# HPS-tree: Um método de acesso para armazenar mapas longos com multi-resolução geométrica e topológica

Maurício Riguette Mediano [1]
Marcelo Gattass [1]
Marco Antônio Casanova [2]

[1] TeCGraf - Grupo de Tecnologia em Computação Gráfica, Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente, 225
CEP 22.453/900 - Rio de Janeiro - RJ - Brasil
{mediano,gattass}@inf.puc-rio.br

[2] Centro de Soluções para Educação Superior e Pesquisa - IBM Brasil
Av. Pasteur, 138 - 8º andar
CEP 22.296/900 - Rio de Janeiro - RJ - Brasil
casanova@vnet.ibm.com

**Abstract.** Access methods play a fundamental role on the performace of geographic databases that typically stores very large maps. This paper introduces a new data structure, called HPS-tree, designed to store geometrical and topological information of very large maps. The proposed structure is designed to efficiently handle the topology and the complete geometry description of the map, with geometrical and topological multiresolution. **Keywords:** Multiresolution, Topological Data Structures, Geographical Databases, Access Methods.

## 1 Introdução

Objetos gráficos complexos em grandes bases de dados de sistemas gráficos necessitam de métodos eficientes para acessar subconjuntos de objetos destas bases de dados, e até mesmo partes desses objetos. A eficiência do acesso a esses objetos complexos e seus conjuntos, está intimamente ligada ao uso desses objetos. Vários métodos de acesso para objetos em uma e duas dimensões tem sido projetados com o objetivo de otimizar conjuntos bem definidos de operadores sobre esses objetos.

A Z-order [Ore86] transforma pontos do espaço n-dimensional em pontos no espaço uni-dimensional, para então indexar os mesmos usando B-trees [Com79]. As R-trees [Gut84, SRF87, BKSS90] são B-trees que indexam conjuntos de retângulos com chaves de busca bi-dimensionais. O Grid-file [NHS84] usa vetores de duas dimensões para indexar pontos e retângulos.

Dois métodos de acesso foram propostos para indexar a geometria completa de objetos espaciais: a TR-tree [SK91, BKSS94] e a família das V-trees [MCD94a, Med95]. A TR-tree indexa geometricamente uma região através de um conjunto de trápezios que formam a região, armazenados em uma R-tree, onde já estão representados os furos da região (tal estrutura não admite edição, pois os trapézios precisariam ser gerados novamente). As V-trees indexam as linhas de cada objeto espacial, admitem edição e são três a quatro vezes mais compactas para representar as mesmas geometrias quando comparadas à TR-tree, o que as torna mais eficientes.

Subdivisões planares podem ser representadas através de um pequeno conjunto de operadores formalmente descritos em [Per95]. Há uma série de estruturas topológicas que implementam de forma eficiente esses operadores, armazenando relações de adjacência entre vértices, faces e arestas. Entre estas estruturas, destacam-se: winged-edge [Bau75], half-edge [Män88], face-edge [Wei86] e HPS [FFGC95].

O objetivo principal deste trabalho é definir uma estrutura de dados capaz de armazenar hierarquias de subdivisões planares muito grandes, otimizando operadores de edição e consulta sobre as mesmas. Desta forma é definido um método de acesso para hierarquias de mapas longos, chamado HPS-tree, baseado em:

- V-trees para otimizar operações geométricas com multi-resolução sobre geometrias de linhas poligonais;
- PV-trees para indexar geometricamente o acesso a regiões formadas por seqüências de linhas poligonais;
- R-trees, para indexar geometricamente o acesso a conjuntos de retângulos;

- uma variante da estrutura HPS, chamada Compact HPS, para armazenar de forma eficiente relações de adjacências em hierarquias de subdivisões planares.

A seção 2 contém preliminares sobre métodos de acesso espaciais e estruturas de dados topológicas. A versão compacta da estrutura de dados topológica HPS é definida na seção 3. A HPS-tree é definida na seção 4 como um método de acesso para mapas longos com multi-resolução geométrica e topológica. As seções 5 e 6 contém as conclusões e agradecimentos, respectivamente.

