Creation of weighted Theissen polygons using Python

Colin Spohn

James Madison University

Follow this and additional works at: https://commons.lib.jmu.edu/honors201019

Part of the Geographic Information Sciences Commons

Recommended Citation
https://commons.libjmu.edu/honors201019/96

This Thesis is brought to you for free and open access by the Honors College at JMU Scholarly Commons. It has been accepted for inclusion in Senior Honors Projects, 2010-current by an authorized administrator of JMU Scholarly Commons. For more information, please contact dc_admin@jmu.edu.
Creation of Weighted Thiessen Polygons Using Python

An Honors Program Project Presented to
the Faculty of the Undergraduate
College of Integrated Science and Technology
James Madison University

by Colin Michael Spohn
August 2015

Accepted by the faculty of the Department of Geographic Science, James Madison University, in partial fulfillment of the requirements for the Honors Program.

FACULTY COMMITTEE:  

Project Advisor: Helmut Kraenzle, Ph.D.
Professor, Geographic Science

Reader: Mace Bentley, Ph.D.
Associate Professor, Geographic Science

Reader: Chris Mayfield, Ph.D.
Assistant Professor, Computer Science

HONORS PROGRAM APPROVAL:

Philip Frana, Ph.D.,
Interim Director, Honors Program

PUBLIC PRESENTATION

This work is accepted for presentation, in part or in full, at [venue] HHS Room 1202 on [date] April 17, 2015.
# Table of Contents

Abstract ................................................................. 1

Thiessen Polygons ......................................................... 1 - 4

Handling Gaps .............................................................. 5

Code Procedure ............................................................ 5 - 10

Application ................................................................. 11 – 16

Conclusion ................................................................. 16

Sources ................................................................. 17

Appendix A ............................................................... 18 – 25

Appendix B ............................................................... 26

Appendix C ............................................................... 27

Acknowledgements ....................................................... 28
Abstract:
This project is an exploration into creating and manipulating Thiessen polygons. Thiessen polygons are zones created around points to show the area closer to each point than any other point in the set. Thiessen polygons are used to extend climate data from individual stations across areas. After creating the new procedure applications of the algorithm were considered.

The manipulation explored in the project focused on allowing the borders of traditional Thiessen polygons to be shifted based on values among the points. These manipulated Thiessen polygons have been dubbed weighted Thiessen polygons.

These weighted Thiessen polygons were created using the Python programming language and ESRI ARCMAP software. Each edge of weighted Thiessen polygons gets shifted so that points with higher values get larger zones proportional to other points.

In order to demonstrate how weighted Thiessen polygons work a set of points corresponding to teams from the National Football League were weighted based on team wins and playoff success over ten year increments starting from the 1966 season that marked the first Superbowl. These results were used to estimate fan bases among teams and shifts over time.

Thiessen Polygons

Thiessen polygons, also known as Voronoi Diagrams, are polygons that can be created from a set of points. The first use of these diagrams in meteorology was by Alfred Thiessen in 1911, and the process now bears his name (Thiessen). The definition of a Thiessen polygon is the area on a plane closer to a particular point than any other point in the set (Weisstein). So
anywhere within a Thiessen polygon is closer to the central place than any other central place.

Figure 1 is a set of Thiessen polygons corresponding to United States weather stations.

Everywhere in green is closer to the red dot in its purple outline than any other red dot.

The edges of each polygon therefore can be defined as the bisector between two points, such that the distance from any point on the line to both relevant points is equal. Using this information it is possible to create Thiessen polygons by finding the edge lines between a point and the other points in its set that outline the smallest area. Figure 2 is an example of this concept; the dotted lines show the temporary lines that get bisected by the solid lines creating the edges of the polygon.

![Figure 1. A network of Thiessen Polygon (Chin et. All)](image)
Creating weighted Thiessen polygons is done by starting with a set of Thiessen polygons and shifting the dividing lines. Shifting these values is something that does not have an exact mathematical definition, in contrast to standard Thiessen polygons exact center. Because of this lack of exact definition, the amount any site should shift is up to the judgment of the user. But the process remains roughly the same. Based on a value each point has, called the z-value, the dividing line will shift towards one of the points. This z-value should exist in every point of the set and the execution of the shifts should be uniform across the set. The shifts can be defined to be a proportion of the two z-values or a flat shift based only on one value being greater or lower than the other. Figure 3 is an example of a weighted polygon using the same
points in Figure 2 with a set of z values. Because of the weighted values the upper right point is no longer an influence on the polygon.

