Computational Geometry: Its objectives and relation to GIS*

Marc van Kreveld
Institute of Information and Computing Sciences
Utrecht University
marc@cs.uu.nl

Abstract

We overview the basics of computational geometry, discuss its relation to GIS, and analyze in which directions research can develop that lies in the intersection of GIS and computational geometry.

1 Introduction

The research area of computational geometry deals with the design and analysis of algorithms for geometric problems. The analysis of algorithms involves understanding how efficiently an algorithm solves a problem. An algorithm is considered efficient if it scales well in the input size, that is, if the time needed to solve an instance with a certain input takes a few minutes, then the time needed on an input that is ten times as large does not get out of hand completely. One of the main objectives of computational geometry is finding the most efficient algorithms for all sorts of geometric problems. These problems can be abstract, or motivated from areas like computer graphics, robot motion planning, and GIS.

We will review the main concepts from algorithms design and computational geometry. These include efficiency analysis, intractability, output-sensitive algorithms, and approximation algorithms. Many basic problems of computational geometry have a direct or indirect use to GIS. For instance, line segment intersection by plane sweep solves map overlay, the most important GIS analysis task, Voronoi diagrams are helpful in neighborhood analysis, Delaunay triangulations are widely used for terrain modeling, and geometric data structures help with efficient spatial indexing in large spatial data sets.

We will next review the limitations for the applicability of computational geometry in GIS. The most important remaining limitation is the fact that many GIS tasks are not easy to formalize into simple, geometric, abstract problems. The objectives are often not clear. We will list a number of problems of this type to indicate where future research can be directed.

2 Computational geometry

The area of computational geometry is about the design and analysis of algorithms for geometric problems [13]. For example, the closest pair problem for a set of points in the plane asks for an efficient method to determine which two points of the set are closest to each other. The input to the problem is some set P of points $\{p_1, \ldots, p_n\}$ specified by their coordinates. The size of the input is the number of points given, here denoted by n. A solution (an algorithm) is a sequence of steps that will find the closest pair of points. An algorithm must be correct, that is, it must determine the closest pair no matter what set P of points in the plane is given.

*This research was supported by the Netherlands Organization of Scientific Research (NWO) as part of the FOCUS grant GADGET.

Efficiency. The efficiency of an algorithm refers to the amount of time needed by the algorithm. Here time is measured in the number of steps. Obviously, it depends on the input set P how much time will be needed. Nearly always, the larger the input set, the more time will be needed by the algorithm. Therefore, we will express the efficiency of an algorithm as a function of the input. For example, the number of steps (assignments, comparisons, additions, ...) could be $4n^2 + 6n + 9$ for some input of n points. Since we cannot express the efficiency for every possible input, we will analyze the most difficult input of size n, so that we know for sure that the algorithm will certainly finish its computations in at most that many steps if it is given an input of size n. This is called worst case analysis.

A possible algorithm for the closest pair problem is the following:

```
q1 = p[1]
q2 = p[2]
for i = 1 to n-1
do for j = i+1 to n
   do if (distance(p[i],p[j]) < distance(q1,q2)) {
          q1 = p[i]
          q2 = p[j]
      }
```

When we analyse the efficiency, we will notice that the number of steps taken by this algorithm depends on n as a quadratic function. If we take an input set of size $2n$ instead of n, the algorithm will roughly take four times as long. Generally we are not interested in the precise number of steps of the most difficult input, but only this scaling behavior. We say that an algorithm takes $O(n^2)$ time (order of n squared time) if the dominant term in the number of steps taken is a quadratic term. If the dominant term is linear in n, an algorithm takes $O(n)$ time. For inputs twice the size, such an algorithm will take roughly twice as long. For difficult problems, we may have to design algorithms that take much more time, like $O(2^n)$. In that case, the algorithm takes twice as long if the input size is only one point more. Such algorithms with exponential running time behavior should be avoided because they simply take too much time in their execution. However, geometric problems exist for which no more efficient algorithms than exponential ones are known.

We can now compare the efficiency of two different algorithms that solve the same problem. The one with a smaller function in $O(..)$ notation is the better algorithm. If the input is large enough, it will be faster than the other. The main goal in computational geometry is to determine the most efficient algorithm for each geometric problem. For example, the most efficient algorithm for the closest pair problem takes $O(n \log n)$ time, and is better than the simple algorithm given above.

