MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14167-TDI/1084

UM SUBSISTEMA EXTENSÍVEL PARA O ARMAZENAMENTO DE GEO-CAMPOS EM BANCO DE DADOS GEOGRÁFICOS

Lúbia Vinhas

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Gilberto Câmara, aprovada em 03 de março de 2006.

INPE
São José dos Campos
2006

519.156

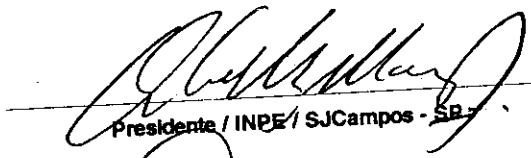
Vinhas, L.
Um subsistema extensível para o armazenamento de geo-campos em bancos de dados geográficos / Lúbia Vinhas. – São José dos Campos: INPE, 2006.
112p. ; (INPE-14167-TDI/1084)

1.Sistemas de Informação Geográfica (SIG). 2.Bancos de dados. 3.Bancos de dados de imagens. 4.Geoinformação. 5.Terralib. I.Título.

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de **Doutor(a)** em

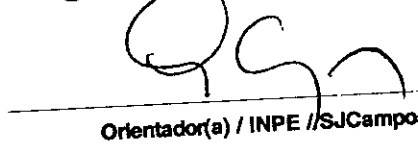
Computação Aplicada

Dr. Antonio Miguel Vieira Monteiro



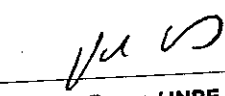
Presidente / INPE / SJCampos - SP

Dr. Gilberto Câmara



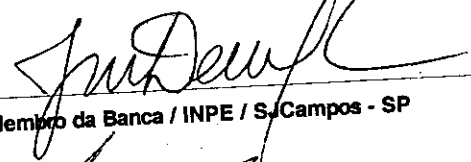
Orientador(a) / INPE / SJCampos - SP

Dr. Rafael Duarte Coelho dos Santos



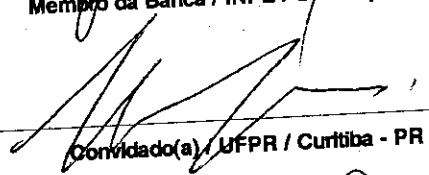
Membro da Banca / INPE / SJCampos - SP

Dr. José Demisio Simões da Silva



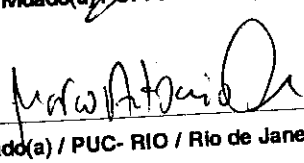
Membro da Banca / INPE / SJCampos - SP

Dr. Marcos Sfair Sunye



Convidado(a) / UFPR / Curitiba - PR

Dr. Marco Antônio Casanova



Convidado(a) / PUC- RIO / Rio de Janeiro - RJ

Aluno (a): **Lúbia Vinhas**

São José dos Campos, 30 de março de 2006

AGRADECIMENTOS

Ao Dr. Gilberto Câmara e Ricardo Cartaxo Modesto de Souza pela orientação e ao Dr. Marco Antônio Casanova pela imensurável ajuda com o texto. Sou grata por sua amizade e confiança. Minha admiração por eles cresce a cada dia.

Ao Antônio Miguel Vieira Monteiro, Karine Reis Ferreira, Gilberto Ribeiro de Queiroz, Juan Garrido, Lauro Hara e Emiliano Castejon pela construção da TerraLib. Para Karine um agradecimento especial por ser uma ótima companheira de sala.

A todos da Divisão de Processamento de Imagens do INPE pela solidariedade e espírito de grupo. Ao Laércio Namikawa pela ajuda na obtenção das referências bibliográficas.

Ao Rui Mauricio da Divisão de Geoprocessamento da Funcate por ter testado e usado extensivamente algumas das soluções desenvolvidas neste trabalho.

Ao INPE e ao seu Programa de Pós-Graduação em Computação Aplicada pela oportunidade.

A minha "familhona" (minha mãe, irmãs e irmãos, sobrinhas, sobrinhos e sobriinhas) por existirem e por todo o apoio e ajuda desde sempre. Em especial à filial São José da família: Heluiza e Benitez, Mara e Paulo minhas irmãs e cunhados, cujas casas estão sempre abertas e por sempre me ajudarem quando preciso.

Aos amigos "de desde o início" com os quais venho dividindo casas, viagens, crianças, alegrias (e algumas tristezas), além de dúvidas e certezas sobre INPE, carreira, ciência ou São José dos Campos, obrigada pela paciência e pela torcida: Sil, Marisa, Maycira, Regina, Tati, Marília, Sergio, Alejandro, Lu, Milton, Ronald e Kevin. Finalmente, aos amigos do meu grupo de capoeira na ADC-INPE pelas horas em que fizeram com que eu me esquecesse que havia uma tese para ser feita.

RESUMO

Esta tese discute a questão da manipulação de dados geográficos do tipo geo-campos. A abordagem adotada baseia-se na especificação de um subsistema semi-completo, que permite acessar e armazenar geo-campos de diversas naturezas, de maneira unificada. Inicialmente, caracteriza-se detalhadamente as representações matriciais de geo-campos. Para essas, apresenta-se um estudo sobre como armazená-las em bancos de dados objeto-relacionais, levantando alguns requisitos e propostas de solução, que consideram questões práticas de eficiência. O resultado materializa-se sob a forma de um subsistema extensível para armazenamento de representações matriciais particionadas e em multiresolução, utilizando bancos de dados objeto-relacionais. Por fim, apresenta-se uma implementação concreta do subsistema, desenvolvida como parte da biblioteca TerraLib, juntamente com alguns exemplos de uso, que demonstram que o subsistema proposto atende a diversos requisitos de manipulação de representações matriciais, especialmente no caso de imagens digitais de sensoriamento remoto.

AN EXTENSIBLE STORAGE SUBSYSTEM FOR GEO-FIELDS IN GEOGRAPHICAL DATABASES

ABSTRACT

This thesis discusses the management of geographical data represented as geo-fields. The approach adopted is to specify a semi-complete subsystem that allows the storage and access to different types of geo-fields in a unified way. Raster representations of geo-fields are first characterized in details. For this kind of representation, a study of the requirements related to their storage in object-relational databases, that includes efficiency considerations, is carried out. The result is materialized as an extensible storage subsystem for tiled multi-resolution raster representations, running on top of object-relational databases. Finally, a concrete implementation, using the TerraLib library, is described, as well as use cases that demonstrate the usefulness of the subsystem, specially when dealing with digital remotely sensed images.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

$\Re$ - Conjunto dos números reais

I - Conjunto dos números inteiros

LISTA DE SIGLAS E ABREVIATURAS

ASCII - *American Standard Code for Information Interchange*

COM - *Component Object Model*

CORBA - *Common Object Request Broker Architecture*

DPI - Divisão de Processamento de Imagens

INPE - Instituto Nacional de Pesquisas Espaciais

OGC - *Open Geospatial Consortium*

SGBD - Sistema de Gerenciamento de Bancos de Dados

SIG - Sistema de Informação Geográfica

SQL - *Structured Query Language*

TIN - *Triangulated Irregular Networks*

XML - *Extensible Markup Language*

UML - *Unified Modelling Language*

CAPÍTULO 1

INTRODUÇÃO

1.1 Sistemas de informação geográfica

Dados geograficamente referenciados, ou simplesmente dados geográficos, são aqueles que possuem uma dimensão espacial, ou uma localização, diretamente ligada ao mundo geográfico real como as imagens de satélites de sensoriamento remoto, os dados de inventários cadastrais, os dados ambientais coletados em campo e os modelos numéricos de terreno. Os sistemas de informação geográfica (SIGs) são sistemas computacionais capazes de capturar, modelar, armazenar, recuperar, manipular, analisar e apresentar dados geográficos (Worboys; Duckham, 2004).

A disciplina chamada de *geoinformática* trata da perspectiva computacional dos SIGs e abrange tópicos como modelagem espacial, bancos de dados geográficos, interfaces homem-máquina para SIG, cartografia digital, processamento digital de imagens de sensoriamento remoto e processamento de dados espaço-temporais. A geoinformática vale-se das ferramentas computacionais mais avançadas, aplicadas de maneira particular ao domínio dos dados geográficos, para desenvolver SIGs capazes de atingir o grau de sofisticação necessário à manipulação de dados geográficos. Do ponto de vista da tecnologia, desenvolver um SIG significa oferecer o conjunto mais amplo possível de estruturas de dados e algoritmos capazes de representar a grande diversidade de concepções do espaço geográfico. Para isso, o espaço geográfico, conforme percebido pelos usuários dos SIGs, deve ser traduzido para os sistemas computacionais. Uma abordagem para a solução desse problema é a adaptação do paradigma dos quatro universos para a ciência da geoinformação (Câmara, 2005). Esse paradigma distingue quatro passos entre o mundo real e sua representação computacional, como mostra a Figura 1.1.

Resumindo de Câmara (2005) o paradigma dos quatro universos, o primeiro universo refere-se a definição dos conceitos diretamente extraídos da realidade; no segundo

distingue-se os modelos lógicos ou construções matemáticas que generalizam os conceitos do universo ontológico; no terceiro os modelos formais são mapeados para estruturas geométricas e alfanuméricas, e algoritmos que realizam operações; finalmente, no universo de implementação estão as questões de arquitetura, linguagens e paradigmas de programação.

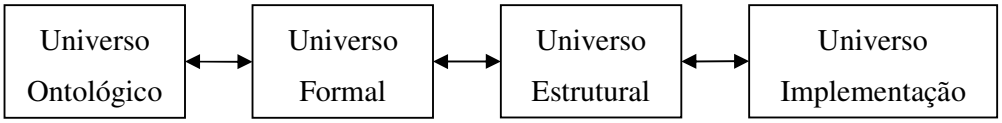


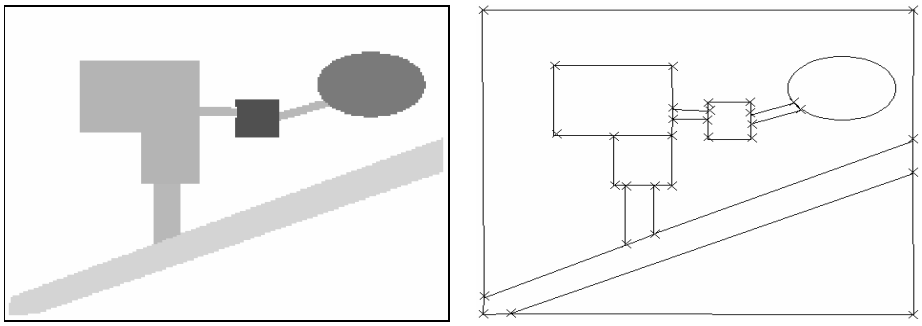
Figura 1.1 – Paradigma dos quatro universos.
FONTE: Adaptado de Câmara (2005).

No universo formal, dois modelos de representação conceitual dos dados geográficos são aceitos: o modelo de geo-campos e o modelo de geo-objetos (Couclelis, 1992; Goodchild, 1992), descritos na seção seguinte.

1.2 Modelos para representação de dados geográficos

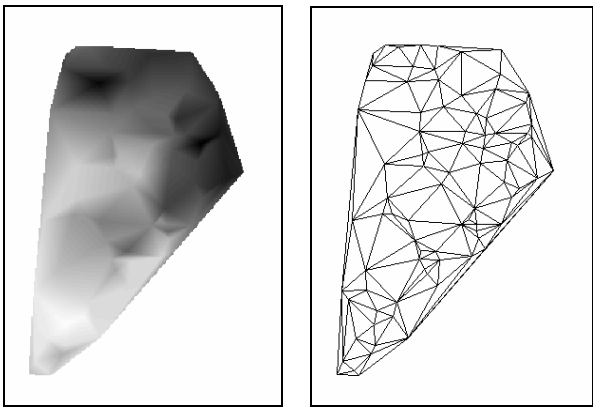
Segundo Câmara (2005), o modelo de geo-campos trata o espaço geográfico como uma superfície contínua sobre a qual variam os fenômenos a serem observados, enquanto que o modelo de geo-objetos representa o espaço geográfico como uma coleção de entidades distintas e identificáveis, onde cada entidade é definida por uma fronteira fechada. De fato, a noção de fronteira diferencia os dois modelos: nos geo-campos as fronteiras são definidas por limitações decorrentes do processo de aquisição, enquanto que nos geo-objetos as fronteiras caracterizam os próprios geo-objetos.

Geo-campos e geo-objetos são mapeados para estruturas de dados de duas naturezas: *vetorial* e *matricial* ou *raster*. A Figura 1.2 ilustra estruturas matriciais e vetoriais para representar um conjunto de geo-objetos: duas construções e um lago próximo a uma estrada. A Figura 1.3 mostra estruturas vetoriais e matriciais para representar um geo-campo: a altimetria de uma região.



(a) matricial (b) vetorial

Figura 1.2 – Representação de geo-objetos: (a) matricial e (b) vetorial.



(a) matricial (b) vetorial

Figura 1.3 – Representação de geo-campos: (a) matricial e (b) vetorial.

Ainda que as duas representações conceituais do dado geográfico tenham, em vários casos, levado à especialização dos SIGs para tratar um dentre os dois modelos de representação, essa escolha não deve ser apenas tecnológica, mas sim decorrente de uma visão conceitual sobre qual a melhor representação do espaço geográfico (Couclelis, 1992). Também existem SIGs híbridos, ou seja, aqueles capazes de tratar dados modelados de acordo com as duas representações. Normalmente, esses SIGs permitem a integração dos dois modelos através de operações que combinam dados de diferentes modelos, ou permitem converter dados de um modelo para outro.

1.3 O desafio de desenvolver inovações em SIGs

A partir das escolhas feitas em cada universo, diferentes abordagens quanto ao modelo de desenvolvimento de SIGs vem sendo propostas, como as orientadas-a-objeto (Câmara, 1995; Egenhofer; Frank, 1992), através de especificações algébricas (Frank; Kuhn, 1995; Winter; Nittel, 2003), ou as baseadas em ontologias (Fonseca; Egenhofer, 1999; Fonseca *et al.*, 2002). Essas abordagens estão mais ligadas ao universo ontológico e formal. Porém especificações e modelos de desenvolvimento mais próximos do universo estrutural e de implementação são necessárias para ajudar os desenvolvedores de SIGs a lidar com questões como eficiência e flexibilidade no projeto de seus sistemas.

Uma iniciativa nesse sentido é a do consórcio OGC – *Open Geospatial Consortium* (Kottman, 1990), criado para promover o desenvolvimento de tecnologias que facilitem a interoperabilidade entre sistemas envolvendo informação espacial e localização. Os produtos do trabalho do OGC são apresentados sob forma de especificações de interfaces e padrões de intercâmbio de dados (Davis Jr. *et al.*, 2005).

Pode-se dizer que existe um avanço maior na proposição de estruturas de dados e algoritmos para materializar o modelo conceitual de geo-objetos do que de geo-campos. Sua representação computacional é geralmente baseada em estruturas de dados vetoriais, sobre as quais já existem um conjunto bem aceito de operadores, entre os quais os operadores topológicos (Egenhofer; Franzosa, 1991). Propostas de modelos de armazenamento e especificações de consultas já foram materializadas em extensões à linguagem SQL – *Structured Query Language* e aos SGBD – Sistemas Gerenciadores de Bancos de Dados comerciais e de domínio público. Uma revisão de alguns desses sistemas pode ser encontrada em Queiroz e Ferreira (2005).

Quanto ao modelo conceitual de geo-campos, as estruturas de dados, modelos de armazenamento e consulta, e as especificações de implementação ainda não estão tão bem estabelecidas. Normalmente, geo-campos representam grandes volumes de dados e é comum que sejam representados tanto por estruturas geométricas vetoriais quanto matriciais. Existe a necessidade da criação de especificações de implementação que

possam ser estendidas para independer da estrutura geométrica. Também é necessário tratar a questão do armazenamento de representações para geo-campos em SGBDs.

Assim, torna-se necessário dar um passo além das especificações abstratas, ou mesmo de implementação e criar sistemas completos ou semi-completos, onde as especificações possam ser revisadas ou até mesmo melhoradas tanto em termos de funcionalidades quanto em termos de volume de dados. Nessa linha, a DPI – Divisão de Processamento de Imagens, do INPE – Instituto Nacional de Pesquisas Espaciais, criou com o projeto TerraLib, uma biblioteca para a construção de aplicativos geográficos (Câmara *et al.*, 2000; Vinhas; Ferreira, 2005). A TerraLib possui código fonte aberto e é distribuída sob a licença LGPL – *Lesser GNU Public License*, via Web no site www.terralib.org. A TerraLib é escrita na linguagem C++ padrão (Stroustrup, 1997), podendo ser compilada nos sistemas operacionais Linux e MS-Windows.

O objetivo do projeto TerraLib é produzir um produto, sem custo, que possa servir como base para o desenvolvimento cooperativo da comunidade de desenvolvedores de SIGs, e que ofereça ferramentas tecnológicas avançadas, muitas vezes ainda não disponíveis nas soluções proprietárias de mercado. Por isso é uma boa alternativa para a prototipação e validação de novas técnicas resultantes da pesquisa e desenvolvimento em ciência da geoinformação, além da criação de aplicações comerciais.

Dadas as características apresentadas acima considera-se a TerraLib como o ambiente adequado para implementar o subsistema descrito nessa tese pelas seguintes razões: ser livre de licença e ter código fonte aberto; ser desenvolvida em linguagem C++ que suporta diversos estilos de programação como orientação-a-objeto ou parametrização de tipos, os quais são necessários para a criação de soluções extensíveis; trabalha com a arquitetura integrada que está diretamente ligada a um dos objetivos dessa tese; finalmente por ser uma biblioteca de componentes e não um único produto final.

1.4 Objetivo do trabalho

Esta tese propõe um subsistema para manipulação de representações matriciais de geo-campos e trata, em particular, da questão do armazenamento de representações

matriciais para geo-campos em SGBDs objeto-relacionais. A tese inclui ainda algumas aplicações concretas utilizando o subsistema.

O subsistema proposto está inserido no contexto da criação de ferramentas para a construção de aplicações geográficas que sejam *open source*. Nesse modelo de produção de sistemas, é fundamental resolver a questão de como fazer com que uma comunidade de desenvolvedores de fato utilize o sistema proposto, resolvendo as dificuldades inerentes ao compartilhamento dos conceitos que o sistema utiliza (Câmara; Fonseca, 2006). Diferentemente de bibliotecas de funções, um subsistema quase completo, que indica os pontos onde pode ser estendido e como pode ser utilizado, é uma alternativa para compartilhar as melhores práticas e decisões encontradas durante o desenvolvimento de um sistema, complementando as especificações abstratas, como as propostas pelo OGC.

Um dos requisitos de projeto da TerraLib foi investir no modelo de arquitetura integrada, ou seja, a utilização de SGBDs relacionais e objeto-relacionais para armazenamento dos dados geográficos em sua componente espacial e convencional. Essa integração é o que permite o compartilhamento de bases de dados, em ambientes corporativos, onde diversas aplicações, customizadas segundo interesses particulares acessam a mesma base. Essa tese estende o modelo de armazenamento de geo-objetos já existente na TerraLib para incluir as representações de geo-campos. Inicialmente trabalha-se com as representações matriciais de geo-campos, porém é o passo inicial no sentido de acrescentar o suporte também as representações vetoriais de geo-campos.

1.5 Organização do documento

Esta tese possui mais quatro capítulos, conforme descrito abaixo:

Capítulo 2: revisa o conceito de geo-campo e suas diferentes representações e formatos de armazenamento, e mostra a especificação OGC para geo-campos;

Capítulo 3: descreve a arquitetura de subsistema extensível para manipulação de geo-campos, que armazena e recupera grandes volumes de dados de forma eficiente. Este

subsistema endereça os problemas relativos ao tratamento de representações matriciais para geo-campos e, em especial, o caso de seu armazenamento em SGBDs.

Capítulo 4: trata da questão do armazenamento de representações matriciais para geo-campos em SGBDs objeto-relacionais;

Capítulo 5: mostra alguns resultados da utilização do subsistema proposto dentro da biblioteca TerraLib;

Capítulo 6: apresenta algumas conclusões e questões relativas a continuação do trabalho.

Cabe salientar que comparações com trabalhos relacionados serão feitas ao longo do texto, especialmente na seção 4.2, que trata dos principais requisitos que devem ser considerados na proposição de um modelo de armazenamento das representações matriciais para geo-campos.

CAPÍTULO 2

DADOS GEOGRÁFICOS DO TIPO GEO-CAMPOS

2.1 Geo-campos e representações de geo-campos

Um *campo geográfico*, ou *geo-campo*, é uma função $f : R \rightarrow A$ tal que $R \subset \Re^2$ é uma região conexa do $\Re^2$, chamada de *domínio* do geo-campo, e A é um conjunto qualquer, chamado de *conjunto de valores* ou *contradomínio* do geo-campo. A região R deve estar definida em um *sistema de localizações geo-referenciado*, definido como um sistema de coordenadas com o qual é possível estabelecer uma relação com pontos da superfície da Terra. De fato, Kemp e Vckovski (1998) argumentam que uma função serve como um modelo explícito, preciso e representativo para campos geográficos.

O contradomínio A do geo-campo pode ser classificado de acordo com o tipo de medida que expressa: *nominal,* onde não existe nenhuma restrição sobre os valores, sendo esses apenas rótulos para os elementos do domínio (normalmente representados por textos ou números inteiros); *ordinal,* onde existe uma relação de ordem entre os valores; *intervalar,* para valores estritamente quantitativos dependentes de um ponto zero arbitrário, ou *razão* (Stevens, 1946) com valores qualitativos que independem de um zero arbitrário. O autor propôs essa classificação a fim de caracterizar o conjunto de operações (matemáticas, lógicas ou de comparação) que fazem sentido sobre essas medidas.

Na prática, um geo-campo possui uma aproximação finita tratável em sistemas computacionais, que por natureza são discretos (Goodchild, 2003). Representações computacionais de geo-campos são tipicamente aproximações no sentido de indicarem o valor (ou uma aproximação do valor) do geo-campo apenas para um subconjunto finito do domínio do geo-campo. Considerando esse subconjunto do domínio como formado por elementos geométricos bidimensionais (células, linhas, triângulos ou polígonos, por exemplo), as representações comumente encontradas nos SIGs são:

1) *grades regulares* (ou *raster*): o geo-campo é representado por um conjunto de células retangulares onde um único valor é atribuído a cada célula, representando o valor do geo-campo na extensão da célula;

2) *pontos amostrais*: o geo-campo é representado apenas em localizações pontuais conhecidas, como estações de medição ou pontos de coleta no campo;

3) *isolinhas*: o geo-campo é representado por linhas ao longo das quais o valor do geo-campo é constante;

4) *subdivisões planares*: especialmente quando o conjunto de valores do campo é finito (por exemplo, os possíveis tipos de solo), o geo-campo é normalmente representado por um conjunto de áreas que não se interceptam e que recobrem todo o domínio do geo-campo. Nesse caso assume-se que o valor do geo-campo é o mesmo dentro da extensão espacial de cada área;

5) *malhas triangulares* ou (*TINs*): representam o geo-campo por um conjunto de triângulos que não se sobrepõem e cobrem totalmente a área do geo-campo. São formados por conexões entre amostras do valor do atributo, com distribuição espacial possivelmente irregular, utilizando algum método de triangulação (de Floriani, 1987).