After making weighted Thiessen polygons, the complete coverage of an area that is found using standard Thiessen polygons is lost. At the intersection of three or more weighted polygons it is typical that there will be a gap. This gap is the result of shifting the midpoints so that when the lines are extended the dividing lines do not meet in exact points but in a number of points close to each other. This problem can be addressed in multiple ways, either using the gaps as important information or filling the gaps with other proximity judgements.
Handling Gaps

One suggested approach to handling these gaps is to recognize the significance of the gaps. The gaps can be a realistic interpretation of the real world. At the intersection of influence between three separate areas there can be a melding of the three areas that makes a majority of any one influence unlikely. The presence of these gaps reflect that reality and can be used to identify spots that may be good for expansion of the network of points.

This approach is not always applicable though. In some cases, complete coverage is a necessity. Achieving this coverage can be done in multiple ways. One method is to extend each polygon border of the gap so that they meet at the center of the gap. This is not something that current tools can currently help with, but is simple enough to do using pen and paper or with software that can edit the files. However, depending on the number of points in the set, it may be time consuming. For a simple and fast fix, overlaying the weighted polygons onto a set of standard Thiessen polygons can work. This will be ignoring the power of the weighted values in favor of proximity which may not match up well. But for quick overviews, it can help address the problem gaps present.

Code Procedure

The procedure for creating weighted polygons involves two separate python modules one called Polygon and one called Line. Polygon reads in coordinates and a z-value from a text file and puts them in a list using the float data type (a numerical data type with decimal places). Each polygon is made on its own, so at the start of every cycle a copy of the list is made that has the central point of the polygon removed. The copied list is sent into Line to create a list of the shifted bisecting lines that will result in the polygons’ borders.
Each line is calculated by subtracting the bordering point’s x and y value from the central point’s x and y value. These values are adjusted based on the proportion selected for the two z-values and added to the bordering point’s x and y value. Then this sum is subtracted from the central point resulting in the dividing point of the two. These two values are added to a list along with the distance from the dividing point and the inverse of the slope between these two points. Then using this slope and the dividing point, the line is extended a large amount used to represent the infinite nature of a line. The x and y value of the end points are added to the list. This list is added to a larger list, and the process is repeated for each bordering point. After the process is finished, the large list is returned to Polygon.

The large list is sorted so that the furthest lines from the central point are at the start of the list. After the list has been sorted, it is sent into a function called whittle. Whittle starts with two lines and calculates their intersect. The intersect point and the two points not excluded from being part of the polygon are added to a list to be vertices. Then, each closer line is compared to the existing lists points searching for intersects. If there is only one intersect, both the intersect point and the point to the other side of the line are added to the vertex list. The intersect point is added to the list, between the two coordinates that represent the ends of the line the new line intersects with, and the end point is added to end of the list. If there are two intersects, they are both added similarly to the single intersect. Figures 4, 5, and 6 show the first part of the whittle process. Figure 4 is the set of possible lines to be added as edges. Figure 5 shows the first two lines being judged, the blue circle is the intersect added to the vertex list and the red circles are the two endpoints that are excluded. Figure 6 is the addition of the next line with the blue circles representing added vertices.
Figure 4. A set of lines to be trimmed.
Figure 5. Evaluating the first two lines

Figure 6. Adding the next line
After a new two points are added to the vertex list, the vertex list is trimmed. Each point gets checked to see if a line between it and the center point intersects any line between neighboring points in the list. If it does intersect, the point is no longer part of the polygon because there is an edge closer to the central point. So this point is eliminated. As this process repeats, the set of lines are turned into a polygon’s vertices with only the closest edges to the central point. This list of vertices are added to a larger list for storage. Figure 7 shows the trimming process, the red dashes are the lines from vertices to the center checked for intersection. The red circle shows the vertex that will be removed because its dashed line intersects an edge line. Figures 8, 9, 10, and 11 show the full process of evaluating vertices to add and the trim. Blue circles are added vertices and red circles are removed vertices.

Figure 7. Trimming the vertex list
The full process is then repeated for each point in the set. After each point has been the central point there should be a list of vertices that create weighted Thiessen polygons when connected in order. This is accomplished using Arcpy (A python library that is part of ESRI’s software suite) to create a shapefile that can be visualized using ArcMap.

