

The Global GIS

Christopher Gold

Geomatics Department,
Laval University, Quebec City,
Quebec, Canada G1K 7P4.
Christopher.Gold@scg.ulaval.ca

Abstract

This paper outlines the concept of a “Global GIS,” and defines various aspects of its development, as well as various options and decisions that must be made. The emphasis is on the advantages and disadvantages of maintaining a global topological structure, and whether topology should be generated “on the fly” in response to a specific query. We first define what we mean by “space” in this context, followed by a description of topological structures and how we may use them in the context of graph traversal problems. We then describe some appropriate data structures. After mentioning some of the real-world problems associated with polygon construction problems, we touch on how graphs may represent change over time.

A global topological structure is then proposed which resolves some of the problems mentioned. This is based on incremental graph updating methods. Some unresolved problems associated with using global topological structures are then discussed, especially those associated with queries, such as polygon overlay, that generate a new local topological structure. We conclude with some preliminary rules, and suggest further work.

Introduction: the problem

This paper outlines some of the issues to be resolved in the development of a truly global GIS. The term ‘GIS’ is used loosely, because our concepts of GIS functionality are based largely on our experience with existing systems, and what we would really like is to be able to simulate as many properties of ‘real-world’ space as possible, within our computing systems. In addition, the term ‘global’ here implies both that the data set is very large, and that we are working on the globe itself: the sphere or ellipsoid. Within that three-dimensional framework we have the gravitational force that gives us our concept of ‘vertical’, which means that for most terrestrial applications we may reasonably restrict ourselves to the two ‘horizontal’ dimensions, in other words the surface of our globe.

What is space?

This space is, for most purposes, continuous, but with local variations in energy and matter. Where the matter is a solid, we think of objects embedded in that space; otherwise we think of fields, where some attribute of energy or matter varies more-or-less continuously over space. Where abrupt boundaries exist we interpret what we observe as due to the existence of solid objects, embedded in our space, and can reason about their form and spatial relationships with respect to other objects. In most other cases we think of 'fields' - some attribute varying over our space, with occasional observations of these attributes. In this case we need to express their spatial relationships, just as if the observations were themselves objects.

Because digital computers use discrete representations, and because people find it easier to classify information than to visualize it as continuously varying, we use various techniques to 'discretize' (classify) this information, either in the attribute domain, or in the spatial domain, or in the time domain. Contour maps, for example, discretize the attribute (elevation) information, whereas political maps discretize the spatial component into polygonal regions. Most maps discretize time, with only occasional map updates. For further discussion of these issues, see Gold (1996).

Thus most of our information about space does not merely consist of defining objects, along with some arbitrary coordinate reference, but at a fundamental level requires some concept of spatial relationship (adjacency, etc.). The use of spatial relationships (a) avoids using arbitrary coordinate systems and (b) provides the tools for local navigation within the particular spatial framework of interest. Item (a) is important because local relationships and measurements are easier to detect than imaginary global coordinates (at least, this was true before the advent of GPS). Item (b) allows the navigation from neighbour to neighbour through a mesh giving the adjacency relationships between members of a selected class of objects.

What are topological structures?

In a global GIS one basic requirement is to have a set of spatial relationships linking our objects or observations in a 'horizontal' sense - in other words, a topological structure. The key question is how to implement it, and at what level, in a 'global' system. Moreover, both in theory and in practice, these topological links must be locally modifiable. On the theoretical level, we live in a world where spatial relationships change frequently, not least because we are navigating in it. At the practical level, a global-sized data set may not have a topology that is built all at once in a 'batch' fashion - it is too big. Indeed, at that scale it is clear that time comes to play a significant role - it will never be complete or up-to-date, and therefore local updating becomes an absolute necessity.

The idea of "topology" given above is that a set of spatial relationships exist between objects that are in some sense neighbours, and this spatial relationship, for the purpose of this paper, is expressed in terms of two local dimensions only. In mathematical terms, we can express this as a graph, with the objects being represented as nodes and the spatial relationships as edges connecting pairs of objects. (See Sedgewick, 1983.) We can go

further: our nodes and edges are embedded in a two-dimensional surface. Usually this is adequate, but it does not allow for cases such as a road crossing a river, or overpasses and underpasses. As these are special cases, we will not include them in our current model. (Most GISs do likewise.) However, the Quad-edge data structure described below may be used to represent these cases if required.

