Moving Window Based Geometry Simplification with Topology Constraints

Shivanth M P, Sandeep Kale, N. L. Sarda, Umesh Bellur, Vishal Goje
{shivanthmp, sanrajkale, nls, umesh, vishalgoje}@cse.iitb.ac.in

GISE Advanced Research Lab
Indian Institute of Technology Bombay
Mumbai, India

ABSTRACT

Geometric simplification is a widely used technique in the field of cartographic generalization. Many algorithms already exist in this field to simplify a geometry so that the number of points in the geometry is reduced while it retains its approximate shape. In this paper, a new algorithm, Moving Window Geometry Simplification (MWGS) algorithm is presented that can be used to reduce a geometry under a given set of constraints. The proposed algorithm uses the technique called sweep line to reduce a geometry and also ensures that none of the constraints are violated. Both computational and algorithmic aspect of the technique are considered for better performance and scaling. The method has important applications in areas relating to map generalization, map compression etc. This algorithm reduced approximately 90% of the points in all the datasets.

Categories and Subject Descriptors

I.3.5 [**COMPUTER GRAPHICS**]: Computational Geometry and Object Modeling Geometric algorithms, languages, and systems

Keywords

Cartographic Generalization,Line Sweeping Algorithm

1. INTRODUCTION

One of the essential ideas in map generalization is smoothing and simplification of a geometry in the map. Many algorithms have been designed to implement unconstrained reduction of geometries. These algorithms basically try to preserve the shape of the geometry while reducing visually redundant points in the geometry. But in many cases we would require that the solution should also work under certain constraints, such as a point which can never fall outside a polygon when it is reduced. An example of a real life constraint is smoothing of a country map when a city lies close to the border of the country. While using unconstrained algorithms for reducing the geometry of the border, the new border might not wrap the city and the city falls outside the boundary or a city that must be outside the boundary of the country being simplified falls inside the boundary of the country after simplification. The paper deals with the problem of geometric simplification where a set of constraints are given and the reduction has to be done such that these constraints are not violated. Generic algorithms like Douglas-Peucker algorithm [2] does not allow such constraints. They also have a flexible parameter ϵ which can be varied to obtain different results. The set of constraints that we consider in this paper are the points in the same plane as that of the geometry, and the reductions has to be carried out so that these points do not change the topological relationship with respect to the polygons containing them. A plain and simple idea would be to introduce the constraint check into every step of reduction in the generic algorithm. This as we see later introduces computational and algorithmic overhead. This paper proposes a new algorithm which is computationally better than the naive approach of checking for constraint violation for every polygon and point. A proof of correctness and a performance analysis of the new algorithm on various measures are also included.

The remainder of this paper consists of six sections. Section 2 presents the problem statement, whereas Section 3 discusses the related work on addressing this category of problems. Section 4 presents the proposed algorithms for geometry simplification, whereas Section 5 is devoted to the computational performance of the proposed algorithms. Finally, Section 6 and 7 includes the result and major concluding remarks respectively.

SIGSPATIAL'14, SIGSPATIAL '14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA Copyright 2014 ACM 978-1-4503-3131-9/14/1...$15.00
http://dx.doi.org/10.1145/2666310.2666424

2. PROBLEM STATEMENT

Given a set of curves L, each curve as a sequence of points, and a set of points C in the same plane - called constraint points, find a simplification of the set of curves such that each point in C maintains their topological relationships with every curve in the set L. Here, simplification of a curve refers to eliminating points from the curve so as to reduce the number of points in the representation of the curve. The curves in L will be non-intersecting except at the end points. The constraint points will never lie on the curves. The Algorithm has been developed as a solution to the SIGSPATIAL CUP 2014 [1].

3. RELATED WORK

There are a number of algorithms used to reduce curves/lines approximated by a certain set of points, the most recog-

nized being Douglas Peucker algorithm [2]. It is a recursive algorithm that removes points which are too close to the "shape of the geometry" and removes all points which do not contribute much to the shape. There are other algorithms using a similar approach with slight variations, including Visvalingam-Whyatt algorithm [3], which tries to remove the minimum area triangles out of all the triangles formed by every 3 consecutive points in the curve. Unlike Douglas - Peucker algorithm which removes all points which do not meet the requirement, this algorithm provides us with the flexibility of removing a fixed number of points from the curve. Other well-known algorithms are Reumann-Witkam [5] algorithm, Lang simplification [5] algorithm etc. None of these algorithms include any external topological constraints and need to be extended suitably.

