



MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-12831-PRE/8121

CAPÍTULO 2

ALGORITMOS GEOMÉTRICOS E RELACIONAMENTOS TOPOLÓGICOS

Clodoveu A. Davis Jr.
Gilberto Ribeiro de Queiroz

Bancos de dados geográficos

INPE
São José dos Campos
2005

2 Algoritmos geométricos e relacionamentos topológicos

Clodoveu A. Davis Jr.

Gilberto Ribeiro de Queiroz

2.1 Introdução

Este capítulo apresenta uma introdução às principais técnicas e algoritmos utilizados na implementação das diversas funções de um sistema de gerência de bancos de dados espaciais (SGDBE), em especial as operações sobre representações vetoriais (pontos, linhas e polígonos), que estão subjacentes a situações típicas, tais como:

- Seleção por apontamento, em que um usuário seleciona um determinado objeto através da interface gráfica;
- Determinação do relacionamento espacial entre dois objetos, tanto para consultas quanto para o estabelecimento de restrições de integridade espaciais no banco de dados;
- Criação de mapas de distância (buffer zones) e solução de problemas de proximidade;
- Sobreposição e aritmética de polígonos para operações de análise espacial.

Essas operações são alvo de estudo de uma área da Ciência da Computação conhecida como Geometria Computacional (Preparata e Shamos, 1985), que procura desenvolver e analisar algoritmos e estruturas de dados para resolver problemas geométricos diversos. Neste particular, tem um ponto importante de contato com a área de projeto e análise de algoritmos, uma vez que também procura caracterizar a dificuldade de problemas específicos, determinando a eficiência computacional dos algoritmos e usando técnicas de análise de complexidade assintótica (Knuth, 1973). Existe também uma

preocupação em desenvolver soluções para problemas clássicos de geometria, construindo estruturas mais apropriadas para a representação geométrica robusta no ambiente computacional, que tem limitações conhecidas quanto à precisão numérica e a capacidade de armazenamento de dados (Schneider, 1997).

2.2 Definições

Em um SGBDE, cada objeto vetorial é codificado usando um ou mais pares de coordenadas, o que permite determinar sua localização. Para entender melhor a maneira como os SGBDE tratam a informação vetorial, são relacionadas a seguir algumas definições fundamentais (Davis Jr., 1997). Como na maioria dos SGBDE, as definições consideram apenas duas dimensões.

- *Ponto*: um *ponto* é um par ordenado (x, y) de coordenadas espaciais.
- *Reta e segmento de reta*: Sejam p_1 e p_2 dois pontos distintos no plano. A combinação linear $\alpha \cdot p_1 + (1 - \alpha)p_2$, onde α é qualquer número real, é uma *reta* no plano. Quando $0 \leq \alpha \leq 1$, se tem um *segmento de reta* no plano, que tem p_1 e p_2 como *pontos extremos*.

A definição de reta e segmento é estritamente geométrica, e nos interessa uma definição mais aplicada. Assim, partimos para o conceito de linha poligonal, que é composta por uma seqüência de segmentos de reta. O mais comum, no entanto, é definir a linha poligonal através da seqüência dos pontos extremos de seus segmentos, ou seja, seus vértices.

- *Linha poligonal*: Sejam v_0, v_1, \dots, v_{n-1} n pontos no plano. Sejam $s_0 = v_0v_1, s_1 = v_1v_2, \dots, s_{n-2} = v_{n-2}v_{n-1}$ uma seqüência de $n - 1$ segmentos, conectando estes pontos. Estes segmentos formam uma *poligonal* L se, e somente se, (1) a interseção de segmentos consecutivos é apenas o ponto extremo compartilhado por eles (i.e., $s_i \cap s_{i+1} = v_{i+1}$), (2) segmentos não consecutivos não se interceptam (i.e., $s_i \cap s_j = \emptyset$ para todo i, j tais que $j \neq i + 1$), e (3) $v_0 \neq v_{n-1}$, ou seja, a poligonal não é fechada.

Observe, na definição da linha poligonal, a exclusão da possibilidade de auto-interseção. Os segmentos que compõem a poligonal só se tocam

nos vértices. Formalmente, poligonais que não obedecem a este critério são chamadas *poligonais complexas*. Estas poligonais podem criar dificuldades na definição da topologia e em operações como a criação de buffers (vide Seção 0).

- *Polígono*: Um polígono é a região do plano limitada por uma linha poligonal fechada.

A definição acima implica que, apenas invertendo a condição (3) da definição de linha poligonal, temos um polígono. Assim, também aqui não é permitida a interseção de segmentos fora dos vértices, e os polígonos onde isto ocorre são denominados *polígonos complexos*. Os mesmos comentários que foram feitos para poligonais valem para os polígonos. Observe-se também que o polígono divide o plano em duas regiões: o interior, que convencionalmente inclui a fronteira (a poligonal fechada) e o exterior.

Estas três entidades geométricas básicas podem ser definidas em uma linguagem de programação usando tipos abstratos de dados. Essa definição inclui tipos abstratos para retângulos e para segmentos, que são bastante úteis nos testes preliminares de alguns algoritmos geométricos. Não foi definido um tipo abstrato específico para polígonos, uma vez que correspondem a poligonais em que o primeiro e o último vértices coincidem. Para as poligonais, foi incluída no tipo uma variável *Retângulo*, para armazenar os limites do objeto segundo cada eixo¹.

estrutura Ponto

início

inteiro x;

inteiro y;

fim;

estrutura Segmento

início

Ponto p1;

Ponto p2;

¹ Este retângulo é usualmente denominado *retângulo envolvente mínimo* (REM), e é o menor retângulo com lados paralelos aos eixos que contém o objeto em questão.

```

fim;
estrutura Retângulo
início
    inteiro x1;
    inteiro y1;
    inteiro x2;
    inteiro y2;
fim;
estrutura Poligonal
início
    inteiro numPontos;
    Retângulo retânguloEnvolventeMínimo;
    Ponto[] vertice;
fim;

```

Programa 2.1 – Tipos abstratos de dados para Ponto, Retângulo e Poligonal.

2.3 Algoritmos básicos

Diversos problemas de geometria computacional utilizam resultados básicos de problemas mais simples em sua solução. Alguns destes resultados básicos vêm da análise geométrica do mais simples dos polígonos, e o único que sempre é plano: o triângulo.

2.3.1 Área de um triângulo

A determinação da área de um triângulo é uma das operações mais básicas empregadas por outros algoritmos. Ela é calculada como a metade da área de um paralelogramo (Figura 2.1) (Figueiredo e Carvalho, 1991). O produto vetorial dos vetores A e B determina a área (S) do paralelogramo com os lados A e B e, portanto, a área do triângulo ABC (que corresponde à metade do paralelogramo) pode ser computada a partir da seguinte equação:

$$S = \frac{1}{2} \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} = \frac{1}{2} (x_a y_b - y_a x_b + y_a x_c - x_a y_c + x_b y_c - y_b x_c) \quad (2.1)$$

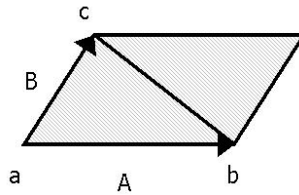


Figura 2.1 – Área do triângulo abc.

A Equação 2.1 fornece outra informação muito útil para os algoritmos de geometria computacional: a orientação dos três pontos que formam o triângulo. Caso a área seja negativa, os pontos a, b e c encontram-se no sentido horário; se positiva, os pontos encontram-se no sentido anti-horário; e se for zero, indica que os três pontos são colineares (estão alinhados).

2.3.2 Coordenadas baricêntricas

Para determinar se um determinado ponto pertence ou não a um triângulo, utiliza-se um método baseado em *coordenadas baricêntricas* (Figueiredo e Carvalho, 1991). De acordo com esse método, cada ponto p do plano pode ser escrito na forma $p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$, onde λ_1 , λ_2 e λ_3 são números reais e $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Os coeficientes λ_1 , λ_2 e λ_3 são denominados *coordenadas baricêntricas* de p em relação a p_1 , p_2 e p_3 .

Os valores de λ_1 , λ_2 e λ_3 podem ser obtidos usando a regra de Cramer, e expressos em termos de áreas de triângulos cujos vértices são p , p_1 , p_2 e p_3 . Temos, portanto:

$$\lambda_1 = \frac{S(pp_2p_3)}{S(p_1p_2p_3)}, \lambda_2 = \frac{S(p_1pp_3)}{S(p_1p_2p_3)} \text{ e } \lambda_3 = \frac{S(p_1p_2p)}{S(p_1p_2p_3)}$$

A análise do sinal das coordenadas baricêntricas indica a região do plano em que se encontra p , em relação ao triângulo $p_1p_2p_3$ (Figura 2.2). Observe-se que, para isso, as áreas devem ser orientadas, ou seja, com sinal.

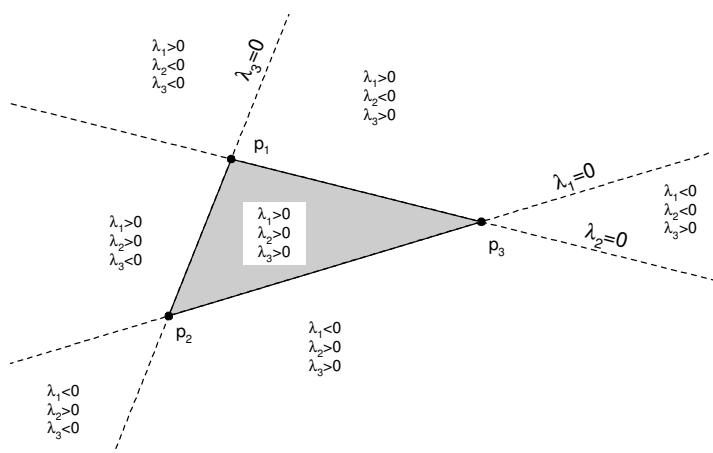


Figura 2.2 – Sinais das coordenadas baricêntricas.

2.3.3 Interseção de retângulos

Diversos algoritmos de geometria computacional se beneficiam de estratégias de implementação que procuram evitar, tanto quanto possível, o uso de procedimentos computacionalmente caros. Assim, os problemas são geralmente resolvidos em duas etapas: uma em que a representação geométrica dos objetos envolvidos é simplificada, e outra, executada apenas se necessário, em que a representação completa é empregada.