If our graph is of this form, not permitting edge crossings, we refer to it as a planar graph, which has various additional properties. One property is that regions may be defined, as the portion of two-dimensional space enclosed by a closed cycle of edges. (We are usually concerned with a minimal region, where the cycle does not enclose any other nodes or edges - a single polygon, for example.) Another property is that all the edges connecting to a particular node can be given in some cyclic order - e.g. anticlockwise around the node. Finally, the number of regions, edges, or nodes may be determined from the other two entities (Euler's Formula: $\text{Regions} + \text{Nodes} - \text{Edges} = 2$, counting the exterior as a region). One example of a planar graph is a polygon map; another is a river network; yet another is a triangulation.

There are various ways of storing these various planar graphs, and much has been written about the various possible data structures used to store and manipulate them in the computer. They are all designed to allow the program to move from one entity (Region, Edge, Node) in the graph to some adjacent entity. Examples of such structures are: Polygon/Arc/Node; Doubly-Connected-Edge-List (DCEL); the Winged-Edge structure; the Quad-Edge structure; and, for specific applications, the triangulation structure. Worboys (1995) gives a good summary. One way to look at the many possibilities is the PAN-graph (Gold, 1988). Here each entity in the graph (Polygons, Arcs, Nodes: equivalent to Regions, Edges, Nodes) may be connected together in various ways, indicating the pointer structure between adjacent graph entities. De Floriani et al. (1995) use the same method. While simple, this figure can readily indicate whether one can reach another entity from a given starting place, how quickly, and with how much storage. Given some knowledge of the kind of queries expected, an appropriate data structure may be defined.

One limitation of these approaches should be noted: they work only for connected graphs. If there are "islands" in a polygon map, for example, and the arcs between the nodes in the data structure graph correspond to the polygon edges in the map, then the presence of the island within some other polygon can not be detected without further processing. This shows up a weakness of this particular data structure. An alternative solution is to work with the "dual" of the original map. Here, each polygon may be represented as a point (the "capital city" perhaps). Then pairs of capital cities of polygons with common boundaries are connected ("roads" between capital cities of adjacent polygons). These form the redefined edges of the new graph. Finally, as before, regions are defined as closed cycles of edges: this gives redefined polygons around each original node. If the map is completely covered with polygons then the resulting graph is connected. In addition, the graph is now a triangulation, except for where four or more regions meet at a point (which can be fixed by adding zero length edges), or where there are islands. Thus, as in Gold (1988), the dual structure may be used as a more convenient representation of the adjacency relationships present in the polygonal map. For non-polygonal maps (river or road networks, for example) the same approach may be used.

Traversing graphs

Within any of the data structures described above, and expressed by some form of PAN graph, we need to be able to navigate through the generated network in some way, in order to respond to various spatially-based queries. Examples of such queries include: finding an adjacent entity (Region, Edge, Node); walking along a particular path through the network to a specified destination (defined by location or object name); detecting all the objects within some zone of interest (defined by coordinates, or by some set of objects, e.g. a polygon boundary); or doing a complete traversal of the whole graph.

An example of the traversal of a whole graph can be seen in the process for the extraction of a set of polygons from a set of digitized arcs. In the usual approach (Burroughs 1986) the digitized “spaghetti” is processed to find all the intersections among all the arcs. After clean up this gives a set of nodes, with arcs defined in (anticlockwise) order around each node. (In practice this process is extremely difficult to implement because of the usually messy results of manual digitizing.) Once this stage has been reached the resulting graph needs to have the region (polygon) information added. Starting at any arc, and examining the node at one end, the next arc clockwise is found. This process is repeated for the node at the other end of this arc, and this continues until we have traversed the polygon in anticlockwise order and have returned to the starting place. The additional relationships (Polygon/Arc, Polygon/Node, and their inverses) are added to the topological structure as required. This process has used Node/Arc and Arc/Node relationships, and the appropriate side of each arc is flagged as used. The opposite (unflagged) side of one of the arcs just processed is then used as a starting point, and the next polygon defined. This continues until the whole graph is processed.

