UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
ESCOLA DE INFORMÁTICA APLICADA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Aplicação Prática para Visualização de Dados Espaciais e Assinaturas 4CRS na Web

Autora:
Cinthia Conti Vanni

Orientador:
Leonardo Guerreiro Azevedo

Aplicação Prática para Visualização de Dados Espaciais e Assinaturas 4CRS na Web

Cinthia Conti Vanni

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Rio de Janeiro - RJ
Janeiro/2012

Aplicação Prática para Visualização de Dados Espaciais e Assinaturas 4CRS na Web

Aprovado em ____/_____/_____

BANCA EXAMINADORA

_____

Prof. Leonardo Guerreiro Azevedo, D.Sc. (UNIRIO)

_____

Prof. Asterio Kiyoshi Tanaka, Ph.D. (UNIRIO)

_____

Prof. Márcio de Oliveira Barros, D.Sc. (UNIRIO)

A autora deste Projeto autoriza a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, ____ de _____ de _____

_____

Cinthia Conti Vanni

Agradecimentos

À minha avó Maria José, que sempre pensou em mim mais que eu mesma, e que, ao me alfabetizar, deu o pontapé inicial em minha vida intelectual.

Aos meus pais, Valéria e Umberto, que muitas vezes se sacrificaram para me proporcionar a melhor educação.

Aos meus irmãos, Renato e Henrique, que hoje são para mim os maiores exemplos de força e perseverança.

À Ana Rachel Fonseca, aquela que me consertou, que me fez querer ter objetivos e me ajudou a traçá-los.

À minha chefe Ana Porthum, pela compreensão e apoio de todas as horas.

Ao meu professor e orientador Leonardo Azevedo, que me incentivou a ser uma melhor aluna e aprendiz, e que reconheceu em mim um potencial que eu mesma desconhecia.

À todos os professores e funcionários da UNIRIO pela paciência e atenção concedidas durante a graduação.

Finalmente, aos meus amigos, que contribuíram para a minha história e que são para toda vida, mesmo que não estejam comigo a vida inteira.

> Significativos em cada etapa de formação. Não estão em nossa frente diariamente, mas estão em nossa personalidade, determinando, de modo imperceptível, as nossas atitudes.
>
> (Fabrício Carpinejar).

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

4CRS - Four-Color Raster Signature

AJAX - Asynchronous JavaScript and XML

API - Application Programming Interface

CSS - Cascade Style Sheets

GIS - Geographic Information System

HTML - HyperText Markup Language

INDE - Infraestrutura Nacional de Dados Espaciais

INPE - Instituto Nacional de Pesquisas Espaciais

JSON - JavaScript Object Notation

KML - Keyhole Markup Language

MBR - Minimum Bounding Rectangle

OGC - Open Geospatial Consortium

PHP - Hypertext Preprocessor

SGBD - Sistema Gerenciador de Banco de Dados

SGBDE - Sistema Gerenciador de Banco de Dados Espaciais

SIG - Sistema de Informação Geográfica

SRID - Spatial Reference System Identifier

XML - Extensible Markup Language

WKB - Well-Known Binary

WKT - Well-Known Text

RESUMO

Este trabalho apresenta a proposta de um passo-a-passo para desenvolver um sistema de visualização de dados espaciais na web. Este passo-a-passo foi implementado com soluções de código fonte aberto, utilizando as linguagens HTML5, PHP e JavaScript, e o banco de dados PostgreSQL juntamente com sua extensão espacial PostGIS, específica para o armazenamento de dados espaciais. Para visualização desses dados, utilizamos o elemento Canvas do HTML5. Para demonstração da proposta foram implementadas visualizações de dados espaciais correspondentes a polígonos do Brasil e assinaturas 4CRS de polígonos, além de ferramentas auxiliares para interação com os dados espaciais, como seleção de polígonos, zoom por ponto e zoom por seleção.

Palavras-chave: Bancos de dados espaciais, Visualização de dados espaciais, PostGIS, Canvas, HTML5, 4CRS.

Capítulo 1: INTRODUÇÃO

O objetivo deste capítulo é contextualizar o assunto abordado neste trabalho, apresentando sua motivação, objetivo e a estrutura dos capítulos.

1.1 Motivação

> "Compreender a distribuição espacial de dados oriundos de fenômenos ocorridos no espaço é um grande desafio para esclarecer questões centrais em diversas áreas do conhecimento, como em saúde, em ambiente, em geologia, em agronomia, entre tantas outras." [Câmara *et al.*, 2002].

> "A Cartografia é a ciência da representação e do estudo da distribuição espacial dos fenômenos naturais e sociais, suas relações e suas transformações ao longo do tempo, por meio de representações cartográficas – modelos icônicos – que reproduzem este ou aquele aspecto da realidade de forma gráfica e generalizada." (Martinelli, 1991:35).

De acordo com as citações acima, concluímos que a análise de dados espaciais é fundamental para um melhor entendimento dos mesmos. Tal análise pode ser realizada por meio de mapas, globos, fotografias e imagens, e pode ser aplicada em diversas áreas do conhecimento.

Segundo Martinelli [1991] *apud* Francischett [2001], a Cartografia é a ciência que representa e investiga conteúdos espaciais e não poderá fazê-los sem o conhecimento da essência dos fenômenos que estão sendo representados, nem sem o suporte das ciências que os estudam.

Existem inúmeros estudos de fenômenos representados por dados espaciais, tornando a visualização destes fundamental para uma total compreensão do que esse conjunto de dados significa.

No Brasil há alguns Ministérios que produzem, inclusive, no contexto de suas atividades e atribuições, dados geoespaciais. Casos típicos são o do IBGE e o das Forças Armadas Brasileiras [Ribeiro, 1996].

O Instituto Nacional de Pesquisas Espaciais (INPE) possui como missão "Produzir ciência e tecnologia nas áreas espacial e do ambiente terrestre e oferecer produtos e serviços singulares em benefício do Brasil" [INPE, 2012]. Dentre os produtos e serviços na área de visualização de dados espaciais oferecidos pelo INPE, se destacam os softwares livres Spring Web[1] e TerraView[2]. Ambos tem como objetivo apresentar um visualizador de dados geográficos com recursos de consulta e análise destes dados. Além dos softwares livres citados, o INPE também fornece uma biblioteca de funções e classes em C++, que auxilia no desenvolvimento de sistemas SIG [INPE, 2011].

Iniciativa do governo federal, a Infraestrutura Nacional de Dados Espaciais (INDE) tem como objetivo de integrar todos os dados geospaciais existentes nas instituições do governo brasileiro, harmonizando-os, disseminando-os e proporcionando o seu uso efetivo. A adoção de padrões e normas para as informações geoespaciais é fundamental para a efetiva integração [INDE, 2011].

Exemplos mais cotidianos são os SIG da Google, que incluem o Google Maps e Google Earth, e oferecem a visualização de dados espaciais em sua forma vetorial, em imagens de satélite e em 3D. A Google também oferece, através de uma API, a possibilidade de conversão de dados em um arquivo KML para integração com o Google Maps e Google Earth [Google Code, 2011]. A linguagem KML, utilizada para armazenar dados geográficos e recentemente incluída, a pedido da Google, no Padrão OGC, é focada na visualização geográfica e possui notações para mapas e imagens [KML, 2011]. O arquivo KML gerado é incluído no Google Maps por meio de uma conta de usuário e seus dados podem ser visualizados pelo usuário tanto no Google Maps quanto no Google Earth, absorvendo todas as funcionalidades que os mesmos oferecem.

Os exemplos citados acima expressam a importância cada vez mais valorizada da visualização de dados espaciais para estudos e análises tanto no meio acadêmico quanto profissional e justificam o investimento em novas e promissoras tecnologias que proporcionem essa visualização.

---

[1] http://www.dpi.inpe.br/spring/portugues/sprweb/springweb.html
[2] http://www.dpi.inpe.br/terraview/index.php

1.2 Objetivo

Este trabalho propõe um passo-a-passo para a visualização de dados espaciais na Web, acessando um banco de dados espaciais. Entende-se por dados espaciais, pontos, polígonos, linhas, regiões, retângulos, superfícies e volumes [Samet, 1990].

A proposta foi implementada utilizando tecnologias de código aberto, demonstrando a viabilidade de implementação em cenários onde não se tem recursos para aquisição de tecnologias fechadas.

Para demonstrar que é possível realizar a visualização de outros tipos de dados baseando-se na proposta apresentada para visualização de dados espaciais, também foi implementada a visualização de Assinaturas Raster de 4 Cores (4CRS) de polígonos.

1.3 Estrutura do trabalho

Este trabalho está dividido da seguinte forma.

O capítulo 1 corresponde a presente introdução, apresenta a motivação e contextualiza o assunto deste trabalho.

O capítulo 2 apresenta os principais conceitos do trabalho, tais como: Dados Espaciais, Bancos de Dados Espaciais e 4CRS.

O capítulo 3 apresenta as tecnologias utilizadas para desenvolver a aplicação.

O capítulo 4 apresenta a implementação do passo-a-passo para desenvolver uma aplicação de visualização de dados espaciais na web.

O capítulo 5 apresenta a aplicação da proposta para dados espaciais e representações de 4CRS dos mesmos.

O capítulo 6 apresenta a conclusão do trabalho.

Os códigos-fonte da implementação desenvolvida neste projeto encontram-se em um repositório online e podem se acessados pelo link http://code.google.com/p/cinthia-conti-projeto-graduacao/.

Capítulo 2: PRINCIPAIS CONCEITOS

Este capítulo tem como objetivo apresentar os conceitos básicos necessários para compreender os assuntos tratados no decorrer deste trabalho, como o significado de Dados Espaciais, Bancos de Dados Espaciais e Assinatura Raster de Quatro Cores (4CRS).

2.1 Dados Espaciais

Dados espaciais consistem de objetos espaciais representados por pontos, linhas, regiões (ou polígonos), janelas (ou retângulos), superfícies, e mesmo dados em dimensões maiores, que podem incluir também a dimensão tempo [Samet, 1990]. Exemplos de dados espaciais são: cidades, rios, rodovias, regiões administrativas, municípios, estados, áreas de utilização do solo, cadeias de montanhas etc. Atributos espaciais frequentemente aparecem juntos com atributos que representam informações não espaciais. Exemplos de dados não espaciais são: nomes de rodovias, endereços, números de telefones, nomes de cidades etc. Como dados espaciais e não espaciais estão intimamente relacionados, não é surpresa que muitas das questões que precisam ser estudadas são na realidade questões de bancos de dados.

2.2 Banco de Dados Espaciais

Segundo [Güting, 1994], Sistemas Gerenciadores de Banco de Dados Espaciais (SGBDE ou Spatial Database Management Systems - SDBMS) provêem a tecnologia de banco de dados fundamental para Sistemas de Informações Geográficas (SIG ou Geographic Information Systems - GIS) e outras aplicações. Um SGBDE possui as seguintes características:

- É um sistema de banco de dados;
- Oferece tipos de dados espaciais (Spatial Datatypes – SDT’s) no seu modelo de dados e na sua linguagem de consulta;

- Suporta tipos de dados espaciais na sua implementação provendo, pelo menos, indexação espacial e algoritmos eficientes para junções espaciais.

A Open Geospatial Consortium (OGC), também conhecida como OpenGIS Consortium, é uma organização sem fins lucrativos dedicada à sistemas *opensource* de geoprocessamento. A missão da OGC é liderar globalmente o desenvolvimento, promoção e harmozinação de padrões e arquiteturas *opensource,* permitindo a integração de dados e serviços geoespaciais com aplicações para o usuário [Reed, 2005]. O processo de desenvolvimento de padrões da OGC criou dois tipos de produtos padrão: Especificações Abstratas e Especificações de Implementação. O propósito da Especificação Abstrata é criar e documentar um modelo conceitual que seja o suficiente para criar a Especificação de Implementação [Reed, 2005].

Uma especificação de elementos simples para SQL foi definida na Especificação Abstrada com o objetivo de definir um esquema padrão em SQL que suporte o armazenamento, recuperação, consulta e atualização de coleções simples de elementos geográficos [OpenGIS Consortium, 1999]. Um elemento simples foi definido para ser tanto um atributo espacial, quanto não-espacial. Coleções simples de elementos geográficos são conceitualmente armazenadas em SGBDs em tabelas com colunas do tipo geometry [OpenGIS Consortium, 1999].

Os modelo de objetos para a classe Geometry e suas subclasses são apresentados na Figura 1. Cada objeto geométrico é associado a um Sistema Espacial Referenciado que descreve a coordenada espacial na qual o objeto geométrico é definido.

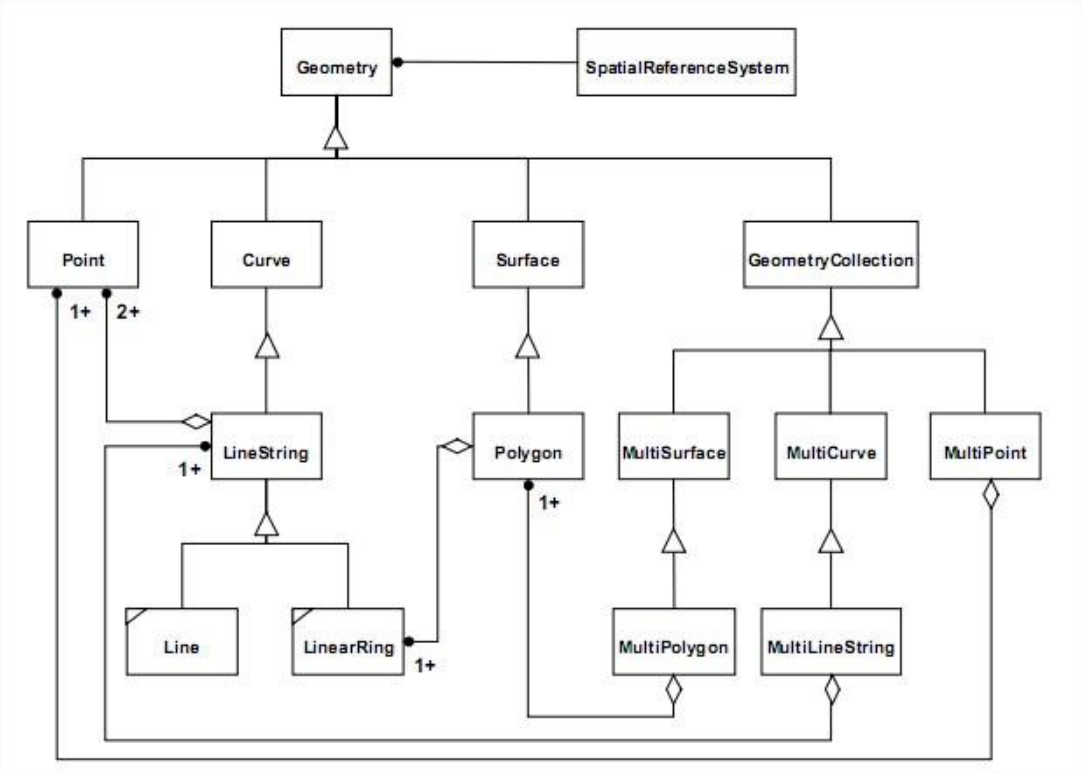

Figura 1 – Hieraquia da Classe Geometry

Existem numerosas aplicações na área de bancos de dados espaciais: supervisão de tráfego, controle aéreo, previsão do tempo, planejamento urbano, otimização de rota, cartografia, agricultura, administração de recursos naturais, monitoramento costeiro, fogo e controle de epidemias [Aronoff, 1989; Tao *et al.*, 2003].