## 2 Preliminares

As consultas a uma base geográfica podem ser de duas naturezas: geométricas ou topológicas. As consultas geométricas são do tipo: *"quais entidades estão em tal região do espaço?"*, e são melhor atendidas pelos métodos de acesso espacial. As consultas topológicas tratam da questão: *"quais entidades são adjacentes a uma tal entidade?"*. As estruturas topológicas procuram responder esta questão com uma busca local.

### 2.1 Métodos de Acesso Espacial

Dada uma seqüência finita de pontos $Q$, uma *linha poligonal* é formada pela união de todos os segmentos reta definidos por pares de pontos consecutivos de $Q$. No contexto deste trabalho, será usado o termo *linha* para denotar uma *linha poligonal*.

As V-trees são uma família de estruturas de dados, inicialmente propostas em [MCD94a] e [MCD94b], com diversas aplicações nas áreas de bancos de dados geográficos e computação gráfica. A V-tree indexa a geometria completa de uma linha, otimizando o acesso a memória secundária e diversas operações geométricas, com ou sem multi-resolução.

Dados dois inteiros $m > 1$ e $n > 1$, uma *V-tree* de ordem $(m, n)$ é uma árvore m-ária tal que:

- todas as folhas estão no mesmo nível;
- cada folha $N$ contém uma seqüência de pontos de tamanho entre $n/2$ e $n$;
- o retângulo envolvente de $N$ é o retângulo envolvente da seqüência de pontos que $N$ armazena;
- cada nó interior tem entre $m/2$ e $m$ filhos, exceto a raiz, que tem entre 2 e $m$ filhos;
- para cada filho $M$ de um nó interior $N$, $N$ contém uma lista de pares da forma $(p, r)$, onde $p$ aponta para $M$ e $r$ é o retângulo envolvente de $M$;
- o retângulo envolvente de $N$ é o retângulo envolvente que cobre todos os retângulos envolventes das entradas em $N$.

Dada uma linha $P$, pode-se armazenar $P$ em uma V-tree $V$ de ordem $(m, n)$, quebrando $P$ em fragmentos consecutivos de tamanho $n$ e armazenando cada fragmento em uma folha de uma V-tree. No processo de quebrar uma linha em fragmentos consecutivos, o último ponto de um fragmento é também o primeiro ponto do fragmento seguinte, tal que os retângulos envolventes dos fragmentos cobrem completamente a linha. Naturalmente, todos os algoritmos que reconstroem $P$ a partir de $V$ precisam estar atentos para o fato de o último ponto de um fragmento ser o primeiro ponto do próximo fragmento (exceto se o fragmento é o último).

Neste ponto, pode-se discutir algumas considerações práticas. Primeiro, os nós interiores e folhas não precisam ocupar páginas de mesmo tamanho, pois elas armazenam entradas de naturezas diferentes. Em outras palavras, os parâmetros $m$ e $n$ a princípio não têm relação. Segundo, pode-se manter as folhas encadeadas e agrupadas em memória secundária para facilitar o acesso a fragmentos consecutivos de uma linha. Se a linha é fechada, as folhas podem ser organizadas como uma lista circular.

O exemplo na Figura 1 mostra a visão de vários níveis de uma V-tree. Em cada um dos níveis 1, 2 e 3 da árvore foi gerada uma aproximação da geometria da linha armazenada nas folhas. A geometria de cada parte de uma linha armazenada em uma subárvore foi aproximada para o centróide do retângulo envolvente da mesma. Quanto mais próximo do último nível da estrutura, maior é a resolução da linha gerada. No nível das folhas (4) a geometria é apresentada com máxima resolução. Outras formas usar multi-resolução geométrica são apresentadas em [Med95].