Other examples of traversing the whole graph include the classic depth-first or breadth-first searches (Sedgewick 1983), which do not require any geometric information and can be guaranteed to visit every node of a connected graph, and the “Visibility Ordering” of a triangulation (Gold and Cormack, 1987, De Floriani et al. 1991). It is also possible to “walk” through a triangulation, from some starting triangle to some destination location, by checking which triangle edges have the destination point on the “outside” edge, and then moving to that triangle and repeating the process, until the enclosing triangle is found (Gold et al., 1977).

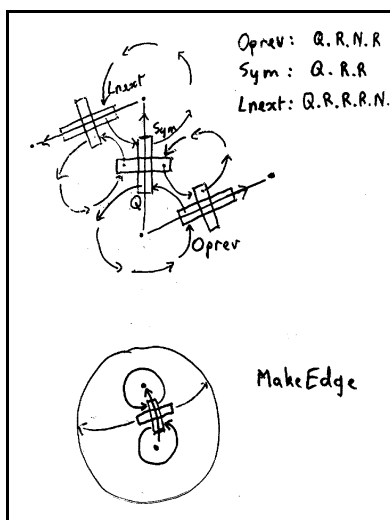


Figure 1. The Quad-edge data structure.

The Quad-edge data structure (Guibas and Stolfi, 1985) allows similar navigation, even though all navigation is from edge to edge only. Its advantages are firstly that there is no distinction between the primal and the dual representation (it is symmetric with respect to edges and faces/polygons), and secondly that all operations are performed as pointer operations only, thus giving an algebraic representation to its operations. (By contrast, operations on triangles require that the three edges of the triangle be tested to establish where the previous triangle

```

TQuad = class
  N : TQuad;           {next edge anticlockwise}
  R : TQuad;           {next 1/4 of edge}
  V : TPoint;          {vertex}
  Index : Integer;     {'name' for debugging}
end;

class function TQuad.MakeEdge(Orig, Dest : TPoint) : TQuad;
var
  Q0, Q1, Q2, Q3 : TQuad;
begin
  {create four new 1/4 edges}
  Q0 := TQuad.Create; Q1 := TQuad.Create;
  Q2 := TQuad.Create; Q3 := TQuad.Create;
  {link the four parts}
  Q0.R := Q1; Q1.R := Q2; Q2.R := Q3; Q3.R := Q0;

  {link 0 & 2 to themselves, 1 & 3 to each other}
  Q0.N := Q0; Q1.N := Q3; Q2.N := Q2; Q3.N := Q1;

  {set pointers to vertices}
  Q0.SetVertex(Orig); Q2.SetVertex(Dest); Result := Q0;
end;

procedure TQuad.Splice(A,B : TQuad); {A, B: input QuadEdges}
var
  Alpha, Beta, An, Bn, Aln, Ben : TQuad;
begin
  {get neighbouring edges: Alpha & Beta in Guibas & Stolff}
  Alpha := A.N.R; Beta := B.N.R;
  An := A.N; Bn := B.N; Aln := Alpha.N; Ben := Beta.N;

  {reconnect the four pointers}
  A.N := Bn; B.N := An; Alpha.N := Ben; Beta.N := Aln;
end;

function TQuad.Oprev : TQuad; {next edge clockwise}
begin
  Oprev := Self.R.N.R;
end;

function TQuad.Sym : TQuad; {other end}
begin
  Sym := Self.R.R;
end;

function TQuad.Lnext : TQuad; {next edge clockwise, other end}
begin
  Lnext := Self.R.R.R.N.R;
end;

function TQuad.Vertex : TPoint; {read vertex}
begin
  Result := Self.V;
end;

procedure TQuad.SetVertex(PtIn : Tpoint); {set vertex}
begin
  V := PtIn;
end;

procedure TQuad.Delete; {disconnect and free an edge}
begin
  Splice(Self, Self.Oprev);
  Splice(Self.Sym, Self.Lnext);
  Self.Free;
end;

function TQuad.Swap : Boolean; {swap a diagonal}
var
  a, b : TQuad;
begin
  Result := False;
  a := Self.Oprev; {get adjacent edges}
  b := Self.Lnext;
  if (a.Sym.Vertex <> b.Sym.Vertex) then begin
    Result := True;

    Splice(Self, a); {disconnect diagonal}
    Splice(Self.Sym, b);

    Splice(Self, a.Lnext); {re-connect diagonal}
    Splice(Self.Sym, b.Lnext);

    Self.SetVertex(a.Sym.Vertex);
    Self.Sym.SetVertex(b.Sym.Vertex); {redefine vertices}
  end;
end;

