UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE GEOCIÊNCIAS - IG
PÓS-GRADUAÇÃO EM GEOCIÊNCIAS APLICADAS E GEODINÂMICA

Desenvolvimento de um método para integrar um segmentador de grandes imagens no banco de dados PostgreSQL

Área de Concentração: Geoprocessamento e Análise Ambiental

TESE Nº 40

SIMONE DUTRA MARTINS GUARDA

Orientador: Prof. Dr. Edson Eyji Sano

Co-orientador: Prof. Dr. Edilson de Souza Bias

Brasília - DF
Junho de 2018

UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE GEOCIÊNCIAS
PÓS-GRADUAÇÃO EM GEOCIÊNCIAS APLICADAS E GEODINÂMICA

Desenvolvimento de um método para integrar um segmentador de grandes imagens no banco de dados PostgreSQL

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Geociências Aplicadas e Geodinâmica da Universidade de Brasília, como requisito parcial para obtenção do título de Doutor.
Área de concentração: Geoprocessamento Aplicado à Análise Ambiental

SIMONE DUTRA MARTINS GUARDA

Orientador: Prof. Dr. Edson Eyji Sano

Co-orientador: Prof. Dr. Edilson de Souza Bias

Brasília - DF
Junho de 2018

BANCA EXAMINADORA

Prof. Dr. Edson Eyji Sano
Embrapa Cerrados - Orientador

Prof. Dr. Henrique Llacer Roig
Universidade de Brasília – Examinador Interno

Prof. Dr. Osmar Abílio de Carvalho Junior
Universidade de Brasília – Examinador Interno

Prof. Dr. Patrick Nigri Happ
Pontifícia Universidade Católica do Rio de Janeiro

Brasília – DF
Junho de 2014

FICHA CATALOGRÁFICA

Guarda, Simone D. M.

Desenvolvimento de um método para integrar um segmentador de grandes imagens no banco de dados PostgreSQL / Simone Dutra Martins Guarda; orientação de Edson Eyji Sano. Co-orientação de Edilson de Souza Bias, 2018.

147 p.

Tese de Doutorado (D) – Universidade de Brasília / Instituto de Geociências, 2018.

1. Segmentação OBIA. 2. SGBD-OR. 3. Classificação OBIA; 4. I Título.

REFERÊNCIA BIBLIOGRÁFICA

Guarda, Simone Dutra Martins. Desenvolvimento de um método para integrar um segmentador de grandes imagens no banco de dados PostgreSQL. Brasília, Instituto de Geociências, Universidade de Brasília, 2018, 147 p. Tese de Doutorado.

CESSÃO DE DIREITOS

NOME DO AUTOR: Simone Dutra Martins Guarda

TÍTULO DA TESE DE DOUTORADO: Desenvolvimento de um método para integrar um segmentador de grandes imagens no banco de dados PostgreSQL.

GRAU: Doutor ANO: 2018

_____

Simone Dutra Martins Guarda
simonedmg@ifto.edu.br

“Procuro semear otimismo e plantar
sementes de paz e justiça.
Digo o que penso, com esperança.
Penso no que faço, com fé.
Faço o que devo fazer, com amor.
Eu me esforço para ser cada dia melhor,
Pois bondade também se aprende! ”

Cora Coralina.

AGRADECIMENTOS

“À VIDA, em Honra à grande força onisciente e onipresente
do Universo manifestada na palavra DEUS.”

Ao Papai e à Mamãe, eterna gratidão pela minha VIDA, AMOR, SUPERAÇÃO, e por TUDO!

Ao meu marido amore Roberto Mauro, pelos nossos filhos e suporte contínuo.

À nossa filha Sara, ao nosso bebê que está com Deus, e ao nosso filho Vitor Roberto, meus presentes da VIDA e motivo desta conquista.

Aos meus professores do Programa de Pós-Graduação em Geociências Aplicadas da UnB, em especial, ao Prof. Dr. Edson E. Sano e ao Prof. Dr. Edilson S. Bias, ambos pelo apoio incondicional e paciência sempre a gratidão; e ao ao Prof. Dr. José Eloi G. Campos, Prof. Dr. Gustavo M. M. Baptista, aoProf. Dr. Henrique L. Roig, Prof. Dr. Paulo R. Meneses e ao Prof. Dr. Alexandre de Amorim T.

Ao IFTO pelo apoio institucional e aos meus amigos (as) do *Campus* Palmas pelo constante incentivo para a obtenção deste novo nível de graduação. Ao colegiado da Coordenação de Informática meu muito obrigada e aos amigos que sustetaram e ajudaram de variadas formas a conclusão do meu doutorado.

Ao INPE, principalmente a todos da Divisão de Processamento de Imagens, pela construção da plataforma TerraLib; e, em especial, à colaboração primordial do Emiliano Castejon, Gilberto Ribeiro de Queiroz e Eric Silva Abreu.

À PUC-RJ, nas pessoas dos responsáveis pelo desenvolvimento do InterIMAGE, representados pelo Prof. Dr. Gilson Alexandre Ostwald Pedro da Costa e Prof. Dr. Patrick Nigri Happ.

À minha Família, em especial de Goiânia-GO e Palmas-TO, pela ancoragem, estímulo, e apoio nesta caminhada. Aos antepassados (as) que

vieram antes. Às minhas irmãs Luciana e a Raquel com seus filhos Alberto e Ubiratan Gabriel; à Madrinha Idecília e prima Juliana. À minhas irmãs de coração Rozane, Márcia, Mariana, Iracélia, Sônia, Viviane, Maria Ivete. Aos amigos João e Dr. Antônio Carlos Cirilo. Às minhas amigas de longa data: Liliam, Simone Maciel, Nolan e Dalvirene pela motivação.

Aos sempre amigos (as), não declarados (as) explicitamente mas também importantes, com os quais venho dividindo aprendizados, viagens, trabalhos, alegrias, poucas tristezas, algumas dúvidas e certezas sobre o doutorado na UnB, carreira, ciência, academia; imensamente agradecida pela colaboração e pela torcida.

Aos colegas de caminhada na UnB que me apoiaram em vários momentos desta jornada, principalmente à Tamires, Débora, Juliane, Carol, Rodrigo Couto, Welber Ferreira, Denilson, Rodrigo Antunes e Rogério Baptista de Sousa.

Aos amigos (as) do meu grupo de tênis pelos momentos juntos em que aliviaram a pressão da tese a ser feita.

A todos (as) professores (as) que tiveram presença na minha formação técnica e acadêmica, ressaltando os da área de Geoprocessamento e Sensoriamento Remoto, representados pelo Prof. Dr. Ricardo R. Dias.

RESUMO

A abordagem de classificação orientada a objetos representa um novo paradigma no processamento de imagens de altas resoluções espaciais, espectrais e temporais, e a abordagem mais comum usada para construir objetos é a segmentação de imagens. Atualmente, o processamento de imagens está avançando na área de análise de imagens baseada em objetos (OBIA ou GEOBIA) que apresenta métodos capazes de explorar, além de atributos espectrais, outros elementos interpretativos como textura, forma ou contexto. Os usuários de recursos computacionais disponíveis exigem que as aplicações possam processar de forma eficaz estas imagens. As soluções buscam melhor desempenho de processamento computacional em aplicações com soluções sequenciais e distribuídas, mas poucas abordam o uso de sistemas gerenciadores de banco de dados. Esta tese propõe-se a explorar a abordagem de especificações de aplicações para integrar o SGBD PostgreSQL/PostGIS e o classificador OBIA do programa InterIMAGE Desktop para processamento de grandes imagens orbitais. O método apresentado é expansível no uso da biblioteca TerraLib 5, com implementação em C++. Os experimentos realizados com as representações matriciais (*raster*) indicaram a viabilidade das aplicações, e podem consolidar-se sob a forma de processos de armazenamento e processamento da segmentação no SGBD.

Palavras-chave: Classificação orientada a objetos. OBIA. Segmentação. Sistemas gerenciadores de banco de dados. InterIMAGE. TerraLib 5. PostgreSQL/PostGIS.

ABSTRACT

The object-oriented classification approach represents a new paradigm in image processing of high spatial, spectral and temporal resolutions, and the most common approach used to construct objects is image segmentation. Currently, image processing is advancing in the area of object-based image analysis (OBIA or GEOBIA) that presents methods capable of exploring, in addition to spectral attributes, other interpretive elements such as texture, form or context. Users of available computing resources require applications to be able to efficiently process these images. The solutions seek better computational processing performance in applications with sequential and distributed solutions, but few address the use of database management systems. This thesis proposes to explore the application specifications approach to integrate the PostgreSQL/PostGIS DBMS and the InterIMAGE Desktop program's OBIA classifier for processing large orbital images. The presented method is expandable in the use of TerraLib 5 library, with implementation in C ++. The experiments performed with the raster representations indicated the feasibility of the applications, and can be consolidated in the form of storage and processing processes of the segmentation in the DBMS.

Keywords: Object-oriented classification. OBIA. Segmentation. Database management systems. InterIMAGE. TerraLib 5. PostgreSQL/PostGIS.

LISTA DE SIGLAS E ABREVIATURAS

ADT	- *Abstract Data Ty*pe
ANSI	- *American National Standards Institute*
API	- *Application Programmers Interface*
ASP.NET	- *Active Server Pages NET*
BLOB	- *Binary Large Object*
CMAKE	- *Cross Plataform Make*
CRA	- Centro Regional da Amazônia
CPU	- *Central Processing Unit*
DML	- *Data Manipulation Language*
DN	- *Digital number*
DPI	- Divisão de Processamento de Imagem
EMBRAPA	- Empresa Brasileira de Pesquisa Agropecuária
ESRI	- *Environmental Systems Research Institute*
FNEA	- *Fractal Net Evolution Approach*
FUNCATE	- Fundação para a Ciência Espacial, Pesquisa Aplicada e Tecnologia
GB	- *Gigabyte*
GDAL	- *Geospatial Data Abstraction Library*
GEOBIA	- *Geographic Object Based Image Analysis*
GIS	- *Geographic Information System*
GIST	- *Generic Index Structure*
GNU	- *General Public License*
ICP	- InterIMAGE *Cloud Platform*
INPE	- Instituto Nacional de Pesquisas Espaciais
ISO	- *International Organization for Standardization*
KByte	- *Kilobyte*
LO	- *Large Objects*
MByte	- *Megabyte*
MIT	- *Massachusetts Institute of Technology*
OGC	- *Open Geospatial Consortium*
OID	- *Object Identifier*

OSGeo - *Open Source Geospatial Foundation*

PDI - Processamento Digital de Imagem

PUC-Rio - Pontifícia Universidade Católica do Rio de Janeiro

S - Segundos

SFSSQL - *Simple Features Specification for SQL - Structured Query Language*

SGDB - Sistemas de Gerenciamento de Banco de Dados

SGBDG - Sistema de Gerenciamento de Banco de Dados Geográficos

SGBDOR - Sistema de Gerenciamento de Banco de Dados Objeto-Relacional

SGBD-R - Sistema de Gerenciamento de Banco de Dados Relacional

SIG - Sistemas de Informações Geográficas

SPRING - Sistema de Processamento de Informações Georreferenciadas

SPT - *Segmentation Parameter Tuning*

SQL - *Structure Query Language*

SRC - Sistemas de Referência de Coordenadas

STL - *Standard Template Library*

TByte - *Terabyte*

TIN - *Triangulated Irregular Network*

UDF - *User-Defined Function*

VB - *Visual Basic*

VHR - *Very High Resolution*

VTK - *Visualization Toolkit*

XML - *Extensible Markup Language*

LISTAS DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

CAPÍTULO 1
INTRODUÇÃO

Os procedimentos tradicionais de classificações de imagens digitais orbitais foram desenvolvidos baseados no conceito do *pixel* como unidade primitiva de informações, ou seja, na abordagem *pixel* a *pixel*, voltada sobretudo para imagens de baixa e moderada resolução espacial. Os avanços tecnológicos que resultaram na melhoria das resoluções espacial e radiométrica do *pixel* evidenciaram as limitações dos métodos automáticos consagrados para a classificação baseada em *pixels*. Diversos autores têm adotado com êxito uma nova abordagem de processamento de imagens de alta resolução espacial em que a unidade primitiva é o objeto, composto de vários *pixels*. Esta técnica é denominada classificação orientada a objetos (BAATZ e SHÄPE, 2000; ZHONG *et al.*, 2005; BLASCHKE, 2010; BLASCHKE *et al*., 2014).

A classificação orientada a objetos almeja integrar os procedimentos de processamento e extração de informações similar ao raciocínio humano, tal como a forma de pensar e de caracterizar padrões de forma intuitiva ao interpretar uma imagem ou apreciar uma paisagem (BAATZ e SCHÄPE, 2000).

O cérebro humano reconhece a forma dos objetos e, em seguida, compara com padrões pré-definidos em nossas mentes. A técnica da segmentação multi-resolução realiza o agrupamento de *pixels* processando propriedades espectrais semelhantes, examinando o tamanho, a homogeneidade espectral, a homogeneidade espacial e a forma do objeto (CHUBEY *et al*., 2006).

Zhong *et al.* (2005) relataram o método de delineamento denominado de segmentação multi-escala, que é um processo que procura delinear os objetos contidos em uma imagem de alta resolução espacial aproximado ao raciocínio humano, e baseia-se no algoritmo multi-resolução proposto por Baatz e Schäpe (2000). A diferença principal se dá ao explorar as múltiplas escalas como se fosse uma segmentação hierárquica por meio dos polígonos.

A área de investigação científica denominada Análise de Imagens Baseada em Objetos Geográficos (GEOBIA - *Geographic Object Based Image Analysis*) apresenta métodos de análise de imagem capazes de explorar, além de atributos espectrais, outros elementos interpretativos como textura, forma ou contexto (HAY e CASTILLA, 2008; BLASCHKE, 2010). O advento de *softwares* comerciais de análise

baseada em objetos geográficos como eCognition® (*DEFINIENS IMAGING*, 2016) e *Feature Analyst*® (*VISUAL LEARNING SYSTEMS*, 2016) e de *softwares* de código aberto como o InterIMAGE (InterIMAGE, 2016) e Sistema de Processamento de Informações Georreferenciadas (Spring) (INPE, 2016), e as limitações dos diversos *softwares* com procedimentos baseados em *pixel* em disponibilizarem resultados de forma precisa para diversas aplicações, fizeram intensificar o interesse na análise de imagens baseada em objetos para dados de sensoriamento remoto (CHUBEY *et al*., 2006; BLASCHKE *et al*., 2014).

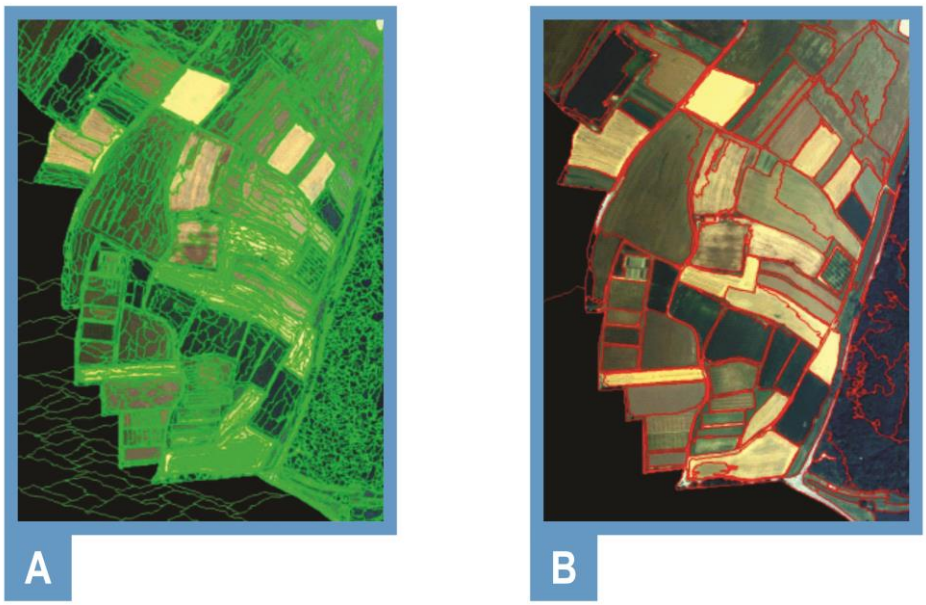
O *software* eCognition® é utilizado para análise de imagens orientadas a objetos e considera o alto grau de relações mútuas e ações em diferentes escalas, como informações de contexto, estrutura semântica e hierárquica (BLASCHKE *et al*., 2000; BENZ *et al*., 2004; TZOTSOS *et al*., 2006; MÖLLER *et al*., 2007; BLASCHKE *et al*., 2010; ALRASSI *et al*., 2016). Entretanto, um forte fator impeditivo para sua ampla utilização é o seu alto custo de aquisição.

Na classificação orientada ao objeto a análise é realizada sobre objetos ou segmentos na imagem e não apenas sobre *pixels*. Na imagem um objeto representa uma entidade que pode ser individualizada, tem atributos próprios e as mesmas propriedades da classe que lhe deu origem (*DEFINIENS IMAGING*, 2016).

Na segmentação de imagens no eCognition® a formação de segmentos é realizada de uma maneira que um padrão geral homogêneo é mantido. Níveis de segmentação mais finos ou grosseiros podem ser criados, respeitando-se a existência de seu nível mais grosseiro (Figura 1) (MANAKOS *et al*., 2000). No eCognition®, o algoritmo de segmentação não só registra o valor único do *pixel*, mas também a continuidade espacial do *pixel*. Os objetos formados possuem o valor da informação estatística dos *pixels* que eles consistem, além das características espaciais (textura e forma) e das informações sobre a topologia em uma tabela de atributos comuns (BAATZ e SCHÄPE, 2000).

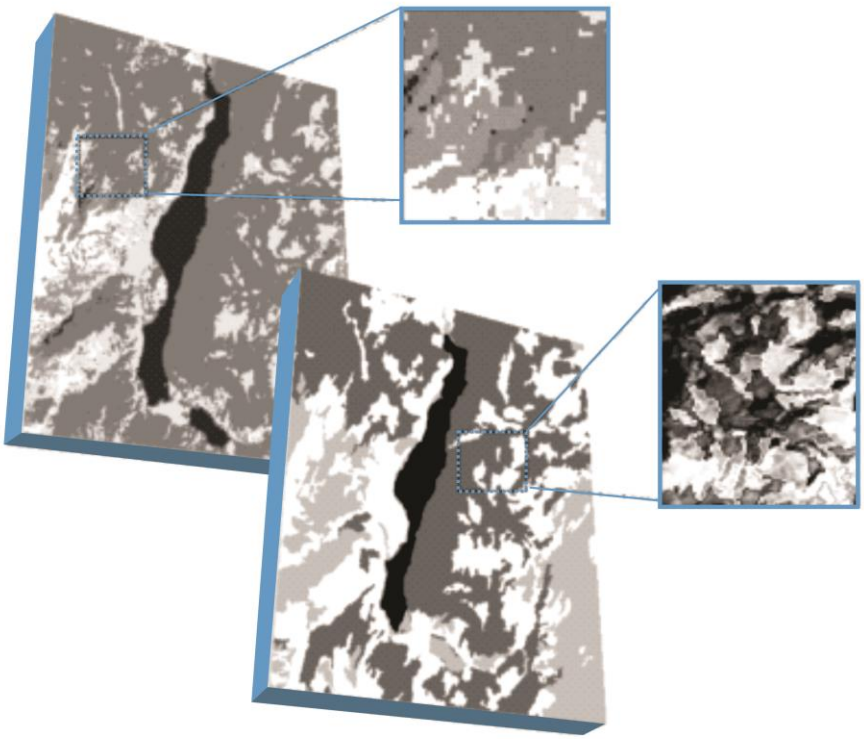
Diversos algoritmos de segmentação têm sido propostos para facilitar a classificação (COLEMAN, 1975; HARALICK e SHAPIRO, 1985). Porém, as soluções de boa qualidade frequentemente apresentam alto custo computacional (WASSENBERG *et al*., 2009). A Figura 2 ilustra o resultado de duas classificações com diferentes parâmetros de processamento.

Figura 1: Segmentação multi-resolução de objetos, fina (A - tamanho médio de objeto: 143 *pixels*) e grossa (B - tamanho médio de objeto: 4.065 *pixels*)



Fonte: Adaptado de Manakos *et al*. (2000).

Figura 2: Classificação “tradicional” baseada em *pixels* (topo) vs. classificação orientada a objetos baseada em segmentação (parte inferior)



Fonte: Adaptado de Blashke *et al*. (2000).

O sistema InterIMAGE é um *framework* de código aberto, de plataforma livre e gratuito, que baseado-se no conhecimento para a interpretação automática de dados de sensoriamento remoto (COSTA *et al*., 2010). O InterIMAGE se tornou uma alternativa interessante de substituição aos pacotes de *software* comerciais de interpretação automática de imagens, porque possui operadores para utilizar *Object-based Image Analysis* - OBIA em processos de classificação de dados de sensoriamento remoto. Além disso, os usuários podem adicionar no InterIMAGE, os seus próprios operadores ou modificar algumas funcionalidades do sistema. Costa *et al.* (2010) definiram o InterIMAGE como um:

> "*framework* multi-plataforma, contando atualmente com implementações para sistemas operacionais LINUX e Windows. O sistema é codificado em C++, utilizando também a estrutura de desenvolvimento de aplicações multiplataforma QT4 (Summerfield, 2010), a biblioteca de classes *Visualization Toolkit* (VTK) (Schroeder *et al*., 2006) e TerraLib (Camara, 2000), e biblioteca de funções desenvolvida no INPE."

O InterIMAGE pode ser usado como um framework de *software* para o desenvolvimento de pesquisas científicas nas áreas de análise de imagens baseadas em objetos. Entretanto, ressalta-se que versão do InterIMAGE 1.43 possui uma limitação para processar e classificar imagem superior a 9 Megapixels (3.000 x 3.000 *pixels*), pois envolvem também outras grandezas como tamanho da rede semântica e o número de objetos a serem trabalhados.

A nova plataforma InterCloud (InterIMAGE Cloud Platform - ICP), que é uma estrutura de código aberto distribuída para a interpretação automática de imagens, e é uma reformulação do sistema InterIMAGE que está sendo desenvolvida para processar grandes conjuntos de dados em *cluster* de computadores físicos ou virtuais. Utiliza-se do pacote *Data Mining Package* que realiza procedimentos de classificação supervisionada envolvendo grande volume de dados em uma infraestrutura distribuída denominada de *Hadoop MapReduce*. O *Apache Hadoop* é uma implementação de código aberto do framework *MapReduce*, proposto pela *Google* (DEAN e GHEMAWAT, 2008). Ele permite o processamento distribuído de conjuntos de dados na ordem de Petabytes entre centenas ou milhares de computadores conectados em rede (KIRAN et al., 2013).

Os Sistemas de Gerenciamento de Banco de Dados Objeto-Relacional (SGBDOR) vêm se consolidando como principal solução de armazenamento de diversos tipos de dados, tais como grandes volumes de dados geoespaciais. De acordo com as especificações do *OpenGIS Consortium* (OGC) em *Simple Features Specification for Structured Query Language* - SFSSQL (POSTGIS, 2016) o PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional de código aberto e de grande porte que oferece suporte a dados espaciais por meio de sua extensão PostGIS para vetores e WKT (*Well-Know Text*) *Raster* para dados matriciais como as imagens de satélite. Neste contexto denomina-se como Sistema Gerenciador de Banco de Dados Geográficos PostgreSQL/PostGIS, e utiliza a linguagem *Structured Query Language* - SQL, procedimentos, funções e consultas complexas entre tabelas e outras características, permitindo o relacionamento entre tabelas do banco com dados espaciais como vetores e matrizes.

Neste contexto, esta tese insere-se na intenção de utilizar os benefícios advindos dos SGBDOR, e objetiva-se a fazer outra implementação do método de segmentação de imagens baseado em OBIA, proposto por Baatz e Schäpe (2000), desenvolvida dentro do *software* InterIMAGE *Desktop*, em um um novo ambiente, neste caso, no Sistema Gerenciador de Banco de Dados Geográficos PostgreSQL/PostGIS.

1.1. Estado da Arte em Sistemas Gerenciadores de Banco de Dados

O volume de dados espaciais disponíveis aumentou, e os desafios de construir plataformas eficientes para processamento estão em:

I - Escalabilidade do sistema. Os sistemas de banco de dados atuais devem ser capazes de processar Petabytes de dados espaciais, armazena-los de forma eficaz e permitir que as aplicações possam recuperá-los eficientemente quando necessário;

II - Interatividade de atuação. O processamento de dados espaciais subjacente do sistema deve permitir formas eficazes de processar o pedido do usuário em um tempo de resposta razoável.

Os conjuntos de dados ambientais atualmente são fornecidos por dispositivos sensores de alta resolução, e podem ser registrados (armazenados) e manipulados

com uso de tecnologias de banco de dados relacionais ou não relacionais, ou ferramentas de base mais específicas.

Eficientes tecnologias estão disponíveis na área de bancos de dados espaciais para processar de forma eficaz conjuntos de dados baseados em entidades espaciais.

O processamento integrado de dados relacionais e de matriz (*raster*) é alcançado pela implementação da linguagem SciQL relatada em (Zhang, Kersten e Manegold 2013), no entanto, o sistema não pode gerenciar a semântica de dimensões de matriz. No MAPAL (*Mapping Analysis Language*) é proposto o gerenciamento da semântica de dimensões de matriz (*raster*) (Villarroya et al. 2016), cuja funcionalidade alcança um processamento declarativo real integrado de dados matriciais e relacionais, incorporando dimensões digitadas como parte dos dados gravados.

Em Eldawy e Mokbel (2015) demonstra-se a abordagem que incorpora o processamento espacial de dados baseados em tecnologias de *big data spatial* (grandes dados espaciais) utilizando como frameworks MapReduce, o Hadoop, com a linguagem de alto nível o Pig Latin. Mas, essa linguagem não é projetada para dados espaciais e não têm suporte para tipos ou funções espaciais. É apresentado o Pigeon, uma extensão espacial para o Pig que fornece funcionalidade espacial, implementado por meio de funções definidas pelo usuário, e permite a integração com funções e operações não espaciais existentes, e é compatível com o padrão *Open Geospatial Consortium* (OGC).

O GeoSpark é um framework de computação em um cluster na memória para processamento de dados espaciais em grande escala (Yu, Wu e Sarwat 2015), que consiste em três camadas: Apache Spark Layer, Camada de Resilient Distributed Datasets (RDD) Espacial e Camada de Processamento de Consulta Espacial. O Apache Spark Layer fornece funcionalidades básicas do Spark que incluem o carregamento / armazenamento de dados para o disco, bem como uma nova abstração de dados chamada conjuntos de dados resilientes (RDDs), para suportar objetos espaciais e executar operações geométricas.

Os Sistemas NoSQL (acrônimo para *Not Only-SQL*) é um termo utilizado para representar uma ampla classe de SGBDs que não se utiliza do modelo relacional para a representação dos dados armazenados (Floratou *et al*., 2012). Pokorny (2011) relata que são projetados para atender a necessidade de armazenamento e

gerenciamento de grandes volumes de dados semi-estruturados ou não-estruturados, sendo classificados de acordo com o modelo que se utiliza para representação de dados, sendo os principais: Chave-valor, orientado a colunas, orientado a documentos, orientado a grafos e XML (*eXtensible Markup Language*).

Busca-se construir o início de uma solução baseada em banco de dados objeto relacional para o processamento integrado de dados matriciais e vetoriais, para fins de segmentação como subsídio a classificação baseada em OBIA.

1.2. Hipótese

O desenvolvimento de aplicações para integrar o sistema gerenciador de banco de dados geográficos PostgreSQL/PostGIS *Raster* e a classificação OBIA pode representar uma solução, no ambiente Desktop, para as limitações que o *software* InterIMAGE apresenta em relação ao tamanho da imagem a ser processada.

1.3. Objetivos

O objetivo geral deste estudo é implementar aplicações para integrar o Sistema Gerenciador de Banco de Dados Geográficos PostgreSQL/PostGIS *Raster* e o sistema InterIMAGE *Desktop* com a implementação/inclusão de um novo operador de segmentação para processamento de grandes imagens de sensoriamento remoto.

Como objetivos específicos, almejam-se:

a) Desenvolver o processamento de segmentação de imagens de satélite, de alta resolução espacial, segundo o conceito de Baatz e Schäpe (2000), no ambiente PostgreSQL/*PostGIS Raster* com desenvolvimento em linguagem de programação C++;
b) Criar uma *Application Programming Interface* (API) para integrar o segmentador baseado no PostreSQL como um operador no InterIMAGE Desktop;
c) Analisar e validar os resultados decorrentes dos processos de segmentação, integração e classificação, decorrentes da integração da plataforma PostgreSQL/PostGIS *Raster* e InterIMAGE 1.43.

1.4. Estrutura da Tese

Esta tese foi estruturada em quatro capítulos organizados conforme se segue:

- ✓ Capítulo 2 - São expostos os principais fundamentos teóricos para a compreensão da tese e dos trabalhos relacionados. São descritos os conceitos de imagens digitais, diferentes técnicas de segmentação de imagens passíveis de serem utilizadas, bem como uma breve explanação sobre sistemas gerenciadores de banco de dados espaciais.
- ✓ Capítulo 3 - São descritos os métodos de adaptação do algoritmo de segmentação proposto por Baatz e Shape (2000) no sistema de gerenciamento de banco de dados geográficos PostgreSQL/*PostGIS Raster*, além dos procedimentos de integração do PostgreSQL para a transferência de resultados de segmentação para o InterIMAGE Desktop.
- ✓ Capítulo 4 - São detalhados os resultados e a discussão referente à implementação e aplicação do segmentador.
- ✓ Capítulo 5 - São detalhadas as principais conclusões deste trabalho, bem como indicações de trabalhos futuros.

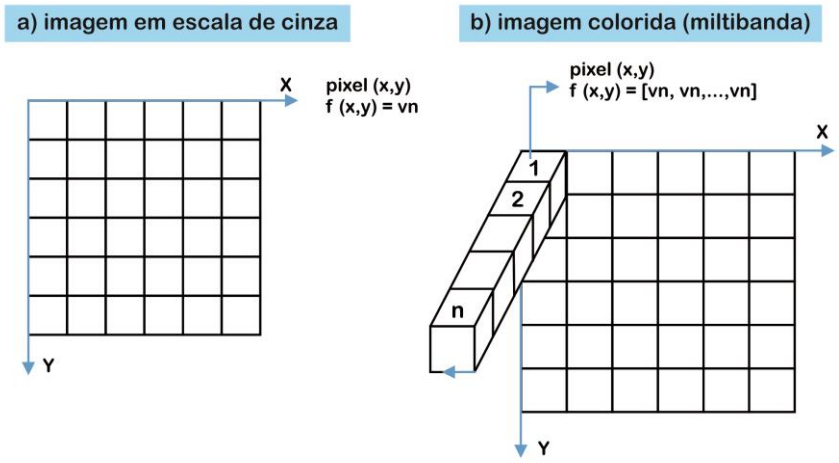
CAPÍTULO 2 - REFERENCIAL TEÓRICO

Neste capítulo são abordados os conceitos de imagens digitais e suas características, segmentação de imagens orbitais e sistema gerenciador de banco de dados geográficos. Esta abordagem faz-se necessária para esclarecimento do problema a ser trabalhado nesta tese, bem como as abordagens das soluções adotadas.

2.1. Imagens digitais

Segundo Gonzalez e Woods (2010), uma imagem digital refere-se a uma função bidimensional de intensidade de luz *f(x,y),* onde *x* e *y* descrevem as coordenadas espaciais e o valor de *f* em algum ponto *(x,y)* corresponde ao nível de cinza (ou brilho) da imagem naquele ponto. Nas imagens com mais de uma banda espectral, os valores de *f* em um determinado ponto *(x,y)* passam a ser uma função vetorial (Figura 3).

Figura 3: Representação bidimensional de uma imagem digital hipotética. n = número de bandas espectrais.



Fonte: Adaptado de Gonzalez e Woods (2010).

Segundo Meneses e Almeida (2012), as imagens de sensoriamento remoto:

> "... devem ser vistas como uma forma de documentos que representam, em escala e sobre um plano 2D, os acidentes e as feições naturais e artificiais da superfície terrestre, a partir da medição de um processo físico da radiação eletromagnética. A

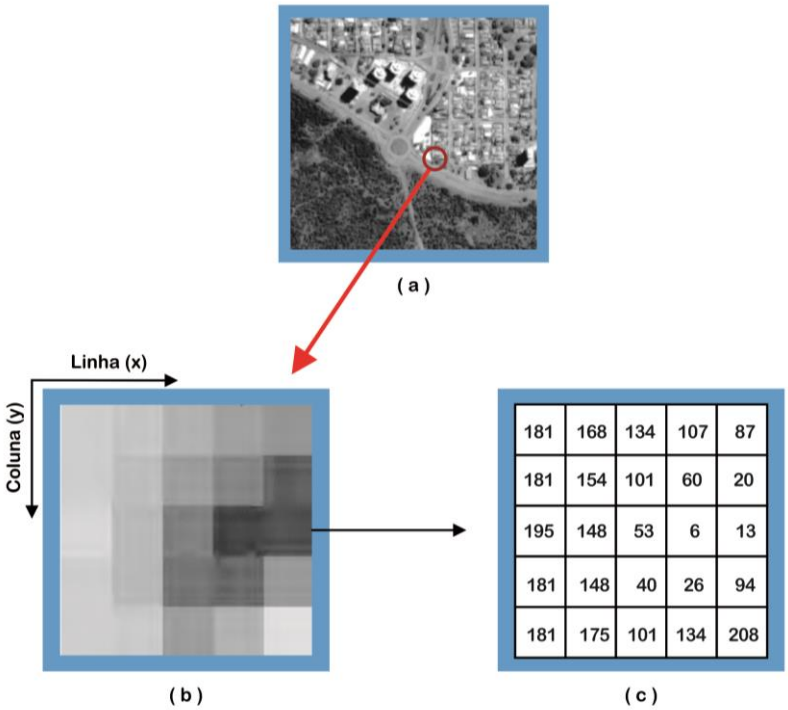
> energia da radiação eletromagnética conduz de forma analógica a informação sobre os objetos e no sensor um conversor analógico/digital converte essa informação em um valor digital, codificado por uma unidade denominada de *pixel*. A forma digital do dado é que possibilita o uso de computadores para processar as imagens, com o objetivo principal de representar porções bem definidas do espaço terrestre, utilizando-se de processamentos matemáticos, estatísticos e probabilísticos dos dados."

Uma imagem f(x,y), para ser possível de ser processada por computadores e exposta na tela do monitor, tem de ser digitalizada tanto espacialmente como em amplitude de níveis de brilho (ou cinza) (MENESES, 2012). A digitalização de coordenadas x,y refere-se à “amostragem de imagem” e a digitalização de amplitude refere-se à “quantização de nível de brilho”. A Figura 4 mostra uma imagem na sua forma digital, o valor da função f(x,y) é representado por um número digital quantizado em bits e que assume somente valores inteiros positivos. Meneses (2012) questionou:

> “ ...por que não executar o processamento em tipos de dados fracionários (*floating point*) que pode representar qualquer valor? O problema é o espaço em disco. Quanto maior o intervalo de valores dos dados, mais espaço em disco é preciso. Por exemplo, tipo de dados em byte usa somente um único byte para representar o valor de cada *pixel*. Tipos de dados em *integer* usam 2 bytes para cada *pixel*, enquanto *floating point* usa 4 bytes. Assim, *floating point* consome duas vezes mais espaço em disco do que o tipo *integer*”.

As imagens de sensoriamento remoto são constituídas por um arranjo em forma de malha ou grade e, por convenção, a origem é sempre no seu canto superior esquerdo (CROSTA, 1993). Para um mesmo sensor, cada *pixel* representa uma área com as mesmas dimensões na superfície imageada. Cada *pixel* possui um atributo numérico *z* que indica o valor digital do mesmo e varia do zero a 2^n, em que *n* é a resolução radiométrica (em bits) do sistema sensor. O valor digital representa a intensidade da energia eletromagnética refletida, emitida ou retroespalhada e medida pelo sensor pela área da superfície imageada correspondente ao tamanho do *pixel*.

Figura 4: Exemplo de uma imagem hipotética mostrando os *pixels* em tons de cinza e os seus correspondentes valores digitais



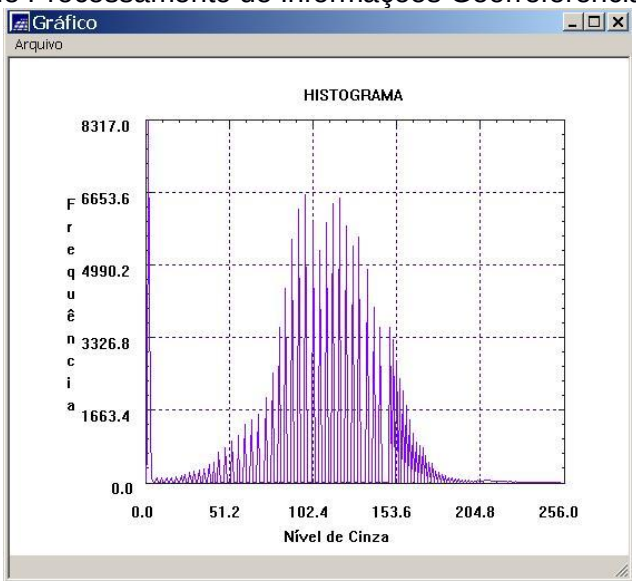
Fonte: Adaptado de Meneses e Almeida (2012).

Os valores digitais de uma determinada imagem podem ser representados na forma de histogramas. O eixo *y* indica a frequência de ocorrência de cada *pixel* na imagem (Figura 5). Portanto, o histograma de uma imagem é a distribuição de suas intensidades, onde o eixo *x* corresponde ao intervalo de valores digitais, e o eixo *f(x)* representa as frequências.

