo (par de snaxels consecutivos) pode ter passado sobre nenhum, apenas um, ou vários nós durante a sua evolução. Para cada elemento do modelo, forma-se um polígono (bounding box) usando posição atual e a anterior do elemento. Este polígono permite determinar rapidamente os nós (vértices do grid ou malha) que podem ter sido queimados (figura 3.5). Para cada nó dentro do polígono, a imagem é particionada em quatro subespaços através das semi-retas, que podem ser vistas nas figuras 3.5, 3.4 a e b. Estas linhas são formadas juntando os nós de um elemento do modelo em suas posições atuais com um nó do grid. Então classifica-se os dois nós do modelo em suas novas posições correntes em um dos quatro subespaços. O algoritmo de classificação consiste essencialmente de vários produtos internos e é extremamente eficiente e inerentemente paralelo. Nas figuras ( 3.4 a-c), está ilustrado a segunda fase do processo de reparametrização. Na figura 3.4 c os novos nós são mostrados em cinza claro. A figura 3.4 d mostra a nova T-snake depois que as duas fases da reparametrização foram

feitas.

#### 3.5.1.5 Algoritmo de Classificação

Nesta seção se descreve um algoritmo eficiente para determinar se um elemento do modelo pode ter passado sobre nenhum ou sobre um nó queimado $v$ durante a fase de deformação. Começa-se particionando o polígono da imagem em quatro subespaços 3.5, formando duas semi-retas $L1$ e $L2$. A linha $L1$ é formada usando o snaxel $p1$ do elemento do modelo(par de snaxels consecutivos) e o nó $v$ e a linha $L2$ é formada usando o snaxel $p2$ e o nó $v$. Durante o passo de deformação (evolução), $p1$ e $p2$ são movidos para novas posições $p1n$ e $p2n$ respectivamente, aproximadamente na direção da normal definida pelos snaxels do modelo $p1$ e $p2$. O passo máximo de cada "snaxel" é restringido a um valor muito menor que a dimensão do domínio da imagem, evitando assim comportamentos degenerados e pode-se assumir que o movimento segue um caminho de linha direto. Desta forma pode-se classificar $p1n$ e $p2n$ em qualquer dos quatro subespaços da figura 3.5. Portanto, tem-se um total de 16 possibilidades a serem consideradas conforme a figura 3.6. Os quatro pontos $p1$, $p2$, $p1n$ e $p2n$ formam um polígono fechado $Q$, que pode ser convexo ou não 3.6. Para tal, duas definições importantes devem ser consideradas [22]:

**1.** Um ponto $p$ é interior a um polígono $Q$ se um raio partindo de $p$ intercepta exatamente uma aresta de $Q$ ou exatamente três arestas de $Q$, ou se $p$ pertence a poligonal definida pelos vértices de $Q$.

**2.** Um nó do grid é rotulado como "queimado" se ele for interior ao polígono $Q$.

De acordo com estas definições, os casos 1, 2, e 7 da figura 3.6 classificam $v$ como queimado. Os casos (3)-(6) e (8)-(12) classificam $v$ como não queimado. Assim, a simples classificação de $p1n$ e $p2n$ em um dos subespaços da figura 3.6 informa imediatamente se $v$ foi queimado, ou não, para a maioria dos casos. O algoritmo de classificação consiste de dois testes de semi-espaços interno e externo tanto para $p1n$ e $p2n$. Um teste de semi-espaço consiste essencialmente de um produto interno entre um ponto $p$ e o "ponto-normal" da equação de uma reta. Além disso, um elemento vizinho de $p1p2$ compartilha uma das semi-retas, $L1$ ou $L2$, e a restrição do máximo movimento do snaxel mencionado anteriormente garante que subespaços degenerados não podem ser formados. Os casos com ambigüidade ( (13)-(16) ) necessitam de um teste adicional. Para cada um destes casos, se executa um teste de semi-espaço interno e externo para o nó $v$ usando uma

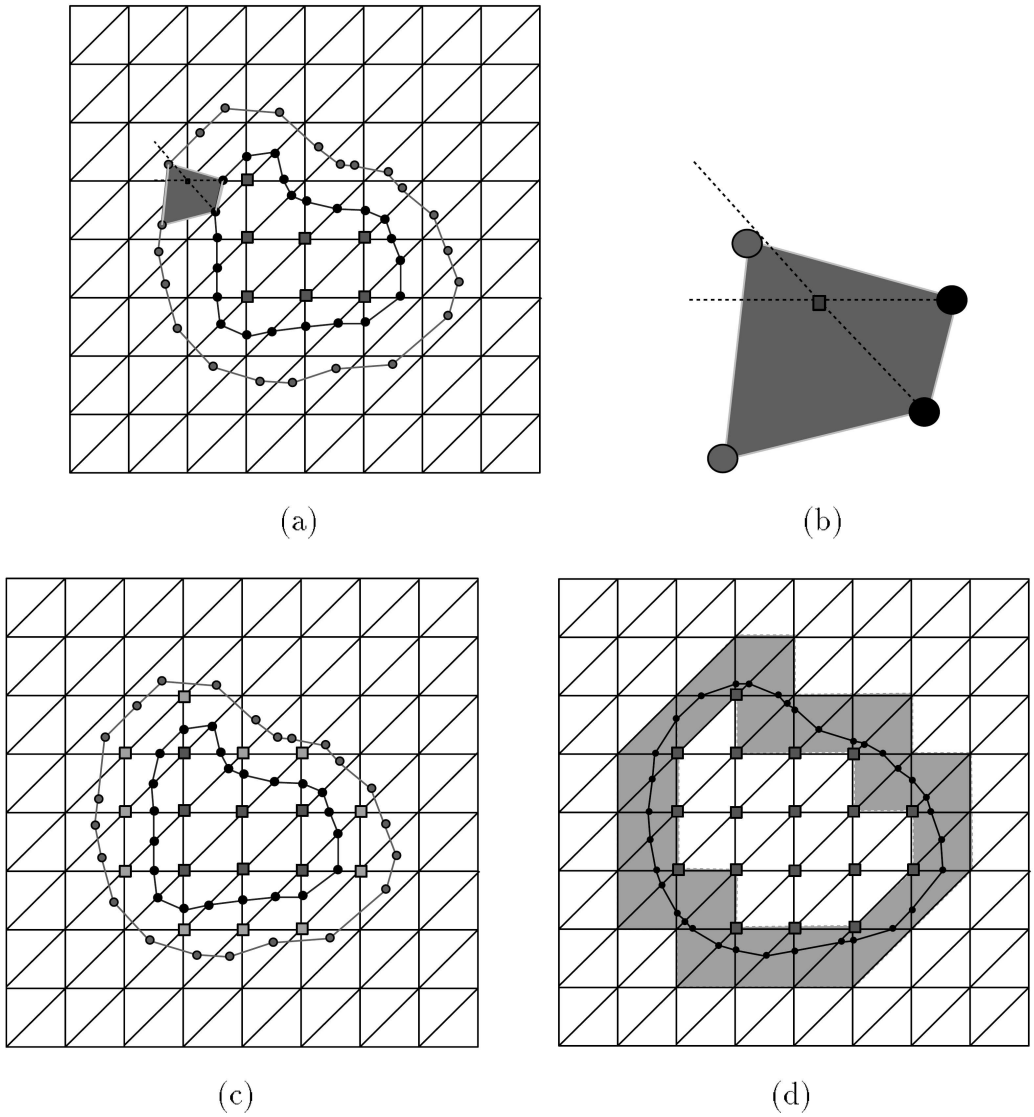


Figura 3.4: Fase 2 da reparametrização do modelo de T-snake: (a) Durante a expansão, a T-snake pode passar por vários vértices do grid, (b) particionamento do subespaço, (c) Modificação da entropia dos novos vértices do grid, (d) Nova T-snake.

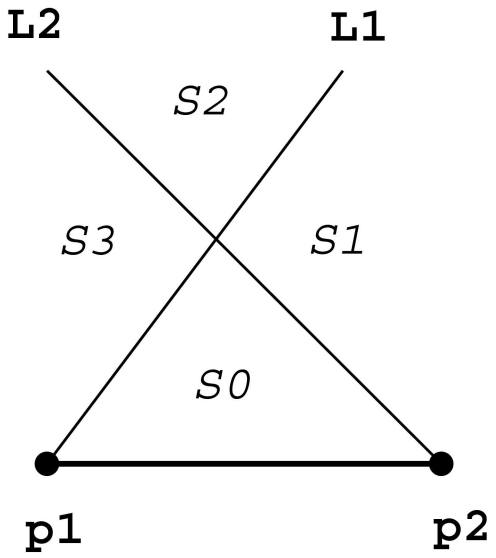


Figura 3.5: Formação de subespaços usados na classificação dos nós.

reta formada pelo elemento do modelo e sua nova posição $p1np2n$ para definir os semi-espaços. Para os casos (13) e (14), $v$ deve ficar no mesmo semi-espaço de $p1$ e $p2$ para $v$ ser classificado como queimado. Para os casos (15) e (16), $v$ deve estar no mesmo semi-espaço de $p1$ e $p2$, respectivamente.

#### 3.5.1.6 Transformações Topológicas

Quando uma T-snake colide consigo mesma ou com outra T-snake, ou quando uma T-snake se divide em duas ou mais partes, uma transformação topológica deve acontecer [22]. A fim de efetuar mudanças topológicas consistentes, decisões precisas devem ser feitas com relação a desconecção e reconecção dos nós da T-snake. O grid simplicial e o processo de reparametrização provê um mecanismo automático e sem ambigüidade para efetuar as reconecções. Ao localizar os nós no interior do grid (e portanto os triângulos de borda do grid), e de acordo com a condição de entropia, e restabelecendo a correspondência do modelo com o grid depois de um passo de deformação (evolução), sempre se pode sem ambigüidade determinar a borda ou "iso-contorno" da nova(s) T-snake(s). Simplesmente se computa os novos elementos do modelo a partir dos sinais dos nós do grid em cada triângulo de borda e a partir destes pontos de intersecção computados na fase 1 da reparametrização, tal que os nós do grid internos e externos destes triângulos são separados pelo elemento do modelo 3.7. Assim, minimizando a envoltura de uma curva de nível (level set) de uma função implícita, o grid simplicial e o processo de reparametri-

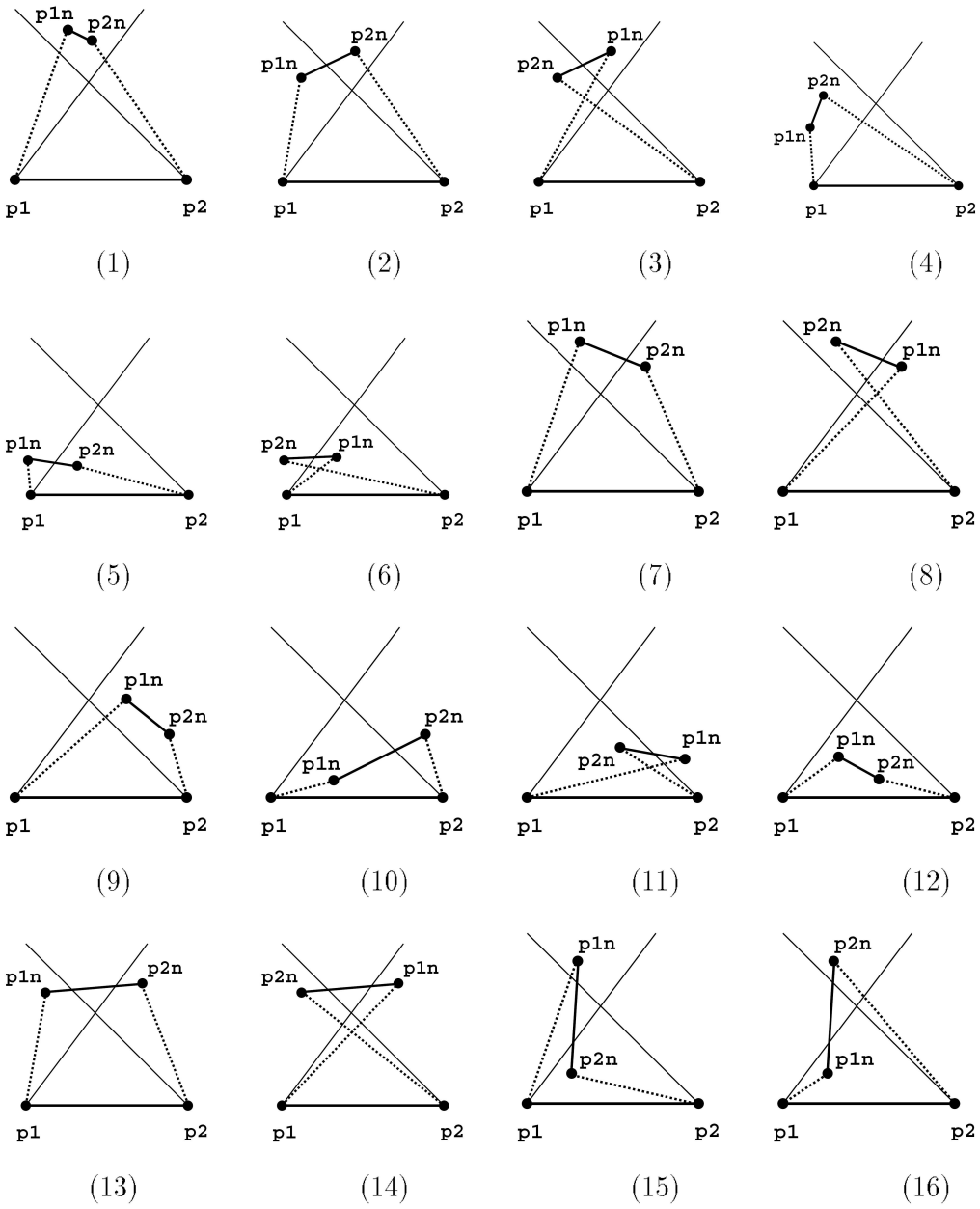


Figura 3.6: Possíveis casos para classificação.

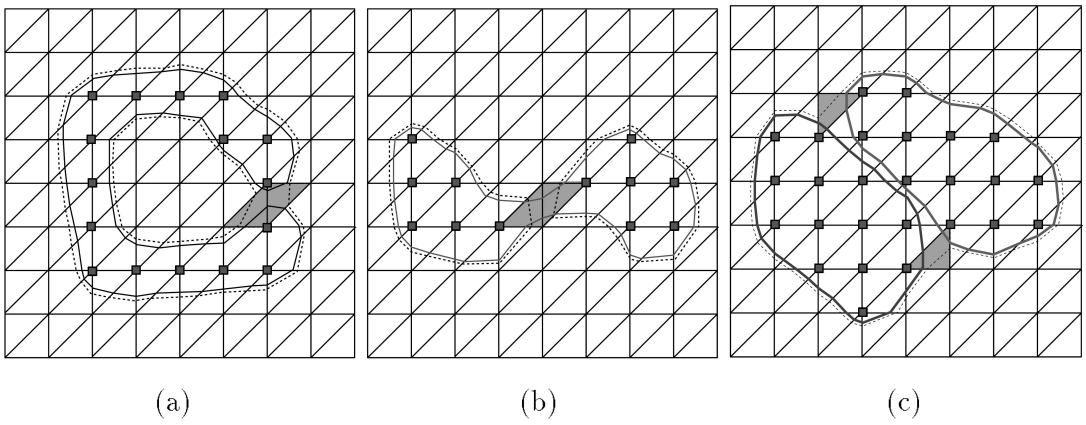


Figura 3.7: Exemplos de transformações topológicas da T-snake: (a) auto-intersecção, (b) compactação e divisão, (c) junção. A(s) T-snake(s) resultante(s) depois das transformaçõws são exibidas como linhas pontilhadas. As reconecções de nós ocorrem automaticamente nos triângulos sombreados de tal forma que os vértices do grid internos e externos estão separados por um elemento do modelo.

zação garantem que as transformações topológicas sejam gerenciadas automaticamente, consistentemente e eficientemente.

## 3.6 O Algoritmo da T-Snakes

Nesta seção apresentamos um algoritmo para implementação do modelo de T-snakes enfocando explicitamente como se deve efetuar o controle de sua topologia. O objetivo é se mostrar que, apesar de toda potencialidade do modelo apresentada nas seções anteriores sua realização computacional encontra algumas dificuldades.

Essencialmente, no enfoque original [21], a evolução da T-snake do passo $k$ — $S_k = [s_{ki}, i = 0, \ldots, I]$ — é computada como indicado abaixo, embora várias variantes desse procedimento sejam possíveis. Os elementos da matriz da *estrutura auxiliar*, que estão relacionados aos vértices da malha, são todos inicializados com zeros (“não visitados”), indicando que todos os vértices começam como “não queimados”. Adicionalmente, $\tau$ será a triangulação $J1$ dos vértices da malha, isto é, a que é obtida cortando-se cada célula ao longo de sua diagonal principal. Uma **aresta de transição** será uma aresta de $\tau$ ligando um vértice visitado a um não visitado. A figura 3.8 representa uma iteração do método. Por questões de visibilidade snaxels de $S_k$ e vértices de $PC_k$ nas arestas diagonais foram excluídos.

**Procedure Evolving_an_original_T-Snake**($\mathbf{S_k} = [\mathbf{s_{ki}}, \mathbf{i} = \mathbf{0}, \ldots, \mathbf{I_k}]$)

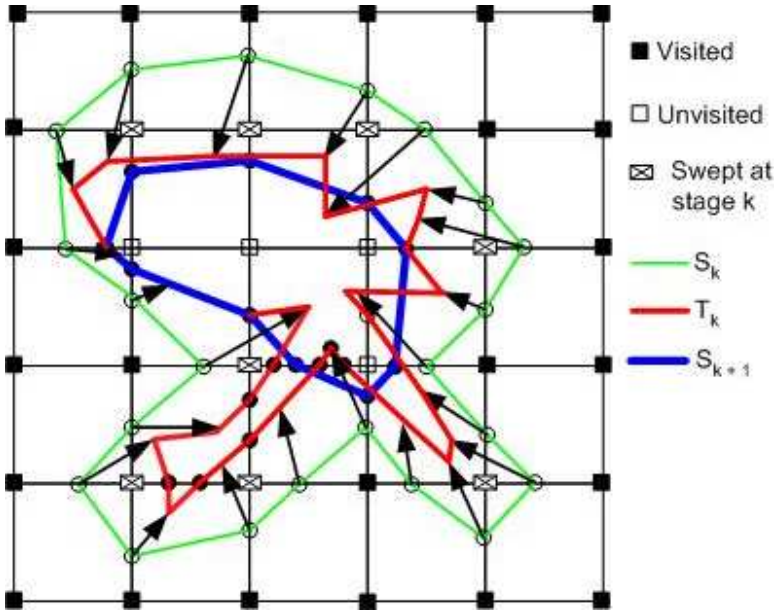


Figura 3.8: Uma iteração da T-snake original.

**Passo** 1. Primeiramente, aplique a cada snaxel $s_{ki}$, o movimento determinado pelo modelo físico empregado. Os pontos obtidos $(t_{ki}, i = 0, \ldots, I_k)$ definem a *Curva Transformada*, $TC_k$. Seja então, $Q_i$ o quadrilátero definido por $s_{ki-1}$, $s_{ki}$, $t_{ki}$, $t_{ki-1}$, $i = 0, \ldots, I$. Esse quadrilátero será chamado de *quadrilátero de varredura* de $[s_{ki-1}, s_{ki}]$, enquanto que o segmento $[s_{ki}, t_{ki}]$ é conhecido como a *raia* de $s_{ki}$, $i = 0, \ldots, I$.

**Passo** 2. Construa a Curva Projetada, $PC_k = [x_j, j = 0, \ldots, J]$ concatenando as arestas $[x_j , x_{j+1}]$, definidas por duas intersecções sucessivas de $TC_k$ com as arestas de $\tau$.

**Passo** 3. Para $i = 0, \ldots, I$, verifique se $Q_i$ contém vértices da malha que ainda não foram queimados(não visitados) e os queime(isto é: acenda o elemento correspondente a ele na estrutura auxiliar).

**Passo** 4. Cada triângulo de $\tau$ ou não tem nenhuma ou tem duas arestas de transição de modo que, conhecendo uma delas, a outra é determinada sem nenhuma ambigüidade. Então, considere que todas as aretas de transição estão inicialmente não marcadas e percorra $PC_k$ completamente. Cada vez que uma aresta de transição $e_0$ não marcada, seja encontrada execute:

4.1 Seja $e = e_0$ e $t$ um dos triângulos adjacentes a $e_0$. Então, repita o seguinte procedimento até voltar para $e_0$.

**A)** Marque $e$ e se $PC_k$ têm vértices sobre $e$, seja $x_e$ uma média deles ou simplesmente escolha $x_e$ entre eles. Se isto não acontecer existe um snaxel de $S_k$ sobre $e$. Nesse caso faça $x_e$ ser a localização deste snaxel. Em qualquer caso guarde $x_e$.

**B)** Obtenha $e'$, a outra aresta de transição pertencente a $t$, e $t'$, o outro triângulo adjacente a $e'$, e atualize $e$ e $t$, com esses elementos.

4.2 Ao retornar para $e_0$, tome a linha poligonal fechada definida pela seqüência de pontos $x_e$ armazenados como uma snake do estágio $k + 1$.

4.3 Se, ao ter percorrido totalmente $PC_k$, nenhuma aresta de transição for encontrada faça $S_{k+1} = S_k$ para manter a snake como um separador entre os conjuntos de vértices queimados(visitados) e não queimados(não visitados).

Assim, uma curva representando cada componente conectada da borda do conjunto visitado é construída a partir de suas intersecções com as arestas de transição encontradas no passo 4.1. Cada uma dessas curvas é transformada em uma nova snake em 4.2.

Agora, considerando que o número de snaxels gerados durante toda a evolução de uma T-snake pode ser de centenas de vários milhares e ainda de vários milhões, se o processo for repetido para cada corte de uma imagem $3D$, explorar todas as possibilidades para reduzir o custo computacional de uma iteração do método se torna um objetivo justificado. No sentido de que pequenas economias elementares obtidas repetidamente um número enorme de vezes podem fazer diferença, as seguintes observações sobre certos pontos do procedimento acima se tornam pertinentes:

**1.** Verificar se os vértices não queimados (não visitados) são cobertos pelos quadriláteros de varredura $Q_i$ — como é feito no passo 3 — pode consumir um tempo considerável, já que isto deve ser feito para cada novo snaxel. Isto é agravado porque ao se usar células triangulares, o conjunto de vértices cobertos por $Q_i$ não é somente uma função das células cortadas por $[t_{ki}, t_{ki-1}]$ mas pode depender da posição dos vértices de $PC_k$. Isto torna necessário determinar a posição de um vértice em relação as raias de $s_{ki-1}$ e $s_{ki}$, e faz com que seja impossível determinar se este vértice deve ser queimado simplesmente consultando uma tabela de dimensões baixas;

**2.** No procedimento dado acima, $PC_k$ pode ser percorrida três vezes. Uma vez no *passo 3*, quando ela está sendo construída, e duas vezes no *passo 4*: uma na busca por arestas

de transição iniciais e uma na determinação de um novo contorno da snake. Curto-circuitando seqüências de vértices consecutivos de $PC_k$ que estão sobre as arestas da malha cortadas por uma nova snake já encontrada, pode-se evitar que eles façam uma visita dupla no *passo 4*. Procurar por uma aresta de transição inicial sem usar o contorno de $PC_k$ como guia, como foi feito aqui, geralmente significa que um número maior de arestas devem ser verificadas;

**3.** O *passo 4.1* requer que dada aresta *e*, um ponto sobre ela seja escolhido para localizar um snaxel que é escolhido. Fazendo isso, entretanto, de forma randômica ou arbitrária, sem levar em conta os cruzamentos com *e* de $PC_k$ ou $S_k$ pode prejudicar o processo de convergência. Em vista disso, deve haver uma estrutura, tal que, estes cruzamentos possam ser acessados quer seja diretamente ou através de uma função de hashing. Isto significa que não é possível trabalhar somente com listas armazenando as linhas poligonais produzidas no estágio corrente além de uma estrutura auxiliar que é um mapa de bits;

**4.** Mover snaxels localizados em arestas diagonais provou-se não ser uma boa opção. O possível ganho em precisão obtido ao considerá-los, não vale o esforço de fazê-los evoluir;

**5.** O uso de uma triangulação $J1$ faz com que curvas com 4 snaxels em um célula quadrada possam ser geradas somente se essa curva não cortar a diagonal principal da célula. Isto faz com que a faixa estreita e longa da figura 3.9(da esquerda, linha cheia) possa ser aproximada pelo processo enquanto que a da figura 3.9(da direita, linha tracejada), que é a figura 3.9(da esquerda) rotacionada de 90°, não possa. Qualquer curva gerada pelo processo que fique muito próxima da segunda curva será fragmentada em pequenos loops cada um envolvendo um vértice da malha diferente.

A partir dessas observações podemos estabeler um **conjunto de objetivos** que devem ser atingidos por um esquema para a implementação das T-snakes com **melhor performance computacional**.

**A) Não verificar se um ponto já foi varrido (visitado, queimado)**, pois isto tem um custo alto. Testar se a célula ou aresta corrrente já foi visitada é certamente menos custoso;

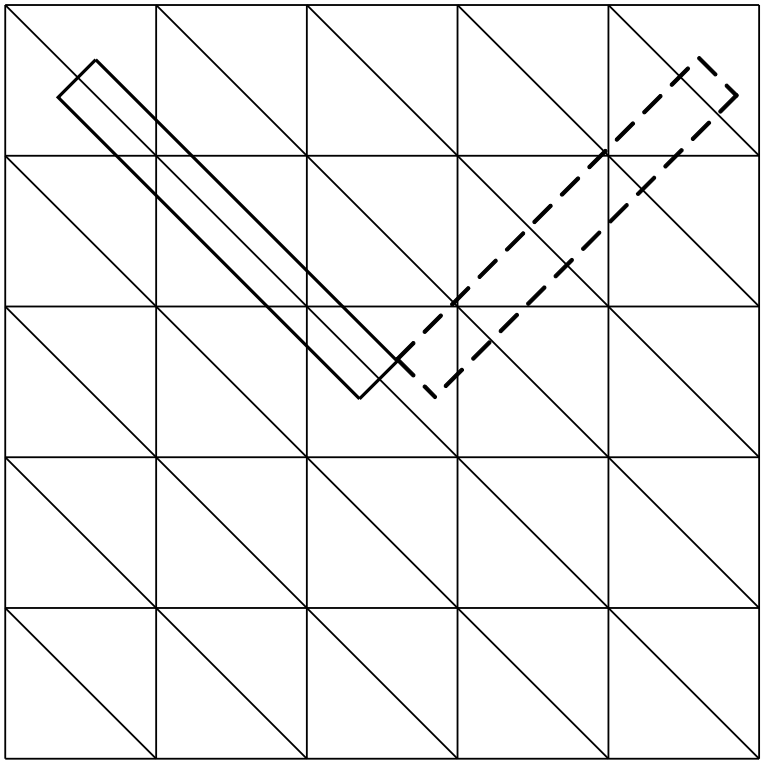

Figura 3.9: O viés imposto pelo uso da triangulação $C - F$.

**B) Não usar triangulações**. A subdivisão da região da imagem empregada deve ser consistituida de células quadradas, que é a solução mais simples para se evitar um víes direcional;

**C) Nenhum histórico** deve ser necessário — Somente a informação gerada em um estágio deve ser processada no estágio. Isto, em particular, torna mais direto fazer o refinamento da malha durante o processo. Isto também faz possível, através de uma organização adequada da informação relativa aos vértices de $PC_k$, dispensar a estrutura auxiliar e ainda assim manter a propriedade de que o trabalho de obter $S_{k+i}$ a partir de $S_k$ seja feito num tempo proporcional ao número de snaxels dessa última snake. Nesse esquema as necessidades de memória não são mais dependentes do tamanho da malha empregada. Esta alternativa, entretanto, não é prática considerando-se o tamanho normal das imagens atualmente;

**D)** Percorrer as curvas relativas a um estágio — $S_k$, $TC_k$ ou $PC_k$ — **só uma vez**. Este requesito pode ainda ser feito mais forte passando a ser expresso por;

**E)** Cada nova snake deve ser determinada **tão logo quanto possível**, isto é, logo após seu último snaxel ser computado.

## 3.7 Um outro modelo para T-Snakes

Um primeiro enfoque com essas propriedades foi introduzido em [4]. O objetivo dessa metodologia é reduzir o máximo possível o esforço gasto no controle da topologia da snake. Isto é alcançado através das seguintes simplificações:

**i.** A fim de não precisar calcular a curva projetada, um snaxel é compelido a se mover somente ao longo da linha da malha sobre a qual ele está[1];

**ii.** Para evitar casos com um topologia problemática, ele não pode ir além do primeiro vértice que encontre. Neste ponto ele é explodido em um snaxel para cada aresta adjacente diferente daquela em que estava. Este esquema faz a curva transformada herdar a simplicidade da snake inicial do estágio. Uma mudança de topologia será feita quando uma aresta contiver dois snaxels não sucessivos

O preço para tal simplificação é pago de diferentes modos — complicando o cômputo dos deslocamentos, introduzindo um viés claro em favor dos vértices da malha e fazendo com que a curva tenda a ficar colada a malha, se se permite que vários snaxels sejam movidos simultaneamente para vértices da malha em uma mesma iteração. Mas o principal problema gerado é o de se reduzir a velocidade de evolução da snake de forma considerável, devido a limitação imposta aos deslocamentos dos snaxels. A versão original propõe que o tempo de um passo seja reduzido de modo que, num caso genérico, um único vértice seja cortado pela snake em uma iteração. Considerando o número de iterações gastos em nossos experimentos isto significa no mínimo dezenas de vezes mais iterações. É bastante óbvio que o deslocamento de um snaxel deve ser limitado para não comprometer o processo de convergência. O truque não é limitá-los mais ainda, somente facilitar o controle topológico. Essa é a proposta do método que será introduzido a partir do próximo capítulo.

[1] Embora isso não seja feito neste trabalho, a curva projetada pode ser substituída por qualquer uma com um snaxel em cada célula cruzada por ela. Obter tal curva a partir de $TC_k$ requer menos computação já que muitos vértices de $TC_k$ podem ser mantidos. Assim, evitar a projeção de $TC_k$ não é uma grande economia.

# Capítulo 4

# Loop Snakes

## 4.1 Background Teórico

### 4.1.1 $\mu$-Curvas e coordenadas "CAP"

Como o foco deste trabalho é o controle da topologia da snake, considera-se que o deslocamento físico de um snaxel é computado por uma "caixa-preta". Assume-se somente, como já é de costume no contexto das T-Snakes, que os deslocamentos físicos têm amplitudes que são menores do que a largura de uma aresta $d$ da célula na $EA$. É claro que o movimento dos snaxels precisa ser limitado para evitar que a snake passe pelos contornos sem detectar a sua presença e o tamanho desse deslocamento está, obviamente, limitado pela precisão com que se deseja efetuar essa detecção. O mesmo se pode dizer em relação as dimensões das células da malha. Por outro lado utilizar uma malha onde as células sejam de uma ordem de grandeza menor que essa precisão seria um desperdício de tempo, pois as T-Snakes obtidas teriam muito mais snaxels que o necessário. Assim temos:

- *O(deslocamento de um snaxel) < O(precisão desejada) $\lesssim$ O(d), o que justifica em termos gerais, a limitação adotada para o deslocamento dos snaxels.*

De agora em diante, $\mu$ se referirá à malha contendo células da $EA$. Portanto $\mu$ será uma malha de células quadradas — cada uma contendo $dxd$ pixels — cobrindo uma imagem cujas dimensões, se adicionando pixels extras se necessário, são assumidas serem múltiplas de $d$.

Chama-se de **$\mu$-curva** qualquer linha poligonal $S$ tal que:

**a)** Seus vértices são pontos onde $S$ intersecta as arestas de $\mu$;

**b)** Nenhum destes vértices coincide com um vértice de $\mu$.

Uma $\mu$-curva é dita regular se for uma curva simples e não tiver mais de um vértice numa mesma aresta de $\mu$. Uma T-Snake e a curva projetada $PC_k$ são exemplos de $\mu$-curvas regular e que pode ser não-regular, respectivamente. Quando uma $\mu$-curva regular $S$ corta as quatro arestas de uma célula, essa célula é dita **dupla** em relação a $S$, dado que contem dois segmentos de $S$.

Para representar uma $\mu$-curva $S = [s_i; i = 0, \ldots, I-1]$ é suficiente ter uma forma de representar seus vértices, que são pontos no interior relativo das arestas de $\mu$. Não importa se essa forma não se aplica a um ponto genérico da região da imagem. Escolheu-se representar cada vértice $s_i$ se usando o sistema *Célula — Aresta da célula — Ponto da aresta (CAP)* cujas coordenadas são(veja a figura 4.1):

**a)** A **coordenada de célula** $C_i = C(s_i)$ indica a $\mu$-célula que contém $[s_i, s_{i+1 \ mod \ I}]$;

**b)** A **coordenada de aresta** $E_i = E(s_i)$ indica qual das quatro arestas de $C_i$ contém $s_i$. $E_i$ é um número de dois bits. Usa-se um bit ($B_i^1$) para indicar a direção da aresta — neste trabalho a direção vertical é indicada por $B_i^1 = 0$ e a horizontal pr $B_i^1 = 1$ — e o outro ($B_i^2$) para indicar se a aresta é adjacente ao vértice superior esquerdo ou não. Aproveitamos para definir $(E(s_i))^{-1}$ que é determinada de forma idêntica a $E(s_i)$ só que em relação a célula $C_{i-1}$;

**c)** seja $u_i$ o vértice da $\mu$-aresta contendo $s_i$ de menor coordenada cartesiana na direção da aresta. A terceira coordenada do sistema — ($p_i$), a coordenada do pixel — é a distância entre $s_i$ e $v_i$ expressa em pixels.

A figura 4.1 ilustra as definições de $C(s_i)$, $E(s_i)$ e $(E(s_i))^{-1}$. A razão para adotar o sistema "CAP", ao invés de se usar coordenadas linha-coluna, é que este permite a detecção e o processamento de loops de forma muito mais direta. De fato, as coordenadas $C_i$ e $E_i$ têm de ser computadas de qualquer forma, não importando o sistema que seja usado. Também, como $C_{i+1 mod I}$ pode ser determinado a partir de $C_i$ e $E_i$, $S$ fica plenamente identificada pela sequüencia $((E_i, p_i), i = 0, \ldots, I-1)$ e um simples $C_i$. Desse modo, apesar do sistema *CAP* empregar três coordenadas, ele produz representações de $\mu$-curvas mais compactas que o sistema *linha-coluna*. Observa-se que um sistema em que a aresta da malha que contem cada $s_i$ é identificada por uma única coordenada, não é apropriado à detecção dos loops formados por uma $\mu$-curva. É que alguns desses loops não são detectados por que a curva voltou a uma mesma aresta de $\mu$.

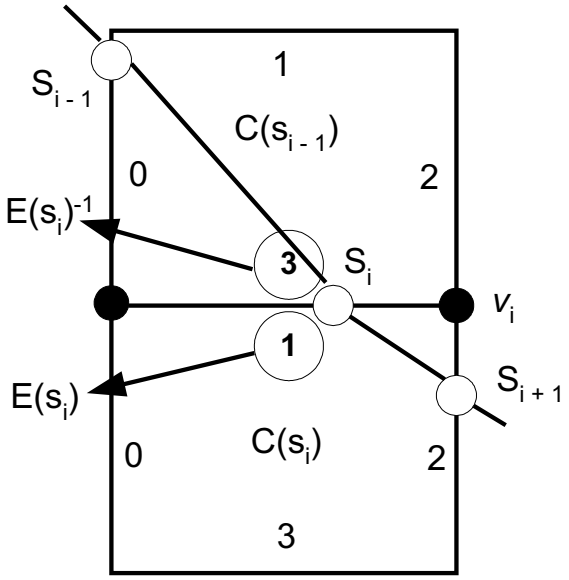


Figura 4.1: Elementos usados no sistema CAP.

Por outro lado usando-se as coordenadas-CAP complica-se levemente a computação dos deslocamentos dos snaxels, mas uma comparação total ainda é favorável a elas. A fim de reduzir o “overhead” imposto à computação do movimento dos snaxels pelo fato de se usar coordenadas-CAP, vários métodos, podem ser adotados. Um deles é descrito abaixo.

**(a) Inicialmente** $d_i = (d_i(0), d_i(1))$, que é $s_{i+1} - s_i$ em coordenadas linha-coluna, é obtido a partir de $(E_k, p_k)$, $k = i, i+1$, através do procedimento abaixo;

**(b) Procedimento:**

**if** $(B_i^1 == B_{i+1}^1)$
  $d_i\,(B_i^1) = p_{i+1} - p_i$
  **if** $(\neg B_i^2)\; d_i\,(\neg B_i^1) = -d$ **else** $d_i\,(\neg B_i^1) = d$
**else**
  **if** $(B_i^2)\; d_i\,(B_{i+1}^1) = p_{i+1} - d$ **else** $d_i\,(B_{i+1}^1) = p_{i+1}$
  **if** $(B_{i+1}^2)\; d_i\,(B_i^1) = -p_i$ **else** $d_i\,(B_i^1) = d - p_i$

**(c) Tendo** $(d_i, i = 0, \ldots, I)$ as forças internas usuais da snake podem ser computadas. Para se obter o campo externo nos snaxels, entretanto, é necessário tê-los como coordenadas linha-coluna que é obtido iterativamente fazendo: $s_{i+1} = s_i + d_i$.

Os vértices da curva transformada, os quais não estão restritos as $\mu$-arestas, são obtidos em coordenadas linha e coluna, mas o processo de geração da curva projetada que é usado,

computa seus vértices já no sistema *CAP* da seguinte maneira:
Seja $\overline{s_i}$ o próximo vértice da curva projetada a ser calculado e suponha que ele pertence ao segmento $[t_{j-1}, t_j]$ da curva transformada. Seja ainda $C_{i-1}$ a coodenada de célula de $s_i$. A aresta $e_i$ de $C_i$ que contém $s_i$ pode ser determinada fazendo-se:

**1)** Compute $\overline{\Delta}_j = (\Delta_j(0), \Delta_j(1)) = t_j - t_{j-1}$ e considere o vértice de $C_{i-1}$ — $z_i$ — dado por $(l_{i-1}, c_{i-1}) + (U(\overline{\Delta}_j))$ onde $(l_{i-1}, c_{i-1})$ são as coordenadas do vértice superior esquerdo de $C_{i-1}$ e $U$ é a função degrau clássica;

**2)** Determinado o lado em que $z_i$ está em relação a reta $\overline{r_i} = s_{i-1} + \lambda\overline{\Delta}_j, \lambda \in \Re$, podemos definir qual das arestas adjacentes a ele deve conter $s_i$. Essa determinação pode ser feita em função do sinal de $\left\langle \overline{\Delta}_j^{\perp}, z_i - s_{i-1} \right\rangle$. Uma vez determinada $e_i$ a determinação de $E_i$ e de $C_i$ é imediata.

Para efeito de identificar a formação de loops e rotulá-los, essas coordenadas são suficientes. A coordenada de pixel de $s_i$ pode também, então, ser obtida determinando-se a intersecção de $\overline{r_i}$ com a reta suporte de $e_i$. É discutível se essa coordenada precisa ser calculada exatamente, mas algum critério na escolha de uma aproximação dela precisa ser utilizado. Simplificações como usar simplesmente o ponto médio de $e_i$ podem dificultar a evolução da T-Snake. Se o deslocamento obtido, empregando-se o campo nesse ponto médio, não for suficientemente grande para que a T-Snake não cruze mais $e_i$, ele voltará a ser repetido e é mesmo possível que a evolução da T-Snake acabe sendo detida nesse ponto.

A alternativa de se re-estruturar os dados da imagem de forma a se armazenar apenas os pixels contidos nas arestas de $\mu$ foi considerada mais trabalhosa porque apenas um percentual desses pixels é ocupado por um snaxel durante toda a evolução da snake e somente para eles se precisa dos dados de imagem.

Se $S = [s_i, i = 1, \ldots, I]$ é uma $\mu$-curva regular, define-se o **vértice externo**(out-vertex) de $s_i$ como o vértice da $\mu$-aresta que contém $s_i$, que está fora da região delimitada por $S$. O **vértice-interno**(in-vertex) de $s_i$ é definido de forma análoga. Daqui por diante se notará, $v_i$ para se referir ao vértice externo de $s_i$.

Tmabém, daqui por diante vai se assumir que os vértices $s_i$ estão ordenados no sentido anti-horário. Tendo-se fixado a direção em que $S$ é atravessada, a determinação de $v_i$ entre os dois vértices da aresta de $s_i$ torna-se uma função de $E_i$ somente, que é definida como está indicado na figura 4.2.

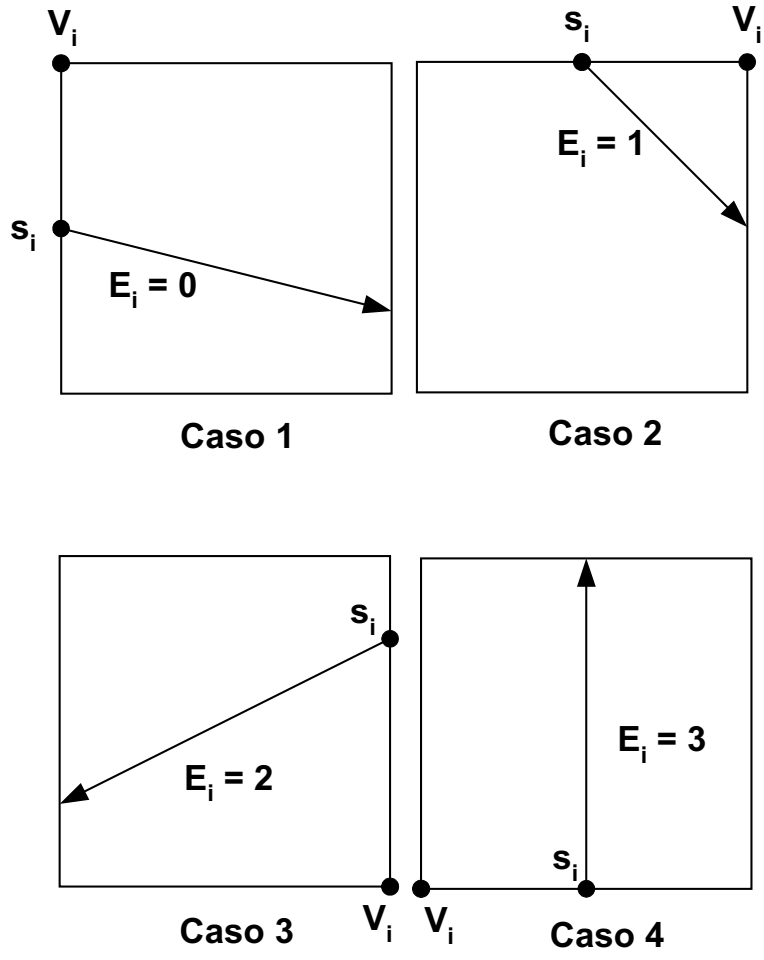


Figura 4.2: $v_i$ em função de $E_i$.

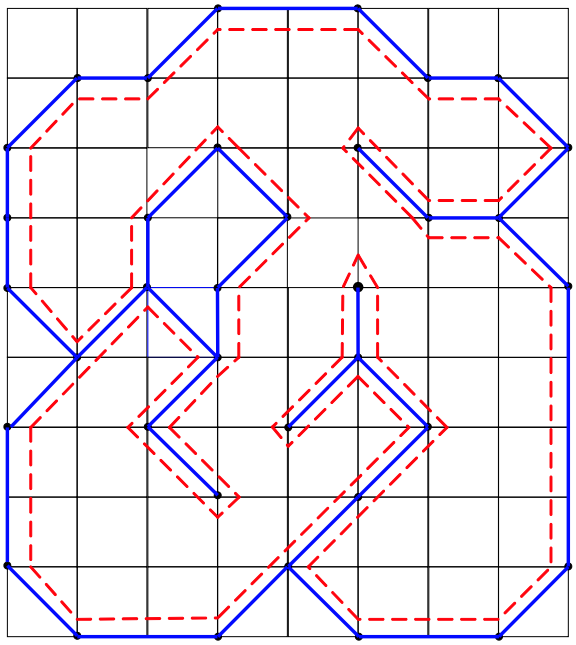

Figura 4.3: $\mu$-dilatação — representada em linha cheia — da curva em linha tracejada.

A $\mu$-**Dilatação** de $S$ — $\mu D(S)$ — é a curva obtida pela substituição de cada $s_i$ por $w_i = v_i + \epsilon.(s_i - v_i)$, onde $\epsilon$ é um real positivo qualquer $< 1/2d$. Isso faz com que $w_i$ e $v_i$ se situem na área ocupada por um mesmo pixel, assumindo que um pixel tem dimensões unitárias. A diferença é que enquanto $v_i$ é uma das extremidades de $e_i$, $w_i$ pertence ao seu interior relativo e portanto pode ser unívocamente representado em coordenadas *CAP*, que são as mesmas de $s_i$, exceto pela coordenada de pixel, que é feita 0 ou $d$ conforme o valor de $E_i$ — 0 se $E_i = 0$ ou 3 e $d$ se $E_i = 1$ ou 2. $v_i$ é substituído na definição de $\mu D(S)$ por $w_i$ para torná-la uma $\mu$-curva, evitando-se que ela tenha vértices repetidos. Sem considerar essa perturbação, $\mu D(S)$ é uma curva composta pelas $\mu$-arestas e diagonais das $\mu$-células que formam seu contorno externo — o *$\mu$-fecho de $S$* — e eventualmente, por loops internos conectados entre si e ao fecho por arestas duplas —isto é, arestas que são atravessadas em ambos os sentidos quando $\mu$D(S) é percorrida — além de ramificções formadas por essas arestas. A figura 4.3 mostra a $\mu$-Dilatação de uma curva com uma topologia mais complexa.

Duas $\mu$-curvas são ditas **equivalentes** se elas cruzam a mesma sequëncia de $\mu$-arestas, como mostrado na figura 4.4.

Sejam $[v_i, i = 0, \ldots, N-1]$ e $[v'_i, i = 0, \ldots, N-1]$ as representações de duas $\mu$-curvas equivalentes $S$ e $S'$ como poligonais. Se $v_i$ e $v'_j$ pertencem a mesma aresta, então dizemos que o vértice $v'_j$ é o correspondente ao vértice $v_j$ em $S'$, $i, j = 0, \ldots, N-1$. Uma $\mu$-

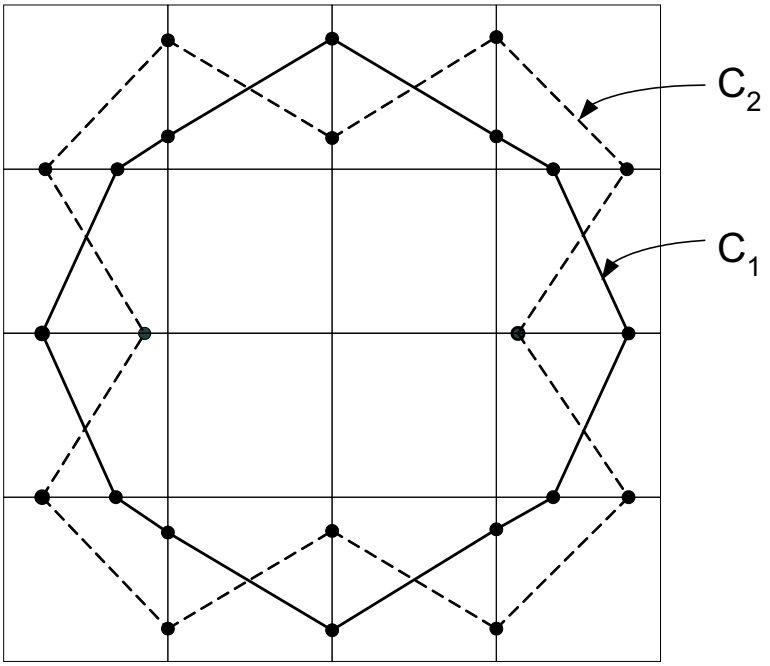


Figura 4.4: As curvas $C_1$ e $C_2$ são ditas $\mu$-equivalentes.

curva estendida difere de uma $\mu$-curva pelo fato de ter vértices no interior das células além daqueles localizados em suas intersecções com as arestas da malha. Os conceitos de equivalência entre $\mu$-curvas e de correspondência entre vértices de $\mu$-curvas equivalentes podem ser levados para o caso de $\mu$-curvas estendidas, meramente ignorando os vértices que não pertencem a arestas da malha.