4. OUR APPROACH - MOVING WINDOW GEOMETRY SIMPLIFICATION (MWGS) ALGORITHM

4.1 Overview

The key idea of our approach is to move a window along the X-axis over the given set of points. At each step, the points inside the window are removed, when its removal does not violate any of the given constraints. For the problem statement given above, the following terminologies are defined

- L: Set of curves that form a set of polygons depicting regions on Map
- C: Set of constraint points (like cities)
- E: Event List, List of all points in all curves sorted along X-axis left to right
- W: The Window of arbitrary size under consideration for simplification from Event List E
- W_L,W_R: Refers to the left or right edge of W (indexes into the array E)
- A: Active List contains segments of the curves in L falling in window W between W_L,W_R constructed from E at every event, A= $\{s_{11}, s_{12}, ..s_{21}, s_{22}..., s_{n1}, s_{nm}\}$ where
  s_{ij}: Segment j may contain one or more consecutive points of curve i falling between window W, segments of same curve are mutually exclusive, shares no points
  $|A|$: Total points in from all segments in A
- C_w: Constraint points within W
- $C_{w_{ij}}$: Constraint points in W that fall within the BB of segment s_{ij}

The essential idea of the algorithm, called Moving Window Geometry Simplifier (MWGS), is as follows:

- Consider only a limited portion of L falling in the window W
- Simplify these portions without violating any constraints due to C and other points in A
- Move the edges of the window to the right, either expanding the window or shifting the window as a whole

Clearly

$$|A| + |C_w| << |L| + |C|$$

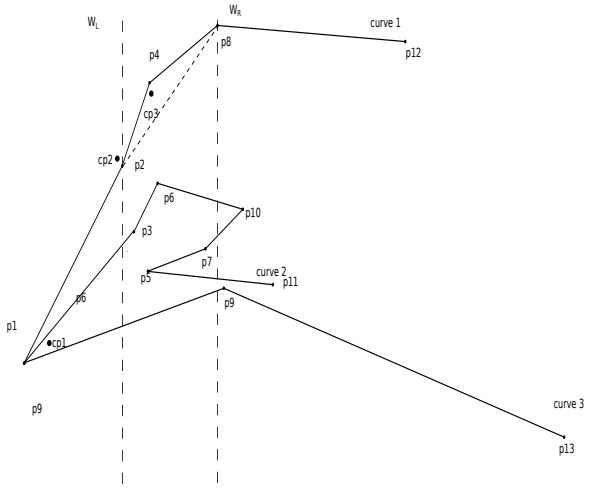

Figure 1: Illustrating our Algorithm

.

Hence we have effectively scaled down the number of comparison required from the points on the whole region to points within the window.

In order to accomplish this sweep along the X-axis, the points on every curve are sorted by their x-value into E and move along this list using an index. As we move from left to right of this list, we keep track of the segments of each curve that we encounter. A segment is a consecutive list of points from the same curve. As soon as we have a segment of 3 (or more) points, we try to reduce the segment by removing the middle point(s) by checking for constraints. The advantage here is that we check for only the constraint points which fall under the x-window of the segment under consideration. This "sweep line" based spatial algorithm or plane-sweep algorithm are extensively used in computational geometry. [4] discusses many applications of this algorithm. A figure explaining the algorithm can be found in Figure 1. The segment $s_{11} = \{p2, p4, p8\}$ of the curve 1 is being reduced. The algorithm in effect considers for all points within the window between W_L and W_R checking if any of them falls inside the triangle formed by the segment. The Active List will contain segments $s_{11} = \{p2, p4, p8\}$, $s_{21} = \{p2, p6\}$, $s_{22} = \{p7, p5\}$. Curve 3 has no active segments in window. Curve 1, 2, 3 shares one end point $p1$.

4.2 Implementation Details

The major task in the implementation of this algorithm is to identify segments on the curve as we move from left to right along X-axis. For this we create a Active List A containing multiple segments of curves. We have a *sweep_line_index* moving along the Event List E (analogous to the sweep line), and for each point encountered we try to find the segment it extends for the corresponding curve. If it does not extend any segment it is added as a new segment of that curve. There might be cases where the point might be extending two segments(one forward and the other backward), then we merge the two segments and make them a single segment. Now in each step, we extend a segment this way, so whenever the segment under consider has a length greater than 2, there is a possibility of reduction of one of the intermediate points in the segment.

