BUT IS IT GIS? [1]

Christopher Gold [2]

Department of Land Surveying and Geo-Informatics, Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Abstract

GIS as we know it is traditionally a two-dimensional static representation of space, closely related to the paper maps that were its inspiration. While various attempts have been made to define a "3D GIS" or a "Dynamic GIS", in recent years other disciplines have moved much faster in this direction – notably some parts of computer science, computer graphics and game development. This paper looks at this situation, and suggests some remedial actions in the geography (and surveying) communities. These consist primarily of a stronger emphasis on teaching computer algorithms as applied to spatial analysis, as well as a familiarity with the basics of computer graphics and spatial data structures. I hope to show that a modest subset of these tools is sufficient to lead towards the modelling of the third dimension and time. However, one worry remains – is this still GIS?

1 Introduction – A Call to Arms

I would like to talk about the underlying processes at work in developing new GIS tools - and how we in the GIS community are in danger of falling even further behind the mainstream of computer science activity.

I will summarize quite a lot of technical tools, coming from GIS, computer science, graphics programming, etc., and give demonstrations of what we have achieved and are working on - mostly in the region of topological "connectedness" and visualization. We will start by reviewing basic 2D algorithms, as more or less accepted by the graphics community, move on to the horribly named "2.5D" stuff, where elevations are pasted onto 2D, continue with "real" 2D surfaces or manifolds, and finally touch on full 3D or "solid" modelling - in addition considering the time dimension, at least in the form of dynamism and interactivity.

However, the moral of all this - there is always a moral - is that it is not at all obvious that this is "GIS'. Indeed, it is not at all obvious that GIS remains relevant. On the one hand it is clear, as David Rhind says, that GIS is becoming "just part of IT". We saw this happening in the 1970s, with "Business Graphics", when pie charts were things of wonder. Obviously a lot of mapping, and simple attribute queries, is already just IT (although it is not yet certain that map construction, as opposed to viewing, has yet gone down that plughole). On the other hand, programming, computational geometry, computer graphics, CAD, game development and many other sub-species of computer spatial analysis and visualization are moving ahead with ferocious speed, and we can not readily keep up with them. We run the real risk of sliding down two slippery slopes simultaneously - financial irrelevance and computer science backwardness. So what can we do? Where do we fit? Should we look for other jobs already?

On the IT side I have little to suggest, although spatial database design, or organizational GIS, offer some hope. If Bill G. thinks that Windows needs GIS, then that is what it will get. However, I see some hope on the outer fringes, where we abandon the fertile (2D) plains to the suits, and escape into the hills of esoteric (and higher dimensional) development - to mix my metaphors madly. We have been left behind by the money-men - we can't afford to be left behind by every teenage game developer as well.

So the rot stops here (there aren't many bucks left). We know we have some special (indeed invaluable) skills: we know what makes sense, and doesn't make sense, in that most confusing of reality's concepts - space itself. We know how far to take Tobler's "Everything is related to

Journal of Geospatial Engineering, Vol. 5, No.2 (December 2003), pp.11-26.

everything else", and when to ignore it. We know when boundaries are meaningful, and when they are just a cartographic joke. We know that slicing things into categories (spatial, temporal, attributal or other "matters metaphorical") messes up our data, but we know we have to do it. So we are useful. The problem is we are mostly old-fashioned - and fashion means computer science skills. Not only that, but no game developer would dream of using either our visual display or our interfaces. We need to catch up. But most of us don't have enough computer science confidence, and our students often live in a maths-free zone.

2 A Way Forward

Luckily, many parts of computer science basics have stabilized in the last few years, and advanced computer systems and graphics are available on any desktop PC. So now is a good time to introduce a set of "advanced" skills into the GIS curriculum – skills that were only available to the experts (or at least specialists) a very few years ago. While computer science moves on faster than ever before, it leaves behind an increasing collection of "established practices" that we relatively "non-specialists" can adopt. Some of these can be summarized as:

a) The basics of good software development have stabilized recently with object-oriented (O-O) programming and design. While there will undoubtedly be a Next Great Thing, we have several years in which we can adopt and use these ideas. In addition, a top-down approach to problem solving fits well within our current best teaching practice. It is many times better than the spaghetti code (or even structured code) previously taught. The trick is: catch them young!
b) Powerful graphics (e.g. OpenGL) are readily available and are easy to use. As with the basics of O-O, a simple introductory program is sufficient to get students started, and to modify for subsequent projects. If you can do 3D graphics in your first exercise, all kinds of possibilities open up.
c) Computational Geometry has dealt with many (not all) of the theoretical issues about line intersections, spatial adjacency, spatial data structures, etc. While it is always fun to develop new techniques, in many simple cases there are well- defined and stable methods.
d) A brief summary of the basics of a course in Data Structures would be helpful. This can be simplified quite a bit, but the basic ideas are very useful.
e) Good development environments are free. All of our teaching work is done with Borland Delphi Personal Edition, downloadable from www.borland.com. (For research work we sometimes use the commercial version, but often this is not necessary.) A strongly typed O-O language like Delphi (Pascal) or Java is much preferable to C/C++, which is highly dangerous to the untrained. (It is not just the language that is important – other issues are rapid compilation, debugging and coding tools, and a good IDE, or Interactive Development Environment.) For the graphics part, OpenGL is either included with the programming package, or may be downloaded separately.
f) Just do it! Even Professors can learn. We have found that careful instructions, a CD with demo programs, and an hour of tutorial help is enough to get students started, even on their home computers.

3 A Study Outline

I am here proposing an approach to programming literacy to teach old dogs new tricks. Rather than describe all possible approaches, in a classical academic fashion, I will select a direction that achieves my objectives – and leave it as an exercise for the reader to explore further. My approach basically consists of stealing from the first chapters of many texts, on the basis that these are simple enough for the non-specialist, even if things get complicated later in the book. I will try and give you a quick background, or course outline. Of necessity, this introduction can supply little more than keywords.

3.1 Objects

The usual: fields, methods, encapsulation and inheritance. The important thing is that this is not procedural programming – it is based on messages: when a button is clicked, for example. Start with top-down design, and worry about how later. Graphics are an excellent way of achieving this, as visual feedback teaches very well. This gives a particular advantage to GIS people, as we are particularly interested in graphical display anyway.

3.2 Graphical Transformations

These are now pretty standard in Computer Graphics. They typically include rotation, scaling and translation (displacement) matrices in 2D and 3D. The key idea is concatenation of the matrices, using homogeneous coordinates. This means that any combination of scaling, rotation, perspective, etc., can be expressed as one matrix, which is used to reposition the object – either because it has moved or because the observer (camera) has changed position. OpenGL has these all built in, together with the idea of a transformation stack - which means that you can save the current position in the coordinate space, create a new position in order to define an object, and then revert to the original position. (While extremely easy to do, this takes a little more explanation to make the applications clear!)

3.3 The First Program

Contrary to many peoples' belief, object-oriented programming is not difficult to understand - unless you learned procedural programming first! The "OnPaint" or equivalent command makes perfect sense in a Windows-type message-passing world. A brief introduction to this, plus one on OpenGL as a state-machine, the provision of a "First Program" to get over the hump of connecting Windows to the graphics, and students can be drawing their very own triangles (in colour) in the first hour. Add a little mathematics about transformation matrices, and the magic of a transformation stack, and a working model solar system can be built in the next hour (Fig. 1), convincing students of their own skills.