```

Table 1. Delphi code for Quad-edge.

was connected. Similar situations arise with other polygon structures.) Its disadvantage is that storage costs are high by comparison with other methods. It is, however, very easy to implement, especially in an object-oriented environment. Figure 1 shows the basic structure, with four branches for each edge of the graph being stored, one for each of the adjacent vertices or faces. Table 1 shows the object definition in Delphi (Object Pascal). Apart from a pointer to the node or face, each branch has a pointer “R” (Rot), connecting the four branches together in anticlockwise order, and a pointer “N” (Next), pointing to the next edge in anticlockwise order around the node or face. Also shown are the low-level operations using N and R: “Sym,” which moves from the current branch to the branch of the same edge that faces in the other direction; “Oprev” which moves to the next branch/edge clockwise around the vertex/face associated with the original branch; and “Lnext” (which equals Sym.Oprev) that finds the next branch/edge clockwise around the other end of the edge.

To show the simplicity of its use, Figure 2 shows the two commands that are used to modify a graph: “Make-edge” to create a new edge on a manifold, and “Splice” to connect/disconnect Quad-edges together. In the simplest case, Splice connects two separate “Next” loops, joining the two nodes together, and at the same time splitting the “Next” loop around the common face. (The process could equally well have split the “Next” loop around a node, and merged the loops around a face: Splice is its own inverse. See the original paper for more details.) Table 1 gives the code for these operations, as well as for three simple example operations using Make-edge and Splice: deleting an edge from a graph, inserting a point in a triangle, and switching the diagonal of a pair of triangles. These operations are sufficient to maintain a triangulation network for simple terrain modelling. Work is continuing on making these operations efficient in secondary (rather than just primary) storage.

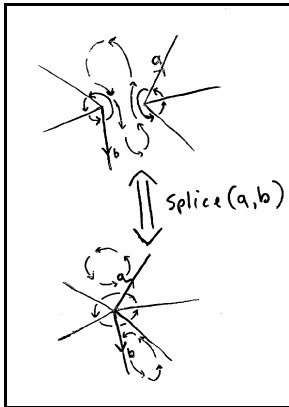


Figure 2. The Splice operation.

Real-world problems with polygon construction

In reality, of course, topology construction in a traditional GIS is a complex operation. It normally consists of a global operation to find all line intersections between the input arcs, followed by various stages of cleanup: removing small dangling arc ends, merging nearby intersections into a single polygon node (see Pullar, 1994), etc. Only then may the resulting graph be traversed to create polygons. Additional problems are the detection of self-intersecting arcs, the generation of unwanted “sliver polygons” if arcs are nearly superimposed, and unconnected islands as described above. (In a traditional GIS, “topology” is largely thought of as a polygon construction problem.)

The nature of the intersection-detection operation indicates a global process rather than an incremental one, and this leads to the necessity of processing the whole “map” at once. Thus most systems are map-sheet based, allowing the processing of a manageable set of data at one time - even if only one arc has been changed from the previous time. This clearly is not appropriate for a global GIS, which would need an incremental algorithm, adding one arc at a time. (It should be noted that if the extent of an update can be well defined, then a “patch” may be created and sewn into the map sheet, thus speeding up the process. All the underlying problems remain, and it is not obvious how to define the update regions.) In addition, since features (usually arcs) cross map-sheet boundaries, a great deal of effort is required to reconnect features when multiple map sheets are needed. Life would be much easier with an incremental, global topology system!

