VISUALIZAÇÃO EFICIENTE DE OBJETOS GRÁFICOS

PAULA FREDERICK

DISSERTAÇÃO DE MESTRADO

TeCGraf
DEPARTAMENTO DE INFORMÁTICA
PUC-Rio

Rio de Janeiro, 01 de Julho de 1999

Paula Frederick

Visualização Eficiente de Objetos Gráficos

Dissertação apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para a obtenção do título de Mestre em Ciências em Informática.

Orientador: Marcelo Gattass

TeCGraf/PUC-Rio
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, 01 de Julho de 1999.

Agradecimentos

Ao orientador e professor Marcelo Gattass pelo apoio e por todo o conhecimento transmitido.

A minha família pelo apoio, compreensão e paciência.

A Maurício Mediano, Newton Sanches e Carla Ferreira pelas sugestões e implementações que foram indispensáveis ao desenvolvimento do trabalho.

A todos os colegas do TeCGraf que de algum modo contribuíram para a realização deste trabalho.

A CAPES pelo auxílio financeiro.

Este trabalho foi desenvolvido no TeCGraf/PUC-Rio, Grupo de Tecnologia em Computação Gráfica da PUC-Rio. O TeCGraf é suportado financeiramente através de projetos principalmente com a PETROBRAS/CENPES.

Resumo

Este trabalho apresenta um estudo sobre a visualização eficiente de figuras grandes, tais como mapas e desenhos CAD. Essas figuras são normalmente formadas por um grande número de objetos gráficos que, em geral, são predominantemente poligonais com muitos pontos. Em computação gráfica, as técnicas convencionais de se desenhar figuras não são capazes de oferecer tempos de resposta interativos quando elas são aplicadas a um grande volume de dados. Portanto, é necessário utilizar mecanismos apropriados para acelerar esse processo.

É feita uma rápida descrição dos objetos gráficos e dos métodos de acesso espaciais. Em seguida, são propostos métodos para armazenar e recuperar eficientemente conjuntos de objetos gráficos bidimensionais em um meio de armazenamento secundário. Esses métodos utilizam estruturas de dados persistentes compostas por uma R-tree, V-trees e Quadtrees. A fim de demonstrar a eficiência das soluções propostas, são mostrados experimentos feitos com dados geográficos reais.

Abstract

This work presents a study on the efficient visualization of large figures, such as maps and CAD drawings. These large figures are usually formed by a great number of graphical objects, generally most of them being polygonal lines with many points. In Computer Graphics, conventional techniques for drawing graphical objects are unable to offer interaction times when applied to a huge data volume. Thus, it is necessary to use appropriate mechanisms for speeding up this process.

A brief description of graphical objects and of spatial access methods is made. Then some methods are proposed for storing and efficiently retrieving sets of two-dimensional graphical objects in a secondary storage media. These methods consist in using persistent data structures composed by an R-tree, V-trees, and Quadtrees. In order to demonstrate the efficiency of the proposed solutions, some experiments done with real geographic data are shown.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Gráficos

Lista de Siglas

API	*Application Program Interface*
CAD	*Computer Aided Design*
CD	*Canvas Draw*
CGM	*Computer Graphics Metafile*
Drops	*Drawing Objects on Persistent Systems*
GDI	*Graphics Device Interface*
IUP	Sistema Portátil de Interface com o Usuário
MBR	*Minimum Boundig Rectangle*
OPS	*Object Persistent System*
SAM	*Spatial Access Methods*
SIG	Sistema de Informação Geográfico
TeCGraf	Grupo de Tecnologia em Computação Gráfica

Capítulo 1

Introdução

A disponibilização para o grande público de atlas geográficos digitais e de estudos ambientais de áreas importantes e/ou de risco tem motivado, ao longo dos últimos anos, uma crescente divulgação de mapas através de CD-ROMs.

Mapas são representações gráficas (figuras) de atributos espaciais de objetos geográficos. Nos sistemas atuais em CD-ROMs, essas figuras são geralmente armazenadas no modo matricial de baixa resolução e não escalável. Essas limitações, entretanto, são muito restritivas para a maioria das aplicações técnico-científicas. A solução apresentada aqui é o uso de mapas no formato vetorial.

Uma figura no formato vetorial pode, em geral, ser composta de linhas, regiões preenchidas, marcas, textos e/ou imagens. Esses elementos podem estar agrupados em objetos que formam grupos semânticos e a figura pode ainda ser composta de diversas camadas (*layers*) sobrepostas.

Os elementos predominantes em mapas vetoriais são poligonais com até milhares de pontos, representando linhas ou regiões. Como geralmente os mapas também possuem muitos elementos, o seu armazenamento resulta em arquivos muito grandes. A visualização interativa dessas figuras na tela de um computador tem como requisito a utilização de métodos de armazenamento de dados capazes de oferecer respostas rápidas na interação com o usuário.

1.1 Requisitos

As interfaces com o usuário de sistemas de navegação sobre mapas e figuras grandes possuem quatro tarefas básicas de interação: *zoom, pan, fit* e seleção. No estudo dos requisitos dessas tarefas com relação ao sistema de armazenamento dos dados, três pontos importantes se destacam:

- *clipping* ou *region query* - no processo de *zoom*, somente parte dos dados é visualizada, logo o tempo de visualização pode ser reduzido recuperando-se somente os dados que estiverem dentro da área de visualização;

- *nível de detalhe* ou *generalização* - em escalas de visualização muito grandes, a resolução do dispositivo gráfico é menor que a resolução dos dados. Conseqüentemente, muitos pontos dos dados originais são mapeados para a mesma posição no dispositivo gráfico, resultando em um grande desperdício de processamento. Portanto, a simplificação dos objetos gráficos de acordo com a resolução do dispositivo de saída gera ganhos de desempenho importantes;

- *memória* - devido ao seu tamanho, não é possível carregar todo o dado em memória principal. É preciso utilizar uma memória secundária, que possui tempos de acesso muito maiores que os da memória principal. Assim, é importante reduzir ao mínimo possível a comunicação de I/O.

Esses requisitos não são atendidos pelos métodos convencionais de armazenamento de figuras da computação gráfica, como o CGM [ISO92], mas sim pelos métodos de indexação de dados baseados em árvores, como a R-tree [Gut84], que são comuns nos Sistemas de Informações Geográficas, SIGs.

1.2 Objetivos

Este trabalho estuda métodos para o armazenamento de grandes volumes de objetos gráficos bidimensionais, de forma que parte desses dados possa ser posteriormente recuperada eficientemente para a sua visualização interativa. São empregados métodos

de acesso espaciais e um subsistema gerenciador de acesso à memória secundária, visando atender aos três requisitos citados na seção anterior.

Para testar as estratégias propostas desenvolvemos o sistema Drops (*Drawing Objects on Persistent Systems*), que tem a finalidade de armazenar e recuperar figuras 2D em arquivos. Ele visa principalmente dar suporte ao desenvolvimento de programas de navegação sobre esses dados. Seu desempenho é comparado com o de um interpretador CGM convencional através de testes realizados sobre conjuntos de dados geográficos reais.

Ainda que esses métodos tenham sido propostos com o objetivo inicial de visualizar mapas grandes, eles podem ser aplicados diretamente a qualquer tipo de figura bidimensional, como desenhos CAD, por exemplo.

1.3 Organização da Dissertação

Este trabalho está organizado em 6 capítulos. O Capítulo 2 define os objetos gráficos, especialmente aqueles que fazem parte da composição dos dados geográficos. No Capítulo 3 são descritos os métodos de acesso espaciais utilizados pelo Drops. No Capítulo 4 é apresentado o sistema proposto, o Drops. No Capítulo 5 estão os experimentos realizados para testar o desempenho do sistema. Nele são apresentadas as bases de dados utilizadas, são descritos os procedimentos dos testes e são mostrados os resultados obtidos. Finalmente, no Capítulo 6 são elaboradas as conclusões baseadas nos resultados dos testes e são feitas sugestões para trabalhos futuros.

Capítulo 2

Objetos Gráficos

Para estudar o processo de armazenamento de dados espaciais é importante conhecer bem os dados envolvidos nesse processo. Isto inclui desde a composição dos conjuntos de dados reais até suas representações dentro do ambiente computacional. Por esse motivo são apresentados neste capítulo alguns conceitos básicos de objetos gráficos. Na seção 2.1 são definidas as diversas etapas do processo de modelagem de dados. Na seção 2.2 são revistas as características das bases de dados utilizadas por Sistemas de Informações Geográficos (SIGs). A seção 2.3 descreve como estes objetos gráficos são representados. Por fim, a seção 2.4 mostra como o formato CGM armazena os objetos gráficos.

2.1 Modelagem de Dados

Segundo Workboys [Wor95], modelagem é o processo pelo qual parte de um domínio (domínio fonte) é mapeado para outro domínio (domínio destino). Esse processo é muito importante em praticamente todas as áreas da computação gráfica, onde ele tem a finalidade de traduzir dados do mundo real para o domínio computacional. O "paradigma dos quatro universos" [GV97] aborda isso dividindo o processo em etapas:

- *universo físico* - é constituído dos objetos do mundo real;
- *universo matemático (conceitual)* - contém descrições abstratas dos objetos do mundo físico;

- *universo da representação* - é constituído por representações discretas dos objetos do universo matemático;
- *universo da implementação* - é constituído pelas estruturas de dados que representam os objetos no meio físico.

A Figura 2.1 ilustra os níveis de abstração deste processo.

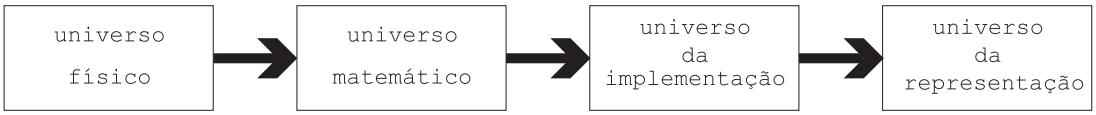


Figura 2.1: Paradigma dos quatro universos

2.2 Sistemas de Informações Geográficas

Os SIGs trabalham com uma ampla variedade de dados geográficos, entre eles estão dados sobre cartografia, cadastros urbanos, redes concessionárias (água, energia e telefone), florestas e muitos outros. Essas informações são constituídas por um conjunto de objetos formados por uma parte gráfica ou geométrica e uma parte descritiva. A primeira parte inclui a sua localização geográfica, sua forma geométrica e mais alguns atributos gráficos (cor, por exemplo), definindo assim o que será apresentado graficamente no dispositivo gráfico. Já a parte descritiva define outras informações, que são semelhantes às que aparecem em qualquer banco de dados convencional, como nome, densidade populacional e área.

As partes descritivas dos objetos geográficos podem ser armazenadas de duas formas: junto com a sua descrição geométrica ou separadas em um banco de dados convencional, sendo que cada uma das partes contém uma referência para a outra. A última estratégia é a mais freqüentemente adotada, pois permite que os dois tipos de dados sejam indexados de formas distintas, otimizando as consultas à base de dados. Quando se considera somente a parte gráfica dos objetos eles também podem ser chamados de objetos gráficos.

Os dados geográficos podem ser provenientes de uma grande diversidade de fontes (ex.: arquivos digitais, sensores, mapas analógicos, etc.) e apresentar uma grande

variedade de formatos. Porém, sob o ponto de vista conceitual, eles podem ser classificados em dois tipos [Câm95, Wor95]:

- *Objetos* - constituem objetos individualizáveis que costumam ter identificação com elementos do mundo real. Também denominados *geo-objetos*, eles têm associados a si um conjunto de atributos descritivos único o objeto como um todo. Sua representação geométrica define a porção do espaço ocupado pelo objeto. Como exemplos podemos citar: áreas indígenas, mapas de vegetação, demarcações de propriedades e unidades federais.

- *Campos* - constituem grandezas distribuídas ao longo do espaço, ou seja, cada posição no espaço está associada a um valor do conjunto de atributos. Os objetos deste modelo são também denominados *geo-campos*. Neste modelo o espaço é normalmente representado por uma grade retangular, porém ele também pode ser representado por outras formas, como, por exemplo, uma triangulação do espaço. Os dados deste modelo são normalmente obtidos através de imagens de satélite, como é o caso de: mapas de classificação dos solos e topografia.

A maioria dos tipos de dados do mundo real pode ser classificada tanto de acordo com o modelo de *objetos* quanto com o modelo de *campos*. Em geral, o formato depende do modo através do qual eles foram obtidos. Entretanto, eles também podem ser convertidos de um formato para outro, pois cada modelo é mais apropriado para determinadas operações realizadas sobre os dados.

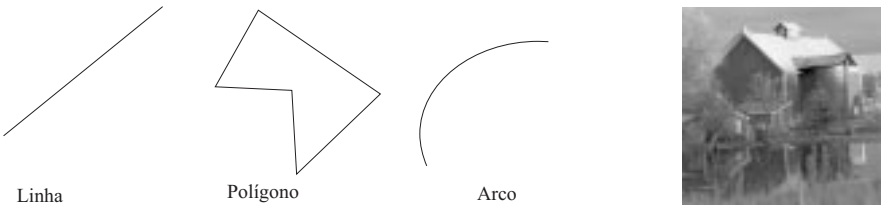
As diferenças entre os dois modelos conceituais implicam também em grandes diferenças nos níveis de abstração restantes. Como conseqüência, grande parte das aplicações de SIG tratam somente um dos dois modelos conceituais.

2.3 Primitivas Gráficas

Os objetos gráficos bidimensionais são representados por um conjunto de primitivas gráficas que são entendidas por qualquer biblioteca gráfica 2D, como, por exemplo, a GDI do Windows ou do XLib. Essas primitivas são divididas em dois grupos distintos:

- primitivas vetoriais - são basicamente marcas (pontos), linhas, linhas poligonais, polígonos (regiões), retângulos, arcos, setores e texto (veja a Figura 2.2(a)). A maioria delas é definida através de um conjunto de pontos. Em SIGs os *geo-objetos* são representados por primitivas vetoriais, mais freqüentemente por pontos, linhas, linhas poligonais e polígonos;

- primitivas matriciais ou *raster* - são os vários formatos de imagens (RGB, RGBA e indexada) - veja a Figura 2.2(b). O objeto é representado por grades de células. Cada célula é referenciada por uma linha e uma coluna, e o seu valor equivale ao atributo do objeto na posição correspondente. Em SIGs os dados matriciais são utilizados para representar os *geo-campos*.



(a) Primitivas vetoriais.

(b) Primitiva matricial.

Figura 2.2: Exemplos de primitiva

2.3.1 Atributos Gráficos

A aparência de todas as primitivas gráficas é determinada por um conjunto de atributos gráficos, sendo que para cada caraterística visual das primitivas existe um tipo de atributo gráfico. Algumas dessas características são: cor, espessura de linha, estilo de linha e fonte, dentre outras. É importante observar que os tipos de atributos utilizados para descrever a aparência de uma primitiva variam dependendo do seu tipo. Mais adiante, na Tabela 4.1 esse fato pode ser visto melhor, pois são mostrados quais os atributos de cada primitiva gráfica do sistema proposto, Drops.

Como pode ser observado na Figura 2.3, os valores dos atributos não variam tanto quanto as descrições geométricas. Portanto o sistema gráfico tem duas opções distintas

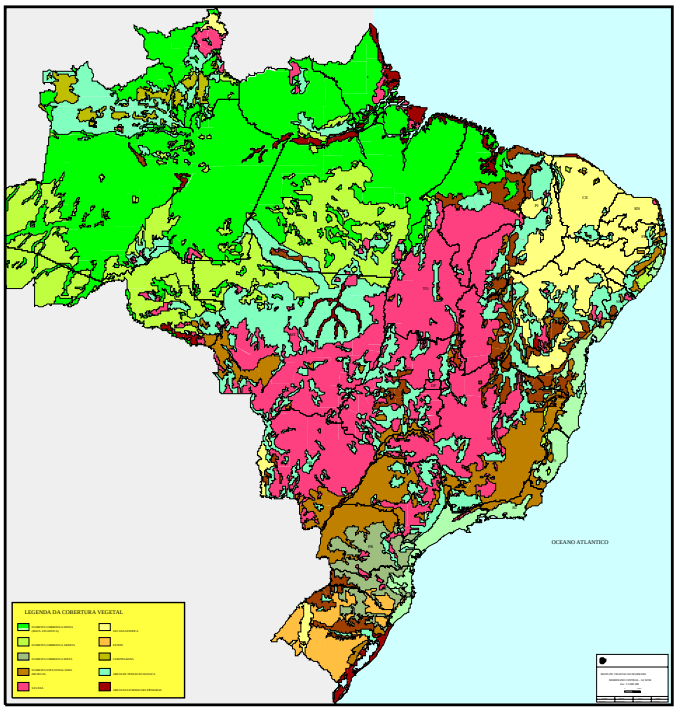


Figura 2.3: Vegetação do Brasil

para tratar esses atributos: através de (1) valor corrente ou (2) índice para tabelas.

No primeiro caso, o sistema guarda internamente um valor para cada tipo de atributo, que são os "atributos correntes". A definição das primitivas é feita somente pela sua descrição geométrica, enquanto que a descrição de sua aparência é obtida a partir dos valores dos atributos correntes. Para modificar a aparência da próxima primitiva a ser definida é necessário modificar os valores dos atributos correntes.

No segundo caso cada primitiva tem armazenado junto com ela um índice para uma tabela que contém os valores relativos ao seu conjunto de atributos. Para economizar espaço de armazenamento essas tabelas podem ser construídas de forma a não terem dois registros de atributos iguais.

O conjunto de atributos possui sua própria tabela resultando em tabelas de: atributos de marca, atributos de linha, atributos de preenchimento e atributos de texto. A seção 4.4 define estas tabelas.

A definição por valor corrente é mais simples de se implementar e resulta em uma descrição menor da figura, uma vez que os atributos não precisam ser definidos individualmente para cada objeto. Porém esse tratamento é depende da ordem de definição

das primitivas, o que constitui um problema quando os objetos são indexados espacialmente, como será discutido mais adiante. Por sua vez, na definição por tabelas, cada objeto tem a referência para os seus atributos, sem que haja a necessidade de armazenar repetidamente conjuntos iguais de atributos.

2.3.2 Camadas

Em sistemas de informações geográficas, é muito comun encontrarmos bases de dados compostas por vários contextos, cujos domínios espaciais se sobrepõem. Como exemplos podemos citar dados que contêm mapas da vegetação, da temperatura e da altitude de uma região ou uma base de dados com as edificações, os lotes e as quadras de uma cidade. Veja o exemplo da Figura 2.4.

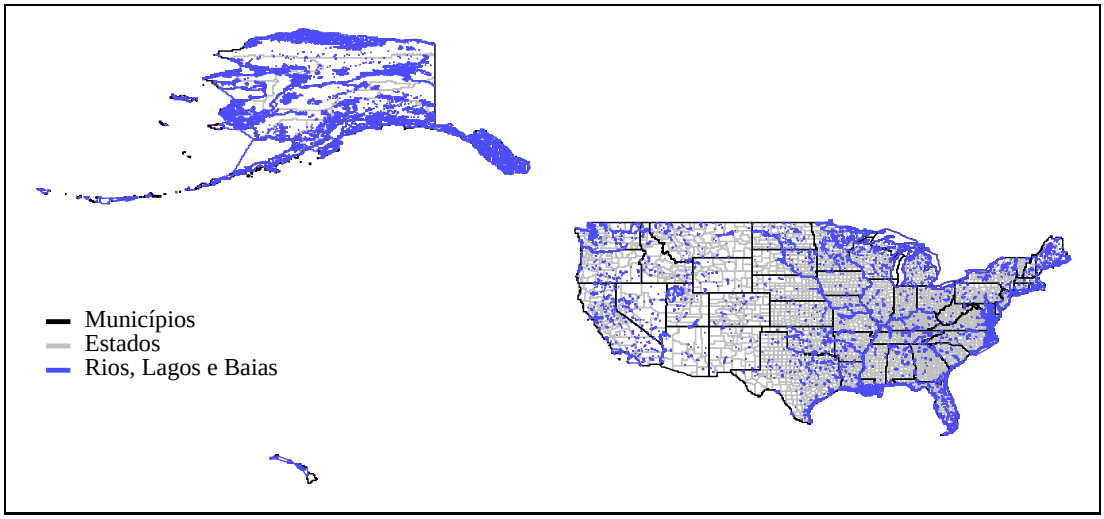


Figura 2.4: Estados Unidos [USG]. Esta base da dados contém três contextos distintos.

Para representar todos esses contextos dentro de uma mesma base de dados é utilizado o conceito de camadas (*layers*), sendo que cada camada corresponde a um contexto diferente. Desta forma, cada camada é formada por um subconjunto de objetos gráficos. Além disso, elas têm um nome de identificação para ser utilizado pelas aplicações e possuem dois estados: ativada ou desativada. Quando ativa, a camada está visível e responde às operações de seleção (*pick*).

Portanto, conjuntos de objetos gráficos que são formados por figuras sobrepostas, podem ser organizados em diversas camadas. Elas estão ordenadas entre si, indicando

a ordem em que serão enviadas ao dispositivo gráfico. A primeira camada é desenhada primeiro, ficando portanto no fundo do desenho.

2.3.3 Formas (*Shapes*)

Formas *shapes* são coleções de primitivas agrupadas geometricamente, de modo que elas possam ser utilizadas diversas vezes dentro da figura. Ao serem definidas, as Formas recebem um identificador (número inteiro). Este deverá ser fornecido em sua instanciação juntamente com uma transformação (translação, rotação e escala), que será aplicada às suas primitivas.

As Formas são utilizadas para representar formas geométricas que aparecem mais de uma vez na base de dados. Deste modo não é preciso redefinir a mesma forma várias vezes, evitando redundância e reduzindo assim o tamanho dos dados.

2.4 Computer Graphics Metafile (CGM)

O CGM (*Computer Graphics Metafile*) [ISO92, Hen93] é um padrão internacional para o armazenamento e a transferência de informações gráficas 2D. Ele aceita dados tanto na representação vetorial quanto na representação matricial. As primitivas que ele suporta são: linhas, polígonos, textos, marcas, símbolos e elementos matriciais. As linhas podem ser linhas disjuntas, uma seqüência de linhas, um arco circular, arco cônico, uma b-spline ou então uma ou mais curvas Bézier. Atualmente a maioria das aplicações gráficas é capaz de gerar e interpretar arquivos no formato CGM.

No contexto do CGM, um meta-arquivo consiste em uma coleção de comandos que descrevem os elementos da figura. Os elementos podem ser descrições geométricas dos objetos gráficos, descrições da aparência dos objetos ou informações necessárias à interpretação do arquivo. Além de dados gráficos, o CGM permite também o armazenamento de outras informações adicionais que são dependentes da aplicação, que não têm nenhum sentido gráfico.

Um arquivo CGM é estruturado em uma série de níveis de abstração (veja a Figura 2.5), e cada elemento tem especificado em que posições do arquivo ele pode se encontrar. O nível meta-arquivo contém uma ou mais figuras independentes que podem

ser acessadas aleatoriamente, enquanto que no nível da figura estão todos os elementos necessários para definir a figura. Os elementos são o menor nível de abstração do meta-arquivo, podendo ter diversas funcionalidades, conforme mencionado anteriormente.