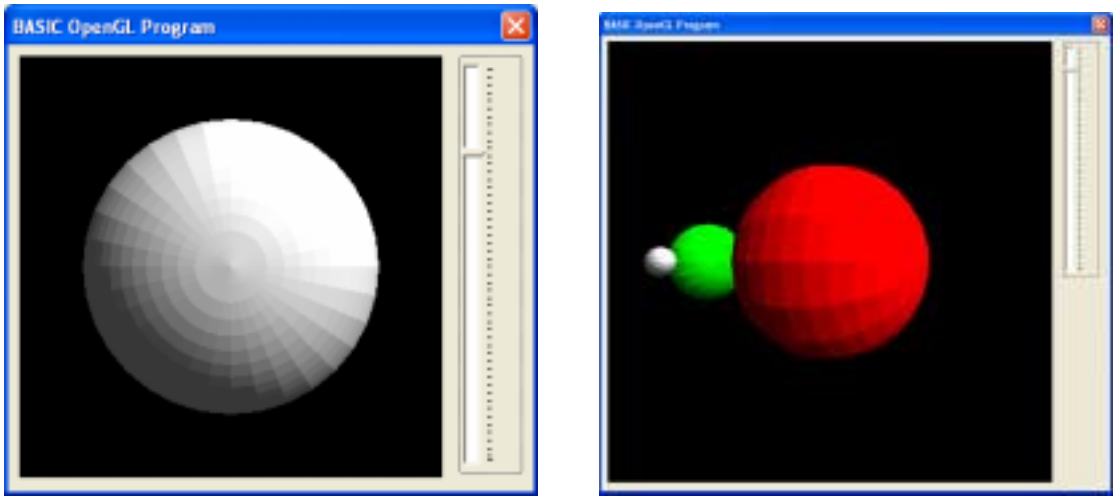

Figure 1: (a) the "First Program" as given; (b) the "Solar System"

3.4 Vector Algebra (Oh No!)

It is most unfortunate that vector algebra has such a bad name! If vectors are thought of as positional, graphical entities, they are easy to follow, and immediately useful. We really only need a few equations: the scalar product to project one vector onto another, the vector product to calculate the normals of triangles for slopes and shading, and the triple scalar product or signed triangle area ("CCW" in computational geometry) in order to find out if a point is to the left or right of a line, or the ordering of the vertices of a triangle. (For Voronoi diagrams or Delaunay triangulations one more test, "InCircle", is needed.) All of these methods work for two, three (or

more) dimensions, and give a standard, robust set of “Geometric Operations”. Almost all common cases for line intersections, area calculations, polygon centroids, linear interpolation in a triangle, TIN model normals, visibility and slopes, and digitizing table transformations and “rubber sheeting” may be done - they are so easy that it is embarrassing that everyone in the spatial sciences is not familiar with them.

Figure 2 shows the next programming projects. Firstly the original sphere is replaced by a triangle, and its vector normal is displayed, and secondly a set of triangles from a TIN are read in and displayed with the appropriate lighting direction. This is not difficult to achieve.

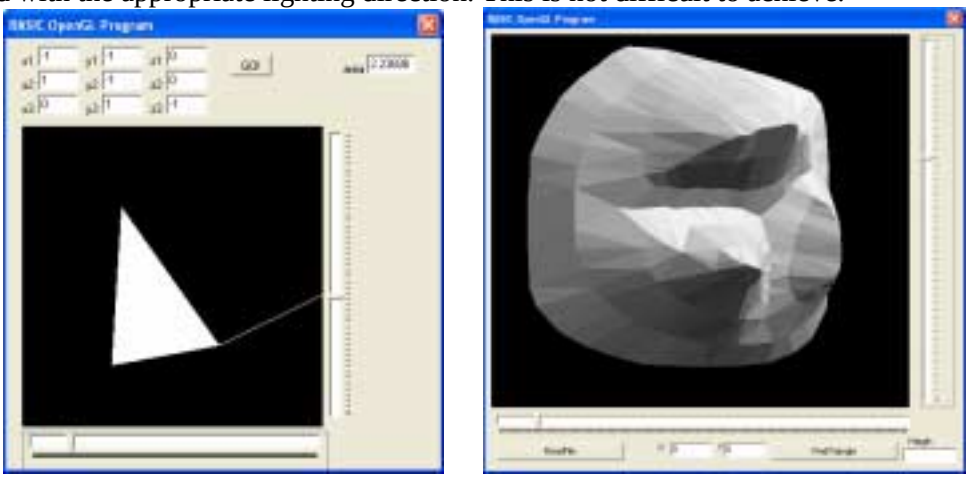

Figure 2: (a) a single triangle and its normal; (b) a set of triangles

3.5 One-dimensional data structures

We should now go back to computer science, and learn about linked lists, stacks, queues and trees. An O-O model makes this almost as easy to do as to describe, and tree traversals (or the Douglas-Peucker line generalization algorithm) provide great examples of divide-and-conquer problem solving. We could have a whole Computer Science course here, but we only need to review one dimensional structures: linked lists, stacks, queues and trees. Delphi’s object reference model is a big advantage here, as we have no need of pointers to connect one node in a data structure to another.

3.6 Two-dimensional structures: graphs

Once we have a grasp of data structures in one dimension, as used in searching and sorting, we may extend ideas where we want to go – into two-dimensional space. We start with connected networks, or graphs, and the basic methods for storing and traversing them. Many of us have seen these techniques in examples such as traffic flow and shortest-path or minimum-spanning-tree analysis. In an O-O environment these techniques are quite simple to implement, the methods are well known, and students are usually surprised by the power of a few algorithms, such as that of Dijkstra, as well as data structures that can easily handle weighted, non-weighted, directed, non-directed, planar and non-planar graphs to solve a variety of practical problems. In the GIS context, the key issue is the recognition of planar graphs, which permit the recognition of “regions” in the plane, defined by cycles of edges.

4 Surface modelling

4.1 Graphs on Surfaces

While the idea of planar graphs is usually understood, we need to borrow a simple idea from topology to take advantage of the power of the concept. The graph is “embedded” in a surface, which may then be distorted or stretched without losing some basic properties. In particular, the connectivity of nodes and edges, and the ordering of edges around each node, must be preserved.

With these properties, for example, it becomes easy to use a breadth-first-search to detect all of the individual “polygons” in a network of boundaries.

Graphs on surfaces (or manifolds) are fundamental to surface modelling – from TINs to dinosaurs. We now have an interlinked structure of nodes, edges and faces, and we need an appropriate data structure to express these “dual” relationships. The “Quad-Edge” structure is perhaps the simplest to use for serious surface modelling, as it contains pointers from each edge to both adjacent vertices and both adjacent faces, as well as to loops around each of them. Having only two operations (“Make-Edge” and “Splice”) it is simple to code – only a few lines in each case. As these structures are appropriate for all embedded graphs, they can form the basis for two-dimensional polygon sets or triangulations, as well as for TIN models, where a height attribute is added to each node. Nor are they limited to the planar case – they may form the basis of complex surface models as used in game development.