O uso do retângulo envolvente mínimo (REM) é uma dessas estratégias. O REM é o menor retângulo com lados paralelos aos eixos coordenados que contém a geometria do objeto. Por exemplo, antes de executar o algoritmo de determinação da interseção entre dois polígonos, comparamos seus REM diretamente. Caso os REM tenham alguma interseção, é *possível* que os polígonos se interceptem, e portanto o algoritmo completo precisa ser executado (Figura 2.3a); caso contrário, é certo que não existe interseção entre os objetos, e portanto a solução é o conjunto vazio (Figura 2.3b).

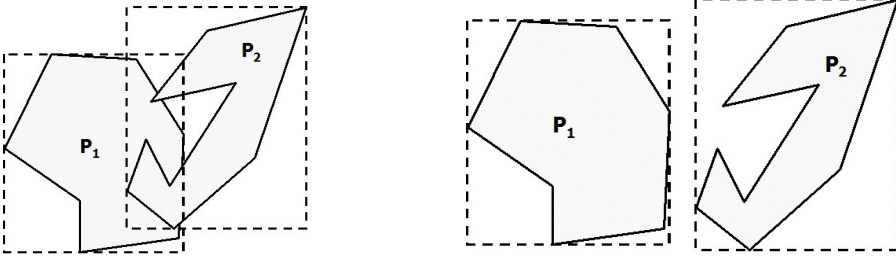


Figura 2.3 – (a) Interseção dos REMs

(b) REMs disjuntos.

O programa 2.2 ilustra este teste.

função interseçãoRetângulos(Ponto A, Ponto B, Ponto C, Ponto D): booleano

início

```
Ponto P, Ponto Q, Ponto Pl, Ponto Ql;
P.x = min(A.x, B.x);
P.y = min(A.y, B.y);
Q.x = max(A.x, B.x);
Q.y = max(A.y, B.y);
Pl.x = min(C.x, D.x);
Pl.y = min(C.y, D.y);
Ql.x = max(C.x, D.x);
Ql.y = max(C.y, D.y);
retorne ((Q.x >= Pl.x) e (Ql.x >= P.x) e
          (Q.y >= Pl.y) e (Ql.y >= P.y));
```

fim.

Programa 2.2 – Interseção de retângulos envolventes mínimos.

2.3.4 Interseção de dois segmentos de reta

Dados dois segmentos a e b, formados pelos pontos p_1p_2 e p_3p_4 (Figura 2.4), respectivamente, deseja-se verificar se eles se interceptam. A solução consiste em testar se os pontos p_1 e p_2 estão de lados opostos do segmento formado por p_3p_4 e também se p_3 e p_4 estão de lados opostos do segmento formado por p_1p_2 . Este problema se conecta com o problema da área de

triângulo, pois, determinar se p_3 está do lado oposto de p_4 em relação ao segmento p_1p_2 , consiste em avaliar o sinal da área dos triângulos formados por $p_1p_2p_3$ e $p_1p_2p_4$. Se os sinais forem contrários, significa que os pontos estão de lados opostos. Se o mesmo for verdadeiro para os triângulos $p_3p_4p_1$ e $p_3p_4p_2$, então, com certeza podemos afirmar que as retas que passam pelos segmentos se interceptam em algum ponto, embora não se possa afirmar ainda que os segmentos têm interseção.

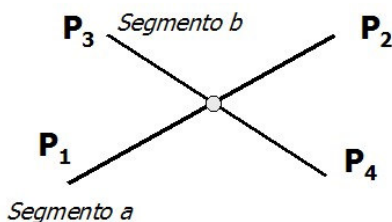


Figura 2.4 – Segmentos que se interceptam.

Em Saalfeld (1987) é discutida uma forma de determinar o ponto de interseção entre dois segmentos baseada na representação paramétrica dos segmentos². Dados dois segmentos formados pelos pontos p_1p_2 e p_3p_4 , respectivamente, e com $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $p_3 = (x_3, y_3)$ e $p_4 = (x_4, y_4)$, o ponto de interseção entre eles é dado por:

$$p_1 + u(p_2 - p_1) = p_3 + v(p_4 - p_3) \quad (2.2)$$

Esta igualdade dá origem a um sistema com duas equações e duas incógnitas (u e v):

$$\begin{cases} x_{intersec\ ao} = x_1 + u(x_2 - x_1) & \text{ou} & x_{intersec\ ao} = x_3 + v(x_4 - x_3) \\ y_{intersec\ ao} = y_1 + u(y_2 - y_1) & \text{ou} & y_{intersec\ ao} = y_3 + v(y_4 - y_3) \end{cases} \quad (2.3)$$

Desenvolvendo o sistema temos:

² A equação paramétrica para um segmento de coordenadas p_1 e p_2 é dada por: $p = p_1 + u(p_2 - p_1)$, onde se $0 < u < 1$ define um ponto localizado entre p_1 e p_2 .

$$u = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \quad (2.4)$$

$$v = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \quad (2.5)$$

Calculados os parâmetros u e v , podemos determinar o ponto de interseção:

$$\begin{cases} x_{\text{intersec\~{a}o}} = x_1 + u(x_2 - x_1) & \text{ou} & x_{\text{intersec\~{a}o}} = x_3 + v(x_4 - x_3) \\ y_{\text{intersec\~{a}o}} = y_1 + u(y_2 - y_1) & \text{ou} & y_{\text{intersec\~{a}o}} = y_3 + v(y_4 - y_3) \end{cases} \quad (2.6)$$

As expressões de u e v , respectivamente 2.4 e 2.5, possuem interpretações importantes. Os denominadores são os mesmos, e, portanto, numa implementação computacional eles deverão ser calculados uma única vez. Se o denominador for zero, as duas linhas são paralelas. Se além do denominador os numeradores de ambos os parâmetros também forem zero, então as duas linhas são coincidentes. Na verdade, as equações paramétricas aplicam-se a linhas e, portanto, só haverá interseção entre os dois segmentos em um ponto localizado sobre ambos, o que significa valores de u e v ambos no intervalo $[0,1]$.

2.3.5 Área de polígonos

A área de um polígono pode ser calculada em tempo linear com relação ao número de vértices, usando um somatório simples, baseado na soma de áreas de triângulos formados entre cada par de vértices consecutivos e a origem do sistema de coordenadas (O'Rourke, 1998):

$$A(P) = \frac{1}{2} \times \sum_{i=0}^{i=n-1} (x_i + x_{i+1}) \times (y_{i+1} - y_i) \quad (2.7)$$

Observe que na expressão acima (Equação 2.7), quando se tem $i = n - 1$, é necessário ter $x_n = x_0$ e $y_n = y_0$. Como no caso dos triângulos, o sinal da área obtida de acordo com esta fórmula indica a orientação dos vértices do polígono. Caso o resultado seja positivo, os vértices estão ordenados no sentido anti-horário, e caso seja negativo os vértices encontram-se no sentido horário.

2.3.6 Centróide de um polígono

O *centro de gravidade* ou *centro de massa*, mais conhecido como *centróide* de um polígono pode ser obtido a partir da sua divisão em triângulos, calculando em seguida a média ponderada dos centros de gravidade dos triângulos usando suas áreas como peso. O centro de gravidade de cada triângulo é simplesmente a média das coordenadas de seus vértices, ou seja, para um triângulo ABC:

$$x_G = \frac{x_A + x_B + x_C}{3} \text{ e } y_G = \frac{y_A + y_B + y_C}{3}$$

Embora este processo seja relativamente simples, pressupõe-se a implementação de um algoritmo de triangulação de polígonos. Os centróides dos triângulos são combinados usando um processo de média ponderada pela área. Assim, o centróide de um polígono formado por dois triângulos T_1 e T_2 , cujos centróides são, respectivamente, (x_{G1}, y_{G1}) e (x_{G2}, y_{G2}) é o ponto (x_G, y_G) , onde

$$x_G = \frac{x_{G1}S(T_1) + x_{G2}S(T_2)}{S(T_1) + S(T_2)} \quad y_G = \frac{y_{G1}S(T_1) + y_{G2}S(T_2)}{S(T_1) + S(T_2)}$$

e o centróide do polígono pode ser determinado de maneira incremental, adicionando um triângulo e seu centróide por vez e calculando as coordenadas do centróide do conjunto.

No entanto, existe uma solução mais simples e independente da triangulação, e que leva em conta triângulos com áreas positivas e negativas, como no cálculo da área do polígono. O mesmo processo de média ponderada pela área pode ser usado, considerando todos os triângulos formados entre um ponto fixo, por exemplo $(0, 0)$, e cada par de vértices sucessivos, (v_i, v_{i+1}) .

Assim, temos que:

$$\begin{aligned}
 A(P) &= \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - y_i x_{i+1}) \\
 x_C &= \frac{\sum_{i=0}^{n-1} (x_{i+1} + x_i) \times (x_i y_{i+1} - y_i x_{i+1})}{3A(P)} \\
 y_C &= \frac{\sum_{i=0}^{n-1} (y_{i+1} + y_i) \times (x_i y_{i+1} - y_i x_{i+1})}{3A(P)}
 \end{aligned} \tag{2.8}$$

O resultado pode ser facilmente implementado em um algoritmo com complexidade $O(n)$, que naturalmente pode fornecer ao mesmo tempo a área do polígono. Mas apesar da simplicidade do processo, não existe garantia de que o centróide será um ponto *pertencente* ao polígono. Caso seja necessário encontrar um ponto interno a um polígono simples dado, pode-se utilizar o seguinte processo, que busca precisamente identificar rapidamente uma diagonal do polígono (O'Rourke, 1998):

- identificar um vértice convexo v_i (por exemplo, o vértice inferior mais à direita)
- para cada outro vértice v_j do polígono verificar:
- se v_j estiver dentro do triângulo $v_{i-1}v_i v_{i+1}$, então calcular a distância $v_i v_j$
- armazenar v_j em q se esta distância for um novo mínimo
- ao final do processo, se algum ponto interior a $v_{i-1}v_i v_{i+1}$ for encontrado, então o ponto médio do segmento qv_i é interior ao polígono; senão, então o ponto médio do segmento $v_{i-1}v_{i+1}$ (ou mesmo o centróide do triângulo $v_{i-1}v_i v_{i+1}$) é interior ao polígono.