### 4.1.2 $\mu$-Intersecções

Seja $S_1 = (s_{1,m}; m = 0, \ldots, M)$ e $S_2 = (s_{2,n}; n = 0, \ldots, N)$ duas $\mu$-curvas e assuma que $T_1 = [s_{1,i}, \ldots, s_{1,i+k}]$ e $T_2 = \left[s_{2,j}, \ldots, s_{2,j+k}\right]$ cortam exatamente as mesmas células.Diz-se que $S_1$ $\mu$-**intersecta** $S_2$ ao longo de $T_1$ se e somente se qualquer $\mu$-curva equivalente a essa linha poligonal cruza $T_2$. É importante observar que, se $S_1$ $\mu$-**intersecta** $S_2$ ao longo de $T_1$ então ao se remover as arestas extremas de $T_1$ e $T_2$ obtêm-se $\mu$-curvas equivalentes, as quais para os propósitos deste trabalho, pode-se assumir que são disjuntas entre si. Nesse caso, ou os segmentos iniciais ou os finais de $T_1$ e $T_2$ e apenas aqueles de um desses pares, devem se intersectar com mostra a figura 4.5. Observe que se $T_1$ e $T_2$ entram em uma célula através da mesma aresta, então para $k = 1, \ldots, k-1$, $s_{1,i+k}$ e $s_{2,j+k}$ estão na mesma aresta, caso contrário $s_{1,i+k}$ pertence a mesma aresta que $s_{2,j+(K-k)}$. Entretanto, se essas duas curvas devem ser parte da curva projetada e considerando-se que ela é gerada aplicando-se um mapeamento contrátil a uma $\mu$-curva regular, no primeiro

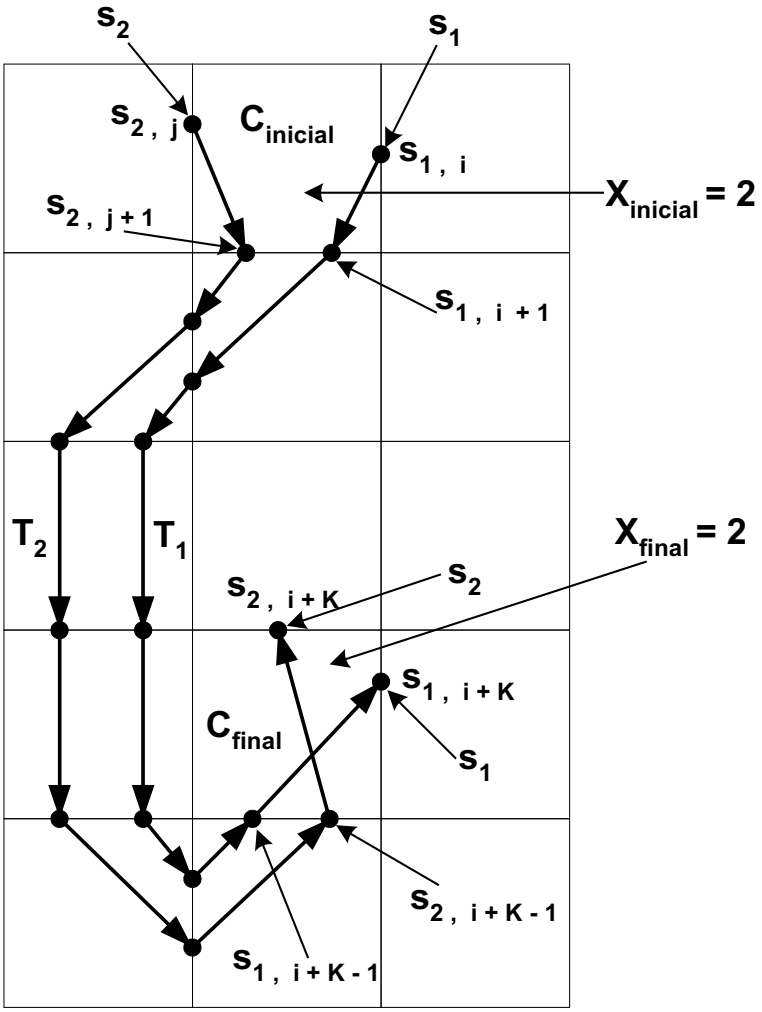


Figura 4.5: Se há uma $\mu$-intersecção em $T_1$ então substituindo, se necessário, $T_1$ e $T_2$ por curvas $\mu$-equivalentes é possível fazer com que elas interceptem apenas em seus segmentos finais, ou iniciais.

caso sempre haverá uma $\mu$-interesecção(ver figura 4.6). No segundo, entretanto, onde o conjunto formado por $T_1$ e $T_2$ é denominado um **gargalo** de $PC_k$, essa intersecção pode existir ou não. Para mostrar como se determina de forma computacionalmente eficiente se ela ocorre ou não nesse segundo caso, precisamos, entretanto, fazer antes algumas considerações. Sejam $s_1$, $s_2$ e $s_3$ três pontos pertencentes a arestas diferentes de uma mesma célula $C$. Defina a função $X$ da seguinte maneira:

- $X(s_1, s_2, s_3) = m$ se $s_m$ é o primeiro entre $s_1$ e $s_2$ a ser atingido quando se percorre o contorno da célula $C$ no sentido horário a partir de $s_3$. Como se consiedera que $s_1, i = 1, 2, 3$ estão em arestas diferentes a função $X$ não depende da posição desses pontos mas simplesmente das arestas da malha a que eles pertencem.

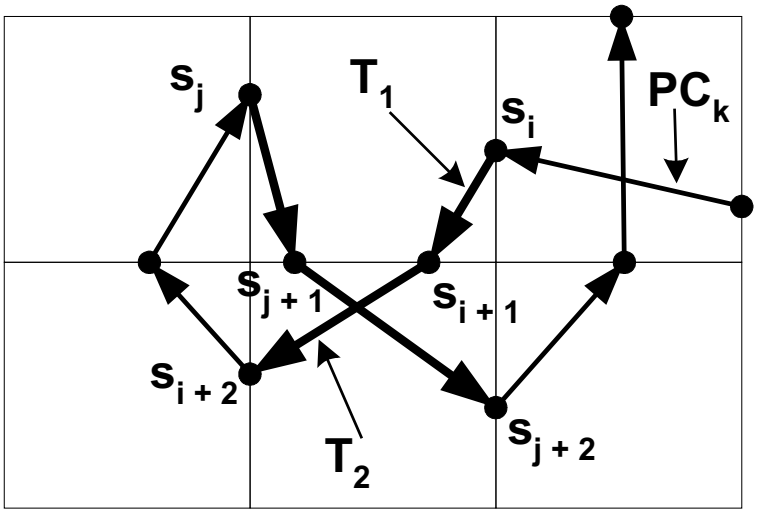


Figura 4.6: Exemplo onde $s_{i+K} = s_{j+K}$, no caso apenas para $K = 1$.

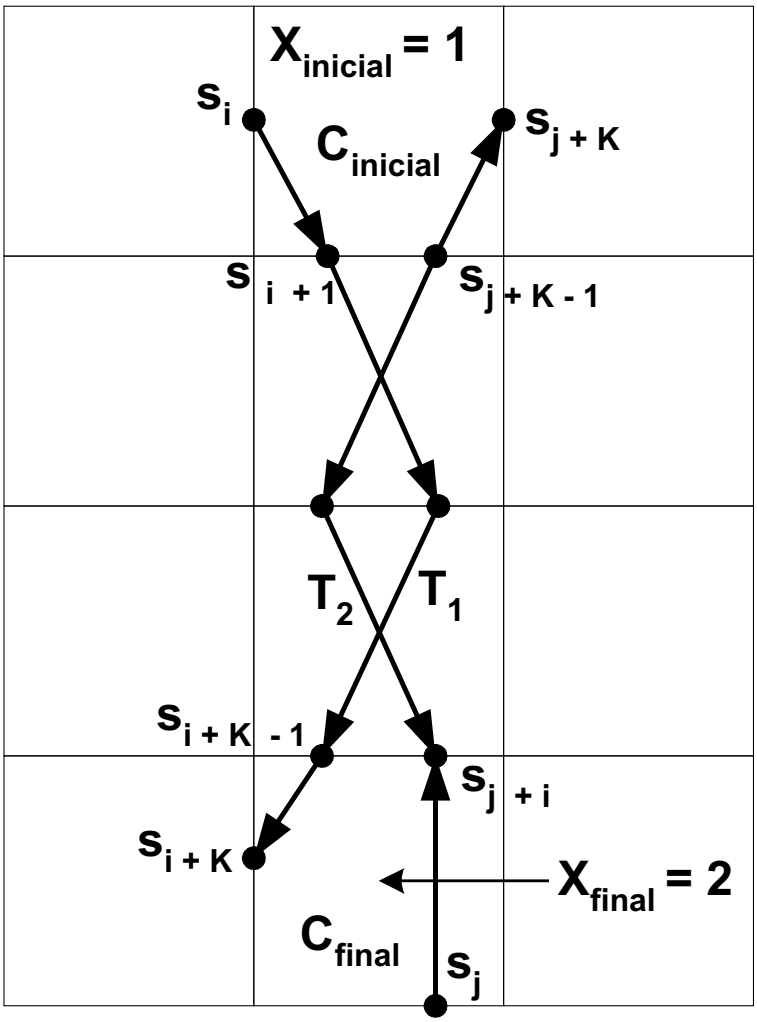


Figura 4.7: Exemplo onde não ocorre uma $\mu$-intersecção, apesar de $T_1$ e $T_2$, efetivamente, se cortarem.

Se $s_1, s_2, s_3$ são vértices de $\mu$-curvas a aresta contendo $s_i, i = 1, 2, 3$ fica perfeitamente identificada por $E'(s_i)$, definida da seguinte maneira:

- $E'(s_i) = E(s_i)$, se a $\mu$-curva contendo $s_i$ está entrando em $C$ em $s_i$ e $E'(s_i) = E(s_i)^{-1}$ se ela estiver saindo de $C$ por $s_i$. O valor $X(s_1, s_2, s_3)$ pode então se computado pelo seguinte algoritmo, onde $E'(s_i)$, $i = 1, 2, 3$ é representado por $E'_i$. A figura 4.8 mostra os 6 casos possíveis para um valor fixo de $E'_3$.

  **Function** $X(s_1, s_2, s_3)$
  **if** $(E'_1 == E'_3 \ mod\ 2\ )$ **do**
    **if** $((E'_2 == (E'_3 + 1)$ **mod** $4)$ $X = 2$; *(Caso 4)*
    **else** $X = 1$; *(Caso 3)*
  **else**
    **if** $(E'_1 == (E'_3 + 1)$ **mod** $4)$ $X = 1$; *(Casos 1 e 2)*
    **else** $X = 2$; *(Casos 5 e 6)*

Voltando a questão de verificar se existe uma $\mu$-intersecção de $S_1$ e $S_2$ em $T_1$, quando $T_1$ e $T_2$ formam um gargalo, considere, inicialmente, os vértices que definem as arestas extremas — $[s_{1,i}, s_{1,i+1}]$ e $\left[s_{2,j+K-1}, s_{2,j+K}\right]$ — de $T_1$ e $T_2$, respectivamente, as quais estão contidas numa mesma célula — $C_{inicial}$. Defina então, $X_{inicial} = X\left(s_{1,i}, s_{2,j+k}, s_{1,i+1}\right)$. Equivalentemente, poderíamos ter $s_{2,j+K-1}$ em lugar de $s_{1,i+1}$ dado que ambos pertencem a uma mesma aresta. De forma análoga, considerando os vértices das outras arestas extremas de $T_1$ e $T_2$ —$[s_{1,i+K}, s_{1,i+K-1}]$ e $[s_{2,i}, s_{2,i+1}]$ — que também estão contidas numa mesma célula — $C_{final}$ — defina: $X_{final} = X(s_{1,i+K}, s_{2,i}, s_{1,i+K-1})$.

Conforme se pode observar nas figuras 4.5 e 4.7, $s_1$ e $s_2$ terão uma $\mu$-intersecção em $T_1$ se e só se, $X_{initial} = X_{final}$ não importa quais sejam as arestas de $C_{inicial}$ e $C_{final}$ cortadas por elas. Independente do número de arestas que compõem o gargalo, pode-se, portanto, identificar a existência de uma $\mu$-intersecção realizando um número fixo de testes — 5, para ser exato:

- 2 no computo de $X_{initial}$, 2 para determinar $X_{final}$ e finalmente a comparação entre os dois.

A existência de gargalos na curva projetada, entretanto, é absolutamente eventual conforme ficou demonstrado no conjunto de testes realizados. Ainda em relação aos gargalos de $PC_k$, a seguinte terminologia será empregada:

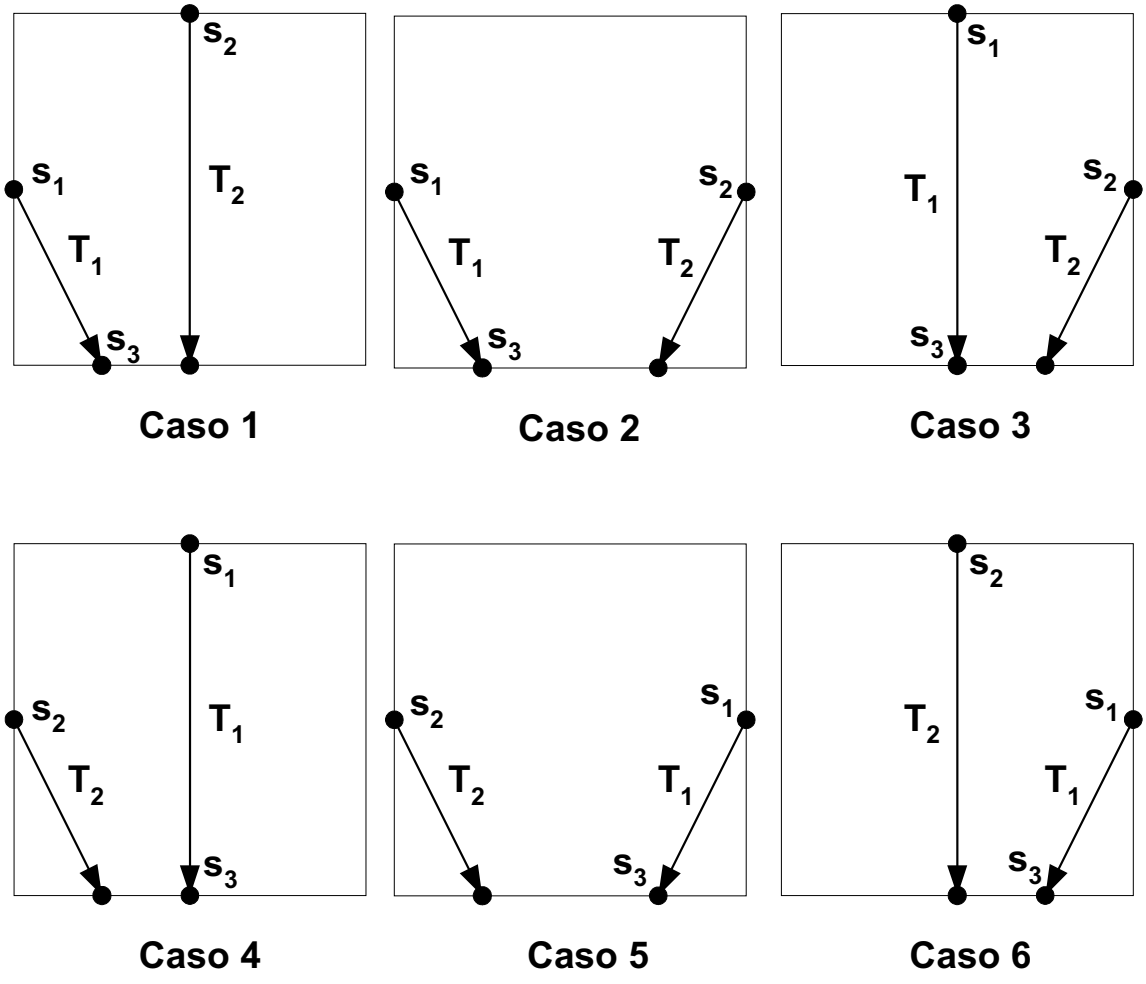


Figura 4.8: Casos a serem considerados na determinação de $X$ quando $E_3' = 3$.

- As curvas $T_1$ e $T_2$, definidas anteriormente, que formam um gargalo $G$ de $PC_k$ serão conhecidas como os **ramos** de $G$ sendo o ramo inicial aquele que é gerado antes, ou seja o formado por vértices de menor índice. O outro será referido como o ramo final.

### 4.1.3 $\mu$-Whiskers e seu tratamento

Dada uma $\mu$-curva $S$=[ $s_i$, $i = 0, \ldots, m$], um $\mu$-whisker de $S$ com tamanho $n$ consiste na curva formada por $2n$ vértices consecutivos de $S$, $s_j$, $s_{j+1}$,..., $s_{j+2n-1}$, tais que $s_{j+k}$,..., $s_{j+2n-1-k}$ estão na mesma aresta da malha, para $k = 0, \ldots, n-1$. O nome $\mu$-whisker deriva do fato de que se identificarmos num único ponto de cada $\mu$-aresta cruzada por $S$, todos os vértices de $S$ contidos nela, então um $\mu$-whisker de $S$ se tornará um whisker do conjunto constituído pela curva resultante dessa identificação e pela região delimitada por ela. A figura 4.9 mostra o exemplo de um $\mu$-whisker. $\mu$-whiskers precisam, obviamente, ser retirados dos loops abertos de $PC_k$ para que as loop-snakes do próximo estágio, tenham a propriedade de possuir um único vértice por aresta da malha. Para evitar que falsos loops sejam determinados, entretanto, eles devem ser removidos dos loops fechados também. Se primeiro geramos $PC_k$ para depois percorre-la para identificar os seus loops, então, assim que a tivermos inteiramente, podemos eliminar todos os seus $\mu$-whiskers, por meio de um procedimento simples como o dado a seguir. Nesse procedimento se assume que $PC_k$ tem $m$ vértices, sendo $v_0$ um deles, e é armazenada numa lista duplamente encadeada.

{ $v = v_0$
**For** ($j = 0$ *to* $m - 1$) **do:**
**if** ($E(v) = (E(v.next))^{-1}$)
{($v.previous$).$next$ = ($v.next$).$next$
$v = v.previous$}
**else**
$v = v.next$ }

Mas assim teríamos de percorrer a curva $PC_k$ uma vez no próprio processo de sua geração, uma vez para eliminar os $\mu$-Whiskers e uma terceira para então, identificar loops e fazer as mudanças topológicas, ou seja, 3 vezes. Como para cada aresta da malha, que é re-cruzada por um $\mu$-whisker, temos que uma nova célula está sendo re-visitada — o que é condição essencial também para a formação de um loop — podemos, em princípio,

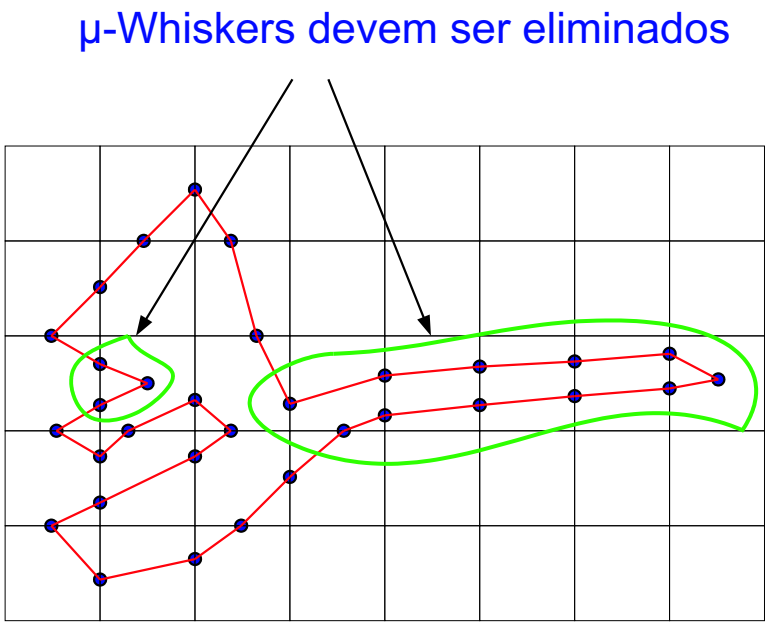


Figura 4.9: Esquilo: $\mu$-whiskers devem ser eliminados.

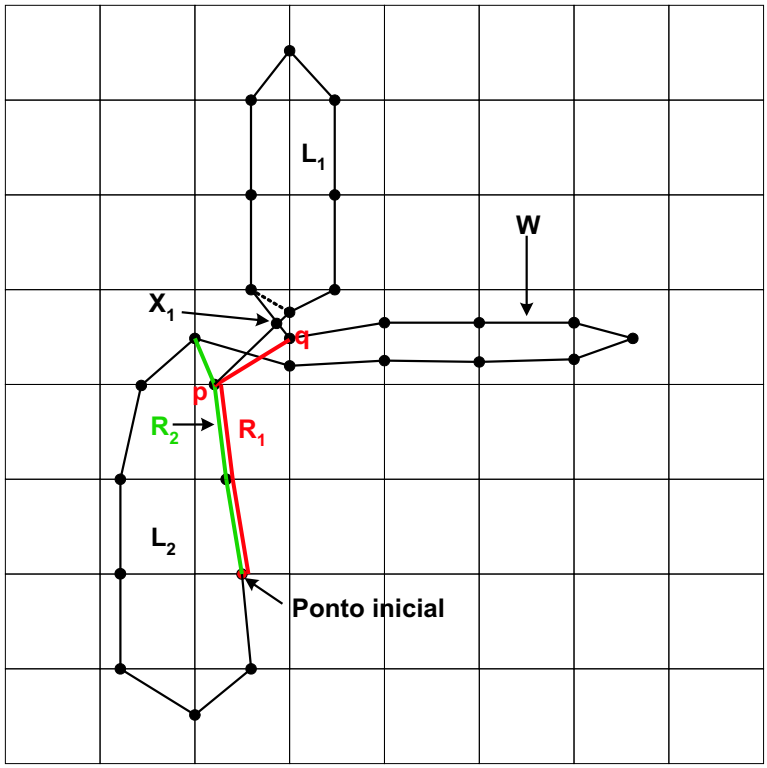


Figura 4.10: Possível erro de rotulação determinado pela eliminação de um $\mu$-whisker.

fazer tanto a procura por loops como a eliminação de whiskers num único percurso de $PC_k$. Mais ainda, podemos tentar fazer isso a medida que os vértices de PCk vão sendo gerados, de forma que ela seja percorrida uma única vez. Essa estratégia é perfeitamente factível, mas tem, entretanto, um problema, que é ilustrado na figura 4.10. No exemplo dessa figura, no momento em que o loop $L_1$ é encontrado se identifica que existe uma $\mu$-intersecção — $X_1$ — entre ele e o resto da $PC_k$ gerada até esse instante, o qual é identificado por $R_1$. Posteriormente, entretanto, com a eliminação do $\mu$-whisker $W$, que só é descoberto depois, essa curva restante passa de $R_1$ para $R_2$, que não tem uma $\mu$-intersecção entre ele e $L_1$. Como não há mais células repetidas depois da eliminação de $W$, o rótulo dos loops $L_1$ e $L_2$ — que é formado ao final do processo — deve ser o mesmo. Ocorre que o procedimento empregado neste trabalho para a rotulação dos loops, obtém o label $L_2$ em função do de $L_1$. Se ele considerar, erroneamente, que existe $\mu$-intersecção entre eles, então, dado que eles são disjuntos, ele indicará que o label de $L_2$ deve ser diferente do de $L_1$. Isso é o que acontece, no caso do exemplo, se ele verificar a possibilidade de existência dessa $\mu$-intersecção no momento que $L_1$ é formado, assumindo que o segmento $[p, q]$ é parte de $L_2$, o que é falso. Operando dessa forma ele dará, então, a $L_2$ um rótulo errado.

A solução para esse problema é, entretanto, bastante simples: Não verificar se a $\mu$-intersecção $X_1$ existe no momento em que se gera $L_1$, mas só fazer isso quando o próximo loop depois de $L_1$ for criado. A partir desse momento, a eliminação de $\mu$-whiskers não determinará que o label de um loop que é obtido em função do de $L_1$, o seja erradamente. Entretanto, para concentrar o foco na identificação e tratamento dos loops , nos procedimentos que devem ser aplicados quando $PC_k$ revisita uma célula, descritos no capítulo 6, não mencionamos essa medida destinada a eliminar a influência da retirada de $\mu$-whiskers na rotulação de loops. Esses procedimentos, conforme se verá, já são suficientemente elaborados sem ela.

### 4.1.4 Mapeamentos Elementares

Chamamos de **mapeamentos elementares** (abreviadamente: *e-map ou e-m*) às transformações que levam a T-snake de um determinado estágio($k$) ou sua $\mu$-dilatação nessa última curva ou naquelas geradas no processo de atualização da snake efetuado nesse estágio. Especificamente vamos fazer uso de quatro desses mapeamentos:

**1)** Aquele que leva a T-snake do estágio($S_k$) na curva *fisicamente transformada* $TC_k$. Vamos nos referir a esse mapeamento por $T_k$.

**2)** O que leva $S_k$ diretamente na *curva projetada* $PC_k$, que será referido por $P_k$.

**3)** $D_k$ , o mapeamento que leva $S_k$ em sua $\mu$-dilatação.

**4)** $\Gamma_k$, o que leva a $\mu$-dilatação de $S_k$ em $PC_k$. Vamos usar $\Gamma_k$ por que é mais fácil obter algumas propriedades desejáveis para ele do que para $P_k$.

$T_k$ e $D_k$ podem ser definidos de maneira óbvia devido a correspondência entre os vértices de $S_k$ e os de $TC_k$ e $\mu$-$D(S_k)$, respectivamente. $P_k$ e $\Gamma_k$, entretanto não tem uma maneira natural de serem definidas. O objetivo inicial é fazê-las serem mapeamentos contínuos, lineares por partes, e que associam pontos que não estão longe um do outro. Esse último requisito ficará melhor definido depois que definirmos mapeamentos $\mu$-limitados na seção 4.1.6.

Abaixo indicamos como obter uma formulação para o mapeamento $\Gamma_k$ que é contínua, e faz com que $\Gamma_k(D_k(s))$ e $T_k(s)$ estejam na mesma célula $\mu$. Tendo $\Gamma_k$ podemos obter $P_k$ fazendo $P_k = \Gamma_k \bullet D_k$.

Na descrição desse algoritmo $v_{k,i}$ vai se referir ao out-vértice de snaxel $s_{k,i}$ de $S_k$. Além disso, daqui por diante, $DP_k$ representará a $\mu$-dilatação de $S_k$. Lembramos que cada $v_{k,i}$ está numa aresta diferente da malha e portanto não há coincidências entre eles.

**Passo I** para cada vértice $r_j$, $j = 0, \ldots, J-1$ de $PC_k$ encontrado que não seja a imagem por $T_k$ de um snaxel, execute o seguinte:

Seja $[T_k(s_{k,i}), T_k(s_{k,i+1})]$ a aresta de $TC_k$ onde $r_j$ está e $v^*$ o último ponto de quebra de $\Gamma_k$ inserido em $[v_{k,i}, v_{k,i+1}]$. Crie, então, um novo ponto de quebra — v — em $(v^*, v_{k,i+1})$ e faça $\Gamma_k(v) = r_j$;

**Passo II** para cada aresta $e_j = [r_j, r_{j+1}]$ de $PC_k$ percorra $DP_k$ de $v_j$ até $v_{j+1}$. Se um ponto $y$, onde a reta suporte de $e_j$ intersecta $DP_k$, é alcançado, crie um novo ponto de quebra de $\Gamma_k$ em $y$ e faça $\Gamma_k(y)$ ser, entre $r_j$ e $r_{j+1}$, o ponto mais próximo de $y$. Até dois pontos como $y$ podem ser encontrados no percurso de $e_j$ como acontece no caso indicado na figura 4.11;

**Passo III** para cada par $(z_l, z_{l+1})$ de pontos de quebra de $\Gamma_k$, consecutivos, definidos nos passos anteriores execute:

Faça $w = \Gamma_k(z_l)$ e percorra $DP_k$ de $z_l$ até $z_{l+1}$. Para cada vértice $v_m$ de $DP_k$ alcançado execute:

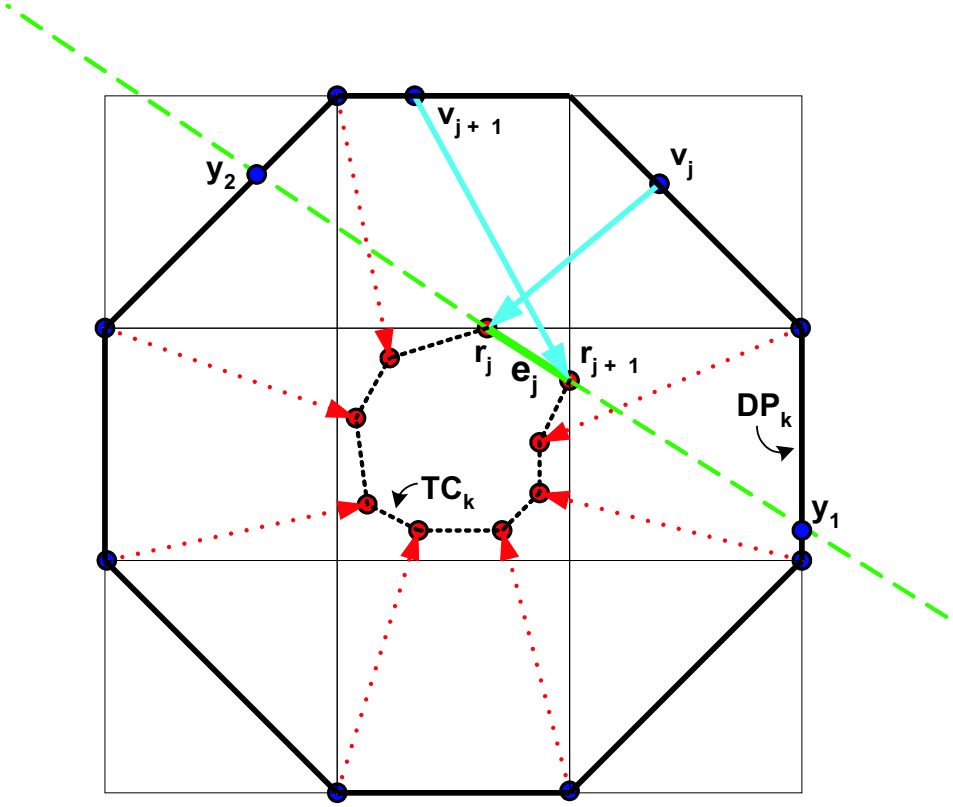


Figura 4.11: Dois pontos de intersecção entre $DP_k$ e a reta suporte.

Faça $\Gamma_k(v_m)$ ser um ponto $w'$ qualquer em $(w, \Gamma_k(z_{l+1}))$ e atualize $w$ fazendo $w = w'$;

**Passo IV** tendo-se definido $\Gamma_k$ em todos os seus pontos de quebra e nos vértices de $DP_k$, estenda sua definição para toda a poligonal $DP_k$ por interpolação linear.

Observe que o exemplo dado na figura 4.11 que se os vértices originais de $DP_k$ que devem ser levados em pontos de $[r_j, r_{j+1}]$ estão em ambos os lados da reta suporte dessa aresta, o mapeamento $\Gamma_k$ obtido pelo método acima não é 1–1. Nessas condições, entretanto, fazer $\Gamma_k$ um homeomorfismo obrigaria a que necessariamente, raias de vértices consecutivos de $DP_k$ se cruzassem e isso é totalmente indesejável para os propósitos desse trabalho. Ao contrário, aqui vai se visar, exatamente, obter mapeamentos, como os definidos na seção seguinte onde esses cruzamentos não ocorrem.

Um mapeamento definido numa $\mu$-curva regular $D$ é dito canônico se sua imagem é uma $\mu$-curva e se ele pode ser reproduzido, aplicando-se o algoritmo acima com $M(D)$ no lugar de $PC_k$. Além dessa designação, vamos usar os seguintes conceitos relativos a mapeamentos elementares e seus componentes:

**1)** Dado um quadrilátero de varredura $Q_i = [v_i, v_{i+1}, M(v_{i+1}), M(v_i)]$ de um mapeamento M, então sua aresta $[v_i, v_{i+1}]$ será referida como a base de $Q_i$ e $[M(v_{i+1}), M(v_i)]$, como o seu topo.

**2)** Dois mapeamentos $M$ e $M'$, que levam uma $\mu$-curva regular $D$ em $\mu$-curvas são ditos $\mu$-equivalentes se :

**1)** $M(D)$ e $M'(D)$, as imagens de $M$ e $M'$, são $\mu$-equivalentes.

**2)** Se $v$ e $v'$ são vértices correspondentes de $M(D)$ e $M'(D)$, respectivamente, então $M^{-1}(v)$ e $M'^{-1}(v')$ estão na mesma aresta de $D$.

### 4.1.5 Mapeamentos Elementares: Ideais e Adequados

Seja $M$ um *e-m* aplicado a uma linha poligonal fechada $S = [s_0, \ldots, s_i, \ldots, s_N = s_0]$ que gera a curva $S' = M(S) = [s'_0, \ldots, s'_i = M(s_i), \ldots, s'_N = s'_0]$. Seja ainda $Q_i$ o quadrilátero de varredura $[s_i, s'_i, s'_{i+1}, s_{i+1}, s_i]$ e $E_i$ a linha poligonal $[s_i, s_{i+1}, s_{i+2(modN)}]$, $i = 0, \ldots, N-1$. Para lidar com a possibilidade de repetição de vértices de $S'$ defina $P_i^+ = \bigcup\{Q_j \mid s'_i = s'_{(i+1) \bmod N} = s'_{(i+2) \bmod N} = \ldots = s'_{(j-1) \bmod N} = s'_j\}$ e $P_i^- = \bigcup\{Q_j \mid s'_i = s'_{(i-1) \bmod N} = s'_{(i-2) \bmod N} = \ldots = s'_{(j+1) \bmod N} = s'_j\}$. Se não há quadriláteros de varredura degenerados $P_i^+ = Q_i$ e $P_i^- = Q_{i-1}$. $P'_i$ se referirá ao conjunto $P_i^+ - (r_i \stackrel{\Delta}{=} [s_i, s'_i])$. Diz-se que $M$ é **ideal** em $s_i$, $i = 1, \ldots, N$ se e somente se:

**(I)** Linhas poligonais $E_i$ e $M(E_i) \subseteq S'$ não se cruzam.

**(II)** $P_i^-$ e $P'_i$ são disjuntos um do outro.

Mapeamentos elementares **adequados** em $s_i$, $i = 1, \ldots, N$ são definidos através da substituição da condição "II)" acima por:

- **II')** $r'_i = (s_i, s'_i)$, $i = 0, \ldots, N-1$, é interior a união de $Q_i$ e $Q_{i-1}$.

$M$ será dito ideal se for ideal em todos os vértices de $S$. Um mapeamento adequado é definido de maneira análoga. A figura abaixo apresenta mapeamentos elementares que não são adequados(figura 4.12(a, b)), adequado mas não ideal(figura 4.12(c)) e um ideais(figura 4.12(d, e)). Um mapeamento ideal certamente é adequado uma vez que $Q_{i-1}$ e $Q_i$ são aderentes a $r_i$, e nesse caso, estão em lados diferentes desse segmento. Um mapeamento adequado $M$ não gera quadriláteros reversos e também será ideal se e somente se $\forall i$, $s'_i$ está sobre a borda externa de $P_i^- \cup P_i^+$. $M$ pode ser transformado em um movimento

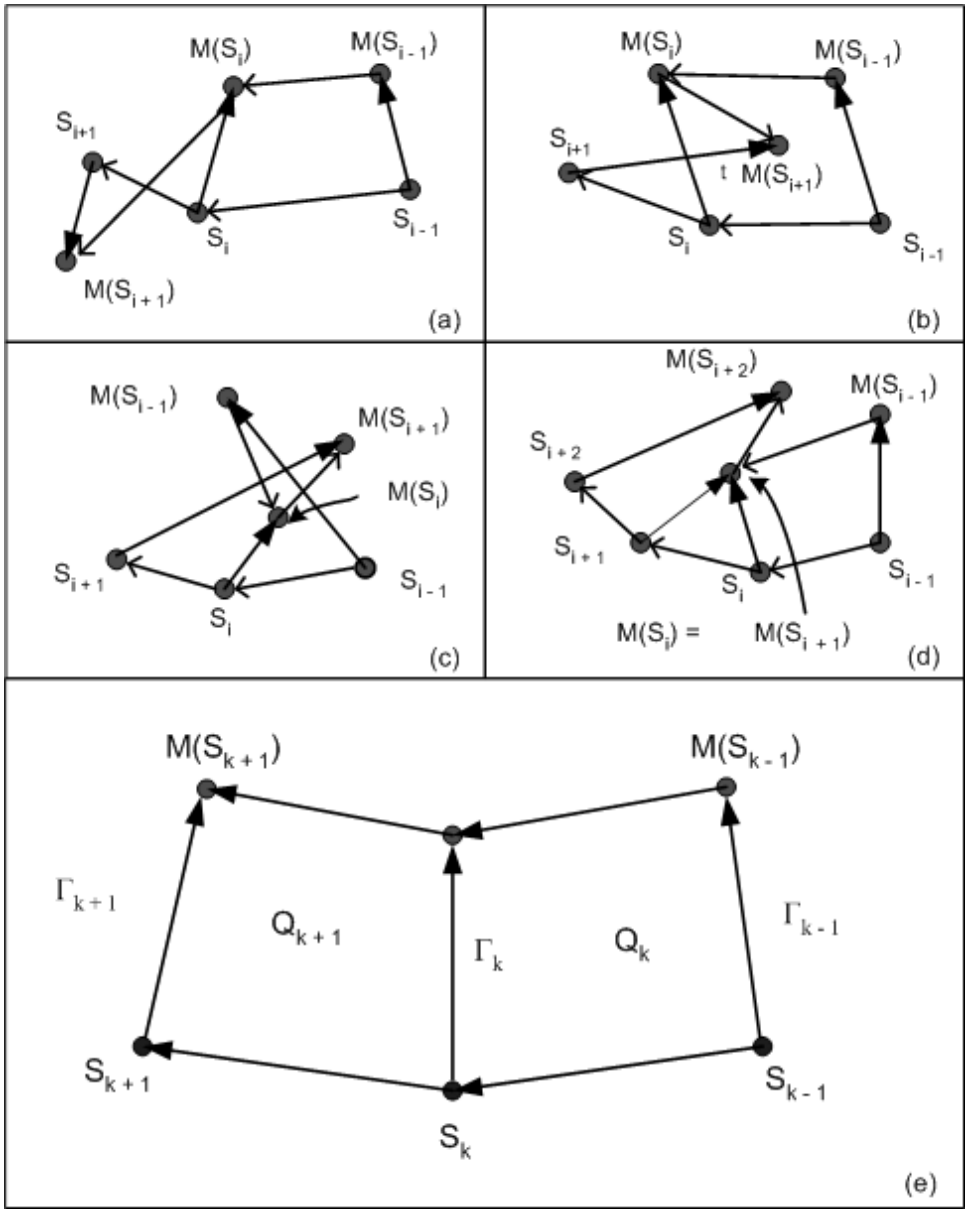


Figura 4.12: Mapeamentos adequados, adequados mas não ideais, ideais.

ideal ($M'$) através da simples introdução para cada $i$, onde a condição $II$ não funciona, e quando $Q_{i-1}$ é não degenerado de novos vértices, $v_i$ em $S$ e $v'_i = M'(v_i)$ em $S'$. $v_i$ pode ser qualquer ponto sobre $(s_{i-1}, s_i)$ enquanto que $v'_i$ deve ser escolhido de modo que:

- **A)** Os quadriláteros $V_1 = [s_{i-1}, s'_{i-1}, v'_i, v_i]$ e $V_2 = [v_i, v'_i, s'_i, s_i]$ são não degenerados e contidos em $Q_{i-1}$.

- **B)** $V_2$ e $P'_i$ são disjuntos.

Obter esses quadriláteros sempre é possível já que $Q_{i-1}$, que será substituído por eles, não é reverso. Além do mais, $v_i$ e $v'_i$, podem estar arbitrariamente próximos de $s_i$ e $s'_i$, respectivamente, de modo que o mesmo acontece entre as curvas transformadas antes e depois da inserção de $v_i$ e $v'_i$.

A condição $B$ implica que $II$ deve valer em $s_i$, pois, $V_2$ se torna $P_i^-$. A condição $A$ acarreta que $II$ também vale em $v_i$ pois se $V_1 - [v_i, v'_i]$ e $V_2$ estão ambos em $Q_{i-1}$, então eles serão disjuntos um do outro. Se $II$ valer em $s_{i-1}$ antes da inserção, também valerá depois porque o $P_{i-1}^+$ depois da inserção é um subconjunto do que era antes. Como $V_1$ e $V_2$ não são nem reversos nem degenerados, $I$ valerá em $s_{i-1}$ e $v_i$. Obviamente, essa condição continuará valendo em $s_i$ pois $M(e_i)$ não foi alterada. Assim $M'$ será ideal tanto em $v_i$ como em $s_i$ e se $M$ já é ideal em $s_{i-1}$, o mesmo acontecerá com $M'$.

Como se pode ver na figura 4.13, a inserção de $v'_i$ faz com que $S'$ tenha um loop contendo $[s'_i, v'_i]$, que pode ser infinitamente pequeno. Na seqüência usar-se-á esse fato, mas considerando-o do ponto de vista oposto, conforme apresentado abaixo:

- **Afirmação 1** Para cada mapeamento adequado $M : S \longrightarrow I$ existe um mapeamento ideal $M'$ tal que $M(S)$ é topologicamente equivalente a curva obtida ao se tirar de $M'(S)$ um conjunto de loops todos eles disjuntos uns dos outros e do restante da curva.

### 4.1.6 Mapeamentos limitados pela malha $\mu$

Seja $s$ qualquer ponto sobre a linha poligonal $S$ e $F_s$, a faceta de $\mu$ de menor dimensão contendo $s$. Chame $N_s$ à união de todas as células que são adjacentes a um vértice de $F_s$. Então, um mapeamento $M$ será dito limitado pela estrutura da malha $\mu$ ou, de forma curta, $\mu$-limitado, se e somente se, $\forall s \in S$, $M(s)$ está no interior de $N_s$. Se $\mu$ é uma triangulação $J1$, como é usado no modelo de T-snakes, $N_s$ consiste de 13, 10 e 6 células supondo que $s$ está respectivamente, no interior de uma célula, no interior relativo de uma aresta ou

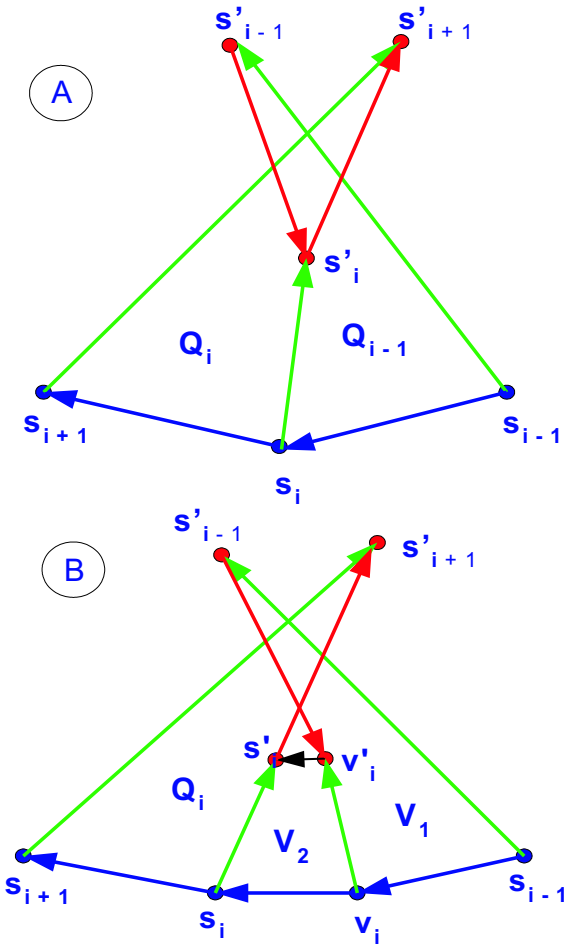


Figura 4.13: Adequado: ideal por meios de uma leve perturbação.

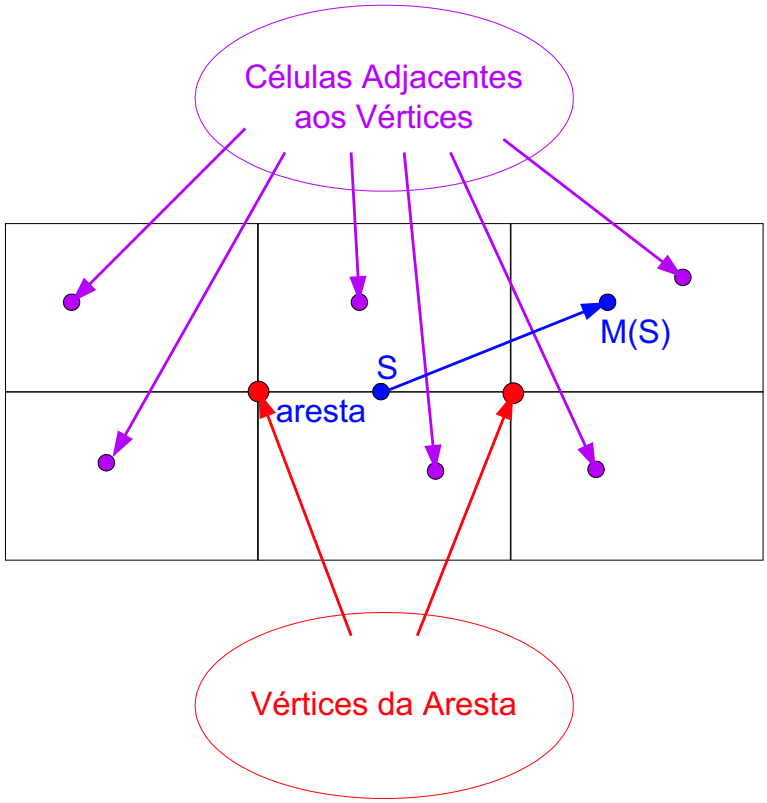


Figura 4.14: Mapeamentos $\mu$-limitados.

coincide com um vértice de $\mu$. Se $\mu$ é uma malha quadrada simples como se assume no caso das loop-snakes, esses números se reduzem a 10, 6 e 4, respectivamente 4.14. Seja $d\mu$ o comprimento de uma aresta de $\mu$. É claro que, se o tamanho de todos as raias de $M$ é menor do que $d\mu$ então $M$ será $\mu$-limitado.

### 4.1.7 A finalidade de se definir essas classes de mapeamentos

As classes de mapeamentos elementares introduzidos acima têm propriedades que simplificam o processo de atualização de uma snake topológica em cada estágio de sua evolução:

**I** A definição de um mapeamento adequado estabelece que nenhum segmento do contorno da região varrida por ele — isto é, da faixa determinada pela união de seus quadriláteros de varredura — está contida numa de suas raias como acontece no exemplo da figura 4.15. Isso significa que no processo de evolução de uma snake topológica que se contrai, onde todos os *e-ms* $P_k$ ou $\Gamma_k$, são adequados a fronteira entre os conjuntos de pontos já percorridos e ainda por serem explorados em um dado estágio, fica totalmente contida na regularização da curva transformada obtida nesse estágio;

**II** Agora, considere que se percorre essa curva, determinando um novo loop a cada vez que se volta a um ponto já visitado que não está num loop, já achado anteriormente. Os loops assim obtidos são todos simples e formarão uma partição da curva. Se $P_k$ ou $\Gamma_k$, usados, além de adequados forem também $\mu$-limitados então a região limitada por qualquer um desses loops ou foi totalmente explorada ou está totalmente inexplorada. De acordo com o caso, rotula-se o loop como *explorado* ou *fechado* no primeiro caso e *inexplorado* ou *aberto* no segundo;

**III** Se $P_k$ ou $\Gamma_k$ acima, são ideais, o rótulo de cada um desses loops pode ser determinado no momento em que o loop é achado, simplesmente observando os rótulos dos loops anteriormente encontrados que são adjacentes a ele, se existir algum. Se não existir, a rotulação do loop é nesse caso, imediata. Em vista disso, excluindo o processo de se achar as auto-intersecções de $PC_k$, pode-se dizer que a rotulação de seus loops torna-se um problema em $1D$, se $P_k$ ou $\Gamma_k$ for ideal. Isto também torna possível atualizar uma snake topológica $S_k$ através de um processo em que essa curva e as obtidas a partir dela — $TC_k$ e $PC_k$ — são percorridas uma única vez. Mais ainda, suponha que se regularize um segmento de $TC_k$ tão logo se obtenha seus dois vértices. Então, se uma divisão(split), ocorre, o resultado acima permite que se desenvolva uma nova componente — $S'$ — de $S_{k+1}$, tão logo seja movimentado o último snaxel — $s_i$ de $S_k$ cuja nova posição assumida — $T_k(s_i)$ — influencia a forma de $S'$. É o que se chama gerar uma snake tão logo seja possível.

**IV** Resultados mais fracos podem ser obtidos se os *e-ms* usados são somente adequados. A diferença é que para rotular corretamente, sem antecessores adjacentes a ele, deve-se fazer uma análise local numa vizinhança de seu ponto de junção com o resto da curva. Para os outros loops o esquema de rotulação mencionada em *III* pode ser mantido.

**V** Como já foi visto, o cumprimento das raias de $T_k$ deve ser menor do que $d$ — o tamanho de uma aresta da malha — para evitar que segmentos da imagem sejam perdidos. No caso em consideração, $P_k$ e $\Gamma_k$ não precisam ter necessariamente essa propriedade mas ambos devem ser $\mu$-limitados. Vários resultados obtidos a seguir requerem essa última condição e não podem ser mantidos se ela for substituída por uma outra do tipo: "os snaxels devem ter raias de magnitude menor do que $d + \epsilon$ ", para qualquer $\epsilon > 0$.

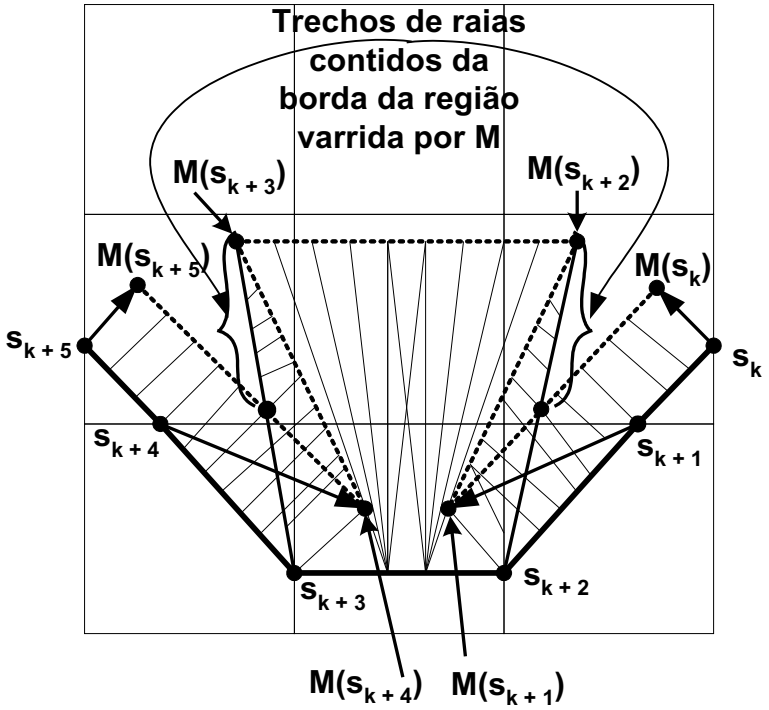


Figura 4.15: Dois pontos de intersecção entre $D_k$ e a reta suporte.