Goodchild (2003) argumenta que as representações 1, 4 e 5 podem ser chamadas de *completas*, uma vez que o valor do geo-campo pode ser consultado diretamente em qualquer localização presente na representação; as representações 2 e 3 podem ser chamadas de *incompletas* pois seus elementos formam um conjunto de localizações com valores que não estão explicitamente armazenadas, onde a consulta ao seu valor normalmente envolve a utilização de algum algoritmo de interpolação.

A escolha de uma determinada representação para um geo-campo pode ser baseada na natureza do fenômeno correspondente, ou ainda decorrente da forma com a qual o geo-campo é medido. Por exemplo, os TINs são apropriados para representar geo-campos correspondentes a altimetria uma vez que melhor aproximam a superfície do terreno,

preservando características particulares do relevo (Figura 2.1). Por outro lado, o modo de funcionamento dos instrumentos de aquisição a bordo de satélites de sensoriamento remoto produz imagens digitais, que são a representação em grades regulares de uma medida: a reflectância associada à porção do terreno sendo observada ou imageada. Já geo-campos que representam medidas ambientais, como valores de temperatura, são obtidos normalmente através de medições em estações de coleta de dados, distribuídas irregularmente sobre uma região, por isso uma representação adequada é a de pontos amostrais. Cada tipo de representação traz suas próprias características, que vão desde formatos particulares de armazenamento em arquivos até maneiras mais eficientes de armazenamento em SGBDs.

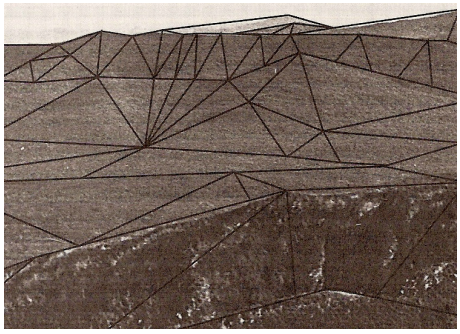
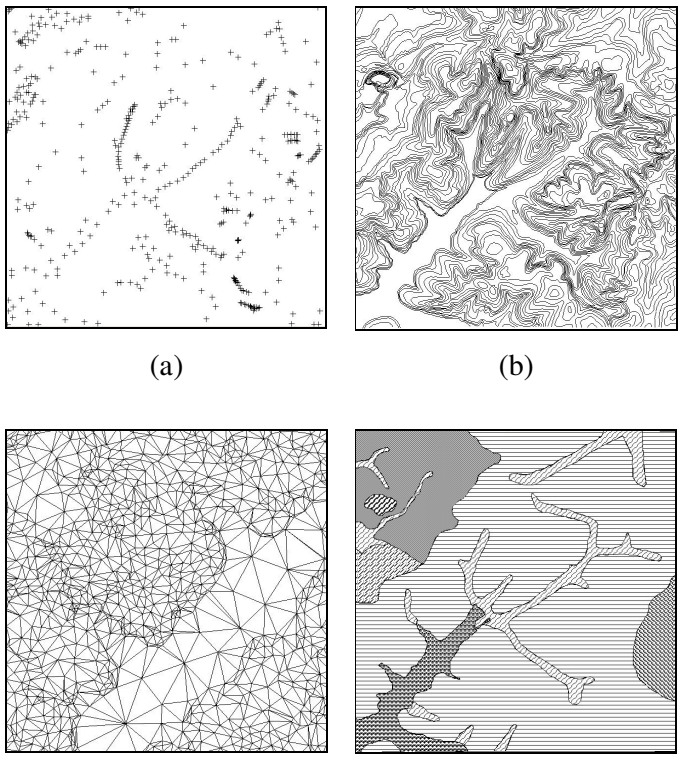


Figura 2.1 – Um TIN sobreposto à fotografia de um terreno.
FONTE: Adaptado de Chrisman (1999).

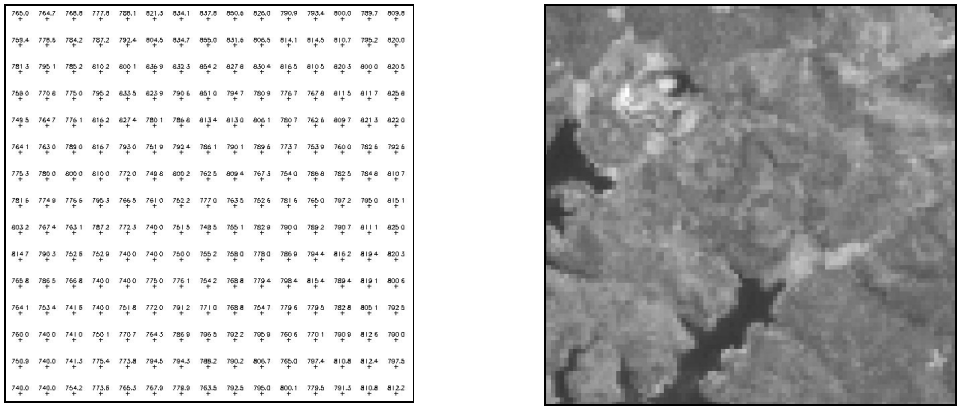
As representações formadas por pontos amostrais, isolinhas, subdivisões planares e TIN são constituídas por estruturas geométricas *vetoriais* (pontos, linhas e polígonos) que armazenam explicitamente os valores das coordenadas que as formam (Figura 2.2). As grades regulares são representadas por estruturas geométricas matriciais, ou *raster*, onde as posições de cada localização dentro da representação não estão explicitamente armazenadas, mas são estabelecidas implicitamente a partir de uma ordem pré-definida dentro da representação (Figura 2.3). As grades regulares serão discutidas em mais detalhe na seção 2.3.



(a) (b)

(c) (d)

Figura 2.2 – Representações vetoriais para geo-campos: (a) pontos amostrais; (b) isolinhas; (c) TIN e (d) divisão planar.



(a) (b)

Figura 2.3 – Representações matriciais para geo-campos: (a) grade regular e (b) imagem.

2.2 Geo-campos e o *Open Geospatial Consortium*

O *Open Geospatial Consortium* – OGC é um consórcio de empresas e instituições governamentais e acadêmicas com o objetivo de promover interoperabilidade entre sistemas envolvendo informação espacial. O OGC define o chamado modelo OpenGIS, um padrão de interoperabilidade para a componente espacial (geometrias) de dados geográficos e também para serviços que utilizam esses dados (Kottman, 1990).

Em uma primeira fase, o OGC especifica o modelo de geometrias e o modelo de serviços sem levar em consideração quaisquer detalhes de implementação, o que resulta em um documento chamado *especificação abstrata*. Numa segunda fase, partes bem definidas da especificação abstrata dão origem a *especificações de implementação,* que sugerem como os conceitos abstratos podem ser implementados usando tecnologias como SQL, Java, XML ou CORBA.

A especificação abstrata para o modelo de geometrias do OGC, orientada a objetos na linguagem UML – *Unified Modelling Language*, define uma classe abstrata chamada *feature*, considerada uma abstração de um fenômeno do mundo real. Mais precisamente, uma *feature* ou *feição geográfica* é um objeto associado a uma localidade relativa à superfície terrestre. A classe *feature* possui duas especializações: *feature with geometry*, relacionado ao conceito de geo-objetos, e *coverage*, relacionado ao conceito de geo-campos (OGC, 2005).

A especialização *coverage* possui uma propriedade chamada de *coverage_function,* cujo valor é uma função, *C_function*, que possui como domínio uma região do espaço e como contradomínio um conjunto qualquer de valores, incluindo conjuntos de vetores n-dimensionais. A Figura 2.4 mostra a classe *coverage*, representando os geo-campos e as suas especializações, conforme definido pelo OGC.

Quanto às propriedades das *coverages*, a especificação também define que a função *C_Function* possui subtipos particulares para cada uma das representações de geo-campos ou, na linguagem da especificação, a cada suporte espacial. Esses subtipos são: *PointC_Function*, *LineStringC_Function* e *SurfaceC_Function*. Os subtipos mostrados

não fazem restrição nem quanto à cardinalidade do suporte espacial nem quanto ao conjunto de valores da função. Quando o domínio da função é considerado finito, a especificação também define os subtipos *DiscretePointC_Function*, *DiscreteLineStringC_Function* e *DiscreteSurfaceC_Function*.

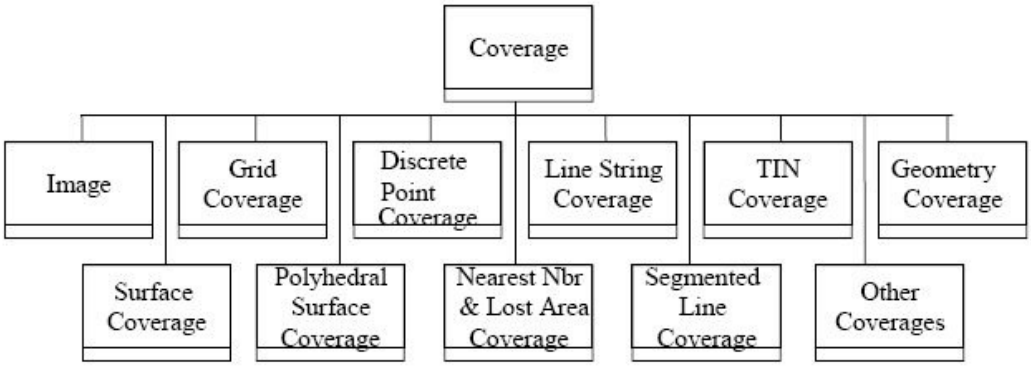


Figura 2.4 – A classe *Coverage* conforme proposta pelo OGC.
FONTE: *Open Geospatial Consortium* (OGC, 1999).

A especificação define que a classe *coverage* deve fornecer três interfaces principais:

1. *avaliação*: representa, de maneira genérica, os métodos pelos quais uma *coverage* pode ser consultada. Tipicamente, um método de avaliação aceita como parâmetro um elemento do domínio espacial e retorna o valor da *coverage* associado (ou calculado) para aquele elemento;

2. *avaliação inversa*: dado um conjunto de valores como parâmetros os métodos de avaliação inversa retornam os pontos do domínio espacial que possuem como valores dentro desse conjunto;

3. *serviços*: devem poder ser criados utilizando a *coverage*. Esses serviços incluem: a observação de uma ou mais propriedades em um ou mais pontos dentro do domínio espacial; a exportação de metadados sobre os tipos dos valores e a extensão espacial da *coverage*; estimativas de qualidade dos valores

representados na *coverage* e informações sobre as funções de interpolação disponíveis.

Essas interfaces não são descritas em mais detalhes, deixando essa tarefa a cargo das especificações de implementação. O OGC fornece uma proposta de especificação de implementação para a classe *grid coverage* (OGC, 2001) voltada para implementações em tecnologia COM e CORBA. A arquitetura da especificação é composta por três pacotes: um pacote geral para a classe *coverage* (CV), um pacote específico para a classe *grid coverage* (GC) e um pacote para o processamento de *grid coverages* (GP) (ver Figura 2.5). Esses pacotes especificam basicamente interfaces para a obtenção dos valores do geo-campo em uma determinada localização, a obtenção de metadados sobre sua representação e uma interface comum para algoritmos de processamento.

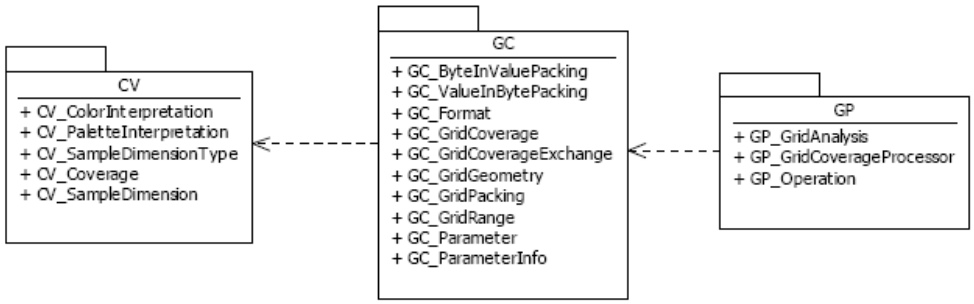


Figura 2.5 – A arquitetura da especificação de implementação das *grid coverage.*
FONTE: Open Geospatial Consortium (OGC, 2001).

A suposição básica sobre a classe *grid coverage* é a de que algum fenômeno relativo à Terra foi observado sobre os vértices de uma grade. Essas observações são um conjunto de valores que formam uma matriz de valores (uni- ou multidimensionais), organizados de alguma forma.

Uma especificação abstrata para geo-campos é bastante complexa e a abordagem adotada pelo OGC já sofreu algumas críticas. Davis Jr. *et al*. (2005) ressaltam que o conjunto de especializações definido para *coverage* é incoerente com a noção funcional a ela atribuída (toda *coverage* possui o atributo *C_Function*). Além disso, induz a que o conteúdo semântico do dado geográfico modelado seja confundido com o seu conteúdo

sintático, ou seja, com a sua representação. Em Winter e Nittel (2003), os autores apresentam uma versão funcional, em linguagem HASKELL, da especificação da classe *coverage*, mais especificamente da especilização *grid coverage*, com o objetivo de comparar a expressividade e aplicabilidade das duas ferramentas de especificação. Os autores concluem que especificações feitas através de linguagens funcionais são mais formais, pois são capazes de expressar regras com consistência semântica. Como a especificação em linguagem funcional é um programa, que pode ser executado e testado, os conceitos definidos na especificação podem ser teoricamente provados.

O OGC especifica uma forma de armazenamento de representações vetoriais de geo-objetos em SGBDs que inclui uma hierarquia de tipos de geometrias vetoriais, operações topológicas e métricas e um esquema de tabelas para o armazenamento de metadados das informações espaciais. Especifica ainda uma extensão da linguagem SQL que permite expressar consultas e manipulação desse tipos (OGC, 2005). Para as *coverages*, essa especificação de armazenamento, operadores ou linguagens de consulta ainda não foi definida.

Nessa tese optou-se por caracterizar a manipulação dos geo-campos através da construção de um subsistema, sobre o qual possam ser implementadas as interfaces previstas na interface da classe *coverage*, prevendo o seu armazenamento em SGBDs e que sirva como um *framework* que facilite a extensão para *coverages* com outras representações que não apenas grades regulares.

2.3 Representações matriciais para geo-campos

2.3.1 Definição de representação matricial

Uma representação matricial de um geo-campo é uma quádrupla:

$$RM = (S, V, M[m,n], T_M[2,3]) \tag{2.1}$$

onde:

S é um sistema de referência espacial plano, associado a um sistema de projeção cartográfica, ou seja, um sistema capaz de mapear localizações sobre a superfície terrestre, chamadas coordenadas geográficas, em uma pontos de uma superfície plana, chamadas coordenadas de projeção;

V é um conjunto qualquer;

$M[m,n]$ é uma matriz regular com *m* linhas e *n* colunas, onde *m* e *n* são inteiros positivos, tomando valores em *V* (os elementos da matriz serão identificados começando em $M(0,0)$ e não em $M(1,1)$);

T_M é uma matriz 2×3 que define os coeficientes de uma matriz de transformação entre as coordenadas linha e coluna de $M[m,n]$e as coordenadas do sistema de referência espacial:

$$T_M = \begin{bmatrix} R_X & Ro_X & X_0 \\ Ro_Y & -R_Y & Y_0 \end{bmatrix} \tag{2.2}$$

onde, R_X e R_Y são números reais, chamados *resolução horizontal* e *resolução vertical* de *RM*; Ro_X e Ro_Y são *fatores de rotação* em relação ao eixos *X* e *Y* respectivamente; e X_0 e Y_0 são números reais, as *coordenadas de origem* de *RM*. Para todo par de coordenadas (c,l) da representação matricial onde $0 \le c < m$ e $0 \le l \le n$, a sua coordenada espacial (x, y) é dada por:

$$\begin{bmatrix} x \\ y \end{bmatrix} = T_M \times \begin{bmatrix} c \\ l \\ 1 \end{bmatrix} \tag{2.3}$$

ou:

$$x = R_X c + Ro_X l + X_0 \tag{2.4}$$

e

$$y = Ro_Y c - R_Y l + Y_0 . \tag{2.5}$$

A transformação inversa é dada por:

$$c = \frac{-R_Y (x - X_0) - Ro_X (l - Y_0)}{Ro_Y Ro_x - R_X R_Y} \tag{2.6}$$

$$l = \frac{Ro_Y (x - X_0) - R_X (l - Y_0))}{Ro_Y Ro_x + R_X R_Y} . \tag{2.7}$$

Uma *célula* de *RM* é um retângulo $Q[i, j] \subseteq \Re^2$, onde $i \in [0, m)$ e $j \in [0, m)$, definido pelo conjunto de pontos $(x, y) \in \Re^2$ tais que $x \in [X_0 + j * R_X, X_0 + (j+1) * R_X)$ e $y \in [Y_0 + iR_Y, Y_0 + (i+1) * R_Y)$.

Uma representação matricial *RM* define um geo-campo $g : Q \to V$ onde $Q \subseteq \Re^2$ é o retângulo definido pelo conjunto de pontos $(x, y) \in \Re^2$ tais que: $x \in [X_0, X_0 + n * R_X)$ e $y \in [Y_0, Y_0 + m * R_Y)$ e $g((x, y)) = M(i, j)$ onde $i = \lfloor (x - X_0) / R_X \rfloor$ e $j = \lfloor (y - Y_0) / R_Y \rfloor$, para $(x, y) \in Q$ (a função $\lfloor t \rfloor$ representa o maior inteiro menor do que *t*). O valor de um elemento da representação *RM* é referenciado por $RM(i, j)$.

2.3.2 Características espaciais da representação

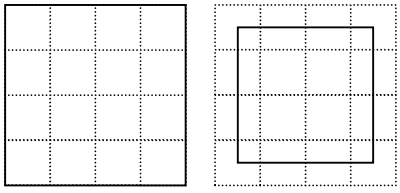
O processo de projeção da superfície terrestre para uma superfície plana é sempre uma aproximação, que carrega algum tipo de distorção inerente a esse processo, por isso existem diferentes projeções adequadas para diferentes aplicações (Snyder, 1987). De maneira geral uma projeção cartográfica define um sistema de equações que permite mapear coordenadas planas em coordenadas geográficas e vice-versa. Os sistemas de coordenadas geográficas definem localizações sobre a intersecção entre paralelos, ou latitudes, e meridianos, ou longitudes. Os paralelos são círculos cujo plano é perpendicular ao eixo dos polos da Terra e os meridianos são círculos ou elipses que contém o eixo de rotação dos polos da Terra. A figura da Terra é aproximada por uma

forma matematicamente tratável, um elipsóide de revolução apropriadamente posicionado em relação ao centro da Terra, chamado Datum Planimétrico (d´Alge, 2004);

Intuitivamente, a matriz *M[m,n]* representa uma região do espaço dividida em células, associadas a uma porção da superfície terrestre através do sistema de referência espacial *S*. As resoluções horizontal e vertical definem a relação entre a extensão de cada elemento da célula e a porção do terreno ao qual está associada. O número de linhas e colunas e a matriz de transformação T_M definem a extensão geográfica do dado matricial, também chamado de *retângulo envolvente*.

Como cada elemento de uma representação matricial possui área, definida pela resolução, muitas vezes é útil identificar não somente o retângulo envolvente, mas também o *retângulo envolvente central*, ou seja, aquele que passa pelo centro de cada célula da representação como mostra a Figura 2.6.

O retângulo envolvente e as resoluções são definidos na mesma unidade da projeção cartográfica dada por *S*. Esses parâmetros são interdependentes, ou seja, cada um deles sempre pode ser inferido a partir dos outros dois. As Figuras 2.7a, 2.7b e 2.7c mostram como a variação dos parâmetros resolução, número de linhas e colunas e coordenada inicial dão origem a três *RM* diferentes.



(a) (b)

Figura 2.6 – Extensão de uma representação matricial: (a) retângulo envolvente e (b) retângulo envolvente central.

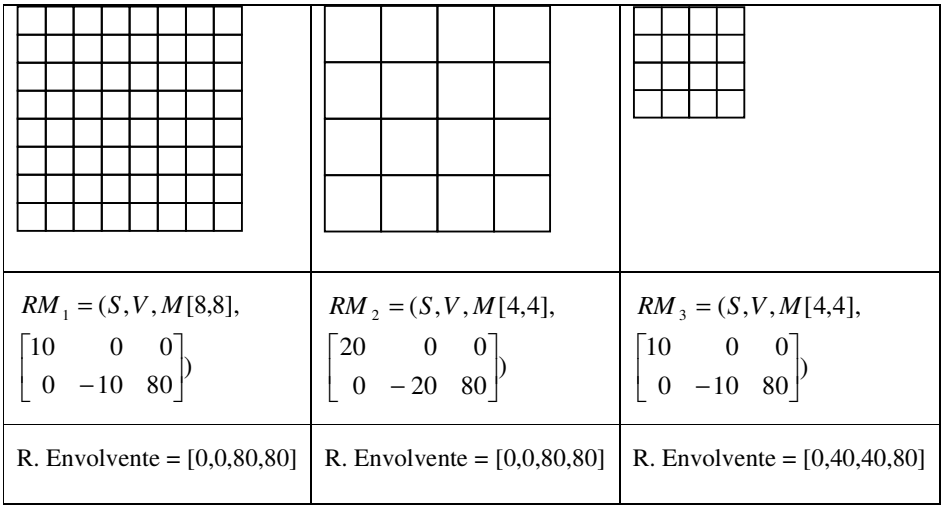
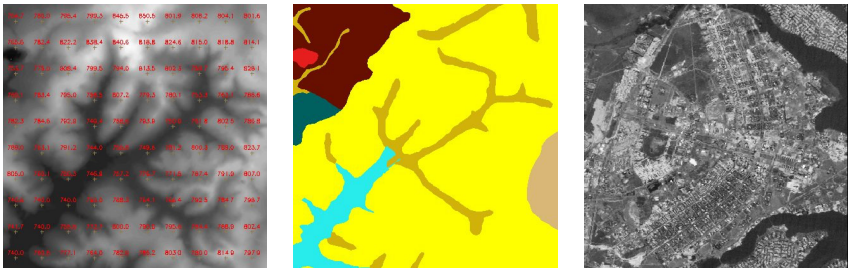


Figura 2.7 – Três representações matriciais diferentes e seus parâmetros: (a) representação original; (b) mesma extensão, resolução diferente e (c) mesma resolução e número de linhas e colunas diferentes.

2.3.3 Características de contradomínio da representação

A Figura 2.8 mostra três exemplos de representações matriciais associadas a diferentes geo-campos com contradomínios unidimensionais associados a diferentes tipos de medida. A Figura 2.8a mostra uma grade de altimetria do terreno, medida numa escala de intervalo cujos valores estão dentro do conjunto dos números reais e cujo zero arbitrário é o nível do mar. A Figura 2.8b mostra uma medida nominal, cujos valores estão associados a classes, ou tipos de solo, as quais estão identificadas por cores. Finalmente, a Figura 2.8c mostra uma imagem de sensoriamento remoto monobanda, medida em uma escala de razão, com valores que estão no intervalo de 0 a 255.



(a) (b) (c)

Figura 2.8 – Exemplo de diferentes contradomínios em representações matriciais: (a) altimetria; (b) mapa de solos e (c) imagem de sensoriamento remoto.