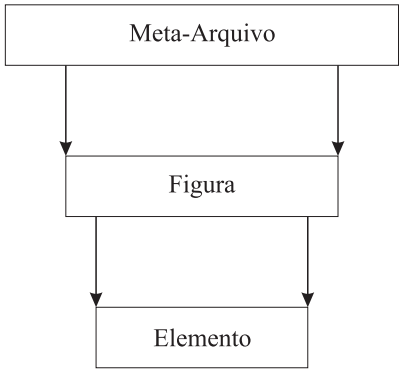


Figura 2.5: Estrutura de um meta-arquivo

O formato CGM possui três formas distintas de codificação: a por caracter, a binária e a textual. Cada formato procura satisfazer determinados requisitos. Na codificação por caracter a intenção é garantir uma codificação compacta e a facilidade de transporte através da rede. Na codificação binária a ênfase é a facilidade de geração e interpretação dos arquivos CGM. Já a codificação textual é legível, visando permitir a edição do arquivo quando não existe nenhum ambiente gráfico apropriado para esta tarefa.

Os objetos gráficos podem ser representados geometricamente por números inteiros (8, 16, 24 ou 32 *bits*), reais de ponto fixo (32 ou 64 *bits*) ou reais de ponto flutuante (32 ou 64 *bits*). Quanto às cores, elas podem ser especificadas no formato indexado ou no formato direto, pela definição de todas as suas componentes. No formato direto a cor pode ser definida através do modelo RGB, do modelo CMYK ou do modelo CIE (XYZ).

É importante mencionar também que no CGM é utilizado o conceito de atributo corrente na associação de atributos às primitivas, e também não existe maneira de se acessar as primitivas aleatoriamente. Portanto, mesmo que a interação envolva somente parte dos dados, é necessário fazer a leitura de todo o arquivo.

Capítulo 3

Métodos de Acesso Espaciais

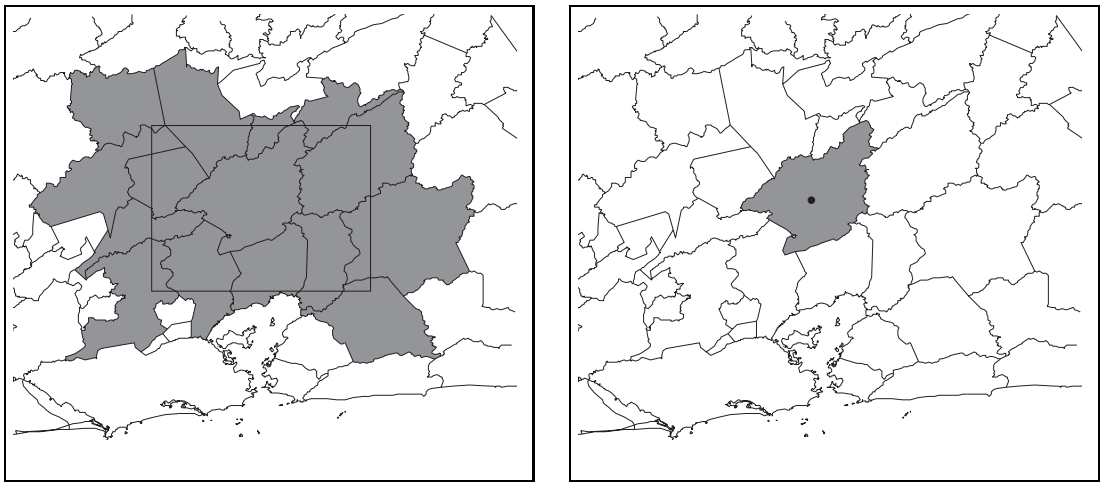
Para manipular eficientemente dados espaciais com um grande nível de complexidade e em grande volume são necessários mecanismos que indexem esses dados, permitindo assim a recuperação de somente uma parte dos dados de acordo com suas localizações e extensões espaciais.

No caso de bancos de dados tradicionais, em que a chave de busca é unidimensional, já existe há muito tempo uma série de métodos de acesso eficientes e bem compreendidos. Entre eles os mais populares são a B-tree [BM72, Com79] e o *hashing* [FNPS79]. Porém, o desenvolvimento de estruturas de indexação para dados multidimensionais é muito mais complicado, pois não existe uma ordem natural entre estes objetos que também preserve as suas proximidades espaciais.

Dentre as diversas operações que são realizadas sobre dados espaciais, as mais comuns são:

- *consulta de intervalo* (*range query* ou *window query*) - dado um retângulo de arestas paralelas aos eixos do sistema de coordenadas, ela obtém todos os objetos que o interceptam (veja a Figura 3.1(a));

- *consulta de ponto* (*point query*) - dado um ponto, ela obtém todos os objetos que o contêm (veja a Figura 3.1(b)). Esta operação é um caso particular da *consulta de intervalo* em que o retângulo tem tamanho zero.



(a) consulta de intervalo (b) consulta de ponto

Figura 3.1: Operações de busca

Outra forma de escrever essas consultas é dada por [San99]:

$$\sigma_{o_{ref},\theta}\, S = \{o \in S \mid o_{ref}\, \theta\, o\} \tag{3.1}$$

onde S é o conjunto de dados de entrada, o_{ref} é o objeto de consulta e θ é a relação que deve ser satisfeita entre o_{ref} e os objetos de saída. Em uma consulta de intervalo o_{ref} é o retângulo de busca e a condição a ser satisfeita é $o_{ref} \cap o \neq \emptyset$. Em uma consulta de ponto o_ref é o ponto de busca e a relação a ser mantida é $o_{ref} \subset o$.

No processo de visualização desses dados, a *range query* é utilizada com a finalidade de fazer o *clipping* de objetos. Nesse caso o retângulo de busca da operação é a área de visualização. A *point query* por sua vez, é equivalente à operação de seleção (*pick*) de objetos.

Atualmente existe uma grande variedade de métodos de acesso espaciais (veja [Sam90] para saber o que foi proposto até 1989 ou [GG98] para obter um levantamento mais recente). Eles são divididos em métodos de acesso a pontos, que armazenam somente pontos, e em métodos de acesso a regiões, que suportam também objetos de tamanho maior do que zero. A maioria desses métodos de acesso leva em conta a utilização de um método de acesso à memória secundária, procurando reduzir o

número de acessos a disco feitos pelo sistema operacional. Conseqüentemente, os objetos são normalmente agrupados em blocos de tamanho igual, denominados páginas, de forma que as transferências de dados de e para o disco sejam feitas em unidades de páginas.

Devido à complexidade que os objetos de extensão diferente de zero apresentam, eles são normalmente aproximados por estruturas mais simples, como o retângulo envolvente (MBR) ou o círculo envolvente, como mostrado na Figura 3.2. Os métodos de acesso indexam somente essas representações simplificadas dos objetos, dividindo assim as operações espaciais em duas etapas (veja a Figura 3.3). Na primeira etapa, também denominada filtragem, os testes são feitos com as aproximações dos objetos. Na segunda etapa são testadas as geometrias dos objetos resultantes da primeira fase.

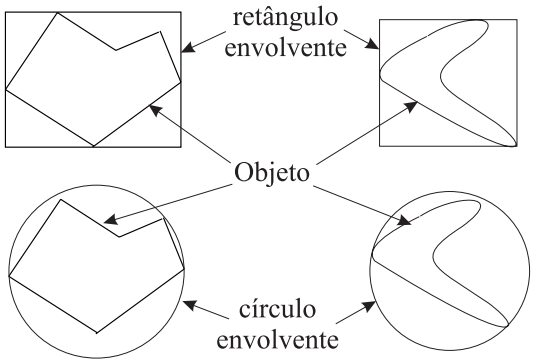


Figura 3.2: Aproximação de objetos

Os métodos de indexação de regiões podem ser classificados da seguinte forma:

1. *transformação* - os objetos são transformados para uma outra representação, e então são indexados por métodos de acesso para pontos. Eles são mapeados para pontos de um espaço de dimensão maior ou para intervalos no espaço uni-dimensional. Os métodos Z-order [Ore86] e Grid-file [NHS84] são alguns exemplos;

2. *sobreposição de regiões* - o espaço de dados é particionado em regiões. É permitido que haja sobreposição de regiões de forma que cada objeto esteja totalmente contido dentro de uma região. Exemplos destes métodos são as R-trees [Gut84] e as R^*-trees [BKSS90];

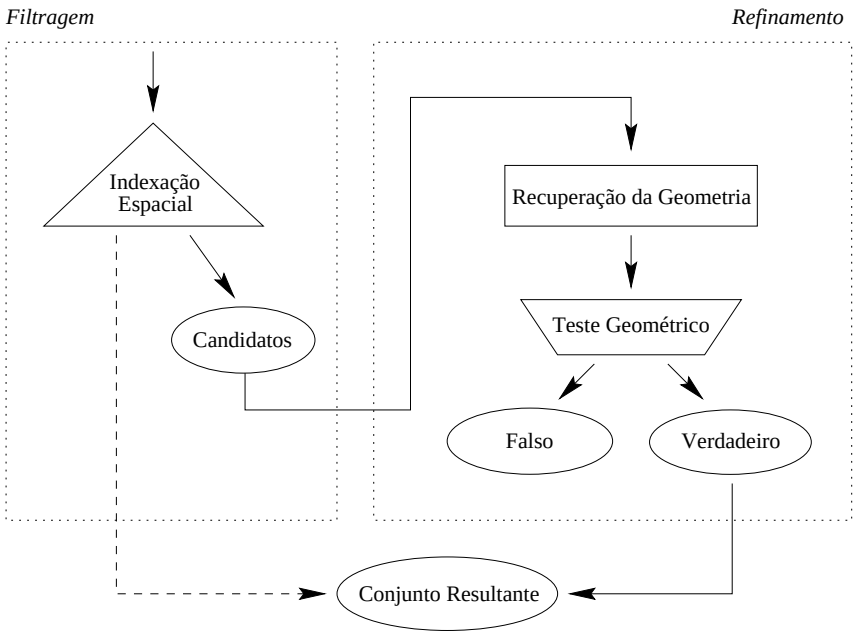


Figura 3.3: As etapas do processo de seleção espacial [BKSS94]

3. *regiões disjuntas* ou *clipping* - o espaço de dados é particionado em regiões disjuntas. Porém pode acontecer que o objeto não esteja totalmente contido dentro de nenhuma região. Neste caso existem duas soluções: a duplicação do objeto ou a decomposição do objeto. Na primeira solução, cada região que intercepta o objeto tem uma referência para ele, enquanto que na segunda solução o objeto é decomposto em pedaços de forma que cada pedaço esteja totalmente dentro de uma região. Um exemplo deste tipo de método é a R^+-tree [SRF87].

Até recentemente, todos os métodos de acesso mencionados tinham somente a função de indexar os objetos espaciais. Ocorre, entretanto, que muitas vezes os próprios objetos são muito grandes e complexos, como no caso de polígonos e linhas poligonais com milhares de vértices ou de imagens com resoluções da ordem de 10000×10000. Nessas situações é importante otimizar também o acesso à geometria de cada objeto, o que pode ser alcançado através da indexação da componente geométrica dos objetos. Além de permitir a manipulação de somente parte dos dados, normalmente os métodos de indexação geométrica também possuem multiresolução geométrica, permitindo assim a obtenção de versões simplificadas dos objetos.

Para linhas poligonais e polígonos foram propostos dois métodos de acesso com

este propósito: a TR-tree [SK91, BKSS94], e a V-tree [MCD94]. A TR-tree indexa a região por meio de um conjunto de trapezóides armazenados em uma R-tree em que os furos das regiões já estão representados. Ela não possui multiresolução geométrica. A V-tree é uma estrutura com multiresolução geométrica que indexa as linhas poligonais que definem as bordas dos objetos espaciais. Por outro lado, para representar imagens, as estruturas mais utilizadas são as Quadtrees [FB74, Sam84, Gar82].

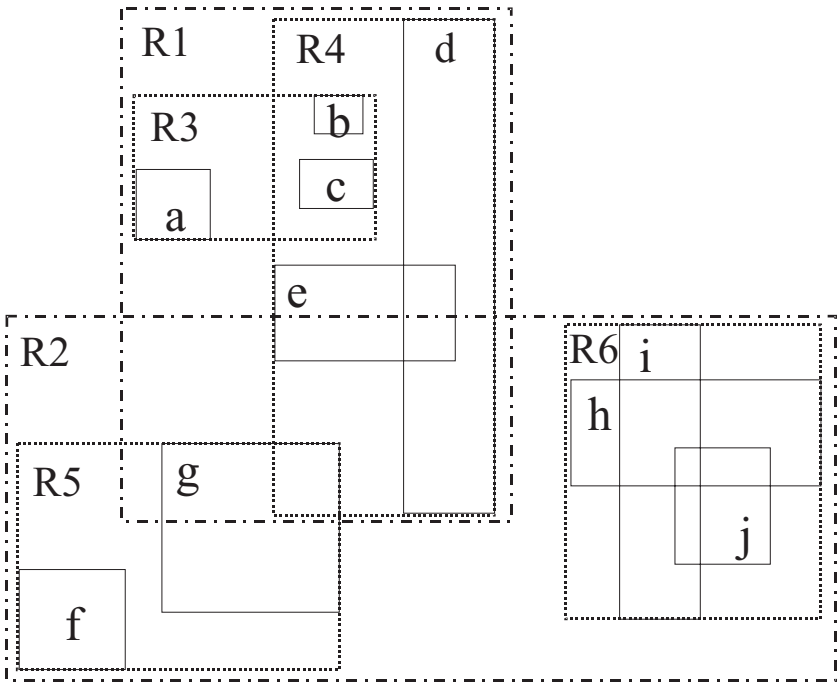
3.1 R-tree

A R-tree [Gut84] é o método de acesso a regiões mais popular existente atualmente. Ela é uma estrutura de dados hierárquica e dinâmica, projetada para suportar eficientemente *range queries*. Ela indexa os objetos geométricos representados por seus retângulos envolventes (MBRs) sem clipá-los nem transformá-los em pontos de dimensões maiores. A maioria dos trabalhos na área de métodos de acesso espaciais (SAM) utiliza a R-tree como modelo para a comparação de desempenho com outras estruturas de indexação. Grande parte da atenção recebida pela R-tree deve-se à sua facilidade de implementação.

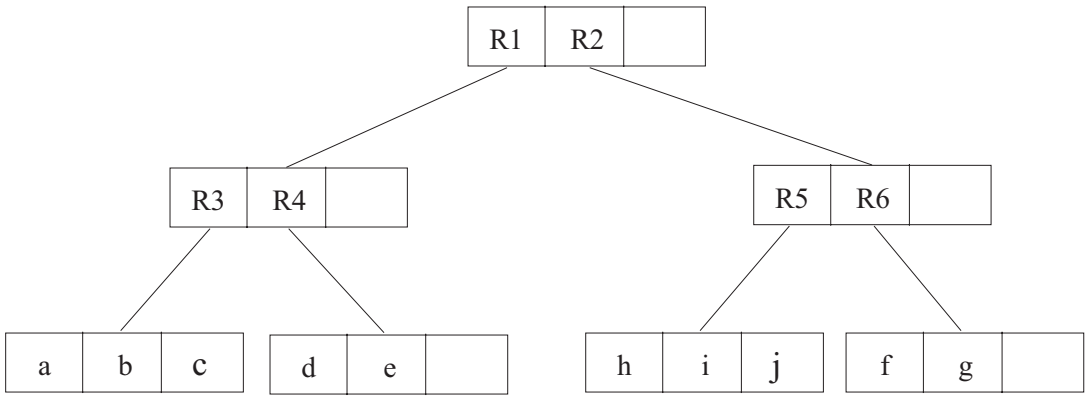
A R-tree é uma extensão da B-tree [Com79] para o espaço multidimensional. Ela representa o espaço de dimensão *n* como uma hierarquia de intervalos *n*-dimensionais (retângulos no caso bidimensional). Assim como a B-tree, ela é uma estrutura balanceada, em que todos os objetos estão armazenados nas folhas da árvore. Toda R-tree possui uma ordem (m, M), indicando que seus nós possuem entre $m \leq \lceil m/2 \rceil$ e M registros.

Todo registro da estrutura é formado pelo par (*r*, *p*), onde *r* é um intervalo (retângulo) e *p* um ponteiro. Nos nós externos (folhas), *r* é o retângulo envolvente do objeto apontado por *p*, e enquanto que, nos nós internos, *r* é o retângulo que envolve todos os retângulos armazenados na sub-árvore apontada por *p*. A Figura 3.4 mostra o exemplo de uma R-tree para um determinado conjunto de dados.

É importante notar que pode haver sobreposição entre os retângulos de um mesmo nível da R-tree. Da mesma forma, um retângulo pode estar espacialmente dentro de diversos nós, porém só pode estar associado a um único nó. Outro ponto que deve



(a) Representação planar



(b) Estrutura

Figura 3.4: Exemplo de uma R-tree

ser observado é que não existe uma R-tree única para cada conjunto de objetos; ela depende da ordem de inserção e de remoção dos elementos e também dos algoritmos utilizados por estas operações.

A operação de busca em uma R-tree é bem trivial. O algoritmo inicia a busca na raiz e em seguida vai descendo recursivamente pelos nós da árvore. Em cada nó visitado verifica-se quais são os retângulos que interceptam o retângulo de busca e, caso o nó seja interno, visita-se as sub-árvores correspondentes aos retângulos interceptados. Caso o nó corrente seja uma folha, os objetos correspondentes são retornados como parte do resultado da operação.

É muito comum que o algoritmo de busca percorra vários caminhos da árvore sem que todos eles realmente levem a objetos que interceptem a área correspondente. Portanto, a redução de tamanho dos retângulos dos nós internos resulta em menos caminhos percorridos durante a operação de busca. Consequentemente o tempo gasto pela operação também será menor. Levando este fato em consideração os algoritmos envolvidos na construção da R-tree, que são a inserção e remoção de elementos, utilizam heurísticas que visam reduzir o número de nós visitados durante a busca.

A operação de inserção em uma R-tree proposta originalmente por Guttman [Gut84] é muito semelhante à de uma B-tree. Porém, este processo envolve duas situações em que a decisão a ser tomada não é tão clara como no caso das B-trees. A heurística de otimização utilizada nesses casos é minimizar a área de cobertura dos nós. A primeira das situações consiste em escolher em qual dos filhos do nó interno corrente o objeto deve ser inserido. A sub-árvore escolhida é sempre aquela cujo retângulo associado necessite a menor expansão de forma a envolver o novo elemento. A segunda situação consiste em decidir como deve ser feita a divisão (*splitting*) do nó em caso de necessidade. Guttman propôs três algoritmos para fazer essa divisão (algoritmos de *splitting*): o expotencial, o quadrático e o linear, em ordem decrescente de qualidade. Seus nomes indicam qual a sua complexidade. Experimentos mostraram que a diferença de desempenho entre eles é muito pequena, não compensando, assim, o uso de um algoritmo muito complexo.

Existe mais uma característica importante na operação de inserção da R-tree. Após a inserção, é necessário atualizar os retângulos de cada uma das sub-árvores que passou

a conter o novo elemento. A atualização deve ser feita de baixo para cima, ou seja, iniciando-se nas folhas da árvore.

Quanto à operação de remoção, ela não será descrita neste trabalho, pois, uma vez que os dados de nosso interesse são estáticos, não existe necessidade de que o sistema proposto suporte remoções de objetos.

Uma série de trabalhos subseqüentes ao de Guttman [Gut84] propõem variantes da R-tree com a finalidade de obter desempenhos de *range queries* melhores do que o da R-tree original. A principal diferença entre a R-tree e suas variantes está nos algoritmos de inserção, sendo que muitas dessas variantes apresentam exatamente a mesma composição que a R-tree original. Entre elas estão a R^+-tree [SRF87], o trabalho de Greene [Gre89], a R^*-tree [BKSS90], a Hilbert R-tree [KF94] e as R-trees compactas (*packed* R-trees) [RL85, KF93, LLE97].

3.2 R-trees Compactas

As R-trees são estruturas de dados dinâmicas, o que significa que seu conteúdo pode sofrer alterações sem que toda a árvore precise ser reconstruída. Por este motivo, ela possui rotinas eficientes de inserção e remoção de objetos. Entretanto, o processo de construção da R-tree, que consiste em inserir um objeto por vez na estrutura, tem a desvantagem de não gerar uma estrutura adequada para o suporte eficiente de *range queries*, resultando assim em árvores que apresentam um sub-aproveitamento do espaço de utilização de seus nós.

Porém, para a construção de uma R-tree a partir de um conjunto de dados previamente definido, pode-se pré-processar os dados de forma a produzir uma árvore mais otimizada. Cientes deste fato, Roussopoulos e Leifker [RL85] propuseram a R-tree compacta para armazenar dados estáticos. As R-trees compactas podem ser vistas como R-trees que utilizam uma técnica de construção diferente, que produz árvores com praticamente 100% de ocupação nos seus nós. A compactação da R-tree resulta em uma estrutura com menos nós, portanto o espaço de armazenamento de seus índices é menor e ela oferece melhor desempenho para operações de busca, pois o número de nós visitados é menor.

Os algoritmos de construção das R-trees compactas são também denominados *algoritmos de compactação* (*packing algorithms*). O processo é feito de baixo para cima, ou seja, primeiro são criadas as folhas e então são criados sucessivamente os nós internos do nível imediatamente superior, até que a raiz seja criada.

O algoritmo cria inicialmente uma lista de registros, sendo que cada registro está associado a um objeto do conjunto de dados, e logo em seguida ordena a lista. Depois ele cria um nó e o preenche com os primeiros elementos da lista, que são então retirados da lista. Esse procedimento é repetido enquanto ainda houver elementos na lista. Ao final desta etapa têm-se as folhas da árvore. Os níveis subseqüentes da árvore são criados da mesma forma, só que desta vez os registros que irão preencher os nós são relativos aos nós criados na etapa anterior.

O critério de ordenação dos elementos depende do algoritmo de compactação. Normalmente a heurística utilizada é a de minimizar a cobertura e a sobreposição dos nós. O critério adotado na proposta original da R-tree compacta [RL85] consiste na ordenação dos retângulos segundo o valor da coordenada x. O trabalho original não especifica de onde tomar esta coordenada, portanto ela pode ser correspondente ao lado esquerdo, ao lado direito ou ao centro dos retângulos. Aqui utilizaremos a coordenada x de seus centros. Daqui por diante chamaremos este método de "ordenação em *X*". A Figura 3.5 ilustra como esse algoritmo funciona.

Atualmente têm mais dois métodos de compactação freqüentemente utilizados, o algoritmo de compactação de Hilbert [KF93] e o STR (*Sort-Tile-Recursive*) [LLE97], que serão descritos a seguir.

3.2.1 Hilbert

A heurística empregada pelo algoritmo de compactação de Hilbert [KF93] na construção da R-tree compacta consiste em utilizar a curva de Hilbert [FR89] para ordenar os retângulos.