Outras definições de centróide consideram que o mesmo se situa “aproximadamente no centro do polígono” (Laurini e Thompson, 1992). O centróide pode ser determinado por diversos processos, como o centro do retângulo envolvente mínimo, o centro de um círculo inscrito ou circunscrito ao polígono. Uma forma freqüentemente usada para determinar um centróide consiste em simplesmente obter a média das coordenadas x e y dos vértices (Figura 2.5).

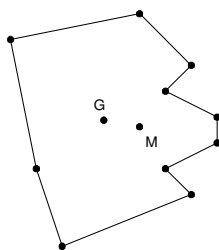


Figura 2.5 – Centróides calculados pela média (M) e como centro de gravidade (G).

2.4 Ponto em polígono

Uma das operações mais comuns em um SIG é determinar se um ponto está no interior de um polígono. Um dos algoritmos mais populares para solução deste problema é o teste do número de cruzamentos entre os segmentos que formam a fronteira do polígono e uma semi-reta (chamada de *raio*), que parte do ponto testado em qualquer direção (Haines, 1994) (Taylor, 1994). Se o número de cruzamentos for par, o ponto encontra-se fora do polígono; se for ímpar, encontra-se dentro. A Figura 2.6 ilustra a idéia desse teste. Conforme pode ser observado, o raio que parte do ponto *Q*, que está dentro do polígono, cruza os segmentos da fronteira um número ímpar de vezes (3 vezes).

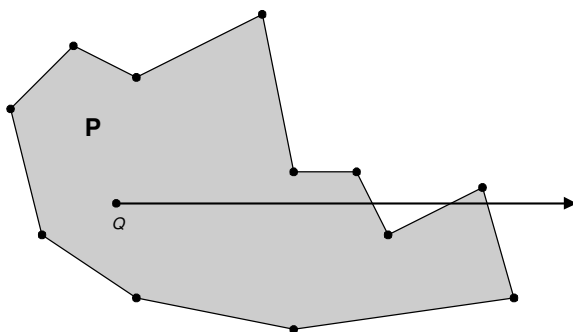


Figura 2.6 – Ponto em polígono.

Apesar da aparente simplicidade desse algoritmo, a sua implementação deve considerar alguns casos particulares (casos degenerados), como:

- a semi-reta passa por uma aresta do polígono (Figura 2.7a);
- a semi-reta passa por um vértice do polígono (Figura 2.7b);
- o ponto Q está sobre a fronteira do polígono (Figura 2.7c);
- o ponto Q coincide com um vértice do polígono (Figura 2.7d).

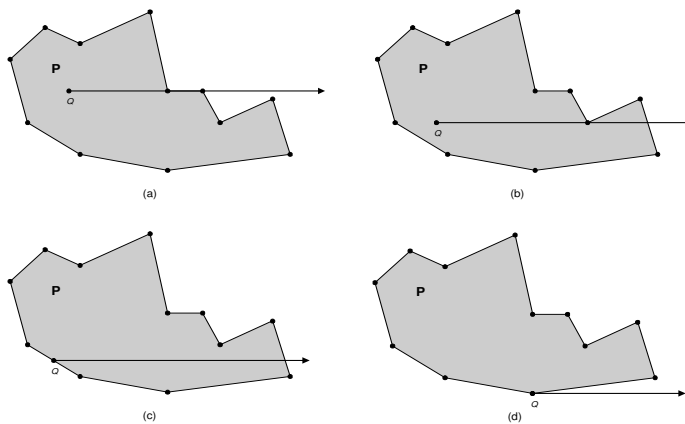


Figura 2.7 – Ponto em polígono: casos degenerados.

Para estes casos, a solução está em adotar um critério para a contagem de interseções de modo que:

- se a reta passa por um vértice, a interseção deve ser considerada apenas se for o vértice com maior ordenada do segmento, e ignorada caso contrário;
- se a reta passa por um segmento do contorno do polígono, nenhuma interseção deve ser considerada;
- se o ponto Q pertence a um segmento do contorno (exceto pontos extremos), considerar como uma interseção.

O caso em que Q coincide com um vértice pode ser tratado pelo primeiro critério. O terceiro critério faz com que todos os pontos da fronteira sejam considerados como pertencentes ao polígono.

Esse algoritmo possui complexidade linear em relação ao número de vértices do polígono. Para uma análise mais aprofundada do problema, o leitor é convidado a ler os trabalhos de Huang e Shih (1997) e Haines (1994).

2.5 Simplificação de poligonais

Muitas entidades do mundo real podem ser modeladas como linhas ou, mais genericamente, poligonais³. Essas entidades são freqüentes em bases de dados geográficas, onde correspondem tipicamente a cerca de 80% do volume de dados vetoriais (McMaster e Shea, 1992). Por isso, o problema de simplificação de linhas é particularmente importante, sendo estudado intensivamente desde os anos 60, quando ocorreram as primeiras experiências com o uso de instrumentos de transcrição de mapas para o computador, como a mesa digitalizadora.

No processo de digitalização de linhas, freqüentemente são introduzidos vértices em excesso, vértices que, se descartados, não provocariam uma alteração visual perceptível na poligonal. Assim, um primeiro objetivo para algoritmos de simplificação de linhas é “limpar” (significativamente, o verbo utilizado em inglês é *weed*, “capinar”) a poligonal de pontos claramente desnecessários, do ponto de vista de sua visualização (Weibel, 1995), mantendo a qualidade de sua aparência gráfica (Peucker, 1975) (Beard, 1991).

Outro objetivo é o de gerar uma nova versão da linha, mais adequada para a representação do mesmo fenômeno geográfico em outra escala, menor que a original. Neste caso, está sendo obtida uma *generalização* da linha (McMaster, 1992). Em uma extensão deste enfoque, existe o interesse em organizar os vértices da poligonal de tal forma que seja possível produzir, dinamicamente, versões generalizadas adequadas para uma escala definida no momento da visualização (van Oosterom, 1993)

³ Deste ponto em diante, será utilizado o termo *poligonal*, em lugar de simplesmente *linha*, para evitar confusão com a definição geométrica da linha reta (infinita).

(van Oosterom e Schenkelaars, 1995), conseguindo portanto gerar múltiplas representações geométricas para o mesmo fenômeno sem introduzir dados redundantes. No entanto, a utilização de métodos e algoritmos desenvolvidos originalmente apenas pensando na redução do número de vértices da linha podem não ser adequados para alcançar o objetivo de generalização (Laurini e Thompson, 1992), em geral por não conseguirem uma boa representação geométrica⁴, e portanto devem ser analisados cuidadosamente quanto a este aspecto.

Assim, o problema de simplificação de linhas consiste em obter uma representação *mais grosseira* (formada por *menos vértices*, e portanto *mais compacta*) de uma poligonal a partir de uma representação mais refinada, atendendo a alguma restrição de aproximação entre as duas representações. Essa restrição pode ser definida de várias maneiras (Li e Openshaw, 1992), mas é em geral alguma medida da proximidade geométrica entre as poligonais, tais como o máximo deslocamento perpendicular permitido (Figura 2.8a) ou o mínimo deslocamento angular permitido (Figura 2.8b). Na Figura 2.8a, o vértice 2 será mantido, uma vez que a distância entre ele e a reta que passa pelos vértices 1 e 3 é superior à permitida. Na Figura 2.8b, o vértice 3 será eliminado, uma vez que o ângulo $\hat{3}24$ é menor que o mínimo tolerável. Uma alternativa mais rara é a área entre as poligonais (Figura 2.8c), onde se estabelece um limite para ao deslocamento de área.

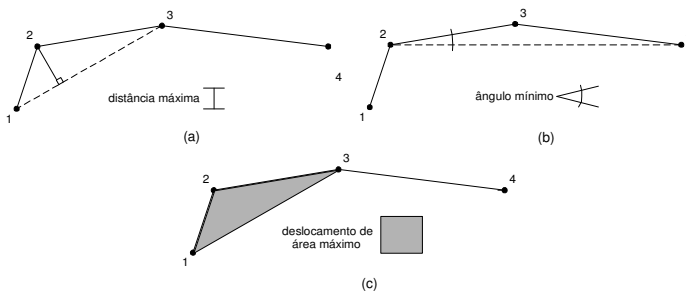


Figura 2.8 – Medidas de proximidade para simplificação de linhas.

⁴ Para auxiliar na manutenção do aspecto natural da poligonal, existem enfoques que integram algoritmos de simplificação com algoritmos de suavização.

Dentre todas as medidas possíveis, a mais utilizada é a distância perpendicular. O conceito de banda de tolerância, apoiado no cálculo de distâncias perpendiculares, é utilizado em grande parte dos algoritmos de simplificação que serão apresentados a seguir. Um problema eventualmente abordado na literatura é a escolha do parâmetro de tolerância (ϵ), e sua correlação com a escala da representação simplificada.

Um critério interessante para a determinação da tolerância é o que usa o tamanho do menor objeto visível em uma determinada escala (Li e Openshaw, 1992). Este tamanho pode ser dado em termos de uma distância medida no espaço de coordenadas do mapa plotado, ou seja, em milímetros do papel, independente da escala utilizada. Assim, é definida uma correspondência linear entre a escala e a tolerância adotada.

Grande parte dos algoritmos de simplificação de poligonais necessita realizar de maneira eficiente cálculos de distância entre um ponto dado e uma reta definida por outros dois pontos. A maneira mais interessante de calcular essa distância é utilizar o produto vetorial, conforme apresentado na Seção 2.3.1, para determinar a área S do triângulo formado por um ponto A e uma reta definida por outros dois (B e C), de acordo com a equação 2.1. Assim, a distância do ponto A à reta definida pelos pontos B e C pode ser calculada como:

$$d = \frac{|S|}{\text{dist}(B, C)}$$

onde $\text{dist}(B, C)$ é a distância euclidiana entre os pontos B e C .

Embora existam muitos algoritmos de simplificação de linhas, envolvendo variados critérios de aproximação e estratégias de processamento, um deles se destaca pela ampla aceitação. Foi proposto em 1973 por Douglas e Peucker (1973), e é reconhecidamente o melhor em termos de preservação das características da poligonal original (Marino, 1979) (McMaster, 1987).