No caso mais geral, V é multidimensional, ou seja, é um conjunto de k-tuplas $\boldsymbol{v} = (v_1, v_2, \ldots, v_k)$, onde $v_i \in A_i$, ou seja, os elementos pertencem a contradomínios particulares, não necessariamente distintos; e k é o número de dimensões do contradomínio do geo-campo. Um exemplo típico de representações multidimensionais onde $A_i = A$, para $i = 1,...,k$ ou seja, todas as dimensões possuem o mesmo contradomínio particular, é o caso das imagens de sensoriamento remoto, que podem ser compostas por um conjunto de bandas espectrais, onde cada banda representa o valor da reflectância medido em uma determinada faixa do espectro eletromagnético. Cada banda pode ser visualizada ou processada independentemente, porém, é necessária a preservação do conceito de que todas as bandas pertencem a uma mesma imagem (Figura 2.9). O geo-campo é a função que mede a reflectância, e cada banda é uma dimensão do contradomínio.

Um exemplo de representações multidimensionais onde os contradomínios particulares podem ser diferentes é o caso dos espaços celulares (Batty, 2000). Espaços celulares são estruturas de dados usadas na construção de modelos de simulação dinâmica em diferentes áreas. Servem para representar fenômenos espaciais complexos onde, a cada localização do espaço é associado um conjunto de atributos diferentes, ou diferentes tipos de medidas, observados sobre uma mesma referência espacial. A Figura 2.10 mostra um exemplo de espaço celular, onde um dos atributos nominais de cada célula está mostrado em forma de imagem com uma graduação de cores, os outros atributos

estão mostrados em forma de tabela na parte inferior da figura. Segundo Carneiro (2004) ferramentas de *software* para modelagem dinâmica cuja arquitetura seja fortemente integrada aos SIGs são de grande importância para a comunidade de modeladores. A caracterização de espaços celulares como representações matriciais de geo-campos com contradomínios particulares diferentes, de forma a inserí-los em um subsistema capaz de armazenar e manipular geo-campos armazenados em bancos de dados geográficos, é mais uma contribuição para a integração entre os SIGs e as ferramentas de modelagem.

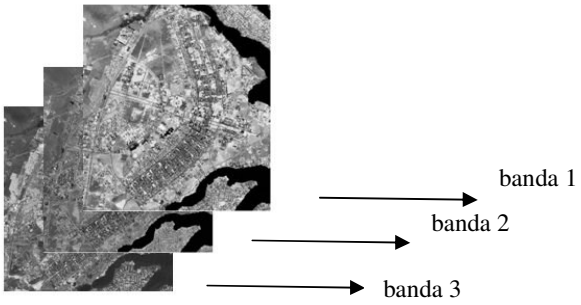
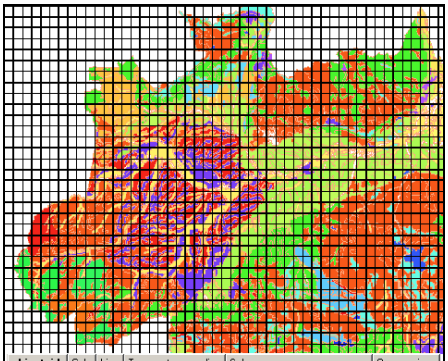


Figura 2.9 – Diferentes bandas de uma mesma cena de imagem.



object_id	Col	Lin	Temperatura_media	Solo	Ocorrencias
C00L00	0	0	27.3	Gleissolos Salicos Orticos	1
C00L01	0	1	28.5	Gleissolos Salicos Orticos	3
C00L02	0	2	21.4	Nitossolos Haplicos Distroficos	0
C00L03	0	3	22.4	Organossolos Mesicos Hemicos	1
C00L04	0	4	27.7	Vertissolos Cromicos Orticos	4
C00L05	0	5	29.8	Organossolos Mesicos Hemicos	2
C00L06	0	6	28.5	Vertissolos Cromicos Orticos	5

Figura 2.10 – Exemplo de um espaço celular.

Os contradomínios particulares A_i podem ser mapeados diretamente para os tipos computacionais usados para armazenar as representações. Por exemplo, a representação

unidimensional mostrada na Figura 2.8a utiliza um tipo `float` para representar o valor do geo-campo; a representação unidimensional mostrada na Figura 2.8b usa o tipo `unsigned char`, que é suficiente, se a representação estiver associada a uma legenda que forneça o conteúdo semântico dos valores qualitativos; para a representação mostrada na Figura 2.8c, é suficiente usar um tipo `unsigned char`.

Por definição uma representação matricial não é esparsa, ou seja, toda célula da matriz possui um valor. Porém, podem existir células nas quais o valor do geo-campo não foi anotado. A Figura 2.11 mostra o exemplo de um geo-campo de temperatura média produzido somente dentro do limite político do Brasil. O dado é representado na forma de imagem em tons de cinza, representada em 8 bits, onde o menor valor do geo-campo é mapeado para 0 (o preto) e o maior valor para o valor 254 (próximo do branco). O contorno da imagem representa a extensão total, ou o retângulo envolvente da representação. Uma forma de se representar, que fora do limite político, não se conhece o valor de temperatura média é através da eleição de um determinado valor, dentro do contradomínio do geo-campo, para ser um indicativo da ausência de informação em certas localizações da representação matricial. No exemplo, o contradomínio do geo-campo é o conjunto $\Re$ dos números reais e o valor 999 indica ausência de informação, que foi mapeado para o valor 255 (o branco).

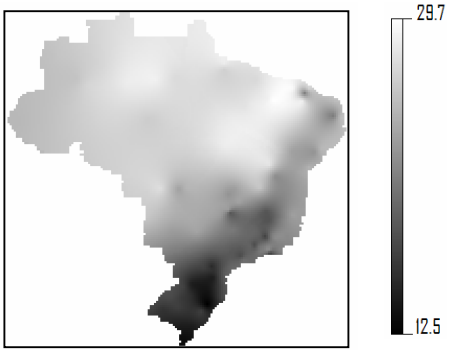


Figura 2.11 – Exemplo de uma representação matricial esparsa.

2.3.4 Formatos de representações matriciais

As representações matriciais podem estar armazenadas em diferentes formatos, com mais ou menos informações sobre a representação. Dentre os formatos mais comuns para representações matriciais de dados geográficos podem ser citados o *Tiff*, *GeoTiff*, *JPEG, JPEG2000* e *raw*.

O *Tiff – Tagged Image File Format* foi desenvolvido para ser o formato padrão de imagens no ambiente comercial. É um formato binário, onde um sistema de marcadores (ou *tags*) é usado para armazenar informações básicas sobre a imagem que contém. Existem marcadores básicos que todos os sistemas devem reconhecer, onde são registradas informações como o número de linhas e de colunas, o número de componentes e número de *bits* por *pixel* da imagem.

Uma alternativa para incluir informações de georeferenciamento, ou seja, de localização de imagens em formato *Tiff* sobre a superfície da terrestre, consiste na criação de arquivos de navegação em coordenadas geográficas, que possuem um formato padrão, com o mesmo nome do arquivo *Tiff*, mas com a extensão ".tfw" ("*tiff world file*"). Esse arquivo é codificado em formato ASCII e possui 6 linhas que descrevem os parâmetros necessários para o mapeamento entre as coordenadas de imagem (linha e coluna) para coordenadas do mundo (Figura 2.12). Deve-se notar que o arquivo de navegação não traz a informação sobre a qual sistema de projeção as coordenadas do mundo referem-se.

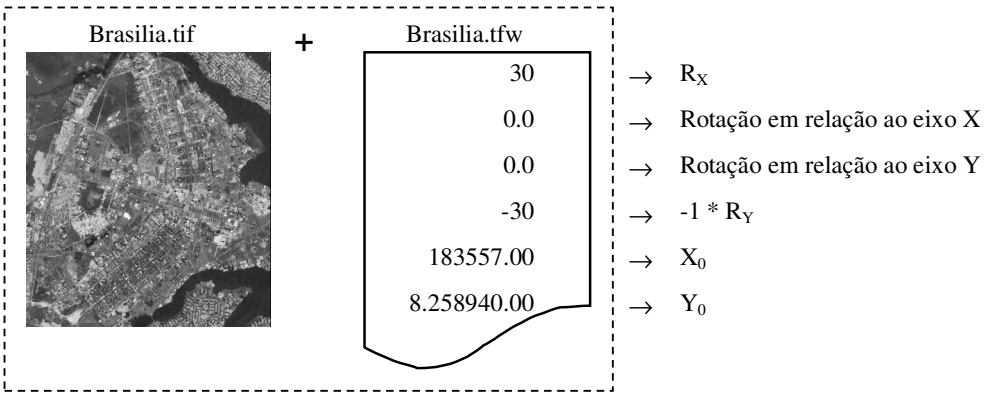


Figura 2.12 – Imagem em formato Tiff georeferenciada com arquivo de navegação e o descrição de seus parâmetros.

O GeoTiff é uma extensão do formato Tiff específica para o armazenamento de dados matriciais geográficos (Ritter; Ruth, 1997). Portanto, todas as informações básicas de um dado em formato Tiff são acessíveis no formato GeoTiff. A diferença é que a especificação do formato GeoTiff define um pequeno conjunto dos marcadores Tiff com o objetivo de descrever as informações cartográficas e geodésicas associadas a uma imagem Tiff. Dessa forma é possível descrever, no próprio formato, o sistema de projeção associado a uma imagem. O OGC propõe o GeoTiff como formato padrão de intercâmbio de dados matriciais (OGC, 1999).

O termo JPEG refere-se a um algoritmo de compressão de imagens digitais, criado pelo *Joint Photographic Experts Group*, de onde vem a sigla. Além disso, tornou-se sinônimo de um formato de armazenamento de imagens comprimidas por JPEG em arquivos que é especificado pelo grupo *Independent JPEG Group*. O JPEG é um algoritmo de compressão com perda, ou seja, uma imagem JPEG descomprimida não é idêntica à imagem original, mas pode ser aceitavelmente similar dependendo da taxa de compressão utilizada. O formato JPEG descreve as informações básicas sobre a imagem como o seu número de linhas e colunas. Como o formato foi proposto para o armazenamento de fotografias, ele é restrito a imagens monocromáticas (uma banda), ou a composições coloridas de três bandas, e suporta somente imagens onde o valor de cada pixel, de cada banda da imagem, está restrito ao intervalo de valores possíveis em

8 bits (unsigned char) de armazenamento. Quando usado para armazenar imagens geográficas, o formato JPEG pode também estar associado a arquivos de navegação, com mesmo formato ao descrito para o formato Tiff (Figura 2.12), porém com a extensão ".jgw".

O formato JPEG2000 (Christopoulos *et al.*, 2000) é uma nova proposta de padrão para codificação de imagens digitais que representa uma evolução frente a expansão das aplicações multimídia e de Internet. Esse padrão buscou definir um sistema de codificação de imagens digitais de diferentes tipos (binárias, monocromáticas, coloridas, com múltiplos componentes), com diferentes características e que atendesse a diferentes modelos de utilização (cliente-servidor, transmissão em tempo real ou criação de bibliotecas de imagens). O formato JPEG2000 utiliza uma forma de compressão totalmente baseada na transformada de *wavelets* ao contrário da transformada do cosseno usada no padrão JPEG original, sendo mais flexível e fornecendo melhor compressão (Usevitch, 2001). A inclusão de informações de geo-referenciamento no formato JPEG2000 pode ser feita através de extensões proprietárias como a GeoJP2™[1] ou através do uso de arquivos GML que contendo a descrição do sistema de georeferenciamento para a imagem. A última a opção é a que está sendo estudada pelo OGC (Cope *et al.*, 2005).

Os formatos *RAW* são formatos de armazenamento binários sem nenhuma formatação. Devem ser acompanhados de descrições a parte com pelo menos os parâmetros básicos sobre o dado e sobre a organização do arquivo como, por exemplo, a forma de entrelaçamento entre os elementos da representação.

Existem ainda outros formatos para representações matriciais como o *MrSID – Multiresolution Seamless Image Database*[2], o ASCII Grid dos produtos da ESRI[3] ou o

[1] O formato GeoJP2™ é um formato proprietário da empresa Mapping Science, Inc. Sua documentação e condições de uso podem ser encontrados em http://www.dimin.net/software/utils.html.
[2] O formato MrSID é proprietário da empresa LizardTech™ e sua documentação pode ser encontrada em http://www.lizardtech.com.
[3] A documentação sobre os formatos de intercâmbio dos produtos da empresa ESRI pode ser encontrada em http://www.esri.com.

ASCII-SPRING Grid do sistema SPRING[4]. Cada formato possui informações básicas ou geográficas sobre a representação, o que define o grau de intervenção de um usuário para manipular as representações.

Representações matriciais de geo-campos também podem estar armazenadas em SGBDs relacionais, por exemplo em campos do tipo binários longos, como se a representação estivesse em um arquivo binários *raw*.

[4] O sistema SPRING é um SIG freeware produzido pelo INPE/DPI. A documentação de seu formato de intercâmbio pode ser encontrada em http://www.dpi.inpe.br/spring.

CAPÍTULO 3

ARQUITETURA DO SISTEMA

Este capítulo descreve um subsistema extensível para manipulação de geo-campos, que armazena e recupera grandes volumes de dados de forma eficiente. O subsistema descrito neste capítulo endereça os problemas relativos ao tratamento de representações matriciais para geo-campos e, em especial, o caso de seu armazenamento em SGBDs.

3.1 Arquitetura geral do subsistema

A arquitetura geral do subsistema, identificando seus componentes, é mostrada de maneira informal na Figura 3.1. O subsistema prevê a existência de três componentes: *aplicações,* ou *serviços*, que contém procedimentos para manipulação das representações matriciais; *interface* que define a interface visível para os membros do componente de aplicações; e *armazenamento* que trata das questões de armazenamento das representações matriciais, seja em arquivos, seja em bancos de dados relacionais, resolvendo questões de acesso e de codificação/decodificação de formatos.

O subsistema possui duas características predominantes. Em primeiro lugar, além de abstrair diferenças internas de representação oferece maneiras padronizadas para desenvolvimento de algoritmos através do uso de técnicas de programação genérica e *iteradores* (Musser; Stepanov, 1988). Em segundo lugar, o subsistema está estruturado segundo o conceito geral de *framework*. De fato, um subsistema mais fechado, como um *framework*, pode ser usado não somente para fornecer funcionalidades, mas também para definir um estilo de programação, transmitindo uma qualidade desejada para o software final, compartilhando conceitos e não apenas APIs – *Application Interfaces*, como as oferecidas por bibliotecas de funções, como o GRASS – Geographic Resource

Analysis Support System (Mitasova; Neteler, 2004) ou a biblioteca GDAL – *Geospatial Data Abstraction Library*[5].

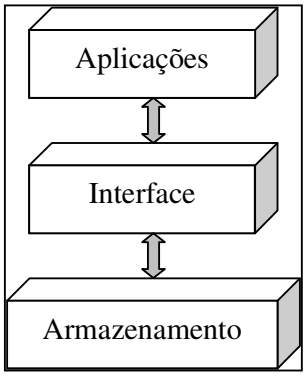


Figura 3.1 – Arquitetura do subsistema para armazenamento de geo-campos.

3.2 Componente interface

3.2.1 Parâmetros da representação matricial

Para abstrair as diferenças de representação considera-se importante a separação das informações sobre o dado, do dado em si. Essas informações são chamadas de *parâmetros* da representação matricial e podem ser derivadas a partir da definição de representação matricial (ver Equação 2.1). Dada uma representação matricial $RM = (S, V, M[m,n], T_M[2,3])$, seus parâmetros podem ser divididos em:

1) *geográficos*: relativos à localização da representação sobre uma área da superfície da Terra. Correspondem a $S, T_M[2,3]$;

2) *geométricos*: relativos à estrutura matricial que dá suporte à representação . Correspondem a $M[m,n]$;

3) *domínio*: representam as características do conjunto de valores *V*;

[5] A GDAL é uma biblioteca que funciona como um tradutor de dados geográficos matriciais. É desenvolvida no modelo *open source* e distribuída pelo endereço http://www.remotesensing.org/gdal/.

4) *armazenamento*: necessários para materializar a representação em um arquivo ou em um banco de dados.

Os parâmetros que descrevem uma representação devem ser expressos em uma estrutura de agregação com uma interface de acesso capaz de garantir que os parâmetros sejam coerentes entre si, utilizando regras definidas como parte da representação. A interface também deve ser capaz de produzir novas informações a partir da combinação das informações explicitamente armazenadas. Os parâmetros são normalmente extraídos diretamente dos formatos de arquivos que contém representações matriciais ou informados diretamente por um usuário de aplicação.

3.2.2 Interface básica

Voltando à definição mostrada no capítulo 2, um geo-campo é uma função que mapeia localizações (na superfície terrestre) em valores tirados de um conjunto *V*. Uma representação matricial de um geo-campo deve preservar essa característica funcional, oferecendo uma interface que permita recuperar o valor do geo-campo em cada uma das localizações definidas pela matriz $M[n,m]$. Como o subsistema pode ser usado também para construir uma representação matricial, a interface possui também a capacidade de atribuir valores de geo-campos a determinadas localizações. Essa interface deve ser geral o suficiente para representar a atribuição e recuperação de valores para diferentes tipos de domínio *V,* incluindo domínios multidimensionais. Assim, a interface deve exportar as seguintes funcionalidades:

- *atribuição* de um valor a uma posição da representação matricial: $M[i, j] = \mathbf{v}$;
- *atribuição* de um valor a uma dimensão de uma posição da representação matricial: $M[i, j][k] = v_k$;
- *recuperação* do valor de uma posição da representação matricial: $\mathbf{v} = M[i, j]$;
- *recuperação de um valor de uma dimensão* de uma posição da representação matricial: $v_k = M[i, j][k]$.

3.2.3 Iteradores para representações matriciais

Iteradores são abstrações capazes de percorrer uma estrutura de dados que agrega um conjunto de elementos, permitindo que esse percorrimento seja independente da representação interna da estrutura. Gamma *et al.* (1995) descrevem iteradores como um padrão de projeto que pode ser aplicado em diferentes domínios e problemas. Os participantes do padrão são: uma estrutura de agregação, que contém os elementos a serem percorridos, e um iterador, o qual é usado para acessar os elementos da estrutura de agregação.

Iteradores são fundamentais para o desenvolvimento de sistemas que utilizam técnicas de programação genérica (Musser *et al.*, 1988), pois são uma forma de separar algoritmos que dependem do percorrimento de um conjunto de elementos de uma coleção da coleção em si. O exemplo mais conhecido de programação genérica é a *STL – Standard Template Libray,* uma biblioteca de funcionalidades gerais que atualmente faz parte do padrão da linguagem C++ (Austern, 1998; Stroustrup, 1997). Um exemplo de programação genérica aplicada a uma biblioteca de processamento de imagens é descrito em Kothe (1999).

Dada uma estrutura de agregação `C` e um iterador `It`, o padrão de projeto baseia-se nos seguintes requisitos:

- `C` deve ser capaz de exportar iteradores para o início da seqüência de elementos;
- deve ser possível verificar se um iterador `It` aponta para o último elemento da seqüência a ser percorrido;
- deve ser possível avançar o iterador `It` para um próximo elemento dentro de `C`;
- deve ser possível obter o elemento para o qual um iterador `It` aponta.

O conceito de iteradores pode ser implementado de diferentes maneiras (Noble, 2000). Os *iteradores externos* são aqueles que usam a interface básica da estrutura de agregação e deixam para os clientes o controle sobre a iteração. Os clientes devem

explicitamente solicitar o avanço do iterador e consultar a posição corrente. Nos *iteradores internos,* os clientes passam uma operação para o iterador e esse é responsável por executá-la sobre todos os elementos que percorre. Os iteradores externos podem ser implementados como objetos aninhados que possuem acesso à representação interna da estrutura de agregação, que por usa vez exporta seus iteradores para os clientes.

Para permitir o uso de programação genérica nas aplicações clientes, a interface de uma representação matricial deve incluir a capacidade de acesso aos seus elementos através de iteradores externos. A estrutura de agregação é a representação matricial e seus elementos são os valores das representações em cada posição. Pode-se construir, por exemplo, iteradores que usam a estrutura matricial da representação para definir as seqüências de percorrimento. O modelo de iterador mais simples percorre a matriz linha-a-linha, começando o percorrimento no elemento $M(0,0)$ e, a cada operação de avanço, move o iterador para a posição na coluna subseqüente até que a última coluna seja atingida quando então o iterador é posicionado na primeira coluna da próxima linha.

O padrão de iterador pode ser refinado em diferentes variações, cada uma com características distintas, mas mantendo os requisitos originais do padrão. Esses iteradores podem fornecer diferentes ordens de percorrimento dentro do domínio (por exemplo, coluna-a-coluna) ou ainda selecionar subconjuntos do domínio que devem ser percorridos (por exemplo, somente aqueles cuja localização espacial está dentro de uma área delimitada por uma geometria).

3.3 Componente de armazenamento

Este componente é responsável por prover acesso eficiente e extensível às representações matriciais armazenadas em arquivos de dados ou SGBDs. Esta funcionalidade é obtida através do conceito de uma fonte de dados acessada através de um decodificador/codificador apropriado.

No caso de dados em arquivos, deseja-se: (1) abstrair o formato particular de armazenamento da representação; (2) extrair os parâmetros característicos da representação do arquivo; e (3) instanciar um decodificador para o formato de forma que a interface mostrada na seção anterior possa ser implementada. No caso das representações armazenadas em SGBDs, deseja-se: (1) extrair os parâmetros da representação através de uma consulta ao modelo conceitual do banco de dados; (2) instanciar um decodificador para a representação que possa interagir com o SGBD para ter acesso aos valores da representação. O capítulo 4 discute em detalhes as questões relativas ao armazenamento e recuperação eficientes de representações matriciais em SGBDs relacionais.

3.4 Componente de aplicações

O componente de aplicações contém funcionalidades de mais alto nível que não representam apenas o acesso ao valor do geo-campo nas posições da representação. Essas funcionalidades incluem:

- *criação* de uma representação em diferentes formatos de arquivos ou em SGBDs;

- *mapeamento* de uma representação em outra, variando características geométricas (por exemplo reamostragem, recorte ou expansão) ou geográficas (por exemplo, remapeamento de projeção) de uma representação;

- *transformação* entre representações, alterando o contradomínio do geo-campo (por exemplo, selecionando apenas uma determinada dimensão do contradomínio).

A lista de funcionalidades descrita acima não engloba todo o conjunto de aplicações que são necessárias para a manipulação de geo-campos, no entanto formam um conjunto representativo das principais funcionalidades. Sua implementação, conforme descrita nesta tese, forma um *framework* que pode ser estendido por outros desenvolvedores de aplicação que necessitem de outras funcionalidades.

3.5 Classes e métodos principais do subsistema

Essa seção descreve o conjunto de classes envolvido na construção do subsistema. Nomes de classes, estruturas de dados e seus métodos são escritos em uma fonte `mono-espaçada`, e os relacionamentos entre as classes são mostrados usando uma notação baseada na OMT (Rumbaugh *et al.*, 1991).

3.5.1 Classes e métodos relativos à descrição de representações matriciais

Inicialmente são definidas duas estruturas de dados: `Coord2D` para representar coordenadas bidimensionais; e `Box` para representar um retângulo envolvente, caracterizado pelas suas coordenadas inferior esquerda e superior direita (Figura 3.2).

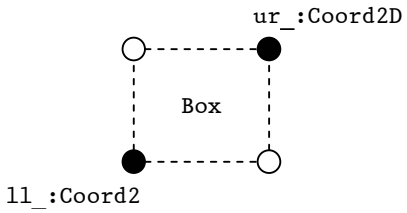


Figura 3.2 – *Box* ou retângulo envolvente.