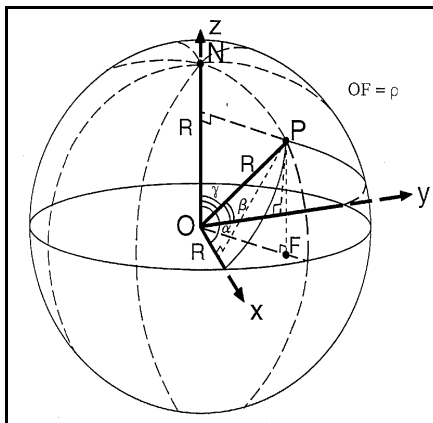


Figure 3. Direction Cosines.

Defining map sheets on the basis of global map coordinates alone ignores the consistency of objects, and in addition breaks the topological connectedness of the global system. A secondary, although relatively minor, problem concerns the coordinate system itself: it must be global (not broken into zones), and it must be continuous throughout (unlike latitude/longitude). Using direction cosines (a vector algebra approach), a seamless coordinate system, simplifies things greatly (Figure 3). We are storing three coordinates for a point instead of two, but there are no discontinuities in the coordinate system (as is the case for longitude at plus or minus 180 degrees, or across UTM zones or the equivalent). Good precision is preserved over the whole globe.

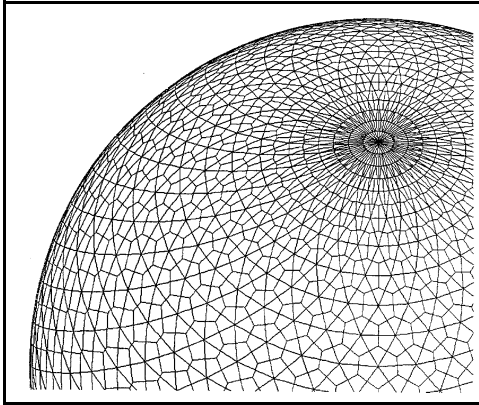


Figure 4. Regular Voronoi cells and Delaunay triangles on part of the sphere.

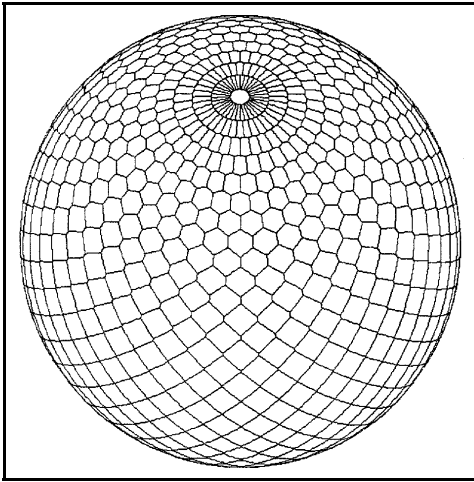


Figure 5. Voronoi cells on the sphere.

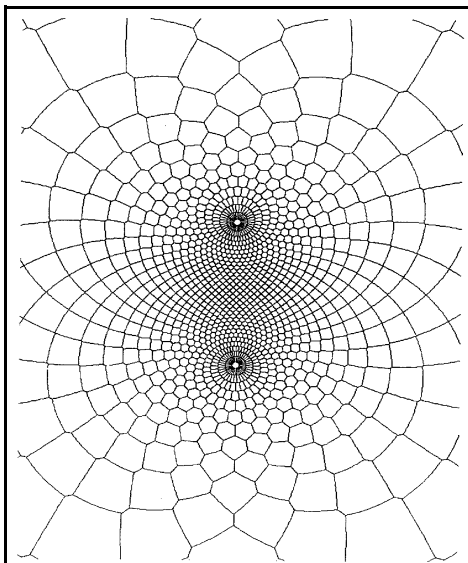


Figure 6. Equatorial stereographic projection of Figure 5.

Time

GISs as we know them have great difficulty in handling change over time, due largely to the map-sheet approach to topology building. It is not easy to add or remove one feature at a time during the update process. Thus time is usually treated as a series of “snapshots,” perhaps at ten-year intervals. What is needed for updating is an incremental (and interactive) approach. The problem here arises out of the complexities in the geometry operations, as the incremental update of a planar graph is not a problem.