Procedimento Douglas-Peucker(linha, numvert, tol)

Procedimento DP(a, f, tol)

início

se ((f - a) == 1) então retorne;

maxd = 0;

```

    maxp = 0;
    para i = a+1 até f-1 faça
    início
    d = distância(linha[i], linha[a], linha[f]);
    se d > maxd então
    início
        maxd = d;
        maxp = i;
    fim se;
    fim para;
    se maxd > tol então
    início
    vértice maxp selecionado;
        DP(a, maxp, tol);
        DP(maxp, f, tol);
    fim
    senão retorne;
    fim;
início
    vértice l selecionado;
    vértice numvert selecionado;
    DP(l, numvert, tol);
fim.

```

Programa 2.3 – Algoritmo Douglas-Peucker.

O algoritmo é recursivo, e a cada passo processa o intervalo de pontos contido entre um vértice inicial (chamado de *âncora*) e um vértice final (denominado *flutuante*). É estabelecido um *corredor* de largura igual ao dobro da tolerância, formando duas faixas paralelas ao segmento entre o âncora e o flutuante (Figura 2.9b), como no algoritmo de Lang. A seguir, são calculadas as distâncias de todos os pontos intermediários ao segmento básico, ou seja, contidos entre o âncora e o flutuante. Caso nenhuma das distâncias calculadas ultrapasse a tolerância, ou seja, nenhum vértice fica fora do corredor, então todos os vértices intermediários são descartados. Caso alguma distância seja maior que a tolerância, o vértice mais distante é preservado, e o algoritmo é reiniciado em duas partes: entre o âncora e o vértice mais distante (novo flutuante), e entre o vértice mais distante (novo âncora) e o flutuante. De acordo com este processo, os pontos tidos como críticos para a geometria da linha, a cada passo, são mantidos, enquanto os demais são descartados.

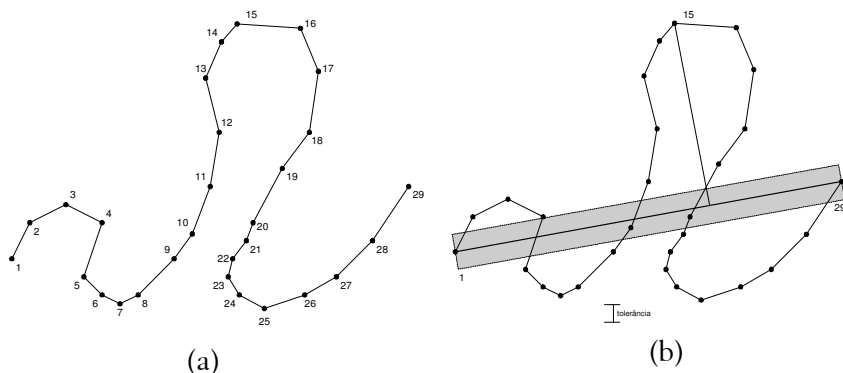


Figura 2.9 – Linha original, 29 vértices (a) e Douglas-Peucker, primeiro passo: seleção do vértice 15 (b).

Para a análise deste algoritmo e dos próximos será utilizada a poligonal da Figura 2.9a, com 29 vértices. As figuras seguintes ilustram melhor o comportamento do algoritmo Douglas-Peucker. Inicialmente, são calculadas as distâncias dos vértices 2 a 28 até a reta definida pelos vértices 1 e 29. O vértice mais distante nesta primeira iteração é o 15, a uma distância muito superior à tolerância (Figura 2.9b). Assim, o vértice 15 é selecionado e o procedimento é chamado recursivamente duas vezes, entre os vértices 1 e 15 e entre os vértices 15 e 29. Continuando pela primeira chamada, o vértice mais distante da reta entre 1 e 15 é o 9, também a uma distância superior à tolerância, e portanto é selecionado (Figura 2.10a). Duas novas chamadas recursivas são feitas, e agora estão empilhados os intervalos 1-9, 9-15 e 15-29. No intervalo 1-9, temos também que preservar o vértice 3, e portanto ficamos na pilha com os intervalos 1-3, 3-9, 9-15 e 15-29 (Figura 2.10b). Analisando agora o intervalo 1-3, verificamos que o vértice 2 pode ser dispensado (Figura 2.11a). Ao final, são preservados os vértices 1, 3, 4, 6, 9, 15, 16, 17, 22, 24, 27 e 29, ou seja, 41% do número original de vértices (Figura 2.11b).

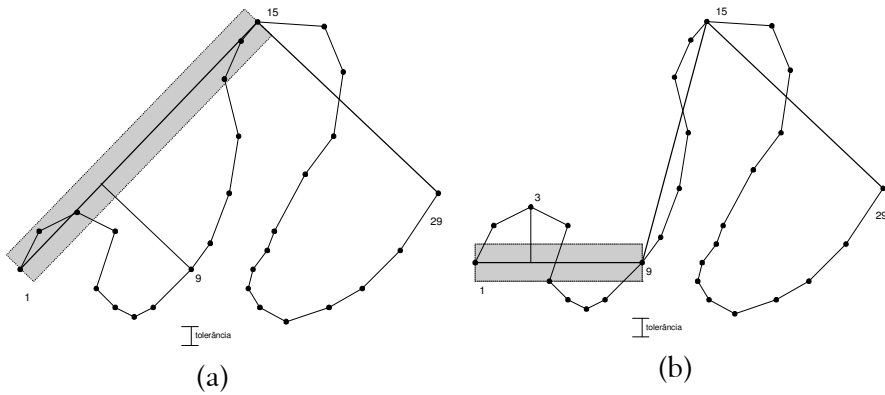


Figura 2.10 – Douglas-Peucker, segundo passo: seleção do vértice 9 (a) e Douglas-Peucker, terceiro passo: seleção do vértice 3 (b).

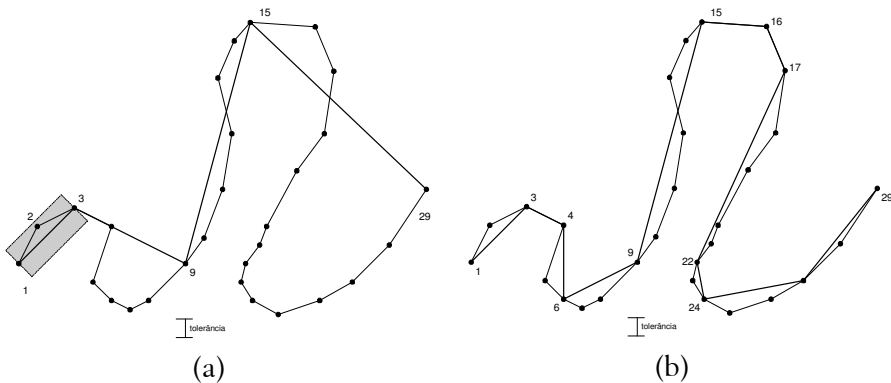


Figura 2.11 – Douglas-Peucker, passo 4: eliminação do vértice 2 (a) e Douglas-Peucker, final (b).

O resultado deste algoritmo é aclamado pela literatura como sendo o que mais respeita as características (ou, como no título do artigo de Douglas e Peucker, a “caricatura”) da linha cartográfica (Marino, 1979). Assim, este algoritmo veio a ser a escolha dos desenvolvedores de software comercial na implementação de funções de simplificação de linhas para processamento pós-digitalização (Li e Openshaw, 1992), ou seja, para limpeza de vértices desnecessários. O uso do algoritmo

Douglas-Peucker em generalização, no entanto, é comprometido pelo seu comportamento em situações de generalização mais radical, ou seja, com tolerâncias maiores. Conforme a situação, o algoritmo pode ser levado a escolher vértices que terminam por deixar a linha com uma aparência pouco natural, com tendência a apresentar “picos” (como no exemplo da Figura 2.11, entre os vértices 17, 24 e 29), com ângulos agudos e mudanças bruscas de direção.

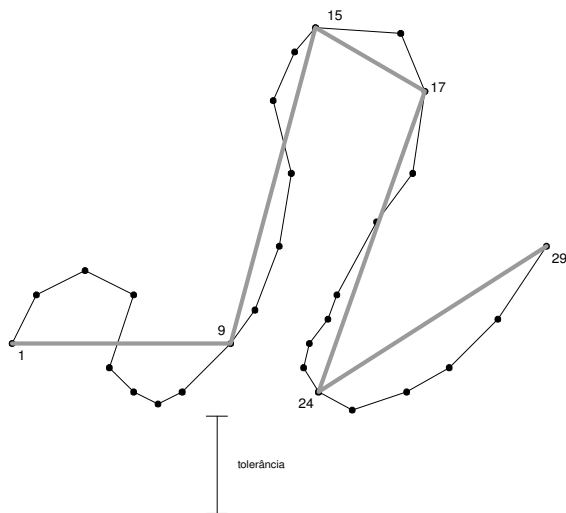


Figura 2.12 – Douglas-Peucker, simplificação radical.

Se a mesma linha da Figura 2.9a for processada novamente com uma tolerância, por exemplo, quatro vezes maior que a apresentada, seriam preservados apenas os vértices 1, 9, 15, 17, 24 e 29, todos pertencentes à solução anterior (Figura 2.12). Portanto, o algoritmo de Douglas-Peucker é hierárquico, pois os pontos são sempre selecionados na mesma ordem, e a tolerância serve para determinar até que ponto o processamento deve ser realizado.

Se a tolerância for igual a zero, todos os vértices serão eventualmente selecionados. O armazenamento das subdivisões nos permite representar

a hierarquia dos vértices em uma árvore binária (van Oosterom, 1993). Em cada nó desta árvore é representado um vértice selecionado, e é armazenado o valor da distância calculado por ocasião da seleção, que corresponde ao valor $maxd$ do algoritmo (Programa 2.3). Tendo sido estabelecido um valor de tolerância, basta caminhar na árvore em preordem para determinar quais vértices serão selecionados. Quando um nó interno contiver um valor de distância inferior à tolerância, o vértice correspondente e todos os descendentes poderão ser eliminados, não sendo necessário continuar com o caminharmento. Observe-se, no entanto, que a árvore binária pode ser bastante desbalanceada, e dificilmente será completa, o que virá a dificultar o seu armazenamento no banco de dados.