A análise de imagens baseada em objetos possibilita a multiplicidade de informações adicionais que podem ser derivadas de objetos de imagem em contraste com a quantidade de informações disponíveis a partir de *pixels* individuais. Um *pixel* geralmente contém um vetor de informações representando cada banda ou camada em um conjunto de dados. As respostas espectrais das imagens digitais estão como números digitais (CHUBEY *et al*., 2006).

Em contrapartida, os objetos de imagem são compostos por grupos de vários *pixels*, permitindo o cálculo de estatísticas agregadas como média e desvio-padrão dos valores digitais subjacentes de um objeto.

Figura 5: Exemplo de um histograma de valores digitais de uma imagem digital, visualizado no *software* Sistema de Processamento de Informações Georreferenciadas (Spring).



Fonte: Adaptado de Meneses e Almeida (2012).

Além das informações baseadas em resolução espectral, podem-se obter as informações baseadas no tamanho, na forma e no contexto do objeto para serem calculadas, bem como as informações pertencentes aos subobjetos ou super-objetos de um objeto, porém, primeiramente estabelecendo-se uma hierarquia de objetos de imagem de vários níveis (CHUBEY *et al*., 2006).

2.2. Segmentação de imagens

A qualidade das informações provenientes da análise de dados de uma imagem muitas vezes está relacionada com a qualidade do tratamento feito sobre os dados originais. As técnicas de extração de informações originadas de imagens de sensoriamento remoto pertencem à área do processamento referida como análise de imagens, podendo ter, como passo inicial, a obtenção de segmentação.

Algumas aplicações necessitam de procedimentos de subdivisões de imagens digitais em objetos ou partes de objetos, ou seja, as suas formas e/ou as suas posições precisam ser determinadas para melhor interpretação das informações contidas na imagem. Este processo denomina-se de segmentação de imagens e constitui-se em determinar a qual objeto pertence cada um dos *pixels* de uma imagem.

Gonzalez e Woods (2010) definiram a segmentação como sendo uma divisão de uma imagem em regiões que satisfaçam as condições de um problema a ser

resolvido, e a segmentação deve parar quando os objetos ou as regiões de interesse da aplicação forem identificados. A segmentação de imagem é basicamente um problema de percepção psicofísica e, portanto, não susceptível de uma solução puramente analítica (NIKHIL e SANKAR, 1993). Portanto, qualquer algoritmo de segmentação, objetivando um resultado satisfatório, independentemente do modelo matemático adotado, deve ser complementado por heurísticas que envolvam informação semântica como conhecimento *a priori* sobre as imagens que estão sendo processadas (YONG *et al.*, 2004).

As abordagens de identificação de descontinuidades ou de similaridades geralmente são utilizadas na segmentação para detectar objetos de uma imagem. Na abordagem de identificação de descontinuidades, investigam-se as mudanças abruptas de valores digitais na forma de linhas ou bordas enquanto na abordagem de similaridades, procura-se agregar em uma mesma classe (objetos), os elementos da imagem que tenham valores similares em determinado conjunto de características. Dentre os exemplos de algoritmos que calculam descontinuidades ou similaridades, têm-se os de limiarização, crescimento de regiões, divisão e fusão de regiões e divisor de águas *(watershed*).

Cita-se, como exemplo da utilização dos resultados de segmentação para fins de classificação de imagens, o projeto TerraClass que está sendo executado a partir de uma parceria entre a Empresa Brasileira de Pesquisa Agropecuária (Embrapa), representada pelas unidades da Embrapa Amazônia Oriental e da Embrapa Informática Agropecuária, e o Centro Regional da Amazônia (CRA) do Instituto Nacional de Pesquisas Espaciais (INPE) (ALMEIDA *et al.*, 2016).

2.2.1. Limiarização

A segmentação por limiarização baseia-se principalmente nas propriedades espectrais em que a variação espectral é representada por um histograma da imagem. A escolha de objetos é feita pela delimitação no histograma em que a identificação do limite adequado entre objeto e *background* é a tarefa principal da identificação.

Para separar um objeto de um *background* em uma imagem monocromática, a segmentação por limiarização consiste em encontrar um valor *T* de intensidade que possibilite a separação dos *pixels* da imagem em duas classes. Os *pixels* que forem

maiores que *T* pertencerão a um grupo e os que forem menores ou iguais pertencerão a outra classe.

Qualquer ponto (*x,y*) na imagem em que *f(x,y)* > *T* é chamado de ponto do objeto, do contrário, o ponto é chamado de ponto de fundo (GONZALES e WOODS, 2010). Quando *T* é uma constante aplicada a uma imagem inteira, o processo é definido como limiarização global, enquanto que quando *T* se altera no decorrer da imagem, usa-se o termo limiarização variável.

2.2.2. Crescimento de regiões

O crescimento de regiões é um processo que reúne os *pixels* ou as sub-regiões em regiões maiores com base em condições predefinidas para o crescimento. Gonzalez e Woods (2010) mencionaram que:

> "A abordagem básica é começar com um conjunto de pontos 'semente' e, a partir deles, fazer as regiões crescerem anexando a cada semente aqueles *pixels* vizinhos que têm propriedades predefinidas semelhantes às das sementes (como os intervalos específicos de intensidade ou cor). A seleção de um conjunto de um ou mais pontos de partida muitas vezes pode ser baseada na natureza do problema."

No procedimento de segmentação por crescimento de regiões, considera-se um *pixel* inicial como sendo uma região. Em seguida, iniciam-se as comparações com as outras regiões adjacentes, validando o limiar de similaridade estabelecido por meio de avaliação de médias em que o limiar define a distância entre elas. Quando a distância é inferior ao limiar pré-estabelecido, as regiões são agregadas, caso contrário, ficam separadas (MENESES e ALMEIDA, 2012).

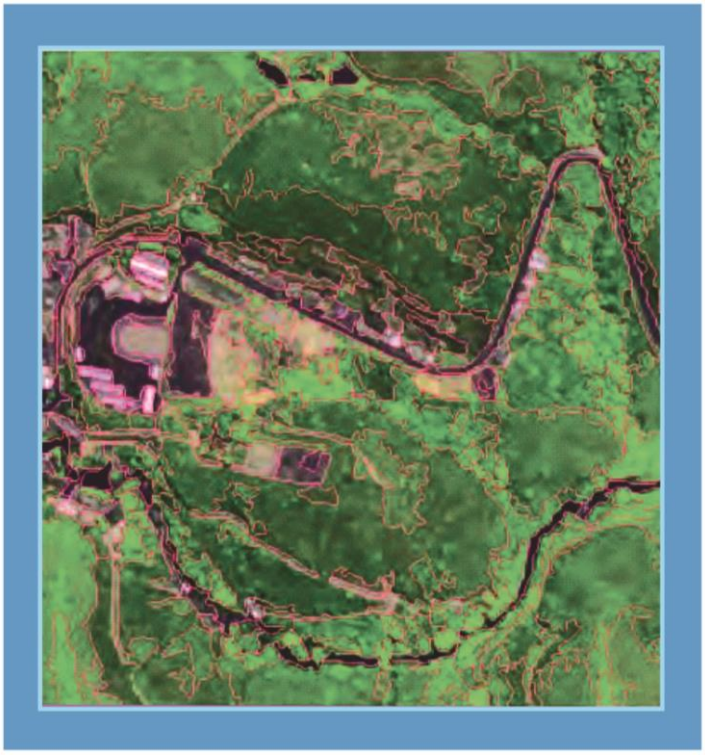
Meneses e Almeida (2012) definiram limiar de similaridade como:

> "o grau de similitude entre o *pixel* candidato à inclusão em um determinado segmento e os parâmetros estatísticos referentes aos *pixels* que já fazem parte do segmento em questão. O tamanho

mínimo dos polígonos, também conhecido por limiar de área, se refere ao número mínimo de *pixels* admitidos em qualquer um dos segmentos."

A Figura 6 ilustra um exemplo de uma imagem segmentada utilizando-se o algoritmo de crescimento de regiões.

Figura 6: Exemplo de uma segmentação por crescimento de regiões.



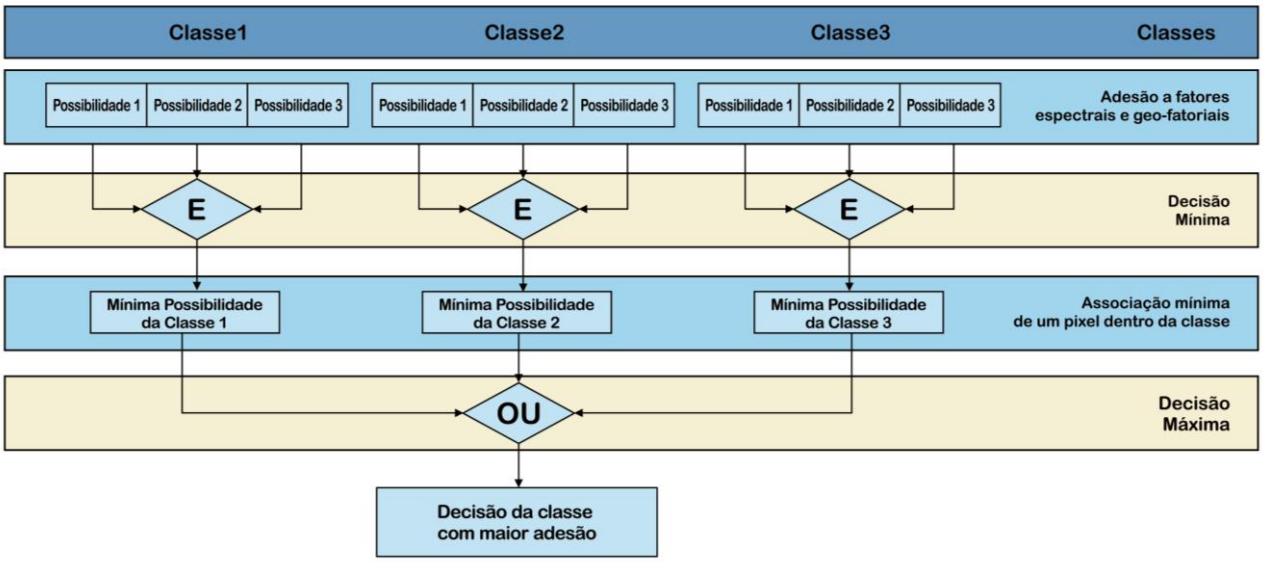
Fonte: Adaptado de Meneses e Sano (2012).

2.2.3. Modelo fuzzy

A segmentação *fuzzy* foi conceituada por Zadeh (1965) como uma técnica baseada em crescimento de regiões que determina um grau de pertinência (entre zero e um) para cada elemento da imagem, indicando o quanto o elemento pertence, ou não, a um determinado objeto existente na imagem. No método *fuzzy,* um elemento de uma imagem segmentada pode pertencer, ao mesmo tempo, a mais de um objeto, ou seja, a segmentação *fuzzy* pode acrescentar limites difusos para objetos.

Na segmentação *fuzzy*, a noção de conectividade *fuzzy* tem a premissa de coesão dos elementos da imagem em um objeto, por meio da atribuição de uma força de conexão para todos os caminhos viáveis entre todos os possíveis pares de elementos da imagem. A Figura 7 apresenta a representação esquemática de uma decisão de classe *fuzzy*.

Figura 7: Aplicação esquemática de uma decisão de classe *fuzzy* com classificação espectral.



Fonte: Adaptado de Förster e Kleinschmit. (2008).

2.2.4. Análise de cluster

A análise de *cluster* implica em um agrupamento de *pixels* no espaço multiespectral. Os *pixels* pertencentes a um conjunto particular são, por conseguinte, semelhantes espectralmente (RICHARDS e JIA, 2006). Para quantificar esse agrupamento, é necessário conceber uma medida de similaridade e a métrica mais utilizada é a de distância simples no espaço multiespectral, conhecida como distância euclidiana. Por similaridade entende-se como o grau de similaridade entre o *pixel* candidato de inclusão, em um determinado segmento, e os parâmetros estatísticos referentes aos *pixels* que já fazem parte do segmento em questão.

2.2.5. Segmentação multi-resolução

A concepção de segmentação multi-resolução proposta por Baatz e Schäpe (2000) adota a noção de evolução de rede fractal (FNEA - *Fractal Net Evolution Approach*), em que a imagem de sensoriamento remoto é considerada como tendo natureza fractal.

A abordagem da evolução rede fractal utiliza a teoria dos conjuntos *fuzzy* para extrair os objetos de interesse, na escala pretendida, concomitantemente segmentando as imagens em escalas finas e grosseiras, e incorpora a semântica da imagem entre os níveis e os seus elementos. O principal desafio e flexibilidade consistem em estabelecer as regras de agregação para as entidades de nível inferior, que resultem em uma melhoria das classificações de imagens e uma nova organização para integrar as regras semânticas no processamento de imagens (BLASCHKE e HAY, 2001).

A etapa da segmentação multi-resolução, em uma classificação orientada a objetos, delimita os objetos que serão classificados em seus específicos níveis de detalhes. A interpretação de uma cena considera não apenas a dimensão espectral, mas também a dimensão espacial, pois as informações contextuais são significativas. Os segmentos, ou objetos, resultantes da segmentação da imagem, baseados em parâmetros espectrais e de forma, podem ser reagrupados em objetos maiores denominados superobjetos, e conservam relações com os segmentos que os constituem em um nível hierárquico mais baixo, os subobjetos.

Em contraposição aos métodos fundamentados em *pixels*, a segmentação da imagem é baseada na estrutura de orientação a objetos em que extrai os objetos de interesse na escala almejada. Em cada escala de segmentação, a importância relativa dos parâmetros espectrais e de forma podem ser definidas através de pesos que variam entre zero e um. No resultado da segmentação, consegue-se uma estrutura hierárquica que descreve a informação de dados simultaneamente em diferentes resoluções. Os objetos estão em uma rede de vizinhança e hierarquia onde cada objeto diferencia os seus vizinhos, subobjetos e superobjetos.

Com o FNEA, os usuários são demandados a apontarem para níveis de escala diferentes, levantando a hipótese de que quase a totalidade de atributos da estrutura de imagem - cor, textura ou forma - são essencialmente dependentes da escala. (BLASCHKE e HAY, 2001).

Uma heurística iterativa como procedimento de otimização visa obter a menor heterogeneidade global possível entre uma imagem. A base para esse parâmetro é o grau de diferenciação entre duas regiões. Como essa diferença diminui, o ajuste das duas regiões diz-se que são mais próximas. Essas diferenças são otimizadas num processo de heurística, comparando os atributos das regiões (BLASCHKE e HAY, 2001).

Dado um determinado espaço de características, dois objetos da imagem são considerados semelhantes quando estão próximos um do outro nesse espaço de características. Dado um espaço de características *d*-dimensional, a heterogeneidade *h* é definida como Equação 1:

$$h = \sqrt{\sum_{d} (f_{1d} - f_{2d})^2} \quad (1)$$

As distâncias podem ser padronizadas pelo desvio-padrão sobre todos os segmentos da função em cada dimensão (BLASCHKE e HAY, 2001) (Equação 2):

$$h = \sqrt{\sum_{d} \left(\frac{f_{1d} - f_{2d}}{\sigma_{fd}} \right)^2} \quad (2)$$

A Equação 3 define a homogeneidade das duas regiões adjacentes, descrevendo a diferença de heterogeneidade *h* das duas regiões anteriores (h_1 e h_2), após uma mesclagem virtual (*hm*). Dada uma definição adequada da heterogeneidade para uma única região, o crescimento de heterogeneidade em uma região deve ser minimizado (BLASCHKE e HAY, 2001).

$$h_{diff} = h_m - \frac{h_1 + h_2}{2} \quad (3)$$

Baazt e Schäpe (2000) propuseram o método de segmentação multi-resolução em um algoritmo de crescimento de regiões.

Demonstrando o uso da técnica de segmentação multi-resolução, apresenta-se, na Figura 8, a segmentação executada em dois níveis de escala: o primeiro, mais detalhado (nível 1), e o segundo, mais geral (nível 2). A segmentação do nível

1 foi elaborada a partir da segmentação no nível 2, ou seja, todas as regiões do nível 1 representam subobjetos das quadras e eixos de rua.

Figura 8: Representação de segmentação multi-resolução realizada em dois níveis de uma paisagem

Fonte: Adaptado de Pinho *et al.* (2005).

Os dois principais componentes da segmentação multi-resolução são:

· A heurística de decisão, para determinar os objetos da imagem em cada passo;

· A definição de homogeneidade de objetos da imagem para calcular o grau de encaixe para um par de objetos da imagem.

A princípio, cada segmento representa apenas um *pixel* da imagem e todos os *pixels* estão associados a algum segmento. Os segmentos crescem na proporção em que são unificados com seus vizinhos, e o mínimo aumento na heterogeneidade é utilizado como método para a seleção do vizinho com o qual o segmento irá se unir. Entretanto, a fusão ocorre somente se o acréscimo da heterogeneidade for menor do que certo limiar. A medida de heterogeneidade possui uma componente espectral e uma morfológica. Para continuar o processamento pode-se utilizar uma das heurísticas de decisão: Melhor Ajuste (*Best Fitting*) ou Melhor Ajuste Mútuo

(*Mutual Fitting*). A heurística de Melhor Ajuste produz-se a fusão do segmento visitado com o seu vizinho mais similar, e na de Melhor Ajuste Mútuo a fusão é completada se a relação de maior similaridade for mútua, isto é, se o segmento visitado for da mesma forma de seu vizinho similiar. Salienta-se qual seja a heurística adotada a fusão encontra-se vinculada ao correspondente aumento da heterogeneidade seja menor ao quadrado do parâmetro do algoritmo denominado de escala (e). A escala estabelece o máximo crescimento de heterogeneidade possível, decorrente da fusão entre os dois segmentos, e que determina indiretamente a dimensão média dos segmentos finais (FERREIRA *et al*., 2013).

O acréscimo da heterogeneidade, resultante da fusão entre dois objetos, faz o direcionamento do algoritmo através das iterações e passa ser definido como fator de fusão. Um componente para a heterogeneidade espectral (h_{cor}) e um componente para a heterogeneidade espacial (h_{forma}) estão contidos no fator de fusão (Equação 4), onde w_{cor} é o peso dado ao parâmetro de cor, variando entre zero e um, e define a importância relativa entre os componentes espectral e espacial:

$$f = w_{\text{cor}} \cdot h_{cor} + \left(1 - w_{\text{cor}}\right) \cdot h_{forma} \tag{4}$$

Happ *et al*. (2010) descreveram a Equação 5 que mostra a formulação da heterogeneidade espectral, onde Obj_1 é o segmento selecionado, Obj_2 é o vizinho analisado e Obj_3 é o segmento resultante da união de Obj_1 com Obj_2. Nessa equação, c é o índice da banda espectral e w_c é um peso arbitrário definido para a banda c; σ é o desvio-padrão dos valores dos *pixels* na banda c, considerando todos os *pixels* pertencentes ao segmento Obj_i; e n é o número de *pixels* em Obj_i, para i = 1, 2 e 3.

$$h_{cor} = \sum_{c} w_c \left(n_{Obj3} \cdot \sigma_c^{Obj3} - \left(n_{Obj1} \cdot \sigma_c^{Obj1} + n_{Obj2} \cdot \sigma_c^{Obj2}\right)\right) \tag{5}$$

A componente morfológica possui o componente relativo à compactação h_{cmpct} e o componente de suavidade h_{suave}. O fator de compactação w_{cmpct} define a importância relativa entre os componentes de compactação e suavidade. A forma é expressada pela Equação 6.

$$h_{forma} = w_{cmpct} \cdot h_{cmpct} + \left(1 - w_{cmpct}\right) \cdot h_{suave} \quad (6)$$

A Equação 7 descreve a componente de compactação, onde $lObj_i$ é o perímetro dos objetos, para i = 1, 2 e 3.

$$h_{cmpct} = n_{Obj3} \cdot \frac{l_{Obj3}}{\sqrt{n_{Obj3}}} - \left(n_{Obj1} \cdot \frac{l_{Obj1}}{\sqrt{n_{Obj1}}} + n_{Obj2} \cdot \frac{l_{Obj2}}{\sqrt{n_{Obj2}}} \right) \quad (7)$$

A componente de suavidade é representada pela Equação 8, sendo $bObj_i$ o perímetro do correspondente retângulo envolvente mínimo, para i = 1, 2 e 3.

$$h_{\text{suave}} = n_{\text{Obj3}} \cdot \frac{l_{Obj3}}{b_{Obj3}} - \left(n_{\text{Obj1}} \cdot \frac{l_{Obj1}}{b_{Obj1}} + n_{\text{Obj2}} \cdot \frac{l_{Obj2}}{b_{Obj2}} \right) \quad (8)$$

Portanto, o crescimento dos segmentos condiciona-se a um critério de heterogeneidade ajustável (HAPP *et al*., 2010). Os ajustes podem ser feitos pela escolha dos parâmetros de escala (*e*), dos pesos das bandas espectrais (*wc)*, do fator de cor (*wcor*) e do fator de compactação (*wcmpct*). Os tamanhos dos segmentos constituídos são influenciados diretamente pelo ajuste do parâmetro de escala (*e*).

A significância de cada banda espectral, e a relevância relativa entre forma e cor, bem como entre compactação e suavidade, são conseguidos pelo ajuste nos parâmetros do algoritmo.

2.3. Sistemas de banco de dados geográficos

Banco de dados é definido como uma coleção de dados que, tipicamente, descrevem as atividades de uma ou mais organizações relacionadas (RAMAKRISHNAN e GEHRKE, 2008). Define-se como uma coleção de dados operacionais armazenados e que são usados por uma determinada organização (DATE, 2004). O banco de dados, por si só, pode ser considerado como o

equivalente eletrônico de um armário de arquivamento. Considerado como um repositório ou recipiente para uma coleção de arquivos de dados computadorizados.

O banco de dados pode ser entendido ainda como uma unificação de vários arquivos que, de outra forma, seriam distintos, e sem redundância parcial ou total entre esses arquivos. De modo geral, os dados de um banco de dados estão integrados e compartilhados. Os dados inseridos em um banco de dados são considerados como "persistentes", pois os mesmos só podem ser removidos por alguma requisição explícita e não como mero resultado, por exemplo, de algum programa finalizando sua execução.

Um modelo de dados é uma descrição conceitual de banco de dados, não incluindo qualquer referência ao *layout* ou à própria estrutura física do banco de dados. O *layout* físico do banco de dados, que descreve como os dados são organizados e armazenados, é chamado de esquema de banco de dados ou simplesmente esquema, que, em termos simples, é a tradução de um modelo de dados em uma representação física que pode ser implementado no computador. Em termos práticos, o esquema é mais conceitual do que físico porque se trata de uma descrição da base de dados, incluindo suas tabelas e as relações entre elas (YEUNG e HALL, 2007).

Date (2004) definiu um sistema de banco de dados como um sistema computadorizado de manutenção de registros, cuja finalidade geral é armazenar informações e permitir que os usuários consultem e atualizem. Os sistemas de banco de dados são projetados para armazenarem grandes blocos de informações. O gerenciamento de sistemas de banco de dados envolve tanto a definição das estruturas para o armazenamento de informações como a provisão dos mecanismos para as manipulações das informações. Concomitantemente, o sistema de gerenciamento deve fornecer segurança das informações armazenadas para que, no caso de falhas do sistema, possam se evitar possíveis resultados anômalos ou acessos não autorizados (SILBERSCHATZ *et al.,* 2006).

Para manter grandes repositórios compartilhados de dados, são utilizados os Sistemas de Gerenciamento de Banco de Dados (SGBD) (HEUSER, 2004). Um SGBD é definido como sendo um conjunto de programas responsáveis pelo gerenciamento de uma base de dados que retira, do usuário, a responsabilidade de gerenciar acesso, manipulação e organização dos dados (SILBERSCHATZ *et al.,*

2006). O SGBD disponibiliza uma interface para que os usuários possam manipular os dados de uma base de dados.

Os sistemas de banco de dados podem ser classificados segundo critérios diferentes (Quadro 1). Convencionalmente, eles são classificados de acordo com os diferentes modelos de dados em que foram construídos. Os modelos inicialmente se dividiam em três categorias que caracterizam a evolução dos sistemas: hierárquicos, de rede e relacionais. Uma quarta classe chamada sistemas de banco de dados orientados a objetos surgiu com os avanços na década de 1990, e é conhecido como tecnologia de orientação a objetos. As evoluções dos modelos apresentaram os sistemas híbridos que foram desenvolvidos aproveitando-se dos conceitos e técnicas relacionais e de orientação a objetos. Tais sistemas são comumente referidos como sistemas de banco de dados objeto-relacional (YEUNG e HALL, 2007).

Quadro 1: Classificação de sistemas de banco de dados.

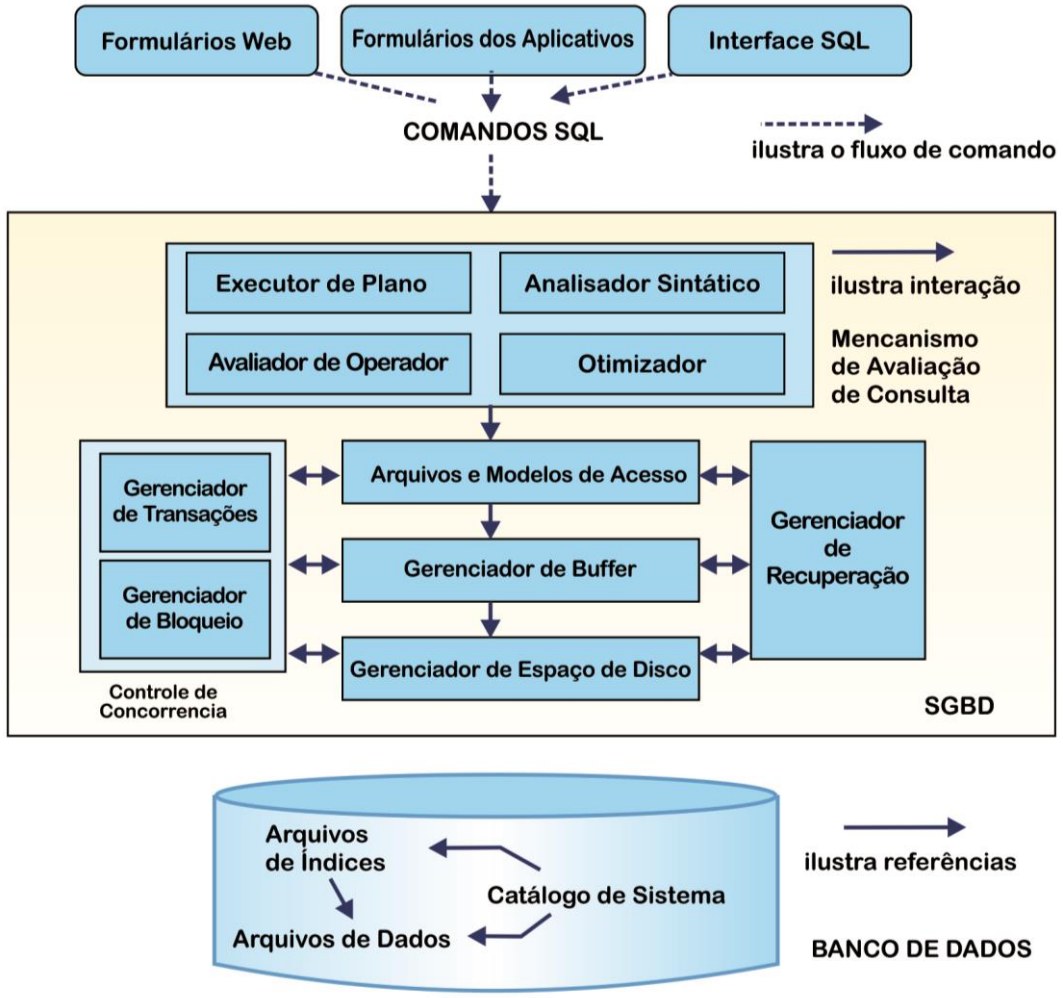
Critério de Classificação	**Categorias**
Modelo de dados	Sistemas hierárquicos Sistemas em rede Sistemas relacionais Sistemas orientados a objetos Sistemas objeto-relacionais
Funções primárias de banco de dados	Armazenamento de dados ou sistemas de inventário Sistemas de transação Sistemas de suporte à decisão
Natureza dos dados	Sistemas de informação espacial Sistemas de informação não-espaciais
Objetivos da informação	Sistemas de custódia e armazenamento de dados (*data wharehouse*) Sistemas orientados a projetos
Plataformas de *hardware* e configurações de sistemas	Sistemas distribuídos Sistemas *desktop*

Fonte: Adaptado de Yeung e Hall (2007).

O catálogo de um SGBD contém informações como a estrutura de cada arquivo, o tipo e o formato de armazenamento de cada item de dados, além de restrições sobre os dados. A informação armazenada no catálogo é chamada de metadados, sendo ela responsável pela descrição da estrutura primária do banco de dados (ELMASRI e NAVATHE, 2011).

Um modelo de dados é uma coleção de construtores de alto nível para descrição dos dados que encobrem vários detalhes de baixo nível do armazenamento (RAMAKRISHNAN e GEHRKE, 2008). O modelo de dados relacional representa o banco de dados como uma coleção de relações. Formalmente, cada relação se assemelha a uma tabela de valores ou, em certa medida, uma linha plana de registros, sendo que cada registro tem uma estrutura linear. Quando uma relação é vista como uma tabela de valores, cada linha da tabela representa uma coleção de valores de dados relacionados. A linha representa um fato que tipicamente corresponde a uma entidade do mundo real ou relacionamento. A Figura 9 ilustra a estrutura simplificada de um SGBD típico com base no modelo de dados relacional.

Figura 9: Arquitetura de um SGBD.



Fonte: Adaptado de Ramakrishnan (2008).

O SGBD-R (relacional) constitui-se de um modelo relacional de dados, em que um banco de dados é organizado como uma coleção de relações, cada qual com atributos de um tipo específico. Os tipos podem ser definidos como números inteiros, de ponto flutuante, cadeias de caracteres, datas e campos binários longos (BLOB - *Binary Large OBject*). A manipulação dos dados desses modelos permite uma variedade de operações (exceto para o tipo BLOB), tais como: aritméticas, de conversão, de manipulação textual e operações com data.

A linguagem de consulta SQL para banco de dados relacional, desenvolvida como parte do projeto System R, da *International Business Machines* (IBM), passou a ser a linguagem de consulta padrão. No final dos anos 1980, a SQL foi padronizada. O padrão atual (SQL: 1999) foi adotado pelo *American National Standards Institute* (ANSI) e pela *International Organization for Standardization* (ISO), (RAMAKRISHNAN e GEHRKE, 2008).

Os sistemas de bancos de dados espaciais incluem-se na definição de banco de dados não-convencionais, pois permitem o armazenamento e manipulação de dados vetoriais e matriciais, com referências espaciais, podendo associar-se a eles os dados tabulares (características dos bancos de dados convencionais). Os sistemas de banco de dados espaciais definem-se como uma classe de sistemas de banco de dados que possuem as seguintes características (GÜTING, 1994):

- ✓ Trata-se de um sistema de banco de dados;
- ✓ Disponibiliza tipos de dados espaciais em seu modelo de dados e linguagem de consulta;
- ✓ Fornece tipos de dados espaciais na sua implementação, suportando pelo menos a indexação espacial e os algoritmos eficientes para junções espaciais.

Os dados espaciais são dados que, por meio de um atributo espacial, representa um local na superfície ou próximo dela, podem ser exibidos, manipulados e analisados (YEUNG e HALL, 2007). Esse atributo espacial é correntemente disponibilizado sob a forma de pares de coordenadas que são registrados em um sistema de coordenadas geográficas e permitem que a posição e a forma de uma particular característica espacial sejam medidas e representadas graficamente.

Um sistema de banco de dados espaciais (geográficos) define os tipos de dados especiais para objetos geométricos e permite que o usuário armazene dados de natureza geográfica. Eles fornecem funções especiais e índices para consulta e

manipulação de dados que podem ser usados por comandos SQL. Entretanto, muitas vezes pode ser utilizado apenas como um recipiente de armazenamento de dados espaciais (OBE *et al*., 2009).

As estruturas de dados utilizadas em sistemas de bancos de dados espaciais podem ser divididas em vetoriais e matriciais. As estruturas vetoriais correspondem às formas básicas de pontos, linhas e áreas (ou polígonos), e são utilizadas para representar as coordenadas das fronteiras de cada entidade geográfica, definidas por suas coordenadas cartesianas. Na estrutura matricial, também denominado de *raster*, a representação do terreno é feita por uma matriz *M(i,j)*, composta por *i* colunas e *j* linhas, e são nomeadas como células, ou mais comumente denominadas de *pixels*. Cada *pixel* apresenta um valor referente ao atributo, além dos valores que definem o número da coluna e o número da linha, correspondendo, quando o arquivo está georreferenciado, às coordenadas cartesianas *x* e *y*, respectivamente. As imagens de satélites, que possuem estrutura matricial, são inseridas com coordenadas geográficas. Essas coordenadas representam um ponto central ou dois pontos, um para o canto superior e outra para o canto inferior.

O SGBD-OR (objeto-relacionais) estende o modelo relacional e propõe a extensão dos mecanismos de indexação sobre os novos tipos de dados geométricos (CÂMARA *et al.*, 2005). Os dados geométricos são comumente entendidos como um ramo da matemática que lida com as propriedades e relações de pontos, linhas, ângulos, superfícies, sólidos e espaços dimensionais.

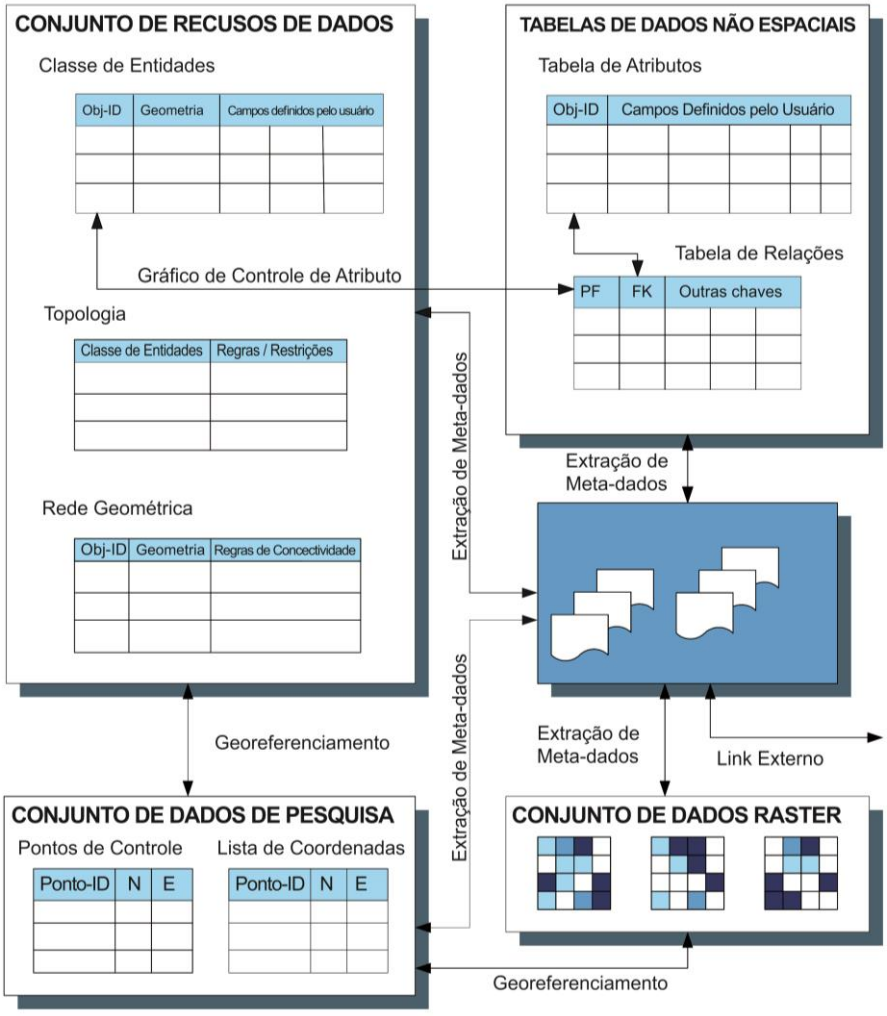
O *Open Geospatial Consortium* (OGC), com a publicação do estudo Especificação de Recursos Simples do OpenGIS para SQL (OGC, 1999), formalizou o uso de dados geométricos e propôs uma hierarquia de tipos de dados espaciais, chamada de modelo de objeto de geometria, que permite que os recursos espaciais sejam representados em um banco de dados. No modelo de objeto de geometria, a palavra "geometria" é usada para representar um recurso espacial como um "objeto" tendo pelo menos um atributo de um tipo geométrico em um banco de dados (YEUNG e HALL, 2007).

Como uma classe raiz do modelo de objeto de geometria da hierarquia, a geometria é uma construção não instanciável, ou seja, não contém nenhuma ocorrência ou instância de recursos do mundo real. A classe de geometria de base possui quatro subclasses: ponto; curva; superfície; e coleção de geometria. Essas subclasses geométricas são construções instanciáveis, ou seja, contêm ocorrências

de instâncias de recursos do mundo real, além de um conjunto definitivo de processos (métodos) que são utilizados para validar suas respectivas propriedades geométricas, definir suas relações espaciais e dar suporte na sua utilização na análise espacial.

A Figura 10 ilustra a estrutura de um sistema de banco de dados espaciais em que se podem armazenar vários tipos de dados espaciais, topologia, dados de atributos e metadados. Os dados espaciais que compartilham os mesmos atributos (dados da mesma classe de recurso) são armazenados em uma única tabela. Nessa tabela, existem dois conjuntos de campos, os campos predefinidos e os campos personalizados. Os campos predefinidos incluem o identificador de recurso (campo FID), a geometria que descreve a forma do recurso e um campo de rastreamento de geometria que registra a área do recurso.

Figura 10: Estrutura de um banco de dados espacial usando um SGBD-OR para o armazenamento de dados espaciais e relacionamentos topológicos



Fonte: Adaptado de Yeung e Hall (2007).

