Polygon Simplification for Location-Based Services Using Population Density

Wonsang Song, Jae Woo Lee, and Henning Schulzrinne
Department of Computer Science, Columbia University
Email: {wonsang, jae, hgs}@cs.columbia.edu

***Abstract*—An important group of location-based services (LBS), including 9-1-1 service, rely on the mapping between a user's location and a service boundary in order to select the appropriate service provider. In such cases, mobile clients can cache the mapping information to reduce service latency and server load. However, caching such a mapping can be burdensome on mobile devices because representing the polygon that defines a service boundary requires a large amount of data.**

We present GeoPS-PD, a polygon simplification algorithm designed for LBS applications. Unlike existing algorithms, GeoPS-PD never produces a false positive, is tunable at runtime for the desired balance between target polygon size and area coverage, and optionally takes into account the population density. We demonstrate the efficacy of GeoPS-PD using the US state boundary data. For New York, GeoPS-PD produces a simplified polygon which is only 3% of the original size, yet covers 95% of the original area, and makes the LBS queries 3.17 times faster.

I. Introduction

The explosive growth of mobile devices with GPS capability has brought location-based services (LBS) into the mainstream. Pedestrians with smartphones looking for nearby restaurants are commonplace in city blocks. Typical LBSs provide the location of desired services based on the distance from a user's current location [1]. There are, however, an important group of LBSs that use the user's location in a slightly different way. Rather than calculating the direct distance to the user's location, they first map the user's location into a predefined area, and then return a service provider responsible for the area. For example, a 9-1-1 call must be routed to the police station whose jurisdictional boundary includes the caller's location.

Such a mapping between the user location and the service boundary is common in many LBS systems, and thus can be implemented using a generic component, such as a Location-to-Service Translation (LoST) [2] system. The mapping server maintains a set of polygons which represent geographic areas, which in turn represent service boundaries. Given a point on earth, the server will find the polygon enclosing the point. When a user uses a mobile device to contact the server, the mobile device can cache polygon information so that multiple queries originating from nearby locations can be processed locally by the mobile device without contacting the server. This reduces the service latency as well as the load on the mapping server. It works well in many LBS situations, such as 9-1-1 calls, where the user's location is relatively static.

However, storing service boundaries and processing them can be a significant burden on typical mobile devices which have limited processing power and storage. The number of vertices of a polygon required to accurately represent a geographic area can be very large. We will refer to the number of vertices of a polygon as *the size of a polygon* for the rest of the paper. For example, the polygon size of the jurisdiction of the Austin police department in Texas is 41,151 [3]. In addition, the large polygon size might be problematic with the limited bandwidth of current mobile networks. The polygon for the Austin police department encoded in Geography Markup Language (GML) [4] is approximately 2 MB.

One way to address this problem is for the mobile client to store a simplified version of a polygon rather than the large original one. A number of polygon simplification algorithms have been developed in digital cartography for map generalization [5], such as the Douglas-Peucker algorithm [6], the Visvalingam-Whyatt algorithm [7], and the bend-simplify algorithm [8]. None of these algorithms, however, satisfies one critical requirement for mappings used in LBS applications: the simplified polygon should be fully enclosed in the original polygon so that the simplified polygon should never produce false positives. For example, if there is an area included in the simplified polygon, but not in the original polygon, a 9-1-1 call from that area will be mistakenly routed to the authority responsible for the original polygon, when in fact, the call has originated from outside of its jurisdiction. This can lead to a significant delay in emergency response.

We present GeoPS-PD, a polygon simplification algorithm designed with LBS in mind. Our GeoPS-PD is based on an algorithm in computer graphics by Cohen-Or *et al.* [9], which generates an inner cover of a shape. We modified Cohen-Or's algorithm for LBS applications, and the resulting GeoPS-PD has the following features:

- It never produces false positives.
- It can be tuned at runtime by providing target polygon size and required area coverage.
- It takes population density into account during the simplification process.