**Application**
In order to evaluate the proposed process, we designed experiments using National Football League team information. The z-values were calculated by assigning one point for a regular season win, two points for a playoff win, and five points for a super bowl victory. In order to accommodate arrival of new teams and relocations of teams, the data was split into ten year periods. If teams shared the same city, they were put into separate trials. So teams like the New York Jets and Giants did not simply split the area based on miniscule differences in location. The collected data started in the 1966 season, the first year with a super bowl held (pro-football-reference.com).

By looking at ten year stretches, one can estimate a person’s team loyalty based on location and era of their first interest in the NFL. By overlaying each successive ten year period, an estimate of team fan bases can be identified that may correspond to marketing regions for merchandise or areas where team’s broadcasts should be shown.

Maps created using this data are shown in figures 12, 13, 14, 15, 16, and 17. Figure 12 is a map for 2006 to 2014 with labels that correspond to the same color patterns, chosen to match team colors with minimal confusion, in future images. Figure 13 is 1996 to 2005, Figure 14 is 1986 to 1995, Figure 15 is 1976 to 1985 and Figure 16 is a layered composite from 1976 to 2014 with more modern results higher in the layering. Figure 17 shows 1966 to 1975 and provide an example of errors. Due to rounding issues that throw of placement of vertices some polygons are clearly in error. By looking at neighboring polygons it is possible to see the proper placement but addressing these errors should be undertaken for future studies.
Figure 12. 2006 - 2014

Figure 13. 1996 - 2005
Figure 14. 1986 - 1995

Figure 15. 1976 - 1985
Using the maps it is possible to make assumptions about a team’s fan base. The first map shows the modern loyalty boundaries and using them you can pinpoint where new fans of each team are likely to live. Until the early 2000s a fan from Texas’s southern coast was likely to root for the Oilers. With the loss of the Oilers and the introduction of the Texans new fans
from the south Texan coast will probably root for the Texans. Poor play recently by teams like the Redskins and Raiders have made it hard to gain new fans and their area of influence shrinks because of it. Gaps like the one in Arkansas in the first map show areas where no team has true support because five teams, the Cowboys, Saints, Chiefs, Rams, and Titans, are all in competition for loyalty.

The composite map makes seeing overlap of eras easy. Battle ground areas like Nevada where 6 different teams, the Rams, Raiders, Chargers, Broncos, Cardinals, and 49ers, are a jumbled mess of lines. Areas where there is no overlap, like the Rocky mountain zone the Broncos dominate, show places where no major changes in loyalty have likely occurred.

By comparing the maps to each other changes in influence can be seen. In the 1970s and 1980s the falcons had a large area with fans including most of Tennessee, South Carolina, Georgia, Alabama, and the Florida panhandle. When teams like the Panthers, Titans, and Jaguars were formed or moved in the 1990s the Falcons fan base shrunk as people started supporting their new local team.

Movement of teams can be seen in examples like the Colts in the 1980s map. There are two distinct polygons, one in Maryland and one in Indiana. When the Colts left Baltimore for Indianapolis people in Baltimore felt betrayed and some stopped supporting the team. In Indianapolis many fans had loyalty to the Bengals and Bears so were unlikely to embrace the Colts right away.

Further applications of this system may include mapping water well usage with the z-value corresponding to output volume of the wells. The procedure could also be used to create rough congressional boundaries without gerrymandering, by putting a set of population centers
as the coordinates and the assign z-values corresponding to estimates of population density.

The political boundaries would need to be restructured slightly to reach equivalent population counts but would eliminate stretched and irregularly shaped districts.

**Conclusion**

Thiessen polygons traditionally correspond just to areas of proximity. The creation of weighted Thiessen polygons by whittling a set of possible edge lines and compiling the resulting polygons together allows analyzing point data in new ways. People do not automatically associate with the closest point so including other factors can better pinpoint zones of influence. In some cases there is no dominant influence in an area as multiple places compete. The NFL example shows how success in a field can attract attention across an area while allowing competitors to grow and compete can cost that influence.
Sources


Appendix A (Code)

Program available for public use under GNU General Public License

Polygon:

# Created by Colin Spohn, James Madison University
# Last Edited 2/22/2015
###

# Imports
import re
import copy
import arcpy
import math
from operator import itemgetter

# Input from file
fo = open("1970NYJOAK.txt", "r")

# Create a file geodatabase in ArcGIS.
arcpy.CreateFileGDB_management( "C:/Users/Colin/Desktop/HON/Polygons", "Polygons.gdb")