As regras de integridade de um sistema de banco de dados espaciais (*geodatabase)* podem serem aplicadas a uma classe de recursos particular em uma única tabela para impor relações topológicas, como adjacência (tais como remoção de lacunas entre polígonos de parcela) e conectividade (por exemplo, garantindo a vinculação adequada entre diferentes segmentos de uma rodovia). Eles também podem ser aplicados a diferentes classes de recursos em tabelas diferentes para manter a integridade da geometria coincidente entre as características espaciais de diferentes classes de recursos (YEUNG e HALL, 2007).

Na década de 1980, os Sistemas de Gerenciamento de Banco de Dados (SGDB) eram definidos em armazenamento puramente baseado em *tuplas* e relações, e os princípios de extensibilidade eram pesquisados para atender as necessidades que envolvem elementos de dados complexos. Nesse período, pesquisas como os conduzidos por Bernstein e Lometo (1987), Osborn (1987) e Stonebraker e Hirohama (1987) apontaram a viabilidade de incorporação aos SGBDs de técnicas já ratificadas pela área de engenharia de *software* na implementação de aplicações, como, por exemplo, o modelo em camadas. Dessas experiências, os SGBDs organizaram-se em um conjunto de camadas conectadas entre si por interfaces simplificadas. Esses métodos permitiam que camadas específicas (identificadas no meio físico para tipos de dados distintos, algoritmos de indexação, mecanismos de armazenamento de dados, e demais) permitissem ser substituídas em tempo de compilação, por camadas mais apropriadas à conjuntura de uso (BATORY, 1987).

Os novos tipos de dados, a princípio compilados como camadas de acoplamento variado (CAREY *et al.,* 1986) e subsequentemente estabelecidos como composições de códigos compilados para execução de operações e especificações armazenadas no dicionário de dados dos SGBDs (STONEBRAKER, 1986), proporcionaram um nível mais alto no processo de modelagem e construção de instâncias de banco de dados, sintetizando a representação e o mapeamento de entidades presentes no domínio do *software* para entidades na esfera da persistência. Surge, desse contexto, o conceito de tipo de dado customizado à aplicação ou tipo de dado abstrato (ADT - *Abstract Data Ty*pe), estendendo o catálogo de tipos de dados padrão para definições e implementações de tipos contextualizados, orientados a aplicações de alvos com características específicas, tais como imagens e dados geométricos e espaciais.

Os tipos customizados passam a ser procedimentos escritos e compilados em diferentes linguagens de programação e interligados à estrutura base do SGBD por meio de uma interface bem definida. Surgiram assim as *User-Defined Functions* (UDFs), interfaces que proveem meios para manipulação de argumentos e valores de retorno, suportada por descrições definidas em dicionário de dados que permitem, em tempo de execução, verificar a correção sintática de chamadas aos procedimentos a partir de instruções *Data Manipulation Language* (DML) (LINNEMANN, 1988; ORDONEZ, 2010). As operações de extensibilidade do SGBD são dirigidas por um catálogo (*catalog-driven*) e são vistos pelo usuário como qualquer outra tabela, e o SGBD armazena suas informações internas nestas tabelas.

2.3.1. Sistema gerenciador de banco de dados objeto-relacional PostgreSQL/PostGIS *Raster*

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional, da categoria de *software* livre, definido em conformidade com os padrões SQL:2011 (ISSO, 2011). As transações realizadas no PostgreSQL, através de leitura e/ou escrita no banco de dados, são implementadas de acordo com os conceitos de atomicidade, consistência, isolamento e durabilidade (ACID). Quando uma transação é completamente executada ou voltada totalmente sem causar efeitos no banco de dados chama-se atomicidade. A coerência nas operações realizadas é consistência. A execução de uma transação que não interfere ou sofre interferência em relação às demais transações em execução é denominada de isolamento. E, os resultados das transações persistidos fisicamente no banco de dados compreendem a durabilidade (PostgreSQL, 2018).

O PostGIS é uma extensão do PostgreSQL com suporte a objetos geográficos em banco de dados que permitem os armazenamentos, as manipulações e análises espaciais. E, que oferece suporte a dados de geometrias vetoriais e *raster* segundo as especificações da *Simple Features Specification for SQL - Structured Query Language* - SFSSQL, bem como, suporte para operações topológicas, operações métricas, esquema de tabelas de metadados das informações espaciais e funções (POSTGIS, 2018). Foi desenvolvido a partir do projeto PostgreSQL, iniciado em 1986 na Universidade da Califórnia em Berkeley, e permite o uso de linguagem de

consulta estruturada SQL. Desenvolvido pela *Refractions Research*, o PostGIS é liberado sob a licença *General Public License* (GNU), que tem, como princípio, um projeto em tecnologia de banco de dados espaciais de *software* livre.

A capacidade de armazenamento de dados do PostgreSQL limita-se ao tamanho máximo disponível em disco rígido para armazenamento de dados, e conceitualmente é definido como ilimitado. Sendo que a grandeza máxima de uma tabela é de 32 TBytes, o tamanho máximo de uma linha é de 1,6 TBytes, a dimensão máxima de uma coluna é de 1 GByte, ao tamanho máximo de linhas por tabela é ilimitado, a medida máxima de colunas por tabela varia entre 250 e 1600 (PostgreSQL, 2018). Muitos rasters são de tamanhos grandes, com quantidade de dados na escala de Megabytes ou Gigabytes. Os tipos *raster* e os de geometrias vetoriais (ponto, linha e polígono) usam a estrutura de indexação denominada de *Generic Index Structure* (GIST).

Através da SQL, que é uma linguagem de definição e manipulação de dados, é possível modificar esquemas e remover tabelas, criar índices, definir restrições de integridade, consultar, inserir, modificar e remover dados no banco de dados (PostgreSQL, 2018).

Por meio de carga dinâmica, o servidor PostgreSQL tem a capacidade de incorporar código escrito pelo usuário, ou seja, o usuário pode criar um arquivo contendo código objeto (por exemplo, uma biblioteca compartilhada), implementando um novo tipo de dado ou função, e o PostgreSQL executa a carga desse módulo, quando requisitado. O código escrito em SQL é ainda mais simples de ser adicionado ao servidor. Essa capacidade de modificar sua operação em tempo de execução torna o PostgreSQL especialmente adequado para a prototipação rápida de novas aplicações e estruturas de armazenamento.

Segundo PostgreSQL (2018), o SGBD-OR armazena informações sobre tipos de dados, funções, métodos de acesso, etc. Essas tabelas podem ser modificadas pelo usuário e, uma vez que as operações do PostgreSQL estão baseadas nessas tabelas, significa que pode-se estender adicionando novos tipos de dados, funções, operadores, métodos de índice e linguagens procedurais, além do fato de ser de código aberto.

O PostGIS foi desenvolvido com ferramentas de interface para usuário com suporte à topologia, validação de dados, transformação de coordenadas geográficas e interface de programação de aplicativos (APIs), onde também há suporte de dados

raster, redes e roteamento, superfície tridimensional e curvas de níveis (POSTGIS, 2018). A extensão denominada de WKT *Raster*, do PostGIS, atualmente denominado PostGIS *Raster,* implementa uma propriedade *raster* como sendo um tipo de dados *geometry* e disponibiliza um conjunto de funções SQL que operam de forma integrada entre vetores e *raster*.

Dentre outras soluções para utilização de SGBD-OR espaciais semelhantes ao PostGIS, destacam-se a da tecnologia ArcSDE (*Spatial Data Server*) da *Environmental Systems Research Institute* - ESRI (2016) e a extensão espacial *Oracle Spatial* do SGDB *Oracle* (CASANOVA *et al.,* 2005; POSTGIS, 2018).

No PostGIS os *rasters* são geralmente divididos em blocos de *pixels* também chamados *tiles* para um melhor desempenho de acesso. Em uma cobertura *raster* PostGIS, cada *tile* é armazenado como um registro na tabela que contém os dados binários. Cada *tile raster* possui especificações de tamanho do *pixel*, largura e altura, georreferenciamento, número de bandas, tipo de *pixel* por banda e valor de *pixel nodata* por banda, as quais são essenciais para realizar operações básicas de *Geograph Informations System (*GIS) *raster* (OBE *et al.,* 2009).

Uma camada *raster* é regularmente bloqueada quando todos os *tiles* têm a mesma largura e altura e estão corretamente alinhados em um *grid* de montagem do tamanho de um *tile*, sem sobreposição e sem intervalo. Dentro de uma cobertura de quadriculação irregularmente bloqueado, *tiles* podem ser dispersos em qualquer lugar, podem se sobrepor e não ter necessariamente a mesma largura e altura. Isso acontece quando se carrega muitos *rasters,* formando um mosaico de sobreposição, ou quando se "*rasteriza"* uma camada de vetor usando um *raster* por geometria (OBE e HSU, 2009).

O PostGIS *Raster* suporta armazenamento de *raster* dentro e fora do banco de dados. Quando o dado *raster* é armazenado dentro do PostGIS pode-se optar por armazenar um arquivo de imagem completa em um único registro, em uma única coluna, ou cortar o arquivo da imagem em *tiles* e armazenar cada *tile* como um registro separado. Os dados não estariam no formato binário *raster* original do qual ele veio, mas convertido para um PostGIS nativo, formando-se o *raster* adequado para a manipulação pelas funções *raster* do PostGIS.

Ao armazenar *raster*s e todos os dados de *pixels* correspondentes no banco de dados, obtêm-se as seguintes vantagens: o dado *raster* fica armazenado no banco de dados; ocorrem validações mais frequentes do que o armazenado fora do banco

de dados; e há garantia da integridade transacional do banco de dados durante a edição de *raster*, leitura rápida de dados, agregação e vetorização. A desvantagem está nos grandes tamanhos dos *raster*s que farão com que os *backups* e restaurações demorem mais tempo.

Existe a opção de armazenar-se apenas a extensão geográfica associada com *rasters*, ou *tiles*, mantendo todos os dados de *pixels* correspondentes fora do banco de dados como arquivos *rasters* em sistema de arquivos do tipo tiff, jpeg ou qualquer formato suportado pela *Geospatial Data Abstraction Library* – GDAL (GDAL, 2018). Os caminhos para sistema de arquivos das rasterizações são armazenados na base de dados juntamente com as informações georreferenciadas. Assim, podem-se consultar os valores de *pixel* com acesso indireto, proporcionando uma maneira fácil de catalogar e indexar os arquivos *raster*. Pode-se ainda compartilhar as varreduras com outras aplicações que precisam ler o dado *raster* a partir do banco de dados. O banco de dados com apenas metadados é menor e mais fácil de fazer *backup* (OBE e HSU, 2011).

Algumas das desvantagens de armazenamento de *raster* fora do banco de dados é que não se tem os benefícios transacionais do banco de dados. Os *rasters* podem ficar excluídos ou movidos para além do banco de dados, o que tornará a base de dados registrada inútil. Precisa-se ter certeza ainda do processo do servidor PostgreSQL poder acessar os arquivos e que a definição do caminho está em uma forma relativa ao servidor (OBE e HSU, 2011).

Um dado *raster* armazenado fora do banco de dados pode ser perfeitamente trabalhado com um *raster* dentro do banco de dados porque funções SQL WKT *Raster* permitem essa comunicação. O WKT *Raster* insere o conceito de objetos *rasters* com características geográficas armazenadas com *tiles* de tamanhos variados ao invés de polígonos e conversão de vetor para *raster* sem perda de informação. A título de exemplo, uma linha é um *tile* que por sua vez é um *raster*; uma tabela é uma camada *raster*; sem metadados; sem máscara, sem múltiplas dimensões, apenas *x* e *y*, mas sem confundir com bandas, pois o WKT *Raster* suporta multibandas *raster* sem pirâmide (OBE *et al.*, 2009).

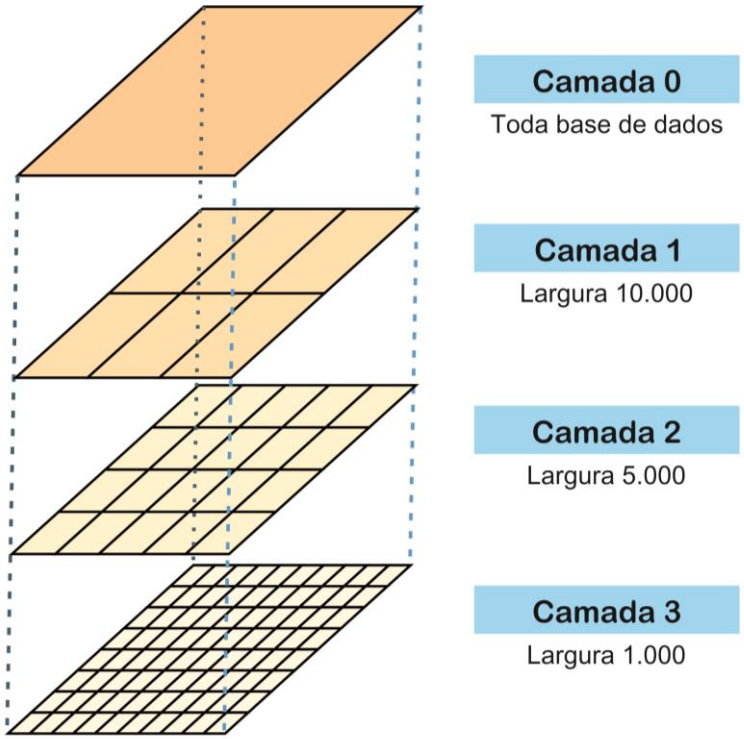
Em geral, as características do PostGIS *Raster* incluem (RAMSEY, 2017):

- ✓ Objetos espaciais adequados (ponto, linha, polígono, multiponto, multilinha, multipolígono, geometria);
- ✓ Indexação espacial (R-árvore com GIST);

- ✓ Funções analíticas simples (área, comprimento, distância);
- ✓ Predicados espaciais como f(geom1, geom2) = V/F, relações entre campos geométricos, intersecciona x disjunto; contém, está contido (dentro), sobrepõe, toca e cruza;
- ✓ Operadores espaciais;
- ✓ Armazenamento de metadados do sistema;
- ✓ Suporte de re-projeção de coordenadas (via Proj4);
- ✓ Ferramentas de importação e exportação de dados.

Portanto, o processo de armazenamento de dados *raster* em *tile* consiste em, conhecendo-se a priori os limites geográficos da base de dados, criar camadas de "*tiles*" de dimensões fixas. A quantidade de camadas e a largura dos *tiles* de cada camada são definidas pelo usuário, de modo a representar a diversidade de dimensões dos objetos geográficos (Figura 11). Cada objeto associa-se a um *tile*, que será o menor *tile* que contiver inteiramente o objeto, formando uma lista. No momento da visualização, testam-se os limites da janela de visualização contra a lista de *tiles*, selecionando aqueles que tenham pelo menos uma extremidade contida no retângulo. Os objetos associados a cada um desses *tiles* serão, então, recuperados na base de dados para apresentação. Com esses limites previamente calculados, preserva-se de percorrer frequentemente uma estrutura de dados.

Figura 11: Exemplo de *raster* em *tiles*



Fonte: Adaptado de Câmara *et al.* (2005).

2.4. Biblioteca TerraLib

A TerraLib é uma biblioteca de classes e funções de Sistema de Informações Geográficas (SIG) que permite um ambiente colaborativo, bem como seu uso para o desenvolvimento de várias ferramentas SIG e está disponível na *internet* como código aberto. Atualmente, a TerraLib está sendo desenvolvida pela Divisão de Processamento de Imagem do Inpe, o Grupo de Tecnologia de Computação Gráfica da PUC-Rio (Tecgraf) e a Fundação para a Ciência Espacial, Pesquisa Aplicada e Tecnologia (Funcate).

A TerraLib permite o desenvolvimento rápido de aplicações geográficas customizadas baseadas em bancos de dados geográficos. Como uma ferramenta de pesquisa, a TerraLib tem como objetivo fornecer um ambiente rico e poderoso para o desenvolvimento de pesquisas em SIG, permitindo o desenvolvimento de protótipos que incluem novos conceitos como modelos de dados espaço-temporais, ontologias geográficas e técnicas avançadas de análise espacial. A TerraLib define um modelo de dados geográficos e fornece suporte para este modelo em uma série de diferentes SGBDs (MySQL, PostgreSQL, Oracle e Access) e é implementado

como uma biblioteca de classes e funções da linguagem C++, que é escrita em ANSI-C++ (TERRALIB, 2017).

A linguagem C++ contém a biblioteca *Standard Template Library* (STL), classes parametrizadas e programação multi-paradigma (STROUSTROUP, 2013), e a biblioteca TerraLib é compilada usando amplamente os mecanismos das funcionalidades da linguagem C++.

Os códigos da biblioteca TerraLib dependem de um conjunto de pacotes de *softwares* de códigos abertos que são utilizados para implementarem suas funções de núcleo, e também para fornecerem suporte aos dados de imagens no PostgreSQL/PostGIS, tendo-se como exemplo a classe *Datasource* da TerraLib que contempla o acesso via PostGIS para dados *raster* e vetoriais.

Os *drivers* de *softwares* de terceiros utilizados na TerraLib são, geralmente, disponibilizados com a biblioteca. Porém, o desenvolvedor de aplicativos para a biblioteca TerraLib deve se apropriar em conhecer esses pacotes ao desenvolver e compilar os algoritmos ao utilizar a biblioteca, bem como considerar as questões de compatibilidade entre a versão da biblioteca TerraLib e as versões de pacotes de *softwares* dependentes.

A biblioteca TerraLib segue um paradigma de programação genérica para o desenvolvimento de bibliotecas reutilizáveis, através da linguagem de programação orientada a objetos C++. A programação genérica é voltada para a construção de algoritmos que independem de um determinado tipo ou estrutura de dados e refere-se a um tipo de abstração no processo de desenvolvimento de *software* e que, no entanto, sejam tão eficientes quanto se construídos acoplados a uma determinada estrutura.

O estilo de desenvolvimento de programação genérica se concentra em encontrar a semelhança entre implementações semelhantes de um algoritmo e disponibilizar abstrações adequadas, permitindo que um único algoritmo genérico seja usado para realizar inúmeras implementações concretas.

Em SIG, tal estilo de programação é extremamente importante, pois um operador precisa lidar com vários conjuntos de dados e procedimentos para realizar uma única tarefa. Por exemplo, vários procedimentos para medir a autocorrelação espacial podem ser adotados para um conjunto de pontos, um conjunto de polígonos, um *Triangulated Irregular Network* (TIN), uma grade ou uma imagem de sensoriamento remoto (IMRAN, 2009). A definição de uma interface de classe é o

resultado da interação com outras classes através das variáveis e métodos que são fornecidos. As outras classes executam a interface de uma classe provendo estrutura, ou seja, dados e estado, e implementações concretas de métodos, isto é, disponibilizando um código que especifica como os métodos funcionam. Um padrão de *design* é uma descrição ou modelo para resolver um problema geo-computacional.

A biblioteca TerraLib adota, no seu desenvolvimento, a programação genérica de acordo com os padrões de *design* descritos por Gamma *et al.* (1995). A seguir, são discutidos os padrões de projeto que foram adotados pelas classes de bibliotecas da TerraLib e os diagramas genéricos modificados.

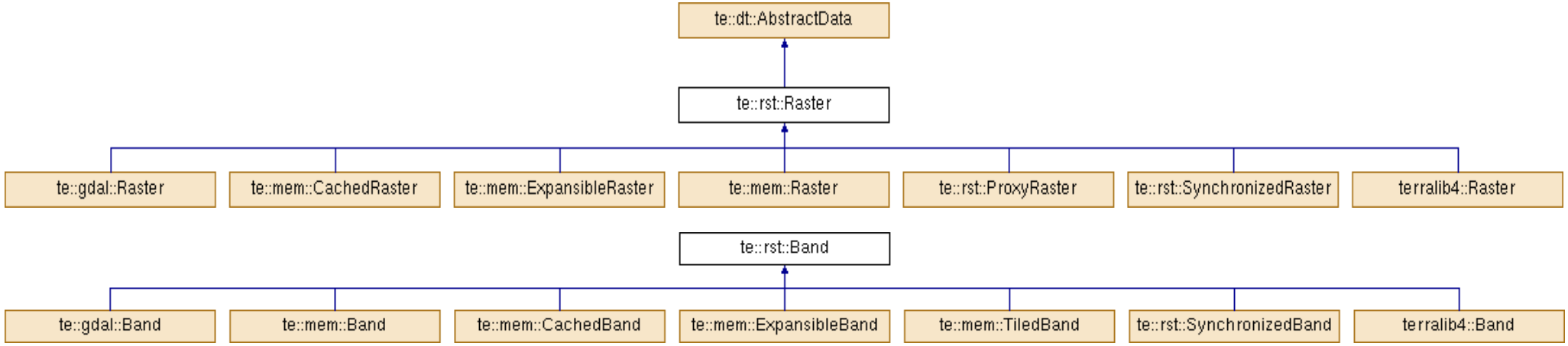
a) Classe de modelo

A linguagem de programação C++ possui um recurso denominado de classe de modelo que permite que as classes operem com tipos genéricos. Os modelos podem ser usados em conjunto com tipos de dados abstratos para permitirem que eles manipulem qualquer tipo de dados. A capacidade de ter uma única classe que pode lidar com diversos tipos de dados diferentes significa que o código é mais fácil de manter e torna as classes mais reutilizáveis.

O tipo de uma classe é definido através do fornecimento de parâmetros em tempo de execução. Isso permite que uma classe possa ser trabalhada em muitos tipos de dados diferentes sem ser reescrito para cada um. O STL, que é uma parte fundamental do C++, tem como objetivo a generalização de estruturas de dados como lista e vetor e iteradores para que possam ser usados para qualquer tipo de dado ou algoritmo fornecido em tempo de execução.

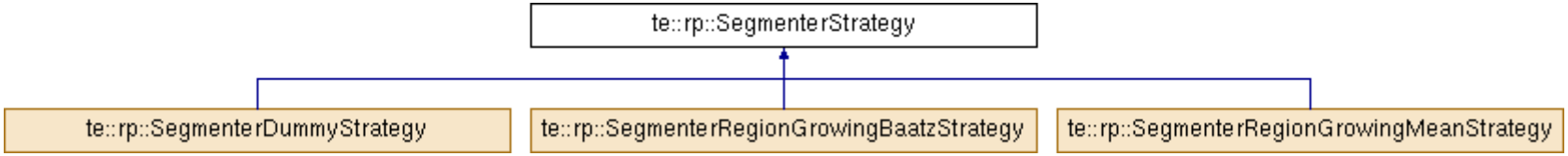
A biblioteca TerraLib adotou a abordagem de classe modelo para definir, por exemplo, um iterador genérico que pode percorrer uma estrutura de dados como uma estrutura genérica de dados de imagem independentemente do formato. A biblioteca TerraLib também usa modelos de classe para definir subtipos de recipientes genéricos. A título de exemplo, nas Figuras 12 e 13, são mostradas as estruturas das classes *raster e* bandas, e de segmentação.

Figura 12: TerraLib - classe abstrata para estruturas de dados *raster* e bandas



Fonte: http://www.dpi.inpe.br/terralib5/codedocs_5.2.1/d6/de2/classte_1_1dt_1_1AbstractData.html#a7acb2108a23406b49a3673330426b588 (TERRALIB, 2017).

Figura 13: TerraLib - classe das estruturas de segmentação



Fonte: http://www.dpi.inpe.br/terralib5/codedocs_5.2.1/db/d47/classte_1_1rp_1_1SegmenterStrategy.html (TERRALIB, 2017).

b) Método do modelo

No conceito de orientação a objeto, a sobrecarga de funções é quando o código de uma função é repetido, com as mudanças pequenas para diferentes parâmetros variando no tipo de dado. A funcionalidade do método do modelo (*template*), na abordagem de programação genérica, é determinada de forma que seja adaptada ao tipo de dados de seu(s) parâmetro(s), excluindo a exigência de reiterar o código genérico.

A generalização que um método do modelo cria para o comportamento de uma função, ou algoritmo, pode ser realizada através da substituição de funções (IMRAN, 2009). Em orientação a objetos, a substituição de funções é uma concepção em que uma nova versão de um método pode ser iniciada por uma extensão de uma classe, com alterações sutis na ação de um método na classe pai. Segundo Gamma *et al.* (1995), um método de modelo consiste em "definir o esqueleto de um algoritmo em uma operação, adiando algumas etapas para subclasses. O método *template* permite que subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do algoritmo".

A definição de herança, em orientação ao objeto, é combinada com a concepção de modelo genérico na programação genérica para estabelecer padrões de *design* reutilizáveis, e a biblioteca TerraLib utiliza-se deste recurso. Então, é determinado um padrão de projeto como modelo, após análise de um impasse geo-computacional e é implementado em duas fases:

- ✓ Uma classe base é a padronização de um algoritmo tido como modelo genérico. Os componentes invariáveis de um algoritmo são implementados com uma classe base.
- ✓ A classe especializada é estabelecida pela extensão da classe base e desempenha os comportamentos variantes.

Na TerraLib, a classe *raster* é uma estrutura de dados *raster* genérica e as operações de varredura genéricas são obtidas de uma instância dessa classe. Essa classe permite estabelecer ou adquirir parâmetros de varredura, como tipo de mosaico e tamanho de bloco.

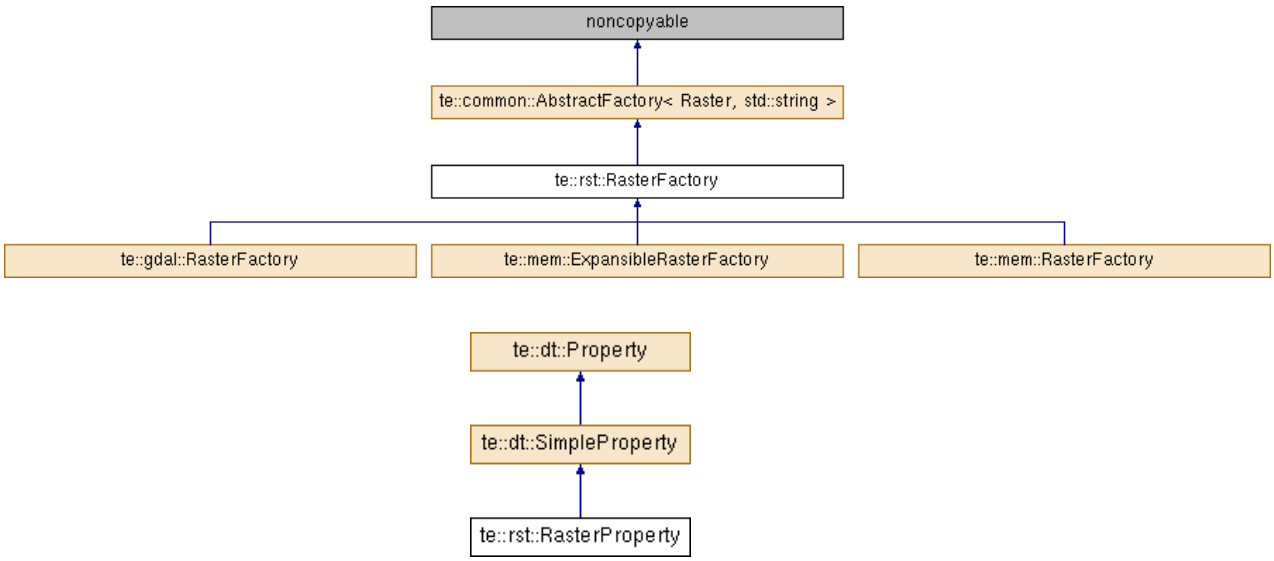
O módulo *raster* disponibiliza a interface básica das classes de manipulação de dados geográficos na forma de dados matriciais especializados, tais como imagens de sensoriamento remoto. Os *drivers,* denominados de implementações desse módulo, são especializados nas manipulações de cada

formato de imagem e sistema de armazenamento. A Figura 14 exemplifica duas classes desse módulo.

Ao importar uma imagem no PostgreSQL, os registros são inseridos na tabela e índices espaciais são estabelecidos para gerenciar o acesso aos dados. As tabelas de imagens importadas, sendo uma tabela para cada imagem importada, serão praticamente estáticas, não havendo problema de desempenho.

Opcionalmente, podem-se importar as imagens como bandas de uma camada *raster*. Nesse caso, o número de tabelas não cresce, porém, as tabelas crescem quando uma nova imagem for inserida como uma banda *raster*. O administrador de banco de dados deverá adotar medidas de gerenciamento de repositório de banco de dados em função do crescimento constante do esquema. Pode-se adicionar o local do diretório da imagem no sistema de arquivos como entrada, criando no banco de dados PostgreSQL, a tabela correspondente, inserindo cada imagem como uma camada separada.

Figura 14: Exemplos de classes do módulo *RasterFactory* e *RasterProperty* na biblioteca TerraLib.



Fontes:
http://www.dpi.inpe.br/terralib5/codedocs_5.2.1/d0/d2c/classte_1_1rst_1_1RasterFactory.html#a55df74b14e750427d963e23e7e602aa7 (TERRALIB, 2017).
e http://www.dpi.inpe.br/terralib5/codedocs_5.2.1/d2/def/classte_1_1rst_1_1RasterProperty.html (TERRALIB, 2017).

Alternativamente, pode-se importar imagens para toda uma área geográfica por meio de um mosaico de imagem para uma única camada *raster* no banco de dados PostgreSQL. Porém, um mosaico de imagens produzirá somente três tabelas dentro do banco de dados, seja qual for a quantidade de imagens importadas para uma camada do banco de dados. Nesse caso, as tabelas serão dinâmicas e proliferarão em cada importação de imagens para a camada mosaico, influenciando no desempenho do banco de dados, e o administrador de banco de dados deverá verificar mecanismos de performance.

O PostgreSQL oferece duas formas de armazenar grandes objetos com relação a cada requisito que é necessário atender: o Bytea e o armazenamento de *Large Objects* (LO). A implementação Bytea é utilizada para armazenamento de objetos com grandes dimensões. É bastante semelhante ao Varchar, mas com características bem distintas, como o armazenamento de dados brutos ou não-estruturados, além de permitir o armazenamento de valores nulos ou até 1 GB de dados.

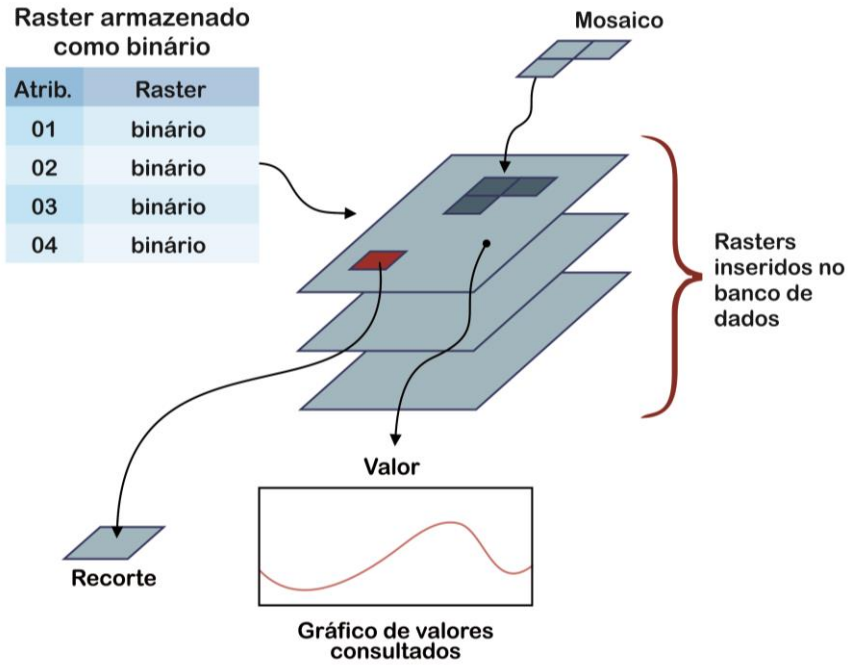
O LO permite o armazenamento de até 4 TBytes, além de oferecer funções que ajudam melhor e mais facilmente na manipulação de dados externos. Por não ser um tipo de dado, mas uma entrada, os LOs são armazenados na tabela de sistema chamada *pg_largeobject* e são partidos em pedaços de tamanho padrão (*tiles*) e armazenados como linhas na base de dados.

Como os LOs não são armazenados nas tabelas do usuário, é criado um *object identifier* (OID) para ser armazenado, ou seja, no momento em que é preciso acessar esses dados, deve-se referenciar o OID que aponta para os registros na tabela pg_largeobject (POSTGIS, 2016). Na Figura 15, tem-se a representação do *raster* no banco no PostGIS.

Um exemplo de aplicação utilizando-se da biblioteca TerraLib é o projeto TerraAmazon que é uma ferramenta de SIG projetada para ser um editor multi-usuário de dados geográficos em tempo real. Possui ferramentas de classificação de uso e cobertura do solo, assim como operações espaciais entre dados vetoriais e mantém informações de tempo de trabalho, visando à gestão de projetos. Uma camada *raster* para toda a região Amazônica foi criada para a aplicação. Suas funcionalidades são extensíveis por meio de *plug-ins*, como o TerraImage (Processamento Digital de Imagem) e TerraPrint (plotagem), que são fornecidos juntamente com o TerraAmazon. Os dados do TerraAmazon são

armazenados em um banco de dados PostgreSQL e modelo TerraLib (TerraAmazon, 2017). No TerraAmazon, imagens podem ser consultadas através de uma instrução SQL direta em tabelas de dados de imagem, ou através de visão (*view*), e obter métodos em um iterador fornecido pela biblioteca TerraLib em uma estrutura de dados de imagem.

Figura 15: Representação de *raster* no PostGIS.



Fonte: PostGIS (2017).

A *Procedural Language Extensions To* SQL ou PL/pgSQL é uma linguagem estruturada estendida da *Structure Query Language* (SQL) disponibiliza comandos que auxiliam as tarefas de programação no sistema gerenciador de banco de dados PostgreSQL. Ela incorpora à linguagem SQL propriedades procedurais e estruturas de controle (PostgreSQL, 2017).

Através de instrução SQL direta em tabela de dados de imagem, pode-se consultá-la no nível de bloco ou *tile*, sendo que o tamanho é decidido quando houver a inserção da imagem. Na consulta, podem-se utilizar os índices espaciais. Outra forma de consulta é através de obtenção de métodos em um iterador fornecido pela biblioteca TerraLib dentro de uma estrutura de dados de imagem. Nesse caso, o desenvolvedor pode acessar dados no nível atômico e um único *pixel* pode ser selecionado e operado explicitamente. O iterador genérico percorre a imagem quando da seleção dos valores de *pixel* para um

algoritmo independente do formato *raster*. O resultado de uma consulta, como dados de varredura (*raster*), pode ser disponibilizado na forma de algoritmo estatístico ou processamento de imagem. O resultado de um algoritmo pode ser tratado na memória como:

- ✓ Disponibilização como entrada para novo algoritmo;
- ✓ Arquivamento no formato de uma imagem no disco ou no banco de dados; e
- ✓ Armazenamento como um registro de banco de dados.

A TerraLib tem um módulo denominado *data access* que possibilita que as aplicações manipulem dados espaciais em diferentes tipos de fontes de dados, abstraindo desde os tradicionais SGBDs até os serviços web da OGC (INPE, 2018). Grande parte das classes dessas camadas são abstratas, isto é, elas definem apenas a interface que deve ser implementada por *drivers* específicos que entendam os detalhes de cada tipo de fonte.

.

CAPÍTULO 3
PROPOSTA DE APLICAÇÕES PARA SEGMENTAÇÃO NO POSTGIS RASTER, INTEGRAÇÃO COM O INTERIMAGE E CLASSIFICAÇÃO OBIA

Neste capítulo é apresentada a proposta de implementação de uma *Application Programming Interface* (API) no programa InterIMAGE Desktop 1.43 para realizar a segmentação OBIA, conceituada por Baatz e Schäpe (2000), no ambiente PostgreSQL/PostGIS *Raster*, e realizar a exportação dos resultados via API para o InterIMAGE Desktop 1.43.

Este capítulo está organizado da seguinte forma: Na Seção 3.1, apresentam-se os dados e *softwares* utilizados; na Seção 3.2, apresenta-se o método do algoritmo de processamento de segmentação Baatz e Schäpe (2000), implementado com Visual Studio C++ na biblioteca TerraLib 5.2.2 e InterIMAGE 1.43, com integração no PostgreSQL/PostGIS *Raster*.

3.1 Materiais

Neste estudo utilizou-se para os procedimentos de processamento uma imagem do satélite GeoEye-1, da cidade de Goianésia - GO, obtida em 09 de julho de 2013, com quatro bandas multiespectrais, com fusão das imagens multiespectrais e pancromática resultando na resolução espacial de 0,5 metros, com faixa de imageamento de 25 km² e tamanho de arquivo de 3,08 GB (Figura 16).

3.2 Métodos

Apresenta-se o método proposto para a segmentação de imagens no sistema gerenciador de banco de dados PostgreSQL/PostGIS e utilização via API pelo programa InterIMAGE Desktop 1.43.

A solução desenvolvida considera a técnica de segmentação Baatz e Schäpe (2000) com execução do algoritmo tendo a imagem (*raster*), de alta resolução espacial, armazenada dentro de um sistema gerenciador de banco de dados objeto-relacional PostgreSQL/PostGIS.

Figura 16: Composição colorida RGB, Red (banda 1) Green (Banda 2) Blue (Banda 3), das bandas obtidas nas faixas espectrais do verde, vermelho e infravermelho próximo da imagem GeoEye-1 da área de estudo.

Fonte: Elaborado pela autora.