Outside the traditional polygon map, there are many other applications whose topological structures deserve consideration. Networks (roads, rivers) possess a topological structure less complete than a polygon set; in addition, closed loops may not represent regions that need to be named, and unconnected nodes at the ends of arcs are permitted. Point data sets initially may be thought of as having no topological structure, but they are very often connected into a triangulation. These structures ought to be updateable on an incremental basis also.

Spatial operations that fall outside traditional GIS need to be considered also; in particular simulation of one form or another where the objects move in space and need dynamic update of their topology. This is a direct continuation of the update problem mentioned above, except that modifications are made as a result of the simulation of the process involved (the “future”), as opposed to the manual updating of the known “past.” A simple example would be atmospheric modelling, where particles representing packets of air would move around and interact with each other. This can be managed with the triangulation structure - in particular the Delaunay triangulation, which is the dual of the simple Voronoi diagram. This provides simple rules for determining the neighbouring points to any particular point, and how to update the topological structure as the points move (Roos, 1991).

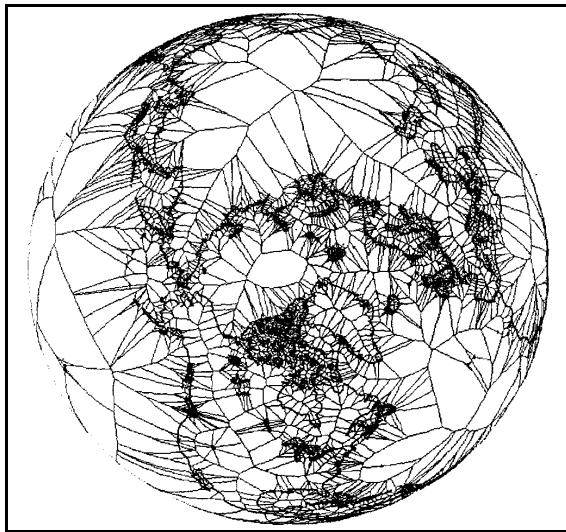


Figure 7. Polar orthographic projection of the continents.

A Global Topological Structure

This example of atmospheric modelling naturally brings us to the question of moving our topological structure onto the sphere (or ellipsoid). The transformation of a simple planar graph is very easy: take the “exterior” region, and consider this to be a finite region on the opposite side of the sphere to the viewpoint. For a triangulation, all exterior vertices may be connected to a point on the opposite side of the sphere. This simplifies management of the boundaries of the map, and provides, as Worboys (1995) calls it, the “perfect 2-space”.

Thus all displayed maps are considered to be projections of the globe - which is quite reasonable! Triangulations and Voronoi diagrams may be displayed (Figures 4 to 7). Dynamic point movement may be managed as described above. The topological structure allows the navigation through the network to locate the enclosing triangle of, or the nearest object to, some specified location. Local updates may then be performed to insert or delete an object at that location.

Topology vs. attributes

Following topology construction issues, but related to them, are concerns about spatial queries. Within a map-sheet-based system simple queries are global for the map sheet in most cases: finding all objects (houses, polygons, etc.) with the desired attributes. This is implicit in a relational data base system: a particular table of attributes will be for the whole map. This is not feasible in a true global system: queries must be restricted first by some spatial component, and only then by attribute. The traditional approach would be by some map-sheet structure as described above, with all its complications.

Another alternative would be to use the global topological structure itself: first finding any object within the region of interest, and then expanding outwards to collect everything within that region. The first stage may be performed by walking through the network as described above. For successive queries this walk will usually be relatively short, as work continues on the region of interest. Nevertheless, this will not always be the case, and some pointer structure may be needed to avoid traversing half the globe to find some initial object within the new region of interest (see Gold, Remmele and Roos, 1995). The second stage involves a simple graph traversal algorithm such as breadth-first-search, terminating at the region’s boundaries, in order to examine all the objects of interest.

Given the premise that a global GIS contains many objects, and that the region of interest is small, this initial step reduces the number of objects examined to a reasonable set. Standard relational queries may then be performed on this reduced set. At present this is not possible within a traditional RDBMS, but ongoing development of object data bases may help in the future.