We implemented GeoPS-PD in the LoST server of our Next Generation 9-1-1 (NG9-1-1) prototype system [10]. Our evaluation shows a significant performance improvement of LBS queries when a mobile client caches simplified polygons instead of original polygons.

The rest of the paper is organized as follows. We discuss the requirements of the polygon simplification algorithm for LBS in Section II. We describe our algorithm in Section III and

the implementation details in Section IV. Section V provides performance evaluation. Lastly, we conclude in Section VI.

II. Requirements of Polygon Simplification Algorithm for LBS

A polygon simplification algorithm for LBS applications must fulfill the following requirements:

- *Reduction*: the size of the simplified polygon should be significantly smaller than the original polygon.
- *Inclusion*: the simplified polygon must be fully enclosed in the original polygon.
- *Coverage*: the simplified polygon should contain most of the area of the original polygon.

A. Reduction

Reduction is the primary purpose of polygon simplification. A simplification algorithm should produce a polygon which consists of a much smaller number of points. The performance of an algorithm can be measured by the ratio of the simplified size to the original size.

B. Inclusion

The inclusion requirement prevents the cached polygon from producing false-positives. A false-positive directs a user to an incorrect service provider, as shown in Figure 1. Assume that a polygon P represents the boundary of a service provider S, and a polygon P' is the simplified version of P. If P' is not entirely enclosed in P, there is an area A which is inside P', but outside P. If a user in A makes an LBS query, she will be directed to S, the wrong service provider, even though her location is outside S's service boundary.

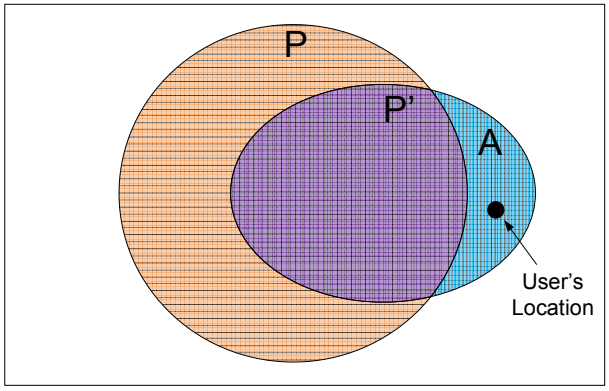


Fig. 1. False positive case.

The simplified polygon must also retain the topology of the original polygon. If there is a hole in the original polygon, it should remain or be replaced by a bigger hole. Otherwise, the simplified polygon will violate the inclusion requirement.

C. Coverage

One disadvantage of using a simplified polygon for the client cache is that the users in some areas near the border will experience constant cache miss due to the fact that the simplified polygon does not cover those particular areas. To minimize the number of users not covered by the simplified polygon, it should cover as much of the original area as possible. The area coverage, which is the ratio of the simplified area to the original area, is another performance measure of a polygon simplification algorithm.

III. GeoPS-PD: Geodesic Polygon Simplification With Population Density

GeoPS-PD is based on Cohen-Or's polygon simplification algorithm [9], which was proposed as the first step to find an inner cover of a non-convex shape in computer graphics. Cohen-Or's algorithm produces a reduced polygon that is fully enclosed within the original polygon, satisfying our inclusion requirement in Section II. We take the core method of Cohen-Or's algorithm, make it tunable, and enhance it to provide better user coverage.

GeoPS-PD takes an adaptive approach in order to produce simplified polygons with better user coverage. The goal is to prevent the simplification process from losing areas where many users make LBS queries. To this end, GeoPS-PD uses query density to guide the polygon simplification process. Query density can be calculated from query history data. Since we do not have such data at this point, we approximate query density using population density.