## 4.2 Tornando um mapeamento em adequado

O primeiro passo é fazer a snake evoluir através de contrações. Primeiramente, considerou-se fazer com que $P_k$ se contraísse. Isso, entretanto, significa que a curva projetada $PC_k = P_k(S_k)$, deve estar no interior do polígono determinado por $S_k$, o que pode ser muito restritivo se queremos tornar $P_k$ adequada também. O problema é que, para $P_k$ ser também adequada, a raia $p_{k,j} = [s_{k,j}, P_k(s_{k,j})]$, $i = 0, \ldots, I$, deve se situar no cone determinado pelo ângulo interno de $S_k$ em $s_{k,j}$. Isto pode ser uma restrição muito forte se esse ângulo for pequeno. Além do mais, verificar se $p_{k,j}$ está dentro desse cone demanda um número de operações ($4\times, 4+, 2$ testes), que é muito mais do que o necessário, como se verá . Em vista desses pontos, resolveu-se considerar apenas fazer $\Gamma_k$ se contrair. Agora, $PC_k$ precisa somente, estar no interior da dilatação de $S_k$. Isto é suficiente para evitar que a snake volte para uma célula que já foi totalmente varrida por ela, embora ela possa retroceder dentro das células que são cruzadas no estágio. Entretanto, em nossos testes esse fato nunca retardou a evolução da snake de forma mensurável. Além disso, o locus da raia $p_{k,j}$ foi extendido para um cone com um ângulo de no mínimo $90^\circ$ e para satisfazer essa nova condição, consideravelmente menos computação é requerida. De fato, isso pode ser alcançado atuando-se em dois momentos. Primeiro, depois de se computar

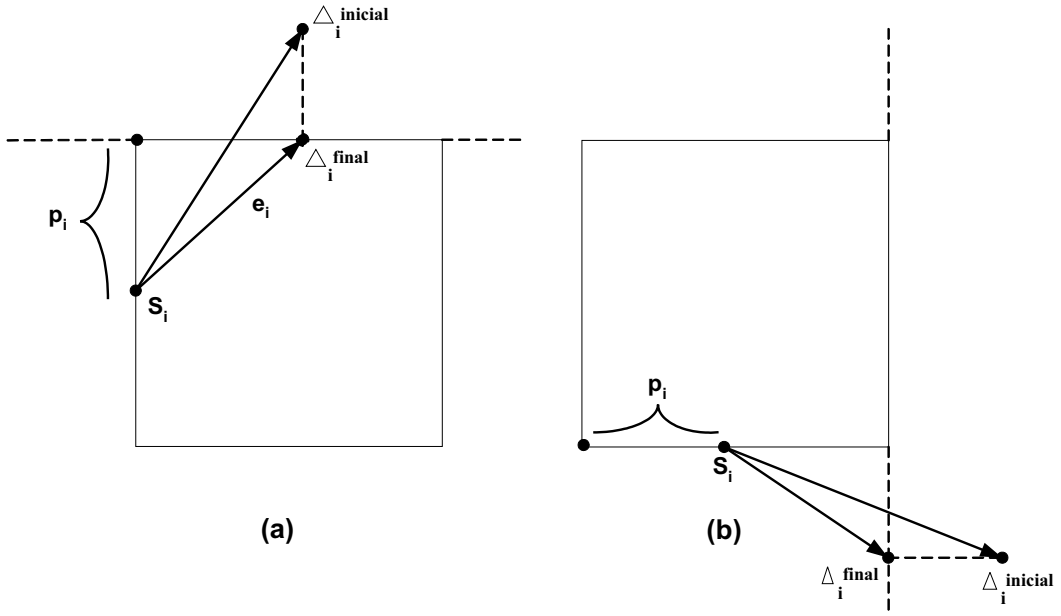


Figura 4.16: Exemplos de aplicação do procedimento *Correcting_Displacements* quando $B_i^2 = 0$ (caso da figura a) e $B_i^2 = 1$ (caso da figura b).

o deslocamento $(\Delta_i(0), \Delta_i(1))$ a ser aplicado ao snaxel $s_{k,i} = (C_i, E_i = (B_i^1, B_i^2), p_i)$, se deve executar o procedimento *Corrrecting_Displacements* abaixo:

**Procedure Correcting_Displacements**

**if** $(B_i^2)$

$\Delta_i\,(B_i^1) =$ **min**$(\Delta_i\,(B_i^1), d - p_i)$

**else**

$\Delta_i\,(B_i^1) =$ **max**$(\Delta_i\,(B_i^1), -p_i)$

A figura 4.16 ilustra a correção efetuada por esse procedimento, quando $B_1^2 = 0$ (figura 4.16.a) e $B_2^1 = 1$ (figura 4.16.b).

Fazendo a correção no valor de $\Delta_i(B_i^1)$, da forma indicada acima, se impede $s_{k,j}$ de se mover para traz além da reta, ortogonal a sua aresta, que passa por seu vértice externo, $v_i$. Observe que não existe restrição quanto a outra coordenada — $\Delta_i(1 - B_i^1)$. Em vista disso, e como estamos assumindo que $||\Delta_j|| < d$ ou mais geralmente, que $T_k$ é $\mu$-limitado, $t_{k,j} = s_{k,j} + \Delta_j$ pode ser um ponto qualquer nas quatro células adjacentes ao vértice interno de $s_{k,j}$. Embora o procedimento acima seja específico da metodologia que estamos introduzindo não é verdade que seu custo computacional inteiro deva ser incluído no da metodologia para efeito de comparações com abordagens alternativas. Ocorre que o procedimento

acima pode ser embutido num que garante que o deslocamento dos snaxels é $\mu$-limitado, sem aumentar seu custo computacional. Esse último procedimento por sua vez pode ser implementado efetuando um único teste a mais — que serve para identificar a direção da aresta em que o snaxel está — do que o que limita a norma_sup dos deslocamentos a um dado valor pré-fixado. Mas usar a norma_sup, no caso, é mais simples do que empregar qualquer outra e alguma limitação para o tamanho dos deslocamentos tem que ser estabelecida, explícita ou implicitamente, de qualquer maneira para que contornos não sejam saltados. Assim, no máximo, esse teste adicional pode ser posto na conta da metodologia empregada aqui. Isso, em função de se fazer os deslocamentos $\mu$-limitados não por causa do procedimento *Correcting_Displacements*.

Se $V_j = [v_{j-1}, v_j, v_{j+1}]$ não contém nenhuma aresta diagonal então a raia $(v_j, t_{k,j})$ estará no cone determinado pelo ângulo interno de $\mu D(S_k)$ em $V_j$. Infelizmente, isso pode não ser verdade se a poligonal $V_j$ contém diagonais de células, o que pode significar que o tratamento acima não consegue assegurar que $T_k(s_k)$ seja interior a $\mu$-dilatação de $S_k$, como acontece no caso da figura 4.14. Isto, entretanto, não é um problema, já que o objetivo é fazer, não $T_k \bullet D^{-1}$, e sim $\Gamma_k = P_k \bullet D^{-1}$, contrátil. Em relação a isso, observe que no exemplo da figura 4.18, apesar de $TC_k$ ter vértices externos a $DP_k$ as suas intersecções com as arestas da malha, que definem a curva projetada, são todas interiores a $DP_k$. Essa retração efetuada pelo próprio processo de projeção sobre a malha evita que tenhamos que considerar, explicitamente, ao corrigir o deslocamento dos snaxels, as restrições determinadas pelas arestas diagonais de $DP_k$. Com isso a correção pode-se restringir a uma coordenada por snaxel.

Se $PC_k$ não cortar as células já totalmente varridas nem as arestas não diagonais de $DP_k$ então ela será inteiramente interior a $DP_k$, o que equivale dizer que $\Gamma_k$ será contrátil. O caso é que o procedimento *Correcting_Displacements* garante apenas que os vértices de $TC_k$ e em conseqüência, também os de $PC_k$, estejam em células ainda não completamente exploradas, mas não impede que os segmentos de $PC_k$ cortem arestas horizontais e verticais de $DP_k$, como ocorre no exemplo da figura 4.19. Situações como a do exemplo dessa figura, serão tratadas em uma segunda intervenção efetuada quando a curva projetada está sendo construída. Observe no exemplo da figura 4.14 a existência de um quadrilátero de varredura $Q_i = [s_{k,i-1}, T(s_{k,i-1}), T(s_{k,i}), s_{k,i}]$ que contém o vértice externo de $s_{k,i}$, que também é o de $s_{k,i-1}$, em seu interior. De fato, devido a limitação do deslocamento dos snaxels, o fato de $PC_k$ cruzar uma aresta horizontal ou vertical de $DP_k$ torna-se equivalente a haver um snaxel cujo out-vértice é coberto por um quadrilátero de

varredura adjacente a ele. Essa situação pode ser caracterizada em termos dos snaxels de $S_k$ e dos vértices de $PC_k$, pela existência de uma configuração constituída pelos seguintes elementos:

**i)** os snaxels $s_{k,i-1}$ e $s_{k,i}$, contidos na célula $C_1$ têm o mesmo vértice externo $v_i$;

**ii)** a intersecção de $[T(s_{k,i-1}), T(s_{k,i})]$ com a célula $C_2$, diagonalmente oposta a $C_1$ em relação a $v_i$, é um segmento delimitado por pontos — $s'_{k,j-1}$ e $s'_{k,j}$ — que estão ambos em arestas adjacentes a $v_i$;

**iii)** além disso, $s_{k,j}$ e $s'_{k,j}$ também devem pertencer a mesma célula, $C$.

Considerando a notação indicada na figura 4.17 pode-se verificar que se a condição **(iii)** é atendida, isto é, se $C(s'_{k,j}) = C(s_{k,i})$ então as demais serão satisfeitas se

$$E(s'_{k,j}) = (E(s_{k,i}) + 1)\ mod\ 4$$

A figura 4.17 se refere ao caso em que $E(s_{k,i}) = 3$, o que implica pela igualdade acima que $E(s'_{k,i}) = 0$. Os demais casos estarão relacionados com rotações dessa figura.

No caso em questão, as três arestas onde $s_{k,i-1}$ pode estar localizado estão identificadas na figura 4.17 pelos números 1, 2 e 3. Para cada célula onde $T_k(s_{k,i-1})$ pode estar, assumindo uma dessas possibilidades, escrevemos o número representativo desse caso junto a um dos vértices da célula. Observe, então que se $E'(s'_{k,j}) = 0$ então, como $s'_{k,j}$ deve pertencer a $T_k(s_{k,i-1}, s_{k,i})$, $T_k(s_{k,i-1})$ deve estar a esquerda da vertical que passa por $s'_{k,j}$, o que, pelo indicado na figura 4.17, só pode acontecer no *caso 3*. Mais ainda, nesse caso $T_k(s_{k,i-1})$ estará abaixo da horizontal que passa por $s_{k,i}$ o que significa que $T_k(s_{k,i-1})$ não está em $C_2$ e como esse ponto tem de estar também a direita da vertical $v_1$ então $TC_k$ entrará em $C_2$ pela aresta de baixo. Teremos, assim, atendido a todos os requisitos expressos em **(i)** e **(ii)**.

Uma vez que uma configuração atendendo as condições acima é detectada, para tornar $v_i$ exterior a curva projetada, deve-se substituir seus vértices $s'_{k,j-1}$ e $s'_{k,j}$ por pontos nas arestas de $s_{k,i-1}$ e $s_{k,i}$, respectivamente. Para minimizar a correção a ser feita pode-se pensar em escolher pontos próximos a $v_i$ como $D_k(s_{k,i-1})$ e $D_k(s_{k,i})$. A escolha mais simples, entretanto, é fazer $s'_{k,j-1} = s_{k,j-1}$ e $s'_{k,j} = s_{k,j}$, o que significa que esses snaxels cujo movimento para trás criou a situação indesejada descrita acima, são trazidos de volta para sua posição original.

Para obter uma forma algorítmica para detectar a configuração descrita acima os seguintes conceitos e notações precisam ser introduzidos:

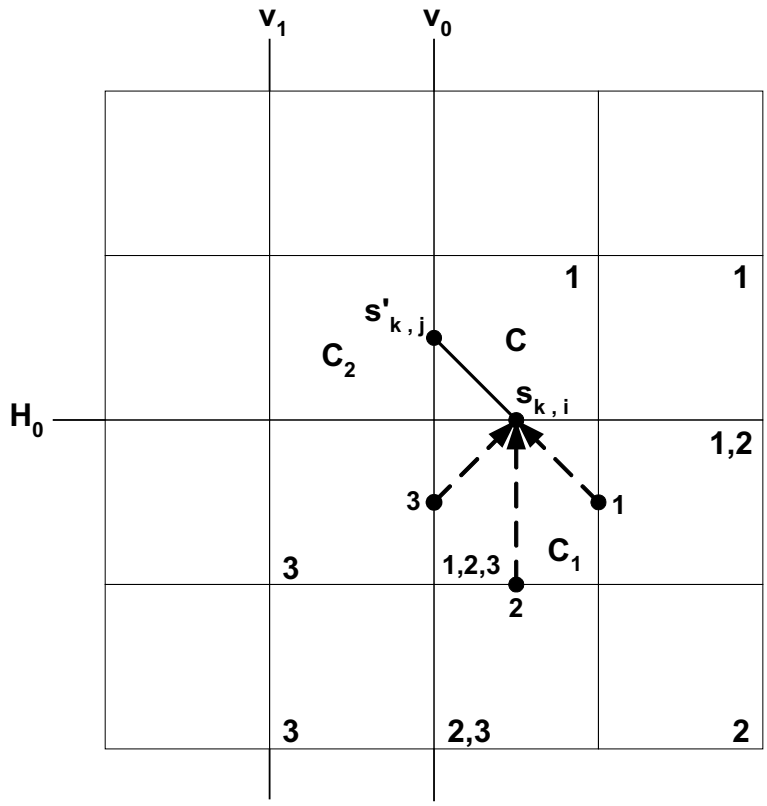


Figura 4.17: Uma intersecção de $PC_k$ e $DP_k$ fica caracterizada pelas condições $C(s'_{k,j}) = C(s_{k,i})$ e $E(s'_{k,j}) = (E(s_{k,i}) + 1)\ mod\ 4$.

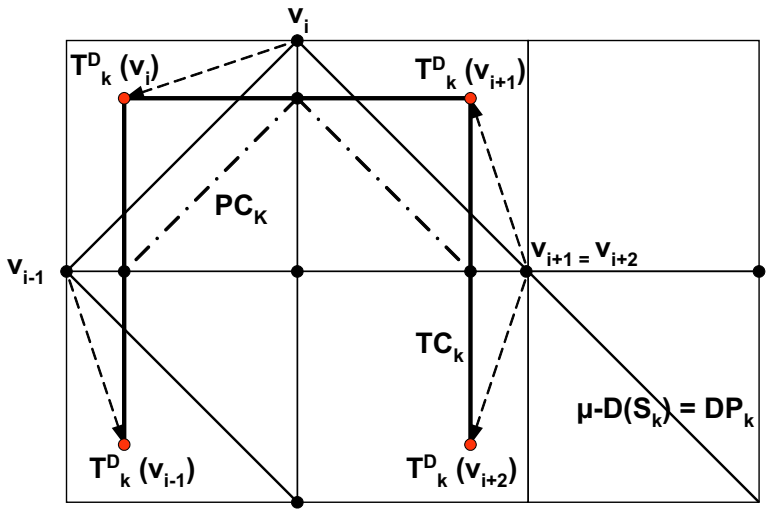


Figura 4.18: Nas células onde a aresta de $\mu$-$D(S_k)$ for uma diagonal, o próprio processo de projeção sobre a malha transforma um mapeamento $T_k^D = T_k \bullet D^{-1}$ que é não contrátil num mapeamento $\Gamma_k$ contrátil.

**A)** *j_cur* e *j_prev* vão indicar os índices do último e do penúltimo vértice da curva projetada, que já foram obtidos.

**B)** A representação em "CAP" do último vértice de $PC_k$ encontrado, será denotado por $(C'\_cur, E'\_cur, p'\_cur)$. A do último snaxel de $S_k$ tratado, será escrita $(C\_cur, E\_cur, p\_cur)$.

**C)** Seja i(j) tal que $s'_{k,\ i(j)} \in [T_k(s_{k,\ i(j)-1}), T_k(s_{k,\ i(j)})]$. Defina *i_cur* e *i_prev*, como sendo valores de i(.) para *j_cur* e *j_prev*, respectivamente. Observe que não necessariamente $i\_prev = i\_cur - 1$.

Por concisão, doravante, escreveremos que uma célula $C$, é seccionada em $i$ para nos referirmos que ela é cortada pelo segmento $[T_k(s_{k,i-1}), T_k(s_{k,i})]$ e que as extremidades desse segmento estão fora dela. Para identificar a configuração acima bastaria, pelo que vimos, testar as condições $C(s_i) = C(s'_j)$ e $E(s'_j) = (E(s'_i) + 1)\ mod\ 4$. Entretanto, dado que nessa configuração devemos ter também que $C_2$ é seccionada em $i$, vamos testar primeiro essa condição por dois motivos:

**1)** Para a maioria as células cruzadas por $PC_k$ ela é falsa o que não necessariamente ocorre com as outras condições.

**2)** Ela precisa ser testada de toda maneira para identificar de forma eficiente uma outra situação indesejável, conforme veremos a seguir.

Tendo em vista todas essas considerações chega-se ao seguinte procedimento:

**Passo 1:** Verifica-se se a última célula cruzada por $PC_k$, como $C_2$ na figura 4.19, é seccionada em *i_cur*. Isto é feito através da comparação de *i_cur* e *i_prev*. A condição $i\_cur = i\_prev$ caracteriza que essa célula seccionada em *i_cur*.

**Passo 2:** Se essa condição for verdadeira, testa-se, então, se $C'\_cur = C\_cur$ e finalmente, se $E'\_cur = (E\_cur + 1)\ mod\ 4$. Se a resposta a esses testes for afirmativa, a configuração acima ocorre e os dois últimos vértices de $PC_k$ devem ser substituídos por $s_{i\_cur}$ e $s_{i\_cur-1}$.

É importante notar que, embora se esteja verificando se um vértice $v_i$ pertence a um quadrilátero de varredura ou não, não há necessidade de se determinar explicitamente sua posição em relação as arestas desse quadrilátero. Tudo é feito fazendo comparações entre índices e coordenadas "CAP" dos snaxels de $S_k$ e dos vértices de $PC_k$.

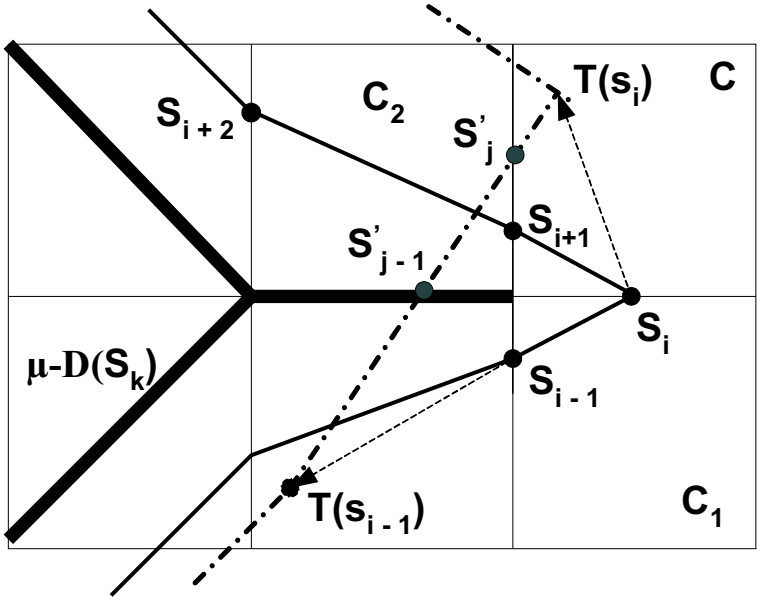


Figura 4.19: Exemplo em que $PC_k$ corta $DP_k$ apesar de ter todos os vértices em células não totalmente exploradas.

As providências descritas até aqui são suficientes para garantir que $\Gamma_k$ satisfará a primeira condição de ser adequado. A segunda condição, entretanto, requer que todos os quadriláteros de varredura reversos sejam eliminados. Infelizmente, isto pode representar muito trabalho e, em vista disto, é preciso trocar o objetivo de tornar $\Gamma_k$ adequado por outro, similar, mas de mais fácil obtenção. Esse novo objetivo consiste em requerer somente que $\Gamma_k$ possua um *e-m* $\mu$-equivalente a ele, $\Gamma^*$, que é adequado. Em relação a estratégia proposta neste trabalho para controlar a topologia de uma T-snake que se contrai, esta mudança não introduz novas dificuldades. Este ponto se torna claro se os dois fatos seguintes forem considerados:

**A)** As estruturas de loop de $\Gamma_k^*(S_k)$ e $\Gamma_k(S_k)$ são equivalentes no sentido de que para cada loop não insignificante, em uma dessas curvas, existe um loop da outra curva que é $\mu$-equivalente a ele.

**B)** Por causa disso $\Gamma_k(S_k)$ herdará de $\Gamma_k^*(S_k)$ todas as boas propriedades que tornam mais fácil rotular os loops de uma curva que é o resultado da aplicação de um *e-m* adequado a uma curva $\mu$-regular, que foram antecipados na seção 4.1.5 e que serão vistos em detalhe no capítulo 5.

No exemplo da figura 4.25 $\Gamma_k$ não é adequado e, já que ele dá origem ao quadrilátero reverso $Q_i$. $\Gamma_k^*$, entretanto, que é $\mu$-equivalente a ele, o é. Assim, o requisito de que $\Gamma_k$ deva

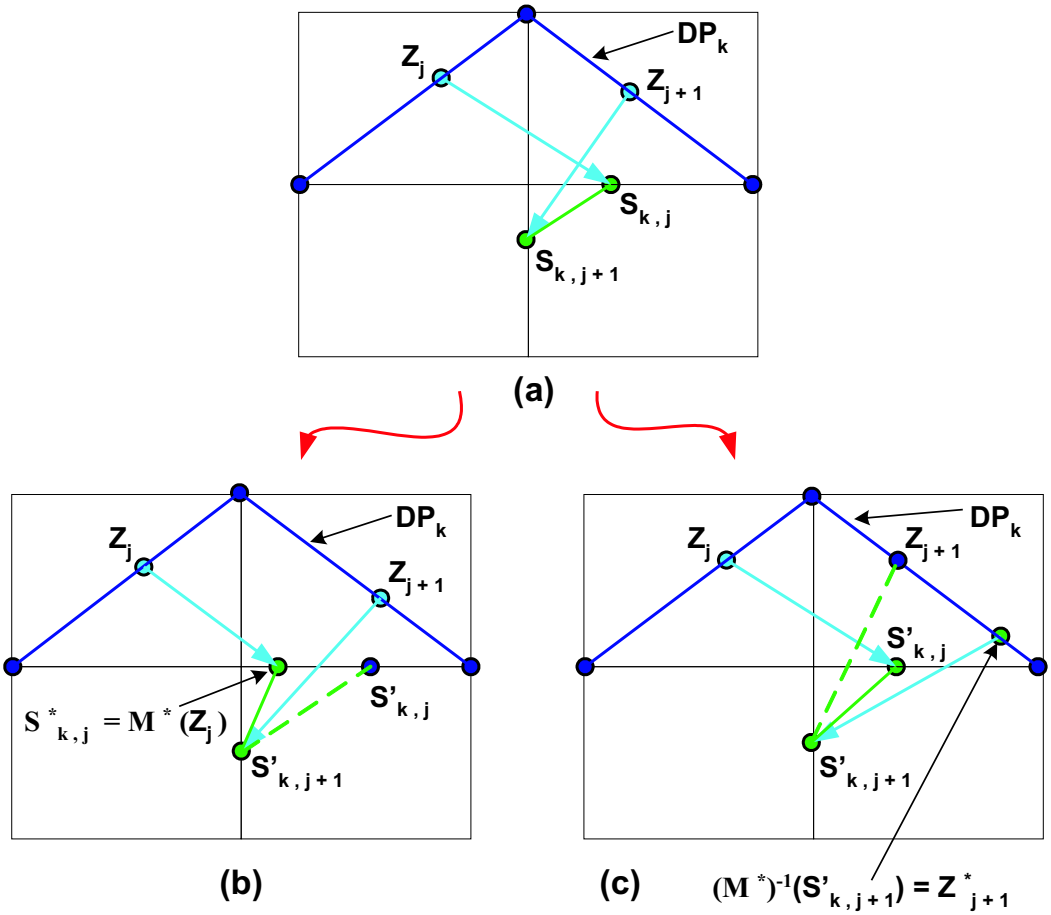


Figura 4.20: Empregando-se a notação usada no lema 4.2.1 temos nas figuras (a), (b) e (c) um trecho dos mapeamentos $M$, $\gamma^j(M)$ e $\gamma^{-j}(M)$ respectivamente.

ser adequado for mantido, alguns dos vértices de $\Gamma_k(S_k)$ precisaram ser reposicionados. O novo objetivo, mais fraco, neste caso, não demanda nenhuma ação.

O resultado abaixo permite concluir que na maior parte dos casos em que $\Gamma_k$ produz quadriláteros reversos, esses quadriláteros podem ser eliminados passando-se a um mapeamento $\mu$-equivalente a ele.

**Lema 4.2.1** *Seja $M$ um mapeamento canônico definido em $DP_k$ e $e_i = [s'_i,\ s'_{(i+1)\ mod\ N}]$ ; $i = 0, \ldots, N-1$, as arestas de $M(DP_k)$. Denote, também, $(M)^{-1}(s_i)$ por $z_i$. Então:*

**I.** *Se as raias que chegam a $s_j$ e $s_{(j+1)\ mod\ N}$ não se interceptam ou se interceptam, mas a reta suporte de $e_j$ corta $(M)^{-1}(e_j)$ — caso ilustrado na figura 4.11— então $M$ não produzirá quadriláteros reversos com topo em $e_j$.*

**II.** *Se essas raias se interceptarem, mas $(M)^{-1}(e_j)$ contiver um vértice $v_i$ de $DP_k$, como no caso da figura 4.20, então haverá mapeamentos $M^j \triangleq \gamma^j(M)$ e $M^{-j} \triangleq \gamma^{-j}(M)$ , $\mu$-equivalentes a $M$ , tais que:*

**1)** $M^j\ (DP_k) = [s'_0, s'_1, \ldots, s'_j, s'^{j}_{j+1} \neq s'_{j+1}, s'_{j+2}, \ldots, s'_{n-1}]$ *e*
$M^{-j}\ (DP_k) = [s'_0, s'_1, \ldots, s'_j \neq s'^{-j}_{j}, s'_{j+1}, s'_{j+2}, \ldots, s'_{n-1}]$.

**2)** *Ou $s^j_{j+1} = s_{k,\ j+1}$ ou $s^{-j}_j = s_{k,\ j}$. Se uma dessas igualdades vale então ela continuará valendo se substituirmos no enunciado do lema 4.2.1, $M$ por qualquer outro mapeamento $\mu$-equivalente a ele.*

**3)** $(M^j)^{-1}\ (s_i) = z_i$ *, se $z_i$ não está na mesma aresta de $DP_k$ que $z_{j+1}$ e $(M^{-j})^{-1}\ (s'_i) = z_i$, se $z_i$ não está na aresta de $z_j$ .*

**4)** *$M^j$ não produz quadriláteros reversos com topo em $e^j_j = [s'_j,\ s^j_{j+1}]$ nem $M^{-j}$ com topo em $e^{-j}_j = [s^{-j}_j,\ s'_{j+1}]$.*

Assuma que $PC_k$ é uma $\mu$-curva significante e aplique o resultado acima para $M = \Gamma_k$. Nesse caso, $M(DP_k) = PC_k$ e $e_i = [s_{k,i},\ s_{k,\ (i+1)\ mod\ N}]$; $i = 0, \ldots, N-1$. Suponha, então, que existe $j$ tal que:

1. $s_{k,j}$ e $s_{k,\ (j+1)}$ não estão numa mesma aresta, o que é sempre possível de se conseguir dado que $PC_k$ é significante;

2. $s^j_{j+1} = s_{k,\ j+1}$, ou seja, tal que $\gamma^j(\Gamma_k)(DP_k)$ também é $PC_k$.

Então aplicando reiteradamente o resultado acima considere a seqüência de mapeamentos:

$X^{j+1} = \gamma^{j+1}(\Gamma_k)$ ,
$X^{j+2} = \gamma^{j+2}(\gamma^{j+1}(\Gamma_k))$,
...,
$X^{n-1} = \gamma^{n-1}(\ldots(\gamma^{j+2}(\gamma^{j+1}(\Gamma_k)))\ldots)$,
$X^0 = \gamma^0(\gamma^{n-1}(....(\gamma^{j+2}(\gamma^{j+1}(\Gamma_k)))...))$,
...,
$X^{j-1} = \gamma^{j-1}(\ldots(\gamma^0(\gamma^{n-1}(\ldots(\gamma^{j+2}(\gamma^{j+1}(\Gamma_k)))\ldots)))\ldots)$

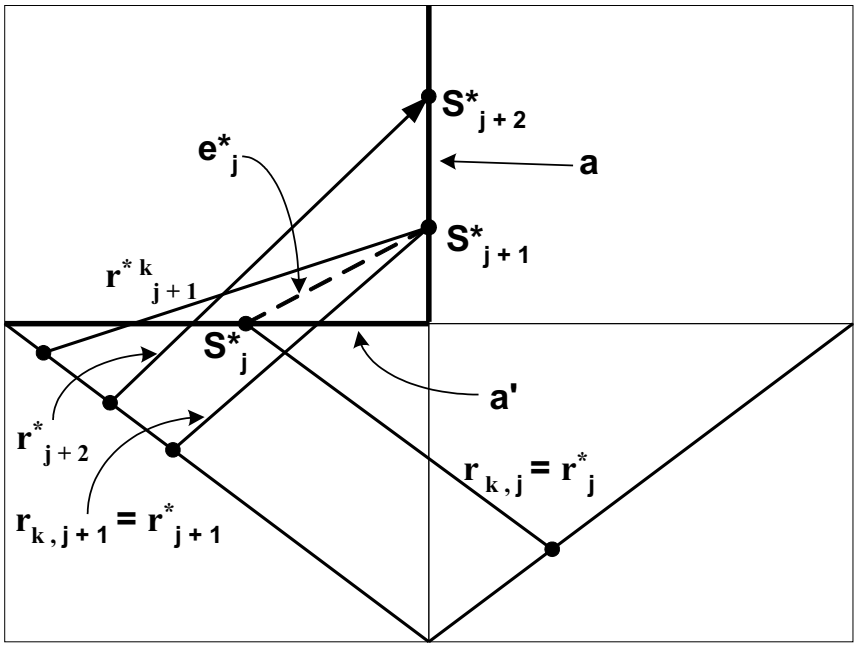


Figura 4.21: Como tornar $\gamma^j(X^{j-1})$ sem quadriláteros reversos.

$X^{j+1}$ não forma um quadrilátero reverso cujo topo é a aresta de sua imagem correspondente $e_{j+1}$. Para $X_{j+2}$, isso é verdade tanto para a aresta correspondente a $e_{j+1}$ como para a correspondente a $e_{j+2}$. E assim por diante até que chegamos a $X^{j-1}$, que pela sua própria construção, não tem quadriláteros reversos com topo nas suas arestas correspondentes a $e_0,\ e_1, \ldots,\ e_{j-1},\ e_{j+1}, \ldots,\ e_{n-1}$. Portanto a única aresta dele que pode conter o topo de um quadrilátero reverso é a correspondente a $e_j$ . Daqui por diante as arestas de $X^{j-1}(DP_k)$, seus vértices e as raias deles considerando-se esse mapeamento serão referenciados, respectivamente por $e_i^*$, $s_i^*$ e $r_i^*$, $i = 0, \ldots, N-1$.

Pelo item II.2, se podemos eliminar os quadriláteros reversos gerados por $Gamma_k$ com topo em $e_j$, sem alterar a sua imagem, como estamos assumindo, o mesmo deve acontecer em relação aos quadriláteros gerados por $X^{j-1}$, com topo em $e_j^*$. Essa aresta tem como vértice final $s_{k,\ j+1}$, que é vértice da curva gerada por todos os mapeamentos da seqüência e cuja raia — $r_{k,\ j+1}$ — se mantém inalterada em todos eles. Se ela precisa ser alterada, passando a $r_{j+1}^{**}$, para eliminar quadriláteros reversos com topo em $e_j^*$ então é porque $r_j^*$ intersecta $r_{k,\ j+1}$. $r_{k,\ j+1}$ e $r_{j+1}^{**}$ ligam pontos da mesma aresta de $DP_k$ a um mesmo ponto numa aresta da malha — $a_j$. Isso significa que elas só podem cruzar ambas uma mesma aresta da malha — $a'_j$ — como mostrado na figura 4.21. Assim se, $r_j^*$ intersecta $r_{k,\ j+1}$ mas não $r_{j+1}^{**}$, isso significa que $s_j^*$ está ou em $a_j$ ou em $a'_j$. Como escolhemos $j$ de forma que $s_{k,j}$ e $s_{k,\ j+1}$ não estejam na mesma aresta da malha e $s_j^*$ e $s_{k,j}$ são vértices

correspondentes temos que $s^*_j \in a'_j$.

Se o mapeamento obtido a partir de $X^{j-1}$, mediante essa troca da raia de $s_{k,\ j+1}$, que é $\gamma^j\ (X^{j-1})$, produz quadriláteros reversos é porque $r^{**}_{j+1}$ corta $r^*_{j+2}$. Como $r^*_{j+2}$ não corta $r_{k,\ j+1}$ temos que, também, $s^*_{j+2}$ deve estar ou em $a_j$ ou em $a'_j$. Como $s^*_j$ e $s^*_{j+2}$ não podem estar ambos em $a'_j$ , temos que, obrigatoriamente, $s^*_{j+2} \in a_j$. Mas, nesse caso é impossível que $r^*_{j+2}$ corte $r^{**}_{j+2}$ e não corte $r_{k,\ j+1}$ dado que essas duas últimas raias chegam a um mesmo ponto de $a_j$. Chegamos assim a uma contradição o que mostra que $\gamma^j\ (X^{j-1})$ que é $\mu$-equivalente a $\Gamma_k$ não gera quadriláteros reversos.

Se não houver $j$ tal que $s^j_{j+1} = s_{k,\ j+1}$, então, tendo em vista o item II.2 do lema 4.2.1, existe $m$ tal que $s^{-m}_m = s_{k,m}$. Nesse caso $X^{-(m+1)} = \gamma(^{-(m+1)}(\ldots(\gamma^{-(n-1)}(\gamma^{-0}(\ldots(\gamma^{-(m-2)}(\gamma^{-(m-1)}(\Gamma_k)))\ldots)))\ldots)$, só pode ter quadriláteros reversos com topo em sua aresta correspondente a $e_m$. A partir daí, um raciocínio análogo ao empregado para mostrar que $\gamma^j(X^{j-1})$ não produz quadriláteros reversos, no caso em que $s^j_{j+1} = s_{k,\ j+1}$, permite concluir que o mapeamento $\gamma^{-m}(X^{-(m-1)})$ possui idêntica propriedade no caso em que $s^{-m}_m = s_{k,m}$. O processo descrito acima é representado, passo a passo, na figura 4.22.

Assim, se $\Gamma_k$ não tem um mapeamento $\mu$-equivalente sem quadriláteros reversos é porque ele não satisfaz as condições requeridas pelo lema 4.2.1, o que só pode acontecer se $PC_k$ tem dois vértices consecutivos — $s'_{k,\ j-1}$ e $s'_{k,j}$ — tais que:

**1)** $s'_{k,\ j-1}$ e $s'_{k,j}$ não estão na mesma aresta da malha;

**2)** a célula $C_j$ contendo $[s'_{k,\ j-1}, s'_{k,j}]$ é seccionada em um dado*i*, o que significa que $[s'_{k,\ j-1}, s'_{k,j}] \subseteq [T_k(s_{k,\ i-1}), T_k(s_{k,i})]$;

**3)** as raias de $\Gamma_k$ terminando em $s'_{k,\ j-1}$ e $s'_{k,j}$ se interceptam. Em vista de **2)** isso significa que $p_{k,\ i-1} = [s_{k,\ i-1},\ T_k(s_{k,\ i-1})]$ e $p_{i,k} = [s_{k,i},\ T_k(s_{k,i})]$ se interceptam.

Suponha que $s'_{k,\ j-1}$ e $s'_{k,j}$ estão ambos em arestas da mesma direção, digamos horizontal e que $s'_{k,j}$ está mais alto que $s'_{k,\ j-1}$. Como as arestas dos dois tem de ser diferente e $[s'_{k,\ j-1}, s'_{k,j}] \subseteq [T_k(s_{k,\ i-1}), T_k(s_{k,i})]$ concluímos que $T_k(s_{k,\ i-1})$ está sob a reta horizontal contendo a aresta de baixo de $Cj$ — $H_0$, na figura 4.23 — e $T_k(s_{k,i})$ está mais alto que a reta contendo a aresta de cima de $C_j$ — $H_1$, na figura 4.23. Nesse caso, $s_{k,i}$ está acima de $H_0$ e $s_{k,\ i-1}$ está abaixo de $H_1$. Assim se $p_{k,\ i-1}$ e $p_{k,i}$ se interceptam num ponto $x$ então:

**1)** $x$ está na faixa entre $H_0$ e $H_1$;

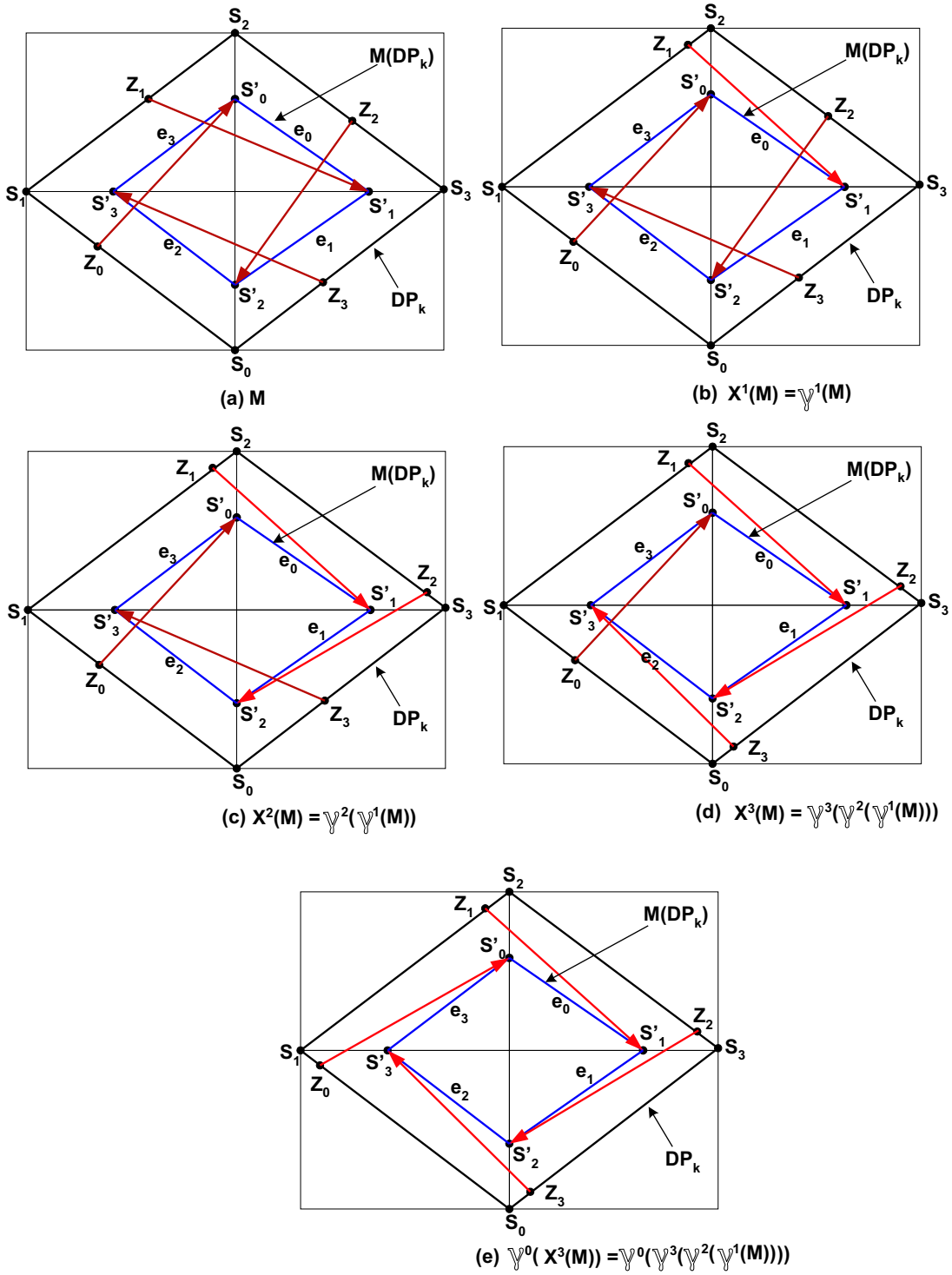


Figura 4.22: Quando $M$ é o mapeamento dado na figura (a) e $j = 0$, os mapeamentos da seqüência abaixo são indicados nas figuras (b), (c) e (d). O Mapeamento obtido eliminando-se o último cruzamento de raias — $\gamma^0(X^3(M))$ — é $\mu$-equivalente a $M$ e não tem quadriláteros reversos.

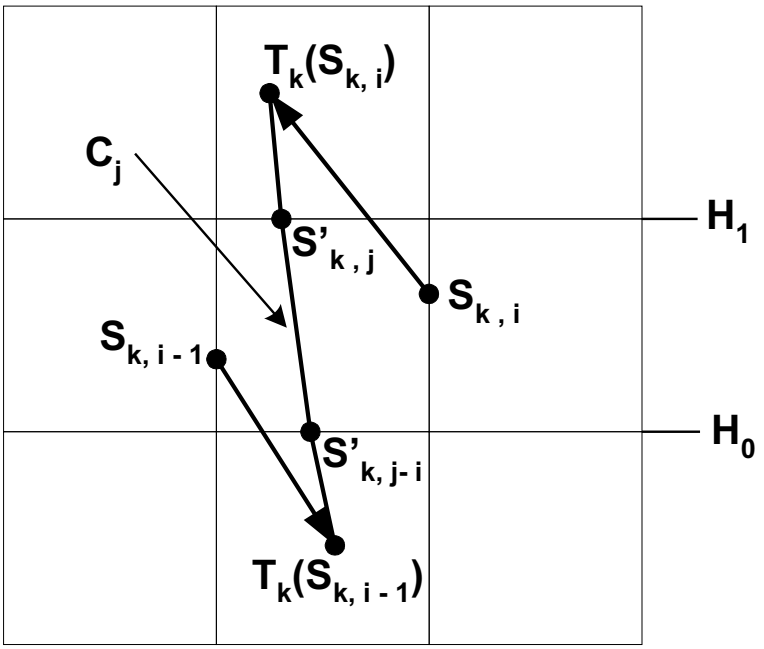


Figura 4.23: Se $s'_{k,j}$ e $s'_{k,\,j-1}$ estão em arestas com a mesma direção, então, como $s_{k,\,i-1}$ e $s_{k,i}$ não podem estar na mesma aresta da malha, então as rais $p_{k,i}$ e $p_{k,\,i-1}$ não podem se interceptar.

**2)** como $T_k(s_{k,\,i-1})$ está abaixo de $H_0$ , ou seja, abaixo de $x$ , então $s_{k,\,i-1}$ está acima de $x$. Como ele deve estar também abaixo de $H_1$, isso permite concluir que $s_{k,\,i-1}$ está em alguma das arestas verticais da malha na faixa entre $H_0$ e $H_1$. Como $T_k$ é $\mu$-limitado, podemos nos restringir as duas entre essas arestas que pertencem a célula $C_x$ contendo $x$;

**3)** idêntica conclusão pode ser obtida, em relação a $s_{k,i}$, considerando que ele deve estar abaixo de $x$ — porque $T_k(s_{k,i})$ está mais alto que $x$ — e acima de $H_0$. Suponha que as arestas de $s_{k,i}$ e $s_{k,\,i-1}$ são diferentes e considere a semi reta com origem em $s_{k,i}$ contendo $T_k(s_{k,i})$. Partindo de $s_{k,i}$, essa semi reta sobe mas se ela intercepta $p_{k,\,i-1}$, então ela vai estar abaixo de $s_{k,\,i-1}$, quando cruzar a aresta vertical que contem esse ponto, ou seja não terá chegado ainda a $T_k(s_{k,i})$ que está acima de $H_1$. Como o restante dessa semi-reta não pode conter $T_k(s_{k,i})$ dado que $T_k$ é $\mu$-limitado. Chegamos, assim a um absurdo que mostra que $s_{k,\,j-1}$ e $s_{k,j}$ não podem estar em arestas com a mesma direção. Eles estão, portanto em arestas que compartilham um vértice $v_i$.

Daqui por diante vamos assumir que isso acontece sem menção explícita a esse fato. Considere, então, os quadrantes determinados pelas linhas horizontal e vertical que passam por $v_j$. Observe que como $C_j$ é seccionada em $i$, então $T_k(s_{k,i})$ e $T_k(s_{k,\,i-1})$ tem de

estar no interior de quadrantes diagonalmente opostos, nenhum dos dois contendo $C_j$. Pela mesma razão, $(\Gamma_k)^{-1}(s'_{k,\ j-1})$ e $(\Gamma_k)^{-1}(s1_{k,j})$ devem estar numa mesma célula $C'$. Levando em conta que $\Gamma_k$ é $\mu$-limitado, $C'$ deve ser uma das seis células onde $(\Gamma_k)^{-1}(s'_{k,\ j-1})$ pode, em função disso, estar e também uma das seis onde $(\Gamma_k)^{-1}(s'_{k,j})$ pode, nesse caso, se localizar. Para atender a essas duas condições, $C'$ deve ser uma das quatro células adjacentes a $v_j$ . Suponha agora que $C'$ esteja no mesmo quadrante que $T_k(s_{k,i})$. Nesse caso, como $T_k(s_{k,\ i-1})$ está no interior do quadrante oposto, $s_{k,\ i-1} \in C'$, só pode estar localizado numa aresta de $C$ contida numa das linhas — horizontal ou vertical — que delimitam o quadrante. Isso significa que $[s_{k,\ i-1}, T_k(s_{k,\ i-1})]$ é disjunto de $C'$. Por outro lado, $p_{k,i}$ está inteiramente contido em $C'$. Como $s_{k,\ i-1} \notin p_{k,i}$ — senão, ou $s_{k,\ i-1} = s_{k,i}$ ou $T_k(s_{k,i})$ não estará no interior do quadrante contendo $C$ — isso significa que $p_{k,i} \cap p_{k,\ i-1} = \oslash$ , contrariando a suposição que essas raias devem se interceptar. Portanto, $C'$ não pode estar no mesmo quadrante que $T_k(s_{k,i})$ e desenvolvendo raciocínio análogo mostraríamos que ela também não pode estar no quadrante de $T_k(s_{k,\ i-1})$. Desse modo, ou $C' = C_j$ ou $C'$ é a célula diagonalmente oposta a ela em relação a $v_j$ - $O_j$ .

Considere, inicialmente que $C' = O_j$ e identifique as arestas de $O_j$ por $a_1$, $a_2$, $a_3$ e $a_4$ seguindo o seguinte critério:

**i)** $a_1$ é aresta não adjacente a $v_j$ com um vértice no quadrante onde está $T_k(s_{k,i})$;

**ii)** $a_2$ indica a aresta caracterizada de forma idêntica a representada por $a_1$ só que com $T_k(s_{k,\ i-1})$ no lugar de $T_k(s_{k,i})$;

**iii)** $a_3$ indica a aresta que está na borda do quadrante que contém $T_k(s_{k,\ i-1})$;

**iv)** $a_4$ é a aresta na borda do quadrante que contém $T_k(s_{k,i})$.

A figura 4.24 representa no caso em que $v_j$ é o vértice inferior esquerdo de $C_j$ e $T_k(s_{k,i})$ está no quadrante a esquerda do que contém $C_j$.

Considerando essa notação e dado que $T_k$ é $\mu$-limitado, temos que $s_{k,j}$ pode estar nas arestas de $O_j$ referidas por $a_1$, $a_3$ e $a_4$. Entretanto, se $s_{k,i}$ está sobre $a_4$, então $p_{k,i}$ não tem pontos abaixo da horizontal que passa por $vj$, enquanto $p_{k,\ i-1}$ está inteiramente abaixo dessa linha, dado que, $s_{k,\ i-1}$ não pode estar também em $a_4$. Isso quer dizer que, $p_{k,i} \cap p_{k,\ i-1} = \oslash$ contrariando a hipótese feita sobre essas raias. Se $s_{k,i}$ está sobre $a_1$, $s_{k,\ i-1}$ não pode estar nem em $a_2$ nem em $a_3$, caso contrário $p_{k,i}$ e $p_{k,\ i-1}$ estarão separadas pela reta contendo uma diagonal de $O'$ que passa por $v_j$ e, portanto também nesse caso não poderão se interceptar. Resta então a possibilidade de $s_{k,\ i-1}$ estar em $a_4$, mas nesse

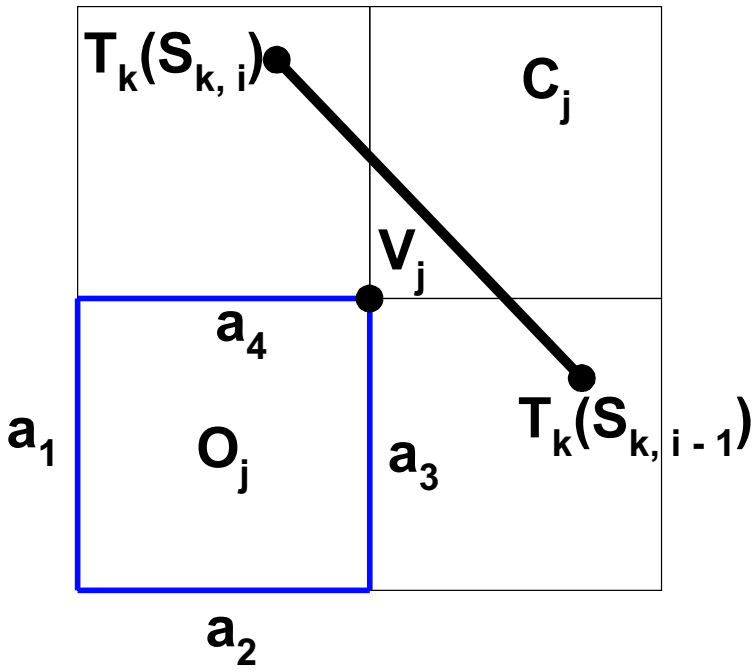


Figura 4.24: Localização das arestas $a_i$, $i = 1, 2, 3, 4$ quando $v_j$ é o vértice inferior esquerdo de $C_j$ e $T_k(s_{k,i})$ está na célula a esquerda de $Cj$.

caso, o procedimento *Correcting_Displacements* impedirá que $T_k(s_{k,i})$ ou $T_k(s_{k,\ i-1})$ esteja na célula onde estamos supondo que ele se localize. Pode-se assim, eliminar também a possibilidade de $s_{k,i}$ estar em $a_1$, o que significa que ele tem, obrigatoriamente de estar sobre $a_3$. Raciocínio inteiramente análogo levaria a conclusão que $s_{k,\ i-1}$ deve estar sobre $a_4$ produzindo-se, então, a configuração representada na figura 4.25, a única possível se $C' = O_j$.

