Geo-information and computational geometry

Peter J.M. van Oosterom and Marc J. van Kreveld (Editors)

Geo-information and computational geometry

Peter J.M. van Oosterom and Marc J. van Kreveld (Editors)

NCG Nederlandse Commissie voor Geodesie Netherlands Geodetic Commission 44

Delft, September 2006

Geo-information and computational geometry
Peter J.M. van Oosterom and Marc J. van Kreveld (Editors)
Nederlandse Commissie voor Geodesie Netherlands Geodetic Commission 44, 2006
ISBN-10: 90 6132 299 5
ISBN-13: 978 90 6132 299 3

Published by: NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission, Delft, The Netherlands
Printed by: Optima Grafische Communicatie, Optima Graphic Communication, Rotterdam, The Netherlands
Cover illustration: Axel Smits

NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission
P.O. Box 5058, 2600 GB Delft, The Netherlands
T: +31 (0)15 278 28 19
F: +31 (0)15 278 17 75
E: info@ncg.knaw.nl
W: www.ncg.knaw.nl

The NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission is part of the Royal Netherlands Academy of Arts and Sciences (KNAW)

Contents

Editorial

Geographic Information Science (GIS) is a multi-disciplinary research area. Contributions from the spatial sciences come from geodesy, geography, and cartography, whereas contributions from computer science come from databases, artificial intelligence, and computational geometry. Furthermore, very different research areas where GIS are used, like spatial planning, archaeology, geology, civil engineering and biology, also perform applied research in GIS.

Computational geometry is all about doing geometric computations by the computer. For this, (additional) theory is being developed based on the foundation of mathematics. Data structures and algorithms are developed to solve geometric problems often with proven worst-case (and some times also with other) time and memory bounds. Computational geometry techniques can be, and are applied in many different domains: vision, path planning, gaming, graphics, robotics, medical image processing, and of course also in GIS.

The seminar 'Geo-Information and Computational Geometry' of the Netherlands Geodetic Commission (jointly organized with Geo Informatie Nederland) was devoted to the relationships between GIS and computational geometry. In a GIS, computations with coordinates are needed, whereas computational geometry is about developing methods to do so. The seminar aimed to improve the understanding of the two research fields, so that interaction in the future is further strengthened. We hope that the study day has resulted in lasting new contacts between all people in the Netherlands with an interest in geometric computing in GIS.

The contributions reflect the diversity of the possible interactions between computational geometry and GIS. The topics of the contributions range from overviews of relevant techniques and tools to solving specific spatial problems in either the object-based (vector) or field-based (raster) domain. This publication is a reflection of the different seminar contributions.

The first paper 'Computational Geometry: Its objectives and relation to GIS' is by Marc van Kreveld (Utrecht University). The analysis of algorithms involves understanding how efficiently an algorithm solves a problem. One of the main objectives of computational geometry is finding the most efficient algorithms for all sorts of geometric problems. He introduces the main concepts and ideas in computational geometry, including efficiency analysis, intractability, output-sensitive algorithms, and approximation algorithms. The basic problems of computational geometry all have a direct or indirect use to GIS. He also indicates why computational geometry is not as useful to GIS as it could be (complicated algorithms, focus on worst-case efficiency, and on well-defined, simple to state problems) and how this is currently improving (available software libraries, simpler algorithms provably efficient under realistic assumptions).

Mark de Berg (TU Eindhoven) addresses one of the issues to make computational geometry techniques more applicable in practice, namely the handling of large data sets that do not fit in main memory (as often more or less implicitly assumed in the description of many data structures and algorithms). In his paper 'I/O- and Cache-efficient Algorithms for Spatial Data' he explains how the hierarchical memory consisting of a disk, main memory, and several levels of cache should be included in data structure and algorithm design. The difference between the times to access these different levels of memory is quite large: the disk is typically about 100,000 times slower than accessing the main memory. In the paper some of the recent results that have been obtained on I/O- and cache-efficient algorithms are discussed with focus on spatial data.

One specific data structure, based on quad-edges, and applied to creating and editing three-dimensional models, is described by Christopher Gold and Rebecca Tse (University of Glamorgan, UK) in their paper 'Quad-Edges and Euler Operators for Automatic Building Extrusion Using

LiDAR Data' (LIght Detection And Ranging). The long-term research objective for their models is to integrate man-made objects with the landscape, so that topological properties, such as connectedness, may be used in applications such as flood modeling. Man-made objects such as buildings, as well as terrain elevation, should be extracted directly from LiDAR data. Their model is a triangle-based boundary description of the relevant objects and earth surface. The model creation and local modifications (updates) is performed on the Quad-Edge data structure by using Euler operators. These operators permit various extrusion operations as well as the manual insertion of bridges and tunnels.

A description of the use computational geometry tools used to solve a few specific cartographic problems is given by Bettina Speckmann (TU Eindhoven) in her paper 'Algorithms for cartograms and other specialized maps'. Cartograms are a useful and intuitive tool to visualize statistical data about a set of regions like countries, states or counties. The size of a region in a cartogram corresponds to a particular geographic variable and therefore the regions generally cannot keep both their shape and their adjacencies. A good cartogram, however, preserves the recognizability in some way. The paper gives a short overview of cartogram algorithms, and focuses in particular on the computation of rectangular cartograms. In a rectangular cartogram each region is represented by a rectangle. An implementation and various tests show that in practice, visually pleasing rectangular cartograms with small cartographic error can be generated effectively. Furthermore, the computation of proportional symbol maps is also discussed briefly.

Three-dimensional topographic modeling is also the topic of the paper by Friso Penninga (TU Delft): 'Constrained tetrahedral models and update algorithms for topographic data'. In contrast to the work of Gold and Tse he does not do this by representing the bounding surfaces, but he represents the three-dimensional objects by sets of tetrahedrons. The whole model then becomes a tetrahedronized irregular network (TEN), the 3D version of the more generally known triangulated irregular network (TIN). The TEN is a well-defined and robust data structure which enables complex processing by separate processing on each primitive first and afterwards joining all these partial results into a final result. In order to represent their borders several edges and faces will be handled as constraints. Updating a topographic dataset therefore equals the addition and removal of constraints within the network. One of the biggest challenges in the realization of such a data structure and corresponding algorithms is to reach acceptable performance, despite the potentially enormous amount of data.

The last paper 'Towards improved solution schemes for Monte Carlo simulation in environmental modeling languages' is by Derek Karssenberg and Kor de Jong (Utrecht University). They deal with the field-based representation of spatial data, in contrast to the object-based representation of spatial data in the other papers. On the most often used field-based data structure, the regular grid, the algorithmic challenges are quite different than their counterparts in the object-based approaches. Environmental modeling languages such as PCRaster are programming languages embedded in GIS to simulate environmental processes. These languages are used to construct dynamic models, also called forward models, which are simulations run forward in time, where the state of the model at time t is defined as a function of its state in a time step preceding t. For future applications, at least two extensions to the languages are required: support of three spatial dimensions (as the real world is often 3D), and inclusion of Monte Carlo simulation techniques (to calculate how input errors propagate to the output of a model).

The seminar day was organized by the subcommittee Geo-Information Models of the Netherlands Geodetic Commission, and was held at Utrecht University on November 14, 2005. We wish to thank the sponsors of the seminar as well for helping to make the day free of costs for all participants. These sponsors are the Netherlands Geodetic Commission, Geo Informatie Nederland (GIN), the Department of Information and Computing Sciences of Utrecht University, and the section GIS technology at the Delft University of Technology. We are further grateful to Frans

Schröder for his support in setting up the seminar, but also for the realization of this publication. Finally, we hope that this publication will help to increase the collaboration between computational geometry and GIS. We express our thanks to all speakers at the seminar, as they are the main contributors to this publication, and the participants for the lively discussions at the seminar.

The editors,

Peter van Oosterom
Marc van Kreveld

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.

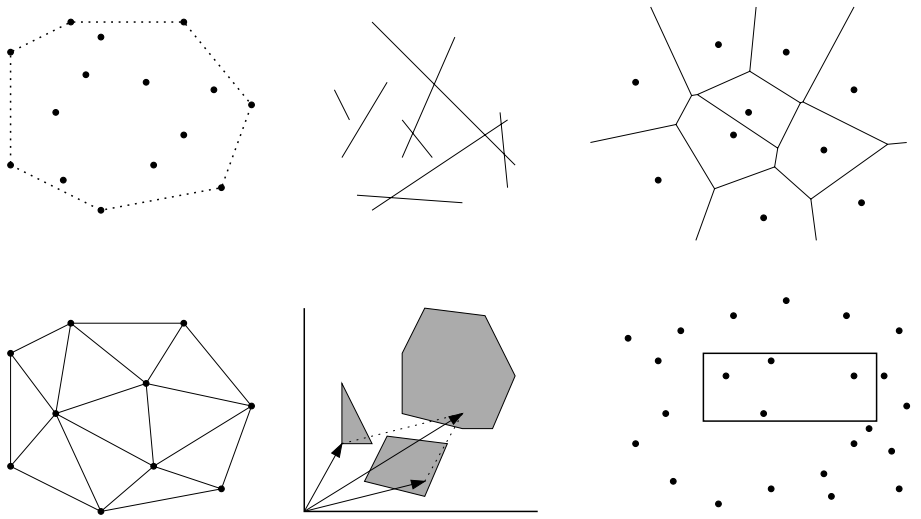

Figure 1: Six basic problems in computational geometry: convex hull, line segment intersection, Voronoi diagram, Delaunay triangulation, Minkowski sum, and rectangular range search.

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

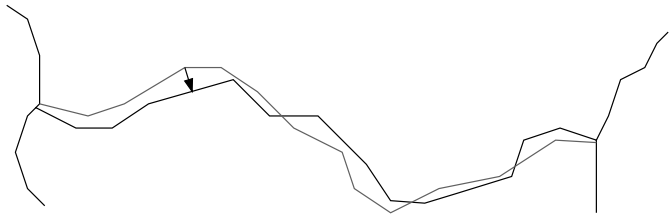

Figure 2: Doubly digitized line and Hausdorff distance.

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.

I/O- and Cache-Efficient Algorithms for Spatial Data

Mark de Berg*

Abstract

I/O- and caching behavior often have a major impact on the performance of algorithms. Traditional algorithms theory, which focusses on the CPU computation time, is therefore not always adequate to predict the performance of an algorithm in practice. We discuss some of the research on so-called I/O-efficient and cache-oblivious algorithms, giving special attention to algorithms and data structures for spatial data.

1 Introduction

In many application areas of computer science—examples are geographic information systems (GIS), robotics, computer graphics and virtual reality, and CAD/CAM—spatial data play a central role. Often the amount of data that needs to be processed in these areas is quite large, and the trend of having larger and larger data sets is still continuing. This is especially true for GIS, where the availability of terrain (and other) data has increased tremendously in recent years. For example, terrain data for 80% of the world's landmass was collected at 30m resolution by the international Shuttle Radar Topography Mission (SRTM) during a nine-day mission in 2000. Using technologies such as LIDAR (laser altimetry; LIght Detection and Ranging) one can nowadays, with relatively little effort, obtain large amounts of terrain data: a fairly small device mounted on a low-flying aircraft can acquire within a short period of time millions of geo-referenced points at a resolution of 1m or less.

The fact that data sets are getting larger and larger means that, even though computer technology is bringing us more powerful machines every year, it is still essential that computational tasks are performed in an efficient manner. Indeed, the size of current data sets implies that the scale-up behavior of algorithms is getting more and more important. Traditional algorithms theory is focussing exactly on this: scale-up behavior is the main concern in both the design and the analysis of algorithms. Thus it seems that algorithms theory is suited perfectly for the job. Unfortunately, this is not quite true. When dealing with massive data sets that do not fit into the computer's main memory, the I/O-behavior rather than the CPU computation time is often determining the actual running time of an algorithm. Indeed, a disk access is easily a factor 100,000 or more slower than an operation on data that is already present in the main memory. This aspect is not taken into account in traditional algorithms theory: here any operation is assumed to take unit time irrespective of where the data is stored. Similar issues play a role when the data fits into the main memory, because also the caching behavior of an algorithm can have a significant impact on its running time. Ideally, one would like to take I/O- and caching behavior into account both in the design and in the analysis of algorithms and data structures.

In recent years the algorithms community has realized the importance I/O- and caching behavior, and researchers are working towards a theory that takes these aspects into consideration. In this paper we review some of the work that has been done in this area, focussing on algorithms for spatial data. We will begin in Section 2 by discussing the *I/O-model*. This model is meant for the setting where the data are stored on disk and disk operations are the main factor determining

*Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands. Email: mdberg@win.tue.nl. Research by MdB is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

the running time of an algorithm. We continue in Section 3 with the *cache-oblivious model*, which is meant to capture all levels of the memory hierarchy—not only disk, but also the various cache levels—of modern computer systems. We give some concluding remarks in Section 4.

2 I/O-efficient algorithms

The model. When data is stored on an external memory device such as a disk, data transfer between the main memory and disk—either to read data from or to write data to disk—usually determines the actual running time of an algorithm. The I/O-model,[1] introduced by Aggarwal and Vitter [4] therefore focusses on data transport between disk and main memory. The model assumes there is a disk with unlimited storage capacity and a main memory of size M (that is, M basic "items" such as integers, pointers, etc. fit into the main memory)—see Fig. 1(a). The model further assumes, as is the case in reality, that data is transported between main memory and disk in *blocks*. The number of items that fit into one block is denoted by B. Thus, when a data item is needed that is not currently stored in the main memory, the block of B items that contains the item is read from disk and transferred to the main memory; when the memory is full some other block is evicted (wrote back to disk) to make room for the new block. The goal of an I/O-efficient algorithm—sometimes also called external-memory algorithm, or EM-algorithm for short—is to solve the computational task at hand with a minimum number of block transfers. To get good I/O-behavior, one needs to ensure two things: the data should be grouped into blocks in a good way (so that when a block is read most of the data in it is useful for the computation) and the times at which a block is needed should be clustered in time (so it pays off to keep a block in main memory for some time). Normally all of this—which data is put together into a block, and which block to evict when room has to be made for a new block—is regulated by the operating system. Although modern operating systems are usually good at this, their blocking and replacement policies are necessarily general purpose; often one can obtain significant speed-up by letting the algorithm itself handle data blocking and transport.

Let's look at a simple example. Suppose we want to search in a set S of N elements, each with a key: given a query value q, we want to find the element $x \in S$ with $key[x] = q$, if it exists. An easy solution to this problem is to store the elements from S in sorted order in an array. A query can then be answered in $O(\log_2 N)$ steps using binary search. What happens if we apply the same solution when the data set is too large to fit into the main memory, so that we have to store the array on disk? The obvious strategy is to group the elements together in order: the first B elements from the array go into a block, the second B elements go into a block, and so on. But this is not a good solution. Only at the end of the binary search, when the algorithm has zoomed in on a set of B elements that are together in a block, do we profit from the blocking; until then a new block is needed for every step. Hence, the total number of I/O's needed to answer a query is roughly $\log_2 N - \log_2 B = \Theta(\log_2 N)$. A much better solution is to use the classical B-tree [18]. This is a search tree, where the nodes have high degree. More precisely, the degree of a node is between t and $2t$, where the value of t is chosen such that a node of degree $2t$ just fits into one block. (Since a node of degree $2t$ stores $2t$ pointers to its children and $2t - 1$ keys, this means $t \approx B/4$.) A search in a B-tree follows a single path down the tree, so the number of blocks needed is bounded by the height of the tree, which is $\log_{B/4} N = O(\log_B N)$. The difference between $\log_2 N$ and $\log_{B/4} N$ is crucial: if, say, $N = 10,000,000$ and $B = 2,000$, then $\log_2 N \approx 23$ while $\log_{B/4} N \approx 3$, so we get a speed-up[2] of almost a factor 10. This shows that it can really pay off to control the grouping of the data into blocks.

We just saw that *searching* in a set of N elements, which takes $O(\log_2 N)$ time in internal memory, can be done using $O(\log_B N)$ I/O's with a B-tree. Another basic result is on *sorting*.

[1] The model of Aggarwal and Vitter is in fact more general than this, as it also considers multiple parallel disks. For simplicity we limit our discussion the the single-disk case.

[2] Here we are cheating a little, because the root of a B-tree is not guaranteed to have degree at least t. On the other hand, one would normally keep the root of the tree and maybe the first level in main memory, so indeed one never needs to do more than 3 I/O's.