2.3 Assinatura Raster de Quatro Cores (4CRS)

Segundo Souza [2001], filtros geométricos consistem na utilização, durante o processamento de consultas espaciais, de representações aproximadas e compactas dos objetos que armazenam suas principais características. Filtros geométricos são utilizados para obter respostas conclusivas de forma mais rápida, a partir do teste das aproximações mais simples ao invés das representações reais dos objetos que, por possuirem uma maior complexidade, geram um elevado custo de processamento em suas consultas.

Segundo BRINKHOFF et al. (1993b) existem três tipos de aproximações: conservadoras, progressistas e generalizadoras. Aproximações

conservadores contêm todo o objeto representado, aproximações progressistas estão inteiramente contidas no objeto e aproximações generalizadoras tentam simplificar o contorno dos objetos.

A Assinatura Raster de Quatro Cores (Four-Colour Raster Signature - 4CRS) foi proposta por Zimbrão e Souza [1998] e é classificada como uma aproximação generalizadora. A 4CRS armazena as principais características dos dados em uma representação aproximada e compacta que pode ser acessada e processada mais rapidamente do que os dados reais. A assinatura corresponde a uma grade de células onde cada célula armazena informação relevante do objeto utilizando poucos bits. A Tabela 1 apresenta os quatro tipos de célula da assinatura 4CRS. Cada tipo corresponde a um percentual do polígono presente na célula. Por exemplo, uma célula do tipo "vazio" corresponde a uma célula que não tem nenhuma interseção com o polígono, enquanto uma célula "cheia" corresponde a uma célula totalmente ocupada pelo polígono. Células do tipo "pouco" correspondem a um percentual de 0% (exclusive) a 50% (inclusive) de interseção do polígono com a célula, enquanto que célula do tipo "muito" corresponde a um percentual de 50% (exclusive) a 100% (exclusive) de interseção. Um exemplo de assinatura 4CRS é apresentado na Figura 2.

Tabela 1 - Tipos de célula 4CRS

Valor	Tipo	Descrição
00	Vazio	A célula não tem interseção com o polígono.
01	Pouco	A célula tem uma interseção de 50% ou menos com o polígono.
10	Muito	A célula tem uma interseção de mais de 50% e menos de 100% com o polígono.
11	Cheio	A célula é totalmente ocupada pelo polígono.

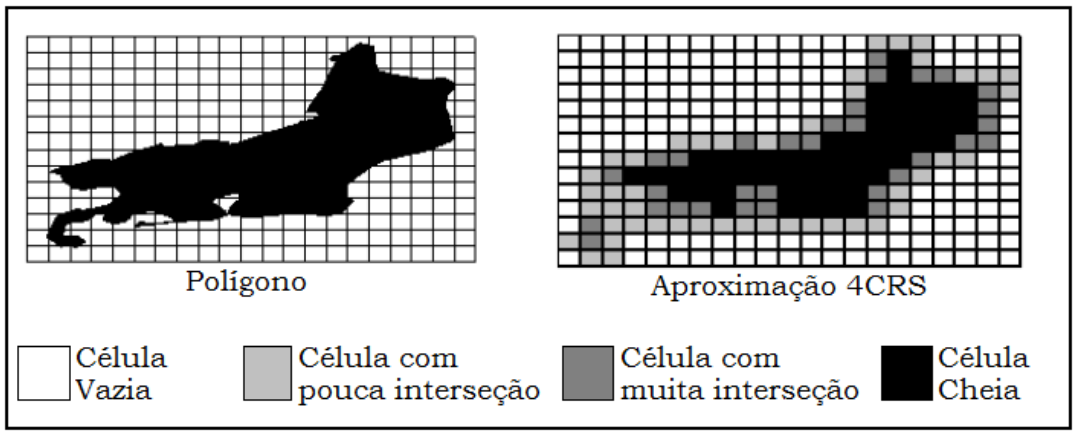

Figura 2 - Exemplo de uma assinatura 4CRS [Azevedo *et al.*, 2004]

A assinatura 4CRS foi concebida principalmente para comparar se polígonos têm interseção de forma mais rápida do que realizar esta comparação considerando os objetos reais [Zimbrão e Souza, 1998]. Apenas podem ser comparadas células de mesmo tamanho e que se sobrepõem perfeitamente, ou seja, assinaturas que tenham a mesma resolução. Para estar de acordo com estes requisitos, a geração de grades deve seguir um padrão pré-definido, como ilustrado na Figura 3. Na Figura 3.a, as assinaturas foram construídas sem seguir um padrão pré-estabelecido. Logo, elas não se sobrepõem. Na Figura 3.b, as assinaturas dos dois objetos foram criadas considerando um padrão. Em outras palavras, existe uma grade universal, e o sistema de coordenadas define a grade.

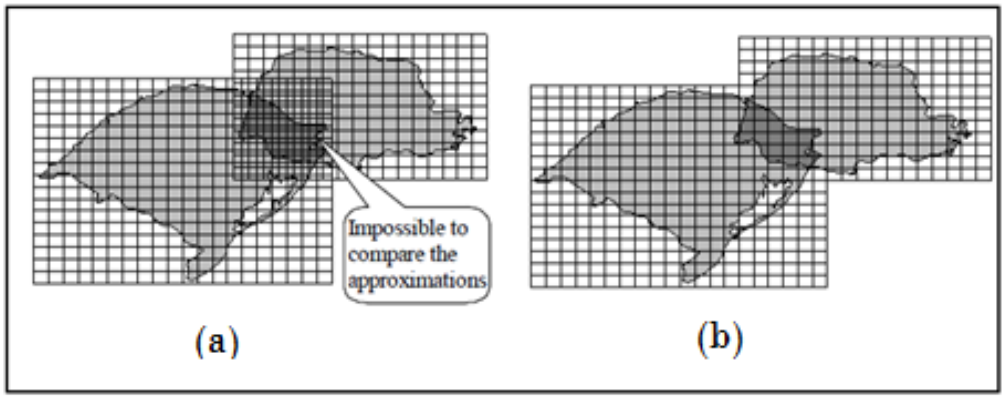

Figura 3 - (a) assinaturas cujas células não se sobrepõem; (b) assinatura cujas células se sobrepõem perfeitamente.

Os requisitos para construção de grades de células podem ser atendidos se for restringido que o tamanho do lado de cada célula seja um valor em potência de dois (2^n) e que os vértices de cada célula sejam múltiplos da

mesma potência de dois ($a \times 2^n$) no sistema de coordenadas. Dessa forma, garantimos que, se duas células de mesmo tamanho se sobrepõem, então elas estão perfeitamente sobrepostas uma na outra, como apresentado na Figura 3.b.

O espaço é dividido em $p \times q$ células com tamanho 2^n, o qual corresponde ao MBR-2^n (*Minimum Bounding Rectangle* ou Retângulo Mínimo Envolvente) do objeto com coordenadas em potência de dois. Em outras palavras, o MBR-2^n corresponde ao MBR cujas coordenadas são múltiplas de potência de dois e que define um espaço de $p \times q$ células com tamanho 2^n. Os vértices do MBR-*2n* são ($2^n a_0$, $2^n b_0$) e ($2^n a_p$, $2^n b_q$), onde a_0, a_p, b_0, b_q e n são números inteiros. Além disso, n é escolhido tal que $(a_p - a_0) \times (b_q - b_0) \leq N$, onde $(a_p - a_0)$ é o número de células no eixo x, $(b_q - b_0)$ é o número de células no eixo y e N é o número máximo de células da grade. Dessa forma, há um limite para o tamanho da assinatura, o qual é definido pelo parâmetro N.

O MBR-2^n é computado baseado no MBR do objeto, truncando suas coordenadas para potências de 2. Neste trabalho, está sendo considerado que as assinatura 4CRS foram geradas previamente. Logo, sabe-se o MBR do objeto e a potência de 2 utilizada para sua geração. Dessa forma, para calcular o MBR-2^n utiliza-se as equações i, ii, iii e iv apresentadas a seguir e a implementação das mesmas na linguagem PHP é apresentada na **Error! Reference source not found.**. Considere que *int* é uma função que retorna a parte inteira resultante da divisão.

$$a_0 = int(\frac{MBR.x.min}{2^n}) \times 2^n \quad \text{(i)}$$

$$a_p = (int(\frac{MBR.x.max}{2^n}) + 1) \times 2^n \quad \text{(ii)}$$

$$b_0 = int(\frac{MBR.y.min}{2^n}) \times 2^n \quad \text{(iii)}$$

$$b_q = (int(\frac{MBR.y.max}{2^n}) + 1) \times 2^n \quad \text{(iv)}$$

A escala da grade pode ser modificada a fim de obter uma representação mais compacta (menor escala) ou com maior precisão (escala mais alta). A mudança de resolução é feita agrupando-se células com maior resolução

(células pequenas) para formar uma célula com menor resolução (célula maior). Isto é feito dessa forma porque não é possível dividir uma célula maior em células menores definindo o percentual do objeto dentro das células menores sem se ter acesso ao objeto real. A Figura 4 apresenta um exemplo de coordenadas de células de assinaturas com tamanho de células diferentes. Sendo a, b, c e d números inteiros. A célula menor, com lado de tamanho 2^n, corresponde às coordenadas: (2^na, 2^nb) do canto inferior esquerdo; (2^na + 2, 2^nb) do canto inferior direito; (2^na + 2, 2^nb + 2) do canto superior direito; e, (2^na, 2^nb + 2) do canto superior esquerdo. Já a célula maior, com lado de tamenho 2^{n+1}, corresponde às coordenadas: (2 × 2^nc, 2 × 2^nd) do canto inferior esquerdo; (2 × 2^nc + 2, 2 × 2^nd) do canto inferior direito; (2 × 2^nc + 2, 2 × 2^nd + 2) do canto superior direito; e, (2 × 2^nc, 2 × 2^nd + 2) do canto superior esquerdo.

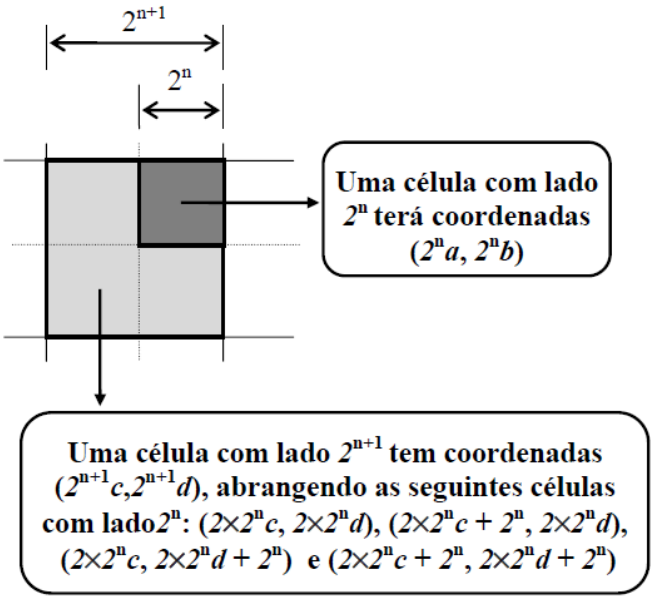

Figura 4 - Exemplo de definição de coordenadas de célula (adaptado de Zimbrão e Souza [1998])

O uso da assinatura 4CRS para processamento de consultas utilizando a arquitetura de Processamento em Múltiplos Passos de Brinkhoff *et al.* [1994] demonstrou bons resultados [Zimbrão e Souza, 1998], bem como o uso da assinatura para processamento aproximado de consultas, como avaliado por Azevedo *et al.* [2004, 2005 e 2006]. Além disso, Antunes e Azevedo [2011] avaliaram o uso da 4CRS para cálculo de similaridade de polígonos e também obtiveram bons resultados. Estes trabalhos apontam a visualização da assinatura

4CRS como trabalho futuro, ou seja, como um trabalho importante, mas que ainda não foi tratado por eles. A visualização da assinatura 4CRS pode auxiliar tanto na depuração e verificação se algoritmos foram implementados corretamente, bem como na montagem de *dashboards* de dados mais simples de mais rápido processamento.

Capítulo 3: TECNOLOGIAS UTILIZADAS

Neste capítulo apresentamos as tecnologias utilizadas no desenvolvimento da aplicação proposta.

3.1 PostGIS

O PostGIS é uma extensão que adiciona suporte a dados espaciais ao SGBD PostgreSQL, habilitando espacialmente o PostgreSQL Server e permitindo que este seja usado como um SGBDE para sistemas de informação geográfica.

O PostGIS segue a Especificação de Elementos Simples para SQL existente na Especificação Abstrata da OpenGIS (Capítulo 2) e foi certificado como compatível para o perfil "Tipos e Funções"[3].

Os objetos GIS suportados pelo PostGIS são um superconjunto dos elementos simples definidos pela OpenGIS. A especificação da OpenGIS define duas formas padrão para expressar dados espaciais: a Well-Known Text (WKT) e a Well-Known Binary (WKB). Exemplos da representação WKT de objetos espaciais são apresentadas a seguir[4]:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOINT(0 0,1 2)
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))

[3] http://postgis.refractions.net/
[4] http://www.postgis.org/docs/

O PostGIS fornece uma biblioteca de funções que auxiliam nas consultas a dados espaciais. As funções *ST_AsText, ST_XMax, ST_YMax, ST_XMin, ST_YMin, ST_Contains* e *ST_Intersects* foram utilizadas na implementação da proposta e são explicadas a seguir.

- text **ST_AsText**(geometry/geography geomA): Esta função retorna a representação WKT(Well-Known Text) de uma geometria/geografia sem o metadado SRID (um exemplo é ilustrado na Figura 5).

```
1. SELECT ST_AsText('0103000000010000000500000000000000000
2. 0000000000000000000000000000000000000000000000000000
3. F03F000000000000F03F000000000000F03F000000000000F03
4. F0000000000000000000000000000000000000000000000000');
5. st_astext
6. ------------------------------
7. POLYGON((0 0,0 1,1 1,1 0,0 0))
8. (1 row)
```

Figura 5 – Exemplo de consulta utilizando a função ST_AsText e seu resultado.

- float **ST_XMax**(geometry A): Esta função retorna a maior coordenada x de um dado geométrico.
- float **ST_YMax**(geometry A): Esta função retorna a maior coordenada y de um dado geométrico.
- float **ST_XMin**(geometry A): Esta função retorna a menor coordenada x de um dado geométrico.
- float **ST_YMin**(geometry A): Esta função retorna a menor coordenada y de um dado geométrico.