Algorithm 1 geometry_generalization (L,C)

```
1: E ← sort_by_x(All points in L)
2: C ← sort_by_x(C)
3: while sweep_line_index < E.size() do
4:     p ← E[sweep_line_index]
5:     n ← curve id of the point p
6:     c ← set of all segments of curve n in A to which point
   p is consecutive in original curve
7:     if |c| = 0 then
8:         s_n1.add(p)
9:         A.add(s_n1)
10:    end if
11:    if |c| = 1 then
12:        s_ni ← c[1]
13:        s_ni.add(p)
14:    end if
15:    if |c| = 2 then
16:        c[1].add(p)
17:        s_ni ← merge(c[1], c[2])
18:        A.add(s_ni)
19:        A.remove(c[1])
20:        A.remove(c[2])
21:    end if
22:    if s_ni.size() > 2 then
23:        remove_point_from_segment(s_ni)
24:    end if
25:    sweep_line_index = sweep_line_index + 1
26: end while
```

The $remove\_point\_from\_segment(segment\ s)$ routine takes a segment and tries to remove points from this segment. This routine will construct triangles from consecutive points. If a triangle does not contain any constraint point or points from other curves then the second point of the triangle can be removed from the segment. If a constraint point 'p' lies inside the triangle, it would mean that removing the second point in the triangle will change the topological relation of p and curve. If a point from one of the other curves lies inside the triangle, the reduction would result in intersection of curves. In no other cases will an intersection of curves occur and hence the constraints for the reductions are always satisfied.

Algorithm 2 remove_point_from_segment(segment s)

```
1: for each consecutive triplet t in s do
2:     min_x ←leftmost_point(t)
3:     max_x ←rightmost_point(t)
4:     cp ←getallPointsBetween(min_x,max_x)
5:     for each Point c in cp do
6:         if not insideTriangle(t,c) then
7:             remove_point(E,t[1])
8:             remove_point(s,t[1])
9:         else
10:            remove_point(s,t[0])
11:        end if
12:    end for
13: end for
```

The implementation of $getallPointsBetween$ returns all the points between min_x and max_x - points from other curves and constraint points. Points from other curves can be obtained by returning all points between min_x and max_x in Event List$E$. For getting constraint points between min_x and max_x any two dimensional metric structure can be used.

Table 1: Results on Different Data Sets for single threaded Implementation

Total Points $\|E\|$	Constraint Points $\|C\|$	Points Removed	Time Taken(ms)
992	26	925	11.934307
1564	127	1432	15.291183
8531	151	7539	52.951826
28014	356	25182	271.408981
28323	1607	23353	251.468033

It may be noted that the only place where a geometric-topology-check is encountered in the whole algorithm is within the insideTriangle routine. This insideTriangle check has been preferred because, unlike a insidePolygon routine, this check is constant time and computationally efficient.

5. PERFORMANCE

The performance of our implementation was tested on a personal computer with Intel i5 quad-core CPU, 8GB RAM running Windows 7 operating system. The data sets used for testing were provided by SIGSPATIAL [1] as a part of GIS CUP 2014. All results were obtained by running implementation on the five data sets provided.

The initial implementation of the algorithm was a naive approach, processing in a single thread. It was observed that as the size of the data increased, the Event List processing took more time and hence there was a necessity to scale down the size of the problem. The running time for the test data sets is shown in Table 1. It may also be noted that the time taken for reduction is low for a set with higher number of constraint points and approximately same number of curve points because constraint violation is checked exhaustively in a window.

An interesting property about the algorithm is that it is highly parallelizable. This concurrency can be achieved by slicing the whole region(from left most point to rightmost point) into 2 or more regions along X-axis and processing these regions individually and simultaneously. Each concurrent instance can process each slice as an independent region and these results can later be merged to obtain the overall simplification. This feature has been used heavily in our implementation for the SIGSPATIAL competition, slicing the whole area into n equal regions along the X-axis and running the MWGS algorithm over these n regions and concurrently.