Na Figura 3.6 estão ilustradas as curvas de Hilbert para grades de dimensões 2x2, 4x4 e 8x8. A curva de Hilbert visita todos os pontos de uma grade *n*-dimensional uma única vez e sem nunca se cruzar, impondo assim uma ordem linear aos pontos multidimensionais. Essa ordem, também denominada valor de Hilbert, é calculada

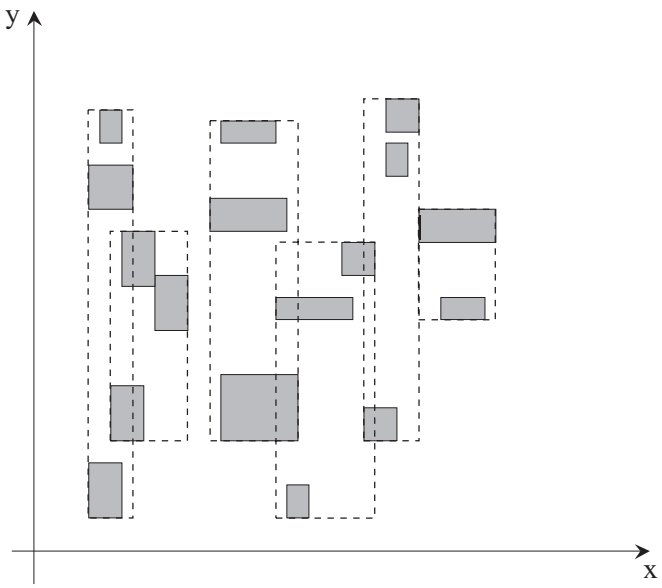
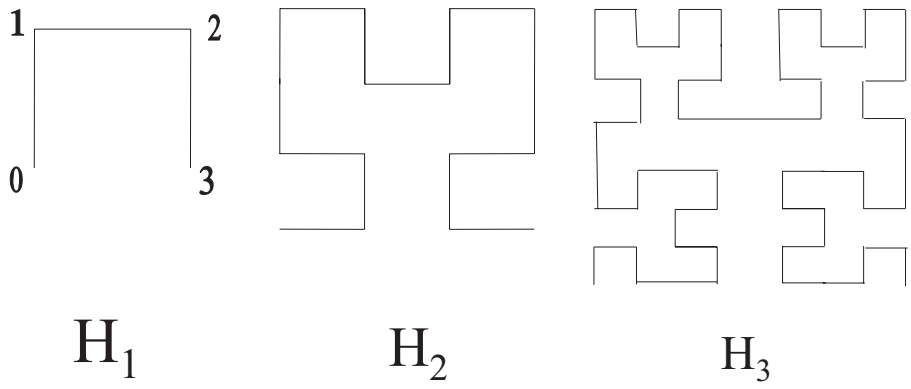


Figura 3.5: Ordenação em X

medindo-se a distância ao longo da curva entre o ponto e a origem da curva. Em [FR89] encontra-se uma descrição mais detalhada da curva de Hilbert e do algoritmo que calcula o valor de Hilbert de um ponto.



(a) Ordem 1 (b) Ordem 2 (c) Ordem 3

Figura 3.6: Curvas de Hilbert

O algoritmo de compactação de Hilbert ordena os retângulos pelo valor de Hilbert dos centros dos retângulos, procurando desta forma proporcionar um agrupamento de dados melhor do que o que é obtido pelo algoritmo de compactação original

(ordenação em X). A Figura 3.7 mostra como os retângulos são agrupados através deste algoritmo.

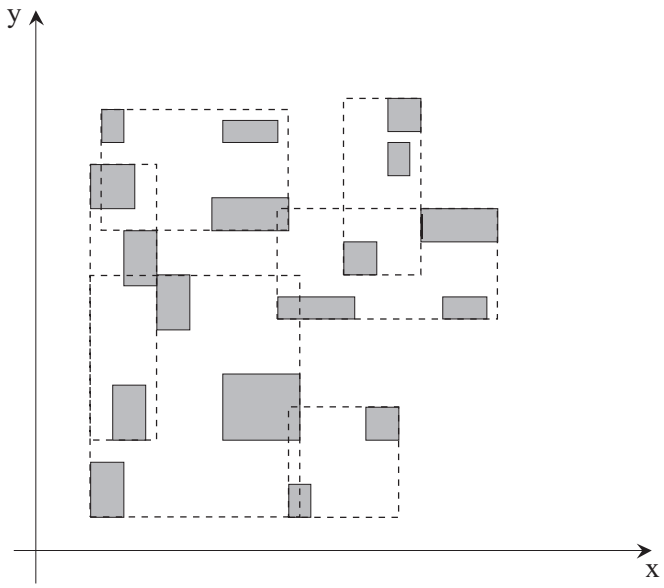


Figura 3.7: Algoritmo de Compactação de Hilbert. Os dados deste exemplo são os mesmos da figura 3.5, mas agora foi utilizado o algoritmo de Hilbert para agrupar os retângulos.

3.2.2 *Sort-Tile-Recursive* (STR)

O algoritmo de compactação STR é o mais recente entre os três mencionados nesta seção. A Figura 3.8 ilustra como o STR agrupa os retângulos. Consideramos que existem r retângulos para serem agrupados em nós que suportam no máximo M elementos. Primeiro ordenam-se os retângulos pela coordenada x de seus centros e então particiona-se os retângulos em $\sqrt{r/M}$ fatias verticais. Cada fatia consiste em uma seqüência de $M * \sqrt{r/M}$ retângulos da lista ordenada, sendo que a última fatia pode ser formada por um número menor de elementos. Depois ordenam-se os retângulos de cada fatia segundo a coordenada y de seus centros e então vai-se agrupando os M elementos consecutivos em nós.

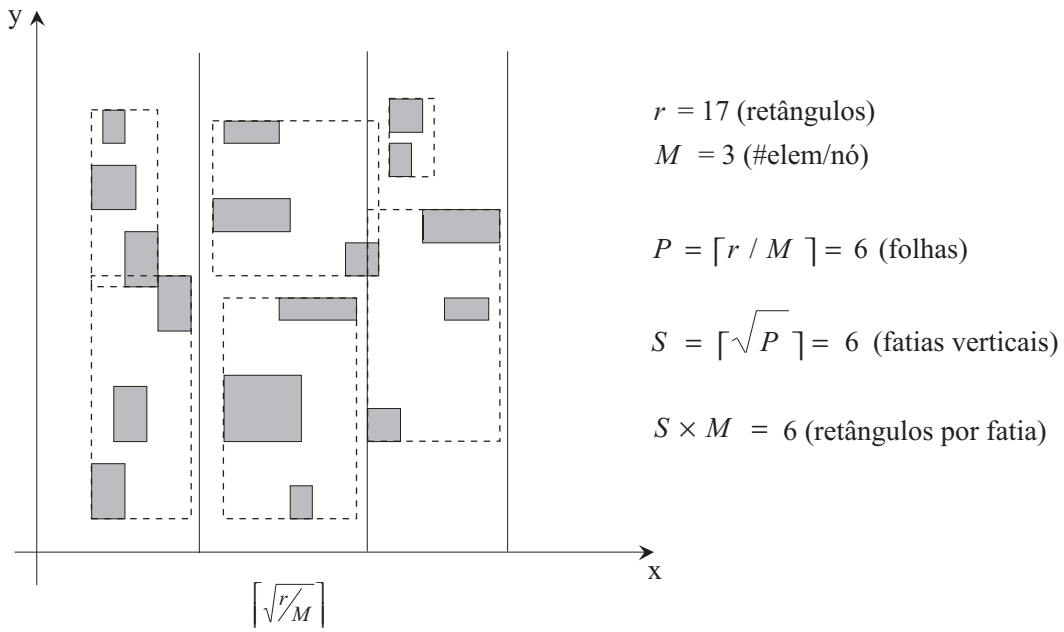


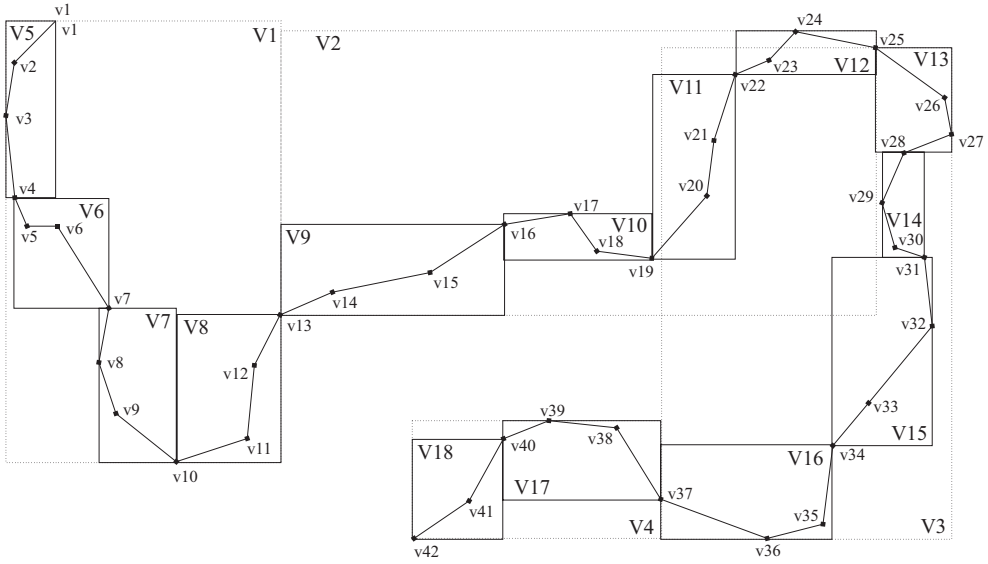
Figura 3.8: *Sort-Title-Recursive*. Estes são novamente os dados da figura 3.5, mas agrupados de acordo com o algoritmo *Sort-Title-Recursive*.

3.3 V-tree

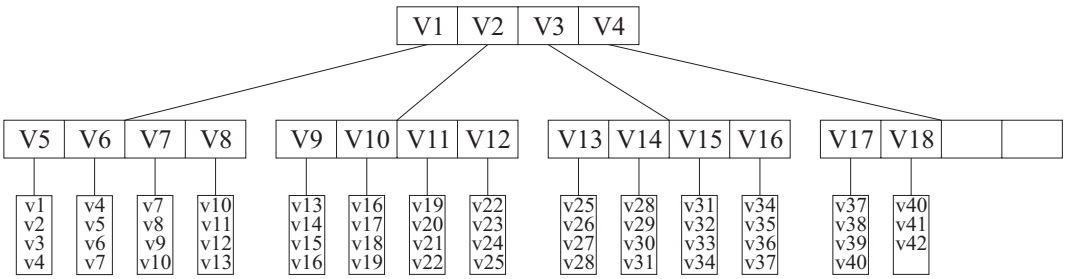
As V-trees [MCD94] são estruturas de dados hierárquicas projetadas para armazenar linhas poligonais longas, que são representadas por uma seqüência de pontos. Sua principal funcionalidade consiste em recuperar eficientemente aproximações de linhas poligonais armazenadas em memória secundária. Além disso, elas otimizam a recuperação de fragmentos da linha poligonal que interceptam uma dada região retangular (*range querie* ou *clipping*) e a operação de ponto em polígono. Esta testa se um dado ponto está dentro de uma linha poligonal fechada que não se cruza.

A estrutura da V-tree é em alguns aspectos semelhante a uma R-tree. Ela é uma árvore balanceada e todos os seus dados estão armazenados nas folhas. Cada folha da V-tree armazena uma seqüência de pontos, que corresponde a um fragmento da linha poligonal, enquanto que seus nós internos são constituídos por registros da forma (*r*, *p*), onde *r* é o retângulo envolvente da sub-árvore apontada por *p*. A ordem (m, M) da V-tree define o número mínimo e máximo de elementos que cada nó pode possuir. A Figura 3.9 mostra o exemplo de uma V-tree.

A construção de uma V-tree a partir de uma linha poligonal já inteiramente de-



(a) Representação planar



(b) Estrutura

Figura 3.9: Exemplo de uma V-tree

finida, ou seja, estática, é bem simples. Primeiro quebra-se a linha em fragmentos de tamanho M armazenando cada um deles em uma folha da árvore. Para que os retângulos envolventes dos fragmentos envolvam toda a linha poligonal, o último ponto de um fragmento deve ser igual ao primeiro ponto do fragmento seguinte. O passo seguinte consiste em agrupar as folhas em nós pegando-se sucessivamente M folhas correspondentes a M fragmentos consecutivos. Com cada grupo forma-se uma árvore, criando-se um novo nó que aponta para todos os elementos do grupo e calculando-se o retângulo envolvente de todos os fragmentos do grupo. Depois agrupam-se as árvores da mesma forma e faz-se de cada grupo uma só árvore. O processo continua até restar

somente uma árvore.

O algoritmo de *range querie* de uma V-tree é semelhante ao das R-trees. Ele percorre a V-tree visitando todas as sub-árvores cujos retângulos envolventes interceptem o retângulo de busca. Chegando em uma folha, ele insere o fragmento correspondente na lista de respostas, sempre considerando que o primeiro ponto do fragmento é o mesmo que o último ponto do fragmento que o antecede.

No processo de recuperação de somente uma aproximação da linha poligonal o procedimento não é muito diferente. O algoritmo não desce na sub-árvore se todas as dimensões do retângulo envolvente forem menores que a precisão dada. Neste caso todos os pontos armazenados na sub-árvore em questão são aproximados pelo centro do retângulo envolvente.

A operação de ponto em polígono é uma variação da apresentada em [PS85]. Traça-se uma linha paralela ao eixo X que parte do ponto e continua para a direita do ponto. Então conta-se quantas vezes ela intercepta o polígono. Se o número de interseções for ímpar, o ponto está no interior do polígono, caso contrário ele está fora. A V-tree otimiza o processo de testar as interseções que o polígono faz com a reta traçada. Se a reta não intercepta o retângulo envolvente da sub-árvore, então não há necessidade de testar os nós da sub-árvore, pois a reta também não intercepta nenhum dos fragmentos armazenados na sub-árvore.

3.4 Quadtree

As Quadtrees [FB74, Gar82, Sam84] são estruturas de dados hierárquicas utilizadas para representar o interior de objetos espaciais bidimensionas. Ela é baseada no princípio de dividir para conquistar. O objeto é decomposto recursivamente em quatro quadrantes de mesmo tamanho até que os quadrantes sejam totalmente homogêneos.

As Quadtrees podem ser utilizadas para representar diversos tipos de dados, entre eles imagens. Aqui descrevemos seu funcionamento somente para o caso de armazenamento de imagens.

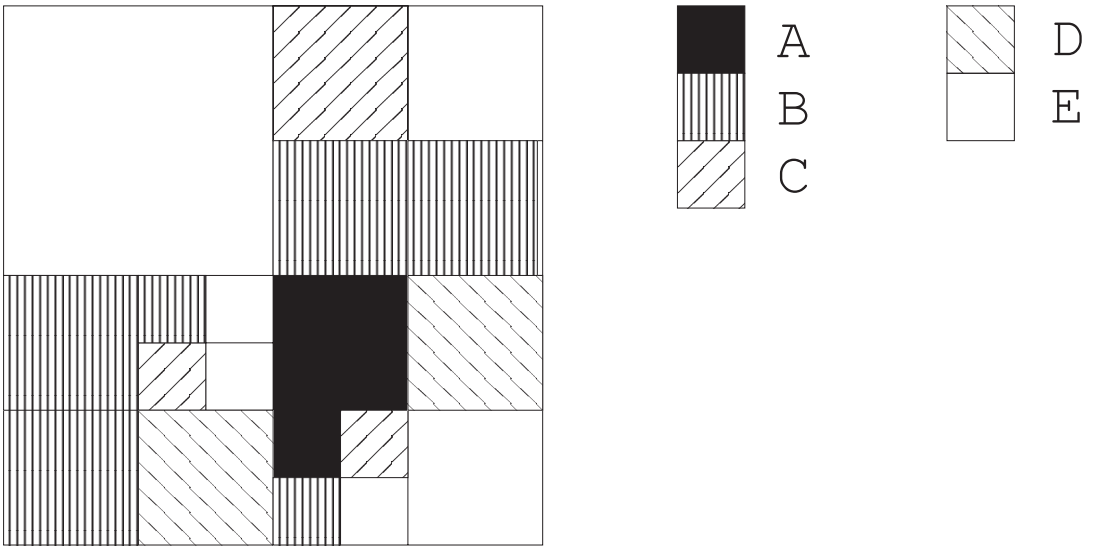
A Figura 3.10 ilustra um exemplo de uma Quadtree para o armazenamento de uma imagem. A representação da Quadtree é geralmente feita por uma estrutura de dados

no formato de uma árvore, em que cada nó é associado a um quadrante. Os nós internos são formados por um valor e quatro ponteiros para os seus nós filhos. Cada um desses filhos corresponde a um quadrante da região associada ao nó, sendo que a raiz da árvore representa todo o objeto. Já o valor armazenado pelo nó corresponde a uma versão simplificada da parte do objeto pertencente ao seu quadrante. Normalmente esse valor é uma média dos valores de seus subquadrantes. As folhas da árvore, por sua vez, representam os quadrantes que não são mais divididos, armazenando o valor real do bloco.

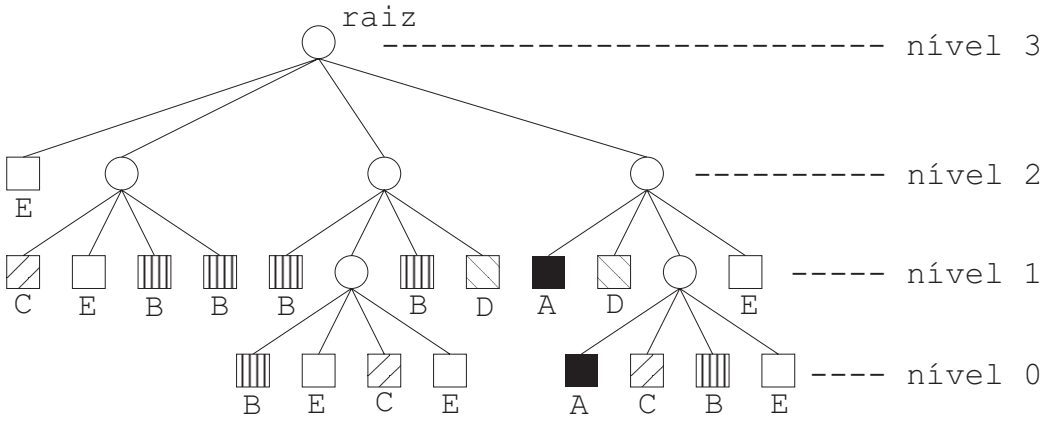
Portanto, é importante observar que a Quadtree armazena os dados com multiresolução e tem a capacidade de focar somente as partes de interesse do objeto. Estas características fazem dela uma estrutura muito útil para o armazenamento de imagens grandes para a sua posterior visualização, pois ela permite a recuperação de versões simplificadas da imagem bem como a recuperação somente do bloco visível da imagem. Caso a área total dos blocos homogêneos da imagem for grande, a Quadtree também serve para comprimi-la.

O algoritmo de busca em uma Quadtree é muito simples. É necessário fornecer o retângulo de busca e a profundidade máxima de busca, esta determina a resolução em que o dado é recuperado. O algoritmo desce a árvore visitando todas as subárvores cujos quadrantes interceptam a região de busca. Ao alcançar uma folha ou ao atingir a profundidade dada recupera-se o valor do nó em questão, acrescentando-o ao resultado. Caso o nó seja interno seus filhos não são visitados.

Normalmente, ao se construir a Quadtree de uma determinada imagem, os blocos não homogêneos são subdivididos até atingirem o tamanho de um *pixel*; desta forma, cada nó (nós internos e folhas) armazena o valor de um *pixel*. Entretanto, no caso de imagens grandes e bem diversificadas (não homogêneas), o tamanho da estrutura fica muito grande, reduzindo assim a sua eficiência. Uma solução para esta situação é a determinar de um limite para a subdivisão dos blocos, o que se faz escolhendo-se as dimensões mínimas de um bloco, ou seja, o quadrante de dimensões mínimas que não será mais subdividido. Para que a definição da imagem não seja perdida, os nós, ao invés de guardar um *pixel*, passam a armazenar blocos com as dimensões mínimas. A principal característica deste método é a obtenção de Quadtrees com profundidades



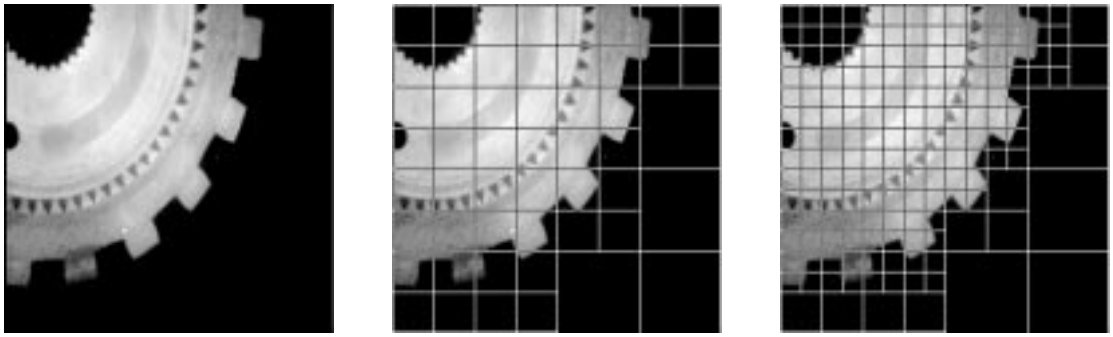
(a) Representação planar de uma imagem formada por 5 cores



(b) Estrutura

Figura 3.10: Exemplo de uma Quadtree

menores e, conseqüentemente de menor tamanho. As Figuras 3.11 e 3.12 ilustram um exemplo desse método, e a Figura 3.13 mostra diversas multiresoluções da imagem.



(a) Imagem Original

(b) Blocos mínimos de 32×32

(c) Blocos mínimos de 16×16

Figura 3.11: Decomposição de uma imagem de 256 cores e com dimensões de 256×256

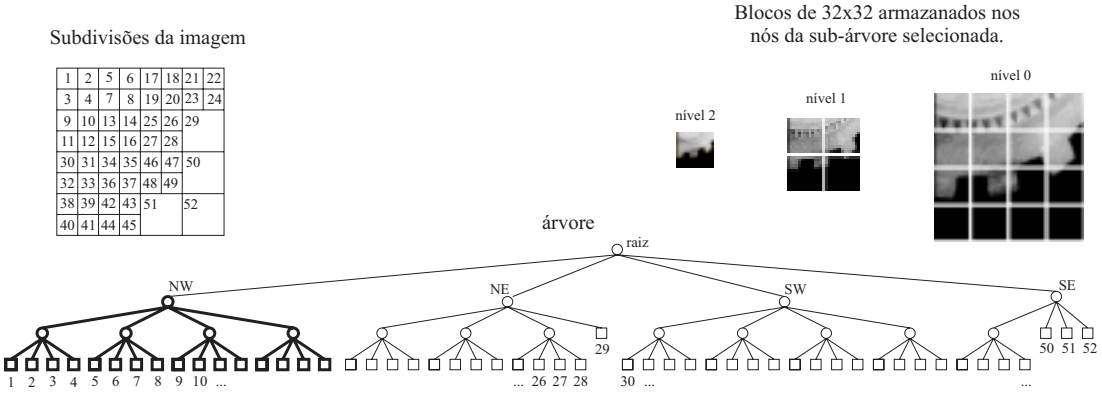


Figura 3.12: Representação da Quadtree da imagem da Figura 3.11, os blocos mínimos são de 32×32

A extensão da Quadtree para dimensões maiores é trivial. No caso do espaço tridimensional ela é denominada Octree e, ao invés de quadrantes, os objetos são divididos em oito octantes.

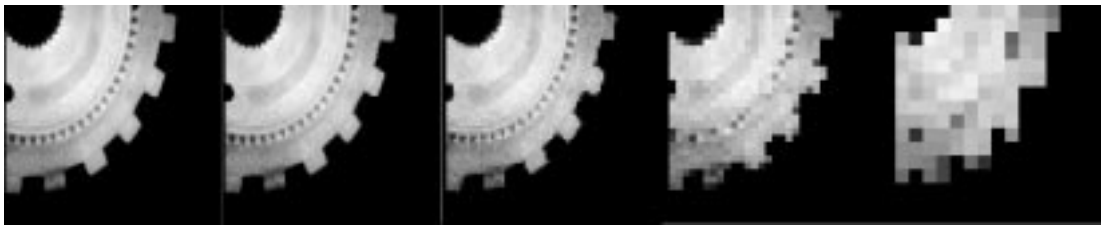


Figura 3.13: Diversas resoluções de uma imagem

Capítulo 4

Drops

Neste capítulo são propostos, através da definição do sistema Drops (*Drawing Objects on Persistent Systems*), os métodos para o armazenamento e recuperação de objetos gráficos. Eles são baseados nos conceitos apresentados nos dois capítulos anteriores. Sua principal característica está na utilização de estruturas de dados persistentes compostas por uma R-tree, V-trees e Quadtrees e de um subsistema de persistência.