# Create geodatabase tables to build the points from.
arcpy.CreateTable_management( "C:/Users/Colin/Desktop/HON/Polygons/Polygons.gdb", "Coordinates")

# Reset the workspace to the proper location.
arcpy.env.workspace = "C:/Users/Colin/Desktop/HON/Polygons/Polygons.gdb"

# Add the coordinates fields for the location and value of each point.
arcpy.AddField_management( "Coordinates", "XValue", "DOUBLE")
arcpy.AddField_management( "Coordinates", "YValue", "DOUBLE")
arcpy.AddField_management( "Coordinates", "ZValue", "DOUBLE")

import PolygonLineWeight
import PolygonLineEqual

def lineIntersection(line1, line2):
    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1])

    def det(a, b):
        return a[0] * b[1] - a[1] * b[0]

    div = det(xdiff, ydiff)
    if div == 0:
        return []
    d = (det(*line1), det(*line2))
x = (det(d, xdiff) / div)
y = (det(d, ydiff) / div)

# check if intersect is on both segments

# check line 1 x values
if round(line1[0][0], 1) == round(line1[1][0], 1):
x1Check = round(line1[0][0], 1) == round(x, 1)
elif line1[0][0] <= line1[1][0]:
x1Check = line1[0][0] <= x <= line1[1][0]
else:
x1Check = line1[1][0] <= x <= line1[0][0]

# check line 2 x values
if round(line2[0][0], 1) == round(line2[1][0], 1):
x2Check = round(line2[0][0], 1) == round(x, 1)
elif line2[0][0] <= line2[1][0]:
x2Check = line2[0][0] <= x <= line2[1][0]
else:
x2Check = line2[1][0] <= x <= line2[0][0]

# check line 1 y values
if round(line1[0][1], 1) == round(line1[1][1], 1):
y1Check = round(line1[0][1], 1) == round(y, 1)
elif line1[0][1] <= line1[1][1]:
y1Check = line1[0][1] <= y <= line1[1][1]
else:
y1Check = line1[1][1] <= y <= line1[0][1]

# check line 2 y values
if round(line2[0][1], 1) == round(line2[1][1], 1):
y2Check = round(line2[0][1], 1) == round(y, 1)
elif line2[0][1] <= line2[1][1]:
y2Check = line2[0][1] <= y <= line2[1][1]
else:
y2Check = line2[1][1] <= y <= line2[0][1]

if (x1Check and x2Check and y1Check and y2Check):
    return [x, y]
else:
    return []

def whittle(point, edges):
    points = []

    # add initial line
    firstPoint = (edges[0][0], edges[0][1])
    secondPoint = (edges[0][4], edges[0][5])
    points.append(firstPoint)
    points.append(secondPoint)

    # cycle through remaining lines
    i = 1
    n = len(edges)
    while i < n:
        # create lines left and right of midpoint
        point1 = (edges[i][0], edges[i][1])
        point2 = (edges[i][2], edges[i][3])
        point3 = (edges[i][4], edges[i][5])
point4 = (edges[i][4], edges[i][5])

line1 = []
line2 = []
line1.append(point1)
line1.append(point2)
line2.append(point3)
line2.append(point4)

leftEndPoint = [edges[i][0], edges[i][1], len(points), 1000]
rightEndPoint = [edges[i][4], edges[i][5], len(points), 1000]

leftIntersects = []
rightIntersects = []

#edges
leftIntersects.append(leftEndPoint)
rightIntersects.append(rightEndPoint)

#check new line against each existing line segment
j = 0
m = len(points) - 1
while j < m:
    intersect1 = []
    intersect2 = []

    point5 = (points[j][0], points[j][1])
    point6 = (points[j+1][0], points[j+1][1])

    line3 = []
    line3.append(point5)
    line3.append(point6)

    intersect1 = lineIntersection(line1, line3)
    intersect2 = lineIntersection(line2, line3)

    if len(intersect1) > 0:
        x = edges[i][2] - intersect1[0]
        y = edges[i][3] - intersect1[1]
        length = math.sqrt(math.pow(x, 2) + math.pow(y, 2))
        intersect1.append(j+1)
        intersect1.append(length)
        leftIntersects.append(intersect1)