Vamos, finalmente mostrar que não precisamos nos preocupar em eliminar os casos em que $C' = C_j$. Para isso, assuma que no texto a seguir, $\Phi_k$ se refere ao mapeamento obtido submetendo-se $\Gamma_k$ ao processo descrito acima para eliminação de todos os quadriláteros reversos que se enquadram nos casos cobertos pelo lema 4.2.1. Esse mapeamento ou é $\gamma^{-j}(X^{-(j-1)})$ ou $\gamma^{j}(X^{(j+1)})$, para algum $j$. Aplique,então a $\Phi_k$ o procedimento a seguir, o qual é ilustrado na figura 4.26:

**Passo 1** para cada par $(s'_j,\ s'_{j-1})$ de vértices consecutivos de $\Phi_k(DP_k)$, que estejam em arestas distintas e cujas imagens inversas estão na mesma célula que o segmento definido por eles, execute:

escolha um ponto $p_j$ no interior relativo do segmento $[(\Phi_k)^{-1}(s'_{k,\ j-1}),\ (\Phi_k)^{-1}(s'_{k,j})]$ e um ponto $q_j$ no interior do triângulo

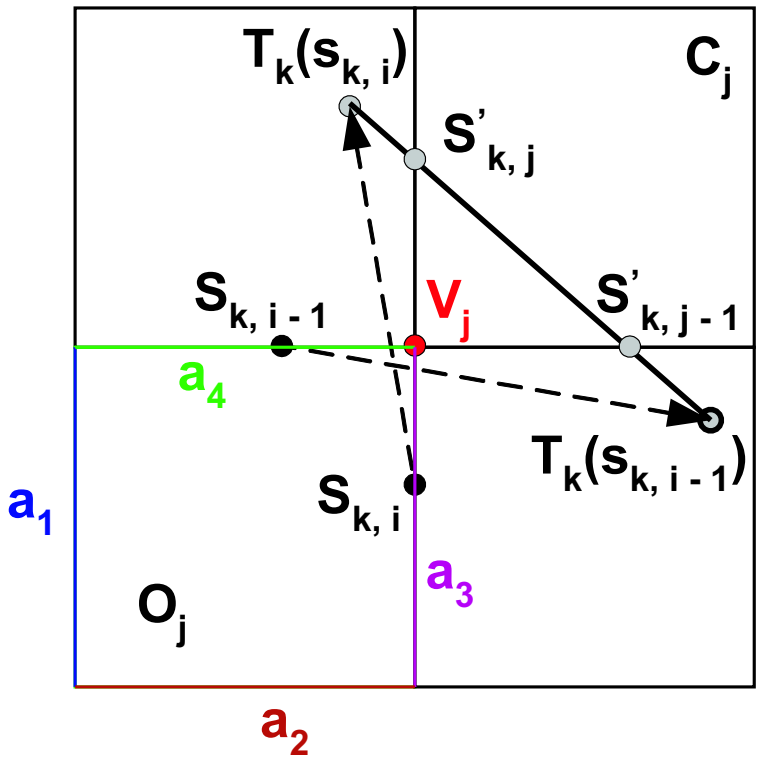


Figura 4.25: Configurações indesejáveis.

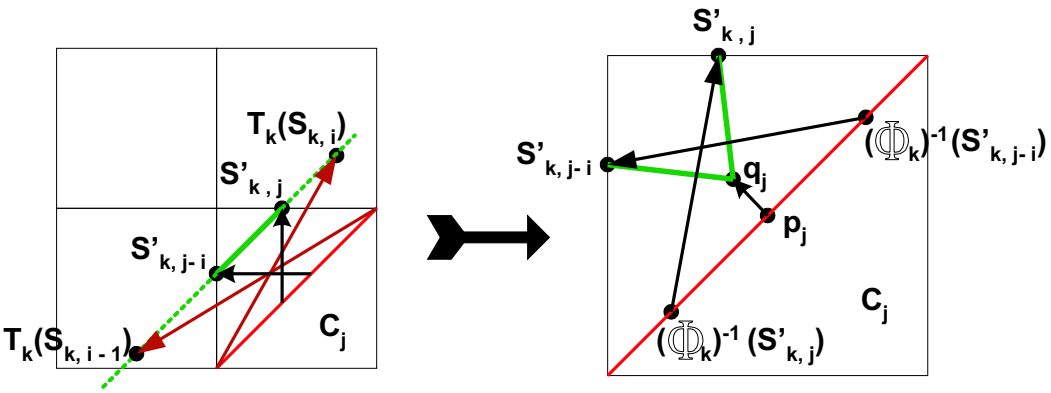


Figura 4.26: Procedimento para eliminar quadriláteros reversos quando $C' = C_j$.

$\Delta_j$, formado por $(\Phi_k)^{-1}(s'_{k,\ j-1})$, $(\Phi_k)^{-1}(s'_{k,j})$ e pelo ponto de intersecção — $x_j$ — das raias de $\Phi_k$ que saem desses pontos;

**Passo 2** obtenha o mapeamento $\varphi(\Phi_k)$ acrescentando a $\Phi_k$ um ponto de quebra adicional em cada $p_j$ definido no passo anterior e fazendo $\varphi(\Phi_k)(p_j) = q_j$.

Assim, para cada $j$ tal que o par $(s'_j,\ s'_{j-1})$ satisfaz as condições indicadas no texto do *passo 1* substituímos o segmento $[s'_j,\ s'_{j-1}]$ pela poligonal $j = [s'_j,\ q,\ s'_{j-1}]$. Isso determina que, ao invés de termos o quadrilátero de varredura reverso $[(\Phi_k^{-1}(s'_{j-1}),\ (\Phi_k)^{-1}(s'_j),\ s'_j,\ s'_{j-1}]$, teremos os quadriláteros $[(\Phi_k)^{-1}(s'_{j-1}),\ p_j,\ q_j,\ s'_{j-1}]$ e $[p_j,\ (\Phi_k)^{-1}(s'_j),\ s'j,\ q_j]$ que são diretos, dado que $q_j \in \Delta_j$. Portanto, para cada $j$ onde as condições do *passo 1* são satisfeitas, $\varphi(\Phi_k)$ será um mapeamento sem quadriláteros reversos com topo em $\rho_j$. Além disso, como $\Delta_j$ está contido em $Cj$, o mesmo acontece com $\rho_j$. Isso e o fato que não alteramos $[(\Phi_k)^{-1}(s'_{k,i})]$, $i = 0, \ldots, N-1$ fazem com que $\varphi(\Phi_k)$ seja $\mu$-equivalente a $\Phi_k$ e portanto $\mu$-equivalente a $\Gamma_k$ . Assim se $\Gamma_k$ for feito contrátil em função do procedimento *Correcting_Displacements* e da eliminação das intersecções de $PC_k$ com $DP_k$, descrita anteriormente, e se ele não gerar uma configuração do tipo representado na figura 4.25, então, o mesmo acontecerá com $\Phi_k$, o que determina que $\varphi(\Phi_k)$ será adequado.

Deve-se observar que $\varphi(\Phi_k)(DP_k)$ não será uma $\mu$-curva, mas, simplesmente, uma $\mu$-curva estendida. Isso, entretanto, não é um problema dado que os resultados que tornam mais fácil a rotulação de um loop de $PC_k$, requerem simplesmente que ela seja equivalente a uma $\mu$-curva, estendida ou não, que seja gerada por um mapeamento adequado.

Desse modo a única configuração que precisa ser evitada, por que não permite que quadriláteros reversos sejam eliminados passando-se a um mapeamento $\mu$-equivalente é aquela que ocorre quando $C' = O_j$ e que é representada na figura 4.25. Quando $s_{k,i}$ e $s'_{k,j}$, são, respectivamente, o snaxel de $S_k$ corrente — $s_{cur}$ — e o último vértice de $PC_k$ encontrado — $s'_{cur}$ — e representando $s_{k,\ i-1}$, $s'_{k,\ j-1}$ por $s_{prev}$ e $s'_{prev}$, essa configuração fica caracterizada por:

**i)** a célula $C$ cruzada por [ $s'_{prev}$, $s'_{cur}$ ] é seccionada em *i_cur*, o que conforme já vimos é indicado pela condição : *i_cur = i_prev*;

**ii)** $s_{cur}$, $s_{prev}$, $s'_{cur}$ e $s'_{prev}$ estão em arestas diferentes, mas todas adjacentes a um mesmo vértice de $C$;

**iii)** $s_{cur}$ e $s'_{cur}$ estão em arestas com a mesma direção, o mesmo ocorrendo com $s_{prev}$ e $s'_{prev}$.

Uma vez detectada essa configuração, para eliminá-la, devemos re-posicionar $T_k(s_{cur})$ de forma que o próximo vértice de $PC_k$ a ser gerado esteja numa aresta diferente da de $s'_{cur}$ mude de aresta ou $C$ deixe de ser seccionada em *i_cur*. Para fazer isso de forma rápida e de forma que essa mudança seja pequena, optamos fazer simplesmente:

$$T_k(s_{cur}) = s'_{cur} \ .$$

e continuamos o processo de geração dos vértices de $PC_k$. Essa re-definição fará com que $C$ deixe de ser seccionada em *i_cur*, mas exigirá que o procedimento de regularização de $TC_k$ lide com situações onde os vértices dessa curva podem estar sobre arestas da malha. Para evitar essas situações, deve-se perturbar $s'_{cur}$ levemente na direção do interior de $C$, antes de fazer a atribuição acima.

Resta estabelecer como detectar essa configuração indesejável de forma eficiente. Vamos mostrar que isso pode ser feito verificando-se além do fato que $C$ é seccionada em *i_cur*, as seguintes condições:

**I)** $E(s_{cur}) = E(s'_{cur})^{-1}$;

**II)** $E(s_{prev}) = E(s'_{prev})^{-1}$.

Vamos demonstrar a seguir, que isso é verdade no caso em que $E(s'_{cur}) = 2$. A demonstração para os demais casos é idêntica a menos de uma rotação dos elementos referidos nela. Arestas, linhas principais e células da malha referenciadas nessa demonstração estão representadas na figura 4.27.

Assuma então que já verificamos que $C$ é seccionada em *i_cur* e que a condição **I** acima é verdadeira. Nesse caso teremos $E(s_{cur}) = 3$. Considerando, então, que já foi aplicado um procedimento para eliminar $\mu$-whiskers, podemos assumir que $s'_{prev}$ e $s'_{cur}$ estão em arestas diferentes, e nesse caso $s'_{prev}$ pode estar em uma das arestas indicadas na figura por $a'_1$, $a'_2$ e $a'_3$. Como $s'_{cur}$ está em $H_0$ e $s'_{prev}$ está acima dela, o mesmo se pode dizer de $T_k(s_{prev})$, o que, como estamos fazendo $T_k$ $\mu$-limitado, significa que $s_{prev}$ está acima de $H_{-1}$. Isso acarreta que $s_{cur}$, que tem de estar numa aresta horizontal está em $H_0$ ou acima dela, pois, como $E(s_{cur}) = 3$ , de $s_{prev}$ para $s_{cur}$, $S_k$ deve subir. Por outro lado, como $E(s'_{cur}) = 2$, $PC_k$ está descendo quando passa por $s'_{cur}$, o que implica que $T_k(s_{cur})$ está abaixo de $H_0$. Em conseqüência, de novo porque $T_k$ é feito $\mu$-limitado, $s_{cur}$ tem de

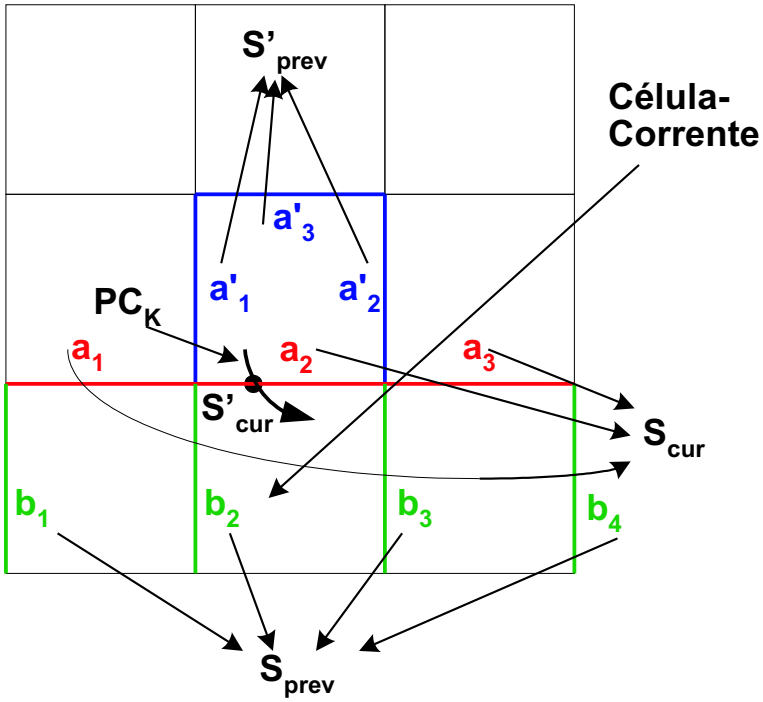


Figura 4.27: Possibilidades para a localização de $s_{cur}$, $s'_{prev}$ e $s_{prev}$.

estar abaixo de $H_1$. Assim, considerando as duas restrições estabelecidas quanto a sua posição, $s_{cur}$, tem, obrigatoriamente, que estar sobre $H_0$. Isso acarreta que $s_{prev}$ deve estar abaixo dessa reta, o que inviabiliza a possibilidade de $s'_{prev}$ estar em $a'_3$. De novo porque $T_k$ é $\mu$-limitado se $s_{cur}$ está em $H_0$, então ela deve se situar numa das arestas da figura 4.27 indicadas por $a_1$, $a_2$ e $a_3$. Caso contrário $T_k([s_{prev}, s_{cur}])$ não poderá intersectar a célula $C$. Pela análise feita até aqui temos, em relação às arestas que podem conter $s_{cur}$ e $s'_{prev}$, seis possibilidades a considerar(três alternativas no caso de $s_{cur}$ e para cada uma delas duas para o caso de $s'_{prev}$). A análise das possíveis localizações de $s_{prev}$ permitirá reduzir, consideravelmente, esse número.

Conforme já vimos $s_{prev}$ tem de estar abaixo de $H_0$. Entretanto, se ele estiver sobre $H_{-1}$ ou abaixo, então, $T_k(s_{prev})$ não poderá estar acima de $H_0$, como é requerido, lembrando uma vez mais que $T_k$ é $\mu$-limitado. Assim, $s_{prev}$ deve estar numa aresta vertical entre $H_0$ e $H_{-1}$. Considerando, então, que $s_{prev}$ e $s_{cur}$ devem estar numa mesma célula, temos que o conjunto de pares de arestas, a primeira contendo $s_{cur}$ e a segunda, $s_{prev}$, que ainda são possíveis, podem ser divididos nas duas categorias que são indicadas abaixo, usando a notação dada na figura 4.27.

**I)** $(a_i\ ,\ b_i),\ i = 1, 2, 3.$

**II)** $(a_i\ ,\ b_{i+1}),\ i = 1, 2, 3.$

Tendo em vista que o procedimento *Correcting_Displacements* já foi aplicado antes, podemos verificar que os pares da categoria **I** também não são possíveis. É que , como estamos assumindo que os vértices de $S_k$ são ordenados no sentido anti-horário, se $s_{prev} \in b_i$ e $s_{cur} \in a_i$, $i = 1, 2, 3$, então o in-vertex de *sprev* estará sobre $H_{-1}$. Em vista disso, as quatro células onde $T_k(s_{prev})$ pode estar, se o deslocamento desse snaxel, foi submetido ao procedimento *Correcting_Displacements*, ficam todas abaixo de $H_0$. O par $(a_3,\ b_4)$ também pode ser descartado em função de:

**i)** Como $T_k$ é $\mu$-limitado, nesse caso $T_k(s_{prev})$ estará a direita da vertical $V_1$.

**ii)** O mesmo acontecerá com $T_k(s_{cur})$, devido ao procedimento *Correcting_Displacements*.

Em vista de **i** e **ii** , $T_k([s_{prev}, s_{cur}])$ não poderá, então, intersectar a célula $C$. Desse modo podemos descartar a possibilidade de $s_{cur}$ estar em $a_3$, restando como pares viáveis, $(a_1\ ,\ b_2)$ e $(a_2\ ,\ b_3)$. Para cada um deles devemos considerar, ainda, as duas possibilidades para a localização de $s'_{prev}$, que pode estar ou em $a'_1$ ou em $a'_2$.

Mas, se $s_{prev}$ está em $b_i$, $i = 2, 3$, então, forçosamente, $s'_{prev}$ estará em $a'_{i-1}$, porque a outra possibilidade implicaria que a raia $[s_{prev},\ T_k(s_{prev})]$ cortasse tanto $V_0$ como $V_1$, o que não é possível com $T_k$ é $\mu$-limitado. Assim, o filtro determinado pela condição **I** e pelo fato de $C$ ser seccionada em *i_cur*, seleciona apenas duas configurações, que no caso em que $E(s'_{cur}) = 2$, são as seguintes.

**i)** $s_{cur} \in a_1$, $s_{prev} \in b_2$, $s'_{cur} \in a_2$ e $s'_{prev} \in a'_1$. Esta configuração é, exatamente, aquela que queremos identificar, com vistas à sua eliminação.

**ii)** $s_{cur} \in a_2$, $s_{prev} \in b_3$, $s'_{cur} \in a_2$ e $s'_{prev} \in a'_2$.

Apenas para diferenciar essas duas configurações é que a condição **II** é usada pois enquanto em **i** temos $E(s_{prev}) = E(s'_{prev})^{-1}$, na com figuração **ii**, $E(s_{prev}) = E(s'_{prev})$. Demonstramos assim que as três condições dadas são suficientes para caracterizar a configuração indesejável. Entretanto, o uso da condição **II** é, na verdade, dispensável. Suponha que não testamos a condição **II** e por causa disso fazemos $T_k(s_{cur}) = s'_{cur}$, também no caso da configuração **ii**. O efeito de fazer isso sobre a curva $PC_k$ poderá, então, ser de um dos dois tipos indicados abaixo:

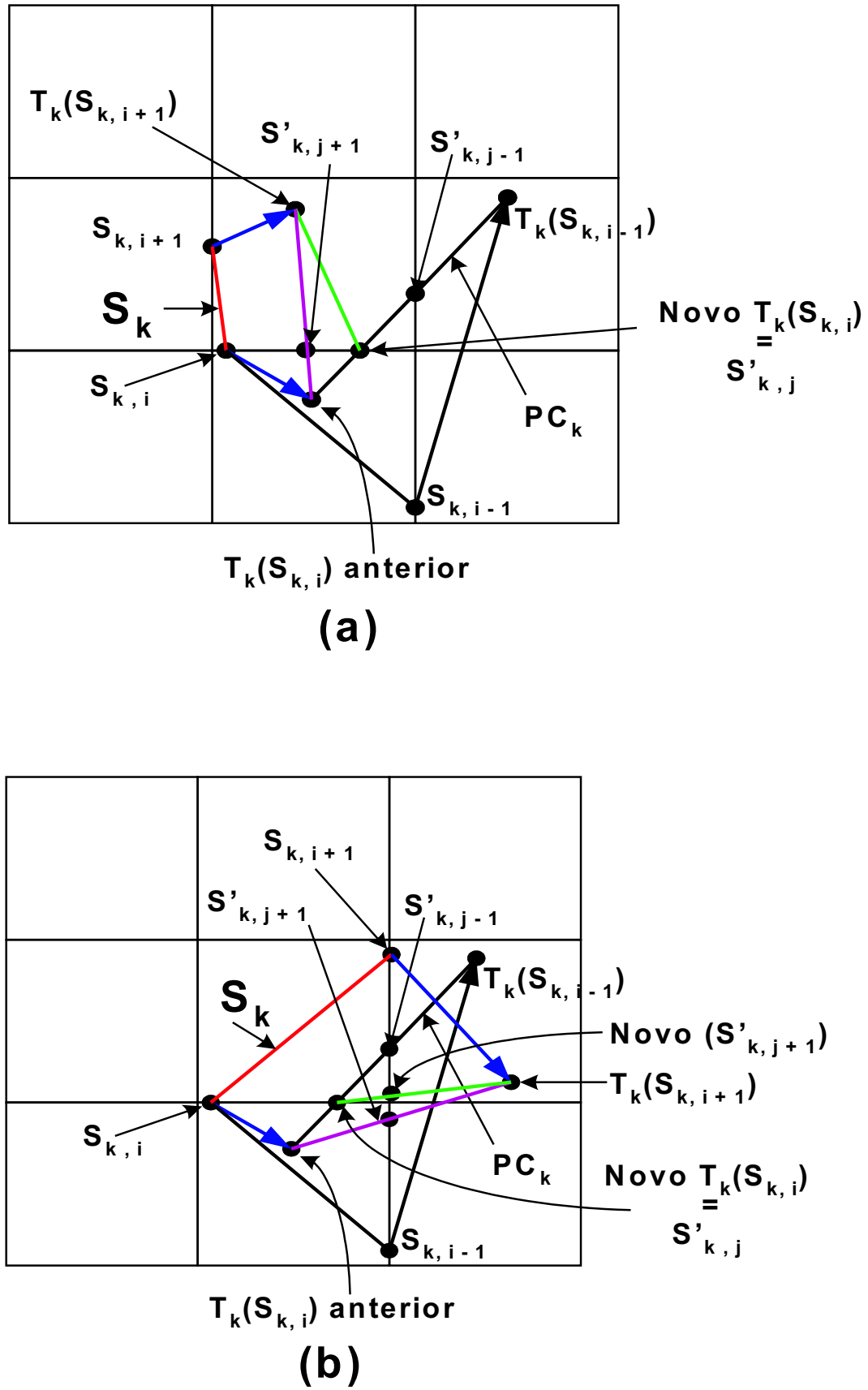


Figura 4.28: Possíveis efeitos de se fazer $T_k(s'_{k,j}) = s'_{k,j}$ também no caso em que $E(s'_{k,\ j-1}) = E(s_{k,\ j-1})$.

**1)** podemos, simplesmente, reduzir o comprimento de um $\mu$-whisker ou o eliminaremos, como acontece no exemplo da figura 4.28.(a);

**2)** podemos ainda, fazer com que o quadrilátero de varredura $[s_i^k,\ s_{i+1}^k,\ (\Gamma_k)(s_i^k),\ (\Gamma_k)(s_{i+1}^k)]$ que eventualmente, cobre o in-vertex de $s_i$ — $v$, na figura 4.28.(b) — seja substituído por um outro que não cobre esse vértice. Mas, dado que $v$ já é coberto por $[s_{i-1},\ s_i,\ (\Gamma_k)(s_{i-1}),\ (\Gamma_k)(s_i)]$, isso não altera o conjunto de vértices cobertos pelos quadrilátero de varredura de $\Gamma_k$. Assim, as bordas dos conjuntos de pontos já explorados num caso e no outro serão $\mu$-equivalentes.

Por conseguinte, podemos considerar que o uso da condição **II** é opcional. Tendo em vista todas as considerações que fizemos nesta seção relativas a eliminar, tanto intersecções de $PC_k$ e $DP_k$, como quadriláteros reversos, não removíveis, pela passagem a um mapeamento $\mu$-equivalente, temos que o seguinte código deve ser executado a cada vértice de $PC_k$ que é gerado. Na apresentação desse código *t_cur* representa $T_k$(*s_cur*) e as demais variáveis já foram definidas anteriormente.

**Procedure Making $\Gamma_k$ Adequate**

```
{if (i_cur = i_prev) then
    if E(s'_cur) = (E(s_cur))^-1)
      t_cur = s'_cur;
    else if (C(s'_cur) = C(s_cur) and E(s'_cur) = E(s_prev))
            { s'_prev = s_prev;
            s'_cur = s_cur;
            }
}
```

Esse procedimento é muito menos oneroso do que testar a cobertura de vértices por quadriláteros de varredura como é feito pelo método clássico para controlar a topologia de uma T-snake. Isso é particularmente verdade se considerarmos que para a maioria dos vértices gerados o teste inicial resulta falso e assim, nada mais é feito.

Finalmente uma palavra a respeito de se tornar $\Gamma_k$, $\mu$-equivalente não a um mapeamento adequado, mas a um mapeamento ideal que possui melhores propriedades no que diz respeito a rotulação dos loops como será visto no próximo capítulo. Nem sempre é possível fazer isso de forma que a imagem gerada seja uma $\mu$-curva, embora, é claro, sempre se possa fazer, se relaxamos essa condição para que essa imagem seja apenas, uma

$\mu$-curva estendida. Mesmo assim, consideramos que detectar as situações em que o mapeamento é apenas adequado, mas não ideal, e torná-lo, então ideal é mais custoso do que as eventuais dificuldades que o fato de não fazermos isso, acarreta com respeito a rotulação dos loops. Essas dificuldades conforme veremos estão restritas às folhas da loop-tree de $PC_k$, a ser definida, precisamente no próximo capítulo. Tornar $\Gamma_k$, $\mu$-equivalente a apenas um mapeamento adequado, é definitivamente, uma solução melhor para o trade-off entre o custo de fazer essa transformação e o de rotular corretamente os loops.

# Capítulo 5

# Loop Trees

## 5.1 Loop-Trees e o Processo de Rotulação

Uma **Loop-tree** de uma curva fechada $C$ sem nenhum ponto de auto-intersecção múltiplo é um grafo orientado que pode ser obtido através do seguinte processo:

0. escolha um ponto $s$ em $C$ que não seja auto-instersecção[1] e uma direção $D$ seja no sentido horário ou no anti-horário para percorrer a curva. Faça também $o = s$;

1. partindo de $o$ percorra $C$ na direção $D$ até alcançar um ponto $x$ que já tenha sido visitado. Crie então, um nó para representar o loop simples ($L$), formado pela parte de $C$ que foi percorrida entre duas visitas a $x$. Se $x \neq s$ simplifique $C$, concentrando todo o loop $L$ em $x$. Faça, então $o = x$ e repita o Passo I. Se $x = s$ execute o Passo II;

2. para cada par de loops $(L, L')$ tal que $L$ tenha sido concentrado em um ponto de $L'$ no Passo I, crie uma aresta orientada partindo do nó que representa $L$ para o que representa $L'$.

Se $x \neq s$, então $x$ é um ponto de auto-intersecção de $C$ e como se está assumindo que todos esses pontos são simples, o out-degree de qualquer nó exceto aquele do loop que contém $s$, será um. Além disso o grafo obtido não pode conter circuitos pois cada loop encontrado é imediatamente colapsado em um ponto o que significa que o nó associado a ele não pode ter nenhum descendente referente a loops gerados depois dele como é o caso de seu nó pai. Em vista desses dois pontos pode-se concluir que o grafo obtido é uma

[1] Estamos assumindo que uma curva fechada é a imagem de uma função contínua localmente inversível, $F$ definida numa circunferência. Percorrer a curva num dado sentido significa percorrer os seus pontos na ordem em que suas imagens inversas por $F$ são atingidas quando a circunferência é percorrida nesse sentido.

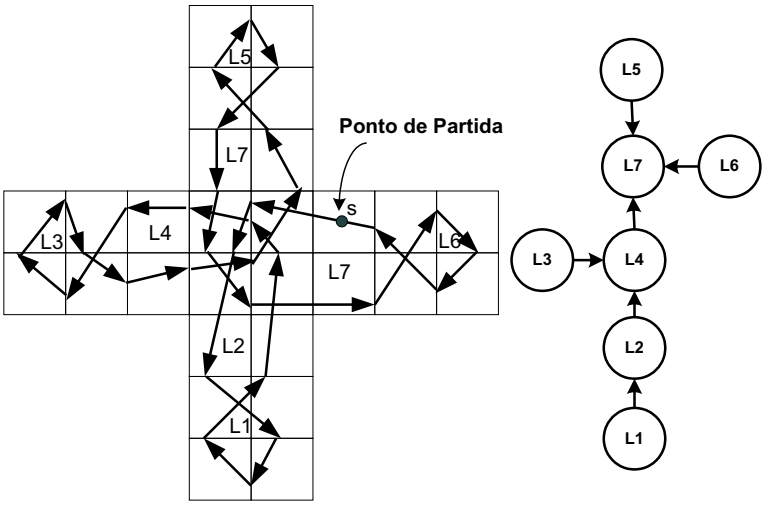


Figura 5.1: Uma curva e sua loop-tree.

arborescência com raiz no nó do loop no qual $s$ está. Uma curva e uma de suas loop-trees podem ser vistos na figura 5.1.

Loop-trees de diferentes topologias ou com a mesma topologia, mas com associações loop-nó diferentes podem ser obtidas para a mesma curva dependendo do ponto inicial escolhido e da circulação usada ao se percorrer a curva.

Essa dependência é ilustrada nas figuras 5.2 e 5.3 dadas a seguir. As loop-trees obtidas quando ser percorre a curva a que se refere a figura 5.2 no sentido horário a partir dos pontos $s$ e $s'$, tem topologias completamente distintas. Essas árvores e os loops correspondentes aos seus nós são representados em 5.2.a e 5.3.b, respectivamente. Na curva a que se refere a figura 5.3, mantendo o mesmo ponto de partida $s$ e variando apenas o sentido de percurso, já obtemos loop-trees de topologias distintas. Se retirarmos dessa curva o loop $L$ indicado na figura, a topologias nos dois casos passariam a ser as mesmas, mas ainda assim em qualquer isomorfismo entre eles os loops associados a nós correspondentes seriam completamente distintos. Observe nas duas figuras que essa falta de unicidade é decorrente da existência de pontos de auto-intersecção como $s_1$ e $s_2$ , no caso da figura 5.2 e $s_3$ e $s_4$, no caso da figura 5.3. Esclarecemos que essas situações podem ocorrer ainda que a curva seja obtida pela aplicação de um mapeamento ideal contrátil a uma curva simples.

Seria muito inadequado se as novas snakes geradas pelo algoritmo na iteração seguinte e que correspondem aos loops abertos da iteração atual, dependessem do snaxel onde o processo de atualização é iniciado ou não da direção na qual a curva $PC_k$ é percorrida. Felizmente, este problema pode ser evitado se $\Gamma_k$ for adequado e contrátil, como pode ser

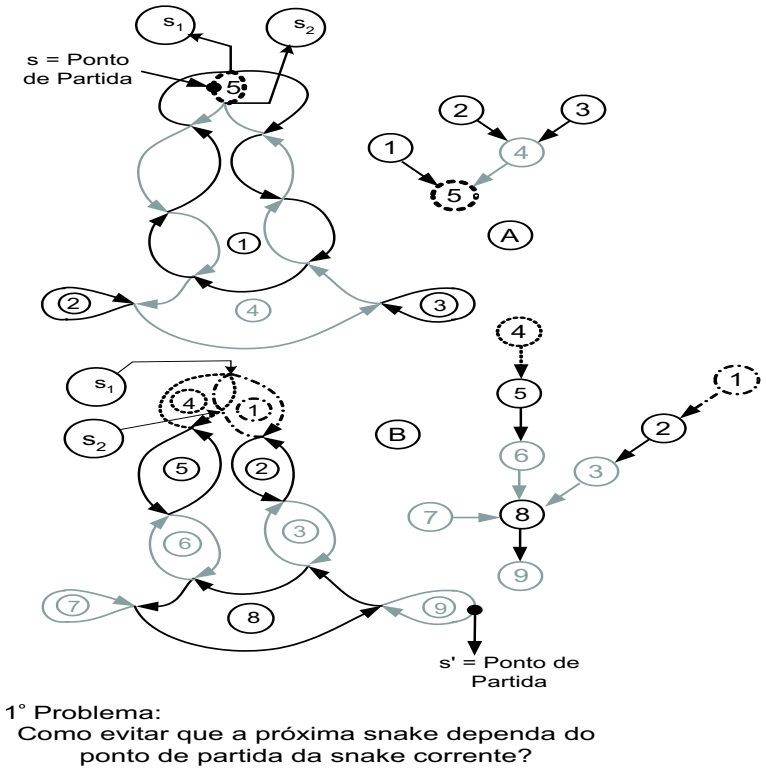


Figura 5.2: Loop-Trees são dependentes do ponto a partir de onde se começa a percorrer a curva.

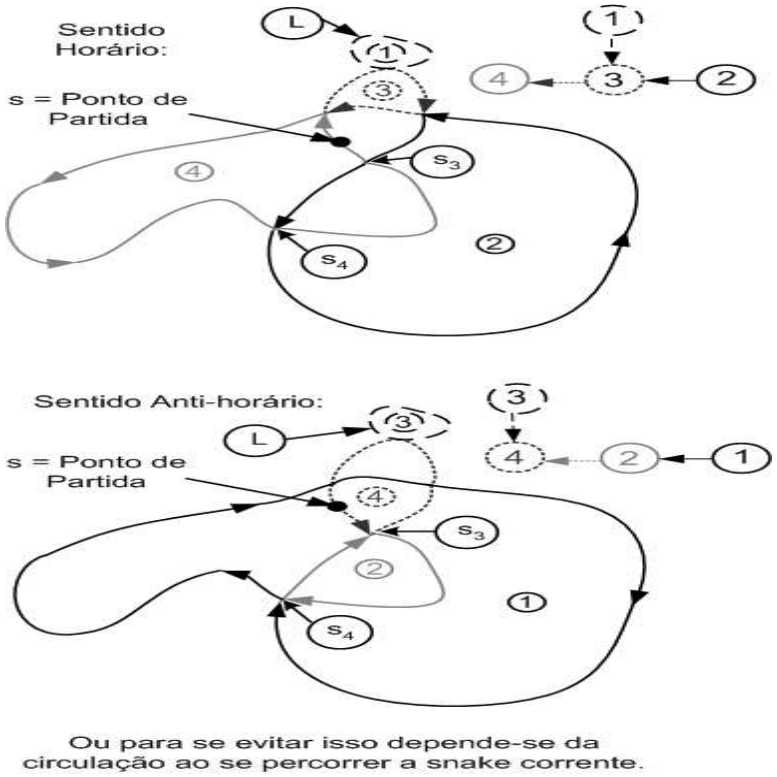


Figura 5.3: Loop-Trees também dependem do sentido usado para percorrer a curva mesmo tomando-se o mesmo ponto de partida.

derivado do lema 5.1.1.

**Lema 5.1.1** *Se $PC_k$ é produzida pela aplicação de um mapeamento $\Gamma_k$ adequado e contrátil a $\mu$-curva regular $\mu - D(S_k)$ e $L$ é um loop representado numa loop-tree $T$ de $PC_k$ e que foi rotulado como aberto então:*

**(i)** *existe um loop $L'$, $\mu$-equivalente a $L$, tal que $L'$ é disjunto de qualquer outro loop representado em $T$;*

**(ii)** *sejam $C_1$ e $C_2$ duas sub-árvores de $T$ obtidas com a retirada do nó de $T$ referente a $L$ e $L_1$ e $L_2$, dois loops de $PC_k$ representados por nós de $C_1$ e $C_2$, respectivamente. Então $L_1$ e $L_2$ tem loops $\mu$-equivalentes que são disjuntos.*

Para simplificar o raciocínio vamos identificar um loop aberto e seus descendentes com seus $\mu$-equivalentes de menor cobertura, isto é que delimitam a menor área possível[2]. Então, os resultados do lema 5.1.1 permitem concluir que qualquer loop aberto achado pelo algoritmo, não somente é disjunto (parte (i) do lema 5.1.1) dos outros, mas também desconecta a curva, se for removido (parte (ii) do lema 5.1.1). Loops que satisfazem essas duas propriedades certamente são representados em qualquer "Loop-Tree" da curva $PC_k$. Assim, as snakes que são desenvolvidas no estágio $k+1$ e, que são exatamente os loops abertos de $PC_k$, se tornam independentes de onde o percurso de $PC_k$ é iniciado e da direção escolhida para percorrê-la. Diferentes loops fechados, entretanto, podem estar representados em duas "Loop-Trees" da mesma curva mas isso não tem importância já que eles não serão mais usados para se determinar a evolução da snake. O exemplo da figura 5.1 serve para ilustrar essa questão. Suponha que a poligonal representada nessa figura é uma curva $PC_k$ e que o contorno da porção da malha representada é exatamente $\mu - D(S_k)$. Observe que, nesse caso, $PC_k$ pode ser obtida a partir de $\mu - D(S_k)$ por um mapeamento ideal. Em relação aos loops representados na árvore de loops da poligonal, os abertos, são os correspondentes às folhas. Eles são disjuntos dos demais e estarão referenciados em qualquer loop-tree da curva que venha a ser obtida começando-se o percurso de um novo ponto de partida ou se invertendo sua direção os loops correspondentes aos demais nós são fechados. É fácil verificar que o loop $L_7$ correspondente a raíz da árvore não estará representado numa loop-tree da curva obtida percorrendo a partir de um ponto de $L_3$. A árvore-de-loop fornece mecanismos para fazer a rotulação dos loops de $PC_k$ dentro da perspectiva, altamente desejável, de que a simples análise da curva — especificamente de suas auto-intersecções — é suficiente para isso. A situação ótima se-

[2]Esta curva está definida se assumirmos tal como foi feito na definição de $\mu - D(S_k)$ que existe entre os pontos de uma aresta da malha diferentes de um de seus vértices $V$, um que seja o mais próximo de $V$.

ria que fosse suficiente examinar apenas o traçado da curva projetada, em si, sem que seja necessário para nenhum de seus pontos — $p$ — nem analisar uma vizinhança de $p$, nem saber onde está ${\Gamma_k}^{-1}(p)$. Nessas condições, se poderia mesmo, gerar primeiro toda a curva projetada e depois rotular os seus loops adequadamente, usando apenas a estrutura auxiliar para encontrá-los. No caso de $\Gamma_k$ ser um mapeamento ideal, essa situação ótima é plenamente atingida, enquanto que para os mapeamentos apenas adequados se faz necessária uma análise local numa vizinhança da célula em que se identificou a formação de um loop correspondente a uma folha da árvore. Essa análise, é feita, empregando-se a estrutura auxiliar e é O(1) não comprometendo, portanto, a performance computacional do método. Vamos, deixar para depois a questão de rotular as folhas da árvore de loops e nos concentrar em rotular os outros loops associados aos outros nós. A idéia é identificar o rótulo de um loop representado numa loop-tree a partir do rótulo de seus filhos, que são determinados primeiro. Se todos os loops são disjuntos entre si, o rótulo de um nó pai de um nó é o oposto do rótulo do nó. No caso geral, entretanto, a situação não é tão favorável mas podemos nos valer dos dois resultados abaixo.

**Lema 5.1.2** *O pai de um nó aberto é um nó fechado, mas nós fechados podem ter tanto um pai aberto como fechado.*

**Lema 5.1.3** *Se ambos, pai e filho, são fechados, eles devem se intersectar.*

Assim, se um nó que tem um filho aberto, o loop que este representa pode ser rotulado como fechado. O caso no qual um nó tem somente filhos fechados, pode ser tratado, se soubermos identificar suas intersecções com os seus filhos. Se elas existirem o nó será fechado, senão será aberto.

Determinar se intersecções entre um pai e um filho fechado, existem ou não — como é necessário para se poder usar o resultado do lema 5.1.3 — pode ser feito com a ajuda da estrutura auxiliar. Basta conhecer os vértices inicial e final de cada loop fechado já encontrado e cujo pai ainda está em processo de construção. Dado um loop fechado considere o conjunto de filhos dele, que também são fechados, já foram gerados, mas que afora seu ponto de junção, ainda não foram interceptados por $PC_k$ até um dado momento da construção dessa curva. Teremos então que como está indicado na figura 5.5 que:

**1)** os filhos desse conjunto vão ser posteriormente cortados pelo pai na ordem inversa àquela em que foram gerados;

**2)** se o vértice inicial de $PC_k$ está fora de um dado filho, o pai vai interceptar esse filho

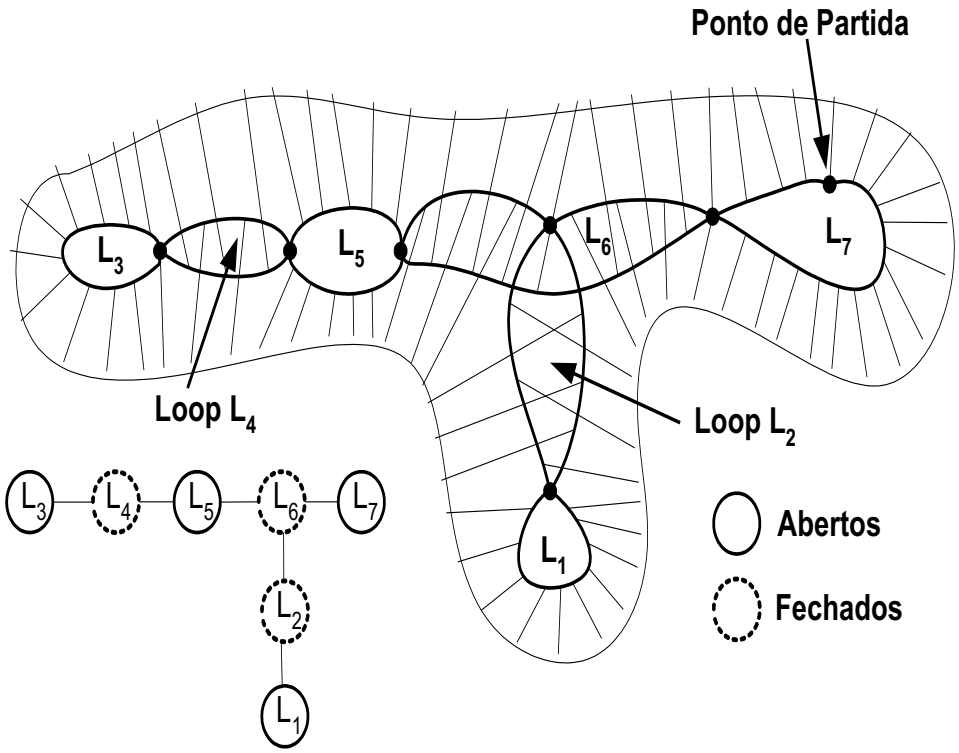


Figura 5.4: Os loops $L_2$ e $L_4$ são fechados mas enquanto o pai de $L_2$ é o loop fechado $L_6$ o pai de $L_4$ é o loop aberto $L_5$. A diferença é que enquanto $L_4$ e $L_5$ são disjuntos $L_2$ e $L_6$ se interceptam como indica o lema 5.1.3.

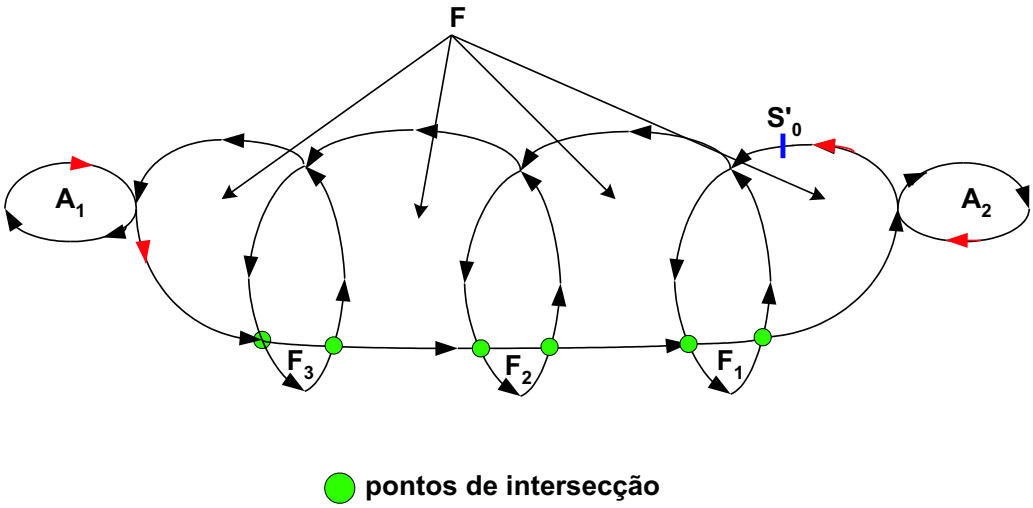


Figura 5.5: O loop fechado $F$ corta seus filhos fechados na ordem inversa aquele em que eles são gerados. Cada um desses filhos é cortado duas vezes.

em, exatamente, dois pontos.

Em vista desses fatos, mantenha numa pilha os loops fechados já gerados tais que:

**1)** o pai na loop-tree de PCk ainda não foi determinado;

**2)** eles não foram ainda intersectados duas vezes pela curva projetada, afora seu ponto de junção.

Fazendo isso é possível determinar a ocorrência das interseções em questão, considerando toda a árvore de loops de $PC_k$, em um tempo proporcional ao número de loops fechados dessa curva. No capítulo 6 se descrevem detalhes operacionais de como isso pode ser feito.

Na seção seguinte estudamos em detalhes os loops. Mostrando quão específica é a sua estrutura conseguimos elementos para fundamentar metodologias de rotulação que não usam a loop-tree de $PC_k$. Nesse estudo vai se assumir que $\Gamma_k$ é sempre adequado e $\mu$-limitado sem que isso precise ser explicitado. Além disso daqui por diante vamos nos referir a loop-tree de $PC_k$ por $LT(PC_k)$.

## 5.2 Loops Fechados

Assumindo que $PC_k$ não tem trechos coincidentes, dado um de seus loops — $L$ — chamaremos de imagem inversa de $L$ por $\Gamma_k$ corrigida, à qual nos referiremos por — $L^{-1}$ — ao conjunto resultante de se excluir de $(\Gamma_k)^{-1}(L)$ seus pontos isolados. Seja $p$ um ponto de intersecção entre loops $L$ e $L'$, ambos fechados, os quais são pai e filho na $LT(PC_k)$ como está mostrado na figura 5.6. Se $p$ também não é o ponto de junção entre os dois, então, uma das raias chegando a $p$ parte de um ponto isolado de $(\Gamma_k)^{-1}(L)$ e portanto não pertence a $L^{-1}$. Esse ponto pertencerá, sim, a $L'^{-1}$. Loops fechados são caracterizados pelo fato de serem inteiramente cobertos pelas raias de $\Gamma_k$ que saem de $L^{-1}$.

Chamamos de uma singularidade de um loop $L$ a um ponto $p$ de seu traçado ao qual chegam mais de uma raia partindo de $L^{-1}$, ou, se a ele chega uma única raia partindo de um ponto de $L^{-1}$ onde $\Gamma_k$ não é ideal. Nesse segundo caso podemos transformar $\Gamma_k$ em ideal de uma forma em que $p$ se torna o ponto de junção entre $L$ e um loop arbitrariamente pequeno. Feito isso, $p$ passará a ter duas raias e como elas podem ser feitas arbitrariamente próximas de sua raia original, diremos que, nesse segundo caso, $p$ tem duas raias coincidentes. Observamos que pela própria maneira como $\Gamma_k$ é construído, apenas vértices de $PC_k$ podem ter raias coincidentes. As singularidades do outro tipo são os pontos de

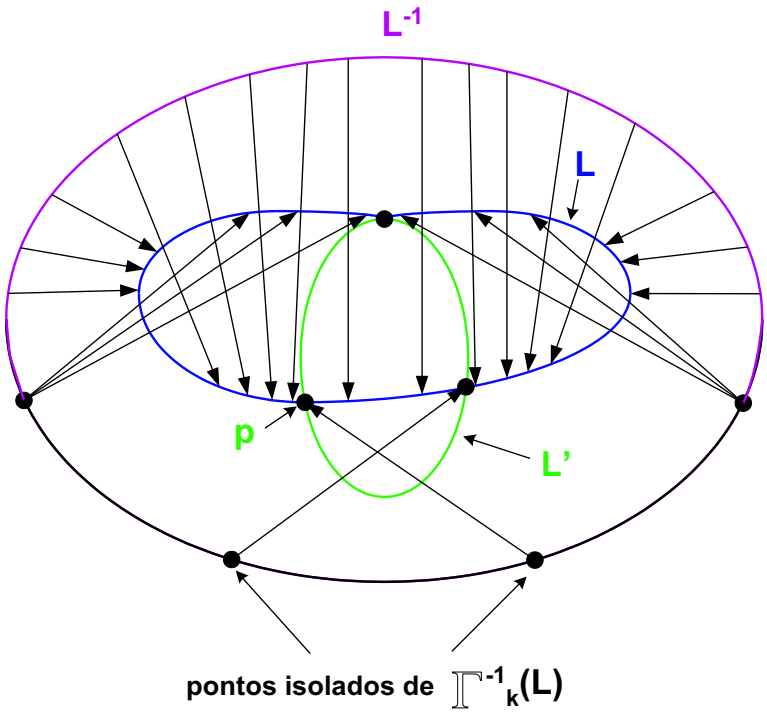


Figura 5.6: $L^{-1}$ é obtido retirando de $\Gamma_k^{-1}(L)$ os pontos isolados.

junção de $L$ com loops adjacentes. Se $L$ é um loop fechado, afora um conjunto discreto de exceções, as raias de $\Gamma_k$ que partem $L^{-1}$, atingem os pontos de $L$ vindo do interior do loop e, portanto, devem cortar uma aresta de $L$ diferente daquela que contém seu ponto final. O conjunto de raias partindo de $L^{-1}$ e que sejam totalmente externas a $L$ é ainda mais reduzido podendo como iremos ver, ser limitado em 4 — duas delas chegando a um ponto e duas a outro. É que se pode mostrar que o ponto final de uma dessas raias tem de ser uma singularidade de $L$ e dessas, podemos excluir os pontos de junção com filhos também fechados. Isso, porque as raias que chegam a eles, obrigatoriamente, tem de cortar $L$. As demais singularidades podem ser divididas em dois grupos:

**1)** aquelas em que há necessariamente raias totalmente externas. É o caso dos pontos com raias coincidentes;

**2)** aquelas em que pode ou não haver uma raia externa: é o caso dos pontos de junção de $L$ com seu pai na $LT(PC_k)$ ou com filhos abertos. Mas se um desses pontos — p — não tem uma raia externa, é possível substituir $\Gamma_k$ por um mapeamento $\mu$-equivalente que tem raias chegando ao ponto de sua imagem correspondente a $p$, que são externas ao loop correspondente a $L$. Observamos que não é possível fazer essa substituição no caso das junções com filhos fechados.

Vamos nos referir ao fato de que, se necessário via uma $\mu$-equivalência, nos dois casos existem raias externas, chamando as singularidades desses dois tipos de externas.

Se, dado um ponto — $p$ — de um loop $L$, conseguirmos garantir, a priori, que ele não é uma singularidade externa de um loop fechado, então, podemos rotular $L$ corretamente apenas verificando se a raia que chega até ele, intersecta o loop em algum outro lugar ou não. No caso em que $PC_k$ tem um loop único — o qual, portanto, não pode ser rotulado a partir do label de seus descendentes na $LT(PC_k)$ — podemos garantir isso, meramente, fazendo $p$ não ser um vértice de $PC_k$. É que se não existem junções então todas as singularidades externas de um loop fechado são pontos com raias coincidentes que só podem ser vértices.

