Visualizing United State's Traffic Flow

Tharald S. Fongaard & Jacob Perricone

Advisor: Professor Alain Kornhauser

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Operations Research and Financial Engineering

Princeton University

12 May 2015

We hereby declare that we are the sole authors of this thesis.

We authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Jacob Perricone	Tharald S. Fongaard

We further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Jacob Perricone	Tharald S. Fongaard

Abstract

By analyzing trip generation in the United States, we will display key statistics in an informative graphic user interface. Specifically, given a node network of the entire nation and trip generation files, we will provide a means to determine the number of vehicles, people, and the average vehicle occupancy on each link at each time throughout the day.

Acknowledgements

Here are the acknowledgments. I'm sure you know what to do here... thanks, and thanks!

To The Hereditary Kingdom of Norway

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Intorduction not yet written.

1.1 First Section

TBD

Chapter 2

Creating The Pixel Grid

The first step of the visualization required that the United States be broken up into pixels so that trips could be easily identified and tracked. Our analysis considers the 48 contiguous states, which together span around 3.8 million square miles. Each square pixel spans an area of .25 miles, implying that around 12 million pixels are needed to uniquely map the entire country. The center of the National grid is at 37°N Latitude, -97.50°W Longitude, placing the grid's centroid close to the geographic center of the continental US along the time meridian.

2.1 Pixel Structure and Algorithm Method

The primary motivation for creating a pixelated grid is to establish a simple method to track, store and retrieve trips within the pixel. Thus each pixel object must contain a unique id, an (i,j) coordinate, and the latitude and longitude of its four corners and its centroid.

2.1.1 Geometric Analysis

In order to algorithmically create a pixelated grid, a methodology of mapping latitude and longitude to (i,j) coordinates was established. Given that along a Meridian each degree of latitude traverses approximately 69.174 miles and that along a Parallel each degree of longitude traverses approximately $69.174 \cdot \cos(Latitude)$ one can map any latitude and longitude coordinate to an (i,j) pixel. To determine the set of (lon, lat) coordinates within a pixel, the angular height and width of each pixel as calculated. The Y-height of a pixel is 0.00722814 degrees Latitude and the X-width of a pixel is 0.00944344 degrees Latitude. Each (i,j) Pixel thus includes all points within:

$$YHeight = 0.00722814 \mid CenterLat = 97.5 \mid CenterLon = 37.0 \tag{2.1}$$

$$\frac{CenterLat + YHeight \cdot i}{\cos(CenterLon + YHeight \cdot j)} \leq Longitude < \frac{CenterLat + YHeight \cdot (i+1)}{\cos(CenterLon + YHeight \cdot (j+1)} \tag{2.2}$$

$$CenterLat + YHeight \cdot j \leq Latitude < CenterLat + YHeight \cdot (j+1) \tag{2.3}$$

The inverse of the equation above was used to map (lon, lat) coordinates to pixel coordinates and was imperative in constraining our algorithm to the US.

$$xPixel = i = floor(138.348 \cdot .sllongitude + CenterLat \cdot \cos(latitude)) \tag{2.4}$$

$$yPixel = j = floor(138.348 \cdot .sllatitude - CenterLon) \tag{2.5}$$

Where floor casts the decimal result as an integer, rounded down, and 138.348 is a correcting constant.

2.1.2 Methodoloy

The southwest and northeast corner of the US provided the initial and final (lon, lat) coordinates of our grid. The initial point (southwest corner) has a longitude of -120.5°and a latitude of -63.5°. The final point (northeast corner) has a longitude of -63.5°and a latitude of 48.9°. Using equation 2.4 and 2.5, the initial and final (i,j) coordinates corresponding to the corners of the nation were found and used as starting and breaking conditions during pixel construction. Then, running through a double for loop indexed by i and j, each (i,j) pair was sent to a Pixel class for creation and then added to a hashtable. The Pixel class takes an (i,j) coordinate in its constructor and using the equations in 2.1.1 determines the longitude and latitude of its four corners and centroid. Furthermore, in order to prevent collisions within the hashtable, each pixel creates a unique id by using its (i,j) coordinates as the input for the Cantor pairing function, which uniquely encodes two natural numbers into a single natural number. Since the Cantor pairing function only holds for positive numbers, i and j are offset by the absolute value of the initial (i,j) point.

$$f_{ij} = (1/2)\cdot(i+|(InitialX+InitialY)|+j)\cdot(i+|(InitialX+InitialY)|+j+1)+j+|(InitialY)|$$

(Cantor Function)

A sketch of the initialization loop is shown below.

```
for i in range(iStart, iEnd):
   for j in range(jStart, jEnd):
            Pixel = createPixel(i,j)
            Pixels.add(Pixel.key(), Pixel)
```