Basic geometric problems. The area of computational geometry has provided efficient algorithms for many different geometric problems. Well-known are (see Figure 1):

- Convex hull problem: for a set of points, determine the smallest convex set that contains all.
- Line segment intersection: for a set of line segments, determine all intersections.
- Voronoi diagram computation: for a set of points, determine the subdivision of the plane into cells such that inside some cell, one and the same point of the set is closest.
- Delaunay triangulation: for a set of points, determine a planar subdivision by creating edges between the input points in such a way that no two edges intersect, all faces are triangles, no more edges can be added with the given constraints, and no circumcircle of any triangle contains an input point in its interior.
- Minkowski sum: for two simple polygons P and Q, compute the shape that consist exactly of the sum of all points of P and all points of Q, where sum is interpreted as the vector sum.

- Rectangular range search: for a set of points in the plane, design a data structure on those points, such that for every axis-parallel query rectangle, all points in the data structure that lie in the query rectangle can be reported efficiently. Algorithms are needed for the construction of the data structure and for the execution of a query.

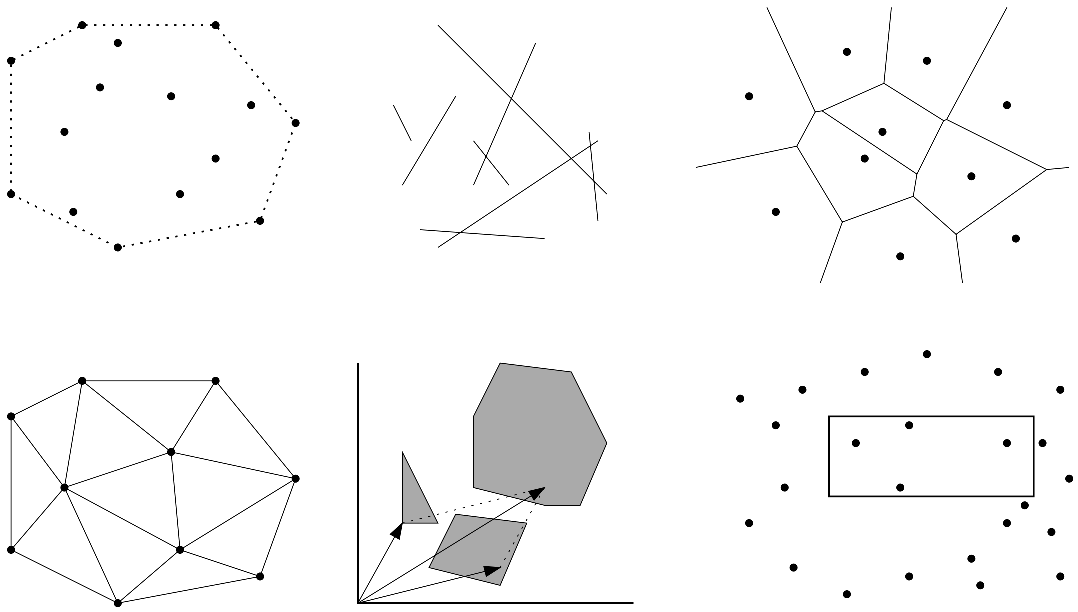

Figure 1: Six basic problems in computational geometry: convex hull, line segment intersection, Voronoi diagram, Delaunay triangulation, Minkowski sum, and rectangular range search.

The problems are illustrated in Figure 1. Such problems are closely related to various well-known GIS operations like map overlay and buffer computation. Map overlay must find all intersections of the line segments that appear in the two thematic layers of which the overlay is constructed. The buffer of a polygon is the same as the Minkowski sum of that polygon with a disk centered at the origin.

Output-sensitive algorithms. In nearly all cases the time needed for an algorithm depends on the input size. In some cases, we also want to take the output size into account. For example, for the line segment intersection problem, any algorithm must take at least quadratic time, because for a set of n line segments, it can be that every pair of segments intersects in a different point. In this case, any correct algorithm must report all of these quadratically many intersection points. However, we would like to develop an algorithm that does not take quadratic time if there are only linearly many intersection points. To capture this in the efficiency analysis, we express the running time of an algorithm not only in the input size n, but also in the output size, denoted by k. The best known algorithm for line segment intersection takes $O(n \log n + k)$ time if there are k intersection points [5, 10]. When k is no more than $O(n \log n)$, the running time comes down to $O(n \log n)$, which is much better than quadratic. The well-known Bentley-Ottmann plane sweep algorithm for line segment intersection takes $O(n \log n + k \log n)$ time [6], which is slightly less efficient, but also output-sensitive.