A Figura 6 apresenta um exemplo de uso da função *ST_Xmax*, a qual retorna o valor 5.

```
1. SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4,5 6 7)'));
2. st_xmax
3. -------
4. 5
```

Figura 6 - Exemplo de consulta que utiliza a função ST_XMax e seu resultado

- boolean **ST_Contains**(geometry geomA, geometry geomB): Esta função verifica se o parâmetro *geomA* contém o parâmetro *geomB*, ou seja, se *geomB* está inteiramente contido em *geomA*, retornando *true* se

positivo e *false* se negativo. A Figura 7 apresenta um exemplo de uso da função *ST_Contains*, no qual o valor *smallc* representa um círculo pequeno e o valor *bigc* um círculo grande. Neste caso, o círculo pequeno não contém o círculo grande e a função retorna *false.*

```
1. SELECT ST_Contains(smallc, bigc) As smallcontainsbig
2. ---------------
3. f
```

Figura 7 - Exemplo de consulta que utiliza a função ST_Contains e seu resultado

- boolean **ST_Intersects**(geometry geomA, geometry geomB): Esta função verifica se dois objetos geométricos se interceptam, ou seja, é uma função boleana que retorna *true* quando os objetos geométricos compartilham algum espaço, e *false* quando não (Figura 8).

```
1. SELECT ST_Intersects('POINT(0 0)'::geometry,'LINESTRING(0 0,0 2)'
   ::geometry);
2. ----------------
3. t
```

Figura 8 - Exemplo de consulta que utiliza a função ST_Intersects e seu resultado

3.2 PHP

PHP (PHP: Hypertext Preprocessor) é uma linguagem de programação Open Source muito utilizada, designada para o desenvolvimento na Web, podendo ser embutida no HTML. O principal objetivo dessa linguagem é permitir que desenvolvedores web codifiquem páginas web dinâmicas de forma rápida. Contudo a linguagem também tem o potencial para ser utilizada em sistemas mais complexos [Achor *et al.*, 2012] (seção *What is PHP?* - http://br.php.net/manual/en/intro-whatis.php).

Na versão 5 do PHP foi adicionada uma série de recursos com o objetivo de proporcionar a implementação de uma modelagem orientada a objetos mais completa, como apresentado por Achor *et al.* [2012] (seção *Classes and Objects* - http://www.php.net/manual/en/language.oop5.php). Alguns recursos são listados a seguir:

- Visibilidade de uma propriedade ou método, prefixando a declaração com as palavras-chave *public*, *protected* ou *private*.

- Abstração de classe ou métodos.
- Palavra-chave *final*, que previne que classes filhas sobrecarreguem um método ou variável.
- Interfaces de objetos, que permitem a criação de código que especifica quais métodos e variáveis uma classe deve implementar, sem ter que definir como esses métodos serão tratados.

3.3 JavaScript

Javascript é a linguagem de programação da Web usada na grande maioria de *websites* modernos e interpretada em grande parte dos *browsers* modernos. É uma linguagem de alto nível, dinâmica, não tipada e bem adequada tanto a orientação a objetos quanto a estilos de programação funcionais [Flanagan, 2011].

O objeto Window é o principal ponto de entrada para todos os recursos JavaScript do lado do cliente. Este objeto representa a janela ou frame do *browser* e pode ser referenciado pelo identificador *window*. Uma das mais importantes propriedades do objeto Window é a *document*, que se refere ao objeto Document, que representa o conteúdo visualizado na janela. Esta propriedade possui métodos importantes, como o *getElementById()* (Figura 9), que retorna o valor armazenado em um elemento único baseado no valor do atributo identificador *id.*

```
1. //Encontra element com id="timestamp"
2. var timestamp = document.getElementById("timestamp");
```

Figura 9 – Exemplo de uso da função getElementById() [adaptado de Flanagan, 2011]

Um dos mais importantes manipuladores de evento do JavaScript é o *onload* do objeto Window. Ele é disparado quando o conteúdo do documento visualizado na janela está estável e pronto para ser manipulado. Um exemplo de uso desta função é apresentado na Figura 10.

```
1. <script>
2. //Não faz nada até que o document inteiro tenha carregado
3. window.onload = function() {
4. / Encontra todos elementos com a class="reveal"
5. var elements = document.getElementsByClassName("reveal");
6. }
7. </script>
```

Figura 10 – Exemplo de uso do evento *onload* [adaptado de Flanagan, 2011]

A serialização de objetos é o processo de converter um objeto para uma string, a qual pode ser recuperada posteriormente. JavaScript fornece as funções nativas *JSON.stringify()* e *JSON.parse()* (Figura 11) para serializar e recuperar objetos JavaScript. Essas funções utilizam o formato intercâmbio de data JSON, explicado na próxima seção.

```
1. o = {x:1, y:{z:[false,null,""]}}; // Define um objeto de teste
2. s = JSON.stringify(o); // s is '{"x":1,"y":{"z":[false,null,""]}}'
3. p = JSON.parse(s); // p é uma cópia de o
```

Figura 11 – Exemplo de uso das funções de serialização [adaptado de Flanagan, 2011]

3.3.1 JSON

JSON, acrônimo para JavaScript Object Notation, é um formato de texto leve para intercâmbio de dados, completamente independente de linguagem de programação. Sua formatação é fácil de ser analisada e gerada por máquinas, além de ser de fácil leitura e escrita. O JSON é constituído pelas estruturas apresentadas a seguir [Crockford, 2012].

- Uma coleção de pares de nomes e valores, correspondendo em várias linguagens a um objeto, registro, estrutura, dicionário, tabela *hash*, lista de chaves e array associativo.
- Uma lista ordenada de valores, sendo identificado na maioria das linguagens de programação como um array, vetor, lista ou sequência.

Um exemplo de objeto JSON em linguagem JavaScript é apresentado na Figura 12.

```
1. {
2.     "name": "Jack (\"Bee\") Nimble",
3.     "format": {
4.         "type":       "rect",
5.         "width":      1920,
6.         "height":     1080,
7.         "interlace":  false,
8.         "frame rate": 24
9.     }
10.}
```

Figura 12 - Exemplo de objeto JSON

As estruturas de dados citadas são universais e todas as linguagens de programação modernas possuem suporte para pelo menos uma delas. No formato JSON, a estrutura de um objeto toma a forma de um conjunto não-ordenado de pares de atributos e valores. A Figura 13 apresenta o esquema para definição de objeto.

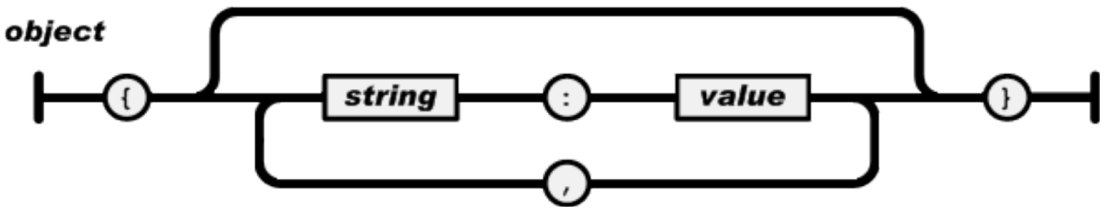

Figura 13 - Estrutura de Objeto JSON [Crockford, 2012]

Por ser um subconjunto de uma notação de objeto da linguagem JavaScript, pode ser utilizado na linguagem sem restrições. A Figura 14 apresenta um exemplo de atribuição do objeto JSON *bindings* à variável *myJSONObject* em linguagem JavaScript. Na Figura 15 é apresentado um exemplo de acesso ao atributo *method* do objeto *bindings*.

```
1. var myJSONObject = {"bindings": [
2.         {"ircEvent": "PRIVMSG", "method": "newURI", "regex":
   "^http://.*"},
3.         {"ircEvent": "PRIVMSG", "method": "deleteURI", "regex":
   "^delete.*"},
4.         {"ircEvent": "PRIVMSG", "method": "randomURI", "regex":
   "^random.*"}
5.     ]
6. };
```

Figura 14 – Exemplo de atribuição de objeto JSON na linguagem JavaScript [Crockford, 2012]

```
1. myJSONObject.bindings[0].method    // "newURI"
```

Figura 15 - Exemplo de acesso a atributos do objeto JSON [Crockford, 2012]

3.3.2 jQuery

Tida como uma linguagem "não muito séria" por muitos desenvolvedores web durante grande parte de seu tempo de vida, JavaScript recuperou seu prestígio nos últimos anos, como resultado de um renovado interesse por alta interatividade, aplicações web da próxima geração e tecnologias AJAX. JavaScript se viu forçada a crescer rapidamente, ao passo que desenvolvedores web deixaram de lado o recurso de copiar e colar código para a conveniência das bibliotecas JavaScript com inúmeros recursos que resolvem as dificuldades de compatibilidade entre os *browsers* e fornecem um novo e aprimorado padrão para o desenvolvimento na web [Bibeault e Katz, 2010].

jQuery é uma biblioteca JavaScript repleta de ferramentas prontas para serem usadas, simplificando o tratamento de documentos HTML, manipulação de eventos, animação e interações AJAX, proporcionando um desenvolvimento web mais rápido e eficaz. Esta biblioteca foi criada com o propósito de alterar a forma de programar na linguagem JavaScript [JQUERY, 2010].

Para utilizar esta biblioteca é necessário incluir o arquivo JavaScript correspondente à biblioteca jQuery na página HTML em que se deseja que métodos jQuery sejam referenciados. A Figura 16 apresenta um exemplo de inclusão da biblioteca.

```
1. <script type='text/javascript' src='js/lib/jquery-1.6.2.js'></script>
```

Figura 16 – Exemplo de include da biblioteca jQuery em uma código HTML.

A biblioteca jQuery possui inúmeras funções que simplificam o desenvolvimento em JavaScript, tornando o código mais limpo e leve. As funções da biblioteca jQuery utilizadas nesse trabalho foram:

- **$("id"):** Recebe como parâmetro o *id* de um elemento da página HTML, oferecendo o mesmo resultado que a função *document.getElementById,* ou seja, retornando uma referência desse elemento para manipulação do mesmo.

- **.hide(“id”)/.show(“id”):** Funções que, respectivamente, escondem e mostram um elemento da página HTML.
- **ajax():** Para esta função existe uma série de parâmetros que podem ser configurados opcionalmente para personalizar uma requisição AJAX e, dentre eles, os seguintes foram utilizados no desenvolvimento desse trabalho:
  - **url**: determina a URL para qual a requisição é enviada;
  - **type**: determina o tipo de requisição HTTP que será enviada, usualmente “POST” ou “GET”, sendo a requisição “GET” definida como padrão;
  - **data**: determina os dados que serão enviados ao servidor, cada dado composto obrigatoriamente por um par de parâmetro e valor;
  - **dataType**: configura o tipo esperado do dado a ser retornado pelo servidor, podendo ser “xml”, “html”, “script”, “json” ou “text”;
  - **context**: determina o contexto dos retornos relacionados ao AJAX;
  - **success**: determina a função a ser executada quando a requisição é realizada com sucesso;
  - **beforeSend**: determina a função a ser executada no intervalo entre o envio da requisição e o recebimento da resposta da mesma;
  - **complete**: determina a função a ser executada quando a requisição é finalizada.

3.4 AJAX

Asynchronous JavaScript and XML (AJAX), nome atribuído em 2005 por Jesse James Garret, é uma técnica utilizada para criar páginas web rápidas, dinâmicas e incrementais [AJAX, 2011]. Não é uma nova linguagem de programação, mas sim uma nova forma de utilizar padrões já existentes. A combinação dos seguintes padrões é utilizada pelo AJAX [W3SCHOOLS, 2011]:

- **Objeto XMLHttpRequest**: Troca dados de forma assíncrona com o servidor;
- **JavaScript/DOM**: Utilizado para mostrar e manipular os dados;
- **CSS**: Utilizado para adicionar estilo aos dados;
- **XML**: Normalmente utilizado como o formato para transmissão dos dados.

O AJAX permite que páginas web sejam atualizadas de forma assíncrona, em segundo plano, trocando pequenas quantidades de dados entre o cliente e o servidor sem a necessidade de recarregar a página web inteira [W3SCHOOLS, 2011].

A Figura 17 apresenta a sequência de uma Requisição AJAX e seus passos são explicados a seguir [Murray e Ball, 2012]:

1. Cliente gera um evento pela interface, como, por exemplo, clicar em um botão, que resulta em uma chamada JavaScript;
2. Um objeto XMLHttpRequest é criado e configurado com um parâmetro de requisição que inclui o *id* do componente que gerou o evento e qualquer valor que o usuário possa ter inserido neste componente, por exemplo, o valor preenchido em um campo de texto;
3. O objeto XMLHttpRequest criado realiza uma requisição assíncrona ao servidor de aplicação, que recebe a requisição e a processa;
4. A página que processou a requisição no servidor retorna um documento XML (na Figura 17, ilustrado como Dados XML) contendo as atualizações que devem ser exibidas no *browser* do cliente;
5. O objeto XMLHttpRequest recebe os dados em XML, os processa e atualiza o HTML apresentando no *browser* do cliente a página web com os novos dados.

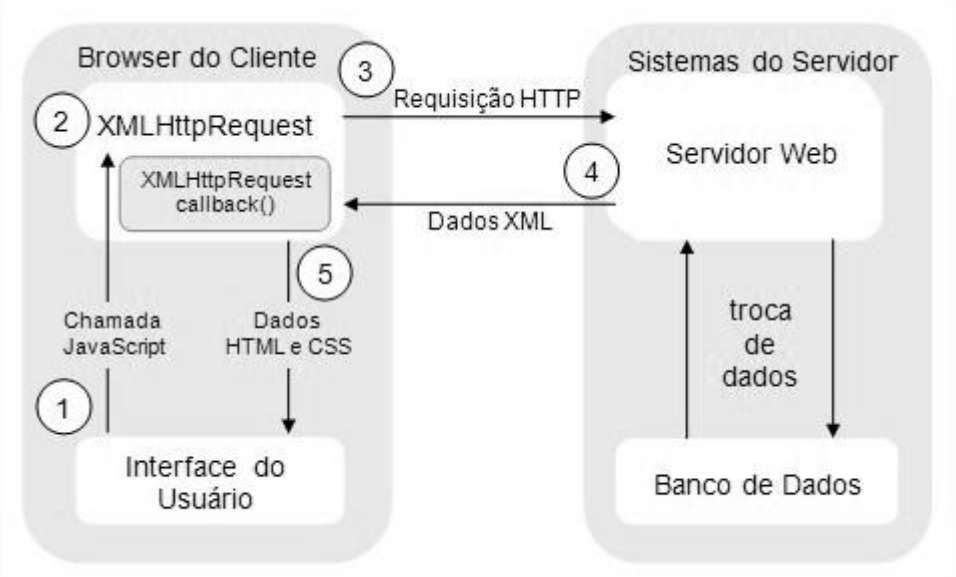

Figura 17 - Sequência de uma Requisição AJAX [Adaptado de Murray e Ball, 2012]

Apesar da letra X em AJAX representar a linguagem de marcação XML, atualmente o JSON (Seção 3.3.1) é mais utilizado como formato de transmissão de dados [MDN, 2012].