2.2 Uploading Data to the SQL Database

Every pixel is added to a SQL database (mySQL). The database is located on a rackspace cloud platform and is divided into several sections. The pixels are added to the "Grid" databse, which has a chema for each state as well as a schema to store pixels that fall outside of the borders of the united states. In the program, each state is stored as a polygon object in a kdtree mapped to its centroid. When a new pixel is created, polygons are returned in order of proximity to the state centroids. Each polygon is then queried until the polygin which contains the pixel centroid is found. This polygon is then queried for its state name, which coresponds to the schema of which the pixel is uploaded to. The pixels are added with a runtime of about 80 pixels per second per thread, which allows us to create about 640 pixels per second with multithreading on two quad core computeres with sufficient heap space. The total creation time for a grid of about 18 million pixels is then about 8hrs. The table structure is shown in the table 2.1. The table is updated with a java database connector (JDBC) package.

Table 2.1: SQL Table Structure

ID	X	Y	C_Long	C_Lat	BL_Long	BL_Lat	BR_Long	BR_Lat	TL_Long	TL_Lat	TR_Long	TR_Lat
20172271942	2694	848	-70,81	43,13	-70,82	43,13	-70,81	43,13	-70,82	43,14	-70,81	43,14
20172271943	2693	849	-70,82	43,14	-70,83	43,14	-70,82	43,14	-70,82	43,14	-70,81	43,14
20172472799	2698	845	-70,78	43,11	-70,79	43,11	-70,78	43,11	-70,78	43,12	-70,78	43,12
20172472800	2697	846	-70,79	43,12	-70,79	43,12	-70,78	43,12	-70,79	43,12	-70,78	43,12
20172472801	2696	847	-70,80	43,13	-70,80	43,12	-70,79	43,12	-70,80	43,13	-70,79	43,13
20172472802	2695	848	-70,80	43,13	-70,81	43,13	-70,80	43,13	-70,81	43,14	-70,80	43,14

Table 2.1 is an extract from Grid.Maine on our rackspace account.

2.3 Visualizing the Grid

The grid can be visualized through our Open Street Map Java Interface. In figure **??** the grid is downloaded from the server and displayed as a layer on top of the map.

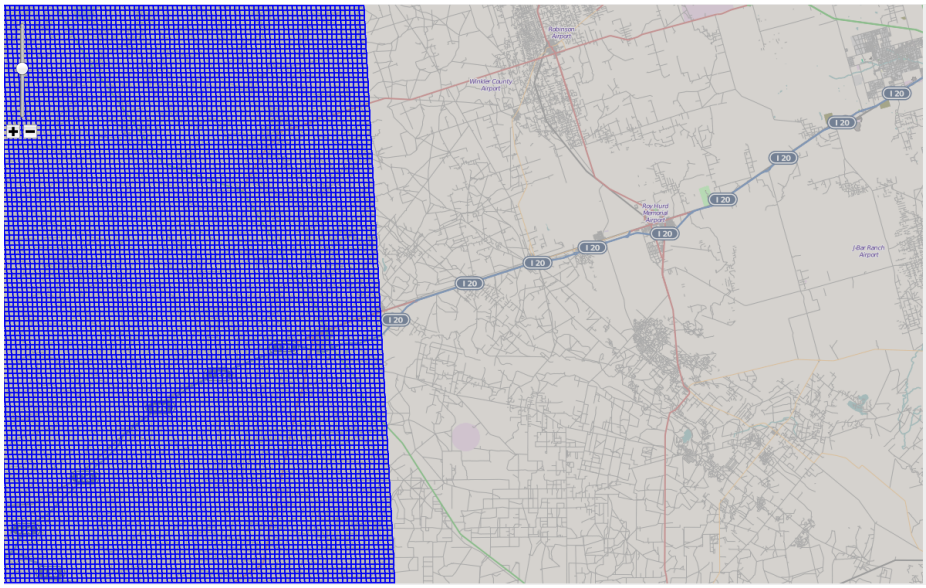

Figure 2.1: The grid displayed on our OSM interface in TExas

When zoomed, the properties of the grid is displayed. The pixels are increasingly non-square as one moves away from the (0,0) pixel.