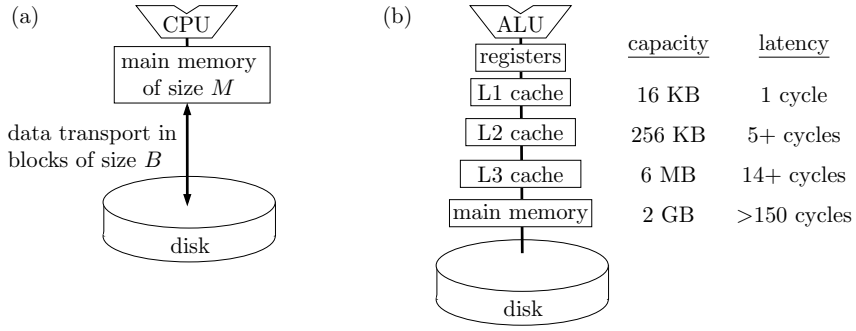

Figure 1: (a) The I/O-model. (b) Memory hierarchy of the Intel(R) Itanium(R) 2 processor.

This can be done using $O((N/B)\log_{M/B} N)$ I/O's [4], while it takes $O(N \log_2 N)$ time in internal memory. Next we discuss some results on I/O-efficient geometric algorithms.

I/O-efficient algorithms and data structures for spatial data. There has been a fair amount of work on I/O-efficient algorithms for spatial data. For example, it has been shown that the Voronoi diagram and the convex hull of a set of N points in the plane can be computed with $O((N/B)\log_{M/B} N)$ I/O's [19], there are results on (dynamic) point location [12], on geometric (and other) data structures [5], etc. It goes beyond the scope of this paper to discuss these and all the other results on I/O-efficient geometric algorithms in detail. We shall confine ourselves to briefly discussing three problems that are more directly relevant for GIS. The interested reader can consult the survey by Breimann and Vahrenhold [16] for more results on I/O-efficient geometric algorithms.

The first problem we discuss is the line-segment intersection problem: compute all intersections in a given set of N line segments in the plane. This problem arises when one wants to do map overlay. In internal memory, there is a simple sweep-line algorithm [15] that solves the problem in $O((N + K)\log N)$ time, where K is the number of intersecting pairs of line segments. This algorithm sweeps a vertical line ℓ over the plane, maintaining the segments intersecting ℓ in order along ℓ in a binary tree $\mathcal{T}$. When ℓ starts to intersect a segment, that segment must be inserted into $\mathcal{T}$; when it stops to intersect a segment, that segment must be deleted from $\mathcal{T}$; and when two segments intersect—these segments must then be neighbors along ℓ, so this can be tested easily—their order must be reversed. There are also several optimal algorithms, which run in $O(N \log N + K)$ time. In the I/O-model, the situation is less rosy. Already the sweep-line algorithm, which is fairly simple in internal memory, is difficult to make I/O-efficient. Instead of using a binary tree to store the segments intersecting ℓ, we could use a B-tree. This way each event during the sweep is handled using $O(\log_B N)$ I/O's, leading to $O((N + K)\log_B N)$ I/O's in total. However, we would like to get a bound like $O((N/B + K/B)\log_B N)$. This is difficult, because the sweep seems inherently sequential: it is hard to avoid doing at least one I/O to handle each event. Still, using a technique called distribution sweeping, it is possible to solve the line-segment intersection problem using $O(((N/B) + (K/B))\log_{M/B} N)$ I/O's [13]. Currently, no I/O-optimal solution, which uses $O((N/B)\log_{M/B} N + K/B)$ I/O's, is known.

The second result we discuss concerns geometric data structures. A well known data structure (or: indexing structure, as it is also called) for storing spatial data on disk is the R-tree, introduced by Guttman in 1984 [20]. An R-tree for a set of rectangles (or other objects) in the plane is a tree whose nodes have high degree, like a B-tree, and whose leaves store the rectangles. An internal node stores a bounding box for each of its subtrees, namely the bounding box of all the rectangles stored in the leaves of that subtree. An R-tree can be used to answer various types of queries; see the book by Manolopoulos *et al.* [23] for an in-depth discussion of R-trees and their applications.

For example, R-trees can be used for orthogonal range queries: report all rectangles intersecting a query rectangle. Many different variants of the R-tree have been proposed—the R*-tree and the Hilbert R-tree being two of the more popular ones—but, even though R-trees perform quite well in practice, none of them had good bounds on the worst-case number of I/O's needed to answer orthogonal range queries. Recently, however, Arge *et al.* [7] managed to develop an R-tree variant, which they call the *PR-tree*, that answers orthogonal range queries using an optimal number of I/O's, namely $O(\sqrt{N/B} + K/B)$ where K is the number of answers.

Finally, we mention some of the work that has been done on terrain models. As mentioned earlier, terrain models are now available at higher and higher resolution, leading to larger and larger data sets. Moreover, flow analysis on terrains is quite important in many applications, for example for hydrologists and country planners while managing or monitoring water resources, possible flood areas, erosion and other natural processes. It is therefore not surprising that flow computations on terrains have also been studied in the I/O-model. Specifically, Arge and co-workers [9, 10, 11] have developed I/O-efficient algorithms for the computation of watersheds, river networks, etc. on terrains. Their algorithms are both efficient in theory (in terms of the number of I/O's they perform), as well as in practice. Compared to existing GIS systems such as ArcInfo and GRASS, their software [26] was between 2 and 1,000 times faster (depending on the specific input).

3 Cache-oblivious algorithms

The model. The I/O-model is suitable when data transfer between main memory and disk is the main bottleneck. It is clear, however, that it is a simplification of reality: modern computer systems have a whole hierarchy of memory levels of decreasing speed but increasing size, ranging from very fast but small registers, through several cache levels and main memory, to the very slow but large disk. Fig. 1(b) gives an example of such a hierarchy, with the size and the latency (i.e. the access time, measured in clock cycles) of the various cache levels and the main memory. Thus, the issue of data transport not only plays a role between disk and main memory, but also between main memory and L3 cache, between L3 cache and L2 cache, and so on. This means that ideally one would like to design algorithms that are efficient on all levels of the memory hierarchy, not just with respect to disk accesses. This seems difficult, however. First of all, each level has its own characteristics (memory size M and block size B) and tuning the algorithm to behave well with respect to each of these parameters is very cumbersome. Actually, the fact that the algorithm has to be tuned to the parameters M and B of the disk is already problematic, because the values of these parameters depend on the specific platform on which the algorithm runs. Even on a fixed platform it is not clear which parameters one should use. (For example, although the size M of the main memory is fixed for the platform, the algorithm will have to share the memory with other processes and so the available memory may vary over time. Also, even though the physical block size is determined by the disk, the existence of a disk cache means that the disk behaves as having a much larger block size.) Second, and more importantly, most of the levels in the memory hierarchy are hardware controlled—an application cannot control which data to group together into a block and which blocks to keep in memory.

The *cache-oblivious model*, introduced by Frigo *et al.* [21], is a very elegant approach to this problem. A cache-oblivious algorithm is designed in an abstract two-level memory model: there is a fast memory of limited size and a slow memory of unlimited storage capacity, and data is transferred between the fast and the slow memory in blocks. The slow memory could represent the disk and the fast memory the main memory, or the slow memory could represent the main memory and the fast memory the L3 cache, and so on. This is very much like the I/O-model, with one crucial difference: the parameters M and B are unknown to the algorithm. For instance, the algorithm cannot explicitly decide to put a set of B items into one block, because it does not know what B is. This way the algorithm will be independent of the specific block and memory sizes of the various memory levels—no tuning of the parameters is necessary because the algorithm does not use these parameters. In the analysis, however, we do use M and B; that is, we analyze the

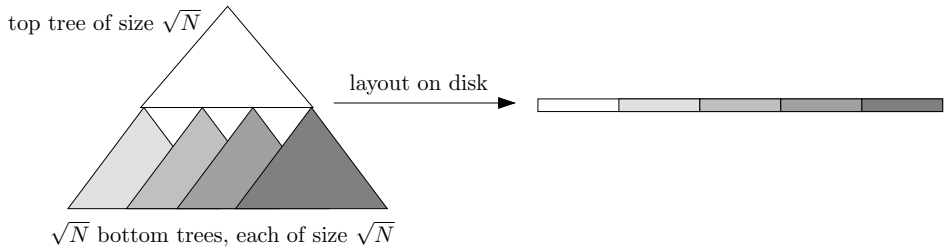

Figure 2: Exponential layout of a binary search tree.

number of block transfers done by the algorithm in terms of N (the input size) and M and B. The beauty is that the analysis then applies to every pair of adjacent levels of the memory hierarchy, since the analysis holds for any M and B values.

At first this may seem impossible: how can we design algorithms without knowing M and B that, when analyzed, magically turn out to have good behavior with respect to M and B? Surprisingly enough, this is possible under some mild assumptions on the blocking and replacement strategy used by the operating system, as discussed next. One important (and realistic) assumption is that blocks are formed according to the order in which data is written. That is, if we write data items $x_1, x_2, \ldots$ in order then the first B items $x_1, \ldots, x_B$ form a block, the second B items $x_{B+1}, \ldots, x_{2B}$ form a block, and so on. We do not know what B is, so we do not know for example if the first item will be in the same block as the 50th item, but we do know that the first block will contain the first B items—whatever the value of B may be. A second assumption is that the operating system uses an optimal block replacement strategy: when a block has to be evicted from the fast memory to make room for another block, the operating system chooses the block for which it takes the maximum time (among all blocks currently in the fast memory) before it is needed again. This may seem like a very unrealistic assumption, but a replacement strategy like LRU can be shown to approximate the optimal strategy quite well. There are a few other technical assumptions, which we will not discuss here—the interested reader can consult the paper by Frigo *et al.* [21] for a discussion and theoretical justification of the assumptions. Instead, let's look at an example to get an impression of what is possible in the model.

Consider the searching problem we discussed before: we want to store a set S of N elements, each with a key, such that for a query value q we can quickly find the element $x \in S$ with $key[x] = q$ (if it exists). For convenience, we call the fast memory in our abstract two-level model the *cache* and the slow memory the *disk*. If we write the elements from S in sorted order to disk, then by assumption blocks will be formed in sorted order. But we have seen that this is not very good, as a binary search will still incur $\Theta(\log_2 N)$ block transfers. In the I/O-model, we could improve this to $O(\log_B N)$ by using a B-tree: we put $\Theta(B)$ keys into a single node so that a node fits into a block. This way we create a search tree whose nodes have degree $\Theta(B)$ and whose depth is $O(\log_B N)$. But we cannot do this in the cache-oblivious model: we do not know the value of B, so we don't know how to choose the number of keys in a node. So what can we do? The trick is as follows. Suppose we have a normal balanced binary search tree $\mathcal{T}$ on the keys. We will define an ordering on the nodes of $\mathcal{T}$. The nodes will be then written to disk (and blocks will be formed) according to this ordering. The ordering is defined recursively as follows. Imagine cutting $\mathcal{T}$ into subtrees at the middle level, as in Fig. 2. We obtain one top tree with $\sqrt{N}$ nodes, and $\sqrt{N}$ bottom trees that also have roughly $\sqrt{N}$ nodes each. The nodes in the top tree come first in the ordering, next come the nodes in the first bottom subtree, then the nodes in the second bottom subtree, and so on—see Fig. 2. Within each subtree (top and bottom) the ordering is defined recursively in the same way. This way of laying out the nodes on disk is called the *exponential layout*, or also *Van Emde Boas layout*. One can prove [24] that this layout produces a blocking of the tree such that any root-to-leaf path visits $O(\log_B N)$ blocks, thus matching the bound achieved by a normal

B-tree. There are also dynamic cache-oblivious B-trees [14].

Although much less algorithms are known in the cache-oblivious model than in the I/O-model, there are optimal cache-oblivious algorithms for a several basic problems. For instance, there is a cache-oblivious sorting algorithm that uses $O((N/B)\log_{M/B} N)$ I/O's [21]. Below we briefly mention some of the cache-oblivious algorithms that have been developed for spatial data.

Cache-oblivious algorithms and data structures for spatial data. Some basic geometric algorithms have been solved optimally in the cache-oblivious model. One example is the problem of computing the convex hull of a planar point set. This is in fact rather easy, because the standard algorithm [15] (Graham's scan) basically only needs a sorting subroutine and a stack. Using an optimal cache-oblivious sorting algorithm and a cache-oblivious stack (a simple array implementation of a stack suffices), we can thus obtain an algorithm that uses $O((N/B)\log_{M/B} N)$ I/O's. Kumar [22] designed a cache-oblivious algorithm for computing the Voronoi diagram in the plane using $O((N/B)\log_M N)$ I/O's. He implemented a simplified version of the algorithm. Although the simplified version is no longer theoretically optimal, it performs well and could compute Voronoi diagrams of up to 300,000,000 points. The distribution sweeping paradigm has also been generalized to the cache-oblivious model, leading to optimal algorithms for a number of problems such as orthogonal line-segment intersection [17]. The normal (non-orthogonal) line-segment intersection problem has not been solved satisfactorily in the cache-oblivious model.

Geometric data structures have received quite some attention in the cache-oblivious model. Much of the work focussed on specialized search structures, in particular for orthogonal range searching [1]. A cache-oblivious variant of the R-tree has also been developed [8]. Although the structure has good worst-case theoretical bounds, its practical efficiency is unclear.

Problems for terrain data have hardly been studied in the cache-oblivious model. On the one hand this is as expected, because terrain data is often so massive that disk accesses are the dominant factor. Hence, the I/O-model may seem sufficient. On the other hand, terrain data is more and more also processed on a wide variety of platforms—from large systems to small handheld devices—so it pays off to have portable code for which parameter tuning is not necessary. Thus it is likely that cache-oblivious algorithms for terrain data will receive more attention in the near future.

4 Concluding remarks

I/O- and caching behavior often have a major impact on the performance of algorithms. Traditional algorithms theory, which focusses on the CPU computation time, is therefore not always adequate to predict the performance of an algorithm in practice. We have discussed two methodologies for dealing with this problem: the I/O-model, which aims to capture the huge difference in the time needed for disk-accesses and for computations on data in internal memory, and the cache-oblivious model, which aims to model multi-level memory hierarchies of modern computer systems. We have given examples of algorithms for spatial data that have been developed for these models.

These two methodologies are in very different stages of their development. For the I/O-model, there is a fair amount of theory, and for a number of basic computational problems I/O-optimal solutions have been developed. (This is not to say that there are no more challenges in this area: especially graph algorithms are still poorly understood in the I/O-model.) Moreover, experiments show that I/O-efficient algorithms theory can really help to speed up computations in practice for massive data sets. However, the model does not take caching behavior into account, and algorithms developed in this model need to be tuned to machine-specific parameters.

In the cache-oblivious model much less is known. The model is very elegant and potentially quite powerful: algorithms that are efficient in this model are automatically efficient with respect to all cache levels (and disk), and no tuning of machine specific parameters is needed. For some basic problems, cache-oblivious solutions have been developed, but much is still open. Furthermore, the

practical relevance of the model is not clear yet. There have been a few encouraging experiments, but much more research in this area is needed to establish the practical viability of the model.

References

[1] P. K. Agarwal, L. Arge, A. Danner, and B. Holland-Minkley. Cache-Oblivious Data Structures for Orthogonal Range Searching. In *Proc. 17th ACM Symposium on Computational Geometry*, pages 237–245, 2003.

[2] P. K. Agarwal, L. Arge, T. M. Murali, K. R. Varadarajan and J. S. Vitter. I/O-efficient algorithms for contour-line extraction and planar graph blocking. In *Proc. 9th Symp. on Discrete Algorithms*, pages 117–126, 1998.

[3] P.K. Agarwal, L. Arge, and K. Yi. I/O-efficient batched Union-Find and its applications to terrain analysis. To appear in *Proc. 22nd ACM Symp. on Computational Geometry*, 2006.

[4] A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM* 31(9):1116–1127, 1988.

[5] L. Arge. External-memory data structures. In J. Abello, P.M. Pardalos, and M. G.C. Resende (eds.). *Handbook of Massive Data Sets*, Kluwer Academic Publishers, pages 313–357, 2002.

[6] L. Arge. External-memory algorithms with applications in geographic information systems. In M. van Kreveld, J. Nievergelt, Th. Roos, and P. Widmayer (eds.). *Algorithmic Foundations of Geographic Information Systems*, Lecture Notes in Computer Science 1340, Springer-Verlag, pages 213–254, 1997.