Intractable problems. If for a problem no algorithm is known that runs in at most a polynomial amount of time (so worse than $O(n^c)$ for any constant c), then the problem is called intractable [17]. The best known algorithm usually needs an exponential amount of time in such a case. A well-known problem of this type is the Euclidean traveling salesperson problem: given a set of n point in the plane, find a shortest tour that visits all points. There are also various cartographic problems that are intractable. For example, many versions of label placement—place as many labels as possible on a map without any overlap—are intractable [16, 27].

Approximation algorithms. If the best known algorithm takes an amount of time that is more than acceptable, then it may be possible to design an approximation algorithm [7]. This is an algorithm that attempts to optimize a particular target, and provably achieves this goal to within a certain factor. An approximation algorithm is useful if it is much more efficient than the best known algorithm that gives an optimal solution, and the approximation factor is good. For example, an algorithm exists that can place at least half as many labels on a map as the maximum possible, and which runs in $O(n \log n)$ time [2]. Also, one can guarantee the placement of at least 2/3 as many labels as the optimum with an algorithm that takes $O(n^5)$ time. Both algorithms are approximation algorithms. A heuristic is like an approximation algorithm, but a heuristic does have a guarantee on how well it achieves its goal.

3 Geometric GIS problems

Many computational problems that must be solved in a GIS are geometric of nature. They occur in all stages of the GIS data cycle. Geometric algorithms are useful in data correction (after data acquisition and input), in data retrieval (through queries), data analysis (like map overlay and geostatistics), and data visualization (for maps and animations). We list a few GIS problems and indicate how computational geometry can provide part of a solution. For a more extensive overview, see [26].

Automatic data correction after manual digitizing. Problems that can occur when digitizing a paper map by hand include spikes, overshoots, undershoots, forgotten boundaries, and

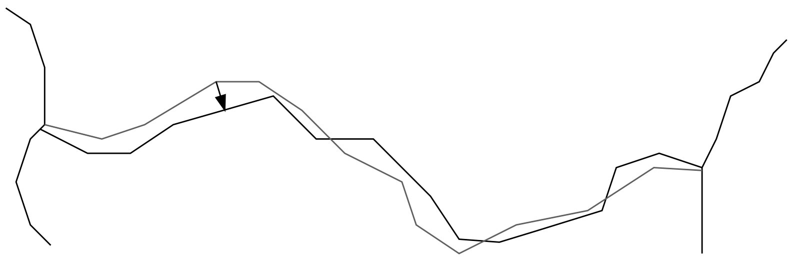

Figure 2: Doubly digitized line and Hausdorff distance.

doubly digitized boundaries. Let us look at the last problem. To automatically detect, among a set of boundaries, whether two are nearly the same and therefore probably doubly digitized, we need to determine the similarity of any two boundaries. There are several similarity measures that can be used, like the Hausdorff distance and the Frechet distance. The Hausdorff distance between two objects is defined as the maximum of all distances from any point on the one object to the closest point on the other object, see Figure 2. With techniques from computational geometry, the Hausdorff distance of two boundaries with n vertices together can be determined in $O(n \log n)$ time [3].

Windowing. When a subset of a data set is selected for visualization or for further analysis, this data must be retrieved from the data base. If the data is organized into a query structure that allows efficient windowing queries, the subset can be extracted without having to go through the whole data set. Usually R-trees are used for this [19, 23]. They were introduced in the database community, but later, in computational geometry, several variations have been described that provably achieve a good query time, even in worst case situations [1]. This was not known yet for previous R-trees.

Spatial data analysis. In spatial data analysis, patterns, clusters, and outliers must be detected [21, 24]. In a set of points, clusters are groups of points close together, and outliers are points that do not fit in the global locations of the points. Many different definitions of outliers exist. To call a point an outlier if it is further than some distance from all other points is not a good idea. If two points lie close together but far from the other points, they are also outliers, but they would not be detected. It is slightly better to deem a point an outlier if, say, its fifth-closest point is further than some distance away. Using higher order Voronoi diagrams [4], the fifth-closest point for every point can be determined efficiently.