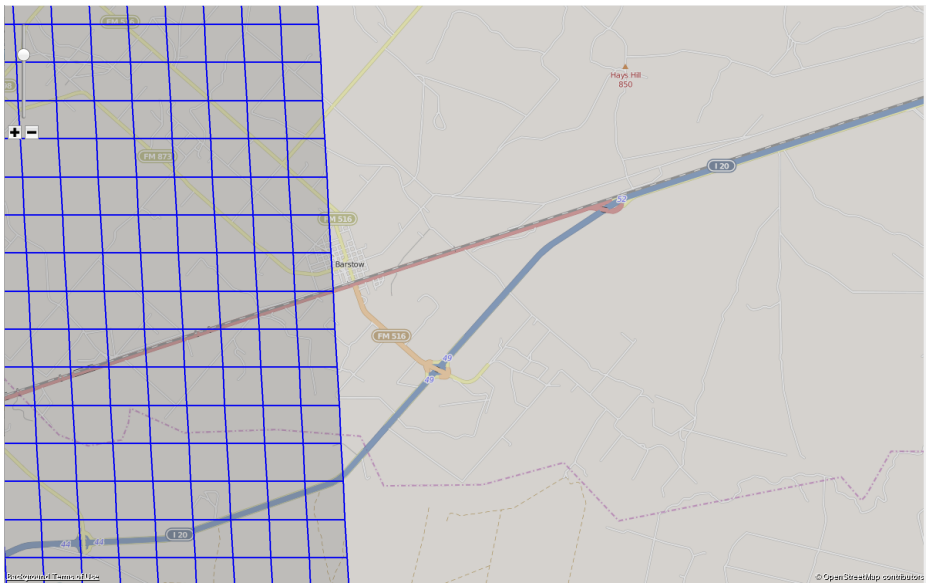

Figure 2.2: The grid displayed on our OSM interface in Texas zoomed in

Appendix A

Code

A.1 Java

A.1.1 CreateGrid.java

```
import java.awt.*;


import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;


import java.awt.geom.*;
import java.util.*;
import java.io.*;


/**
 * Created by Tharald Fongaard and Jacob Perricone on 10/02/15.
 */
```

```
public class CreateGrid {
    private SeparateChainingHashST<String,SeparateChainingHashST<Double,Pixelizati
    private XMLReader xmlReader;
    private SeparateChainingHashST<Double,String> keyState;

    public CreateGrid() throws ClassNotFoundException, SQLException {
        xmlReader = new XMLReader();
        pixels = new SeparateChainingHashST<String, SeparateChainingHashST<Double,
        keyState = new SeparateChainingHashST<Double, String>();

        for(String state: xmlReader.getAllStates()){
            SeparateChainingHashST<Double,Pixelization> stateTable = new SeparateC
            pixels.put(state,stateTable);
        }

        int iStart;
        int iEnd;
        int jStart;
        int jEnd;

        double initiallat = 24.544091;
        double initallon = -120.5;
        double finallon = -106.25;
        double finallat = 48.987386;

        iStart = (int) Math.floor(138.348*(initallon + 97.5)*Math.cos(Math.toRadia
        jStart = (int) Math.floor(138.348*(initiallat - 37.0));
```

```
iEnd = (int) Math.floor(138.348*(finallon + 97.5)*Math.cos(Math.toRadians(
jEnd = (int) Math.floor(138.348*(finallat - 37.0));

String url = "jdbc:mysql://127.0.0.1:3306/Grid";

for(int i = iEnd; i >= iStart; i--){
    Class.forName("com.mysql.jdbc.Driver");
    Connection m_Connection = DriverManager.getConnection(url, "tharald",
    StdOut.println("" + i + " / " + iEnd);
    for(int j = jStart; j <= jEnd; j++){
        Pixelization pixel = new Pixelization(i,j);
        double key = pixel.get_id();
        Point2D cent = pixel.get_centroid();
        String state = xmlReader.getClosestState(cent);

        Point2D bl = pixel.get_bl();
        Point2D br = pixel.get_br();
        Point2D tr = pixel.get_tr();
        Point2D tl = pixel.get_tl();
        int x = i;
        int y = j;

        PreparedStatement total = m_Connection.prepareStatement("REPLACE I

        //ID
        total.setDouble(1, key);
```

```
    //X/Y
    total.setInt(2, x);
    total.setInt(3, y);

    //Centroid Long/Lat
    total.setDouble(4, cent.x());
    total.setDouble(5, cent.y());

    //Bottom Left Long/Lat
    total.setDouble(6, bl.x());
    total.setDouble(7, bl.y());

    //Bottom Right Long/Lat
    total.setDouble(8, br.x());
    total.setDouble(9, br.y());

    //Top Left Long/Lat
    total.setDouble(10, tl.x());
    total.setDouble(11, tl.y());

    //Top Right Long/Lat
    total.setDouble(12, tr.x());
    total.setDouble(13, tr.y());

    total.executeUpdate();
}
m_Connection.close();
```

```
    }


}

public Iterable<String> getStates(){
    return pixels.keys();
}



public double Size(){
    return pixels.size();
}

public Iterable<Double> getKeys(String state){
    return pixels.get(state).keys();
}

public int getX(double key){
    String state = keyState.get(key);
    Pixelization pixel = pixels.get(state).get(key);
    return pixel.get_x();
}

public int getY(double key){
    String state = keyState.get(key);
```

```
    Pixelization pixel = pixels.get(state).get(key);
    return pixel.get_y();
}

public Point2D getPixelCentroid(double key){
    String state = keyState.get(key);
    Pixelization pixel = pixels.get(state).get(key);
    Point2D returnPoint = pixel.get_centroid();
    return returnPoint;
}

public Point2D getPixelBr(double key){
    String state = keyState.get(key);
    Pixelization pixel = pixels.get(state).get(key);
    Point2D returnPoint = pixel.get_br();
    return returnPoint;
}

public Point2D getPixelBl(double key){
    String state = keyState.get(key);
    Pixelization pixel = pixels.get(state).get(key);
    Point2D returnPoint = pixel.get_bl();
    return returnPoint;
}

public Point2D getPixelTr(double key){
    String state = keyState.get(key);
```

```
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_tr();
        return returnPoint;
    }

    public Point2D getPixelTl(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_tl();
        return returnPoint;

    }

    public static void main(String[] args) {
        CreateGrid grid = null;
        try {
            grid = new CreateGrid();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        StdOut.println("Done");
    }


}
```