Os dois lemas dados a seguir, relativos às singularidades externas(abreviadamente : $s-es$) de um loop fechado, são válidos se assumirmos que o primeiro vértice de $PC_k$ está fora do interior da região delimitada por ele. Chamemos essa condição de qualificação de $C1$. Ela sempre foi satisfeita durante toda a evolução das snakes, em todos os testes que realizamos. Mas, mesmo que ela,não seja atendida resultados similares aos desses lemas, só que de formulação mais elaborada, podem ser obtidos.

**Lema 5.2.1** *Se $\Gamma_k$ é adequado e $\mu$-limitado, um loop fechado de $PC_k$ que satisfaça a condição $C1$ tem exatamente duas singularidades externas. As raias partindo de $L^{-1}$ e chegando a pontos do loop, que não essas singularidades, o seccionam em duas partes cada uma contendo uma das $s-es$.*

Isso significa que, nas condições do lema 5.2.1, um loop fechado tem no máximo dois filhos abertos ou singularidades com raias coincidentes. Mais ainda, se esse loop não for a raiz de $LT(PC_k)$ então como a junção com seu pai também é uma singularidade externa, esse número se reduz a um. Em contrapartida um loop fechado pode ter um número de filhos fechados da ordem do número de vértices de $PC_k$, como, aliás, também ocorre com os loops abertos.

Se $\Gamma_k$ é $\mu$-limitado a região definida por um loop fechado não pode conter nenhuma célula inteira. Esse fato, entretanto, não é suficiente para caracterizar a particularidade dessas regiões que fica melhor traduzida pelo lema dado a seguir.

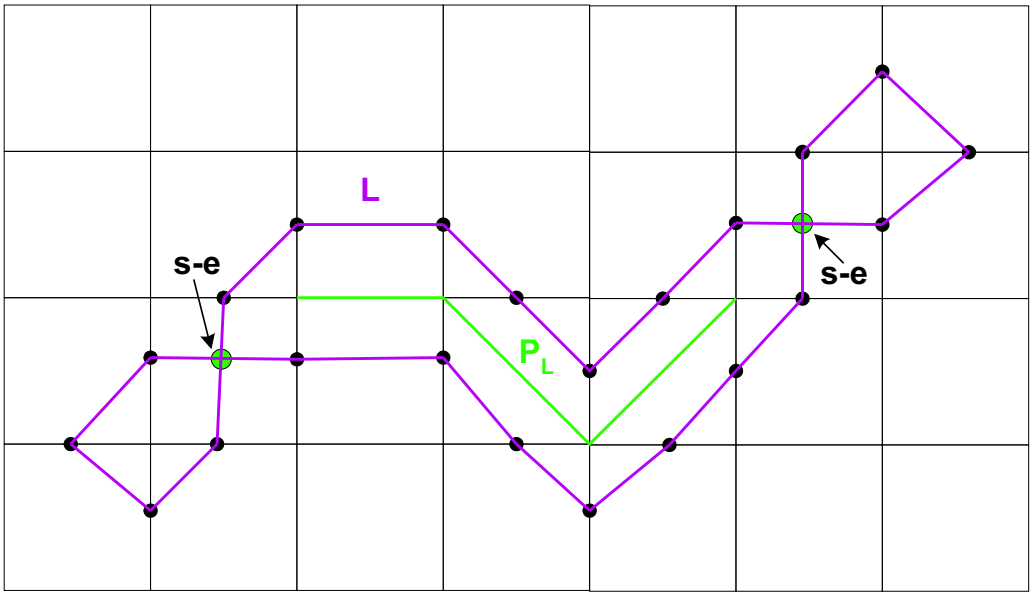


Figura 5.7: Loop fechado $L$ e a poligonal $P_L$.

**Lema 5.2.2** *Dado um loop fechado L considere o conjunto — $A_L$ — formado pelas:*

**1)** *arestas da malha interiores a L;*

**2)** *diagonais de células duplamente cruzadas por L e que são interiores a ele.*

*Se $\Gamma_k$ é adequado e $\mu$-limitado e L satisfaz a condição C1 então:*

**1)** *os segmentos em $A_L$ formam uma poligonal simples — $P_L$ — cujas extremidades são vértices das células contendo as s – es de L. Ver figura 5.7;*

**2)** *nenhuma aresta de $P_L$ é interior a outro loop mas as poligonais relativas aos loops fechados que são filhos de L tem uma extremidade num vértice não extremo de $P_L$.*

O fato de que os segmentos de $A_L$ formam uma única poligonal simples aberta e não uma estrutura com ramificações, permite eliminar configurações, como a representada na figura 5.8.(a), que satisfazem a condição de não serem mais largas que uma célula e que, portanto, numa primeira análise seriam cabíveis. Além disso, dado que as singularidades externas do loop devem estar nas extremidades dessa poligonal, isso significa que a junção com o pai ou com um filho aberto não pode estar em células disjuntas dessas extremidades o que elimina outros casos, igualmente plausíveis a primeira vista, como o representado na figura 5.8.(b).

Atendidas as condições dos lemas 5.2.1 e 5.2.2 temos que, a grosso modo, um loop fechado é uma faixa estreita em torno de uma poligonal formada por arestas e diagonais da malha. Ela se estende, se o loop não é a raiz da $LT(PC_k)$, desde sua junção com o pai

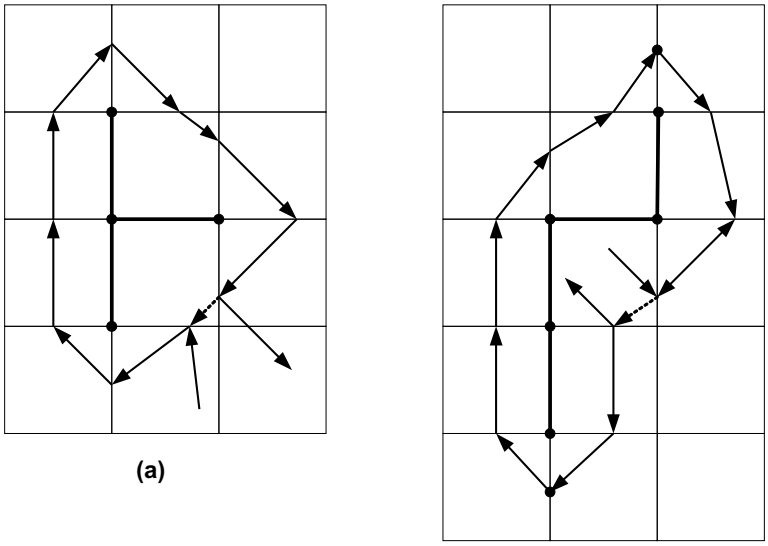


Figura 5.8: Configurações impossíveis para um loop fechado devido ao lema 5.2.1: O caso da figura A não pode ocorrer porque o conjunto de arestas internas ao loop tem uma bifurcação. O da figura B não é possível porque a junção do loop não está numa célula adjacente a uma extremidade de $A_L$.

nessa árvore até aquela com um filho aberto ou até um vértice com raias coincidentes. Se o loop for a raiz, ela vai de um filho aberto a outro ou a um ponto com raias coincidentes ou então, liga dois pontos com essa propriedade.

O fato da estrutura dos loops fechados ser bem menos diversificada do que, em princípio, se poderia pensar, permite demonstrar a validade de formulações simples para rotular um loop, que não utilizam a loop-tree de $PC_k$. Essas formulações, em vez disso, simplesmente analisam o que acontece na vizinhança da junção de um loop com o pai. Algumas delas serão descritas na seção seguinte onde se descrevem procedimentos para rotular as folhas de $LT(PC_k)$. Em relação a esses procedimentos, ressaltamos que são os resultados desta seção que permitem que se chegue ao lema 5.3.2, que é onde se fundamenta aquele que é proposto para rotular folhas que são loops próprios de $PC_k$.

## 5.3 A rotulação de Folhas

Sabendo como rotular as folhas de $LT(PC_k)$ teríamos como começar um processo que, empregando os lemas 5.1.2 e 5.1.3, nos permitiria rotular todos os nós dessa árvore. Em relação a isso, o resultado a seguir mostra que isso pode ser feito de forma imediata se $PC_k$ for a imagem de um mapeamento ideal e contrátil.

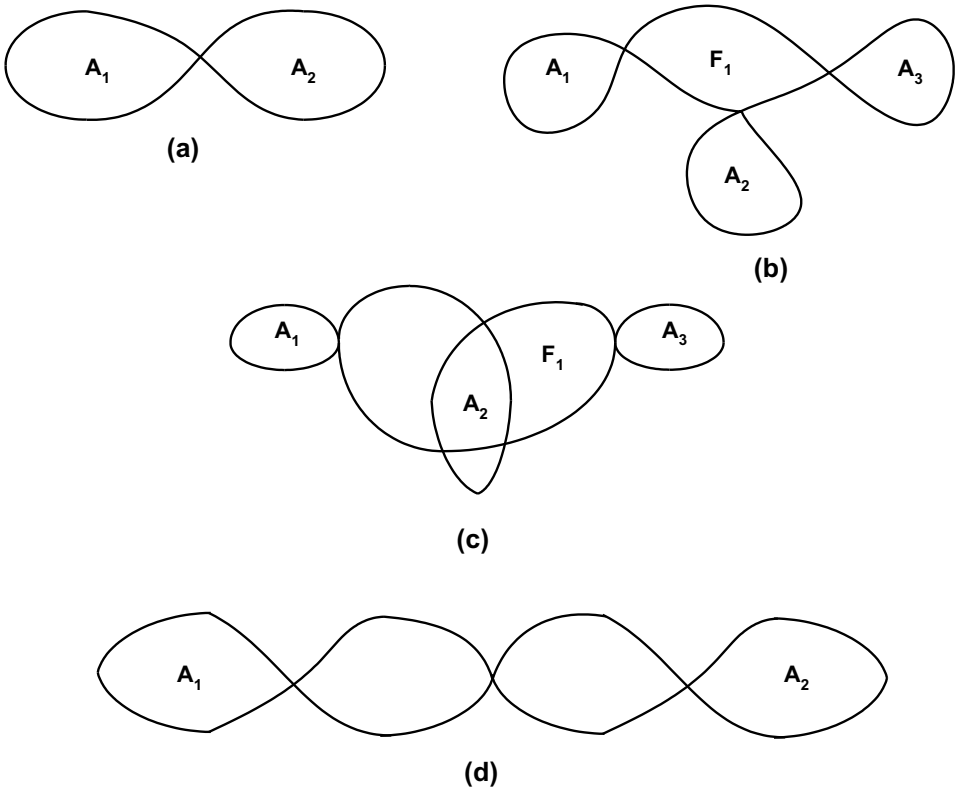


Figura 5.9: Topologias impossíveis para $PC_k$ quando $\Gamma_k$ é ideal.

**Lema 5.3.1** *Se $\Gamma_k$ for $\mu$-equivalente a um mapeamento ideal e contrátil então toda a folha de $LT(PC_k)$ será um loop aberto.*

Na grande maioria das instâncias de evolução de uma T-snake, a curva projetada tem um único loop, que, em vista desse resultado, seria aberto, se $\Gamma_k$ fosse ideal. Assim, tornar $\Gamma_k$ ideal é sensivelmente vantajoso no que se refere a rotulação. Não fizemos isso neste trabalho pelo custo adicional que isto implica, conforme já foi esclarecido no final do capítulo 4. Para mostrar o alcance desse resultado, observe que, associado a resultados anteriores, ele elimina a possibilidade de $PC_k$ ter uma série de topologias simples, como as indicadas nas figuras 5.9.(a,b,c e d), mostradas a seguir, no caso de $\Gamma_k$ ser ideal.

A exclusão de cada uma dessas topologias se dá pelas seguintes razões: na configuração da figura 5.9.(a) teríamos dois loops abertos cujos nós seriam adjacentes na $LT(PC_k)$. Na da figura 5.9.(b) o loop $F1$ teria de ser fechado porque o nó relativo a ele é adjacente a folhas da loop-tree de $PC_k$. Mas, nesse caso, teríamos um loop fechado adjacente a três loops abertos que não é possível pelo lema 5.2.1. No caso da figura 5.9.(c) temos uma folha, portanto, um aberto cortado por outro loop. Finalmente a figura 5.9.(d) mostra que seqüências constituídas por um número par de loops delimitando regiões cujos interiores

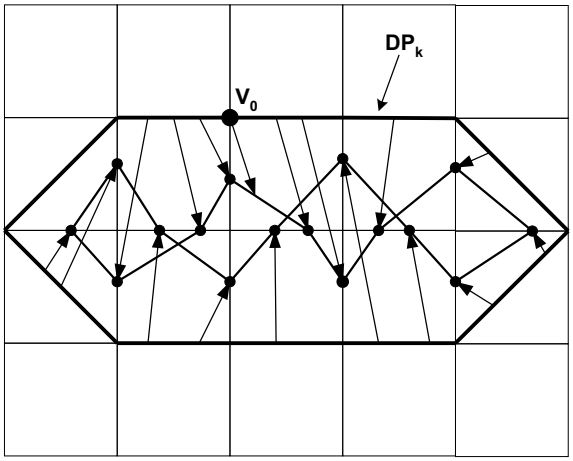


Figura 5.10: Exemplo de mapeamento adequado gerando uma curva cuja loop-tree tem uma folha que é um loop fechado — o da esquerda — e um aberto — o da direita.

são todos disjuntos, também não são possíveis. Ordenando os loops conforme eles vão sendo atingidos por um percurso da curva iniciado a partir de um ponto de uma folha os loops teriam de ser alternadamente abertos e fechados e assim não haveria como a outra folha ser também aberta. Chamamos a atenção para o fato de que, devido a possibilidade de haver pontos com raias coincidentes, todas essas topologias são possíveis se $\Gamma_k$ for apenas adequado.

Ao contrário do que acontece se $\Gamma_k$ for um mapeamento ideal, se ele for apenas adequado então $L - T(PC_k)$ pode ter folhas que correspondam tanto a loops abertos como fechados, como acontece com o exemplo da figura 5.10. A identificação do rótulo correto dessas folhas pode ser obtida, na maioria dos casos, aplicando-se o Lema 5.3.2, dado a seguir, o qual requer, entretanto que alguns conceitos sejam introduzidos.

Estamos considerando que os vértices de $PC_k$ são determinados e indexados conforme eles vão sendo atingidos quando se percorre $TC_k$ no sentido anti-horário a partir de $T_k(s_0)$. Em vista disso, os vértices de um loop aberto de $PC_k$ são atingidos em um percurso do loop no sentido anti-horário, feito a partir de seu vértice inicial, na ordem dada por seus índices. Já para os vértices de um loop fechado, a ordem dada pelos índices é aquela em que eles são atingidos quando se percorre o loop no sentido horário a partir do vértice inicial. Assim, orientando-se as arestas de um loop do vértice de menor índice para o de maior, a região delimitada por $L$ estará a direita delas, se o loop for aberto e a esquerda se o loop for fechado.

Sejam, então mais uma vez, $s'_{k,i}$, $i = 0, \ldots, N-1$ os vértices de $PC_k$. O **vértice à esquerda** de $s'_{k,i}$ — $v_e(s'_{k,i})$ - é o vértice da aresta da malha contendo $s'_{k,i}$ que fica à

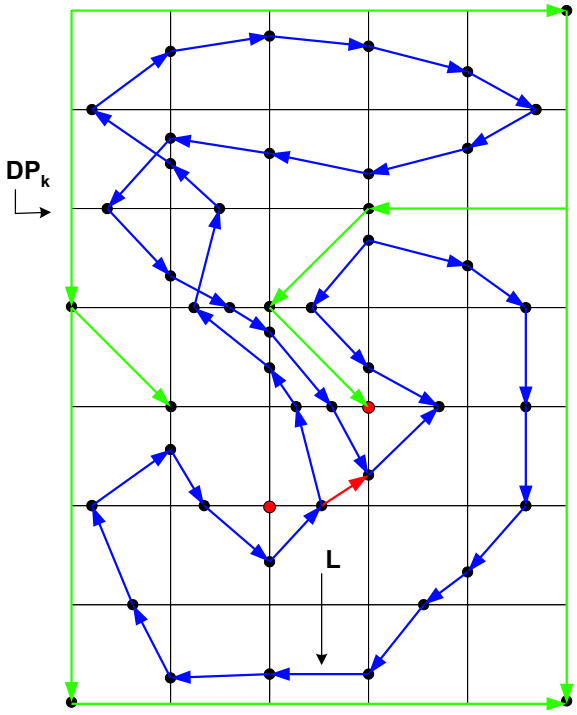


Figura 5.11: Exemplo de um loop aberto que intercepta todas as células adjacentes aos vértices a direita de suas duas extremidades.

sua esquerda, considerando-se a orientação de $PC_k$ dada pela ordem dos índices de seus vértices. O **vértice à direita** de $s'_{k,i}$ — $v_d(s'_{k,i})$ — é definido de forma análoga. Como os loops formados são todos $\mu$-curvas regulares, o que significa que a aresta da malha contendo $s'_{k,i}$ não é cruzada pelo loop em nenhum outro ponto, então, pelo exposto acima, podemos concluir que se $s'_{k,i}$ pertencer a um loop fechado, $v_e(s'_{k,i})$ deve ser interior ao loop e $v_d(s'_{k,i})$ externo a ele. No caso de $s'_{k,i}$ estar num loop aberto ocorre o contrário — $v_d(s'_{k,i})$ é interno e $v_d(s'_{k,i})$, exterior ao loop.

$v_e(s'_{k,i})$ pode ser determinado a partir das coordenadas *CAP* de $s'_{k,i}$. Ele é o vértice da célula $C(s'_{k,i})$ que é adjacente às suas arestas identificada por $E(s'_{k,i})$ e $(E(s'_{k,i}) + 1)$ *mod* 4. Considerando a célula do outro lado da aresta de $s'_{k,i}$,ele pode ser identificado pertencer às arestas dessa célula dadas por $(E(s'_{k,i}))^{-1}$ e $(E(s'_{k,i})^{-1} + 1)$ *mod* 4. Se $s'_{k,i}$ está num loop fechado então $v_e(s'_{k,i})$ será interior a ele, o que significa que as quatro células adjacentes a $v_e(s'_{k,i})$ devem ser cruzadas pelo loop. Como, obviamente, duas delas são as que contem $s'_{k,i}$, testar essa condição, pode ser feito simplesmente consultando os elementos da *EA* correspondentes às outras duas células adjacentes a $s'_{k,i}$. Entretanto, como o exemplo da figura 5.11 mostra, a condição acima é necessária mas, se for verificada apenas para um vértice do loop, pode não ser suficiente para que o loop seja fechado. Se ela for verdadeira

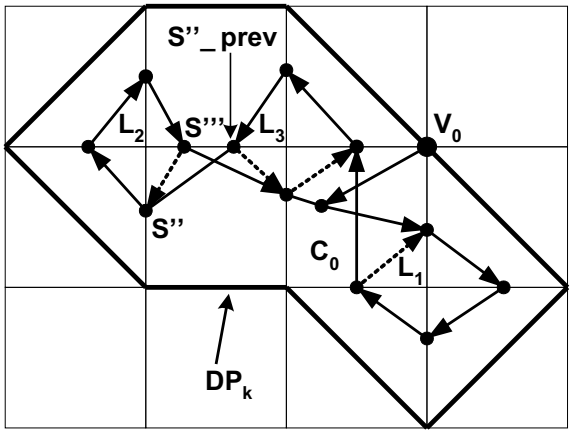


Figura 5.12: Enquanto $L_1$ e $L_3$ são loops sem antecessor, $L_2$ possui antecessor. Observe que no caso de $L_2$ *s''_prev* está definido.

para todos os vértices do loop, é claro que o loop tem de ser fechado,mas usar esse fato impediria que a rotulação de um loop fosse efetuada em tempo *O(1)*. Para conseguir isso precisamos de uma condição um pouco mais elaborada, que requer novas definições.

Vamos chamar de **Loop em Construção(LeC)**, num dado instante da determinação de $PC_k$ à poligonal definida pelos vértices de $PC_k$ que não pertencem a nenhum loop, gargalo ou $\mu$-whisker já determinado até esse instante. Isto é a parte já gerada, mas ainda não resolvida, de $PC_k$, até esse momento. Observe que com a descoberta de novos loops o vértice de menor índice do *LeC* pode mudar mas sem sair da célula inicial, isto é, daquela contendo $T_k(s_0)$.

Um **loop com antecessor** é todo aquele que cuja detecção não elimina completamente o *LeC* . Em outras palavras, é aquele que não contem o vértice inicial do traçado do *LeC* antes de sua determinação, pois se contiver o vértice inicial, conterá todo o *LeC*. O exemplo da figura 5.12 tem loops com e sem antecessor. No restante dessa seção trabalharemos apenas com loops que tenham antecessor e por concisão não faremos menção explícita a essa propriedade.

Para fazer os loops serem $\mu$-curvas regulares — inclusive o *LeC* pois é dele que sairão os loops que ainda serão encontrados — quando se detecta um novo loop, duas arestas do traçado anterior do *LeC* são eliminadas e substituídas por outras duas, uma fazendo parte do loop e outra do novo traçado do *LeC*, determinado pela remoção do loop. A figura 5.13 ilustra os três casos possíveis quanto a realização dessa troca de arestas e que correspondem a situações em que:

**i)** $PC_k$ volta a uma aresta da malha, chegando a ela pelo lado por onde saiu na vez

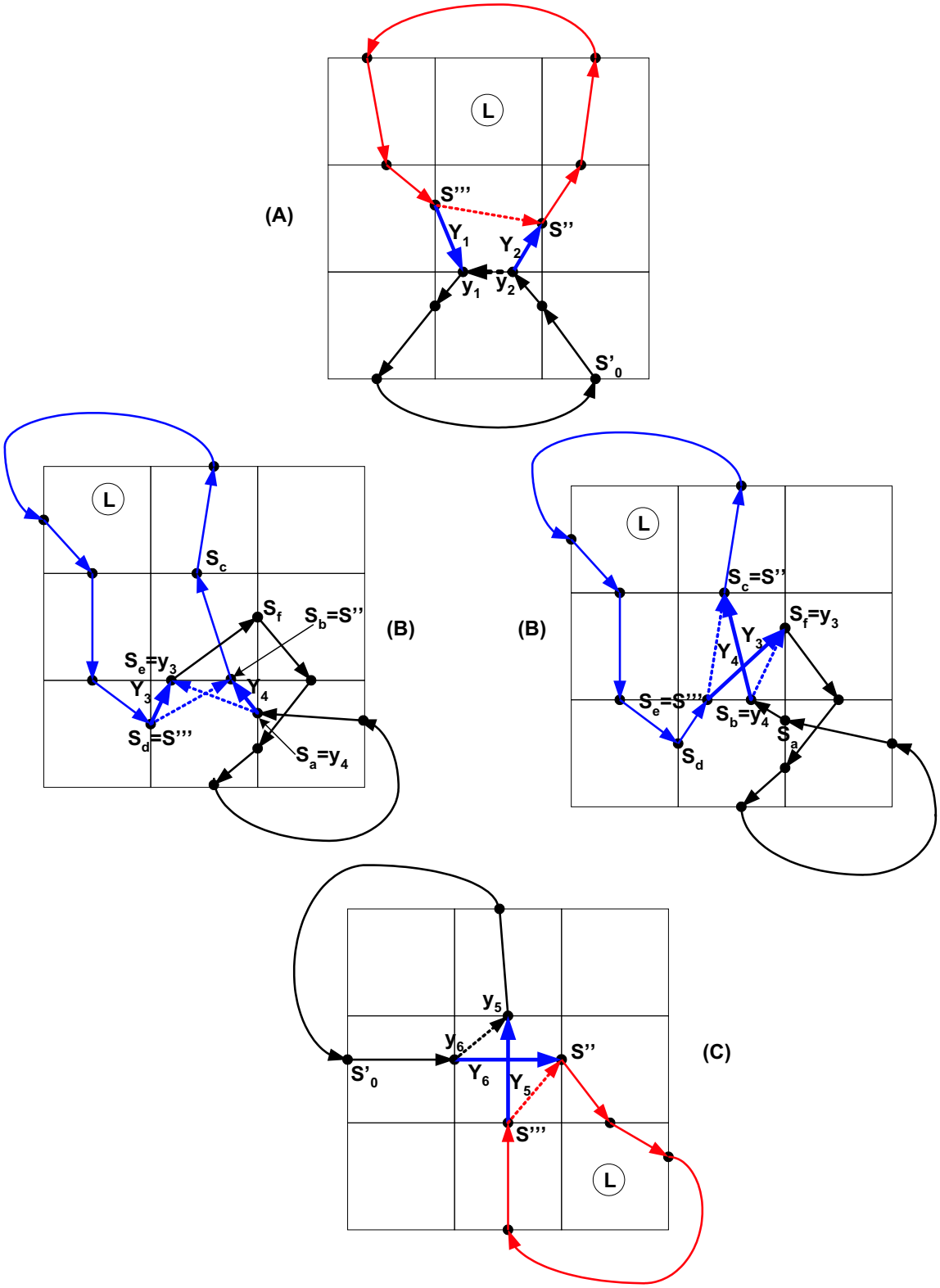


Figura 5.13: Conexões indicadas por $Y_i$ e encaixes indicados por $y_i$, $i = 1, \ldots, 6$ em cada um dos três casos possíveis em relação a troca de arestas. $s''$ e $s'''$ indicam o nó inicial e final do loop em cada caso. No caso da figura (B) temos duas possibilidades para a definição de conexões e encaixes. Adotou-se a alternativa descrita na figura de cima.

anterior.Ver figura 5.13.(A);

**ii)** $PC_k$ volta a uma aresta da malha, mas pelo mesmo lado em que chegou na vez anterior. Ver figura 5.13.(B). Nesse caso temos duas opções para tornar o loop uma $\mu$-curva regular. A primeira é ligarmos $s_d$ a $s_b$ para formar o loop e sa a se para que o *LeC* não sofra interrupção. Na outra ligamos se a sc para formar o loop e $s_b$ a $s_f$ para que o *LeC* se mantenha contínuo.Daqui por diante consideraremos sempre que a primeira alternativa é utilizada;

**iii)** o *LeC* cruza todas as arestas de uma célula o que caracteriza a formação de um *knot-loop*. Ver figura 5.13.(C).

As duas arestas que são eliminadas, no processamento efetuado quando se acha um loop, ligam ambas, um vértice do loop a um do novo traçado do *LeC*, e serão chamadas aqui, as conexões do loop. Os vértices das conexões que pertencem ao *LeC*, serão referidos, daqui por diante,como os encaixes do loop. Conexões dos loops mostrados nas figuras 5.13.(A,B e C) são identificados nessas figuras por $Y_i$, $i = 1, \ldots, 6$. $y_i$ indica o encaixe adjacente a $Y_i$. Temos agora todos os elementos necessários par enunciar o lema 5.3.2 e para mostrar como ele pode ajudar a rotular as folhas de $LT(PC_k)$.

**Lema 5.3.2** *Todo loop L com antecessor associado à uma folha de $LT(PC_k)$ tem pelo menos um encaixe situado numa aresta malha que é totalmente exterior a ele.*

Observe que se $L$ for do caso **i** ela terá ambos os encaixes numa mesma aresta da malha que pelo lema 5.3.2 tem de ser, então totalmente externa a ele. Se $L$ for do caso **iii** seus encaixes estarão em arestas diferentes mas ambas serão exteriores ao loop. No caso **ii**, entretanto um dos encaixes está necessariamente numa aresta onde também se localiza um vértice extremo do loop e portanto apenas a aresta contendo o outro encaixe pode ser exterior a ele. Em qualquer dos três casos, entretanto, a aresta contendo o antecessor do vértice inicial de $L$ no traçado do *LeC* antes da detecção desse loop, será sempre externa a ele. Essa aresta será referida daqui em diante por $a_e(L)$.

Como, em qualquer caso, as conexões de um loop estão na mesma célula que as suas extremidades, as quais, por sua vez, ficam em arestas diferentes, então, $a_e(L)$ é sempre adjacente ou ao vértice esquerdo ou ao vértice direito de pelo menos uma das extremidades. Pelo que vimos anteriormente, se ela for adjacente ao vértice direito de uma extremidade então $L$ terá de ser fechado. Se fosse aberto esse vértice teria de ser ao mesmo tempo interior e exterior ao $L$S. De forma análoga se $a_e(L)$ for adjacente ao vértice esquerdo de das extremidades de $L$ então $L$ deverá ser aberto.

Vamos descrever, agora, de como pode ser testado de forma rápida se o vértice a esquerda de uma extremidade de $L$ pertence a $a_e(L)$ ou não. Sejam, então, $s''$ e $s'''$, respectivamente, as extremidades inicial e final de $L$, e $s''_{prev}$, o antecessor de $s''$ no $LeC$, no momento que $L$ é determinado. Note que como $L$ tem antecessor $s''_{prev}$ está definido. Como $a_e(L)$ não pode ser a aresta contendo a extremidade, então o resultado do teste só pode ser verdadeiro se $a_e(L)$ for a outra aresta da célula — $CL$ — que contem as conexões de $L$, que é adjacente ao vértice em questão. Essa outra aresta é identificada como aresta de $CL$ por $(E(s'') + 1)^{-1}$ $mod$ 4, no caso de $s''$ e $(E(s''')^{-1})$ $mod$ 4 no caso de $s'''$. A diferença nas duas expressões deriva do fato que $PC_k$ está saindo de $CL$ em $s''$ e entrando em $s'''$. Como $a_e(L)$ é a aresta de $s''_{prev}$, sua identificação como aresta de $CL$ é dada por $E(s''_{prev})$ dado que $PC_k$ entra em $CL$ por esse vértice, resulta que o teste pode ser feito meramente comparando expressões envolvendo coordenadas de arestas de vértices de $PC_k$. Explicitamente, o resultado do teste será verdadeiro determinando que $L$ é um loop aberto se:

$$E(s''_{prev}) = (E(s'') + 1)^{-1} \; mod \; 4$$
$$\text{ou } E(s''_{prev}) = (E(s''')^{-1}) \; mod \; 4.$$

Caso contrário o resultado será falso indicando que L é um loop fechado.

Falta agora analisar o caso em que uma folha $L$ é um loop sem antecessor. Antes de fazer essa análise, entretanto, a seguinte consideração é necessária:

**a)** movendo-se em ambos os sentidos ao longo de $TC_k$, podemos gerar primeiramente $s'_{k,0}$ e $s'_{k,\ M-1}$ a partir daí executar o seguinte procedimento:

**b)** faça $m = 0$ e enquanto $s'k, m$ e $s'_{k,\ M-1-m}$ estiverem na mesma aresta, elimine $s'_{k,m}$ e $s'_{k,\ M-1-m}$, gere $s'_{k,\ m+1}$ e $s'_{k,\ M-2-m}$ e incremente $m$.

Ao final fazendo $PC_k$ ser a curva definida pelos vértices não eliminados — que é $\mu$-equivalente a inicial já que apenas podamos um $\mu$-whisker, $M = M - 2m$ e $s'_{k,\ i+m}$ ser o vértice inicial de $PC_k$ podemos garantir que ele e seu antecessor no novo traçado de $PC_k$, que agora podemos referenciar por $s'_{k,\ M-1}$, estarão em arestas distintas. Observe que a aplicação do procedimento acima não faz com que se percorra $PC_k$ mais de uma vez, por que, obviamente, não iremos voltar aos vértices que foram eliminados. Daqui por diante vamos assumir que $s'_{k,0}$ e $s'_{k,\ M-1}$ não estão na mesma aresta.

Assumindo isso, considere, inicialmente, que $s'_{k,\ M-1} \notin L$. Nesse caso, passando o vértice inicial para $s'_{k,\ M-1}$, o loop $L$ que não tinha antecessor passa a ter dois encaixes

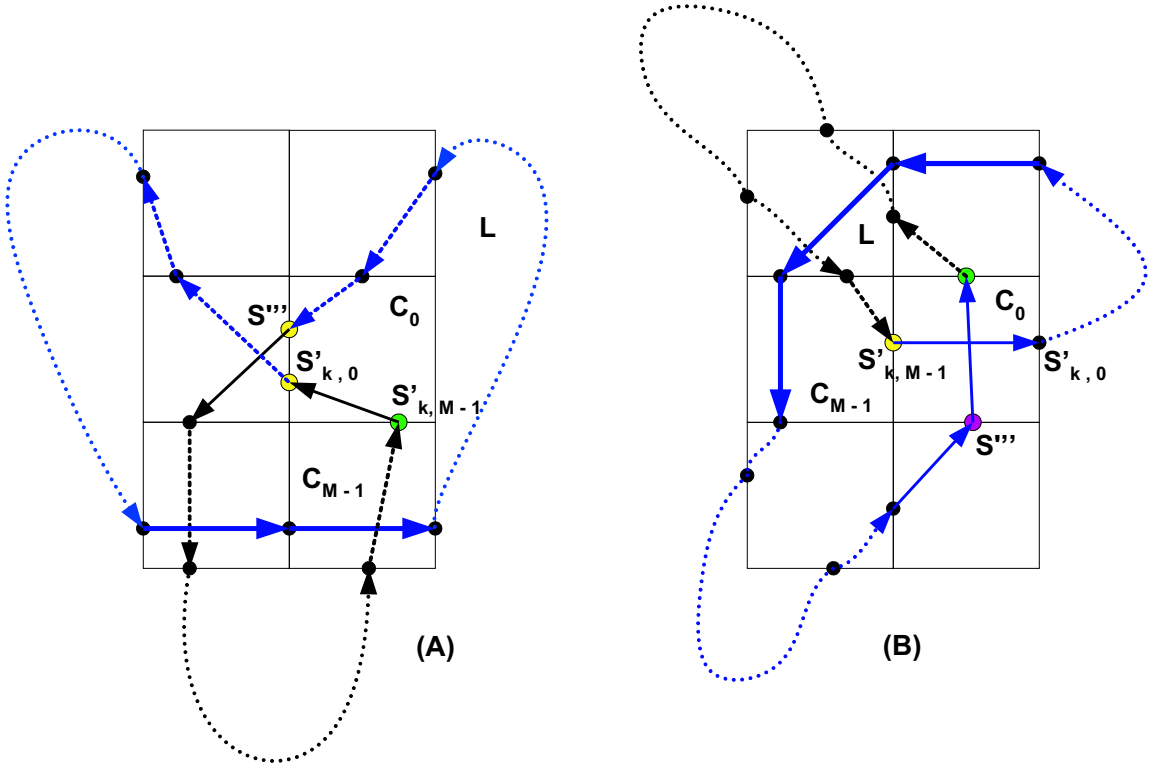


Figura 5.14: Configurações onde loops sem antecessor contém $s'_{k,\ M-1}$. O da figura (a) se refere ao caso em que $L$ volta a uma aresta outra vez pelo mesmo lado. O da figura (b) representa uma situação similar a da formação de um knot-loop.

sendo um deles $s'_{k,\ M-1}$. Então, se o vértice a esquerda de uma das extremidades de $L$ estiver na aresta — $e_{M-1}$ — contendo $s'_{k,\ M-1}$ o loop será fechado. Diferentemente se essa aresta contiver o vértice a direita da extremidade não se pode afirmar que o loop será aberto.

É que enquanto um loop mesmo sem antecessor que seja aberto jamais pode conter um encaixe, um fechado pode. Para que isso aconteça,entretanto, é necessário uma configuração que rodada de um múltiplo de 90° se torne $\mu$-equivalente a uma das duas representadas nas figuras 5.14.(a e b), pelo menos em seus trechos obrigatórios, representados nessas figuras com traço cheio.

A configuração representada na figura 5.14.(a) se refere ao caso em que $L$ é detectado, porque $PC_k$ cruza uma aresta duas vezes no mesmo sentido. A da figura 5.14.(b) é relativa a situação em que $L$ é determinado porque $PC_k$ atravessa a célula inicial — $C_0$ — cruzando as duas arestas dela que tem direção diferente da de $s'_{k,0}$. Em relação às trocas de arestas que são efetuadas na criação de $L$, tudo se passa como se $s'_{k,0}$ fosse precedido por um vértice de na aresta de $C_0$ oposta a dele. Ou seja, como se $L$ fosse um *knot-loop*.

Seja agora $C_{M-1}$ a célula separada de $C_0$ por $e_{M-1}$ e observe que os segmentos obriga-

tórios indicados nas figuras, ligam pontos — $s'_{k,j}$ e $s'_{k,\ j+1}$ — em arestas de $C_{M-1}$ que tem direção diferente de $e_{M-1}$. Um resultado referente a esses segmentos, que é importante para nossos propósitos, é o fato de que, considerando, naturalmente, o caso a que se referem as figuras em que eles aparecem, se $L$ contiver um deles então terá forçadamente de ser fechado. Este resultado pode ser demonstrado, simplesmente observando que se mudamos o vértice inicial para $s'_{k,\ M-1}$, então, antes que $L$ fosse formado iríamos identificar primeiro um loop que não é $\mu$-equivalente a ele, constituído pela parte inicial de $L$ até $s'_{k,j}$ seguida da poligonal $[s'_{k,j},\ s'_{k,\ M-1}, s'_{k,0}]$. Conforme já vimos, loops cuja detecção depende do ponto de onde começamos a percorrer $TC_k$ não podem ser abertos. O resultado acima nos permite adotar o seguinte procedimento para rotular um loop $L$, sem antecessor mas não contendo $s'_{k,\ M-1}$:

**1)** aplique o procedimento descrito para rotular loops com antecessor com $s''_{prev} = s'_{k,\ M-1}$, obtendo um rótulo inicial para $L$;

**2)** se o rótulo obtido foi aberto, verifique se $L$ cruza consecutivamente as arestas de $C_{M-1}$ transversais à $e_{M-1}$. Essa verificação pode ser feita se consultando o elemento da estrutura auxiliar relativo a $C_{M-1}$ e a lista de vértices de $PC_k$. Em caso afirmativo mude o rótulo de $L$ para fechado, caso contrário, mantenha o rótulo aberto.

Fica faltando então apenas considerar o caso de uma folha $L$ que é um loop sem antecessor e que além disso contém $s'_{k,\ M-1}$. Como $L$ é uma folha de $PC_k$, que, nesse caso, conterá $s'_{k,\ M-1}$ e $s'_{k,0}$ resulta que ele tem de ser a própria curva $PC_k$ a menos, talvez, de um ou mais $\mu$-whiskers.

Por simplicidade vamos assumir que esses $\mu$-whiskers não existem, esclarecendo, entretanto que é possível tratar o caso em que eles estão presentes sem sacrificar a eficiência do processo.

Assumindo isso, $s'_0$ e $s'_{M-1}$, são, os vértices inicial e final de $L$. Seja então $s_{ini}$ o snaxel inicial de $S_k$ e $v_{ini} = D_k(s_{ini})$. Por concisão, quando for conveniente vamos identificar $v_{ini}$ com o vértice da malha do qual ele está junto e então, fará sentido dizer, uma aresta adjacente a $v_{ini}$. Quando não for conveniente, consideraremos $v_{ini}$ distinto desse vértice, e diremos por exemplo, que $e_{ini}$ é a aresta de $v_{ini}$. É claro que $\Gamma_k(v_{ini}) \in e'_0 = [s'_0,\ s' - M - 1]$.

Suponha, inicialmente que o próprio mapeamento $\Gamma_k$ é adequado. Nesse caso dentre todas as raias de $\Gamma_k$ que chegam a pontos de um loop fechado, só as que chegam a vértices desse loop — especificamente, aquelas que começam em pontos onde $\Gamma_k$ não é ideal — podem ser disjuntas do interior do loop. Isso quer dizer que toda a raia de $\Gamma_k$, chegando

a um ponto $p$ que não é vértice de $L$ tem de cruzar uma de suas arestas diferente daquela contendo $p$. Se necessário perturbando a localização de $T_k(s_{ini})$ podemos assumir que ele não se encontra numa aresta da malha. Pela própria maneira como $\Gamma_k$ é construído, isso será suficiente para garantir que $\Gamma_k(v_{ini})$ não será um vértice de $PC_k$.

Feito isso, podemos verificar se $L$ é um loop fechado ou aberto, simplesmente, verificando se a raia $r_{ini} = [v_{ini},\ \Gamma_k(v_{ini})]$ corta ou não uma de suas arestas diferente de $e'_0$. Essa intersecção, se existir, será única. Daqui por diante, $x_{ini}$ se referirá ao ponto onde ela se dá.

É interessante notar que não temos a posição exata de $\Gamma_k(v_{ini})$. Sabemos, apenas que ele está em $e'_0$. Mas como conhecemos essa aresta e $v_{ini}$, sabemos porque lado de $e'_0$, $r_{ini}$ chega a $\Gamma_k(v_{ini})$. Isso e, naturalmente, o conhecimento das arestas da malha contidas em $C_0$ ou adjacentes a ela, que são cortadas por $L$, nos permite determinar se a interseção procurada, de fato existe.

Se $v_{ini} \in C_0$ essa intersecção não existirá e em conseqüência $L$ será aberto, a menos que $C_0$ seja dupla e $v_{ini}$ o vértice de $C_0$ que não é adjacente às arestas de $s'_0$ e $s'_{M-1}$. Nesse último caso $L$ será fechado. Se $v_{ini}$ está numa célula — $C_{ini}$ — vizinha a $C_0$, a análise já não é tão simples. É preciso classificar os casos possíveis conforme às condições $C_1$, $C_2$, $C_3$ e $C_4$, dadas a seguir, que eles satisfazem.

**C 1)** $C_{ini}$ não contem nem $s'_0$ e nem $s'_{M-1}$. Nesse caso $x_{ini}$ existe se e só se $L$ tem um vértice em $e_{ini}$. Ver figura 5.15.(a);

**C 2)** $C_{ini}$ contem $s'_0$ ou $s'_{M-1}$ e as arestas da malha contendo esses dois pontos e $e_{ini}$ tem todas três um vértice em comum. Nesse caso também vale a afirmação do caso anterior. O exemplo da figura 5.15.(b) ilustra esse caso;

**C 3)** $C_{ini}$ não é dupla, contem uma extremidade — s" — de $e'_0$, mas $e_{ini}$ não satisfaz a outra condição requerida em **C2**. Nesse caso a afirmação do caso **C1** será válida se $e_{ini}$ e $e'_0$ estiverem do mesmo lado da reta suporte — $\sigma_i ni$ — de $[v_{ini},\ s'']$. Se essas arestas estiverem em lados diferentes, valerá a firmação contrária, ou seja, que $x_i ni$ existirá se e só se $L$ não intersectar $e_{ini}$. As figuraa 5.15.(a) e 5.15.(d) ajudam a distinguir esses dois casos;

**C 4)** $C_{ini}$ e $e_{ini}$ atendem a **C3**, exceto por $C_{ini}$ ser dupla. Nesse caso se $e_{ini}$ e $e'_0$ estiverem do mesmo lado de $\sigma_{ini}$ então $x_{ini}$ sempre existirá. Se elas estiverem em lados diferentes de $\sigma_{ini}$, não haverá essa intersecção. Os dois casos estão representados nas

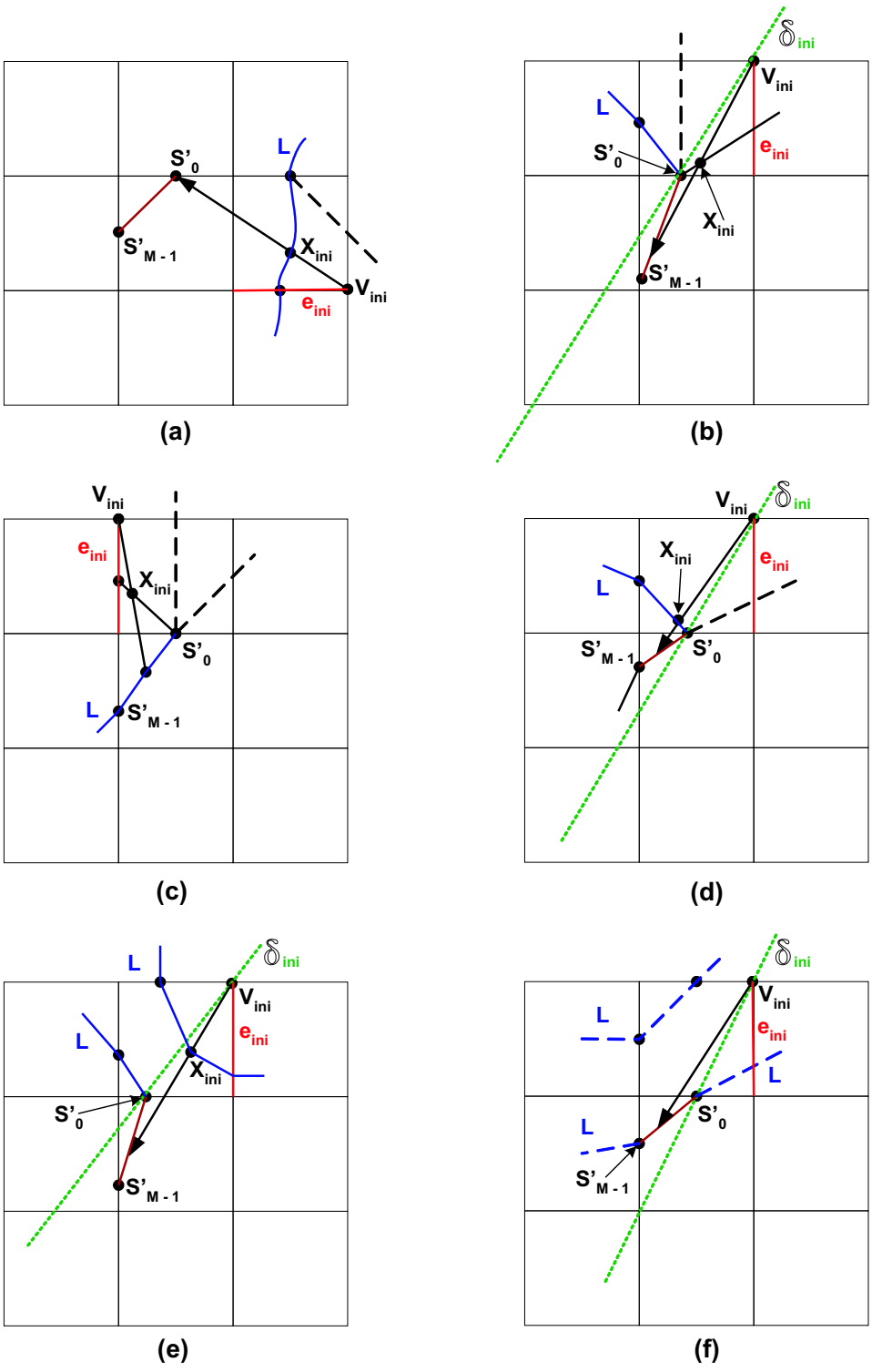


Figura 5.15: Cada um dos casos referidos nas condições $C1$—$C4$. As alternativas para o traçado de $L$ representadas em linha cheia intersectam a raia que sai de $v_{ini}$. As representadas em linha tracejada são disjuntas dessa raia.

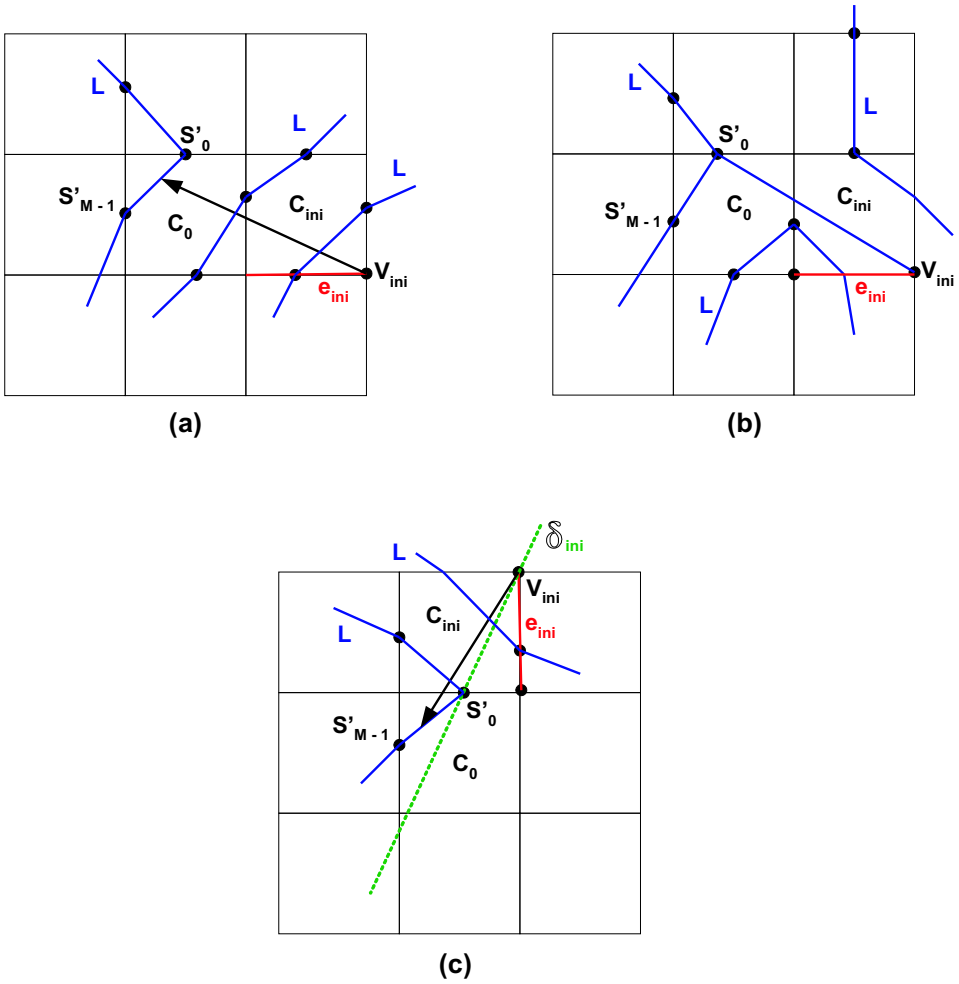


Figura 5.16: Casos que não são possíveis quando $e_{ini}$ é dupla.

figuras 5.15.(e) e 5.15.(f). Devemos observar que se $C_{ini}$ é dupla uma série de casos como os indicados na 5.16.(a, b e c) não são possíveis.

Portanto se $\Gamma_k$ é adequado teríamos assim, um mecanismo para determinar se $x_{ini}$ existe ou não e em função disso saber qual é o rótulo de $L$. O problema é que com o procedimento dado no capítulo anterior podemos garantir apenas que $\Gamma_k$ é $\mu$-equivalente a um mapeamento adequado — $\Gamma_k^*$ — mas não que ele próprio seja adequado. Em vista disso $PC_k$ pode ter um loop abertos tal que a raia de $\Gamma_k$ chegando a um de seus pontos, o corta em outro ponto, como mostra a figura 5.17.