Há pelo menos duas maneiras de representar fronteiras de faces. Uma delas é através de uma linha (seqüência de pontos) e outra é através de uma *cadeia* (seqüência de arestas). A forma mais simples é através de uma linha. Na segunda forma, arestas pertencentes a faces distintas são compartilhadas. Ao representar fronteiras de faces como seqüências de pontos, é necessário que uma mesma seqüência de pontos relativa a fronteira de duas faces seja repetida na fronteira de cada face. Ao representar faces como cadeias de arestas, evita-se esta redundância, pois para representar uma aresta comum a duas faces, basta que as duas cadeias que representam a fronteira das duas faces possuam uma referência para a aresta comum.

Para armazenar a fronteira de uma face como

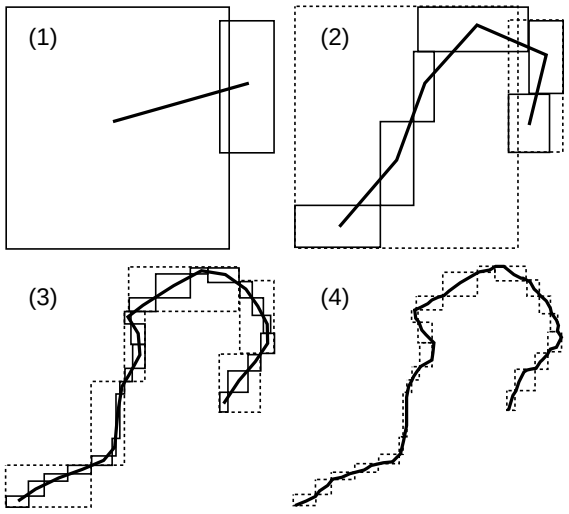


Figura 1: Vários níveis de uma V-tree de ordem(4,4).

uma cadeia de arestas, define-se uma estrutura chamada *PV-tree* [Med95], uma variante da V-tree que, em vez de armazenar seqüências de pontos, armazena cadeias (seqüências) de arestas.

Dado um inteiro $m > 1$, uma *R-tree* de ordem $(m)$ é uma árvore m-ária tal que:

- todas as folhas estão no mesmo nível;
- cada nó interior tem entre $m/2$ e $m$ filhos, exceto a raiz, que tem entre 2 e $m$ filhos;
- para cada filho $M$ de um nó $N$, $N$ contém uma lista de pares da forma $(p, r)$, onde $p$ é uma referência para $M$ e $r$ é o retângulo envolvente de $M$;
- o retângulo envolvente de $N$ é o retângulo envolvente que cobre todos os retângulos envolventes das entradas em $N$;
- quando $M$ é uma folha, $r$ é um retângulo envolvente (uma aproximação) de uma região $r$ do $\Re^2$ armazenada na R-tree e $p$ é uma referência para $r$.

Ao percorrer uma R-tree que indexa um conjunto de regiões do $\Re^2$, é possível desprezar subárvores inteiras que não interceptam uma determinada região de consulta, quando o retângulo envolvente da subárvore não intercepta a região. Tal propriedade é válida também para V-trees. Nas folhas da R-tree, cada região é aproximada para um retângulo envolvente. Quando o retângulo envolvente da região não é suficiente para resolver uma operação espacial, a geometria completa da região é recuperada.

### 2.2 Estruturas de dados topológicas

De acordo com [Cav92], topologia de conjuntos de pontos, uma das subáreas de modelagem geométrica, pode ser definida como o estudo das propriedades invariantes por homeomorfismos. Homeomorfismo pode ser imaginado como uma transformação elástica que preserva adjacências, além de outras propriedades topológicas.

Seja $S$ uma *subdivisão planar* do $\Re^2$, $S$ é composta por conjuntos de vértices, arestas e faces, onde: um *vértice* é um mesmo ponto do $\Re^2$ que pertence a fronteira de uma ou mais arestas de $S$; uma *aresta* é uma linha que separa duas faces de $S$; e uma *face* é uma seqüência alternada de vértices e arestas que formam uma das partições de $S$. Pontos e linhas que não pertencem a fronteira de nenhuma face são considerados como *detalhes* da região [PD95].