A referência espacial *S* de uma representação para geo-campos é representada em uma interface `Projection` que define métodos de conversão entre coordenadas de projeção plana e coordenada geográficas, ou geodésicas. Como as coordenadas geográficas não são absolutas, ou seja, dependem de um datum planimétrico, torna-se necessário incluir no subsistema a classe `Datum` (d´Alge, 2004), que se relaciona com a interface de projeção por um relacionamento de agregação. Os métodos que devem ser publicamente oferecidos pela interface `Projection` estão mostrados na Tabela 3.1.

TABELA 3.1 – Os métodos da interface `Projection`.

Método	Descrição
`LL2PC`	Converte uma coordenada geodésica para coordenada de projeção
`PC2LL`	Converte uma coordenada de projeção para coordenada geodésica

A conversão entre coordenadas de duas projeções planas diferentes, *P* e *P'*, é sempre feita em dois passos. Primeiro, converte-se de *P* para um sistema de coordenadas geográficas *G*; em seguida, converte-se de *G* para *P'*. Cada derivação concreta da interface representa um sistema de projeção particular com parâmetros particulares.

Os parâmetros que definem uma representação matricial são representados em uma classe extensível chamada `RasterParameters`. A Tabela 3.2 mostra os métodos exportados pela classe de parâmetros da representação.

TABELA 3.2 – Métodos dos parâmetros da representação.

Método	Descrição
`setProjection`	Associa uma instância de projeção à representação
`getProjection`	Retorna instância de projeção associada à representação
`setTransformationMatrix`	Define os valores de T_M
`setNLines`	Define o número de linhas na representação
`setNColumns`	Define o número de colunas na representação
`getNLines`	Retorna o número de linhas na representação
`getNColumns`	Retorna o número de colunas na representação
`getResolutionY`	Retorna a resolução vertical da representação
`getResolutionX`	Retorna a resolução horizontal da representação
`getGeometricParameters`	Retorna o os parâmetros geométricos associados à representação: matriz de transformação, número de linhas e colunas
`getGeometricParameters(Box)`	Retorna o os parâmetros geométricos associados a uma área de interesse retangular, sobre a representação
`getBoundingBox`	Retorna o retângulo envolvente associado à representação
`getBox`	Retorna o retângulo envolvente central associado à representação
`coord2Index`	Converte uma coordenada no sistema S para o sistema de coordenadas de linhas e colunas da representação
`index2Coord`	Converte uma coordenada no sistema de coordenadas de linhas e colunas da representação para o sistema S

O conjunto de características relativas ao contradomínio de um geo-campo é composto de três tipos de informações: a interpretação semântica do geo-campo (por exemplo, imagem de sensoriamento remoto multi-espectral); a dimensionalidade do contradomínio do geo-campo; e o tipo associado a cada dimensão do contradomínio. Outra característica importante do contradomínio é a escolha de valores que representam ausência de informação sobre o geo-campo em algumas localizações da representação matricial.

O conjunto de características relativas ao armazenamento do geo-campo acrescenta aos parâmetros o conceito geral de uma fonte de dados. A Tabela 3.3 mostra os métodos que devem ser incluídos na classe de parâmetros para acessar as informações de domínio e de armazenamento.

TABELA 3.3 – Continuação dos métodos da interface de parâmetros.

Método	Descrição
`setDataSource`	Associa o nome de uma fonte de dados à representação
`getDataSource`	Retorna o nome de uma fonte de dados à representação
`setInterpretation`	Define a interpretação semântica da representação
`getInterpretation`	Retorna a interpretação semântica da representação
`setDimension`	Define a dimensionalidade de V
`getDimension`	Retorna a dimensionalidade de V
`setDataType`	Define o tipo computacional de cada elemento de uma dimensão de V
`getDataType`	Retorna o tipo computacional de cada elemento de uma dimensão de V
`getDummy`	Define o valor indicativo de ausência de informação em uma dimensão de V
`setDummy`	Retorna o valor indicativo de ausência de informação em uma dimensão de V

3.5.2 Classes e métodos relativos ao acesso a representações matriciais

O ponto central do subsistema é a classe `Raster`, que registra os parâmetros das representações matriciais disponíveis. Instâncias dessa classe podem ser construídas diretamente a partir de uma fonte de dados, quando é possível extrair dessa fonte os

parâmetros da representação, ou a partir de uma instância da classe `RasterParameters`, previamente construída. Além da descrição da fonte de dados (ou de uma instância de parâmetros), um indicador do modo de construção também é informado. Nesse indicador é possível especificar se a representação está sendo construída ou se já existe e está sendo consultada com ou sem permissão de modificação de seus valores. A classe `Raster` possui uma interface capaz de armazenar parâmetros, recuperar e atribuir valores do geo-campo em localizações e exportar iteradores. A Tabela 3.4 mostra os métodos exportados pela classe.

TABELA 3.4 – Métodos da classe `Raster`.

Método	Descrição
`Raster(DataSource,mode)`	Cria uma representação raster a partir de o nome de uma fonte de dados
`Raster(RasterParameters,mode)`	Cria uma representação raster a partir de um conjunto de parâmetros
`init`	Inicializa o sistema de decodificação da representação
`clear`	Libera o sistema de decodificação da representação
`setParameters`	Associa um conjunto de parâmetros a uma representação
`getParameters`	Retorna um conjunto de parâmetros a uma representação
`getElement`	Atribui o valor da representação em uma localização
`setElement`	Recupera o valor da representação em uma localização
`begin`	Retorna um iterador para o início de uma sequência de percorrimento da representação
`end`	Retorna um iterador para o final de uma sequência de percorrimento da representação

As representações podem estar em memória, ou armazenadas em arquivos em disco, em diferentes formatos, ou em tabelas de um SGBD com seu próprio formato, etc. Todos os meios de armazenamento devem ser tratados uniformemente no código do cliente, de forma a promover flexibilidade e facilitar a melhoria no gerenciamento do armazenamento à medida que o subsistema evolui sem a necessidade de atualizar o

código do cliente. O conceito de *decodificadores de formato* fornece a abstração-chave quanto ao armazenamento e decodificação de formato.

Decodificadores são responsáveis por acessar ou criar representações, recuperar ou inserir os parâmetros que as descrevem, instanciar e gerenciar dispositivos de memória ou quaisquer outras estruturas de dados necessárias para permitir o acesso aos elementos da representação. São representados na classe abstrata `RasterDecoder`, que pode ser materializada em classes concretas especializadas para cada formato, ou em um sistema de armazenamento particular, como exemplificado na hierarquia mostrada na Figura 3.3. A Tabela 3.5 mostra os métodos abstratos definidos pela interface `RasterDecoder`.

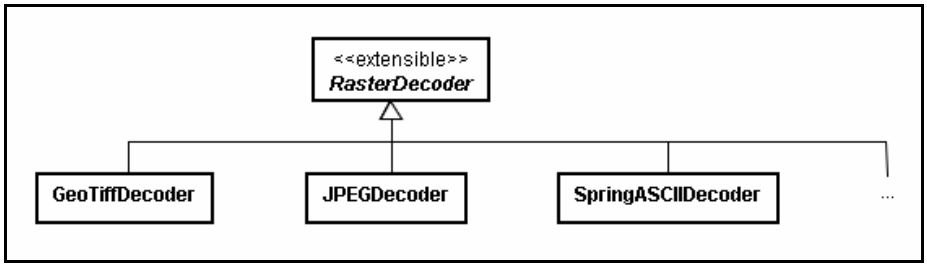


Figura 3.3 – Hierarquia de decodificadores de representação.

TABELA 3.5 – Métodos da classe `RasterDecoder`.

Método	Descrição
saveParameters	Associa um conjunto de parâmetros a uma representação
loadParameters	Retorna o conjunto de parâmetros de uma representação
getElement	Atribui o valor da representação em uma localização
setElement	Recupera o valor da representação em uma localização
init	Inicia estruturas internas e dispositivos de entrada e saída necessários para a disponibilização da estrutura de decodificação
clear	Libera estruturas internas e dispositivos de entrada e saída necessários para a disponibilização da estrutura de decodificação

A Figura 3.4 mostra um diagrama de relacionamento entre as classes descritas acima. O estereótipo <<*extensible*>> nas classes `RasterParameters` e `Projection` indica que

extensões ao subsistema podem acrescentar métodos e atributos à implementação destas classes básicas a fim de oferecer novas funcionalidades.

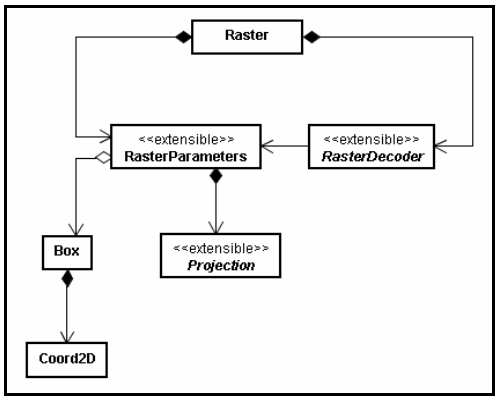


Figura 3.4 – Diagrama de relacionamento entre as classes do subsistema.

A Figura 3.5a mostra uma descrição simplificada, em pseudocódigo, de um procedimento de percorrimento de uma imagem armazenada em um arquivo (em formato GeoTiff, por exemplo). A chamada ao método que inicializa o objeto decodificador de formato faz com que os parâmetros da representação e o acesso aos elementos fiquem disponíveis para essa rotina que representa um cliente do subsistema. A Figura 3.5b mostra o mesmo percorrimento feito através de iteradores.

Exemplo 1: Listar os valores dos *pixels* de uma imagem em arquivo usando *loop*

```
Rst = Raster(fileName)          {Instancia um objeto Raster a partir do nome do arquivo}
Rst.init                                                          {Inicia a decodificação}
nLines = Rst.getParameters.nlines          {Acessa os  parâmetros da representação}
nColumns = Rst.getParameters.ncolumns
for c=0 to nLines do                                  { Recupera o valor de toda localização}
    for l=0 to nColumns do
      val = Rst.getElement(c,l)
    endfor
endfor
```

(a)

Exemplo 2: Listar os valores dos *pixels* de uma imagem em arquivo usando iterador

```
Rst = Raster(fileName)          {Instancia um objeto Raster a partir do nome do arquivo}
Rst.init                                          {Inicializar um objeto decodificador}
iterator = Rst.begin                            {Cria iterador para o primeiro elemento}
itEnd = Rst.end                                  {Cria iterador para o último elemento}
while (itBegin != itEnd) do                       {Recupera o valor de todo elemento}
  iterator.getElement(val)                                        {visitado pelo iterador}
  iterator.moveNext
endwhile
```

(b)

Figura 3.5 – Algoritmo para percorrer uma imagem: (a) usando loop e (b) usando iteradores.

Uma parte importante da classe `Raster` é a definição do decodificador mais apropriado para cada formato de representação. Quando a fonte de dados é um arquivo, normalmente é possível inferir qual o formato do dado a partir de sua extensão. Em outros casos essa informação é passada ao subsistema por algum tipo de interação com o usuário. Ou seja, a criação do decodificador concreto deve ser adiada até o tempo de execução. Essa situação é um problema recorrente no desenvolvimento de sistemas e que pode ser solucionado através de técnicas chamadas genericamente de *fábricas de objetos* (Alexandrescu, 2001). Assim, o subsistema define o conceito de uma fábrica de decodificadores para representações, que é única em todo o subsistema. Cada decodificador concreto deve ser registrado na fábrica indicando o identificador que o caracteriza unicamente. A criação de um decodificador concreto é sempre feita através de um método da fábrica, informando qual seu identificador. A Figura 3.6 mostra o relacionamento entre as classes apresentadas anteriormente e o decodificador de formatos.

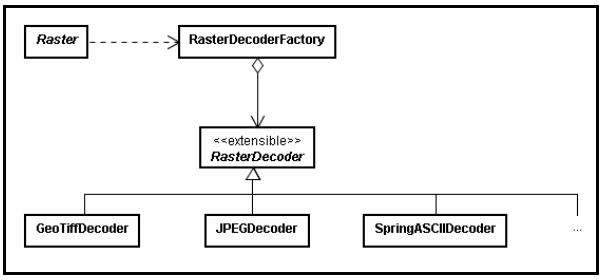


Figura 3.6 – Fábrica de decodificadores.

Essa solução é baseada no padrão de projeto *Abstract Factory* (Gamma *et al.*, 1995), cuja vantagem principal é separar a criação de objetos de sua utilização, permitindo que a introdução de novos objetos similares no sistema não altere o código já existente. O subsistema mantém uma única instância da fábrica de decodificadores, o que pode ser garantido fazendo uso do padrão de projeto *Singleton*[6].

A questão da escolha do identificador apropriado, único, para cada decodificador não é tratada no padrão de projeto, sendo deixada a cargo do restante do sistema. Nesse caso, como é bastante útil associar decodificadores a fontes de dados cujos formatos podem ser inferidos a partir de seus nomes, propõe-se que a identificação seja dada por nomes, ou cadeias de caracteres. Para permitir a associação de diferentes extensões ao mesmo formato (por exemplo, o formato JPEG pode estar associado à extensão ".jpg" ou ".JPEG"), o subsistema prevê a existência de um dicionário de sinônimos para a identificação de cada decodificador, que pode ser gerado a partir de arquivos de configuração. Esse dicionário pode ser acessado por objetos da classe `Raster` de forma que essa possa inferir qual o identificador adequado a uma dada fonte de dados.

O método `init` da classe `Raster` é responsável pela interação com a fábrica de decodificadores, encontrando a identificação do decodificador mais apropriado para uma representação e solicitando a sua construção. Após a chamada desse método, pode-se verificar o seu valor de retorno: sucesso representando que foi possível associar um

[6] Os padrões de projeto (Gamma *et al.*, 1995) tem como objetivo descrever soluções orientadas a objeto para problemas recorrentes no projeto de sistemas. Eles introduzem um vocabulário comum entre desenvolvedores que documentam os sistemas facilitando sua compreensão. Outros exemplos do uso padrões de projeto em aplicações relacionadas ao domínio dos sistemas de informação geográfica podem ser encontrados em Gordillo *et al.* (1999), Câmara (2001) e Laurini*, et. al.* (2003).

decodificador concreto à representação e que esse pode definir ou recuperar valores em todas as suas posições; fracasso indica que a representação não está disponível, ou um decodificador para ela não pode ser criado. O método `clear` é responsável por liberar o decodificador associado à representação e, portanto, tornar a representação indisponível.

3.6 Aplicações

Nessa seção são apresentados os conceitos mais importantes em termos de aplicações escritas usando o subsistema proposto nessa tese, adotando-se uma notação de pseudocódigo.

3.6.1 Instanciando uma representação

A instanciação no subsistema de uma representação já existente em um arquivo é feita através da criação de um objeto da classe `Raster` e sua inicialização. A Figura 3.7 mostra o pseudocódigo que exemplifica a instanciação de uma representação armazenada em um arquivo em um formato com parâmetros (Figura 3.7a) e em um arquivo em um formato sem parâmetros (Figura 3.7b). No segundo caso, a estrutura de parâmetros deve ser preenchida de acordo com informações que não podem ser retiradas do próprio arquivo de dados. A Figura 3.8 mostra os diagramas de seqüência relativos aos programas mostrados na Figura 3.7.

O procedimento para a criação de novos arquivos de representação é o mesmo, variando somente o modo especificado no nome da fonte ou no conjunto de parâmetros. Relativo a esse modo, o método `init` encapsula todo o controle de acesso a arquivos, além da alocação de recursos de memória necessários para instanciar a representação dentro do subsistema. O método `clear` tem a responsabilidade inversa, ou seja, liberar memória e dispositivos de armazenamento quando a representação não for mais utilizada.

Exemplo 3: Instanciar uma representação por nome de arquivo

```
sourceName = "imagem.tif"
Rst = Raster(sourceName, 'r')
if (Rst.init) do
{usa método getElement}
endif
Rst.clear
```

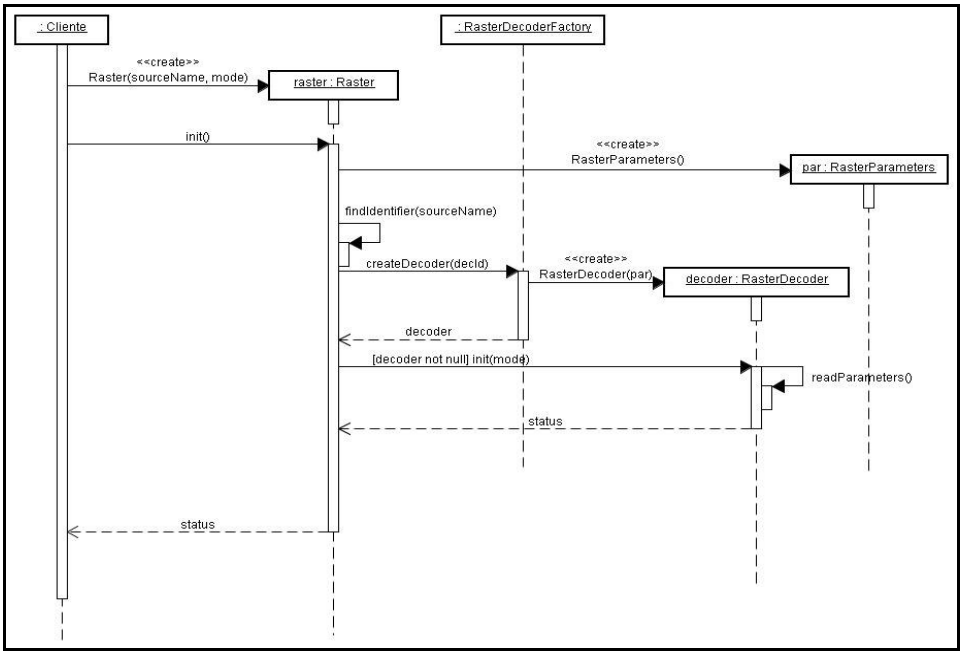
(a)

Exemplo 4: Instanciar uma representação por conjunto de parâmetros relativos

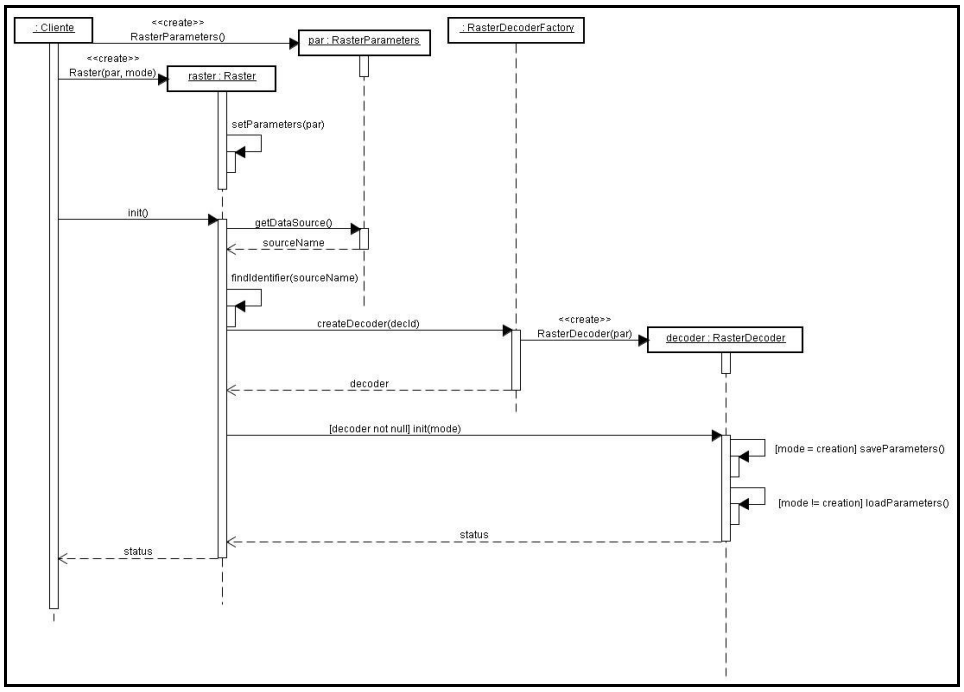
```
par = RasterParams()
par.setNLines(512)
par.setNColumns(512)
...
setDataSource("imagem.raw", 'r')
Rst = Raster(par)
if (Rst.init) do
{ usa método getElement }
endif
Rst.clear
```

(b)

Figura 3.7 – Peudo-código para instanciar uma representação para leitura: (a) a partir de uma fonte de dados e (b) a partir de um conjunto de parâmetros.



(a)



(b)

Figura 3.8 – Criação de uma representação matricial a partir de uma fonte de dados.

3.6.2 Transformações geométricas e geográficas

A transformação, ou mapeamento, de uma representação em outra com características geométricas ou geográficas diferentes está concentrada em um único componente do subsistema, implementado como uma classe. Essa classe é chamada de `RasterRemap` e é uma variação do padrão de projeto *command*, definido em (Gamma *et al.*, 1995), que permite o encapsulamento de uma requisição em um objeto, permitindo assim a parametrização de clientes por diferentes requisições além de controlar o tempo de execução da requisição. Os clientes do subsistema criam instâncias de um objeto `RasterRemap` passando todas as informações necessárias para a execução da transformação e esse objeto pode ser manipulado, ou reaproveitado, em diferentes pontos do fluxo de execução das aplicações cliente do subsistema. A classe `RasterRemap` possui dois parâmetros principais, a representação original (entrada) e a nova representação (saída), e um método `apply`, que executa o mapeamento.

Esse método é responsável por verificar as diferenças entre os parâmetros geométricos e geográficos das duas representações e executar uma de três possíveis ações:

1) *cópia*: quando as duas representações são geométrica e geograficamente idênticas;

2) *reamostragem*: quando as duas representações são diferentes geometricamente. Inclui também a alteração no domínio da representação:

3) *recorte*: quando o domínio da representação é reduzido;

4) *expansão*: quando a extensão da representação é aumentado. Nesse caso, as novas localizações, ou novos pontos do domínio, são atribuídos um valor representativo de falta de informação;

5) *remapeamento* de projeção: quando as duas representações são diferentes geograficamente.

A classe `RasterRemap` utiliza a interface da classe `Raster` para executar suas funcionalidades. Cada uma dessas funcionalidades é um método da classe `RasterRemap` como mostra a Tabela 3.6.

TABELA 3.6 – Métodos da classe `RasterRemap`.

Método	**Descrição**
`setInputRaster`	Define a representação de entrada
`setOutputRaster`	Define a representação de saída
`apply`	Executa o mapeamento
Métodos internos chamados pelo método `apply`	
`checkGeometryCompatibility`	Verifica qual a diferença entre a geometria das duas representações
`copy`	Copia uma representação para outra
`resample`	Executa uma reamostragem entre as duas representações
`remap`	Executa um remapeamento entre as duas representações

A utilização dessa classe será explicada através de um conjunto de exemplos. O primeiro exemplo, na Figura 3.9, mostra como a classe pode ser usada para duplicar uma representação, mas mudando o formato de armazenamento, que é inferido a partir do nome da fonte de dados, como mostrado na seção anterior.

O exemplo na Figura 3.10 mostra como criar uma nova representação, com uma extensão diferente, por exemplo, para fazer um recorte em uma área (retângulo) de interesse. Finalmente, o exemplo na Figura 3.11 mostra como criar uma nova representação a partir de uma existente, alterando seu sistema de projeção cartográfica. Deve-se observar que o método responsável por redefinir a matriz de transformação na nova projeção é aquele que define qual é a projeção cartográfica associada à representação.