A.1.2 Pixelization.java

```
import java.util.*;

import java.io.*;

import java.awt.*;

import java.lang.Math;


public class Pixelization {

double id;

int x;

int y;

double CENTERLON = -97.5;

double CENTERLAT = 37;

double YHEIGHT = 0.00722814;

double XWIDTH =  0.00944344;

Point2D bottomleft;

Point2D bottomright;

Point2D topleft;

Point2D topright;

Point2D centroid;


public Pixelization(int x, int y) {

this.x = x;

this.y = y;

this.id = createid(x, y);

this.bottomleft = createbottomleft(x, y);

this.bottomright = createbottomright(x,y);

this.topleft = createtopleft(x,y);
```

```
this.topright = createtopright(x,y);
this.centroid = createcentroid();
}
private double createid(int a, int b) {
        double returnn = 0.5*(a+195593+b+1724)*(a+195593+b+1724+1)+b+1724;
       // StdOut.println(returnn);
        return returnn;
    }

private Point2D createbottomleft(int i, int j) {
double lon, lat;
double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
lon = (CENTERLON + (YHEIGHT*i)/cosangle);
lat = (CENTERLAT + YHEIGHT*j);
Point2D p = new Point2D(lon, lat);
return p;
}
private Point2D createbottomright(int i, int j) {
i++;
double lon, lat;
double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
lon = (CENTERLON + (YHEIGHT*i)/cosangle);
lat = (CENTERLAT + YHEIGHT*j);
Point2D p = new Point2D(lon, lat);
return p;
}
```

```
private Point2D createtopleft(int i, int j) {
j++;
double lon, lat;
double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
lon = (CENTERLON + (YHEIGHT*i)/cosangle);
lat = (CENTERLAT + YHEIGHT*j);
Point2D p = new Point2D(lon, lat);
return p;
}

private Point2D createtopright(int i, int j) {
j++;
i++;
double lon, lat;
double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
lon = (CENTERLON + (YHEIGHT*i)/cosangle);
lat = (CENTERLAT + YHEIGHT*j);
Point2D p = new Point2D(lon, lat);
return p;
}

private Point2D createcentroid() {
double lon, lat;
lon = (bottomleft.x() + bottomright.x())/2;
lat = (bottomleft.y() + topleft.y())/2;
Point2D p = new Point2D(lon, lat);
return p;
```

```
}

public Point2D get_br() {
return bottomright;
}

public int get_x() {
return x;
}

public int get_y() {
return y;
}

public double get_id() {
return id;
}

public Point2D get_bl() {
return bottomleft;
}

public Point2D get_tl() {
return topleft;
}

public Point2D get_tr() {
```

```
return topright;
}

    public Point2D get_centroid(){
        return centroid;
    }

    public static void main(String[] args) {
     double lat = Double.parseDouble(args[0]);
     double lon = Double.parseDouble(args[1]);

     System.out.println("Starting lat: " + lat + " , Starting lon: " + lon);
     Point2D p = new Point2D(0.9,1.9);
     Pixelization pixel = new Pixelization(lat, lon, p);
     System.out.println("Lat : " + pixel.get_brLat() + "Long + " + pixel.get_brLon

    }
}
```