The introduction of time into the picture complicates things further. If it is a question of querying the database at some particular point in its updating process, rather than after the completion of some “snapshot” of the whole system as mentioned above, the approach just described remains unchanged. Queries that are constrained in space as well as time, and examine any of the past states of the region, remain a research topic. However, if we consider that the number of time states remains small within a local region, then the above approach may be preserved and the lifespan of the objects may be treated as attributes. This, however, will not preserve the topological structuring at some past time: questions such as “When were these two objects adjacent?” can not be handled.

The Overlay Problem

Other questions arise when the traditional tools of polygon overlay and buffer zones are required. These are normally batch operations performed on the whole topological unit: usually the map sheet. In a global system the objects within the region of interest must be extracted by queries, as described above, and then the new topological structure created for that region. This last may be a batch operation, or based on local incremental topology modification. Nevertheless, the effect is to produce a new topological space (or structure) for the region of interest, for a particular query. Thus the concept of global topological spaces becomes less clear.

The Classification Problem

This brings us to the key question of this paper: to what extent are global topological structures appropriate in an ideal system? For our purposes, an ideal system would be organized on the globe (the sphere or ellipsoid), possess very large numbers of map objects, have some topological structure connecting these objects, and be capable of handling frequent small, local map updates. Due to the difficulties encountered in rejoining adjacent separate map sheets, it would be preferable to avoid this problem by having a seamless topology and coordinate system. (For the purpose of this exercise the problems of paging this data structure into secondary storage will be ignored, as being a separate problem.)

When is a Global Topological Space Useful?

The primary question concerns the number of basic topological spaces (overlays) that are required for the project under consideration. Different overlays are used to separate different classes of objects: objects of the same class will be treated in the same way and, presumably, two objects of the same class can not occupy the same spatial location. Objects (or fields) of different classes can do so, and may therefore be combined in the usual overlay operation to produce new overlays. However, there is an important consideration:

does the operation of classification of a set of objects in a single overlay require that the resulting classes be stored in multiple overlays? At one level they need not: they remain spatially non-overlapping after being reclassified. (Think of the reclassification of a choropleth map.) On another level they may: adjacency queries may be between buildings of the same type (such as police stations), rather than involving intervening buildings of another type. This issue may be resolved in practice by using more complex operations on the topological space than that of the basic adjacency properties of the simple graph. In our police station example the stations are not adjacent in terms of the buildings layer, but they are accessible through that layer and the adjacency query may be resolved analytically.

Thus it appears that the answer to our primary question depends on how much local topology construction is required for queries (overlay, etc.). This in turn depends on the classification of object classes. If there are many types of objects, then the question becomes: do we have a separate global topology for each class? If yes, we can have adjacency type queries within each class, but local batch construction for inter-class queries. (Overlay is an inter-class query, buffer-zone is not.) If we need to classify our objects on the fly, (e.g. polygon reclassify and merge) then it is not obvious whether the results of this operation are good for local adjacency queries only, or need to be subdivided into their own topological spaces. At first sight it appears that, since the results can not be superimposed, they can be processed in a single topological space, as with our tentative approach to the police station problem above. (One basic rule for topological spaces is that objects in that space may not occupy the same locations: if that can possibly occur, then separate spaces are required.) Thus the primary question is: how many topological spaces are required? Can we make general rules, or do we need to perform semantic analysis for each specific application?

Summary

Within a single topological space a global system is appropriate. Where multiple topological spaces are required, then comparisons between them (often overlay operations) generate further spaces. These would be small-scale, one-off local operations, and would not need to be preserved beyond their use as a response to a query. This seems to be reasonably good for traditional polygonal maps, which are one representation of field-type data. Other fields would be continuous, with individual observations, and the single topological space would seem appropriate. For discrete objects a lot depends on the assumption of two-dimensionality. Trees are probably not on houses, but cars are usually on roads. In both cases however there will be the need to extract subsets of the operations in a single topological space in order to answer specific local queries: buffer zones around primary roads only, for example, or the distances between pubs or police stations. These are again one-off operations for specific queries.

The function of a global topological space therefore is to contain the objects of a particular (spatially non-overlapping) category. For simple cases, such as simulating atmospheric movement, this should be enough. In other cases several spaces should be maintained, and local topological structures generated for specific queries. The alternative is to subdivide the globe into map-sheets, creating many complications.