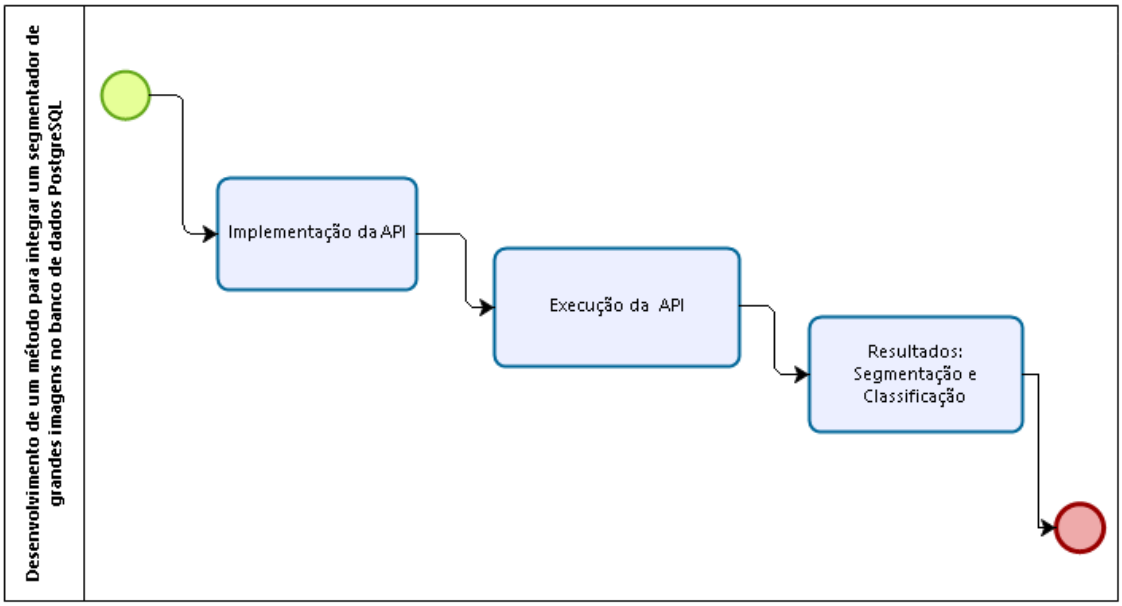
Implementaram-se as técnicas a partir dos estudos sobre segmentação envolvendo o conceito de GEOBIA utilizando-se a linguagem C++ no banco de dados geoespacial PostgreSQL/PostGIS. A configuração do ambiente de implementação consistiu na instalação e configuração dos *softwares* listados no Apêndice A.

O *framework* InterIMAGE é implementado com um repositório de operadores chamado TerraAIDA, definidos como classes e funções de *software* fornecidas pela biblioteca TerraLib, e funcionam como programas externos por seu mecanismo de controle.

O InterIMAGE 1.43 utiliza em sua compilação a biblioteca TerraLib4, e na construção do protótipo da API Segmentação em PostgreSQL/PostGIS *Raster*, foi necessária uma adaptação do ambiente de programação para a TerraLib 5.2.2, uma vez que a nova biblioteca fornece as abstrações necessárias otimizadas para manipulação de dados geoespaciais através de dois de seus módulos: *raster* e *data access* (INPE, 2018), e outro fator preponderante deu-se pela descontinuidade de manutenção da TerraLib4 por parte do INPE.

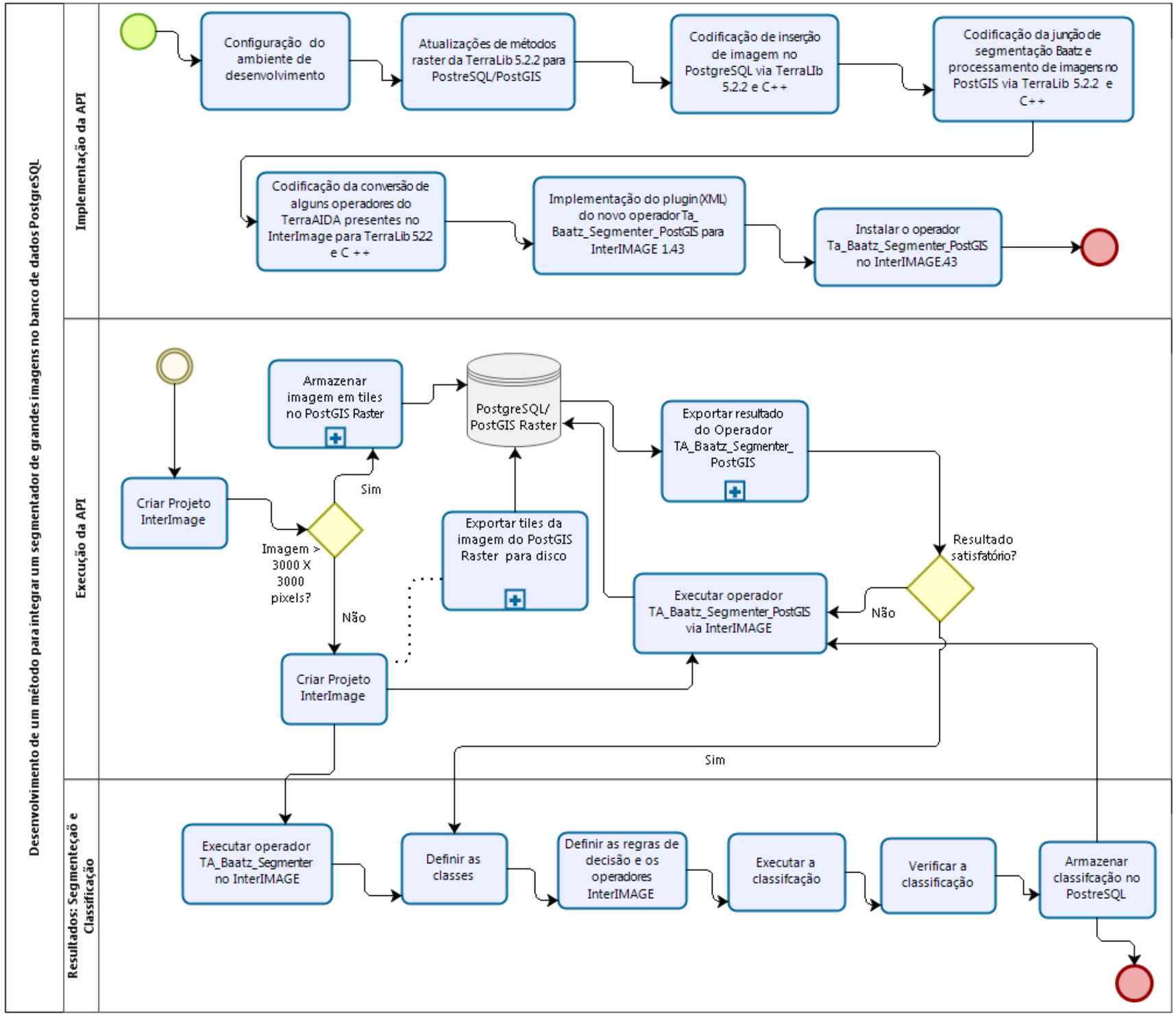
Para alcançar os resultados propostos, duas etapas metodológicas foram necessárias e uma de resultados: fluxo geral (Figura 17) e detalhamento da implementação e execução da API (Figura 18). Nos próximos subtítulos serão explanados cada subetapa de forma detalhada.

Figura 17: Fluxograma de implementação da API para segmentação no PostgreSQL/PostGIS *Raster* e posterior exportação no InterIMAGE Desktop.



Fonte: Elaborado pela autora.

Figura 18: Fluxograma de execução da API Segmentação no PostgreeSQL/PostGIS *Raster*.



Fonte: Elaborado pela autora.

3.2.1 Implementação da API

Os arquivos das implementações da API estão todos nos Apêndices de A à F, e no endereço https://github.com/orgs/ifto-palmas/ estarão disponíveis os códigos fontes.

3.2.1.1 Configuração do ambiente de desenvolvimento

A arquitetura tecnológica para implementar a API comtemplou a estrutura física de um notebook ACER, modelo Aspire F 15, Intel® Core™ i7-7500U 2.7 GHz com Turbo Boost up to 3.5 GHz, memória 16 GB DDR4, disco rígido de 2000 GBytes, e placa de vídeo NVIDIA® GeForce® 940MX com 4 GBytes VRAM dedicada.

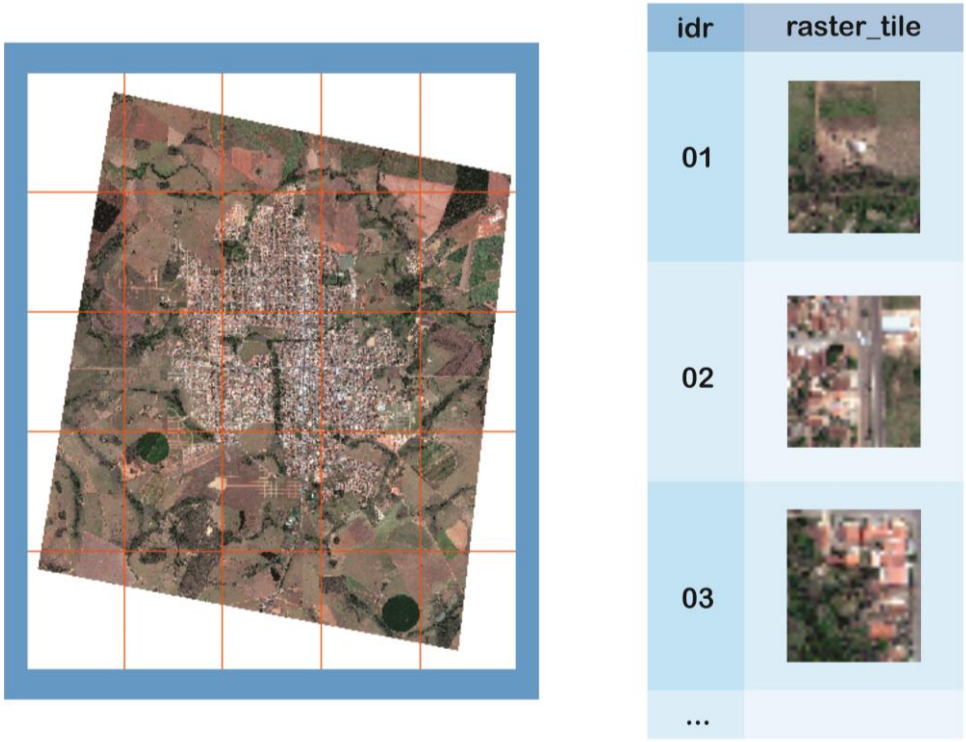
Os principais *softwares* necessários às implementações dos aplicativos, no ambiente do sistema operacional Windows 10, foram: Microsoft Visual Studio C++ 2013, PostgreSQL versão 9.6.5-1, PostGIS versão 2.3.2, TerraLib 5.2.2 e InterIMAGE versão1.43. E, a listagem completa com maiores detalhamentos estão no Apêndice A.

3.2.1.2 Atualização de métodos de manipulação de banco de dados na TerraLib 5.2.2

Para atingir o objetivo de processar a segmentação com a imagem armazenada em banco de dados, houve a necessidade de atualizar a TerraLib 5.2.2, que consistiu em habilitar a classe *Raster* para poder ser manipulada no sistema gerenciador de banco de dados PostgreSQL/PostGIS *Raster*, e foi realizada principalmente pela equipe de desenvolvimento do INPE.

Para o processamento de armazenamento da imagem no PostgreSQL/PostGIS *Raster* foi realizada a divisão da imagem em *tiles*. Segundo Sample e Ioup (2010), a padronização de dimensões horizontais e verticais em partes iguais e o uso de potência de dois descomplicam os cálculos no processamento da divisão em *tiles*. Na Figura 19, é ilustrada a forma de armazenamento da imagem em *tiles* como tabela no sistema gerenciador de banco de dados.

Figura 19: Exemplo de subdivisão da imagem GeoEye-1, da área de estudo, em *tiles* de 1.024 *pixels* x 1.024 *pixels*.



Fonte: Elaborado pela autora.

Os algoritmos de criação de *tiles* utilizam os seguintes procedimentos (Sample, 2010):

1. Escolha do nível de base para o conjunto de *tiles*;
2. Determinação dos limites geográficos do conjunto de arquivos, com base nos limites das imagens de origem;
3. Determinação dos limites do conjunto de blocos nas coordenadas do bloco;
4. Inicialização do mecanismo de armazenamento do bloco;
5. Iteração sobre as coordenadas do conjunto de mosaicos. Para cada bloco, faz-se o:
   a. Cálculo dos limites geográficos do bloco específico.
   b. Iteração sobre as imagens de origem. Para cada imagem de origem, faz-se a:
      i. Determinação se a imagem de origem específica cruza o bloco que está sendo criado.
      ii. Se a imagem de origem e o bloco se cruzarem:

a. Verificação do cache para a imagem de origem. Se não estiver no cache, armazenamento no disco e no cache.
b. Extração dos dados de imagem necessários da imagem de origem e armazenamento na imagem lado a lado.

c. Armazenamento da imagem completada no mecanismo de armazenamento do bloco.

6. Limpeza do cache da imagem de origem.
7. Finalização do mecanismo de armazenamento do bloco.

3.2.1.3 Junção dos códigos de segmentação na TerraLib 5.2.2 e de métodos de manipulação de imagens no PostgreSQL/PostGIS *Raster*

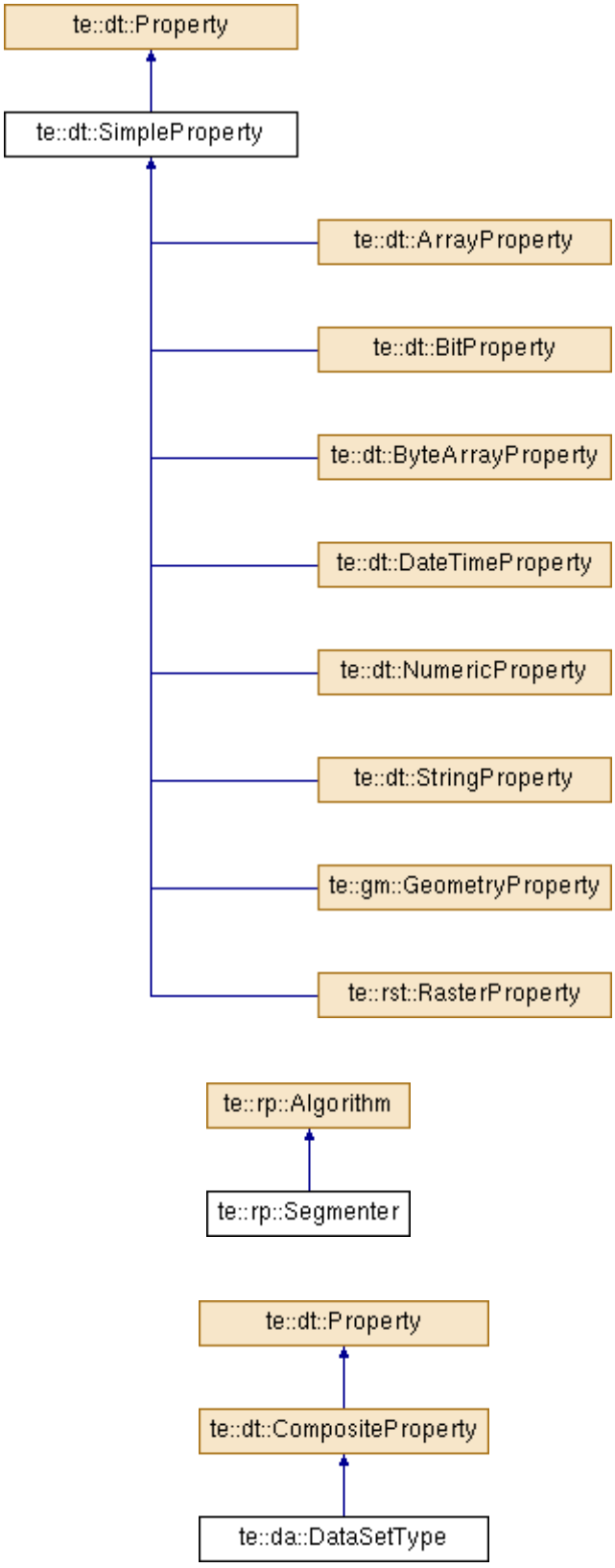
Em um primeiro momento foi desenvolvido o procedimento de inserção de imagem de sensoriamento remoto no PostgreSQL/PostGIS via TerraLib 5.2.2 e o Visual Studio 2013 C++.

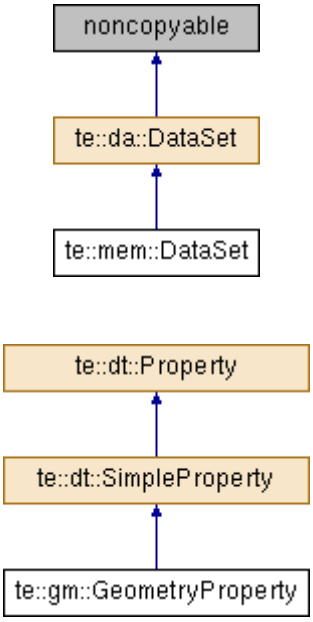
Em seguida realizou-se a compilação do algoritmo de segmentação acompanhado dos métodos de manipulação e processamento de imagem armazenada no PostgreSQL/PostGIS *Raster*. Este processo inclui, na chamada da função, a passagens de valores (parâmetros) de acesso ao banco de dados (usuário, senha, porta, tabela) e os de segmentação (número de bandas, peso das bandas, cor, escala e compacidade).

O operador *TA_Baatz_Segmenter* da TerraAIDA, presente no InterIMAGE 1.43, necessitou de atualização para poder ser executado na versão da TerraLib 5.2.2, com a adaptação do algoritmo para uso com o PostgreSQL/PostGIS *Raster,* tendo a implementação no Visual Studio 2013 C++.

Na prototipação da API classes, e os métodos, da TerraLib 5.2.2 foram utilizadas para implementação do ambiente de dados das simulações de processamento da segmentação, no PostgreSQL/PostGIS *Raster*, bem como a opção de copiar os resultados gerados no processamento para um repositório de dados, fora do PostGIS *Raster*, a serem indicadas pelo usuário. As principais classes utilizadas no método estão representadas na Figura 20.

Figura 20: Representação das classes utilizadas no método de segmentação





Fonte: http://www.dpi.inpe.br/terralib5/codedocs_5.2.1/df/d2b/group__rp.html (TERRALIB, 2017).

3.2.1.4 Implementação do plugin *TA_Baatz_Segmenter_PostGIS* (XML) no InterIMAGE 1.43

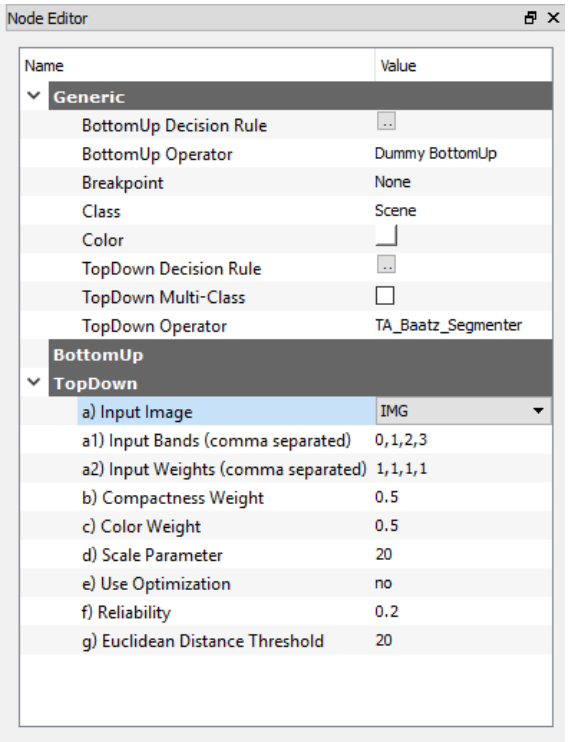
O operador *TA_Baatz_Segmenter* do InterIMAGE 1.43, executa o processamento da segmentação baseada no conceito de Baatz e Schäpe (2000), em que cada segmento gerado representa uma hipótese a ser analisada pelo próximo nó de rede semântica. Os parâmetros de entrada para o operador *TA_Baatz_Segmenter*, no InterIMAGE 1.43, devem atender aos requisitos do Quadro 2. Na Figura 21, tem-se a representação da tela de entrada de valores.

Quadro 2: Parâmetros de entrada do operador *TA_Baatz_Segmenter* no InterIMAGE.

Parâmetro	Tipo	Descrição	Valores Válidos	Nota
Imagem de entrada	tif	Introduz o nome do arquivo de imagem	Um nome de arquivo de imagem válido	O tipo de imagem deve ser suportado pelo segmentador
Entrada das bandas	*String*	Uma lista separada por vírgulas de canais/bandas de imagens de entrada utilizadas	De zero até o número de canais na imagem menos 1	Ex .: Uma imagem com 3 bandas Aceitar valores de 0 a 2. Exemplo: 0,1,2
Pesos de entrada	*String*	Uma lista separada por vírgulas dos pesos de canais/bandas de entrada usados		Exemplo: 0.1,2.4,1.3
Compacidade Peso	Ponto	Atributo de compacidade Baatz	(0, 1)	Quando o critério forma é superior a

	flutuante			zero, pode-se determinar se os objetos serão mais compactos ou suavizados
Cor Peso	Ponto flutuante	Atributo de cor Baatz	(0, 1)	
Parâmetro de escala	Ponto flutuante	Atributo da escala Baatz	Qualquer valor real positivo	Determina a máxima heterogeneidade permitida para os objetos, influencia o tamanho médio dos objetos gerados
Uso otimização	Booleana	Divide ou não a imagem em pedaços e segmenta cada peça individualmente		Padrão: *No*
Confiabilidade	Ponto flutuante	Maior prioridade será dada aos nós com pesos mais altos nos casos em que existam sobreposições geográficas	(0, 1)	
Distância Euclidiana	Ponto flutuante	A distância euclidiana mínima entre cada segmento	Maior que 0	Esse parâmetro é necessário para mesclar segmentos de bloco adjacentes quando a opção de otimização está habilitada

Figura 21: Tela de entrada do operador *TA_Baatz_Segmenter* no InterIMAGE



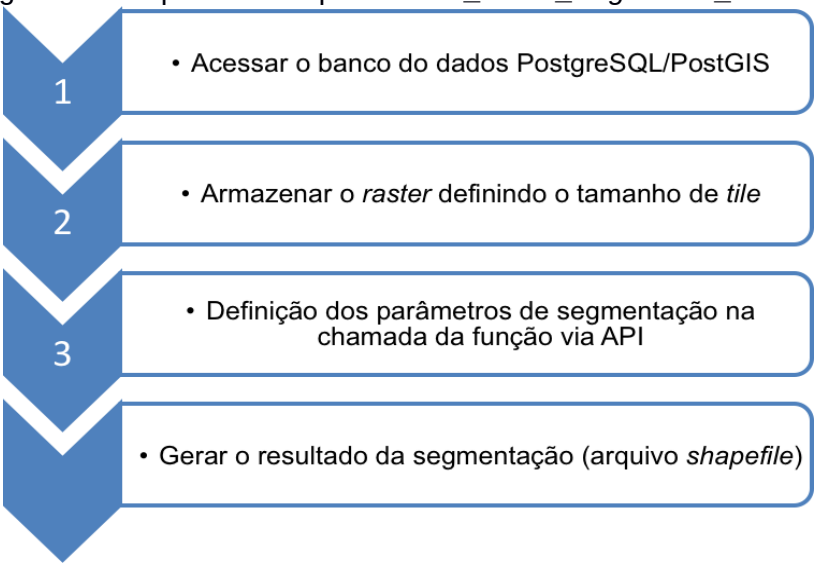
Fonte: Elaborado pela autora.

Como proposta de solução para segmentação de imagens (*raster*) com tamanhos superiores a 3.000 x 3.000 *pixels*, para o *software* InterIMAGE 1.43, implementou-se uma adaptação do algoritmo de segmentação de Baatz e Schäpe (2000). O algoritmo é parte de um conjunto de códigos exemplos da biblioteca TerraLib 5.2.2. O exemplo base de código utilizado é a classe *SegmenterStrategy*, no método *SegmenterRegionGrowingBaatzStrategy* e os parâmetros *SegmenterRegionGrowingBaatzStrategy::Parameters* para os elementos *m_bandsWeights*, *m_colorWeight*, *m_compactnessWeight*, *m_minSegmentSize*, e *m_segmenesSimilarityThreshold*, presentes no código implementado do Anexo D. No desenvolvimento foram acrescentados ao código elementos para o armazenamento da imagem de sensores remotos e o processamento da segmentação de dados *raster* no PostgreSQL/PostGIS *Raster*. E, denominou-se o código de *TA_Baatz_Segmenter_PostGIS*.

No processamento do algoritmo de segmentação, os primeiros resultados são armazenados como *raster* podem serem armazenados no PostGIS *Raster* em *tiles*, bem como em arquivos no disco rígido. O resultado da segmentação (*raster*) é então transformado em formato vetorial, como registros na tabela PostGIS *Raster*, e em seguida convertidos para o tipo *shapefile*, sendo exportados para disco rígido na pasta destinada pelo usuário, e estarão disponíveis para acesso no InterIMAGE 1.43. A sequência de processamento do operador *TA_Baatz_Segmenter_PostGIS* tem as seguintes premissas (Figura 22).

Um programa, na linguagem procedural PLPgSQL, foi desenvolvido para realizar a divisão da imagem de acordo com o sistema de referência de coordenadas e o tamanho dos *tiles* definidos pelo usuário.

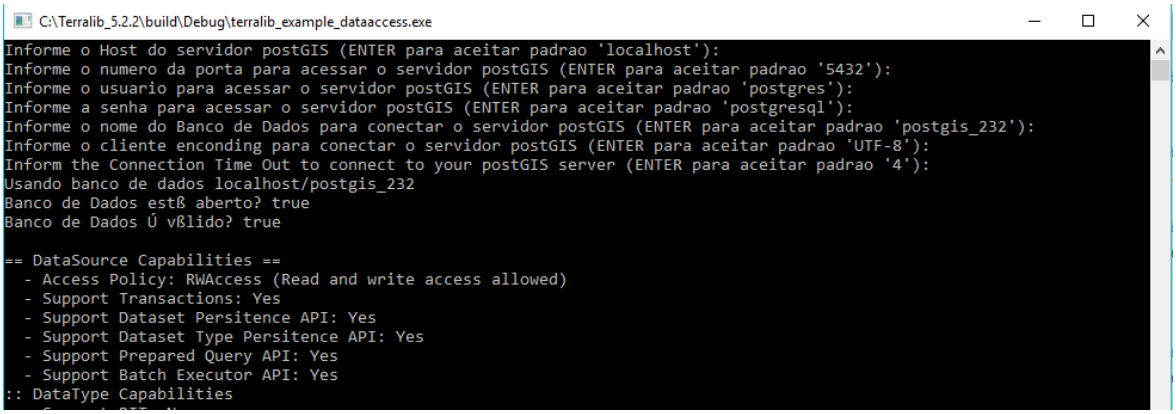
Figura 22: Sequência do operador *TA_Baatz_Segmenter_PostGIS*



Fonte: Elaborado pela autora.

Parte da execução do código de inserção do arquivo *raster* pode ser observada na representação da Figura 23, e a codificação completa encontra-se no Apêndice C.

Figura 23: Execução de entrada de dados *raster* no PostGIS via TerraLib



Fonte: Elaborado pela autora.

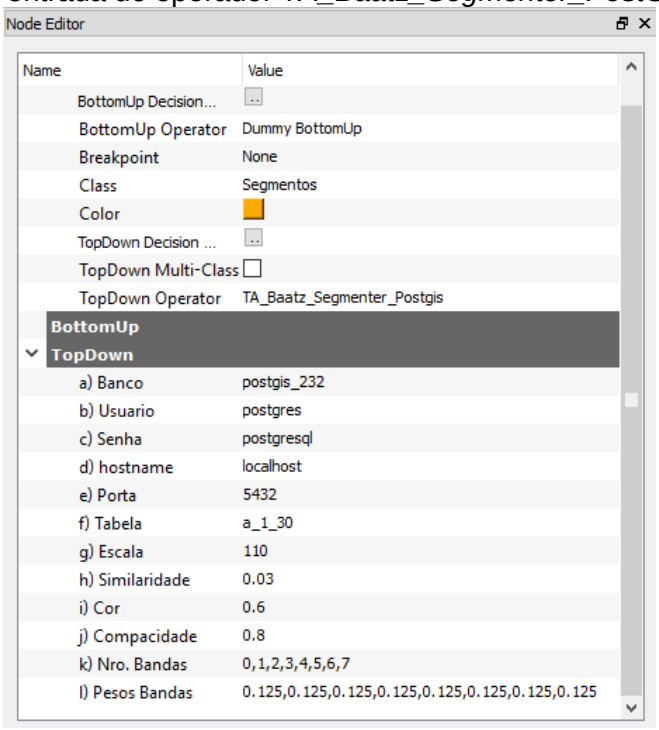
3.2.1.5 Instalar o plugin *TA_Baatz_Segementer_PostGIS* (XML) no InterIMAGE

No InterIMAGE qualquer operador precisa conter a descrição que determina como deve-se manuseá-lo. Essa descrição precisa ser codificada em arquivos XML com extensão *op* e colocada na pasta */share/data/operators/*,

dentro do diretório raiz do InterIMAGE. Basicamente, a descrição XML do operador diz ao sistema sobre como montar a linha de comando para executar o programa executável associado.

Na Figura 24, é mostrada a representação da tela de entrada de definição dos parâmetros do operador *TA_Baatz_Segmenter_PostGIS*, no *software* InterIMAGE.

Figura 24: Tela de entrada do operador *TA_Baatz_Segmenter_PostGIS* no InterIMAGE



Fonte: Elaborado pela autora.

Os parâmetros de entrada para o *TA_Baatz_Segmenter_PostGIS* são:

- ✓ Parâmetros de conexão ao PostgreSQL/PostGIS *Raster*: nome do banco, usuário, senha, nome da máquina, porta, nome da tabela em que a imagem (*raster*) está armazenada;
- ✓ Parâmetros de segmentação:
  - Número de bandas;
  - Peso das bandas - O peso dado a cada banda, quando aplicável (nota: a soma dos pesos das bandas deve ser sempre 1) ou um vetor vazio que indique que todas as bandas têm o mesmo peso;
  - Escala - Tamanho de segmento mínimo positivo (número de *pixels* - padrão: 100);

- Segmentos de similaridade - Valores mais baixos para mesclar apenas os segmentos mais parecidos. Valores mais altos permitem que mais segmentos sejam mesclados (intervalo de valores válidos: valores positivos - padrão: 0,03);
- Cor - Peso atribuído ao componente de cor, padrão: 0,9, intervalo válido: [0,1];
- Compacidade - Peso atribuído ao componente de compacidade, padrão: 0,5, intervalo válido: [0,1];

✓ Execução da segmentação.

No algoritmo implementado na API *TA_Baatz_Segmenter_PostGIS* estão definidos como padrão os seguintes parâmetros:

- Processamento em blocos – Definição do tipo de processamento, se será ou não em blocos. Se sim, o processamento *threaded* será executado (melhor com sistemas multi-core ou multiprocessador (padrão: *true*);
- Tamanho máximo do bloco - A imagem de entrada será dividida em blocos com essa largura para o processamento. Esse parâmetro indica o tamanho máximo do bloco lateral (largura ou altura), o padrão: 0 - o tamanho será definido seguindo os recursos do sistema atual e o número dos processadores físicos).

A codificação que habilita a API para a passagem dos parâmetros do operador *TA_Baatz_Segmenter_PostGIS* é mostrada, em parte, na Figura 25.

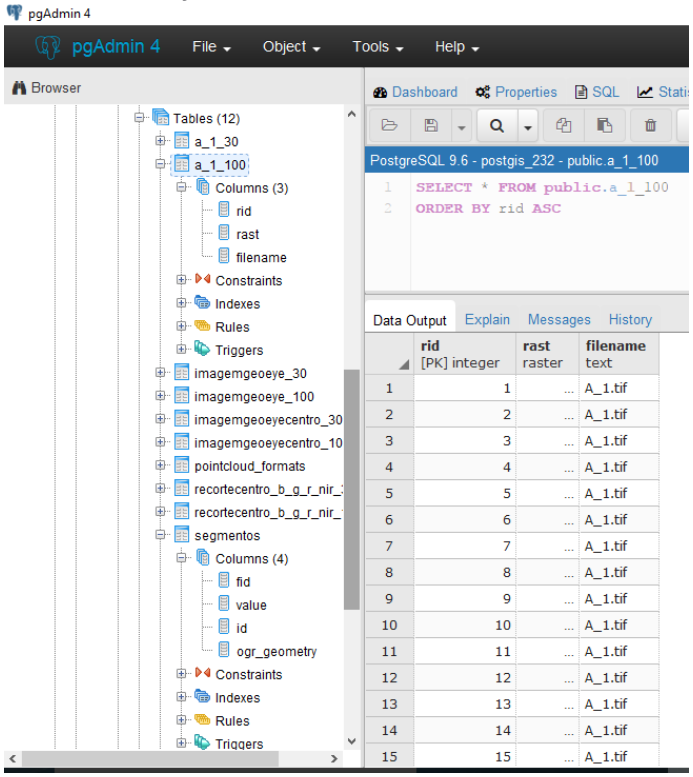
Figura 25: Exemplo de listagem de parte do código da API

```
1
2  <operator type=topdown class=gis name="TA_Baatz_Segmenter_Postgis" cmd="C:/Terralib_5
3    <attribute name=dbname label="a) Banco" value="postgis_232">
4    <attribute name=username label="b) Usuario" type=string value="postgres">
5    <attribute name=password label="c) Senha" type=string value="postgresql">
6    <attribute name=hostname label="d) hostname" type=string value="localhost">
7    <attribute name=port label="e) Porta" type=integer value=5432>
8    <attribute name=tabela label="f) Tabela" type=string value=>
9    <attribute name=scale label="g) Escala" type=integer value=100>
10   <attribute name=similarity label="h) Similaridade" type=double value=0.03>
11   <attribute name=color label="i) Cor" type=double value=0.4>
12   <attribute name=compact label="j) Compacidade" type=double value=0.7>
13   <attribute name=nroband label="k) Nro. Bandas" type=double value=>
14   <attribute name=pesoband label="l) Pesos Bandas" type=double value=>
15 </operator>
```

Fonte: Elaborado pela autora.

A codificação completa da implementação da API para entrada dos dados (parâmetros da segmentação) e codificação de saída dos resultados (*shapefile*) no InterIMAGE encontram-se no Apêndice F. Os dados armazenados no PostGIS *Raster* estão representados, em parte, na Figura 26. A codificação da função SQL que separa os vetores da segmentação resultante por tamanho de *tiles* se encontra no Apêndice E.

Figura 26: Representação dos dados armazenados no PostGIS *Raster*



Fonte: Elaborado pela autora.

3.2.2 Execução da API no InterIMAGE

Estão descritos nos subitens próximos as subetapas de execução da API *TA_Baatz_Segmenter_PostGIS* para fins de validação do método proposto.

3.2.2.1 Criação do projeto no InterIMAGE

O ponto de partida desta fase é a criação do projeto no InterIMAGE. Primeiramente, deve-se observar se a imagem possui tamanho superior a 3.000 x 3.000 *pixels*. Caso afirmativo, será necessário acionar o comando de armazenar a imagem no PostGIS *Raster*. Em seguida, uma função no PostGIS *Raster* é acionada, através de uma função SQL denominada de *stored procedure* (procedimento armazenado), que é um código executável ou *script* dentro do PostgreSQL/PostGIS em que pode ser executado sempre que se precisar do procedimento. A imagem original que se encontra armazenada em disco nessa fase é importada para dentro do PostgreSQL/PostGIS com a definição dos tamanhos dos *tiles*. Em seguida, após o armazenamento, um comando SQL *trigger* (gatilho) é disparo, e os registros da imagem em *tiles*, é exportado para o disco rígido.

As respectivas imagens do disco, em *tiles*, serão adicionadas no projeto para uso no processamento das segmentações e ou nas classificações. Na divisão da imagem em *tiles,* é considerado o Sistema de Referência de Coordenadas (SRC), da imagem original, que é a referência de sistemas de coordenadas cartográficas. Esse método provê maior robustez em relação à qualidade do resultado final, pois mantém a mesma referência de origem quando diferentes processos de segmentação são necessários.

Após o carregamento das imagens em *tiles*, no projeto, o algoritmo de segmentação pode ser acionado com as definições dos dados de entrada pela API que foi desenvolvida com TerraLib 5.2.2 e Visual Studio C++.

Importante ressaltar que os parâmetros definidos no operador da API *TA_Baatz_Segmenter_PostGIS* são repassados ao programa e o processamento da segmentação ocorre sobre a imagem completa. O resultado da segmentação cria um registro no PostGIS *Raster* no tamanho integral da imagem base dos cálculos dos parâmetros.

3.2.2.2 Exportação da segmentação do PostGIS

Em seguida à execução da API *TA_Baatz_Segmenter_PostGIS*, são exportados, do PostGIS *Raster*, a segmentação em arquivos em formato *shapefile*, na dimensão dos *tiles* da imagem. Nesse processo, não há ocorrências de formações indesejáveis ao longo de todas as fronteiras dos *tiles*, pois o processamento é realizado sobre toda a imagem.

A API desenvolvida possibilita que os resultados, segmentos e imagens em *tile*s estejam disponíveis para serem importados, ou adicionados, via InterIMAGE. Os arquivos de segmentação podem ser utilizados para realização de testes de amostras das classes a serem definidas no projeto de classificação.

3.2.2.3 Validação da segmentação

Os parâmetros para o operador *TA_Baatz_Segmenter_PostGIS* que são levados em consideração são: compacidade, cor e escala. As quatro bandas do sensor (0 = *Blue*, 1 = *Green*, 2 = *Red*, 3 = Infravermelho), da imagem desta área de estudo, foram inseridas e definidas na API com o mesmo peso, definindo a relevância de cada banda na classificação. A compacidade caracteriza pela forma de agrupamento de *pixel* de cada objeto classificado e é representada por um valor. Um valor menor representa um objeto menos compactado e um valor maior representa um objeto mais compacto.

No processo de validação das segmentações realizou-se a divisão dos resultados das segmentações da imagem original particionando-as em tamanhos dos *tiles* da imagem armazenada no PostGIS *Raster*. O código da função armazenada no PostgreSQL que realiza esta atividade está descrita no Apêndice E.