Na seção 4.1 são discutidas as estruturas de dados utilizadas para o armazenamento dos dados. A seção 4.2 define como deve ser feita a conversão de figuras no formato convencional, como o CGM, para o formato do Drops. Na seção 4.3 é descrito o subsistema de persistência do Drops. Por fim, na seção 4.4 é definido o conjunto de elementos suportados pelo sistema.

4.1 Estruturas de Acesso

A estrutura proposta é composta basicamente por uma estrutura principal, que trabalha com todo o conjunto de objetos espaciais, e por outras estruturas secundárias que armazenam cada um dos objetos espaciais.

A estrutura primária tem a função de indexar espacialmente o conjunto de objetos espaciais, permitindo ao usuário recuperar somente os objetos que interceptem a área de interesse. Dada a natureza diversificada de cada objeto, que pode ser composto por linhas, regiões, imagens, marcas e/ou textos, essa indexação espacial primária é feita

com base no menor retângulo envolvente alinhado com os eixos, o MBR (*Minimum Bounding Rectangle*). A estrutura de dados principal é uma R-tree [Gut84]. A escolha da R-tree é justificada por ela ser um método de acesso eficiente à região para as *range queries*.

Como, porém, os mapas em CD-ROM são figuras estáticas (não sofrem atualizações), não há necessidade de utilizar uma estrutura de dados dinâmica como a R-tree convencional. O sistema Drops implementa, além da R-tree convencional, as R-trees compactas [RL85, KF93, LLE97]. Como a diferença entre elas está somente no algoritmo de construção da árvore, a escolha se faz por um parâmetro do sistema.

O fato da R-tree ser uma estrutura de dados hierárquica faz com ela também possa ser utilizada para otimizar operações de multiresolução. Para que isto ocorra, é preciso que os registros de cada nó tenham uma versão simplificada do conjunto de objetos armazenados na sua sub-árvore. A solução adotada neste trabalho foi a de acrescentar uma cor no registro de cada nó interno. Desta forma, cada sub-conjunto de objetos é simplificado pelo seu retângulo envolvente preenchido com essa cor, sendo que a cor descreve sua aparência e o MBR descreve a geometria dessa versão simplificada. Quando o retângulo envolvente de uma sub-árvore cabe dentro de uma área pequena (por exemplo, um *pixel*), a sub-árvore não precisa ser recuperada na operação de desenho. Quando o tamanho da caixa envolvente de um elemento for pequena mas a aproximação for inaceitável entram em jogo as estruturas secundárias. Para a descrição da aparência da simplificação é utilizada somente a cor, que é um atributo essencial, porém a inclusão de outros atributos é trivial. Também por simplicidade, é pega a cor de um dos objetos do subconjunto correspondente para preencher o retângulo.

No Drops as estruturas secundárias são utilizadas para armazenar os objetos mais complexos, como é normalmente o caso das linhas poligonais, dos polígonos e dos diversos tipos de imagens. Essas estruturas secundárias representam os objetos em multiresolução, permitindo assim a recuperação de versões simplificadas dos mesmos. Nesse caso existem métodos específicos para cada tipo de objeto. São utilizadas V-trees [MCD94] para armazenar as linhas poligonais e os polígonos e Quadtrees [FB74, Gar82, Sam84] para as imagens.

Os atributos dos objetos gráficos são armazenados em tabelas em vez de serem

armazenados junto com as primitivas, com a finalidade de reduzir o volume final do arquivo. São construídas quatro tabelas, cada uma para um grupo de atributos (marcas, linhas, preenchimento e texto), e cada objeto possui uma referência para os seus atributos na tabela correspondente.

No processo de armazenamento dos dados em disco é preciso levar em consideração a posição no arquivo onde cada dado é armazenado, pois a redução no tempo de acesso ao disco implica também na redução do deslocamento efetuado pela cabeça de leitura do disco. Portanto é importante que os dados lidos seqüencialmente também estejam armazenados seqüencialmente em disco. Isto não acontece inicialmente com o Drops, pois os objetos e parte da estrutura são armazenados à medida que eles são inseridos no Drops. Logo, é preciso reordenar todo o arquivo para que a estrutura de dados e os objetos gráficos não fiquem armazenados alternadamente dentro do arquivo e, além disso, para que os objetos espacialmente próximos também estejam próximos dentro do arquivo.

É importante observar que todas as estruturas de dados envolvidas na composição da estrutura do Drops oferecem uma série de alternativas em sua construção que podem afetar muito o desempenho da estrutura. Com a preocupação de otimizar ao máximo o processo de recuperação de dados, essas alternativas foram deixadas como opções de construção da estrutura. No Capítulo 5 todas essas alternativas serão exploradas com o objetivo de se descobrir qual resulta em melhores desempenhos. Entre as opções estão a utilização ou não de V-trees para armazenar as linhas poligonais e polígonos, a utilização ou não de Quadtrees para armazenar imagens, a escolha da ordem da R-tree, a escolha da ordem das V-trees e a reordenação ou não dos dados no disco.

4.2 Conversão de CGM para Drops

Devido à disponibilidade de mapas e figuras no formato CGM, torna-se necessário que o sistema Drops trate da conversão de um modelo para outro. O fato do Drops indexar os dados em vez de armazená-los de forma seqüencial faz com que, na conversão de formatos do tipo CGM para o Drops, a ordem original dos objetos seja perdida. Porém, nos casos em que há sobreposição de objetos, manter a ordem original durante a sua

visualização é fundamental para preservar a integridade da imagem resultante (veja a Figura 4.1). O Drops apresenta duas soluções para este problema: a ordenação dos objetos e a utilização de camadas.

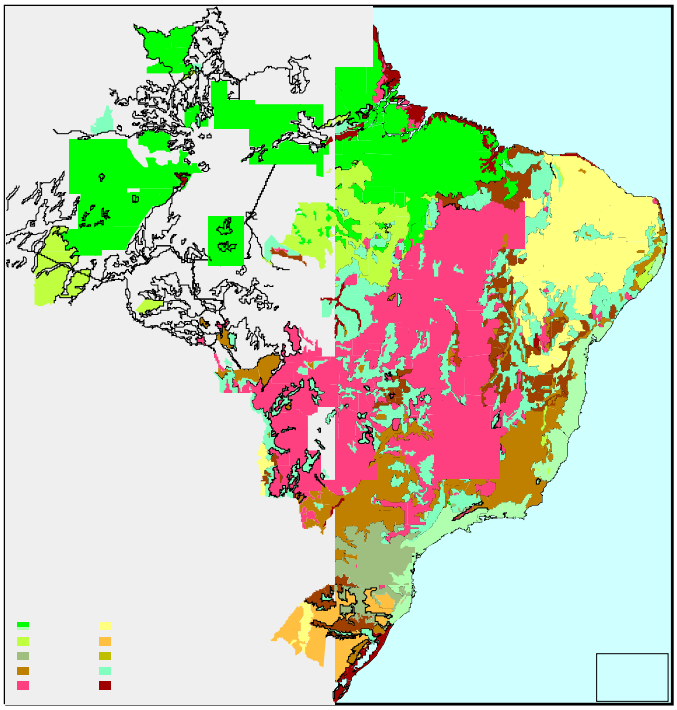


Figura 4.1: Este é o mesmo mapa da Figura 2.3, mas suas primitivas foram desenhadas na ordem em que elas foram recuperadas da R-tree.

No primeiro caso é armazenado junto com a descrição geométrica dos objetos o valor de um contador que indica sua ordem original. Quando os dados são recuperados do arquivo, primeiro são obtidos o contador e a referência dos objetos. Os objetos são então ordenados, utilizando o algoritmo de *Quick Sort*, em função de sua ordem original. Somente depois disto é que os objetos são recuperados.

Já no segundo caso, os objetos são armazenados em diversas camadas. Eles devem ser separados de forma que não haja sobreposição entre os objetos pertencentes à mesma camada, fazendo assim com que a única preocupação seja manter a ordem original das camadas. O Drops trata internamente o conceito de camadas através da utilização de uma R-tree distinta para cada camada, de forma que cada conjunto de objetos seja indexado separadamente. Esta solução de ordenação é na maioria dos casos suficiente para resolver o problema de sobreposição dos objetos, uma vez que normalmente os

dados em bases de dados geográficas já se encontram organizados originalmente em camadas.

É importante mencionar que esse problema de sobreposição também é abordado nos métodos de acesso espacial baseados na técnica de regiões disjuntas (*clipping*), como a R^+-tree [SRF87], e nos métodos de múltiplas camadas [GG98], como a Multilayer-Grid-File [SW88] e a R-File [HSW90]. Neste caso é necessário organizar os objetos de forma que não haja sobreposição entre os objetos dentro da mesma camada. Entretanto, apesar desses métodos particionarem o espaço em regiões disjuntas, eles não foram projetados com o propósito manter a ordenação original entre os objetos indexados.

4.3 OPS

Todas as estruturas de dados do Drops utilizam um subsistema de persistência, o OPS (*Persistent Object Storage Subsistem*), proposto e implementado por Mediano [Med97], para o gerenciamento da memória secundária.

O OPS oferece um ambiente extensível e configurável desenvolvido em C++ sob a forma de uma biblioteca. Sua API é totalmente orientada a objetos, sendo constituída principalmente por classes virtuais puras sobre as quais são modelados os elementos envolvidos no processo de acesso ao meio de armazenamento definitivo.

Esse processo é modelado pelo OPS a partir de três abstrações: o *objeto persistente*, o *meio persistente* e o *gerenciador*. Os *objetos persistentes* são aqueles que têm a propriedade de existir (persistir) entre as execuções das aplicações que fazem uso deles. Os *meios persistentes* são aqueles onde são armazenados e recuperados os objetos persistentes (disco, memória principal, fita, etc.). O *gerenciador* é o responsável por fazer a alocação e a liberação da área utilizada pelos objetos persistentes no meio persistente. Essas abstrações são implementadas em um modelo de camadas que permite que tanto a manipulação dos dados persistentes quanto a sua alocação sejam feitas de forma transparente para a aplicação.

O OPS permite também a combinação de vários meios de acesso e de vários gerenciadores. Para tanto o gerenciador pode, em vez de acessar diretamente o meio

persistente, trabalhar em cima de outro gerenciador, permitindo assim o uso de outros gerenciadores para fazer controles de concorrência e controles de versão, entre outras funções. Quanto ao meio persistente, ele pode funcionar também como uma camada em cima de outro meio persistente, geralmente armazenando temporariamente parte dos dados, como é o caso do *cache* em memória principal. A configuração da combinação de meios e gerenciadores que a aplicação utiliza pode ser feita em tempo de execução. Essa arquitetura está ilustrada na Figura 4.2.

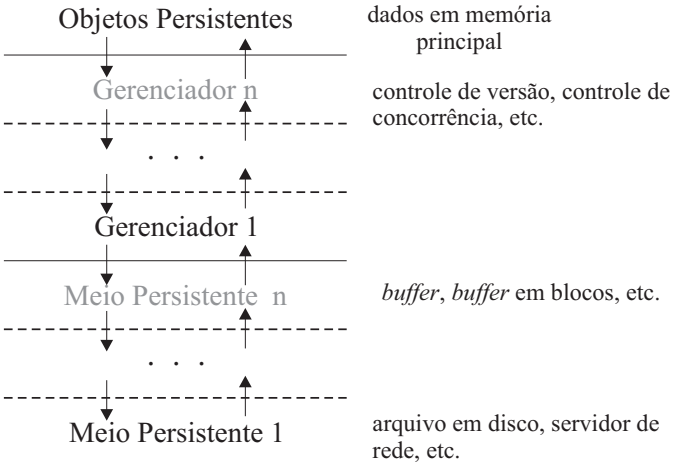


Figura 4.2: Arquitetura em Camadas do OPS

Além das classes virtuais puras, o OPS oferece algumas classes derivadas de meios e gerenciadores, visando assim oferecer um conjunto inicial de técnicas de acesso a bancos de dados. O Drops utiliza alguns desses meios e gerenciadores previamente definidos pelo OPS. Na Figura 4.3 está ilustrada a configuração utilizada pelo Drops. Ele emprega um arquivo como meio de armazenamento definitivo (camada inferior), e um *buffer* em blocos que tem a função de guardar os dados mais recentemente acessados em memória principal. O meio *buffer* em blocos é constituído por uma quantidade pré-determinada de blocos de mesmo tamanho da memória principal.

4.4 Elementos

O conjunto de elementos do Drops pode ser dividido em três classes funcionais: os elementos primitivos, os atributos e os elementos de agrupamento. A classe de elementos

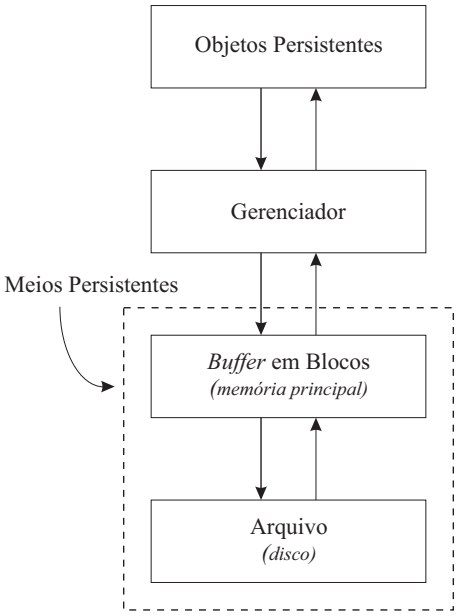


Figura 4.3: Arquitetura de Persistência do Drops

primitivos foi escolhida de forma a ser muito semelhante à classe das primitivas da biblioteca gráfica *Canvas Draw* [Tec98], visando tornar o mapeamento do Drops para o *Canvas Draw* trivial, facilitando o seu uso na tarefa de desenho.

4.4.1 Primitivos

Os elementos primitivos contêm a descrição geométrica dos objetos gráficos. Eles são divididos em dois grupos, o das primitivas vetoriais e o das primitivas matriciais. A seguir estão listadas todas as primitivas vetoriais, sendo que ao lado de cada elemento estão as funções necessárias para a sua definição. Na implementação atual do Drops são utilizados *floats* de 32 *bits* (mantissa de 24 *bits*, expoente de 6 *bits* e 1 *bit* para o sinal) para armazenar as coordenadas das posições no espaço.

- Marca (Ponto) – *Mark(float x, float y)*
  É uma marca localizada no ponto *(x, y)*.

- Linha – *Line(float x0, float y0, float x1, float y1)*
  É uma linha que vai de *(x0, y0)* até *(x1, y1)*.

- Linha Poligonal – *Begin(OPEN_LINES ou CLOSED_LINES), Vertex(float x, float y), End()*
  É uma seqüência de linhas, formada por uma série de vértices conectados. Sua definição é feita através das três funções mostradas acima. A função *Begin* indica o início da definição da linha poligonal. Se o valor do parâmetro passado for *CLOSED_LINES*, o último vértice será conectado ao primeiro. A função *Vertex* acrescenta um vértice à definição da linha poligonal e a função *End* finaliza a definição.

- Arco – *Arc(float xc, float yc, float w, float h, float angle1, float angle2)*
  Define um arco de uma elipse alinhada com o eixo. O par de coordenadas *(xc, yc)* indica a posição do centro da elipse. As dimensões *w* e *h* são os eixos elípticos X e Y, respectivamente. Os ângulos *angle1* e *angle2* definem o início e o fim do arco. O arco começa no ponto *(xc + (w/2) * cos(angle1), yc + (h/2) * sin(angle1))* e termina em *(xc + (w/2) * cos(angle2), yc+(h/2) * sin(angle2))*. Os valores dos ângulos são fornecidos em graus.

- Caixa – *Box(float xmin, float xmax, float ymin, float ymax)*
  Define um retângulo preenchido. O seu canto inferior esquerdo está em *(xmin, ymin)* e seu canto superior direito está em *(xmax, ymax)*.

- Polígono – *Begin(FILL), Vertex(x, y), End()*
  É um polígono formado através da conexão de uma seqüência de vértices e pelo preenchimento de seu interior. Seu processo de definição é semelhante ao de uma linha poligonal, a única diferença está no parâmetro repassado para a função *Begin*, que tem de ser *FILL*. Esse parâmetro indica que o último vértice será conectado ao primeiro e a primitiva será preenchida.

- Setor – *Sector(float xc, float yc, float w, float h, float angle1, float angle2)*
  Define um arco preenchido de uma elipse alinhada com os eixos. É semelhante ao arco, a única diferença está no fato de ele ser preenchido.

- Texto – *Text(int x, int y, unsigned char *text)*
  Define o texto *text* na posição *(x, y)*.

4.4.2 Atributos

Os atributos são fornecidos pela aplicação utilizando o conceito de valor corrente, como é comum em sistemas gráficos como o *Canvas Draw*. Para reduzir o armazenamento interno, o Drops coloca estes atributos em tabelas, sem registros repetidos, e apenas uma referência para eles é colocada em cada primitiva.

Os atributos do Drops são:

- Cor – *Color(long int color)*
  Define a cor no formato RGB que será utilizada para desenhar a primitiva. Ela é armazenada em uma variável do tipo *long int* (32 *bits*), onde são destinados 8 *bits* para cada componente. Conseqüentemente somente 24 *bits* da variável contêm alguma informação útil. A cor é utilizada por todas as primitivas vetoriais.

- Modo de Escrita – *WriteMode(int mode)*
  São três os modos de escrita: *REPLACE, XOR* e *NOT_XOR*. O modo *REPLACE* é o padrão. Este atributo é armazenado em um *char* (8 *bits*) e também é utilizado por todas as primitivas vetoriais.

- Tipo de Marca – *MarkType(int type)*
  Pode assumir os valores *PLUS, STAR, CIRCLE, X, BOX, DIAMOND, HOLLOW_CIRCLE, HOLLOW_BOX, HOLLOW_DIAMOND*. São utilizados 8 *bits* para armazenar estes dados.

- Tamanho de Marca – *MarkSize(float mark)*
  Configura o tamanho da marca (em coordenadas de mundo). O valor é armazenado em 32 *bits*.

- Estilo de Linha – *LineStyle(int style)*
  Pode assumir os valores: *CONTINUOUS, DASHED, DOTTED, DASH_DOT, DASH_DOT_DOT*. São utilizados 8 *bits* para armazenar este valor.

- Espessura de Linha – *LineWidth(float width)*
  Configura a espessura (em coordenadas de mundo) da linha. São utilizados 32 *bits* para o seu armazenamento.

- Cor de Fundo – *Background(long int)*
  Define a cor de fundo das primitivas preenchidas. Seu valor é armazenado da mesma forma que o atributo Cor, mencionado anteriormente.

- Opacidade – *BackOpacity(unsigned char opacity)*
  Configura a opacidade de fundo para as primitivas de preenchimento baseadas em cores de frente e fundo. Ela pode ser *OPAQUE* (opaca) ou *TRANSPARENT* (transparente). Se for transparente, somente a cor de frente será desenhada; caso contrário, a cor de fundo é desenhada apagando tudo o que estiver atrás. O valor padrão é *TRANSPARENT*. Seu valor é armazenado em um *char* (8 *bits*).

- Estilo Interior – *InteriorStyle(unsigned char style)*
  Tem efeito somente sobre as primitivas preenchidas. Pode assumir os valores *SOLID, HOLLOW ou HATCH*. O valor padrão é SOLID. É armazenado em um *char* (8 *bits*).

- *Hatch – Hatch(unsigned char style)*
  Seleciona um estilo *hatch* pré-definido e configura o estilo interior para *HATCH*. Os estilos pré-definidos são: *HORIZONTAL, VERTICAL, FDIAGONAL, BDIAGONAL, CROSS ou DIAGCROSS*. Seu valor é armazenado em um *char* (8 *bits*).

- Fonte – *Font(unsigned char typeface, unsigned char style, float size)*
  Seleciona a fonte do texto. Ela pode assumir os valores *SYSTEM, COURIER, TIMES_ROMAN ou HELVETICA*. Seu valor é armazenado em um *unsigned char* (8 it bits). Sua definição é feita juntamente com a definição de estilo e tamanho de fonte, que serão mencionados a seguir, através da função *Font*.

- Estilo de Fonte – *Font(unsigned char typeface, unsigned char style, float size)*
  Pode assumir os valores *PLAIN, BOLD, ITALIC ou BOLD_ITALIC*. É utilizado um *unsigned char* (8 *bits*) para armazená-lo.

- Tamanho de Fonte – *Font(unsigned char typeface, unsigned char style, float size)*
  Define o tamanho da fonte em coordenadas de mundo. É utilizado um *float* (32 *bits*) para armazenar seu valor.

- Alinhamento de Fonte – *TextAlignment(unsigned char alignment)*
  Define o alinhamento vertical e horizontal de um texto. Ele pode ser *NORTH, SOUTH, EAST, WEST, NORTH_EAST, NORTH_WEST, SOUTH_EAST, SOUTH_WEST, CENTER, BASE_LEFT, BASE_CENTER ou BASE_RIGHT*. São utilizados 8 *bits* para armazenar seu valor.

Não houve muita preocupação em economizar espaço no armazenamento dos valores dos atributos, pois foi observado que o total de espaço ocupado pelos atributos equivale a uma fração muito pequena do tamanho do arquivo de elementos primitivos.

Os atributos são agrupados conforme mostra a Tabela 4.1

grupos	atributos	primitivas associadas
Atributos de Marca (*Mark*)	cor, modo de escrita, tipo de marca e tamanho de marca	marca
Atributos de Linha (*Pen*)	cor, modo de escrita, estilo de linha e espessura de linha	linha, linha poligonal e arco
Atributos de Preenchimento (*Brush*)	cor, modo de escrita, cor de fundo, opacidade e estilo interior	caixa, polígono e setor
Atributos de Texto (*Font*)	cor, modo de escrita, cor de fundo, opacidade, fonte, estilo de fonte, tamanho de fonte e alinhamento de fonte	texto

Tabela 4.1: Grupos de atributos do Drops.

4.4.3 Imagens

As primitivas matriciais do Drops são imagens. Cada tipo de imagem armazena as informações de cores de cada célula em dos seguintes formatos: RGB, RGBA e indexado.

As primitivas *raster* são:

- Imagem RGB – *ImageRGB(int iw, int ih, unsigned char *r, unsigned char *g, unsigned char *b, float x, float y, float w, float h)*

Define uma imagem RGB (*True Color*). Cada um de seus componentes é fornecido separadamente em um vetor de *bytes*. Os parâmetros *iw* e *ih* são as dimensões da imagem, ou seja, o número de células que ela possui ao longo de cada dimensão. O seu canto inferior esquerdo está localizado na posição *(x, y)*, e *w* e *h* são as dimensões do retângulo onde a imagem está localizada.

- Imagem RGBA – *ImageRGBA(int iw, int ih, unsigned char *r, unsigned char *g, unsigned char *b, unsigned char *a, float x, float y, float w, float h)*
  Define uma imagem RGBA. Ela é idêntica à imagem RGB, exceto pelo fato de possuir um componente de cor a mais, o alfa, que determina a transparência da célula.

- Imagem Indexada – *ImageMap(int iw, int ih, unsigned char *index, long int palette, float x, float y, float w, float h)*
  É semelhante à imagem RGB, exceto pelo fato de suas cores estarem no formato indexado (256 cores). É fornecida a paleta de cores da imagem, que contém no máximo 256 cores, e também o índice de cor de cada célula.