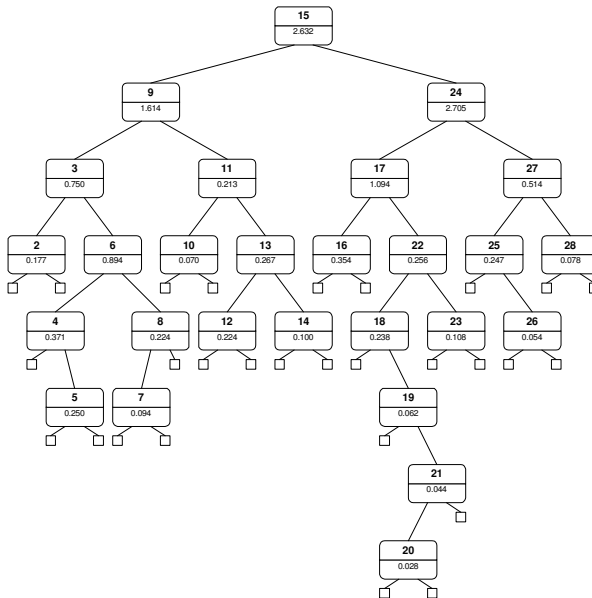


Figura 2.13 – Árvore binária formada a partir do exemplo da Figura 2.9 a.

2.6 Interseção de conjuntos de segmentos

O problema de se determinar os pontos de interseção entre um conjunto de segmentos é um dos mais importantes no caso de um SIG que trabalha com representação vetorial. Isso porque operações como união, interseção, diferença e as operações que avaliam o relacionamento topológico necessitam determinar esses pontos como uma das primeiras etapas de seus processamentos, sendo esta a de maior consumo de processamento. No que segue, são apresentados algoritmos para resolução deste problema.

2.6.1 Plane sweep

Shamos e Hoey (1976) apresentam um dos primeiros trabalhos discutindo o problema de interseção entre objetos com base na análise de complexidade. Eles fornecem um algoritmo para um problema similar que é determinar se, em um conjunto de n segmentos, há pelo menos um par que se intercepte.

A idéia para solução desse problema vem da análise de intervalos em uma dimensão. Considere-se que, em vez de n segmentos, tenha-se n intervalos entre números reais, do tipo $[x_L, x_R]$, onde $x_L \leq x_R$. Uma solução exaustiva seria analisar todos os n^2 pares de intervalos existentes, comparando-os sempre dois a dois, e interrompendo o processamento assim que a primeira interseção fosse detectada.

No entanto, uma maneira mais eficiente de resolver o problema é construir uma lista ordenada dos valores extremos dos intervalos, tomando o cuidado de identificá-los como sendo L ou R , de acordo com sua situação no intervalo. Assim, não haverá interseção alguma entre os intervalos se e somente se a lista ordenada contiver uma seqüência alternada de L s e R s: $L R L R \dots L R L R$. Em qualquer outra situação, pode-se afirmar que existe superposição entre algum par de intervalos. Esta solução tem complexidade computacional da ordem de $O(n \log n)$, uma vez que é dominada pela ordenação dos valores extremos.

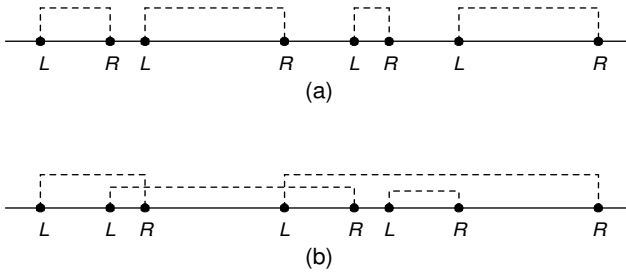


Figura 2.14 – Verificação de interseção em intervalos na reta.

Em duas dimensões, o problema torna-se um pouco mais complicado, já que não existe maneira de produzir uma ordenação adequada para segmentos no plano. A técnica empregada é clássica na geometria computacional, e é denominada de *varredura do plano* (*plane sweep*). Esta técnica faz uso de duas estruturas de dados básicas, uma para registrar a situação da linha de varredura (*sweep line status*), e a outra que registra eventos ocorridos durante a varredura (*event-point schedule*).

A idéia consiste em deslocar uma reta vertical pelo conjunto de segmentos, buscando identificar inversões na ordem em que esta reta encontra dois segmentos quaisquer. Para implementar esta idéia, é necessário definir uma nova relação de comparação, da seguinte forma: considere-se dois segmentos s_1 e s_2 no plano, sendo que s_1 não intercepta s_2 . Diz-se que s_1 é *comparável* a s_2 se, para alguma abscissa x , existe uma linha vertical que intercepta tanto s_1 quanto s_2 . Assim, diz-se que s_1 está *acima de* s_2 em x se, naquela abscissa, a interseção da reta com s_1 está acima da interseção da reta com s_2 . Esta relação é denotada como $s_1 >_x s_2$. Na Figura 2.15, temos as seguintes relações: $s_3 >_v s_2$; $s_4 >_v s_3$; $s_4 >_v s_2$; $s_4 >_w s_2$; $s_4 >_w s_3$; $s_2 >_w s_3$.

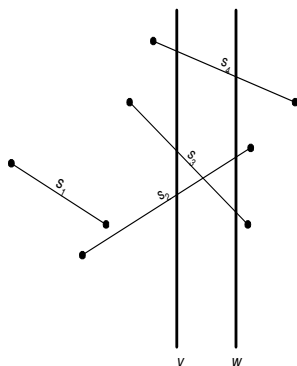


Figura 2.15 – Relação de ordenação entre segmentos.

Com esta relação é construída uma ordenação total dos segmentos, que muda à medida em que a linha é deslocada da esquerda para a direita. Nesse processo de varredura do plano, três coisas podem ocorrer:

- o ponto extremo à esquerda de um segmento é encontrado; o segmento é, portanto, inserido na ordenação;
- o ponto extremo à direita de um segmento é encontrado; o segmento é, portanto, retirado da ordenação;
- um ponto de interseção entre dois segmentos s_1 e s_2 foi encontrado; portanto, s_1 e s_2 trocam de posição na ordenação.

Observe-se que, para que s_1 e s_2 possam trocar de posição, é necessário que exista algum x para o qual s_1 e s_2 são consecutivos na ordenação. O algoritmo usa este fato, testando apenas elementos consecutivos, à medida em que novos eventos vão sendo detectados conforme descrito acima.

Portanto, é necessário operar duas estruturas de dados no processo. A primeira (*sweep line status*) é a responsável por manter a ordenação das interseções dos segmentos com a linha de varredura, e é usualmente implementada como um dicionário ou como uma árvore *red-black* (Cormen et al., 1990). As operações que o *sweep line status* deve suportar são inserção (insere, complexidade $O(\log n)$), exclusão (exclui, também $O(\log n)$), e duas funções para determinar qual segmento está imediatamente acima e imediatamente abaixo de um segmento dado na

ordenação (acima e abaixo, $O(1)$). A segunda estrutura de dados (*event-point schedule*) é responsável por manter a seqüência das abscissas que serão analisadas pela linha de varredura, e é implementada como uma fila de prioridades. Deve suportar as clássicas operações de inclusão (insere), retirada do elemento de mais alta prioridade (min) e uma função que testa a presença de um determinado elemento na estrutura (membro), todas com complexidade $O(\log n)$.

Inicialmente, as abscissas dos pontos extremos dos segmentos são ordenadas e inseridas no *event-point schedule*. Em seguida, as abscissas são retiradas a partir da menor, e são realizadas as seguintes operações:

- Se a abscissa corresponder a um ponto extremo à esquerda de algum segmento, inserir o segmento no *sweep line status*. Verificar se existem interseções entre este segmento e os segmentos que estão imediatamente acima e abaixo dele na linha de varredura. Caso exista interseção, a abscissa do ponto de interseção deve ser calculada e inserida no *event-point schedule*, caso já não pertença a ele.
- Se for um ponto extremo à direita, excluir o segmento do *sweep line status*. Verificar se existem interseções entre os segmentos que estão imediatamente acima e abaixo dele na linha de varredura. Caso exista interseção (que estará necessariamente à direita do ponto extremo), a abscissa do ponto de interseção deve ser calculada e inserida no *event-point schedule*, caso já não pertença a ele.
- Se for um ponto de interseção entre dois segmentos, trocar a posição destes segmentos no *sweep line status*. Informar a existência de um ponto de interseção e suas coordenadas.

O algoritmo possui complexidade sub-ótima, $O(n \log n + k \log n)$, onde k é o número de interseções. Um dos motivos para que ele não atinja o limite inferior de $n \log n + k$ é que os pontos de interseção são reportados na ordem x , que é a ordem na qual eles são inseridos na fila de eventos. A complexidade desse algoritmo depende não só do número de segmentos de entrada, mas também do número de interseções reportadas. Esse algoritmo pertence a uma classe conhecida como algoritmos sensíveis à saída, sendo apresentado originalmente por Bentley e Ottmann (1979).

2.6.2 Algoritmos de interseção por partição do espaço

Andrews et al. (1994) e Andrews e Snoeyink (1995) apresentam uma comparação entre métodos advindos da Geometria Computacional, e métodos desenvolvidos pela comunidade SIG (métodos pragmáticos) para resolver esse problema. Os algoritmos testados por eles foram agrupados em duas categorias: algoritmos por partição espacial e algoritmos por ordenação espacial.

Nos algoritmos da primeira classe, o espaço é subdividido em regiões, e os segmentos são atribuídos às regiões interceptadas por cada um deles. As interseções são computadas entre os segmentos de cada região, normalmente empregando um algoritmo de força bruta. A idéia é a aplicação de heurísticas que realizem filtros, diminuindo o número de segmentos a serem testados.

Os algoritmos agrupados na segunda classe são os baseados em estratégias de Geometria Computacional, onde a preocupação é com o desenvolvimento de algoritmos onde a análise de complexidade de pior caso possua uma boa complexidade. O algoritmo do *plane sweep* pode ser classificado nesta categoria.

Os testes realizados no trabalho deles mostram que embora os algoritmos da primeira classe não garantam uma eficiência no pior caso como os da Geometria Computacional, eles acabam tirando proveito da característica dos dados de um SIG: segmentos curtos, espaçados, com poucas interseções por segmento e uniformemente distribuídos no plano. Dessa forma, eles acabam sendo mais eficientes (velozes) do que os da segunda classe.