3.5 HTML5

A HTML sempre foi definida como a linguagem de marcação da web. Originalmente, esta linguagem foi desenvolvida para descrever documentos científicos. Porém, no decorrer dos anos, seu uso se estendeu para a descrição de inúmeros tipos de documentos [W3C, 2011].

Em 2007, a W3C (World Wide Web Consortium) formou um grupo de trabalho para desenvolver as especificações da quinta versão da HTML, a HTML5. Algumas novidades do HTML5 são elementos como:

- **video e áudio**: Utilizados para conteúdo multimídia;
- **SVG**: Que possibilita a visualização de arquivos do tipo SVG;
- **Canvas**: Utilizado para renderizar gráficos de forma dinâmica.

A história do elemento Canvas e suas características são apresentadas no item a seguir.

3.5.1 Canvas

Originalmente criado pela Apple para uso em seus aplicativos *dashboard* [MDN, 2011], o Canvas é um aplicativo de desenho em 2D, utilizado para desenhar gráficos diretamente em navegadores web via linguagens de script, como JavaScript, sem a necessidade de plugins como Flash ou Java [MDN, 2011]. Foi introduzido na quinta revisão da linguagem de marcação HTML, a HTML5, e possui suporte na maioria dos navegadores web, inclusive no Internet Explorer em sua versão 9 beta [MDN, 2012]. A HTML5 define o elemento <canvas> como uma tela de bitmap dependente de resolução, que pode ser utilizada para renderizar dinamicamente gráficos, gráficos de jogos e outros tipos de imagens [Pilgrim, 2010].

O elemento <canvas> possui dois atributos *width* e *height*. O espaço de coordenadas do Canvas é controlado por estes atributos e possui os valores padrões de 300 para o *width* e 150 para o *height*, os quais são interpretados como CSS pixels. Na Figura 18, é ilustrado como funciona o sistema de coordenadas do Canvas e podemos observar que, diferente de um plano cartesiano, o sistema de coordenadas do Canvas é disposto de cima para baixo e não possui pontos negativos.

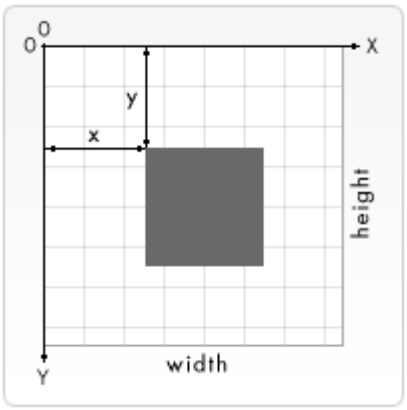

Figura 18 - Sistema de coordenadas do Canvas [MDN, 2012]

Neste projeto, as equações i e ii foram implementadas para realizar a conversão dos pontos dos polígonos e das assinaturas 4CRS para o sistema de coordenadas do Canvas. A equação i obtém a coordenada *x* do Canvas e a equação

ii obtém a coordenada *y* do mesmo. Nota-se que, como o sistema de coordenadas do Canvas é disposto de cima para baixo, é necessário corrigir a coordenada *y* do ponto.

$$canvas.x = \frac{point.x - extent.x.min}{extent.x.max - extent.x.min \ \times canvas.width} \quad \text{(i)}$$

$$canvas.y = \frac{point.y - extent.y.min}{(\ extent.y.max - extent.y.min \ \times -canvas.height)} + canvas.height \quad \text{(ii)}$$

Na Figura 19 vemos as etapas para converter a coordenada y de um plano cartesiano para o plano de coordenadas do Canvas. Na Figura 19.a temos o objeto disposto em um plano cartesiano. A Figura 19.b mostra o mesmo objeto se desenhado diretamente no Canvas, sem a conversão. Na Figura 19.c, com a multiplicação pela altura negativa do Canvas, o objeto se encontra em sua forma correta mas fora do plano, sendo necessário somar a coordenada y a altura do Canvas para obtemos o objeto corretamente visualizado, como mostra a Figura 19.d.

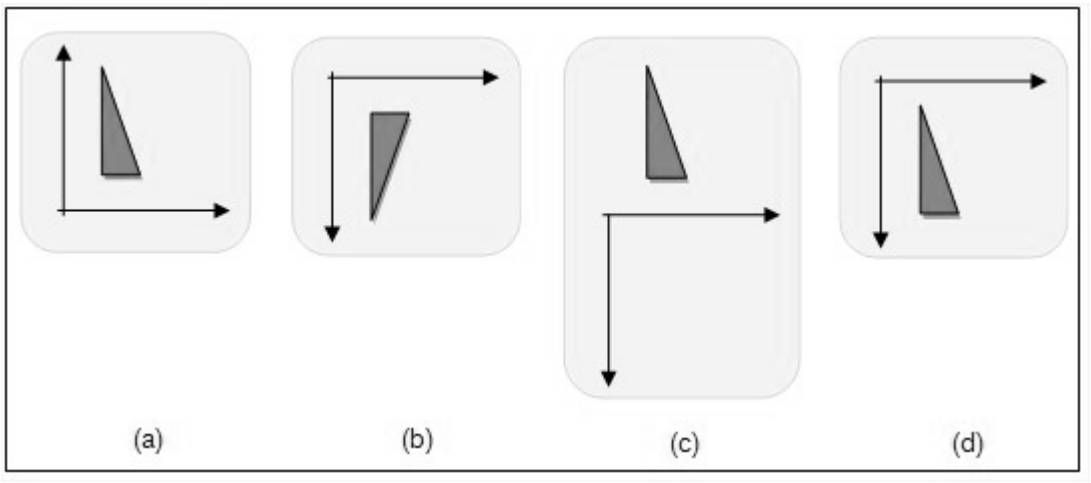

Figura 19 – Correção da coordenada y de plano cartesiano para plano do Canvas

A Figura 20 apresenta um exemplo do uso do elemento Canvas. Observe que quando a página é carregada (evento *onload* no *body* da página HTML) a função JavaScript *draw* é invocada e a curva de Bezier apresentada na Figura 21 é desenhada.

(d)

```
1. <html>
2.  <head>
3.   <script type="application/javascript">
4. function draw() {
5.   var canvas = document.getElementById("canvas");
6.   var ctx = canvas.getContext("2d");
7.
8.   ctx.fillStyle = "red";
9.
10.  ctx.beginPath();
11.  ctx.moveTo(30, 30);
12.  ctx.lineTo(150, 150);
13.  ctx.bezierCurveTo(60, 70, 60, 70, 70, 150);
14.  ctx.lineTo(30, 30);
15.  ctx.fill();
16.}
17.   </script>
18. </head>
19. <body onload="draw()">
20.   <canvas id="canvas" width="300" height="300"></canvas>
21. </body>
22.</html>
```

Figura 20 – Exemplo Elemento Canvas em código HTML[MDN, 2011]

Figura 21 - Desenho exibido na página HTML correspondente ao código apresentado na Figura 20.

O JavaScript oferece funções específicas para criar desenhos no Canvas e a seguir apresentamos as funções que foram utilizadas nesse trabalho.

- **fillRect**(xmin, ymin, width, height): Esta função é utilizada para desenhar retângulos preenchidos com uma determinada cor. Os parâmetros *xmin* e *ymin* determinam o ponto inicial onde o retângulo será desenhado e os parâmetros *width* e *height* representam, respectivamente, a largura e altura em CSS pixels que o desenho do retângulo irá possuir;
- **strokeRect**(xmin, ymin, width, height): Está função é semelhante a função *fillRect* e desenha as bordas de retângulos;

- **moveTo**(x, y): A função *moveTo* é utilizada para iniciar o desenho de um segmento de reta. Seus parâmetros *x* e *y* representam o ponto inicial de onde partirá o segmento;
- **lineTo**(x, y): Esta função é utilizada para dar continuação ao segmento de reta iniciado pela função *moveTo*. Com esta função é possível desenhar linhas e polígonos com inúmeros pontos.

As tecnologias tratadas neste capítulo foram utilizadas no desenvolvimento da proposta apresentada a seguir. Os métodos e funções utilizados na implementação da proposta foram previamente descritos neste capítulo, com o objetivo de proporcionar uma melhor comprensão do código gerado.

Capítulo 4: PASSO-A-PASSO PARA A VISUALIZAÇÃO DE DADOS ESPACIAIS

O passo-a-passo para visualização de dados espaciais na Web é composto de 6 etapas descritas abaixo e segue a arquitetura cliente/servidor de uma aplicação web com acesso a um banco de dados tradicional utilizando o recurso do AJAX para realizar as requisições ao servidor, como exemplificado na Figura 22.

1. Cliente solicita a visualização de determinado dado espacial, enviando uma requisição ao servidor de aplicação;
2. Servidor de aplicação recebe a solicitação;
3. Servidor solicita execução de consulta pelo banco de dados espaciais. A consulta é executada e os dados são retornados para o servidor de aplicação;
4. Servidor de aplicação recebe os dados do banco de dados e os converte para serem exibidos no sistema de coordenadas do Canvas;
5. Servidor converte os dados em formato para transmissão na web para o cliente;
6. Cliente recebe dados codificados do servidor e os manipula para visualização dos mesmos no Canvas.

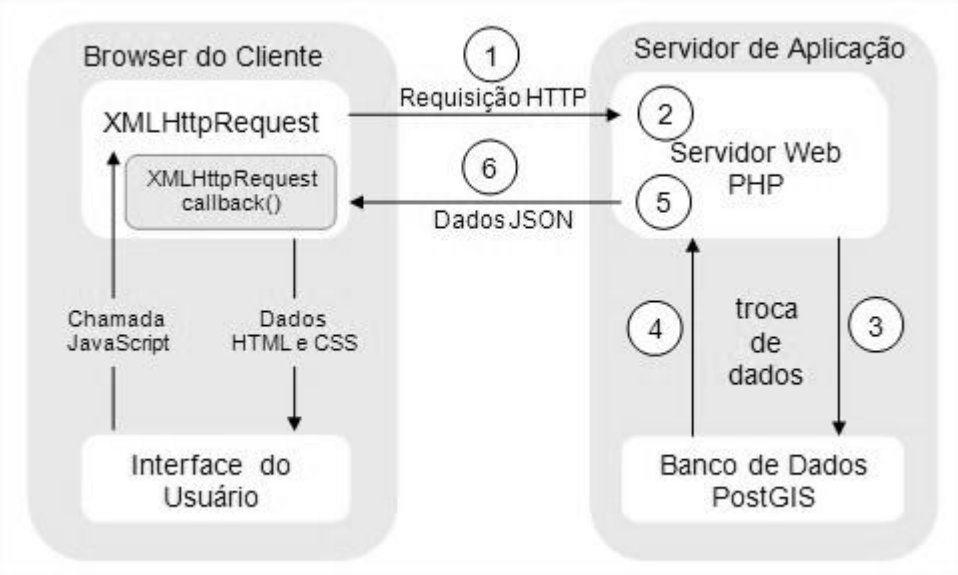

Figura 22 - Modelo de arquitetura Cliente/Servidor

Para a implementação do passo-a-passo proposto neste trabalho foram adotadas as seguintes tecnologias, cujos principais conceitos foram apresentados no Capítulo 3.

- **PHP** (Seção 3.2). A linguagem PHP foi utilizada para implementar componentes que são executados no servidor de aplicação, realizam consultas no banco de dados espaciais a partir de requisições enviadas pelo cliente e retornam os dados obtidos nessas consultas para o mesmo.
- **JavaScript** (Seção 3.3). A linguagem JavaScript foi utilizada para implementar métodos que executam no cliente e enviam as requisições desejadas pelo mesmo ao servidor de aplicação. Nas implementações das funções JavaScript as seguintes tecnologias são também utilizadas:
  - **JSON** (Seção 3.3.1). Este formato leve de intercâmbio de dados entre cliente e servidor permite que no JavaScript os dados sejam manipulados em forma de objetos, facilitando o acesso a eles;
  - **jQuery** (Seção 3.3.2). Utilizando as funções existentes nesta biblioteca é possível simplificar as funções JavaScript (por exemplo, a função *jQuery.ajax())* que envia requisições ao servidor de aplicação utilizando a tecnologia AJAX (Seção 3.4).
- **HTML5** (Seção 3.5). Linguagem de marcação utilizada no desenvolvimento da página de visualização pelo cliente. A principal funcionalidade do HTML5 utilizada foi o **Canvas** (Seção 3.5.1), no qual os dados recebidos pelo cliente são visualizados dinamicamente, através das funções JavaScript disponibilizadas para tal.

A seguir são explicados os detalhes da implementação de cada passo citado acima.

4.1 Cliente solicita visualização de Dado Espacial

Na implementação da solicitação do cliente foi utilizado JavaScript, juntamente com a biblioteca jQuery e os recursos que ela proporciona. A função *ajax*, disponibilizada nesta biblioteca, foi utilizada para realizar requisições HTTP assíncronas ao servidor de aplicação, evitando dessa forma o *reload* da página.

Quando a página HTML é chamada, a função *onload* da mesma é executada. Esta função instancia o elemento <*canvas*> e configura o seu atributo *context* (Figura 23).