A winged-edge [Bau75], a primeira estrutura proposta para representar as relações de adjacência em subdivisões planares, é composta por listas de faces, vértices e arestas. As estruturas face-edge [Wei86] e half-edge [Män88] introduzem descritores, chamados de laços e usos que permitem representar de forma mais clara topologias de faces com furos, quando comparadas à winged-edge. O HPS [FFGC95] pode ser considerado uma evolução das estruturas anteriores por prover recursos de multiresolução topológica. Em comum, estas estruturas possuem o fato de serem baseadas em arestas, pois, em modelagem manifold, cada aresta em uma subdivisão planar possui um número constante de relações de adjacência: dois vértices e duas faces adjacentes. A Figura 2 mostra as relações entre entidades em estruturas de dados baseadas em arestas.

Uma hierarquia de subdivisões planares, segundo [FFGC95], pode ser definida através de duas regras que mantém a consistência vertical dentro da hierarquia. A regra de *consistência top-down* garante que cada entidade $x$ está totalmente contida em alguma entidade $y$ para cada nível acima do nível de $x$. Diz-se que $x$ é *coberta* por $y$. A segunda regra, chamada de *consistência bottom-up*, garante que cada entidade $x$ é formada pela união de entidades a cada nível abaixo do nível de $x$. Diz-se que $x$ é refinada em níveis inferiores.

A estrutura de dados HPS [FFGC95] é composta por modelos, faces, laços, usos, arestas e vértices:

- a entidade *modelo* é composta de uma lista de faces (*face_tree*), uma lista de arestas (*edge_tree*), uma lista de vértices (*vertex_list*), um indicador do nível da hierarquia ao qual pertence (*num_level*) e duas referências que formam uma lista duplamente encadeada de modelos (*next*,*prev*);

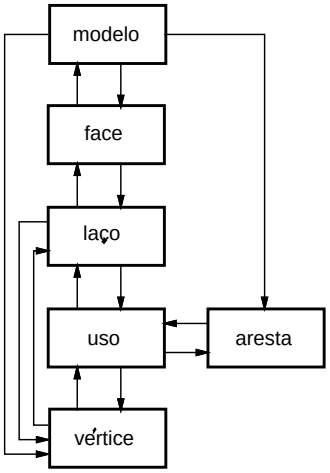


Figura 2: Organização de estruturas de dados topológicas baseadas em arestas.

- a entidade *face* é composta por uma referência para o modelo ao qual pertence (*model_ptr*), uma referência para o laço externo da *face* (*lout_ptr*), uma referência para a lista de laços internos (furos) da *face* (*loop_list*), uma referência para a face pai (*parent*), uma referência para uma face filha (*child*) e duas referências que formam uma lista duplamente encadeada de faces (*next*,*prev*);
- a entidade *laço* possui uma referência para a face a qual pertence (*face_ptr*), uma referência para um dos usos que formam o laço (*use_ptr*) e duas referências que formam uma lista duplamente encadeada de laços internos a este (*next*,*prev*);
- a entidade *uso* de aresta possui uma referência para a aresta da qual ela é um uso (*edge_ptr*), uma referência para o laço ao qual o uso pertence (*loop_ptr*), uma referência para o vértice de origem (*vertex_ptr*) e duas listas de referências para os dois usos de arestas adjacentes em cada nível igual ou inferior na hierarquia de modelos (*nexts*,*prevs*);
- a entidade *aresta* possui duas referências para os dois usos da aresta, uma referência para a aresta pai (*parent*), uma referência para uma aresta filha (*child*) e duas referências que formam uma lista duplamente encadeada de arestas (*next*,*prev*);
- a entidade *vértice* possui uma referência para uma entidade adjacente (*use_ptr*) e e duas referências que formam uma lista duplamente encadeada de vértices (*next*,*prev*);

A Figura 3 mostra um exemplo de hierarquia de subdivisões planares onde as faces, arestas e vértices se encontram dispostos de acordo com a Figura 4. As faces $f111$ e $f112$ são um refinamento de $f11$, bem como: $f11$ e $f12$ para $f1$; $a111$ e $a112$ para $a11$; $a11$ e $a12$ para $a1$; e $a61$ e $a62$ para $a6$.