Os exemplos mostram a utilidade da separação dos parâmetros das representações dos dados em si. Na criação de uma representação a partir de outra, os parâmetros da representação original são aproveitados, sofrem pequenas alterações e podem ser usados para gerar a nova representação. Os clientes do subsistema precisam conhecer e utilizar um número pequeno de métodos da classe de parâmetros, enquanto que detalhes de armazenamento interno e a lógica de processamento das representações ficam escondidas na classe `Raster` e na classe `RasterRemap`.

Exemplo 5: Cópia de representações

```
Rst = Raster("imagem.tif", 'r')                    {Instancia representação existente}
Rst.init
parOut = rst.getParameters
parOut.setDataSource("imagem.jpg", 'c')            {Instancia nova representação}
Rst_copy = Raster(parOut)
Rst_copy.init
copier = RasterRemap(Rst, Rst_copy)                {Copia representações}
copier.apply
Rst_copy.clear
Rst.clear
```

Figura 3.9 – Cópia de representações.

Exemplo 6: Recorte de representação

```
Rst = Raster("imagem.tif", 'r')              {Instancia representação existente}
Rst.init
roi = Box{ xll, yll, xur, yur }
Rst.getParameters.getGeometricParameters(roi,TM_out,nlines,ncols)
parOut.setDataSource("imagem_recorte.tif", 'c')
parOut.setNLines(nlines)                        {Instancia nova representação}
parOut.setNcolumns(ncols)                              {com domínio menor}
parOut.setTransformationMatrix(TM_out)
Rst_clip = Raster(parOut, 'c')                  {Instancia nova representação}
Rst_clip.init
clipper = RasterRemap(Rst, Rst_clip)
clipper.apply                                          {Recorta representação}
Rst_clip.clear
Rst.clear
```

Figura 3.10 – Recorte de uma região de interesse retangular de uma representação.

Exemplo 7: Mudança de projeção de representações

```
Rst = Raster("imagem.tif", 'r')              {Instancia representação existente}
Rst.init
parOut = Rst.getParameters
parOut.setDataSource("imagem_reproj.tif",'c')
parOut.setProjection(projMercator)              {Instancia nova representação}
Rst_merc = Raster(parOut)                        {com uma projeção diferente}
Rst_merc.init
reprojector = RasterRemap(Rst, Rst_merc)          {Reprojeta representação}
reprojector.apply
Rst_merc.clear
Rst.clear
```

Figura 3.11 – Mudança de projeção de uma representação.

Outro ponto a ser ressaltado é que a criação de representações a partir de uma representação original pode incluir variações em mais que uma característica geométrica

simultaneamente. A Figura 3.12 mostra um exemplo onde se deseja criar uma nova representação, com um domínio menor, em um formato diferente e em uma projeção diferente. Novamente, a nova representação é totalmente definida a partir da definição correta de seus parâmetros e a classe `RasterRemap` executa todo o processamento.

Exemplo 8: Recorte, remapeamento e mudança de formato

```
Rst = Raster("imagem.tif", 'r')                    {Instancia representação existente}
Rst.init
roi = Box { xll, yll, xur, yur }
parOut = Rst.getParameters
parOut.setDataSource("imagem_nova.jpg", 'c')                    {Muda formato}
parOut.getGeometricParameters(roi,TM_out,nlines,ncols)
parOut.setNLines(nlines)                          {Define retângulo de interesse}
parOut.setNcolumns(ncols)
parOut.setTransformationMatrix(TM_out)
parOut.setProjection(projMercator)                              {Muda projeção}
Rst_t = Raster(parOut, 'c')                       {Instancia nova representação}
Rst_t.init
transf = RasterRemap(Rst, Rst_t)          {Cria nova representação com modificações}
transf.apply
Rst_t.clear
Rst.clear
```

Figura 3.12 – Combinação de transformações geométricas.

3.6.3 Transformações entre contradomínios

A classe `RasterRemap` foi desenvolvida com o objetivo de transformar representações variando as suas características geométricas e geográficas, que tem a ver com o domínio dos geo-campos e suas representações. Mas existem outras transformações que alteram as características do contradomínio do geo-campo. Por exemplo, supondo que o geo-campo tenha um contradomínio unidimensional definido sobre o conjunto $\Re$, deseja-se criar uma nova representação a partir dessa, porém com um contradomínio unidimensional definido sobre o conjunto I dos números inteiros. A transformação de

um valor real para um valor inteiro pode ser feita de diferentes maneiras como, por exemplo, descartando a parte decimal, ou aplicando uma função do tipo $f : \Re \to \mathrm{I}$ qualquer.

Para atender a essa necessidade, o subsistema prevê o conceito de um *transformador de contradomínio* entre duas representações, que pode ser acoplado à classe `RasterRemap`, aumentado a capacidade de mapeamento para além de características somente geométricas. Os transformadores de contradomínio podem armazenar estados ou variáveis próprias e podem ser alterados mediante interações com outras partes do subsistema, independentemente do mapeamento geométrico.

Um transformador de contradomínio funciona como um *functor* generalizado. Ou seja, uma abstração que representa a invocação de um processamento encapsulado em um objeto com tipo e valor semântico (Alexandrescu, 2001). Esses objetos armazenam processamentos como se fossem valores, de forma que podem ser passados como parâmetros de funções ou métodos e invocados em um ponto distante de seu ponto de criação. Todo *functor* possui um único ponto de início como um método ou um operador da classe. Os *functores* também são conhecidos como o padrão de projeto *command* (Gamma *et al.*, 1995).

No subsistema proposto os transformadores de domínio são implementados através de uma classe extensível chamada `RasterTransform`. A classe `RasterTransform` pode tratar de diferentes aspectos no mapeamento entre duas representações: a transformação entre tipos associados a uma dimensão particular, a associação entre dimensões de dois contradomínios ou ainda a seleção de um valor específico dentro de uma dimensão do contradomínio. A interface da classe define o método `apply` como o ponto de execução da funcionalidade. Esse método recebe como parâmetros as identificações das duas localizações geometricamente equivalentes entre as duas representações, definidos pelos métodos da classe `RasterRemap`. Estende-se a interface da classe `RasterRemap` para aceitar um objeto da classe `RasterTransform`. Nos exemplos anteriores de mapeamento entre representações, mostrados nas Figuras 3.10, 3.11 e 3.12, vale que, quando não especificado, existe um transformador de contradomínio *default* que produz

para a representação de saída um contradomínio idêntico ao da representação de entrada. Na Figura 3.13 é mostrado um exemplo de uma possível hierarquia de transformadores de contradomínio.

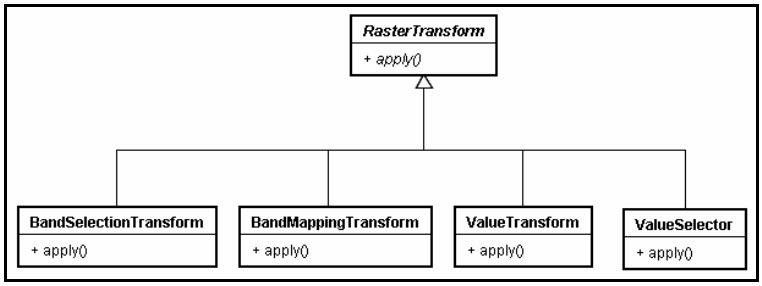


Figura 3.13 – Transformadores de contradomínio.

A Figura 3.14 mostra o exemplo de uso de um transformador entre dois contradomínios onde o primeiro está associado ao conjunto dos números reais e o segundo ao conjunto dos números inteiros. O transformador utiliza uma função linear para fazer o mapeamento entre os dois conjuntos, com parâmetros de ganho e *offset* definidos arbitrariamente. Esse transformador é passado então à classe de mapeamento encarregada de resolver as diferenças espaciais entre as representações.

Exemplo 9: Recorte, remapeamento, mudança de formato e mudança de contradomínio

```
Rst = Raster("imagem.tif", 'r')                    {Instancia representação existente}
Rst.init
roi = Box{ xll, yll, xur, yur }
parOut = Rst.getParameters
parOut.setDataSource("imagem_nova_linear.jpg",'c')
parOut.getGeometricParameters(roi,TM_out,nlines,ncols)
parOut.setNLines(nlines)
parOut.setNcolumns(ncols)
parOut.setTransformationMatrix(TM_out)
parOut.setProjection(projMercator)              {Define transformações de domínio}
Rst_new = Raster (parOut, 'c')                     {Instancia nova representação}
Rst_new.init
lienarTranf = RasterTransform(gainl, offsetl)
transf = RasterRemap(Rst, Rst_new)
transf.setTransformer(linearTransform)     {Define transformações de contradomínio}
transf.apply                                          {Transforma  representação}
Rst_new.clear
Rst.clear
```

Figura 3.14 – Mapeamento geométrico com transformação de contradomínio.

Os transformadores também podem ser usados para construir predicados de seleção de dimensões do contradomínio, ou mais especificamente ainda de valores particulares dentro do contradomínio como mostra o exemplo da Figura 3.15.

Os transformadores de contradomínio podem ser combinados entre si. Por exemplo, pode-se extrair uma dimensão do contradomínio e modificá-la por uma transformação linear, o que pode levar a um aumento indesejável no número de classes derivadas dos transformadores de contradomínio. Uma alternativa para evitar esse problema é o uso novamente de um padrão de projeto chamado *decorator* descrito em (Gamma *et al.*, 1995) como um objeto capaz de acrescentar responsabilidades adicionais a um objeto dinamicamente, fornecendo uma alternativa mais flexível para a extensão de funcionalidades a uma classe que o uso de derivação.

Exemplo 10: Seleção de dimensão

```
Rst = Raster("imagem.tif", 'r')                    {Instancia representação existente}
Rst.init
parOut = Rst.getParameters
parOut.setDataSource("imagem_mono.jpg",'c')
parOut.setDimension(1)
Rst_new = Raster (parOut, 'c')                     {Instancia nova representação}
Rst_new.init
bandSelector = RasterTransform(3)      {Seleciona apenas a 3ª dimensão do contradomínio}
transf = RasterRemap(Rst, Rst_new)
transf.setTransformer(bandSelector)          {Define transformações de contradomínio}
transf.apply                                       {Transforma representação}
Rst.clear
Rst_new.clear
```

Figura 3.15 – Seleção de uma dimensão do contradomínio.

3.7 Resumo do subsistema

A interface `RasterDecoder` concentra os principais pontos de extensibilidade do subsistema ao abstrair as questões de formatos e dispositivos de armazenamento em uma única classe. O conceito de fábricas de decodificadores permite que a inclusão de suporte a novos formatos e dispositivos não implique em mudanças nas aplicações que utilizam o subsistema.

O conceito de iteradores possui a responsabilidade de fornecer o percorrimento entre as localizações da representação matricial uma por uma, de acordo com regras bem definidas, como por exemplo, baseadas em relacionamentos topológicos entre as localizações da representação matricial e geometrias associadas a geo-objetos. Esse conceito pode ser estendido para outras regras de percorrimento, seja para representações matriciais ou outros tipos de representações para geo-campos.

Voltando à especificação de interfaces da classe *coverage* da seção 2.2, pode-se dizer que o item 1, avaliação, já está implementado na interface especificada na classe

`Raster`; o item 2, avaliação inversa, pode ser implementado através dos transformadores de contradomínio que fazem seleção de dimensões ou de valores; e o item 3, serviços, estão em parte implementados diretamente na classe `Raster` e na classe `RasterParameters`, e outros podem ser implementados em camadas acima do subsistema, que esconde detalhes mais próximos de características de armazenamento e formatos.

A Figura 3.16 mostra uma visão geral das classes envolvidas no subsistema projetado nessa tese, agrupando-as em suas três componentes principais. Pode-se observar que o subsistema é baseado em um número pequeno de classes, que por sua vez possuem um número pequeno de métodos públicos, o que torna o subsistema mais fácil de ser utilizado por aplicações clientes.

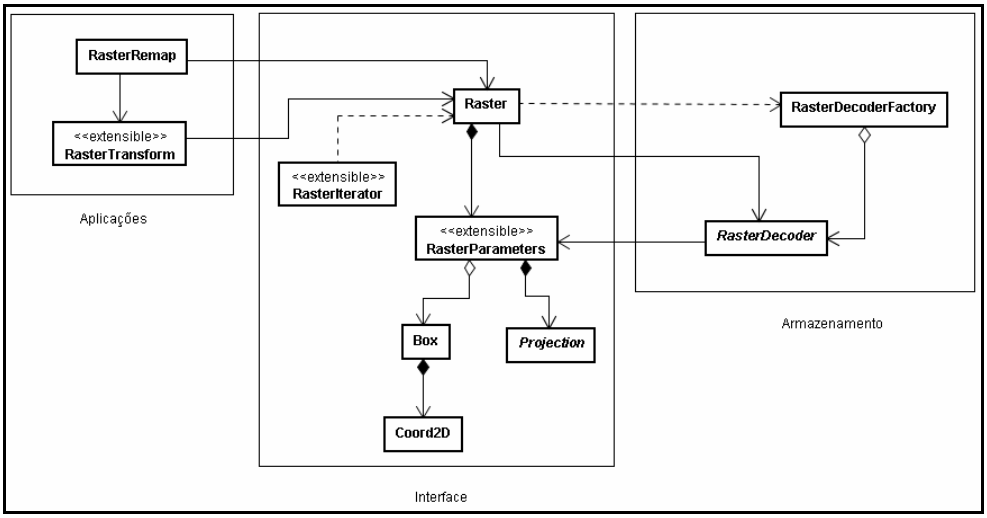


Figura 3.16 – Uma visão geral do subsistema.

CAPÍTULO 4

ARMAZENAMENTO EM BANCOS DE DADOS

4.1 Bancos de dados geográficos

Há dois modelos de arquitetura comumente adotados para acomodar a interação entre SIGs e SGBDs. Em uma *arquitetura dual*, o SGBD trata somente da componente descritiva do dado geográfico (que possui somente dados convencionais), enquanto que, em uma *arquitetura integrada,* o SGBDs também oferece suporte para o gerenciamento da componente espacial do dado geográfico.

Na arquitetura integrada, a manipulação da componente espacial pode ser feita de três maneiras: utilizando *campos longos* (ou BLOBs) para armazenar os dados espaciais; utilizando *extensões espaciais* construídas sobre o SGBD, as quais fornecem tipos espaciais, operadores e métodos de acesso específicos; e, finalmente, *combinando* as duas opções anteriores, ou seja, utilizando os BLOBs para armazenar tipos espaciais que a extensão não oferece. Uma discussão detalhada sobre os modelos de arquiteturas, suas vantagens e desvantagens pode ser encontrada em Ferreira (2005).

Podemos citar como exemplo de extensões espaciais comerciais os produtos *Oracle Spatial* (Ravada; Sharma, 1999) e o *IBM Spatial Extender* (Adler, 2001). Como exemplos de extensões espaciais de licença livre, ou seja, *freeware*, temos a extensão espacial *PostGIS* para o SGBD PostgreSQL (Santilli *et al.*, 2005) e os tipos espaciais disponíveis no SGBD MySQL (Karlsson, 2004). Embora largamente baseadas nas especificações do OpenGIS (McKee; Buehler, 1998), estas implementações possuem variações relevantes entre os modelos de dados, a semântica dos operadores espaciais e os mecanismos de indexação. Dessas extensões, apenas o Oracle Spatial, a partir da versão 10g, oferece a capacidade de tratar as estruturas matriciais para representação de geo-campos (Farley, 2003). Esse pode ser considerado um indicativo de que o suporte às representações de geo-campos pode ainda ser considerado como um assunto em desenvolvimento na área de bancos de dados geográficos (Shekkar; Chawla, 2003).

A abordagem mais comum para o gerenciamento de representações matriciais para geo-campos tem sido a criação de servidores especializados, como é o caso do PARADISE (Patel *et al.*, 1997) e do RASDAMAN (Baumann *et al.*, 1998). Sua principal vantagem é o ganho em desempenho, especialmente para o caso de grandes bases de dados, o que é muito comum quando se tratam de representações matriciais de geo-campos. Porém, um servidor especializado em geo-campos, e que pode não ser o mais adequado para tratar geo-objetos, pode ser uma desvantagem para o SIG, visto que as aplicações geográficas freqüentemente manipulam geo-campos conjuntamente com geo-objetos.

Esse capítulo trata da questão do armazenamento de geo-campos em SGBDs, a segunda parte do subsistema proposto nessa tese, tendo em vista a utilização de uma arquitetura integrada combinada, ou seja, onde as geometrias vetoriais são armazenadas utilizando-se os recursos oferecidos pelas extensões espaciais e as geometrias matriciais são armazenadas em BLOBs. As funcionalidades para manipulação de geometrias matriciais são fornecidas por uma camada externa ao SGBD, de modo a complementar os recursos ausentes, até o momento, nas extensões.

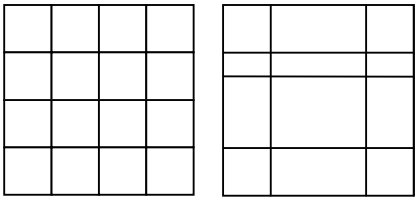
4.2 Características gerais do armazenamento de representações matriciais

Essa seção levanta os principais requisitos que devem ser considerados na proposição de um modelo de armazenamento das representações matriciais utilizando somente campos binários longos em um SGBD relacional. Ou seja, esta seção não considera a existência de tipos abstratos ou extensões espaciais por parte do SGBD

4.2.1 Particionamento

Duas características importantes das representações matriciais, especialmente aquelas que representam mosaicos de imagens de satélite, são: (1) seu tamanho excede a capacidade de memória principal dos sistemas; e (2) são tipicamente acessadas apenas parcialmente através de consultas espaciais. Por isso, qualquer sistema de armazenamento deve suportar, de forma transparente, o particionamento de representações em blocos menores e disjuntos, individualizáveis, a partir dos quais seja possível reconstruir a representação total posteriormente.

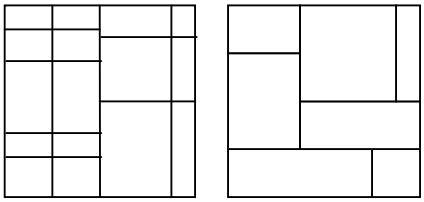
Um particionamento é *alinhado* quando é definido por planos paralelos aos eixos do domínio espacial, e portanto ortogonais entre si. Um particionamento alinhado é *regular* quando os planos são eqüidistantes (Figura 4.1a). Casos particulares de particionamento regular são aqueles em que toda a representação está inclusa em um único bloco, adequado quando a representação pode ser considerada pequena e também os casos de blocos criados ao longo de uma direção (por exemplo, ao longo de uma linha ou de uma coluna). Um particionamento alinhado é *irregular* quando os planos não são equidistantes (Figura 4.1b).



(a) regular (b) irregular

Figura 4.1 – Particionamento alinhado de representações.

Um particionamento é *não alinhado* quando contém blocos adjacentes com vértices não coincidentes. Um particionamento é *totalmente não alinhado* quando todos os blocos adjacentes contém vértices não coincidentes; caso contrário, é *parcialmente não alinhado*. A Figura 4.2 mostra exemplos de particionamento parcialmente e totalmente não alinhado.



(a) parcialmente (b) totalmente

Figura 4.2 – Particionamento não alinhado de representações.

Mais precisamente, dada uma representação matricial $RM = (S, V, M[m,n], T_M[2,3])$ de um geo-campo, um *particionamento* de *RM* é uma quádrupla $RB = (S, V, \beta, T_M[2,3])$, onde β é um conjunto finito de *blocos* da forma $B = (ID, (b_{x0}, b_{y0}), w, h)$, onde

- *ID* é um identificar único de *B* dentro do conjunto β (a definição exata de *ID* é deixada em aberto);
- $0 \le b_{x0} < m$ e $0 \le b_{y0} < n$;
- *w* e *h* são inteiros definindo, respectivamente, a largura e a altura do bloco.

Exige-se ainda que β *cubra totalmente M de forma não ambígua*, ou seja, dados inteiros *i* e *j* tais que $0 \le i < m$ e $0 \le j < n$, existe um único bloco $B = (ID, (b_{x0}, b_{y0}), w, h)$ em β tal que $b_{x0} \le i < b_{x0} + w$ e $b_{y0} \le i < b_{y0} + h$.

No caso do particionamento alinhado regular, *w* e *h* são os mesmos para todos os blocos que compõem a representação.

4.2.2 Indexação dos blocos

O padrão de acesso às representações matriciais armazenadas no banco de dados é dividido basicamente em:

1) acesso a uma das representações que correspondem aos níveis mais altos da pirâmide;
2) acesso a parte de uma das representações, ou a um subconjunto de seu domínio, normalmente definido por uma janela retangular;
3) acesso a elementos individuais da representação original;
4) acesso em uma determinada seqüência aos elementos de uma representação original.

Observando os padrões de acesso mostrados acima, nota-se que a identificação de cada bloco, de cada representação, deve ser capaz de responder basicamente a duas consultas:

Q1. "*Qual o bloco que contém o elemento (i,j) da representação*?"

Q2. "*Quais os blocos de uma representação preenchem uma determinada região geográfica*?"

A primeira consulta exige que o subsistema seja capaz de mapear a localização de um elemento na representação para o identificador do bloco que o contém. A segunda consulta exige que o subsistema seja capaz de responder a uma consulta espacial do tipo seleção por região (Ferreira *et al.*, 2005). Para que isso possa ser feito de maneira eficiente, os blocos devem possuir uma indexação espacial. Esse problema é abordado como um problema de criação de uma identificação de blocos que represente um mapeamento entre um espaço bidimensional, a extensão espacial do bloco, para um espaço unidimensional de identificadores inteiros, os quais podem ser mantidos em estruturas indexadas tradicionais em SGBDs como por exemplo B-Trees (Gaede; Guenther, 1998). Essa identificação deve ser de tal forma que blocos que são comumente requisitados conjuntamente, por serem espacialmente adjacentes, possuam identificações adjacentes e, portanto, estejam agrupados no banco de dados.

As curvas de preenchimento de espaço (Sagan, 1994) tem sido extensivamente usadas para o mapeamento de um espaço multidimensional para um espaço unidimensional. Intuitivamente, uma *curva de preenchimento* é uma função que visita todos os pontos de um espaço multidimensional, exatamente uma vez, e em uma ordem pré-determinada. Dessa forma, a curva de preenchimento impõe uma ordem linear sobre os pontos do espaço multidimensional. A Figura 4.3 ilustra graficamente algumas curvas de preenchimento em um espaço bidimensional e mostra como cada curva determina uma ordem de percorrimento dos pontos do espaço.

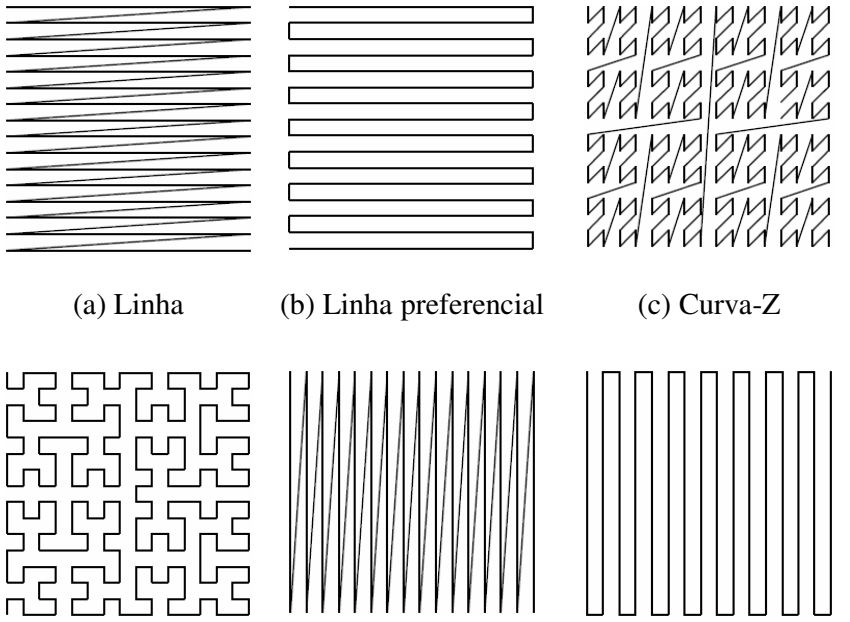


Figura 4.3 – Exemplos de curvas de preenchimento de espaço.