- **Linha 5.** A função *document.getElementById* atribui à variável global *canvas* uma referência ao elemento que possui o atributo *id* igual a "canvas" no código em HTML da página index.php. Dessa forma o elemento identificado pelo *id* que está estático na página HTML pode ser alterado dinamicamente utilizando JavaScript.
- **Linha 6.** A variável global *context* é definida a partir do contexto de renderização do Canvas, obtido pelo método *getContext()*. Através desse método é possível acessar o contexto de renderização e as funções de desenho do mesmo.

```
1. var canvas;
2. var context;
3.
4. window.onload = function(){
5.     canvas = document.getElementById("canvas");
6.     context = canvas.getContext("2d");
7. }
```

Figura 23 – Função onload é executada quando a página HTML está estável

A função *ajax* é executada quando a função *obterDadosEspaciais* (Figura 21) é chamada. A Figura 24 apresenta um exemplo de invocação desta função utilizando o evento *onload* da janela.

```
1. window.onload = function(){
2.      obterDadosEspaciais("obterDadosEspaciais","Norte");
3. }
```

Figura 24 – Invocação do método obterDadosEspaciais na página HTML

O código apresentado na Figura 25 é executado assim que a página HTML se encontra estável, trazendo os municípios da região norte do Brasil. Os parâmetros *type, url, data, dataType e context* (linhas 4 a 12) definem as configurações necessárias para que a requisição seja enviada ao servidor. Os parâmetros *success, beforeSend e complete* (linhas 14 a 22) definem o comportamento da função após o envio da requisição ao servidor de aplicação. O que se refere ao primeiro passo, em que o cliente solicita a visualização do dado espacial, é explicado a seguir:

- **Linha 3.** *$* indica que a biblioteca jQuery é referenciada. O método *.ajax* indica que a função *.ajax* desta biblioteca é utilizada;
- **Linha 4.** Define como "POST" o tipo de requisição HTTP a ser enviada ao servidor de aplicação;
- **Linha 5.** Define que o arquivo php "php/src/control/control.multipoligono.php" receberá a requisição a ser executada no servidor de aplicação;
- **Linhas 6 a 10.** Define o nome dos parâmetros e o conteúdo dos mesmos que serão enviados na requisição ao servidor. Neste caso, o valor do parâmetro *func* indica qual operação será realizada pela página php no servidor (correspondente ao passo 2 da arquitetura). Exemplos de funcionalidades executadas por esta página são: seleção de polígono, zoom por ponto, zoom por seleção;
- **Linha 11.** Determina que os dados que serão retornados pelo servidor estão no formato JSON;
- **Linha 12.** Determina que *document.body* é o elemento de contexto para todos os retornos relacionados ao AJAX.

```
1.  function obterDadosEspaciais(func, filtro){
2.
3.      $.ajax({
4.          type: "POST",
5.          url: "php/src/control/control.multipoligono.php",
6.          data: ({
7.              func: func,
8.              filtro: filtro,
9.              extensao: extensaoHistorico[extensaoHistorico.length-1]
10.         }),
11.         dataType: "json",
12.         context: document.body,
13.
14.         success: function(data){
15.             desenhaMultiPoligono(func, data);
16.         },
17.         beforeSend: function () {
18.             $('#loading').show();
19.         },
20.         complete: function () {
21.             $('#loading').hide();
22.         }
23.     });
24. }
```

Figura 25 - Método que envia solicitação de visualização de dados espaciais ao servidor de aplicação

4.2 Servidor recebe solicitação

O servidor de aplicação recebe a requisição feita pelo cliente na página requisitada pelo mesmo. Neste caso, o cliente requisitou a execução da página "control.multipoligono.php" (linha 5 da Figura 25). Esta página (apresentada na Figura 26) recebe os parâmetros enviados via AJAX (*func* e *filtro*) e, a partir do valor do parâmetro *func*, invoca uma operação de acesso a dados. Esta página (Figura 26) é explicada abaixo:

- **Linha 1.** O arquivo dao.multipoligono.php é incluído na página para possibilitar que um objeto da classe MultiPoligonoDAO seja instanciado posteriomente;
- **Linhas 3 e 4.** Os valores dos parâmetros *func* e *filtro* são atribuídos, respectivamente, às variáveis *$func* e *$filtro*, tornando o acesso a esses valores mais simplificado;
- **Linha 6.** A condição verifica se o valor da variável *$func* é igual ao texto *"obterDadosEspaciais"* e, sendo confirmado, as linhas de código dentro da condição são executadas;

- **Linha 7.** O parâmetro *$filtro* é passado para o método *obterDadosEspaciais*, que retorna uma lista de objetos do tipo MultiPoligono, atribuído a variável *$multipoligonos;*
    - O método *obterDadosEspaciais* é executado através da instância da classe MultiPoligonoDAO acessada pelo método *getInstancia,* no qual foi implementado o padrão de projeto Singleton, que garante a existência de apenas uma instância da classe MultiPoligonoDAO e provê um ponto de acesso global a mesma [PHP, 2012].
- **Linha 8.** A lista de objetos MultiPoligono, contida na variável *$multipoligonos*, é convertida para JSON, através da função do PHP *json_encode,* e retornada ao cliente através da função *echo,* também do PHP.

```
1. include_once("../model/dao.multipoligono.php");
2.
3. $func = $_REQUEST['func'];
4. $filtro = $_REQUEST['filtro'];
5.
6. if ($func == 'obterDadosEspaciais') {
7.     $multipoligonos = MultiPoligonoDAO::getInstancia()->
   obterDadosEspaciais($filtro);
8.     echo json_encode($multipoligonos);
9. }
```

Figura 26 - Script no servidor de aplicação que recebe requisição enviada pelo cliente

4.3 Servidor solicita execução de consulta pelo Banco de Dados Espaciais

A execução da consulta pelo banco de dados espaciais é feita pelo método *obterDadosEspaciais* da classe MultiPoligonoDAO, apresentado na Figura 27 e explicado a seguir. Este método é invocado pela página "control.multipoligono.php" (linha 7 da Figura 26).

- **Linha 3.** Na string da consulta SQL é definida a tabela (d_geometry) em que os dados espaciais serão consultados, bem como o atributo (g_region) que será comparado para filtrar estes dados. A função do

PostGIS *ST_AsText* é utilizada para converter os dados do tipo geometry, armazenados no banco de dados espaciais, para uma forma textual, possibilitando o acesso aos dados espaciais pelo PHP;

- **Linha 8.** Uma conexão com o banco de dados espaciais é instanciada para que a consulta SQL possa ser executada;
- **Linha 9.** A consulta SQL é executada no banco de dados e os dados retornados pela mesma são atribuídos a variável *$result;*
- **Linha 11.** A extensão obtida pelo método *getExtensao* (Figura 28), explicado no próximo passo, é atribuída à variável *$extensao*.
    - Extensão é um retângulo de limite mínimo (xmin, ymin e xmax, ymax) definido por pares de coordenadas de uma fonte de dados. Todas as coordenadas desse conjunto de dados estão contidas dentro desse limite [WADE, 2006];
- **Linhas 13 e 14.** Para cada linha de resultado obtido pela consulta SQL, um objeto MultiPoligono é criado e, nas linhas seguintes, ele é tratado e inserido numa lista de objetos MultiPoligono;
- **Linha 15.** O método *converterWKTtoPoligonos* (apresentado na Figura 29 e explicado na próxima seção) retorna os dados espaciais convertidos para o sistema de coordenadas do Canvas;
- **Linha 16.** O atributo *poligonos* do objeto *$multipoligono* é definido;
- **Linha 17.** O objeto *$multipoligono* é inserido numa lista de objetos do tipo MultiPoligono;
- **Linha 19.** A lista de MultiPoligonos é retornada pelo método.

```
1. public function obterDadosEspaciais($filtro) {
2.
3.     $sql = "SELECT ST_AsText(g_map) as wkt
4.             FROM d_geometry
5.             WHERE g_region='" . $filtro . "'
6.             ORDER BY sky_geometry";
7.
8.     $this->conexao = new Connection();
9.     $result = $this->conexao->executeSQL($sql);
10.
11.    $extensao = $this->getExtensao(null);
12.
13.    while ($row = pg_fetch_row($result)) {
14.        $multipoligono = new MultiPoligono();
15.        $poligonos = $this->converterWKTtoPoligonos($row[0], $extensao);
16.        $multipoligono->poligonos = $poligonos;
17.        $listamultipoligonos[] = $multipoligono;
18.    }
19.    return $listamultipoligonos;
20.}
```

Figura 27 - Método que obtém os dados espaciais do banco de dados espaciais

4.4 Servidor recebe dados do Banco de Dados

Como explicado no seção 3.5.1, existem peculiaridades no sistema de coordenadas do Canvas para as quais os dados obtidos na consulta ao banco de dados precisam receber tratamento, tornando assim a visualização dos mesmos possível. É preciso adaptar proporcionalmente a extensão dos dados espaciais com a extensão do Canvas (que representa a janela onde os dados serão exibidos) e, para tal, cada ponto contido no polígono dos dados espaciais deve ser convertido para um ponto do Canvas. Para essa conversão, é preciso ter o conhecimento da extensão dos dados espaciais e do tamanho definido para o Canvas, que também determina o limite do mesmo.

O método *getExtensao*, que é chamado na linha 11 da Figura 27 e mostrado na Figura 28, determina a extensão dos dados espaciais obtidos. Para tal, utilizamos as funções do PostGIS *ST_XMax*, *ST_YMax*, *ST_XMin* e *ST_YMin*, apresentadas na seção 3.1. Criando uma consulta combinando as funções citadas, conseguimos obter a extensão do conjunto de dados espaciais. A consulta que obtém as coordenadas mínimas e máximas é apresentada das linhas 3 a 7 da Figura 28. O resultado obtido é armazenado em um objeto do tipo Extensao que possui dois pontos como atributos.

```
1. public function getExtensao($id) {
2.     $sql = "SELECT MAX(ST_XMAX(g_map)) as xmax,
3.             MAX(ST_YMAX(g_map)) as ymax,
4.             MIN(ST_XMIN(g_map)) as xmin,
5.             MIN(ST_YMIN(g_map)) as ymin
6.             FROM d_geometry";
7.
8.     $result = $this->conexao->executeSQL($sql);
9.     $row = pg_fetch_row($result);
10.
11.    $max = new Ponto();
12.    $max->x = round($row[0], 4);
13.    $max->y = round($row[1], 4);
14.
15.    $min = new Ponto();
16.    $min->x = round($row[2], 4);
17.    $min->y = round($row[3], 4);
18.
19.    $extensao = new Extensao();
20.    $extensao->max = $max;
21.    $extensao->min = $min;
22.
23.    return $extensao;
24.}
```

Figura 28 - Método que obtém os limites da extensão dos dados espaciais

Na Linha 15 da Figura 27, o método *converterWKTtoPoligonos* visualizado na Figura 29 é chamado, e é nele que a conversão dos dados espaciais para o sistema de coordenadas do Canvas é realizada e os objetos instanciados. Este método foi desenvolvido considerando as equações apresentadas na seção 3.5.1. As partes importantes do método *converterWKTtoPoligonos* (Figura 29**)** são explicadas abaixo:

- **Linha 26 a 37.** O loop apresentado da linha 26 a 37 é executado para cada polígono existente no conjunto de dados. Nele, cada ponto do polígono é convertido para um ponto do sistema de coordenadas do Canvas, considerando a altura e largura do Canvas, bem como os limites dos dados do banco. Pelo fato do sistema de coordenadas do Canvas ter suas coordenadas *x* e *y* começando de cima para baixo no topo esquerdo (seção 3.5.1), e os pontos dos dados do banco começarem de baixo pra cima, no canto esquerdo, é necessário multiplicar a coordenada *y* por -1, para que a visualização dos dados não fique invertida.

```
1. public function converterWKTtoPoligonos($wkt, $extensao) {
2. 
3.     //correção para zoom proporcional do município
4.     $distanciaX = abs($extensao->max->x - $extensao->min->x);
5.     $distanciaY = abs($extensao->max->y - $extensao->min->y);
6. 
7.     if ($distanciaX > $distanciaY) {
8.         $largura = $this->canvasW;
9.         $altura = ($distanciaY * $largura) / $distanciaX;
10.    } else {
11.        $altura = $this->canvasH;
12.        $largura = ($distanciaX * $altura) / $distanciaY;
13.    }
14. 
15.    $wkt = substr_replace($wkt, "", 0, 15);
16.    $wkt = substr($wkt, 0, -3);
17.    $wkt = explode(")),((", $wkt);
18. 
19.    //para cada poligono
20.    for ($i = 0; $i < sizeof($wkt); $i++) {
21. 
22.        $wkt = explode(",", $wkt[$i]); //separa os pontos do poligono
23.        //para cada ponto do poligono
24. 
25.        for ($j = 0; $j < sizeof($wkt); $j++) {
26. 
27.            $xy = explode(" ", $wkt[$j]); //separa o X e Y do ponto
28. 
29.            $xc = (round($xy[0], 4) - $extensao->min->x) / ($extensao-
   >max->x - $extensao->min->x) * $largura;
30.            $yc = ((round($xy[1], 4) - $extensao->min->y) / ($extensao-
   >max->y - $extensao->min->y) * -$altura) + $altura;
31. 
32.            $ponto = new Ponto();
33.            $ponto->x = round($xc, 4);
34.            $ponto->y = round($yc, 4);
35.            $pontos[] = $ponto;
36.        }
37.        $poligono = new Poligono();
38.        $poligono->pontos = $pontos;
39.        $poligonos[] = $poligono;
40.    }
41.    return $poligonos;
42. }
```

Figura 29 - Método que converte cada ponto dos dados espaciais em um ponto do sistema de coordenadas do Canvas.

4.5 Servidor converte dados em formato para transmissão na Web

A lista de objetos MultiPoligono, retornada pelo método *obterDadosEspaciais* na linha 19 da Figura 28, é atribuída à variável

$multipologonos na linha 8 da Figura 26 e codificada para o tipo JSON na linha 9 da mesma figura para ser enviada ao cliente.

4.6 Cliente recebe dados do Servidor

Voltando à Figura 25, das linhas 14 a 22 são definidas as ações a serem tomadas após o cliente ter recebido os dados do servidor. Tais linhas são explicadas abaixo

- **Linha 14 e 15**. No parâmetro *success* é atribuída uma função que executa o método *desenhaMultiPoligono* (Figura 30) ;
- **Linha 17.** O parâmetro *beforeSend* invoca uma função que insere uma ampuleta na visualização, simbolizando que a requisição foi enviada e está sendo processada pelo servidor**;**
- **Linha 20.** O parâmetro *complete* finaliza a função invocada no parâmetro *beforeSend.*

A função *desenhaMultiPoligono*, apresentada na Figura 30, recebe como parâmetro o conjunto de dados retornados do servidor de aplicação. Estes dados são compostos por uma lista de objetos MultiPoligono.

Cada objeto MultiPoligono contém uma lista de objetos Polígono, e para cada um a função *desenhaPoligono* é chamada. A função desenhaMultiPoligono é explicada a seguir.

- **Linha 4.** Ao atributo *strokeStyle* do contexto de renderização é definida a cor da linha que será desenhada no Canvas, neste caso a cor preta, representada pelo código hexadecimal “#000000”;
- **Linhas 7, 9 e 14.** Semelhante ao atributo *strokeStyle*, o atributo *fillStyle* define a cor de preenchimento do polígono que será desenhado no Canvas;
- **Linha 12.** Para apagar os dados anteriormente visualizados, é necessário reinicializar o atributo *width* do Canvas atribuindo ao mesmo o valor *canvas.width*.