A.1.3 Polygon.java

```
import java.awt.*;

/*************************************************************************
 *  Compilation:  javac Polygon.java
 *  Execution:    java Polygon
 *  Dependencies: Point2D.java
 *
```

```
 *  Implementation of 2D polygon, possibly intersecting.
 *
 *
 *  Original Obtained from Princeton University algs4 library
 *  This version has been altered from the original
 ************************************************************************/

public class Polygon {
    private int N;        // number of points in the polygon
    private Point2D[] a;    // the points, setting points[0] = points[N]
    private String name;
    private Color color;

    // default buffer = 4
    public Polygon(String state, String colour) {
        N = 0;
        a = new Point2D[4];
        name = state;
        color = Color.decode(colour);

    }


    // double size of array
    private void resize() {
        Point2D[] temp = new Point2D[2*N+1];
        for (int i = 0; i <= N; i++) temp[i] = a[i];
```

```
    a = temp;
}

// return size
public int size() { return N; }

// draw polygon
public void draw() {
    for (int i = 0; i < N; i++)
        a[i].drawTo(a[i+1]);
}



// add point p to end of polygon
public void add(Point2D p) {
    if (N >= a.length - 1) resize();   // resize array if needed
    a[N++] = p;                        // add point
    a[N] = a[0];                       // close polygon
}

// return the perimeter
public double perimeter() {
    double sum = 0.0;
    for (int i = 0; i < N; i++)
        sum = sum + a[i].distanceTo(a[i+1]);
    return sum;
}
```

```
public Iterable<Point2D> getAll(){
    Stack<Point2D> stack = new Stack<Point2D>();
    for (int i = 0; i < N; i++){
        stack.push(a[i]);
    }
    return stack;

}

// return signed area of polygon
public double area() {
    double sum = 0.0;
    for (int i = 0; i < N; i++) {
        sum = sum + (a[i].x() * a[i+1].y()) - (a[i].y() * a[i+1].x());
    }
    return 0.5 * sum;
}

// does this Polygon contain the point p?
// if p is on boundary then 0 or 1 is returned, and p is in exactly one point
// Reference: http://exaflop.org/docs/cgafaq/cga2.html
public boolean contains2(Point2D p) {
    int crossings = 0;
    for (int i = 0; i < N; i++) {
        int j = i + 1;
        boolean cond1 = (a[i].y() <= p.y()) && (p.y() < a[j].y());
```

```
            boolean cond2 = (a[j].y() <= p.y()) && (p.y() < a[i].y());
            if (cond1 || cond2) {
                // need to cast to double
                if (p.x() < (a[j].x() - a[i].x()) * (p.y() - a[i].y()) / (a[j].y()
                    crossings++;
            }
        }
        if (crossings % 2 == 1) return true;
        else                    return false;
    }

    // does this Polygon contain the point p?
    // Reference: http://softsurfer.com/Archive/algorithm_0103/algorithm_0103.htm
    public boolean contains(Point2D p) {
        int winding = 0;
        for (int i = 0; i < N; i++) {
            int ccw = Point2D.ccw(a[i], a[i+1], p);
            if (a[i+1].y() >  p.y() && p.y() >= a[i].y())  // upward crossing
                if (ccw == +1) winding++;
            if (a[i+1].y() <= p.y() && p.y() <  a[i].y())  // downward crossing
                if (ccw == -1) winding--;
        }
        return winding != 0;
    }


    // return string representation of this point
```

```
public String toString() {
    if (N == 0) return "[ ]";
    String s = "";
    s = s + "[ ";
    for (int i = 0; i <= N; i++)
        s = s + a[i] + " ";
    s = s + "]";
    return s;
}

public String getName(){
    return name;
}

public Point2D centroid(){
    double cX = 0;
    double cY = 0;
    for (int i = 0; i <= N; i++){
        cX = cX + a[i].x();
        cY = cY + a[i].y();
    }
    cX = cX/N;
    cY = cY/N;
    Point2D rPoint =new Point2D(cX,cY);
    return rPoint;
}
```

```
public Color color(){
    return color;
}


// test client
public static void main(String[] args) {
    int N = 10;

    // a square
    Polygon poly = new Polygon("STATE", "#ff0000");
    poly.add(new Point2D(5, 5));
    poly.add(new Point2D(9, 5));
    poly.add(new Point2D(9, 9));
    poly.add(new Point2D(5, 9));

    System.out.println("polygon    = " + poly);
    System.out.println("perimeter  = " + poly.perimeter());
    System.out.println("area       = " + poly.area());

    System.out.println("contains(5, 5) = " + poly.contains(new Point2D(5, 5)))
    System.out.println("contains(9, 5) = " + poly.contains(new Point2D(9, 5)))
    System.out.println("contains(9, 9) = " + poly.contains(new Point2D(9, 9)))
    System.out.println("contains(5, 9) = " + poly.contains(new Point2D(5, 9)))
    System.out.println("contains(7, 5) = " + poly.contains(new Point2D(7, 5)))
    System.out.println("contains(5, 7) = " + poly.contains(new Point2D(5, 7)))
    System.out.println("contains(7, 9) = " + poly.contains(new Point2D(7, 9)))
```

```
        System.out.println("contains(9, 7) = " + poly.contains(new Point2D(9, 7)))

        // generate N random points in the unit square and check what fraction are
        int yes = 0;
        for (int i = 0; i < N; i++) {
            int x = (int) (10 * Math.random());
            int y = (int) (10 * Math.random());
            Point2D p = new Point2D(x, y);
            if (poly.contains(p)) yes++;
            if (poly.contains(p) != poly.contains2(p)) System.out.println("differe
        }

        // true answer is = 0.16 (depends on how boundary points are handled)
        System.out.println("Fraction in polygon = " + 1.0 * yes / N);


    }
}
```