Mesmo assim se vini estiver na própria célula $C_0$ podemos aplicar o procedimento descrito anteriormente para esse caso. Se $v_{ini} \notin C_0$ precisamos de um pouco mais de informação, para, junto com o conhecimento das arestas contendo $s'_0$, $s'_{M-1}$ e $v_{ini}$ e das cortadas por $L$ em $C_0$ e $C_{ini}$ decidir qual deve ser o rótulo do loop. De fato, nesse caso,empregar uma única raia pode não ser suficiente.

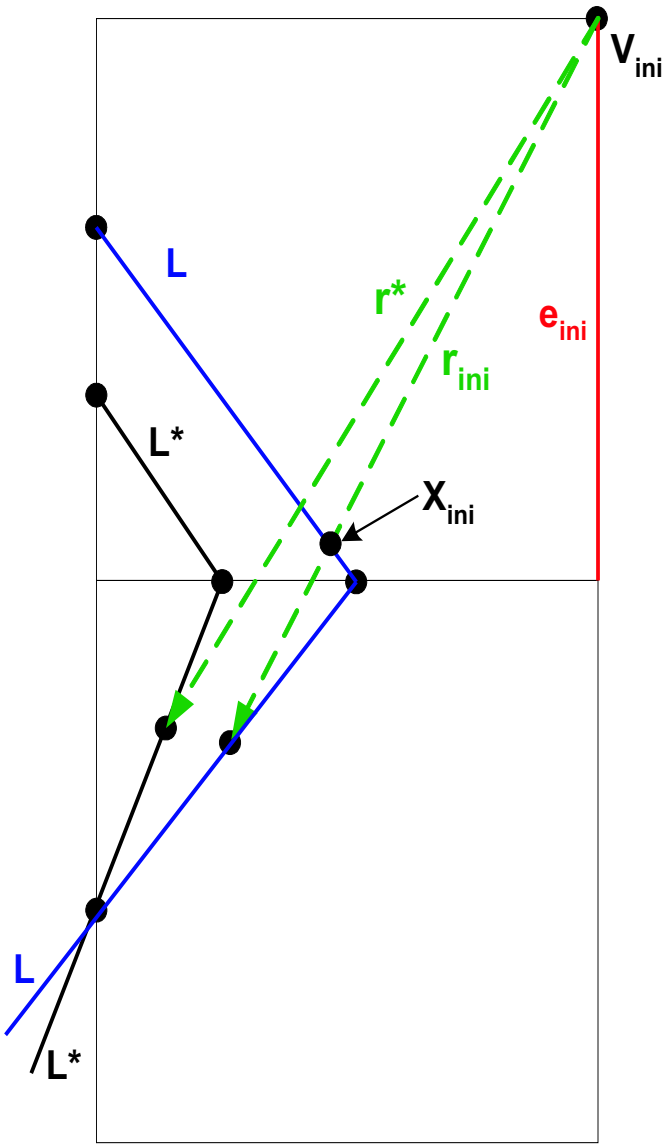


Figura 5.17: Dado que $\Gamma_k$ é apenas $\mu$-equivalente a uma mapeamento adequado $\Gamma_k^*$ então um loop aberto $L$ de $PC_k$ pode ser cortado por uma de suas raias. Isso, entretanto, não pode acontecer com seu correspondente $L^*$ na curva gerada por $\Gamma_k^*$.

O raciocínio a ser empregado agora é o seguinte: seja $L^*$, a curva gerada por $\Gamma_k^*$. Como $\Gamma_k^*$ é adequado, $L^*$ será fechado ou aberto conforme exista ou não um ponto — $x^*$ — de intersecção da raia de $\Gamma_k^*$ partindo de $v_{ini}$, com uma aresta de $L^*$ diferente de $e_0^*$ — aquela que corresponde a $e_0'$. Como o rótulo de $L$ deve ser o mesmo de $L^*$, se conseguimos decidir se $x^*$ existe ou não, ele estará perfeitamente determinado. Devemos nos concentrar portanto em decidir sobre a existência de $x^*$ ou não e não mais sobre a de $x_{ini}$.

Entretanto, se $C_{ini}$ e $e_{ini}$ satisfazem às condições **C1** e **C2** dadas acima, não precisamos de nenhuma informação adicional para decidir sobre a existência dessa intersecção, porque, como a definição acerca da existência de $x_{ini}$ independe da orientação de $e_0'$, os resultados obtidos para $L$ serão os mesmos para qualquer curva $\mu$-equivalente a ele e, em particular, para $L^*$.

No caso das condições **C3** e **C4** entretanto a definição sobre a ocorrência de $x_{ini}$ depende da orientação de $e_0'$. Sejam, então, $s^*$ o vértice de $L^*$ na aresta entre $C_0$ e $C_{ini}$, e $\sigma^*$, a reta suporte de $[v_{ini},\ s^*]$. Chame de $H_0$ e $H_0^*$, aos semi-planos determinados, respectivamente, por $\sigma$ e $\sigma^*$, que contem $e_{ini}$. $H_1$ e $H_1^*$ serão os outros semiplanos determinados por essas retas. Pode acontecer então que $e_0'$ esteja em $H_i$ e $e_0^*$ em $H_{1-i}$, $i = 0$ ou 1, como mostrado na figura 5.18. Isso fará com que se aplicássemos o procedimento acima para o casos que se enquadram nessas condições, o resultado para $L$ seria um e para $L^*$, outro. Entretanto, se usando o fato que $\Gamma_k^*$ é adequado conseguirmos impedir que $e_0^*$ esteja de um dado lado de $\sigma^*$, então a existência ou não de $x^*$ estará perfeitamente definida também nesses casos.

Para isso considere um ponto — $s_1$ — de $[\Gamma_k^*(v_{ini}),\ s^*] \subseteq e_0^*$. Podemos fazer esse ponto suficientemente próximo de $\Gamma_k^*(v_{ini})$ de modo que $w_1 = (\Gamma_k^*)^{-1}(s_1)$ pertença a uma aresta adjacente a $v_{ini}$. Nesse caso, a poligonal obtida concatenando $[\Gamma_k^*(v_{ini}),\ s_1]$ e $[s_1,\ w_1]$ secciona inteiramente o triângulo formado por $v_{ini}$, $\Gamma_k^*(v_{ini})$ e $s_1$, o que determina que a raia $r^* = [v_{ini},\ \Gamma_k^*(v_{ini})]$ só pode chegar por um dos lados a $e_0^*$ sem cruzar a raia $[s_1,\ w_1]$ como deve acontecer dado que $\Gamma_k^*$ é adequado. Podemos assim decidir sobre a existência ou não de $x^*$, porque suprimindo um lado de $x_0^*$ eliminamos uma das alternativas que podem ocorrer no caso das condições **C3** e **C4**.

Observe na figura 5.19 que se $w_1$ e $C_0$ estiverem do mesmo semiplano determinado pela reta suporte de $e_{ini}$, então o caso impedido será aquele em que $e_0^*$ e $e_{ini}$ estão em lados diferentes de $\sigma^*$. Resulta então que $L^*$ será fechado apenas se cortar $e_{ini}$ e o mesmo podemos dizer de $L$ que é $\mu$-equivalente a ele. Se $w_1$ e $C_0$ estiverem em lados opostos então o caso que não será possível será aquele em que $e_0^*$ e $e_{ini}$ estão do mesmo lado de

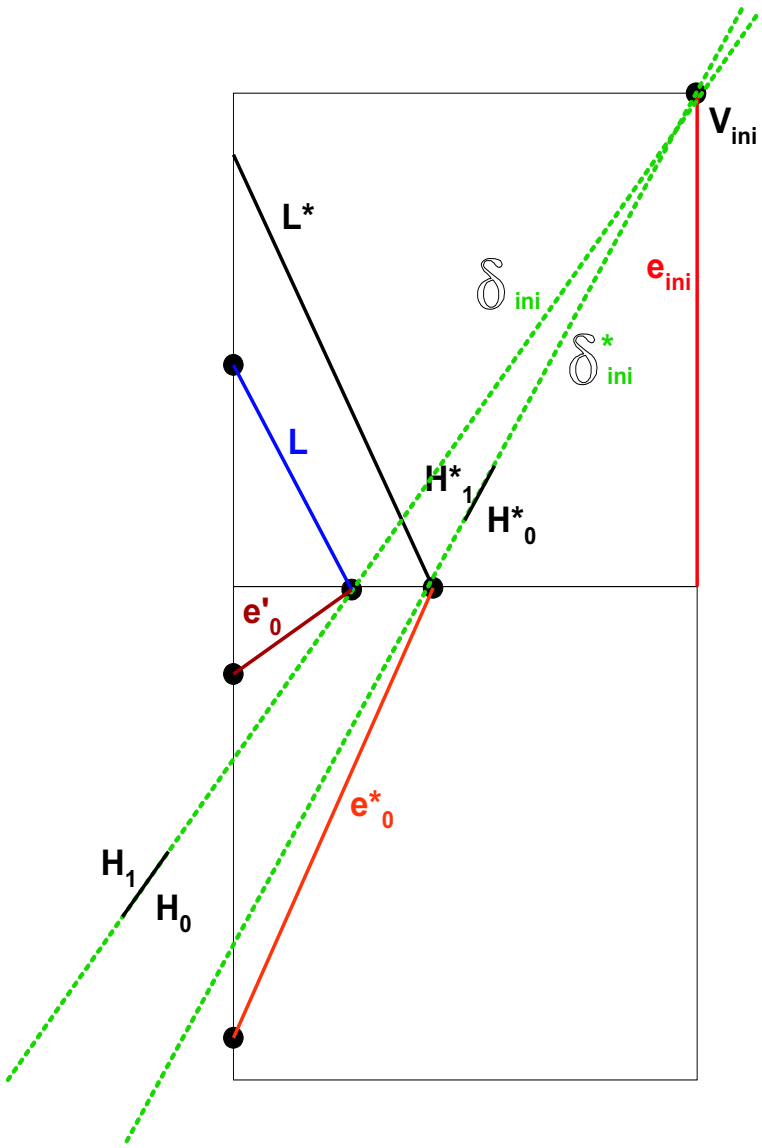


Figura 5.18: Situação em que $e'_0$ está em $H_1$ e $e^*_0$ em $H^*_0$.

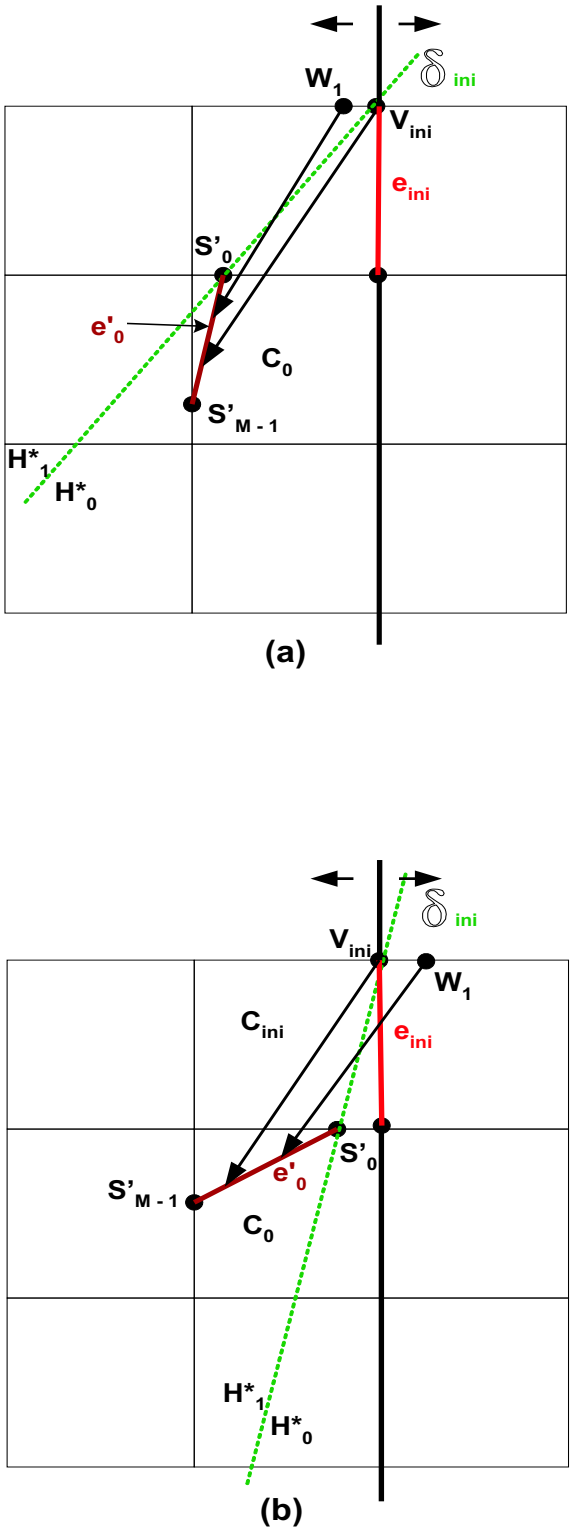


Figura 5.19: Conforme $w_1$ esteja do mesmo lado que $C_0$ ou não, com respeito a reta suporte de $e_{ini}$, $e'_0$ tem de estar respectivamente em $H_0^*$ (figura (a)) ou em $H_1^*$ (figura (b)).

$\sigma^*$. Em função disso $L^*$ e, portanto, $L$ será fechado se não cortar $e_{ini}$ . É claro que não precisamos saber a posição exata de $w_1$. Basta saber de que lado ele esta'em relação a reta suporte de $e_{ini}$. Como esse lado é exatamente aquele de $C(s_{ini})$, acrescentando essa informação às listadas anteriormente temos uma maneira correta de obter o rótulo de $L$ em tempo $O(1)$. De forma heurística, a menos que os segmentos que se busca encontrar numa imagem tenham muitas ramificações , se $PC_k$ formar um único loop então ele pode ser rotulado como aberto se seu número de vértices for maior que um mínimo pré-estabelecido.

# Capítulo 6

# Revisitando uma Célula

## 6.1 Introdução

O processo constrói a curva-$PC_k$, vértice por vértice seguindo as regras especificadas na secção 4.2 e sucessivamente vai atualizando a "estrutura auxiliar" com uma referência para o último vértice determinado, isso até que uma célula é revisitada. Nesse ponto as possibilidades de que a célula tenha se tornado uma célula dupla ou um loop tenha sido formado, determinando, nesse caso, a realização de um split, tem de ser consideradas. O trabalho de explicar como são tratadas todas as possíveis situações que podem ocorrer em uma célula revisitada são tratadas, será facilitado pela introdução de alguns conceitos, dados abaixo, e por uma descrição detalhada de como a "estrutura auxiliar" pode ser organizada.

Embora o conceito de Loop em Construção já tenha sido introduzido no capítulo anterior vamos reapresentá-lo aqui dentro de um outro enfoque. Durante o processo de atualização de uma Loop-Snake, o **Loop_em_Construção** — *LeC* — é a linha poligonal definida pelos vértices de $PC_k$, que não pertencem a nenhum loop já detectado, na ordem em que eles foram gerados. O *LeC* pode ser considerado como a raiz da Loop-Tree da parte de $PC_k$ já construída. Ele é, obviamente, uma curva regular, exceto no momento em que com o acréscimo de um vértice um loop é formado. Como esse loop é, então, imediatamente extraído dele, ele volta logo a ser uma curva $\mu$-regular. Se $x$ é um vértice de $PC_k$ então o $LeC(x)$ se referirá ao *LeC* no momento em que $x$ foi determinado. A figura 6.1 ilustra esses conceitos.

Um loop **válido** é um que está totalmente contido no *LeC* enquanto um não válido compartilha parte de seu contorno com um loop já encontrado anteriormente e por esta razão não pode ser adicionado a árvore de loops — ver figura 6.2. Loops não válidos eventualmente são formados porque as referências na Estrutura Auxiliar aos vértices de

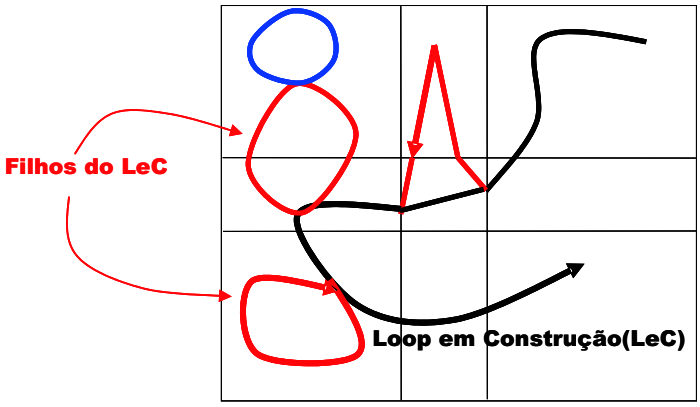


Figura 6.1: Tratando Loops: Elementos Principais.

um loop recém encontrado não são apagadas. Isto, entretanto, não é um problema, porque o algoritmo tem meios de distinguir entre loops válidos e não-válidos. Basicamente, o procedimento, executado a cada célula que é revisitada, para efetuar essa distinção é o seguinte:

**(i)** na criação de um loop os índices de seus vértices inicial e final, ou seja, aqueles de menor e de maior índice entre todos os vértices do loop são armazenados;

**(ii)** quando $PC_k$ revisita uma célula $C$ pode-se, então, verificar se o segmento $s$, segundo o qual $C$ foi intersectada por $PC_k$ na visita anterior, está dentro do $LeC$ ou de um loop já formado. Isso é feito comparando os índices dos vértices de $s$ com os dos vértices extremos dos filhos do $LeC$ na loop-tree $\mu$-regular. Se o índice de um vértice de $S$ estiver entre os dois vértices inicial e final de um filho do $LeC$, o loop que acabou de ser gerado é declarado, inválido. Caso contrário, ele é considerado válido e processado, determinado-se seu rótulo e se ele é separado do $LeC$ por uma auto-intersecção de $PC_k$ além de se guardar os dados a ele referentes como descrito em **(i)**. Fazer com que a validade ou não dos $m$ loops gerados num estágio de evolução da snake possa ser verificada em $O(m)$ depende, no entanto, de como é armazenada a informação relativa aos loops já gerados, bem como da estrutura auxiliar usada, a qual será tratada na próxima seção. Um maior detalhamento sobre como devem ser identificados loops válidos e inválidos será apresentado, futuramente na descrição da função *Check_whether_the_LeC_contains*.

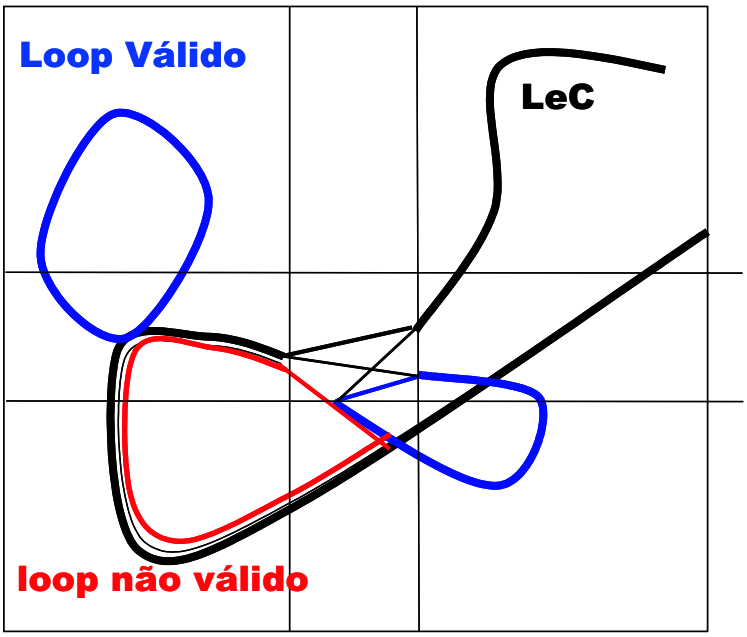


Figura 6.2: Tratando Loops: Loops não válidos devem ser identificados para que não sejam introduzidos em $LT(PC_k)$.

### 6.1.1 Estrutura de Dados Utilizada

Diferentes modelos são possíveis para a Estrutura Auxiliar. O mais simples é uma matriz de elementos de dois bits, cada um associado a uma célula da malha. O conteúdo destes elementos pode ser -1, 0 ou 1. O valor -1 indica que a célula não foi atingida ainda, por nenhuma *PC* gerada desde o início do processo de evolução da snake. Os outros dois valores indicam a paridade do estágio no qual a célula foi atingida pela última vez. O fato de que existem células não visitadas no estágio corrente, cujos elementos têm o mesmo código que os daquelas que foram visitadas, pode ser contornado. Estas células foram alcançadas pela última vez em um estágio anterior com a mesma paridade do corrente e como o processo de evolução das snakes é de contração, isto significa que elas são exteriores à snake do estágio que precede o atual.

Agora, considere que para cada estágio $k$ da evolução da snake, $PC_k$ é totalmente interior a $\mu$-dilatação de $S_{k-1}$. Nessas condições, é bastante claro que $PC_k$ não pode alcançar essas células. Como $S_{k-1}$ e $S_k$ podem ser coincidentes, o meio de garantir que $PC_k$ é interior a $\mu - D(S_{k-1})$ é torná-la interior a $\mu - D(S_k)$. Essa propriedade pode ser satisfeita mesmo que $S_{k+1}$ seja interior a $S_k$, pois isso não estabelece restrições sobre os loops fechados de $PC_k$. Para atendê-la se torna necessário limitar por $d\sqrt{2}/2$, o deslocamento máximo dos snaxels em células duplamente cortadas por $s_k$ e com uma diagonal

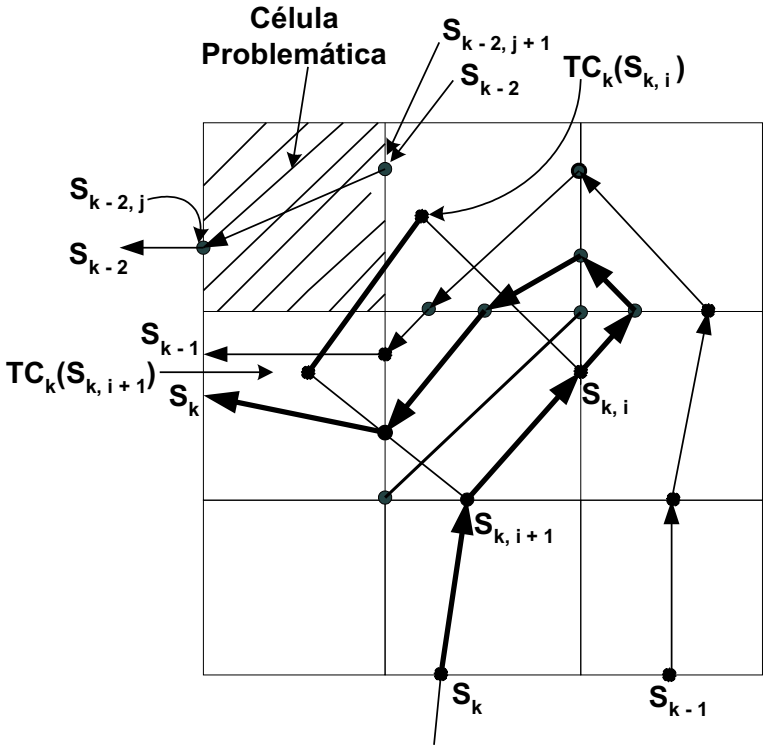


Figura 6.3: $TC_k$ e $PC_k$ retornam para a célula problemática que foi visitada pela última vez no estágio $k-2$.

interior a ela. A figura 6.3 ilustra o problema que pode ocorrer nessa situação. Sejam $s_{k,i}$ e $s_{k,\,i+1}$ os snaxels de $S_k$ que estão em $C$. Somente se uma configuração similar a da figura ocorre, o lugar de $u_{k,j} = [T_k(s_{k,i}), T_k(s_{k,i+1})]$, pode intersectar um célula —- $C'$ — que foi visitada pela última vez por uma curva projetada em um estágio $l$ anterior a $k-1$. Nesse caso $C'$ será simétrica a $C$ em relação a um de seus vértices $v$. Se $l$ tem a mesma paridade de $k$ e $PC_k$ cruza $C'$, uma falsa revisita será indicada consultando-se o elemento de $C'$ na $E-A$. Entretanto, se o deslocamento de $s_{k,i}$ e $s_{k,i+1}$ são limitados por $d\sqrt{2}/2$, $C'$ será separada de $r_{k,i}$ pela reta passando por $v$ que é paralela a diagonal de $C$ interior a $S_k$. Conseqüêntemente nenhuma falsa revisita será determinada neste caso.

Uma outra clara limitação ao se usar uma estrutura tão simples é o fato de que, quando uma célula repetida é detectada, o *LeC* tem de ser percorrido andando-se para traz até que ele volte a essa célula. Somente então é possível decidir se um novo loop realmente foi achado ou se a célula simplesmente se tornou uma célula dupla. A fim de se obter a assim chamada, propriedade de se percorrer $PC$ uma só vez é necessário ter, para cada célula revisitada, uma ligação direta entre os vértices de $PC$ gerados em duas visitas sucessivas a ela. Esta ligação pode ser provida através da manutenção, para cada célula $C$, de um código que identifica *Last_Vertex*[$C$], o

último vértice de uma curva projetada já gerado — seja no estágio corrente ou num anterior, que pertence a $C$. Este código é composto de duas partes conforme explicado abaixo:

**A)** para evitar que a snake fique presa em um mínimo local de energia, se uma célula $C$ é cruzada por um número pré-definido, digamos $K$, de snakes de iterações consecutivas, os snaxels cuja coordenada da célula é $C$ ou são congelados ou deslocados por forças artificiais para fora de $C$. A primeira parte do código de identificação do Last_Vertex[C] é dado por $(k+1) mod (K+1)$, onde $k$ é o estágio no qual esse vértice foi gerado. O objetivo de se aumentar o tamanho dos blocos dentro dos quais o estágio é identificado por um código diferente, de 2, como ele era no caso da estrutura minimalista apresentada antes para $K+1$ é unicamente evitar que, por se compactar excessivamente a informação contida em $Last\_Vertex[.]$, restrições adicionais precisem ser impostas ao deslocamento máximo de snaxels, além de que eles devam ser $\mu$-limitados[1]. Valores típicos de $K$ são números entre 3 e 7. Essa primeira parte do código será referida como *L*ast_Vertex[C].stage;

**B)** a segunda parte contém o índice do *L*ast_Vertex[C] na lista de vértices da $PC_k$ corrente. Essa lista, daqui por diante referida simplesmente como *L*ista de Vértices ou *LV*, será assumida como sendo uma lista duplamente encadeada embora também pudesse ser simplesmente encadeada. Neste último caso, entretanto, seria necessário ter uma lista auxiliar para indicar as descontinuidades determinadas na seqüência dos vértices do *LeC* pelos loops já formados.

*L*ast_Vertex[.] será uma matriz ou vetor conforme seja feita a identificação de uma célula e constitui toda a chamada Estrutura Auxiliar que é empregada na implementação feita neste trabalho. Um outro item será incluído nos registros da *LV* como o resultado do seguinte raciocínio: células duplas contém dois segmentos do *LeC* e o processo deve ter meios de acessar a ambos. A alternativa mais simples para se lidar com essa questão seria duplicar a Estrutura Auxiliar de modo que dois vértices, um em cada segmento, pudessem ser referenciados por cada célula dupla. Entretanto como o número de vértices de $PC_k$ gerados numa iteração da snake, geralmente é muito menor do que o das células da malha, é preferível adicionar um item "(.old)" a estrutura dos registros da *LV*. *L*ast_snaxel[C].old informará o último snaxel do primeiro segmento do *LeC* em *C*,

[1]Essa limitação, entretanto, é necessária se pretendemos empregar outras propriedades como o próprio lema 5.1.1

se *C* for dupla e o item deve ter valor nulo no registro de outros vértices. Com este novo item o processo pode-se valer do fato de que o *L*ast_snaxel[C].old não é nulo, para saber que uma célula *C* sendo revisitada já uma célula dupla. Assim, uma estrutura que indique explicitamente quais células são duplas se torna desnecessária.
Ao incorporar o item .old, a estrutura de registros da *LV* finalmente tem a seguinte forma:

**typedef {**

short int edg; //coordenada da aresta do snaxel
short int pix; //coordenada do pixel do snaxel
LV *nxt; //↑ para o próximo snaxel em PC
LV *prv; //↑ para o snaxel anterior em PC
LV *old; //↑ para o último snaxel anterior de uma célula dupla
**}**
**LV [];**

### 6.1.2 Detectando um Loop e Dividindo a curva-PC

Os pontos que estão na mesma aresta são considerados indistingüíveis e, em vista disso, se *LeC* retornar para uma aresta *e*, considera-se que um novo loop foi formado. Também existem loops que não revisitam nenhuma aresta da malha, os quais são determinados pelas autointersecções(knots)de $PC_k$. Esses loops são identificados por uma configuração onde uma célula tem dois snaxels-$PC_k$ sucessivos em suas arestas verticais e outros dois sucessivos em suas arestas horizontais. O fato de que a caracterização de um tipo de loop está relacionado a uma aresta e o outro tipo a uma célula determina que eles podem ser tratados de maneiras diferentes. Vamos iniciar com o caso de uma aresta repetida:
Uma configuração típica neste caso é descrita na figura 6.5. $s_{j-m}, \ldots, s_j, s_i, \ldots, s_{i+m}$ com $J < i$ são vértices do *LeC* que formam o que se chama de **gargalo**. Essa configuração é caracterizada por:

A) $s_{i+n}$ e $s_{j-n}$ estão sobre a mesma aresta, $n = 0, \ldots, m$.

B) Se o *LeC* entra em uma célula $C_n$ em $s_{i+n}$ então, no momento em que esse vértice foi gerado, $Last\_vertex[C_n] = s_{j-n}$.

Quando $s_i$ é gerado, o loop $L_0$, delimitado por $s_{j+1}$ e $s_{i-1}$, é identificado e qualquer vértice do gargalo deve ser descartado por estar na mesma aresta de um outro vértice do

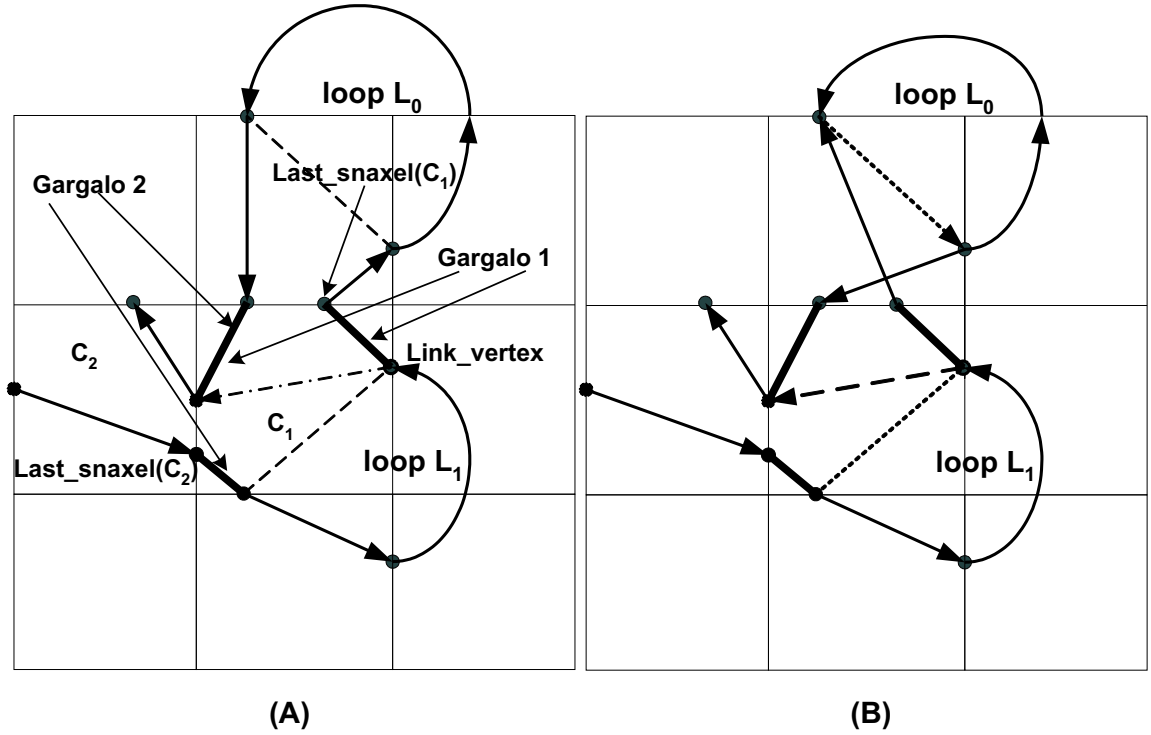


Figura 6.4: Exemplos de um gargalo cujo ramo inicial é interrompido por um loop $L_1$. Na figura (A), $L_1$ e $L_0$ têm o mesmo rótulo. Na figura (B) esse rótulo é diferente.

*LeC*. Entretanto, o tratamento de um gargalo não se resume a isso porque para rotular corretamente o loop, ainda a ser formado, contendo $s_{i+m+1}$ é necessário verificar se $PC_k$ tem uma $\mu$ -auto - intersecção em $s_i, \ldots, s_{i+m}$. Se essa intersecção não existe o rótulo deste loop deve ser o mesmo de $L_0$ e ambos devem ser associados a um mesmo nó da Loop-tree. Eles são, de fato, parte de um loop maior que foi subdividido em componentes que são $\mu$-curvas regulares. Por outro lado, se a $\mu$ -auto - intersecção existe, $L_0$ se torna um filho do *LeC* e suas extremidades devem ser colocadas na pilha de seu rótulo. A existência de uma $\mu$ -auto - intersecção deve ser feita através do uso da função $X$, introduzidas na subseção 4.1.2. O tratamento completo de um gargalo pode ser descrito como segue:

A) no início do gargalo: Determine $X_{ini} = X(E(s_i), E(s_{i-1}), E(s_{j-n-1}))$ e ache o rótulo de $L_0$;

B) no fim do gargalo: Ache $X_{end} = X(E(s_{i+n}), E(s_{i+n+1}), E(s_{j-n-1}))$: se $(X_{ini} == X_{end})$, a $\mu$ -auto - intersecção existe e o loop de $PC_k$ — contém possivelmente vários loops $\mu$-regulares — associado a um nó da loop-tree que está sendo construída, foi totalmente determinada. Por outro lado, esse loop ainda tem uma parte no *LeC*. Em qualquer caso $s_{j-n}$ e $s_{i+m}$ devem ser colocados na pilha(rótulo de $L_0$) junto

com a informação relativa a ocorrência ou não de uma $\mu$ -auto - intersecção dentro do gargalo. Também, $s_{j-n-1}$ e $s_{i+m+1}$ devem ser ligados a fim de manter o *LeC* conectado;

C) nenhuma ação é requerida nos snaxels não extremos do gargalo exceto atualizar o vértice que deve ser ligado ao próximo a ser gerado se, com a criação desse vértice, o gargalo termina;

D) o caso mais complicado onde existe um loop entre dois elementos consecutivos de $s_{j-m}, \ldots, s_j$, o que está descrito na figura 6.4(A) determina que, com a adição de um simples vértice do *LeC*, um gargalo termina e um outro começa. Assim, neste caso, as ações indicadas acima em **B** — para o primeiro gargalo — e em **A** — para o segundo — devem ser feitas em seqüência.

Além desse caso onde a aresta repetida é aquela do *Last_vertex*[*C*], também existe a situação onde o vértice corrente — $s_i$ — está sobre a aresta de *L*ast_vertex[C].prev - $s_{j-1}$ na figura 6.6. Gargalos não podem ocorrer nesse segundo caso onde sempre existe uma $\mu$-intersecção entre o loop *L* formado e o *LeC* a menos que o loop seja seguido imediatamente por um $\mu$-whisker que acaba mudando as suas conexões como no exemplo descrito na figura 4.10. Como não se sabe sobre a existência desse $\mu$-whisker, no momento em que o loop é achado, ele é processado da mesma forma que ocorre em qualquer caso. Para tornar possível evitar que essa ocorrência do $\mu$-whisker corrompa o processo de rotulação, deve-se computar $X(E(s_i), E(s_{i-1}), E(s_j))$. Eventualmente, se o segundo caso de repetição de uma aresta determina o fim de um gargalo, o tratamento específico para essa situação, dado em **B** acima, é aplicado.

Além da memória constituída pela informação na Estrutura Auxiliar o processo precisa lembrar de dois aspectos relativos ao snaxel imediatamente anterior. Ele precisa saber se esse snaxel está sobre uma aresta repetida e se este estava na terceira aresta cruzada pelo *LeC* na célula imediatamente anterior. A resposta a essas perguntas é fornecida pelas variáveis boleanas *P*rv_sx.in_a_bottleneck e *X*ed_edgs_in_Prv_Cel.

Deve ser comentado que não existe nenhum modo de que a curva-PC retorne para uma aresta de uma célula dupla que contém um snaxel que já não é mais válido. Assim, se o snaxel corrente está sobre uma aresta repetida não existe outra possibilidade dele ser explorado numa célula revisitada.

Como já foi explicado, o algoritmo faz uso do fato de que esse *C*urrent_Snaxel_nxt já foi computado para antecipar o processamento que seria realizado quando esse snaxel se

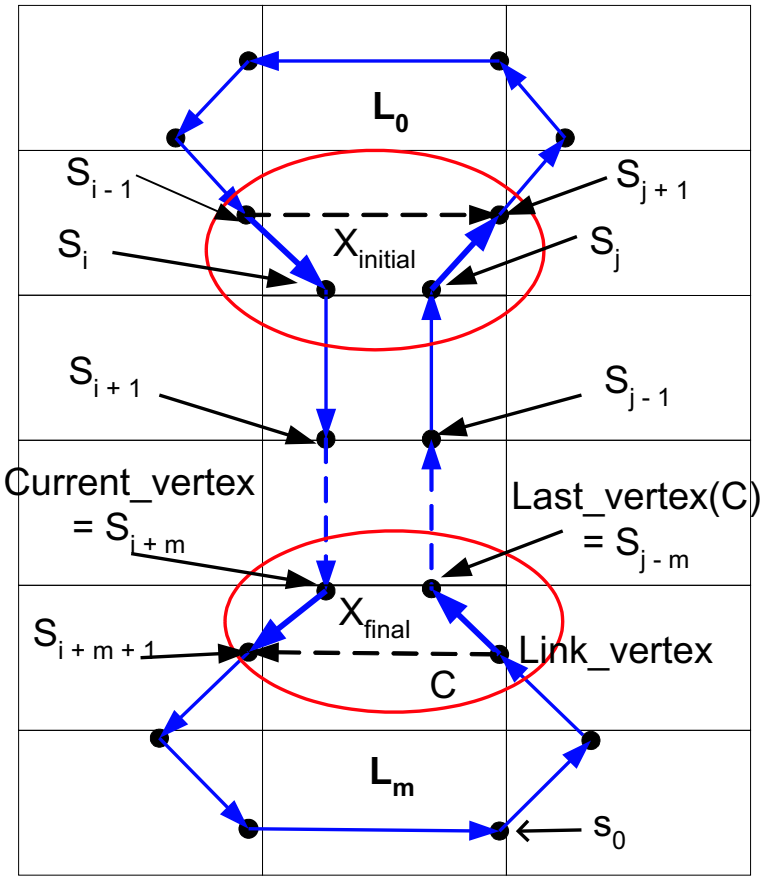


Figura 6.5: Tipos de Loops: Loops terminados numa aresta repetida(gargalo).

tornasse o snaxel corrente, se através desse snaxel a curva-PC entra em uma célula que não foi visitada ainda.

**Function Check_whether_the_Vertex_is_on_a_Repeated_Edge(Parameter_Vertex)**
**If** (*Check_whether_the_LeC_contains*(*Parameter_Vertex*,"seeking for valid loops")) **do:**
{**If** ((*Current_Vertex.edge* == (*Parameter_Vertex.edge*)$^{-1}$) **do**
{**If** (¬(*Previous_Vertex_and_Last_Vertex_of_Previous_Cell_are_on_the_same_edge*)) **do:**
{*Label = Label_the_loop_between*(*Parameter_Vertex.next,Current_Vertex.previous*)
$X_{initial}$=X(*Parameter_Vertex.next, Current_Vertex.previous, Current_Vertex*)}
**Else If** (*Parameter_Vertex* ¬ = *Link_Vertex*) **do:**
{$X_{final}$=X(*Link_Vertex, Current_Vertex, Current_Vertex.previous*)
**If** ($X_{final}$ == $X_{initial}$) *Set Self_Intersection* **else** *unset Self_Intersection*
*Push_onto_Stack*(*Link_Vertex,Current_Vertex,Label*)
*Label = Label_the_loop_between*(*Parameter_Vertex.next, Link_Vertex*)
$X_{incial}$=X(*Parameter_Vertex.next, Link_Vertex, Current_Vertex*)}
*Link_Vertex = Parameter_Vertex.previous*
*Set Previous_Vertex_and_Last_Vertex_of_Previous_Cell_are_on_the_same_edge*
**Return TRUE**}
**If** (*Current_Vertex.edge* == ((*Parameter_Vertex*).*previous*).*edge*) **do:**
{**If** (*Previous_Vertex_and_Last_Vertex_of_Previous_Cell_are_on_the_same_edge*) **do:**
{$X_{final}$=X(*Link_Vertex, Current_Vertex,* (*Current_Vertex*).*previous*)
**If** ($X_{final}$==$X_{initial}$) (*Set Self_Intersection* **else** *unset Self_Intersection*
*Push_onto_Stack*(*Link_Vertex,Current_Vertex,Label, Self_Intersection*)
*Link Link_Vertex to Current_Vertex* }
*Label = Label_the_loop_between*(*Parameter_Vertex.previous,*(*Current_Vertex*).*previous*)
*Push_onto_Stack*((*Parameter_Vertex*).*previous,*(*Current_Vertex*).*previous,Label,1*)
*Link ((Parameter_Vertex.previous).previous) to Current_Vertex*}
**Return TRUE**}}
**Return FALSE**}

Apesar de uma considerável repetição dos conceitos e argumentos empregados no texto que introduz essa rotina vamos descrevê-la detalhadamente, a seguir, comando por comando. Fazemos isso, em especial, porque esse trabalho enfoca especialmente a implementação da metodologia empregando Loop-Snakes. Por simplicidade não consideraremos inicialmente as possibilidade de haver $\mu$-whiskers imediatamente em seqüência aos loops encontrados e que com a sua eliminação poderiam até fazer com que as condições que determinaram a geração do loop deixem de existir. Esse caso será tratado, especificamente, depois.

Seja $C$, a célula da malha onde $PC_k$ está entrando através do último vértice gerado (*Current_Vertex*). O único parâmetro desse procedimento (*Parameter_Vertex*) é sempre o último vértice do *LeC* que está em uma aresta de $C$. Ele pode entretanto, ser indicado de duas maneiras distintas:

**i)** usualmente ele é dado pr *Last_Vertex*[*C*], o último vértice de $PC_k$ já gerado;

**ii)** caso o vértice definido em *i* não pertença mais ao *LeC* e *C* seja uma célula dupla, então *Parameter_Vertex* será o vértice através do qual $PC_k$ deixou *C* na visita imediatamente anterior. Este vértice é indicado em *Last_Vertex*[*C*].old.

A primeira providência a ser tomada é verificar a validade de *Parameter_Vertex*, isto é, se ele ainda pertence ao *LeC*, caso contrário loops sem representação na Loop-tree poderão ser gerados. Essa verificação é feita pela função lógica *Check_whether_the_LeC_contains*(.) que será descrita posteriormente. Confirmada essa validade testa-se, então se $PC_k$ retorna a uma aresta já visitada de *C* ou não. Em relação a isso há duas possibilidades a serem consideradas.

**i)** a aresta revisitada é aquela onde está o *Parameter_Vertex*. Essa situação fica caracterizada pela condição (*Current_Vertex.edge* == (*Parameter_Vertex.edge*)$^{-1}$) — condição $C_1$. O uso do operador $(.)^{-1}$ é necessário pelo fato de que enquanto $PC_k$ está deixando *C* através do *Parameter_Vertex*, ela está entrando em *C* pelo *Current_Vertex*;

**ii)** a aresta re-visitada é aquela onde está o vértice de $PC_k$ imediatamente anterior ao que é parâmetro. A condição $C_2 \equiv$ *Current_Vertex.edge* == ((*Parameter_Vertex*).*previous*).*edge* indica essa situação. Neste caso o operador $(.)^{-1}$ é desnecessário porque a curva projetada está entrando em *C* em ambos os vértices.

Comecemos por analisar o caso **i**. Este caso determina que se está percorrendo corretamente um gargalo, como foi definido anteriormente. Se a condição *Previous_Vertex_and_Last_Vertex*$_o f$*_Previous_Cell_are_on_the_same_edge* for falsa então o vértice corrente está dando início a um novo gargalo. As ações que se fazem necessárias quando essa circunstância específica ocorre são:

**i)** rotular adequadamente o loop formado pelo *LeC* entre os vértices anterior ao corrente e posterior ao *Parameter_Vertex*. No caso do gargalo mostrado na figura 6.5 quando o vértice corrente for $S_i$, a rotina será chamada tendo como parâmetro $s_j$ e o loop em questão será definido pelos vértices $s_{j+1}, \ldots, s_{i-1}$. Essa rotulação é feita pela rotina *Label_the_loop_between* que utiliza como parâmetros apenas os vértices de índices extremos do loop e será descrita posteriormente;

**ii)** calcular $X_{initial} = X(Parameter\_Vertex.next, Current\_Vertex.previous, Current\_Vertex)$;
: conforme estabelecido no capítulo 4, na seção 4.1.2, a função $X$ tem como parâmetros três vértices de uma $\mu$-curva que estão em arestas diferentes de uma mesma célula. Ela indica qual entre os dois primeiros é atingido antes quando se percorre o contorno da célula a partir do terceiro no sentido anti-horário. No cômputo de $X_{initial}$ os dois primeiros parâmetros de $X$ são exatamente os vértices de índices extremos do loop $L_0$. O terceiro pode ser qualquer ponto da primeira aresta da malha a ser cruzada pelo gargalo. Conforme indicado no capítulo 4, na seção 4.1.2, tendo calculado $X_{initial}$ é possível determinar ao final do gargalo se ele contem uma $\mu$-auto-intersecção de $PC_k$. Se não existir essa intersecção $L_0$ e o loop regular adjacente ao final do gargalo —$L_m$ na figura 6.5 — estarão associados ao mesmo nó da loop-tree e, portanto, terão o mesmo rótulo. Caso haja intersecção eles estarão associados a nós diferentes e o rótulo de $L_m$ deverá ser calculado usando os resultados do capítulo 5.

Em qualquer vértice em que se detecte a condição de que se está correntemente num gargalo três ações devem ser executadas independente de quaisquer condições adicionais. As duas primeiras se referem a atualização de variáveis que serão utilizadas no processamento que dever ser executado quando o próximo vértice é gerado.

**i)** a variável *Previous_Vertex_and_Last_Vertex_of_Previous_Cell_are_on_the_same_edge* deve ser setada para indicar, quando da geração do vértice seguinte que no imediatamente anterior se estava num gargalo. Assim, caso esse novo vértice ainda esteja no gargalo tem-se como saber que não é preciso executar os procedimentos específicos de um início de gargalo. Ou ao contrário, caso ele não esteja mais no gargalo pode-se então saber que os procedimentos próprios de um final de gargalo devem ser realizados;

**ii)** além disso, é preciso atualizar o chamado vértice de ligação, *Link_Vertex*. Por definição esse vértice é aquele que deve ser conectado ao próximo vértice a ser gerado caso esse não pertença mais ao gargalo, para manter a conexidade do *LeC*. Durante todo o percurso do segundo ramo do gargalo, que é onde ele é detectado o *Link_Vertex* vai sendo sucessivamente substituído pelo seu antecessor. Considere a chamada à função feita na criação de um vértice qualquer desse ramo e observe que após essa criação o *Link_Vertex* deve apontar sempre o antecessor do *Parameter_Vertex* empregado nessa chamada. Assim no exemplo da figura 6.5 ele

é feito $s_{j-1}$ depois que $s_i$ é criado, $s_{j-2}$ depois que $s_{i+1}$ é gerado e assim por diante até que com a geração de $s_{i+m+1}$, o gargalo acaba e $s_{i+m+1}$ é ligado ao valor do *Link_Vertex* no momento dessa geração que é $s_{j-m-1}$;

**iii)** além disso a função deve retornar no caso desses vértices, sempre o valor verdadeiro.

O *Link_Vertex* determinado em um dado vértice do gargalo permite ainda identificar se no vértice seguinte temos simultaneamente o final de um gargalo ($G_1$) e o começo de um segundo ($G_2$) tal como acontece no exemplo da figura 6.4. Essa situação especial que só pode ocorrer se o vértice corrente, o que é parâmetro e o de ligação estiverem numa célula dupla fica caracterizada pela condição $Parameter\_Vertex \neq Link\_Vertex$. Essa condição determina que entre os primeiros ramos de ($G_1$) e ($G_2$), existe um loop — $L_1$, na figura 6.4 — cujos vértices de índices extremos são o sucessor do *Parameter_Vertex* e o próprio *Link_Vertex*. Para que o rótulo desse loop possa ser determinado adequadamente usando os resultados do capítulo 5, entretanto é preciso determinar antes se ($G_1$) contem uma $\mu$-auto-intersecção. Isso, requer que sejam efetuados os procedimentos específicos de um final de loop para o caso de ($G_1$). No caso, esses procedimentos consistem de:

**i)** Computar:

$$X_{final} = X(Link\_Vertex, Current\_Vertex, Current\_Vertex.previous)$$

observe que o vértice de ligação é o último vértice do *LeC* que é gerado antes do primeiro ramo de ($G_1$) começar. O vértice corrente é obviamente o primeiro depois que o segundo ramo acabou e seu antecessor portanto, é o vértice de maior índice em ($G_1$). Isso é suficiente para assegurar que esses três vértices estão em arestas diferentes de uma mesma célula e assim a função $X$ pode ser aplicada a eles;

**ii)** comparando o valor de $X_{final}$ com o do $X_{initial}$, computado no início do gargalo ($G_1$) podemos decidir se esse gargalo contem uma $\mu$-auto-intersecção de $PC_k$ ou não. ($G_1$) conterá uma dessas auto-intersecções se $X_{final} == X_{initial}$ indicando que vértices adjacentes a um mesmo ramo do gargalo são atingidos primeiro nos dois percursos de contorno de células empregadas na definição dessas variáveis;

**iii)** caso ($G_1$) contenha uma auto-intersecção a variável $Self\_Intersection$ deve ser ligada. Caso contrário ela é desligada. Os exemplos das figuras 6.4(A) e (B) mostram que as duas possibilidades podem ocorrer. Definida essa variável e já tendo:

**1)** o valor do rótulo do loop adjacente aos dois ramos de $(G_1)$ — $L_0$, na figura 6.4(a) —, que foi computado quando $(G_1)$ foi, inicialmente, detectado;

**2)** os vértices extremos de $(G_1)$: *Link_Vertex.next* e *Current_Vertex.previous*.

temos todos os elementos que compõem o registro relativo a $L_0$ que deve ser acrescentado à pilha onde estão representados todos os filhos do *LeC* na loop-tree $\mu$-regular. Literalmente, esse acréscimo é feito pelo comando:

$$Push\_onto\_Stack(Link\_Vertex.next, Current\_Vertex.previous, Label, Self\_Intersection)$$

Isso encerra o tratamento específico de fim de gargalo referente a $(G_1)$.