- **Linha 18.** A extensão dos dados é armazenada em um array para ser utilizada posteriormente;
- **Linha 16 a 24.** O método *desenhaPoligono* é chamado para cada polígono contido dentro de um multipoligono**.**

```
1.  function desenhaMultiPoligono(func, data){
2.
3.      var multipoligono;
4.      context.strokeStyle  = '#000000';
5.
6.      if(func == "obterDadosEspaciais")
7.          context.fillStyle = "#F5DEB3";
8.      if(func == "marcarPonto")
9.          context.fillStyle = "#F4A460";
10.
11.     if(func == "zoomSelecao" || func == "zoomPonto"){
12.         canvas.width = canvas.width;
13.         context.fillStyle = "#F5DEB3";
14.     }
15.
16.     for (var j in data){
17.         if(j == 0 && func !="obterDadosEspaciais" && func
    !="marcarPonto"){
18.          extensaoHistorico.push(data[j]);
19.         }
20.         multipoligono = data[j];
21.         for(var i in multipoligono.poligonos){
22.             desenhaPoligono(multipoligono.poligonos[i]);
23.         }
24.     }
25. }
```

Figura 30 - Método que desenha no Canvas do HTML5

A função *desenhaPoligono* é descrita a seguir:

- **Linha 3.** o método *beginPath* do contexto de renderização indica que um desenho no Canvas será iniciado;
- **Linha 4.** O atributo *lineWidth* define a espessura da linha que será desenhada no Canvas;
- **Linhas 5 a 16**. Cada ponto do objeto polígono é desenhado no Canvas separadamente. Para cada polígono, o primeiro ponto é posicionado no Canvas pela função *moveTo*, que recebe as coordenadas x e y do ponto como parâmetro. A partir do segundo ponto do polígono, a função *lineTo*, que também recebe as coordenadas x e y como parâmetros, é utilizada para desenhar os pontos no Canvas;

- **Linha 14**. O método *fill* é chamado, indicando que os polígonos desenhados serão preenchidos com a cor definida previamente pelo atributo *fillStyle;*
- **Linha 15**. O método *stroke*, semelhante ao método *fill*, indica que a borda que delimita os polígonos também será desenhada, de acordo com a cor definida previamente pelo atributo *strokeStyle*.

```
1.  function desenhaPoligono(poligono){
2.
3.      context.beginPath();
4.      context.lineWidth = 0.3;
5.      for (var i in poligono.pontos){
6.
7.          if(i == 0){
8.              context.moveTo(poligono.pontos[i].x,poligono.pontos[i].y);
9.          }
10.         else{
11.             context.lineTo(poligono.pontos[i].x,poligono.pontos[i].y);
12.         }
13.     }
14.     context.fill();
15.     context.stroke();
16. }
```

Figura 31 – Função que desenha polígono no Canvas

O passo-a-passo proposto tem como objetivo exemplificar as etapas necessárias para desenvolver uma aplicação web para visualização de dados espaciais. No Capítulo 5 a seguir veremos a aplicação desse passo-a-passo para um determinado conjunto de dados espaciais e analisaremos o resultado obtido.

Capítulo 5: APLICAÇÃO DA PROPOSTA

5.1 Preparação do Ambiente

O ambiente de desenvolvimento preparado para desenvolver a aplicação proposta é composto das seguintes versões de softwares listadas abaixo:

- PHP versão 5.3.6 (http://www.php.net/downloads.php);
- Apache versão 2.2.17 (http://httpd.apache.org/download.cgi);
- PostgreSQL versão 9.0(http://www.postgresql.org.br/downloads);
- PostGIS versão 1.5 (http://www.postgis.org/download/).

Para executar a aplicação adequadamente, foi necessária a preparação de um servidor, no qual os softwares listados foram instalados. Nenhuma configuração adicional a padrão foi realizada.

5.2 Modelagem do sistema com UML

A modelagem de dados na linguagem UML foi utilizada para identificar as classes, seus atributos e os relacionamentos entre elas. A Figura 32 apresenta o diagrama de classes correspondente ao modelo conceitual das classes implementadas para a visualização de dados espaciais sem considerar o modelo da assinatura 4CRS.

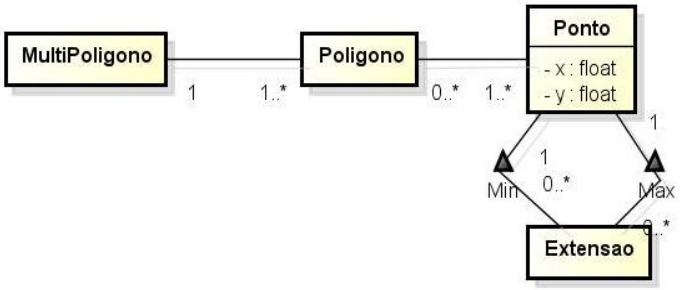

Figura 32 – Diagrama de classes

5.3 Detalhes de Uso da Proposta

Esta proposta abrange a visualização de dados espaciais correspondentes a polígonos e as assinaturas 4CRS destes polígonos. O passo-a-passo apresentado no Capítulo 4 foi inicialmente construído para a exibição de polígonos. Contudo, novos métodos e classes foram adicionados para realizar a exibição das assinaturas 4CRS.

5.3.1 Visualização de Dados Espaciais

Os dados espaciais escolhidos para visualização correspondem aos municípios do Brasil, onde cada município é representado por um dado espacial do tipo *multipolygon*.

Para armazenar estes dados foi criado um banco de dados denominado *municipios*, e nele a tabela *d_geometry*, na qual foram incluídos dados espaciais e não espaciais relativos aos municípios do Brasil. O script para criação da tabela d_geometry é apresentado na Figura 33. A tabela contém dados espaciais e escalares e a coluna g_map corresponde ao dado espacial.

```
1.  CREATE TABLE d_geometry (
2.
3.  sky_geometry integer NOT NULL,
4.  g_city character varying(50),
5.  g_state character varying(50),
6.  g_nation character varying(25),
7.  g_region character varying(25),
8.  g_map geometry,
9.
10. CONSTRAINT enforce_dims_g_map CHECK ((ndims(g_map) = 2)),
11. CONSTRAINT enforce_geotype_g_map CHECK (((geometrytype(g_map) =
    'MULTIPOLYGON'::text) OR (g_map IS NULL))),
12. CONSTRAINT enforce_srid_g_map CHECK ((srid(g_map) = (-1)))
13. );
```

Figura 33 - Script da tabela d_geometry

Uma vez que o passo-a-passo apresentado foi construído inicialmente para exibir polígonos, não houve necessidade de nenhuma alteração no código implementado para exibir os polígonos armazenados na tabela d_geometry.

Como resultado da aplicação do passo-a-passo, a Figura 34 apresenta os dados espaciais armazenados visualizados no Canvas.

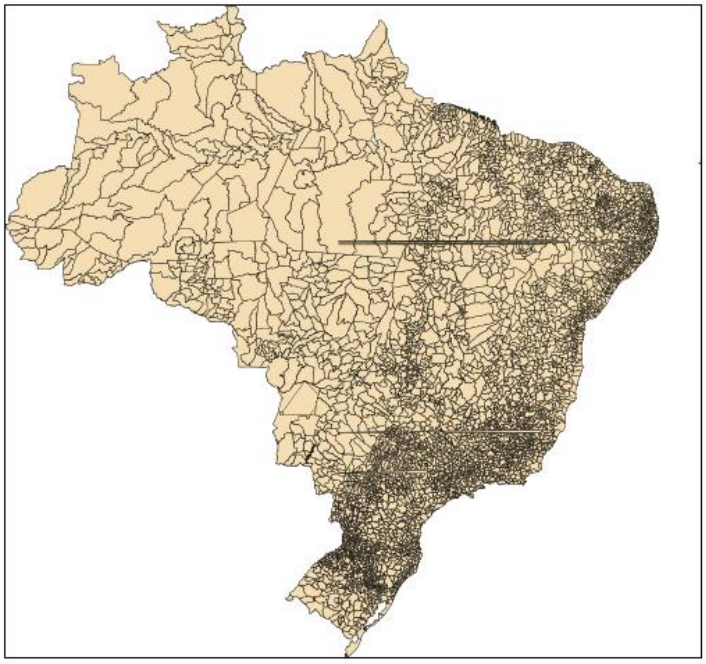

Figura 34 – Exemplo de visualização de dados espaciais no Canvas do HTML5

5.3.2 Visualização de Assinaturas 4CRS

Para as assinaturas 4CRS, a implementação desenvolvida no passo-a-passo teve que ser alterada considerando alguns aspectos. A assinatura 4CRS de um polígono caracteriza um novo objeto e para a manipulação de seus dados foi necessário criar uma nova classe chamada Raster com seus respectivos atributos. Além dos atributos que compõem uma assinatura 4CRS, explicados na seção 2.3, um novo atributo *celula* foi criado para receber dados referentes à célula inicial de uma assinatura 4CRS, calcula dos a partir dos atributos *xmin*, *ymin* e *size* da mesma assinatura.

Foram criados métodos de acesso ao banco de dados para obter os dados das assinaturas e criar os objetos. O atributo *celula*, único que não se encontra armazenado na tabela raster, é definido por dois pontos já convertidos para visualização no Canvas, que representam o ponto mínimo e máximo da primeira célula de uma assinatura. Este atributo, juntamente com o atributo *weight*, será responsável pela visualização da assinatura 4CRS no Canvas.

A posição da primeira célula é identificada pelo seus pontos mínimo e máximo, que definem sua extensão. O ponto mínimo pode ser obtido pelo MBR da

assinatura (xmin, ymin), e o ponto máximo pela soma das coordenadas do ponto mínimo com o tamanho (*size*) da célula, ou seja, *xmax = xmin+size* e *ymax = ymin + size*. Estes pontos são convertidos para pontos em potência de dois e, em seguida, convertidos para as coordenadas do Canvas.

O método *converterCelula*, apresentado na Figura 35 e semelhante ao método *converterWKTtoPoligonos* explicado no passo-a-passo, foi criado para realizar essa conversão. Este método recebe como parâmetro a primeira célula da assinatura e o objeto do tipo Extensão, que representa a extensão em potência de 2 de todas as assinaturas. A partir do conhecimento da largura e altura do Canvas e dos pontos mínimo e máximo da célula inicial da assinatura, podemos redimensionar os mesmos como apresentado na Figura 35.

```
1. public function converterCelula($cel, $ext) {
2.
3.     $w = MultiPoligonoDAO::getInstancia()->canvasW;
4.     $h = MultiPoligonoDAO::getInstancia()->canvasH;
5.
6.     $cel[xmax] = ($cel[xmax] - $ext->min->x) / ($ext->max->x - $ext-
   >min->x) * $w;
7.     $cel[ymax] = (($cel[ymax] - $ext->min->y) / ($ext->max->y - $ext-
   >min->y) * -$h) + $h;
8.
9.     $cel[xmin] = ($cel[xmin] - $ext->min->x) / ($ext->max->x - $ext-
   >min->x) * $w;
10.    $cel[ymin] = (($cel[ymin] - $ext->min->y) / ($ext->max->y - $ext-
   >min->y) * -$h) + $h;
11.
12.    return $cel;
13.}
```

Figura 35 – Método que converte pontos das células para serem exibidos no Canvas

Também foi necessária a criação de novas funções em JavaScript. A função *desenhaGeoGrid* (Figura 36) foi criada para desenhar as assinaturas 4CRS em posição relativa a todas as assinaturas. Esta função recebe como parâmetro um objeto Raster em formato JSON e a partir dos seus atributos, realiza a visualização.

A partir dos atributos *dx* e *dy*, que representam respectivamente o número de linhas e colunas da assinatura, criamos um loop para desenhar cada célula do grid. A função *fillRect*, do Canvas explicada na Seção 3.5.1, é responsável por desenhar cada célula do grid no Canvas.

```
1. function desenhaGeoGrid(data){
2.     var numRows = data.dy;
3.     var numCols = data.dx;
4.     var weight = data.weight;
5.     var colors = weight.split(" ");
6.     var grid = data.grid;
7.     var xpos = grid[0].xmin;
8.     var ypos = grid[0].ymin;
9.     var size = grid[0].xmax - grid[0].xmin;
10.
11.    for(var i=0; i<numRows; i++){
12.        for(var j=1; j<=numCols; j++){
13.
14.            defineCores4CRS(colors[i]);
15.            if(colors[i] != 0){
16.                contextRaster.fillRect(xpos,ypos,size,size);
17.            }
18.            xpos+=size;
19.        }
20.        ypos-=size;
21.        xpos = grid[0].xmin;
22.    }
23.}
```

Figura 36 – Função que desenha assinatura raster 4CRS no Canvas

No banco de dados municípios foi adicionada uma tabela denominada raster (Figura 37), na qual foram inseridas as assinaturas 4CRS de polígonos correspondentes a municípios da região Norte do Brasil e os polígonos dos mesmos dados. A visualização das assinaturas 4CRS e dos polígonos dos quais elas foram geradas é exibida na

Figura 38 e as colunas da tabela raster (Figura 37) são descritas da seguinte forma:

- **id_raster**: identificador da assinatura;
- **id**: identificador do objeto espacial a partir do qual foi gerada a assinatura raster;
- **type**: indica o tipo da assinatura. Neste trabalho, o tipo será sempre 4CRS;
- **x_min, x_max, y_min, y_max**: corresponde às coordenadas do MBR do objeto a partir do qual foi gerada a assinatura;
- **size**: corresponde ao tamanho do bloco em potência de 2, ou seja, tamanho da borda da célula;
- **dx**: indica o número de células do grid no eixo x;

- **dy**: indica o número de células do grid no eixo y;
- **weight**: corresponde a uma cadeia de caracteres separados por um espaço, onde cada caracter representa o tipo de cada célula do grid, sendo 0 para vazio, 1 para pouco, 2 para muito e 3 para cheio. Por exemplo, se o grid tem 4 células, esta cadeia de caracteres armazenaria 0 1 2 3, indicando os tipos vazio, pouco, muito e cheio para cada uma das células do grid;
- **g_map**: indica a geometria do polígono do qual a assinatura 4CRS foi gerada.

```
1.  CREATE table raster(
2.
3.  id_raster SERIAL NOT NULL,
4.  id INT NOT NULL,
5.  type INT NOT NULL,
6.  x_min INT NOT NULL,
7.  y_min INT NOT NULL,
8.  x_max INT NOT NULL,
9.  y_max INT NOT NULL,
10. size INT NOT NULL,
11. dx INT NOT NULL,
12. dy INT NOT NULL,
13. weight VARCHAR NOT NULL,
14.
15. CONSTRAINT raster_pk   PRIMARY KEY(id_raster)
16. );
17. SELECT AddGeometryColumn('raster', 'g_map', -1, 'MULTIPOLYGON', 2 );
```

Figura 37 - Script da tabela raster

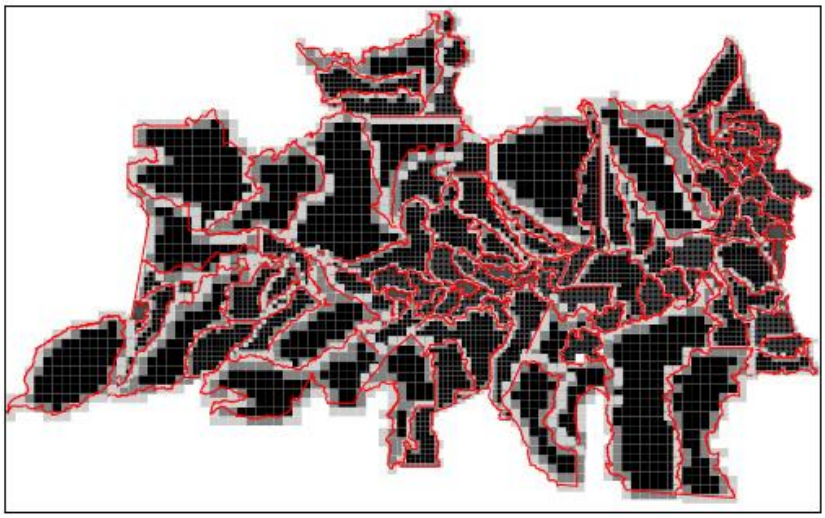

Figura 38 – Visualização das assinaturas raster 4CRS e seus polígonos

5.4 Ferramentas Auxiliares

Com o objetivo de proporcionar uma maior interação entre o usuário e os dados espaciais visualizados no Canvas, foram desenvolvidas três ferramentas auxiliares - ferramenta para seleção de polígono, ferramenta para zoom por ponto e ferramenta de zoom por seleção - as quais são apresentadas a seguir.

