Polygon Detection from a Set of Lines

Alfredo Ferreira Manuel J. Fonseca Joaquim A. Jorge
Department of Information Systems and Computer Science
INESC-ID/IST/Technical University of Lisbon
R. Alves Redol, 9, 1000-029 Lisboa, Portugal
alfredo.ferreira.jr@inesc-id.pt, mjf@inesc-id.pt, jorgej@acm.org

Abstract

Detecting polygons defined by a set of line segments in a plane is an important step in the analysis of vectorial drawings. This paper presents an approach that combines several algorithms to detect basic polygons from a set of arbitrary line segments. The resulting algorithm runs in polynomial time and space, with complexities of $O((N+M)^4)$ and $O((N+M)^2)$ respectively, where N is the number of line segments and M is the number of intersections between line segments. Our choice of algorithms was made to strike a good compromise between efficiency and ease of implementation. The result is a simple and efficient solution to detect polygons from lines.

Keywords

Polygon Detection, Segment Intersection, Minimum Cycle Basis

1. INTRODUCTION

Unlike image processing, where data consist of raster images, the proposed algorithm deals with drawings in vector format, consisting of line segments. This requires completely different approaches, such as described in this paper.

To perform polygon detection from a set of line segments we divide this task in four major steps. First we detect line segment intersections using the Bentley-Ottmann algorithm [13]. Next step creates a graph induced by the drawing, where vertices represent endpoints or proper intesection points of line segments and edges represent maximal relatively open subsegments that contain no vertices. The third step finds the Minimum Cycle Basis (MCB) [16] of the graph induced in previous step, using the algorithm proposed by Horton [12]. Last step constructs a set of polygons based on cycles in the previously found MCB. This is straight-forward if we transform each cycle into a polygon, where each vertex in the cycle represents a vertex in the polygon and each edge in the cycle represents an edge in the polygon.

In sections 2 and 3 we describe the four steps of our method. Section 4 presents the whole algorithm, followed by experimental results in section 5. Finally in section 6 we discuss conclusions and future work.

2. INTERSECTION REMOVAL

In a vector drawing composed by a set of line segments there might exist many intersections between these segments. To detect polygonal shapes we have to remove proper segment intersections, thus creating a new set of line segments in which any pair of segments share at most one endpoint.

2.1. Finding line segment intersections

The first step of our approach consists in detecting all M intersections between N line segments in a plane. This is considered one of the fundamental problems of Computational Geometry and it is known that any algorithm, within the model of algebraic decision tree, have a lower bound of $\Omega(N \log N + M)$ time to solve it [3, 5].

In [1] Balaban proposes two algorithms for finding intersecting segments, a deterministic and asymptotically optimal for both time $O(N \log N + M)$ and space $O(N)$ algorithm and a simpler one that can perform the same task in $O(N \log^2 N + M)$-time. Before that, Chazelle and Edelsbrunner [5] reached a time optimal algorithm $O(N \log N + M)$ with space requirement of $O(N + M)$. The randomized approach devised by Clarkson and Shor [6] produced a algorithm for reporting all intersecting pairs that requires $O(N \log N + M)$ time and $O(N)$ space.

In 1979 Bentley and Ottmann proposed an algorithm that solved this problem in $O((N+M) \log N)$ time and $O(N+M)$ space [13]. This algorithm is the well-known Bentley-Ottmann algorithm and after more than 20 years it is still widely adopted in practical implementations because it is easy to understand and implement [15, 11]. In realizing that this is not the most complex part of our aproach, we decide to use the Bentley-Ottmann algorithm, since its complexity is pretty acceptable for our purposes and its published implementations are quite simple.

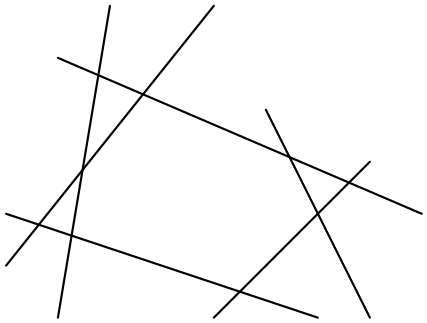

Figure 1. Set Φ of line segments

2.2. Removing line segment intersections

The next step of our approach is to remove all proper intersections between line segments, dividing each intersected segment in sub-segments without proper intersections, only sharing endpoints. In order to find and remove intersections, performing at once the first two steps of our approach, we use a robust and efficient implementation of the Bentley-Ottmann algorithm, described by Bartuschka, Mehlhorn and Naher [2] that computes the planar graph induced by a set of line segments. Their implementation, represented in this paper by COMPUTE-INDUCED-GRAPH, computes the graph G induced by set Φ in $O((N+M)\log N)$ time. Since this algorithm is quite long we choose not to present it here. We refer our readers to [2] for a detailed description.