The starting point for any local operation on a topological space in traditional GIS is to find a map sheet containing some initial location. If a global structure is used, the first step is to find the closest object to the initial location, and then perform an outwards-search to cover the region of interest. This may be done by a walk through the structure (which could be long), or else by adding a simple hierarchical structure to give a close initial estimate. Having obtained a spatially restricted region, local topological spaces may be generated by overlay or other operations in order to respond to particular queries. A major technical question concerns the storage and retrieval of attribute information: where there are no permanent regions (map sheets) it is not obvious how to adapt to RDBMS-type structures.

The arguments given above would be appropriate for any appropriate topological structure that can be represented as a graph, possibly stored as Quad-edges, and can be updated using local operations. Nevertheless, the ideas were organized with the Voronoi structure in mind (for example Gold and Condal, 1995), as it has been shown to be maintainable by local update, and can readily be generated on the sphere. Thus the move towards a global GIS would require a locally-updateable topological structure, perhaps of that type. It would be necessary to maintain a topological space for each base class of objects/fields, defined on the basis that duplicate objects may not exist in the same space at the same location. Topological queries between classes, such as overlay, would require the generation of a temporary topological structure over the region of interest. For some single-space operations, such as flow simulation or interpolation, this would not be necessary.

Where temporary topological structures are required, it is less clear whether a global topological structure is necessary. On the positive side, it eliminates the worries over local coordinate systems and map edge-matching. On the negative side, maintenance might be excessive if most queries required local temporary topological structures. This would reduce the value of the global structure to data maintenance and extraction of the region of interest. Nevertheless, their benefits for the modelling of simple data types warrant experimentation with the approach for those applications, with extension to complex data types and queries if warranted at a later date.

Acknowledgments

This research was supported in part by an NSERC/AIFQ Industrial Research Chair in Geomatics. The author is also very grateful to Mr. Mir Mostafavi for providing examples, from his own research work, of Voronoi diagrams on the sphere.

References

Burroughs, P., 1986. Principles of Geographical Information Systems for Land Resources Assessment. Oxford University Press.

De Florian, L., B. Falcidieno, G. Nagy and C. Pienovi, 1991. On sorting triangles in a Delaunay tessellation. *Algorithmica*, v. 6, pp. 522-532.

De Floriani, F., E. Puppo, P. Magilla, 1995. Technical aspects of spatial data. IN: Andrew Frank (ed.), Geographical Information Systems - Materials for a Post-Graduate Course - Vol. 2: GIS Technology. Department of Geoinformation, Technical University of Vienna.

Gold, C.M., T.D. Charters and J. Ramsden, 1977. Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. *Computer Graphics*, v. 11, June 1977, pp. 170-175.

Gold, C.M., 1988. PAN Graphs: an aid to GIS analysis. *International Journal of Geographical Information Systems*, v. 2, pp. 29-42.

Gold, C.M., 1996. An event-driven approach to spatio-temporal mapping. *Geomatica*, v. 50, pp. 415-424.

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

Gold, C.M. and S. Cormack, 1987. Spatially ordered networks and topographic reconstructions. *International Journal of Geographical Information Systems*, v. 1, pp. 137-148.

Gold, C.M., P. Remmele and T. Roos, 1996. Fully dynamic and kinematic Voronoi diagrams in GIS. *Algorithmica* (in press).

Guibas, L. and J. Stolfi, 1985. Primitives for the manipulation of general subdivisions, and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, v. 4, pp. 74-123.

Pullar, D., 1994. A Tractable Approach to Map Overlay. Ph.D. Dissertation, University of Maine at Orono, 134 p.

Roos, T., 1991. Dynamic Voronoi diagrams. Ph.D. Dissertation, University of Würzburg, 213 p.

Sedgewick, R., 1983. *Algorithms*. Addison-Wesley, Reading, Mass., 551 p.

Van Oosterom, P., 1993. *Reactive Data Structures for Geographic Information Systems*. Oxford University Press.

Worboys, M.F., 1995. *GIS - A Computing Perspective*. Taylor and Francis Ltd., London, 376 p.