5.4.1 Ferramenta de Seleção

A ferramenta de seleção de polígonos consiste em, ao clicar com o botão direito do mouse em algum ponto do Canvas, se o ponto estiver contido dentro de um polígono ou mais visualizados, estes são destacados com uma nova cor. As etapas para realizar a seleção de polígonos são semelhantes às apresentada no Capítulo 4 e ocorrem da seguinte forma:

1. O usuário informa a ação que deseja executar selecionando o *checkbox* referente a opção “Marcar Polígonos”;
2. O usuário clica com o cursor na área desenhada do Canvas que deseja marcar, disparando o evento *onclick* do Canvas;
3. O evento *onclick* executa uma função JavaScript que captura as coordenadas x e y da posição em que o evento foi disparado;
4. O mesmo evento executa a função *obterDadosEspaciais* (seção 4.1), enviando o ponto capturado como parâmetro;
5. A função *obterDadosEspaciais* envia a requisição ao servidor de aplicação. A requisição é composta pelo nome da função que será executada no servidor, atribuído ao parâmetro *func*, e o ponto capturado, atribuído ao parâmetro *filtro*;
6. O servidor recebe a requisição e, a partir dos parâmetros recebidos, instancia um objeto MultiPoligonoDAO, executando o seu método *marcarPoligono*;

7. O método *marcarPoligono* realiza a consulta ao banco de dados espaciais e retorna esses dados, que são convertidos para o formato JSON;
8. Os dados já convertidos são retornados ao cliente, na função *obterDadosEspaciais*, que chama a função que os desenha no Canvas.

Na consulta realizada no banco de dados, a função *ST_Contains*, explicada na Seção 3.1, é utilizada para consultar os polígonos existentes que contém o ponto escolhido pelo usuário. A consulta SQL é exemplificada na Figura 39 e o seu resultado é ilustrado graficamente na Figura 40.

```
1. SELECT sky_geometry as id, ST_AsText(g_map) as wkt
2. FROM d_geometry
3. WHERE ST_Contains(g_map, GeomFromText('POINT(-1533.656825
   27.508300620623)'))
```

Figura 39 – Consulta SQL que retorna os polígonos que contém o ponto informado

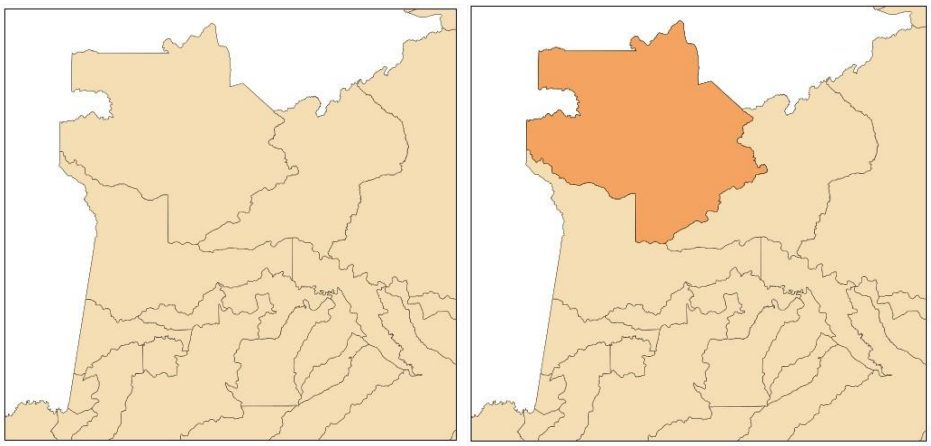

Figura 40 – Dados visualizados antes e após a seleção

5.4.2 Ferramenta de Zoom por Ponto

A ferramenta de zoom por ponto é semelhante à ferramenta de seleção. Sua diferença básica é que a extensão dos dados é alterada, passando a ser a extensão dos dados onde o ponto capturado está contido. Dessa forma, os dados retornados são visualizados em um tamanho maior, proporcionalmente a extensão dos mesmos. Em outras palavras, apenas os polígonos que contenham o ponto

clicado são exibidos no Canvas e a extensão de exibição corresponde à extensão dos polígonos selecionados.

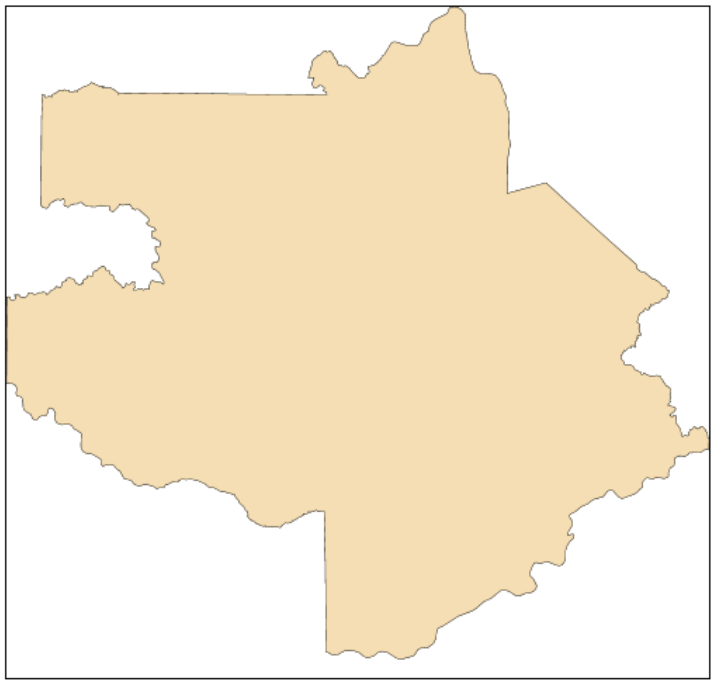

Figura 41 – Polígono visualizado após a execução da ferramenta de zoom por ponto

5.4.3 Ferramenta de Zoom por Seleção

A ferramenta de zoom por seleção consiste em, ao selecionar uma área dentro do Canvas, os polígonos que são interceptados ou estão contidos nessa área são visualizados em um tamanho maior, proporcionalmente a extensão dos mesmos.

A função *ST_Intersects* do PostGIS é ideal para criar a ferramenta de zoom por seleção, pois ela não só retorna os polígonos que estão contidos no polígono informado, como também os que são interceptados por ele. Podemos observar o uso dessa função na Figura 42. A Figura 43 apresenta o resultado da realização de um zoom por seleção realizado pela consulta apresentada na Figura 42.

```
1. SELECT sky_geometry as id, ST_AsText(g_map) as wkt
2. FROM d_geometry
3. WHERE ST_Intersects(ST_GeomFromText('MULTIPOLYGON(((962.8492 -
   2433.8630681419,962.8492 -2875.8934729139,1404.9532 -
   2875.8934729139,1404.9532 -2433.8630681419,962.8492 -
   2433.8630681419)))'), g_map)
```

Figura 42 – Consulta que retorna os polígonos interceptados pelo polígono informado

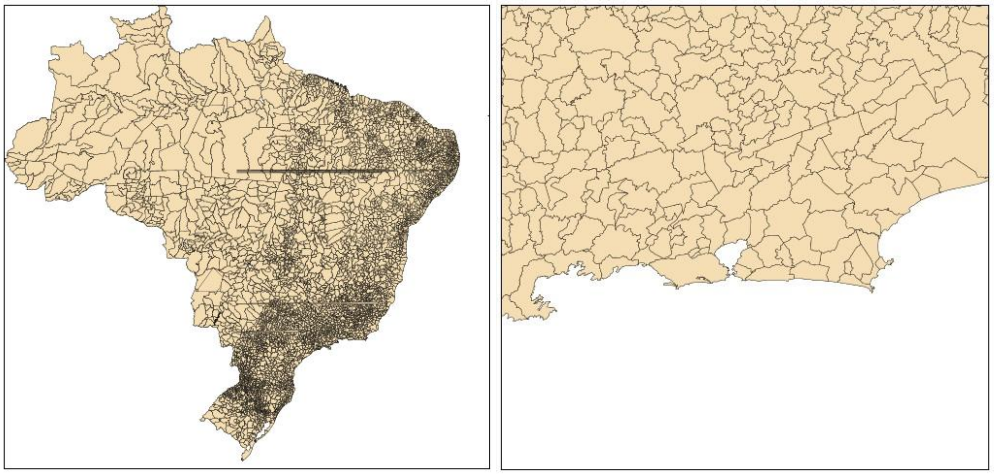

Figura 43 – Dados espaciais antes e após realização do zoom por seleção

5.5 Exemplo de uso da proposta

5.5.1 Verificação de inconsistência em dados espaciais

A visualização de dados espaciais auxilia na verificação de inconsistências nos dados que são difíceis de serem identificados analisando apenas os dados gerados. Por exemplo, a Figura 44 apresenta os municípios do Brasil com um polígono mal formado da região Nordeste. Um dos erros é fácil de visualizar neste mapa, pois são traços que não fazem parte de um município e passam por cima de outros municípios. No entanto, utilizando a ferramenta de zoom por seleção na região Nordeste, chega-se ao município que está com erro, como apresentado na Figura 45.

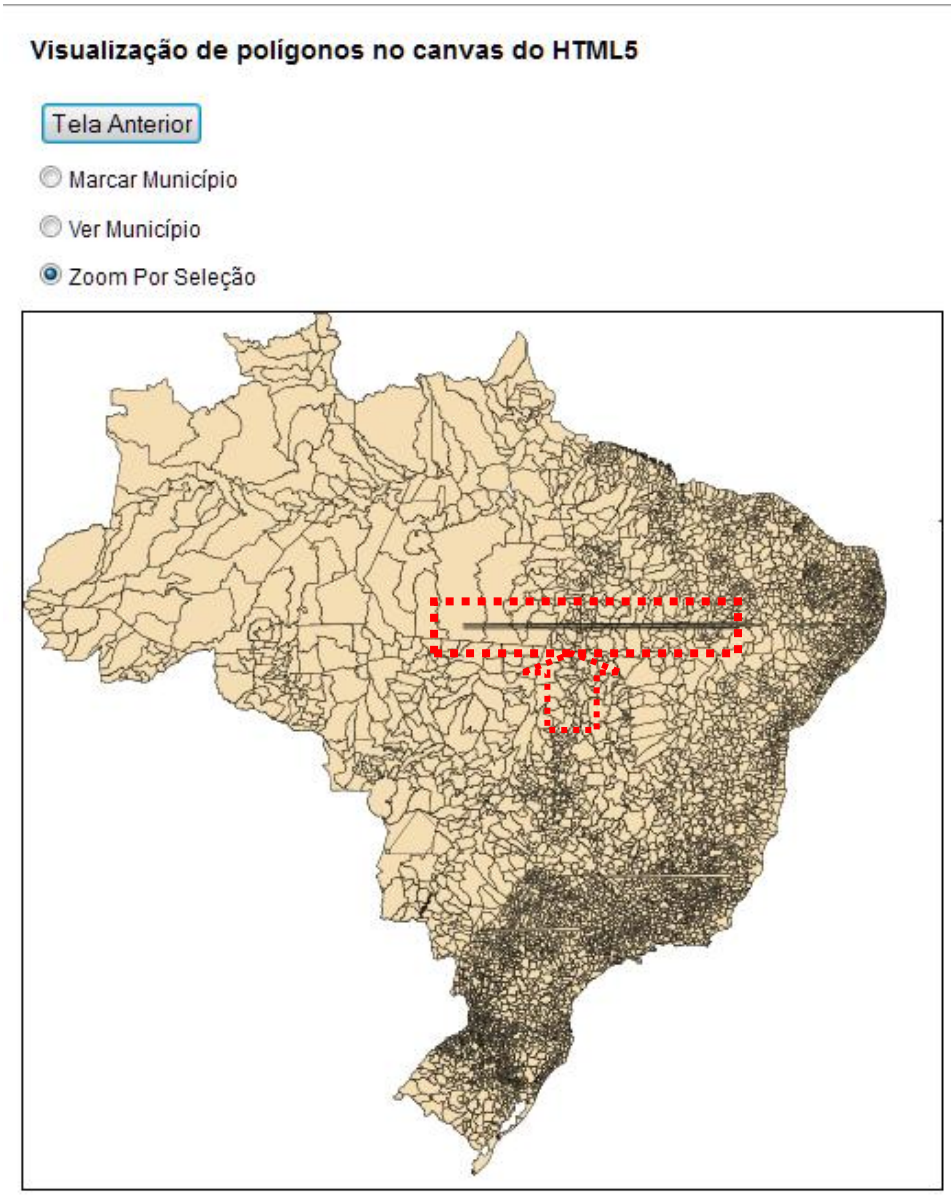

Figura 44 – Dados espaciais com problemas

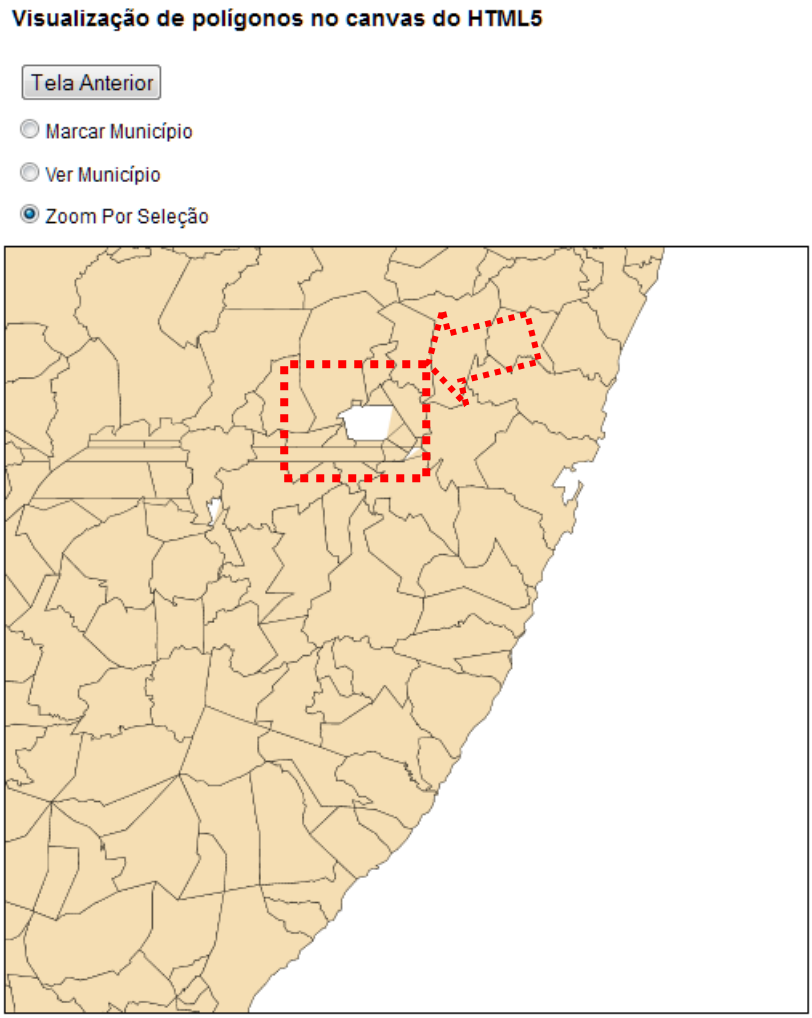

Figura 45 – Polígono com erro

5.5.2 Verificação de assinaturas 4CRS geradas

Outro exemplo importante do uso da proposta é na verificação se uma assinatura foi gerada corretamente. Considere, por exemplo, que se deseja verificar se a assinatura para o objeto apresentado na Figura 46 foi gerada corretamente. Neste caso, realiza-se Zoom na assinatura e verifica-se se o tipo da célula atribuído na assinatura corresponde ao percentual do objeto dentro da célula. A Figura 47 apresenta o zoom na parte superior da assinatura 4CRS. Observe que a célula do topo da assinatura é marcada com cinza claro (tipo pouco) e que realmente há uma interseção de menos que 50% do objeto com a célula. Na realidade, a interseção é mínima e isto pôde ser visualizado facilmente.