3.2.2.4 Definição das classes e regras

No *software* InterIMAGE 1.43 para a classificação orientada a objeto criou-se uma rede semântica com base nas definições de classes definidas nas referências de Antunes (2018), e que estão representadas na Figura 27. Optou-se, após o nó inicial (nó pai), por criar-se os nós vegetação, rasteira, amianto,

cerâmica clara, cerâmica escura, cobertura metálica, solo exposto, asfalto, pavimento de concreto, sombra e não classificado.

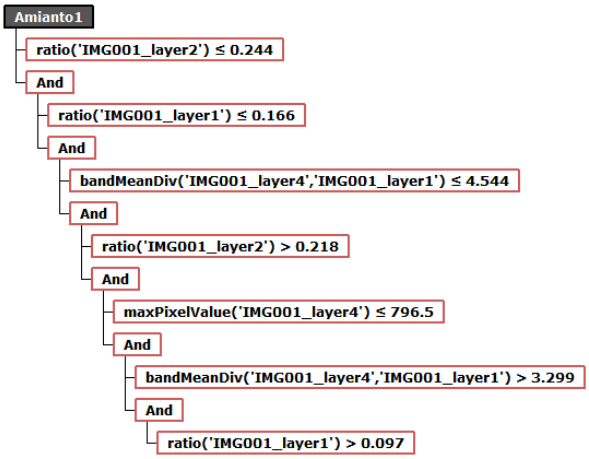
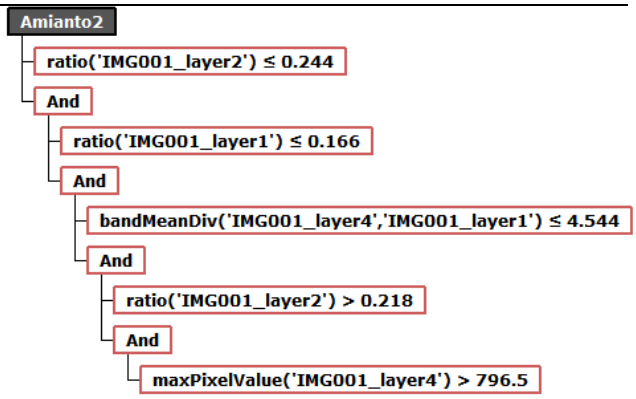
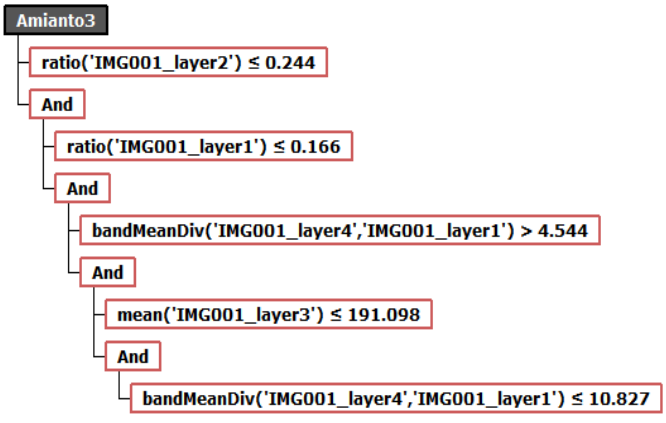
Atribuiu-se ao operador *topdown,* do nó vegetação, o operador *TA_NDVI_Segmenter*, de modo a classificar a vegetação arbórea. O operador gera hipóteses com base no Índice de Vegetação por Diferença Normalizada (NDVI), que é a diferença normalizada entre as bandas espectrais no infravermelho próximo (acentuado pelo espalhamento da radiação eletromagnética incidente na vegetação verde) e no vermelho (acentuada pela absorção da radiação pela clorofila).

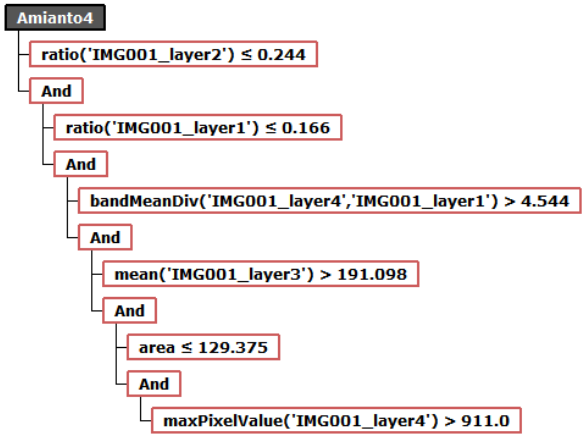
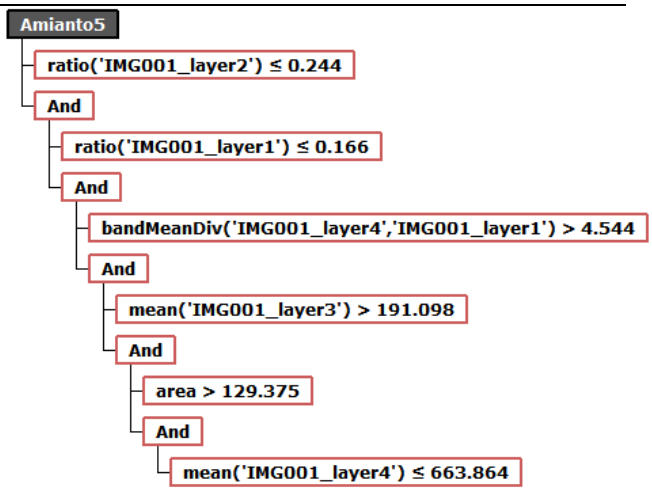
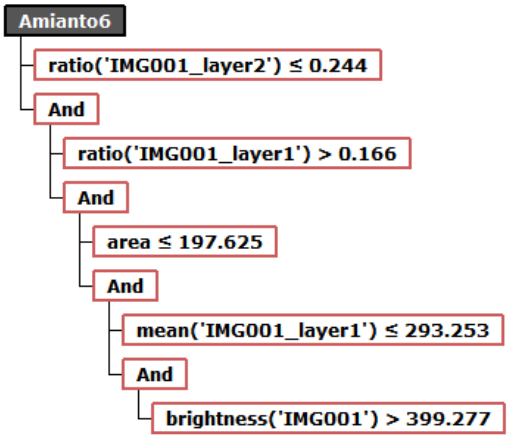
Para o nó sombra, ficou estabelecido o operador *TA_Arithmetic* que é uma operação aritmética (soma e divisão) das imagens de entrada, no caso, as bandas 1, 2, 3 e 4 do sensor, que resultam na separação das áreas escuras na imagem. Para o nó nao_classificado, o operador atribuído foi *dummy topdown*. Este operador tem como resultado uma região que é igual à região de interesse definida no nó pai do nó ao qual está associado, e não permite regra de decisão (INTERIMAGE, 2010). Para os demais nós foram selecionados o operador *TA_Baatz_Segmenter* em que foram atribuídas as regras de decisões no operador *topdown decision*, definidos no Quadro 03.

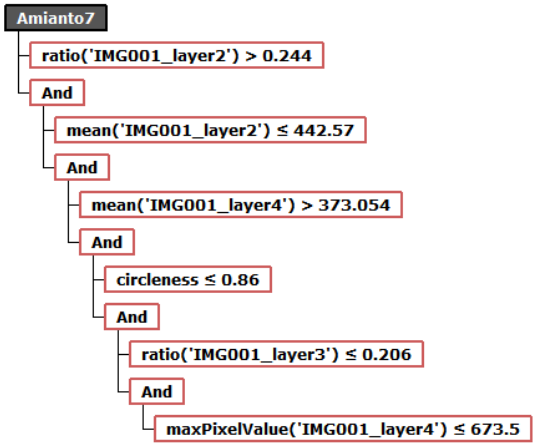
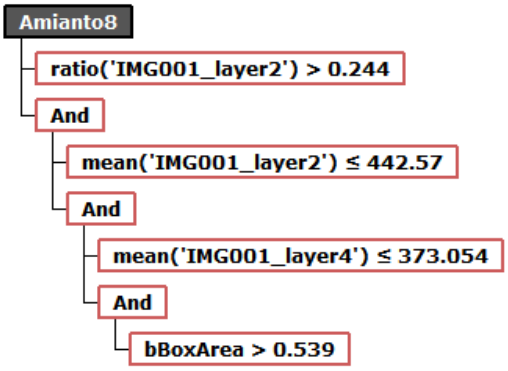
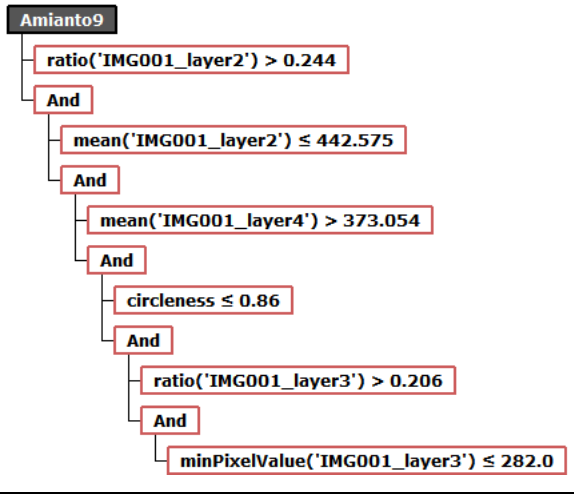
Figura 27: Classes de uso e cobertura de terras da área de estudo consideradas na etapa de classificação orientada a objetos no InterIMAGE Desktop

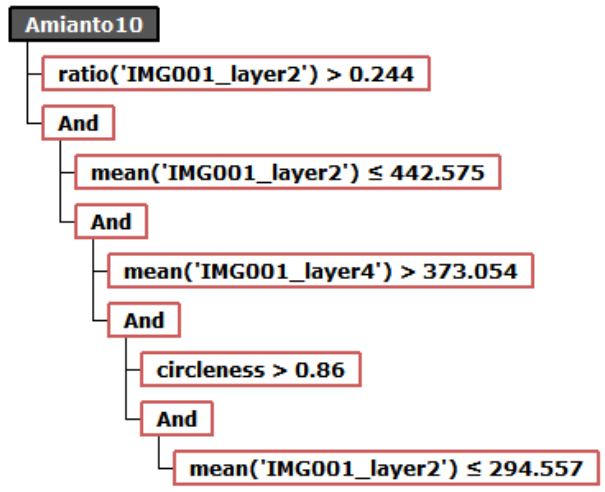
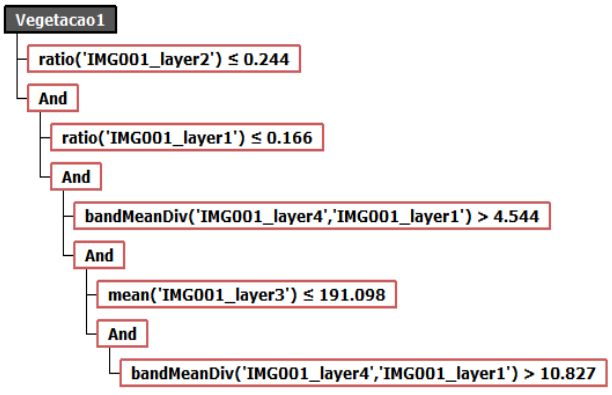
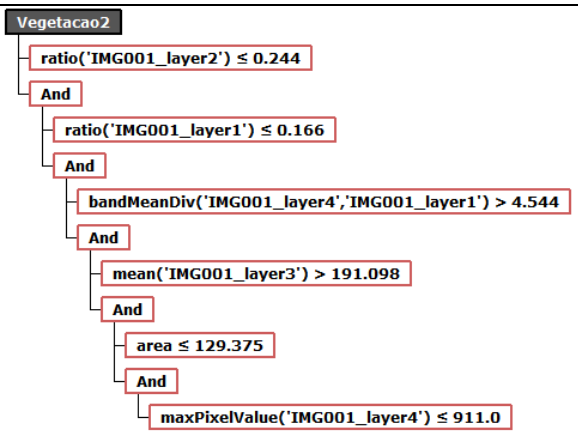
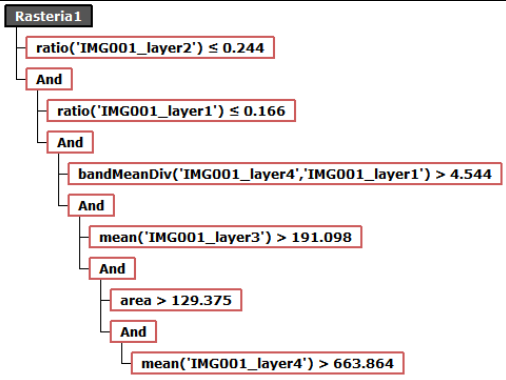
Fonte: Adpatado de Antunes *et al*. (2018).

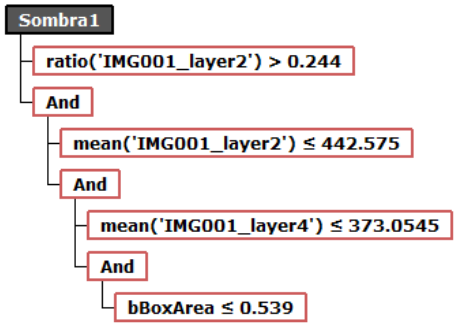
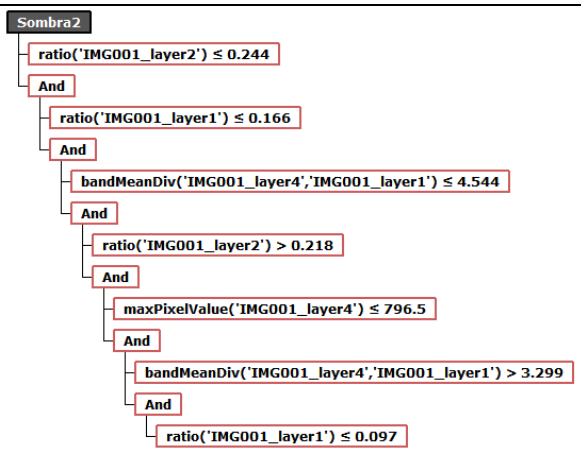
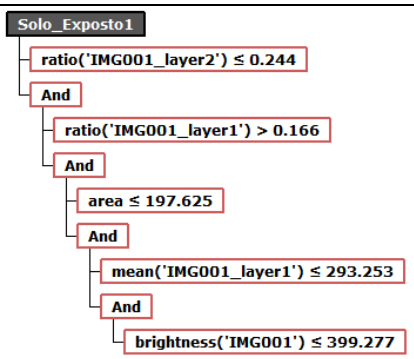
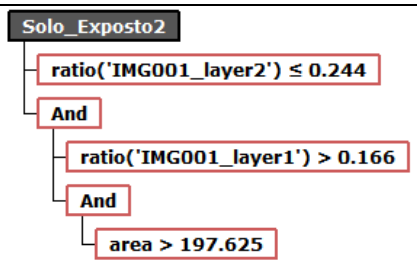
Quadro 3: Regras para classificação no InterIMAGE.

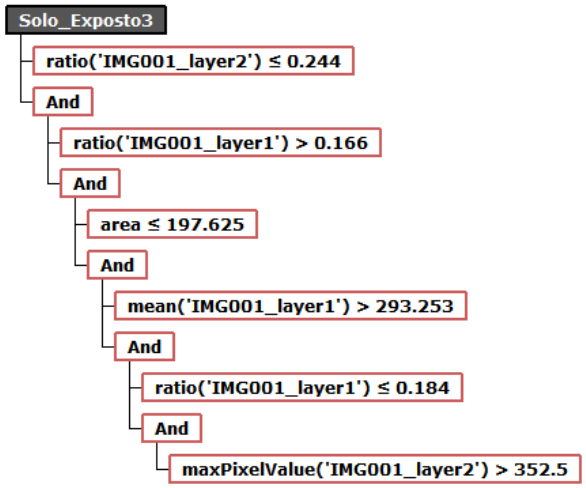
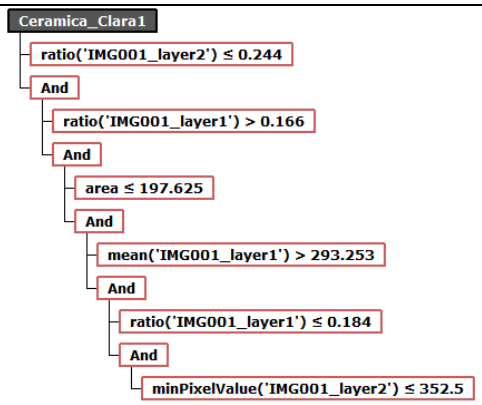
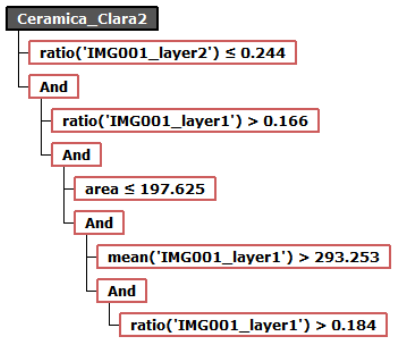
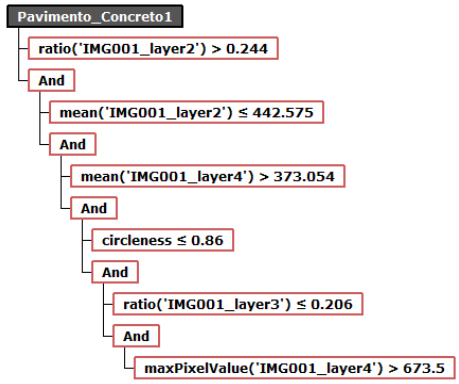
Regra	Tela *Node Editor*
Amianto1 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1')<= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 4.544 AND ratio('IMG001_layer2') > 0.218 AND maxPixelValue('IMG001_layer4') <= 796.5 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 3.299 AND ratio('IMG001_layer1') > 0.097	
Amianto2 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 4.544 AND ratio('IMG001_layer2') > 0.218 AND maxPixelValue('IMG001_layer4') > 796.5	
Amianto3 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 4.544 AND mean('IMG001_layer3') <= 191.098 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 10.827	

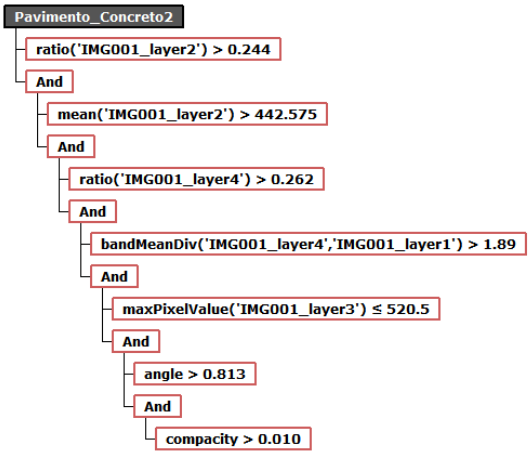
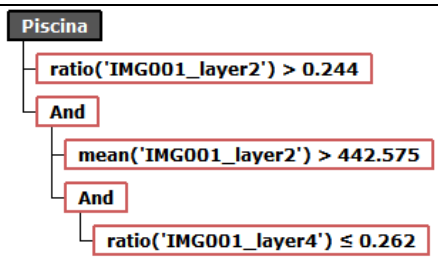
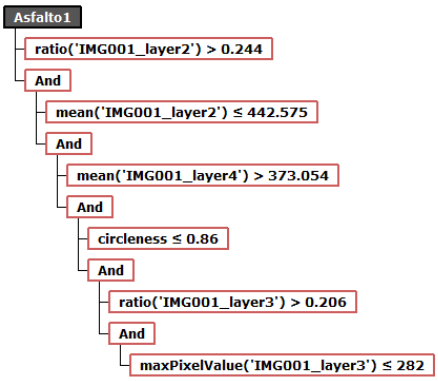
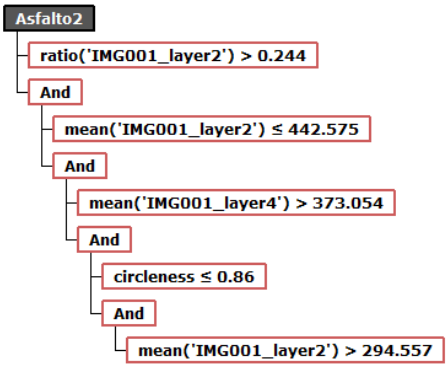
Amianto4 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 4.544 AND mean('IMG001_layer3') > 191.098 AND 'area' <= 129.375 AND maxPixelValue('IMG001_layer4') > 911.0	
Amianto5 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 4.544 AND mean('IMG001_layer3') > 191.098 AND 'area' > 129.375 AND mean('IMG001_layer4') <= 663.864	
Amianto6 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') > 0.166 AND 'area' <= 197.625 AND mean('IMG001_layer1') <= 293.253 AND brightness('IMG001') > 399.277	

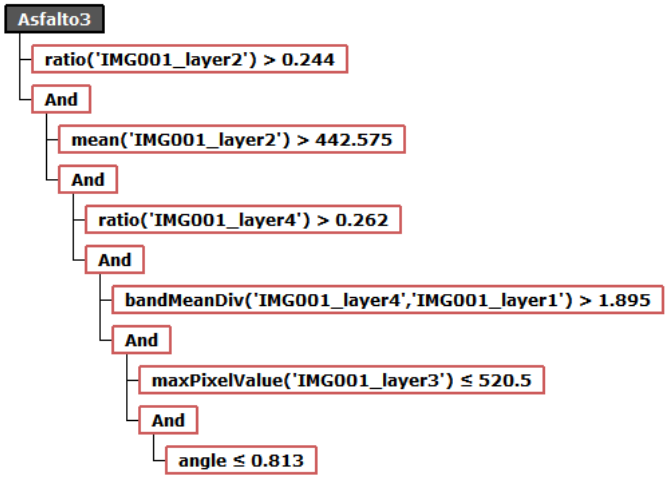
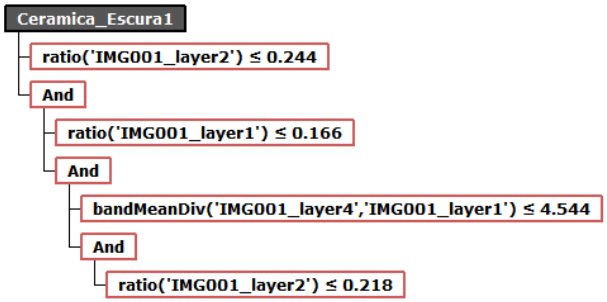
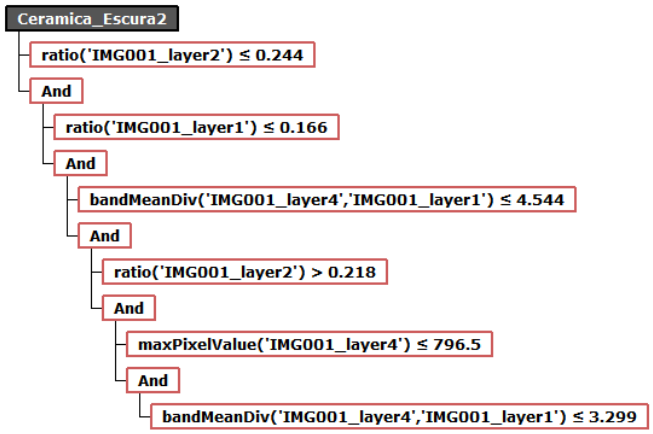
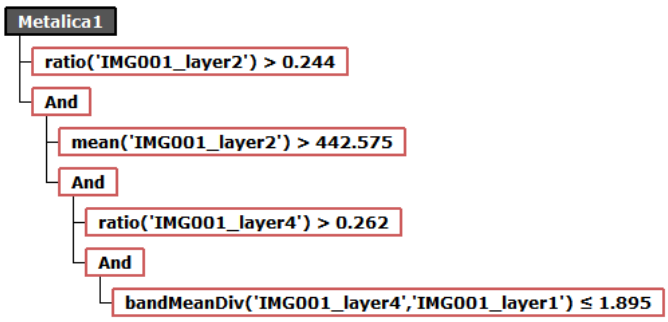
Amianto7 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.57 AND mean('IMG001_layer4') > 373.054 AND circleness <= 0.86 AND ratio('IMG001_layer3') <= 0.206 AND maxPixelValue('IMG001_layer4') <= 673.5	
Amianto8 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.57 AND mean('IMG001_layer4') <= 373.054 AND bBoxArea > 0.539	
Amianto9 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.57 AND mean('IMG001_layer4') > 373.054 AND circleness <= 0.86 AND ratio('IMG001_layer3') > 0.206 AND minPixelValue('IMG001_layer3') <= 282.0	

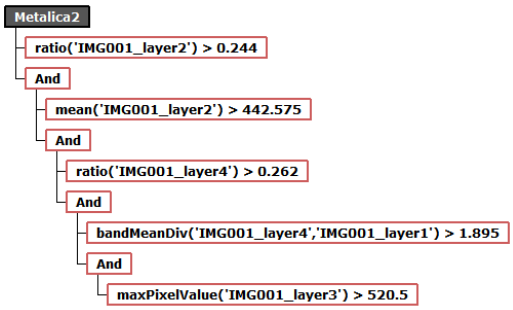
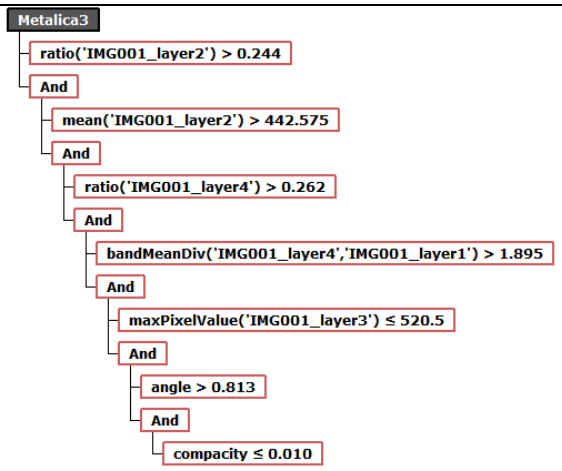
Amianto10 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.57 AND mean('IMG001_layer4') > 373.054 AND circleness > 0.86 AND mean('IMG001_layer2') <= 294.557	
Vegetacao1 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 4.544 AND mean('IMG001_layer3') <= 191.098 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 10.827	
Vegetacao2 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 4.544 AND mean('IMG001_layer3') > 191.098 AND 'area' <= 129.375 AND maxPixelValue('IMG001_layer4') <= 911.0	
Rasteira1 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 4.544 AND mean('IMG001_layer3') > 191.098 AND 'area' > 129.375 AND mean('IMG001_layer4') > 663.864	

Sombra1 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.575 AND mean('IMG001_layer4') <= 373.0545 AND 'bBoxArea <= 0.539	
Sombra2 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 4.544 ratio('IMG001_layer2') > 0.218 AND maxPixelValue('IMG001_layer4') <= 796.5 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 3.299 AND ratio('IMG001_layer1') <= 0.097	
Solo_exposto1 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') > 0.166 AND 'area' <= 197.625 AND mean('IMG001_layer1') <= 293.253 AND brightness('IMG001') <= 399.277	
Solo_exposto2 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') > 0.166 AND 'area' > 197.625	

Solo_exposto3 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') > 0.166 AND 'area' <= 197.625 AND mean('IMG001_layer1') > 293.253 AND ratio('IMG001_layer1') <= 0.184 AND maxPixelValue('IMG001_layer2') > 352.5	
Ceramica_clara1 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') > 0.166 AND 'area' <= 197.625 AND mean('IMG001_layer1') > 293.253 AND ratio('IMG001_layer1') <= 0.184 AND minPixelValue('IMG001_layer2') <= 352.5	
Ceramica_clara2 ratio('IMG001_layer2') <= 0.244 AND ratio('IMG001_layer1') > 0.166 AND 'area' <= 197.625 AND mean('IMG001_layer1') > 293.253 AND ratio('IMG001_layer1') > 0.184	
Pavimento_Concreto1 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.575 AND mean('IMG001_layer4') > 373.054 AND circleness <= 0.86 AND ratio('IMG001_layer3') <= 0.206 AND maxPixelValue('IMG001_layer4') > 673.5	

Pavimento_Concreto2 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') > 442.575 AND mean('IMG001_layer4') > 0.262 AND maxPixelValue('IMG001_layer3') <= 520.5 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 1.895 AND 'angle' > 0.813 AND 'compacity' > 0.010	
Piscina ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') > 442.575 AND ratio('IMG001_layer2') <= 0.262	
Asfalto1 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.575 AND mean('IMG001_layer4') > 373.054 AND circleness <= 0.86 AND mean('IMG001_layer3') > 0.206 AND maxPixelValue('IMG001_layer3') <= 282.0	
Asfalto2 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') <= 442.575 AND mean('IMG001_layer4') > 373.054 AND circleness <= 0.86 AND mean('IMG001_layer2') > 294.557	

Asfalto3 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') > 442.575 AND ratio ('IMG001_layer4') > 0.262 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 1.895 AND maxPixelValue('IMG001_layer3') <= 520.5 AND 'angle' <= 0.813	
Ceramica_escura1 ratio('IMG001_layer2') <= 0.244 AND ratio ('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 4.544 AND ratio ('IMG001_layer2') <= 0.218	
Ceramica_escura2 ratio('IMG001_layer2') <= 0.244 AND ratio ('IMG001_layer1') <= 0.166 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 4.544 AND ratio ('IMG001_layer2') > 0.218 AND maxPixelValue('IMG001_layer4') <= 796.5 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 3.299	
Metalica1 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') > 442.575 AND ratio ('IMG001_layer4') > 0.262 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') <= 1.895	

Metalica2 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') > 442.575 AND ratio ('IMG001_layer4') > 0.262 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 1.895 AND maxPixelValue('IMG001_layer3') > 520.5	
Metalica3 ratio('IMG001_layer2') > 0.244 AND mean('IMG001_layer2') > 442.575 AND ratio ('IMG001_layer4') > 0.262 AND bandMeanDiv('IMG001_layer4','IMG001_layer1') > 1.895 AND maxPixelValue('IMG001_layer3') <= 520.5 AND 'angle' > 0.813 AND 'compacity' <= 0.010	

Fonte: Elaborado pela autora.

3.2.2.5 Processamento da segmentação no InterIMAGE

Com a rede semântica definida e os respectivos parâmetros para as classes, procedeu-se a classificação orientada a objetos no InterIMAGE.

3.2.2.6 Validação da classificação

No processo de verificação das classificações realizaram-se observações dos artefatos nas bordas dos *tiles* para os resultados obtidos.

3.2.2.7 Armazenamento dos resultados no PostGIS

O PostgreSQL e suas extensões espaciais do PostGIS *Raster* apresentam-se como uma ferramenta para armazenamento e consultas de imagens de satélite e dados vetoriais, possibilitando ao usuário consultar valores reais de dados contidos na imagem do satélite, e realizar-se consultas que interagem entre informações em formato vetor e matricial.

Os resultados das classificações podem ser armazenados no PostGIS *Raster*, na mesma estrutura do banco em forma de imagem e vetores. Pode-se

executar a função de unir os registros das segmentações incluídas no ambiente do SGBD, e também exportar a tabela da unificação dos segmentos para gravação de um único arquivo armazenado em disco.

Os procedimentos de armazenamento, importação e exportação são realizados via *stored procedure* (funções armazendas) e triggers (gatilhos) programadas no PostreSQL/PostGIS *Raster*.

3.2.3 Comparação de tempo de processamento das segmentações (InterIMAGE e PostgreSQL /PostGIS *Raster*)

Para a comparação de tempos de processamentos no PostgreSQL/PostGIS *Raster* foram realizadas segmentações utilizando os operadores *TA_Baatz_Segmenter* (processamento exclusivo no InterIMAGE e com dados armazenado em disco rígido) e o *TA_Baatz_Segmenter_PostGIS* (operador adicionado no InterIMAGE e processamento realizado no PostGIS *Raster* sendo com imagem armazenada em tabela no formato *raster*). Os tempos, definidos na unidade de segundos, foram medidos a cada execução dos operadores após a definição dos parâmetros de testes, e estabeleceu-se um valor fixo para os blocos de processamento.

Na fase de desenvolvimento da API, para o operador *TA_Baatz_Segmenter_PostGIS,* optou-se pela indicação *default* da Terralib 5 os padrões de processamento para os dados *raster* de segmentação como:

- *m_enableThreadedProcessing* = *true* → Se for verdade, o processamento encadeado será executado (melhor com sistemas *multi-core* ou *multi-processadores* (padrão: true).

- *maxSegThreads* = 0 → O número máximo de encadeamentos do segmentador simultâneo (padrão: 0 - localizado automaticamente);

- *m_enableBlockProcessing* = *true* → Se *true*, a imagem original será dividida em pequenos blocos, cada uma será segmentada independentemente e o resultado será mesclado (se possível) no final (padrão: *true*).

- m_blocksOverlapPercent = 10 → Percentual de blocos com sobreposição.

- m_maxBlockSize = 0 → A imagem de entrada será dividida em blocos com essa largura para processamento, este parâmetro informa o tamanho lateral

máximo do bloco (largura ou altura), o padrão: 0 - o tamanho será definido seguindo os recursos atuais do sistema e número de processadores físicos).

Os parâmetros da API *TA_Baatz_Segmenter_PostGIS,* da biblioteca TerraLib, são executados sob as características de definição de memória e blocos do PostgreSQL.

A extensão PostGIS *Raster* possibilita a manipulação dos dados *raster* sobre as funcionalidades do PostgreSQL que possui um sistema de cache, ou seja, de armazenamento temporário em memória, dos blocos recuperados em uma determinada seleção para processamento. A finalidade deste sistema é evitar consultas ao PostgreSQL de acessos consecutivos ao mesmo bloco da representação de dados, ou a elementos diferentes de um mesmo bloco, pois esta operação tornar-se-ia onerosa ao processamento.

Conjuntamente, o sistema de memória temporária mantém um *array* associativo ou mapa, na memória principal, entre identificadores de blocos e uma posição de memória com seu conteúdo. A chave do *array* associativo é a mesma identificação de blocos usada para armazená-lo no banco de dados.

Os números de blocos que podem ser armazenados ao mesmo tempo caracteriza a memória temporária. E, o valor do número de blocos pode ser definido na aplicação, bem como devem ser levados em consideração os recursos de memória disponíveis.

CAPÍTULO 4
RESULTADOS E DISCUSSÕES

Apresentam-se as segmentações de uma imagem do satélite GeoEye-1 armazenada no sistema gerenciador de banco de dados e em disco, respectivamente, no PostgreSQL/PostGIS *Raster* e no InterIMAGE, aplicando-se o algoritmo de segmentação, segundo Baatz e Schäpe (2000). Mostram-se conjuntamente os tempos decorridos dos processamentos e os exemplos de resultados das classificações.

4.1 Segmentação – API Segmenter Baatz no PostGIS e InterIMAGE

É importante esclarecer que a API *TA_Baatz_Segmenter_PostGIS* não é uma aplicação completa, e nem definitiva, porque constitui-se somente de uma parte de um processo que poderá levar à construção de partes complementares no sistema InterIMAGE Desktop, de forma a atender às demandas existentes.

Na segmentação da imagem GeoEye-1, as quatro bandas do sensor foram inseridas no sistema com o mesmo peso, definindo-se igual relevância de cada banda na segmentação.

Os experimentos foram realizados variando-se os parâmetros de escala, peso de cor e peso de compacidade e obtendo-se o tempo de processamento para cada processamento.

Os processamentos foram padronizados para serem calculados em blocos de tamanhos 10, sendo este parâmetro estabelecido no algoritmo da API *TA_Baatz_Segmenter_PostGIS*. Padronizaram-se os parâmetros de processamentos: otimização = no, confiabilidade = 0,2 e distância euclidiana relativa = 20. Os valores apresentados nas Tabelas 1 e 2 referem-se ao processamento realizado no ambiente PostGIS *Raster,* executados a partir da API *TA_Baatz_Segmenter_PostGIS*.

Na utilização do operador *TA_Baatz_Segmenter_PostGIS* ficaram definidos os valores de similaridade para os experimentos, realizados nas Tabelas 1 e 2, no padrão fornecido pela TerraLib 5.2.2 que é 0.03. O valor de similaridade é a semelhança entre segmentos vizinhos para mesclá-los ou não.

Tabela 1: Parâmetros de experimentos aplicados na API TA_Baatz_ Segmenter_PostGIS

TA_ Baatz_ Segmenter_PostGIS	Imagem GeoEye-1 - Experimento 1	Imagem GeoEye-1 - Experimento 2	Imagem GeoEye-1 - Experimento 3	Imagem GeoEye-1 - Experimento 4
Escala	100	100	70	60
Cor	0.8	0.8	0.5	0.5
Compacidade	0.2	0.2	0.5	0.5
Número de bandas	4	4	4	4
Peso das bandas	1	1	1	1
Imagem em *pixels* (linhas x colunas)	19.404 x 21.360	19.404 x 21.360	19.404 x 21.360	19.404 x 21.360
Tamanho (MB)	3.080	3.080	3.080	3.080
Tiles armazenados no PostGIS em *pixels* (linhas x colunas)	2048 x 2048	1024 x 1024	1024 x 1024	1024 x 1024
Linhas armazenadas na tabela *raster* do PostGIS	110	399	399	399
Tempo de processamento (s)	98.320	97.200	172.840	216.220
Tamanho do arquivo vetorial gerado (*shapefile*) (GB)	4,30	4,297	4,45	4,67

Fonte: Elaborado pela autora.

Tabela 2: Parâmetros de experimentos aplicados no InterIMAGE – API TA_Segmenter_Baatz_PostGIS

TA_ Baatz_ Segmenter_PostGIS	Imagem GeoEye-1 – Parte – Experimento 5	Imagem GeoEye-1 - Parte - Experimento 6	Imagem GeoEye-1 - Parte - Experimento 7
Escala	100	70	60
Cor	0.5	0.5	0.5
Compacidade	0.5	0.5	0.5
Número de bandas	4	4	4
Peso das bandas	1	1	1
Imagem em *pixels* (linhas x colunas)	989 x 1560	989 x 1560	989 x 1560
Tamanho (MB)	9,68	9,68	9,68
Tiles armazenados no PostGIS em *pixels* (linhas x colunas)	1024 x 1024	1024 x 1024	1024 x 1024
Linhas armazenadas na tabela raster do PostGIS	2	2	2
Tempo de processamento (s)	190	170	179
Tamanho do arquivo vetorial gerado (shapefile) (MB)	14,4	17,1	18,5

Fonte: Elaborado pela autora.