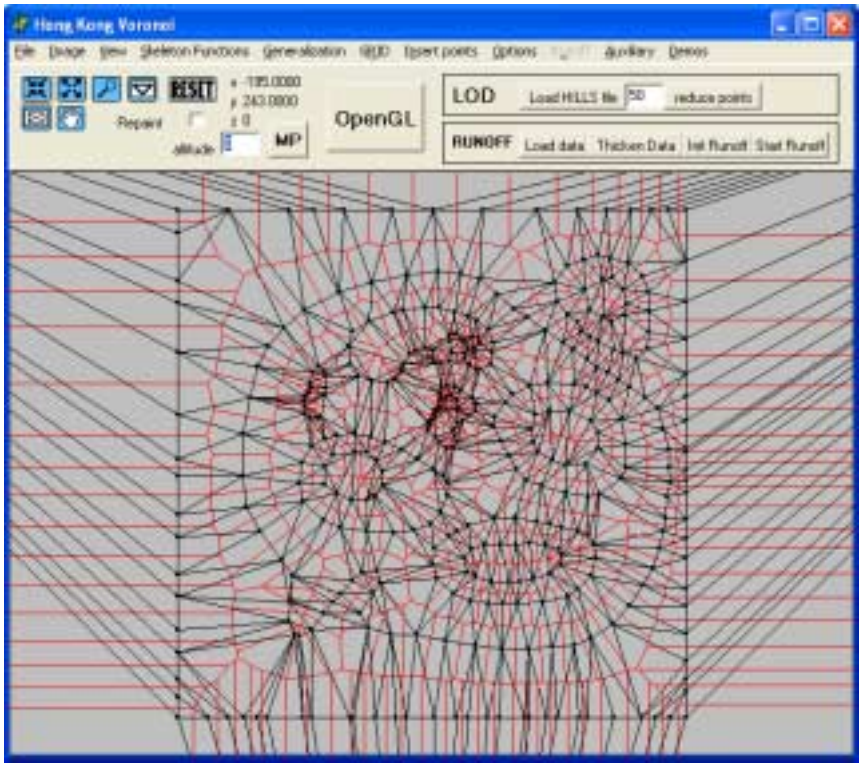

Figure 3: The Voronoi diagram and Delaunay triangulation

In most cases, the Delaunay triangulation is the preferred definition of a “good” triangulation, partly because of its relationship to the Voronoi diagram, and partly because it is therefore locally stable, and in most cases inserting, deleting or moving a vertex will only make local changes to the structure. Incremental algorithms for inserting and deleting vertices may be developed, based on the previously mentioned “CCW” and “InCircle” tests (Fig. 3). While more complex, and theoretically more efficient, algorithms may be developed, the simple incremental algorithm is usually found to be sufficiently fast and quite robust. When used with the Quad-Edge structure students can create their own TIN models fairly easily. Fig. 4 shows a 3D model based on the exercise shown in Fig. 2.

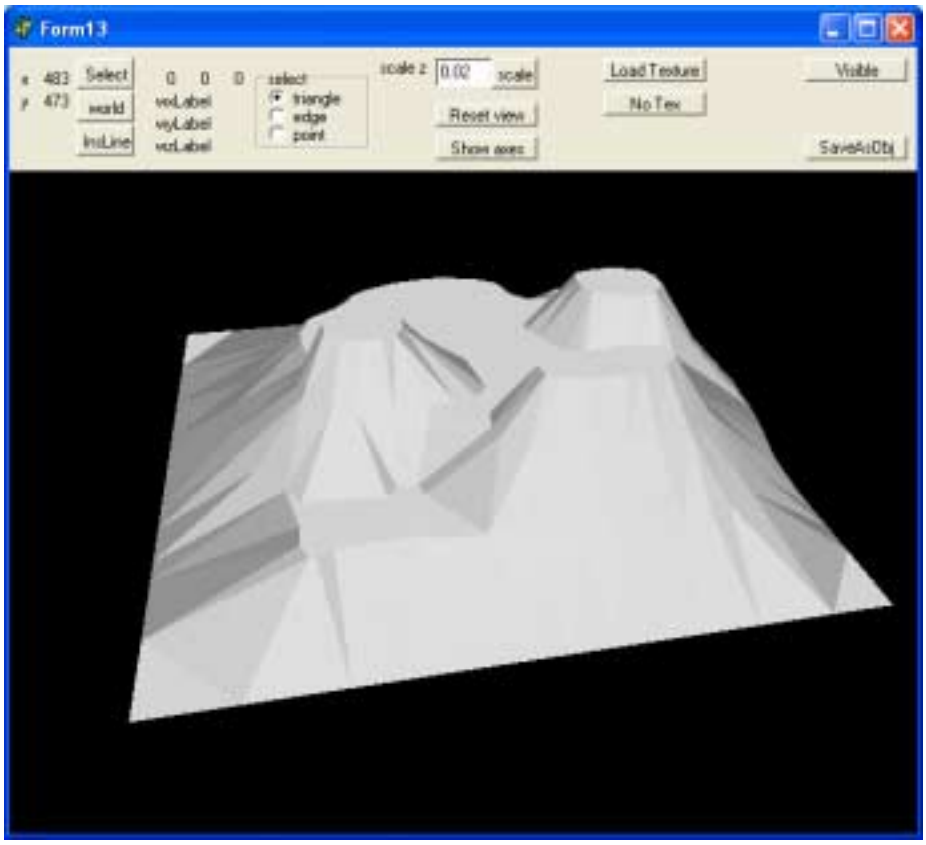

Figure 4: A 3-D view of the data of Fig. 3.

Fig. 5 attempts to illustrate that the Voronoi diagram need not just be static – it may be dynamic or kinetic as well. It is possible to move the central point in this image (a boat, perhaps, between the two "river banks") while maintaining the Voronoi/Delaunay properties and the associated spatial relationships. This has obvious potential applications in navigation and collision detection.

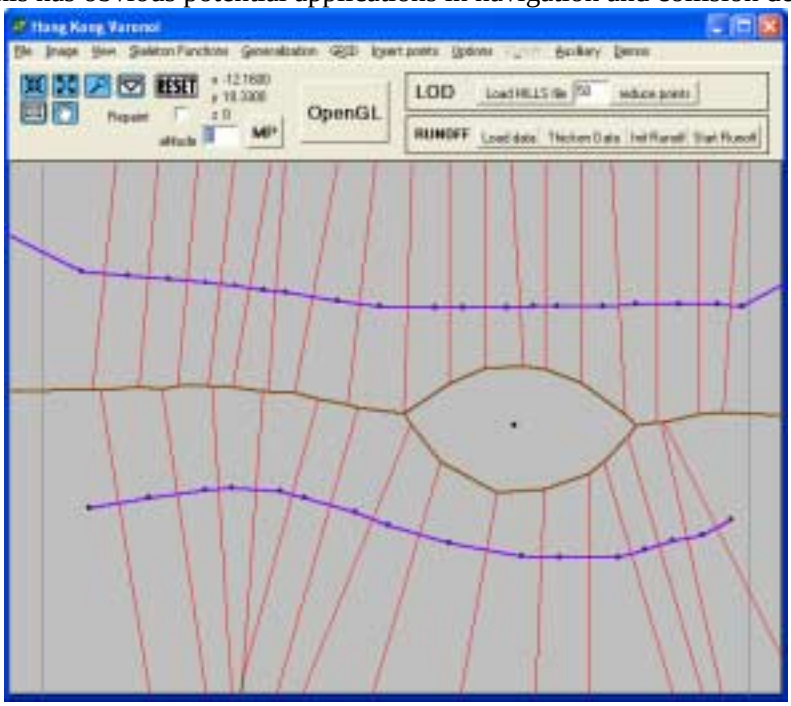

Figure 5: Navigation using dynamic/kinetic Voronoi diagrams

Fig. 6 is another view of Fig. 3, where we can see that the original data were in the form of digitized contour lines. Based on the recognition that each Voronoi edge must have a dual Delaunay edge, a simple test may be used to test whether the Delaunay edge or the Voronoi edge should be drawn. In the first case we have a portion of the "crust", connecting the data points that form the contours. In the second case a portion of the "skeleton" is drawn, forming the medial axis separating different portions of the contours. This elementary test permits a variety of useful applications. In our terrain modelling application, we can use these skeleton points to address the problem of all the "flat" triangles in Fig. 4. As a first step, skeleton points are guaranteed to break up all flat triangles formed at ridges, valleys, pits and summits. As a second step, these points may be assigned elevations based on the relative sizes of their circumcircles. We can thus create an enriched point data set, which generates the 3D view shown in Fig. 7. As can be seen, ridges and valleys can be formed from the original contours in quite a plausible fashion. (In fact, they are based on the hypothesis of equal side-wall slopes, which is an acceptable model in many real-world cases.)