Outro trabalho que realiza testes semelhantes é apresentado por Pullar (1990), onde é mostrado que uma técnica baseada no *Fixed Grid*, também pertencente à categoria dos algoritmos por partição, é bastante competitiva em relação aos algoritmos baseados no *plane sweep*, apesar da complexidade de pior caso ser bem maior, $O(n^2)$.

Nos trabalhos de Akman et al. (1989), Franklin et al. (1988) e Franklin et al. (1989) é apresentado um algoritmo baseado no *fixed grid*. Dados dois conjuntos de segmentos, um vermelho e outro azul, os segmentos da primeira linha são cobertos por uma grade retangular fixa (*fixed grid*). Cada segmento vermelho é associado às células da grade por

onde ele passa. Os pontos de interseção são determinados procurando para cada segmento azul a lista de células por onde ele passa e então utilizando um algoritmo de força bruta esses pontos são determinados.

Um dos pontos-chaves desse algoritmo é a determinação da resolução da grade. Ela pode ser determinada a partir da média do comprimento dos segmentos. Franklin et al. (1988) mostram que utilizando-se de parâmetros estatísticos, como a média do comprimento dos segmentos, a grade se adapta bem aos dados de entrada.

2.7 União, interseção e diferença de polígonos

Operações sobre polígonos são de fundamental importância em SIG. Através da detecção e processamento da união, interseção e diferença de polígonos, diversos tipos de operações, conhecidas como em conjunto como *polygon overlay*, são viabilizadas. São operações fundamentais para análise espacial, usadas em situações em que é necessário combinar ou comparar dados colocados em camadas distintas. Por exemplo, considere-se uma consulta como “identificar fazendas em que mais de 30% da área é de latossolo roxo”. Para executar esta análise, é necessário combinar uma camada de objetos poligonais (os limites de propriedades rurais) com outra (o mapa de tipos de solo), para obter uma nova camada, de cujo conteúdo podem ser selecionados diretamente os objetos que atendem ao critério de análise colocado.

Algumas vezes, o *polygon overlay* é definido como uma operação topológica, ou seja, que é executada sobre dados organizados em uma estrutura de dados topológica. As funções de processamento de polígonos que serão descritas a seguir são utilizadas em sistemas não topológicos, ou em situações em que o processamento é feito de maneira isolada, como na criação e uso de *buffers* (vide Seção 0).

Para realizar operações sobre polígonos, é interessante aplicar um passo preliminar de detecção rápida da possibilidade de interseção entre os polígonos. Assim, se não for possível que dois polígonos P e Q tenham interseção, então podemos concluir diretamente que $P \cup Q = \{P, Q\}$, $P \cap Q = \emptyset$, $P - Q = P$ e $Q - P = Q$. Uma maneira simples de testar se dois polígonos têm ou não interseção é usar inicialmente o teste de interseção dos retângulos envolventes mínimos (Seção 0).

No caso geral, operações de união, interseção ou diferença entre dois polígonos simples podem gerar diversos polígonos como resultado. Mais ainda, os polígonos resultantes poderão conter buracos. A Figura 2.16 contém exemplos de produção de múltiplos polígonos e de polígonos com buracos em operações de interseção, união e diferença.

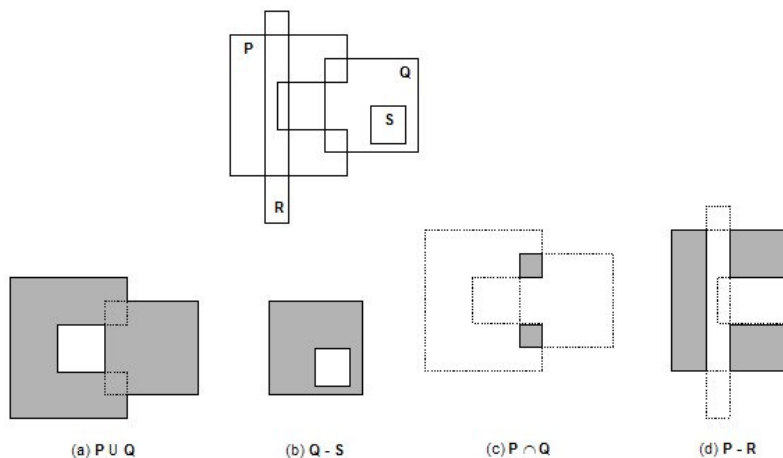


Figura 2.16 – Operações sobre polígonos produzindo buracos e múltiplos polígonos.

Apresentaremos aqui um método proposto por Margalit e Knott (1989). Esse algoritmo é sensível à orientação dos polígonos, e exige que os vértices de ilhas sejam codificados em um sentido (por exemplo, anti-horário) e os vértices de buracos sejam dispostos no sentido inverso (horário). Isto coincide com a convenção usada para calcular a área de polígonos, conforme apresentado na Seção 2.3.5.

Tabela 2.1 – Orientação dos polígonos de acordo com a operação

Polígonos		Operações			
P	Q	$P \cap Q$	$P \cup Q$	$P - Q$	$Q - P$

ilha	ilha	<i>manter</i>	<i>manter</i>	<i>inverter</i>	<i>inverter</i>
ilha	buraco	<i>inverter</i>	<i>inverter</i>	<i>manter</i>	<i>manter</i>
buraco	ilha	<i>inverter</i>	<i>inverter</i>	<i>manter</i>	<i>manter</i>
buraco	buraco	<i>manter</i>	<i>manter</i>	<i>inverter</i>	<i>inverter</i>

O algoritmo tem seis passos, que serão descritos a seguir.

1. *Normalizar a orientação* dos polígonos de entrada P e Q , e inverter a orientação de Q dependendo do tipo de operação e da natureza (ilha ou buraco) dos dois polígonos de entrada, de acordo com a Tabela 2.1.
2. *Classificar os vértices*, verificando se cada um está *dentro*, *fora* ou *na fronteira* do outro polígono, usando o teste de ponto em polígono (Seção 2.4). Inserir os vértices assim classificados em duas listas circulares, PL e QL , onde aparecerão em seqüência, de modo a definir as arestas por adjacência.
3. *Encontrar as interseções* entre arestas dos dois polígonos, usando o teste de interseção de n segmentos (Seção 0). Inserir os pontos de interseção na posição apropriada em PL e QL , classificando-os como *na fronteira*. A partir deste ponto, teremos um conjunto de *fragmentos de arestas* em lugar das arestas originais. É necessário cuidar do caso especial de interseção ao longo de uma aresta comum, ou parte dela. Neste caso, ambos os pontos extremos da aresta devem ser classificados como *na fronteira* e inseridos nas listas.
4. *Classificar os fragmentos de arestas* (definidos pelos pares de vértices) formados em PL e QL com relação ao outro polígono, entre *interior*, *exterior* ou *na fronteira*. Não é necessário realizar novamente o teste de ponto em polígono. Uma aresta pode ser considerada *interior* ao outro polígono caso pelo menos um de seus vértices esteja classificado como *dentro*. Da mesma forma, uma aresta pode ser classificada como *exterior* ao outro polígono caso pelo menos um de seus vértices esteja classificado como *fora*. Se ambos os vértices estiverem classificados como *na fronteira*, então é necessário verificar a situação de um ponto interno ao segmento (por exemplo, seu ponto médio). Se este ponto

estiver fora do outro polígono, então a aresta é classificada como *exterior*. Se o ponto estiver dentro do outro polígono, então a aresta é classificada como *interior*. Se o ponto estiver na fronteira, a aresta é classificada como *fronteira*.

Arestas na fronteira constituem um caso degenerado, que requer tratamento especial. Se existe um fragmento de aresta na fronteira de P , então necessariamente existe também um na fronteira de Q . Estes fragmentos podem estar orientados na mesma direção ou em direções opostas. A implementação pode decidir o que fazer nestes casos, ou seja, se interseções com dimensão de segmento ou de ponto serão ou não retornadas. Se as interseções como segmento forem retornadas, serão formadas por um ciclo de duas arestas sobrepostas, cada uma em uma direção. Interseção em um ponto será retornada como um ciclo de duas arestas, cada uma em uma direção, ligando dois vértices sobrepostos. Desta forma preserva-se a topologia do resultado (sempre cadeia fechada de segmentos), mas em SIG é mais interessante detectar estes casos e retornar objetos da dimensão adequada (no caso, ponto)⁵.

5. *Selecionar e organizar as arestas* para formar os polígonos de resultado. Este processo de seleção é baseado na combinação das duas listas em uma, denominada RL , usando apenas as arestas que interessam para a operação, conforme definido na Tabela 2.2.
6. *Construir os polígonos de resultado*, selecionando uma aresta e , com base em seu ponto final, procurar em RL sua continuação, até fechar o polígono. Repetir o processo, eliminando de RL a cada passo as arestas utilizadas, até que RL fique vazia.

Os polígonos resultantes manterão a orientação adotada para ilhas e buracos.

⁵ Para uma análise mais completa, inclusive com as combinações de hipóteses nos casos de ilhas e buracos, vide (Margalit e Knott, 1989).

Tabela 2.2 – Tipos de arestas para seleção de acordo com o tipo de operação e os tipos de polígonos de entrada

Polígonos		Operações							
		$P \cap Q$		$P \cup Q$		$P - Q$		$Q - P$	
P	Q	P	Q	P	Q	P	Q	P	Q
ilha	buraco	interior	interior	exterior	exterior	exterior	interior	interior	exterior
ilha	buraco	exterior	interior	interior	exterior	interior	interior	exterior	exterior
buraco	ilha	interior	exterior	exterior	interior	exterior	exterior	interior	interior
buraco	buraco	exterior	exterior	interior	interior	interior	exterior	exterior	interior

2.8 Mapas de distância (buffer zones)

Outra operação importante para um SIG é a construção de mapas de distância ou *buffer zones*, que são áreas construídas ao redor de objetos mantendo uma certa distância. A Figura 2.17 ilustra a idéia dessas operações para pontos, linhas e polígonos respectivamente.

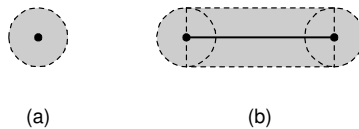


Figura 2.17 – *Buffers* elementares ao redor de ponto (a) e segmento (b).