    if len(intersect2) > 0:
        x = edges[i][2] - intersect2[0]
        y = edges[i][3] - intersect2[1]
        length = math.sqrt(math.pow(x, 2) + math.pow(y, 2))
        intersect2.append(j+1)
        intersect2.append(length)
        rightIntersects.append(intersect2)

    j = j + 1

#repeat for last point to first point
intersect1 = []
intersect2 = []
point5 = (points[len(points)-1][0], points[len(points)-1][1])
point6 = (points[0][0], points[0][1])

line3 = []
line3.append(point5)
line3.append(point6)

intersect1 = lineIntersection(line1, line3)
intersect2 = lineIntersection(line2, line3)

if len(intersect1) > 0:
x = edges[i][2] - intersect1[0]
y = edges[i][3] - intersect1[1]
length = math.sqrt(math.pow(x, 2) + math.pow(y, 2))
intersect1.append(len(points))
intersect1.append(length)
leftIntersects.append(intersect1)

if len(intersect2) > 0:
x = edges[i][2] - intersect2[0]
y = edges[i][3] - intersect2[1]
length = math.sqrt(math.pow(x, 2) + math.pow(y, 2))
intersect2.append(len(points))
intersect2.append(length)
rightIntersects.append(intersect2)

#get shortest intersecting segment on top
leftIntersects = sorted(leftIntersects, key=itemgetter(3))
rightIntersects = sorted(rightIntersects, key=itemgetter(3))

#add left intersect
leftPoint = (leftIntersects[0][0], leftIntersects[0][1])
points.insert(leftIntersects[0][2], leftPoint)

#add right intersect
rightPoint = (rightIntersects[0][0], rightIntersects[0][1])
if leftIntersects[0][2] <= rightIntersects[0][2]:
    points.insert(rightIntersects[0][2]+1, rightPoint)
else:
    points.insert(rightIntersects[0][2], rightPoint)

#add new line if only one intersect
if len(leftIntersects) == 1 and len(rightIntersects) > 1:
    points.remove(leftPoint)
    points.insert(rightIntersects[0][2]+1, leftPoint)

#add new line if only one intersect
if len(rightIntersects) == 1 and len(leftIntersects) > 1:
    points.remove(rightPoint)
    points.insert(leftIntersects[0][2]+1, rightPoint)

#remove
j = 0
m = len(points)
while j < m:
    if (points[j] != leftPoint and points[j] != rightPoint):
        newLine = [leftPoint, rightPoint]
        midLine = [(point[0], point[1]), (points[j][0], points[j][1])]
        checkIntersect = []
        checkIntersect = line_intersection(newLine, midLine)
        if len(checkIntersect) > 0:
            points.remove(points[j])
            j = j - 1
            m = m - 1
        j = j + 1
        i = i + 1

pointsCleaned = []
checked = set()
for pair in points:
    if pair not in checked:
        pointsCleaned.append([pair[0], pair[1]])
        checked.add(pair)
return pointsCleaned

# Define the list to store coordinates
coord = []
# Read the first line
line = fo.readline()
# Loop through the text file adding new sets of coordinates to the list
while line:
    parts = re.split(':|,', line)
    coord.append([float(parts[3]), float(parts[5]), float(parts[7])])
    line = fo.readline()
# Create values to be used in loops
i = 0
n = len(coord)
polygons = []
polygonShapes = []

while i < n:
    extraCoords = copy.deepcopy(coord)
    extraCoords.remove(coord[i])
    edges = PolygonLineWeight.createLines(extraCoords, coord[i])
    edges = sorted(edges, key=itemgetter(7), reverse=True)
    # whittle
    polygon = whittle(coord[i], edges)
polygons.append(polygon)
i = i + 1

for shape in polygons:
polygonShapes.append(arcpy.Polygon(arcpy.Array([arcpy.Point(*eachPair) for eachPair in shape])))
arcpy.CopyFeatures_management(polygonShapes, "C:/Users/Colin/Desktop/HON/Polygons/Polygons.gdb/Weight.Weight")