- *Pixel – Pixel(float x, float y, long int color)*
  Define a cor RGB de um *pixel* que se encontra na posição *(x, y)* em coordenadas de mundo.

4.4.4 Agrupamento

Os métodos de agrupamento do Drops são:

- Forma – *int BeginShape(void), EndShape(void), Shape(int id, float x, float y, float rot, float scale)*
  Uma forma é definida através das funções *BeginShape* e *EndShape*. Todos os objetos gráficos definidos entre a chamada a essas duas funções fazem parte da forma. A função *BeginShape* retorna o identificador da nova forma. A instanciação de uma forma, por sua vez, é feita com a função *Shape*, passando como parâmetros o identificador da forma e a translação, a rotação e a escala a serem aplicadas sobre as primitivas que a compõem.

- Camada – *Layer(char *name)*

  A camada agrupa os objetos gráficos que estão dentro de um mesmo contexto. Toda camada tem associados um nome e um número, cujo o valor é a ordem em que ela foi criada. A função *Layer* cria a camada de nome *name* e a torna corrente. Desta forma os elementos pertencem sempre à última camada definida. No processo de visualização, os números das camadas determinam a ordem em que elas serão recuperadas do arquivo. Pode-se configurar cada camada como visível ou invisível; as camadas invisíveis não serão recuperadas do arquivo. O padrão é que todas as camadas sejam visíveis.

Capítulo 5

Experimentos

A fim de conferir a eficiência do sistema proposto implementamos uma biblioteca capaz de gerar e ler arquivos no formato Drops. A partir dela desenvolvemos aplicações para converter dados armazenados em outros formatos gráficos (ex.: CGM, LIN, Shp, etc.) para o formato Drops, além de uma aplicação para visualizar dados no formato Drops. A Figura 5.1 ilustra esta aplicação.

Tanto a biblioteca quanto as aplicações foram desenvolvidas na linguagem C++, utilizando o sistema de interface IUP [Tec99] e a biblioteca gráfica CD - *Canvas Draw* [Tec98]. A estratégia de implementação constituiu em colocar o sistema Drops como um controlador *driver* configurável do *Canvas Draw*. Com isto ganha-se conectividade com todos os formatos de arquivos gráficos suportados pelo CD (Figura 5.2). Com a portabilidade das ferramentas utilizadas, os testes podem ser feitos tanto na plataforma UNIX quanto em Windows. Optamos por utilizar o sistema Linux rodando em Pentium PC.

Foram realizadas diversas séries de experimentos numéricos, cada uma visando testar uma característica específica do sistema Drops. Inicialmente essas características podem ser divididas em multiresolução e *clipping*. Contudo, na multiresolução podemos distinguir quando ela é realizada no nível da R-tree e no nível dos objetos, que por sua vez é diferente para cada tipo de objeto. Por sua vez, o *clipping* é influenciado pela forma com que a R-tree foi construída e pelo fato de ser feito sobre o conjunto de objetos ou sobre o objeto em si.

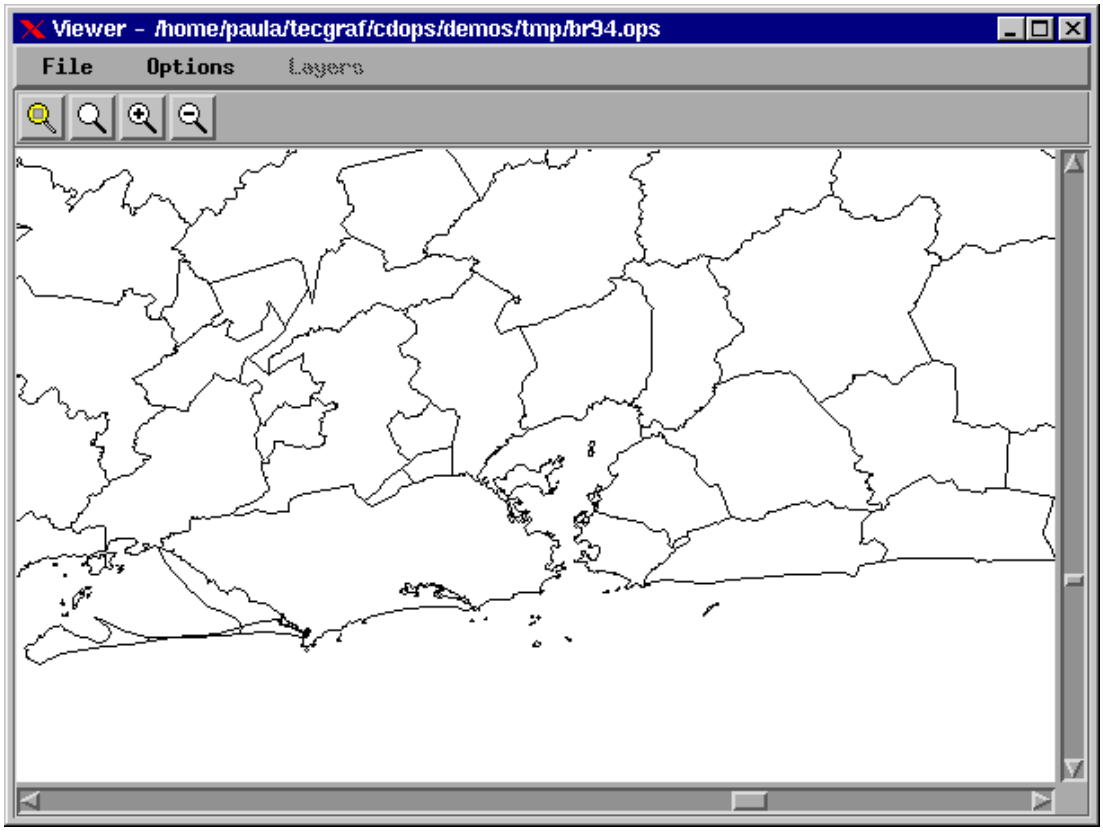


Figura 5.1: Aplicação de Visualização

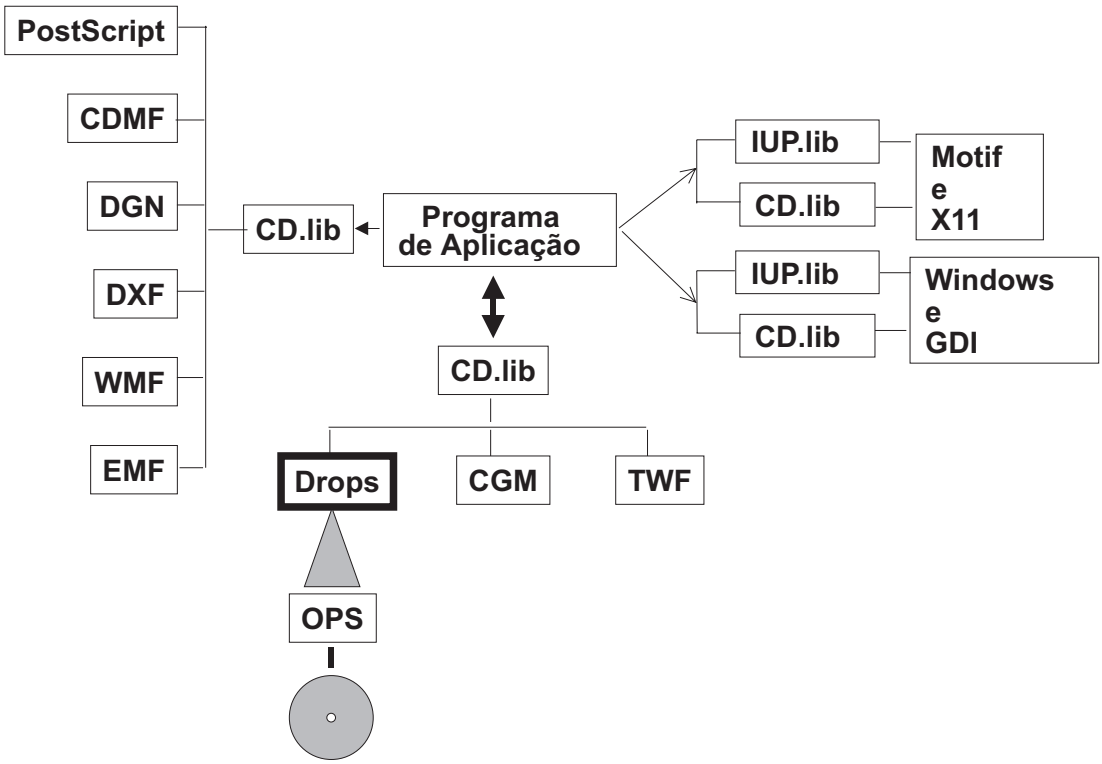


Figura 5.2: Organização do Canvas Draw.

Para comparar o desempenho do sistema Drops, todos os testes realizados foram também feitos com o controlador CGM do CD no formato de arquivo binário. Um ponto importante desta comparação é que, enquanto o Drops armazena coordenadas em números de ponto flutuante, o controlador CGM codifica esses números como inteiros de 16 *bits*. Isso naturalmente reduz o espaço em memória utilizado pelo CGM.

A medida de desempenho normalmente utilizada para medir a eficiência de índices em Banco de Dados é o número de acessos a páginas de disco. Isto porque, em uma consulta, geralmente o tempo de CPU é muito menor que o tempo de acesso à memória secundária. É também comum contar o número de nós visitados na árvore índice como um indicador do número de acessos à página de disco. Isto porque normalmente cada nó corresponde a uma página de disco. Com o sistema de *cache* ativado, esta correspondência deixa de ser verdadeira devido à "bufferização" que é empregada nestes casos [LL96].

Neste trabalho estamos interessados em comparar a eficiência do controlador Drops quando comparado com o controlador CGM. Como este é um arquivo seqüencial que é lido ou gravado todo de uma vez, resolvemos considerar o tempo de CPU como nossa medida principal. O número de nós visitados também é apresentado para complementar a informação.

Todos os testes de tempo deste trabalho lêem dados ou do sistema Drops ou de arquivos CGM e geram arquivos CGM, através dos controladores do *Canvas Draw*. Como estimamos que o tempo foi constante para todos os testes, ele não influencia a análise relativa das diversas opções.

5.1 Dados

Neste trabalho foram utilizados os seguintes conjunto de dados:

- *BR Indígena* - Mapa das áreas indígenas do Brasil (Figura 5.3). Obtido a partir de um arquivo no formato CGM.
- *BR Municípios* - Mapa de municípios do Brasil [IBG] (Figura 5.4). Obtido de um CD-ROM disponibilizado pelo IBGE no formato Shape File. Convertido

para o formato LIN (Apêndice A).

- *Ásia* - Mapa da Ásia [CIA], constituído pelas fronteiras do continente (Figura 5.5). Obtido na WWW no formato BIN (Apêndice A).

- *Madureira* - Mapa das edificações do bairro de Madureira (Figura 5.6). Este mapa não está completo. Seu dado original é formado por uma série de arquivos Shape File que foram montados em arquivo só.

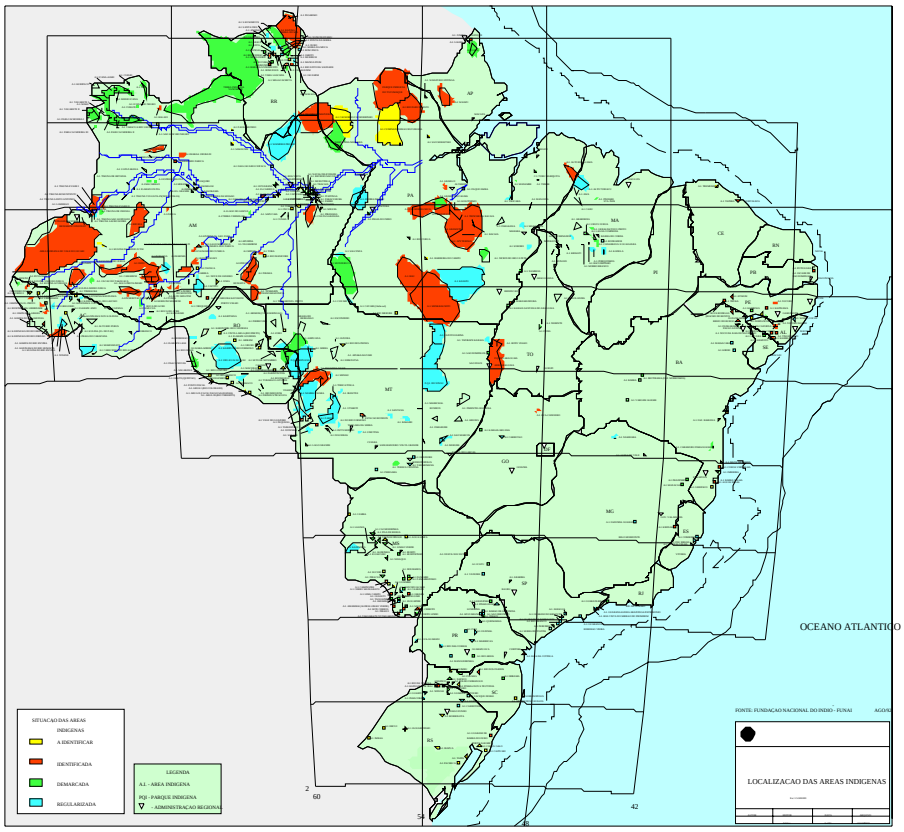


Figura 5.3: Mapa das Áreas Indígenas do Brasil

Estes dados de teste foram escolhidos dentre os dados reais disponíveis por suas características diversificadas, algumas das quais estão ilustradas na Tabela 5.1. O mapa BR Indígena não é um arquivo grande mas contém uma grande variedade de tipos de objetos gráficos e de atributos. Os outros possuem basicamente linhas poligonais de cor preta. Os arquivos BR Municípios e Ásia têm como característica marcante o alto nível de complexidade das suas linhas poligonais, como pode ser verificado pelas

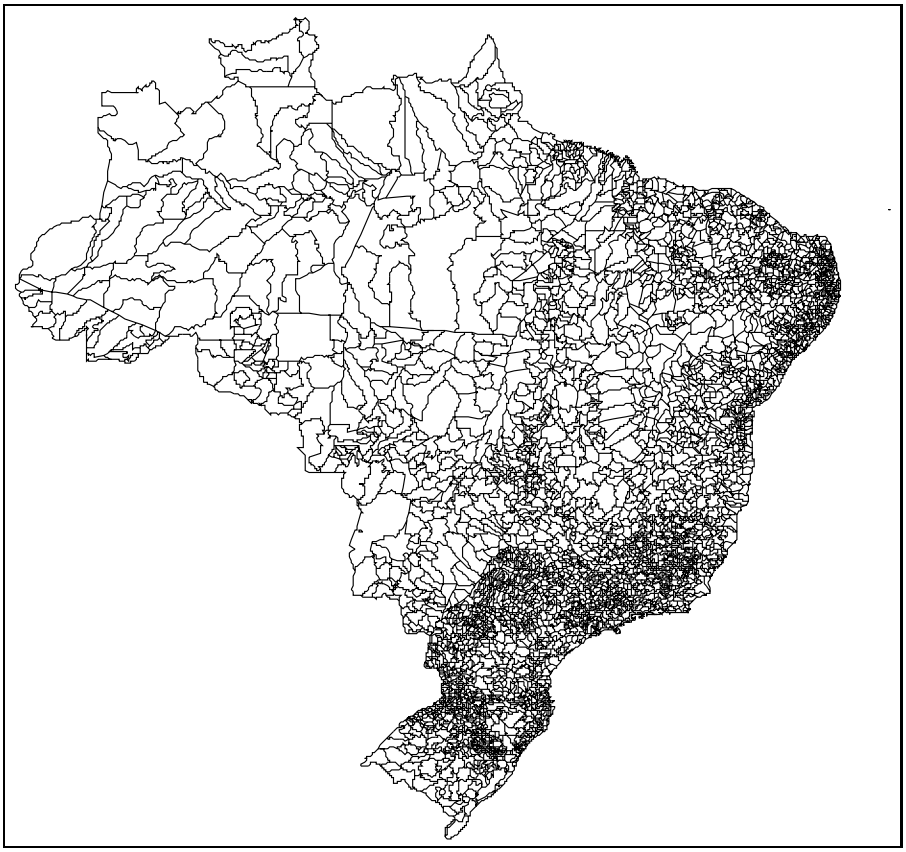


Figura 5.4: Mapa dos Municípios do Brasil

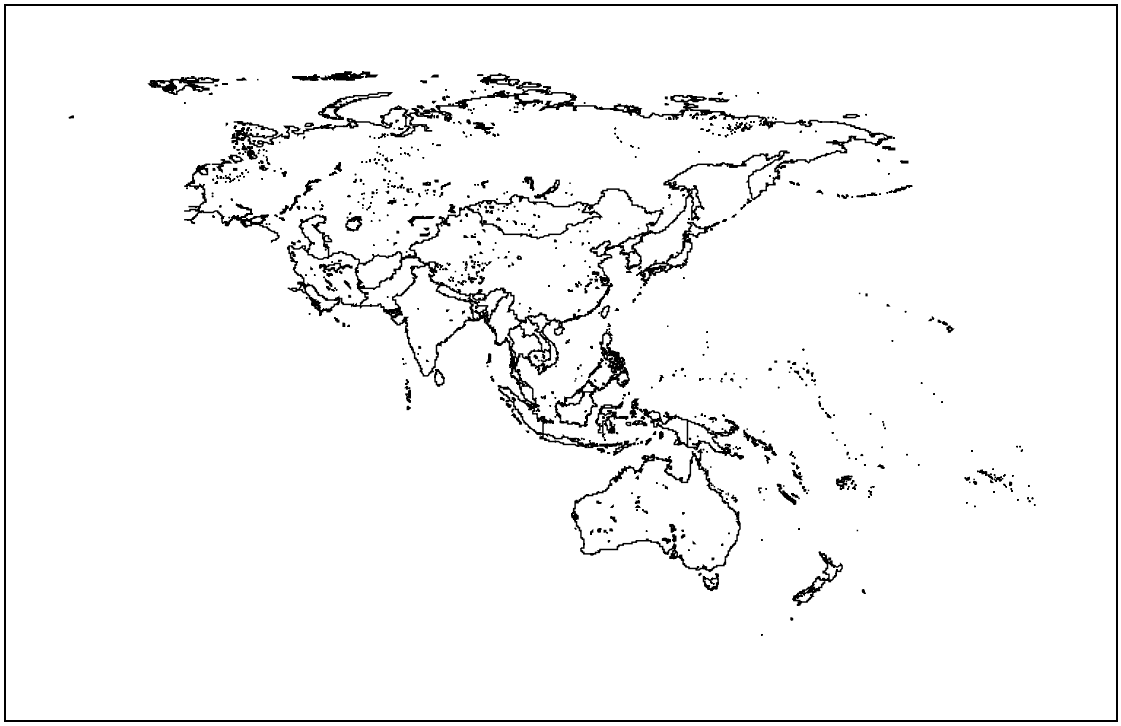


Figura 5.5: Mapa da Ásia

(a) Dado completo (b) *Zoom* sobre uma pequena área do dado

Figura 5.6: Edificações de Madureira

médias de vértices por linha poligonal. Por sua vez, os dados de edificação apresentam um grande número de objetos, que são relativamente simples.

Dado	**Formato**	**Tamanho (Kb)**	**Número de Objetos**	**Média de Vértices por Poligonal**	**Número de Atributos**
BR Indígena	CGM	829	2243	111	31
BR Municípios	BIN	5665	16596	43	1
Ásia	BIN	8088	9533	108	1
Madureira	SHP	17247	63062	8	1

Tabela 5.1: Características dos Dados

5.2 *Clipping* **(***Zoom***) ou** *Range Queries*

O método mais comum para testar o desempenho das *range queries* consiste em fazer uma série de consultas variando-se o tamanho da área dos retângulos de consulta, com valores que vão de 0% a no máximo 80% da área total do dado. Para cada porcentagem de área é calculada a média das respostas obtidas de várias consultas uniformemente distribuídas dentro do domínio espacial de dados. Com base neste modelo, alguns autores [KF93, LL96] propõem métodos analíticos para computar o desempenho das R-trees para as *range queries* utilizando como parâmetros suas características geométricas. Na Tabela 5.2 estão listadas algumas características geométricas das diversas R-trees construídas para o armazenamento dos conjuntos de dados.

Nos experimentos desta seção o procedimento difere ligeiramente do proposto por [KF93, LL96]. Enquanto os autores procuram distribuir uniformemente no domínio o centro da janela de *zoom*, aqui nós procuramos manter o centro em um ponto fixo localizado na regiões mais densas do dado. Acreditamos que o método proposto produz um resultado médio, enquanto que o nosso tende a estimar o pior caso. Com os resultados foram produzidos gráficos de porcentagem de área de visualização x tempo.

5.2.1 Influência da Ordem da R-tree nas *Range Queries*

Os Gráficos 5.1 e 5.2 mostram o tempo e o número de nós visitados em função da área de visualização para a variação da ordem (min e max) da R-tree. Apesar dos gráficos de tempo não apresentarem diferenças significativas, a R-tree (20, 50) reduziu significativamente o número de nós visitados. Nos gráficos que seguem essa foi a ordem utilizada.

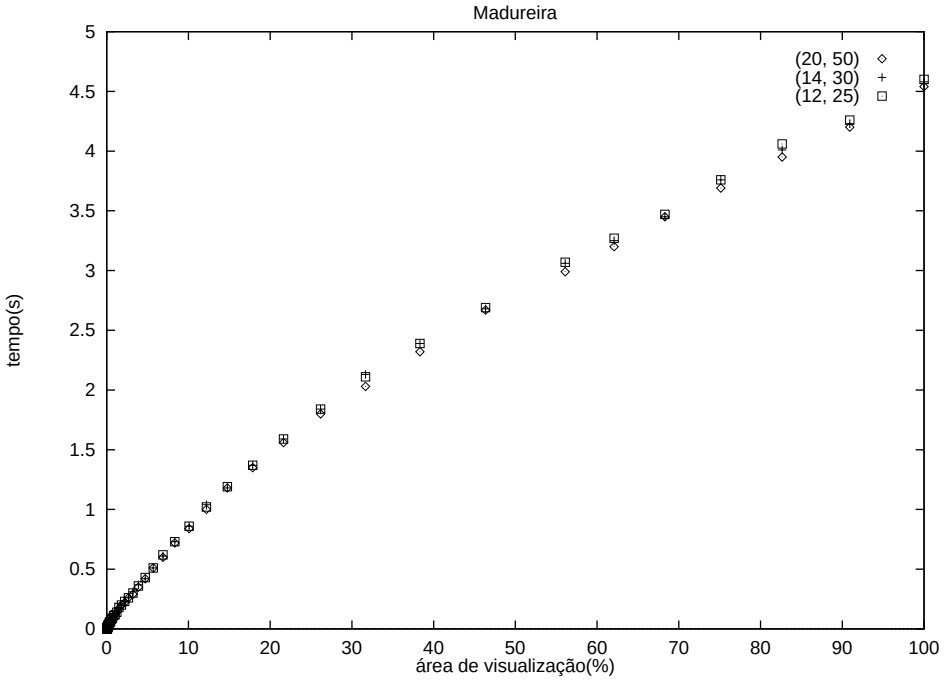


Gráfico 5.1: Madureira - variação da ordem da R-tree (tempo x área de visualização)

5.2.2 Comparação Drops versus CGM para *Range Queries*

Os Gráficos 5.3, 5.4, 5.5 e 5.6 mostram o tempo em função da área de visualização. Note-se que nestes experimentos não foi utilizada a multiresolução (tolerância zero), embora as V-Trees tenham sido utilizadas para o armazenamento das linhas poligonais longas (BR Indígena, BR Municípios e Ásia).