A.1.4 XMLreader.java

```
import jdk.internal.org.xml.sax.SAXException;
import org.w3c.dom.Document;
import org.w3c.dom.*;



import javax.xml.parsers.DocumentBuilder;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.awt.*;
import java.io.IOException;



/**
 * Created by Tharald on 17/02/15.
 */
public class XMLReader {
    private KdTreeST<Polygon> treeST;
    private SeparateChainingHashST<String,Polygon> states;



    public XMLReader(){
        treeST = new KdTreeST<Polygon>();
        states = new SeparateChainingHashST<String, Polygon>();

        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance

        try{
            DocumentBuilder dBuilder = builderFactory.newDocumentBuilder();
            Document document = dBuilder.parse(XMLReader.class.getResourceAsStream
            Document centroidsDocument = dBuilder.parse(XMLReader.class.getResourc
            document.normalize();
            centroidsDocument.normalize();
```

```
NodeList centroids = centroidsDocument.getElementsByTagName("states");
Node centroidRoot = centroids.item(0);
Element centroidElementRoot = (Element) centroidRoot;

NodeList rootNodes = document.getElementsByTagName("states");
org.w3c.dom.Node rootNode = rootNodes.item(0);
org.w3c.dom.Element rootElement = (org.w3c.dom.Element) rootNode;

NodeList statesList = rootElement.getElementsByTagName("state");
NodeList centroidList = centroidElementRoot.getElementsByTagName("stat

for(int i = 0; i < statesList.getLength();i++){
    org.w3c.dom.Node theState = statesList.item(i);
    org.w3c.dom.Element stateElement = (org.w3c.dom.Element) theState;
    //StdOut.println("This state is " + stateElement.getAttribute("nam
    Polygon newState = new Polygon(stateElement.getAttribute("name"),

    NodeList pointList = stateElement.getElementsByTagName("point");
    for(int j = 0; j < pointList.getLength();j++){
        org.w3c.dom.Element pointElement = (org.w3c.dom.Element) point
        Point2D newPoint = new Point2D(Double.parseDouble(pointElement
        newState.add(newPoint);
        //StdOut.println("Lat: " + pointElement.getAttribute("lat") +
    }
    Node statecentroid = centroidList.item(i);
    Element centroidElement = (Element) statecentroid;
    Point2D centroid = new Point2D(Double.parseDouble(centroidElement.
```

```
            treeST.insert(centroid,newState);
            states.put(stateElement.getAttribute("name"),newState);

        }
        Polygon nostate = new Polygon("NoState","#ff0000");
        states.put("NoState", nostate);


    }
    catch (ParserConfigurationException e){
        e.printStackTrace();
    }
    catch (org.xml.sax.SAXException e){
        e.printStackTrace();
    }
    catch (IOException e){
        e.printStackTrace();
    }
}

public Polygon getStatePolygon(String state){
    return states.get(state);
}

public Iterable<Polygon> getAllPolygons(){
    return treeST.getAllValues();
}
```

```
public String getClosestState(Point2D queryPoint){
    Point2D nearestPoint = treeST.nearest(queryPoint);
    Polygon nearestState = treeST.get(nearestPoint);
    if(nearestState.contains(queryPoint)){
        return nearestState.getName();
    }
    else{
        for(Polygon pol:treeST.getAllValues()){
            if(pol.contains(queryPoint)){
                return pol.getName();
            }
        }
    }
    return "NoState";
}

public int size(){
    return treeST.size();
}

public Iterable<String> getAllStates() {
    return states.keys();

}

public Iterable<Point2D> getAllPoints(){
```

```
    return treeST.getAllPoints();
}
public void drawTree(){
    treeST.draw();
}



public static void main(String[] args){
   XMLReader xmlReader = new XMLReader();
    StdOut.println(xmlReader.size());
    StdDraw.setXscale(-136,-66);
    StdDraw.setYscale(20.54,53);
    /*for(String s: xmlReader.getAllStates()){
        StdOut.println(s);
    }

    StdDraw.setPenColor(Color.GREEN);
    int points= 0;
    for(Point2D p:xmlReader.getAllPoints()){
        p.draw();
        points++;
    }
    */
    for(Polygon p:xmlReader.getAllPolygons()){
        StdDraw.setPenColor(p.color());
        p.draw();
```

```
        }
        //StdOut.println(points);
        //xmlReader.drawTree();
        StdDraw.setPenColor(Color.BLACK);
        StdDraw.setPenRadius(.006);

        for(Point2D p:xmlReader.getAllPoints()){
            p.draw();
        }

    }
}
```