A determinação do *buffer* ao redor de um ponto é feita de forma direta, como uma circunferência de raio d (Figura 2.17a). O *buffer* ao redor de uma linha é formada pela união de *buffers* elementares (Figura 2.17b) definidos para cada segmento da linha. Esses *buffers* elementares são formados a partir de semicircunferências traçadas nas extremidades dos segmentos (uma em cada extremidade). Utilizando o algoritmo de

união (Seção 2.7) podemos combinar esses *buffers* até formar o resultado final da linha (Figura 2.18a).

O *buffer* de polígonos (Figura 2.18b) é semelhante ao de linha, com a diferença de que é possível gerar *buffers* negativos (voltados para o interior do polígono).

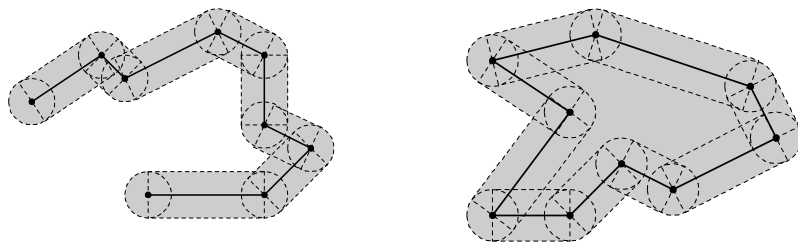


Figura 2.18 – *Buffer* ao redor de linha (a) e polígono (b).

2.9 Relacionamentos topológicos

A grande importância da caracterização dos relacionamentos topológicos entre estruturas vetoriais é poder atribuir um contexto semântico aos algoritmos geométricos. Para especificá-los, inicialmente definiremos as geometrias vetoriais como elementos do \mathcal{R}^2 , considerado como espaço topológico. Assim, um *ponto* é simplesmente um elemento de \mathcal{R}^2 . Uma *linha* L é um conjunto de pontos conectados. Uma *ilha* ou *linha circular* é uma linha em que o ponto inicial é igual ao ponto final. A fronteira de L , denotada por δL , é o conjunto dos pontos inicial e final, caso L não seja uma ilha, ou o conjunto vazio, em caso contrário. O interior de L , denotado por L^o , é composto pelos demais pontos. Uma *região* A é um conjunto de pontos com um interior conectado, denotado por A^o , uma fronteira conectada, denotada por δA , e um único exterior conectado, denotado por A^- . Assim, as regiões consideradas não têm “buracos”.

Os relacionamentos topológicos podem ser definidos com base em um modelo, chamado *matriz de 4-interseções* (ver a Figura 2.19), que considera oito relações topológicas binárias, representando a interseção

entre a fronteira e o interior de duas geometrias (Egenhofer e Franzosa, 1995).

Para definir relacionamentos topológicos entre geometrias com estruturas mais complexas, como regiões com ilhas e separações, é necessário estender a matriz de 4-Interseções para também considerar o exterior de uma geometria (Egenhofer e Herring, 1991). O novo modelo, chamado de *matriz de 9-Interseções* (ver Figura 2.20), considera então o resultado da interseção entre as fronteiras, interiores e exteriores de duas geometrias. Maiores detalhes sobre relações topológicas entre regiões com ilhas podem ser encontrado em (Egenhofer et al., 1994).

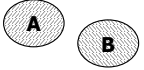
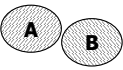
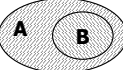
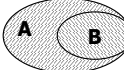

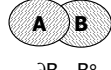
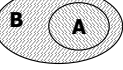

 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \end{matrix}$ <p>disjoint</p>	 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} -\emptyset & \emptyset \\ \emptyset & \emptyset \end{pmatrix} \end{matrix}$ <p>meet</p>	 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} \emptyset & \emptyset \\ -\emptyset & -\emptyset \end{pmatrix} \end{matrix}$ <p>contains</p>	 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} -\emptyset & \emptyset \\ -\emptyset & -\emptyset \end{pmatrix} \end{matrix}$ <p>Covers</p>
 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} -\emptyset & \emptyset \\ \emptyset & -\emptyset \end{pmatrix} \end{matrix}$ <p>equal</p>	 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} -\emptyset & -\emptyset \\ -\emptyset & -\emptyset \end{pmatrix} \end{matrix}$ <p>overlap</p>	 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} \emptyset & -\emptyset \\ \emptyset & -\emptyset \end{pmatrix} \end{matrix}$ <p>inside</p>	 $\begin{matrix} \partial A & \begin{matrix} \partial B & B^\circ \end{matrix} \\ A^\circ & \begin{pmatrix} -\emptyset & -\emptyset \\ \emptyset & -\emptyset \end{pmatrix} \end{matrix}$ <p>Covered By</p>

Figura 2.19 – Matriz de 4-Interseções para relações entre duas regiões.

Fonte: (Egenhofer et al., 1994).

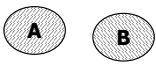


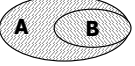




 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ disjoint	 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \neg\emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ meet	 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ contains	 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \neg\emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \neg\emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ covers
 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ equal	 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ overlap	 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \emptyset & \neg\emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \emptyset & \neg\emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix}$ inside	 $\partial B \quad B^\circ \quad B^-$ $\partial A \begin{pmatrix} \neg\emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^\circ \begin{pmatrix} \neg\emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ $A^- \begin{pmatrix} \neg\emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{pmatrix}$ covered by

Figura 2.20 – Matriz de 9-Interseções para relações entre duas regiões. Fonte: (Egenhofer e Herring, 1991).

Nos modelos citados acima, os resultados das interseções são avaliados considerando os valores vazio ou não-vazio. Há várias situações em que é necessário considerar as dimensões das interseções não vazias. Por exemplo, certo estado X só considera um outro estado Y como vizinho se eles têm pelo menos uma aresta em comum. Neste caso, para encontrar os vizinhos do estado X, não basta saber quais estados “tocam” ou são “adjacentes” a ele, mas sim se o resultado da interseção entre eles é uma aresta.

Para acomodar estas situações, novos modelos foram definidos, levando em consideração as dimensões dos resultados das interseções não vazias, como o *modelo para relações topológicas binárias detalhadas* (Egenhofer, 1993), baseado na matriz de 4-interseções, e a matriz de 9-Interseções estendida dimensionalmente (DE-9IM), baseada na matriz de 9-interseções (Paiva, 1998).

Clementini et al. (1993) estenderam a abordagem da matriz de 4-interseções de forma a incluir a informação da dimensão da interseção. No espaço bidimensional, a dimensão da interseção pode ser vazia, um ponto, uma linha ou uma região. Este modelo contempla assim um conjunto de 52 relacionamentos topológicos, o que não é conveniente do ponto de vista do usuário. Para equacionar este problema, os

relacionamentos topológicos foram agrupados em cinco mais gerais – *touch*, *in*, *cross*, *overlap*, *disjoint* – que são sobrecarregados, ou seja, que podem ser usados indistintamente para ponto, linha e região. Estes relacionamentos são definidos da seguinte forma:

touch: aplica-se a pares de geometrias dos tipos região/região, linha/linha, linha/região, ponto/região e ponto/linha:

$$\langle \lambda_1, touch, \lambda_2 \rangle \Leftrightarrow (\lambda_1^o \cap \lambda_2^o = \emptyset) \wedge \\ ((\partial \lambda_1 \cap \lambda_2^o \neq \emptyset) \vee (\lambda_1^o \cap \partial \lambda_2 \neq \emptyset) \vee (\partial \lambda_1 \cap \partial \lambda_2 \neq \emptyset))$$

in: aplica-se a pares de geometrias com qualquer combinação de tipos:

$$\langle \lambda_1, in, \lambda_2 \rangle \Leftrightarrow (\lambda_1^o \cap \lambda_2^o \neq \emptyset) \wedge (\lambda_1^o \cap \lambda_2^- = \emptyset) \wedge (\partial \lambda_1 \cap \lambda_2^- = \emptyset)$$

cross: aplica-se a pares de geometrias dos tipos linha/linha e linha/região. No caso de linha/região, temos:

$$\langle L, cross, R \rangle \Leftrightarrow (L^o \cap R^o \neq \emptyset) \wedge (L^o \cap R^- \neq \emptyset)$$

No caso de linha/linha, temos:

$$\langle L_1, cross, L_2 \rangle \Leftrightarrow \dim(L_1^o \cap L_2^o) = 0$$

overlap: aplica-se a pares de geometrias dos tipos região/região e linha/linha. No caso de região/região, temos:

$$\langle A_1, overlap, A_2 \rangle \Leftrightarrow (A_1^o \cap A_2^o \neq \emptyset) \wedge (A_1^o \cap A_2^- \neq \emptyset) \\ \wedge (A_1^- \cap A_2^o \neq \emptyset)$$

No caso de linha/linha, temos:

$$\langle L_1, overlap, L_2 \rangle \Leftrightarrow (\dim(L_1^o \cap L_2^o) = 1) \wedge (L_1^o \cap L_2^- \neq \emptyset) \\ \wedge (L_1^- \cap L_2^o \neq \emptyset)$$

disjoint: aplica-se a pares de geometrias com qualquer combinação de tipos:

$$\langle \lambda_1, disjoint, \lambda_2 \rangle \Leftrightarrow (\lambda_1^o \cap \lambda_2^o = \emptyset) \wedge (\partial \lambda_1 \cap \lambda_2^o = \emptyset) \wedge \\ (\lambda_1^o \cap \partial \lambda_2 = \emptyset) \wedge (\partial \lambda_1 \cap \partial \lambda_2 = \emptyset)$$

2.10 Determinação do relacionamento topológico

Nesta seção apresentaremos um algoritmo simples para a determinação dos relacionamentos topológicos entre dois polígonos (A e B), segundo a matriz de 9-Interseções (descrita na seção anterior). Ele utiliza uma combinação dos algoritmos geométricos apresentados anteriormente para determinar as interseções entre interior, fronteira e exterior dos dois polígonos. O algoritmo possui seis etapas, que estão descritas a seguir e são ilustradas na Figura 2.21.