Os tempos de processamento dos experimentos realizados, representados nas Tabelas 1 e 2, consideraram diferentes tamanhos de imagens, diferentes tamanhos dos *tiles* e as parametrizações dos atributos para segmentação apresentaram-se com valores diferenciados. Estes resultados de tempos de processamentos estão relacionados aos tamanhos das imagens à serem processadas no PostGIS *Raster*, bem como em função do uso de memória e

CPU (Unidade Central de Processamento) alocada para processamento, e disponibilização do equipamento para rodar principalmente este processo.

Ainda, o tempo de processamento das segmentações está condicionado ao sistema de particionamento da representação do *raster*, e vetores, armazenados e/ou gerados, conjuntamente com a indexação espacial dentro do sistema de gerenciamento de banco de dados. Nas Tabelas 1 e 2 a definição dos blocos da representação para fins de processamento foram estabelecidos em 10.

Quando a aplicação solicita o acesso a um bloco que ainda não está na memória temporária, caso não existam posições livres para acomodá-lo, o SGBD-OR PostgreSQL/PostGIS *Raster* e conjuntamente com o sistema operacional realizam uma operação de substituição de blocos, que consiste na escolha de um bloco para ser descartado, liberando seu espaço para o novo bloco solicitado.

Na Tabela 3 encontram-se os parâmetros utilizados para obtenção da segmentação no ambiente do InterIMAGE, via operador *TA_Baatz_Segmenter*. A segmentação no InterIMAGE é realizada somente com processamento e armazenamento dos resultados em disco rígido. Na Tabela 3, os tempos de processamento e tamanho de arquivos resultantes responderam às variações dos parâmetros: escala, cor e compacidade.

Tabela 3: Parâmetros de experimentos aplicados no InterIMAGE 1.43 - TA_Segmenter_Baatz – processamento em disco

TA_Segmenter_Baatz	**Imagem GeoEye-1 Parte**	**Imagem GeoEye-1 – Parte – Experimento 8**	**Imagem GeoEye-1 – Parte – Experimento 9**	**Imagem GeoEye-1 – Parte – Experimento 10**
Escala	70	100	70	60
Cor	0.5	0.5	0.5	0.5
Compacidade	0.5	0.5	0.5	0.5
Número de bandas	4	4	4	4
Peso das bandas	1	1	1	1
Imagem em *pixels* (linhas x colunas)	19.404 x 21.360	989 x 1560	989 x 1560	989 x 1560
Tamanho (MB)	3.080	9,68	9,68	9,68
Tempo de processamento (s)	Não há	38,14	41,04	48,42
Tamanho do arquivo vetorial gerado (shapefile) (MB)	Não há	17,8	29,0	36,0

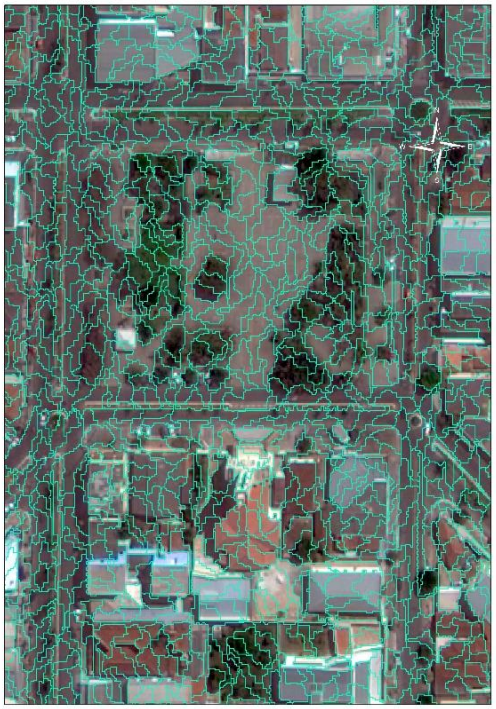
Fonte: Elaborado pela autora.

Para observação dos resultados, escolheu-se uma parte da imagem, compreendida na área urbana, e o seu recorte está representado na Figura 28. Nas Figuras 29, 30 e 31 estão representadas as segmentações obtidas na execução do operador *TA_Baatz_Segmenter_PostGIS* no ambiente do InterIMAGE processado no PostGIS *Raster*. As Figuras 32, 33 e 34 representam parte dos resultados das segmentações realizadas somente no InterIMAGE, utilizando-se o operador *TA_Baatz_Segmenter*.

Figura 28: Parte da imagem GeoEye da área urbana de Goianésia

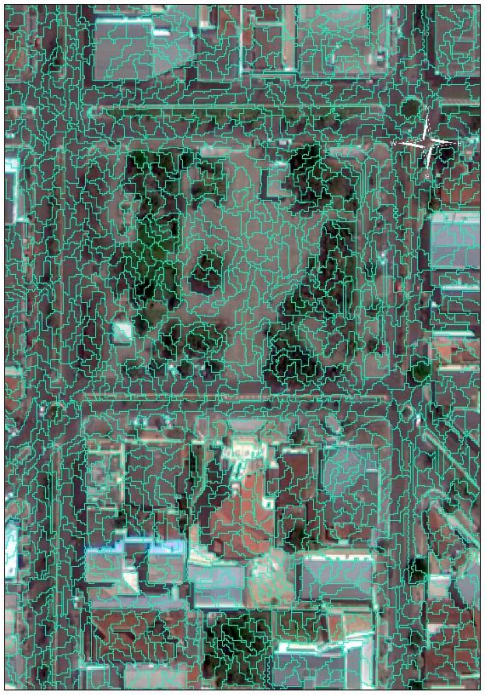
Fonte: Elaborado pela autora.

Figura 29: Segmentação no PostGIS *Raster* com os parâmetros: escala 100 – cor 0,5 – compacidade 0,5



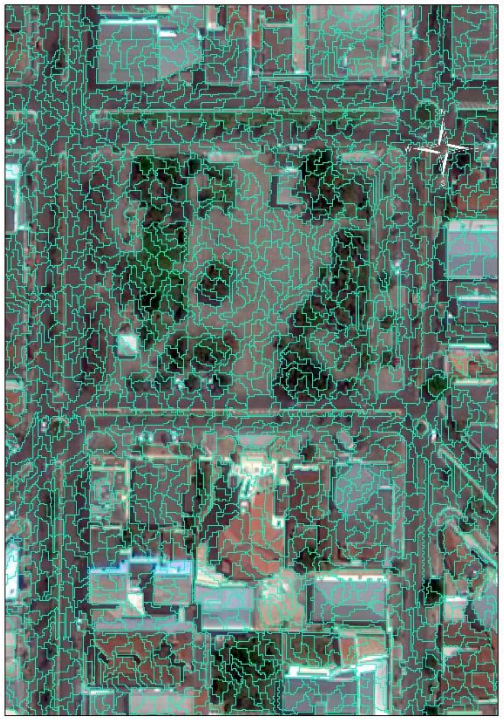
Fonte: Elaborado pela autora.

Figura 30: Segmentação no PostGIS *Raster* com os parâmetros: escala 70 – cor 0,5 – compacidade 0,5



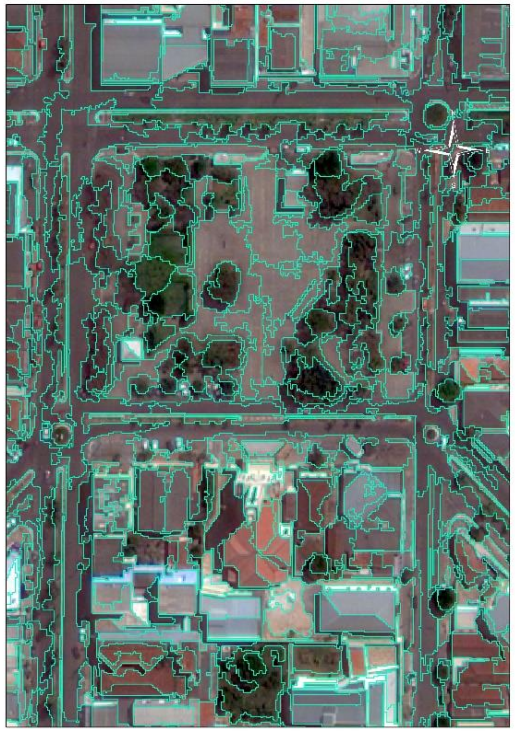
Fonte: Elaborado pela autora.

Figura 31: Segmentação no PostGIS *Raster* com os parâmetros: escala 60 – cor 0,5 – compacidade 0,5



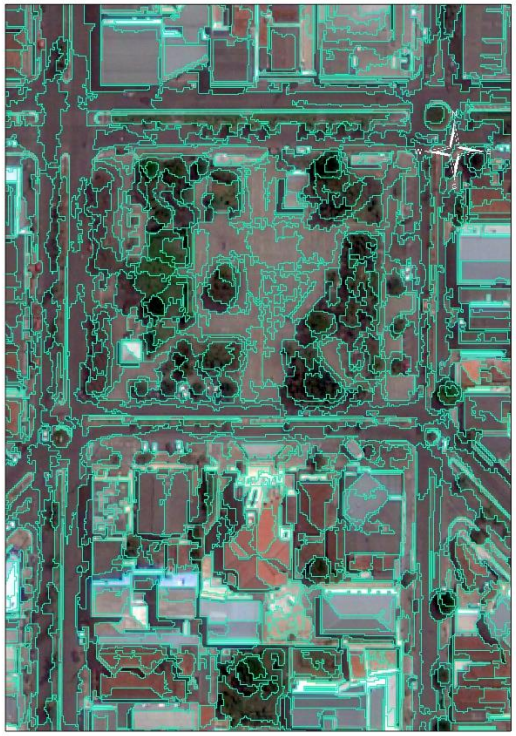
Fonte: Elaborado pela autora.

Figura 32: Recorte segmentação no InterIMAGE com os parâmetros: escala 100 – cor 0,5 – compacidade 0,5



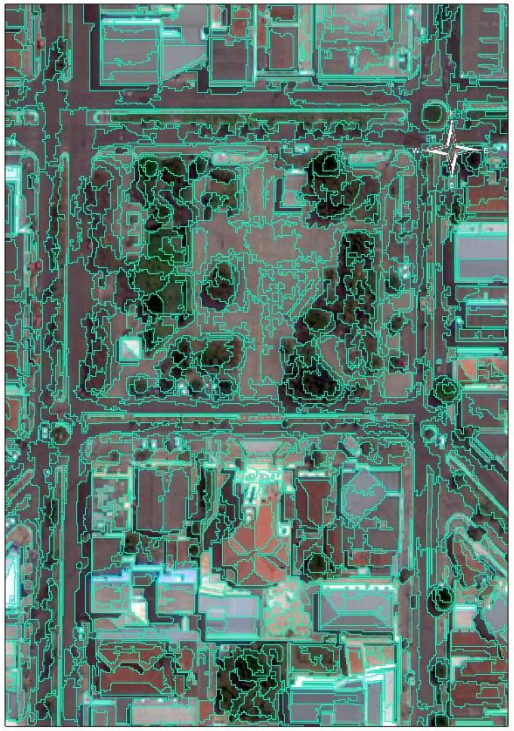
Fonte: Elaborado pela autora.

Figura 33: Recorte segmentação no InterIMAGE com os parâmetros: escala 70 – cor 0,5 – compacidade 0,5.



Fonte: Elaborado pela autora.

Figura 34: Recorte segmentação no InterIMAGE com os parâmetros: escala 60 – cor 0,5 – compacidade 0,5



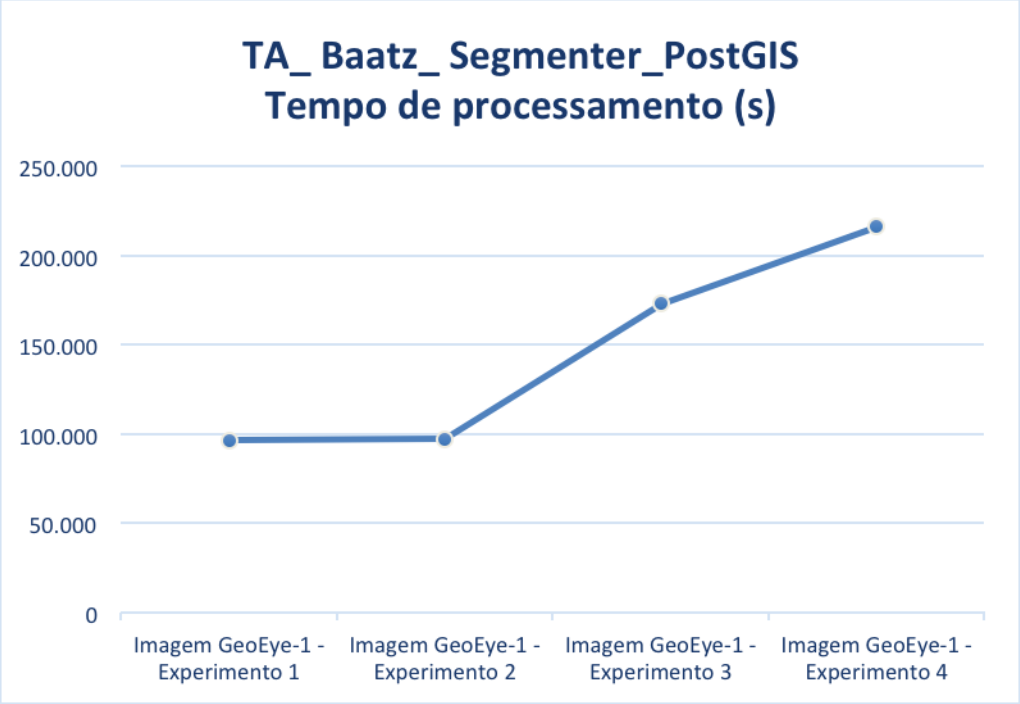
Fonte: Elaborado pela autora.

As Figuras 29, 30 e 31, respectivamente, demonstram os resultados de segmentos da API do operador *TA_Baatz_Segmenter_PostGIS*. E, apresentaram segmentos mais detalhados em função da variação da escala e do parâmetro de similaridade definido para 0.03, disponível pela redefinição na adequação e otimização da TerraLib 5.2.2 para a classe *SegmenterRegionGrowingBaatzStrategy* utilizada.

As segmentações apresentadas nas Figuras 32, 33 e 34 estão condicionadas ao processamento do operador *TA_Baatz_Segmenter* e às variações de definições de escalas.

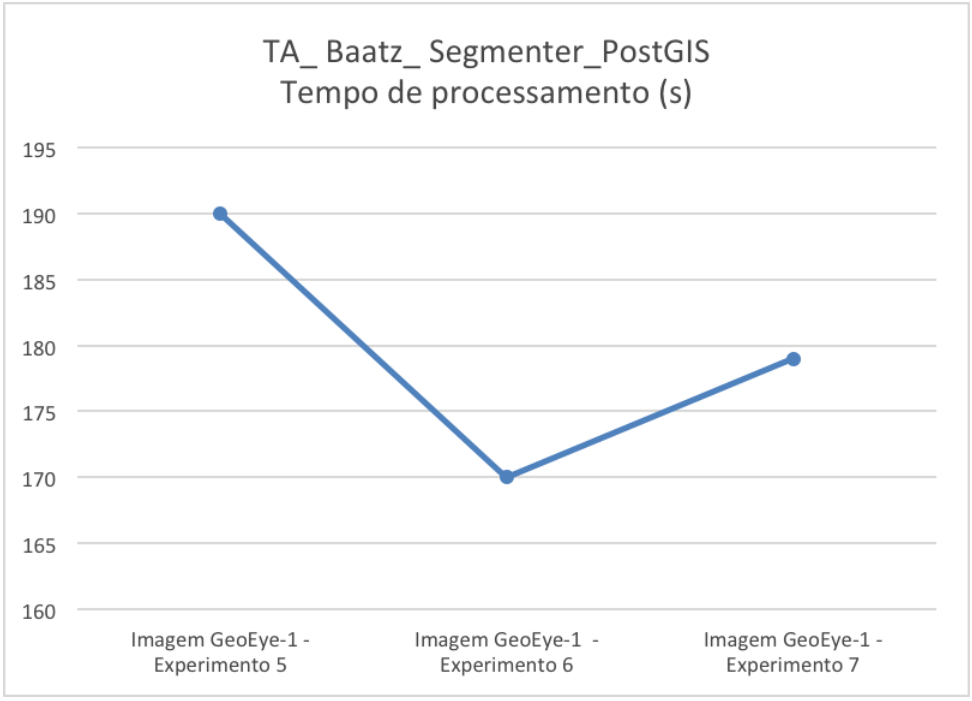
As Figuras 35 e 36 demonstram, individualmente, os resultados de tempos de processamentos da segmentação no PostGIS *Raster.*

Figura 35: Tempo de processamento (em segundos) da Tabela 1



Fonte: Elaborado pela autora.

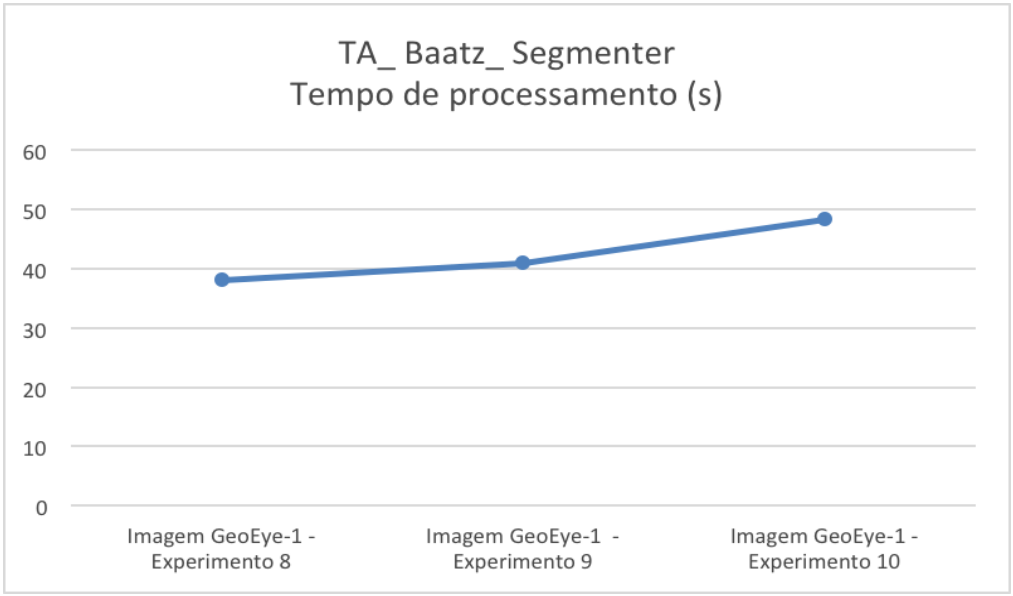
Figura 36: Tempo de processamento (em segundos) da Tabela 2



Fonte: Elaborado pela autora.

A Figura 37 demonstra o tempo decorrido para o processamento, em segundos, da segmentação realizada somente no ambiente do InterIMAGE. O parâmetro escala foi o único que foi modificado nos três experimentos (8, 9 e 10), e demais parâmetros definidos na tabela ficaram com os mesmos valores.

Figura 37: Tempo de processamento (em segundos) da Tabela 3.



Fonte: Elaborado pela autora.

Todos os tempos decorridos nos experimentos, ou seja, o processamento das segmentações, derivam dos valores de parâmetros escolhidos (escala, cor e compacidade), configuração de memória dedicada para a computação dos cálculos e a disponibilidade de dedicação exclusiva do processador para a execução dos processamentos.

Na Tabela 4, optou-se por apresentar todos dados dos processamentos no SGBD-OR PostGIS *Raster* e no InterIMAGE, e foram representados em forma de gráfico na Figura 38.

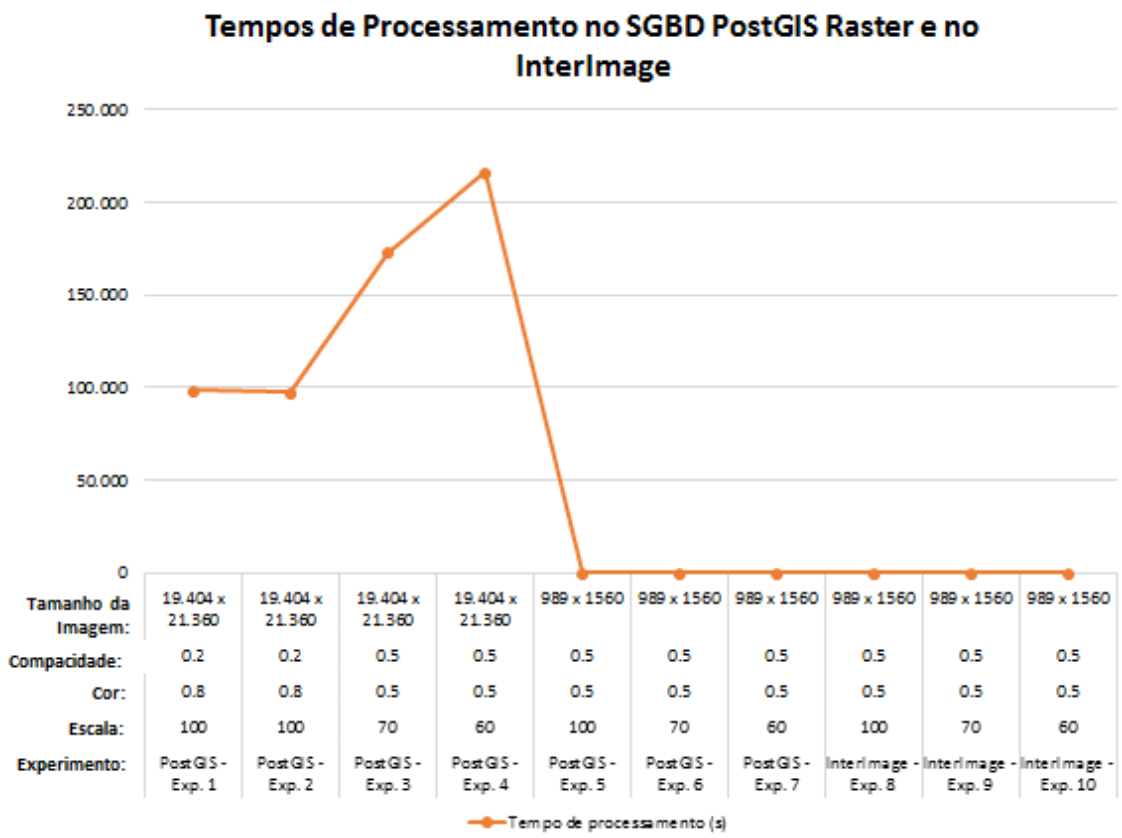
O gráfico da Figura 38 denota os tempos de processamento entre o SGBD-OR PostGIS *Raster* e o InterIMAGE. Ressalta-se que todos os parâmetros devem ser observados, principalmente o tamanho das imagens e a variçãod e escala de processamento. Destacando, que todos os parâmetros dos experimentos influenciam diretamente nos resultados de tempos de processamentos decorridos.

Tabela 4: Parâmetros de processamento no SGBD-OR PostGIS *Raster* e InterIMAGE

Operador ->		**TA_ Baatz_ Segmenter_ PostGIS**	**TA_ Baatz_ Segmenter_PostGIS**	**TA_ Baatz_ Segmenter_PostGIS**	**TA_ Baatz_ Segmenter_PostGIS**	**TA_ Baatz_ Segmenter_PostGIS**	**TA_ Baatz_ Segmenter_PostGIS**	**TA_ Baatz_ Segmenter_PostGIS**	**TA_ Baatz_ Segmenter**	**TA_ Baatz_ Segmenter**	**TA_ Baatz_ Segmenter**
Parâmetros	**Item**	**Exp. 1**	**Exp. 2**	**Exp. 3**	**Exp. 4**	**Exp. 5**	**Exp. 6**	**Exp. 7**	**Exp. 8**	**Exp. 9**	**Exp. 10**
Escala	A	100	100	70	60	100	70	60	100	70	60
Cor	B	0.8	0.8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Compacidade	C	0.2	0.2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Imagem em *Pixels* (linhas x colunas)	D	19.404 x 21.360	19.404 x 21.360	19.404 x 21.360	19.404 x 21.360	989 x 1560	989 x 1560	989 x 1560	989 x 1560	989 x 1560	989 x 1560
Tamanho (MB)	E	3.080	3.080	3.080	3.080	9,68	9,68	9,68	9,68	9,68	9,68
Tempo de processamento (s)	H	98.320	97.200	172.840	216.220	190	170	179	38,14	41,04	48,42

Fonte: Elaborado pela autora.

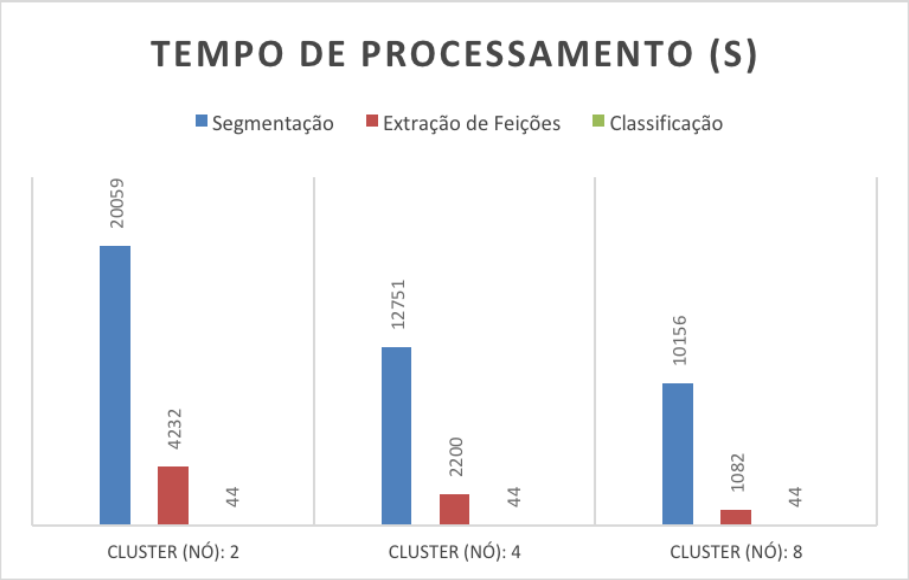
Figura 38: Tempo de processamento da Tabela 4



Fonte: Elaborado pela autora.

Em Antunes (2016), e representado na Figura 39, mostra-se o resultado da segmentação da mesma imagem da área deste estudo, neste caso a imagem completa (19.404 x 21.360 *pixels*), porém, o processamento foi realizado no InterCloud, ou InterIMAGE *Cloud Platform*. O principal objetivo da InterIMAGE *Cloud Platform* é o desenvolvimento científico e tecnológico na análise de imagens de sensoriamento remoto (SR), focada na análise de imagens baseada em objetos (GEOBIA) e na interpretação de imagens em um ambiente distribuído e escalável de forma a ser capaz de processar grandes volumes de dados. Compõem os principais tópicos: (1) a representação e processamento do conhecimento explicitamente representado na interpretação das imagens de SR; (2) métodos de segmentação de imagens SR, extração de atributos e classificação de objetos em ambientes distribuídos; (3) métodos de análise multitemporal de imagens de SR em ambientes distribuídos; e (4) métodos de visualização avançada de imagens e objetos armazenados de forma distribuída em *clusters* de computadores (INTERIMAGE *CLOUD PLATFORM,* 2018).

Figura 39: Tempo de processamento no InterCloud



Fonte: Adptado de Antunes (2016).

Segundo Antunes (2106), o InterCloud pode explorar a escalabilidade fornecida pelos serviços de infraestrutura de computação em nuvem comercial, permitindo a interpretação de conjuntos de dados de sensoriamento remoto muito grandes de maneira eficiente. Este ambiente de plataforma de interpretação de imagens foi projetado para ser executado em grades de computadores (*clusters* físicos ou infraestrutura de computação em nuvem).

O tempo de resposta do processamento da segmentação de Antunes (2016), apresentado no gráfico da Figura 39 e que está representado na cor azul, mostra-se viável e rápido comparando-se ao tempo do gráfico da Figura 35. Constata-se que o InterCloud, para processar grandes volumes de dados nas arquiteturas modernas escaláveis, sendo um sistema distribuído, se sobressai pela viabilidade no tempo da segmentação, mas tem-se a necessidade para o uso da solução ter que programar-se com os custos financeiros para o processamento.

A solução da API *TA_Baatz_Segmenter_PostGIS* contrapõe-se como sendo uma solução de processamento em uma única máquina com uso de plataformas de *softwares* livres, e sem custos adicionais.

4.2 Resultados das segmentações

Com o propósito de verificar os resultados do funcionamento do método implementado denotam-se nas Figuras 40, 41, 42 e 43 as visualizações das bordas dos segmentos dos *tiles* e se encontram artefatos.

Utilizou-se como exemplo a imagem GeoEye, da área de estudo, definida para um recorte de *tiles* de 1.024 x 1.024 *pixels*, e a porção escolhida foram dois *tiles* representativos da área urbana para verificar os resultados do método. A Figura 40 representa uma parte do resultado de segmentação processada no PostGIS *Raster*, com os valores dos parâmetros de escala igual a 70, compacidade em 0,5 e cor no valor de 0,5.

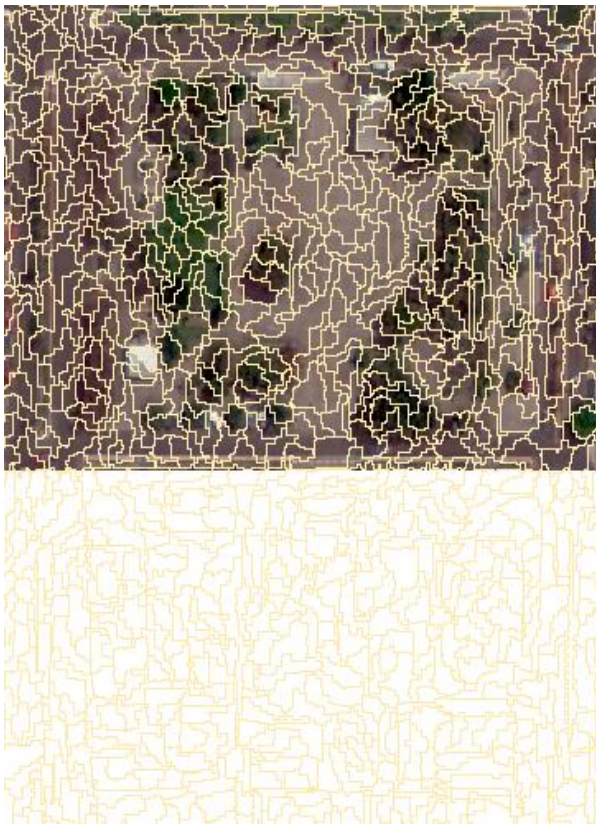
O processamento da segmentação via API *TA_Baatz_Segmenter_PostGIS* é realizado sobre toda a imagem, mesmo que a imagem esteja armazenada em *tiles* no SGBD-OR PostGIS *Raster*. A segmentação resultante tem como limite o tamanho da imagem original, e os segmentos podem ser armazenados ou exportados nos tamanhos definidos pelos *tiles* da imagem armazenada.

A Figura 41 representa a visualização de um *tile* da imagem e a sobreposição da segmentação gerada no PostGIS *Raster*. Para destacar os segmentos gerados denotam-se na Figura 42 dois *tiles* da imagem e a sobreposição das duas segmentações (delimitadas pelos mesmos *tiles*).

Figura 40: Parte da segmentação com os parâmetros de escala 70 – cor 0,5 – compacidade 0,5

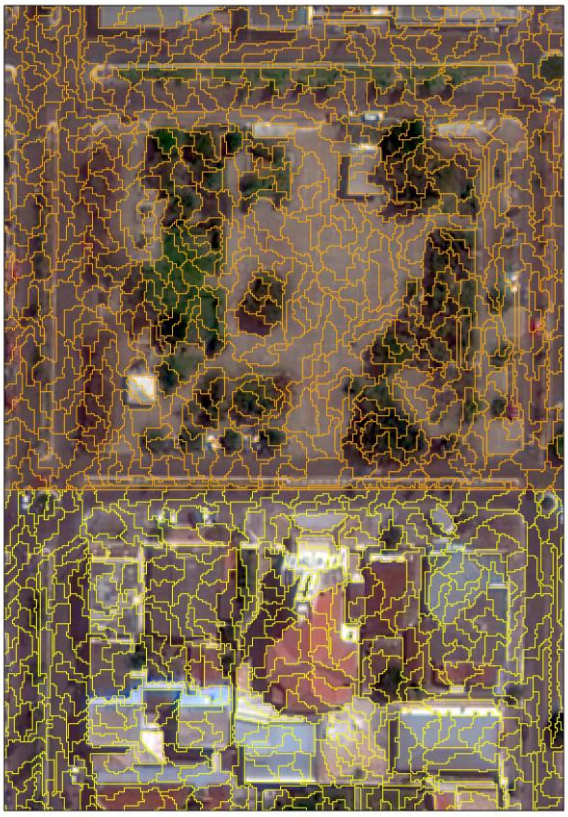
Fonte: Elaborado pela autora.

Figura 41: Imagem em *tile* e segmentação com os parâmetros de escala 70 – cor 0,5 – compacidade 0,5



Fonte: Elaborado pela autora.

Figura 42: Imagens em *tiles* e segmentação recortada no tamanho dos *tiles,* com os parâmetros de escala 70 – cor 0,5 – compacidade 0,5



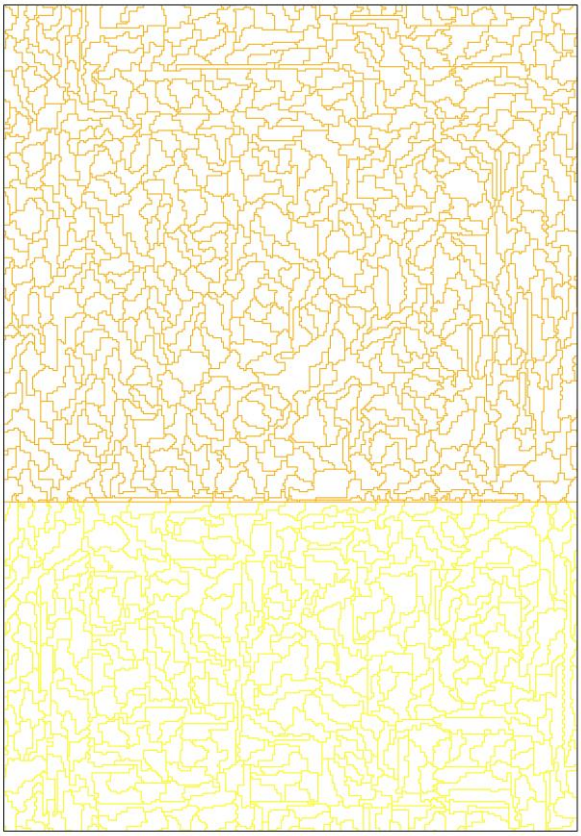
Fonte: Elaborado pela autora.

Os limites das bordas entre os resultados das segmentações, considerando-se o tamanho dos *tiles*, podem ser vistos na Figura 43.

Os experimentos realizados comprovaram ser possível processar a segmentação, baseado no método de Baatz e Schäpe (2000), de imagens da alta resolução espacial, no ambiente de gerenciamento de banco de dados PostgreSQL/PostGIS *Raster*.

Portanto, esta forma de processar o método de segmentação, definido neste estudo, não introduz artefatos indesejáveis decorrentes do processamento no ambiente SGBD PostGIS *Raster*. Ressaltando que uma imagem de 3,2 GBytes foi processada na sua totalidade nos experimentos processados.

Figura 43: Segmentação recortada no tamanho dos *tiles,* com os parâmetros de escala 70 – cor 0,5 – compacidade 0,5



Fonte: Elaborado pela autora.

4.3 Classificação por segmentação multirresolução

Utilizou-se o *software* InterIMAGE v. 1.43 para a classificação orientada a objetos - OBIA. Com os parâmetros usados no processo de segmentação, e identificados nas segmentações os melhores resultados, os mesmos foram utilizados para realização das classificações.

A rede semântica criada foi estabelecida a partir das definições de Antunes (2016). Para a caracterização das classes de interesses foram consideradas as características de ocupação do solo tais como: coberturas de edificações, solo exposto, vegetação, etc. E, para qualificar cada classe foram testados os parâmetros de cor, de forma, de tamanho, de textura e de localização. As regras estabelecidas adviram das definições do minerador WEKA em Antunes (2018).

O Quadro 4 apresenta os dados de entrada para definição dos parâmetros dos operadores *TA_Baatz_Segmenter*, *TA_NDVI_Segmenter* e *TA_Arithmetic* do *software* InterIMAGE 1.43. Para as classes de vegetação e sombra não foram

coletadas amostras, porque foram definidos os operadores para classificação, *TA_NDVI_Segmenter* e *TA_Arithmetic*, respectivamente. As classes da Figura 27 foram parametrizadas pelas regras criadas no *Node Editor – TopDown Decision Rule*, totalizando 32 classes.