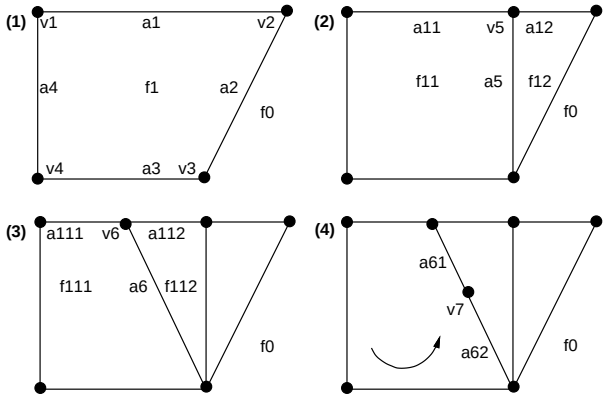


Figura 3: Um exemplo de hierarquia de subdivisões planares.

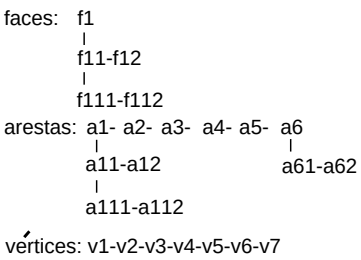


Figura 4: Disposição de faces, arestas e vértices em árvores e listas, na estrutura HPS.

A estrutura HPS (com algumas modificações a serem discutidas na seção 3) será a estrutura topológica adotada no trabalho, em fução de prover recursos de multi-resolução topológica.

## 3 A estrutura Compact HPS

Como objetos longos tentem a ser armazenados por partes em unidades de armazenamento em memória secundária, torna-se mais adequado compôr mapas longos a partir de pedaços de tamanho fixo, para então recuperá-los por partes com a ajuda dos métodos de acesso. Por esse motivo torna-se adequado rever a estrutura HPS para conter apenas descritores de tamanho fixo.

A entidade uso na estrutura HPS possui atributos que são listas (nexts e prevs), o que torna o

```
algorithm nextuse
  input: s, M
  output: next
begin
  atribua a next o próximo uso no
                        último nível de s;
  while level(next) > M do
    if (parent(next) <> nil) then
      atribua a next o mesmo uso
                        da aresta pai associada a s;
    else
      atribua a next o próximo uso
                        do uso oposto de s;
  end-while
end nextuse
```

Figura 5: Algoritmo nextuse(s,M).

tamanho da entidade variável. Estes atributos são listas porque cada uso pode ter dois usos adjacentes em cada modelo da hierarquia ao qual o uso pertence (igual ou abaixo do seu nível).

Se ao armazenar um conjunto fixo $C$ dessas adjacências entre usos fosse possível descobrir as outras adjacências, bastaria armazenar este conjunto fixo $C$. Informalmente, ao armazenar as duas referências para os dois usos adjacentes no nível de maior resolução (*nextl* e *prevl*) é possível alcançar todos os outros níveis (através das demais relações topológicas armazenadas na estrutura HPS). Por definição, o nível de maior resolução é composto por todas as arestas dos níveis superiores com ou sem refinamento. A partir das níveis mais refinados é possível alcançar os níveis menos refinados usando a relações de adjacência disponíveis na estrutura HPS.

Dado um uso $s$ em um nível de busca $M$, o algoritmo nextuse (Figura 5) mostra como alcançar o próximo uso de $s$ no mesmo nível, a partir do próximo uso de $s$ no nível de maior resolução. O algoritmo tenta subir na hierarquia da aresta corrente ou girar em torno do vértice que separa os dois usos até encontrar o primeiro uso de nível maior ou igual a $M$. Na estrutura HPS a operacão de girar em torno de um vértice é feita tomando o próximo uso da aresta oposta ao uso corrente. Um algoritmo simétrico pode ser construído para encontrar os usos anteriores.