A.1.5 TestFrame.java

```
/**
 * Created by Tharald on 22/02/15.
 */

import org.openstreetmap.gui.jmapviewer.*;
import org.openstreetmap.gui.jmapviewer.events.JMVCommandEvent;
import org.openstreetmap.gui.jmapviewer.interfaces.*;
import org.openstreetmap.gui.jmapviewer.tilesources.BingAerialTileSource;
import org.openstreetmap.gui.jmapviewer.tilesources.MapQuestOpenAerialTileSource;
import org.openstreetmap.gui.jmapviewer.tilesources.MapQuestOsmTileSource;
import org.openstreetmap.gui.jmapviewer.tilesources.OsmTileSource;

import java.awt.*;
```

```
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.IOException;
import java.sql.*;
import java.util.*;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class TestFrame extends JFrame implements JMapViewerEventListener {

    private static final long serialVersionUID = 1L;

    private JMapViewerTree treeMap = null;

    private JLabel zoomLabel=null;
    private JLabel zoomValue=null;
```

```
private JLabel mperpLabelName=null;
private JLabel mperpLabelValue = null;

public TestFrame() {

    super("TrafficFlow");
    XMLReader xmlreader = new XMLReader();
    setSize(400, 400);

    treeMap = new JMapViewerTree("Zones");

    // Listen to the map viewer for user operations so components will
    // recieve events and update
    map().addJMVListener(this);

    // final JMapViewer map = new JMapViewer(new MemoryTileCache(),4);
    // map.setTileLoader(new OsmFileCacheTileLoader(map));
    // new DefaultMapController(map);

    setLayout(new BorderLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setExtendedState(JFrame.MAXIMIZED_BOTH);
    JPanel panel = new JPanel();
    JPanel panelTop = new JPanel();
    JPanel panelBottom = new JPanel();
    JPanel helpPanel = new JPanel();
```

```
JPanel sqlPanel = new JPanel();




mperpLabelName=new JLabel("Meters/Pixels: ");
mperpLabelValue=new JLabel(String.format("%s",map().getMeterPerPixel()));

zoomLabel=new JLabel("Zoom: ");
zoomValue=new JLabel(String.format("%s", map().getZoom()));

add(panel, BorderLayout.NORTH);
add(helpPanel, BorderLayout.SOUTH);
panel.setLayout(new BorderLayout());
panel.add(panelTop, BorderLayout.NORTH);
panel.add(panelBottom, BorderLayout.SOUTH);
add(sqlPanel,BorderLayout.EAST);
JLabel helpLabel = new JLabel("Use right mouse button to move,\n "
        + "left double click or mouse wheel to zoom.");
helpPanel.add(helpLabel);
JButton button = new JButton("setDisplayToFitMapMarkers");
button.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        map().setDisplayToFitMapMarkers();
    }
});
```

```
JComboBox<TileSource> tileSourceSelector = new JComboBox<>(new TileSource[
        new OsmTileSource.CycleMap(), new BingAerialTileSource(), new MapQ
tileSourceSelector.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        map().setTileSource((TileSource) e.getItem());
    }
});
JComboBox<TileLoader> tileLoaderSelector;
try {
    tileLoaderSelector = new JComboBox<>(new TileLoader[] { new OsmFileCac
} catch (IOException e) {
    tileLoaderSelector = new JComboBox<>(new TileLoader[] { new OsmTileLoa
}
tileLoaderSelector.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        map().setTileLoader((TileLoader) e.getItem());
    }
});
map().setTileLoader((TileLoader) tileLoaderSelector.getSelectedItem());
panelTop.add(tileSourceSelector);
panelTop.add(tileLoaderSelector);
final JCheckBox showMapMarker = new JCheckBox("Map markers visible");
showMapMarker.setSelected(map().getMapMarkersVisible());
showMapMarker.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        map().setMapMarkerVisible(showMapMarker.isSelected());
    }
```

```
});
panelBottom.add(showMapMarker);
///
final JCheckBox showTreeLayers = new JCheckBox("Tree Layers visible");
showTreeLayers.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        treeMap.setTreeVisible(showTreeLayers.isSelected());
    }
});
panelBottom.add(showTreeLayers);
///
final JCheckBox showToolTip = new JCheckBox("ToolTip visible");
showToolTip.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        map().setToolTipText(null);
    }
});
panelBottom.add(showToolTip);
///
final JCheckBox showTileGrid = new JCheckBox("Tile grid visible");
showTileGrid.setSelected(map().isTileGridVisible());
showTileGrid.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        map().setTileGridVisible(showTileGrid.isSelected());
    }
});
panelBottom.add(showTileGrid);
```

```
final JCheckBox showZoomControls = new JCheckBox("Show zoom controls");
showZoomControls.setSelected(map().getZoomContolsVisible());
showZoomControls.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        map().setZoomContolsVisible(showZoomControls.isSelected());
    }
});
panelBottom.add(showZoomControls);
final JCheckBox scrollWrapEnabled = new JCheckBox("Scrollwrap enabled");