In this implementation the vertices of G represent all endpoints and proper intersection points of line segments in Φ, and the edges of G are the maximal relatively open subsegments of lines in Φ that do not contain any vertex of G. The major drawback of this implementation lies in that parallel edges are produced in the graph for overlapping segments. We assume that Φ contains no such segments. Considering, for example, the set Φ shown in Figure 1, COMPUTE-INDUCED-GRAPH will produce the graph G, depicted in Figure 2, where each edge represents a non-intersecting line segment.

3. POLYGON DETECTION

Detecting polygons is similar to finding cycles on the graph G produced in the previous step.

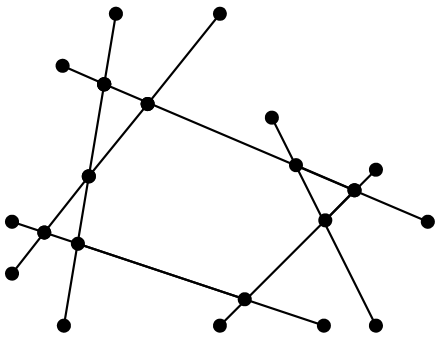

Figure 2. Graph G induced by Φ

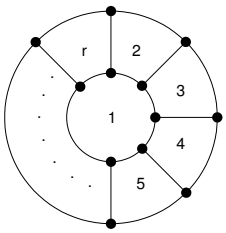

Figure 3. A planar graph with a exponential number of cycles

3.1. All Cycles of a Graph

The first known linear-time algorithm for listing all cycles of a graph was presented by Syslo [16]. This algorithm requires $O(V)$ space and $O(V \times C)$ time, where V is the number of vertices and C the number of cycles in G. Later Dogrusöz and Krishnamoorthy proposed a vector space algorithm for enumerating all cycles of a planar graph that runs in $O(V^2 \times C)$ time and $O(V)$ space [8]. Although asymptoticaly slower, this algorithm is much simpler than Syslo's and is amenable to parallelization. Unfortunately, the total number of cycles in a planar graph can grow exponentially with the number of vertices [14]. An example of this situation is the graph presented in Figure 3. In this case, the number of cycles, including the interior region numbered 1, is $O(2^r)$ with $r = k/2 + 1$, where k is the number of vertices, since one can choose any combination of the remaining regions to define a cycle [8]. This is why it is not very feasible to detect all polygons that can be constructed from a set of lines. In this paper, we choose just to detect the minimal polygons, those that have a minimal number of edges and cannot be constructed by joining other minimal polygons.

3.2. Minimum Cycle Basis of a Graph

Considering that we just want to detect the minimal polygons this can be treated as searching for a Minimum Cycle Basis (MCB). So, the second step of our approach consists in obtaining a MCB of graph G. A cycle basis is defined as a basis for the cycle space of G which consists entirely of elementary cycles. A cycle is called elementary if it contains no vertex more than once. The dimension of the cycle space is given by the *cyclomatic number* $\nu = E - V + P$ [9, 4], where E is the number of edges and V the number of vertices in G and P is the number of connected components of G.

Horton presented the first known polynomial-time algorithm to find the shortest cycle basis of a graph, which runs in $O(E^3V)$ time [12] or in $O(E^4)$ on simple planar graphs [10], which is the case. While assimptotically better solutions have been published in the literature, the Bentley-Ottmann algorithm is both simple and usable for our needs. The pseudo-code of this algorithm is listed in MINIMUM-CYCLE-BASIS and shortly described bellow. A further detailed description of this algorithm and concepts behind it can be found in [12].