[7] L. Arge, M. de Berg, H.J. Haverkort, and K. Yi. The Priority R-tree: A practically efficient and worst-case-optimal R-tree. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 347–358, 2004.

[8] L. Arge, M. de Berg, H.J. Haverkort, and K. Yi. Cache-oblivious R-trees. In *Proc. 21st ACM Symp. on Computational Geometry*, pages 170–179, 2005.

[9] L. Arge, L. Toma, and J.S. Vitter. I/O-Efficient Algorithms for Problems on Grid-based Terrains. *Journal of Experimental Algorithmics* 6, 2001.

[10] L. Arge, J. Chase, P. Halpin, L. Toma, D. Urban, J. S. Vitter, and R. Wickremesinghe. Flow computation on massive grid terrains. *GeoInformatica* 7(4):283–313, 2003.

[11] L. Arge, A. Danner, H. Haverkort, and N. Zeh. Computing Pfafstetter Labellings I/O-Efficiently. In *Abstracts 1st Workshop on Massive Geometric Data Sets*, Pisa, 2005.

[12] L. Arge and J. Vahrenhold. I/O-Efficient Dynamic Planar Point Location. *Computational Geometry: Theory and Applications* 29(2):147-162, 2004.

[13] L. Arge, D.E. Vengroff, and J.S. Vitter. External-memory algorithms for processing line segments in geographic information systems. In *Proc. 3rd Annual European Symposium on Algorithms*, Lecture Notes in Computer Science 979, Springer-Verlag, pages 295–310, 1995.

[14] M.A. Bender, E.D. Demaine, and M. Farach-Colton. Cache-Oblivious B-Trees. In *Proce. 41st IEEE Symp. on Foundations of Computer Science*, pages 399–409, 2000.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geomtry: Algorithms and Applications*, 2nd edition. Springer-Verlag, 2000.

[16] C. Breimann and J. Vahrenhold. External memory computational geometry revisited. In: U. Meyer, P. Sanders, J. Sibeyn (Eds.). *Algorithms for Memory Hierarchies*. Lecture Notes in Computer Science 2625, Springer-Verlag, pages 110–148, 2003.

[17] G.S. Brodal and R. Fagergerg. Cache-oblivious distribution sweeping. In *Proc. Annual International Colloquium on Automata, Languages, and Programming*, pages 426–438, 2002.

[18] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[19] M.T. Goodrich, J.-J. Tsay, D.E. Vengroff, and J.S. Vitter. External-memory computational geometry. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 714–723, 1993.

[20] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[21] M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Symp. on Foundations of Computer Science*, pages 285-298, 1999.

[22] P. Kumar. *Clustering and Reconstructing Large Data Sets*. PhD thesis, Stony Brook University, 2004.

[23] Y. Manolopoulos, A. Nanopoulos, A.N. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications*. Springer-Verlag, 2006.

[24] H. Prokop. *Cache-Oblivious Algorithms*. Masters thesis, MIT. 1999.

[25] J.S. Vitter. External memory algorithms and data structures: Dealing with MASSIVE data. *ACM Computing Surveys* 33(2):209–271, 2001.

[26] Computations on Massive Grids: The TerraFlow Project. http://www.cs.duke.edu/geo*/terraflow/

Quad-Edges and Euler Operators for Automatic Building Extrusion Using LIDAR Data

Christopher Gold and Rebecca Tse

GIS Research Centre, School of Computing, University of Glamorgan,
Ponrypridd CF37 1DL Wales, UK. cmgold@glam.ac.uk, rtse@glam.ac.uk

Our long-term research objective is to integrate man-made objects with the landscape, so that topological properties, such as connectedness, may be used in applications such as flood modelling. Building information should be extracted directly from LIDAR data.

Several steps are required. Building data points should be separated from the terrain data points by identifying building boundaries, either from cadastral information or by various edge-detection techniques. Bare earth topography is obtained, either from existing information or by filtering the LIDAR data. Data points inside the building boundaries are then processed to give an average building height, and to estimate the roof shape. The buildings are then constructed by treating the terrain TIN model as a CAD type b-rep surface, and performing local modification of the Quad-Edge data structure by using Euler operators. These operators permit various extrusion operations, as well as the manual insertion of bridges and tunnels.

Two approaches have been taken to the separation of building data from terrain data. Unlike the traditional approach, where walls are represented by non-vertical triangles, our first approach is to insert cadastral boundaries into our TIN by adding sufficient extra points to ensure that the boundaries follow triangle edges. Euler operators are then used to generate a second boundary (the tops of the walls) which is then extruded upwards. A more experimental method uses a coarser network of Voronoi cells that sample the underlying LiDAR data. These cells are then perturbed until each one is entirely part of the (higher) building or (lower) terrain. Building blocks are then extracted.

Data points interior to the building may be used to estimate the height of a supposedly flat-roofed building, or they may be used to extract the form of the roof. This is attempted by calculating the vector normals of “roof” triangles, and calculating the eigenvalues of their direction cosines. For simple roof shapes this will give the primary roof orientation, and the pattern of normals indicates the roof form. This information may then be integrated into the surface model.

Introduction

LIDAR (Light Detection and Ranging) data is widely used to construct 3D terrain models to provide realistic impressions of the urban environment and models of the buildings. The latest airborne laser scanning technology allows the capture of very dense 3D point clouds from the terrain and surface features. The most common method is to remove all the buildings, trees and terrain objects and generate a bare-earth model. Then building boundaries are extracted from the LIDAR data points. The buildings are reconstructed using CAD software and pasted on top of the bare-earth model. Our approach integrates the buildings with the topography.

We first describe using Ordnance Survey Landline data for building reconstruction. This involves inserting boundary points into the TIN model and then using Euler operators (based on

the Quad-edge data structure) to extrude the building from the TIN. We then suggest an alternative way, the Voronoi diagram, to extract building outlines from the raw LIDAR data. An additional Euler operator may then be added to permit the insertion of bridges and holes within our structures. Finally, we suggest methods for roof reconstruction within individual buildings.

Building Reconstruction by Using LiDAR Data

Airborne laser scanning allows the capture of very dense point clouds, permitting 3D building reconstruction. This is an active research topic in GIS. However different steps are involved before actually using the laser scanning data. A filtering algorithm is used to generate a bare-earth model which removes all the buildings, trees and terrain objects. Morphologic filtering is one of the algorithms which are commonly used to solve this problem, for example slope based filtering (Vosselman, 2000) and modified slope based filtering (Roggero, 2002).

Constructive Solid Geometry (CSG) and VRML are the two common methods for modelling and rendering the buildings on the terrain surface respectively. Suveg and Vosselman (2004) used CSG to generate a complex building with the Boolean operations of union, intersection and differences. Rottensteiner and Briese (2003) used VRML to display the generated buildings. However the topological relationships are not kept during the construction. All of the rendered buildings are superimposed on the terrain surface without actually being connected to it. If the topological connectivity is preserved, more kinds of spatial analyses can be performed.

Adding Buildings to a TIN Using Boundary Data

Several steps are used to create a 3D terrain model. They are:

1. Create a TIN model with the filtered LiDAR data. Filter the laser scanning data by one of the methods mentioned above.
2. Add boundary data to the terrain surface.
3. Calculate the average heights of the buildings from the LiDAR data.
4. Extrude the building by using CAD-type Euler Operators while keeping the topological connectivity.

Line Tracing Algorithm

In order to model buildings, we need to estimate the ground surface at the foot of the walls described by the OS Mastermap data. We also need to have triangle edges that follow these lines.

The line tracing algorithm is:

1. Insert two points on the terrain surface.
2. Check if any triangle edge connects these two points.
    i. Stop if this is true.
    ii. Insert a point half way between these two points if no edge connects them.
3. For each half of the line repeats step 2 recursively until points A and B are connected by triangle edges.

We add the points on the terrain surface and estimate the height of the points by using the surface interpolation method of (Dakowicz and Gold, 2002).

Building Extrusion Using Euler Operators

We start by creating a 2.5D TIN model and Euler Operators are used to extend the TIN (Tse and Gold, 2002). Building boundaries are added on the terrain surface. Then the boundaries are extruded from the ground surface to the height of the building using Euler Operators.

The Basic Quad-Edge Data Structure

We used the Quad-Edge data structure (Guibas and Stolfi, 1985) as the basis of our model. More particularly, the Quad-Edge structure was used to implement a set of Euler Operators that were sufficient for the maintenance of surface triangulations. According to Weiler (1988), if a topological representation contains enough information to recreate nine adjacency relationships without error or ambiguity, it can be considered a sufficient adjacency topology representation. These Euler Operators form the basis of the standard (two-dimensional) incremental triangulation algorithm. In addition, Euler Operators can serve to generate holes within our surfaces, thus permitting the modelling of bridges, overpasses etc. that are so conspicuously lacking in the traditional GIS TIN model. The individual Quad-Edge and Euler Operators take only a few lines of code each. "Make-Edge" and "Splices" are the two basic operations on the Quad-Edge structure, which may be formed from four connected "Quad" objects (Fig. 1). Every Quad has three pointers:

- N - link to next Quad ("Next") anticlockwise around a face or a vertex
- R - link to next 1/4 Edge ("Rot") anticlockwise around the four Quads
- V - link to vertex (or face)

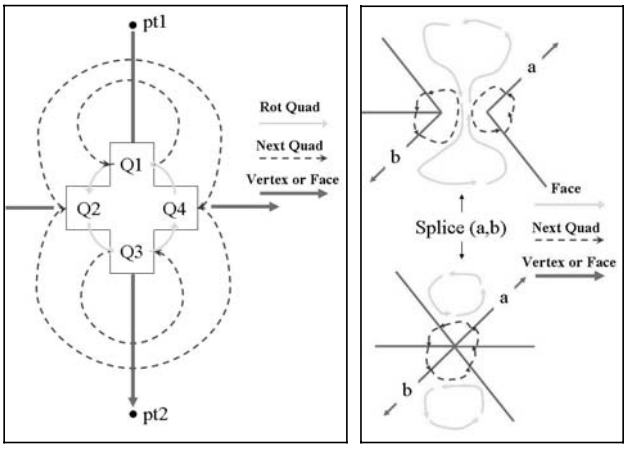

Figure 1. a) MakeEdge operator b) Splice operator.

Implementation of Euler Operators Using the Quad-Edge Data Structure

Five spanning Euler Operators (plus the Euler-Poincaré formula) suffice to specify the number of elements in any boundary representation model (Braid et al., 1978) In TINs there are no loops (holes in individual faces), so these conditions will not be considered further. Thus four spanning Euler Operators suffice for TINs with tunnels or bridges. "Make Edge Vertex Vertex Face Shell" (MEVVFS) creates an initial shell (this is the same as “MakeEdge” above), and its

inverse "Kill Edge Vertex Vertex Face Shell" (KEVVFS) removes it. "Make Edge Face" (MEF) and its inverse "Kill Edge Face" (KEF) creates and kills an edge and a face. "Split Edge Make Vertex" (SEMV) splits an edge and creates a vertex, and its inverse is "Join Edge Kill Vertex" (JEKV) - see Fig. 2.

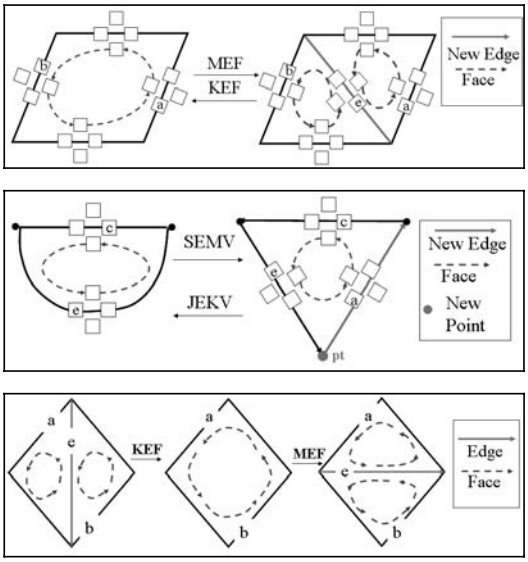

Figure 2. a) MEF b) SEMV c) Swap.

Implementation of the TIN Model Using Euler Operators

In the TIN model we have three main functions, which are:

1. Create a "First Triangle" big enough to contain all the data points and its inverse kill the first triangle
2. Insert, or its inverse, delete a point
3. Swap an edge

In creating the First Triangle, three points are needed as input. Three different Euler Operators are used: MEVVFS, MEF and SEMV. 3 points (pt1, pt2, and pt3) are input to create the First Triangle and 3 edges (e1, e2, and e3) are the output. MEVVFS creates the first edge e1. MEF creates a new edge e3. SEMV splits edge e3. To kill the First Triangle, JEKV joins edge e2 and e3. KEF kills edge e3 and a face. KEVVFS kills the last edge e1, giving an empty space (see Fig. 2).

Fig. 2 also shows inserting and its inverse, deleting, a new point - which makes three edges and two faces. MEF creates an edge N4. SEMV splits an edge N4. In the last step MEF creates a new edge N6. We use KEV and JEKV to delete a point. To delete a point, we use MEF to kill an edge and face. Then we use JEKV to join two edges and kill a vertex. We use KEF to kill edge N4 and its associated face.

Swap is a procedure for swapping two edges inside the TIN model. For the Delaunay Triangulation, we use the "in-circle" test to test the triangle, and use the swap operator to change edges.

The Delaunay Triangulation is based on the empty circumcircle criterion. Fig. 2c shows the steps for Swap using Euler Operators. We need to input an edge e to be changed. KEF kills the edge and MEF creates the edge. It swaps the edge between two triangles.

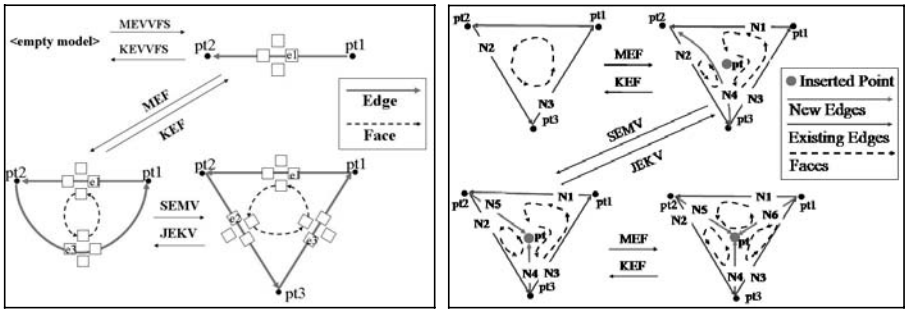

Figure 3. a) Creating "First Triangle" b) Inserting/deleting a point.

Building Extrusion Using Euler Operators

We used one more Euler Operator to simplify the extrusion procedure. "Make Zero-Length Edge Vertex" (MZEV) and its inverse "Kill Zero-Length Edge Vertex" (KZEV) are used to create and kill a zero length edge and a vertex.

We selected a rectangle with two triangles and a common edge to extrude a building. MEF creates a face and edge N1, which runs from point pt1 to pt4 in Fig. 4. Three face loops are inside the selected rectangle.

MZEV can be replaced by using MEF, SEMV and KEF. MZEV can simplify the procedures. MZEV is used to split point pt1 in two pieces. Point pt5 is created and edge N5 is created. Point pt5 is vertically on the top of point pt1. They have the same x and y coordinates, but different height values. The height of point pt5 is equal to the building height.

Three more MZEV operators are used (more if the building has more than four corners) until the entire building's corner points are split. MZEV split pt2, pt3, and pt4 by adding pt6, pt7 and pt8 respectively in Fig. 4. Fig. 5 shows the result for the University of Glamorgan campus.

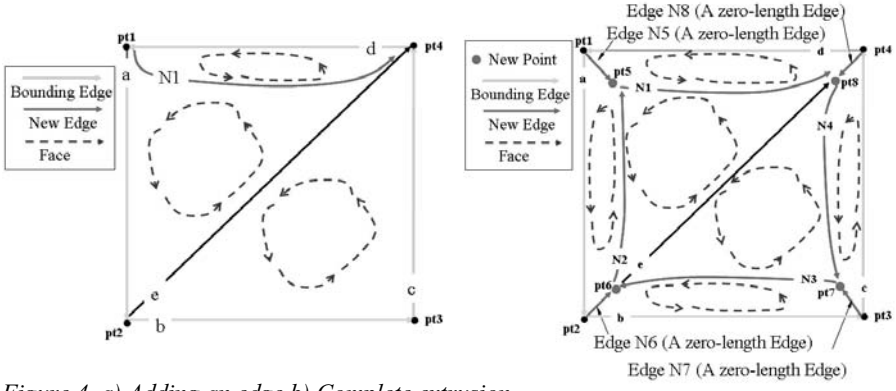

Figure 4. a) Adding an edge b) Complete extrusion.