O procedimento de início de gargalo que tem de ser efetuado, em seguida, para $(G_2)$ consiste em:

**i)** Rotular o loop existente entre os primeiros ramos de $(G_1)$ e $(G_2)$, o que é feito por

$$Label\_the\_loop\_between(Parameter\_Vertex.next, Current\_Vertex.previous)$$

e guardar o resultado para o uso no final de $(G_2)$.

**ii)** Computar

$$X_{incial} = X(Parameter\_Vertex.next, Link\_Vertex, Current\_Vertex)$$

Observe que com a eliminação do trecho de $PC_k$ contendo $(G_1)$ e $(L_0)$, o *Link_Vertex* deveria passar a ser o vértice anterior ao corrente[2]. Fazendo isso os parâmetros dessa chamada à função $X$ seriam os mesmos da que gerou $X_{initial}$ no tratamento do caso em que se inicia um gargalo e que já foi descrito.

Com isso encerramos a descrição dos procedimentos que precisam ser efetuados quando $C_1$ ocorre. Passamos então a tratar do caso da condição $C_2$.

Nesse caso, desconsiderando inicialmente a possibilidade dele ser seguido por um $\mu$-whisker sempre existe uma $\mu$-auto-intersecção de $PC_k$ dentro do conjunto constituído pelas poligonais definidas por:

**a)** O vértice corrente e os adjacentes a ele em $PC_k$.

[2]A eliminação desse trecho não é completada pelo procedimento, fazendo-se a ligação direta entre o vértice corrente e o *Link_Vertex*, por que se sabe que ao final de $(G_2)$ um trecho maior, incluindo esse segundo gargalo deverá ser eliminado.

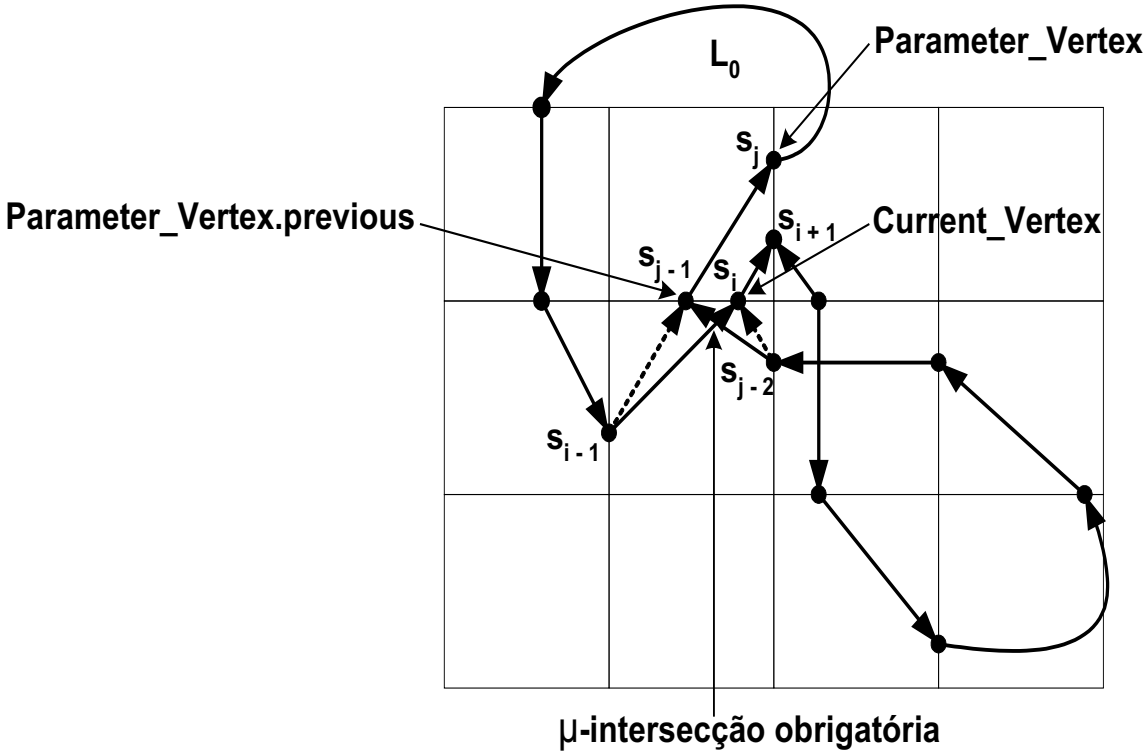


Figura 6.6: Figura mostrando um loop determinado pela condição $C_2$ (*Current_Vertex*).*edge* == ((*Parameter_Vertex*).*previous*).*edge*.

**b)** O *Parameter_Vertex* e os dois que o antecedem em $PC_k$(ver figura 6.6).

Temos de fato duas opções para a definição do loop. Ou tomamos como seus vértices extremos os antecessores do vértice corrente e do parâmetro ou o *Parameter_Vertex* e o vértice corrente. Adotamos a primeira alternativa porque a segunda traria o seguinte pequeno problema:

- Para restabelecer a conectividade do *LeC* precisaríamos conectar diretamente o (*Parameter_Vertex*).*previous* ao (*Current_Vertex*).*next*. Como esse último vértice ainda não foi gerado e pode não vir a sê-lo — o que ocorre quando a determinação da curva projetada acaba no vértice corrente — precisaríamos, ao identificar que a geração da curva terminou, substituir essa conexão por uma entre o antecessor do *Parameter_Vertex* e o primeiro vértice de $PC_k$ a ser gerado. Teríamos assim, um trabalho adicional desnecessário dado que ele não é requerido pela outra opção.

Aproveitamos a oportunidade para comentar que o código faz, sim, referências, ao antecessor de um vértice de $PC_k$. Se esse vértice é o primeiro a ser gerado, seu vértice anterior não estará definido. Essa situação pode, no entanto, ser contornada criando um vértice artificial — $v_0$ cujas coordenadas *CAP* são as seguintes:

**i)** a coordenada de célula é a identificação da célula_inicial de $PC_k$;

**ii)** a coordenada de aresta tem um valor diferente dos que identificam qualquer aresta de célula — 0,1,2 ou 3. Com isso $v_0$ jamais será incluído dentro de um loop, seja ele determinado pela repetição de aresta ou um knot-loop, e em conseqüência as rotinas que efetuam uma mudança topológica jamais farão referência ao seu antecessor;

**iii)** a coordenada de pixel pode ser qualquer.

Esse vértice é ligado ao primeiro a ser obtido pelo processo de geração de $PC_k$ e assim eliminamos a possibilidade do antecessor de um vértice referido nas rotinas mencionadas acima não estar definido.

Dado, então que nas condições que estamos considerando sempre há uma $\mu$-auto-intersecção e devido a opção feita em relação aos extremos do loop encontrado, o tratamento do caso determinado pela condição $C_2$ consiste dos seguintes comandos:

**i)** rotular corretamente o loop formado, cujos vértices extremos conforme já foi visto, são os antecessores do vértice corrente e do *Parameter_Vertex* (ver figura 6.6). Observe que, ao contrário do caso em que ocorre um gargalo nenhum vértice é desconsiderado — ou fazem parte do loop ou continuam sendo vértices do *LeC*;

**ii)** dado que nas condições assumidas esse loop tem sempre uma $\mu$-auto-intersecção com o *LeC* a variável $Self\_Intersection$ deve valer sempre 1. Portanto, o registro relativo ao loop a ser incorporado ao stack contendo os filhos $\mu$-regulares do *LeC* é constituído por;

$$((Parameter\_Vertex).previous, (Current\_Vertex).previous, Label, 1)$$

**iii)** para recuperar a conectividade do *LeC* ligamos diretamente (*Parameter_Vertex*).*previous* ao vértice corrente. Observe que como (*Parameter_Vertex*).*previous* tem a mesma coordenada de aresta que o vértice_corrente ele não pode ser o vértice artificial — $v_0$ — que é o único sem antecessor. Portanto, o antecessor de (*Parameter_Vertex*).*previous* está bem definido;

**iv)** finalmente, também nesse caso a função deve retornar o valor verdadeiro.

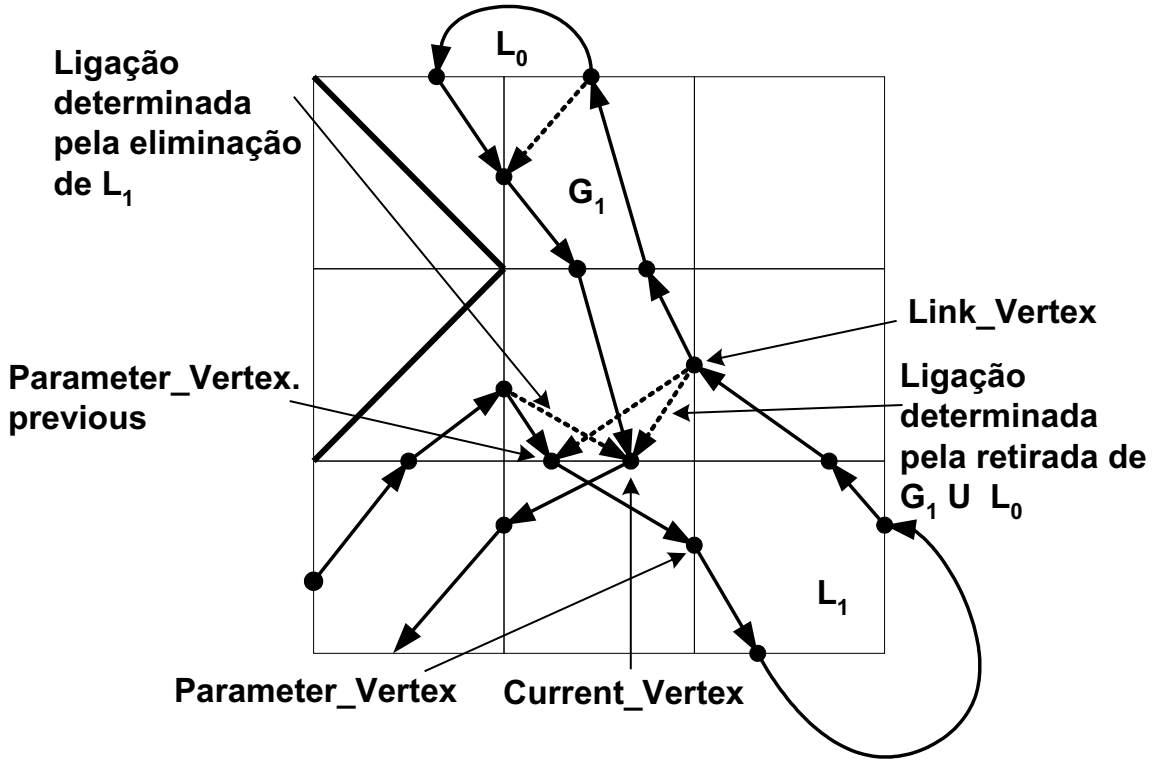


Figura 6.7: Exemplo de gargalo encerrado por vértice satisfazendo a condição $C_2$.

Pode acontecer ainda, tal como ocorre no caso da figura 6.7 que a condição $C_2$ determine o fim de um gargalo ($G_1$). Nesse caso o procedimento específico de fim de gargalo deve ser executado para ($G_1$) antes de se tratar, o próprio loop determinado por $C_2$. Esse procedimento consiste dos três comandos já descritos quando tratamos do caso em que um gargalo acaba e outro simultaneamente de inicia. Inclusive os parâmetros das rotinas chamadas são os mesmos. Através da variável

$$Previous\_Vertex\_and\_Last\_Vertex_of\_Previous\_Cell\_are\_on\_the\_same\_edge$$

sabemos que um gargalo foi encerrado com o acréscimo do vértice_corrente e os três comandos mencionados acima são:

- $X_{final} = X(Link\_Vertex, Current\_Vertex, Current\_Vertex.previous)$
- **If** $(X_{final} == X_{initial})$ $Set\ Self\_Intersection$ **else** $unset\ Self\_Intersection$
- $Push\_onto\_Stack(Link\_Vertex, Current\_Vertex, Label, Self\_Intersection)$

Finalmente, devemos fazer com que o *Link_Vertex* passe a ser o antecessor do vértice_corrente, restabelecendo a conectividade do *LeC* que foi perdida com a retirada de ($G_1$) e do trecho de $PC_k$ entre seus dois ramos.

Encerrando a descrição da rotina se o *Parameter_Vertex* não é mais válido, ou as condições $C_1$ e $C_2$ são simultaneamente falsas a função deve retornar o valor falso.

**Procedure Check whether the Vertex is on_a_non_Repeated_Edge(Parameter_vertex)**
{**If** (*Previous_Cell_has_3 crossings*) **do:**
{ **If** (*Parameter_vertex.edge* = ((*Parameter_vertex*).*previous*).*edge*) **do:**
{ **If** (*Check_whether*(*Last_vertex_in_previous_cell*, "is on the LeC")) **do:**
{*Label = Label_the_loop_between*(*Last_vertex_in_previous_cell*,(*Parameter_vertex*).*previous*)
*Push_onto_Stack*(*Last_vertex_in_previous_cell*,(*Parameter_vertex*).*previous*, *Label*)
*Link* ((*Last_vertex_in_previous_cell*).*previous*)*to Parameter_vertex*}}
**Else If** (*Check_whether_the_LeC_contains*(*Last_vertex_in_previous_cell*, "searching_for_new_double_cells") **do:**
{*Push_onto_a_Stack*(*Last_vertex_in_previous_cell, Parameter_vertex*, "Double Cell")
{(*Parameter_vertex*).*old = Last_vertex_in_previous_cell* }}
**Else**
{ **If** *(Previous_vertex_and_Last_vertex_of_Previous_Cell_are_on_the_Same_Edge*) **do:**
{$X_{final}$ = X(*Link_vertex, Parameter_vertex*,(*Parameter_vertex*).*previous*)
**If** ($X_{final}$ == $X_{initial}$) *Push_onto_Stack*(*Link_vertex, Parameter_vertex, Label*)
*Link Link_vertex to Parameter_vertex* }
*Last_vertex_in_previous_cell = Current_cell .Last_vertex*}}

Se o *LeC* entra numa célula $C_i$, cruzando no vértice $v_i$ uma aresta ($e_i$) não atingida por ele antes, então dois casos distintos, referenciados aqui por I e II precisam ser tratados. No caso I a célula cruzada imediatamente antes — $C_{i-1}$ — já foi visitada anteriormente pelo *LeC* mas também o vértice $v_{i-1}$ através do qual ele entra nessa célula pela última vez está situado numa aresta da malha ($e_{i-1}$) que não foi atingida antes. $C_i$ pode ser uma célula que já foi cruzada pela curva projetada através de um segmento que não corta $e_i$, como é o caso da primeira chamada à rotina feita pela procedure *Revisiting_a_Cell* ou mesmo ser uma célula que está sendo atingida pela primeira vez em $v_i$, como é o caso da segunda chamada feita à rotina em *Revisiting_a_Cell*. Ver figuras 6.8 e 6.9 abaixo.

O caso I é identificado pelo fato da variável de controle *Previous_Cell_has_3_crossings* ter sido setada quando da geração de $v_{i-1}$, ao se constatar que a situação descrita acima envolvendo $v_{i-1}$, $e_{i-1}$ e $C_{i-1}$ ocorre. Dentro do caso I há duas possibilidades, referidas por A e B, que tem de ser consideradas. Na primeira $e_{i-1}$ e $e_i$ são paralelas o que pode ser determinado pela condição:

*Parameter_vertex.edge = = (Parameter_vertex.previous).edge*

Como a célula $C_{i-1}$ já foi visitada antes pelo LeC e suas arestas $e_{i-1}$ e $e_i$ , não, pode-se inferir que o último segmento ($\sigma$) de $PC_k$ em $C_{i-1}$ gerado antes de $v_{i-1}$ liga pontos das

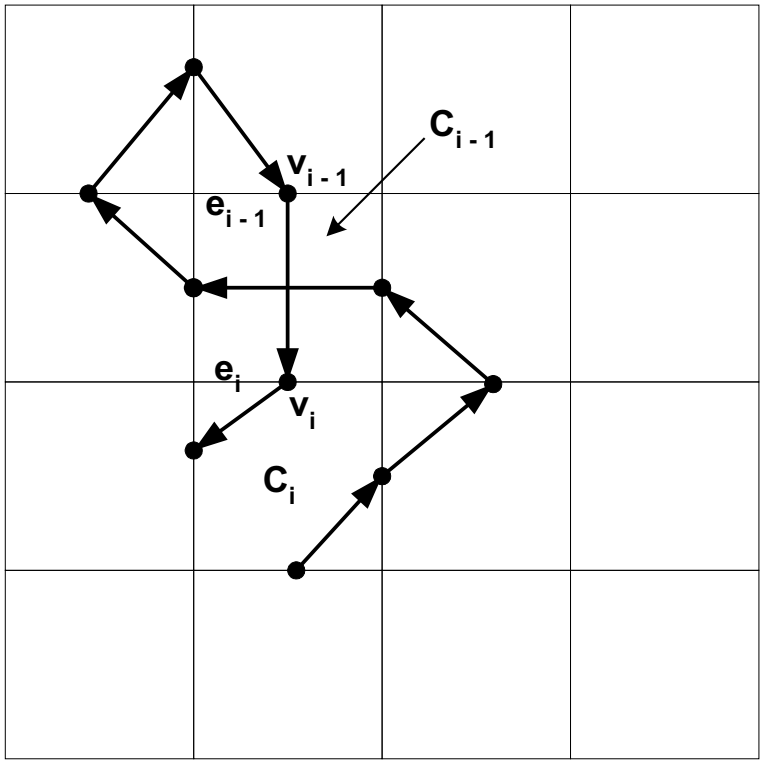


Figura 6.8: Caso em que $C_i$ já foi visitada pela curva $PC_k$.

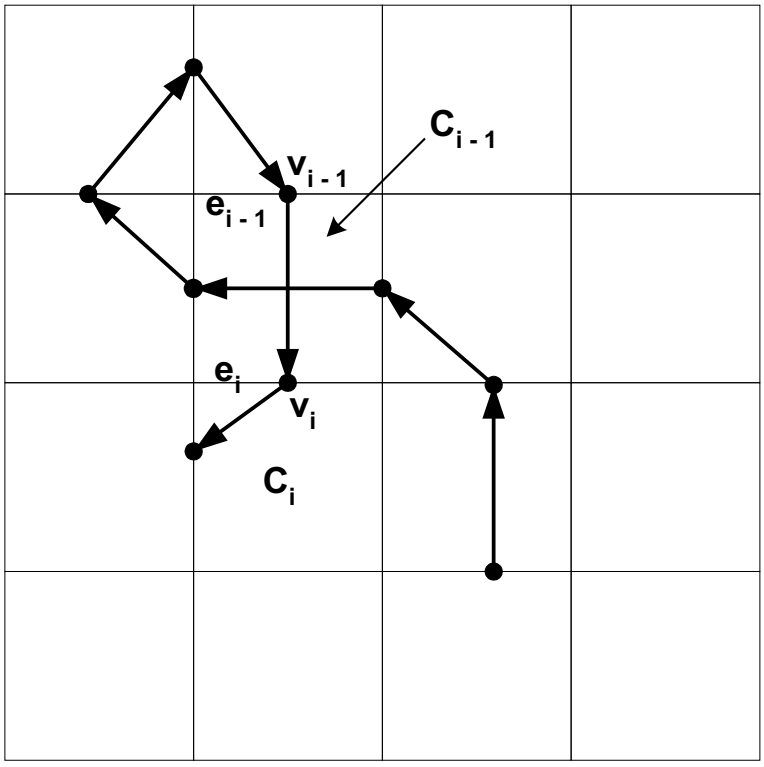


Figura 6.9: $C_i$ está sendo atingida pela primeira vez por $PC_k$ no vértice $v_i$.

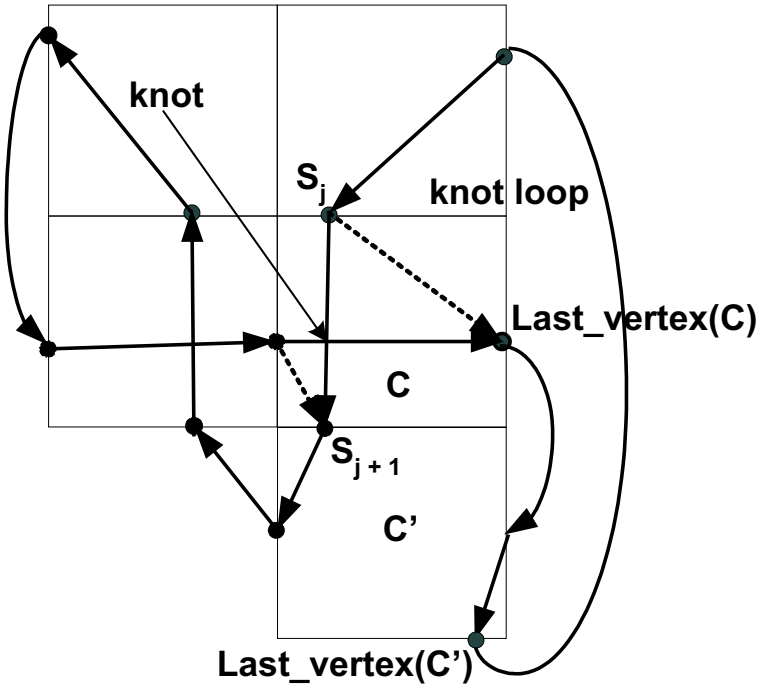


Figura 6.10: Exemplo de um *Knot-loop*.

arestas de $C_{i-1}$ que tem direção diferente da de $e_{i-1}$ e $e_i$. Desse modo com a adição de $[v_{i-1}, v_i]$ se gera claramente um auto-cruzamento de $PC_k$, o qual não pode ser removido se substituindo a curva projetada por qualquer outra $\mu$-equivalente a ele. Se o loop formado for válido ele será, portanto, um *knot-loop*. Ver figura 6.10. Verificar a validade desse loop significa, simplesmente, checar se o segmento, ou equivalentemente, seu vértice final *Last_vertex_in_previous_cell*, de fato ainda pertencem ao *LeC*. Isso é feito pela procedure *Check_whether_the_LeC_contains* e como no caso do loop não ser válido, isso vai significar a ocorrência de uma intersecção do *LeC* com um filho fechado dele determinado anteriormente, essa procedure é chamada com a indicação de se verificar especificamente essas intersecções. A outra opção para o segundo parâmetro dessa procedure é verificar se uma célula dupla válida está sendo gerada, que conforme iremos ver, será empregada no caso da possibilidade B.

No caso do Loop ser válido seu rótulo será determinado pela função *Label_the_loop_between* aplicada às suas extremidades dadas pelo nó final de — *Last_vertex_in_previous_cell* — e $v_{i-1}$ = *Parameter_vertex.previous*. Essas extremidades são então armazenadas na pilha relativa ao label do loop e o novo traçado do *LeC*, decorrente da retirada do novo loop é determinado ligando-se diretamente o vértice anterior a seu vértice inicial, que é, exatamente, o vértice inicial de $\sigma$ —

*Last_vertex_in_previous_cell . previous* — ao vértice posterior ao seu vértice final, que é $v_i$ = *Parameter_vertex*.

Na possibilidade B do caso I, $e_{i-1}$ e $e_i$ tem direções diferentes. Lembrando uma vez mais que nas condições do caso I $C_{i-1}$ já foi visitada por $PC_k$ antes da geração de $v_{i-1}$, $e_{i-1}$ e $e_i$, não, teremos que o segmento, definido acima, que não pode, então, cruzar essas arestas, deve ligar agora, pontos de arestas de $C_{i-1}$ com direções diferentes. Isso significa que $\sigma_i = [v_{i-1}, v_i]$ e $\sigma$ são separados por uma diagonal de $C_i$ sendo portanto disjuntos. Assim, com a inclusão de $v_i$ a intersecção de $PC_k$ com $C_{i-1}$ passa a conter dois segmentos disjuntos. $\sigma_i$ pertence, obviamente, ao *LeC*. Se o mesmo acontecer com $\sigma$, $C_{i-1}$ se tornará um célula dupla válida. Para tanto, basta checar seu vértice final — *Last_vertex_in_previous_cell* pertence ao *LeC*. Isso é testado pelo procedimento *Check_whether_the_LeC_contains*, agora tendo como foco distinguir novas células duplas válidas de não válidas. Nesse último caso $\sigma$ está contido em um loop já encontrado que pode ser aberto ou fechado. Isso acarreta que, enquanto no caso de validação de um loop apenas o conteúdo da pilha fechada precisava ser analisada, para validar uma célula dupla tanto o conteúdo da pilha fechada como o da pilha aberta precisam ser considerados. Nas figuras 6.11 e 6.12 se representam casos em que se tem, respectivamente, uma dupla célula válida e uma não válida. Caso a célula dupla seja válida, $v_i$ = *Parameter_vertex* é armazenado numa pilha — *Double_Cells Stack* — que contem os últimos vértices gerados em células duplas válidas, isto é, duplamente cruzados pelo *LeC*. Estabelecemos também uma ligação direta entre $v_i$ e o vértice final de $\sigma$ fazendo:

$$Parameter\_vertex.old = Last\_vertex\_in\_previous\_cell$$

Isso nos permitirá recuperar todos os vértices do *LeC* em $C_{i-1}$ tendo $v_i$. Poderemos, portanto, identificar esses vértices para, por exemplo, na iteração seguinte, quando os loops abertos da iteração corrente forem tornados snakes, restringir mais estritamente o movimento de snaxels em células duplas contendo uma diagonal interior a snake.

No caso II, a célula $C_{i-1}$ não atende às condições requeridas no caso I, o que significa que ela pode tanto: i) não ter sido atingida por $PC_k$ antes de $v_{i-1}$ ou ii) $PC_k$ pode já ter cruzado $e_{i-1}$ antes da criação desse vértice, dado, que, por hipótese ele não pode ter cruzado $e_i$. Nas figuras 6.13 e 6.14, respectivamente, se representam os casos i e ii. No caso de $e_{i-1}$ ser uma aresta repetida,a geração de $v_i$, pode encerrar um gargalo, situação que fica caracterizada pelo fato do vértice anterior ($v_{i-1}$) estar na mesma aresta do último vértice de $PC_k$ a ser gerado em $C_{i-1}$ . A ocorrência dessa situação é identificada pelo fato da variável de

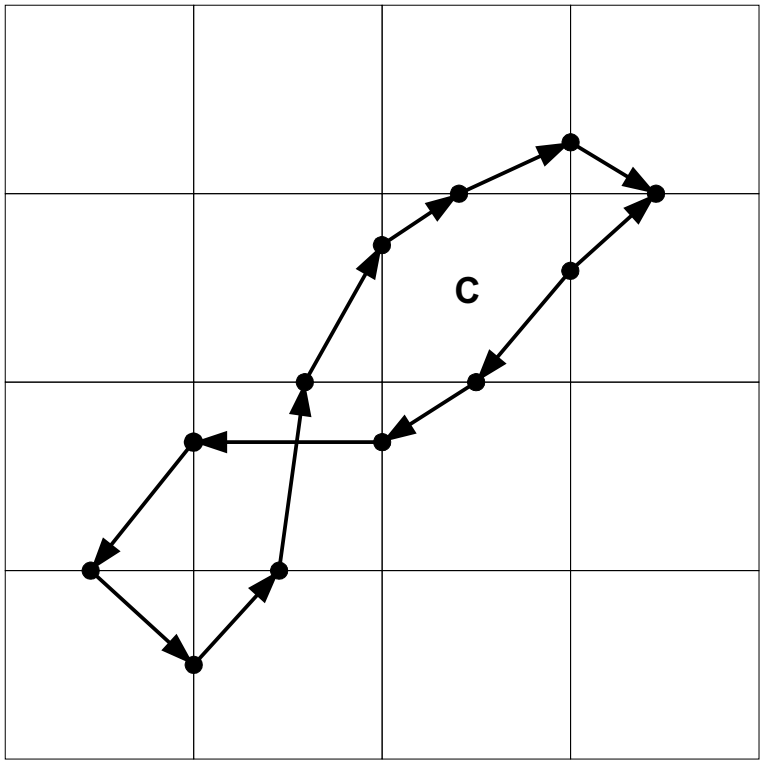


Figura 6.11: A célula *C* é uma célula dupla válida.

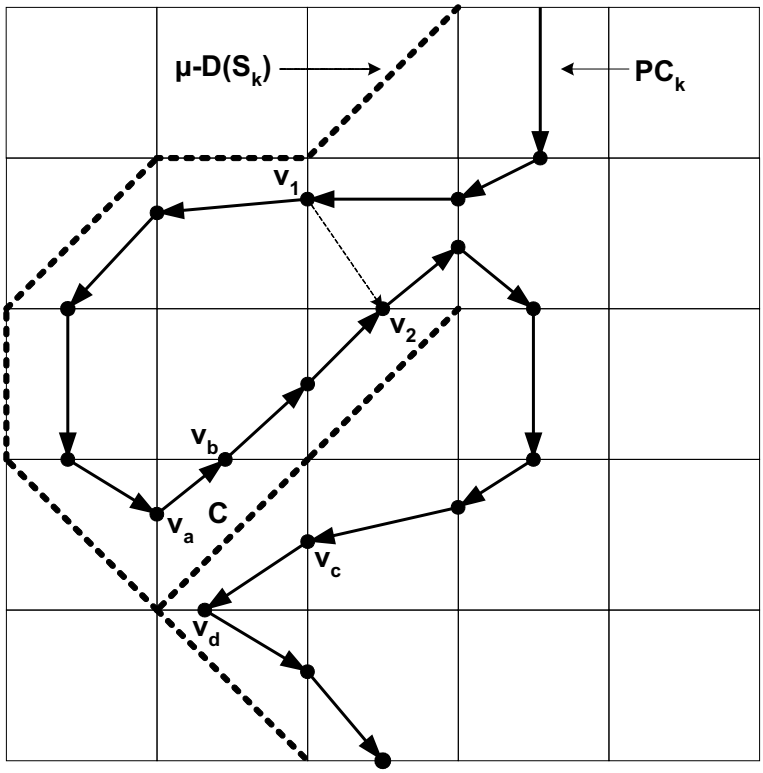


Figura 6.12: A célula *C* não é uma célula dupla válida por que quando o segmento $[v_c, v_d]$ é gerado, $[v_a, v_b]$ não pertencem mais ao *LeC* pois já faz parte de loop delimitado por $v_1$ e $v_2$.

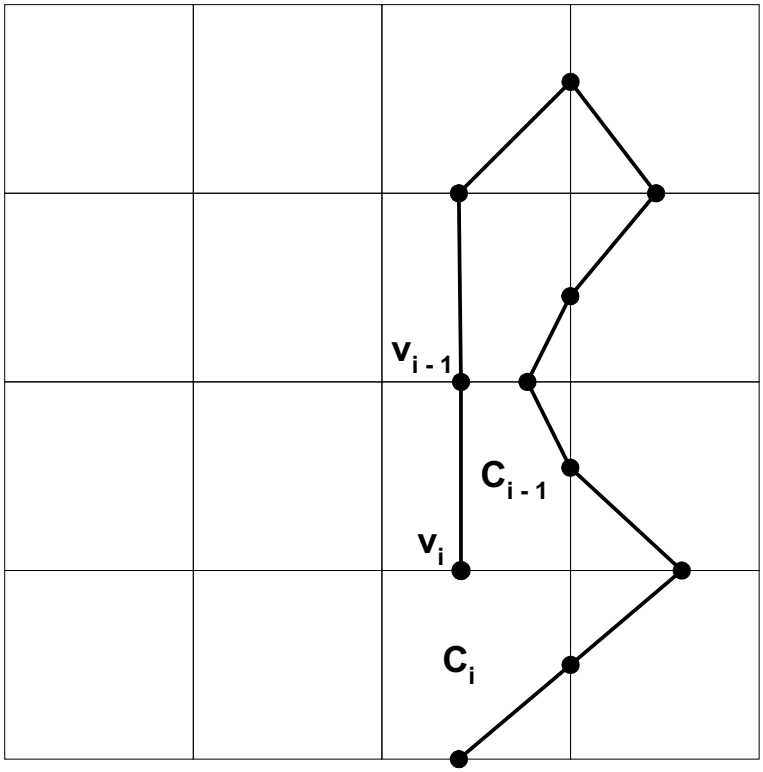


Figura 6.13: Caso em que $e_{i-1}$ já foi atingida por $PC_k$. A geração de $v_i$ determina então, o fim de um gargalo e requer que o tratamento específico desse caso seja realizado.

controle *Previous_vertex _and_ Last_vertex_of_Previous_Cell_are_on_the_Same_Edge* estar setada . Isso acontecendo, os procedimentos que devem ser efetuados ao final de um gargalo para identificar se um novo nó da árvore deve ser gerado e atualizar os apontadores que determinam o novo traçado do *LeC* são executados. Afora isso, diante da possibilidade de, ao se gerar o vértice seguinte, se tenha, exatamente, o caso I, em qualquer dos casos i) ou ii), se deve prover o valor de *Last_vertex_in_previous_cell*, que será usado então fazendo:

*Last_vertex_in_previous_cell = Current_cell .Last_vertex*

### 6.1.3 Validando, Rotulando e Processando um Loop

Uma folha *L* da Loop-Tree pode ser identificada pelo fato de que seu vértice inicial foi criado depois de qualquer um em um loop já formado. Seu rótulo deve ser determinado por um dos dois esquemas exibidos na secção 5.3 . Agora, vamos considerar o modo como um loop *L* não-folha é rotulado. Se um novo loop intersecta um outro achado anteriormente, então, através do lema 5.1.3 , este deve ser fechado. Essa intersecção

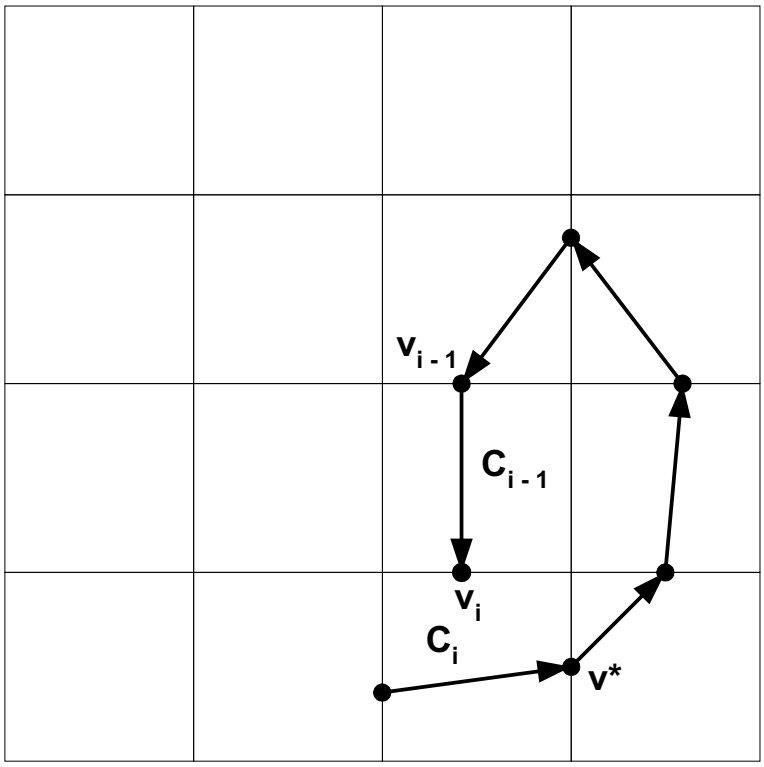


Figura 6.14: Caso em que $C_{i-1}$ não foi atingida antes de $v_{i-1}$. Nesse caso a única preocupação é fazer *L*ast_Vertex_In_Previous_Cell = $v^*$, para uso por esta mesma rotina após a geração de $v_{i+1}$.

existe se e somente se o loop é iniciado antes da última intersecção do *LeC* com um de seus filhos. Se este não for o caso, o próximo resultado que pode ser tentado para rotular *L* é o lema 5.1.2 que estabelece que qualquer loop que tem um filho aberto é fechado. Um filho aberto existe se e somente se o vértice inicial de *L*, *I*ni_vertex, foi gerado.

```
Function Check_whether_LeC_contains(parameter_vertex)
{ While (Parameter_vertex < (Top_of_Stack).initial_vertex)
        { If (Top_of_Stack). is_separated_from_the_LeC_by_a_self_intersection)
            If (((Top_of_Stack).label == "OPEN") ||
            ((Top_of_Stack).label == "CLOSED_WITH_INTERSECTION"))
               Label_of_the_parent_of_the_Last_Element_Removed_from_Stack = "CLOSED";
            Else
                 Label_of_the_parent_of_the_Last_Element_Removed_from_Stack = "OPEN";
        Else
             Label_of_the_parent_of_the_Last_Element_Removed_from_Stack = (Top_of_Stack).label;
        Pop_out_of_Stack (Top_of_Stack);}
        If (Parameter_vertex ≤ (Top_of_Stack).final_vertex)
          { If ((Top_of_Stack).label == "CLOSED")
              (Top_of_Stack).label = "CLOSED_WITH_INTERSECTION";
          Return FALSE;}
        Else
            Return TRUE;}
```

A função *Check_wether_LeC_contains*, tem as duas seguintes finalidades:

**i.** verificar se o vértice dado como parâmetro pertence ao *LeC*;

**ii.** atualizar a variável global *Label_of_the_parent_of_the_Top_of_Stack* que indica o rótulo que o pai do loop no topo da pilha deve ter em função do label desse loop. Conforme iremos ver, no momento que um loop verdadeiro é determinado, seu label será o valor corrente dessa variável;

Comecemos por analisar os possíveis valores do parâmetro da rotina. Nessa análise *C* representa a célula corrente e *C′*, a anterior a ela. Há três possibilidades para o *Parameter_Vertex*:

**a.** ele é o último vértice anteriormente gerado em *C*. Isso ocorre no caso do *LeC* voltar a uma aresta;

**b.** ele é o vértice final do penúltimo segmento de $PC_k$ em *C*. Isso ocorre quando a célula corrente já foi cruzada pelo menos duas vezes e o último segmento de $PC_k$ *C* encontra-se dentro de um loop já encontrado. Tanto nesse caso como no anterior a função permite evitar que um seja gerado falso loop por repetição de aresta;

**c.** ele é o último vértice de $C'$ gerado antes do antecessor do vértice corrente. Isso acontece quando $C'$ está sendo re-visitada mas o vértice corrente e seu antecessor estão em arestas dela que não foram cruzadas pelo *LeC* antes. Nesse caso a função se destina evitar que falsos knot-loops sejam criados.

Deve-se esclarecer que o uso dessa função é necessário porque, para conferir maior agilidade computacional ao processo, os vértices de loops já encontrados não são apagados, como é feito pelo procedimento dado no capítulo anterior que gera a árvore de loops. Mais especificamente, não se apaga na *estrutura_auxiliar* a referência a um deles. Com isso se consegue cumprir o propósito de só percorrer a curva projetada uma única vez. Desse modo, a possibilidade de $PC_k$ voltar a uma célula que ela já havia atingido dentro de um loop já encontrado precisa ser distinguida daquela em que versão corrente do *LeC* volta a uma célula já visitada por ele, determinando a geração de um novo loop válido. O seguinte rationale é utilizado para fazer essa distinção de forma que o tempo computacional despendido com isso durante a atualização de toda a snake num dado estágio de sua evolução seja da ordem do número de loops verdadeiros, gerados nesse estágio:

**i.** Obviamente um loop filho do *LeC* só é iniciado depois que o anterior é terminado e o intervalo definido pelos índices de seus nós iniciais e finais contém os índices de todos os seus descendentes. Esse último fato, em particular, determina que um falso loop iniciado no *Parameter_Vertex* ocorre se e só se esse vértice tiver sido gerado entre os vértices inicial e final de um filho do *LeC*.

**ii.** Considere a seqüência de eventos que poderia determinar a criação de falsos loops incluindo partes do *LeC* corrente, ou seja, as situações em que ele e um loop que é filho dele cortam a mesma aresta ou em que uma aresta dele e uma de um filho formam uma cruz dentro de uma dada célula. A ordenação dos filhos do *LeC* envolvidos nesses eventos, segundo a cronologia em que eles acontecem durante a geração de $PC_k$, é, exatamente, a inversa daquela obtida considerando-se os momentos em que esses filhos são iniciados ou completados.

Esse fato garante o seguinte: Suponha que $L'$ é um filho do *LeC* corrente cujo vértice inicial é posterior ao *Parameter_Vertex* de uma chamada a essa rotina. Nesse caso, dessa chamada em diante, enquanto $L'$ se mantiver como filho do *LeC*, a curva projetada não formará nenhum falso loop que contenha uma parte dele. Isso é verdade porque:
Se a re-visita a uma célula que determinou a chamada a rotina, acarretar a criação de

um novo loop verdadeiro — $L''$ — esse loop começará em *Parameter_Vertex* e assim, como o vértice inicial de $L'$ é posterior a ele, a partir desse momento $L'$ passará a ser um filho de $L''$ e não mais do *LeC*. Portanto, se ele continua sendo filho do *LeC* depois dessa re-visita é porque ela não acarretou a geração de um novo Loop. Isso implica que essa chamada foi determinada pela formação de um falso loop envolvendo um filho do *LeC*, $L'''$, o qual, por **ii.**, deve, necessariamente, ter sido gerado antes de $L'$ . Por conta disso, ele continuará sendo um filho do *LeC*, enquanto $L'$ o for. Assim, se $PC_k$ vier posteriormente a formar um falso loop incluindo parte de $L'$, e este ainda for filho do *LeC* isso irá contradizer o resultado expresso em **ii.**, pois $L'''$ terá, então, estado nessa situação antes apesar de ter sido gerado primeiro que $L'$.

Esse resultado sugere que a detecção de falsos loops pode ser efetuada de forma eficiente armazenando numa pilha os loops verdadeiros gerados no momento de sua criação e retirando da pilha aqueles cujo vértice inicial tiver um índice maior que o de *Parameter_Vertex*. Em relação a composição dessa pilha deve-se observar que:

**iii.** Ela é inteiramente composta por filhos do *LeC* corrente pois no momento que o pai definitivo de um loop $L$ na $LT$ — $\mu$-regular é gerado é feita uma chamada à rotina com *Parameter_Vertex* sendo seu vértice inicial. Nessa chamada , se $L$ ainda estiver na pilha será retirado dela pois foi gerado depois desse vértice inicial.

**iv.** Em função do resultado da observação **ii.**, não há perda de informação útil para distinguir um loop verdadeiro de um falso, por se retirar da pilha os loops gerados depois do *Parameter_Vertex*.

**v.** Seja $F_L$, o número de loops falsos que $PC_k$ forma com partes do loop $L$ e que são formados, enquanto L é um filho do LeC. Seja ainda $V_L$, o número de loops verdadeiros determinados quando $L$ é o último filho do *LeC* gerado antes de seu vértice inicial. Considerando todas as chamadas à rotina feitas no estágio $k$, o número de comparações entre o nó inicial de $L$ e o *Parameter_Vertex* que precisam ser realizadas — $n_L$ — é limitado por: $V_L + F_L + 1$. Seja $V$ o número total de loops verdadeiros gerados no estágio $k$, como $F_L$ é limitado por 2, temos somando o valor de $n_L$, obtido para cada um desses loops que:

$$\sum_L n_L = \sum_L (V_L + F_L + 1) = \sum_L (F_L + l) \leq V + 3V = 4V, \tag{6.1}$$

Como o número de comparações com os vértices finais dos loops da pilha é, certamente, menor que o realizado em relação aos nós iniciais temos que o número de comparações feitas por todas as chamadas à rotina feitas no estágio $k$ é o $(V)$. Se não usássemos uma estrutura e critérios adequados, a classificação dos loops em falsos ou verdadeiros poderia ser quadrática, mais especificamente da ordem do produto dos números de nós verdadeiros e falsos.

Tendo em vista os resultados acima para fazer, pura e simplesmente, a distinção entre loops falsos e verdadeiros basta empregar a simples seqüência de comandos abaixo

{**While** (*Parameter_vertex* < (em Top_of_Stack)*.initial_vertex)*
{.....
*Pop_out_of_Stack (Top_of_Stack)*;}
**If** (*Parameter_vertex* ≤ (*Top_of_Stack).final_vertex)*
{.....
**Return FALSE** ; }
**Else**
**Return TRUE** ;}

Nessa seqüência a função *Pop_out_of_Stack*, simplesmente remove *Top_of_Stack* da pilha e sua implementação, obviamente, depende de como essa estrutura é montada. *Top_of_Stack* não precisa é claro ser um apontador como indicado nessa seqüência.

Se a retirada de elementos da pilha, feita nos moldes dados acima, não traz conseqüências em relação a se poder classificar os loops como falsos ou verdadeiros, o mesmo não se pode dizer em relação a rotulá-los. Afinal, essa retirada faz com que no momento em que um loop $L$ é confirmado, teremos excluído da pilha todos os seus filhos, cujos rótulos são necessários para rotular $L$. De fato, considerando que $PC_k$ é obtida por um mapeamento adequado, então, para rotular $L$ se ele não for uma folha da Loop-Tree precisamos:

**I)** conhecer o label de um único filho se esse filho não for separado dele por uma $\mu$-intersecção, ou senão, se for aberto;

**II)** saber se um filho fechado é disjunto do loop ou o intercepta, no sentido de formar um loop falso com ele.

Portanto, podemos rotular um loop tendo informações adequadas em relação a um só de seus filhos. Suponha, então que guardamos sempre o label que o pai de $L_U$, o último loop que foi retirado da pilha deve ter em função do label de $L_U$ e das informações disponíveis até o momento concernentes a posição de $L_U$ em relação ao *LeC* — especificamente: se $L_U$ e o *LeC* se $\mu$-intersectam, formam um falso loop ou são disjuntos. Observe que essa informação pode ter variado enquanto $L_U$ estava na pilha, pois quando ele foi posto lá, certamente, não formava um falso loop com o *LeC*. Tendo em vista, mais uma vez, a observação **ii.**, um falso loop contendo parte de $L_U$ só pode ocorrer quando ele for o topo da pilha e apenas se ele for fechado, isso tornará diferente o rótulo de seu pai. Portanto para verificar se o rótulo a ser atribuído ao pai de $L_U$ mudou, durante a sua estada na pilha, precisamos apenas verificar a posição relativa do topo da pilha em relação ao *LeC* no caso dele ser fechado. Para manter um registro de que essa mudança ocorreu mudamos o label do topo para "fechado e formando um falso loop".
Ele manterá o rótulo fechado apenas se for disjunto do *LeC*. Essas medidas são implementadas pelo código:

**If** (*Parameter_vertex* ≤ (*Top_of_Stack*).*final_vertex*)
{**If** ((*Top_of_Stack*).*label* == "CLOSED")
(*Top_of_Stack*).*label* = "CLOSED_WITH_INTERSECTION"
.....}

Assim quando $L_U$ é retirado da pilha em função de seu label, que agora pode ser, aberto, fechado ou fechado e formando um falso loop e da informação relativa a existência ou não de uma $\mu$-intersecção com o *LeC* será possível obter diretamente o rótulo de seu pai, usando os resultados da seção 5.1. Esse rótulo, é guardado numa variável global cuja a atribuição, usando esses resultados, é feita pelo código:

{**If** ((*Top_of_Stack*) . *is_separated_from_the_LeC_by_a_self_intersection*)
**If** (((*Top_of_Stack*).*label* == "OPEN") ||
((*Top_of_Stack*).*label* == "CLOSED_WITH_ INTERSECTION")
*Label_of_the_parent_of_the_Last_Element_Removed_from_Stack* = "CLOSED" ;
**Else**
*Label_of_the_parent_of_the_Last_Element_Removed_from_Stack* = "OPEN" ;
**Else**
*Label_of_the_parent_of_the_Last_Element_Removed_from_Stack* = (*Top_of_Stack*).*label* ;
.....}

Observe, finalmente, que no momento em que um loop $L$, que não é uma folha da LT $\mu$-regular, é confirmado como verdadeiro, o último loop que foi retirado da pilha,

é certamente um de seus filhos e portanto quando $L$ é confirmado, seu rótulo já estará determinado na variável indicada acima. Para ver isso, considere $L'$, o primeiro filho de $L$ a ser iniciado. Pela observação **ii.** esse loop não pode ser retirado da pilha na chamada a rotina em que um falso loop envolvendo um outro filho do $LeC$ — $L''$ — é detectado. Isso por que, nesse caso, $L''$ deveria ser iniciado antes de $L'$ o que significa, pela própria definição de $L'$, que ele teria de começar antes $L$. Não haveria, então como o $LeC$ vir a formar depois um loop, como $L$, que se inicia entre $L''$ e $L'$. Portanto $L'$ é retirado da pilha quando um loop verdadeiro do qual ele é filho, ou seja $L$, é encontrado. Assim, $L'$ é retirado da pilha na chamada que confirma que $L$ é um loop verdadeiro, e de novo pela sua definição tem de ser o último loop a ser removido da pilha nessa chamada.

A **Procedure Push_onto_Stack** simplesmente coloca no topo da pilha *Stack* um registro com dados a respeito do loop que foi encontrado constituído pelos seus vértices inicial e final, rótulo e a informação de se ele faz uma $\mu$-intersecção com o $LeC$ ou não. Esses dados possibilitarão encontrar o label do pai do loop em $LT$. As funções **Top_of_Stack** e **Pop_out_of_Stack**, podem ter implementações diversas mas sua finalidade dispensa explicações.

Falta apenas a função que efetivamente dá rótulo ao loop. Em vista do que já vimos ela pode ser implementada de uma forma extremamente compacta como a dada abaixo:

**Function Label_the_Loop_between(Initial_vertex, Final_vertex)**
{**If** (*Initial_vertex > Final_vertex_of_the_Last_loop_found*))
*Label = Label_the_Leaf_between*( *Initial_vertex, Final_vertex*);
**Else**
*Label = Label_of_the_parent_of_the_last_loop_removed_from_Stack*;
*Final_vertex_of_the_Last_loop_found = Final_vertex*;
**Return** *Label*;}

A condição (*Initial_vertex > Final_vertex_of_the_Last_loop_found*) indica que o loop encontrado é uma folha. Nesse caso a função *Label_the_Leaf_between* fará a sua rotulação empregando um dos três métodos descritos na seção 5.3, conforme ele seja um loop com antecessor, sem antecessor mas não $\mu$-equivalente a $PC_k$ ou a própria curva projetada. Caso contrário o label do loop será o dado por *Label_of_the_parent_of_the_last_loop_removed_from_Stack*, ou seja rótulo que é o obtido a partir do último filho do loop a ser retirado de *Stack*, considerando se a junção desse filho com o loop se dá ou não através de uma $\mu$-intersecção ou ele corta esse filho, que então,

seria fechado, em outro ponto. Afora a rotulação propriamente dita, temos de atualizar a variável *Final_vertex_of_the_Last_loop_found* o que nos permitirá detectar possíveis futuras folhas.