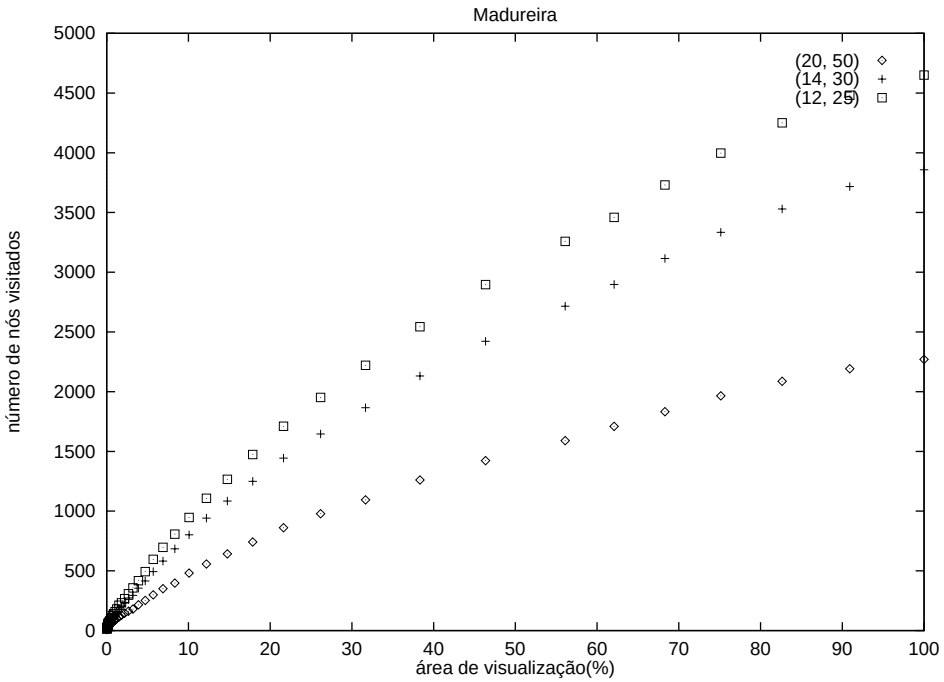


Gráfico 5.2: Madureira - variação da ordem da R-tree (nós visitados x área de visualização)

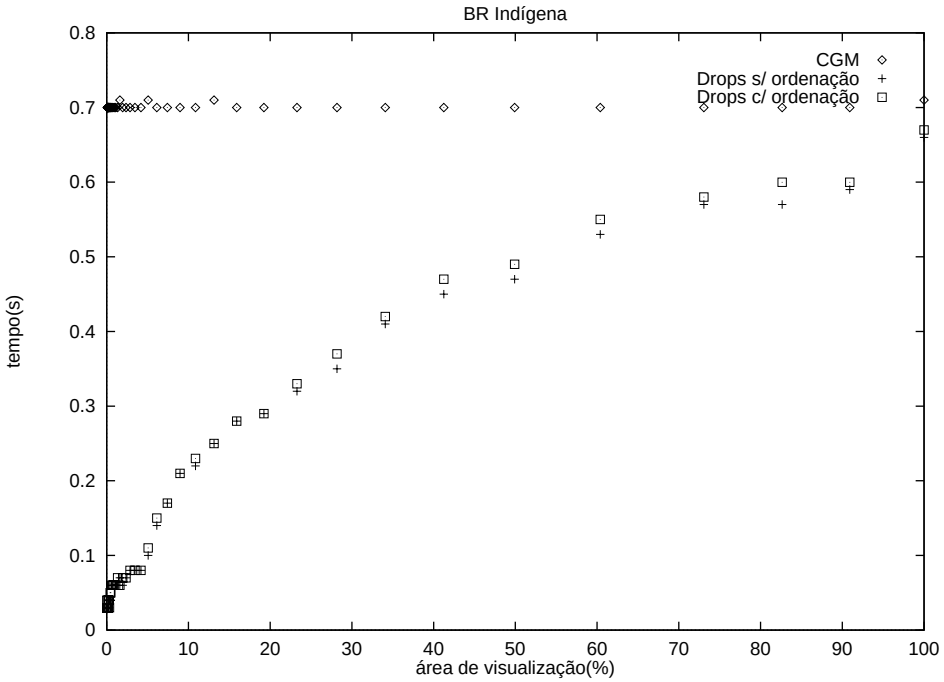


Gráfico 5.3: BR Indígena - Drops versus CGM

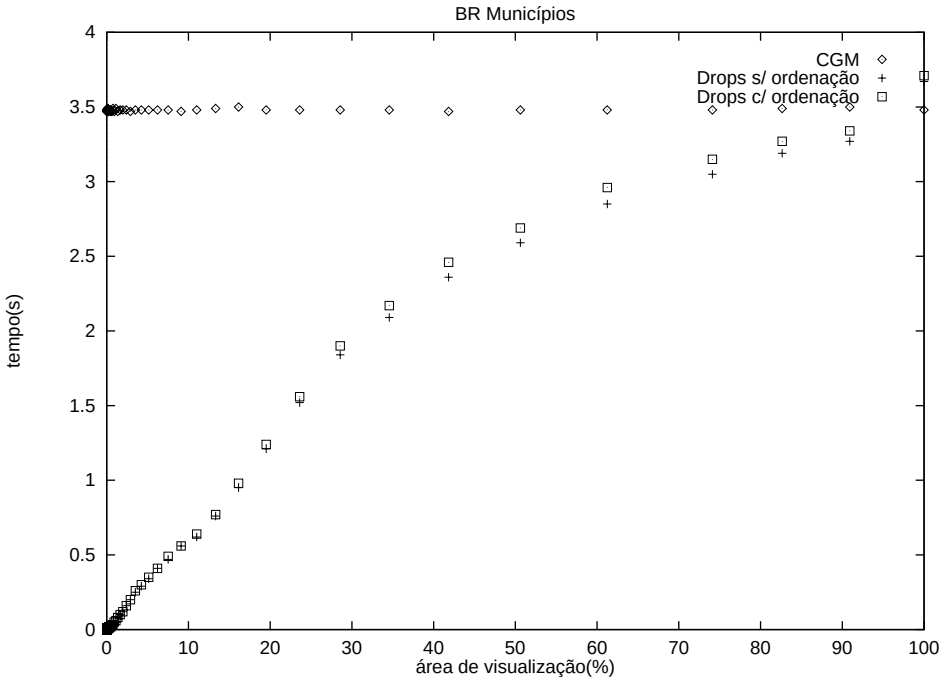


Gráfico 5.4: BR Municípios - Drops versus CGM

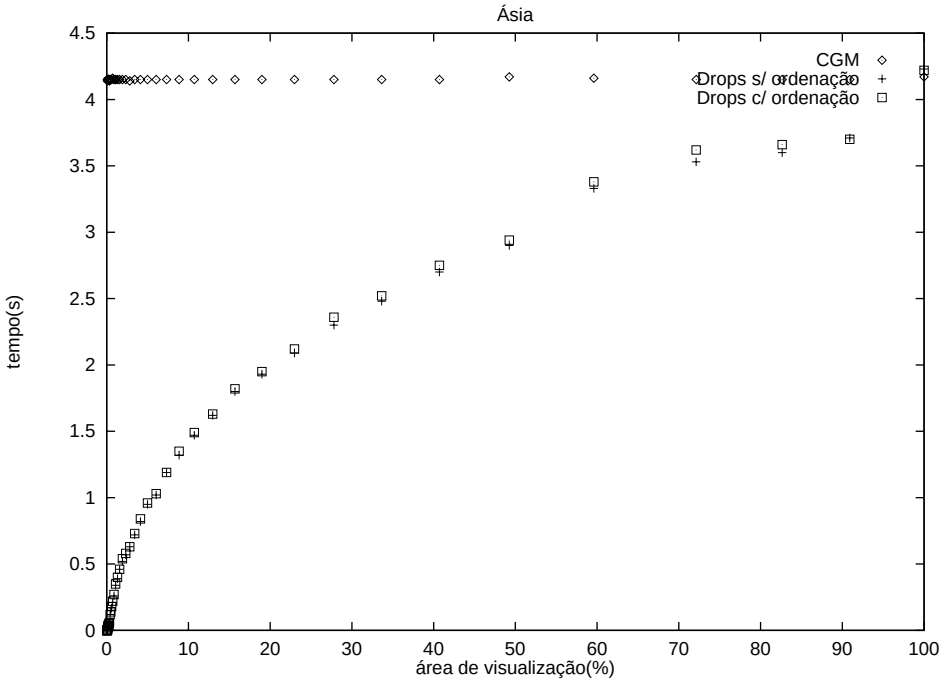


Gráfico 5.5: Ásia - Drops versus CGM

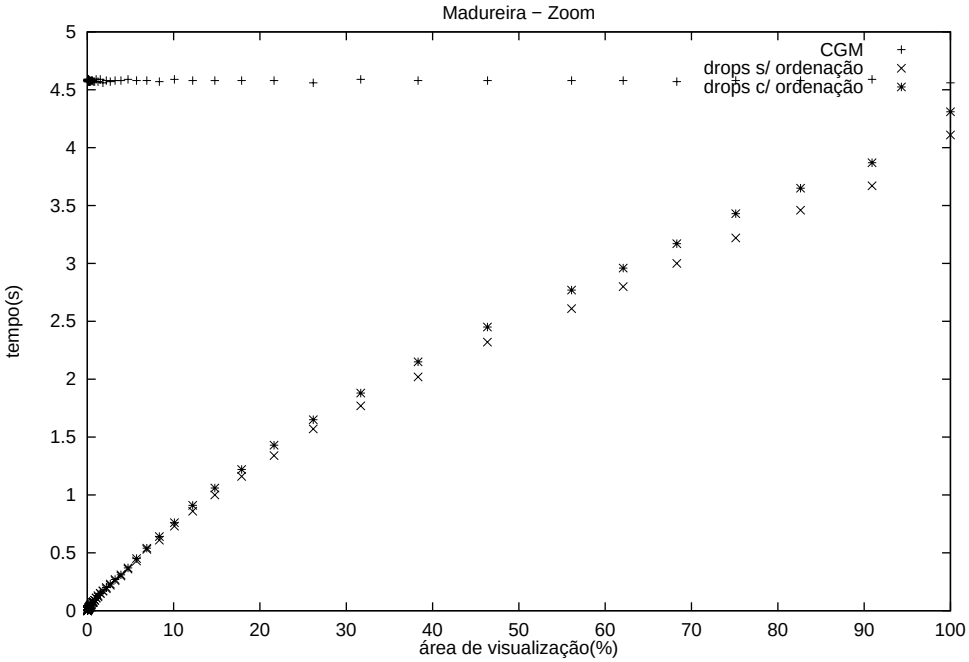


Gráfico 5.6: Madureira - Drops sem V-trees versus CGM

5.2.3 Uso das R-trees Compactas nas *Range Queries*

A Tabela 5.2 mostra o tamanho, a profundidade e a porcentagem de ocupação dos nós para as seguintes versões da R-tree: clássica de Guttman [Gut84], ordenada em X [RL85], *packed* com Hilbert [KF93] e STR [LLE97].

Os Gráficos 5.7, 5.8, 5.9, 5.10, 5.11 e 5.12 mostram o tempo e o número de nós visitados em função da área de visualização para diversas versões da R-tree. Note-se que a utilização destas reduz significativamente o número de nós visitados mas praticamente não apresenta qualquer efeito significativo sobre o tempo de processamento.

5.2.4 Efeito da Reordenação nas *Range Queries*

O Gráfico 5.13 mostra o gráfico de tempo x área de visualização utilizando a R-tree dinâmica e a Hilbert R-tree reordenadas e não reordenadas.

	R-tree Dinâmica				R-tree Compacta			
Dados	T	D	N	O	T	D	N	O
BR Indígena	1436	3	81	57.4	1400	2	45	100.0
BR Municípios	6480	3	607	56.4	6220	2	339	99.9
Ásia	8520	3	303	64.9	8412	3	195	99.8
Madureira	7220	4	2269	57.6	6264	3	1288	99.9

Legenda

T tamanho (Kb)
D profundidade
N número de nós
O % de ocupação dos nós

Tabela 5.2: Características geométricas das R-trees construídas para os dados

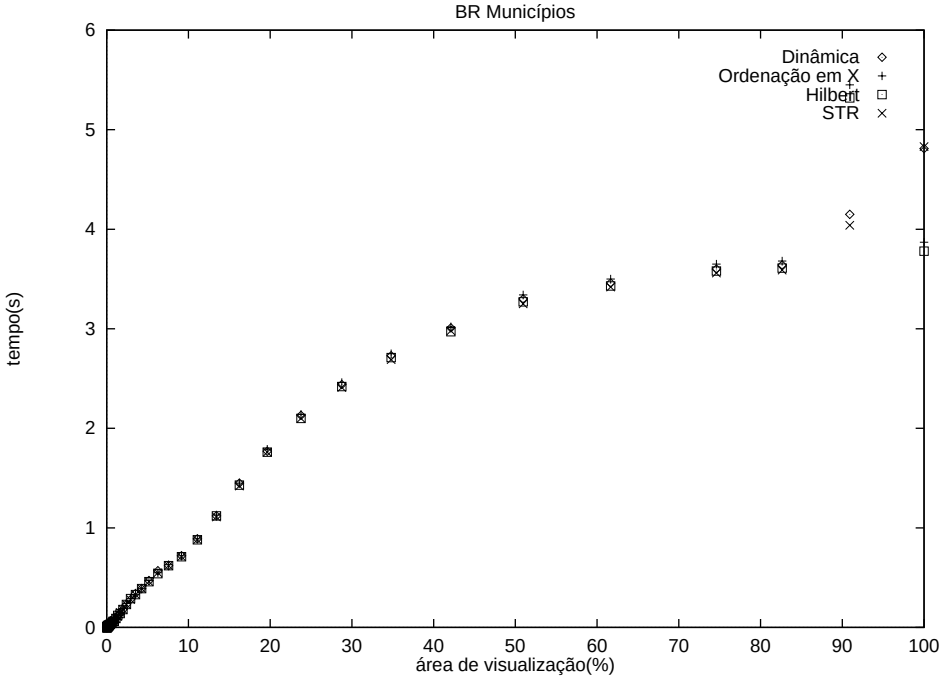


Gráfico 5.7: BR Municípios - variação da versão da R-tree (tempo x área de visualização)

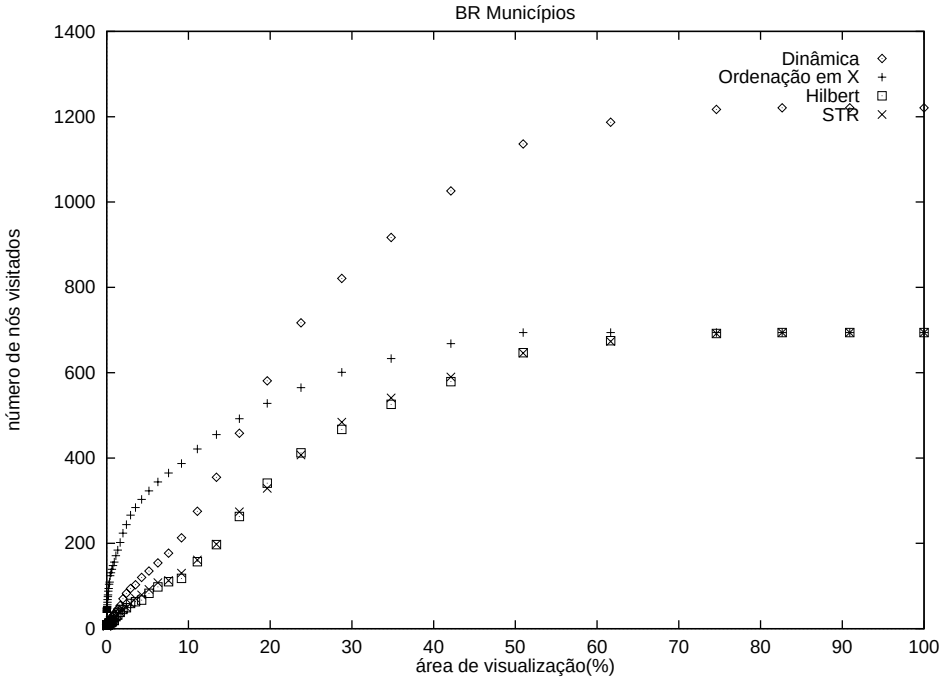


Gráfico 5.8: BR Municípios - variação da versão da R-tree (nós visitados x área de visualização)

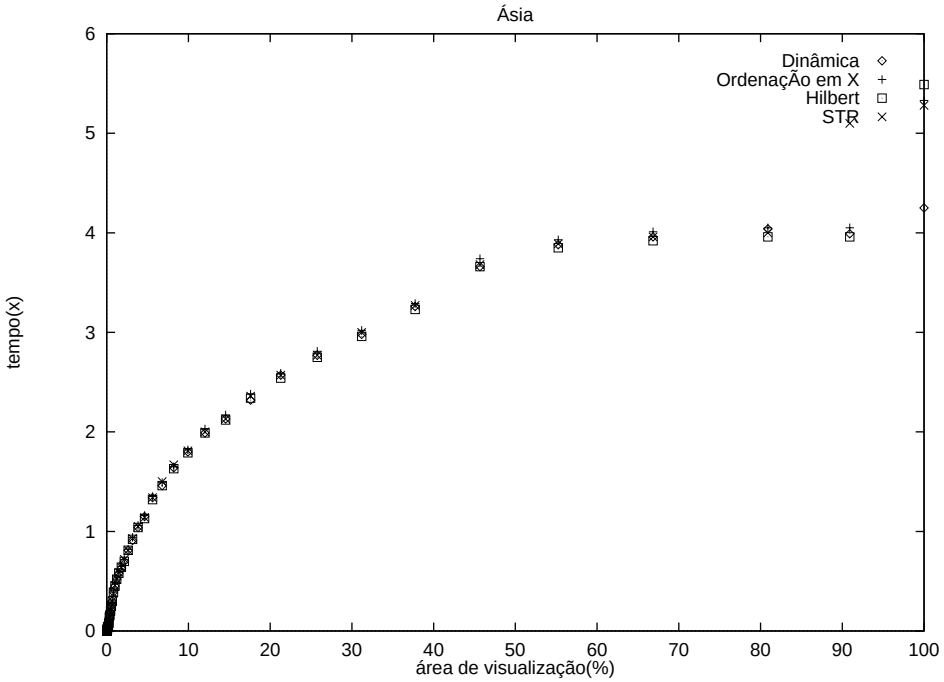


Gráfico 5.9: Ásia - variação da versão da R-tree (tempo x área de visualização)

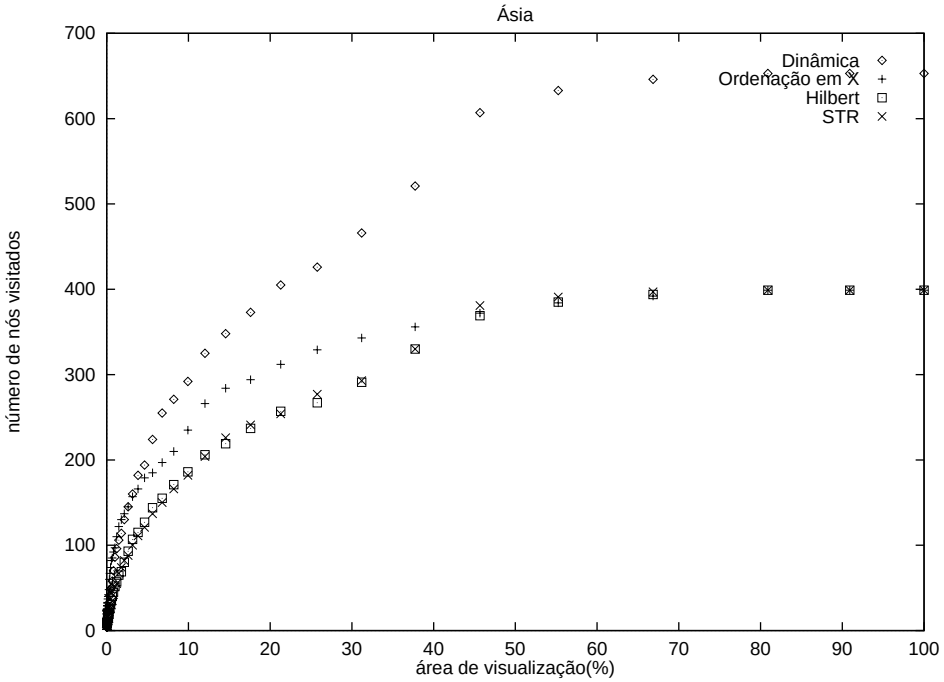


Gráfico 5.10: Ásia - variação da versão da R-tree (nós visitados x área de visualização)

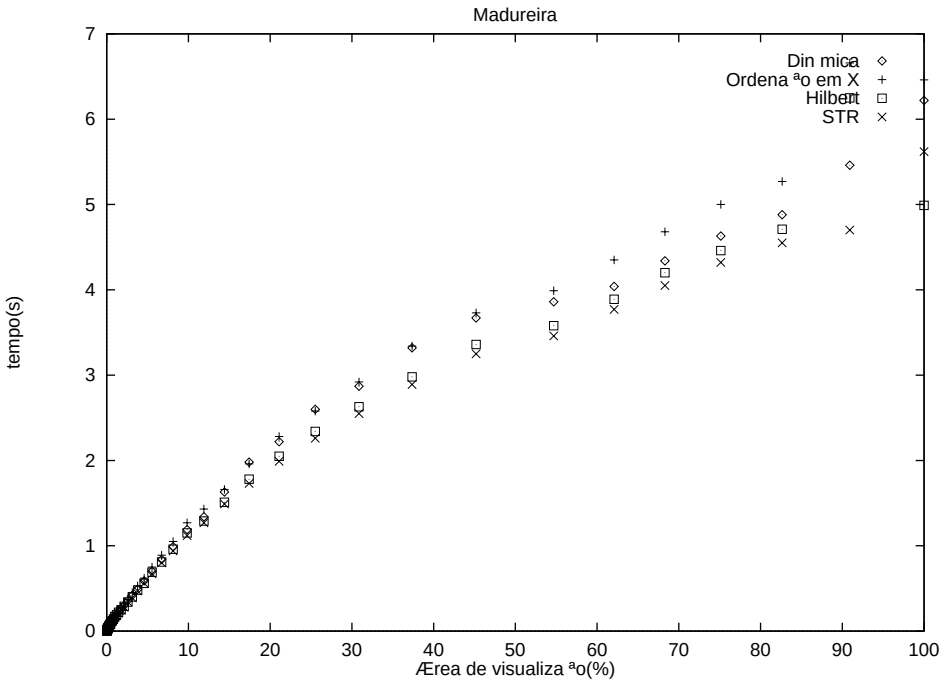


Gráfico 5.11: Madureira - variação da versão da R-tree (tempo x área de visualização)