# Create values to be used in loops
i = 0
n = len(coord)
polygons = []
polygonShapes = []

while i < n:
    extraCoords = copy.deepcopy(coord)
    extraCoords.remove(coord[i])
    edges = PolygonLineEqual.createLines(extraCoords, coord[i])
    edges = sorted(edges, key=itemgetter(7), reverse=True)
    #whittle
    polygon = whittle(coord[i], edges)

    polygons.append(polygon)
i = i + 1

for shape in polygons:
polygonShapes.append(arcpy.Polygon(arcpy.Array([arcpy.Point(*eachPair) for eachPair in shape])))
arcpy.CopyFeatures_management(polygonShapes, "C:/Users/Colin/Desktop/HON/Polygons/Polygons.gdb/Equal.Equal")

Line:

###
#Lines
#
#Created by Colin Spohn, James Madison University
#Last Edited 1/14/2015
###

# Imports
import re
import arcpy
import math
def midpoints(coord, point):
    # Create values to be used in loops
    i = 0
    n = len(coord)
    newCoords = []

    while i < n:
        x = point[0] - coord[i][0]  # center x minus other x
        y = point[1] - coord[i][1]  # center y minus other y

        midX = coord[i][0] + ((coord[i][2]*x)/(point[2]+coord[i][2]))
        midY = coord[i][1] + ((coord[i][2]*y)/(point[2]+coord[i][2]))

        x2 = point[0] - midX
        y2 = point[1] - midY

        length = math.sqrt(math.pow(x2, 2) + math.pow(y2, 2))
        slope = getSlope(point[0], point[1], coord[i][0], coord[i][1])

        if slope == 9999999999999999.0:
            perpendicular = 0
        elif slope == 0:
            perpendicular = 9999999999999999.0
        else:
            perpendicular = -1.0/slope

        midPoint = (midX, midY, perpendicular, length)

        newCoords.append(midPoint)
        i = i + 1
    return newCoords

def getSlope(x1, y1, x2, y2):
    if x2 == x1:
        slope = 9999999999999999.0
    else:
        slope = (y2 - y1)/(x2 - x1)
    return slope

def createLines(coords, point):
    midPoints = midpoints(coords, point)
    edges = []

    # Create values to be used in loops
    i = 0
    n = len(midPoints)
    lines = []

    # take each midpoint and turn it into a possible edge line
    while i < n:
        line = extend(midPoints[i][0], midPoints[i][1], midPoints[i][2], midPoints[i][3])
edges.append(line)
i = i + 1

return edges

#extend a line, from a single point and slope, in both directions
def extend(x, y, slope, distance):
    if slope < 0:
        positiveX = (math.sqrt(100000.0/(1+math.pow(slope, 2))))-x)*(-1)
        positiveY = -(math.sqrt(100000.0-math.pow(x-positiveX, 2))-y)*(-1))
        negativeX = x + (x - positiveX)
        negativeY = (y + (y - positiveY))
        line = [positiveX, positiveY, x, y, negativeX, negativeY, slope, distance]
    elif slope == 9999999999999999.0:
        positiveX = x
        positiveY = y + 1000
        negativeX = x
        negativeY = y - 1000
        line = [positiveX, positiveY, x, y, negativeX, negativeY, slope, distance]
    elif slope ==0:
        positiveX = x + 1000
        positiveY = y
        negativeX = x - 1000
        negativeY = y
        line = [positiveX, positiveY, x, y, negativeX, negativeY, slope, distance]
    else:
        positiveX = (math.sqrt(100000.0/(1+math.pow(slope, 2))))-x)*(-1)
        positiveY = (math.sqrt(100000.0-math.pow(x-positiveX, 2))-y)*(-1))
        negativeX = x + (x - positiveX)
        negativeY = (y + (y - positiveY))
        line = [positiveX, positiveY, x, y, negativeX, negativeY, slope, distance]
    return line
# Appendix B (Journal of Progress)

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2013</td>
<td>First meetings with advisor, discussion of Fractals</td>
</tr>
<tr>
<td>January – February 2014</td>
<td>Further meetings and research on Thiessen Polygons and Fractals</td>
</tr>
<tr>
<td>March 2014</td>
<td>Creation of program to make simple Fractals as test case</td>
</tr>
<tr>
<td>April 2014</td>
<td>Development of proposal</td>
</tr>
<tr>
<td>August 2014</td>
<td>Creation of Thiessen Polygons in real space using pen and paper</td>
</tr>
<tr>
<td>September 2014</td>
<td>First attempt to create Thiessen Polygons with a program, abandoned in favor of multiple class design</td>
</tr>
<tr>
<td>October 2014</td>
<td>Brief return to Fractals before decision to concentrate solely on Thiessen Polygons</td>
</tr>
<tr>
<td>November 2014</td>
<td>Three class implementation of Thiessen Polygons, capable of making standard polygons but abandoned because could not be properly shifted to weighted format</td>
</tr>
<tr>
<td>December 2014</td>
<td>Pen and graphing paper experiments with weighted polygons and two class implementation of Thiessen Polygon code</td>
</tr>
<tr>
<td>January 2015</td>
<td>First successful examples of Weighted Polygons, recognizing and addressing the gap problem</td>
</tr>
<tr>
<td>February 2015</td>
<td>Application of real world data to the procedure</td>
</tr>
<tr>
<td>March 2015</td>
<td>Writing of Paper</td>
</tr>
<tr>
<td>April 2015</td>
<td>Paper revisions and approval by committee members</td>
</tr>
</tbody>
</table>
## Appendix C (NFL Data)