Cartographic generalization. In the process of cartographic generalization of maps, map features may have to be displaced, or aggregated into larger objects [22]. Aggregation is only possible for features that are close and are of the same type, like two buildings or two forest patches. It is important know the proximity of the features to detect the need for displacement, and the possibility for aggregation. Ware et al. [28] use the constrained Delaunay triangulation to capture proximity, and at the same time, the constrained Delaunay triangulation provides triangles that lie between two objects that can be aggregated, and can be added to join them. The most efficient method to compute the constrained Delaunay triangulation was developed by Chew and runs in $O(n \log n)$ time [11].

4 Limitations of the use of computational geometry for GIS

There are several reasons why computational geometry is not as useful to GIS as it could be. A first reason is that the algorithms developed in computational geometry are often rather complex, and they require a large effort to implement. This is partially because they must be provably efficient for all possible inputs, and partially because geometric degeneracies must be dealt with. A second problem with the usefulness comes from the efficiency analysis, which is based on worst-case inputs to the algorithm. The theoretical efficiency of any algorithm depends on the time needed on the data set for which it is slowest. However, such worst-case data sets may be rather artificial, and never appear in real-world applications. The third problem with the applicability of computational geometry lies in the abstraction of the original problem. Many problems that come from practice require several criteria to be met simultaneously, or criteria that should be met "at least to some extent". For example, when constructing a cartogram, maintaining recognizability as much as possible is an important issue. This leads to vague problem statements, and the general practice in computational geometry is to consider well-defined, simple-to-state problems mostly.

In the last decade, several of the problems mentioned above have been addressed. Nowadays, there are software libraries with geometric primitives and algorithms, alleviating the effort to implement [15]. They also solve the robustness issues largely. The second issue has been addressed through the development of relatively simple algorithms are inefficient in the worst case, but provably efficient under realistic assumptions on the input [12]. Unfortunately, the third issue is still largely unresolved. Computational problems where various, partially conflicting criteria must be respected to some extent, are typical in the more advanced GIS functionalities. The correct problem formulation becomes the first concern. Only after a study of appropriate formalizations has been done, algorithmic solutions can be studied. At the same time, implementation and testing are needed to evaluate formalizations, and refine them. We list a few of such problems next.

Spatial interpolation in mixed environments. Many geographic variables are measured at various locations in the field, and then interpolated to get a complete picture. For example, noise level measurements are done at point locations. It is possible to apply any of the many spatial interpolation techniques, like by triangulation, moving windows, and Kriging [9], but one should take into account that noise travels differently over water, over fields, and through forests.

Therefore, the coordinates of the points and the Euclidean distances are not the only factors that determine how to interpolate. New interpolation models and algorithms should be developed for such cases.

Spatio-temporal data mining. Spatio-temporal data mining is about discovering patterns in spatio-temporal data sets [21, 25]. Such data can be sequences of (location, time) pairs for moving entities. Patterns include grouping of a sufficiently large subgroup at a moment in time (flocking), or moving in a particular common direction for a subgroup [18, 20]. Other patterns of interest that have not been defined and investigated properly are patterns like recurring flocking, where a subgroup comes together more than once.

Specialized maps. In the area of quantitative mapping, many interesting types of map exist, like cartograms, flow maps, and dot maps [14]. Such maps generally need to satisfy various constraints at once. For example, cartograms (of the contiguous area type) need to show the countries with the proper size, the adjacencies of the countries must be maintained, the original shape must be recognizable, and the locations of the countries must be preserved. These constraints are conflicting, and fulfilling one may make another be unacceptable. It is unclear what the proper balance between the constraints is, and therefore it is difficult to develop fully automated methods to compute high-quality maps of this type.

Terrain reconstruction. Digital elevation modeling deals with generating a model for a terrain in a GIS that is realistic [29]. The model is built on an initial data set, which can consist of points with elevations, or digitized contour lines with elevations, possibly in combination with drainage and slope information. Also, there may be high-level information like no abrupt slope changes anywhere. To generate a terrain that fits with the input data, the high-level information, and is realistic in the sense that no artifacts appear, is again a problem where several conflicting criteria have to be taken into account. The modeling problem and corresponding algorithms are a continuing direction of future research that has hardly been addressed from the computational geometry perspective.