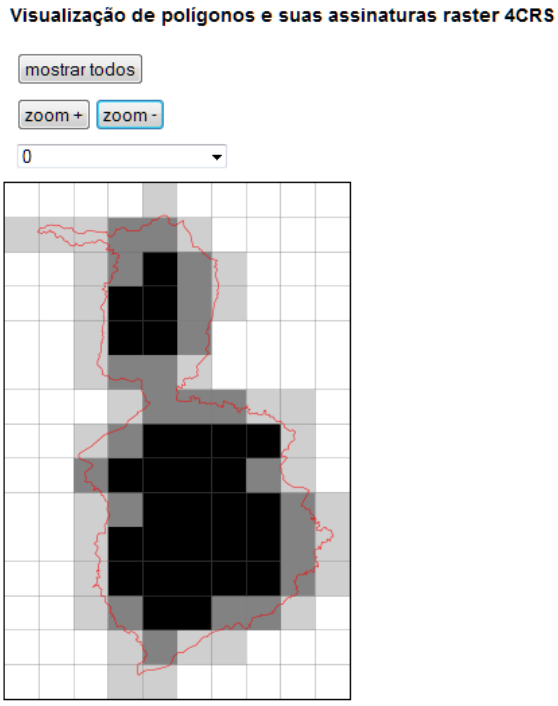

Figura 46 – Visualização individual de uma assinatura 4CRS

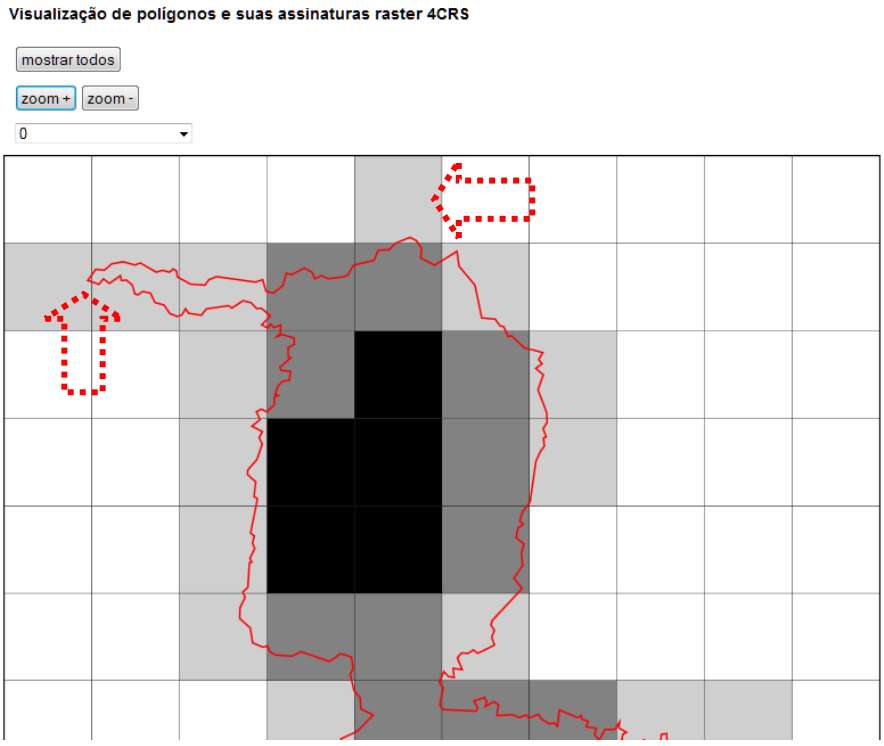

Figura 47 – Zoom da parte superior da assinatura apresentada na Figura 46

A partir do exemplos de uso da proposta apresentados neste capítulo, podemos observar a importância da visualização de dados espaciais em determinados contextos, auxiliando na confirmação da qualidade dos dados armazenados, que antes não eram visualizados. Além disso, as ferramentas de auxilio na visualização como o zoom por seleção se mostraram extremamente úteis para uma visualização mais detalhadas dos dados armazenados.

Capítulo 6: CONCLUSÃO

A análise de dados espaciais é fundamental para um melhor entendimento dos mesmos. Tal análise pode ser realizada por meio de mapas, globos, fotografias e imagens, sendo de suma importância para realização desta análise a visualização dos dados.

Este trabalho apresentou um passo-a-passo para a visualização de dados espaciais na Web, acessando um banco de dados espaciais correspondente a polígonos do Brasil. Para ilustrar o uso da proposta em outros tipos de dados, foi também implementada a visualização de assinaturas 4CRS [Zimbrão e Souza, 1998]. A escolha deste tipo de dado se deu aos bons resultados que indicam o uso desta assinatura.

Dessa forma, a implementação do passo-a-passo proposto confirma que é viável utilizar tecnologias de código fonte aberto para visualizar dados espaciais na Web.

Além disso, comprovamos que, para determinados casos, a visualização de dados espaciais é importante na identificação de inconsistências dados espaciais armazenados, como podemos observar nas seções 5.5 e 5.5.2. Antunes e Azevedo [2011] apontam a necessidade de se ter uma visualização da assinatura raster para depuração e análise de resultados.

Como trabalhos futuros propomos:

- Criação de um sistema de dados georreferenciados referentes ao mapa do mundo, adicionando novas funcionalidades como navegação *drag and drop*, medidor de distância e desenhador de áreas.
- Inclusão de visualização de dados armazenados em arquivos KML.
- Aprimoramento das ferramentas de zoom e seleção.
- Inclusão de novas funcionalidades para visualização: visualizar outros objetos espaciais, como polilinhas e pontos;

- Execução de testes experimentais comparando abordagem com outras tecnologias, em relação a desempenho, escalabilidade e facilidade de uso.

Capítulo 7: REFERÊNCIAS

ACHOR, M., BETZ, F., DOVGAL, A. et al. **PHP Manual.** The PHP Group, 2012. Disponível em <http://www.php.net/manual/en/language.oop5.php> . Acessado em 9 jan. 2012.

ANTUNES, L. C. R., AZEVEDO, L.G. **Computing Polygon Similarity from Raster Signatures.** In: *XII Brazilian Symposium on GeoInformatica* (GeoInfo 2011), Campos do Jordão, Brazil, p. 27-38, 2011.

ARONOFF, S., **Geographic Information Systems**. 1 ed. Ottawa, Canada, WDL Publications, 1989.

AZEVEDO, L. G., MONTEIRO, R. S., ZIMBRÃO, G., SOUZA, J. M.. **Approximate Spatial Query Processing Using Raster Signatures**. In: *VI Brazilian Symposium on GeoInformatica* (GeoInfo 2004), Campos do Jordão, Brazil, p. 403-421, 2004.

AZEVEDO, L. G., ZIMBRÃO, G., SOUZA, J. M., GÜTING, R. H. **Estimating the Overlapping Area of Polígono Join**. In: International Symposium on Advances in Spatial and Temporal Databases, v. 1, Angra dos Reis, Brazil, p. 91-108, 2005.

AZEVEDO, L. G., ZIMBRÃO, G., SOUZA, J. M. **Approximate Query Processing in Spatial Databases Using Raster Signatures**. In: Advances in Geoinformatics. 1 ed., v. 1, Springer-Verlag, Berlin Heidelberg New York , p. 69-85, 2006.

AZEVEDO, L. **Filtros Raster Para Junções de Polilinhas**. Rio de Janeiro, 2001

BIBEAULT, B; KATZ, Y. **JQuery in action**. 2nd ed. Stamford: Manning Publications Co., 2010.

BRINKHOFF, T., KRIEGEL, H. P, SEEGER, B., 1993a. **Efficient processing of Spatial Joins Using R-Trees**. In: Proceedings of the 1993 ACM-SIGMOD Intl. Conference, Washington, DC, USA, Maio, 1993.

CÂMARA, G.; MONTEIRO, A. M.; FUCKS, S. D.; CARVALHO, M. S. Análise espacial e geoprocessamento. In: FUCKS, Suzana Druck; CARVALHO, Marilia Sá; CÂMARA, Gilberto; MONTEIRO, Antonio Miguel Vieira (Ed.). **Análise espacial de dados geográficos.**. São José dos Campos: INPE, 2002. p. 26. Disponível em: <http://urlib.net/sid.inpe.br/sergio/2004/10.07.14.45>. Acessado em 25 jan. 2012.

CROCKFORD, D. **JSON:** The Fat-Free Alternative to XML. Dezembro, 2006. Disponível em <http://www.json.org/fatfree.html>. Acessado em 10 jan. 2012.

CROCKFORD, D. **Introducing JSON.** Maio, 2009. Disponível em <http://www.json.org>. Acessado em 10 jan. 2012.

FLANAGAN, D. **JavaScript**: the definitive guide. 6th ed. Sebastopol: O'Reilly Media Inc., 2011.

FRANCISCHETT, M. N.. **A cartografia no ensino de geografia a apredizagem mediada**. (Tese de Doutorado) Universidade Estadual Paulista Faculdade de Ciências e Tecnologia. 2001. p. 9.

GOOGLE CODE. **KML documentation introduction**. Google, 2011. Disponível em <http://code.google.com/apis/kml/documentation/>. Acessado em 9 jan. 2012.

GÜTING, R. H., **An Introduction to Spatial Database Systems**. The International Journal on Very Large Data Bases, v. 3, n. 4 (Oct), pp. 357-399, 1994.

INDE. **Boletim da INDE** Ano 1, nº 1, Maio de 2011. Ministério do Planejamento, Orçamento e Gestão Comissão Nacional de Cartografia, Infraestrutura Nacional de Dados Espaciais. Disponível em http://www.inde.gov.br/wp-content/themes/boletim_inde_a4.pdf. Acessado em 17 abr. 2012.

INPE. **Plano Diretor 2011-2015**. Ministério da Ciência e Tecnologia, Instituto Nacional de Pesquisas Espaciais. São José dos Campos, Julho de 2011. Disponível em<http://www.inpe.br/noticias/arquivos/pdf/Plano_diretor_miolo.pdf>. Acessado em 17 abr. 2012.

JQUERY. **JQuery.ajax(url [, settings]).** The jQuery Project, 2010. Disponível em <http://api.jQuery.com/jQuery.ajax/#jQuery-ajax-settings>. Acessado em 7 jan. 2012

JQUERY.AJAX(). In: **JQuery**: write less, do more. Disponível em <http://api.jquery.com/jQuery.ajax/#jQuery-ajax-settings>. Acessado em 5 jan. 2012.

KML. **OGC KML.** Open Geospatial Consortium, 2011. Disponível em <http://www.opengeospatial.org/standards/kml/>. Acessado em 9 jan. 2012.

MDN. **Canvas**. Mozilla Developer Network, 2012. Disponível em <https://developer.mozilla.org/en/HTML/Canvas>. Acessado em 9 jan. 2011.

MDN. **Canvas Tutorial**. Mozilla Developer Network, 2012. Disponível em <https://developer.mozilla.org/en/Canvas_tutorial>. Acessado em 13 jan. 2012.

MDN. **AJAX: Asynchronous JavaScript + XML.** Mozilla Developer Network, Novembro, 2011. Disponível em < https://developer.mozilla.org/en/AJAX>. Acessado em 09 jan. 2012.

MURRAY, G.; BALL, J. **Including AJAX Functionality in a Custom JavaServer Faces Component.** Disponível em <http://www.oracle.com/technetwork/java/javaee/tutorial-jsp-140089.html#whatis>. Acessado em 10 jan. 2012.

OpenGIS Consortium, Inc. **OpenGIS® Simple Features Specification For SQL** Revision 1.1. In: *OpenGIS Project Document 99-049*, Maio de 1999.

Open Geospatial Consortium, Inc. **The OGC® Abstract Specification Topic 0: Abstract Specification Overview Version 5**. In: OGC Project Document Number 04-084, Wayland, Massachusetts, 2005.

PILGRIM, M. **HTML5**: up and running. Sebastopol: O'Reilly Media Inc., 2010.

SAMET, H., **The Design and Analysis of Spatial Data Structure**. Addison-Wesley Publishing Company, 1a ed., Boston, Massachusetts, 1990.

SOMMER, S; WADE, T. **A to Z GIS**: An illustrated dictionary of geographic information systems. Redlands: ESRI Press, 2010.

TAO, Y., SUN, J., PAPADIAS, D., **Selectivity estimation for predictive spatio-temporal queries**. In: Proceedings of the 19th International Conference on Data Engineering, pp. 417-428, Bangalore, India, 2003.

W3C. **HTML5: A vocabulary and associated APIs for HTML and XHTML.** World Wide Web Consortium (W3C), Maio, 2011. Disponível em <http://www.w3.org/TR/html5/>. Acessado em 9 jan. de 2012.

W3C. **The Canvas Element.** World Wide Web Consortium (W3C), Maio, 2011. Disponível em http://www.w3.org/TR/html5/the-canvas-element.html#the-canvas-element. Acessado em 12 jan. 2012.

W3SCHOOLS. **AJAX introduction**. Disponível em <http://www.w3schools.com/ajax/ajax_intro.asp>. Acessado em 9 jan. 2012.

ZIMBRAO, G., SOUZA, J. M., 1998, "A Raster Approximation for Processing of Spatial Joins". In: *Proceedings of the 24rd International Conference on Very Large Data Bases*, pp. 558-569, New York City, New York, USA, Aug.

RIBEIRO, GILBERTO. **Metadados Geoespaciais Digitais.** Universidade Federal do Rio de Janeiro. Dezembro, 1996. Disponível em < http://www.geomatica.eng.uerj.br/docentes/gilberto/_media/qualify2_gilberto.pdf> . Acessado em 25 abr. 2012.