The ALL-PAIRS-SHORTEST-PATHS finds the shortest

MINIMUM-CYCLE-BASIS(G)
1 $\Gamma \leftarrow$ *empty set*
2 $\Pi \leftarrow$ ALL-PAIRS-SHORTEST-PATHS(G)
3 **for each** v **in** VERTICES(G)
4 **do for each** (x, y) **in** EDGES(G)
5 **do if** $\Pi_{x,v} \cap \Pi_{v,y} = \{v\}$
6 **then** $C \leftarrow \Pi_{x,v} \cup \Pi_{v,y} \cup (x, y)$
7 add C to Γ
8 ORDER-BY-LENGTH(Γ)
9 **return** SELECT-CYCLES(Γ)

paths between all pairs of vertices in graph G and can be performed in $O(V^3)$ time and $O(V^2)$ space using Floyd-Warshall or Dijkstra algorithms [7]. ORDER-BY-LENGTH orders the cycles by ascending length and can be implemented by any efficient sorting algorithm. This is a non-critical step because it has a $O(V\nu \log V)$ upper bound in time complexity, which is insignificant in comparision with other steps of this algorithm.

In SELECT-CYCLES we use a greedy algorithm to find the MCB from Γ set of cycles. To do this Horton [12] suggests representing the cycles as rows of a 0-1 incidence matrix, in which columns correspond to the edges of the graph and rows are the incidence vectors of each cycle. Gaussian elimination using elementary row operations over the integers modulo two can then be applied to the incidence matrix, processing each row in turn, in ascending order of the weights of cycles, until enough independent cycles are found.

This step dominates the time complexity from other steps, since it takes $O(E\nu^2V)$ time. Knowing that G is always a simple planar graph we can conclude that as a whole the MINIMUM-CYCLE-BASIS algorithm has a worst case upper bound of $O(E\nu^2V) = O(E^3V) = O(E^4)$ operations and a space requirements of $O(V^2)$.

Figure 4 shows an example of Γ, the set of cycles resulting from applying the MINIMUM-CYCLE-BASIS to graph G shown in Figure 2.

3.3. Polygon construction

The last step of our approach consists in constructing a set Θ of polygons from the MCB. An algorithm to perform this operation can easily run in $O(CV)$ time, where C is number of cycles in MCB. Such an algorithm is listed in POLYGONS-FROM-CYCLES which returns a set Θ of polygons.

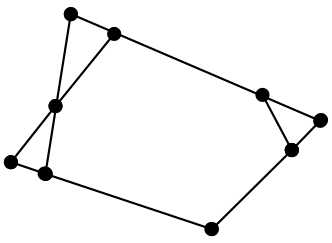

Figure 4. Shortest cycle basis Γ of graph G

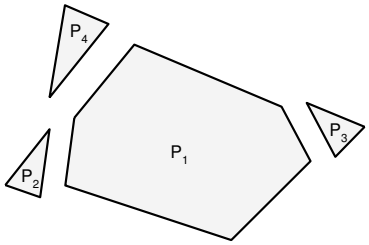

Figure 5. Set Θ of polygons detected from Φ

POLYGONS-FROM-CYCLES(Γ)
1 $\Theta \leftarrow$ *empty set*
2 **for each** C **in** Γ
3 **do** $P \leftarrow$ *new polygon*
4 **for each** v **in** VERTICES(V)
5 **do** add vertex v to P
6 add polygon P to Θ
7 **return** Θ

Figure 5 illustrates the resulting set Θ of polygons generated by applying POLYGONS-FROM-CYCLES to Γ depicted in Figure 4.

4. ALGORITHM OUTLINE

We can now outline DETECT-POLYGONS. This algorithm is able to detect a set Θ of polygons from a initial set Ψ of line segments. To perform this task we pipeline the algorithms referred in previous sections for line segment intersection removal, MCB finding and cycle-to-polygon conversion.

DETECT-POLYGONS(Ψ)
1 $G \leftarrow$ COMPUTE-INDUCED-GRAPH(Ψ)
2 $\Gamma \leftarrow$ MINIMUM-CYCLE-BASIS(G)
3 $\Theta \leftarrow$ POLYGONS-FROM-CYCLES(Γ)
4 **return** Θ

As refered in section 2.2, COMPUTE-INDUCED-GRAPH runs in $O((N + M) \log N)$ time and $O(N + M)$ space. The SHORTEST-CYCLE-BASIS runs in $O(V^4)$ operations and has a space requirement of $O(V^2)$, making this the critical step in the complexity of this algorithm, since the POLYGONS-FROM-CYCLES just needs $O(CV)$ time.

Since the number V of vertices in the graph is no greater than the sum of line endpoints ($2 \times N$) with detected intersections M, we can then conclude that the proposed algorithm has time and space complexities of $O(V^4) = O((N + M)^4)$ and $O(V^2) = O((N + M)^2)$, respectively.