Completamos assim uma descrição detalhada dos procedimentos que tratam os loops formados, seja para validá-los ou não ou para lhes dar o rótulo correto. Na seção seguinte vamos apresentar, finalmente, o procedimento que gerencia todo o processo a ser efetuado quando uma célula é re-visitada por $PC_k$.

## 6.2 Gerenciamento do processo efetuado quando se re-visita uma célula.

```
Revisiting_the_cell_of(Parameter_Vertex)
If (Check_whether_the_Vertex_is_on_a_Repeated_Edge(Parameter_Vertex))
   Unset Previous_Cell_has_Three_Crossings;
Else If(Parameter_Vertex.old ¬ = nil)
        {Check_whether_the_Vertex_is_on_a_Repeated_Edge(Parameter_Vertex.old);
        Unset Previous_Cell_has_Three_Crossings;}
Else
     { Check_whether_the_Vertex_is_on_a_Non_Repeated_Edge(Current_Vertex);
     Set Previous_Cell_has_Three_Crossings;
     Next_Vertex = Generate_Next_Vertex_of_PC_from( Transformed_Curve, Current_Vertex_of_TC);
     If (¬(Last_Vertex[C(next_vertex)].stage = Current_stage))
        {If Previous_Vertex_and_Last_Vertex_of_the_Previous_Cell_are_on_the_same_Edge or
            Previous_Cell_has_Three_Crossings }
        Check_whether_the_Vertex_is_on_a_Non_Repeated_Edge(Next_Vertex);
     Unset Previous_Cell_has_Three_Crossings;
     Unset Previous_Vertex_and_Last_Vertex_of_the_Previous_Cell_are_on_the_same_Edge;}
```

A procedure *Revisiting_the_Cell* dada acima é responsável por fazer com que a aplicação dos procedimentos que verificam a possibilidade de formação de um loop, seja feita na ordem correta e com os parâmetros corretos. Seu único parâmetro é sempre o último vértice da célula re-visitada gerado antes do corrente. Deve-se observar que ela tem uma formulação mais elaborada do que em princípio se esperaria, pelo fato de que pode ser necessário aplicar os procedimentos citados mais de uma vez, numa mesma re-visita de $PC_k$ a uma célula. Isso acontece, por razões diferentes, nos dois casos abaixo:

**1)** quando uma célula — $C$ — que já é dupla é re-visitada outra vez. Nesse caso, certamente $PC_k$ volta a uma aresta — $e$ —já intersectada por ela e assim, um novo loop será formado se $PC_k$ ainda tiver um vértice válido em $e$. É claro que para processar esse loop é preciso saber qual é esse vértice. As quatro opções para esse vértice são acessíveis a partir de *Last_Vertex*[$C$]. Explicitamente, elas são, afora o próprio *Last_Vertex*[$C$] : *Last_Vertex*[$C$].*prev*, *Last_Vertex*[$C$].*old*, (*Last_Vertex*[$C$].*old*).*prev*.

Ocorre que a rotina *Check_whether_the_Vertex_is_on_a_Repeated_Edge*(abreviadamente: *CVRE*) pode identificar o vértice em questão apenas se ele for o que é passado a ela como parâmetro ou seu antecessor. Assim fazendo numa primeira chamada, esse parâmetro ser *Last_Vertex*[*C*], procuramos esse vértice entre os determinados por $PC_k$ em sua última passagem por *C* feita anteriormente. Deveríamos agora, em princípio, considerar duas possibilidades:

(a) se nenhuma das arestas cruzadas nessa passagem for a do vértice corrente, então uma nova chamada com *Last_Vertex*[*C*].*old* como parâmetro determinará se ele ou seu antecessor é o vértice procurado e se ele é válido;

(b) o vértice procurado foi determinado na primeira passagem, mas se revelou não válido e então, não haveria necessidade de se fazer a segunda chamada. Esse caso, entretanto, não é possível nas condições em que estamos trabalhando. Assim, não precisamos distinguir um caso do outro e se o resultado da primeira chamada de *CVRE* não for verdadeiro, ela é chamada de novo.

Essas considerações e a constatação óbvia, de que se $PC_k$ re-visita uma aresta da malha então, a condição expressa pela variável *Previous_Cell_has_Three_Crossings* não se verifica, justificam o bloco inicial de procedimentos de *Revisiting_the_Cell_of* constituído por:

**If** *(Check_whether_the_Vertex_is_on_a_Repeated_Edge*(*Parameter_Vertex*))
  **Unset** *Previous_Cell_has_Three_Crossings*;
**Else If**(*Parameter_Vertex.old* ¬ = *nill*)
  {*Check_whether_the_Vertex_is_on_a_Repeated_Edge*(*Parameter_Vertex.old*);
  **Unset** *Previous_Cell_has_Three_Crossings*;}

Deve-se observar ainda que a rotina *Check_whether_the_Vertex_is_on_a_non_Repeated_Edge* (abreviadamente: *CVNRE*) já parte do pressuposto que as arestas de *C* não são repetidas e, por isso, deve ser chamada só depois que isso foi constatado usando *CVRE*.

**2)** a célula *C*, que estamos considerando como a que está sendo correntemente visitada, é aquela indicada pela coordenada de célula do vértice corrente de $PC_k$. Portanto ela é a célula onde essa curva está entrando pelo vértice corrente. Em vista disso, a rotina CVNRE precisaria ser chamada mesmo quando a célula corrente não tiver sido visitada antes. Isso aconteceria, por exemplo, quando *Previous_Cell_has_Three_Crossings* fosse verdade. É que só depois da geração do vértice de entrada dessa célula e então constatando que ela não tinha sido atingida antes, é que poderíamos concluir que com

o acréscimo desse vértice, ou a célula anterior se havia tornado dupla ou um knot-loop tinha sido formado. Ambos esses casos requerem um processamento feito por *CVNRE*.

Situação idêntica ocorreria se o vértice anterior estivesse num gargalo. Nesse caso, apenas com a geração do último nó e verificando que por ele se estava entrando numa célula não visitada, é que poderíamos constatar que esse gargalo tinha acabado. Deveria, então, ser executado o procedimento específico dessa circunstância o que *CVNRE* também faz. Desse modo, para toda célula atingida por PCk, estivesse ela sendo re-visitada ou não, precisaríamos testar o valor de *Previous_Vertex_and_Last_Vertex_of_the_Previous_Cell_are_on_the_same_Edge* e de *Previous_Cell_has_Three_Crossings*. Como a imensa maioria das células é cruzada por PCk uma única vez, seria interessante podermos nos livrar desses testes, ainda que isso demande um acréscimo no procedimento que tem de ser efetuado quando uma célula é re-visitada. A alternativa que escolhemos para fazer isso foi a seguinte:

- quando se constata que a célula corrente contém 3 vértices pertencentes ao *LeC* — e, por isso, necessariamente em arestas diferentes — é claro que a variável booleana que expressa essa condição deve ser setada. Mas além disso, o próximo vértice de $PC_k$(*Next_Vertex*) é gerado. Pode-se, então verificar se a próxima célula a ser atingida (*C*(*Next_Vertex*)) ainda não foi visitada antes pela $PC_k$ atual. Explicitamente, isso é feito verificando se ela ainda não foi atingida por alguma curva projetada ou se já foi, se o ultimo estágio em que isso ocorreu é anterior ao atual — condições que são verificadas por (*Last_Vertex*[*C*(*Next_Vertex*)].*stage Last_Vertex*[*C*(*Next_Vertex*)].*stage* ¬ = *Current_stage*). Caso o resultado desse teste seja verdadeiro assim como as condições associadas às variáveis booleanas indicadas acima,essas variáveis são zeradas e *Check_whether_the_Vertex_is_on_a_Non_Repeated_Edge* é chamada de novo, agora com *Next_Vertex* como parâmetro, para efetuar ou o procedimento próprio de um fim de gargalo, fazer o split de um knot-loop ou simplesmente indicar que a célula anterior(*C*(*Current_Vertex*)) é dupla. Esclarecemos que essa nova chamada deve ser feita apenas se *C*(*Next_Vertex*) não tiver sido visitada antes. Senão, em casos como o da figura 6.15, deixaríamos de verificar que $PC_k$ já cruzou antes a aresta de *Next_Vertex*. Esse raciocínio é o que embasa o bloco

{ *Check_whether_the_Vertex_is_on_a_Non_Repeated_Edge*(*Current_Vertex*);
**Set** *Previous_Cell_has_Three_Crossings*;

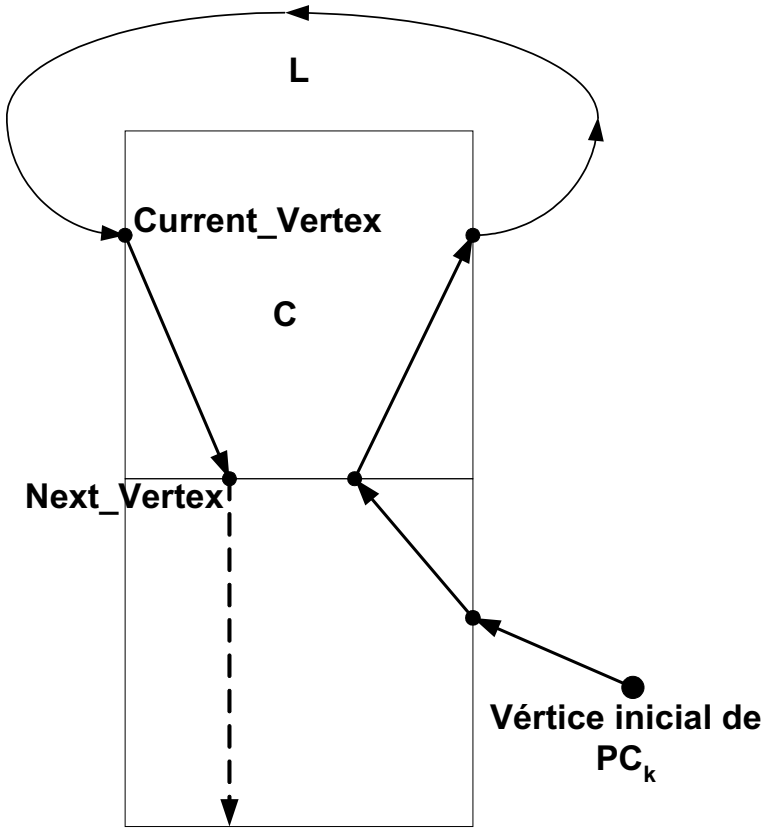


Figura 6.15: Se *CVNRC* for chamada, sucessivamente, para *Current_Vertex* e *Next_Vertex*, a célula *C* seria, erradamente, feita dupla e o loop *L* não seria encontrado.

*Next_Vertex = Generate_Next_Vertex_of_PC_from( Transformed_Curve, Current_Vertex_of_TC*);
**If** (*Last_Vertex*[*C*(*next_vertex*)].*stage = Current_stage*)
{**If** *Previous_Vertex_and_Last_Vertex_of_the_Previous_Cell_are_on_the_same_Edge* **or**
*Previous_Cell_has_Three_Crossings* }
*Check_whether_the_Vertex_is_on_a_Non_Repeated_Edge*(*Next_Vertex*);
**Unset** *Previous_Cell_has_Three_Crossings*;
**Unset** *Previous_Vertex_and_Last_Vertex_of_the_Previous_Cell_are_on_the_same_Edge*;}

que trata do caso em que o *LeC* volta a uma célula sem re-cruzar nenhuma de suas arestas.

Descrevemos neste capítulo todo um conjunto de procedimentos para detectar loops, fazê-los $\mu$-regulares e rotulá-los corretamente. Esse conjunto é muito mais elaborado do que um simples procedimento para traçar uma curva de nível de uma função cujo valor é conhecido nos vértices da malha, como aquele que dá o traçado final de uma T-snake.
O procedimento dado aqui tem alguns complicadores como a determinação das $\mu$-intersecções contidas na curva projetada. Observamos que isso é feito com a única finalidade de possibilitar que a rotulação de um loop possa ser feita corretamente, a partir dos labels de seus filhos na $LT(PC_k)$. O uso de técnicas para rotular os loops que dispen-

sem essa determinação, poderá fazer o código de algumas rotinas ficar bem mais curto. Mas, em última análise, a maior complexidade do processo descrito neste capítulo é o preço a pagar por se estar num único percurso da snake, gerando as curvas transformada e projetada e ainda detectando e rotulando os loops dessa última. Entretanto, o acréscimo em termos de custo computacional que o emprego desse processo pode acarretar é pouco significativo considerando-se que, conforme veremos no próximo capítulo, durante toda a evolução da snake apenas um percentual ínfimo de células é visitado mais de uma vez pela curva projetada do estágio corrente. Em vista disso esse acréscimo é largamente compensado pela economia feita no tratamento que tem que ser aplicado a cada snaxel que se movimenta por se usar o procedimento simplíssimo para fazer o mapeamento $\Gamma_k$ $\mu$-equivalente a um adequado, que é descrito no capítulo 4.

# Capítulo 7

# Resultados

## 7.1 Estatísticas e Exemplos de Segmentação

Comparando-se o desempenho computacional do enfoque descrito neste trabalho com a dos outros modelos de snakes topologicamente adaptativas, isto poderia nos levar a conclusões injustas, uma vez que não se teve acesso às implementações feitas por outros proponentes desses modelos e não se pode assegurar que nosso método particular é comparável aos demais.

Preferiu-se argumentar em favor do enfoque apresentado aqui, utilizando-se as estatísticas obtidas numa série de testes que foram feitas para validar o enfoque. Para uma melhor avaliação do esforço global requerido por essa metodologia, os snaxels foram classificados em função do número de operações feitas, no momento em que eles foram gerados, somente para manter o controle da topologia da snake. Especialmente eles foram distribuídos nas seguintes classes:

**A)** Snaxels nos quais o método detectou que uma célula foi totalmente seccionada;

**B)** Snaxels na classe *A* nos quais a necessidade de uma ação corretiva foi observada;

**C)** Snaxels nos quais o procedimento “Visited_Cell”(revisita uma célula) foi chamado.

**D)** Número total de Snaxels gerados.

O número de snaxels em cada uma dessas classes foi computado para imagens dos cinco grupos seguintes, cada um com suas dificuldades particulares:

**I)** imagens sintéticas contendo vários objetos com uma forma não convexa(figura: 7.1);

**II)** imagens com ruído que têm um contorno não muito bem definido(figura: 7.7);

**III)** imagens com muitos objetos a serem detectados onde um processo de refinamento da malha foi necessário para separar alguns desses objetos(figura: 7.5);

**IV)** imagens de células onde o "background" é consideravelmente texturizado ou contém estruturas que não são o objetivo da segmentação da snake(figura: 7.4);

Os resultados obtidos são apresentados na tabela 7.1 junto com o número total de snaxels gerados para cada grupo de imagens:

| | Imagens I | Imagens II | Imagens III | Imagens IV |
|---|---|---|---|---|
| Grupo A | 142.605 | 937.837 | 360.687 | 425.618 |
| Grupo B | 1.806 | 95.357 | 22.453 | 44.866 |
| Grupo C | 6.359 | 8.540 | 12.541 | 3.808 |
| Total D | 557.282 | 2.847.313 | 1.183.100 | 1.212.631 |

Tabela 7.1: The number of snaxels computed.

Pode-se observar na tabela 7.1 que o número dos snaxels mais custosos, isto é, aqueles onde o processamento mais complexo de "Visited_cell", foi disparado, é extremamente pequeno em relação ao número total de snaxels, nunca atingindo a 1%. Além disso, os snaxels no grupo *A* são mais do que 25% do total, o que significa que para os outros, no máximo 75% dos snaxels, a única ação especificamente importante para o controle da topologia da snake é testar se $i\_cur = i\_prev$. Também, para os snaxels em $A - B$, que são no máximo 1.2% dos snaxels em *A*, a ação tomada é limitada a quatro testes. A partir das linhas dos grupos *B*, *D* e *E* pode-se perceber que o número de situações que determinam quaisquer trabalho extra além desses testes, é realmente insignificante. Mesmo os snaxels criados desnecessariamente, do grupo *C* são extremamente pequenos. Os números da tabela 7.2 se referem às iterações de snakes problemáticas ao ínvés de snaxels problemáticos. Esta indica para as mesmas classes dos exemplos da tabela 7.1, as seguintes estatísticas:

**A)** o número de iterações onde um split ocorreu;

**B)** o número de iterações onde o Loop-Tree *PC* tinha mais do que três nós;

**C)** o número de snakes dessas árvores que têm folhas fechadas;

| | Imagens I | Imagens II | Imagens III | Imagens IV |
|---|---|---|---|---|
| Grupo A | 11 | 95 | 155 | 242 |
| Grupo B | 0 | 0 | 0 | 0 |
| Grupo C | 1 | 9 | 7 | 40 |
| Total D | 10.531 | 18.594 | 33.811 | 4.443 |

Tabela 7.2: Snakes Problemáticas.

**D)** o número total de snakes geradas nos exemplos da classe.

Exceto nos exemplos, especialmente construídos para esse propósito, Loop-Trees com não mais de cinco nós não foram encontradas. Isto permite concluir que os resultados apresentados aqui relativos a rotulação de nós de Loop-Trees não triviais dificilmente podem ser empregados. Entretanto a força dessa teoria é que esta provê mecanismos a serem aplicados se uma situação mais complexa eventualmente apareça, o que requer pouco esforço em snaxels simples. Isto é como uma garantia de baixo prêmio e como uma garantia, geralmente o melhor é não utilizá-la. O enfoque se torna eficiente, não para tratar situações raramente elaboradas de uma forma eficiente, mas para ser capaz de tratá-las se somente algumas precauções são tomadas em uma situação padrão. Com relação a essas precauções o fato de que este enfoque lida com loops fechados faz a diferença. Outros esquemas têm que prover contextos onde eles não são necessários. Essas estatísticas validam a estratégia “lazy” empregada.

| | Images I | Images II | Images III | Images IV |
|---|---|---|---|---|
| Tempo(seg) | 11 | 32 | 23 | 27 |

Tabela 7.3: Tempo de Execução.

Na tabela 7.3 pode-se ver ver o tempo de execução para todo o processamento necessário para a geração das Loop-Snakes. Estes tempos foram obtidos utilizando-se um computador pessoal com as seguintes características:

- processador AMD Athlon (TM) XP 1800+ ;
- memória RAM 512 MB ;
- sistema operacional Red Hat 9 .

Os tempos obtidos no sistema operacional Windows XP correspondem ao dobro dos tempos do Red Hat 9.

## 7.2 Resultados de Segmentação

Pode-se ver o resultado deste trabalho aplicado a diferentes tipos de imagens, conforme classificadas na tabela 7.1.

Nas imagens 7.1, 7.2 se procura segmentar as formas geométricas de cor escura em um fundo branco. Já na imagem 7.3 não foi feito nenhum filtro, procurando-se segmentar o máximo possível dentro de um determinado threshold, ao contrário do que acontece na imagem 7.4 onde se procura filtrar as snakes pelo número de pontos que a curva contém a fim de se pegar as duas maiores células. Na imagem 7.5 com um determinado threshold se procura segmentar todas as células, sendo estas bem parecidas em forma e tamanho. Mas isto não foi suficiente para se obter uma boa segmentação visto que várias células ficaram agrupadas. Desta forma foi feito uma subdivisão da malha a fim de se segmentar estas células que ficaram agrupadas (imagem 7.6). Já a imagem 7.7 é muito ruidosa. Nesta dentro de um determinado threshold procurou-se ver tudo o que seria segmentado. Aplicando-se um threshold mais específico e limitando-se o número de pontos da snake consegui-se segmentar o grupo de células mais escuras no centro da imagem 7.8 e procurou-se da mesma forma segmentar o contorno mais exterior da maior célula 7.9.

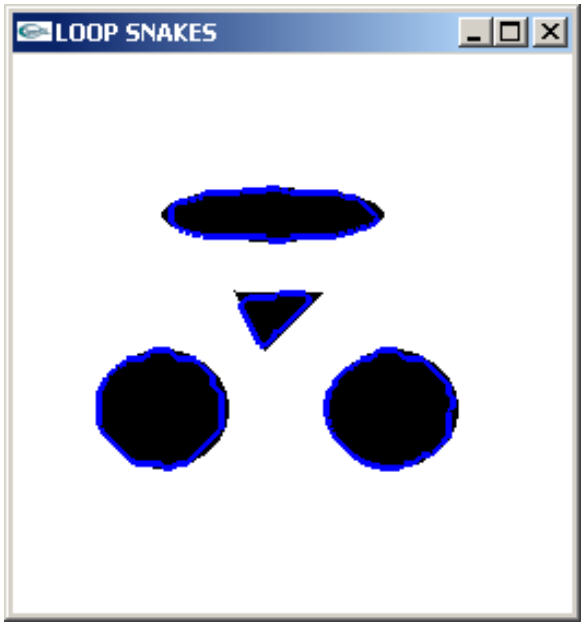


Figura 7.1: Imagem sintética.

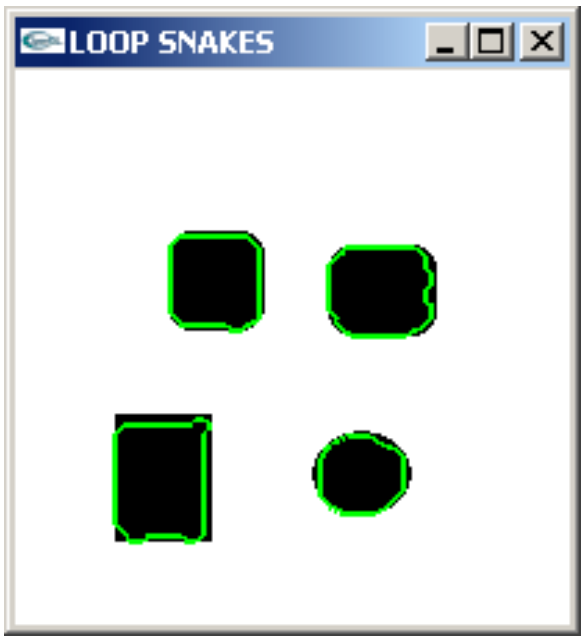


Figura 7.2: Imagem sintética.

## 7.3 Interface Utilizada

A interface permite a visualização da imagem a ser segmentada bem como as curvas que foram geradas e sua respectiva animação. O usuário também pode modificar vários ítens de controle a fim de obter um melhor resultado. A interface da figura 7.10 é relativa ao programa que gera a snake inicial e o tipo inicial em função das dimensões da imagem e do pixel. Na figura 7.11 pode-se ver a interface do programa principal, que permitem a visualização da imagem que está sendo segmentado, e controles para os tipos de ponto, linha, grid, propriedades das curvas, variáveis de controle e controles para animação das snakes e exibição das segmentações com e sem refinamento. Já na figura 7.12 pode-se ver os quatro primeiro detalhes da interface principal.

**a** no item de menu *S*howfigure controla-se a exibição da figura;

**b** nos ítens de menu *L*ine Properties e Point Properties controla-se se será feita a exibição das snakes com linhas ou pontos, bem como suas respectivas dimensões e cores. Na opção *G*rid Properties pode-se exibir ou não o grid bem como determinar sua dimensão e cor;

**c** já nos ítens de menu *C*urves Properties e Function In() Properties existem várias variáveis de controle a de se determinar como a curva irá evoluir e que regiões são de interesse para segmentação;

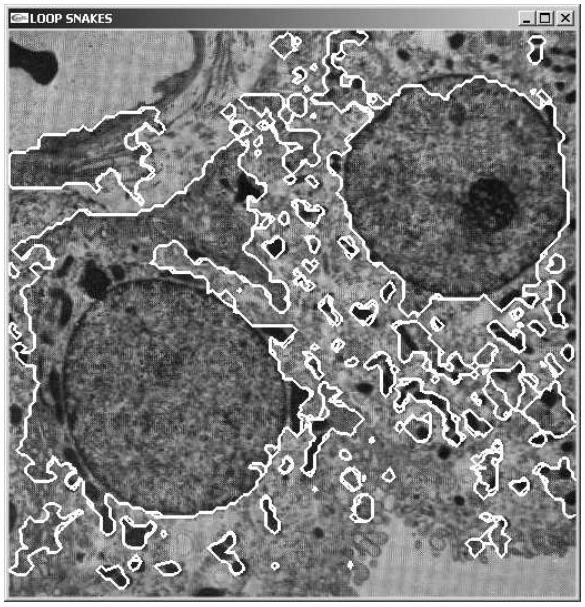


Figura 7.3: Imagens de células onde o “background” é consideravelmente texturizado.

**d** no item de menu *S*tatistics tem-se várias estatísticas que serão apresentadas nas tabelas;

**e** no último item de menu *I*mage Controls 1 tem-se o controle das animações das curvas, podendo-se se exibir e animar todas as curvas encontradas bem como exibir aquelas que não sofreram refinamento ou tiveram refinamentos a fim de se obter melhores resultados.

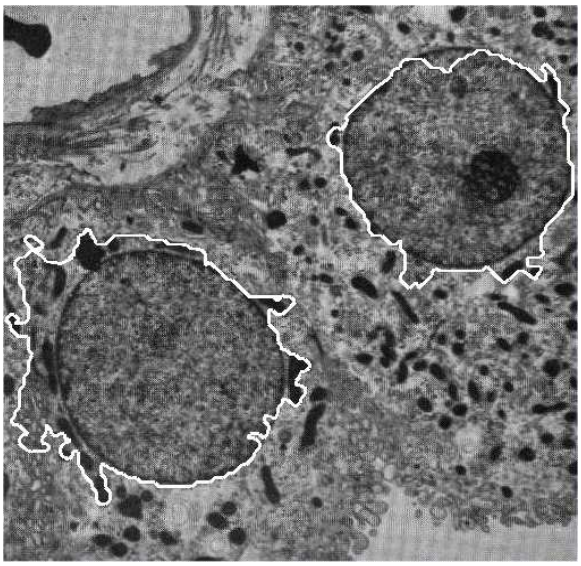

Figura 7.4: Mesma imagem 7.3 com filtro por tamanho da célula.

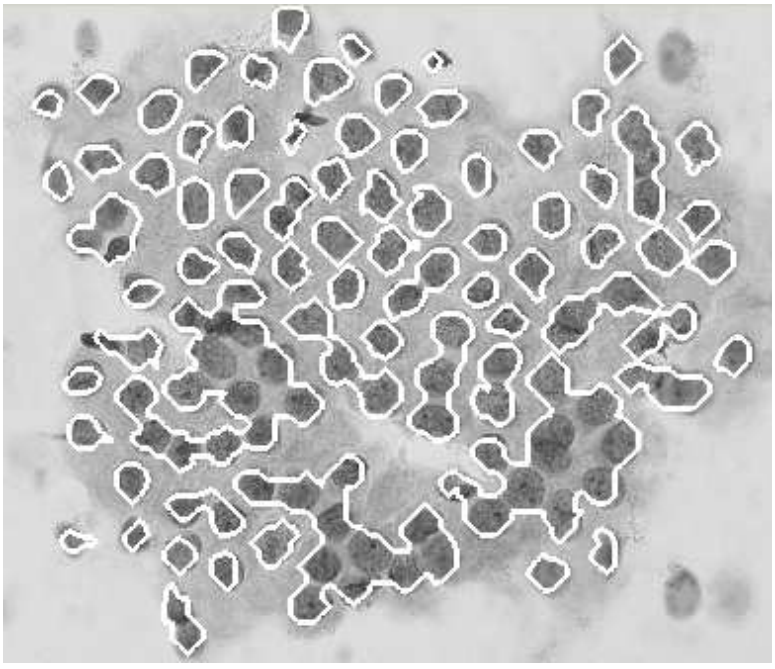

Figura 7.5: Imagens com muitos objetos a serem detectados.

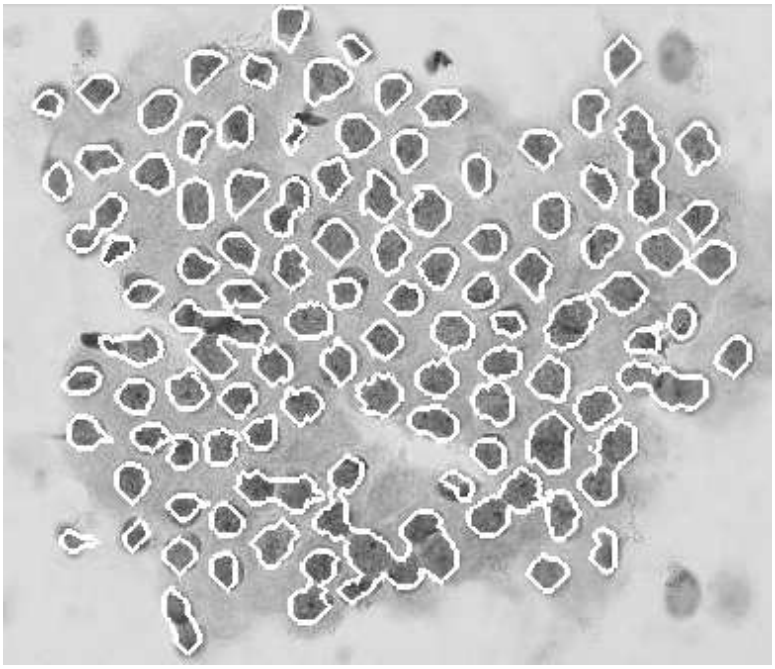

Figura 7.6: Mesma imagem 7.5, onde um processo de refinamento da malha foi necessário para separar alguns desses objetos.

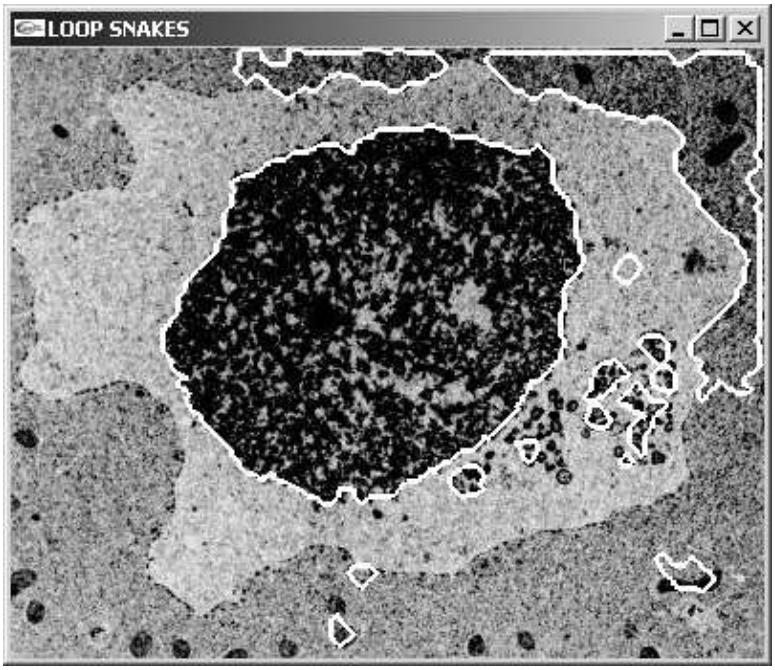


Figura 7.7: Imagem com ruído.

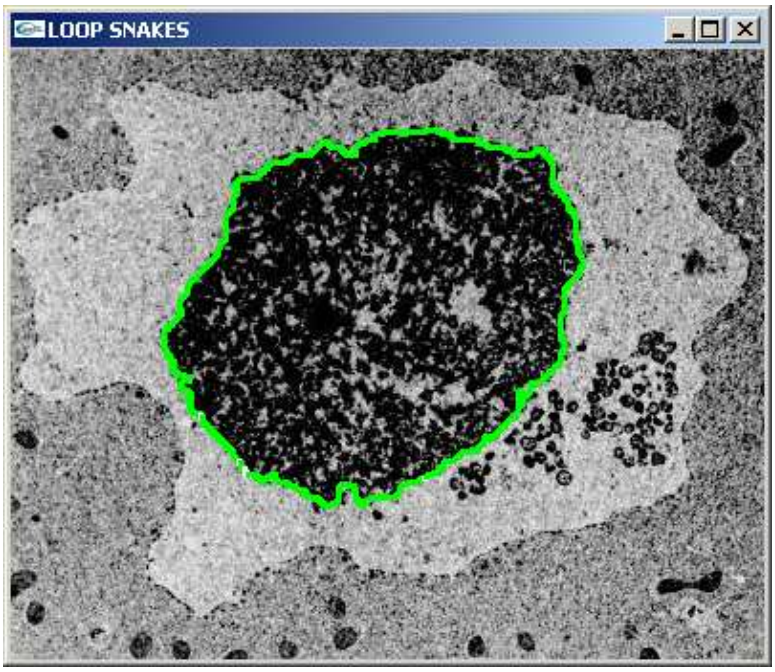


Figura 7.8: Mesma imagem 7.7 com ruído, com filtro na região mais interna.

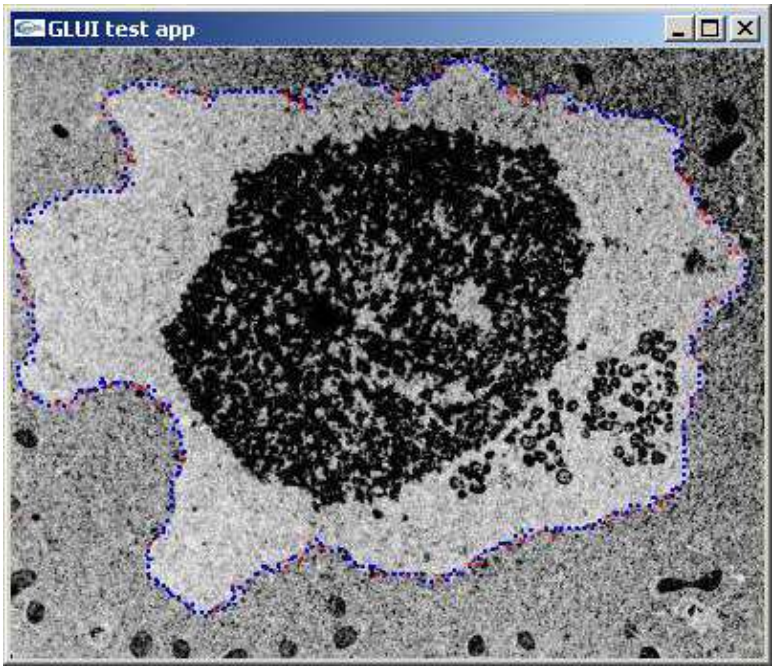


Figura 7.9: Mesma imagem 7.7 com ruído, com filtro na região mais externa.

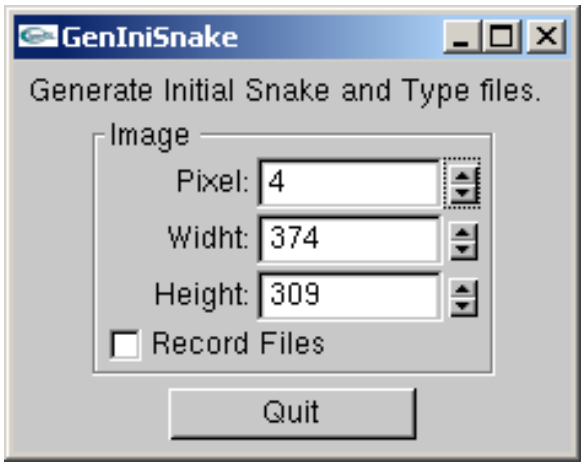


Figura 7.10: Gera snake e tipo inicial.

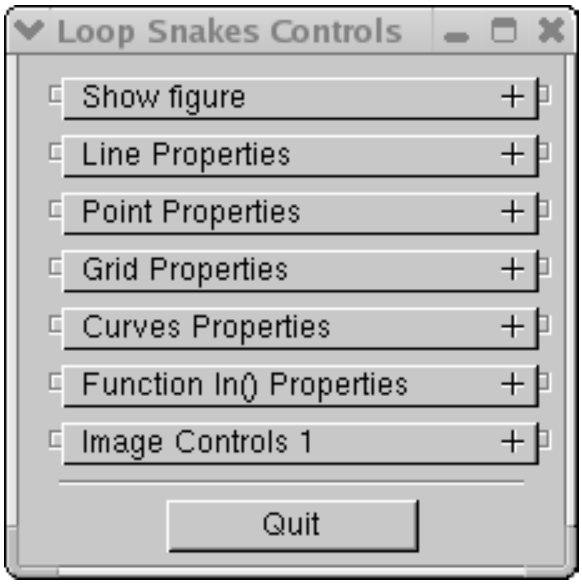


Figura 7.11: Interface do programa principal.

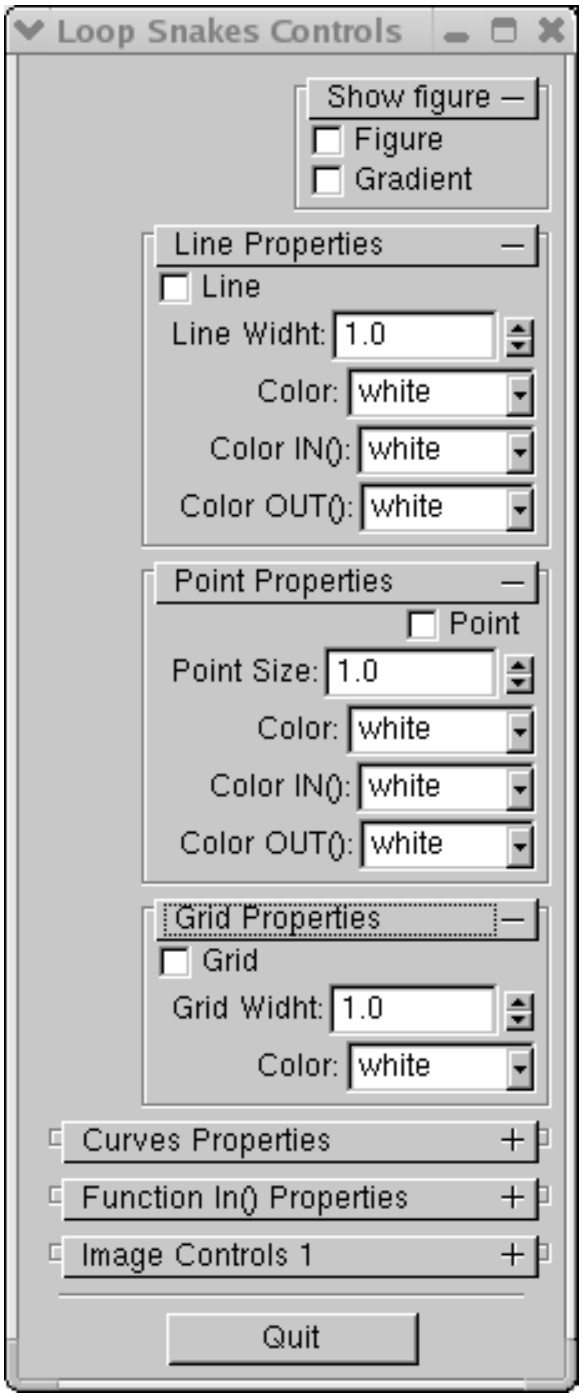


Figura 7.12: interface do programa principal: os quatro primeiro detalhes.

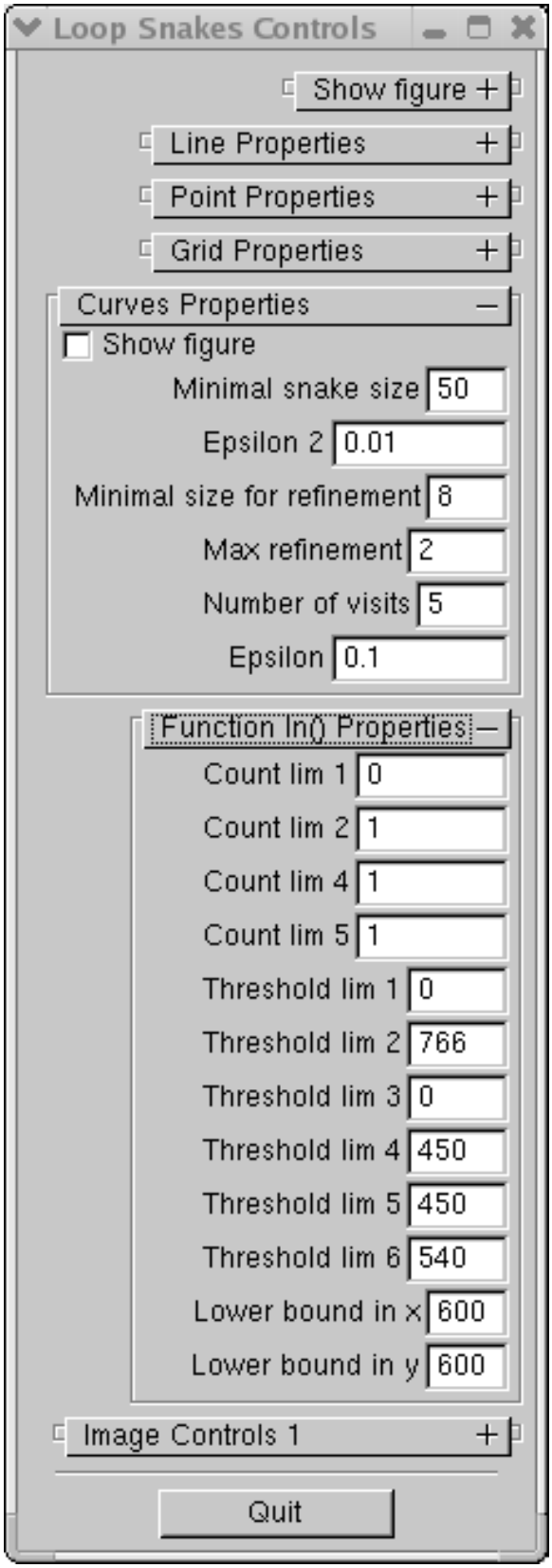


Figura 7.13: interface do programa principal: o quinto e o sexto detalhe.

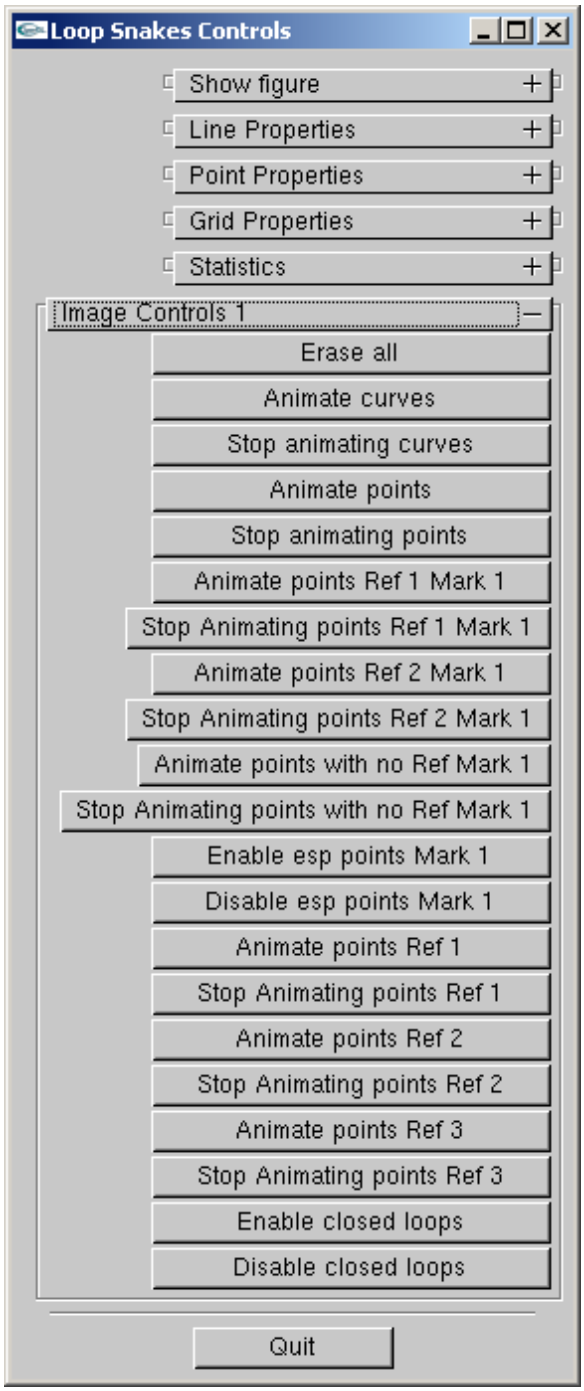


Figura 7.14: interface do programa principal com o último detalhe

# Capítulo 8

# Conclusões

## 8.1 Conclusões e Trabalhos Futuros

Uma teoria específica foi desenvolvida nos capítulos 4, 5 e 6, para suportar a tentativa de melhorar o processo de controlar a topologia da T-snake, o que é apresentado neste trabalho. Baseado nessa teoria foi possível criar uma metodologia que satisfaz a cinco propriedades desejadas indicadas no final do capítulo 3 na secção 3.6 e também tem um ganho computacional em relação aos métodos existentes o que se pode concluir a partir das tabelas da secção 7.1.

Com relação a trabalhos futuros, a possibilidade de expandir os resultados obtidos aqui para T-surfaces gerando imagens 3D pode ser considerado no seguinte contexto: Se $\Gamma_k$ é adequado o trabalho de queimar os vértices da malha pode ser reduzido a se queimar os out-vertex de cada vértice de $PC_k$ gerado. Esta propriedade pode ser mantida para T-surfaces e pode dar origem a um método para desenvolvê-los mais rápido do que os existentes.

Existem também variantes em 2D do método ainda a serem exploradas. A mais promissora nem mesmo requer que a curva projetada seja determinada.

# Referências Bibliográficas


[1] E. L. Algower and K. Georg. Introduction to numerical continuation methods. *Springer-Verlag, Heidelberg*, 1990.

[2] Taciana Amorim Vanderlei Wheydislane Yáskara Ramos dos Santos Ana Carolina, Luciano do Rêgo Galvão. Uma ferramenta de extração de bordas utilizando t-snakes. *Computer Graphics*, 21(4):163–169, July 2003.

[3] Ricardo Farias Cláudio Esperança Gilson Giraldi Antônio Oliveira, Saulo Ribeiro. Loop snakes: Snakes with enhanced topology control. *SIBGRAPI, Simpósio Brasileiro de Computação Gráfica e Processamento de Imagem*, pages 364–371, 2004.

[4] Bischoff and L. Kobbeit. Snakes with topology control. *The Visual Computer*, 2003.

[5] A. Black and A. Yuille. Active vision. *MIT Press*, 1993.

[6] F.; T. Caselles, V. ; Catte and F. Dibos. A geometric model for active contours. *Numerishe Mathematik 66*, 1993.

[7] Laurent D. Cohen. On active contour models and balloons. *CVGIP: Image Underst.*, 53(2):211–218, 1991.

[8] L.D. Cohen and I. Cohen. Finite element methods for active contour models and ballons for 2d and 3d images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(11):1131–1147, 1993.

[9] A. Witkin D. Terzopoulos and M. Kass. Constraints on deformable models: Recovering 3d shape and nonrigid motion. *Artificial Intelligence*, 36(1):91–123, 1988.

[10] Ingrid Carlbom D. Terzopoulos and Kristen M. Harris. Computer-assisted registration, segmentation, and 3d reconstruction from images of neuronal tissue sections. *IEEE Transactions on Medical Imaging*, 13(2):351–362, 1994.

[11] E. Segawa Gang Xu and S. Tsuji. Robust active contours with insensitive parameters. *Pattern Recognition*, 27(7):879–884, 1994.

[12] Edilberto; OLIVEIRA Antonio A. F. GIRALDI, Gilson A.; STRAUSS. A boundary extraction approach based on multi-resolution methods and the t-snakes framework. *In Proc. of International Symposium on Computer Graphics, Image Processing and Vision, SIBGRAPI'2000*, 2000.

[13] Gilson A. Giraldi. Tese de doutorado: T-snakes duais e inicializaÇÃo de modelos deformÁveis. ufrj (universidade federal do rio de janeiro), coppe/sistemas. 2000.

[14] S. R. Gunn and M. S. Nixon. A robust snake implementation: A dual active contour. *IEEE Trans. Pattern Anal. Mach. Intell*, 19(1):63–68, 1997.

[15] Steve R. Gunn. Dual active contour models for image feature extraction. 1996.

[16] Ip and Shen. An affine-invariant active contour model (ai-snake) for model-based segmentation. *Image and Vision Computing*, 16(2):135–146, 1998.

[17] J. Ivins, J. ; Porrill. Statistical snakes: Active region models. *In Proc. 5th British Machine Vision Conf.(BMVC'94)*, pages 377 – 386, 1994.

[18] V. A. Kovalevsky. Elements of algebraic topology. pages 141–161, 1984.

[19] F. Leitner and P. Cinquin. From splines and snakes to snakes splines. *IN C. Laugier, editor, Geometric Reasoning: From Perception to Action*, 708 of Lectures Notes in Computer Science:264–281, 1993.

[20] T. McInerney. Topologically adaptable deformable models for medical image analysis. *Ph.D. thesis, Department of Computer Science, University of Toronto*, 1995.

[21] T. McInerney and D. Terzopoulos. Topologically adaptable snakes. *Proc. Of the Fifth Int. Conf. On Computer Vision (ICCV'95), Cambridge, MA, USA*, pages 840–845, june 1995.

[22] Tim McInerney and Demetri Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4(2):73–91, 2000.

[23] Andrew Witkin Michael Kass and Demetri Terzepoulos. Snakes: Active contour models. 21(4):163–169, July 1988.

[24] S. Osher and J.A. Sethian. Fronts propagation with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1998.

[25] N. Rougon and F. Prêteux. Deformable markers: Mathematical morphology for active contour models control. *Proc. In Image Algebra and Morphological Image Processing II*, 1568:78 – 79, 1991.

[26] J.A. Sethian. Curvature and the evolution of fronts. *Commun. in Mathematical Physics*, 101:487–499, 1985.

[27] M. A. Snyder. On the mathematical foundations of smoothness constraints for the determination of optical flow and for surface reconstruction. *IEEE Trans. on Pattern Analusis and Mach. Intell.*, 13(11):1105–1114, 1991.

[28] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(4):413–424, 1986.

[29] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, 1988.

[30] Vincent Torre Tomaso Poggio and Christof Koch. Computational vision and regularization theory. *Nature*, 317(26):314–319, 1985.

[31] G. Wahba. Spline models for observational data. *SIAM, Philadelphia, PA*, 1990.