De acordo com o exemplo na Figura 3, para alcançar o next de $a3$ no nível 4, atribui-se a *next* o próximo uso de $a3$ no nível de maior resolução, no caso $a62$. Como o nível de $a62$ é 4, o algoritmo encerra com sucesso. Caso se deseje alcançar o next de $a3$ no nível 3, atribui-se a *next* o mesmo uso da aresta pai de $a62$, no caso $a6$. Como o nível de $a6$ é 3, o algoritmo encerra com sucesso. Caso se deseje alcançar o next de $a3$ no nível 2, atribui-se a *next* o próximo uso do uso oposto de $a6$, no caso $a5$. Como o nível de $a5$ é 2, o algoritmo encerra com sucesso. Caso se deseje alcançar o next de $a3$ no nível 1, atribui-se a *next* o próximo uso do uso oposto de $a5$, no caso $a2$. Como o nível de $a2$ é 1, o algoritmo encerra com sucesso.

Pode se considerar ineficiente percorrer caminhos no nível mais refinado, como o caminho do exemplo $a3, a62, a6, a5, a2$, para descobrir que $a2$ é o próximo uso adjacente a $a3$ no nível 1. Para contornar este problema, permite-se que cada uso referencie também os usos próximo e anterior, *nextm* e *prevm*, no nível no qual é instanciado. Dado um uso $s$, só haverá necessidade de percorrer o nivel de maior precisão para descobrir o próximo uso de $s$, quando $s$ estiver armazenada em um nível acima do nível da hierarquia que está sendo acessado. Em contrapartida, será mais simples editar a estrutura, pois há menos redundância entre as relações de adjacência de usos.

Baseada na estrutura HPS, chama-se de *Compact HPS* uma estrutura similar onde: a lista de referências nexts é substituída por nextm e nextl; e a lista de referências prevs é substituída por prevm e prevl.

## 4 Armazenamento de Mapas

A estrutura de armazenamento proposta para mapas, chamada *HPS-tree*, armazena uma hierarquia de subdivisões planares muito grandes. Para tanto, a estrutura combina os métodos de acesso V-tree, PV-tree e R-tree com a estrutura topológica Compact HPS.

A Figura 6 mostra a organização da HPS-tree. Quando comparada com a organização de outras estruturas topológicas baseadas em arestas (Figura 2), pode-se notar como a HPS-tree é acrescida pelos métodos de acesso para retângulos (R-tree), geometrias de arestas (V-tree) e seqüências de arestas (PV-tree).

As contribuições feitas pela HPS-tree à Compact HPS são as seguintes:

- Na entidade modelo, as referências para as árvores de faces (face_tree), arestas (edge_tree) e a lista de vértices (vertex_list), passam a referenciar três R-trees que contém o primeiro nível das árvores de faces e arestas, e lista completa de vértices, respectivamente.
- A referência para a lista de laços internos (loop_list) na entidade face passa a referenciar

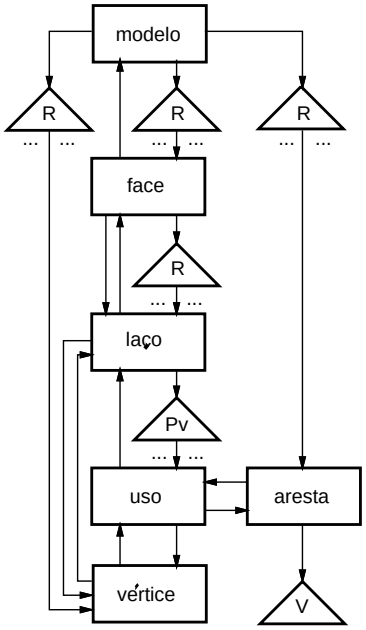


Figura 6: Organização da estrutura HPS-tree.

uma R-tree que referencia o mesmo conjunto de laços. Deixam de existir na entidade laço as referências next e prev.