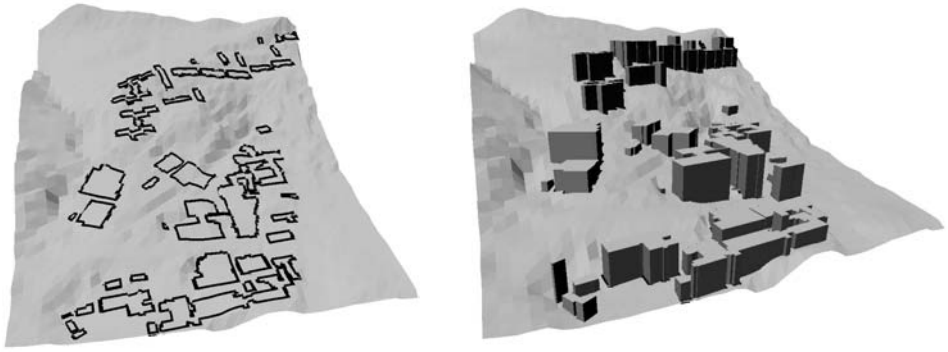

Figure 5. a) Building boundaries b) Extruded buildings.

Voronoi City Modelling

Though many researchers are interested in automatic filtering and building extraction algorithms when boundaries are not available, there is still room for improvement. With the use of the Delaunay Triangulation and its dual, the Voronoi Diagram, we suggest an alternative way to solve the problem. Several steps are used to reconstruct buildings using "Voronoi City Modelling". They are:

1. Sample the raw LIDAR data to give a lower density of data points.
2. Create the Voronoi Diagram of the sampled data points.
3. Assign the z-value of the sampled Voronoi centres to the average of the original data points falling within the cell.
4. Pick a sampled Voronoi cell.
5. Compare the picked sampled Voronoi cell with its neighbour cells.
6. Move the cell, average the data points inside, and see whether it increases the height differences compared with its neighbours. If so, keep it at its new location.
7. Repeat steps 4 to 6 until every cell is processed.
8. Display the sampled Voronoi cells in a 3D view.
9. Iterate the whole process from steps 1 to 7 until it shows the best building shapes in the 3D view.
10. Extrude all the Voronoi cells using Euler Operators.
11. Merge Voronoi Cells with similar heights.
12. Show all the building blocks on the terrain.

Fig. 6 shows the initial Voronoi cells and Fig. 7 shows the result where the building outlines have more or less formed after a few iterations.

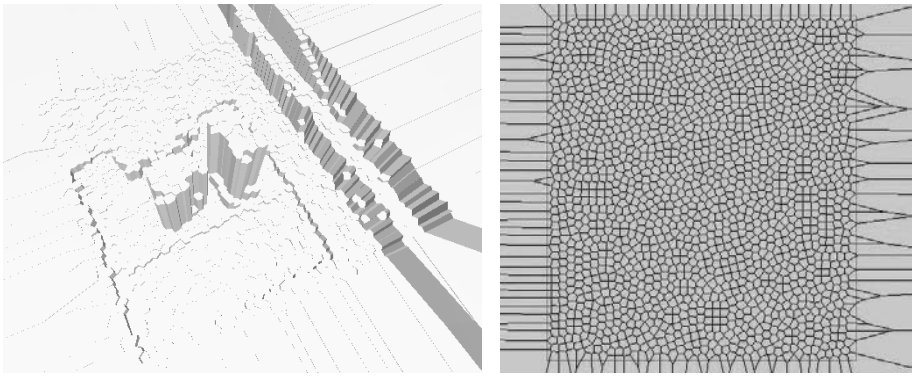

Figure 6. Voronoi City Modelling – initial state.

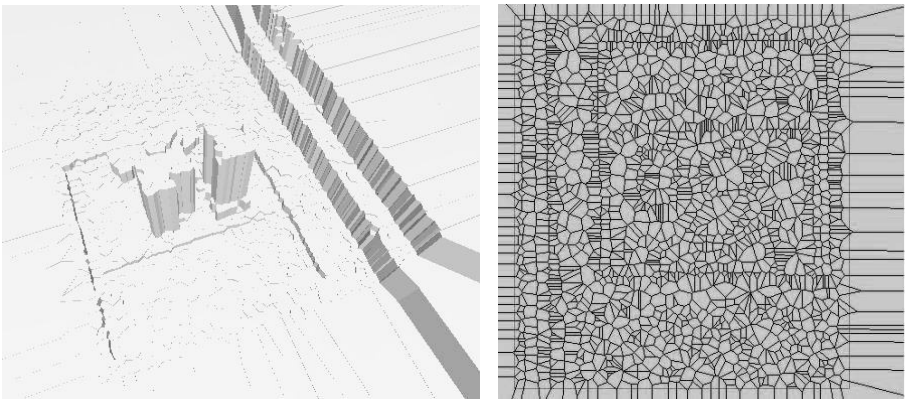

Figure 7. Voronoi City Modelling – after several iterations.

Bridges and Tunnels

We may then perform further spatial analysis on our city model because the topological connectivity is preserved. Interactive editing is allowed as in Tse and Gold (2002).

We created a TIN model and extruded buildings using Euler operators. We can then use an additional Euler operator MEHKF (Make Edge Hole Kill Face) to extend the TIN model with bridges and tunnels (Fig. 8). This method keeps the topological connectivity. The details can be found in Tse and Gold (2002). Fig. 9 shows some simple examples.

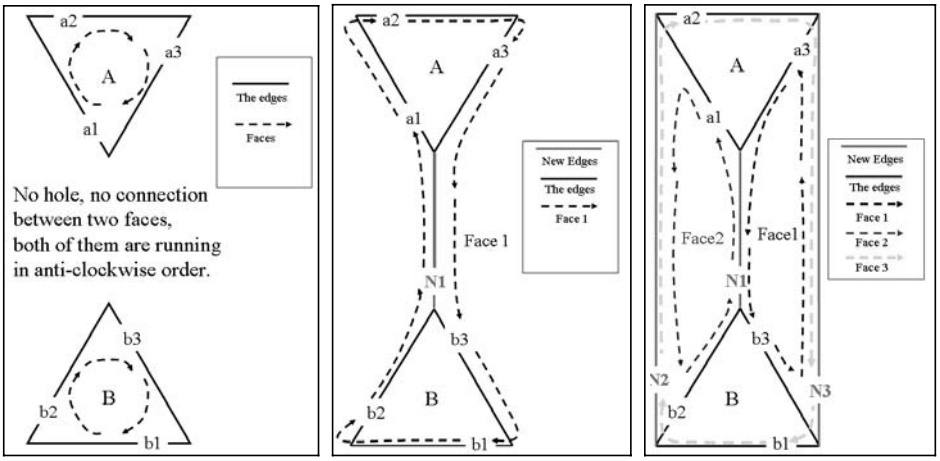

Figure 8. a) Initial condition b) Bridge or hole started using MEHKF c) Extra faces added.

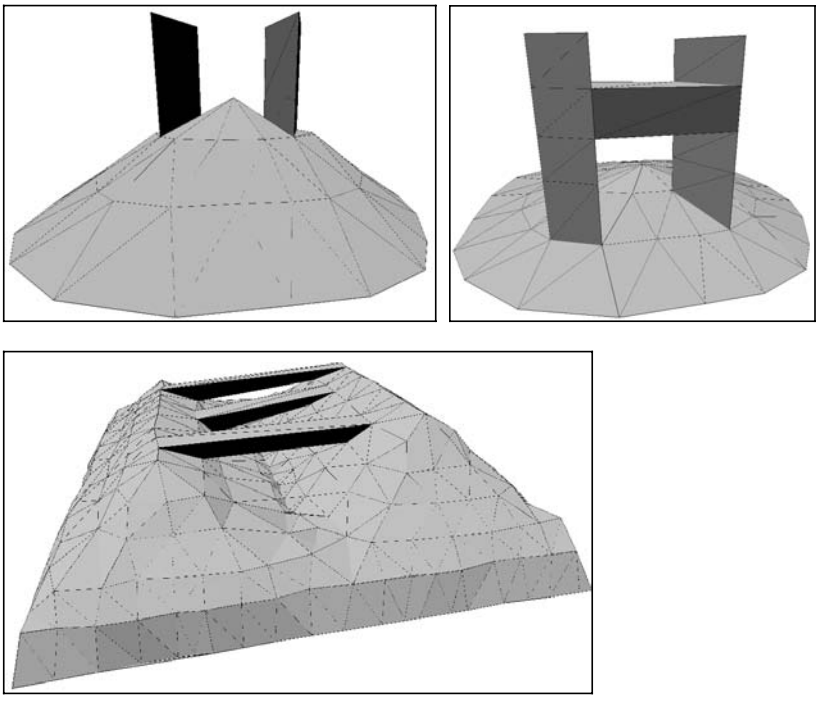

Figure 9. Simple structures created by extrusion and bridge/hole operations.

Conclusions and Future Work

We have explored the possibility of reconstructing buildings and terrain using LIDAR data. This is simplified by using the Quad-edge structure and Euler operators for TIN construction and building extrusion given building boundary data.

The Voronoi City Modelling approach may produce good results where boundary data is absent, but it is still being refined. When we have the extruded buildings on the terrain, we are attempting to reconstruct the roof from the interior LIDAR points. Using concepts from structural geology, we first calculate the vector normals of "roof" triangles, and then calculate the eigenvalues of their direction cosines. For simple roof shapes the smallest eigenvector will give the primary roof orientation, and the pattern of normals indicates the roof form. This information may then be integrated into the surface model.

References

Braid, I.C., Hilyard R.C. and Stroud, I.A., 1978. Stepwise construction of polyhedra in geometric modelling. In: Brodlie, K.W. (ed.), Mathematical Methods in Computer Graphics and Design, Harcourt Brace Jovanovitch, Leicester, p. 123-141.

Dakowicz, M. and Gold, C. 2002. Extracting Meaningful Slopes from Terrain Contours. In Proceedings of Computational Science - ICCS 2002, Lecture Notes in Computer Science vol. 2331, (Amsterdam, The Netherlands), p. 144-153.

Guibas, L. and Stolfi, J, 1985. Primitives for the manipulation of general subdivisions and the computations of Voronoi diagrams. ACM Transactions on Graphics 4(2), p. 74-123.

Roggero, M. 2002. Airborne laser scanning: clustering in raw data. In Proceedings of IAPRS (Annapolis, MD), vol XXIV-3/W4, p. 227-232.

Rottensteiner, F. and Briese, C. 2003. Automatic Generation of Building Models from LIDAR Data and the Integration of Aerial Images. In Proceedings of the ISPRS working group III/3 workshop '3-D reconstruction from airborne laserscanner and InSAR data' (Dresden, Germany, Institute of Photogrammetry and Remote Sensing Dresden University of Technology).

Suveg, I. and Vosselman, G. 2004. Reconstruction of 3D Building Models from Aerial Images and Maps, ISPRS Journal of Photogrammetry & Remote Sensing, 58(3-4), p. 202-224.

Tse, R.O.C. and Gold, C.M. 2002. TIN Meets CAD - Extending the TIN Concept in GIS. In Proceedings of Computational Science - ICCS 2002, International Conference, Proceedings of Part III. Lecture Notes in Computer Science (Amsterdam, the Netherlands), p. 135-143.

Vosselman, G. 2000. Slope based filtering of laser altimetry data. In Proceedings of IAPRS (Amsterdam, The Netherlands), XXXIII, Part B3, p. 935-942.

Weiler, K.J., 1988. Boundary graph operators for non-manifold geometric modelling topology representations. Geometric Modelling for CAD Applications, 1988.

Algorithms for cartograms and other specialized maps

Bettina Speckmann
Department of Mathematics and Computer Science
TU Eindhoven
speckman@win.tue.nl

Abstract

Automated cartography generates a large number of challenging algorithmic problems which fall squarely into the area of computational geometry. In this note we review several algorithms for specialized map construction that were generated by the computational geometry community. We will focus in particular on the construction of (rectangular) cartograms, but also discuss algorithms for proportional symbol maps.

1 Introduction

Computational Geometry [6] is a branch of theoretical computer science that considers the design and analysis of algorithms and data structures for problems that are geometric in nature. It is a very successful area of algorithms research and experienced a rapid growth during the last two decades. The methods, data structures, and algorithms developed by the computational geometry community find application in many areas—which include robotics, databases, computer graphics, molecular biology, and geographic information systems. Automated cartography in particular generates a large number of challenging algorithmic problems—including cartographic generalization, label placement, and specialized map construction—which fall squarely into the area of computational geometry.

The approaches and algorithms developed by computational geometers tend to be combinatorial in nature but sometimes involve iterative elements as well. In this note we review several algorithms for specialized map construction that were generated by the computational geometry community. We will focus in particular on the construction of (rectangular) cartograms, but also discuss algorithms for proportional symbol maps.

2 Cartograms

Cartograms are a useful and intuitive tool to visualize statistical data about a set of regions like countries, states or counties. The size of a region in a cartogram corresponds to a particular geographic variable. The most common variable is population: in a population cartogram, the sizes of the regions are proportional to their population. In a cartogram the sizes of the regions are not the true sizes and hence the regions generally cannot keep both their shape and their adjacencies. A good cartogram, however, preserves the recognizability in some way.

Globally speaking, there are four types of cartogram. The standard type—also referred to as contiguous area cartogram—has deformed regions so that the desired sizes can be obtained and the adjacencies kept. Algorithms for such cartograms were given, among others, by Tobler [23], Dougenik et al. [9], Kocmoud and House [15], Edelsbrunner and Waupotitsch [10], and Keim et al. [14]. We will provide more details on the combinatorial algorithm of Edelsbrunner and Waupotisch in the following subsection.

The second type of cartogram is the non-contiguous area cartogram [11, 18]. The regions have the true shape, but are scaled down and generally do not touch anymore. Sometimes the scaled-down regions are shown on top of the original regions for recognizability. A third type of cartogram is based on circles and was introduced by Dorling [8]. The fourth type of cartogram is the rectangular cartogram introduced by Raisz in 1934 [20]. Each region is represented by a single rectangle, which has the great advantage that the sizes (area) of the regions can be estimated

much better than with the first two types. However, the rectangular shape is less recognizable and it imposes limitations on the possible layout. Hybrid cartograms of the first and fourth type exist as well. Here, regions are rectilinear polygons with a small number of vertices instead of rectangles.

Quality criteria. Whether a (contiguous) cartogram is good is determined by several factors. One of these is the *cartographic error* [9, 10], which is defined for each region as $|A_c - A_s| / A_s$, where A_c is the area of the region in the cartogram and A_s is the specified area of that region, given by the geographic variable to be shown. The following list summarizes the most important quality criteria:

- Average cartographic error.
- Maximum cartographic error.
- Correct adjacencies.
- Suitable relative positions.
- Original shape is recognizable or maximum aspect ratio is bounded (rectangular cartograms).

These requirements are conflicting and we cannot expect to simultaneously satisfy all criteria well.

2.1 A combinatorial approach to contiguous cartograms

Edelsbrunner and Waupotitsch [10] presented a combinatorial approach to cartogram construction. Their algorithm is based on simplicial complexes in the plane. It matches regions with a surplus of area with regions that have a deficit and uses paths of triangles to define deformations that let one region grow at the cost of another one shrinking (see Fig. 1). Unfortunately this approach does not produce visually pleasing output if all inflations and deflations are performed in one step. Therefore the authors used a heuristic together with their combinatorial approach which changes the area of each region by only a small amount during each iterative step.

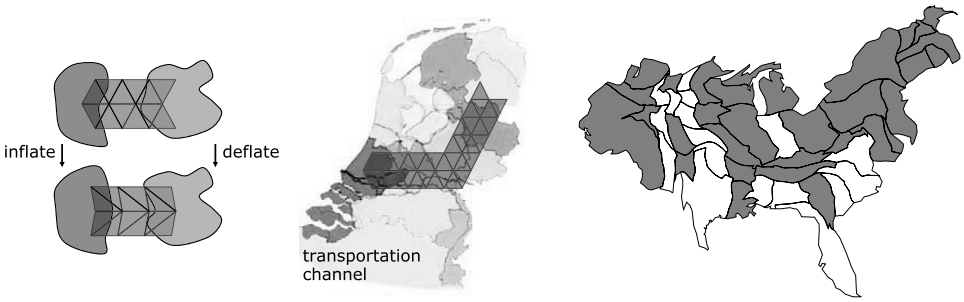

Figure 1: Inflating and deflating regions (left); transporting area via a path of triangles (middle); the electorial votes in the 1992 US presidential election, the shaded states show a majority for Bill Clinton (from Edelsbrunner and Waupotitsch [10]).