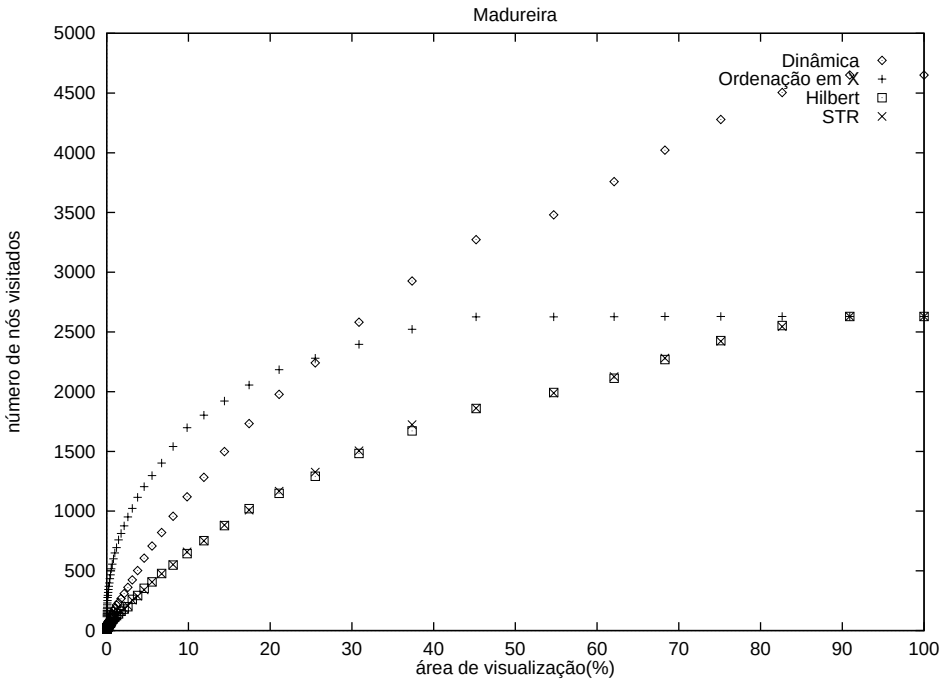


Gráfico 5.12: Madureira - variação da versão da R-tree (nós visitados x área de visualização)

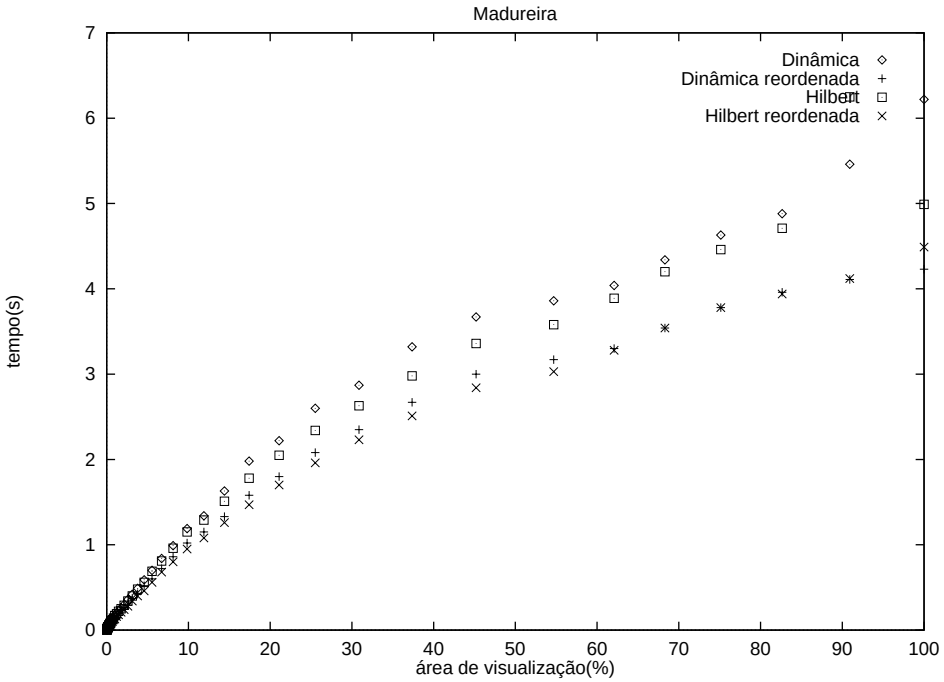


Gráfico 5.13: Madureira - reordenação da estrutura

5.3 Multiresolução

Os experimentos apresentados nesta seção visam verificar o desempenho dos mecanismos de multiresolução adotados pelo Drops, tanto no nível da R-tree quanto no nível dos objetos. Para isso o processo de visualização foi repetido várias vezes variando-se a resolução de saída dos dados. Foi calculado o tempo gasto por cada interação e então foram elaborados os gráficos resolução x tempo para a apresentação dos resultados.

A janela de exibição utilizada nos testes foi de 700x500 *pixels*. O valor da resolução de visualização foi definido no espaço de coordenadas dos dados, pois assim o resultado obtido é independente das dimensões da janela de visualização. Porém, para facilitar a avaliação da perda visual nos gráficos, está indicado também o que as resoluções representam em *pixels* na janela 700x500. É importante observarmos o comportamento dos resultados quando admitimos erros de 1, 2 ou 3 *pixels*, pois para estes valores praticamente não é possível perceber diferenças na visualização dos dados. O teste com o Drops foi feito duas vezes, uma com "buferização" do OPS e a outra sem. Como o teste corre diversas vezes, o *buffer* passa a operar a partir da segunda rodada.

Os experimentos foram divididos em três tipos. No primeiro tipo foi testada a multiresolução realizada somente pela R-tree. Para isto, o dado foi construído sem utilizar nem V-trees nem QuadTrees. Nos outros experimentos testamos a multiresolução ao nível dos objetos, desligando a multiresolução da R-tree e armazenando os objetos com multiresolução geométrica, ou seja, V-trees para polígonos e QuadTrees para imagens.

A Tabela 5.3 mostra os tamanhos dos arquivos de dados nos formatos que foram utilizados nos testes. Os arquivos CGM são bem menores que os Drops pois o Drops armazena também todas as estruturas de indexação dos objetos. Devemos destacar ainda que a precisão utilizada no CGM é somente de 16 *bits*, enquanto que o Drops utiliza 32 *bits* para o armazenamento das coordenados dos objetos.

Dado	**Tamanho (Kb)** *formato CGM*	**Tamanho (Kb)** *s/ V-trees*	**Tamanho (Kb)** *c/ V-trees*
BR Indígena	829	1436	2092
BR Municípios	3025	6480	11148
Ásia	4174	8520	12968
Madureira	2858	7220	10472

Tabela 5.3: Tamanhos dos arquivos de dados no formato CGM e no formato Drops sem V-trees e com V-trees

5.3.1 Multiresolução com a R-tree

Nos Gráficos 5.14, 5.15 e 5.16 estão os resultados dos experimentos de Multiresolução com a R-tree. Os testes não foram realizados com o mapa BR Indígena pois, devido à sua variedade de atributos, a construção da R-tree com multiresolução é bem mais complicada. Pelos resultados pode-se observar que todos os três mapas apresentaram grandes ganhos de desempenho principalmente nas primeiras variações de resolução (de 0 a 3 *pixels*): depois disto o ganho passa a ser pequeno. O dado de Madureira foi o que apresentou o melhor ganho com este tipo de multiresolução.

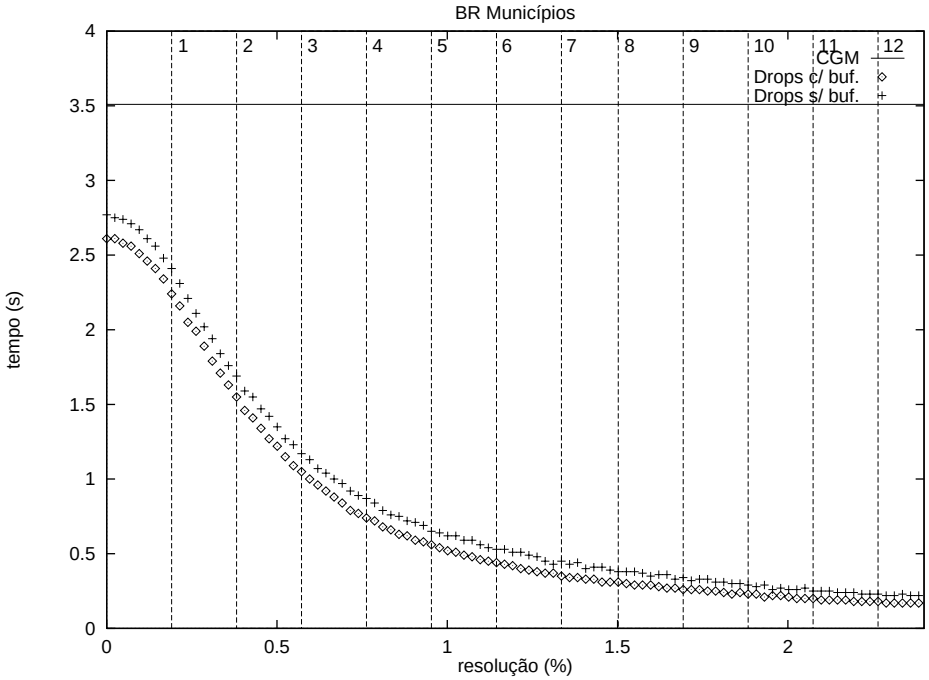


Gráfico 5.14: BR Municípios - multiresolução na R-tree

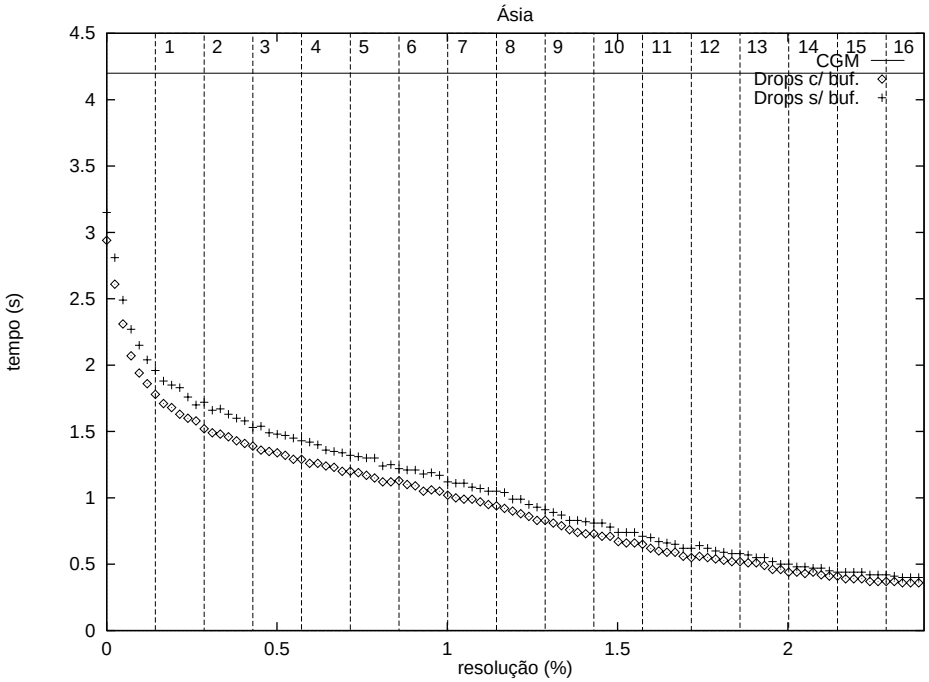


Gráfico 5.15: Ásia - multiresolução na R-tree

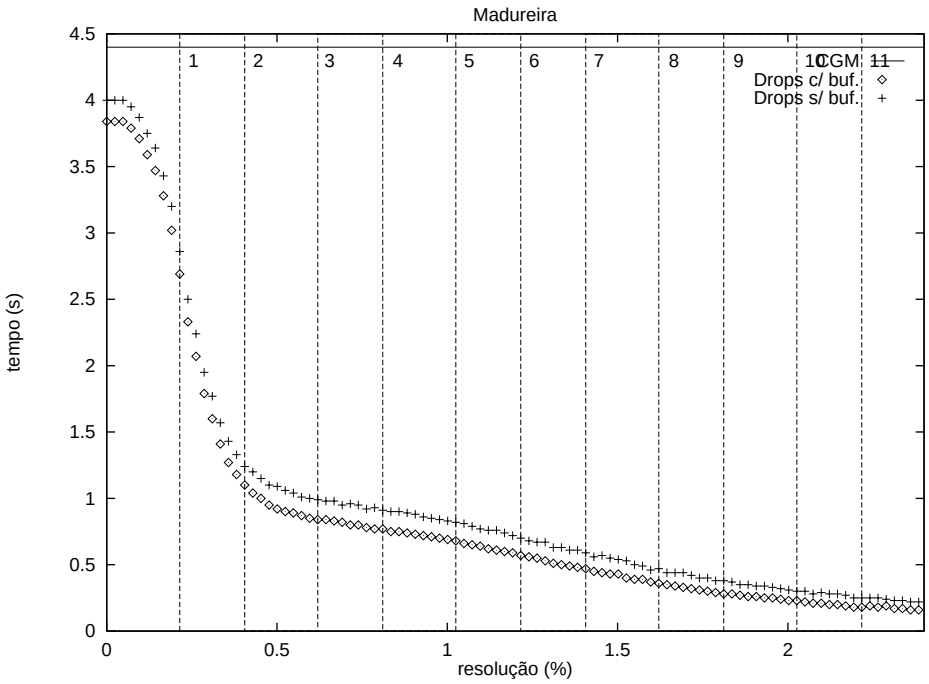


Gráfico 5.16: Madureira - multiresolução na R-tree

5.3.2 Multiresolução com as V-trees

Os resultados dos testes de multiresolução somente com as V-trees estão nos Gráficos 5.17, 5.18, 5.19 e 5.20. Neste caso somente os dados de BR Municípios e da Ásia tiveram ganhos importantes de desempenho. O mapa da Ásia foi o que obteve melhor resultado, enquanto que o mapa BR Indígena teve uma pequena melhora de desempenho. Como era previsto, no mapa de Madureira a multiresolução por V-trees praticamente não fez diferença uma vez que seus objetos são em sua maioria retângulos.

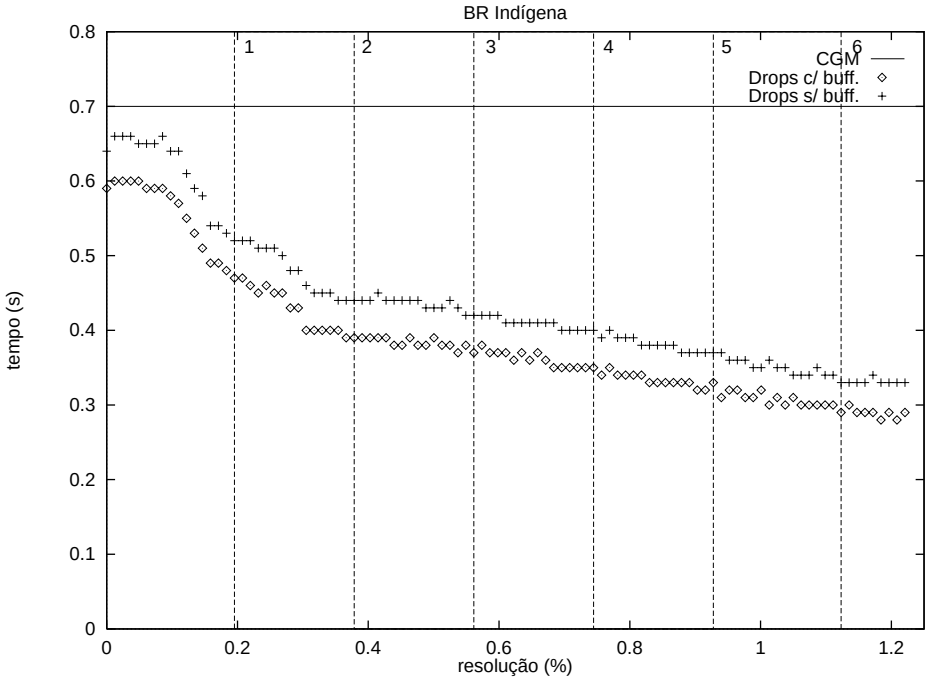


Gráfico 5.17: BR Indígena - multiresolução na V-tree

5.3.3 Multiresolução com a R-tree e as V-trees

Os Gráficos 5.21, 5.22 e 5.23 mostram os resultados dos experimentos em que foram utilizadas as duas multiresoluções juntas. Para os dados de BR Municípios e da Ásia o melhor resultado foi obtido com a utilização das duas multiresoluções em conjunto. Já para o dado de Madureira o terceiro resultado apresentou uma piora de desempenho em relação à utilização da multiresolução somente com R-trees. Este resultado era

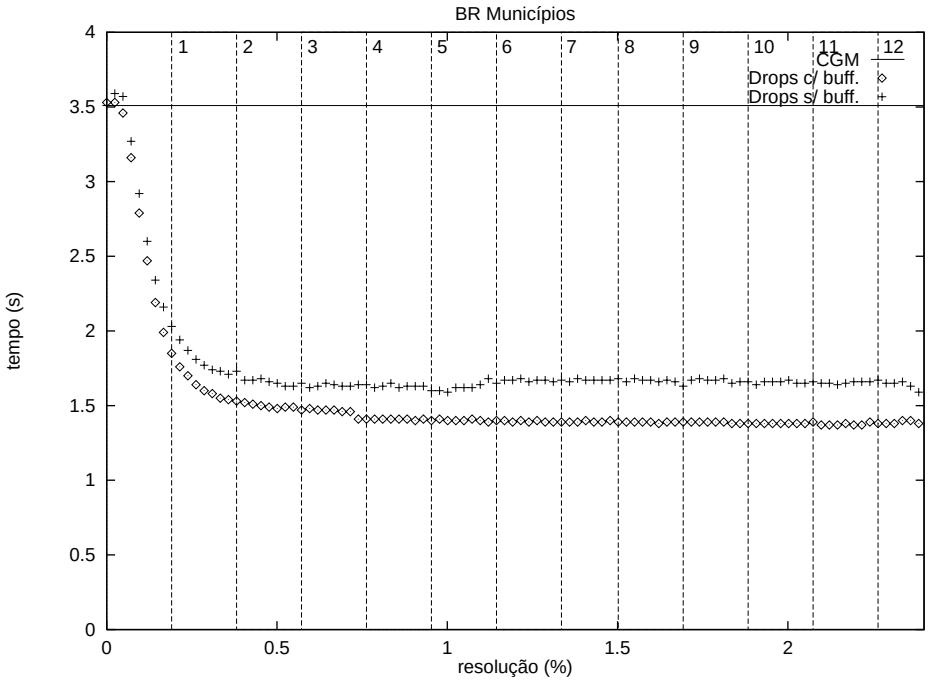


Gráfico 5.18: BR Municípios - multiresolução na V-tree

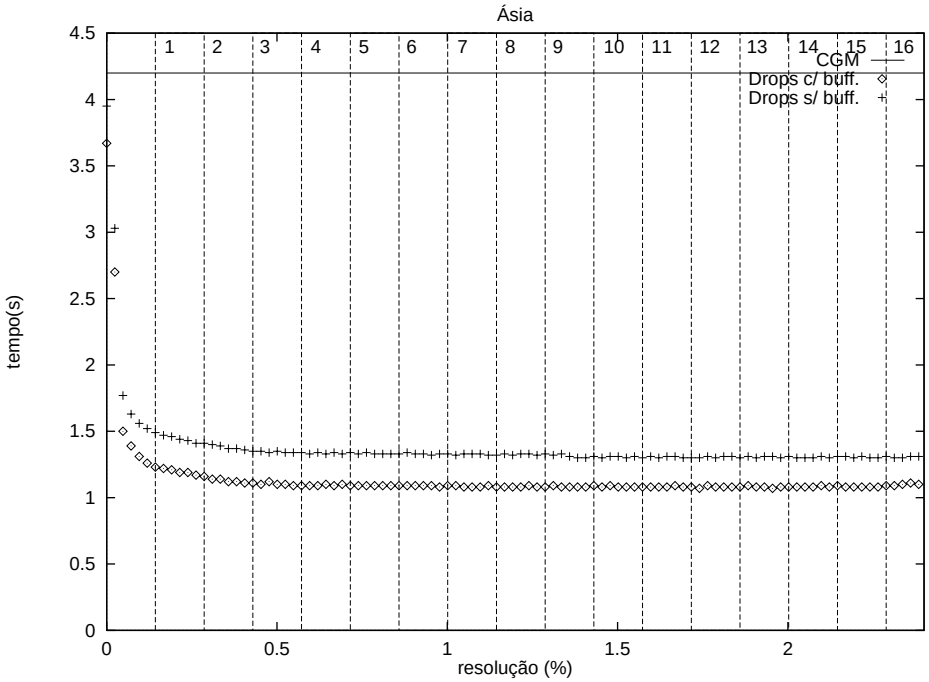


Gráfico 5.19: Ásia - multiresolução na V-tree

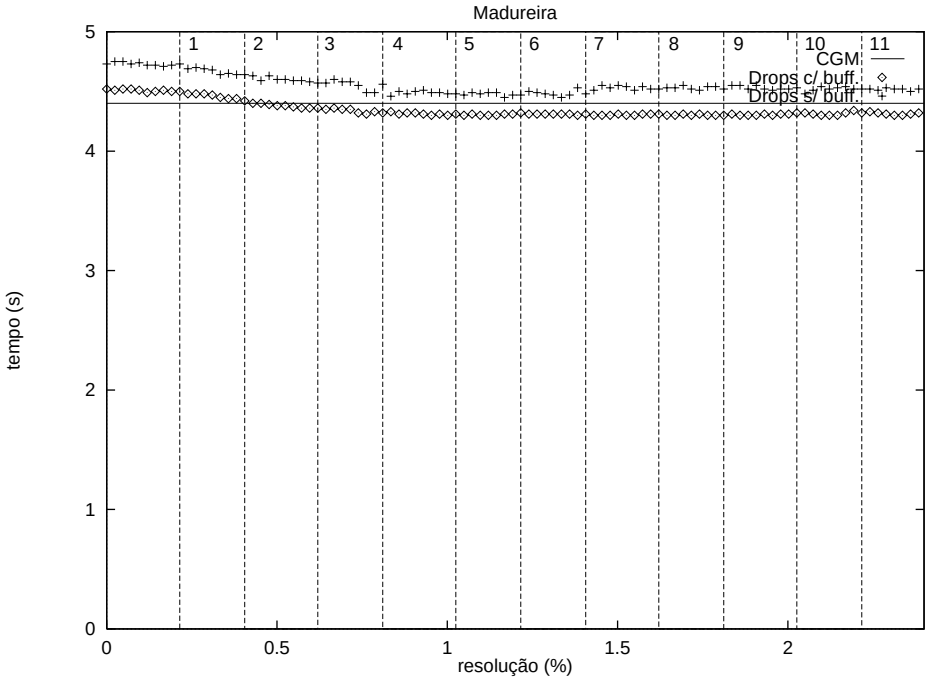


Gráfico 5.20: Madureira - multiresolução na V-tree

esperado pois, além do fato de que a multiresolução das V-trees não serve para este tipo de dado, elas dobram o tamanho do arquivo de dados.

5.3.4 *Range Queries* com Multiresolução

Os Gráficos 5.24, 5.25, 5.26 e 5.27 mostram o tempo para o uso acoplado de *zoom* e multiresolução com base na R-treee [Gut84] e nas V-Trees [MCD94].