While there is not space here to provide more illustrations, the crust and skeleton are also valuable in the estimation of the centrelines of polygonal objects (e.g. character recognition, or the veins of a leaf), for the estimation of preliminary watersheds based on the hydrography alone, and in the extraction of topology from scanned maps.

We now have a proper TIN model by just adding elevations. However, good surface models are non-trivial, and we need to look carefully at our desired surface properties, our type and distribution of data, and the various interpolation techniques. Irregularly distributed data that is nonetheless precise takes careful handling.

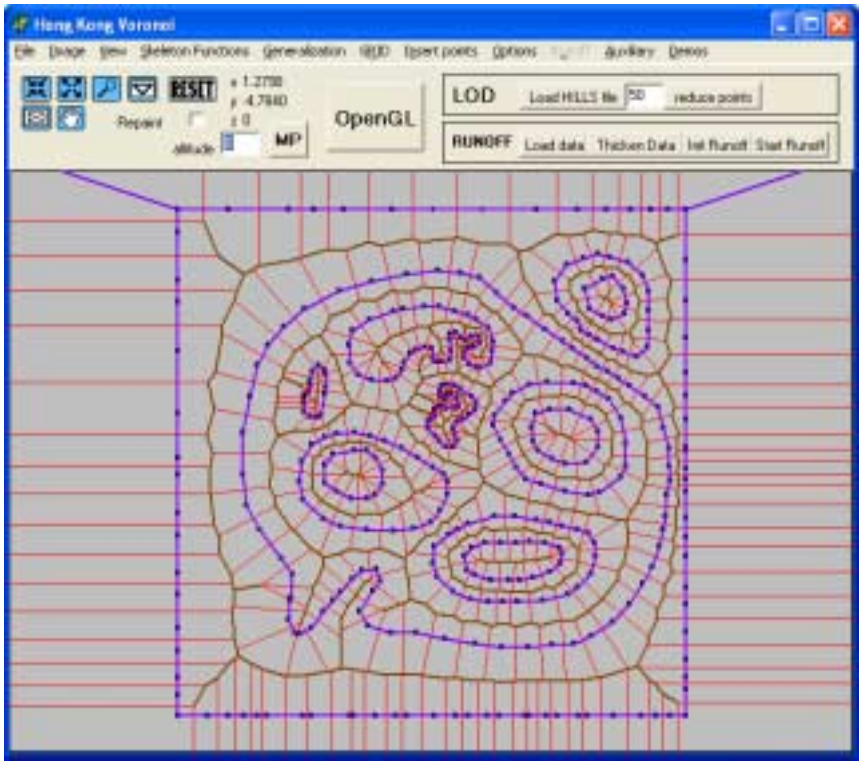

Figure 6: Crust and Skeleton derived from Fig. 3.

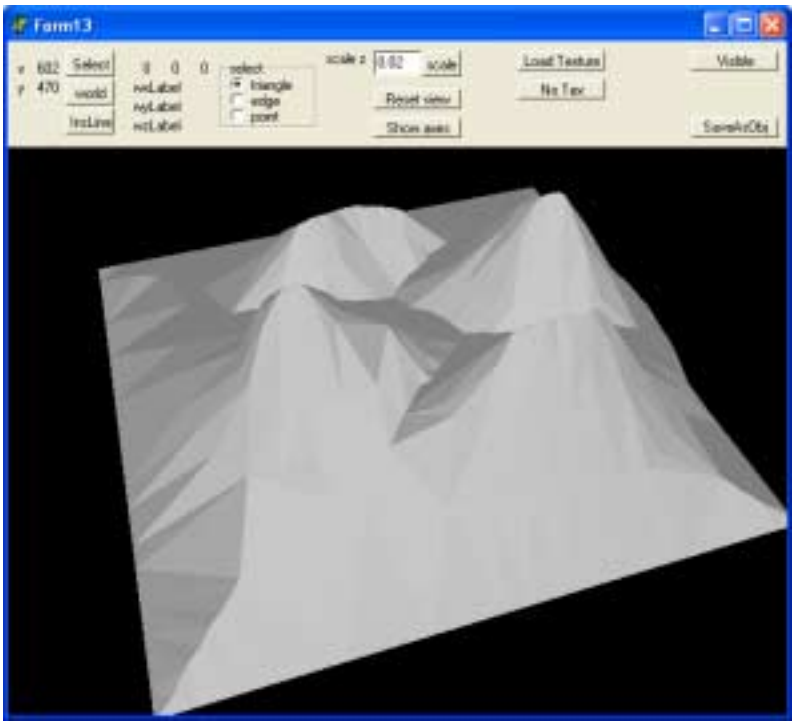

Figure 7: Terrain model with added ridges and valleys.

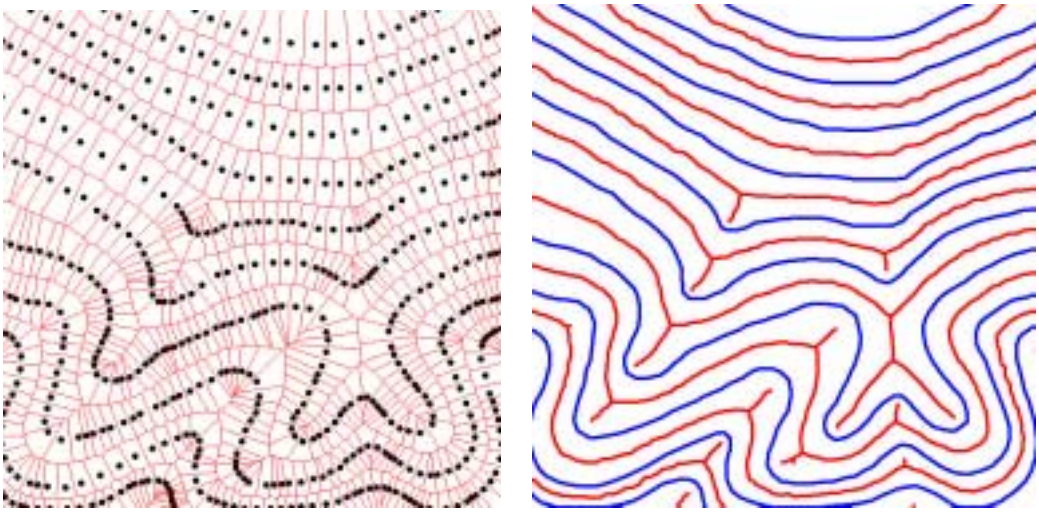

Figure 8: a) Contour data and Voronoi cells; b) Crusts and skeletons

4.2 Terrain interpolation – in particular from contours

One particularly interesting problem arises frequently in GIS projects. Despite modern data capture technologies, we are frequently required to extract our terrain models from previously-acquired contour data. This poses specific problems. The data densities along contours are much higher than between them, so many local averaging techniques fail. In addition, TIN models have "flat triangles" where the triangles connect points of equal height. We can use our crust and skeleton techniques to handle this. Fig. 8 shows a simple contour map with Voronoi cells around the data points, and the resulting crusts and skeletons. The skeleton branches break up the flat triangles, and heights are estimated. Thus the main form of our imaginary terrain is defined. Fig. 9 shows the triangulated surface before and after insertion of the skeleton branch points.