2.2 Rectangular cartograms

Rectangular cartograms are closely related to *floor plans* for electronic chips. Floor planning aims to represent a plane graph $\mathcal{G}$ by its *rectangular dual* which is a partition of a rectangle into rectangular regions whose dual graph is $\mathcal{G}$ (see Fig. 2 right). If the graph $\mathcal{G}$ is a triangulated plane graph without separating triangles—a separating triangle is a 3-cycle with vertices both inside and outside the cycle—then a rectangular dual always exists [2, 16] and can be computed in linear time [13].

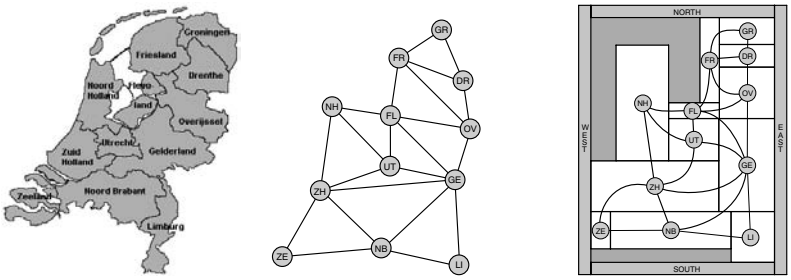

Figure 2: The provinces of the Netherlands, their adjacency graph, a population cartogram—here additional "sea rectangles" were added to preserve the outer shape.

The dual or adjacency graph of most maps is a triangulated graph, because usually at most three regions meet at any one point. Separating triangles occasionally arise, for example, Luxembourg does not border any sea and is incident to only three countries. This implies that a purely rectangular cartogram with correct adjacencies does not exist for Europe. Also note that although every triangulated plane graph without separating triangles has a rectangular dual this does not imply that an error free cartogram for this graph exists.

Tobler states in a recent survey, "Thirty-five years of computer cartograms" [24], that none of the existing cartogram algorithms are capable of generating rectangular cartograms. However, even more recently van Kreveld and Speckmann presented the first algorithms for rectangular cartogram construction [17, 22]. These algorithms are iterative and semi-combinatorial and are sketched below.

Algorithmic Outline. Assume that we are given an administrative subdivision into a set of regions. The adjacencies of the regions can be represented in a graph $\mathcal{G}$, which is the face graph of the subdivision.

1. Preprocessing: The face graph $\mathcal{G}$ is in most cases already triangulated (except for its outer face). In order to construct a rectangular dual of $\mathcal{G}$ we first have to process internal vertices of degree less than four and then triangulate any remaining non-triangular faces.

2. Directed edge labels: Any two nodes in the face graph have at least one direction of adjacency which follows naturally from their geographic location. While in theory there are four different directions of adjacency any two nodes can have, in practice only one or two directions are reasonable and, in fact, there is only a small number of adjacent regions where more than one direction is reasonable. The algorithms go through all possible combinations of direction assignments and determine which one gives a correct or the best result. We call a particular choice of adjacency directions a *directed edge labeling.*

Observation 1 *A face graph F with a directed edge labeling can be represented by a rectangular dual if and only if*

1. *every internal region has at least one North, one South, one East, and one West neighbor, and*
2. *when traversing the neighbors of a node in clockwise order starting at the western most North neighbor we first encounter all North neighbors, then all East neighbors, then all South neighbors and finally all West neighbors.*

A realizable directed edge labeling constitutes a *regular edge labeling* for F as defined in [13] which immediately implies our observation.

Figure 3: Highway kilometers of the US; the population of Europe (country codes according to the ISO 3166 standard).

3. Rectangular layout: To actually represent a face graph together with a realizable directed edge labeling as a rectangular dual we have to pay special attention to the nodes on the outer face since they may miss neighbors in up to three directions. To compensate for that we add four special regions NORTH, EAST, SOUTH, and WEST, as well as *sea rectangles* that help to preserve the original outline of the subdivision. Then we can employ the algorithm by He and Kant [13] to construct a *rectangular layout*, i.e., the unique rectangular dual of a realizable directed edge labeling.

4. Area assignment: For a given set of area values and a given rectangular layout we would like to decide if an assignment of the area values to the regions is possible without destroying the correct adjacencies. Should the answer be negative or should the question be undecidable, then we still want to compute a cartogram that has a small cartographic error while maintaining reasonable aspect ratios and relative positions.

In [17] three algorithms are described that compute a cartogram from a rectangular layout. This work is extended in [22] where a fourth algorithm is introduced. The first algorithm presented in [17] is the simple segment moving heuristic, which loops over all maximal segments in the layout and moves each with a small step in the direction that decreases the maximum error of the adjacent regions. After a number of iterations, one can expect that all maximal segments have moved to a locally optimal position. However, there is no proof that the method reaches the global optimum or that it even converges. Secondly, if the rectangular layout is L-shape destructible (see [17]) then one can compute a zero-error cartogram if one exists. The third method is based on bilinear programming and can produce a cartogram with minimum maximal error, provided a good bilinear program solver is available. Unfortunately, bilinear programs are notoriously difficult and none of the available solvers is guaranteed to work [1].

In [22] area assignment is formulated as a linear program which takes only the vertical or only the horizontal segments into account. The algorithm then alternatingly solves a linear program for the vertical and the horizontal segments. The segment moving heuristic and the linear programming based approach were implemented—the latter using the well-known CPLEX program [4]. Both methods can relax the adjacency constraints to compute cartograms with lower cartographic error at the cost of mild disturbances in the adjacencies.

Although the segment moving heuristic often gives aesthetically pleasing cartograms with small error (see Figure 3 (left)), the linear programming based approach in general produces cartograms with a lower cartographic error, a smaller aspect ratio, and a better global shape. Furthermore, it is also able to handle L-shaped regions in a cartogram (see Figure 3 (right)) and to compute satisfactory cartograms of all countries of the World (see Figure 4), whereas the previous method often does not.

Figure 4: A population cartogram of the World .

2.3 Rectilinear cartograms

Rectilinear cartograms are a generalization of rectangular cartograms where regions can be rectangles, or L-shapes, or any other type of rectilinear polygon. Recall that even if a plane triangulated graph $\mathcal{G}$ has a rectangular dual then this does not imply that an error free cartogram for $\mathcal{G}$ exists. Figure 5 shows an example where correct adjacencies must be sacrificed in order to get a small cartographic error. However, if we allow the regions to be rectilinear polygons then a cartogram with zero cartographic error and correct adjacencies exists for any plane triangulated graph $\mathcal{G}$ and any assignment of areas to regions.

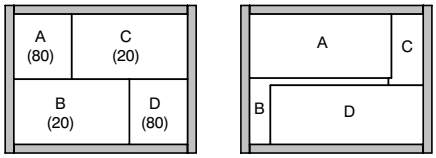

Figure 5: An input for which no rectangular cartogram has zero error and correct adjacencies.

More specifically, de Berg et al. [5] very recently proved the following. Let $\mathcal{G} = (V, E)$ be a plane triangulated graph—the adjacency graph of the input map—where each vertex is assigned a positive weight—the preferred area of the region that corresponds to this vertex. A rectilinear dual of $\mathcal{G}$ is a partition of a rectangle into $|V|$ simple rectilinear regions, one for each vertex, such that two regions are adjacent if and only if the corresponding vertices are connected by an edge in E. A rectilinear cartogram is a rectilinear dual where the area of each region is equal to the weight of the corresponding vertex.

Figure 6: Highway kilometers of the US; gross domestic product (GDP) of Europe.

Theorem 1 *Every vertex-weighted plane triangulated graph $\mathcal{G}$ admits a rectilinear cartogram of constant complexity, that is, a cartogram where the number of vertices of each region is constant.*

The proof presented in [5] is constructive and the underlying algorithm has been implemented. It produces regions of very small complexity—in fact, most regions are rectangles (see Figure 6).

3 Proportional Symbol Maps

Proportional symbols maps, also known as *graduated symbol maps*, are a well established cartographic tool to visualize quantitative data that is associated with specific (point) locations. A symbol, most commonly a disk or a square, is scaled such that its area corresponds to the data value associated with a point and then placed at exactly that point on a geographic map. The spatial distribution of the data can then be observed by studying the spatial distribution of the differently sized symbols. Typical data that are visualized in this way include the magnitude of earthquakes (see Fig. 7), the production of oil wells, or the temperature at weather stations.

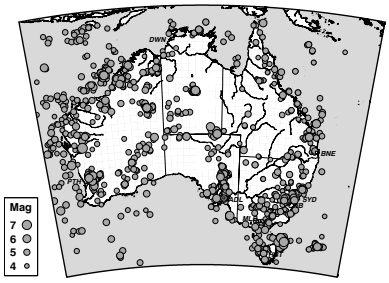

Figure 7: A proportional symbol map depicting Australian earthquakes of size > 4.0 on the Richter scale [19].

A proportional symbol map communicates its message via the sizes of its symbols—both the actual size of the symbols and the ratio between symbol sizes. There exists a large amount of theory and user studies that discuss which sizing communicates the difference between quantities in the most effective way (see the books by Dent [7] and Slocum et al. [21] for an extensive overview). While it is commonly agreed upon that a map should appear "neither 'too full' nor 'too empty'" it is unclear how much the symbols on a proportional symbol map should overlap, but every "good" map will contain at least some overlapping symbols.

In principle any two- or three-dimensional shape can be used as a symbol on a proportional symbol map. However, circles (transparent) and disks (opaque) are used most frequently, since they are visually stable, they conserve map space, and users do prefer them. Although opaque symbols obscure each other and also the map below them, users indicate [12] a preference for opaque symbols. Clearly there are many different ways to arrange opaque symbols with respect to each other and any choice of (partial) order makes some symbols more visible than others. Cabello et al. [3] very recently studied the algorithmic question how to arrange a given set of overlapping disks such that all of them can be seen "as well as possible". Below we first give a more detailed problem description and then sketch some of their results.

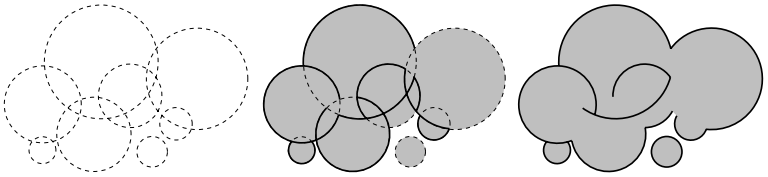

Figure 8: An arrangement $\mathcal{S}$, a drawing with $\mathcal{S}$ visible, and a bounded drawing.

Definitions and notation. Let S be a set of n disks $D_1, \ldots, D_n$ in the plane. We denote by $\mathcal{S}$ the arrangement formed by the boundaries of the disks in S. A *drawing* $\mathcal{D}$ of S is a sub-arrangement of $\mathcal{S}$ which is drawn on top of the filled interiors of the disks in S. Not every drawing is suitable for the use on a proportional symbol map. A suitable drawing needs to include at least the boundary of the union of the disks in S. It should be locally correct at the vertices: every vertex v of the drawing is formed by the intersection of the boundaries of two disks D_i and D_j; a drawing is locally correct at v if it corresponds locally around v to stacking D_i onto D_j or vice versa. Furthermore, a suitable drawing must have only correct faces: a face of the drawing is correct if there is an order in which all disks in S that contain the face can be drawn on top of each other such that the face appears. We call drawings that satisfy these conditions *face correct*.

Figure 9 shows that even a face correct drawing can still have an "Escher-like" quality which we would like to avoid on a proportional symbol map. Hence we need to enforce even stronger requirements on what constitutes a proper drawing. We consider two types of drawings.

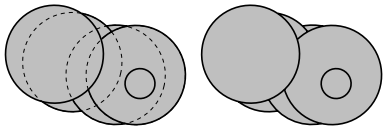

Figure 9: A face correct drawing shown with and without $\mathcal{S}$ visible.

Physically realizable drawings. A face correct drawing is physically realizable if and only if for every face f of $\mathcal{S}$ there exists a total order on the disks in S_f (the disks in S that contain f) such that the topmost disk is visible and the orders associated with any two faces of $\mathcal{S}$ do not conflict. That is, the order in which the disks in S are stacked upon each other is uniquely determined at every face of $\mathcal{S}$ and no two of such orders conflict. In particular, any two or more disks that have a common intersection have a unique ordering.

Stacking drawings. A stacking drawing is a natural restriction of a physically realizable drawing and also the one most frequently found on proportional symbol maps. A physically realizable drawing $\mathcal{D}$ is a stacking drawing if there exists a total order on the disks in S such that $\mathcal{D}$ is the result of stacking the disks in this order.

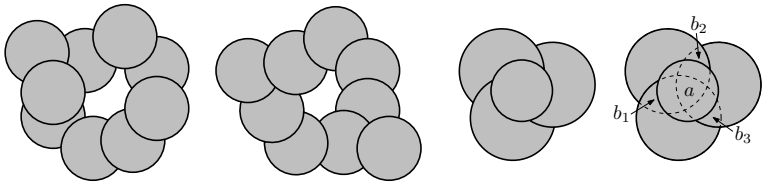

Figure 10: A stacking drawing (left), a physically realizable drawing that is not a stacking drawing (middle), a drawing that may seem physically realizable, but is not—any order for face a will conflict with one of b_1, b_2, or b_3 (right).

Quality of a drawing. Intuitively, a good drawing should enable the viewer to see at least some part of all disks and to judge their sizes as correctly as possible. The accuracy with which the size of a disks can be judged is proportional to the portion of its boundary that is visible. This leads us to the following two optimization problems. Assume that we are given a set S of n disks $S_1, \ldots S_n$.

Max-Min: Find a physically realizable or a stacking drawing that maximizes the minimum visible boundary length of each disk, that is, $\max \min_{1 \leq i \leq n} \{\text{visible length of the boundary of } S_i\}$.

Max-Total: Find a physically realizable or a stacking drawing that maximizes the total visible boundary length over all disks.

Figure 11 illustrates why we consider only visible boundary length and not visible area of disks. The boundary of the center disk is completely covered but a significant part of its area is still visible. It is, however, impossible to judge its size or to determine the location of its center. Figure 12 shows that a stacking drawing can be arbitrarily much worse than a physically realizable drawing with respect to the Max-Min problem. At least half of the boundary of every disk in Figure 12 (left) is visible, whereas the lowest disk in any stacking drawing is covered by its two neighbors and hence has only a very short visible boundary.

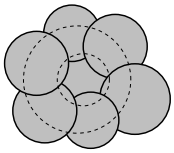

Figure 11: Visible perimeter is more important than visible area.

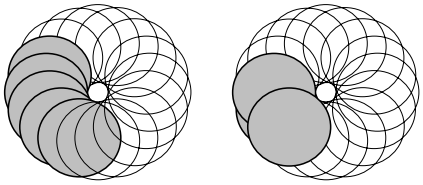

Figure 12: An optimal physically realizable drawing (left), an optimal stacking drawing for the same disks (right).

Cabello et al. [3] showed that for physically realizable drawings both the Max-Min and the Max-Total problems are *NP-hard*, that is, they belong to a class of problems that can—most likely—not be solved in polynomial time. But the Max-Min problem for stacking drawings can be solved in $O(n^2 \log n)$ time by the following approach: repeatedly choose the disk that has most of its boundary visible if it were to be placed at the bottom. This disk can be determined in $O(n \log n)$ time by the clever use of augmented segment trees. If no point in the plane is covered by more than a constant number of disks then the problem can even be solved in $O(n \log n)$ time because the arrangement of the disks has only linear complexity in this case. There is currently no efficient algorithm known for the Max-Total problem for stacking drawings. Cabello et al. also implemented four different methods for computing stacking orders and experimentally evaluated their quality. Their solution to the Max-Min problem performed best also with respect to the visual quality of the maps.

4 Conclusions

This note discussed several algorithms for specialized map construction that were generated by the computational geometry community. In particular, it described algorithms for (rectangular) cartograms and proportional symbol maps. Automated cartography is a rich source of interesting and challenging geometric problems that brings together researchers from cartography and computer science. There is a wealth of problems that remain to be explored—for example, related to animated and 3D cartography—which will ensure a continuation of this fruitful collaboration.