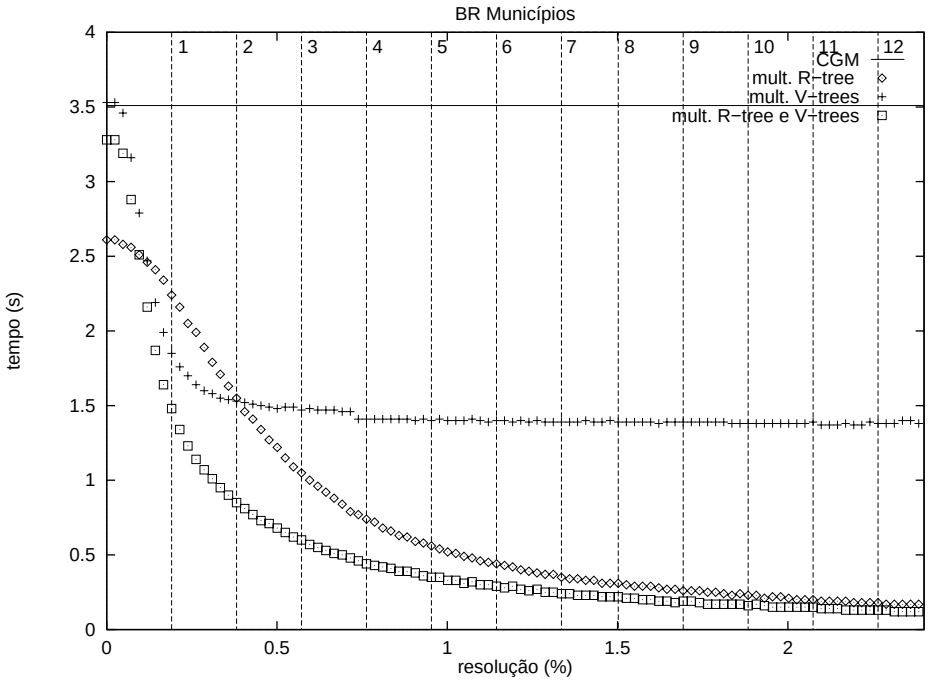


Gráfico 5.21: BR Municípios - multiresolução na R-tree e V-tree

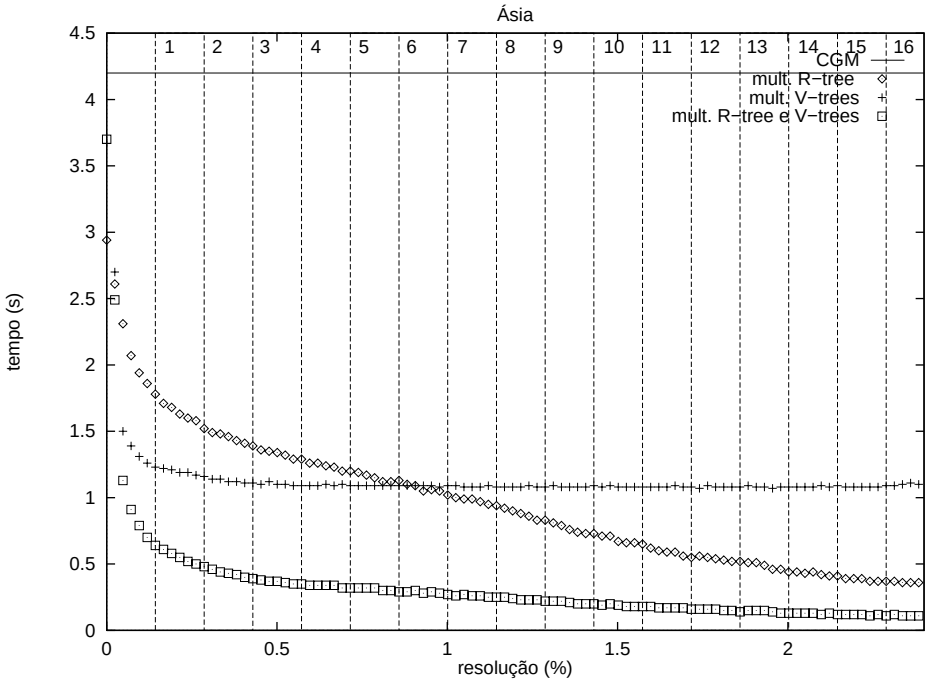


Gráfico 5.22: Ásia - multiresolução na R-tree e V-tree

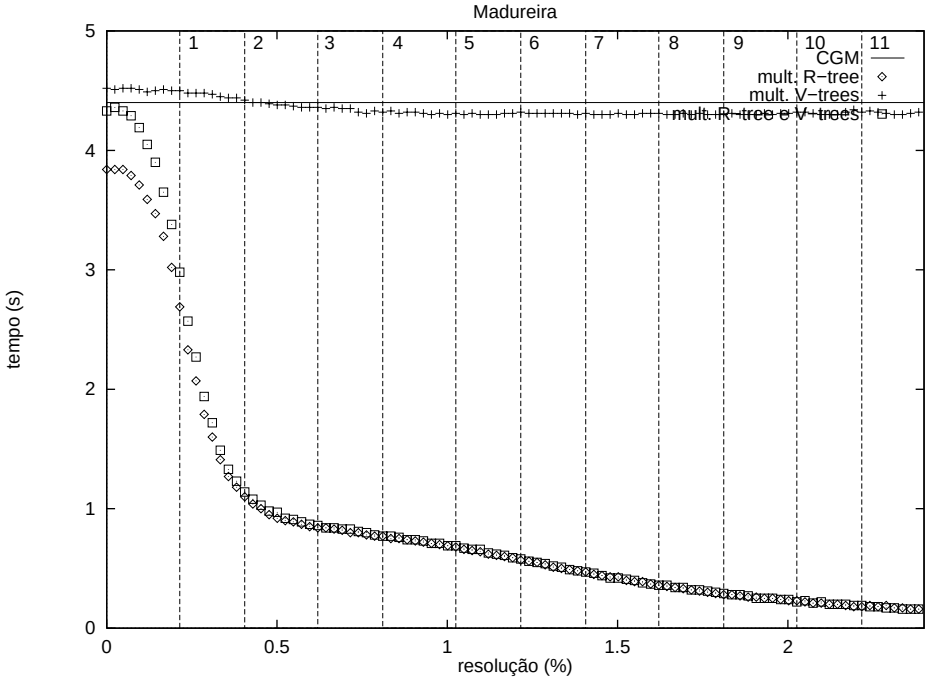


Gráfico 5.23: Madureira - multiresolução na R-tree e V-tree

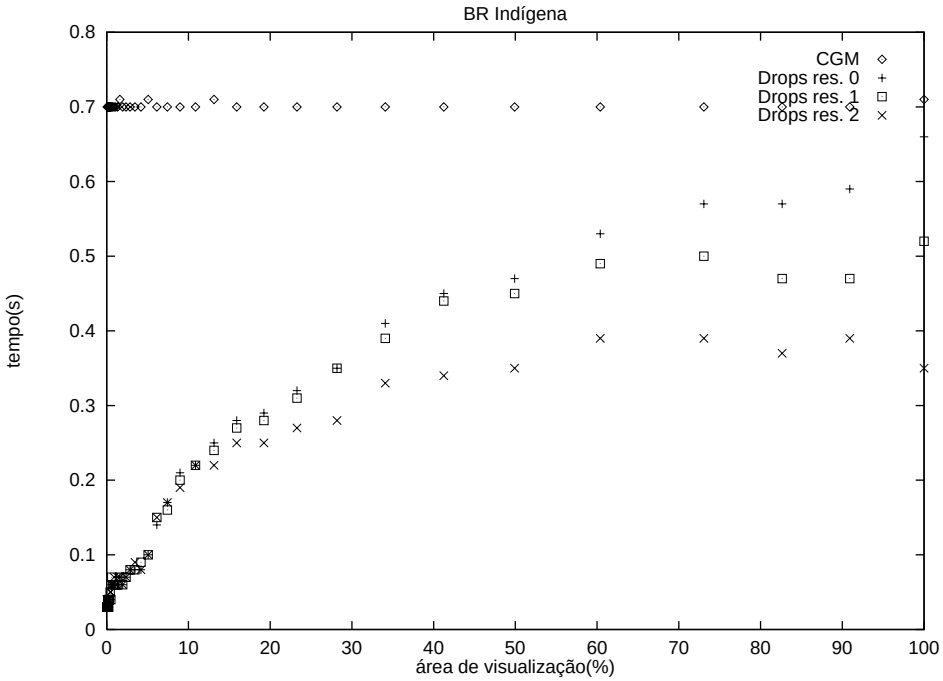


Gráfico 5.24: BR Indígena - *zoom* com multiresolução

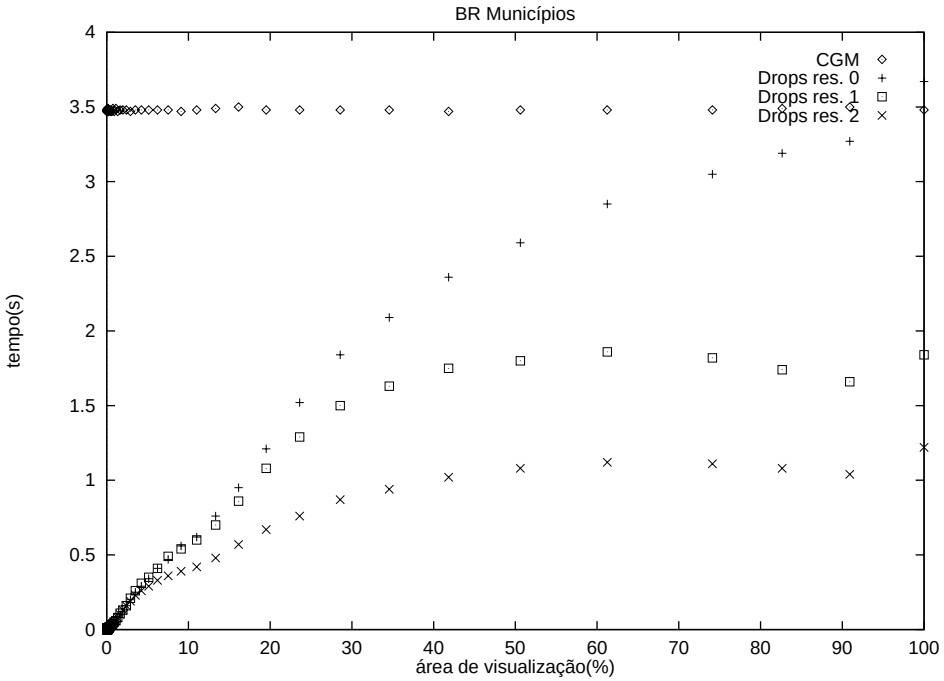


Gráfico 5.25: BR Municípios - *zoom* com multiresolução

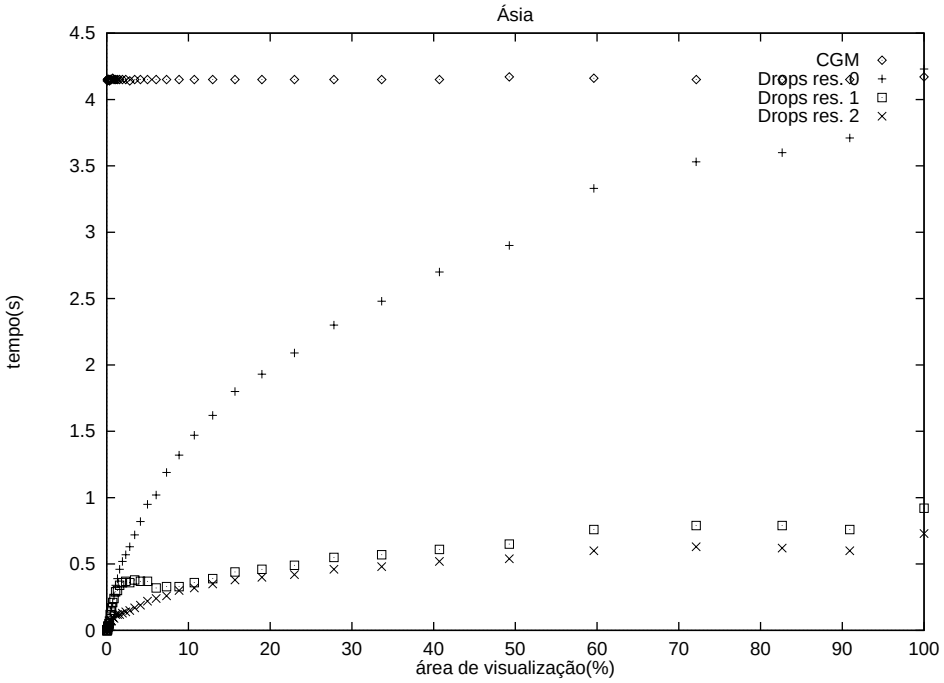


Gráfico 5.26: Ásia - *zoom* com multiresolução

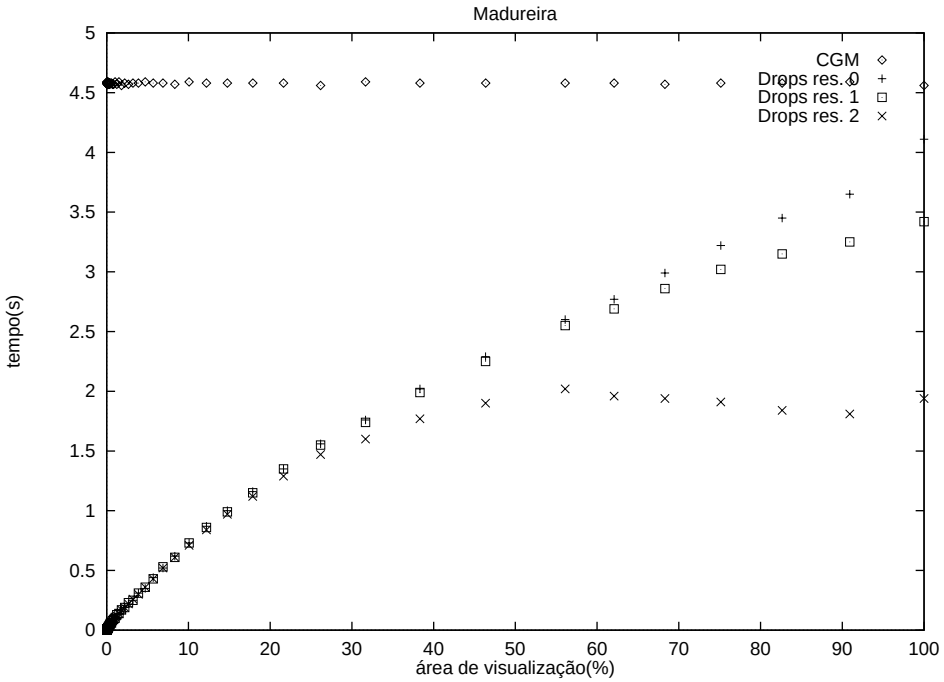


Gráfico 5.27: Madureira - *zoom* com multiresolução

Capítulo 6

Conclusões e Sugestões para o Futuro

Procuramos avaliar a idéia de trabalhar as R-trees e suas variantes para a indexação de um conjunto de dados. Para isto implementamos o sistema Drops sobre o OPS [Med97].

A partir dos experimentos numéricos podemos traçar algumas conclusões. A primeira é que, no equipamento de teste, as medidas de nós visitados não refletiram diretamente no tempo de processamento. Uma explicação possível pode dizer respeito ao sistema de discos rápidos (SCSI com *cache*).

A R-tree de no mínimo 20 e no máximo 50 elementos por nó foi a que apresentou o melhor resultado dentre as estudadas.

Na comparação com o arquivo CGM, em geral todas as versões da R-tree apresentaram melhor desempenho, principalmente quando a área de interesse é pequena.

A reordenação dos dados para preservar a fidelidade das sobreposições não é uma tarefa que degrade significativamente o tempo de acesso.

As versões clássica e ordenada em X da R-tree apresentam um desempenho pior que as versões Hilbert e STR.

O uso da R-tree mostrou-se bastante eficiente para a tarefa de multiresolução. Mesmo para valores pequenos de tolerância a árvore reduziu bastante o tempo de acesso. Para valores de tolerância maiores os novos ganhos não foram tão significativos. Devemos notar que a multiresolução relevante é a que apresenta pequena tolerância onde os erros perceptuais também são reduzidos (ou inexistentes).

As V-trees apresentaram um comportamento semelhante ao das R-trees na tarefa de apoiar a multiresolução dos dados que se baseiam em poligonais grandes. Nos dados compostos de muitos objetos simples (retângulos) o efeito foi negativo.

O efeito combinado da R-tree com as V-trees apresentou um ganho significativo quando os dados se baseavam em poligonais de muitos pontos.

O uso das R-trees compactas [RL85, KF93, LLE97] reduz significativamente o número de nós visitados.

Por fim pode-se concluir que multiresolução das V-trees traz benefícios para dados com objetos complexos, enquanto que a multiresolução das R-trees é vantajosa para dados compostos por grandes quantidades de objetos pequenos.

6.1 Sugestões para Trabalhos Futuros

A continuação natural deste trabalho é o aprofundamento do estudo de multiresolução para imagens baseada na QuadTree. Esta facilidade foi implementada no Drops mas ainda precisa de mais desenvolvimento e testes.

A multiresolução de objetos compostos de objetos primitivos tipo regiões, marcas e textos coloridos também precisa de um tratamento para determinar automaticamente a cor do retângulo equivalente.

A estratégia para suporte de camadas e de *shapes* também precisa ser investigada mais profundamente. As camadas foram implementadas da forma simples e as *shapes* não foram consideradas.

Também deve-se investigar mais profundamente a razão da pouca melhora em tempo quando o número de nós visitados é reduzido significativamente.

Finalmente deve-se estudar o suporte das idéias do Drops para outras tarefas comuns na interação com o usuário como seleção (*pick*).

Apêndice A

Formato BIN

O Formato BIN é um formato binário extremamente simples que armazena um conjunto de linhas poligonais no espaço bidimensional. Também conhecido por LIN, ele não suporta qualquer tipo de atributo (descrição de aparência) nem outro tipo de elemento geométrico.

Para cada linha poligonal é armazenado primeiro seu número de vértices, seguido pelos vértices. O número de vértices é armazenado no formato inteiro de 32 *bits*. Cada vértice é constituído por dois *floats* de 32 *bits* correspondendo a suas coordenadas x e y. As linhas poligonais são armazenadas seqüencialmente no arquivo.

Referências Bibliográficas

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

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, e Bernhard Seeger. Multi-step processing of spatial joins. Em *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 197–208, maio de 1994.

[BM72] R. Bayer e E. M. McCreight. Organization and maintence of large ordered indices. *Acta Informatica*, 1(3):173–189, 1972.

[Câm95] Gilberto Câmara. *Modelos, Linguagens e Arquiteturas para Banco de Dados Geográficos*. Tese de Doutorado, Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, 1995.

[CIA] Cia world data bank. ftp://setftp.stanford.edu/pub/World_Map/.

[Com79] D. Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–131, junho de 1979.

[FB74] R. A. Finkel e J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[FNPS79] R. Fagin, J. Nievergelt, N. Pippenger, e R. Strong. Extendible hashing: A fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[FR89] Christos Faloutsos e S. Roseman. Fractals for secondary key retrieval. Em *Proceedings of the ACM Conference on Principles of Database Systems*, 1989.

[Gar82] Irene Gargantini. An effective way to represent QuadTrees. *Communications of the ACM*, 25:905–910, 1982.

[GG98] Volker Gaede e Oliver Gúther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, junho de 1998.

[Gre89] Diane Greene. An implementation and performance analysis of spatial data acess methods. Em *Proceedings of the Fifth IEEE International Conference on Data Engineering*, pp. 606–615, 1989.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. Em *Proceedings of the ACM SIGMOD Conference on Data Engineering*, pp. 47–56, 1984.

[GV97] Jonas Gomes e Luiz Velho. *Image Processing for Computer Graphics*. Springer-Verlang, New York, NY, 1997.

[Hen93] Mumford A.M. Henderson, L.R. *The CGM Handbook*. Academic Press Inc., 1993.

[HSW90] A. Hutflesz, H. Six, e P. Widmayer. The r-file: An efficient access structure for proximity queries. Em *Proceedings of the Sixth IEEE International Conference on Data Engeneering*, pp. 372–379, 1990.

[IBG] Malha municipal digital do Brasil - situação em 1991 e 1994. Base de dados em CD.

[ISO92] ISO/IEC 8632:1992. *Information Technology - Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information*, 1992. Part 1 - Functional Specification (ISO8632-1), Part 2 - Character Encoding (ISO8632-2), Part 3 - Binary Encoding (ISO8632-3), Part 4 - Clear Text Encoding (ISO8632-4).

[KF93] I. Kamel e C. Faloutsos. On packing R-trees. Em *Proceedings of the 2nd International Conference on Information and Knowledges Management (CIKM-93)*, pp. 490–499, novembro de 1993.

[KF94] Ibrahim Kamel e Christos Faloutsis. Hilbert R-tree: An improved R-tree using fractals. Em *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 500–509, 1994.

[LL96] Scott T. Leutenegger e Mario A. Lopez. The effect of buffering on the performance of R-Trees. Relatório técnico, Mathematics and Computer Science Department, University of Denver, Denver, CO 80208-0189, 1996.

[LLE97] S. T. Leutenegger, M. A. Lopez, e J. Edgington. STR: A simple and efficient algorithm for R-tree packing. Em *Proceedings of the Thirteenth International Conference on Data Engeneering*, pp. 497–506, 1997.

[MCD94] Maurício Riguette Mediano, Marco Antônio Casanova, e Marcelo Dreux. V-tree - a storage method for long vector data. Em *Proceedings of the 20th VLDB Conference*, 1994.

[Med97] Maurício Riguette Mediano. OPS: Um subsitema extensível e configurável de armazenamento de objetos persistentes. Relatório Técnico MCC37/97, Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, 1997.

[NHS84] J. Nievergelt, H. Hinterberguer, e K. C. Sevick. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[Ore86] J.A. Orenstein. Spatial query processing in an object-oriented database system. Em *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 326–336, maio de 1986.

[PS85] F. Preparata e M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlang, 1985.

[RL85] N. Roussopoulos e D. Leifker. Direct spatial search on pictorial databases using packed r-trees. Em *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, pp. 17–31, maio de 1985.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, junho de 1984.

[Sam90] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Massachutsetts, 1990.

[San99] Newton Cunha Sanches. Temporary data structures in spatial query processing. Dissertação de Mestrado, Departamento de Informática PUC-Rio, março de 1999.

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

[SRF87] Timos Sellis, Nick Roussopoulos, e Christos Faloutsos. The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects. Em *Proceedings of the 13th VLDB Conference*, pp. 507–518, setembro de 1987.

[SW88] H. Six e P. Widmayer. Spatial searching in geometric databases. Em *Proceedings of the Fourth IEEE International Conference on Data Engeneering*, pp. 496–503, 1988.

[Tec98] Tecgraf - PUC-Rio. *CD - Canvas Draw - Uma Biblioteca Gráfica 2D - Versão 3.6*, abril de 1998.

[Tec99] Tecgraf - PUC-Rio. *IUP - Sistema Portátil de Interface com o Usuário - Versão 1.8*, janeiro de 1999.

[USG] United states geographic survey (usgs). http://www-atlas.usgs.gov.

[Wor95] Michael F. Workboys. *GIS: A Computing Perspective*. Taylor & Francis Ltd., 4 John Street, London, WC1N 2ET, 1995.

Visualização Eficiente de Objetos Gráficos

Dissertação de Mestrado apresentada por **Paula Frederick** em 01 de julho de 1999 ao Departamento de Informática da PUC-Rio e aprovada pela comissão julgadora formada por:

Prof. Marcelo Gattass
Orientador
Departamento de Informática - PUC-Rio

Prof. Marco Antonio Casanova
Departamento de Informática - PUC-Rio

Prof. Gilberto Câmara Neto
INPE

Prof. Luiz Fernando Martha
Departamento de Engenharia Civil - PUC-Rio

Prof. José Lucas Mourão Rangel Netto
Departamento de Informática - PUC-Rio

Visto e permitida a impressão.
Rio de Janeiro,

Coordenador dos Programas de Pós-Graduação
e Pesquisa do Centro Técnico Científico