- Quando a entidade adjacente à entidade laço é um uso, a referência para a lista de usos (use_ptr) passa a referenciar uma PV-tree que referencia a mesma seqüência de usos.

- Quando a aresta não possui refinamentos, a entidade aresta passa a referenciar uma V-tree que armazena a geometria associada à aresta. Quando uma aresta possui refinamentos, a sua geometria passa a ser formada pela geometria dos seus refinamentos.

Os triângulos R, Pv e V na Figura 6 mostram como as R-trees, PV-trees e V-trees se adaptam ao modelo de estruturas topológicas baseadas em arestas da Figura 2, respectivamente. As R-trees, imediatamente abaixo da entidade modelo, indexam geometricamente conjuntos de faces, vértices e arestas, o que otimiza consultas a conjuntos dessas entidades com restrições geométricas. A R-tree, entre as entidades face e laço, indexa geometricamente o conjunto de laços internos de cada face. Este método permite otimizar consultas com restrições geométricas a esse grupo de entidades. A PV-tree, entre as entidades laço e uso, indexa geometricamente a seqüência de usos que forma cada laço. Por fim, a V-tree abaixo da entidade aresta indexa a seqüência de pontos que forma a geometria da aresta, provendo multi-resolução geométrica.

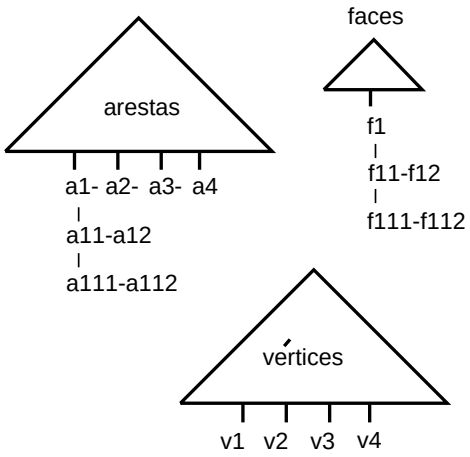


Figura 7: Disposição de faces, arestas e vértices na HPS-tree.

A Figura 7 mostra a disposição das faces, vértices e arestas da HPS-tree no nível (1) do exemplo na Figura 3. Ao comparar com o mesmo exemplo disposto na estrutura HPS (Figura 4), nota-se a possibilidade de procurar as entidades, através de métodos de busca com restrição geométrica, usando R-trees.

Os métodos utilizados para indexar os conjuntos de laços internos de uma face e as seqüências de usos de um laço, se justificam devido a frequência com a qual essas entidades aparecem em mapas geográficos. Por exemplo, em mapas de sistemas de informações geográficas, as faces que representam rios podem ser formadas por milhares de arestas de faces que representam as margens dos rios. As regiões de seringais que aparecem no meio da floresta são outro exemplo, onde faces contém centenas de pequenos furos.

As operações de edição de mapas são semelhantes as operações de edição da estrutura HPS. A única diferença é que em vez de percorrer listas para procurar os vértices, arestas e faces próximos a um ponto ou que interceptam uma região do espaço, as operações com restrições geométricas sobre conjuntos dessas entidades são otimizadas percorrendo as R-trees. As operações de inserção, remoção e consulta em R-trees estão bem definidas em [Gut84, RL85, SRF87, BKSS90, HNP95]. A questão das V-trees e das PV-trees é ortogonal a esta discussão, pois a Compact HPS trata apenas da topologia de mapas, deixando em aberto como são implementadas as operações sobre a geometria das arestas. O armazenamento e consequente otimização

do acesso a geometria de arestas e faces usando V-trees e PV-trees são abordadas em [Med95].

## 5 Conclusão

A estrutura topológica Compact HPS é definida como uma variante da estrutura HPS que possui um número fixo de adjacências entre usos. Tal característica simplifica a nova estrutura, tornando-a mais adequada para o uso em memória secundária.