A. Population density

In general, urban areas have higher population density than rural areas. In New York State, for example, New York City has a population density of 26,402 per mile^2, while Hamilton County has 3.1 per mile^2 [11]. A polygon simplification algorithm for LBS applications should take into account the population density so that it does not lose a highly populated area during the simplification process. A simplified polygon that lost a small fraction of New York City would result in a lot more cache misses than the one that lost the same amount of area of Hamilton County.

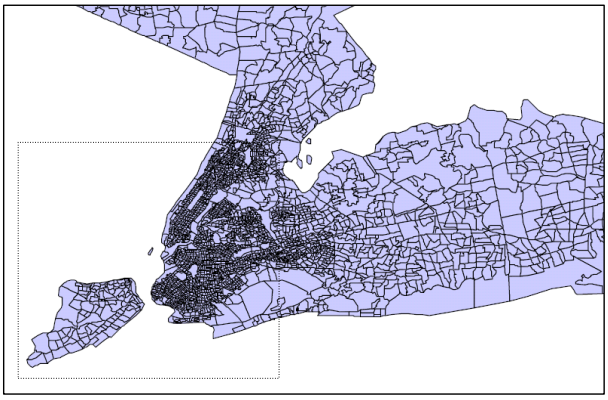


Fig. 2. Census tracts around New York City.

In order to take the population density into account, GeoPS-PD uses the census tracts boundary information published by the U.S. Census Bureau. A census tract is a small geographic region used by the Census Bureau. The census tracts cover the

entire United States and one census tract generally contains between 1,500 and 8,000 people [12]. Since a single census tract contains a comparable number of people, the spatial sizes of census tracts depend on their population density. Figure 2 shows that the census tracts in New York City are much smaller in size than those in the nearby counties.

B. Polygon simplification

Algorithm 1 describes GeoPS-PD in pseudo-code.

Algorithm 1 GeoPS-PD($P_{org}, N_{target}, R_{req}$)

```
1:  P ← P_org
2:  P' ← P
3:  while (area(P')/area(P_org) > R_req) do
4:    if size(P') ≤ N_target then
5:      return (success, P')
6:    end if
7:    P' ← {p_1}
8:    p_prev ← p_1
9:    for i = 2 to size(P) do
10:     triangle T = {p_prev, p_i, p_{i+1}}
11:     if P does not contain T
        or area(T)/area(P) ≥ ε
        or num_tracts_in(T) ≥ δ then
12:       // p_i is kept in simplified polygon
13:       add p_i to P'
14:       p_prev ← p_i
15:     else
16:       // p_i gets removed
17:     end if
18:   end for
19:   P ← P'
20: end while
21: // failed to reach N_target while keeping R_req
22: return (failure, P')
```

There are three input parameters: P_{org}, N_{target} and R_{req}. P_{org} is the original polygon which consists of a set of vertices, $p_1, p_2, \ldots, p_n$. N_{target} is the desired polygon size. A successful execution of the algorithm will return a simplified polygon whose size is less than or equal to N_{target}. R_{req} is the required area coverage. For example, if R_{req} is 0.9, GeoPS-PD must produce a simplified polygon that covers at least 90% of the original area. If it cannot reach N_{target} while keeping R_{req}, GeoPS-PD will return its best effort.

For each point p_i in the polygon P, GeoPS-PD considers if p_i can be eliminated in the simplified polygon P' by checking three conditions (line 11). First, we consider the triangle T formed by p_i and its two neighboring points p_{prev}, p_{i+1}. The point p_{prev} is the last point kept. If T is fully enclosed in P, T indicates a convex region in P, and thus p_i can be safely eliminated without violating the inclusion requirement. Second, we make sure that eliminating p_i does not lose too big an area. We define an acceptable loss threshold, ϵ, which is typically a small percentage (we used 0.2% in our evaluation), and compare it with the ratio of the area of T to the area of P. Lastly, we take the population density into consideration. We count the number of census tracts within T and compare it with a predefined threshold δ (we used 10 in our evaluation). We eliminate p_i only if T contains fewer census tracts than δ. This prevents the simplified polygon from losing highly populated areas. The algorithm runs repeatedly until the simplified polygon's size reaches N_{target} (line 4) or the area coverage drops below R_{req} (line 3).