1. Avaliar o relacionamento entre os REM dos polígonos A e B . Nessa avaliação podemos empregar as estratégias apresentadas por Clementini et al. (1994) que consiste basicamente no estabelecimento de um mapeamento entre os relacionamentos topológicos dos REMs e das geometrias exatas. No caso, por exemplo, de dois polígonos A e B que estejam sendo testados para ver se A contém B , podemos rapidamente descartar essa possibilidade caso o REM de A não contenha o REM de B . Caso contrário vamos à próxima etapa;
2. Determinar os pontos de interseção entre os dois polígonos. Isso pode ser feito utilizando algum dos algoritmos apresentados na Seção 0. Esta etapa nos informa se há ou não interseção entre as fronteiras dos objetos.
3. Se não houve interseção na etapa anterior (etapa 2), então devemos testar qualquer ponto do polígono A , num teste de ponto em polígono (Seção 2.4), com o polígono B , para determinar a localização de A em relação a B . Este teste precisa ser feito também com um ponto de B em relação a A :
 - a. Se o ponto do polígono A estiver dentro de B então A encontra-se dentro de B (relacionamento *inside*).
 - b. Caso contrário, se o ponto de B estiver dentro de A então A contém B (relacionamento *contains*).
 - c. Caso contrário, os polígonos são disjuntos (relacionamento *disjoint*).
4. Se houve interseção na etapa 2, devemos realizar a fragmentação da fronteira de A , em relação aos pontos de interseção.

5. Depois, verificamos a localização de cada um dos fragmentos em relação ao polígono B. Podemos utilizar o teste de ponto em polígono, tomando um ponto do fragmento que não esteja na extremidade.
6. Com base na localização dos fragmentos, as interseções entre fronteiras, interiores e exteriores podem ser inferidas:
 - a. Se houve fragmentos dentro e fora do polígono B então os dois polígonos se sobrepõe (relacionamento *overlaps*);
 - b. Se houve fragmentos somente dentro e na fronteira do polígono B, então o polígono A é coberto pelo polígono B (*covered by*).
 - c. Se houve fragmentos somente fora e na fronteira do polígono B, temos que decidir se os polígonos se tocam ou se A cobre B. Isso pode ser feito fragmentando a fronteira do polígono B, como na etapa 4 e testando a localização dos fragmentos de B em relação a A (etapa 5). Se houver fragmentos dentro de A, então A cobre B (relacionamento *covers*) senão A toca B (relacionamento *touches*).
 - d. Se todos os fragmentos encontram-se na fronteira de B, então os polígonos são iguais (relacionamento *equals*).

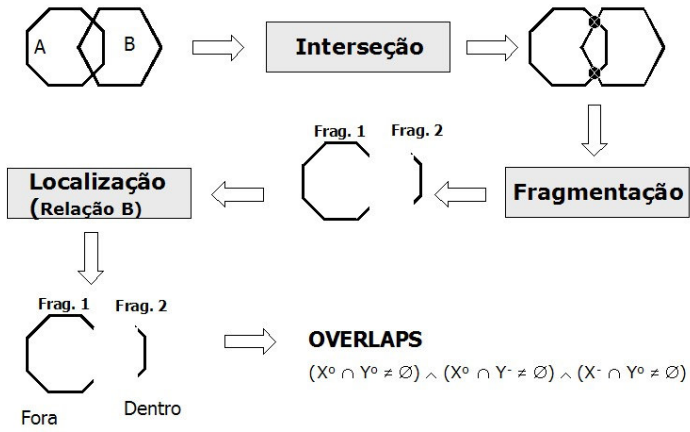


Figura 2.21 – Determinação do relacionamento topológico.

Referências

- AKMAN, V.; FRANKLIN, W. R.; KANKANHALLI, M.; NARAYANASWAMI, C. Geometric computing and uniform grid technique. **Computer-Aided Design**, v. 21, n. 7, p. 410-420, set. 1989.
- ANDREWS, D. S.; SNOEYINK, J. Geometry in GIS is not combinatorial: segment intersection for polygon overlay. In: Annual Symposium on Computational Geometry, 11., 1995, Vancouver. **Proceedings**. Canada: British Columbia, 1995, p. 424-425.
- ANDREWS, D. S.; SNOEYINK, J.; BORITZ, J. CHAN, T.; DENHAM, G.; HARRISON, J.; ZHU, C. Further comparison of algorithms for geometric intersection problems. In: International Symposium on Spatial Data Handling, 6., 1994, Edinburgh. **Proceedings**. Edinburgh: Taylor and Francis, 1994, p. 709-724.
- BEARD, K.; Theory of the cartographic line revisited: implications for automated generalization. In: **Cartographica**. 1991. v. 28, cap. 4, p. 32-58.
- BENTLEY, J. L.; OTTMANN, T. A. Algorithms for reporting and counting geometric intersections. **IEEE Transactions on Computers**, v. C-28, n. 9, p. 643-647, set. 1979.
- CLEMENTINI, E.; DI FELICE, P.; VAN OOSTEROM, P., 1993. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In: ABEL, D.; OOI, B. C., eds., **SSD '93: Lecture Notes in Computer Science**, v. 692: New York, NY, Springer-Verlag, p. 277-295.
- CLEMENTINI, E.; SHARMA, J.; EGENHOFER, M. Modelling topological spatial relations: strategies for query processing. **Computers & Graphics**, v. 18, n. 6, p. 815-822, 1994.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to algorithms**. E.U.A.: McGraw-Hill, 1990.
- DAVIS Jr., C.; Uso de vetores em GIS. **Fator GIS**, v. 21, n. 4, p. 22-23, 1997.
- DOUGLAS, D. H.; PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a line or its caricature. **The Canadian Cartographer**, v. 10, n. 2, p. 112-122, 1973.
- EGENHOFER, M. A Model for Detailed Binary Topological Relationships. **Geomatica**, v. 47, p. 261-273, 1993.

- EGENHOFER, M.; P. DI FELICE; CLEMENTINI, E. Topological Relations between Regions with Holes. **International Journal of Geographical Information Systems**, v. 8, n.2, p. 129-144, 1994.
- EGENHOFER, M.; FRANZOSA, R. On the Equivalence of Topological Relations. **International Journal of Geographical Information Systems**, v. 9, n.2, p. 133-152, 1995.
- EGENHOFER, M.; HERRING, J. Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. Orono, ME: Department of Surveying Engineering, University of Maine, 1991.
- FIGUEIREDO, L. H.; CARVALHO, P. C. P. **Introdução à geometria computacional**. Rio de Janeiro: IMPA, 1991.
- FRANKLIN, W. R.; CHANDRASEKHAR, N.; KANKANHALLI, M.; SESHAN, M.; AKMAN, V. Efficiency of uniform grids for intersection detection on serial and parallel machines. In: *Computer Graphics International*, maio, 1988, Geneva, Switzerland. **Proceedings**. Berlin Heidelberg: Springer-Verlag, 1988, p. 51-62.
- FRANKLIN, W. R.; CHANDRASEKHAR, N.; KANKANHALLI, M.; SUN, D.; ZHOU, M.; WU, P. YF Uniform grids: a technique for intersection detection on serial and parallel machines. In: *Auto Carto 9*, Baltimore, Mariland. **Proceedings**. Baltimore, Maryland: American Congress on Surveying and Mapping, 1989, p. 100-109.
- HAINES, E. Point in polygon strategies. In: HECKBERT, P. S. (Org.). **Graphics gems IV**. Boston, E.U.A.: Academic Press, 1994. p. 24-46.
- HUANG, C.; SHIH, T. On the complexity of point-in-polygon algorithms. **Computers & Geosciences**. v. 23, n. 1, p. 109-118, 1997.
- KNUTH, D. E. **The art of computer programming, vol. 1: fundamental algorithms**. Boston, Massachusetts: Addison-Wesley, 1973.
- LAURINI, R.; THOMPSON, D. **Fundamentals of spatial information systems**. : Academic Press, 1992.
- LI, Z.; OPENSHAW, S. Algorithms for automated line generalization based on a natural principle of objective generalization. **International Journal of Geographic Information Systems**, v. 6, n. 5, p. 373-389, 1992.
- MARGALIT, A.; KNOTT, G. D. An algorithm for computing the union, intersection or difference of two polygons. **Computers & Graphics**, v. 13, n. 2, p. 167-183, 1989.

- MARINO, J. S.; Identification of characteristic points along naturally occurring lines: an empirical study. **The Canadian Cartographer**, v. 16, n. , p. 70-80, 1979.
- MCMMASTER, R. B.; SHEA, K. S. Generalization in digital cartography. **Association of American Geographers**, 1992.
- O'ROURKE, J.. **Computacional geometry in C**. Cambridge: Cambridge University Press, 1998.
- PAIVA, J. A. C. Topological Equivalence and Similarity in Multi-Representation Geographic Database. University of Maine, 1998.
- PEUCKER, T. K.; A theory of the cartographic line. In: **International yearbook of cartography**. 1975. cap. 16, p. 134-143.
- PREPARATA , F. P.; SHAMOS , M. I. **Computational geometry an introduction**. New York: Springer-Verlag, 1985.
- PULLAR, D. Comparative study of algorithms for reporting geometrical intersections. In: International Symposium on Spatial Data Handling, 4., 1990, Zurich. Proceedings... Edinburgh: Taylor and Francis, 1990, p. 66-76.
- SAALFELD, A. It doesn't make me nearly as CROSS - some advantages of the point-vector representation of line segments in automated cartography. **International Journal of Geographical Information Systems**, v. 1, n. 4, p. 379-386, 1987.
- SCHNEIDER, M. **Spatial data types for database systems**. Berlin Hidelberg: Springer-Verlag, 1997.
- SHAMOS, M. I.; HOEY, D. Geometric intersection problems. In: Annual IEEE Symposium on Foundations of Computer Science, 17., oct. 1976, Houston, Texas. **Proceedings**. New York: IEEE, 1976, p. 208-215.
- TAYLOR, G. E.; Point in Polygon Test. **Survey Review**, v. 32, n. 254, p. 479-484, 1994.
- VAN OOSTEROM, P.; **Reactive data structures for geographic information systems**. : Oxford University Press, 1993.
- VAN OOSTEROM, P.; SCHENKELAARS, V. The development of an interactive multi-scale GIS. **International Journal of Geographical Information Systems**, v. 9, n. 5, p. 489-507, 1995.
- WEIBEL, R.; Map generalization in the context of digital systems. **Cartography and Geographical Information Systems**, v. 22, n. 4, p. 3-10, 1995.