References

[1] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming - Theory and Algorithms*. John Wiley & Sons, Hoboken, NJ, 2nd edition, 1993.

[2] J. Bhasker and S. Sahni. A linear algorithm to check for the existence of a rectangular dual of a planar triangulated graph. *Networks*, 7:307–317, 1987.

[3] S. Cabello, H. Haverkort, M. van Kreveld, and B. Speckmann. Algorithmic aspects of proportional symbol maps. In *Proc. 14th European Symposium on Algorithms*, 2006. To appear.

[4] CPLEX: High-performance software for mathematical programming and optimization. `http://www.ilog.com/products/cplex/`.

[5] M. de Berg, E. Mumford, and B. Speckmann. On rectilinear duals for vertex-weighted plane graphs. In *Proc. 13th Int. Sympos. on Graph Drawing*, number 3843 in LNCS, pages 61–72, 2005.

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

[7] B.D. Dent. *Cartography - thematic map design*. McGraw-Hill, 5th edition, 1999.

[8] D. Dorling. *Area Cartograms: their Use and Creation*. Number 59 in Concepts and Techniques in Modern Geography. University of East Anglia, Environmental Publications, Norwich, 1996.

[9] J. A. Dougenik, N. R. Chrisman, and D. R. Niemeyer. An algorithm to construct continous area cartograms. *Professional Geographer*, 37:75–81, 1985.

[10] H. Edelsbrunner and E. Waupotitsch. A combinatorial approach to cartograms. *Comput. Geom. Theory Appl.*, 7:343–360, 1997.

[11] S. Fabrikant. Cartographic variations on the presidential election 2000 theme, 2000. `http://www.geog.ucsb.edu/~sara/html/mapping/ election/map.html`.

[12] T.L.C. Griffin. The importance of visual contrast for graduated circles. *Cartography*, 19(1):21–30, 1990.

[13] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theor. Comp. Sci.*, 172:175–193, 1997.

[14] D.A. Keim, S.C. North, and C. Panse. Cartodraw: A fast algorithm for generating contiguous cartograms. *IEEE Trans. Visu. and Comp. Graphics*, 10:95–110, 2004.

[15] C.J. Kocmoud and D.H. House. A constraint-based approach to constructing continuous cartograms. In *Proc. Symp. Spatial Data Handling*, pages 236–246, 1998.

[16] K. Koźmiński and E. Kinnen. Rectangular dual of planar graphs. *Networks*, 5:145–157, 1985.

[17] M. van Kreveld and B. Speckmann. On rectangular cartograms. In *Proceedings 12th European Symposium on Algorithms*, number 3221 in LNCS, pages 724–735, 2004.

[18] J.M. Olson. Noncontiguous area cartograms. *Prof. Geographer*, 28:371–380, 1976.

[19] Queensland University Advanced Centre for Earthquake Studies. `http://www.quakes.uq.edu.au`.

[20] E. Raisz. The rectangular statistical cartogram. *Geogr. Review*, 24:292–296, 1934.

[21] Terry A. Slocum, Robert B. McMaster, Fritz C. Kessler, and Hugh H. Howard. *Thematic Cartography and Geographic Visualization*. Prentice Hall, 2nd edition, 2003.

[22] B. Speckmann, M. van Kreveld, and S. Florisson. A linear programming approach to rectangular cartograms. In *Proc. 12th International Symposium on Spatial Data Handling*, pages 529–546, 2006.

[23] W. Tobler. Pseudo-cartograms. *The American Cartographer*, 13:43–50, 1986.

[24] W. Tobler. Thirty-five years of computer cartograms. *Annals of the Assoc. American Cartographers*, 94(1):58–71, 2004.

Constrained tetrahedral models and update algorithms for topographic data

Ir. F. Penninga
OTB Research Institute, section GIS Technology
Delft University of Technology
F.Penninga@otb.tudelft.nl

Introduction

Most current topographic products are limited to representing the real world in only two dimensions. As the real world exists of three dimensional objects, which are becoming more and more complex due to increasing multiple land use, accurate topographic models have to cope with the third dimension. The overall goal of this research is to extend current topographic modeling into the third dimension. Applications of 3D modeling are not limited to the terrain surface and objects built directly on top or beneath it, as geological features and air traffic or telecommunication corridors can be modeled too.

Most initiatives on developing 3D GIS focus on supporting visualization, often in Virtual Reality-like environments. One of the objectives of this 3D modeling research is to enable 3D analysis as well, as this traditional GIS-strength lacks until now in most 3D GIS approaches. Another important assumption within this research follows from the required wide variety of applications of topographic data. As topography is ranked high in the spatial data infrastructure hierarchy, one cannot optimize the data model for one specific purpose. One has to be able to serve the complete range of user applications, regardless whether these applications require for instance optimal visualization capabilities or optimal analytical capabilities.

3D Topography

The development of a 3D topographic data model is both demand and supply driven. 3D modeling is not only required for accurate modeling of increasingly complex real world objects, as is the case in multiple land use areas as illustrated in Figure 1.

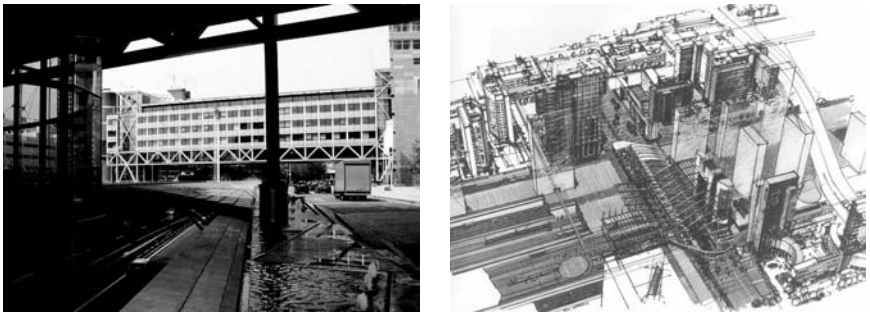

Figure 1. Multiple land use: offices at Utrechtse baan Den Haag (left) and plans for Amsterdam WTC (right).

Other important aspects in the increasing need for 3D modeling are the rising awareness of the importance of sustainable urban environments (requiring 3D planning and 3D analysis) and the need for better data for emergency services and disaster response. Disaster management (both natural and non-natural disasters) gained a lot of attention since 9/11, the Christmas 2004 tsunami in Asia and the flooding after hurricane Kathrina in New Orleans.

On the demand side the availability of the AHN (Actueel Hoogtebestand Nederland), a high density height point data set obtained with airborne laser altimetry, is an important factor in the development of 3D models. Figure 2 shows a part of the AHN in Zuid-Limburg. As the AHN contains on average one height point for every $16m^2$ the spatial resolution is high enough to enable extraction of for instance building heights. Building extraction can even be automated when laser scan data of higher resolution is available. Due to current developments in laser scanning technology laser scanners can measure up to 50 times more points per second compared to the mid-nineties, when the AHN resolution was chosen (Vosselman, 2005). Also terrestrial laser scanning offers the possibility to model objects in 3D with more detail, even indoor topography can be introduced.

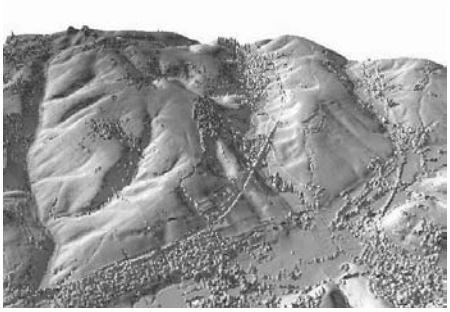

Figure 2. AHN offers high resolution height data.

3D modeling

Selection of 3D primitive

In 3D modeling one needs a 3D primitive (a volume) beside points, lines and faces to represent 3D objects accurately. Earlier research proposed amongst others using simplexes (point, line, triangle, tetrahedron) (Carlson 1987), points, lines, surfaces and bodies (3D Formal Data Structure (FDS)) (Molenaar 1990, Molenaar 1992), combining Constructive Solid Geometry (CSG) and a B-rep. (de Cambray 1993) and integrating a 2.5D Triangulated Irregular Network (TIN) with 3D FDS (Pilouk 1996). In applications polyhedrons are often used as 3D primitive (Zlatanova 2000, Stoter 2004).

In this research simplexes are the primitives of choice. A great advantage of using these simplexes is the well-defined character of the mutual relationships: a kD simplex is bounded by k+1 (k-1)D-simplexes (Pilouk 1996). This means that for instance a 2D simplex (a triangle) is bounded by three 1D simplexes (edges) and a 3D simplex (tetrahedron) is bounded by four 2D simplexes (triangles). The second important advantage of simplexes is the flatness of the faces, which enables one to describe a face using only three points. The third advantage is that every simplex, regardless its dimension, is convex, thus making convexity testing unnecessary. This quality simplifies point-in-polygon test significantly. The price for this comes with increased modeling complexity. Compared to for instance using polyhedrons as 3D primitive it will be clear that there exists a 1:1 relationship between a 3D feature (for instance a building) and its

representation (the polyhedron), but that there will be a 1:n relationship between this 3D feature and its tetrahedrons. However, as long as one is able to hide this complexity from the average user, the advantages will overcome this drawback. To further illustrate the strength of using well-defined primitives, consider a real estate tax application that determines the tax assessment based on the volume of the building. In order to automate this process, a formula for determining volumes is required. Designing a formula capable of determining a polyhedron's volume is more complex due to the unlimited variation in shape. Contrarily, implementing a formula for the volume of a tetrahedron is straightforward, it only has to be applied several times as a building will be represented as a set of tetrahedrons. This repetition is however exactly what computers are good for. As a result the TEN (Tetrahedronized Irregular Network) is selected as data structure.

Modeling concept

The development of the current modeling concept is described in (Penninga, 2005). The modeling concept is based on two basic observations:

- The ISO 19101 Geographic information - Reference model defines a feature as an 'abstraction of real world phenomena'. These real world phenomena have by definition a volumetric shape. In modeling often a less-dimensional representation is used in order to simplify the real world. Fundamentally there are no such things as point, line or area features; there are only features with a point, line or area representation (at a certain level of abstraction/generalization).
- The real world can be considered to be a volume partition. A volume partition can be defined (analogously to a planar partition) as a set of non-overlapping volumes that form a closed modeled space. As a consequence objects like 'air' or 'earth' are explicitly part of the real world and thus have to be modeled.

As a result the real world features will be modeled as volumes (set of tetrahedrons) in a TEN structure. The option to represent certain feature types in less dimensional representations belongs in the digital cartographic model and not already in the digital landscape model. Based on this one might wonder whether less-dimensional representations are even allowed in the new modeling approach, for instance using a face instead of a volume. The answer is positive, but only in special cases. Looking at the real world one can see that the features that are represented by faces are actually marking a border between two volume objects. For instance an area labeled as 'wall' might still be represented as a face, despite its actual thickness in reality. However it's important to realize that this face marks the transition between two volumes: 'building' at one side and 'air' at the other side of the face. In the new modeling approach these volumes play a central role. The faces marking the borders between volumes might still be labeled, for instance as 'wall' or 'roof', but semantically they do not bound the building anymore, as the building in itself is represented by a volume, with neighboring volumes that represent air, earth or perhaps another adjacent building. One can say that area features, such as walls, are derivatives of volume features, as they cannot exist without the presence of these volume features. At this time it is not decided yet whether only first order derivatives (area features) are useful or that second (line features) and third order derivates (point features) should be supported too.

The UML class diagram of the proposed method is shown in Figure 3. The concept of derived features is modeled by treating these classes as association classes. For instance an area feature is an association class between two (adjacent) volume features. Also visible in the UML class diagram is that features are stored as constraints in the TEN structure. On algorithm level these constraints are all on the edge level, as current triangulation / tetrahedronization algorithms are

only capable of handling constraint edges. A third aspect that is shown in the diagram is that every tetrahedron should represent a volume feature, whereas for instance not every triangle represents an area feature. This full volume partition idea results in modeling 'air' and 'earth' in between of topographic features as well.

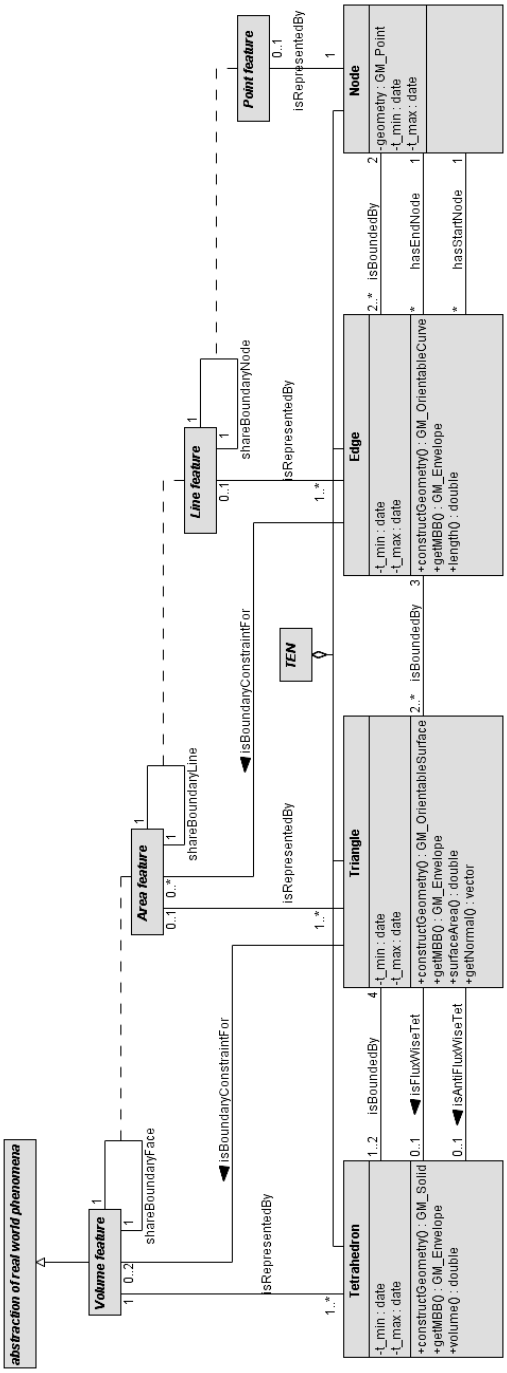

Figure 3. UML class diagram of the 3D Topography TEN model.

At this point it might seem that also modeling 'air' and 'earth' in addition to all common topographic features is a very rigid approach of modeling, more serving the abstract goal of 'clean' modeling than an actual useful goal. This is however not the case. These air and earth objects do not just fill up the space between features of the other types, but are often also subject of analyses, such as noise and odor modeling and flooding analysis. Another great advantage is the flexibility introduced by these features, as they enable future extensions of the data model. Figure 4 shows some examples of features that can be included in the model in the future, such as air traffic corridors and petroleum reservoirs.

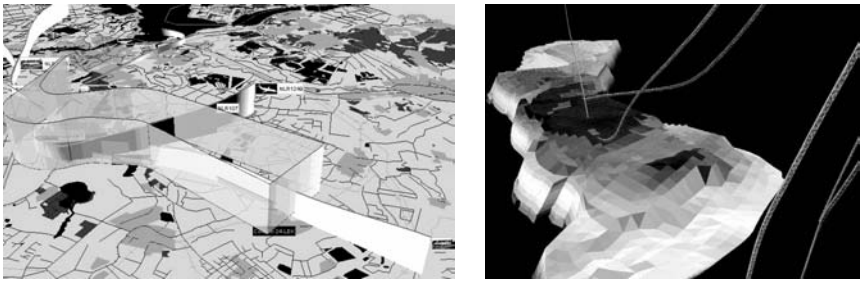

Figure 4. Air traffic corridors near Schiphol Amsterdam Airport (left) and a 3D petroleum reservoir (Ford and James, 2005) (right).

Another possible future extension is the addition of indoor topography to the model. Information on the basic interior layout of a building can be very useful for emergency services, especially the fire brigade. Indoor geo information will become more and more available due to the introduction of terrestrial laser scanning. This technique also enables measurements in complex 3D situations such as highway interchanges, see Figure 5.