Note that, for clarity, the pseudo-code omits a few details of our implementation. Checking for population density can be turned on and off. Determining when to terminate the algorithm is slightly more complicated than it is shown in Algorithm 1 because we make sure that the area does not dip below R_{req} during the last iteration, and that the loop terminates when the algorithm fails to converge. We also take into account various holes in a polygon caused by lakes or autonomous jurisdictions.

IV. Implementation

We applied GeoPS-PD to the LoST server. We use the LoST server to route emergency calls in our NG9-1-1 system, but it is a generic component which can be used for any LBS application that maps the user's location to the service boundary. The client-side application first determines its location and sends a request to the LoST server. The server then replies with a response containing the appropriate service provider's URL and a polygon representing the service boundary. After receiving the response, the LoST client caches the polygon for later queries.

A. Client

We implemented our LoST client on an Android Dev Phone 1, equipped with Qualcomm 528 MHz processor and 192 MB RAM [13]. It communicated with the LoST server using 802.11g wireless LAN.

The XML description of a polygon in a LoST response message was converted to two arrays–one for latitude and the other for longitude of double precision floating point numbers, and then stored as BLOBs in a SQLite database along with other information. Before making a LoST request, the client first checks the user's location against all polygons stored in the database. For point-in-polygon test, we used the `PNPOLY` function [14].

B. Server

The LoST server hosted on a Dell PowerEdge 1950 Server, equipped with dual Intel Xeon 1.6 GHz CPUs and 2 GB RAM running Linux 2.6.18. We implemented the LoST server as a Java Servlet running on Apache Tomcat [15] web application server. We used the PostgreSQL database [16] to store LoST mappings. A service boundary was stored in the database as a geometric object. PostGIS extension [17] adds geometric objects and functions to the PostgreSQL database. PostGIS provides the `ST_Contains` function for point-in-polygon test.

C. GeoPS-PD function

We implemented the GeoPS-PD algorithm as a function written in PL/pgSQL [18], the procedural language for the PostgreSQL database system. We compute the simplified polygon for each service boundary using the GeoPS-PD function, and store it in the database along with the original polygon. The computation is done offline.

D. Geographical data set

We used the actual state boundaries of the United States as the polygons representing service boundaries. In our NG9-1-1 system, 9-1-1 calls are first routed to a statewide emergency service routing proxy (ESRP), and the ESRP further routes the calls to local authorities [19]. From the user's perspective, therefore, the state boundaries make up the 9-1-1 service boundaries.

We used the state boundary data available from the U.S. Census Bureau [20]. The original data file is in the ESRI shapefile [21] format. We loaded it into the PostgreSQL database using the `shp2sql` data loader included in PostGIS. The census tracts data file was loaded in the same way.

V. Evaluation

A. Polygon simplification

We evaluated the performance of GeoPS-PD using the size and area coverage of the simplified polygons. We did not measure the running time of GeoPS-PD. The running time of the algorithm is not critical because the simplified polygons need to be recalculated only when the service boundaries change and the calculation is done offline.

For our evaluation, we picked five US states (or state equivalent areas), and simplified their boundaries using GeoPS-PD. The five states are District of Columbia, Utah, Massachusetts, New York, and Texas. They represent various points in the polygon size spectrum, ranging from 229 to 6,534.