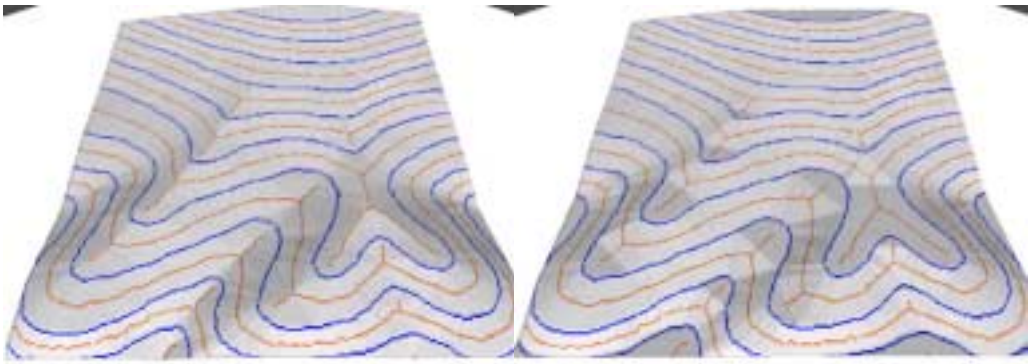

Figure 9: a) Surface with flat triangles; b) After insertion of skeleton points

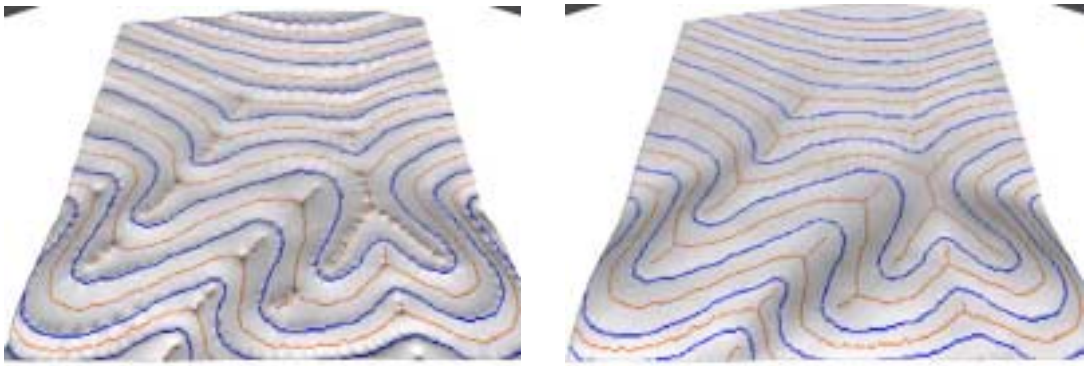

Figure 10: Interpolation a) gravity model; b) Sibson method with slopes

4.3 Slopes – the ignored factor

In real applications accuracy of slopes is often more important than accuracy of elevation. However, simple weighted averages often produce poor slope values, as in Fig. 10a. However, in the weighted-average process we can replace the height of each neighbouring data point by the value of a function involving the data point height and local slope, which greatly improves things. Our best results have been with Sibson interpolation (also called natural neighbour, or area-stealing) where the query location is inserted into the Voronoi diagram (cf. Fig. 5) and the areas lost from the neighbouring Voronoi cells form the basis of the weighted average (Fig. 10b). By now we are back in the realm of computational geometry, where a variety of easily implementable algorithms allow us to readily insert and delete points, giving us a dynamic data structure.

4.4 Flow Modelling

One of the obvious cases where good slope modelling is important is in runoff modelling. Once we have a good interpolation scheme we can build a cell structure for finite difference flow between cells. This is usually done on a grid, but the bias of the grid directions produces unfortunate artefacts, such as parallel rivulets in vertical, horizontal or diagonal directions. A better method is to densify the original data (of Figs. 3, 6 and 7) with random points of fixed rejection radius and interpolated values, as in Fig. 11a. Fig. 12 shows the interpolated surface based on these random locations, and Fig. 11b gives the result of the runoff model after a fixed number of iterations.

5 Beyond Two Dimensions

5.1 TIN model with quad-edge data structures and Euler operators

Nevertheless, underneath it all, this is still 2D, dependent on (X, Y) location. Any self-respecting game now sports 3D monsters, modelled with triangular meshes. Armed with our Quad-Edge data structures, and a little knowledge of CAD system design, we may redesign our TIN model with Euler Operators - which permits us to extend our surface with extruded buildings, as well as

bridges and tunnels. This is particularly useful when we wish to do surface runoff modelling, as buildings, dams, etc. are part of the landscape, and affect our water flow.

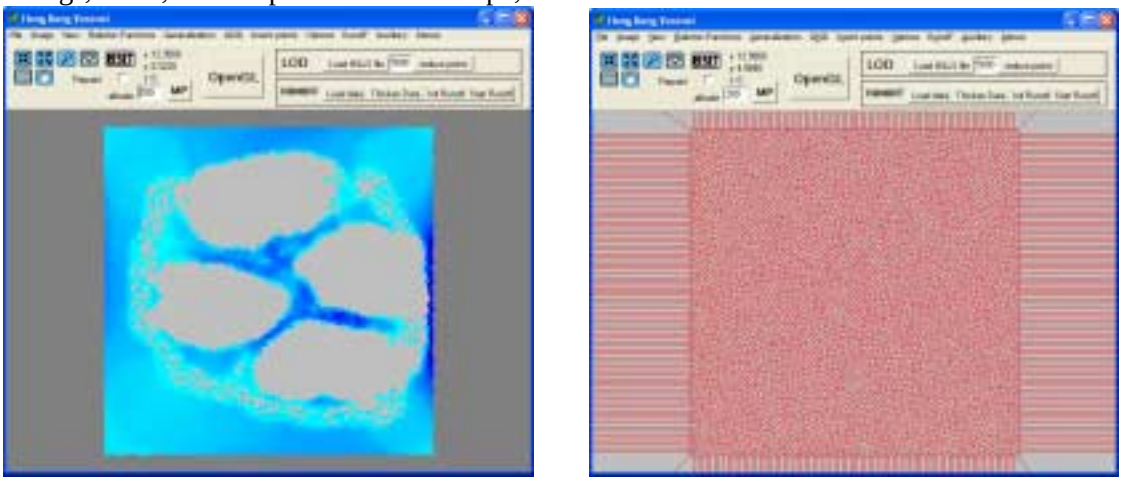

Figure 11: a) Densified mesh of Fig. 6; b) Finite-difference runoff

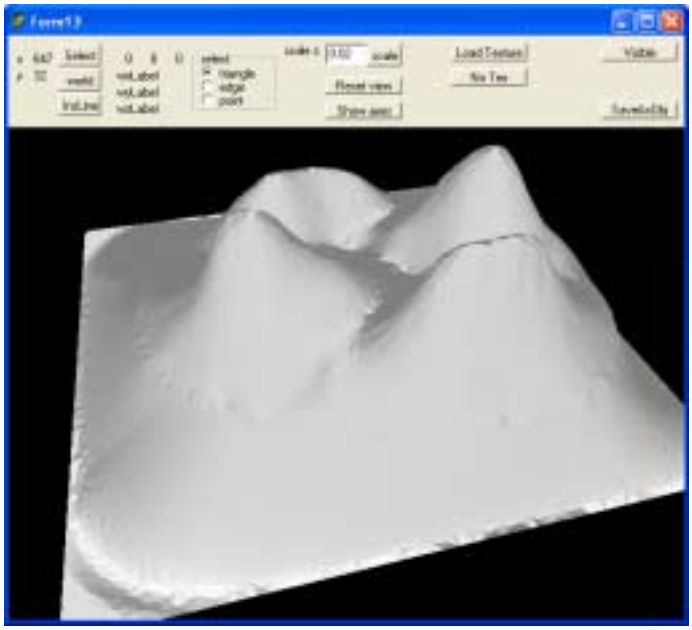

Figure 12: Interpolated surface model from the mesh of Fig. 11a

Euler operators were developed for CAD systems to guarantee that the surface of some three-dimensional model remained connected when modifications were made. They are used today in the mesh models used for computer games or films, and are usually based on the "winged-edge" data structure. However, in our recent work we have shown that elegant and greatly simplified methods may be developed based on the simple TIN model (now a closed polyhedron, not a modified plane) and the Quad-Edge data structure.