scrollWrapEnabled.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        map().setScrollWrapEnabled(scrollWrapEnabled.isSelected());
    }
});
panelBottom.add(scrollWrapEnabled);
panelBottom.add(button);

panelTop.add(zoomLabel);
panelTop.add(zoomValue);
panelTop.add(mperpLabelName);
panelTop.add(mperpLabelValue);

add(treeMap, BorderLayout.CENTER);

//

LayerGroup pixelGroup = new LayerGroup("States");
```

```
pixelGroup.setVisible(true);
SeparateChainingHashST<String, Layer>  states = new SeparateChainingHashST



String[] options = new String[51];
int i = 0;
for(String s: xmlreader.getAllStates()) {
    Layer layer = pixelGroup.addLayer(s);
   states.put(s,layer);
   Polygon pol = xmlreader.getStatePolygon(s);


    List<Coordinate> list = new ArrayList<Coordinate>();
    for(Point2D p: pol.getAll()){
        list.add(c(p));
    }



    MapPolygon thisState = new MapPolygonImpl(layer,s,list);
    map().addMapPolygon(thisState);
    treeMap.addLayer(layer);
    options[i] = s;
    i++;



}
JComboBox jComboBox = new JComboBox(options);
```

```
sqlPanel.add(jComboBox);

 JButton getPixels = new JButton();
 getPixels.addActionListener(new ActionListener() {

     public void actionPerformed(ActionEvent e) {
         try {
             Class.forName("com.mysql.jdbc.Driver");
         } catch (ClassNotFoundException e1) {
             e1.printStackTrace();
         }
         String url = "jdbc:mysql://127.0.0.1:3306/Grid";
         try {
             Connection m_Connection = DriverManager.getConnection(url, "th
             PreparedStatement total = m_Connection.prepareStatement("SELEC
             ResultSet resultSet = total.executeQuery();
            while(resultSet.next()){
                double bl_long = resultSet.getDouble(6);
                double bl_lat = resultSet.getDouble(7);
                double br_long = resultSet.getDouble(8);
                double br_lat = resultSet.getDouble(9);
                double tl_long = resultSet.getDouble(10);
                double tl_lat = resultSet.getDouble(11);
                double tr_long = resultSet.getDouble(12);
                double tr_lat = resultSet.getDouble(13);
                String state = (String) jComboBox.getSelectedItem();
                List<Coordinate> points = new ArrayList<Coordinate>();
```

```
                points.add(c(bl_lat, bl_long));
                points.add(c(br_lat,br_long));
                points.add(c(tr_lat,tr_long));
                points.add(c(tl_lat,tl_long));


                MapPolygonImpl  pix =  new MapPolygonImpl(states.get(state)
                map().addMapPolygon(pix);

            }



        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        //StdOut.println("Connection Successful");


    }
});

sqlPanel.add(getPixels);
```

```
        // map.setDisplayPosition(new Coordinate(49.807, 8.6), 11);
        // map.setTileGridVisible(true);

        map().addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getButton() == MouseEvent.BUTTON1) {
                    map().getAttribution().handleAttribution(e.getPoint(), true);
                }
            }
        });

        map().addMouseMotionListener(new MouseAdapter() {
            @Override
            public void mouseMoved(MouseEvent e) {
                Point p = e.getPoint();
                boolean cursorHand = map().getAttribution().handleAttributionCurso
                if (cursorHand) {
                    map().setCursor(new Cursor(Cursor.HAND_CURSOR));
                } else {
                    map().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
                }
                if(showToolTip.isSelected()) map().setToolTipText(map().getPositio
            }
        });
    }
```

```
@Override
public void processCommand(JMVCommandEvent command) {
    if (command.getCommand().equals(JMVCommandEvent.COMMAND.ZOOM) ||
            command.getCommand().equals(JMVCommandEvent.COMMAND.MOVE)) {
        updateZoomParameters();
    }
}

private void updateZoomParameters() {
    if (mperpLabelValue!=null)
        mperpLabelValue.setText(String.format("%s",map().getMeterPerPixel()));
    if (zoomValue!=null)
        zoomValue.setText(String.format("%s", map().getZoom()));
}

private JMapViewer map(){
    return treeMap.getViewer();
}
private static Coordinate c(double lat, double lon){
    return new Coordinate(lat, lon);
}
private static Coordinate c(Point2D p){
    return new Coordinate(p.y(),p.x());
}


public static void main(String[] args) {
```

```
        // java.util.Properties systemProperties = System.getProperties();
        // systemProperties.setProperty("http.proxyHost", "localhost");
        // systemProperties.setProperty("http.proxyPort", "8008");
        new TestFrame().setVisible(true);
    }
}
```