In fact, it may be noted that the time taken to complete one phase (that is the sweep line crosses the rightmost point in the region) depends on the number of points in each region and hence reducing the number of points for each phase, using region 'slicing', will reduce the time taken. In fact after processing these slices, there are more possibilities for reduction on the edges of these slices, which can be reduced on a

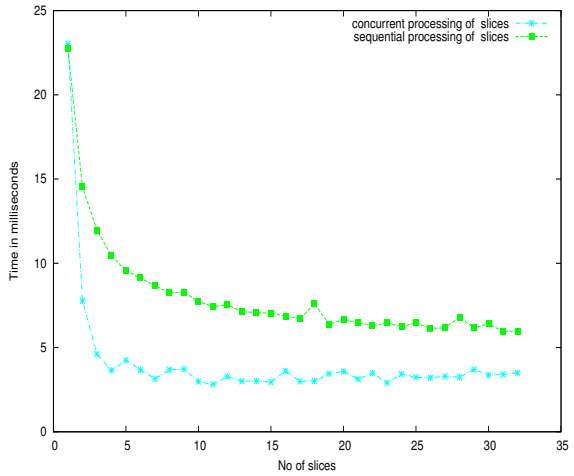

Figure 2: Impact of Parallelizing the algorithm

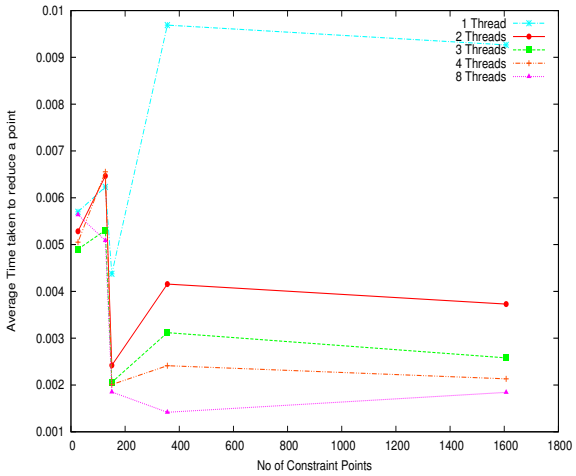

Figure 3: Comparison on Parallelizing the algorithm

separate pass. It is advisable to split the entire region into n regions, where n is the number of cores available for processing. Figure 3 shows the impact of parallelizing code. The average amount of time taken for reducing points reduces as number of threads are increased.

It was observed that I/O was taking considerable time compared to total simplification time. So, in order to optimize I/O, we materialized producer consumer paradigm as, reading file as producer and parsing the curves as consumer. I/O and parsing took almost 60% of the total runtime. Table 2 gives details on the I/O performance before and after implementing the producer consumer paradigm, where T_s denotes time without I/O threading and T_t denotes time with I/O threading. It is interesting that the only geometric operation done in the whole algorithm is 'point-in-triangle' which is highly optimizable unlike a generic 'point-in-polygon' algorithm. Clearly, the parallelized version does better than the sequential version. In fact the performance of parallel version depends on the available cores on the system, hence its advantage stabilizes after it reaches eight threads on an 8 core system.

Table 2: I/O Times for different input sizes

Total Points $\|E\|$	Constraint Points $\|C\|$	T_s(ms)	T_t(ms)
992	26	8.296340	3.137035
1564	127	10.783785	2.643387
8531	151	22.067841	3.836287
28014	356	40.215298	7.873544
28323	1607	46.305616	8.770454

6. RESULT

The implementation of MWGS algorithm could reduce more than 90% of the total number of points in all test cases. In fact the idea of trying to reduce maximum number of point in one pass of the data a good 'number-of-reduced-points to time' ratio. The results for different data sets can be seen in Table 3. From the last two rows of the table, it may be noted that the number of constraint points and curve topology has a major impact on the performance of the algorithm.

Table 3: Overall Performance

Total Points $\|E\|$	Constraint Points $\|C\|$	Points Removed	Time Taken(ms)
992	26	884	4.904054
1564	127	1308	8.561237
8531	151	7538	21.094188
28014	356	25179	87.295281
28323	1607	23336	95.198980

7. CONCLUSIONS AND FUTURE WORK

The paper introduced a new algorithm named MWGS algorithm which can be used for reducing general curves under constraints. The algorithm was shown to be highly parallelizable and was quick and performed exceptionally well on large number of constraint points. Note that the algorithm can be improved by using efficient data structures such as spatial trees to efficiently find points in a window. Even the implementation of parallelized code can be improved by using more robust slicing algorithms that ensure equal load on parallel instances.

8. REFERENCES

[1] Problem statement for sigspatial 2014. `http://mypages.iit.edu/~xzhang22/GISCUP2014/index.php`.

[2] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[3] J. D. W. M Visvalingam. Line generalisation by repeated elimination of the smallest area. *CISRG Discussion Paper Series No 10*, 1992.

[4] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases: With Application to GIS*. Series in Data Management Systems. Morgan Kaufmann Publishers, 2002.

[5] W. Shi and C. Cheung. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43(1):27–44, 2006.