Ainda que nenhuma curva de preenchimento seja capaz de gerar uma ordem tal que pontos adjacentes no espaço estejam adjacentes na curva, considera-se que produzem uma ordem aceitável, até certo nível, para ser usada na geração de índices espaciais. Ou seja, a ordem gerada é tal que uma região no domínio multidimensional é mapeada para um intervalo, ou apenas alguns intervalos, no domínio unidimensional.

Abel (1990) em um estudo comparativo, sugere que a curva de Hilbert e a curva-Z são as mais promissoras quanto a preservação da proximidade espacial. Porém, as curvas de coluna e linha preferenciais podem ser úteis quando o padrão de acesso esperado é o acesso a representação inteira passando por todos os elementos nessa ordem, como acontece em algumas operações de manipulação de representações, por exemplo de processamento de imagens. Asano (1997) define as chamadas curvas de preenchimento recursivas e prova matematicamente qual o número máximo de operações de recuperação necessárias para responder a algumas consultas espaciais. Lawder (2001) e Bartholdi (2001) mostram como gerar os rótulos associados a ordem definidas nas curvas de preenchimento, uma vez que mesmo que as curvas sejam bem definidas, a

geração de rótulos unidimensionais únicos, que mantém a ordem descrita na curva, pode depender de algoritmos custosos. Mokbel et al. (2003) apresentam uma revisão do uso das curvas de preenchimento para a geração de índices unidimensionais para domínios multidimensionais e propõe uma ferramenta sistemática de escolha da melhor curva de preenchimento para uma determinada aplicação.

4.2.3 Multiresolução

Dada uma representação $RM = (S, V, M[m,n], T_M[2,3])$, uma *representação em multiresolução,* ou *uma pirâmide de multiresolução*, de *RM* consiste de um conjunto de K representações $RM_i = (S, V, M[m_i, n_i], T_{Mi}[2,3])$ tais que, para $i = 1,\ldots,K$:

1) RM_i possui o mesmo suporte espacial S e o mesmo contradomínio V que a representação original *RM*;

2) existem inteiros f_i e g_i, definindo o *relacionamento de multiresolução* entre as representações RM e RM_i, tais que

   a) $m_i = \lceil m / f \rceil$ e $n_i = \lceil n / g \rceil$ (onde a função $\lceil x \rceil$ representa o menor inteiro maior que x);

   b) a matriz de transformação $T_{M_i} = \begin{bmatrix} R_{Xi} & Ro_{xi} & X_{0i} \\ Ro_{Yi} & -R_{Yi} & Y_{0i} \end{bmatrix}$ relaciona-se com a matriz de transformação $T_M = \begin{bmatrix} R_X & Ro_X & X_0 \\ Ro_Y & -R_Y & Y_0 \end{bmatrix}$ de tal forma que $R_{Xi} = f_i * R_X$ e $R_{Yi} = g_i * R_Y$.

   c) $g_i \leq g_{i+1}$ *e* $f_i \leq f_{i+1}$, sendo que pelo menos um deles deve ser estritamente menor.

Informalmente, pode-se dizer que cada representação da multi-resolução é uma representação com resolução degradada, que cobre a mesma extensão da representação original, porém com um número menor de elementos. Os índices *i* de uma pirâmide de

multi-resolução costumam ser chamados de *níveis* na pirâmide, de tal forma que quanto mais alto o nível da pirâmide menor o número de elementos e mais degradada, em relação a original, é a sua resolução. As técnicas de particionamento e multiresolução devem ser usadas conjuntamente, ou seja, as versões com resolução degradada da representação devem ser também particionadas da mesma forma que a resolução original (ver Figura 4.4).

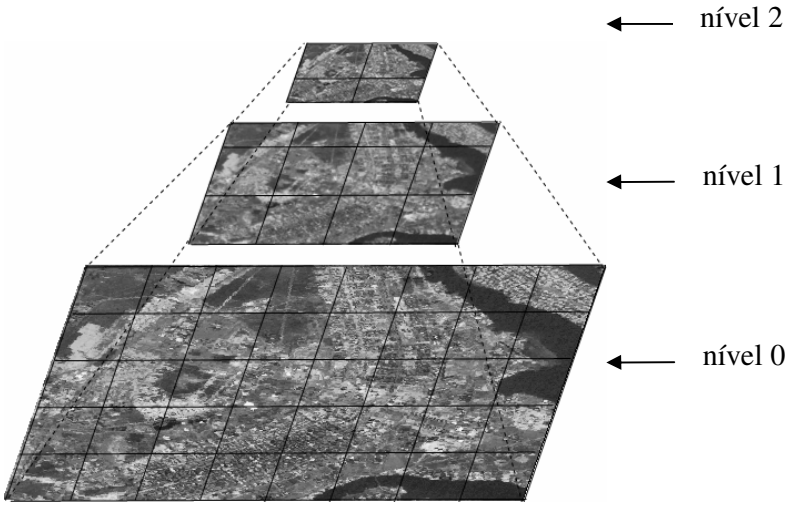


Figura 4.4 – Pirâmide de multiresolução de uma representação.

Essa combinação permite resolver duas questões contraditórias: capacidade de armazenamento de um volume grande de dados e demanda por alto desempenho no acesso a eles. As aplicações passam ao subsistema as informações necessárias para que esse decida qual representação, dentro de uma pirâmide de multiresolução, é mais apropriada, ou suficiente, e dentro dessa qual conjunto de blocos é suficiente para recobrir uma região de interesse.

A criação da pirâmide de multiresolução pode utilizar diferentes métodos para a geração de cada representação em resolução degradada, utilizando alguma técnica de re-amostragem como, por exemplo: vizinho mais próximo, bilinear, ou por convolução cúbica (Parker *et al.*, 1983). O método de reamostragem por vizinho mais próximo é o mais rápido e gera apenas valores dentro do contradomínio da representação original. Porém, especialmente no caso de representações de imagens de sensoriamento remoto,

esse método pode gerar artefatos visuais indesejáveis. Os métodos de reamostragem bilinear ou por convolução cúbica, por envolverem interpolação, podem gerar valores diferentes dos valores existentes na representação original. Porém, seu resultado pode ser visualmente mais agradável, no caso de imagens de sensoriamento remoto, ou representar variações espaciais mais suaves, que melhor representam fenômenos naturais, no caso de outros tipos de geo-campos. Vale lembrar que, os métodos de re-amostragem que envolvem interpolação de valores não podem ser aplicados em geo-campos cujo contradomínio é uma medida nominal, ou seja, são aplicáveis somente em geo-campos com contradomínio numérico (inteiros ou reais). Assim, ainda que possam gerar valores que não estavam associados a nenhum elemento da representação original, ainda pertencem ao mesmo contradomínio (inteiros ou reais).

4.2.4 Memória temporária

Uma vez definidos o sistema de particionamento das representações e sua indexação espacial dentro do banco de dados, o sistema de consulta a essa representação deve incluir um sistema de cache, ou seja, de armazenamento temporário em memória, dos blocos recuperados em uma determinada seleção. Esse sistema tem por finalidade evitar consultas ao SGBD para acessos consecutivos ao mesmo elemento da representação, ou a elementos diferentes de um mesmo bloco, uma vez que essa é uma operação custosa. Inspirado no modelo de memória virtual dos sistemas operacionais, o sistema de memória temporária mantém um *array* associativo ou mapa, na memória principal, entre identificadores de blocos e uma posição de memória com seu conteúdo. A chave do *array* associativo é a mesma identificação de blocos usada para armazená-lo no banco de dados.

A memória temporária é caracterizada pelo número de blocos que pode armazenar simultaneamente, valor que pode ser definido por uma aplicação cliente do sistema levando em conta os recursos de memória disponíveis, ou mesmo o padrão de acesso à representação esperado. Por exemplo, supondo que a representação vá ser acessada sequencialmente por linha, é interessante a manutenção na memória temporária de, pelo menos, o número de blocos necessários para formar uma linha da representação. Uma

maneira de melhorar ainda mais a eficiência do uso da memória temporária é a manutenção da identificação do bloco corrente dentro da memória temporária, uma vez que, tipicamente, os elementos dentro de um mesmo bloco são acessados sequencialmente, o bloco corrente pode otimizar o custo de buscar um bloco na memória temporária.

O acesso do elemento $RM(i, j)$ de uma representação RM é feito da seguinte forma:

1) A partir de (i, j), encontrar a identificação *ID* do bloco ao qual o elemento pertence.

2) Verificar se *ID* é o bloco corrente na memória temporária. Se for retornar o valor do elemento (i, j).

3) Se *ID* não for o bloco corrente, verificar se ele está em outras posições da memória temporária. Se existir, transformá-lo no bloco corrente e retornar o valor do elemento (i, j).

4) Se o bloco não existir na memória temporária, verificar se a memória temporária possui espaço para mais um bloco:

    a) Em caso afirmativo, recuperar o bloco com a identificação *ID* do SGBD, trazer para a memória temporária e torná-lo o bloco corrente e retornar o valor do elemento (i, j).

    b) Em caso negativo, escolher um dos blocos para ser descartado e trazer para a memória temporária e torná-lo o bloco corrente e retornar o valor do elemento (i, j).

4.2.5 Compressão

Devido ao grande volume de dados que as representações matriciais para geo-campos ocupam, uma funcionalidade importante é a capacidade de compressão da representação antes do seu armazenamento no SGBD. Especialmente no caso da criação do conjunto

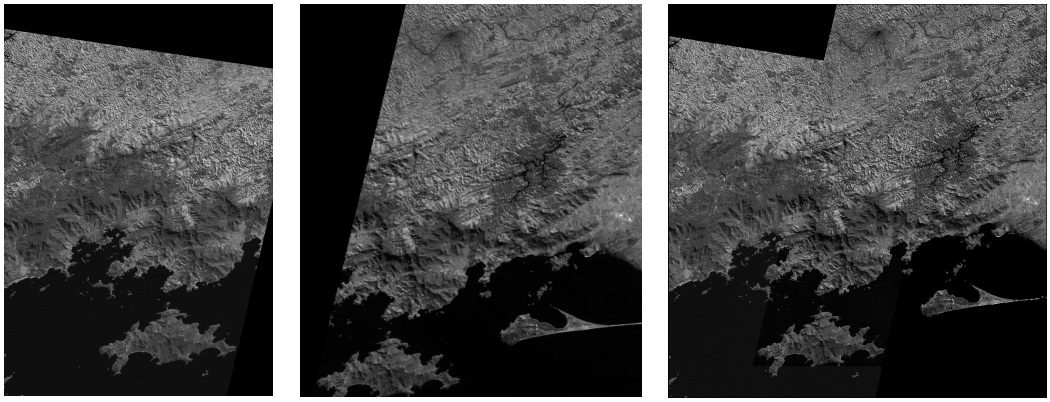
de representação em multi-resolução, isso é particularmente desejável, pois cada representação, além da original, implica em um volume extra a ser armazenado. Nesse caso, considera-se que cada bloco será comprimido individualmente.

Diferentes algoritmos de compressão podem ser usados, escolhidos de acordo com a sua capacidade de redução ou ainda características próprias quanto a perda ou aplicabilidade. O método JPEG de compressão e uma variação do método LZ77 (Ziv; Lempel, 1977) disponível na biblioteca Zlib[7], são exemplos de algoritmos de compressão que podem ser usados para comprimir representações. O JPEG é um método de compressão com perda, aplicável somente às representações matriciais onde o domínio é composto de valores representáveis em 8 bits; o método implementado na Zlib é um método sem perda, aplicável a representações com quaisquer contradomínio.

4.2.6 Mosaico

Outro requisito que deve ser levado em consideração quando se trata do armazenamento de representações matriciais em bancos de dados refere-se ao suporte a operações de inserção em uma representação já criada, nesse caso chamadas de operações de mosaico. O termo mosaico é derivado do exemplo típico de concatenação de diversas cenas de imagens de sensoriamento remoto, distribuídas em arquivos separados, que devem ser unidas para formar uma única representação dentro do banco de dados, como mostra a Figura 4.5. Genericamente, o mosaico de representações deve ser entendido como a concatenação de duas ou mais representações, com o mesmo contradomínio e com uma referência espacial coerente, formando uma única representação cujo domínio contenha o domínio das duas representações sendo concatenadas.

[7] Zlib é uma biblioteca de compressão de dados, livre de patentes e disponível para diversas plataformas e sistemas operacionais. Sua descrição e seu código podem ser encontrados em www.zlib.org.



(a) (b) (c)

Figura 4.5 – Mosaico de duas cenas de imagens de sensoriamento remoto: (a) e (b) cenas separadas e (c) mosaico resultante.

4.2.7 Metadados

Uma outra classe de informações sobre as representações são os metadados, ou seja, informações adicionais sobre o geo-campo, principalmente para facilitar a construção de ferramentas de consulta baseadas em conteúdo. Podem ser armazenados por exemplo, estatísticas como valor mínimo, máximo, médio ou histogramas de geo-campos numéricos (ordinal, intervalar ou razão), ou descrições semânticas do contradomínio de geo-campos nominais. Por isso, cada representação deve possuir uma identificação única dentro do banco de dados.

4.3 Modelo do bancos de dados

Uma vez levantados os requisitos gerais e apontadas soluções de forma abstrata, essa seção descreve um modelo de dados relacional para o armazenamento de representações em bancos de dados relacionais com suporte ao armazenamento de BLOBs (ou *Binary Long Objects*).

O modelo relacional propõe organizar os dados em relações (ou tabelas), cujos atributos (ou colunas) são de um tipo de dados específico, como números inteiros, de ponto flutuante, cadeias de caracteres e BLOBs (Ferreira *et al.*, 2005). Os tipos de dados, com exceção do tipo BLOB, oferecem uma variedade de operações, como operações de

conversão entre tipos, manipulação textual e operações com data. Os sistemas de gerência de banco de dados *relacionais* são aqueles que seguem o modelo relacional. Estes sistemas tipicamente adotam a linguagem SQL como linguagem de consulta.

Conforme a discussão da seção 4.2, o armazenamento em SGBDs relacionais de representações matriciais, especialmente representações em multiresolução com particionamento, requer a definição das relações, e seus atributos, que serão utilizadas para representar os blocos e os parâmetros da representação, e o relacionamento de multiresolução entre representações.

O modelo adotado para as representações matriciais é parte de um modelo conceitual mais genérico para bancos de dados geográficos, que define como os dados geográficos, geo-campos ou geo-objetos, estão organizados. Um modelo largamente utilizado para bancos de dados geográficos baseia-se na noção de *camada*, ou *plano de informação*, definida como uma agregação de geo-campos ou geo-objetos, ou ambos, de uma mesma região do espaço e compartilhando o mesmo sistema de referência espacial. Esse modelo é flexível o suficiente para permitir que um plano de informação agregue representações matriciais e vetoriais de geo-campos, ainda que esse trabalho esteja concentrado nas representações matriciais de geo-campos. Este modelo é usado, por exemplo, nas extensões espaciais Oracle Spatial e PostGIS.

Considera-se que uma representação matricial armazenada em um banco de dados está sempre associada a um plano de informação. Um plano de informação pode conter assim uma ou mais representações ou pirâmides de multiresolução (ver Figura 4.6). Como a representação matricial não se limita somente ao conjunto de blocos que contém os valores do geo-campo, o sistema de armazenamento das representações deve ser capaz de persistir seus parâmetros geométricos, geográficos e de contradomínio para recuperar a representação conforme definida anteriomente. Os parâmetros relativos a características de armazenamento particionado e em multiresolução também devem ser armazenadas no banco de dados e incluem: o tipo de particionamento, indicação do algoritmo de compressão, o número de níveis e quais os fatores que caracterizam as representações em multiresolução.

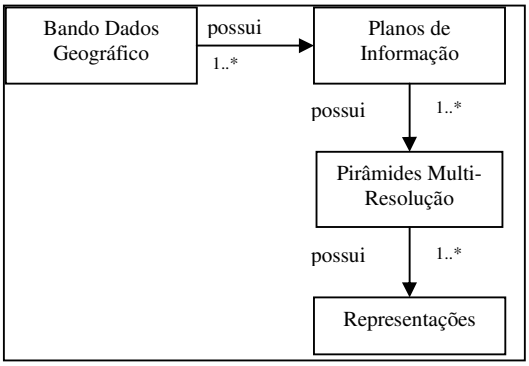


Figura 4.6 – Banco de dados geográfico com pirâmides de multi-resolução.

O esquema lógico apresentado na Figura 4.6 deve ser mapeado para um esquema físico, ou seja, uma estruturação de tabelas dentro do banco de dados. Um modelo para esse mapeamento está mostrado na Figura 4.7.

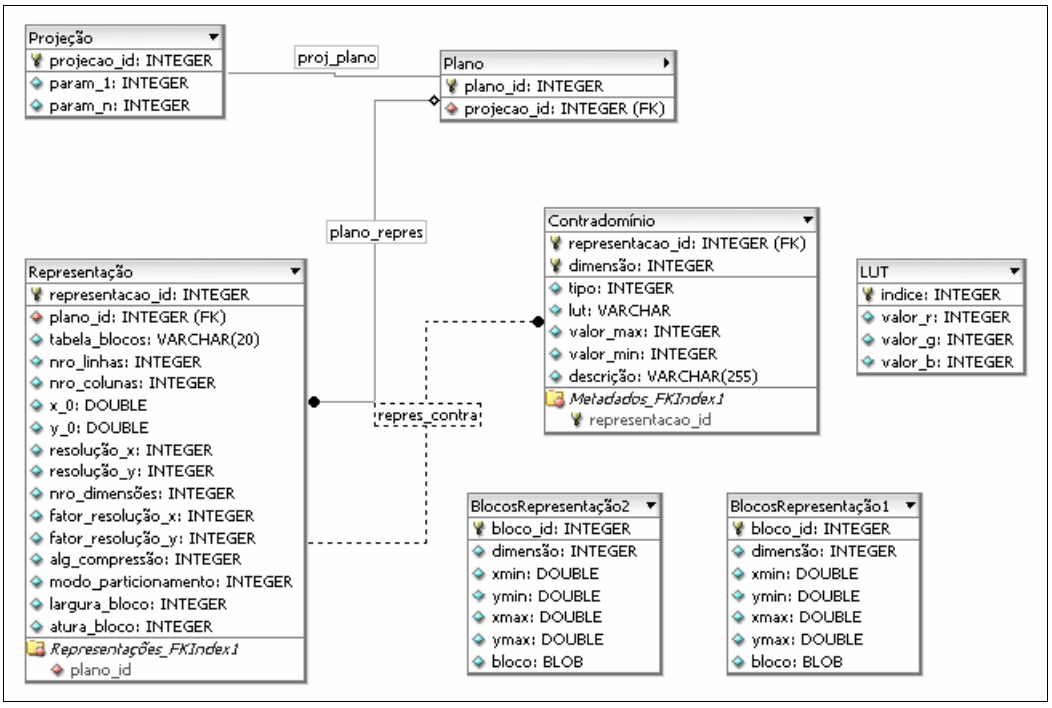


Figura 4.7 – Esquema de armazenamento de representações matriciais.

Observa-se que cada representação R é armazenada em uma relação ou tabela diferente T_R (na Figura 4.7, temos por exemplo as tabelas `BlocosRepresentação1` e `BlocosRepresentação2`). Cada registro da tabela T_R armazena um bloco da representação R, em uma dimensão do domínio V. Para cada bloco é armazenada tanto sua identificação única quanto seu retângulo envolvente. A identificação única, para o caso de particionamento alinhado regular pode ser derivada de uma curva de preenchimento como mostrado na seção 4.2.2. O armazenamento explícito do retângulo envolvente é útil para o caso de particionamentos não regulares. Nesse caso devem ser criados índices espaciais sobre o retângulo envolvente e não sobre a identificação única. As informações sobre o contradomínio são relativas a cada dimensão por isso são armazenadas em uma tabela separada.

4.4 Subsistema de armazenamento de representações

Essa seção mostra como o armazenamento de representações matriciais em bancos de dados relacionais está representado dentro do subsistema mostrado no Capítulo 3.

O conceito de memória temporária é implementado através da classe chamada `VirtualMemoryDecoder` cujo objetivo é otimizar o acesso a representações que permitem particionamento. A classe mantém em memória um mapa de blocos, representados pela classe `RasterBlock`, e é capaz de localizar dentro de um bloco um elemento da representação. A Tabela 4.1 mostra os métodos definidos pela classe `VirtualMemoryDecoder`. Observa-se que essa classe não pode ser instanciada, pois os métodos `getRasterBlock` e `putRasterBlock` são virtuais e representam o acesso efetivo a blocos da representação. A memória temporária possui uma capacidade pré-definida, ou seja, é limitada quanto ao número de blocos que pode manter simultaneamente. O método `getFreeBlock` é responsável pelo mecanismo de paginação da memória temporária. Quando o subsistema solicita o acesso a um bloco que ainda não está na memória temporária, caso não existam posições livres para acomodá-lo, realiza uma operação de substituição de blocos, que consiste na escolha de um bloco para ser descartado liberando seu espaço para o novo bloco solicitado.

TABELA 4.1 – Métodos classe `VirtualMemory`.

Método	Descrição
getRasterBlock(block_id)	Carrega um bloco do armazenamento permanente
putRasterBlock(block_id)	Salva um bloco no armazenamento permanente
getElementInBlock(block_id,i,j)	Recupera o valor do elemento (i,j) no bloco block_id
setElementInBlock(block_id,i,j)	Insere o valor do elemento (i,j) no bloco block_id
getFreeBlock	Recupera um espaço na memória temporária para ser alocado a um bloco
insertBlock(block_id)	Insere um bloco na memória temporária

A classe `DatabaseDecoder` é uma especialização da classe `RasterDecoder` que implementa as funcionalidades específicas de mapeamento entre elementos da representação e os blocos, e a sua recuperação ou armazenamento no banco de dados, utilizando a classe `VirtualMemory` para gerenciá-los em memória. A classe `DatabaseDecoder` é responsável por conhecer o modelo conceitual do banco de dados geográfico onde a representação está armazenada, até chegar na relação onde os blocos estão armazenados e recuperá-los utilizando os mecanismos de indexação definidos. A interface `CompressionStrategy` representa um algoritmo de compressão de blocos para seu armazenamento no SGBD. Cada algoritmo de compressão é uma instância concreta dessa classe. A Figura 4.8 mostra o relacionamento entre essas classes.

A Tabela 4.2 mostra os métodos que a `DatabaseDecoder` exporta, além daqueles já definidos na classe base `RasterDecoder`. Os métodos implementam a interação do decodificador com o modelo conceitual do banco onde a representação está armazenada, localizando-a em relação a um plano de informação, unicamente identificado dentro do banco de dados. Essa classe esconde do restante do subsistema detalhes do modelo físico, ou seja, das relações, atributos e sistema de indexação definido no armazenamento.