Quadro 4: Parâmetros para as classes no InterIMAGE

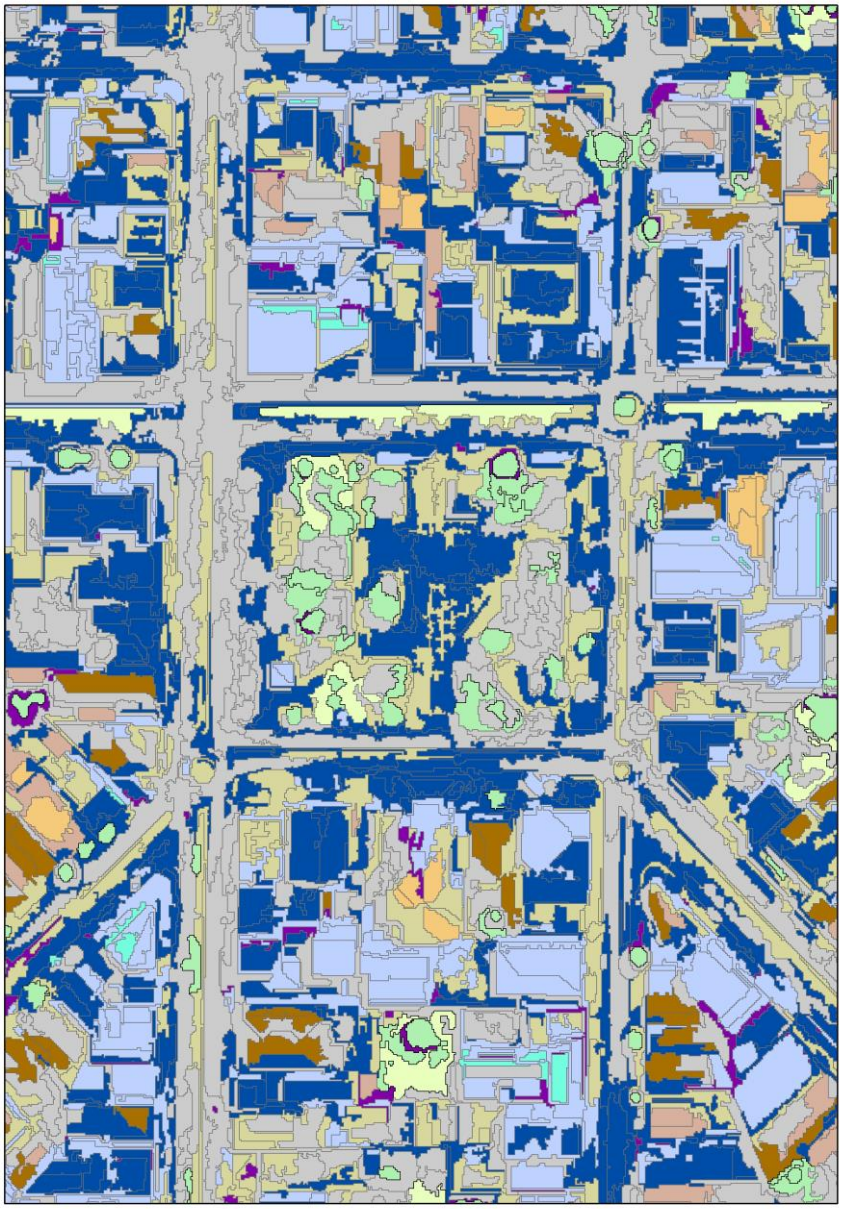
Classe	Bandas	Peso	Operador	Compacidade	Cor	Escala
Vegetação	0,1,2,3	1,1,1,1	*TA_NDVI_Segmenter*	NDVI: 0.65		
Rasteira	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Amianto	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Cerâmica Clara	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Cerâmica Escura	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Metálica	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Solo Exposto	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Piscina	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Asfalto	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Pavimento Concreto	0,1,2,3	1,1,1,1	*TA_ Baatz_Segmenter*	0.5	0.5	70
Sombra	0,1,2,3	1,1,1,1	*TA_Arithmetic*	(R0:1+R0:1+R0:2+R0:3)/4		
Não classificado	0,1,2,3	1,1,1,1	*Dummy TopDown*	*Reliability:* 0.1		

Fonte: Elaborado pela autora.

A Figura 44 apresenta o resultado da classificação no interIMAGE dos parâmetros estabelecidos na segmentação e as regras de decisão das classes definidas em Antunes (2018).

As classificações geradas para os *tiles* da imagem original são processadas independentes e apresentam artefatos, contínuos ou não, ao longo das fronteiras, ou bordas, dos *tiles.*

Figura 44: Classificação resultante da segmentação com os parâmetros de escala 70 – cor 0,5 – compacidade 0,5.



Fonte: Elaborado pela autora.

A ausência de comunicação entre os processos de classificação dos *tiles* podem apresentar como resultados ocorrências de artefatos, ou formações indesejáveis, ao longo das fronteiras dos *tiles*, e na Figura 45 apresenta-se um exemplo deste tipo de possível ocorrência.

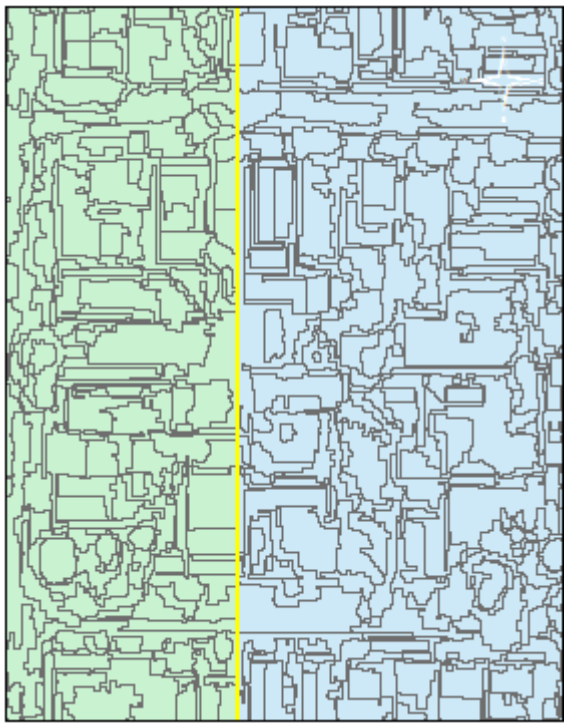
Observa-se na Figura 45 que alguns polígonos vizinhos nas bordas dos *tiles* não se tocam, e ou estão com classificações divergentes onde deveriam ser iguais. A fim de resolver esse problema, novas iterações com mudanças de regras da rede semântica devem ser consideradas, ou uma etapa de pós-processamento deve ser realizada.

Como sugestão do pós-processamento, primeiramente, pode-se realizar a união dos resultados da classificação de um *tile* com o *tile* vizinho. Após a união deve-se proceder com a verificação dos artefatos rotulados diferentes entre as fronteiras dos *tiles* unidos. A etapa seguinte poderá ser a decisão sobre a classe ser contígua, ou não, e caberá ao analista identificar quais artefatos deverão permanecer, serem alterados ou excluídos. É um processo simples que tende a apresentar um maior número de iterações do analista para obtenção do resultado final.

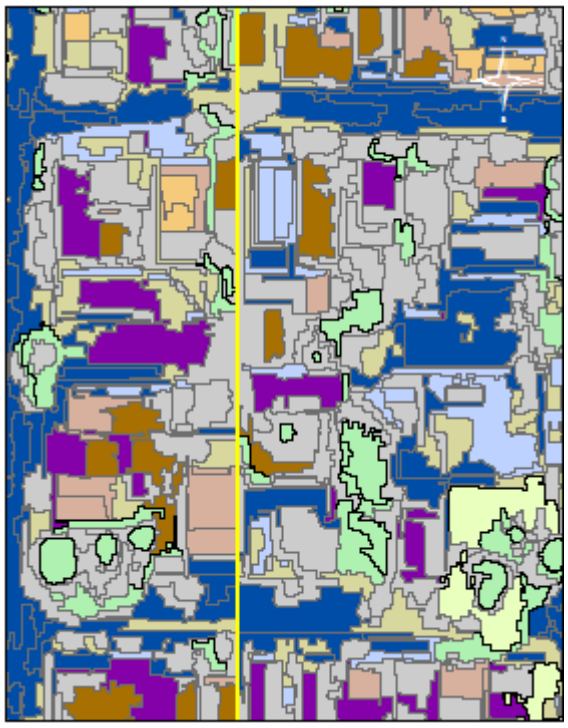
Figura 45: Classificação e as bordas dos *tiles* com exemplo de artefatos presentes.

a. Duas imagens (tiles) vizinhas, e em amarelo a linha que demonstra a divisão dos tiles.

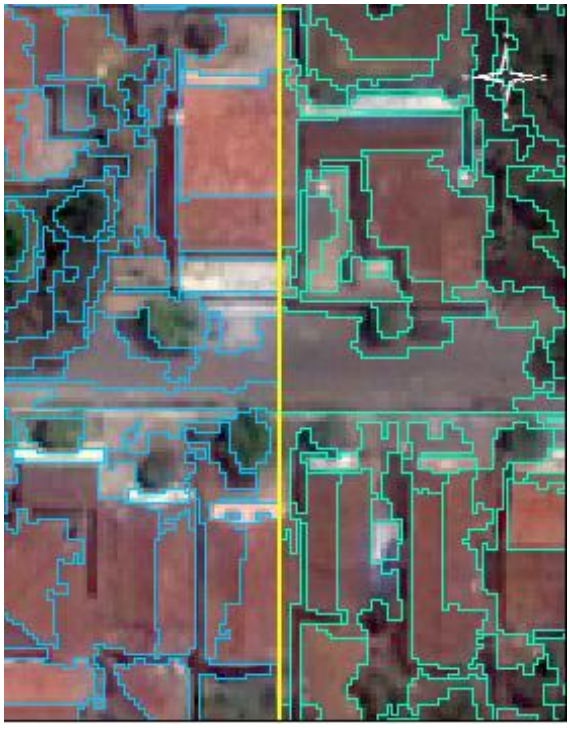
b. Duas imagens (tiles) e os segmentos classificados (tiles). Em azul parte de imagem, em verde parte de outra imagem, e a linha de divisão em amarelo.



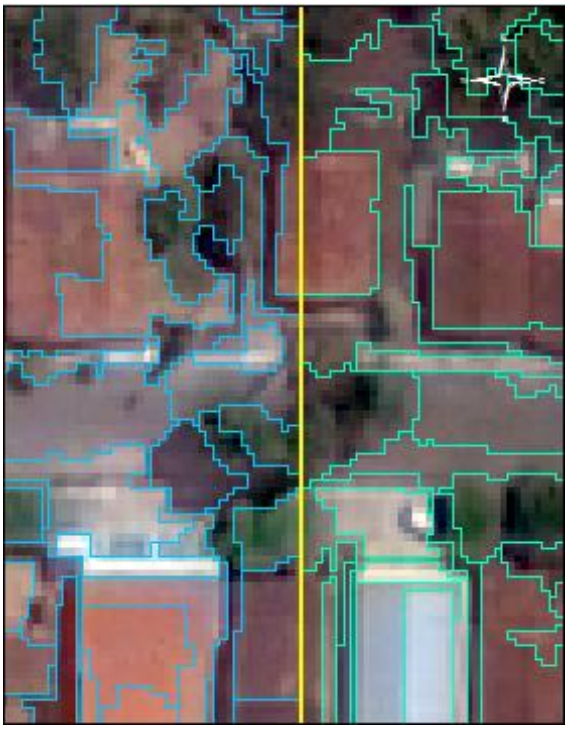
c. Duas classificações (tiles) vizinhas.



d. Duas classificações (tiles) vizinhas.



e. Exemplo de artefatos entre duas classificações entre os tiles.

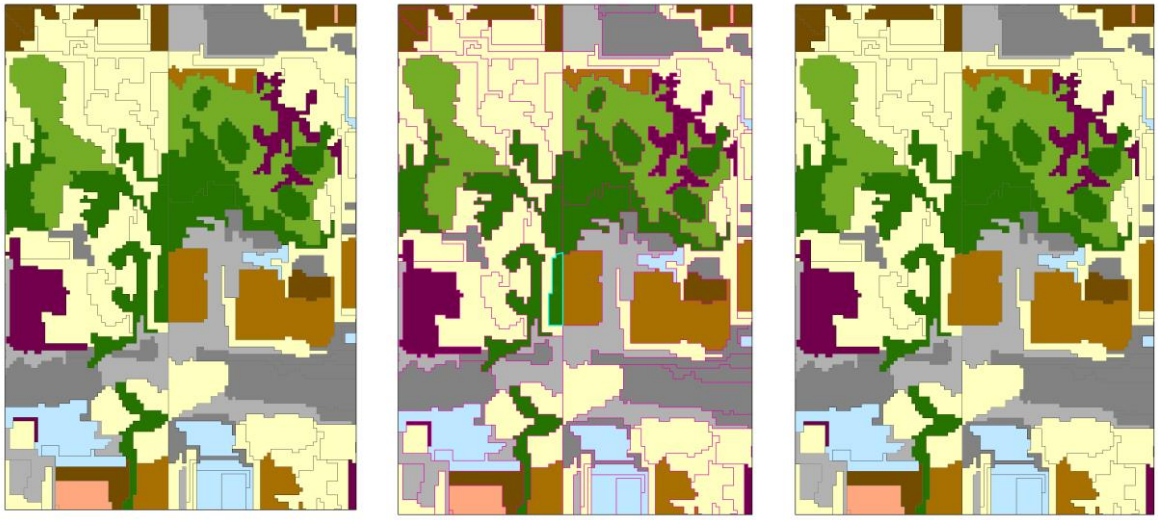


f. Outro exemplo de ocorrências de artefatos entre duas classificações entre os tiles

Fonte: Elaborado pela autora.

Aos demais resultados das classificações em *tiles* dever-se-á proceder conforme descrito anteriormente para obtenção da composição da classificação da imagem completa, ou seja, junção de todas as classificações dos *tiles*. Poderá ser utilizado um *software* de SIG, e no exemplo da Figura 46 foi utilizado o QGIS v.2.18.13.

Figura 46: Classificação e pós-processamento para correção de artefatos presentes.



a. Ocorrência de artefato, resultando em classificação errada.

b. Edição do artefato: divisão do polígono para nova classificação.

c. Resultado: polígono alterado para nova classe.

Fonte: Elaborado pela autora.

A segunda sugestão é refazer novos testes de classificações, com mudanças nos parâmetros ou regras para as classes, e executar o processamento com estes novos valores. E em seguida, proceder novamente com a verificação visual das bordas dos *tiles*. Caso ocorram novamente os artefatos nas bordas procede-se conforme a primeira sugestão até a conclusão da classificação.

A média de tempo estimada para processamento de cada *tile*, da classificação dos parâmetros da Figura 42, ficou em torno de 1.410 segundos (s). Para a imagem particionada em 1024 X 1024 *pixels* têm-se 399 *tiles*, totalizando para a imagem completa um tempo médio total de 562.590 s para a classificação total.

Comparando-se o tempo total médio para as classificações de todos os *tiles* da imagem completa da área de estudo, parte representados na Figura 42, com os tempos obtidos por Antunes (2016), representados na Figura 37, têm-se valores 562.590 s *versus* 44 s. Portanto, o tempo gasto para a classificação

segundo Antunes (2016) foram muito rápidos em comparação aos tempos dos experimentos desta tese.

O tempo de processamento no SGBD PostgreSQL pode ser significativo porque uma execução de uma transação computacional precisa passar pelo analisador de sintaxe da linguagem, o *driver*, o analisador de sintaxe do banco de dados, o planejador, o executor, o analisador de sintaxe novamente, retorna para a interface, passa pelo manipulador de dados do *driver* e para o cliente da aplicação. O PostgreSQL tem um *overhead* significativo por transação, incluindo o log de saída e as regras de acesso que precisam ser ajustadas em cada transação. Porém, o PostgreSQL consegue processar consultas complexas e longas com transações com vários comandos com fácil resolução de conflitos de concorrência.

Uma proposta melhoria de desempenho do processamento é utilizar um medidor de desempenho do PostgreSQL, como por exemplo o comando *standard – o pgbench*, para perceber quais os melhores parâmetros de configuração. Podem-se alterar os parâmetros e testar, observando-se qual a combinação de parâmetros que melhor desempenho se consegue. Complementando, devem-se testar os melhores parâmetros do *buffer* do banco de dados (*shared_pool, buffer cache, log buffer*, etc.), e a sugestão para prevenir frenquentes *reloads* é colocar objetos grandes e muito acessados em memória.

Para melhorar o desempenho de SGBDs principalmente têm-se: melhorar o hardware, com CPUs - *Central Process Unit* (Unidade Central de Processamento), RAM - *Random Access Memory* (Memória de Acesso Aleatório), discos mais novos, rápidos e confiáveis. E, a outra é otimizando as consultas realizadas nos bancos (usando *Vacuum, Vacuum Analyze*, *Explain,* criando *Clusters*, entre outros).

No computador as informações são manipuladas pelos registradores da CPU, pelo cache da CPU, pela memória RAM e pelos discos rígidos. Efetivamente, a otimização em bancos de dados envolvem aumento da quantidade de informações úteis na RAM, prevenindo acesso a disco sempre que possível.

Existem dois tipos de configuração de memória no PostgreSQL, a compartilhada e a individual. A compartilhada tem um tamanho fixo, ela é alocada sempre que o PostgreSQL inicializa e então é compartilhada por todos

os clientes. Já a memória individual é tem um tamanho variável e é alocada separadamente para cada conexão feita ao SGBD.

As classificações obtidas dos *tiles*, da imagem original, podem ficar gravadas em disco, ou armazenadas no PostGIS *Raster*. Na opção de os arquivos das classificações serem armazenados no PostGIS *Raster*, são realizadas as inserções dos dados em tabelas de vetores. Posteriormente, pode-se realizar a operação de *merge*, função que une as feições de mesmas características, das tabelas, para unificar a classificação. Tem-se, ainda, a opção de exportar esta unificação como um só arquivo para gravar em disco rígido. Estas funções são nativas do PostgreSQL/PostGIS *Raster*.

Para a visualização do arquivo completo da classificação dos *tiles* deve-se utilizar um *software* de sistemas de informações geográficas, como por exemplo o QGIS, e caso necessário para a elaboração de layout de impressão.

CAPÍTULO 5
CONCLUSÃO

Uma nova forma de processamento de segmentação, segundo o conceito OBIA, de imagem de alta resolução espacial foi apresentada nesta tese por meio de uma implementação de API desenvolvida em C++ para processar dentro de um sistema gerenciador de banco de dados com extenção geográfica. A base principal de bibliotecas da implementação da aplicação fez uso da TerraLib 5.2.2.

A API proposta serviu de complemento para interação da biblioteca TerraLib 5.2.2 com outros sistemas, neste caso o InterIMAGE 1.43, e um novo modelo de integração baseado em banco de dados geográficos, como o PostgreSQL. Portanto, amplia-se a disponibilidade de ferramentas aos usuários de *software* livre para a construção e utilização de aplicativos geográficos. E, esta integração elaborada deu-se através de inclusão de novos processos a sistemas já consolidados.

Com o desenvolvimento da API *TA_Baatz_Segmenter_PostGIS* possibilitou-se processar a segmentação segundo o conceito de Baatz e Schäpe (2000), configurado o armazenamento de imagem de alta resolução espacial em sistema gerenciador de banco de dados geográficos, sendo que nesta tese, utilizou-se o PostgreSQL/PostGIS *Raster*. A *interface* implementada fornece, via *driver*, o acesso aos dados *raster*, e com passagem de parâmentros para execução da segmentação, bem como os mecanismos de acesso de usário e senha.

O modelo de armazenamento de *raster* no sistema gerenciador de banco de dados geográficos leva em conta questões relativo à eficiência no seu armazenamento e recuperação de dados, condicionados a existência de extensão espacial no SGBD.

A API respondeu ao processamento de arquivos *raster* de tamanhos superiores a 3.000 x 3.000 *pixels* através de acionamento no ambiente do InterIMAGE 1.43, bem como à passagem de parâmetros como de similaridade, compacidade, escala, cor, e processamento do *raster* armazenado em *tiles* definidos pelo usuário. Os resultados apresentados sofrem alteração se houver

mudança dos parâmetros de processamento relativos aos parâmetros mencionados aqui.

A biblioteca TerraLib 5.2.2 possibilitou a interação com novos componentes sendo usados diretamente como objetos. Neste sentido, pode-se acessar o sistema de gerenciamento de banco de dados geográficos de forma segura e executar funções de geoprocessamento enquanto lê-se, ou manipulam-se, os dados. A TerraLib 5.2.2 tem uma funcionalidade ampla para análises espaciais integradas, e os experimentos realizados demonstraram a possibilidade de realização completa da segmentação de dados *raster* de tamanhos de arquivos superiores a 3 GB no sistema gerenciador de banco de dados.

A possibilidade do PostgreSQL/PostGIS *Raster* armazenar imagens particionadas, e em multirresolução, permitem que o sistema gerenciador de banco de dados geográficos possa ser acessado por aplicações desktop, web, ou aplicações multi-usuários.

Complementando, a verificação dos resultados obtidos no processamento da segmentação, utilizaram-se os parâmetros e dados gerados (segmentos) para a classificação dos *tiles* da imagem com definição de classes e da rede semântica estabelecida. Os resultados das classificações mostraram-se satisfatórios visivelmente nas comparações estabelecidas. As conexões (junções) dos dados dos resultados dos *tiles* segmentados e classificados foram possíveis de serem realizados e armazenados em disco e no PostGIS *Raster*.

Os tempos de processamento ficaram condicionados às variações dos parâmetros de escala, cor, compacidade, os tamanhos das divisões da imagem em *tiles*, e às padronizações das variáveis de processamento discutidos no item 3.2.3, resultando em valores significativos em relação ao tempo de execução da API.

Um fator que pode interferir na perfomance é em qual sistema operacional está rodando a aplicação, neste caso o Windows 10, porque depende diretamente das configurações de memória disponível e disco rígido – com espaço para processamento e necessidade de desfragmentação do disco para melhorar o desempenho.

Tecnologias como o ICP e sistemas de gerenciadores de banco de dados não convencionais, tais como o NoSQL podem ser soluções alternativas para a resolução da questão do processamento da segmentação de imagem de alta

resolução espacial. Entretanto, faz-se necessário pesquisar quais atendem às operações geográficas para dados matriciais (*raster*) e vetores.

Ressalta-se que no desenvolvimento do trabalho foi necessário a utilização de um tempo considerável para instalar e configurar as ferramentas e *softwares* essenciais para as implementações das aplicações, bem como demandado tempo nos processamentos para atingir-se os objetivos do trabalho.

Concluindo, os métodos incrementados nesta tese encontram-se alinhados com o funcionamento do InterIMAGE 1.43 e podem ser acoplados como operadores externos.

5.1 Trabalhos futuros

As sugestões de trabalhos futuros estão nas áreas de incrementar as operações de classificação do InterIMAGE 1.43 para serem processadas no ambiente PostGIS *Raster*, e acoplamento de novas funcionalidades como: acesso à leitura de *raster* com tamanho superior a 3.000 x 3.000 *pixels*; leitura de geometrias resultantes de segmentação de *raster* com tamanho superior a 3.000 *pixels* x 3.000 *pixels* com acesso automatizado para dados armazenados em sistemas gerenciadores de banco de dados geográficos PostgreSQL; automatização dos processos para integração com o *software* Weka; e aplicação para construção de matriz de confusão das classificações.

De forma complementar, é desejável a realização de experimentos envolvendo uma avaliação sistemática da qualidade da segmentação em relação ao tamanho dos *tiles*. Os resultados da segmentação, com o processamento do *raster* armazenado em SGBDs, devem ser comparados com os resultados advindos da versão armazenada em disco, a fim de garantir maior validação de qualidade. Também, propõem-se experimentações com imagens orbitais ópticas de maiores resoluções espaciais.

Complementando, sugere-se a implementação em ambiente de SGBD para outras técnicas de segmentações, tal como crescimento de regiões, demonstrando as adaptabilidades acopladas às funcionalidades de sistemas gerenciadores de banco de dados com extensão geográfica.

Um aspecto importante é avaliar o tempo computacional associado à execução da segmentação com dados *raster* armazenados em sistemas

gerenciadores de banco de dados geográficos. O Sistema Operacional (SO) também interfere na performance de tempo de processamento das funcionalidades da API e PostgreSQL, indicando uma necessidade de avaliação da adoção do SO e sua respectiva versão.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, C. A.; COUTINHO, A. C.; ESQUERDO, J. C. D. M.; ADAMI, M.; VENTURIERI, A.; DINIZ, C. G.; DESSAY, N.; DURIEUX, L.; GOMES, A. R. High spatial resolution land use and land cover mapping of the Brazilian Legal Amazon in 2008 using Landsat-5/TM and MODIS data. **Acta Amazonica**, v. 46, n. 3, p. 291-302, 2016.

ALRASSI, F.; SALIM, E.; NINA, A.; ALWI, L.; DANOEDORO, P.; KAMAL, M. GEOBIA for land use mapping using WorldView2 image in Bengkak Village Coastal, Banyuwangi Regency, East Java. **IOP Conference Series: Earth and Environmental Science**, v. 47, 2016, 12 p.

ANTONIO, M. M. Z. **Uma metodologia para análise de imagens usando segmentações específicas por classe**. Tese de doutorado em Engenharia Elétrica. Rio de Janeiro: PUC-Rio, 2013, 169 p.

ANTUNES, R. R.; HAPP, P. N.; BIAS, E. S.; BRITES, R. S.; COSTA, G. A. O. P.; FEITOSA, R. Q. *An object-based image interpretation. Application on cloud computing infrastructure.* In: ***GEOBIA 2016: Solutions and Synergies***, 14 - 16 Sept. 2016, University of Twente. Faculty of Geo-Information and Earth Observation (ITC).

ANTUNES, R. R.; BIAS, E. D. S.; COSTA, G. A. O. P. D.; BRITES, R. S. Object-based analysis for urban land cover mapping using the InterImage and the Sipina free software packages. **Boletim de Ciências Geodésicas**, v. 24, n. 1, p. 1-17, 2018.

BAATZ, M.; SCHÄPE, A. Multiresolution segmentation: an optimization approach for high quality multi-scale image segmentation. In: **Angewandte Geographische Informations verarbeitung**, 12. Wichmann-Verlag, Heidelberg, 2000. Disponível em: <http://www.ecognition.com/sites/default/files/405_baatz_fp_12.pdf>. Acessado em: 13/05/2018.

BATORY, D. Principles of database management system extensibility. **IEEE Database Engineering Bullet**, v. 2, p. 40-46, 1987.

BENZ, U. C.; HOFMANN, P.; WILLHAUCK, G.; LINGENFELDER, I.; HEYNEN, M. Multi-resolution, object-oriented fuzzy analysis of remote sensing data for GIS-ready information. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 58, n. 3, p. 239-258, 2004.

BERNSTEIN, P. A.; LOMET, D. B. Principles of database management system extensibility. **IEEE Database Engineering Bullet**, v. 10, p. 2-9, n. 2, 1987.

BLASCHKE, T. Object-based image analysis for remote sensing. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 65, n. 1, p. 2-16, 2010.

BLASCHKE, T.; HAY, G. J. Object-oriented image analysis and scale-space: theory and methods for modeling and evaluating multiscale landscape structure. **International Archives of Photogrammetry and Remote Sensing**, v. 34, n. 4, p. 22-29, 2001.

BLASCHKE, T. LANG, S. LORUP, E. STROBL, J. ZEIL, P. Object-oriented image processing in an integrated GIS/remote sensing environment and perspectives for environmental applications. **Environmental Information for Planning, Politics and the Public**, v. 2, p. 555-570, 2000.

BLASCHKE, T.; HAY, G. J.; KELLY, M.; LANG, S.; HOFMANN, P.; ADDINK, E.; FEITOSA, R. Q.; MEER, F. V. D.; WERFF, H. V. D.; COILLIE, F. V.; TIEDE, D. Geographic object-based image analysis–towards a new paradigm. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 87, p. 180-191, 2014.

CÂMARA, G.; DAVIS, C.; CASANOVA, M. A.; QUEIROZ, G. R. D. **Bancos de Dados Geográficos**. Curitiba: MundoGEO, 2005.

CAMARA, G.; SOUZA, R. M.; PEDROSA, B. M.; VINHAS, L.; MONTEIRO, A. M.; PAIVA, J. C.; CARVALHO, M. T.; GATTASS, M. **TerraLib: Technology in support of GIS innovation**. In: Workshop Brasileiro de GeoInformática, São Paulo, Brasil, p. 126-133, 2000.

CAREY, M. J.; DEWIT, D. The architecture of the EXODUS extensible DBMS. In: 1986. International Workshop on Object-Oriented Database Systems. **Proceedings.** IEEE Computer Society Press, 1986, p. 52-65.

CASANOVA, M.; CÂMARA, G.; DAVIS, C.; VINHAS, L.; QUEIROZ, G. R. **Bancos de Dados Geográficos**. MundoGeo, 2005. 506 p. Disponível em: <http://www.dpi.inpe.br/livros/bdados/>. Acesso em: 02/05/2014.

CHUBEY, M. S.; FRANKLIN, S. E.; WULDER, M. A. Object-based analysis of Ikonos-2 imagery for extraction of forest inventory parameters. **Photogrammetric Engineering & Remote Sensing**, v. 72, n. 4, p. 383-394, 2006.

COLEMAN, G. B.; ANDREWS, H. C. Image segmentation by clustering. **Proceedings of the IEEE**, v. 67, n. 5, p. 773-785, 1979.

COSTA, G. A. O. P.; FEITOSA, R. Q.; FONSECA, L. M. G.; OLIVEIRA, D. A. B.; FERREIRA, R. S.; CASTEJON, E. F. Knowledge-based interpretation of remote sensing data with the InterIMAGE system: major characteristics and recent developments. In: INTERNATIONAL CONFERENCE ON GEOGRAPHIC OBJECT-BASED IMAGE ANALYSIS (GEOBIA 2010), 3. Ghent, Bélgica. **Proceedings...** The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, v, XXXVII, 2010, 6 p.

CROSTA, A. P. **Processamento Digital de Imagens de Sensoriamento Remoto**. Campinas: UNICAMP, 1993.

DATE, C. J. **Introdução a Sistema de Banco de Dados**. Rio de Janeiro: Elsevier Brasil, 2004.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, v. 51, n. 1, p. 107-113, 2008.

DEFINIENS IMAGING. **eCognition**. Disponível em: <http://www.ecognition.com/>. Acesso em: 03/10/2016.

DEY, V.; ZHANG, Y.; ZHONG, M. A review on image segmentation techniques with remote sensing perspective. ISPRS TC VII Symposium – 100 Years ISPRS, Vienna, Austria, July 5–7, 2010. **Proceedings…** IAPRS, Vol. XXXVIII, Part 7A, 2010.

ELDAWY, A.; MOKBEL, M. F. **Pigeon: A spatial mapreduce language**. In: 2014 IEEE 30th International Conference on Data Engineering (ICDE). IEEE, 2014. p. 1242-1245.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. Boston: Pearson-Addison-Wesley, 6ª ed., 2010.

ESRI. Environmental Systems Research Institute. **ArcGIS Professional GIS for the desktop**, v. 10.4, 2016.

FERREIRA, R. **Uma abordagem multiescalar, multicritério para a segmentação de imagens**. Dissertação de mestrado em Engenharia Elétrica, Rio de Janeiro: PUC-Rio, 2011, 113 p.

FERREIRA, R. da S.; COSTA, G. AOP; FEITOSA, R. Q. **Avaliação de critérios de heterogeneidade baseados em atributos morfológicos para segmentação de imagens por crescimento de regiões**. Boletim de Ciências Geodésicas, v. 19, n. 3, 2013.

FÖRSTER, M.; KLEINSCHMIT, B. **Object-based classification of QuickBird data using ancillary information for the detection of forest types and NATURA 2000 habitats**. Springer Berlin Heidelberg, 2008, p. 275-290.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**, New Jersey: Addison-Wesley, 1995.

GDAL. **Geospatial Data Abstraction Library**. Disponível em: <http:// www.gdal.org/>. Acesso em: 21/07/2018.

GONZALEZ, R. C.; WOODS, R. E. **Processamento Digital de imagens**. Pearson, São Paulo, 3ª ed., 2010.

GÜTING, R. H. An introduction to spatial database systems. **The International Journal on Very Large Data Bases**, v. 3, n. 4, p. 357-399, 1994.

HAPP, P. N.; FERREIRA, R. S.; BENTES, C.; COSTA, G. A. O. P.; FEITOSA, R. Q. Multiresolution segmentation: a parallel approach for high resolution image segmentation in multicore architectures. **The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, v. 38, n. 4, p. C7, 2010.

HARALICK, R. M.; SHAPIRO, L. G. Survey: image segmentation techniques. **Computer Vision, Graphics, and Image Processing**, v. 29, n. 1, p. 100-132, 1985.

HAY, G. J.; CASTILLA, G. Geographic Object-Based Image Analysis (GEOBIA): A new name for a new discipline. In: BLASCHKE, T.; LANG, S.; HAY, G. (Eds.), **Object Based Image Analysis**. Nova York: Springer, 2008, p. 93–112.

HEUSER, C. A. **Projeto de Banco de Dados**. Porto Alegre: Sagra Luzatto, 5ª ed., 2004.

IMRAN, M. **Extending an open source spatial database with geospatial image support: An image mining perspective**. Tese de doutorado, International Institute for Geo-information Science and Earth Observation (ITC). Enschede, The Netherlands, 2009.

INPE. **Manual do Spring. Tutorial de Geoprocessamento. Classificação de imagens**. São José dos Campos: INPE, 1997. Disponível em: <http://www.dpi.inpe.br/spring/portugues/tutorial/classific.html>. Acesso em: 16/05/2016.

INPE. **TerraLib 5 Code Documentation.** São José dos Campos: INPE, 2018. Disponível em: <http://www.dpi.inpe.br/terralib5/wiki/doku.php?id=start>, acesso em: 26/03/2018.

InterIMAGE. **InterIMAGE – Interpreting images freely**. Laboratório de Visão Computacional da Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro, Brasil, 2010. Disponível em: http://www.lvc.ele.puc-rio.br/projects/interimage/index.html. Acesso em: 03/10/2016.

InterIMAGE Cloud Platform. **InterIMAGE Cloud Platform**. Laboratório de Visão Computacional da Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro, Brasil, 2010. Disponível em: http://www.lvc.ele.puc-rio.br/wp/?p=2419. Acesso em: 12/05/2018.

ISO. International Organization for Standardization. **ISO/IEC 9075-1:2011: Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework)**, 2011. Disponível em: https://www.iso.org/standard/53681.html. Acesso em: 16/07/2018.

KIRAN, M.; KUMAR, A.; MUKHERJEE, S.; PRAKASH, R. Verification and validation of MapReduce program model for parallel support vector machine. **International Journal of Computer Science Issues**, v. 10, n. 3, p. 317-325, 2013.

LINNEMANN, V.; KUSPERT, K.; DADAM, P.; PISTOR, P.; ERBE, R.; KEMPER, A.; SUDKAMP, N.; WALCH, G.; WALLRATH, M. Design and implementation of an extensible database management system supporting user defined data types and functions. In: VLDB '88, Los Angeles, California, USA. 14th International Conference on Very Large Data Bases. **Proceedings…** San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, p. 294-305.

MANAKOS, I.; SCHNEIDER, T.; AMMER, U. A comparison between the ISODATA and the eCognition classification methods on basis of field data. **International Archives of Photogrammetry and Remote Sensing**. v. 33, p. 133-139, 2000.

MENESES, P. R. Formatos das Imagens de Sensoriamento Remoto. In: MENESES, P. R.; ALMEIDA, T. (Orgs.). **Introdução ao Processamento de Imagens de Sensoriamento Remoto.** Brasília: CNPq/UnB, 2012, p. 77-81.

MENESES, P. R.; ALMEIDA, T. **Introdução ao Processamento de Imagens de Sensoriamento Remoto**. Brasília: UnB/CNPq, 2012, 276 p.

MENESES, P. R.; SANO, E. E. Classificação Pixel a Pixel de Imagens. In: MENESES, P. R.; ALMEIDA, T. (Orgs.). **Introdução ao Processamento de Imagens de Sensoriamento Remoto.** Brasília: CNPq/UnB, 2012, p. 191-208.

MÖLLER, M.; LYMBURNER, L.; VOLK, M. The comparison index: A tool for assessing the accuracy of image segmentation. **International Journal of Applied Earth Observation and Geoinformation**, v. 9, n. 3, p. 311-321, 2007.

NIKHIL, R. P.; SANKAR, K. P. A review on image segmentation techniques. **Pattern Recognition**, v. 26, n. 9, p. 1277-1294, 1993.

OBE, R. O.; HSU, L. S. **PostGIS in Action: MEAP Edition Manning Early**. Manning Publications Co. Access Program, 2009, 425 p. Disponível em: <http://www.manning.com>. Acesso em: 30/05/2014.

OBE, R.; HSU, L**. PostGIS in Action**. Stanford. 2011.

OPEN GIS **Consortium. OpenGIS simple features specification for SQL**. Technical Report Revision, v. 1, 1999.

ORDONEZ, C. Statistical model computation with UDFs. **IEEE Transactions on Knowledge and Data Engineering**, v. 22, n. 12, p. 1752-1765, 2010.

OSBORN, S. Principles of database management system extensibility. **IEEE Database Engineering Bullet**, v. 10, n. 2, p. 10-15, 1987.

PAHL, M. **Arquitetura de um sistema baseado em conhecimento para a interpretação de dados de sensoriamento remoto de múltiplos sensores**. Tese de doutorado em Sensoriamento Remoto. São José dos Campos: INPE, 2008.

PASSO, D. P. **Análise da qualidade de classificadores para identificação de alvos urbanos em imagens de alta resolução espacial: uma aplicação com as imagens do satélite Worldview II.** Dissertação de mestrado em Geociências Aplicadas. Brasília: UnB, 2013. 106 f.

PINHO, C. M. D.; FEITOSA, F. F.; KUX, H. Classificação automática de cobertura do solo urbano em imagem IKONOS: Comparação entre a abordagem pixel-a-pixel e orientada a objetos. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO (SBSR), 12. Goiânia, Go. **Anais…** São José dos Campos: INPE, p. 4217-4224, 2005**.**

PIRKELBAUER, P.; SOLODKYY, Y.; STROUSTRUP, B. **Open multi-methods for C++.** In: 6th INTERNATIONAL CONFERENCE ON GENERATIVE PROGRAMMING AND COMPONENT ENGINEERING. **Proceedings…** ACM, 2007, p. 123-134.