Other typical GIS aspects that should be considered in the context of computational geometry as well are computations with imprecise coordinates, computations that deal with (spatial) scale of geographical phenomena and processes, and indeterminate boundaries of features [8]. Examples of indeterminate boundaries are boundaries of mountain ranges when there is a transition zone with the lower surrounding land, and the definition of the extent (or size) of an island when there are tidal changes.

5 Conclusions

Research in computational geometry has proved to be very useful to many computational problems that appear in GIS. In order to extend its usefulness, more complex problems need to be addressed. Often, the precise conditions and constraints that characterize a good solution have not been formulated, because this seems to be difficult. Attempts should be made at devising good formalizations, which can then be tested experimentally. The objective of trying several different formalizations is to find out which ones give the most useful results in practice. If a formalization is useful, research in computational geometry can help to design efficient algorithms. It can be that a formalization leads to a problem that is computationally intractable. In that case, either a different formalization must be designed, or an approximation algorithm, although it is possible that heuristics are satisfactory in practical cases. Whether a particular algorithm is efficient enough also depends on the application. If the computational problem must be solved on-line, while the user of a GIS is waiting, efficiency is much more important than for off-line problems, like computation of maps to be published in atlasses.

References

[1] P. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H.J. Haverkort. Box-trees and R-trees with near-optimal query time. *Discrete & Computational Geometry*, 28:291–312, 2002.

[2] P.K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11:209–218, 1998.

[3] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell.*, 13:251–266, 1995.

[4] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[5] Ivan J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[6] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[7] M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 296–345. PWS Publishing Company, Boston, MA, 1997.

[8] P.A. Burrough and A.U. Frank, editors. *Geographic Objects with Indeterminate Boundaries*, volume 2 of *GISDATA*. Taylor & Francis, London, 1996.

[9] P.A. Burrough and R.A. McDonnell. *Principles of Geographical Information Systems.* Oxford University Press, New York, 1998.

[10] Bernard Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[11] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[12] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 294–303, 1997.

[13] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[14] Borden D. Dent. *Cartography – thematic map design.* Wm. C. Brown Publishers, 1996.

[15] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.

[16] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[18] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *GIS 2004: Proc. of the 12th ACM Sympos. on Advances in GIS*, pages 250–257, 2004.

[19] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf. Principles Database Systems*, pages 47–57, 1984.

[20] P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In P.F. Fisher, editor, *Developments in Spatial Data Handling: 11th Int. Sympos. on Spatial Data Handling*, pages 201–215, 2004.

[21] H.J. Miller and J. Han, editors. *Geographic Data Mining and Knowledge Discovery.* Taylor & Francis, London, 2001.

[22] J.-C. Müller, J.-P. Lagrange, and R. Weibel, editors. *GIS and Generalization – Methodology and Practice*, volume 1 of *GISDATA*. Taylor & Francis, London, 1995.

[23] Jürg Nievergelt and Peter Widmayer. Spatial data structures: Concepts and design choices. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 725–764. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[24] D. O'Sullivan and D.J. Unwin. *Geographic Information Analysis.* Wiley, 2003.

[25] J.F. Roddick and K. Hornsby, editors. *Temporal, Spatial, and Spatio-Temporal Data Mining – First International Workshop TSDM 2000.* Number 2007 in Lecture Notes in Artificial Intelligence. Springer, 2001.

[26] M. van Kreveld. Geographic Information Systems. In J.E. Goodmann en J. O'Rourke, editor, *Handbook of Discrete and Computational Geometry*, chapter 58, pages 1293–1314. Chapman & Hall/CRC, Boca Raton, 2004.

[27] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13:21–47, 1999.

[28] J.M. Ware, C.B. Jones, and G.L. Bundy. A triangulated spatial model for cartographic generalisation of areal objects. In *Proceedings of COSIT*, pages 173–192, 1995.

[29] R. Weibel and M. Heller. Digital terrain modelling. In D. J. Maguire, M. F. Goodchild, and D. W. Rhind, editors, *Geographical Information Systems – Principles and Applications*, pages 269–297. Longman, London, 1991.