O método de acesso para mapas proposto, chamado HPS-tree, une os recursos da estrutura topológica Compact HPS com os métodos de acesso para retângulos, linhas e faces; R-trees, V-trees e PV-trees, respectivamente. Desta forma, a HPS-tree permite armazenar hierarquias de mapas muito grandes, otimizando o acesso a subconjuntos de vértices, arestas e faces de um mapa longo, com restrições geométricas (R-trees). São otimizadas também operações geométricas e operações que utilizem multi-resolução geométrica envolvendo arestas e faces (V-trees e PV-trees).

As relações topológicas entre vértices, arestas e faces são tratadas através das informações de adjacência armazenadas pela Compact HPS. Estas informações de adjacências armazenadas, permitem que a HPS-tree mantenha de forma eficiente a consistência topológica de hierarquias de subdivisões planares de um mapas muito grandes.

## 6 Agradecimentos

Este trabalho foi parcialmente sustentado pelos projetos: PROTEM/GEOTEC financiado pelo CNPQ; e TeCGraf financiado pelo CENPES/Petrobrás. O primeiro autor gostaria de agradecer também a CAPES pelo financiamento de sua bolsa de estudos. Waldemar Celes Filho, Paulo Cezar Carvalho e Luiz Fernando Martha contribuiram discutindo diversos pontos do trabalho. Aos três os autores agradecem.

## Referências Bibliográficas

[Bau75] B. G. Baumgart. Winged-edge polyedron representation. In *AFIPS Proc.*, number 44 in 1, pages 589–596, 1975.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Retangles. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 322–332, May 1990.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-Step Processing of Spatial Joins. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 197–208, May 1994.

[Cav92] Paulo Roma Cavalcanti. *Criação e Manutenção de Subdivisões do Espaço*. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica, Março 1992.

[Com79] D. Comer. The Ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–131, June 1979.

[FFGC95] Waldemar Celes Filho, Luiz Henrique Figueredo, Marcelo Gattass, and Paulo Cezar Carvalho. A topological dada structure for hierarquical planar subdivisions. Technical Report CS-95-53, Department of Computer Science, University of Waterloo, 1995.

[Gut84] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD Conference on Data Engineering*, pages 47–56, June 1984.

[HNP95] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized Search Trees for Database Systems. Technical report, Submitted to VLDB, 1995.

[Män88] Martti Mäntylä. *Solid Modelling*. Computer Science Press, 1988.

[MCD94a] Maurício Riguette Mediano, Marco Antônio Casanova, and Marcelo Dreux. A Family of Storage Methods for Geographic Data. In *9º Simpósio Brasileiro de Bancos de Dados*, Setembro 1994.

[MCD94b] Maurício Riguette Mediano, Marco Antônio Casanova, and Marcelo Dreux. V-tree - A Storage Method for Long Vector Data. In *Proceedings of the 20th VLDB Conference*, September 1994.

[Med95] Maurício Riguette Mediano. *V-trees: Um Método de Armazenamento para Dados Vetoriais Longos*. PhD thesis, Pontifícia Universidade Católica, Rio de Janeiro, Fevereiro 1995.

[NHS84] J. Nievergelt, H. Hinterberguer, and K. C. Sevick. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[Ore86] J.A. Orenstein. Spatial Query Processing in an Object-Oriented Database System. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 326–336, May 1986.

[PD95] Enrico Puppo and Giuliana Dettori. Towards a formal model for multiresolution spatial maps. In *Proceedings SSD*, 1995.

[Per95] Ronaldo Marinho Persiano. Representação Computacional de Subdivisões Planares. Technical report, Universidade Federal do Rio de Janeiro, 1995.

[RL85] Nick Roussopoulos and Daniel Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-trees. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 17–31, May 1985.

[SK91] Ralf Schneider and Hans-Peter Kriegel. The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations. In *Proceedings of 7th Workshop on Computational Geometry*, pages 507–518, 1991.

[SRF87] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The $R^+$-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th VLDB Conference*, pages 507–518, September 1987.

[Wei86] Kevin Weiler. *Topological Structures for Geometric Modeling.* PhD thesis, Renssealer Polytechnic Institute, August 1986.