The usual "easy" Voronoi algorithm is incremental: find the enclosing triangle for the new point, split the triangle, and test the edges recursively to see if they are Delaunay (having an empty circumcircle); if the test fails, switch the diagonal of the relevant triangle pair and continue. The mesh initialization, "split" and "switch" operations on the data structure may be performed using the operators. Initialization (MEVVFS – Make Edge, Vertex, Vertex, Face, Shell) creates a new object, and its inverse KEVVFS deletes it. MEF makes a new Edge and Face, and KEF removes them, as shown in Fig. 13. SEMV Splits an Edge and Makes a Vertex, and JEKV does the inverse, as in Fig. 14.

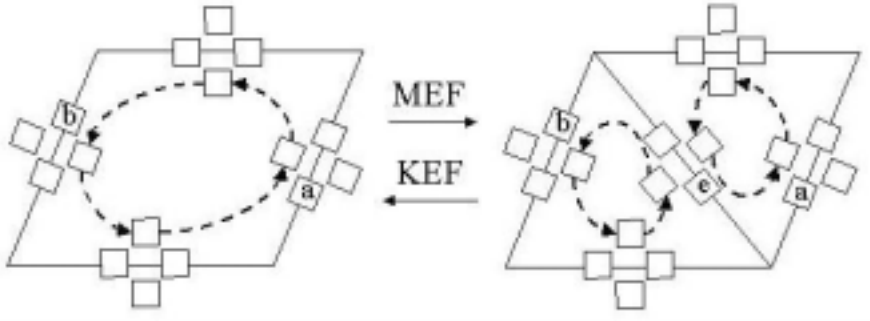

Figure 13: The MEF and KEF Euler Operators

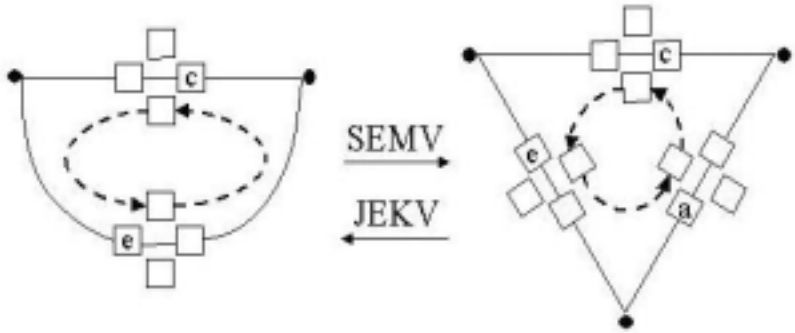

Figure 14: The SEMV and JEKV Euler Operators.

The advantage of Euler Operators, however, comes with the additional operators MEHKF to make an Edge and Hole, and in the process Kill a Face, and its inverse KEHMF. Fig. 15 shows the basic operation. In Fig. 15a two non-adjacent triangles are connected by an edge – removing the faces of these triangles to become a hole, and adding a single face inside the hole. In Figs. 15b and 15c MEF is used to form the other faces of the hole. These operations are identical whether a hole or a bridge is being formed. Fig. 16 shows a hole formed in a TIN, and expanded by edge swaps. Fig. 17 shows two extruded buildings, and a bridge formed between them.