TABLE I
GeoPS-PD results for 5 states.
($N_{target} = 100, R_{req} = 0.9, PD = off$)

State	Original size	Simplified size	Simplification ratio (%)	Area coverage (%)
DC	229	61	26.64	97.01
UT	533	97	18.20	98.12
MA	1,421	100	7.04	92.55
NY	3,093	94	3.04	95.05
TX	6,534	91	1.39	94.77

Table I shows the performance of GeoPS-PD. The population density option (PD) was turned off. The simplification ratio is the ratio of the simplified polygon's size to the original size in percentage, and the area coverage is the ratio of the simplified polygon's area to the original area. In the case of New York, for example, the original polygon size is 3,093 and the simplified polygon size is 94, which is only about 3% of the original size but still covers more than 95% of the original area.

TABLE II
States that were affected by PD.

State	Cities with high population density
California	San Francisco, Long Beach, and San Diego
Florida	St. Petersburg
Illinois	Chicago
Kentucky	Covington
Michigan	Detroit
Missouri	St. Louis
New York	New York City
Pennsylvania	Philadelphia
Texas	McAllen
Virginia	Arlington
Washington	Seattle

When we ran GeoPS-PD with the population density option turned on, 11 out of the 50 states produced different results. Table II shows the 11 states along with the densely populated cities in each state that caused the differences.

TABLE III
Simplification of New York with and without PD.
($N_{target} = 100, R_{req} = 0.9$)

	Polygon size	Area coverage (%)	Census tracts included
Original polygon	3,093	100.00	4,711
Simplified w/o PD	94	95.05	2,611
Simplified with PD	100	95.43	4,186

The results for New York clearly demonstrate the effectiveness of the population density option. Table III compares the results for New York with and without the population density option. While the polygon size and the area coverage remain similar between the two cases, the number of census tracts included in the simplified polygon is increased by 60%, jumping from 2,611 to 4,186. Put another way, the simplification using population density covers 6.3 million more people, according to the U.S. Census Bureau's statistics which states that the average population of all census tracts is about 4,000 [12]. This is about 1/3 of the total population of New York State.

Figure 3 illustrates how such a huge difference in population coverage is possible when the area coverage is almost same. Figure 3(a) shows that, without the population density option, most of New York City gets cut off during simplification. Figure 3(b) shows that the population density option prevents the loss of New York City.