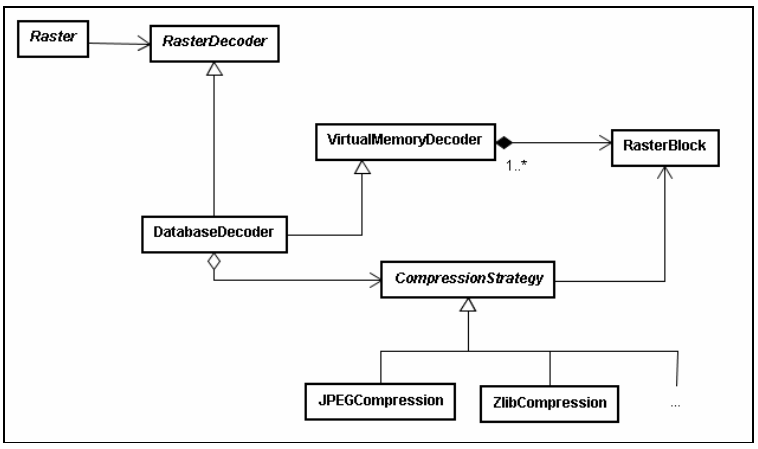


Figura 4.8 – A classe `DatabaseDecoder` e seus relacionamentos.

A seleção por janela é uma consulta espacial típica em bancos de dados geográficos. Consiste em, dado um retângulo *R* com os lados paralelos aos eixos do domínio espacial e um conjunto de objetos espaciais *D*, determinar todos os objetos em *D* cujas geometrias estão contidas em *R* (Ferreira *et al.*, 2005). No caso de uma representação matricial *RM* a seleção por janela consiste em recuperar todos os elementos de *RM* cuja localização está dentro de *R*. Como os blocos da representação não podem ser divididos internamente dentro do banco de dados, essa consulta significa a recuperação de todos os blocos cuja extensão intercepta o retângulo *R*. A seleção espacial funciona como uma etapa de redução sobre o espaço de busca dos elementos de *RM*; porém, cada elemento de cada bloco deve ser testado em memória quanto se está contido no retângulo *R*.

O resultado de uma consulta a uma relação em um banco de dados relacional é um conjunto de tuplas, ou registros, da relação que atendem ao critério especificado na consulta. A classe `DatabaseDecoder` fornece um mecanismo de acesso ao resultado de uma seleção por janela e consulta ao seu resultado através de dois métodos: o método `windowQuerySelecion` disponibiliza o conjunto de registros, nesse caso blocos, resultante de uma seleção por janela, um a um em seqüência ordenada, escondendo detalhes do modelo de armazenamento ou da SQL usada para a consulta. Subseqüentes chamadas ao método `fetchRasterBlock` retornam um bloco e uma valor indicando se existem mais blocos resultantes na seleção. A qualquer momento a seleção corrente pode ser descartada através da chamada ao método `clearWindowQuerySelection`.

Cada bloco da seleção por janela pode ser utilizado e descartado, ou armazenado na memória temporária, ou em alguma outra estrutura de armazenamento de blocos definida pela aplicação.

TABELA 4.2 – Métodos classe `DatabaseDecoder`.

Método	Descrição
`block_id = codifyId(i,j)`	Calcula a identificação do bloco que contém certo elemento da representação
`getRasterBlock(block_id)`	Recupera um bloco do banco de dados
`putRasterBlock(block_id)`	Salva um bloco no banco de dados
`windowQuerySelecion(box)`	Recupera um conjunto de blocos que intercepta uma região
`fetchRasterBlock`	Recupera sequencialmente os blocos selecionados em uma seleção por região
`clearWindowQuerySelection`	Libera a seleção de blocos

A seleção por janela foi mostrada nessa seção como uma operação da especialização `DecoderDatabase` da classe dos decodificadores. Porém, como é uma das operações mais características da manipulação de representações matriciais, esse conjunto de métodos também faz parte da interface da classe `RasterDecoder` (descritos na Tabela 3.5). Dessa forma podem ser reimplementados em outros decodificadores de dados em arquivos em formatos que permitam particionamento, como é o caso dos formatos JPEG2000, GeoTiff ou MrSID.

CAPÍTULO 5

IMPLEMENTAÇÃO NA TERRALIB

Essa seção descreve uma implementação concreta do subsistema mostrado nos capítulos anteriores. Essa implementação faz parte da biblioteca TerraLib, uma biblioteca para a construção de aplicativos geográficos e é o principal resultado dessa tese.

5.1 A biblioteca TerraLib

5.1.1 Arquitetura da biblioteca

A Figura 5.1 mostra a arquitetura geral da biblioteca. Existe um módulo central, chamado *kernel*, composto de estruturas de dados espaço-temporais, suporte a projeções cartográficas, operadores espaciais, e especificações de interfaces de acesso a SGBDs e arquivos de dados geográficos, além de mecanismos de controle de visualização. No módulo composto de *drivers* as interfaces genéricas são implementadas em classes. Esse módulo também contém rotinas de decodificação de dados geográficos em formatos abertos e proprietários. As funções de análise espacial são implementadas utilizando as estruturas do *kernel*. Finalmente, sobre esses módulos podem ser construídas diferentes interfaces aos componentes da TerraLib para diferentes ambientes de programação além da própria linguagem C++, como por exemplo Java ou COM.

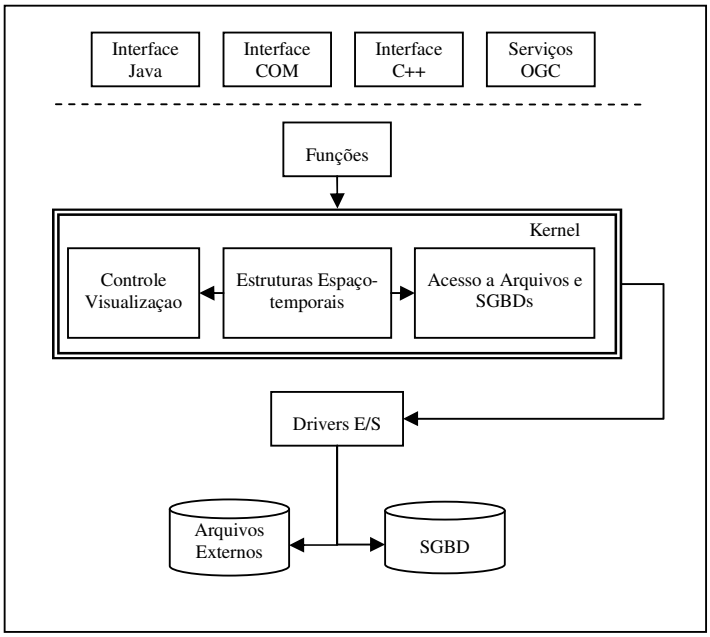


Figura 5.1 – A arquitetura da TerraLib.

5.1.2 O modelo conceitual da TerraLib

A TerraLib propõe não somente um modelo de armazenamento de dados geográficos em um SGBD, mas também um modelo conceitual de banco de dados geográfico, sobre o qual são escritos seus algoritmos de processamento. As principais entidades que formam o modelo conceitual são:

1) *Banco de Dados*: representa um repositório de informações contendo tanto os dados geográficos quanto o seu modelo de organização. Um banco de dados pode ser materializado em diferentes SGBDs, comerciais ou de domínio público. O único requisito da TerraLib é que o SGBD possua a capacidade de armazenar campos binários longos, ou uma extensão própria capaz de criar tipos abstratos espaciais, e que possa ser acessado por alguma camada de *software*;

2) *Plano de Informação*: representa uma estrutura de agregação de um conjunto de informações espaciais que são localizadas sobre uma região geográfica;

3) *Representação*: trata do modelo de representação da componente espacial dos dados de um plano e pode ser baseado em geometrias vetoriais ou matriciais;

4) *Projeção Cartográfica*: serve para representar a referência geográfica da componente espacial dos dados geográficos;

5) *Tema*: serve principalmente para definir uma seleção sobre os dados de um plano;

6) *Vista*: serve para definir uma visão particular de um usuário sobre o banco de dados. Uma vista define quais temas serão processados ou visualizados conjuntamente.

As entidades que formam o modelo conceitual estão representadas tanto nas classes que compõe a biblioteca (Figura 5.2) quanto nas tabelas do banco de dados (Figura 5.3).

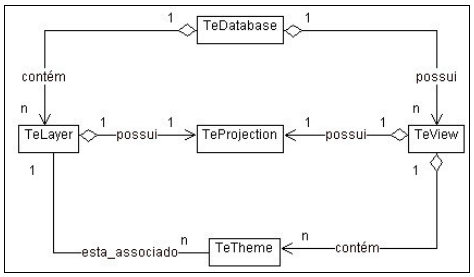


Figura 5.2 – Relacionamento entre as classes do modelo conceitual.

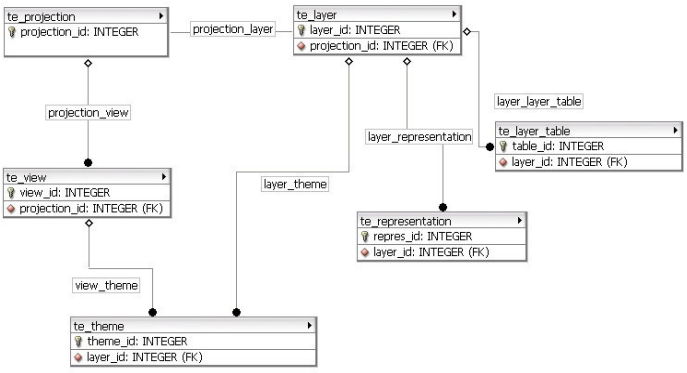


Figura 5.3 – Modelo de um banco de dados TerraLib.

5.1.3 O modelo de geometrias vetoriais

As geometrias vetoriais de TerraLib são construídas a partir de coordenadas bi-dimensionais representadas na classe chamada de `TeCoord2D`, combinadas através do padrão composite como mostra a Figura 5.4, formando as geometrias de pontos, linhas, anéis e polígonos. Todas as geometrias podem ser combinadas em conjuntos formando os conjuntos de pontos, de linhas e de polígonos.

Resumindo pode se dizer que em um banco TerraLib existem planos de informação os quais possuem representações para a sua componente espacial. Para o caso de geo-objetos as representações são formadas por conjuntos de geometrias vetoriais do tipo `TeGeometry`.

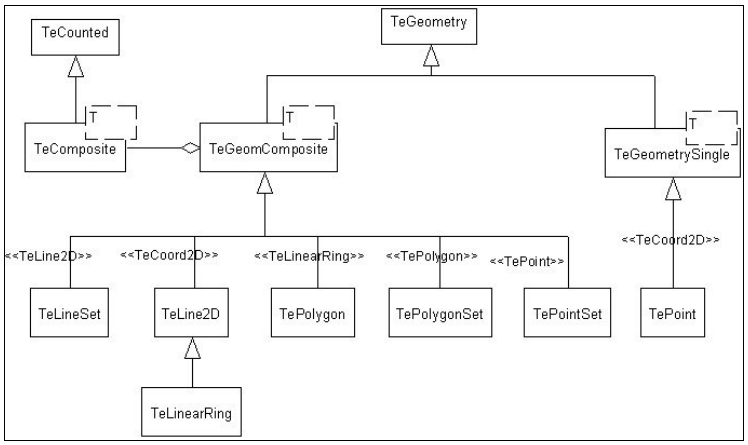
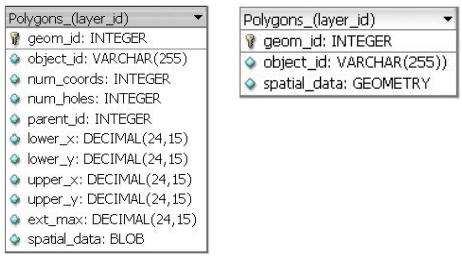


Figura 5.4 – Tipos geométricos vetoriais da TerraLib.
FONTE: Adaptado de Queiroz (2003).

O modelo de armazenamento das geometrias no banco de dados leva em conta questões relativas à eficiência no seu armazenamento e na sua recuperação, e também a existência de extensão espacial no SGBD. Para os SGBDs sem extensão espacial as tabelas de geometrias do tipo linhas e polígonos armazenam, além do campo do tipo BLOB, campos para armazenar o mínimo retângulo envolvente da geometria, os quais são indexados por mecanismos convencionais pelo SGBD. Para os SGBDs com extensão espacial, a coluna com o dado espacial é indexada espacialmente pelo

mecanismo oferecido pela extensão. A Figura 5.5 mostra a diferença entre uma tabela de geometria do tipo polígono criada em um banco sem extensão espacial e em um banco com extensão espacial. No segundo caso o tipo "GEOMETRY" representa o tipo espacial fornecido pela extensão.



(a) (b)

Figura 5.5 – Tabelas para o armazenamento de geometrias do tipo polígono: (a) em bancos sem extensão espacial e (b) em bancos com extensão espacial. FONTE: Adaptado de Ferreira (2002).

5.1.4 A interface genérica de acesso a um banco de dados

A classe `TeDatabase` é uma interface abstrata de manipulação do banco de dados através da qual é possível inserir, alterar e recuperar as entidades do modelo conceitual e os dados geográficos. Ela contém todos os métodos para criar as tabelas de metadados, as tabelas de geometrias e de atributos convencionais. A classe `TeDatabase` também é capaz de submeter ao banco comandos SQL resolvendo variações entre as diferentes SQL implementadas nos SGBDs (Ferreira *et al.*, 2002). Essa classe foi extendida, nesse trabalho, para fornecer o suporte a manipulação de representações matriciais em bancos de dados.

A TerraLib atualmente fornece uma implementação concreta para a interface para os gerenciadores MySQL (Widenius *et al.*, 2002), Oracle (Ravada *et al.*, 1999), Access e

SQLServer[8] via a biblioteca MS-ADO[9], PostgreSQL e PostGIS (Santilli *et al.*, 2005), em uma arquitetura de classes mostrada na Figura 5.6.

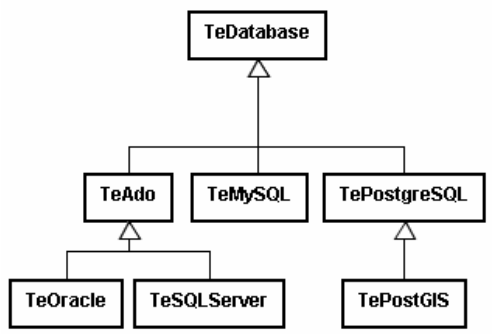


Figura 5.6 – Implementações concretas da interface `TeDatabase`.

FONTE: Adaptado de TerraLib (2006).

A classe `TeDatabase` fornece um conjunto de operações espaciais em seus métodos, que possuem uma implementação *default* utilizando algoritmos geométricos em memória e que são reimplementados nos *drivers* específicos para SGBDs com extensão espacial para utilizar os operadores definidos na própria extensão (Ferreira *et al.*, 2002).

5.2 A instanciação do subsistema na TerraLib

A componente de interface do subsistema proposto no capítulo 3 foi implementada diretamente em classes da TerraLib sendo chamadas de `TeRaster` e `TeRasterParams`, com a definição dos métodos mostrados na seção 3.2. A classe de parâmetros utiliza a interface `TeProjection`, que já existia na TerraLib e que possui especializações para as principais projeções cartográficas usadas em SIGs. Também foram aproveitadas as classes que representam uma coordenada bidimensional e um retângulo envolvente e que já existiam na TerraLib (`TeCoord2D` e `TeBox`, respectivamente).

[8] SGBD relacional comercial produzido pela empresa Microsoft, maiores informações podem ser obtidas no endereço http://www.microsoft.com/sql.
[9] ADO- *ActiveX Data Objects* é uma objeto COM da Microsoft usado para acessar dados em diferentes fontes, incluindo SGBDs, que pode ser usado através de diferentes linguagens de programação. Maiores informações podem ser obtidas no endereço http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/dasdkadooverview.asp

Para resolver as diferenças de tipos entre cada dimensão do contradomínio, nos métodos `getElement` e `setElement` da classe `TeRaster` o tipo do parâmetro relativo ao valor do elemento é sempre do tipo `double`, uma vez que todos os tipos numéricos podem ser convertidos para dupla precisão sem perda de precisão. O mesmo é valido para os métodos que atribuem e recuperam conjuntos de valores relativos a todas as dimensões do contradomínio.

O modelo de iterador que percorre uma representação em uma ordem pré-definida, por linhas ou por colunas, foi implementado como um iterador interno da classe `TeRaster`. Seguindo esse modelo, Ferreira (2003) estendeu essa classe para um iterador que percorre os elementos da representação cujas localizações estão dentro de uma geometria do tipo polígono da TerraLib.

Para a decodificação de formatos de representações em arquivos, a interface de decodificadores foi acrescentada à biblioteca, como a interface `TeDecoder`. A identificação de cada formato é feita por um nome, que pode ser instanciada a partir de extensões de arquivos. A fábrica de decodificadores juntamente com o dicionário de identificadores foi incluída em uma função da biblioteca chamada `TeInitRasterDecoders`. A Tabela 5.1 mostra a lista de decodificadores concretos implementados na TerraLib.

A classe `TeDatabaseDecoder` é o decodificador para representações matriciais em bancos de dados TerraLib. As representações matriciais são consideradas como mais um tipo de geometria, ou seja, como uma representação geométrica de um plano de informação dentro do banco de dados TerraLib. Essa classe implementa apenas estratégias de particionamento alinhado regular:

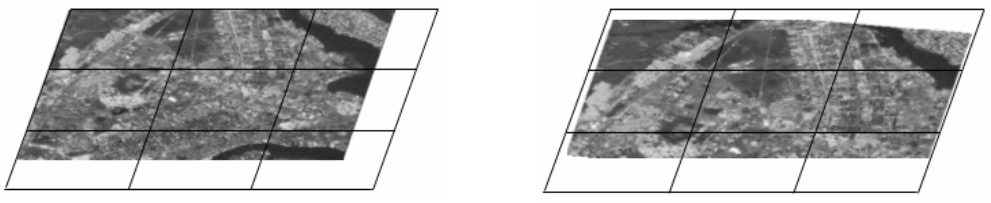
1) *domínio matricial*: os planos que definem que o pariticionamento passam pela posição (0,0) de $M[m,n]$ e pelas posições (c,l), onde c e l são múltiplos de w e h maiores que 0;

2) *domínio geográfico*: os planos que definem o particionamento passam pelas posições (c,l) onde onde c e l são múltiplos de $w*R_X$ e $h*R_Y$ respectivamente. Ou seja, são múltiplos da largura do bloco em coordenadas geográficas.

TABELA 5.1 – Decodificadores concretos implementados na TerraLib.

Decodificadores	Identificador	Extensão	Aplicação
`TeDecoderMemory`	"MEM"	---	Representações matriciais em memória
`TeDecoderJPEG`	"JPEG"	".jpg" ".jpeg"	Representações matriciais em arquivos no formato JPEG.
`TeDecoderTIFF`	"TIFF"	".tif" ".tiff"	Representações matriciais em arquivos no formato GeoTIFF ou TIFF
`TeDecoderASCIIGRID`	"ASCIIGRID"	".ascii" ".txt"	Representações matriciais em arquivos no formato ASCII-GRID
`TeDecoderSPR`	"SPR"	".spr"	Representações matriciais em arquivos no formato ASCII-Spring.
`TeDecoderMemoryMap`	"MEMMAP"	".raw"	Representações matriciais em arquivos binários raw, limitados a 2 Gigabytes.
`TeDecoderFile`	"FILE"	".bin"	Representações matriciais em arquivos binários *raw* de qualquer tamanho.
`TeDecoderMrSID`	"MRSID"	".sid"	Representações matriciais em arquivos no formato MrSID *raw*
`TeDecoderDatabase`	"DB"	---	Representações matriciais em um banco de dados TerraLib.

A Figura 5.7 ilustra o particionamento de uma representação seguindo as duas estratégias. A identificação dos blocos é feita de acordo com sua ordem dentro do particionamento gerado.



(a) (b)

Figura 5.7 – Duas estratégias de particionamento: (a) baseada em $M[m,n]$ e (b) baseada na extensão espacial de RM.

Nos dois casos observa-se que o particionamento inclui blocos onde existem localizações para as quais não existem valores do geo-campo, a essas posições é atribuído o valor indicativo de falta de informação, como no caso das representações esparsas. A estratégia de particionamento baseada na extensão espacial da representação permite a construção de mosaicos, uma vez que a identificação dos blocos depende de coordenadas geográficas que não se alteram com a expansão da representação, decorrente da operação de mosaico.

A identificação dos blocos no caso da estratégia de particionamento no domínio matricial é feita seguindo uma curva de preenchimento por linha. O decodificador implementa as consultas por região encontrando os intervalos de identificadores que dentro interceptam a região de interesse. Para o caso do particionamento no domínio geográfico a consulta por região leva em conta os campos que contém o retângulo envolvente dos blocos e não o identificador único.

A memória temporária é inicialmente definida para conter o número de blocos necessários para acessar uma linha da representação, e possui uma estratégia de substituição de blocos bastante simplificada: quando existe necessidade de abrir posições na memória temporária o bloco menos recentemente utilizado é substituído.

A solução desenvolvida na TerraLib foi utilizada na construção de bancos de dados com mosaicos de fotografias de algumas cidades pela Funcate – Fundação de Ciência, Aplicações e Tecnologias Espacias que utiliza a TerraLib em seus projetos de gestão municipal. Os mosaicos são compostos de fotografias áreas com resolução de 0.25, 0.16 e 0.01 metros, armazenados em bancos de dados TerraLib construídas sobre os SGBDs SqlServer e PostreSQL + PostGIS. Por exemplo, a cidade de Santos foi recoberta por um mosaico de fotos de 0.1 metros de resolução que ocupa um armazenamento total de aproximadamente 8 Gbytes. O armazenamento de imagens particionadas e em multiresolução permite que esses bancos de dados possam ser acessados por aplicações web, ou aplicações multi-usuários dentro de um ambiente corporativo.

A criação das pirâmides de multiresolução está concentrada em uma função que atualmente pode utilizar duas estratégias para a reamostragem: a de vizinho mais próximo e a bilinear. Não existe limitação quanto ao número de níveis de resolução que podem ser criados.

A função de remapeamento serve de base para várias outras funções dentro da TerraLib, entre elas na importação de dados para o banco como mostra a Figura 5.8. As rotinas de visualização também utilizam a função de remapeamento, através da instanciação da interface de representação matricial como um decodificador para a área de desenho, associada ao pacote de visualização utilizada no sistema. A Figura 5.9 mostra o exemplo de uma instanciação do sistema de decodificação para a área de desenho fornecida por um sistema específico (no exemplo o pacote Qt[10]). A forma como as representações são visualizadas é controlada por transformadores de contradomínio que adequam o contradomínio do geo-campo as características do *display*.

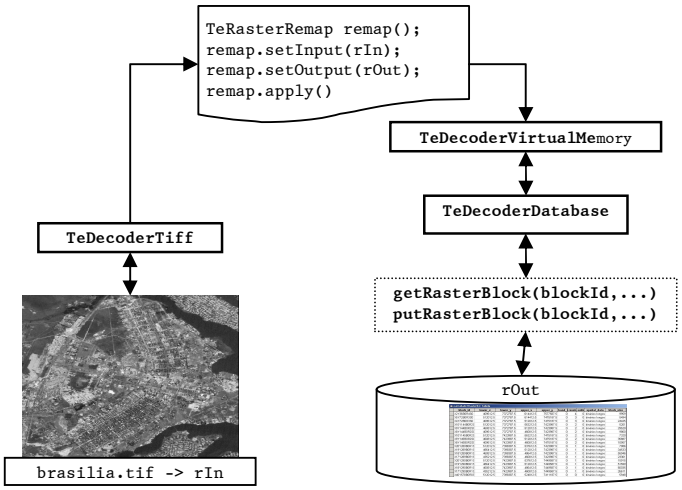


Figura 5.8 – Esquema da importação de uma representação.