5. EXPERIMENTAL RESULTS

The algorithm proposed in this paper was implemented in C++ and tested in a Intel Pentium III 1GHz 512MB RAM computer running Windows XP . We tested the algorithm with sets of line segments created from simple test drawings, technical drawings of mechanical parts and hand-sketched drawings. Table 1 presents the results obtained from these tests.

Lines	Intersections	Nodes	Edges	Time (ms)
6	9	21	24	10
36	16	58	68	50
167	9	169	177	3986
286	47	389	376	8623
518	85	697	679	36703
872	94	1066	10050	128995
2507	10	2407	2526	1333547

Table 1. Results of algorithm tests

Based on these results we conclude that performance is acceptable for on-line processing in sets with less than three-hundred lines like hand-sketches or small-size technical drawings. If the line set have about 2500 lines the algorithm will take more than twenty minutes to detect the polygons. Still this remains a feasible solution for batch processing of medium-size technical drawings.

6. CONCLUSIONS and FUTURE WORK

The proposed algorithm is used for polygon detection in vector drawings to create descriptions based on spatial and topological relationships between polygons. Other use is detecting planar shapes in sketches. Both applications have been implemented as working prototypes used for shape retrieval and architectural drawing from sketches.

The algorithm presented here detects in polynomial time and space, all minimal polygons that can be constructed from a set of line segments. This approach uses well-known and simple to implement algorithms to perform line segment intersection detection and to find a MCB of a graph, instead of using more efficient but less simpler methods.

Indeed there is considerable room for improvement in the presented algorithm, namely through the use of more recent, complex and efficient algorithms. Further work may be carried out regarding the detection and correction of rounding errors resulting from finite precision computations.

7. ACKNOWLEDGMENTS

We thank to Professor Mukkai S. Krishnamoorthy from Rensselaer Polytechnic Institute, New York, for his very helpful suggestions.

This work was funded in part by the Portuguese Foundation for Science and Technology, project 34672/99 and the European Commission, project SmartSketches IST-2000-28169.

References

[1] Ivan J. Balaban. An optimal algorithm for finding segment intersections. In *Proceedings of the 11th Annual ACM Symposium Comp. Graph.*, pages 211–219. ACM, 1995.

[2] Ulrike Bartuschka, Kurt Mehlhorn, and Stefan Naher. A robust and efficient implementation of a sweep line algorithm for the straight line segment intersection. In *Proceedings of Workshop on Algorithm Engineering*, pages 124–135, Venice, Italy, September 1997.

[3] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium Theory of Computing*, pages 80–86. ACM, 1983.

[4] Augustin-Louis Cauchy. Recherche sur les polyèdres. *J. Ecole Polytechnique*, 9(16):68–86, 1813.

[5] Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39:1–54, 1992.

[6] Ken Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, ii. *Discrete and Computational Geometry*, 4:387–421, 1989.

[7] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, McGraw-Hill, 2nd. edition, 1990.

[8] U. Dogrusöz and M. Krishnamoorthy. Cycle vector space algorithms for enumerating all cycles of a planar graph. Technical Report 5, Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, New York 12180 USA, January 1995.

[9] Leonhard Euler. Elementa doctrinae solidorum. *Novi Commentarii Academiae Scientiarum Petropolitanae*, 4:109–140, 1752.

[10] David Hartvigsen and Russel Mardon. The all-pairs minimum cut problem and the minimum cycle basis problem on planar graphs. *SIAM Journal on Computing*, 7(3):403–418, August 1994.

[11] John Hobby. Practical segment intersection with finite precision output. *Computational Geometry: Theory and Applications*, 13(4), 1999.

[12] J.D.Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM Journal on Computing*, 16(2):358–366, April 1987.

[13] J.L.Bentley and T.Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, pages 643–647, 1979.

[14] Prabhaker Mateti and Narsingh Deo. On algorithms for enumerating all circuits of a graph. *SIAM Journal on Computing*, 5(1):90–99, March 1976.

[15] Joseph O'Rourke. *Computational Geometry in C*, chapter Section 7.7 "Intersection of Segments", pages 264–266. Cambridge University Press, 2nd edition, 1998.

[16] Maciej M. Syslo. An efficient cycle vector space algorithm for listing all cycles of a planar graph. *SIAM Journal on Computing*, 10(4):797–808, November 1981.