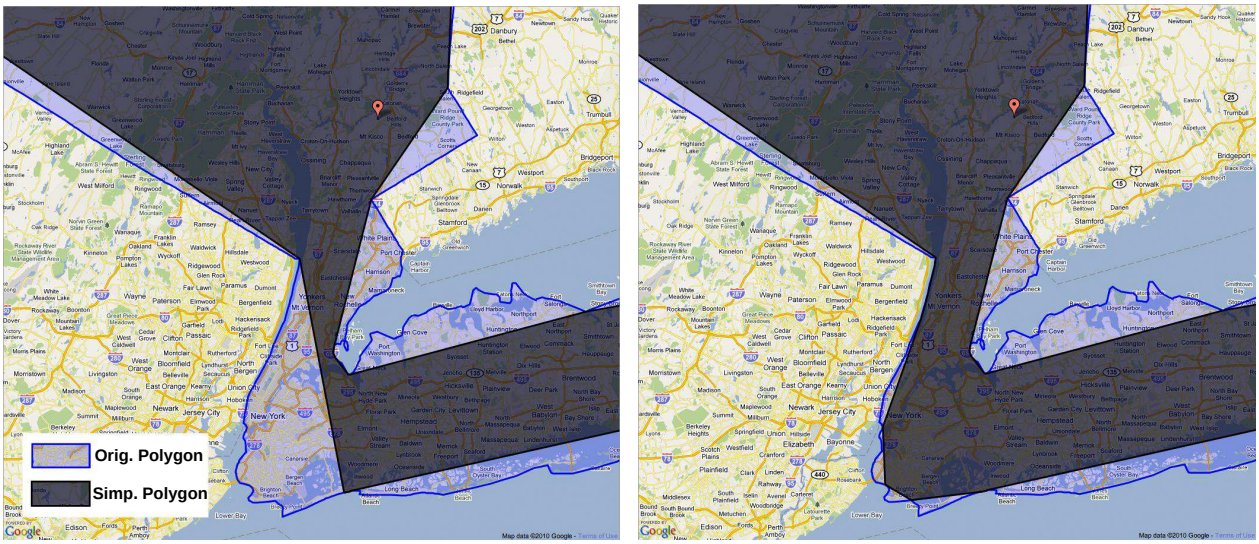
B. LoST query performance

In order to measure the performance benefit of caching simplified polygons, we analyzed the expected LoST query time for two cases. In one case, the LoST server sends the original polygon which the client then caches. In the other, the server sends the client the simplified polygon. The simplified polygon was computed in advance.

The expected query time is given by the following formula:

$$E(Query\ time) = Hit\ time + Miss\ penalty \times Miss\ rate$$

We measured $Hit\ time_{org}$, $Miss\ penalty_{org}$, $Hit\ time_{simp}$ and $Miss\ penalty_{simp}$ as follows. (org and $simp$ denote the



(a) without population density option (b) with population density option

Fig. 3. Polygon simplification of New York State near New York City.

original and simplified polygon being cached, respectively.) For the miss penalty, we let the client issue a query with an empty cache and measured the response time. At that point, the client cached the polygon that it received from the server. We let the client issue the same query again, which caused a cache hit, to measure the hit time. We selected the query location to ensure the cache hit.

For the case of original polygons, the only cache miss will occur for the first query in each service boundary. All subsequent queries within the same boundary will result in cache hits. Thus, as the number of query increases, the expected query time converges to the hit time:

$$E(Query\,time_{org}) \rightarrow Hit\,time_{org}$$

For the case of simplified polygons, a cache miss will occur every time the query location is in a region that is included in the original polygon but not in the simplified polygon. The total percentage of such regions in a service boundary is $1 - Area\ coverage$, which is the $Miss\ rate_{simp}$. The expected query time for the simplified polygon case becomes:

$$E(Query\,time_{simp}) = Hit\,time_{simp} + Miss\,penalty_{simp} \times (1 - Area\ coverage)$$

TABLE IV
LOST QUERY TIME FOR 5 STATES. (IN MILLISECONDS)

State	Original Polygon		Simplified Polygon			
	Size	Hit time	Size	Hit time	Miss penalty	Miss rate
DC	229	67	61	43	466	0.029
UT	533	84	97	58	511	0.018
MA	1,421	148	100	61	516	0.074
NY	3,093	263	100	59	527	0.045
TX	6,534	566	91	46	526	0.051

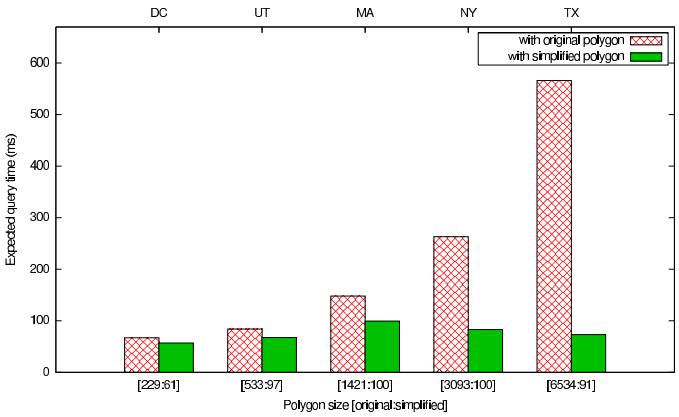


Fig. 4. Expected LoST query time for 5 states.

Table IV shows our measurements for five states: hit time, miss penalty, miss rate, and the polygon sizes. Figure 4 compares the resulting query times between the two cases. The expected LoST query times using simplified polygons are faster in all five states, and the bigger the original polygon size, the larger the performance improvement.