<table>
<thead>
<tr>
<th>Team</th>
<th>2010s</th>
<th>2000s</th>
<th>1990s</th>
<th>1980s</th>
<th>1970s</th>
<th>Lat</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Washington</td>
<td>54</td>
<td>81</td>
<td>112</td>
<td>109</td>
<td>82</td>
<td>38.8951</td>
<td>-77.0366</td>
</tr>
<tr>
<td>NYG</td>
<td>101</td>
<td>85</td>
<td>114</td>
<td>65</td>
<td>51</td>
<td>40.7127</td>
<td>-74.0059</td>
</tr>
<tr>
<td>Philly</td>
<td>86</td>
<td>103</td>
<td>91</td>
<td>78</td>
<td>48</td>
<td>39.95</td>
<td>-75.1666</td>
</tr>
<tr>
<td>Dallas</td>
<td>88</td>
<td>76</td>
<td>116</td>
<td>130</td>
<td>124</td>
<td>32.7758</td>
<td>-96.7966</td>
</tr>
<tr>
<td>Carolina</td>
<td>71</td>
<td>87</td>
<td>7</td>
<td></td>
<td></td>
<td>35.2269</td>
<td>-80.8433</td>
</tr>
<tr>
<td>Atlanta</td>
<td>79</td>
<td>81</td>
<td>63</td>
<td>67</td>
<td>46</td>
<td>33.755</td>
<td>-84.39</td>
</tr>
<tr>
<td>New Orleans</td>
<td>102</td>
<td>65</td>
<td>91</td>
<td>51</td>
<td>32</td>
<td>29.9647</td>
<td>-90.0705</td>
</tr>
<tr>
<td>Tampa</td>
<td>52</td>
<td>102</td>
<td>48</td>
<td>48</td>
<td></td>
<td>27.9472</td>
<td>-82.4586</td>
</tr>
<tr>
<td>Chicago</td>
<td>84</td>
<td>66</td>
<td>101</td>
<td>93</td>
<td>47</td>
<td>41.8369</td>
<td>-87.6847</td>
</tr>
<tr>
<td>GreenBay</td>
<td>111</td>
<td>119</td>
<td>79</td>
<td>66</td>
<td>86</td>
<td>44.5133</td>
<td>-88.0158</td>
</tr>
<tr>
<td>Detroit</td>
<td>50</td>
<td>57</td>
<td>74</td>
<td>62</td>
<td>67</td>
<td>42.3313</td>
<td>-83.0458</td>
</tr>
<tr>
<td>Minnesota</td>
<td>69</td>
<td>101</td>
<td>96</td>
<td>82</td>
<td>103</td>
<td>44.9833</td>
<td>-93.2667</td>
</tr>
<tr>
<td>St. Louis</td>
<td>43</td>
<td>100</td>
<td>7</td>
<td></td>
<td></td>
<td>38.6272</td>
<td>-90.1977</td>
</tr>
<tr>
<td>LARams</td>
<td>64</td>
<td>100</td>
<td>101</td>
<td>34</td>
<td>101</td>
<td>34.05</td>
<td>-118.25</td>
</tr>
<tr>
<td>Arizona</td>
<td>79</td>
<td>58</td>
<td>44</td>
<td></td>
<td></td>
<td>33.45</td>
<td>-112.067</td>
</tr>
<tr>
<td>St.L Card</td>
<td>11</td>
<td>67</td>
<td>68</td>
<td></td>
<td></td>
<td>38.6272</td>
<td>-90.1977</td>
</tr>
<tr>
<td>San Fran</td>
<td>87</td>
<td>90</td>
<td>152</td>
<td>94</td>
<td>71</td>
<td>37.7833</td>
<td>-122.417</td>
</tr>
<tr>
<td>Seattle</td>
<td>99</td>
<td>90</td>
<td>73</td>
<td>74</td>
<td></td>
<td>47.6097</td>
<td>-122.333</td>
</tr>
<tr>
<td>New England</td>
<td>137</td>
<td>136</td>
<td>63</td>
<td>85</td>
<td>45</td>
<td>42.358</td>
<td>-71.0636</td>
</tr>
<tr>
<td>NYJ</td>
<td>77</td>
<td>84</td>
<td>65</td>
<td>71</td>
<td>73</td>
<td>40.7127</td>
<td>-74.0059</td>
</tr>
<tr>
<td>Buffalo</td>
<td>58</td>
<td>76</td>
<td>120</td>
<td>56</td>
<td>52</td>
<td>42.9047</td>
<td>-78.8494</td>
</tr>
<tr>
<td>Miami</td>
<td>61</td>
<td>96</td>
<td>95</td>
<td>113</td>
<td>104</td>
<td>25.7877</td>
<td>-80.2241</td>
</tr>
<tr>
<td>Indy</td>
<td>117</td>
<td>98</td>
<td>71</td>
<td>9</td>
<td></td>
<td>39.7666</td>
<td>-86.15</td>
</tr>
<tr>
<td>Balt Colts</td>
<td>47</td>
<td>98</td>
<td>39.2833</td>
<td></td>
<td></td>
<td>39.2833</td>
<td>-76.6166</td>
</tr>
<tr>
<td>Tennessee</td>
<td>69</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td>36.1666</td>
<td>-86.7833</td>
</tr>
<tr>
<td>HoustOil</td>
<td>8</td>
<td>90</td>
<td>71</td>
<td>51</td>
<td>51</td>
<td>29.7627</td>
<td>-95.383</td>
</tr>
<tr>
<td>Jacksonville</td>
<td>55</td>
<td>98</td>
<td>4</td>
<td></td>
<td></td>
<td>30.3369</td>
<td>-81.6613</td>
</tr>
<tr>
<td>Houston</td>
<td>74</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td>29.7627</td>
<td>-95.383</td>
</tr>
<tr>
<td>Baltimore</td>
<td>113</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td>39.2833</td>
<td>-76.6166</td>
</tr>
<tr>
<td>Pittsburgh</td>
<td>103</td>
<td>119</td>
<td>95</td>
<td>116</td>
<td>86</td>
<td>40.4416</td>
<td>-80</td>
</tr>
<tr>
<td>Cleveland</td>
<td>48</td>
<td>36</td>
<td>88</td>
<td>74</td>
<td>82</td>
<td>41.4822</td>
<td>-81.6697</td>
</tr>
<tr>
<td>Cincy</td>
<td>73</td>
<td>61</td>
<td>70</td>
<td>77</td>
<td>55</td>
<td>39.1</td>
<td>-84.5166</td>
</tr>
<tr>
<td>Kansas City</td>
<td>58</td>
<td>89</td>
<td>96</td>
<td>58</td>
<td>96</td>
<td>39.0997</td>
<td>-94.5783</td>
</tr>
<tr>
<td>San Diego</td>
<td>96</td>
<td>64</td>
<td>79</td>
<td>88</td>
<td>56</td>
<td>32.715</td>
<td>-117.163</td>
</tr>
<tr>
<td>Oakland</td>
<td>43</td>
<td>81</td>
<td>8</td>
<td>82</td>
<td>117</td>
<td>37.8044</td>
<td>-122.271</td>
</tr>
<tr>
<td>LARad</td>
<td>79</td>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td>34.05</td>
<td>-118.25</td>
</tr>
<tr>
<td>Denver</td>
<td>88</td>
<td>128</td>
<td>103</td>
<td>98</td>
<td>51</td>
<td>39.7618</td>
<td>-104.881</td>
</tr>
</tbody>
</table>
Acknowledgements

Thank you to everyone in the Geographic Science and Computer Science departments at JMU for running such tremendous program.

A special thanks to Dr. Kraenzle my advisor, and Dr. Bentley and Dr. Mayfield my thesis readers.