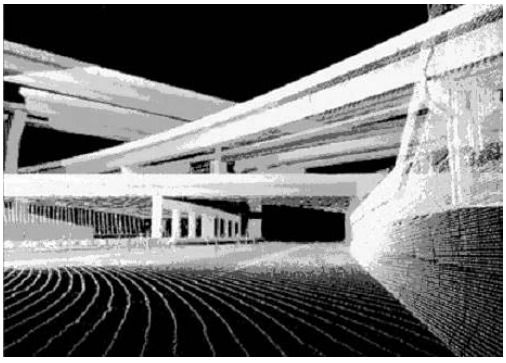

Figure 5. Scan of highway interchange obtained by terrestrial laser scanning.

3D Topography modeling: implementation

In the presented 3D Topography TEN model the topographic features are stored as constraints in the tetrahedronization. The creation of the model starts with four initial tetrahedrons, as can be seen in Figure 6: two air and two earth tetrahedrons with an initial earth surface. As a first step height data from digital elevation models will be used to refine the earth surface. If necessary ill-shaped tetrahedrons will be reshaped and Steiner points might be added. The last step is

to insert separately tetrahedronized topographic features into the model. Therefore an incremental algorithm for updating the TEN model is required.

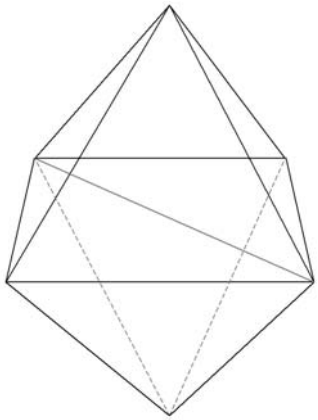

Figure 6. The four initial tetrahedrons (two 'air' and two 'earth').

3D Topography and Computational Geometry

Amount of data

If one considers the tetrahedronization of the building in Figure 7, it will be clear that storing the building in a TEN requires a lot of storage. In Table 1 the required number of tetrahedrons, triangles, edges and nodes is compared to the number of volumes, faces, edges and points in a polyhedron approach.

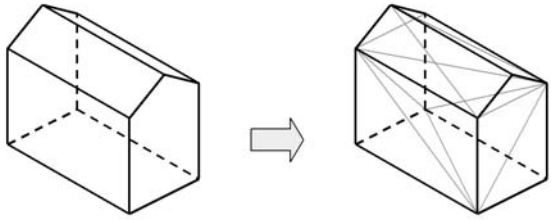

Figure 7. Tetrahedronized building.

Building as polyhedron	Building as TEN
(1 volume)	8 tetrahedrons
7 faces	24 triangles
(15 edges)	25 edges
(10 points)	10 nodes

Table 1. Comparison between polyhedron and TEN model of the building.

In order to reach acceptable performance it has to be decided which relationships (as modeled in the class diagram in Figure 3) will be stored explicitly, as performance requirements do not tolerate full storage of all possible relationships. Several approaches exist in 2D to reduce storage

requirements of TINs by either working with an edge- or a triangle based approach, in which not both triangles, edges and nodes are stored explicitly. However, in the 3D situation and in the case of constraints in the TEN this is very difficult.

Updating the topography model: requirements from a computational geometry perspective

In the developed data model all data is stored in a spatial database. The most straight forward implementation consists of four tables with nodes, edges, triangles and tetrahedrons and a table with volume features. The set of representing simplexes is stored in this last table for every feature, so for the building from Figure 7 references are present to the eight tetrahedrons. To ensure the correct representation of the building in the TEN model one needs to enforce the boundary faces of this building to be present. As triangulation algorithms can only handle constrained edges, so a set of constrained edges is required. This set of constrained edges forms a complete surface triangulation of the building.

If one wants to remove this building, for instance because it is demolished, the record from the volume feature table can be deleted. At the same time the TEN needs to be updated. The constraints on the edges of the surface triangulation can be removed, but only if this building is the only feature that is bounded by this constrained edge. In the case of the demolished building constrained edges on the building's floor also bound the earth surface and therefore need to remain present in the TEN model. The tetrahedrons that were previously representing the building now need to be re-classified, in this case most likely just as 'air'. This reclassification is necessary to maintain the volume partition. At this moment the building is entirely removed from the model, both on TEN and on feature level, but the deletion process is not finished. As a last step it is necessary to check whether the TEN can be simplified by creating larger tetrahedrons or can be optimized by creating better-shaped tetrahedrons. As an alternative one might delete directly all edges that were part of the building, except for (constrained) edges that also contribute to the shape of other features. The resulting hole in the TEN needs to be re-triangulated and the created tetrahedrons will be linked to the 'air' feature.

If one wants to add a feature (for instance the same building) to the model its surface first needs to be triangulated. The resulting edges are the input for the tetrahedronization. This tetrahedronization is performed separately from the TEN network. The complete set of edges is then inserted into the TEN model by an incremental tetrahedronization algorithm. As a last step the volume feature table needs to be updated. A new record is created which links the building to the representing tetrahedrons and the previous 'air' tetrahedrons on the specific location are removed.

Now that the update process is described the algorithm requirements can be extracted. For creating and maintaining the TEN an incremental algorithm is required. Due to the potential enormous amount of data this incremental algorithm has to work in the database and should preferably impact the TEN structure as locally as possible. In the TEN all simplexes should be explicitly available, as the tetrahedrons represent volume features, the triangles contain most topological relationships, the edges contain the constraints and the nodes contain the geometry. Attempts to work for instance with implicit edges as is done with TINs would seriously complicate some of the required analysis or editing operations. Being able to store a TEN as compact as possible is might be nice, but in this 3D topography research interest is not only in maintaining a TEN but also in altering and querying the structure, which requires more functionality. Another requirement is the need for numerical stability through detection and repair of ill-shaped triangles

and tetrahedrons. Shewchuk has performed a lot of research (Shewchuk 1997, Shewchuk 2004) in the field of Delaunay mesh refinement in both two and three dimensions.

Conclusions

The volume approach in 3D topographic data modeling offers several advantages, amongst other good analytical and computational capabilities. However the TEN approach will lead to very large data sets. In order to overcome this drawback fast and reliable algorithms are required. The constrained tetrahedronized irregular network needs to be updated by an incremental algorithm that will also guarantee well-shaped triangles and tetrahedrons to avoid numerical instability. Despite the conceptual advantages of the 3D TEN approach the success of this approach completely depends on the degree in which the algorithms are capable of querying, analyzing and altering with acceptable performance.

References

Carlson, E., 1987. Three-dimensional conceptual modeling of subsurface structures. In: Auto-Carto 8, pp. 336-345.

de Cambray, B., 1993. Three-dimensional 3D) modeling in a geographical database. In: Auto-Carto 11, pp. 338-347.

Ford, A. and P. James, 2005. Integration of 3D petroleum datasets in commercial GIS. In: Agile 2005, 8th Conference on Geographic Information Science, pp. 411-418.

Molenaar, M., 1990. A formal data structure for three dimensional vector maps. In: 4th International Symposium on Spatial Data Handling, Zurich, pp. 830-843.

Molenaar, M., 1992. A topology for 3D vector maps. In: ITC Journal 2, pp. 25-33.

Penninga, F., 2005. 3D Topographic data modeling: Why rigidity is preferable to pragmatism. In: Spatial Information Theory, Cosit 2005, pp. 409-425.

Pilouk, M., 1996. Integrated Modeling for 3D GIS, PhD thesis, ITC Enschede.

Shewchuk, J.R., 1997. Delaunay refinement mesh generation, PhD thesis, Carnegie Mellon University.

Shewchuk, J., 2004. General-Dimensional Constrained Delaunay and Constrained Regular Triangulations I: Combinatorial Properties. To appear in: Discrete & computational Geometry. Available at: http://www-2.cs.cmu.edu/jrs.

Stoter, J., 2004. 3D Cadastre, PhD thesis, Delft University of Technology.

Vosselman, G., 2005. Sensing Geo-information, Inaugural address, ITC Enschede.

Zlatanova, S., 2000. 3D GIS for urban development, PhD thesis, Graz University of Technology.

Towards improved solution schemes for Monte Carlo simulation in environmental modeling languages

Derek Karssenberg and Kor de Jong

Department of Physical Geography, Faculty of Geosciences, Utrecht University, PO Box 80115, 3508 TC Utrecht, the Netherlands. Tel. +31 30 2532768, Fax +31 30 2531145,
Email: d.karssenberg@geo.uu.nl, http://pcraster.geo.uu.nl.

1.1. Introduction

Numerical environmental models simulating landscape changes through time with equations representing physical, chemical, or biological landscape processes (Karssenberg & De Jong, 2005; Wainwright & Mulligan, 2004) are important tools for environmental research, planning, and management. Although numerical environmental models have much in common with the group of software tools known as Geographical Information Systems (GIS, Burrough & McDonnell, 1998) – both environmental models and GIS deal with spatial data – there are at least two large differences. The first difference is in the number of dimensions represented. Most GIS systems restrict their data models to two spatial dimensions, whereas numerical environmental models need to deal with at least five data dimensions: three spatial dimensions, the time dimension, and a dimension to represent uncertainty in a Monte Carlo framework. The second difference between GIS and environmental models is the type of functions on the data required. While GIS is strong in functions for static analysis, management and presentation of vector and raster data, numerical environmental models need to incorporate functions representing spatio-temporal processes occurring in a landscape such as numerical solution schemes of differential equations. These functions need to deal with additional dimensions such as the time dimension.

As a result of these two differences, run times of numerical environmental models are often much larger than these of GIS applications, since numerical environmental models include relatively complicated functions on data volumes which are often even larger than in most GIS applications. So, minimizing run times is a very relevant issue when coding environmental models. This requires specialist knowledge in the disciplines of informatics, computational geometry, and numerical mathematics. Since model builders, which are environmental scientists with knowledge of landscape processes, often do not have this knowledge, model construction is preferably done with high level environmental modeling languages (Karssenberg, 2002). These are languages providing building blocks for model construction with built-in optimization schemes developed by specialists in the abovementioned disciplines. The environmental scientists construct their models by combining these building blocks instead of programming everything from scratch. An example of such a language is the PCRaster language (PCRaster, 2006) and the recently developed PCRaster Python library (Karssenberg & De Jong, subm.). Concepts and examples presented in this paper are taken from these languages although many other environmental modeling languages exist with similar concepts (c.f., Karssenberg, 2002).

This paper starts with a description of the multiple dimensions involved in numerical environmental modeling and functions required operating on data in these dimensions. The second part of the paper discusses future challenges for designing better stochastic environmental modeling languages that minimize runtimes involved in calculating parameters describing the sampling distributions of output variables generated by Monte Carlo simulation.

1.2. Stochastic dynamic spatial modeling

1.2.1. Process representation, discretisation of temporal and spatial dimensions

To mimic changes in a landscape through time, numerical environmental models use the rules of cause and effect, with a discretisation of time in time steps. The state of the model variables, which are attributes in one, two, or three dimensional space, at time t+1 is defined by their state at t and a function f. Similar descriptions can be found in (Beck, Jakeman, & McAleer, 1993; Karssenberg & De Jong, 2005; Wainwright & Mulligan, 2004):

$$Z_{1..m}(t+1) = \mathrm{f}(Z_{1..m}(t), I_{1..n}(t), t, P_{1..l}) \tag{1}$$

The model variable(s) $Z_{1..m}$ belong to coupled processes, and therefore have feedback in time, for instance stream water level. The model variable(s) $I_{1..n}$ are simple inputs to the model, for instance incident rainfall in a runoff model. The function f with associated parameters $P_{1..l}$ models the change in the state of all model variables over the time step t to t+1 and it can be either an update rule, explicitly specifying the change of the state variable over the time slice (t, t+1), for instance a rule based function such as cellular automata (e.g., Toffoli, 1989), or alternatively a derivative of a differential equation describing the change of the state variables as a continuous function (c.f. Wainwright & Mulligan, 2004). It may also include probabilistic rules when model behaviour is better described as a stochastic process. The function f is evaluated for each time step, which can be written in pseudo code as:

```
for each time step:                                    (2)
  evaluate f
```

To represent spatial variation, the model variables are attributes in two or three dimensional space, referred to as map variables or block variables, respectively (Figure 1). For representing continuous fields, most packages for environmental modeling discretise the lateral dimensions in regular grid cells. PCRaster discretises the third dimension with an irregular discretisation (ragged array) using voxels with variable thickness (Figure 1, Karssenberg & De Jong, 2005).

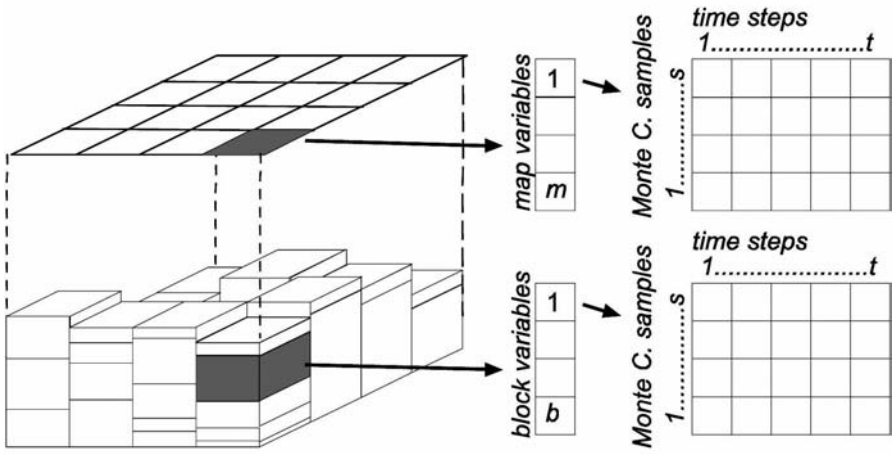

Figure 1. Representation of spatial dimensions, time, and Monte Carlo samples in environmental modeling. Left, map and block; centre, list of map or block variables; right, matrix with values for each time step and each Monte Carlo sample, stored for each map or block variable, for each cell or voxel.

To represent the change in state of a model over each time step, environmental modeling languages come with a library of preprogrammed functions on map and block variables. These functions may represent spatial interaction such as required in cellular automata models, lateral flow over a digital terrain model or simple processes without spatial interaction (c.f. Karssenberg & De Jong, 2005). The preprogrammed functions are the building blocks of the model: the model builder represents f in equation (1) by combining the functions in an iterative script. To illustrate this, consider the construction of a rainfall-runoff model that simulates surface runoff of water as a result of incident rainfall. The spatial variation in rainfall, runoff, infiltration, and other state variables is represented by using map variables. The processes involved generating surface runoff are programmed by combining functions on maps, in order to represent f in equation 1. By running these functions for each time step, a simulation can be made of the temporal change in rainfall, runoff, and related processes.

1.2.2. Stochastic modeling with Monte Carlo simulation

In many cases, environmental models include probabilistic rules or have inputs or parameters that are given as spatial probability distributions as is the case in error propagation modeling (c.f. Crosetto & Tarantola, 2001; Heuvelink, 1998). The aim of stochastic modelling is to derive the probability distributions (or parameters describing these) of the stochastic model variables $Z_{1..n}(t)$ from the stochastic inputs, parameters, or probabilistic rules, and to store the probability distributions of these model variables which are of interest, i.e. the model output variables. For complex environmental models, this can only be approximated with Monte Carlo simulation modelling (Heuvelink, 1998). Monte Carlo simulation involves two steps: Step (1). For each Monte Carlo sample: run f(·) (eq. 1) for all time steps with realizations of stochastic parameters or inputs, and store the realizations of the model output variables $Z_{1..n}$ in which the interest lies. Step (2). Compute descriptive statistics (e.g., mean, variance, quantiles) from the model outcomes of each sample, for each model output variable and each time step t, or for a selection of model variables and time steps. In pseudo code, Monte Carlo simulation for environmental models can be written as (see also Figure 2):

```
1     for each sample:
        for each timestep:                                    (3)
3          evaluate f
      compute descriptive statistics of output variables
```

This nested iteration will provide each variable and each cell or voxel with a value for each time step and Monte Carlo sample, which means that the array on the right side of Figure 1 is filled. In Monte Carlo simulation, the calculation of descriptive statistics typically involves an aggregation over the samples of the Monte Carlo dimension: for each cell or voxel, a statistical value such as the mean or a quantile is calculated from the values of the Monte Carlo samples for that cell or voxel (Figure 3C). The example presented in the previous section will make this clear. Consider the rainfall-runoff model predicting the change in surface runoff through time, for each grid cell. It is run in Monte Carlo mode now to represent uncertainty in incident rainfall. Each Monte Carlo sample represents a realization of the model with a possible spatial distribution of runoff. To calculate the uncertainty in the surface runoff for each time step and each cell, the model variable surface runoff is aggregated over the Monte Carlo dimension, by calculating for instance the variance of the surface runoff values over the Monte Carlo dimension. This is done for each cell and each time step.