To see where the performance improvement comes from, let us look more closely at the numbers for New York. The expected query time using the simplified polygon is 3.17 times faster–82 ms versus 263 ms. This comes from the large difference in hit time–59 ms versus 263 ms. It takes a long time to access and process the cache of the original polygon due to the large size of 3,093 points. On the other hand, the simplified polygon case incurs a heavy penalty of 527 ms on every cache miss, but the low cache miss rate of 0.045 mitigates the effect of the penalty on the expected query time.

VI. Conclusion

We presented GeoPS-PD, a polygon simplification algorithm for LBS. GeoPS-PD produces a simplified polygon that is fully enclosed in the original one so it never produces false positives. It can be tuned at runtime by providing target polygon size and required area coverage. It also provides an option to take the population density into consideration, which can result in better population coverage in certain cases.

Our measurements with real US state boundary showed that GeoPS-PD produces simplified polygons that are much smaller in size but still cover most of the original areas. Enabling the population density option prevented GeoPS-PD from losing highly populated areas near the state borders. We have shown that the performance of LBS queries can be improved if mobile clients cache simplified polygons produced by GeoPS-PD, especially when the size of the original service boundary is large.

VII. Acknowledgments

This project was funded in part by the NSF grant #0751094.

References

[1] A. Küpper, *Location-Based Services: Fundamentals and Operation*. Wiley, 2005.

[2] H. Schulzrinne, H. Tschofenig, A. Newton, and T. Hardie, "LoST: A Protocol for Mapping Geographic Locations to Public Safety Answering Points," in *Proc. IEEE IPCCC*, 2007.

[3] "Texas 9-1-1 GIS Collaboration Portal - Download GIS Data," https://tx911map.911.state.tx.us/download/.

[4] Open Geospatial Consortium, Inc., "OpenGIS Geography Markup Language (GML) Encoding Standard," 2007.

[5] R. B. McMaster and K. S. Shea, *Generalization in Digital Cartography*. Association of American Geographers, 1992.

[6] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, December 1973.

[7] M. Visvalingam and J. D. Whyatt, "Line generalisation by repeated elimination of points," *Cartographic Journal*, vol. 30, no. 1, pp. 46–51, June 1993.

[8] Z. Wang and J. Muller, "Line Generalization Based on Analysis of Shape Characteristics," *Cartography and Geographic Information Science*, vol. 25, no. 1, pp. 3–15, Jaunary 1998.

[9] D. Cohen-Or, S. Lev-Yehudi, A. Karol, and A. Tal, "Inner-cover of Non-convex Shapes," *International Journal of Shape Modeling*, vol. 9, no. 2, pp. 223–238, December 2003.

[10] W. Song, J. Y. Kim, H. Schulzrinne, P. Boni, and M. Armstrong, "Using IM and SMS for Emergency Text Communications," in *Proc. IPTComm*, 2009.

[11] U.S. Census Bureau, "Census 2000 Summary File 1," 2001.

[12] U.S. Census Bureau, *Geographic Areas Reference Manual*, November 1994.

[13] B. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution," in *Proc. HotOS*, 2009.

[14] W. R. Franklin, "PNPOLY - Point Inclusion in Polygon Test." [Online]. Available: http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html

[15] "Apache Tomcat," http://tomcat.apache.org/.

[16] "PostgreSQL," http://www.postgresql.org/.

[17] "PostGIS," http://postgis.refractions.net/.

[18] "PL/pgSQL - SQL Procedural Language." [Online]. Available: http://developer.postgresql.org/pgdocs/postgres/plpgsql.html

[19] H. Schulzrinne and R. Marshall, "Requirements for Emergency Context Resolution with Internet Technologies," RFC 5012, 2008.

[20] "Census 2000 State and State Equivalent Areas Cartographic Boundary Files." [Online]. Available: http://www.census.gov/geo/www/cob/st2000.html

[21] "ESRI Shapefile Technical Description," 1998. [Online]. Available: http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf