Applications of Analytical Cartography[*†]

Wm Randolph Franklin

Electrical, Computer, and Systems Engineering Dept.,
Rensselaer Polytechnic Institute, Troy, New York 12180–3590

and

The National Science Foundation

wrf@ecse.rpi.edu
http://www.ecse.rpi.edu/Homepages/wrf

April 20, 2000

Abstract

Several applications of analytical cartography are presented. They include terrain visibility (including visibility indices, viewsheds, and intervisibility), map overlay (including solving roundoff errors with C++ class libraries and computing polygon areas from incomplete information), mobility, and interpolation and approximation of curves and of terrain (including curves and surfaces in CAD/CAM, smoothing terrains with overdetermined systems of equations, and drainage patterns). General themes become apparent, such as simplicity, robustness, and the tradeoff between different data types. Finally several future applications are discussed, such as the lossy compression of correlated layers, and just good enough computation when high precision is not justified.

[*] *to appear*, Cartography and Geographic Information Systems, 2000
[†] http://www.ecse.rpi.edu/Homepages/wrf/research/gisapps/gisapps.pdf

Contents

List of Figures

1 Introduction

Analytical cartography's progress has been distinguished since *mathematikoi* measured the circumference of the Earth, (Aristotle, 350 BC), which may have happened before 2600 B.C., (Stecchini, 2000). Throughout its history, it has relied on, and even forced the development of applied mathematics. Today, techniques developed in computer science enable its advance.

This paper samples several applications in analytical cartography that have benefited from computer science, including terrain visibility, map overlay, mobility, and interpolation and approximation. Visibility includes using industrial engineering sampling techniques quickly to identify the most visible potential observers, and alternating visibility index determination with viewshed determination to site a set of observers who, jointly, will cover the complete area of interest. It also includes intervisibility, or the pairwise visibility of a group of observers. Map overlay leads to a discussion of roundoff errors during computation, and one solution, the use of C++ class libraries, such as LEDA and CGAL, that permit computations with datatypes such as rational numbers, which permit the exact intersection of two lines. Map overlay also includes determining the areas of polygons using only information about the set of vertex-edge incidences.

Interpolation and approximation includes a discussion of theory, mentioning some counterintuitive results. It describes how the CAD/CAM community handles curves and surfaces. It also discusses terrain elevation interpolation, including using the solution of an overdetermined system of equations, with Matlab, to cause greater smoothness and to infer local peaks inside topmost contours. It also summarizes drainage determination.

Some common themes become apparent, including the interplay between theory and application, and factors in data structure and algorithms design, such as the importance of simplicity, the importance of robustness, the use of Simulation of Simplicity to remove geometric degeneracies, and the choice of different representation formats.

Finally some representative unsolved problems are presented, such as just good enough computation, and lossily compressing correlated layers of data while maintaining important relations.

2 Applications

This section samples applications in analytical cartography that have benefitted, or might benefit from research in computer science. Many interesting applications are omitted; Müller (1991) contains a valuable collection of other problems.

2.1 Terrain visibility

Consider a terrain elevation database, and an observer, $\mathcal{O}$. The *viewshed* is the terrain visible from $\mathcal{O}$ within some distance R of $\mathcal{O}$. The observer might be situated at a certain height above ground level, and might also be looking for targets at a certain height. Figure 1 on the following page shows a viewshed with error bars for a region in northeastern New York State. The observer, at Mt Marcy, the highest point, is marked by a white square near the lower left. This figure was produced by a program that classifies the grid cells according to how likely they are to be visible.

The elevation of the almost certainly visible cells is indicated by colors. Cells that are probably visible are indicated by colors overlaid with widely spaced black lines. These would form the most likely viewshed. Cells that are probably hidden are colored according to their elevation, but overlaid

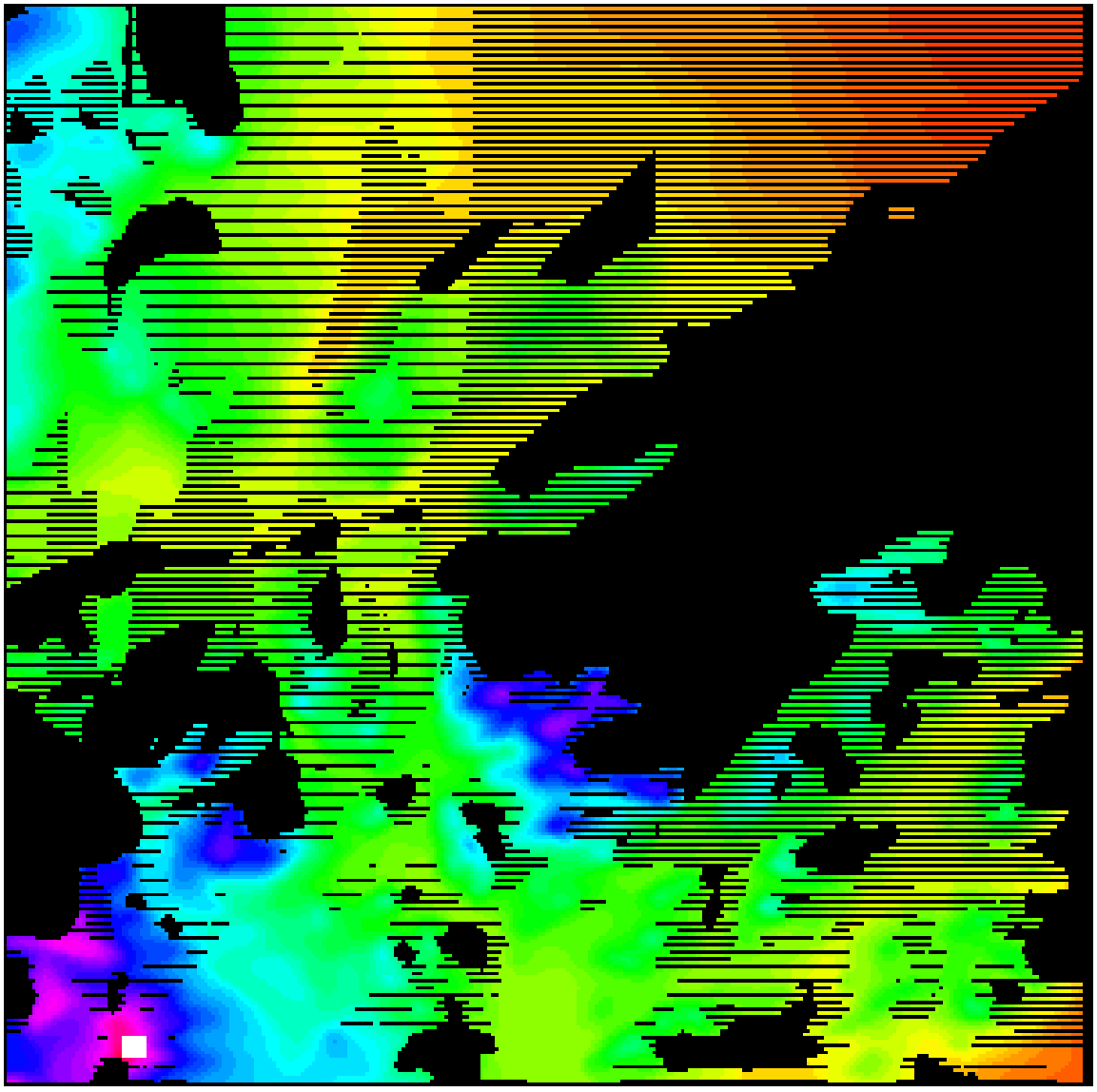

Figure 1: Viewshed with Uncertainty

with closely spaced black lines. Finally, cells that are almost certainly hidden from the observer are colored solid black.

The *visibility index* of $\mathcal{O}$ is the fraction of the disk of radius R centered on $\mathcal{O}$ that is visible from $\mathcal{O}$. Ray (1994) presents new algorithms and implementations of the visibility index.

Siting radio transmitters is one application of terrain visibility. Here, the identities of the observers of highest visibility index are of more interest than their exact visibility indices, or than the visibility indices of all observers. Locating points at which to hide is a corollary application. For example, we may wish site a forest clearcut to be invisible to observers driving on a highway sited to give a good view. Statistical sampling techniques from production quality control in industrial engineering are useful to identify the most visible observers, as follows.

1. Start with a potential observer $\mathcal{O}$. We wish to know whether $\mathcal{O}$ has, with a given probability, at least a given minimal visibility index.
2. Select N random target points in the disk of radius R around $\mathcal{O}$.
3. Determine how many are visible from $\mathcal{O}$ by running lines of sight. Let that number be K.
4. Note that the expected visibility index of $\mathcal{O}$ is $\mu = K/N$, and the standard deviation is $\sigma = \sqrt{\mu(1-\mu)/N}$.
5. If the expected visibility index is high enough with sufficient probability, then accept $\mathcal{O}$ as a useful observer. E.g., since in a normal distribution, $p(z > 1.6) = 0.05$, for a 95% probability that the visibility index is at least 0.5, we need $\mu - 1.6\sigma > 0.5$.
6. Otherwise, if the expected visibility index is low enough with sufficient probability, then reject $\mathcal{O}$, select another potential observer and repeat from step 2 above.
7. Otherwise, we don't yet know what to do with $\mathcal{O}$, so select more random targets, and repeat from step 3 above.

The process is fast enough to iterate through a complete 1201×1201 level-1 DEM.

Combining two separate algorithms so that each executes alternately is another useful technique. In this case viewshed determination for a specific observer, and selection of the most visible observers, can be combined to site a set of observers that, jointly, will cover the whole terrain. The process of selecting a set $\mathcal{S}$ of observers goes as follows.

1. Initially $\mathcal{S} = \{\}$.
2. Find a set, $\mathcal{P}$, of very visible potential observers, as described above.
3. Calculate all their visibility indices.
4. Pick the most visible one, insert it into $\mathcal{S}$, and determine its viewshed, $\mathcal{V}$.
5. Delete any observers in $\mathcal{P}$ that are also in $\mathcal{V}$.
6. Pick the most visible remaining observer.

7. Add it to the set of observers, and find the union of its viewshed with the other observers in $\mathcal{S}$.

8. Repeat until this union viewshed is the entire terrain.

The set of observers $\mathcal{S}$ might be fine-tuned with a technique reminiscent of a multiple stepwise linear regression with independent variable addition and deletion, as follows.

1. Find the observer in $\mathcal{S}$ whose viewshed covers the smallest area not covered by other observers. With luck this observer might cover no new area at all (since later observers' viewsheds covered its viewshed completely).

2. Delete it.

3. If there is now some uncovered area, then insert a new observer as described earlier.

4. Repeat.

Efficiently calculating areas of sets of intersecting and uniting polygons is required to make this process feasible. That is covered in the next section on overlay.

2.2 Map Overlay

Consider two layers in a vector database, such as soil types and watersheds. We wish to *overlay* them to create new polygons, each of which contains only one soil type and watershed. Roundoff errors and short-circuit area calculations are two computer science issues resulting from this overlay. Wu (1987) performs overlays in the Prolog language.

2.2.1 Roundoff Errors, and $C++$ Class Libraries

The two layers frequently contain edges almost in common, as when an edge common to the two layers, was digitized separately for each layer. The result is "sliver" polygons and floating roundoff errors, (Veregin, 1989; White, 1977), which worsen as the digitization accuracy improves. The proper solution is to recognize that this is one edge. However there are also tools to write computer programs that are more tolerant of roundoff errors.

LEDA, the *Library of Efficient Datatypes and Algorithms*, (LEDA, 2000; Mehlhorn and Näher, 1995), is a C++ class library that "provides a sizable collection of data types and algorithms in a form which allows them to be used by non-experts. This collection includes most of the data types and algorithms described in the text books of the area", according to the manual. Its number types include, among others, arbitrary-length integers, rational numbers defined as the quotient of two integers, big floating point numbers, and algebraic numbers defined as roots of polynomials. Rational numbers allow the intersection of two lines to be stored exactly. A map overlay program using rationals will be somewhat slower, but will have no roundoff errors. Algebraic numbers allow the intersection of a line and a circle to be stored exactly. A coordinate of $\sqrt{2}$ is stored as the expression $\sqrt{2}$ and not as $1.414\ldots$. When squared, the result is exactly 2.

Built-in data types include 1-D and 2-D arrays, stacks, queues, trees, hash tables, dictionaries, and graphs. Implemented algorithms include shortest path and maximum flow on graphs, and

Voronoi and 3-D convex hull algorithms in geometry. A user can change a variable's types after coding an algorithm, by changing a few declarations and recompiling. Finally, there are tools to create graphical user interfaces, many demonstration programs, and considerable documentation.

CGAL, the *Computational Geometry Algorithms Library*, is another C++ class library, which implements more specifically geometric operations such as 2D and 3D Voronoi diagrams, multidimensional searching, etc., (CGAL, 2000; Fabri et al., 1996).

These, and other similar, packages allow the designer to delay critical decisions about what number type to use, to experiment more easily, and hence greatly to increase productivity.

2.2.2 Polygon Area Determination From Partial Information

Determining the overlaid polygon's areas, perhaps to interpolate data from one layer to another, is one application of overlaying, (Franklin et al., 1994). Tobler (1999) smooths the data with partial differential equations to achieve a similar effect. In this case, suppose that layer $\mathcal{A}$ has census polygons, labeled $\mathcal{A}_i$, including their areas, a_i, and populations, p_i. Layer $\mathcal{B}$ has soil types. We wish to estimate the population of each soil polygon, $\mathcal{B}_j$, perhaps to estimate agricultural land being taken out of production for housing. Given c_{ij}, the area of intersection of $\mathcal{A}_i$ and $\mathcal{B}_j$, then one estimate of the population of $\mathcal{B}_j$ is

$$\sum_i \frac{p_i c_{ij}}{a_i}$$

(This assumes that the population of $\mathcal{A}_i$ is evenly distributed throughout it.) Calculating c_{ij} would seem to require having the actual intersection of $\mathcal{A}_i$ and $\mathcal{B}_j$. However, with a careful choice of data structure and algorithm, this is not true since most of the topological structure of the polygon is unnecessary to determining its area. In the present case, the necessary information includes only the set of incidence relations of vertices and edges, and, for each one, this information:

1. the location of the vertex,
2. the direction that the edge leaves the vertex, and
3. which side of this edge ray is inside the polygon.

Knowing how the vertices pair up to form edges, and how the edges are joined into nested loops is unnecessary. This greatly shortens the program and speeds the computation. The implementation executes in expected time linear in the number of input edge segments plus output polygons, (Franklin et al., 1994). The concept also extends to finding areas of boolean combinations of many polygons, (Narayanaswami and Franklin, 1992).

2.3 Mobility

Consider a database of terrain elevation and ground conditions, such as soil type and tree stem size. Now, determine the best feasible route for a hiker, truck, tank, or whatever from point $\mathcal{A}$ to $\mathcal{B}$. Avoid the temptation to make the problem unrealistic while simplifying it enough to solve it. Battles have been won, such as in the Ardennes forest in 1940, when one side traversed an area that the other side thought was impassable.

The simplest mathematical formulation of this is as a shortest path graph traversal problem, perhaps using a Dijkstra algorithm, although if the roads have capacity limits the situation is more complicated. Then there are transportation capacity paradoxes such as Braess (1968), where adding a new road increases every driver's delay. Mobility in 3-D is of interest to the robotics community, (Akman, 1985).

The recent *intervisibility* problem concerns planning the joint mobility of a group. Assume that a platoon of soldiers wishes to move from $\mathcal{A}$ to $\mathcal{B}$, having regard to the considerations described above, plus two new constraints. First, the soldiers wish to remain pairwise visible to each other in order to communicate via VHF radio. Second, the soldiers wish to remain dispersed so that there is never a possible single other observer who can simultaneously see them all.

2.4 Interpolation and Approximation

Drawing a meaningful line or surface through (for interpolation), or near (for approximation) a sparse set of points remains an engaging problem. In this section, we will first see some interpolation theory, then some practice from the CAD/CAM community, and finally some issues in terrain reconstruction.

2.4.1 Interpolation theory

The mathematics of interpolation is counterintuitive, and the desired properties often mutually contradictory. Consider the Lagrangian interpolation of an $N-1$-degree polynomial to fit N points in 2D. Even though the generated curve passes through the data points, between them it will traverse far from the line joining adjacent pairs of points. That is, the curve lacks the useful *variation minimization* property.

Further, moving one point drastically changes the whole curve, so Lagrangian interpolation does not have *local control*. Figure 2 on the next page shows interpolating a 11-th degree polynomial through 12 evenly spaced points, half with $y = 0$, and half with $y = 1$. Note that the curve's maximum is $y = 6$, much larger than the data points' range of $-1 \leq y \leq 1$. Figure 3 on the following page shows the effect of moving the point $(6, 0)$ to $(6, 1)$. The whole curve largely inverts.

As another example, see the Catmull-Rom-Overhauser interpolating spline shown in Figure 7 on page 14. This cubic spline is C^2. In order to maintain that condition while interpolating control point $\mathcal{P}_2$, the curve is forced to swing wide before the point. Also, although the control points $\mathcal{P}_9$, $\mathcal{P}_{10}$, and $\mathcal{P}_{11}$ are collinear, the continuity requirements at $\mathcal{P}_9$ and $\mathcal{P}_{11}$ force the interpolating curve not to be a straight line.

More complicated interpolation methods are counterintuitive in more subtle ways. In fact, a completely "reasonable" interpolation method does not exist; the concept can be shown to be self-contradictory.

2.4.2 Curves and Surfaces in CAD/CAM

The *Computer Aided Design / Computer Aided Manufacturing* (CAD/CAM) community has extensive experience in approximating lines and surfaces with curves. Typically they use sections of parametric cubics whose tangents and curvatures match at the joints. Besides airplane fuselages, ship hulls, and auto bodies, even PostScript and TrueType fonts are designed thus. These techniques

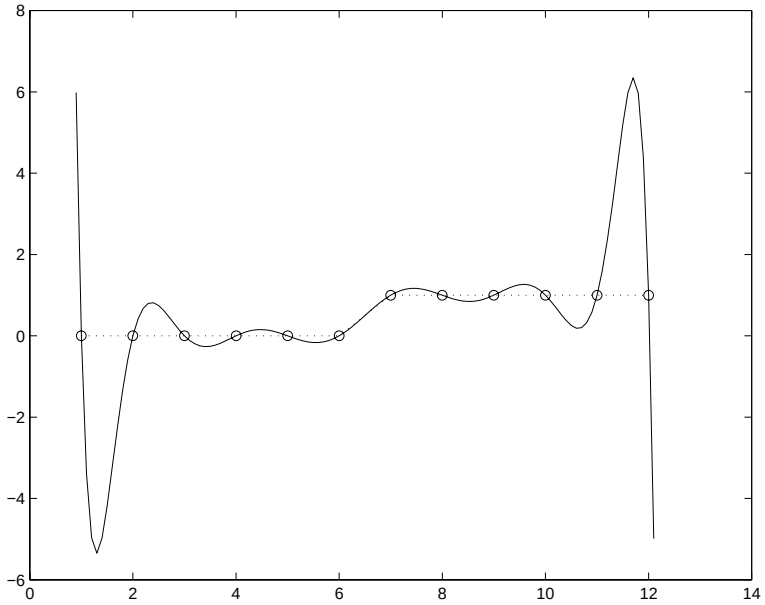

Figure 2: Interpolating 12 Points

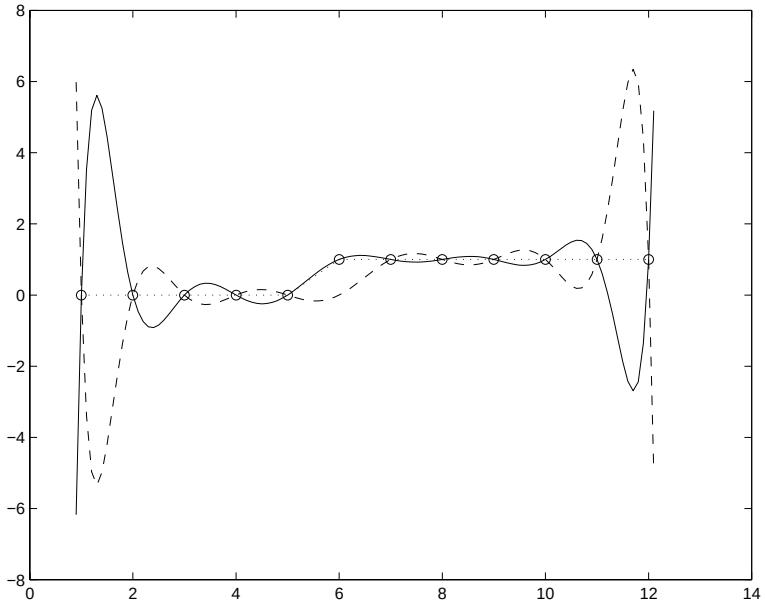

Figure 3: Moving One Point

will be useful in analytical cartography to the extent that curves and surfaces in this domain are also smooth. Fortunately, the available data have often been smoothed during data collection and sampling. Farin (1993) is an excellent text on these curves and surfaces. Franklin (1992) describes their applications to analytical cartography in more detail.

The first design question is, what form of mathematical equation should be used to represent a curve, such as a coastline or contour line? Ordinary differential equations (ODEs), explicit, implicit, and parametric curves are 4 possibilities. The ODE formulation which is $y' = -y/x$ for a circle of radius 1 centered at the origin, is often used for time-space curves, but is rare in curve design. In an explicit equation, y is a function of x, as in the half-circle $y = \sqrt{1 - x^2}$. In an implicit equation, we have a function of x and y that is zero for points on the curve. For example, a circle is $x^2 + y^2 = 0$. A parametric equation has a parameter, t, and both x and y are functions of t. For example, the circle can be represented parametrically thus:

$$x = \frac{2t}{t^2 + 1}, \qquad y = \frac{t^2 - 1}{t^2 + 1}, \qquad -\infty \leq t \leq \infty$$

Explicit representations are not used because they can represent only a single-valued function, and that even ceases to be single-valued when rotated. In an implicit representation, determining points on the curve is difficult. Given one point, it is possible to step along the curve and generate successive points on the same curve segment. However, it is difficult to determine the curve's topology; how many curve segments there are. For example, consider these four curves; note that the only difference is the constant at the end.

$$y^2 = x^3 - 3x - 3$$

$$y^2 = x^3 - 3x - 2$$

$$y^2 = x^3 - 3x$$

$$y^2 = x^3 - 3x + 2$$

Their plots are in Figure 4 on the following page. Note that the first curve has one infinite line, but the second has one infinite line plus a single isolated point at $(-1, 0)$, the third has an infinite line and a closed line, and the fourth has an infinite line that crosses itself.

A more important problem with an implicit representation is that we usually don't want an infinite curve, but rather a finite segment, which requires also storing the segment's limits, perhaps as two other intersecting implicit equations.

Therefore, the parametric representation is preferred. The next decision is whether to represent a complicated shape with one high-degree curve, or instead a sequence of low-degree curves, designed to join with sufficient continuity that the joints are invisible. A single high degree curve is undesirable for several reasons, which we'll illustrate with an explicit representation for simplicity. As shown above in the interpolation section, when an $N - 1$ degree polynomial is interpolated through N points, between the points, the polynomial may take very large values. Indeed, the size

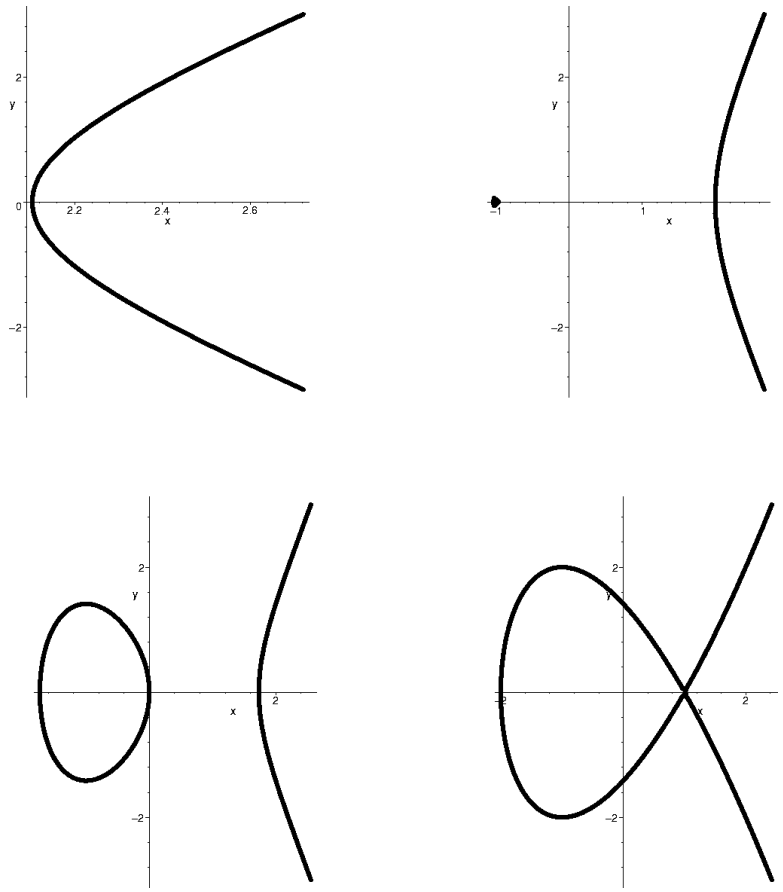

Figure 4: Four Topologically Different Implicit Plots, Whose Equations Differ Only in a Constant

of these excursions can grow exponentially with N. Next, the generated polynomial can be very sensitive to small changes in the input points. Indeed, this problem can be expressed as the solution of a system of linear equations, where the determinant is almost singular. Finally, the generated polynomial is very sensitive to numerical roundoff errors during the computation.

Therefore, a *spline*, or a curve composed of a sequence of low degree curves, is the preferred solution. If the joint between two adjacent segments is to be invisible, then the radius of curvature should be continuous across the joint. That is called G^2 continuity. To simplify the mathematics at the cost of using up a degree of freedom at each joint, a slightly stronger condition, that the second derivative be continuous across the joint, is often used instead. This is called C^2 continuity.

If the spline segments are to join with C^2 continuity, and the segments are not really all the same quadratic curve, then each segment must be at least a cubic polynomial. Its equation is as follows.

$$x = \sum_{i=0}^{3} a_i t^i, \qquad y = \sum_{i=0}^{3} b_i t^i, \qquad 0 \le t \le 1$$

By varying the parameter t, we can easily generate points on the segment. The curve is defined by 8 coefficients, or degrees of freedom. Requiring the user to specify a_i and b_i explicitly would be hostile, so better interfaces are used. In the *Hermite* form, the user specifies the 8 degrees of freedom as the two endpoints, and the vector derivatives at the endpoints. In the *Bézier* form, the user specifies a 4-point *control polygon*. The generated curve starts at the first point, goes near the second and third points, and ends at the fourth point. If the control points are $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, then the curve is

$$\mathcal{P}(t) = \sum_{i=0}^{3} w_i \mathcal{P}_i$$

where

$$w_i = \binom{3}{i} t^i (1-t)^{3-i}$$

$\binom{3}{i}$, pronounced 3-choose-i, is $\frac{3!}{i!(3-i)!}$. Since $\sum_i w_i = 1$ the curve is always inside the convex hull of the four points. Therefore, if the control polygon is relatively flat, then so will be the curve, which is desirable. Figure 5 on the next page shows 4 control polygons and their corresponding Bézier curves.

A cubic *B-spline* curve contains a sequence of cubic Bézier segments joined with C^2 continuity. That severely limits the number of degrees of freedom of each segment. For instance, the last control point of one segment must be identical to the first control point of the next. Each segment will be affected by only four control points, and each control point will affect only four segments. Segment Q_i will be affected by points $P_{i-3}, \ldots, P_i$, and control point P_i will affect segments $Q_i, \ldots, Q_{i+3}$. This desirable concept is called *local control*.

The B-spline does not go through any control points, even the end ones, unless some control points are superimposed or duplicated. A double control point reduces the continuity and the corresponding knot from C^2 to C^1. A triple control point reduces the continuity to C^0, i.e., the curve has a corner here, and it goes through the control point.

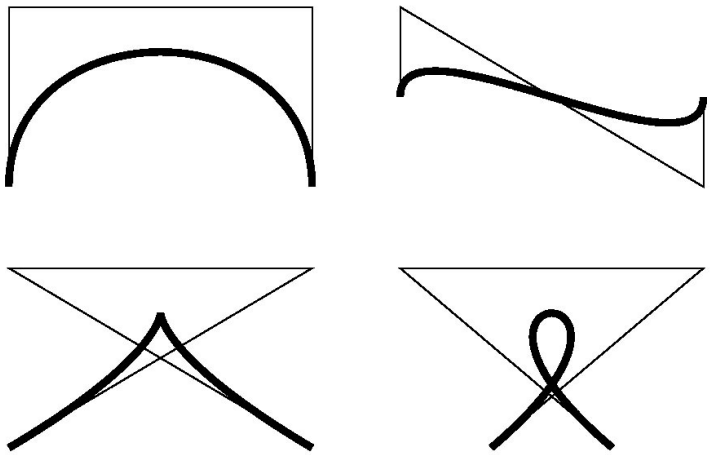

Figure 5: Some Bézier Control Polygons and Corresponding Curves

Figure 6 on the following page shows an example of a B-spline on ten control points, together with the control polygon ($P_0P_1\ldots$) and the joints between the segments ($K_3K_4\ldots$).

B-splines' not going through their control points is acceptable to a freeform designer, but less desirable when we wish to fit a spline to some data. The Catmull-Rom-Overhauser splines go through, or interpolate, their control points. In addition, the tangent to the spline at control point P_i is in the direction $P_{i-1}P_{i+1}$. Thus each segment is defined by two points and two derivatives, which give the correct number of degrees of freedom. Figure 7 on the next page shows an example on the same control polygon as in Figure 6 on the following page.

The easiest way to extend these ideas to a surface is to construct it as a grid of *patches*, each patch a parametric Bézier surface with two parameters u and v, and the following equation.

$$x = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}u^iv^j, \qquad y = \sum_{i=0}^{3}\sum_{j=0}^{3} b_{ij}u^iv^j,$$

$$z = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{ij}u^iv^j, \qquad 0 \le u \le 1,\ 0 \le v \le 1$$

There are 48 degrees of freedom: a_{ij}, b_{ij}, c_{ij}. A more user-friendly interface is to specify a *control grid* of 4×4 3-D points $\mathcal{P}_{ij}$. Then a general point on the surface is

$$\mathcal{P}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \binom{3}{i}\binom{3}{j} u^i(1-u)^{3-i}v^j(1-v)^{3-j}\mathcal{P}_{ij}$$

$$0 \le u \le 1,\ 0 \le v \le 1$$

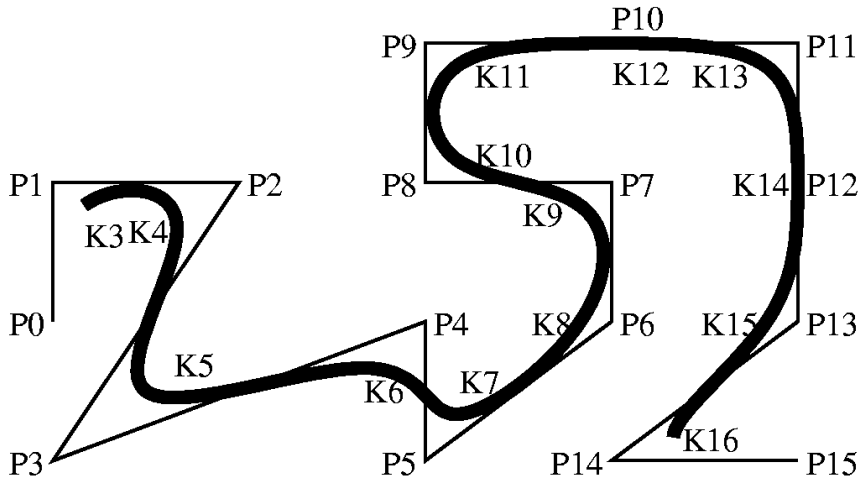

Figure 6: B-spline Approximating Ten Control Points

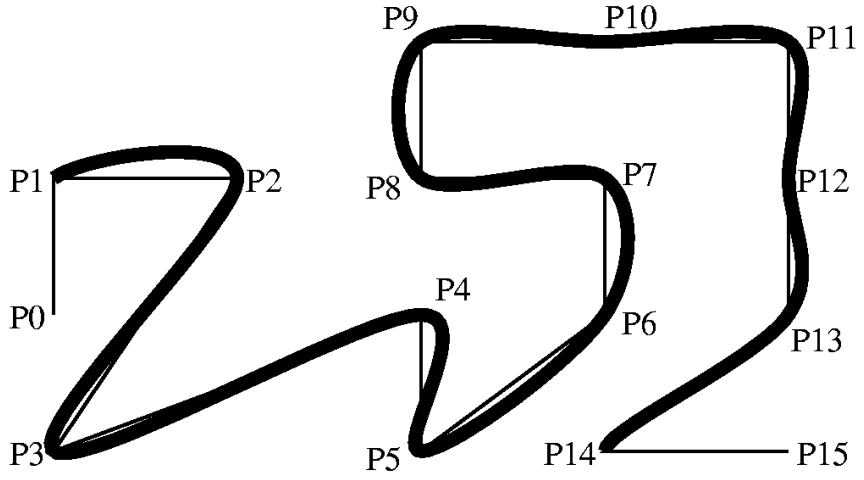

Figure 7: Catmull-Rom-Overhauser Spline Interpolating Ten Control Points

Adjacent patches are joined with C^2 continuity, so that the joints are invisible.

Triangular spline patches are more powerful than the, above, much more common, Cartesian product square patches, and more easily adapt to regions of varying resolution. However, the mathematics is more complicated and fewer software packages are available yet. Pedrini (2000) is investigating approximating a terrain surface with a TIN composed of triangular splines.

2.4.3 Terrain Elevation Interpolation

Assume that we have a digital terrain elevation model (DEM) dataset represented by a matrix of $N \times N$ elevations, many of which may be missing. For a level-1 DEM, $N = 1201$. Lam (1983) surveys many methods. Common operations include interpolating missing elevations, smoothing the data, and determining drainage networks. Voronoi diagrams, (Gold, 1990) are particularly capable of processing very unevenly spaced data. Wood and Fisher (1993) assesses interpolation accuracy. Douglas (1983) interpolates contours by running 8 rays from each unknown point until they hit known points, then postprocessing the surface to smooth it. Gousie (1998) presents several new methods. The first interpolates intermediate contours between the known ones. Another, based on *lofting* in CAD/CAM, interpolates gradient lines in x and y, then smooths the elevations along each of them by interpolating splines in z.

This section considers one conceptually easy method for raster data, which is to set up a system of $M = N^2$ linear equations to solve for either the unknown elevations or the flow through each cell. To interpolate a missing point, let z_i be the unknown elevation, and z_l, z_r, z_u, and z_d be elevations of its four neighbors, some or all of which may also be unknown. This is reminiscent of the use of matrices for surface smoothing in (Tobler, 1966). Here, we have the equation

$$z_l + z_r + z_u + z_d - 4z_i = 0 \tag{1}$$

This is equivalent to saying that the surface satisfies the Laplacian partial differential equation,

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0$$

Since z_i depends directly on only 4 other points, its line in the matrix of all the coefficients of the system of linear equations of the elevations will have 5 nonzeros, and $N^2 - 5$ zeros. When we solve this system of linear equations for the unknown heights, it is necessary to utilize the fact that, of the N^4 entries, only $5N^2$ are nonzero. *Matlab*, a commercial interactive system designed to process matrices, is one very useful tool for processing such *sparse* (mostly zero) matrices, efficiently. Matlab solves such problems for $N = 500$ or more. For larger N, *multigrid* techniques can handle N over $10,000$.

The above method will have an equation for each unknown point, to make it the average of its neighbors, which makes the generated surface smooth there. However, since the known points are known, they are not necessarily the average of their neighbors, and so the surface is probably not smooth there. This is especially important when there are many adjacent known points, e.g., points defining contour lines. The surface's slope will not be continuous across the contours. Also, in the usual case where the lower contours are longer than the higher ones, the generated surface will sag between the contours, as if pulled down by gravity. Therefore, it would be desirable also to make

the known points to be the average of their neighbors, but that would lead to more equations than unknowns. If there are M points, of which K have known elevations, we can set up a system of $M + K$ equations with M unknowns as follows.

1. Pretend that all the M points have unknown elevations z_i.
2. Create an equation for each z_i, setting it to be the average of its neighbors as in equation 1 above.
3. For each of the K points whose elevation e_i we do know, create an additional equation

$$z_i = e_i \tag{2}$$

The resulting system of equations is $Az = b$ where A is a $(M + K) \times M$ matrix. z is a $M \times 1$ vector, and b a $(M + K) \times 1$ vector. Although the system is overdetermined, a solution that minimizes the sum of the RMS errors in the equations, is possible. That is, find z to minimize $(Az - b)^t(Az - b)$, where v^t is the transpose of the matrix or vector v. The solution can be expressed as $z = (A^tA)^{-1}A^tb$ although computationally better methods are used. Matlab can solve these systems. In this least squares fit, the equation $z_i = e_i$ no longer has the same effect as $100z_i = 100e_i$. The latter form will have 100 times the weight in the solution, so that z_i will be much closer to e_i. Hence, we can choose the relative importance of accuracy versus smoothness, point-by-point.

Figure 8 on the next page shows this effect. Subfigure (a) shows the four nested square contour lines to be approximated. The contours were made square to make the problem harder. As the surface tries to remain continuous at the corners, it will become inaccurate. Subfigure (b) shows the Lagrangian interpolation of the points between the contours. Note how the surface normals are not continuous across the contours, and that the contours are quite visible, which is undesirable. Subfigure (c) shows an overdetermined solution, with the smoothing equations, (1 on the preceding page), made equally as important as the known-elevation equations (2). This surface is quite accurate. However, since we can see the contours, especially along the silhouette, the surface is not smooth enough. Some true surfaces may have slope discontinuities, but probably not exactly at contour lines. On the other hand, this surface is higher inside the innermost contour, which is very desirable, and which most interpolation schemes do not do. This inferred peak was caused by forcing the continuity of slope across that contour.

Subfigure (d) shows an overdetermined solution with the equations (1) have 10 times the weight of the equations (2). There is no evidence of the original contour lines. but the surface interpolates the contours less accurately. The following table shows the mean absolute error and maximum error for various weights ρ. In all the surfaces, the points of maximum error occur at the corner of the contour lines. The rareness of the very inaccurate points is shown by the large ratio of maximum error to mean absolute error.

ρ	mean abs error	max error
0.01	0.01	0.19
1	0.37	3.8
3	1.4	8.4
10	4.4	13.3

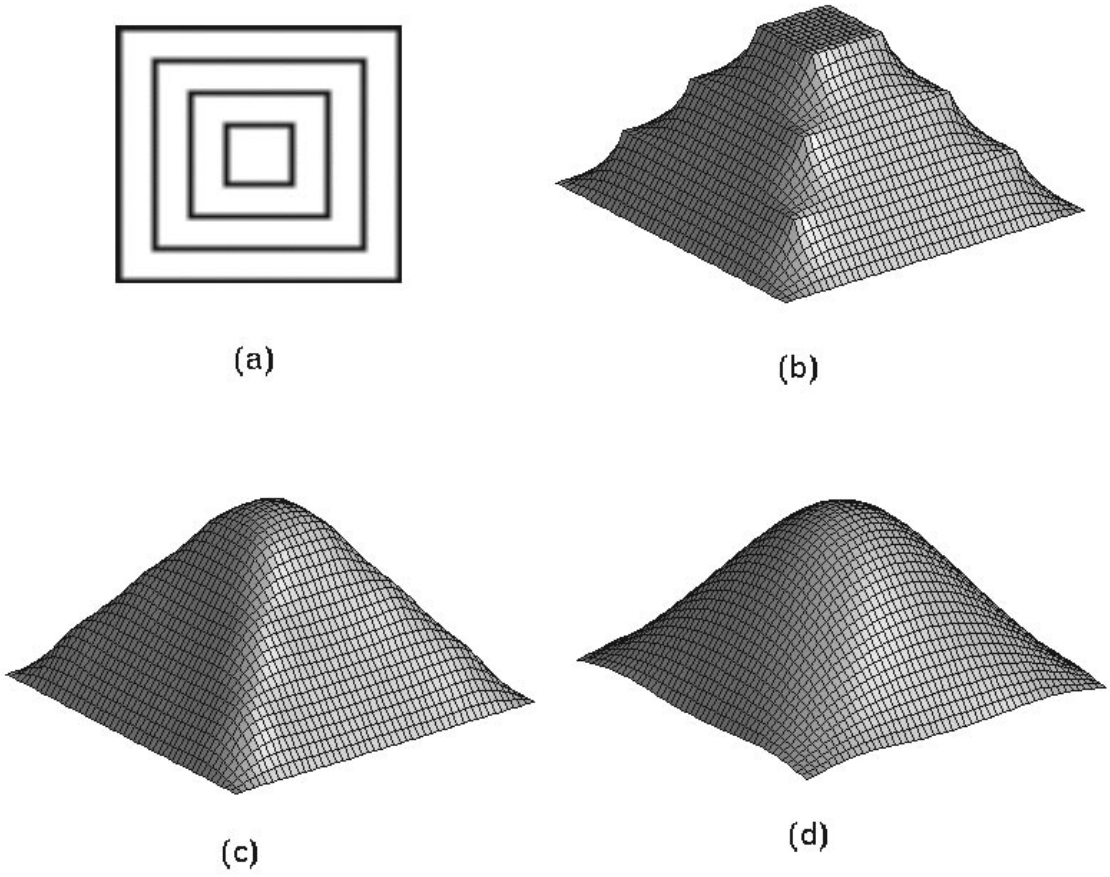

Figure 8: Overdetermined Interpolation. (a): The Square Contours to be Interpolated. (b): Lagrangian Interpolation. (c): Overdetermined Solution, R=1. (d): Overdetermined Solution, R=10.

This surface generation method appears to be very promising. Although the mathematics are quite simple, with only one parameter (the relative weight of the two equation types), the generated surfaces have peaks inside topmost contours, and show little or no evidence of the contours. The main limitation, as this is written, is that the builtin Matlab routines cannot find overdetermined solutions on grids of more than about 200×200 points. However, that limit is expected to be exceeded shortly.

2.4.4 Drainage

Consider a terrain elevation database, and assume some precipitation pattern. The problem is to determine the *drainage patterns*, where the streams and rivers will flow, (Mark, 1984; McCormack et al., 1993). According to (López, 1997), which has a detailed problem description with many references, drainage patterns have been constructed in hydrology at least as far back as by R Rothe in 1915.

On a grid, we may formalize the problem as follows. Let r_i be the rain falling on cell i. The variable i takes values from 1 to N^2 for an $N \times N$ grid of cells. Let $a_{ij} = 1$ if the water in cell i flows to cell j, otherwise 0. Let f_i be the unknown flow out of cell i, which is composed of the flow into the cell plus the rain. Then

$$f_i = r_i + \sum_j f_j a_{ij}$$

This can be rewritten thus:

$$r = f(I - A) \tag{3}$$

where f and r are vectors of N^2 elements, I is the $N^2 \times N^2$ identity matrix, and A is the $N^2 \times N^2$ matrix of the a_{ij}. This is a sparse system of linear equations. A Matlab function implementing this complete process, from input elevations to output flows, takes 50 lines of code. Processing a 301×301 grid takes ten CPU minutes on a 233-MHz Pentium. The advantage of Matlab compared to the alternative of explicitly calculating the water flow with a C program is simplicity, but it's a little too simple.

It assumes that each cell's water should flow to its lowest neighbor, provided that that neighbor is lower than the current cell. (Removing that restriction causes loops in the water flow, which leads to an inconsistent set of equations.) However, the large number of local minima trap so much water that major river systems are prevented from forming. Various solutions are possible. US Geological Survey (1996) fills in these local depressions, after checking that they are not an actual sink, such as Lake Chad.

One efficient solution is a *connected component* algorithm from computer science. Let us define two grid cells to be connected if they are adjacent and water can flow from either one to the other, or if they are connected to cells that are connected. Therefore, each drainage basin forms one connected component. Connected components can be quickly determined from the adjacency relationships even on grids larger than 15000×15000 cells. Finally, some rule must still be applied to process the water accumulating in each component, by modifying A and re-solving equation (3).

3 Themes

Several themes may be apparent from the previous applications.

1. *Interplay between theoreticians and applications:* Cartographers need computer science theoreticians to find algorithms to solve problems such as efficiently calculating Voronoi diagrams. The theoreticians need the cartographers to tell them what's worth solving, and to stop them from idealizing problems until they are unrealistic.

   This loop is imperfectly closed. For a model of how things might operate better, consider physics and astronomy. The theoretical physicists predict some phenomenon, such as the gravitational red-shift, which the experimentalists then look for. In addition, the experimentalists observe anomalous phenomena, such as, in the 19th century, the precession of Mercury's orbit, which the theoreticians then try to explain. Not every anomaly leads to interesting theory; there were several other 19th century orbital oddities, which had classical explanations and therefore are now forgotten. Nevertheless, every so often general relativity results.

2. *Algorithm and data structure design principles:* Various rules for creating software in analytical cartography are worth considering, especially since they are often ignored.

   (a) *Efficiency:* This includes both asymptotic rate-of-growth time, and actual time on the largest possible datasets, in contrast to on the toy-sized examples seen too often. Worst-case efficiency, where an adversary selects the worst possible input, is easier to analyze. However, expected efficiency, under some probability distribution, choosing which is itself a hard question, is more useful.

   (b) *Simplicity:* There is a temptation to complexify ideas, adding bells and whistles, since bigger is obviously better. Wrong! "You need a lot of self-confidence to do simple things." Einstein said that an idea should be as simple as possible, but no simpler. If a data structure can hardly be explained to another person, then it will be difficult to implement, correctly. There are two ways to design something: It may be so simple that there are obviously no errors. Or, it may be so complex that there are no obvious errors.

   Optimal algorithms in Computer Science are often quite simple. When inserting a key into a hash table, *location = modulo(key, tablesize)* is generally fine. An excellent virtual memory page-replacement strategy is to page out the least recently used (LRU) page. On the World Wide Web, the Hypertext Transport Protocol (HTTP) has largely replaced the more complicated File Transport Protocol (FTP) and Gopher. Hypertext Markup Language (HTML) has prevailed over PostScript and LaTeX, even though they can display text and equations in ways that HTML cannot.

   Simple algorithms have the advantage that we don't spend all our time getting the program syntactically legal, let alone executing correctly. After getting the algorithm implemented correctly, there is time to play with the program to understand its behavior and improve it. Simple algorithms also often parallelize more easily.

   Simple data structures are also often smaller, so more data elements will fit into the processor's cache memory. Cache is much faster, and much smaller, than main memory, but this important design consideration is often ignored, (Bentley, 2000). Finally, simplicity leads to greater parallelizability, (Kankanhalli, 1990; Narayanaswami, 1991).

Complexity is tolerable in the algorithms' analysis and design, if that makes the implementation simple, as *Simulation of Simplicity* (SoS) demonstrates, (Edelsbrunner and Mücke, 1990). Simulation of Simplicity handles degenerate geometric cases in algorithms. A *degeneracy* is a geometric coincidence, such as a point incident on a line. If that point is (x_0, y_0), and the line is defined by the two points (x_1, y_1) and (x_2, y_2), then there is a degeneracy when the determinant

$$\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0$$

Consider a point-in-polygon algorithm designed for general points, where a ray is extended from the test point, p, up to infinity, the number of polygon edge crossings is counted, and p is inside if and only if the number is odd. There is a degeneracy when the ray crosses a polygon vertex. SoS transforms this algorithm into an algorithm that is still correct. Its method is to pretend to perturb the coordinates by an infinitesimal amount, so that the ray doesn't quite touch the vertex, and the degeneracy vanishes. Each perturbation uses a different order of infinitesimal, so that new degeneracies are not created.

An *infinitesimal*, ϵ, in this context, is a positive number that is smaller than any positive number we can use for a coordinate. The difference between any two coordinates, if positive, will be larger than ϵ. A second-order infinitesimal, ϵ^2 is smaller than any first-order infinitesimal. If we add ϵ to the test point's x-coordinate, then the coordinate cannot now be equal to the x-coordinate of any vertex. Therefore, the only practical effect is on conditional tests for equality. SoS analyzes the infinitesimal's effect those changes on these boolean tests, and changes the code accordingly. In the point-in-polygon case, the only change is that one '$<$' test becomes a '$\leq$'. In this case, though not always, the code is no larger or slower; all the complexity lay in the code derivation. One unexpected property of such code, since its derivation is unobvious to other programmers, is that they may easily corrupt it when translating to another language. This provides a crude form of intellectual property protection.

(c) *Robustness:* Data are imprecise, and sometimes completely wrong. Floating point numbers have roundoff errors. Real data may correlate in ways worse than if it were random. For example, consider a sweep-line algorithm testing a street network for intersections. An algorithm designed to handle the number of active streets that would occur if streets were random might fail on a regular grid, such as in Chicago. An fragile implementation may process small, test, cases, while failing on large, realistic, examples, perhaps because a neglected roundoff error deep inside the code caused a topological inconsistency that, much later, was fatal.

(d) *Things we know that are not so:* According to Will Rogers, these are worse than the things that we know that we are ignorant of. Example: It's a truism that compressing a large database in one piece is much more efficient than partitioning it into several small pieces and compressing them separately. The advantage of the latter would be that we could recover a small piece of the data without the time and space needed to uncompress the whole database.

However that isn't generally true. E.g., the *gzip* program compressed the 28058-byte source file for one version of this paper into a 11515-byte file. When it was split into seven pieces that were compressed separately, the total of their sizes was 11500 bytes, which is actually slightly smaller! The image processing community is aware of this; the JPEG image compression partitions the image into 8×8 pixel blocks, which it compresses separately.

Higher points in a terrain not necessarily being more visible than lower points is another example of unobviousness, (De Floriani et al., 1986; Franklin and Ray, 1994).

(e) *Interplay between different formats:* It's not clear, certainly not in advance, what the appropriate data structure may be. Although this author followed Peucker's and Douglas's ideas to implement the first TIN in cartography, in 1973 (Franklin, 1973; Peucker and Chrisman, 1975), has worked extensively with, the competing, elevation matrices and grids, and has looked at contour interpolation, he is still not certain which of the best method to represent terrain elevations. Then there is the issue of formal standards for data transfer, (Moellering, 1991). The Spatial Data Transfer Standard, (US Dept of the Interior and US Geological Survey, 2000), designed for portably transferring spatial data, allows for several data types, such as topological vectors, 2-D raster data sets, transportation nets, and points. Guerrieri (1989) presents one general methodology for portable software.

4 Future Applications

The research needs of analytical cartography remain considerable; this section describes only a sampling.

1. The question of the proper representation of terrain elevation data is still open. De Floriani (1987) uses Triangulated Irregular Networks (TINs). Goodchild (1980) shows how complex geomorphological terrain, including scale-variance and the long-range correlations of river basins can't be modeled directly by fractals, without extensions. Arrays of elevations, such as the USGS Digital Elevation Model (DEM). Gittings (1994) has a large catalog of digital elevation data. This format appears much more compact and easier to work with. However, the resolution is inherent in the data structure. Either technique may be made hierarchical; (De Floriani et al., 1984) shows hierarchical TINs. Compact representations for the TIN topology are available, (Speckmann and Snoeyink, 1997), which answer the objection that, in the TIN, the topology can require ten times the storage that the geometry does.

2. Is a conceptually deeper representation, based on the geomorphological forces that created the terrain, possible. Might we devise a basis set of operators, such as uplift, and downcut? Then we might deduce the operators that created the particular terrain under consideration, and store them. This approach is quite successful in mathematics, where various different sets of basis functions, each with particular strengths, are used. They include *sin* and *cos* functions used in Fourier transforms, square wave functions used in Walsh transforms, and *arccos* functions used in Chebyshev approximations. While all the above basis sets can also be used for terrains, the hope is that exploiting the richer structure of geomorphology will lead to more economical representations.

3. Slightly differently, can we represent the terrain by the features that people would use to describe it? Again, this is not a novel idea, but it may be solvable. The problem is that we can't just say that there is a hill over there, it is necessary to specify the hill in considerable detail. That might take more space than simply listing all the elevations with a grid or TIN.

4. The major creative force for terrestrial terrain is water erosion. This does not apply to the moon, to Jupiter's moons, or to Venus, data for which is becoming increasingly available, (Zimbelman and Tanaka, 1999). Since those surfaces' terrains should be statistically different, what impact will this have on the speed and output distribution of our algorithms and data structures?

5. We often have a database containing separate layers of information, such as both coastline and elevation, or both elevation and slope, or both contours and rivers. These layers are correlated with each other. E.g., rivers should cross contours perpendicularly.

   Note that we might want to store a layer explicitly even when it can apparently be derived from another layer. For example, the slope is the magnitude of elevation's derivative. However taking the derivative increases any errors. Since some questions, such determining whether a helicopter can land and also take off, are sensitive to the slope, we may want to store it explicitly.

   One difficult but important problem is how to lossily compress this database, while maintaining internal consistency. If the layers are compressed separately, they will be inconsistent when restored. This can be seen in some commercial PC-mapping products, where blue pixels may occur on the land side of a coastline, and rivers intersect contours obliquely.

   This idea can be generalized to include other desirable data restrictions. For instance, when reconstructing a surface, not creating or destroying gulleys, which may affect mobility, and in which people may hide, may be more important than minimizing the RMS error.

6. Can analytical cartography help us to develop three-dimensional data structures in geology? There are serious technical difficulties. Geometry in 2D differs from 3D in several respects, such as the following. A 2D Voronoi diagram on N points has a linear number of edges, while a 3D Voronoi diagram on N points may easily have $N^2/4$ faces. (This occurs if $N/2$ points are on the X-axis, and the other $N/2$ points are in a circle in the $X = 0$ plane. In the 3D Voronoi diagram, every point in the first half is adjacent to every point in the second half.)

   Again, all (2D) polygons are decomposable into triangles by adding only interior edges, whole not all (3D) polyhedra are decomposable into tetrahedra by adding only interior faces. Finally, for polygons, all such decompositions have the same number of triangles, while some polyhedra can be decomposed different ways into different numbers of tetrahedra.

7. Establishing error bounds on output as a function of approximations in the algorithm and uncertainties in the data is critical, (Chang and Tsai, 1991; Fisher, 1993). For example any visibility algorithm on a DEM must decide how to interpolate elevations when lines of sight pass between posts. When this algorithm is used to site an observer, what are the odds that he will have blind spots that we didn't calculate?

8. Consider *just good enough* computation, or, how do we turn input uncertainty and output sensitivity to our advantage? Given the above-mentioned input uncertainty and algorithm

sensitivity, precise output is not warranted. This apparent problem is, rather, an opportunity, to design faster, just accurate enough, algorithms. A sufficient quantitative speedup will enable a qualitative growth in the set of solvable problems, perhaps including the intervisibility problem defined earlier.

9. Finally, an open theoretical issue is why some simple algorithms, which have intolerable worst-case times, work so well in practice. Edge segment intersection and visible surface determination with a uniform grid are examples, (Franklin, 1981; Franklin et al., 1989). What does *in practice* mean? It's unfairly optimistic to assume the data to be uniform and uncorrelated, but unfairly pessimistic to make no assumptions at all. Most analytical cartography implementations that everyone uses every day can be made to fail by an adversary who selects the worst possible input.

   We tolerate that because we work in the real world, not the theoretical world of algorithms analysis. Nevertheless, a theoretical characterization of these algorithms might enable them to perform better in the real world.

5 Summary

The various applications presented above do not function in isolation, but are energized by a synergy between them. Techniques developed to solve one problem assist the solution of another. Analytical cartography draws on computer science, but also presents it with new difficult, yet worthwhile, problems to solve. Finally, the research needs continue because old solutions become dated as larger databases become available to be processed by faster machines.

6 Acknowledgements

This paper has greatly benefitted from the reviewers' detailed helpful suggestions. It was supported by the National Science Foundation, CISE/C-CR.

References

Akman, V. (1985), Shortest paths avoiding polyhedral obstacles in 3-dimensional euclidean space, PhD thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute. UMI abstract no. 85-28,651.

Aristotle (350 BC), *On the Heavens*, Vol. II.

Bentley, J. L. (2000), Cache-conscious algorithms and data structures. Bell Labs.

Braess, D. (1968), 'Uber ein paradoxon aus der verkehrsplanung [a paradox of traffic assignment problems]', *Unternehmensforschung* **12**, 258–268. (For many refs, see `http://homepage.ruhr-uni-bochum.de/Dietrich.Braess/`).

CGAL (2000), 'The CGAL home page', `http://www.cs.uu.nl/CGAL/`.

Chang, K.-T. and Tsai, B.-W. (1991), 'The effect of DEM resolution on slope and aspect mapping', *Cartography and Geographic Information Systems* **18**(1), 69–77.

De Floriani, L. (1987), 'Surface representations based on triangular grids', *Visual Comput.* **3**(1), 27–50.

De Floriani, L., Falcidieno, B., Nagy, G. and Pienovi, C. (1984), 'Hierarchical structure for surface approximation', *Comput. Graph. (UK)* **8**(2), 183–193.

De Floriani, L., Falcidieno, B., Pienovi, C., Allen, D. and Nagy, G. (1986), A visibility-based model for terrain features, *in* 'Proc. 2nd Internat. Sympos. Spatial Data Handling', pp. 235–250.

Douglas, D. H. (1983), The XYNIMAP family of programs for geographic information processing and thematic map production, *in* B. S. Wellar, ed., 'Proceedings of Auto-Carto Six', Vol. II, pp. 2–14.

Edelsbrunner, H. and Mücke, E. P. (1990), 'Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms', *ACM Trans. Graph.* **9**(1), 66–104.

Fabri, A., Giezeman, G.-J., Kettner, L., Schirra, S. and Schönherr, S. (1996), The CGAL kernel: A basis for geometric computation, *in* M. C. Lin and D. Manocha, eds, 'Proc. 1st ACM Workshop on Appl. Comput. Geom.', Vol. 1148 of *Lecture Notes Comput. Sci.*, Springer-Verlag, pp. 191–202. `http://www.cs.ruu.nl/CGAL/Papers/wacg.ps.gz`.

Farin, G. (1993), *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*, Academic Press.

Fisher, P. F. (1993), 'Algorithm and implementation uncertainty in viewshed analysis', *Int. J. Geographical Information Systems* **7**, 331–347.

Franklin, W. R. (1973), 'Triangulated irregular network program', `ftp://ftp.cs.rpi.edu/pub/franklin/tin73.tar.gz`.

Franklin, W. R. (1981), 'An exact hidden sphere algorithm that operates in linear time', *Comput. Graph. Image Process.* **15**, 364–379.

Franklin, W. R. (1992), Tutorial on curve fitting for GIS, *in* D. H. Douglas, ed., 'Proceedings, SORSA'92 Symposium and Workshop'.

Franklin, W. R., Narayanaswami, C., Kankanhalli, M., Sun, D., Zhou, M.-C. and Wu, P. Y. (1989), Uniform grids: A technique for intersection detection on serial and parallel machines, *in* 'Proceedings of Auto Carto 9: Ninth International Symposium on Computer-Assisted Cartography', Baltimore, Maryland, pp. 100–109.

Franklin, W. R. and Ray, C. (1994), Higher isn't necessarily better: Visibility algorithms and experiments, *in* T. C. Waugh and R. G. Healey, eds, 'Advances in GIS Research: Sixth International Symposium on Spatial Data Handling', Taylor & Francis, Edinburgh, pp. 751–770.

Franklin, W. R., Sivaswami, V., Sun, D., Kankanhalli, M. and Narayanaswami, C. (1994), 'Calculating the area of overlaid polygons without constructing the overlay', *Cartography and Geographic Information Systems* pp. 81–89.

Gittings, B. (1994), 'Digital elevation data catalogue', On the WWW, at `http://www.geo.ed.ac.uk/home/ded.html`, and posted regularly to `Comp.infosystems.gis`.

Gold, C. M. (1990), Space revisited: back to the basics, *in* 'Proc. 4th Internat. Sympos. Spatial Data Handling', pp. 175–189.

Goodchild, M. (1980), 'Fractals and the accuracy of geographical measures', *Mathematical Geology* **12**, 85–98.

Gousie, M. B. (1998), Contours to digital elevation models: grid-based surface reconstruction methods, PhD thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute.

Guerrieri, E. (1989), A methodology for software transportability, PhD thesis, Rensselaer Polytechnic Institute. Computer & Systems Engineering Department.

Kankanhalli, M. (1990), Techniques for Parallel Geometric Computations, PhD thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute.

Lam, N. S. (1983), 'Spatial data interpolation methods: A review', *American Cartographer* **10**(2), 129–149.

LEDA (2000), 'Leda research', `http://www.mpi-sb.mpg.de/LEDA/`.

López, A. M. (1997), 'Ridge/valley-like structures: Creases, separatrices and drainage patterns', `http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/LOPEZ/cvonline.html`.

Mark, D. M. (1984), 'Automated detection of drainage networks from digital elevation models', *Cartographica* **21**, 168–178.

McCormack, J. E., Gahegan, M. N., Roberts, S. A., Hogy, J. and Hoyle, B. S. (1993), 'Feature-based derivation of drainage networks', *Int. J. Geographic Information Systems* **7**(3), 263–279.

Mehlhorn, K. and Näher, S. (1995), 'LEDA: a platform for combinatorial and geometric computing', *Commun. ACM* **38**(1), 96–102. `http://www.mpi-sb.mpg.de/guide/staff/uhrig/leda.html`.

Moellering, H. (1991), Research issues relating to the development of cartographic database transfer standards, *in* (Müller, 1991).

Müller, J. C., ed. (1991), *Advances in Cartography*, Pergamon.

Narayanaswami, C. (1991), Parallel Processing for Geometric Applications, PhD thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute. UMI no. 92-02201.

Narayanaswami, C. and Franklin, W. R. (1992), Boolean combinations of polygons in parallel, *in* 'Proceedings of the 1992 International Conference on Parallel Processing'.

Pedrini, H. (2000), An Adaptive Method for Terrain Approximation based on Triangular Meshes, PhD thesis, Rensselaer Polytechnic Institute, Electrical, Computer, and Systems Engineering Dept.

Peucker, T. K. and Chrisman, N. (1975), 'Cartographic data structures', *The American Cartographer* **2**(1), 55–69.

Ray, C. K. (1994), Representing Visibility for Siting Problems, PhD thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute.

Speckmann, B. and Snoeyink, J. (1997), Easy triangle strips for TIN terrain models, *in* 'Proc. 9th Canad. Conf. Comput. Geom.', pp. 239–244.

Stecchini, L. C. (2000), 'Ancient measurements of the circumference of the earth', `http://www.interpres.cz/metrolog/measures/measurements.htm`.

Tobler, W. (1966), Numerical map generalization, *in* J. D. Nystuen, ed., 'Discussion papers of the Michigan interuniversity community of mathematical geographers', Vol. 8, `http://www.imagenet.org/`.

Tobler, W. (1999), 'Converting administrative data to a continuous field on a sphere', `http://www.sbg.ac.at/geo/idrisi/gis_environmental_modeling/sf_papers/to%bler_waldo/tobler_waldo.html`.

US Dept of the Interior and US Geological Survey (2000), 'SDTS (Spatial Data Transfer Standard) home page', `http://mcmcweb.er.usgs.gov/sdts/`.

US Geological Survey (1996), 'Global drainage basins database', `http://grid2.cr.usgs.gov/dem/global.html`.

Veregin, H. (1989), Error modeling for the map overlay operation, *in* M. Goodchild and S. Gopal, eds, 'The Accuracy of Spatial Databases', Taylor & Francis, pp. 3–18.

White, D. (1977), A new method of polygon overlay, *in* 'An Advanced Study Symposium on Topological Data Structures for Geographic Information Systems', Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, MA, USA, 02138.

Wood, J. D. and Fisher, P. F. (1993), 'Assessing interpolation accuracy in elevation models', *IEEE Computer Graphics And Applications* **13**, 48–56.

Wu, P. Y. F. (1987), Polygon Overlay in Prolog, PhD thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute.

Zimbelman, J. R. and Tanaka, K. L. (1999), Planetary mapping activities and plans of the national aeronautics and space administration (NASA), *in* C. P. Keller, ed., 'ICA1999: 19th International Cartographic Conference', International Cartographic Assoc., Ottawa.