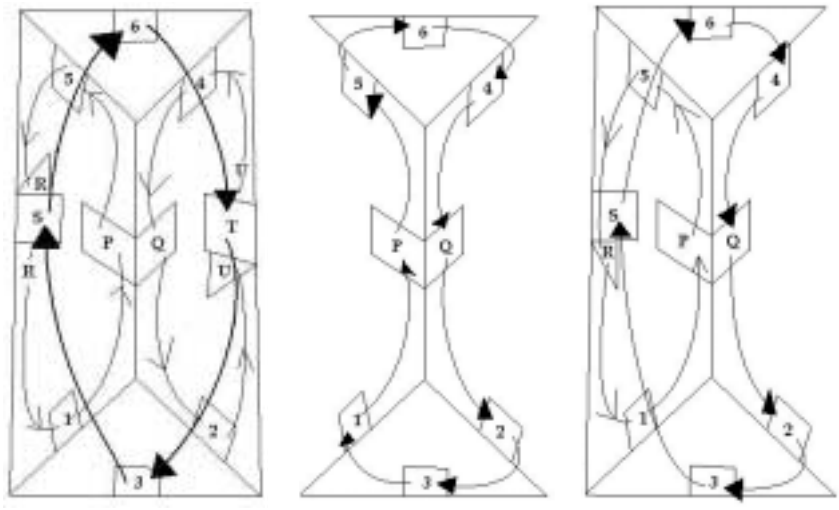

Figure 15: Creating a hole with the MEHKF, MEF and MEF Euler Operators

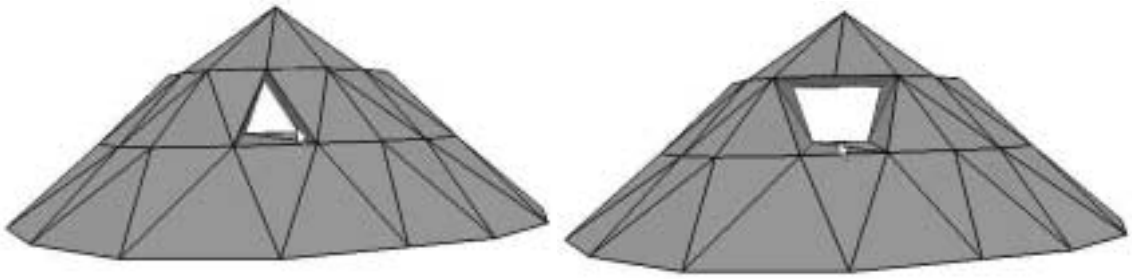

Figure 16: a) A triangular hole; b) A hole enlarged by edge swaps

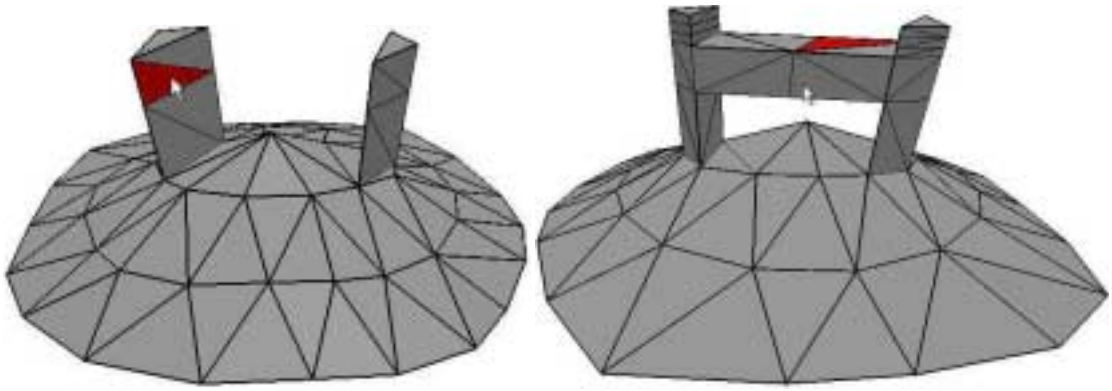

Figure 17: a) Two simple buildings extruded; b) a bridge connects them

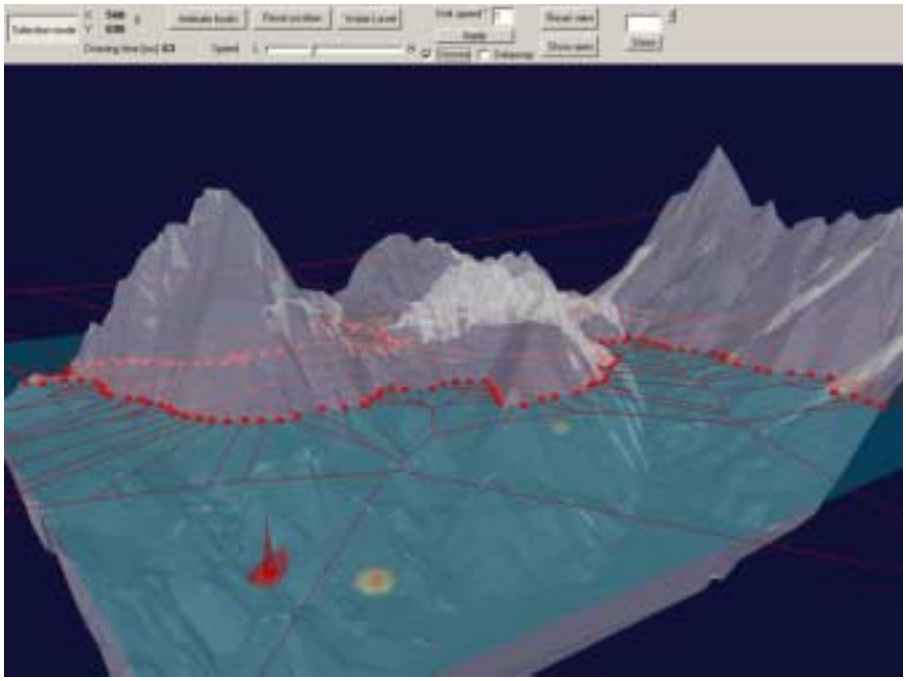

Figure 18: A Prototype Marine GIS

5.2 Marine GIS

An interesting related development is the idea of a "Marine GIS". Part of this would be fully 3D, but part of the concept is surely the idea that, unlike on land, things move all the time. Thus "kinetic" data structures are needed, allowing navigation, collision detection and the existence of ephemeral objects and obstacles. The moving-point Voronoi diagram is useful here - for 2D navigation, real-time depth estimation from rapidly changing data, real-time queries, moving-cell flow modelling and perhaps line-segment Voronoi diagrams for boundaries. This work is ongoing,

and it is not at all clear what a marine GIS really is, and what tools it needs to have. However, Fig. 18 shows a preliminary idea, based on multiple surface meshes. One mesh represents the terrain, implemented with Quad-Edges as described previously. A second mesh is a moving-point Voronoi diagram in two dimensions, representing the water surface. As can be seen, moving objects (boats) and static objects (points on the coastline) are placed in the dynamic layer – indeed, the intersection of the topography/bathymetry with the sea surface is defined dynamically, to take account of tidal changes. Boats may navigate so as to avoid (or, at present, merely detect) collisions based on the dynamic layer. They may also navigate on the basis of bathymetry, as their relation to the topographic layer is preserved, and the appropriate interpolation (and slope) calculations may be performed to lead towards the deepest channel.

5.3 Volumetric Modelling

From here we can move on to full 3D modelling - not 2D surface manifolds, as above, but volumetric modelling of arbitrarily distributed data. Our 2D models can be extended to 3D Voronoi/Delaunay structures, using the “InSphere” test equivalent to the two-dimensional InCircle test described above. We use a tetrahedral data structure, as Quad-Edges are not appropriate here. However, this presents us with a problem: how do we visualize our results? OpenGL translucency helps here, and Fig. 19 shows a set of points in 3D, together with the 3D Voronoi diagram outlined in red, and one Voronoi cell highlighted in blue. Fig. 20 shows the tetrahedral data structure on the left, and another 3D Voronoi cell on the right.

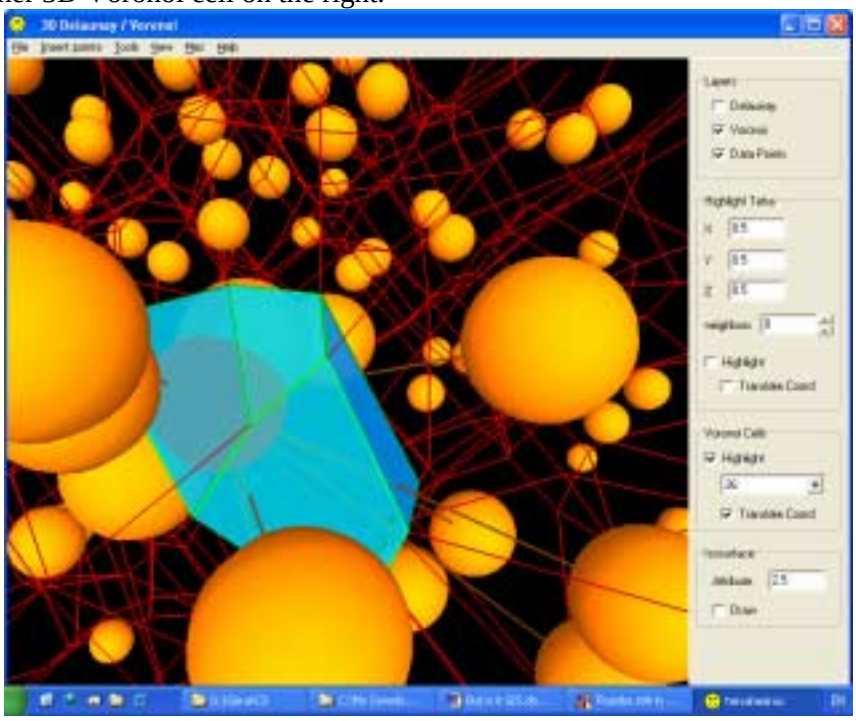

Figure 19: The Three-Dimensional Voronoi Diagram, with one cell highlighted

In practice, a volumetric 3D model of this type is relatively new, and the possible applications are largely speculative. In the marine context, a dynamic model could be used for modelling and collision detection, or for the examination of changes in water parameters, such as temperature or pollution. In practice, visualization requires translucency, and isosurface extraction (for example using the “marching tetrahedra” algorithm) give the equivalent of contours in 3D (Fig. 21). We are now approaching the basics of a true “3D GIS” (with the addition of links to our attribute database) - but what should it do? As yet we are unclear. Applications include 3D point location, full 3D Sibson interpolation, extraction of 3D crusts and skeletons from point clouds, and more.