[10] O Qt é um pacote de rotinas C++ para a criação de aplicativos da empresa Trolltech, que entre outros fornece uma biblioteca de classes para a construção de interfaces, incluindo áreas de desenho gráfico. Maiores informações podem ser obtidas no site: http://www.trolltech.com.

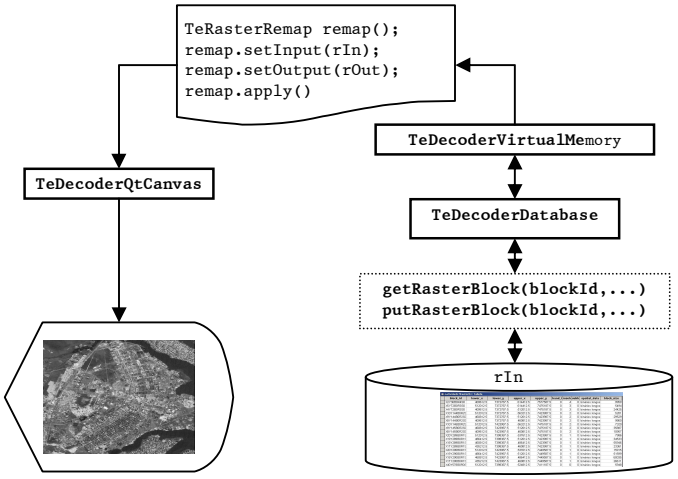


Figura 5.9 – Esquema de visualização de uma representação do banco de dados.

Uma parte do projeto TerraLib, é voltada para a criação de um repertório de algoritmos de processamento de imagens. Essa extensão chamada de TerraLibPDI é cliente do subsistema, de forma que todos os algoritmos são escritos utilizando a interface `TeRaster` e seus iteradores. Essa extensão também inclui funcionalidades previstas no pacote de serviços previstos na especificação OGC mostrada na seção 2.2, como a recuperação de uma área de interesse, ou a exportação de representações em diferentes formatos (TerraLib, 2006).

A interface `TeRaster` também foi usada para a integração da TerraLib com o pacote estatístico R[11], permitindo acrescentar aos aplicativos SIGs construídos sobre a TerraLib ferramentas estatísticas avançadas, não disponíveis no *kernel* da TerraLib. Andrade (2005) descreve como foi feita a implementação dessa interface e dá como exemplo a criação de uma superfície de Krigeagem, armazenada dentro de um banco de dados TerraLib como uma representação matricial. Na mesma linha, um trabalho em desenvolvimento está integrando a TerraLib à biblioteca de modelagem para distribuição espacial de espécies openModeller[12] que atualmente manipula dados de

[11] O R é um pacote que fornece uma linguagem e um ambiente para processamentos estatísticos. Maiores informações podem ser obtidas no site http://www.r-project.org.

[12] O openModeller é uma biblioteca de modelagem espacial que fornece um método uniforme para a modelagem de padrões de distribuição usando uma variedade de algoritmos de modelagem. Maiores informações podem ser obtidas no site http://openmodeller.sourceforge.net/.

entrada e saída matriciais armazenados em arquivos, permitindo novamente que esses possam ser obtidos, ou gerados, em bancos de dados TerraLib.

CAPÍTULO 6

CONCLUSÕES

6.1 Conclusões

Esta tese definiu, inicialmente, as noções de geo-campo e de representação de geo-campo e caracterizou mais detalhadamente a noção de representação matricial para geo-campos. A partir destas definições, propôs um subsistema capaz de manipular, de maneira unificada, várias instâncias de representações matriciais com características diversas. Esse subsistema prevê que os parâmetros que caracterizam as representações devem ser manipulados independentemente dos dados que formam as representações. Essa independência está prevista nas classes e interfaces propostas no subsistema, de forma que novas características, como novos formatos de arquivos de representação, possam ser incorporados ao sistema de maneira mais fácil. As decisões em termos de projeto de sistemas são documentadas, também com o intuito de facilitar a extensão do subsistema.

Em seguida, a tese abordou como representações matriciais podem ser armazenadas em SGBDs objeto-relacionais. Da análise feita, foram levantados alguns requisitos que consideram questões práticas de eficiência relacionadas ao armazenamento e recuperação desses dados em SGBDs. A tese especificou um esquema conceitual relacional para armazenamento de representações matriciais através de duas estratégias: criação de pirâmides de multiresolução e particionamento de grandes representações. A tese mostra que essa combinação de estratégias é uma solução promissora que permite atingir o grau de eficiência de manipulação desejado. Além disso, permite a exploração de características dos SGBDs em combinação com estratégias de otimização do lado das aplicações, para criar soluções eficientes e robustas em bancos de dados geográficos.

Por fim, uma implementação concreta do subsistema proposto foi desenvolvida como parte da biblioteca TerraLib, atendendo a um dos objetivos dessa tese, que é o da

criação de ferramentas para a construção de aplicações geográficas que sejam livre de licença e com código fonte aberto. A implementação foi totalmente incorporada à biblioteca e é utilizada como base para uma série de funcionalidades da TerraLib como a biblioteca de algoritmos de processamento digital de imagens de sensoriamento remoto. O subsistema proposta serviu como base também para interação da biblioteca TerraLib com outros sistemas, aumentando a disponibilidade de ferramentas disponíveis aos usuários de software livre para a construção de aplicativos geográficos.

6.2 Trabalhos futuros

A indexação de blocos através das curvas de preenchimento é a alternativa mais promissora para a recuperação eficiente de representações particionadas em bancos de dados sem extensão espacial. Os testes executados com os mosaicos de fotografias áreas mostraram bons resultados, mas ainda é necessário investigar em mais detalhe as estratégias do armazenamento otimizado dos blocos, que passa pelas questões levantadas em 4.2.2, através de estudos experimentais ou da utilização de técnicas de comparação publicadas na literatura, como a de Mokbel (2003). Combinada a essa necessidade, pretende-se investigar melhores estratégias de gerência da memória temporária, ou *buffer* de blocos, adaptáveis a padrões de acesso determinados por algoritmos de processamento.

Pretende-se estender o subsistema desenvolvido para outros tipos de representações para geo-campos, como os TINs. Existem na literatura diversos trabalhos que tratam da criação de multiresoluções para representações de geo-campos baseadas em TINs, como os de Yang (2005) e (Pedrini, 2001). Voigtmann (1997) e (Kidner *et al.*, 2000) apresentam inclusive algumas estruturas de dados para representar os TINs, que são mais adequadas para a construção de multiresoluções. Pretende-se estender o sistema de decodificadores para essas estruturas, além de estudar como adaptá-las para o seu armazenamento em bancos de dados. As representações de geo-campos por pontos amostrais e isolinhas podem se valer de estruturas de indexação disponíveis de geometrias vetoriais para geo-objetos (Davis Jr.; Ferreira, 2005a). Nesse caso, o desafio é fazer com que o subsistema incorpore, de maneira flexível, algoritmos de interpolação

necessários para atender a sua componente de interface, ou seja, como calcular o valor de um geo-campo em uma localização.

Pretende-se também definir métodos de armazenamento semelhantes ao MrSID, que definem formatos de arquivo com interfaces específicas, e que tenham os benefícios do armazenamento piramidal particionado, sem o uso de banco de dados. Esse estudo deverá levar em conta tanto o caso das representações matriciais, quanto vetoriais de geo-campos.

Seguindo o conceito da interface genérica de acesso a SGBDs da TerraLib, pretende-se construir uma interface entre a classe `TeDecoderDatabase` e a extensão espacial do Oracle Spatial 10g, que possui uma proposta de armazenamento de representações matriciais.

A TerraLib já possui uma especificação de armazenamento para espaços celulares baseada em estruturas vetoriais. Pretende-se criar um decodificador específico que integrar essa representação ao componente de interface do subsistema incluindo também as características de particionamento e multiresolução inexistentes no armazenamento atual.

REFERÊNCIAS BIBLIOGRÁFICAS

Abel, D. J.; Mark, D. M. A comparative analysis of some 2-dimensional orderings. **International Journal of Geographical Information Systems**, v. 4, n. 1, p. 21–31, 1990.

Adler, D. W. IBM® DB2® Spatial Extender - Spatial data within the RDBMS. In: Very-Large Database Conference, 27., 2001, Roma, Italy. **Proceedings...** Disponível em: http://www.cse.iitb.ac.in/dbms/Data/Conferences/vldb2k1/087_687.pdf. Acesso em: 23 dez. 2005.

Alexandrescu, A. **Modern C++ design**. Generic programming and design patterns applied. Upper Saddle River, NJ: Addison-Wesley, 2001. 352 p.

Andrade, P. R.; Ribeiro Jr., P. J.; Fook, K. D. Integration of statistics and geographic information systems: the R/TerraLib case. In: Brazilian Symposium on GeoInformatics, 7., 2005, Campos do Jordão, SP. **Anais Eletrônicos...** Nov. 2005. Disponível em: http://www.geoinfo.info/geoinfo2005/papers/P74.PDF. Acesso em: 25 Jan. 2006.

Asano, T.; Ranjan, D. R., T. ; Welzl, E.; Widmayer, P. Space-filling curves and their use in the design of geometric data structures. **Theoretical Computer Science**, v. 181, n. 1, p. 3–15, 1995.

Austern, M. **Generic programming and the STL**: using and extending the C++ Standard Template Library. Reading, MA: Addison-Wesley, 1998. 548 p.

Bartholdi, J. J.; Goldsman, P. Vertex-labeling algorithms for the Hilbert space-filling curve. **Sofware**: practice and experience, v. 31, n. 5, p. 395–408, 2001.

Batty, M. GeoComputation using cellular automata. In: Openshaw, S.; Abrahart, R. J. (Eds.). **GeoComputation**. London: Taylor&Francis, 2000, p. 95–126.

Baumann, P.; Dehmel, A.; Furtado, P.; Ritsch, R.; Widmann, N. The multidimensional database system RasDaMan. In: ACM International Conference on Management of Data (SIGMOD), 1998, Seattle, USA. **Proceedings...** New York: ACM Press, 1998. p. 575–577.

Câmara, G. **Modelos, linguagens e arquiteturas para bancos de dados geográficos** Computação Aplicada. 265 p. Doutorado (Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1995.

___. Representação computacional de dados geográficos. In: Casanova, M. A.; Câmara, G.; Davis Jr., C. A.; Vinhas, L.; Queiroz, G. R. (Eds.). **Bancos de dados geográficos**. Curitiba: MundoGeo, 2005. Cap. 1, p. 11–52.

Câmara, G.; Fonseca, F. Information policies and open source software in developing countries. **Journal of the American Society for Information Science and Technology** No prelo.

Câmara, G.; Souza, R. C. M.; Pedrosa, B.; Vinhas, L.; Monteiro, A. M. V.; Paiva, J. A.; Carvalho, M. T.; Gattass, M. TerraLib: technology in support of GIS innovation. In:

Simpósio Brasileiro em Geoinformática, 2., 2000, São Paulo, SP. **Anais...** Disponível em: http://www.tecgraf.puc-rio.br/geoinfo2000/anais/019.pdf. Acesso em: 06 jan. 2006.

Câmara, G.; Souza, R. C. M.; Pedrosa, B. M.; Vinhas, L.; Monteiro, A. M. V.; Paiva, J. A. C.; Carvalho, M. T.; Raoult, B. Design patterns in GIS development: the TerraLib experience. In: Simpósio Brasileiro de GeoInformatica, 3., 2001, Rio de Janeiro, RJ. **Anais ...** Disponível em: http://www.geoinfo.info/portuguese/geoinfo2001/papers/pag93a99.pdf. Acesso em: 12 dez. 2005.

Carneiro, T. G. S. **Uma arquitetura para modelagem ambiental empírica baseada nas teorias dos autômatos celulares, híbridos e situados.** São José dos Campos: INPE, 2004. 55 p. Disponível em: http://www.dpi.inpe.br/gilberto/teses/proposta_tiago.pdf. Acesso em: 20 dez. 2005.

Chrisman, N. A transformational approach to GIS operations. **International Journal of Geographical Information Science**, v. 13, n. 7, p. 617–637, Out. 1999.

Christopoulos, C.; Skodras, A.; Ebrahimi, T. The JPEG2000 still image coding system: an overview. **IEEE Transactions on Consumer Electronics**, v. 46, n. 4, p. 1103–1127, 2000.

Couclelis, H. People manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS. In: Frank, A. U.; Campari, I.; Formentini, U. (Eds.). **Theories and methods of spatio-temporal reasoning in geographic space**. Berlin: Springer, 1992, p. 65–77.

d'Alge, J. C. L. Cartografia para geoprocessamento. In: Câmara, G.; Davis Jr., C.; Monteiro, A. M. V. (Eds.). **Introdução à ciência da geoinformação**. São José dos Campos: INPE, 2004. Cap. 6. Disponível em: http://www.dpi.inpe.br/gilberto/livro/introd/. Acesso em: 6 jan. 2006.

Davis Jr., C.; Queiroz, G. R. Métodos de acesso para dados espaciais. In: Casanova, M. A.; Câmara, G.; Davis Jr., C. A.; Vinhas, L.; Queiroz, G. R. (Eds.). **Bancos de dados geográficos**. Curitiba, PR: MundoGEO, 2005. Cap. 6, p. 213–231.

de Floriani, L. Surface representations based on triangular grids. **The Visual Computer**, v. 3, n. 1, p. 27-50, 1987.

Egenhofer, M.; Franzosa, R. Point-set topological spatial relations. **International Journal of Geographical Information Systems**, v. 5, n. 2, p. 161–174, 1991.

Egenhofer, M. J.; Frank, A. U. Object-oriented modeling for GIS. **URISA Journal**, v. 4, n. 2, p. 3–19, 1992.

Farley, J. **Oracle Database 10g** – managing geographic raster data using GeoRaster. 2003. Oracle Technical White Paper. Disponível em: www.oracle.com/technology/products/spatial/pdf/10g_georaster_twp.pdf. Acesso em: 7 fev. 2006.

Ferreira, K. R.; Casanova, M. A.; Câmara, G.; Oliveira, O. F. Arquiteturas e linguagens. In: Casanova, M. A.; Câmara, G.; Davis Jr., C. A.; Vinhas, L.; Queiroz, G. R. (Eds.). **Bancos de dados geográficos**. Curitiba: MundoGeo, 2005. Cap. 5, p. 169–201.

Ferreira, K. R.; Queiroz, G. R.; Paiva, J. A. C.; Souza, R. C. M.; Câmara, G. Arquitetura de Software para Construção de Bancos de Dados Geográficos com SGBD Objeto-Relacionais. In: Simpósio Brasileiro de Banco de Dados, 17., 2002, Gramado, RS. **Proceedings...** Porto Alegre: UFRGS, 2002. p. 57–67.

Fonseca, F.; Egenhofer, M. Ontology-driven geographic information systems. In: ACM GIS Symposium, 7., 1999, Kansas City, USA. **Proceedings...** New York: ACM Press, 1999. p. 14–19.

Fonseca, F. T.; Egenhofer, M.; Agouris, P.; Câmara, G. Using ontologies for integrated geographic information systems. **Transactions in GIS**, v. 6, n. 3, p. 231–257, 2002.

Frank, A. U.; Kuhn, W. Specifying open GIS with functional languages. In: Egenhofer, M. J.; Herring, J. R. (Eds.). **Advances in Spatial Databases**. Berlin: Springer, v. 951, 1995, p. 184–195.

Gaede, V.; Guenther, O. Multidimensional Access Methods. **Computing Surveys**, v. 30, n. 2, p. 170–231, 1998.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Design patterns: elements of reusable object-oriented software**. Reading, MA: Addison-Wesley, 1995.

Goodchild, M. F. Geographical data modeling. **Computers and Geosciences**, v. 18, n. 4, p. 401–408, 1992.

___. Geographic information science and systems for environmental management. **Annual Review of Environment and Resources**, v. 28, p. 493–519, 2003.

Gordillo, S.; Balaguer, F.; Mostaccio, C.; Neves, F. D. Developing GIS applications with objects: a design patterns approach. **GeoInformatica** v. 3, n. 1, p. 7–32, 1999.

Karlsson, A. **GIS and spatial extensions with MySQL** 2004. Disponível em: http://dev.mysql.com/tech-resources/articles/4.1/gis-with-mysql.html. Acesso em: 10 jan. 2006.

Kemp, K. K.; Vckovski, A. Towards an ontology of fields. In: International Conference on GeoComputation, 3., 1998, Bristol, UK. **Proceedings...** [s.l.] MySQLLab, Disponível em: http://www.geocomputation.org/1998/60/gc_60.htm. Acesso em: 18 jan. 2006.

Köthe, U. Reusable software in computer vision. In: Jähne, B.; Haußecker, H.; Geißler, P. (Eds.). **Handbook on computer vision and applications**. San Diego: Academic Press, v. 3, 1999, p. 103–132.

Kottman, C. A. The Open GIS Consortium and progress toward interoperability in GIS. In: Goodchild, M. E.; Egenhofer, M.; Fegeas, R.; Kottman, C. A. (Eds.). **Interoperating Geographic Information Systems**. Norwell, MA: Kluwer, 1990, p. 39–54.

Laurini, R.; Bazzocco, J.; Gordillo, S.; Rossi, G.; Catalina, M. Designing adaptable geographic objects for mobile applications. In: International Conference on Web Information Systems Engineering Workshops (WISEW'03), 4., 2003, Lyon, França. **Proceedings...** Washington: IEEE Computer Society, p. 217–224.

Lawder, J. K.; King, P. J. H. Querying multi-dimensional data indexed using the Hilbert space-filling curve. **ACM SIGMOD Record**, v. 30, n. 1, p. 19–24, 2001.

McKee, L.; Buehler, K. (Ed.). **The OpenGIS® guide** – introduction to interoperable geoprocessing. Wayland, MA: Open GIS Consortium, Inc., 1998. 103 p. Disponível em: http://gis.geo.fhm.edu/klauer/gi/OpenGISGuide980629.pdf. Acesso em: 13 dez. 2005.

Mitasova, H.; Neteler, M. GRASS as open source free software GIS: accomplishments and perspectives. **Transactions in GIS**, v. 8, n. 2, p. 145–154, 2004.

Mokbel, M. F.; Aref, W. G.; Kamel, I. Analysis of multi-dimensional space-filling curves. **GeoInformatica** v. 7, n. 3, p. 179–209, 2003.

Musser, D. R.; Stepanov, A. A. Generic programming. In: International Symposium on Symbolic and Algebraic Computation: ISAAC'88, 1988, Rome, Italy. **Proceedings...** Berlin: Springer-Verlag, p. 13–25.

Noble, J. Iterators and encapsulation. In: Technology of Object-Oriented Languages and Systems (TOOLS'33), 2000, Mont Saint-Michel, França. **Proceedings...** Washington: IEEE Computer Society, p. 431–442.

OPEN GIS CONSORTIUM. **The OpenGIS specification model, topic 6**: the coverage type and its subtypes. 1999. 67 p. Disponível em: http://www.opengeospatial.org/specs/. Acesso em: 06 jan. 2006.

___. **OpenGIS implementation specification**: grid coverage revision 1.0. 2001. 71 p. Disponível em: http://www.opengeospatial.org/specs/. Acesso em: 06 jan. 2006.

___. **OpenGIS® implementation specification for geographic information** – Simple feature access – Part 2: SQL option. 2005. 51 p. (OGC 05-126). Disponível em: http://www.opengeospatial.org/specs/. Acesso em: 06 jan. 2006.

Parker, J. A.; Kenyon, R. V.; Troxel, D. E. Comparison of interpolating methods for image resampling. **IEEE Transactions on Medicam Imaging**, v. MI-2, n. 1, p. 31–39, 1983.

Patel, J.; DeWitt, D. Partition based spatial-merge join. In: ACM International Conference on Management of Data (SIGMOD), 1996, Montreal, Canada. **Proceedings...** New York: ACM Press, p. 259–270.

Queiroz, G. R. **Algoritmos geométricos para banco de dados geográficos**: da teoria à prática na TerraLib. 2003. 145 p. INPE-12150-TDI/971. Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2003.

Queiroz, G. R.; Ferreira, K. R. SGBD com extensões espaciais. In: Casanova, M. A.; Câmara, G.; Davis Jr., C. A.; Vinhas, L.; Queiroz, G. R. (Eds.). **Bancos de dados geográficos**. Curitiba, PR: MundoGEO, 2005. Cap. 8, p. 267–303.

Ravada, S.; Sharma, J. Oracle8i spatial: experiences with extensible databases. **Lecture Notes in Computer Science**, v. 1651, p. 355–359, 1999.

Ritter, N.; Ruth, M. The GeoTiff data interchange standard for raster geographic images. **International Journal of Remote Sensing**, v. 18, n. 7, p. 1637–1647, 1997.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. **Object-oriented modeling and design**. Englewood Cliffs, NJ: Prentice-Hall, 1991. 500 p.

Sagan, H. **Space-filling curves**. Berlin: Springer-Verlag, 1994. 193 p.

Santilli, S.; Hodgson, C.; Ramsey, P.; Lounsboury, J.; Blasby, D. **PostGIS manual** 2005. Disponível em: http://postgis.refractions.net/documentation. Acesso em: nov., 2005.

Shekkar, S.; Chawla, S. **Spatial databases** - a tour. Upper Saddle River, NJ, USA: Prentice-Hall, 2003. 262 p.

Snyder, J. P. **Map projections** - a working manual. Washington, DC, USA: United States Government Printing Office, 1987.

Stevens, S. S. On the theory of scales of measurement. **Science**, v. 103, p. 677-680, 1946.

Stroustrup, B. **The C++ programming language**, 3rd edition. Reading, MA: Addison-Wesley Publishing Company, 1997. 911 p.

TerraLib. **TerraLib source code cocumentation** - v 3.1.2 São José dos Campos, 2006. Disponível em: http://www.dpi.inpe.br/terralib/html/rv/index.html. Acesso em: 27 fev. 2006.

Usevitch, B. E. A Tutorial on modern lossy wavelet image compression: foundations of JPEG 2000. **IEEE Signal Processing Magazine**, v. 18, p. 22–35, 2001.

Vinhas, L.; Ferreira, K. R. Descrição da TerraLib. In: Casanova, M. A.; Câmara, G.; Davis Jr., C. A.; Vinhas, L.; Queiroz, G. R. (Eds.). **Bancos de dados geográficos**. Curitiba, PR: MundoGeo, 2005. Cap. 12, p. 397–439.

Widenius, M.; AB MySQL; Axmark, D. **MySQL reference manual** O'Reilly, 2002. 814 p. ISBN 0596002653.

Winter, S.; Nittel, S. Formal information modelling for standardisation in the spatial domain. **International Journal of Geographical Information Science**, v. 17, n. 8, p. 721–741, 2003.

Worboys, M.; Duckham, M. **GIS a computing perspective** Second edition. Boca Raton, Florida, USA: CRC Press, 2004. 426 p.

Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. **IEEE Transactions on Information Theory**, v. 23, n. 3, p. 337–343, maio, 1977.