POSTGIS. **PostGIS Manual**. Disponível em: <http://postgis.net/docs/>. Acesso em: 20/07/2018.

POSTGRESQL. **PostgreSQL**. Disponível em: <http://www.PostgreSQL.org>. Acesso em: 18/07/2018.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de Gerenciamento de Banco de Dados.** 3ª ed., AMGH Editora, 2008.

RAMSEY, P. **PostGIS 2.3.1 Manual**. Disponível em: <http:// http://download.osgeo.org/postgis/docs/postgis-2.3.1.pdf>. Acesso em: 30 jan. 2017.

RICHARDS, J. A.; JIA, X. **Remote Sensing Digital Image Analysis**. **An Introduction.** Germany: Springer, 2006.

SAMPLE, J. R.; IOUP, E. Image Tile Creation. In: SAMPLE, J. T.; IOUP, E. **Tile-Based Geospatial Information Systems**. Springer, Boston, MA, 2010, p. 81-95.

SCHROEDER, W.; MARTIN, K.; LORENSEN, B. The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. 4ª ed., Kitware, 2006.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados**. São Paulo: Makron Books, 2006.

SPT. Segmentation Parameter Tuner. Disponível em: < http://www.lvc.ele.puc-rio.br/wp/?p=1403>. Acesso em: 10/06/2017.

STONEBRAKER, M. Inclusion of new types in relational data base systems. In: SECOND INTERNATIONAL CONFERENCE ON DATA ENGINEERING. Washington, DC, USA. **Proceedings**… IEEE Computer Society, 1986, p. 262-269.

STONEBRAKER, M. A. J.; HIROHAMA, M. Principles of database management system extensibility. **IEEE Database Engineering Bullet**, v. 10, p. 16-23, 1987.

STROUSTRUP, B. **C++ Programming Language**. Pearson Education, 2013.

SUMMERFIELD, M. **Advanced Qt Programming: Creating Great Software with C++ and Qt 4**, Addison-Wesley, 500 p., 2010.

TERRA AMAZON. **TERRA AMAZON**. Disponível em: <http://www.TerraLib.org/>. Acesso em: 02/06/2017.

TERRALIB. **TERRALIB**. Disponível em: <http://terraamazon.org/index.php/pt/sobre />. Acesso em: 16/01/2017.

TERRACLASS. **Projeto Terraclass, 2014**. Disponível em: <http://www.inpe.br/noticias/noticia.php?Cod_Noticia=3302>. Acesso em: 15/05/2017.

TZOTSOS, A.; ARGIALAS, D. MSEG: A generic region-based multi-scale image segmentation algorithm for remote sensing imagery. In: ASPRS 2006 Annual Conference, **Proceedings**… Reno, Nevada. 2006.

Villarroya, S. Viqueira, J.R.R. Regueiro, M.A. Taboada, J.A. and Cotos, J.M. **SODA: A framework for spatial observation data analysis**. Distributed and Parallel Databases, 34(1). 2016, pages 65-99.

VISUAL LEARNING SYSTEMS. **Feature Analyst**. Disponível em: <http://www.vls-inc.com/>. Acesso em: 03/10/2016.

WASSENBERG, J.; MIDDELMANN, W.; SANDERS, P. An efficient parallel algorithm for graph-based image segmentation. In: INTERNATIONAL CONFERENCE ON COMPUTER ANALYSIS OF IMAGES AND PATTERNS (CAIP 2009), 13. , Munster, Alemanha. **Proceedings**… 2009, p. 1003-1010.

YEUNG, A. K. W.; HALL, G. B. **Spatial Database Systems: Design, Implementation and Project Management**. Springer Science & Business Media, 2007.

YONG, X.; FENG, D.; RONGCHUN, Z. **Optimal selection of image segmentation algorithms based on performance prediction**. Sydney: Australian Computer Society, Inc. 2004.

YU, J.; WU, J.; SARWAT, M. **Geospark: A cluster computing framework for processing large-scale spatial data**. In: Proceedings of the 23rd SIGSPATIAL

International Conference on Advances in Geographic Information Systems. ACM, 2015. p. 70.

Zhang, Y. Kersten, M.L. and Manegold, S. **SciQL: Array Data Processing Inside An RDBMS**. In Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'13). 2013. p. 1049 -1052. ACM, New York, NY, USA.

ZADEH, L. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338-353, 1965.

ZHONG, C.; ZHONGMIN, Z.; DONGMEI, Y.; RENXI, C. Multi-scale segmentation of the high resolution remote sensing image. In: Geoscience and Remote Sensing Symposium, 2005. IGARSS'05. **Proceedings..**. 2005 IEEE International. IEEE, 2005. p. 3682-3684.

APÊNDICE A – *Softwares* utilizados na configuração do ambiente de desenvolvimento da API

a) Sistema Operacional Windows;

b) Pacote de programas da Microsoft Visual Studio C++ 2013 para desenvolvimento de *software* especialmente dedicado ao .NET Framework e às linguagens Visual Basic (VB), C, C++, C# (C Sharp), J# (J Sharp) e a plataforma do ASP.NET - *Active Server Pages NET*.;

c) Conjunto de ferramentas CMake versão 3.9.2 64 bits para controlar o processo de compilação de *software* usando arquivos de configuração independentes de plataforma e compilador e gerar *makefiles* e espaços de trabalho nativos que podem ser usados no ambiente de compilação escolhido;

d) Biblioteca Boost, versão 1.65.1, onde se estendem a funcionalidade da biblioteca TerraLib e linguagem de programação C++;

e) OSGeo4W: É uma biblioteca com distribuição binária de um amplo conjunto de *software* geoespacial de código aberto para ambientes Windows;

f) Função de filtro UNIX (acrônimo) padrão Proj-4.9.3 que converte as coordenadas de latitude e longitude em coordenadas cartesianas (e vice-versa). Trata-se de uma API em C para desenvolvedores de *software* que incluem transformação de coordenadas em seu próprio ambiente;

g) Biblioteca de tradutores GDAL versão 2.2.1 para formatos de dados geoespaciais de *rasters* e vetores que é lançado sob uma licença *Open Source* de estilo X/MIT pela *Open Source Geospatial Foundation*. Por ser uma biblioteca, apresenta um único modelo de dados abstratos de *raster* e um modelo de dados de resumo de vetor único para o aplicativo de chamada para todos os formatos suportados. Ele também vem com uma variedade de utilitários de linha de comando para tradução e processamento de dados;

h) Sistema de gerenciamento de banco de dados geográficos PostgreSQL versão 9.6.5-1;

i) Extensão PostGIS versão 2.3.2;

j) Interface gráfica PgAdmin 4 do PostgreSQL;

k) Biblioteca de classes e funções TerraLib 5.2.2, escrita na linguagem C++, para a construção de aplicações geográficas. É distribuída na internet com código fonte aberto e livre de licença ou restrição de uso;

l) *Framework* multi-plataforma InterIMAGE versão1.43, escrito em linguagem de programação C++, atualmente com implementações para sistemas operacionais LINUX e Windows. Esse *framework* conta com suporte à integração de operadores de processamento de imagem no processo de interpretação. Tais operadores são tratados como programas externos por seu mecanismo de controle, podendo ser codificados em qualquer linguagem de computador. Oferece um repositório de operadores, chamado TerraAIDA (http://www.dpi.inpe.br/TerraAIDA), montado com classes de *software* e funções fornecidas pela biblioteca TerraLib 5.

APÊNDICE B - Código de conexão ao banco de dados PostGIS via Terralib 5

```
// Examples
#include "DataAccessExamples.h"

// TerraLib
#include <terralib/dataaccess/datasource/DataSourceFactory.h>
#include <terralib/datatype.h>

// STL
#include <iostream>
#include <exception>

std::unique_ptr<te::da::DataSource> GetPostGISConnection()
{
	// let's give the minimal server connection information needed to connect to the database server
	std::string aux, user, password, host, port, path, query;
	std::string strURI = "pgsql://"; // The base of the URI

	std::cout << "Informe o Host do servidor postGIS (ENTER para aceitar padrao \'localhost\'): ";
	std::getline(std::cin, aux);
	host = aux.empty() ? "localhost" : aux;

	std::cout << "Informe o numero da porta para acessar o servidor postGIS (ENTER para aceitar padrao \'5432\'): ";
	std::getline(std::cin, aux);
	port = aux.empty() ? "5432" : aux;

	std::cout << "Informe o usuario para acessar o servidor postGIS (ENTER para aceitar padrao \'postgres\'): ";
	std::getline(std::cin, aux);
	user = aux.empty() ? "postgres" : aux;

	std::cout << "Informe a senha para acessar o servidor postGIS (ENTER para aceitar padrao \'postgresql\'): ";
	std::getline(std::cin, aux);
	password = aux.empty() ? "postgresql" : aux;

	std::cout << "Informe o nome do Banco de Dados para conectar o servidor postGIS (ENTER para aceitar padrao \'postgis_232\'): ";
	std::getline(std::cin, aux);
	path = aux.empty() ? "postgis_232" : aux;

	std::cout << "Informe o cliente enconding para conectar o servidor postGIS (ENTER para aceitar padrao \'UTF-8\'): ";
	std::getline(std::cin, aux);
	query = aux.empty() ? "&PG_CLIENT_ENCODING=" + te::core::CharEncoding::getEncodingName(te::core::EncodingType::UTF8) : aux;
```

```
	std::cout << "Inform the Connection Time Out to connect to your postGIS server
(ENTER para aceitar padrao \'4\'): ";
	std::getline(std::cin, aux);
	query += aux.empty() ? "&PG_CONNECT_TIMEOUT=4" :
"&PG_CONNECT_TIMEOUT=" + aux;

	strURI += user + ":";
	strURI += password + "@";
	strURI += host + ":";
	strURI += port + "/";
	strURI += path + "?";
	strURI += query;

	// create a data source using the data source factory
	std::unique_ptr<te::da::DataSource> ds =
te::da::DataSourceFactory::make("POSTGIS", strURI);

	try
	{
		ds->open();
	}
	catch (const std::exception& e)
	{
		std::cout << "Banco de Dados " << host << "/" << path << " não pode ser
usado!\n Entre com os parâmetros corretos para a conexão!\n";
		std::cout << "Erro: " << e.what() << std::endl;
		ds.reset();
		return ds;
	}
	catch (...)
	{
		std::cout << "Banco de dados " << host << "/" << path << " não pode ser
usado!\n Entre com os parâmetros corretos para a conexão!\n";
		ds.reset();
		return ds;
	}

	std::cout << "Usando banco de dados " << host << "/" << path << std::endl;
	ds->close();
	return ds;
}

void PostGISExample()
{
	try
	{
		std::unique_ptr<te::da::DataSource> ds = GetPostGISConnection();
		if (!ds.get())
			return;

		ds->open();

		std::cout << "Banco de Dados esta aberto? " << std::boolalpha << ds-
>isOpened() << '\n';
```

```
		std::cout << "Banco de Dados e valido? " << std::boolalpha << ds->isValid() << '\n';

		// retrieve the data source capabilities and print it
		std::cout << std::endl;
		PrintDataSourceCapabilities(ds.get());

		// printing some datasets from the datasource
		std::vector<std::string> dsets = ds->getDataSetNames();

		if (dsets.empty())
		{
			std::cout << "Banco de Dados nao tem tabelas.\n";
			return;
		}

		// let's check one of them (or all)
		std::cout << "\nEstes são " << dsets.size() << " os conjuntos de tabelas no banco de dados: \n";
		for (size_t i = 0; i<dsets.size(); ++i)
			std::cout << '\t' << i + 1 << ':' << dsets[i] << std::endl;

		// check point: retrieving the data from a dataset of the datasource
		while (true)
		{
			std::cout << "\nSelecione um banco de dados de 1 até " << ds->getNumberOfDataSets() << " para ver os dados (0 para nenhum): ";
			int n;
			std::cin >> n;
			if (n<1 || n>ds->getNumberOfDataSets())
				break;
			PrintDataSet(dsets[n - 1], ds->getDataSet(dsets[n - 1]).get());
		}

		system("PAUSE");
	}


	catch (const std::exception& e)
	{
		std::cout << std::endl << "An exception has occurred in the PostGIS Example: " << e.what() << std::endl;
	}
	catch (...)
	{
		std::cout << std::endl << "An unexpected exception has occurred in the PostGIS Example!" << std::endl;
	}
}
```

APÊNDICE C – Código de cópia de dados para o PostGIS via Terralib 5

```
// Examples
#include "DataAccessExamples.h"

// TerraLib
#include "../Config.h"
#include <terralib/dataaccess.h>
#include <terralib/geometry.h>

// STL
#include <iostream>

std::unique_ptr<te::da::DataSource> GetPostGISConnection();

void CopyingData()
 {
  try
 {
    // let's take the input dataset from a shape file
    std::string connInfo("file://");
    std::string data_dir = TERRALIB_DATA_DIR;

    std::string aux("");
    std::cout << "Informe a localização do shapefile (ENTER para aceitar o padrao \'" <<
(data_dir + "/shape/segmentos.shp") << "\'): ";
    std::getline (std::cin, aux);
    if (!aux.empty())
      connInfo += aux;
    else
      connInfo += data_dir + "/shape/segmentos.shp";

    std::unique_ptr<te::da::DataSource> dsOrigin =
te::da::DataSourceFactory::make("OGR", connInfo);
    dsOrigin->open();

    if (!dsOrigin->isValid())
    {
      std::cout << "Nao se pode acessar o shapefile.\n";
      return;
    }

    // get a transactor to interact to the data source origin
    std::auto_ptr<te::da::DataSourceTransactor> tOrigin = dsOrigin->getTransactor();

    std::vector<std::string> datasets = tOrigin->getDataSetNames();
    std::auto_ptr<te::da::DataSet> datasetOrigin = tOrigin->getDataSet(datasets[0]);
    std::auto_ptr<te::da::DataSetType> dtOrigin = tOrigin->getDataSetType(datasets[0]);

    std::unique_ptr<te::da::DataSource> dsDestination = GetPostGISConnection();
    if (!dsDestination.get())
      return;
```

```
    dsDestination->open();

    // get a transactor to interact to the data source
    std::auto_ptr<te::da::DataSourceTransactor> tDestination = dsDestination-
>getTransactor();

    // create and save datasettype in the datasource destination
    te::da::DataSetType* newDataSet = static_cast<te::da::DataSetType*>(dtOrigin-
>clone());
    newDataSet->setName("public.segmentos");

          GetFirstGeomProperty(newDataSet)->setSRID(31983);
    GetFirstGeomProperty(newDataSet)->setGeometryType(te::gm::GeometryType);

    std::cout << std::endl << "Inicia a copia..." << std::endl;
    std::map<std::string, std::string> options;

          if (tDestination->dataSetExists("public.segmentos"))
    {
      tDestination->dropDataSet("public.segmentos");

    }

    tDestination->begin();
    tDestination->createDataSet(newDataSet,options);
    tDestination->add(newDataSet->getName(), datasetOrigin.get(),options);
    tDestination->commit();
    std::cout << std::endl << "Copia finilizada..." << std::endl;

  }

  catch(const std::exception& e)
  {
    std::cout << std::endl << "An exception has occurred in the Copy Example: " <<
e.what() << std::endl;
  }
  catch(...)
  {
    std::cout << std::endl << "An unexpected exception has occurred in the Copy
Example!" << std::endl;

  }
}
```

APÊNDICE D – Código TA_Baatz_Segmenter_PostGIS

```
#include "RPExamples.h"

// TerraLib
#include <terralib/raster.h>
#include <terralib/dataaccess.h>
#include <terralib/datatype.h>
#include <terralib/geometry.h>
#include <terralib/memory.h>
#include <terralib/rp.h>

#include <terralib/datatype.h>

// STL
#include <iostream>
#include <stdio.h>
#include <string>
#include <stdlib.h>
#include <cstdio>
#include <cstdio>
#include <memory>
#include <map>
#include <sstream>
#include <fstream>
#include <limits>

#include <libpq-fe.h>

bool getMaskRaster(const std::string& mask_file_name,
        double geoWest, double geoNorth,
        double geoEast, double geoSouth,
        std::unique_ptr< te::rst::Raster >& mask_raster_ptr)
{
        if (mask_file_name.empty()) return false;
        if (!(geoNorth >= geoSouth)) return false;
        if (!(geoEast >= geoWest)) return false;

        std::FILE* file_ptr = std::fopen(mask_file_name.c_str(),
                "rb");
        if (file_ptr == 0) return false;

        char filetype[3];
        unsigned int lines = 0;
        unsigned int cols = 0;
        char dummy_char = 0;

        if (6 != std::fscanf(file_ptr, "%2s%1c%u%1c%u%1c",
                filetype,
                &dummy_char,
                &cols,
                &dummy_char,
                &lines,
```

```
            &dummy_char)) {

            std::fclose(file_ptr);

            std::cout << "Invalid file header";
            return false;
        }

        if (std::string("P4") != filetype) return false;

        /* Allocating the raster */

        std::vector< te::rst::BandProperty * > bandsProps;
        bandsProps.push_back(new te::rst::BandProperty(0, te::dt::UCHAR_TYPE));

        te::gm::Envelope* mbr = new te::gm::Envelope(geoWest, geoSouth, geoEast,
geoNorth);

        te::rst::Grid* grid = new te::rst::Grid(cols, lines, mbr);

        mask_raster_ptr.reset(te::rst::RasterFactory::make("MEM",
            grid, bandsProps,
            std::map< std::string, std::string >(), 0, 0));

        /* Reading data */

        unsigned int col = 0;
        unsigned int linedatasize = (cols / 8) +
            ((cols % 8) ? 1 : 0);
        unsigned char* linedata = new unsigned char[linedatasize];
        unsigned int lineindex = 0;
        te::rst::Raster& mask_raster = (*mask_raster_ptr);
        double value = 0;

        const unsigned char char_1 = 1;
        const unsigned char char_2 = 2;
        const unsigned char char_4 = 4;
        const unsigned char char_8 = 8;
        const unsigned char char_16 = 16;
        const unsigned char char_32 = 32;
        const unsigned char char_64 = 64;
        const unsigned char char_128 = 128;

        for (unsigned line = 0; line < lines; ++line)
        {
            if (1 == std::fread(linedata, linedatasize, 1, file_ptr)) {
                col = 0;

                for (lineindex = 0; lineindex < linedatasize;
                    ++lineindex) {

                    if (col < cols) {
                        value = ((linedata[lineindex] & char_128) ? 0 : 255);
                        mask_raster.setValue(col, line, value, 0);
                        ++col;
```

```
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_64) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_32) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_16) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_8) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_4) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_2) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }

                        if (col < cols) {
                                value = ((linedata[lineindex] & char_1) ? 0 : 255);
                                mask_raster.setValue(col, line, value, 0);
                                ++col;
                        }
                }
        }
        else {
                std::fclose(file_ptr);
                delete[] linedata;

                std::cout << "Error reading data";
                return false;
        }
}

std::fclose(file_ptr);
```

```
        delete[] linedata;

        return true;
}
//-------------

// descricao dos campos
//
// class - class associated to the object.
// llx - lower left x coordinate of the object bounding box.
// lly - lower left y coordinate of the object bounding box.
// urx - upper right x coordinate of the object bounding box.
// ury - upper right y coordinate of the object bounding box.
// size - object size (in pixels).
// xCenter - x coordinate of the object centroid.
// yCenter - y coordinate of the object centroid.
// xGeoCenter - x geo-coordinate of the object centroid.
// yGeoCenter - y geo-coordinate of the object centroid.
// membership or p - confidence in the object with regard to its classification.

void saveToShp(std::vector<te::gm::Geometry*> polygons, std::vector< double >&
polygonsValues,
        const std::string& shpBaseFileName, const te::rst::Grid& rasterGrid)

{


        const double rasterPixelArea = rasterGrid.getResolutionX() *
rasterGrid.getResolutionY();


        std::auto_ptr<te::da::DataSetType> dataSetTypePtr1(new
te::da::DataSetType(shpBaseFileName));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("value", te::dt::INT64_TYPE,
true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("id", te::dt::INT64_TYPE,
true));

        // inicio acréscimo campos

        dataSetTypePtr1->add(new te::dt::SimpleProperty("llx", te::dt::DOUBLE_TYPE,
true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("lly", te::dt::DOUBLE_TYPE,
true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("urx", te::dt::DOUBLE_TYPE,
true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("ury", te::dt::DOUBLE_TYPE,
true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("size", te::dt::DOUBLE_TYPE,
true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("xCenter",
te::dt::DOUBLE_TYPE, true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("yCenter",
te::dt::DOUBLE_TYPE, true));
```

```
        dataSetTypePtr1->add(new te::dt::SimpleProperty("xGeoCenter",
te::dt::DOUBLE_TYPE, true));
        dataSetTypePtr1->add(new te::dt::SimpleProperty("yGeoCenter",
te::dt::DOUBLE_TYPE, true));

        // fim acréscimo campos

        dataSetTypePtr1->add(new te::gm::GeometryProperty("polygon", polygons[0]-
>getSRID(),
                te::gm::PolygonType, true));

        std::auto_ptr<te::da::DataSetType> dataSetTypePtr2(new
te::da::DataSetType(*dataSetTypePtr1));

        std::auto_ptr< te::mem::DataSet > memDataSetPtr(new
te::mem::DataSet(dataSetTypePtr1.get()));

        double centroidCol = 0;
        double centroidRow = 0;

        for (unsigned int polygonsIdx = 0; polygonsIdx < polygons.size(); ++polygonsIdx)
        {
                te::mem::DataSetItem* dsItemPtr = new
te::mem::DataSetItem(memDataSetPtr.get());

                rasterGrid.geoToGrid(polygons[polygonsIdx]->getCentroid().x,
                        polygons[polygonsIdx]->getCentroid().y, centroidCol,
centroidRow);

                dsItemPtr->setInt64(0, polygonsValues[polygonsIdx]);
                dsItemPtr->setInt64(1, polygonsIdx);

                //Novos campos

                // llx
                dsItemPtr->setDouble(2, polygons[polygonsIdx]->getMBR()-
>getLowerLeftX());

                // lly
                dsItemPtr->setDouble(3, polygons[polygonsIdx]->getMBR()-
>getLowerLeftY());

                // urx
                dsItemPtr->setDouble(4, polygons[polygonsIdx]->getMBR()-
>getUpperRightX());

                // ury
                dsItemPtr->setDouble(5, polygons[polygonsIdx]->getMBR()-
>getUpperRightY());

                // size in pixels
                dsItemPtr->setDouble(6, ((te::gm::Polygon*)polygons[polygonsIdx])-
>getArea() / rasterPixelArea);

                // xCenter
```

```
                dsItemPtr->setDouble(7, centroidCol);

                // yCenter
                dsItemPtr->setDouble(8, centroidRow);

                // xGeoCenter
                dsItemPtr->setDouble(9, polygons[polygonsIdx]->getCentroid().x);

                // yGeoCenter
                dsItemPtr->setDouble(10, polygons[polygonsIdx]->getCentroid().y);

                //Fim Novos campos

                dsItemPtr->setGeometry(11, (te::gm::Geometry*)polygons[polygonsIdx]-
>clone());

                memDataSetPtr->add(dsItemPtr);
        }

        remove((shpBaseFileName + ".shx").c_str());
        remove((shpBaseFileName + ".shp").c_str());
        remove((shpBaseFileName + ".prj").c_str());
        remove((shpBaseFileName + ".dbf").c_str());


        std::string connInfo("file://" + shpBaseFileName + ".shp");

        std::unique_ptr<te::da::DataSource>
dsOGR(te::da::DataSourceFactory::make("OGR", connInfo));
        dsOGR->open();

        memDataSetPtr->moveBeforeFirst();

        te::da::Create(dsOGR.get(), dataSetTypePtr2.get(), memDataSetPtr.get());


        dsOGR->close();

        //return;
}
//--------inicio label image

void saveLabeledImageFile(const std::string& output_image_file_name_str,
        const te::rst::Raster& raster, unsigned int raster_channel)
{

//      std::cout << "caminho " << output_image_file_name_str << std::endl;
//      std::cout << "numero de bandas " << raster_channel << std::endl;
//      std::cout << "raster bandas " << raster.getNumberOfBands() << std::endl;
//      std::cout << "nro linhas " << raster.getNumberOfRows() << std::endl;
//      std::cout << "nro colunas " << raster.getNumberOfColumns() << std::endl;

        if (!(raster.getNumberOfBands() > (int)raster_channel)) throw;

        unsigned int nlines = (unsigned int)raster.getNumberOfRows();
```

```
	unsigned int ncols = (unsigned int)raster.getNumberOfColumns();
	unsigned int curr_line = 0;
	unsigned int curr_col = 0;
	double curr_value_double = 0;
	int curr_value_int = 0;
	size_t sizeofint = sizeof(int);

	/* Finding min and max raster values */

	double max_value_dbl = std::numeric_limits< double >::max() * (-1.0);
	double min_value_dbl = std::numeric_limits< double >::max();

	for (curr_line = 0; curr_line < nlines; ++curr_line)
	{
		for (curr_col = 0; curr_col < ncols; ++curr_col)
		{
			raster.getValue(curr_col, curr_line, curr_value_double,
				raster_channel);

			if (curr_value_double > max_value_dbl) {
				max_value_dbl = curr_value_double;
			}
			if (curr_value_double < min_value_dbl) {
				min_value_dbl = curr_value_double;
			}
		}
	}

	unsigned int max_value_uint = (unsigned int)max_value_dbl;
	unsigned int min_value_uint = (unsigned int)min_value_dbl;

	/* Creating file */

	std::FILE* file_ptr = std::fopen(output_image_file_name_str.c_str(),
		"wb");

	if (file_ptr == 0)
	{
		std::cout << "Cannot create file";
		system("PAUSE");
		return;
	}

	/* Writing header */

	std::fprintf(file_ptr, "F5\n");

#if BYTE_ORDER == LITTLE_ENDIAN
	std::fprintf(file_ptr, "L\n");
#elif BYTE_ORDER == BIG_ENDIAN
	std::fprintf(file_ptr, "B\n");
#else
#error "ERROR: Endianess format detection error"
#endif
```

```
	std::fprintf(file_ptr, "%u %u\n", ncols, nlines);

	std::fprintf(file_ptr, "%u %u\n", min_value_uint, max_value_uint);

	/* Writing image data */

	for (curr_line = 0; curr_line < nlines; ++curr_line) {
		for (curr_col = 0; curr_col < ncols; ++curr_col) {
			raster.getValue(curr_col, curr_line, curr_value_double,
raster_channel);
			curr_value_int = (int)curr_value_double;

			if (1 != std::fwrite(&curr_value_int, sizeofint, 1, file_ptr)) {
				std::cout << "Error wrinting file";
				std::fclose(file_ptr);
				return;
			}
		}
	}

	std::fclose(file_ptr);
}

//--------Fim label image

//void Segmenter(){}
void Segmenter(char* dbname, char* username, char* password, char* hostname, char*
port, char* tabela, char* scale, char* similarid, char* color, char* compact, char*
nrobandas, char* pesobandas)
{
	// ------------------------------------------------------------------------------------------------------
----
	// Region growing segmenter (with Baatz features) example
	try
	{
		// Hora
		time_t rawtime;
		struct tm * timeinfo;
		time(&rawtime);
		timeinfo = localtime(&rawtime);
		printf("Data inical atual do sistema: %s", asctime(timeinfo));

		//std::cout << "Segmentacao Baatz - exemplo usando Raster no PostGIS"
<< std::endl << std::endl;

		// open input raster

		// Inicia abertura de imagem no banco e saída em disco

		std::map<std::string, std::string> inputRasterInfo;

		//std::cout << std::endl << "Ponto 1 - Abertura de imagem no banco" <<
std::endl;

		std::stringstream ss;
```

```
		ss << "PG:host=" << hostname << " port=" << port << " dbname='" <<
dbname << "' user='" << username << "' password='" << password << "' table=" <<
tabela << " schema='public' mode=2";

		std::string s = ss.str();

		inputRasterInfo["URI"] = s;

		//std::cout << s;

		boost::shared_ptr< te::rst::Raster >
inputRasterPointer(te::rst::RasterFactory::open(
			inputRasterInfo));
		if (inputRasterPointer.get() == 0) return;

		/* Access a raster datasource to create the output raster */

		//std::cout << std::endl << "Ponto 2 - Define Saida em disco" << std::endl;

		std::map<std::string, std::string> outputRasterInfo;

		std::map<std::string, std::string> outputRasterInfoLabel;


		outputRasterInfo["URI"] = TERRALIB_DATA_DIR
"/geotiff/result_img_seg.tif";

		outputRasterInfoLabel["URI"] = TERRALIB_DATA_DIR
"/plm/result_img_seg_plm.plm";

		// Finaliza abertura de imagem no banco e saída em disco
		// define segmentation parameters
		// input parameters
		te::rp::Segmenter::InputParameters algoInputParameters;
		//abre parametro no disco
		//			algoInputParameters.m_inputRasterPtr = rin;
		//abre parametro no banco
		algoInputParameters.m_inputRasterPtr = inputRasterPointer.get();
		/*Tratamento do número de bandas*/
		std::string input = nrobandas;
		std::istringstream nrob(input);
		std::string token;
		int qtd = 0;
		while (std::getline(nrob, token, ',')) {
			//std::cout << token << '\n';
			int vl = atoi(token.c_str());
			//std::cout << vl << '\n';
			algoInputParameters.m_inputRasterBands.push_back(vl);
			qtd++;
		}
		// link specific parameters with chosen implementation
		// strategy specific parameters (m_minSegmentSize: size of the smallest
segment to be created;
```

```
            // m_segmentsSimilarityThreshold: similarity between neighboring segments to
merge them or not)
                te::rp::SegmenterRegionGrowingBaatzStrategy::Parameters
segparameters;
                te::rp::SegmenterRegionGrowingBaatzStrategy::Parameters
m_minSegmentSize;

                //-----------------------------------------
                //      segparameters.m_minSegmentSize = 80; //Escala
                std::stringstream st1;
                st1 << scale;
                int v1;
                st1 >> v1;
                segparameters.m_minSegmentSize = v1;
                //-----------------------------------------
                //              segparameters.m_segmentsSimilarityThreshold = 0.03;
//Similaridade
                std::stringstream st2;
                st2 << similarid;
                double v2;
                st2 >> v2;
                segparameters.m_segmentsSimilarityThreshold = v2;
                //-----------------------------------------
                // Peso das bandas -> iguais
                //              segparameters.m_bandsWeights.clear();
                //segparameters.m_bandsWeights.resize(8, 0.125);
                /*Tratamento do peso de bandas*/
                std::string inputp = pesobandas;
                std::istringstream pesb(inputp);
                std::string tokenp;
                //std::cout << inputp;
                //float total = 0;
                while (std::getline(pesb, tokenp, ',')) {

                        float vl = ::atof(tokenp.c_str());
                        //total = total + vl;
                        //std::cout << vl <<  '\n';
                        segparameters.m_bandsWeights.push_back(vl);
                }
                //std::cout << total;
                //exit;
                //-----------------------------------------
                //              segparameters.m_colorWeight = 0.7; // Cor
                std::stringstream st3;
                st3 << color;
                double v3;
                st3 >> v3;
                segparameters.m_colorWeight = v3;
                //-----------------------------------------
                segparameters.m_compactnessWeight = 0.7; // Compacidade
                std::stringstream st4;
                st4 << compact;
                double v4;
                st4 >> v4;
                segparameters.m_compactnessWeight = v4;
```

```
                //-----------------------------------------
                algoInputParameters.m_strategyName = "RegionGrowingBaatz";
                algoInputParameters.setSegStrategyParams(segparameters);
                //Processar a imagem em blocos (Emiliano) (Segmenter.h)
                algoInputParameters.m_enableThreadedProcessing = true;
                algoInputParameters.m_maxSegThreads = 0;
                // blocagem
                // algoInputParameters.m_enableBlockProcessing = false;
                algoInputParameters.m_enableBlockProcessing = true;

                algoInputParameters.m_blocksOverlapPercent = 10;
                algoInputParameters.m_maxBlockSize = 0;
                // output parameters
                // the output can be a previously created raster (in this
case,outputRasterInfo)
                te::rp::Segmenter::OutputParameters algoOutputParameters;
                //algoOutputParameters.m_rInfo - As informações necessárias para criar
o raster
                algoOutputParameters.m_rInfo = outputRasterInfo;
                //      algoOutputParameters.m_rType = "GDAL";
                // algoOutputParameters.m_rType - Tipo de fonte de dados de raster de
saída
                algoOutputParameters.m_rType = "EXPANSIBLE";
                // execute the algorithm
                te::rp::Segmenter seginstance;
                if (!seginstance.initialize(algoInputParameters)) throw;
                //std::cout << std::endl << "Ponto 3 - Segmentacao" << std::endl;
                if (!seginstance.execute(algoOutputParameters)) throw;
                //std::cout << std::endl << "Ponto 4 - Copiar raster para gerar vetor" <<
std::endl;
                //*algoOutputParameters.m_outputRasterPtr - Um ponteiro do raster de
saída gerado(imagem do rótulo)
                if (!te::rp::Copy2DiskRaster(*algoOutputParameters.m_outputRasterPtr,
outputRasterInfo["URI"])) throw;
                //std::cout << std::endl << "Ponto 5 - Gerar Shapefile" << std::endl;
                // Gera vetor
                std::vector<te::gm::Geometry*> polygons;
                std::vector< double > polygonsValues;
                algoOutputParameters.m_outputRasterPtr->vectorize(polygons, 0, 0,
                &polygonsValues);
                //std::cout << std::endl << "Ponto 6 - salvar shapefile" << std::endl;
                saveToShp(polygons, polygonsValues, "../../../_exec_seg/seg_img",
                        *algoOutputParameters.m_outputRasterPtr->getGrid());
                saveLabeledImageFile(outputRasterInfoLabel["URI"],
*algoOutputParameters.m_outputRasterPtr, 0);
                // Fim Gera vetor
                // clean up
                // Hora fim
                time(&rawtime);
                timeinfo = localtime(&rawtime);
                printf("Data final atual do sistema: %s", asctime(timeinfo));
                std::cout << "Pronto!" << std::endl << std::endl;
        }
        catch (const std::exception& e)
        {
```

```
                std::cout << std::endl << "Ocorreu uma exceção em Segmenter(): " <<
e.what() << std::endl;
        }
        catch (...)
        {
                std::cout << std::endl << "Ocorreu uma exceção inesperada em
Segmenter()!" << std::endl;
        }
        return;

}
```

APÊNDICE E – Código da função de separação dos segmentos gerados no PostGIS e exportação para disco

```
CREATE OR REPLACE FUNCTION public.fn_tile2shp(
    p_tile_id integer,
    p_tx_tabela_raster text,
    p_tx_tabela_vetor text)
  RETURNS boolean AS
$BODY$
DECLARE
    v_srid_raster integer;
    c_cursor_segmentos record;
    v_command text;

BEGIN
execute 'create schema if not exists '||p_tx_tabela_raster;



execute 'create table if not exists '||p_tx_tabela_raster||'.export_commands (
                            command text
        )';
v_command:= 'pgsql2shp -f C:/_saida_shapes/'||p_tx_tabela_raster||'_'||p_tile_id||' -h
localhost -u postgres -P postgresql postgis_232
'||p_tx_tabela_raster||'.'||p_tx_tabela_raster||'_'||p_tile_id;

execute 'insert into '||p_tx_tabela_raster||'.export_commands (command)
    values ($1)' using v_command;

execute 'select st_srid(rast) from '||p_tx_tabela_raster||' where rid = $1' into v_srid_raster
                    using p_tile_id;
          --raise exception '%', v_srid_raster;

execute 'drop table if exists '||p_tx_tabela_raster||'.'||p_tx_tabela_raster||'_'||p_tile_id;


execute 'create table '||p_tx_tabela_raster||'.'||p_tx_tabela_raster||'_'||p_tile_id||' as
    select gid, st_transform((st_setsrid(geom,$1)) ,$1) as geom from
'||p_tx_tabela_vetor||' a
    join '||p_tx_tabela_raster||' b on st_setsrid(geom,$1) && ST_Envelope(rast) and
st_within(st_setsrid(geom,$1),ST_Envelope(rast))
    where rid = $2
    --and gid not in (select gid from '||p_tx_tabela_raster||'.'||p_tx_tabela_raster||'_control)

    union all

    select a.gid,
st_transform(ST_CollectionExtract(st_intersection(st_envelope(rast),st_transform((st_set
srid(geom,$1)) ,$1)),3),$1) as geom from '||p_tx_tabela_vetor||' a
    join '||p_tx_tabela_raster||' b on  st_overlaps(st_setsrid(geom,$1),ST_Envelope(rast))
    where rid = $2
    '
```

```
        using v_srid_raster, p_tile_id;

    return true;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

APÊNDICE F – Código do operador TA_Baatz_Segmenter_PostGIS no InterImage

```
<operator type=topdown class=gis name="TA_Baatz_Segmenter_Postgis"
cmd="C:/Terralib_5.2.2/build/Debug/terralib_example_rp \"@dbname@\"
\"@username@\" \"@password@\" \"@hostname@\" \"@port@\" \"@tabela@\"
\"@scale@\" \"@similarity@\" \"@color@\" \"@compact@\" \"@nroband@\"
\"@pesoband@\" " runglobal=false >

  <attribute name=dbname label="a) Banco" value="postgis_232">

  <attribute name=username label="b) Usuario" type=string value="postgres">

  <attribute name=password label="c) Senha" type=string value="postgresql">

  <attribute name=hostname label="d) hostname" type=string value="localhost">

  <attribute name=port label="e) Porta" type=integer value=5432>

  <attribute name=tabela label="f) Tabela" type=string value=>

  <attribute name=scale label="g) Escala" type=integer value=100>

  <attribute name=similarity label="h) Similaridade" type=double value=0.03>

  <attribute name=color label="i) Cor" type=double value=0.4>

  <attribute name=compact label="j) Compacidade" type=double value=0.7>

  <attribute name=nroband label="k) Nro. Bandas" type=double value=>

  <attribute name=pesoband label="l) Pesos Bandas" type=double value=>

</operator>
```