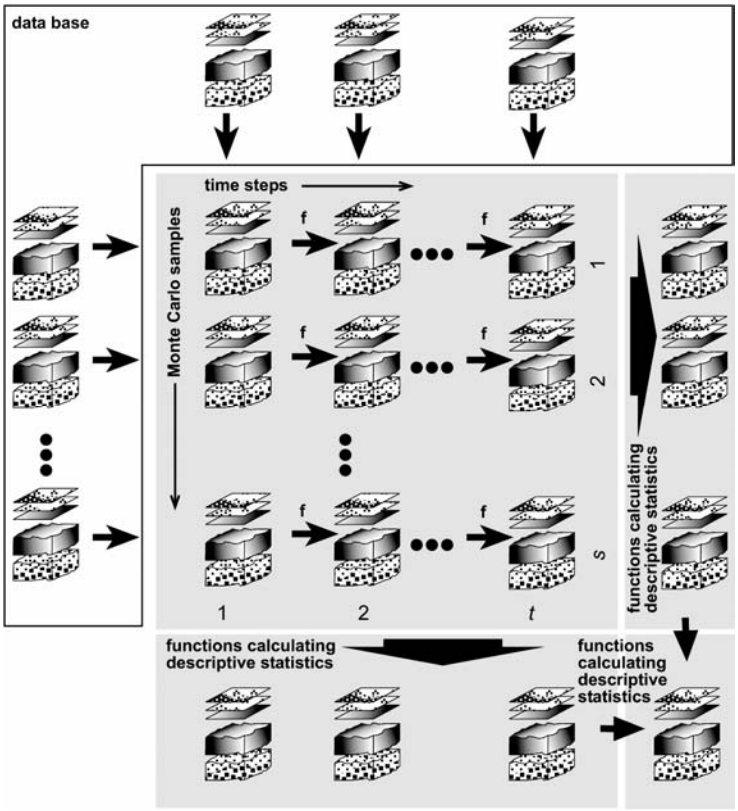

Figure 2. Concepts for temporal, two or three dimensional, and stochastic modeling.

In addition to the calculation of descriptive statistics over the Monte Carlo dimension, descriptive statistics can be calculated over time and/or over space. Consider for instance the calculation of peak runoff in the example rainfall-runoff model. This involves an aggregation over time (Figure 2B) where the maximum runoff is calculated over the time steps, for each cell and each Monte Carlo loop. Calculating average runoff values for different land use types is an example of aggregation over space (Figure 2A), where average values need to be calculated over a certain set of cells, for each time step and each Monte Carlo loop. Finally, aggregation needs to be done in certain cases over more than one dimension. Consider for instance the calculation of the median peak runoff, which involves an aggregation over time, calculating the peak runoff for each Monte Carlo sample, and an aggregation over the Monte Carlo dimension, calculating the median of the peak runoff values over the Monte Carlo loops.

The development of environmental modeling languages with built-in functionality for Monte Carlo simulation is still in its infancy. Recently, the PCRaster team developed the PCRaster Python library (Karssenberg & De Jong, subm.) which can be considered as one of the first languages that can do Monte Carlo simulation with dynamic spatial environmental models.

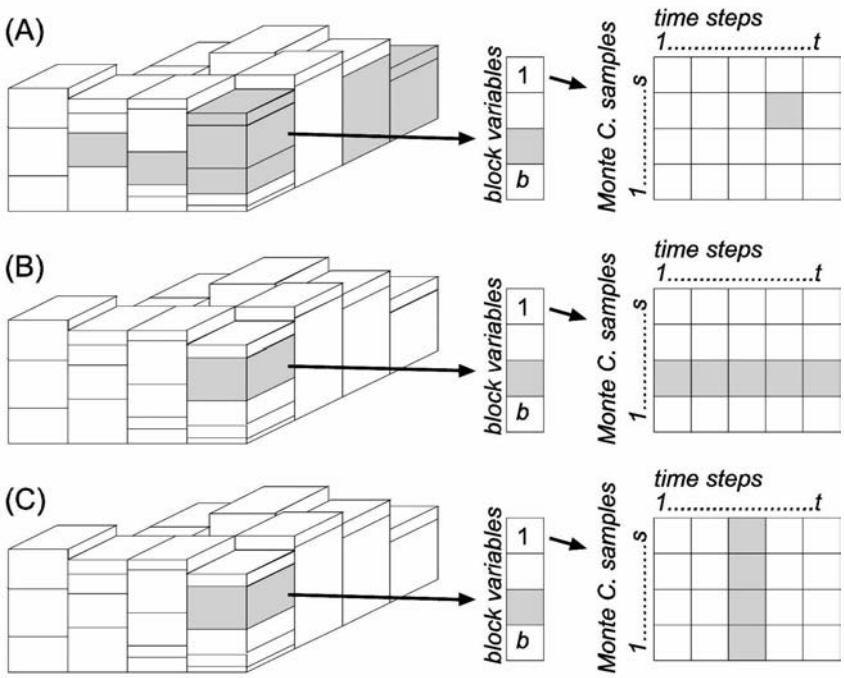

Figure 3. Functions calculating descriptive statistics, aggregating over A) space, B) time, C) the stochastic dimension, i.e., Monte Carlo samples.

1.3. Minimizing run times involved with Monte Carlo simulation

1.3.1. Examples of solution schemes reducing I/O

As noted in the introduction, the run times of environmental models can be large due to the relatively large number of calculations involved in evaluating f (equation 1), the large data sets used, and the large number of evaluations of f required, which is equal to the number of time steps times the number of Monte Carlo loops. Although decreasing run times of f is still a major challenge when designing new environmental modeling languages, we will focus our discussion here on the run times associated with the calculation of descriptive statistics. Unlike run times of f being mainly dependent on the speed of the processor, run times associated with calculating descriptive statistics are largely dependent on memory, since limited memory or inefficient use of memory may require I/O which will increase the run times significantly. An example will make this clear. A prototype implementation of the environmental modeling language known as the PCRaster Python Library (Karssenberg & De Jong, subm.) uses the following order of calculation for calculating descriptive statistics over the Monte Carlo dimension of a map variable.

```
1     for each sample:
2       for each time step:
3         evaluate f
4         write map variable to disk
5     for each cell:                                                    (4)
6       for each time step:
7         for each sample:
```

```
8           read cell value from disk and store in memory
9         compute descriptive statistics of variable
10        release cell values from memory
```

This approach writes the map variable for which statistics need to be calculated to hard disk (line 3 in the scheme above), storing the map variable as a separate file for each time step and each Monte Carlo sample. In line 5-10, these files are opened to calculate descriptive statistics of the variables: the scheme aggregates for each time step over the samples, one cell at a time. Since the calculation is done cell-by-cell, a file needs to be opened, read from, and closed for each cell, Monte Carlo sample and time step. This means that a file is opened a number of 10^7 up to 10^{15} times, which is calculated as $t \cdot s \cdot c$, with t, number of time steps (typically 10^1-10^4), s, number of samples (10^2-10^3), c, number of cells or voxels per variable (10^4-10^8). The advantage of this approach is that the number of cells, time steps and Monte Carlo samples is not limited by the memory available, since the amount of memory required equals $s \cdot m$, which is in the order of magnitude of 10^2 byte – 1 kB using the typical values given above and a memory per cell/voxel value (m) of 4 byte. The main disadvantage of this approach is the long run times since much time is spent on I/O.

A significant decrease of run times is reached when all data required for calculating statistics are kept in memory. Since an extensive evaluation of all possible approaches is beyond the aim of this paper, I provide two obvious approaches here as an invitation to the discussion below. The first one nests the time step loop inside the loop over the Monte Carlo samples:

```
1     for each sample:
2       for each time step:
3         evaluate f                                                    (5)
4         keep map variable in memory
5     for each time step:
6       compute descriptive statistics of variable
7       release cell values from memory for current time step
```

This approach aggregating for each time step over the samples, one map at a time, requires a memory of $t \cdot s \cdot c \cdot m$ which is 10 MB up to 100 TB (tera byte, 10^{12} Byte) using the typical values given above. In practice, most desk top computers will not have sufficient memory for this approach, although it will work when the data in one or more dimensions is small. An alternative approach would be to do the loop over the Monte Carlo samples nested inside the loop over the time steps:

```
1     for each time step:
2       for each sample:
3         evaluate f                                                    (6)
4         keep map variable in memory
5       compute descriptive statistics of map variable
6       release values from memory for current time step
```

This approach requires much less memory, equal to $s \cdot c \cdot m$, which is typically 1 MB – 100 MB.

1.3.2. Factors involved in choosing optimal solution schemes

The examples in the previous section show that the straightforward solution scheme (4) currently applied in the PCRaster Python Library is not in all cases the best scheme considering calculation time. In many cases, other schemes can be applied that do not involve I/O speeding up the calculation. It is a major challenge for future research to develop better environmental

modeling languages that select the best approach for calculating descriptive statistics given the model script developed by the model builder, the size of the input data, and the properties of the computer used. As a first step towards such a 'clever' environmental modeling language, I discuss below the main factors which are important in selecting the best approach for calculating descriptive statistics, whereby focus is on schemes that minimize I/O.

Type of descriptive statistics required. In the solution schemes (4-6) it was assumed that the calculation of a value describing the distribution of a variable requires that all data reside in memory. Although this is required for calculating the exact value of quantiles, this is not needed for the calculation of statistics such as mean or variance since these can be calculated on-line (incrementally). On-line calculation means that the statistical value is repetitively updated using single data values which are sequentially read and released from memory. When descriptive statistics need to be calculated that allow for on-line calculation, solution scheme (6) can be adjusted:

```
1     for each time step:
2       for each sample:
3         evaluate f                                                    (7)
4         keep map variable in memory
5         update descriptive statistics of map variable
6         release values from memory for current time step and sample
```

The size of the memory required in this scheme is $c \cdot m$, typically 10 KB - 100 MB, which is orders of magnitude smaller than the memory required for solution scheme (6). This example shows that the type of descriptive statistics that needs to be calculated is an important factor in choosing the solution scheme. In general, on-line methods for calculation of statistics will be preferable. When exact calculation of a certain statistic is not possible with an on-line method, it should be considered to include on-line methods that return approximations of the statistic (e.g., Pearl, 1981) in environmental modeling languages.

Size of the data in each of the dimensions. The memory required for each of the solution schemes depends on the number of time steps and Monte Carlo samples, the number of cells or voxels, and/or the number of variables for which descriptive statistics need to be calculated. Certain schemes are possible (and very fast) with a small number of data, while the same schemes may become impossible with large data sets due to memory overflow. Note that in many cases descriptive statistics are not required for all coordinates in all dimensions. For instance, calculating surface runoff at the outflow point of a catchment requires the calculation of descriptive statistics at a single cell, the cell corresponding to the outflow point, only. Other cells can be ignored in the solution scheme which will decrease the amount of memory required.

Dimensions over which aggregation is required. In error propagation modeling, calculation of descriptive statistics is in most cases required over the Monte Carlo dimension on a cell-by-cell basis, resulting in descriptive statistics for each time step and each cell (or voxel) as discussed so far. Sometimes, it may be required to calculate statistics over the spatial or temporal dimension, too. Aggregation over the spatial dimension require relatively little memory in the order of magnitude of $c \cdot m$ since it can be done directly after evaluating the function f:

```
1     for each sample:
2       for each time step:                                             (8)
3         evaluate f, keep map variable in memory
4         calculate spatial stats of variable and release from memory
```

Calculation of descriptive statistics over the time dimension can be done following (8) when calculation can be done on-line, for instance calculating the maximum value of runoff over the time steps, for each cell. When descriptive statistics over the time dimension need to be calculated that require to have all data in memory (e.g., quantiles), the map variable needs to be kept in memory over all time steps, requiring $t \cdot c \cdot m$ of memory:

```
1     for each sample:
2       for each time step:                                          (9)
3         evaluate f, keep map variable in memory
4       calculate spatial stats of variable and release from memory
```

Occurrence of multiple variables for which aggregation is required. In many cases, aggregation needs to be done for a number of different variables. Apart from increasing the total memory needed, this may complicate the solution schemes since conflicting solution schemes may be required for different variables. For instance, aggregation of a variable over the time dimension may require solution scheme (8), while aggregation over the Monte Carlo samples may require scheme (7). In this case, a solution scheme needs to be chosen by comparing efficiency of different possible schemes.

Occurrence of corresponding values. In some cases, a map or block variable may contain cells with exactly similar starting values and process equations, resulting in similar cell values over one, or multiple, dimensions. For instance, the amount of infiltration may be similar in all cells in a certain soil class, and as a result all cells in that class will have similar cell values. In this case, the calculation of descriptive statistics can be restricted to a single calculation for each soil class, instead of calculating statistics for each individual cell in the class. This principle can be used to decrease the amount of memory required in calculating descriptive statistics.

Memory requirements. In cases when the solution scheme requiring the smallest amount of memory results in a required memory that is larger than the available memory, a scheme needs to be applied that does I/O. Solution schemes need to be developed that result in shortest run times in this situation.

Memory required for the process model. In addition to the memory required for calculating descriptive statistics, the evaluation of the function f for each time step may need a large amount of memory, both for the evaluation of f itself and for storing variables at the end of each time step that are required as input to f for the next time step. So, by comparing different solution schemes, the amount of memory involved in the evaluation of f for each possible scheme needs to be considered, too. For instance, solution scheme (6) may be very efficient in terms of memory use related to calculating descriptive statistics, but it may not be as efficient as solution scheme (5) when memory is considered required for evaluating f. The point is that in scheme (5) all variables that need to be stored at the end of a time step as input to the next time step require to reside in memory for all Monte Carlo samples. This requires a memory in the order of magnitude of $s \cdot c \cdot m \cdot v$, with v, the number of variables that need to be stored.

1.4. Final remarks and conclusions

We showed that existing environmental modeling languages provide all functionality required for constructing models that simulate changes through time in two and three spatial dimensions. In addition, the PCRaster Python Library includes a framework for Monte Carlo simulation with these models, whereby functions can be used to calculate descriptive statistics of stochastic variables. Although this framework makes it much easier to do Monte Carlo simulation for environmental scientists compared to the situation whereby everything needs to be programmed

from scratch, the framework does not solve the problem of long runtimes associated with Monte Carlo simulation. It is a major challenge for the GIS research community to develop better environmental modeling languages that optimize, in terms of run times, the solution of stochastic models developed with these languages. The list of factors that need to be taken into account when choosing optimal solution schemes provided here should be considered as a first step towards such faster environmental modeling languages.

1.5. References

Beck, M. B., Jakeman, A. J., & McAleer, M. J. (1993). Construction and evaluation of models of environmental systems. In M. B. Beck, A. J. Jakeman & M. J. McAleer (Eds.), Modelling change in environmental systems. New York: John Wiley & Sons Ltd.

Burrough, P. A., & McDonnell, R. A. (1998). Principles of Geographical Information Systems. Oxford: Oxford University Press.

Crosetto, M., & Tarantola, S. (2001). Uncertainty and sensitivity analysis: tools for GIS-based model implementation. International Journal of Geographical Information Science*,* 15(5), 415-437.

Heuvelink, G. B. M. (1998). Error Propagation in Environmental Modelling with GIS. London: Taylor & Francis.

Karssenberg, D. (2002). The value of environmental modelling languages for building distributed hydrological models. Hydrological Processes*,* 16(14), 2751-2766.

Karssenberg, D., & De Jong, K. (2005). Dynamic environmental modelling in GIS: 1. Modelling in three spatial dimensions. International Journal of Geographical Information Science*,* 19(5), 559-579.

Karssenberg, D., & De Jong, K. (subm.). Modelling landscape dynamics with Python. International Journal of Geographical Information Science.

PCRaster. (2006). PCRaster internet site. Retrieved jan 01, 2006, from http://pcraster.geo.uu.nl

Pearl, J. (1981). A space-efficient on-line method of computing quantile estimates. Journal of Algorithms*,* 2, 164-177.

Toffoli, F. (1989). Cellular automata machines. Cambridge, Massachusetts: MIT Press.

Wainwright, J., & Mulligan, M. (2004). Environmental Modelling. Chichester: Wiley.