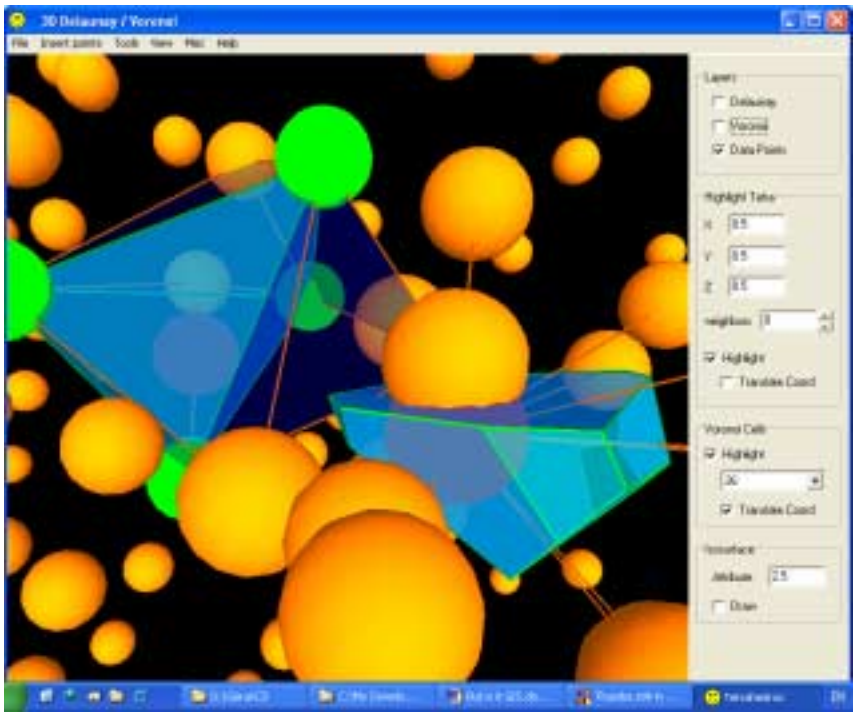

Figure 20: A Tetrahedral Element on the left, and a Voronoi Cell on the right

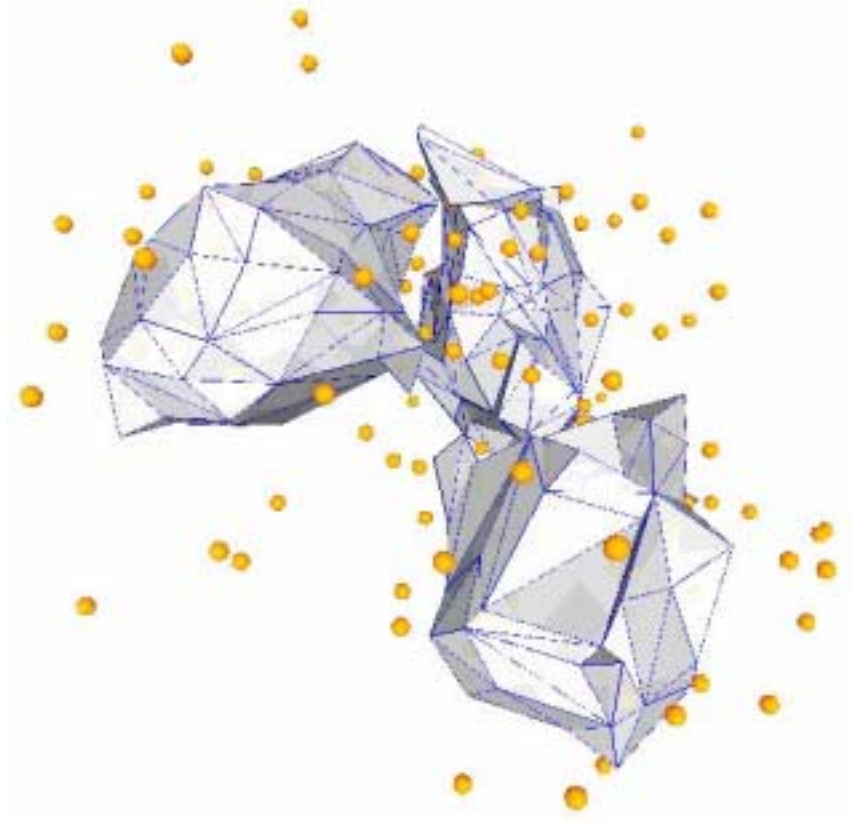

Figure 21: Isosurface extraction from 3D point data

6 But is it GIS?

In this presentation we have seen the development of a variety of spatial analysis and visualization techniques, based on stealing ideas from the introductions to a variety of disciplines. We have seen how a basic grounding in vector and matrix algebra, graphics, object-oriented programming and

data structures can leads us to a stage where we can develop some of the 2D, dynamic, kinetic, interactive, manifold-based tools, plus true 3D visualization, that we have been talking about. But it is not clear that we are talking about GIS any more – games: yes; 3D modelling: yes; simulation: yes. But there has been little reference here to standard GIS operations – indeed, it is not clear how many 2D spatial analysis questions translate to 3D. Perhaps GIS is inherently 2D and, like business graphics twenty years ago, is just becoming IT.

Which brigs us back to the initial question. This presentation has been about spatial analysis questions, dependent on both an understanding of the spatial relationships and on the latest computer science techniques. But is it GIS? And is it something that we, and our students, are prepared for?

Bibliography

Amenta, N., Bern, M. and Eppstein, D., 1998, The crust and the beta-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, **60**, pp. 125-135.

Aurenhammer, F., 1991, Voronoï diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, **23**, pp. 345-405.

Blum, H., 1967, A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, edited by Whaten-Dunn, W., (Cambridge, Mass.: MIT Press), pp. 153-171.

Blum, H., 1973, Biological shape and visual science (Part 1). *Journal of Theoretical Biology*, **38**, pp. 205-287.

Blum, H. and Nagel, R.N., 1978, Shape description using weighted symmetric axis features. *Pattern Recognition*, **10**, pp. 167-180.

Fritts, M.J., Crowley, W.P. and Trease, H.E., 1985, *The Free-Lagrange Method. Lecture Notes in Physics*, **238**, (Berlin: Springer-Verlag).

Gold, C.M., 1989, Surface interpolation, spatial adjacency and GIS In *Three Dimensional Applications in Geographic Information Systems*, edited by Raper, J., (London: Taylor and Francis), pp. 21-35.

Gold, C.M., 1991, Problems with handling spatial data - the Voronoï approach. *CISM Journal*, **45**, pp. 65-80.

Gold, C.M., 1997, Simple topology generation from scanned maps. In *Proceedings of Auto-Carto 13*, Seattle, Washington, (ACM/ASPRS), pp. 337-346.

Gold, C.M., 1999, Crust and anti-crust: a one-step boundary and skeleton extraction algorithm. In *Proceedings of the ACM Conference on Computational Geometry*, Miami, Florida, (ACM).

Gold, C.M., Charters, T.D. and Ramsden, J., 1977, Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. In *Computer Graphics* **11**, *Proceedings of Sigraph '77*, pp. 170-175.

Gold, C.M. and Condal, A.R., 1995, A spatial data structure integrating GIS and simulation in a marine environment. *Marine Geodesy*, **18**, pp. 213-228.

Gold, C.M., Nantel, J. and Yang, W., 1996, Outside-in: an alternative approach to forest map digitizing. *International Journal of Geographical Information Systems*, **10**, pp. 291-310.

Gold, C.M. and Snoeyink, J., in press, A one-step crust and skeleton extraction algorithm. *Algorithmica*.

Guibas, L., Mitchell, J.S.B. and Roos, T., 1991, Voronoï diagrams of moving points in the plane. In *Proceedings, 17th. International Workshop on Graph Theoretic Concepts in Computer Science*: *Lecture Notes in Computer Science*, **570**, (Berlin: Springer-Verlag), pp. 113-125.

Guibas, L. and Stolfi, J., 1985, Primitives for the manipulation of general subdivisions and the computation of Voronoï diagrams. *Transactions on Graphics*, **4**, pp. 74-124.

Lam, N. S-N., 1983, Spatial interpolation methods: a review. *American Cartographer*, **10**, pp. 129-149.

Narasihman, T.N. and Witherspoon, P.A., 1976, An integrated finite-difference method for analyzing fluid flow in porous media. *Water Resources Research*, **12**, pp. 57-64.

Okabe, A., Boots, B. and Sugihara, K., 1992, *Spatial Tessellations: Concepts and Applications of Voronoï Diagrams*, (Chichester: John Wiley and Sons).

Roos, T., 1990, Voronoï diagrams over dynamic scenes. In *Proceedings of the Second Canadian Conference on Computational Geometry*, Ottawa, pp. 209-214.

Sibson, R., 1981, A brief description of natural neighbour interpolation. In *Interpreting Multivariate Data*, edited by Barnett, V. (New York: John Wiley and Sons), pp. 21-36.

[1] This paper is based on the author's keynote speech at GISRUK 2002 in Sheffield, UK. The numbering of the headings is added by the editor.

[2] Professor Chris Gold has just accepted a European Union Marie Curie Chair, in GIS Algorithms, and will be based at the University of Glamorgan, UK.