Minimal Spatial Representations

W. Randolph Franklin

ECSE Dept, 6026 JEC, Rensselaer Polytechnic Institute, 110 8th St, Troy NY 12180 USA
Email: mail@wrfranklin.org, Web: http://wrfranklin.org/

Abstract

This paper argues for representing geometric objects like polygons and planar graphs (maps) using no global topology, and using only the minimal local topology. Good representations include the set of directed labeled edges, and the set of vertex-edge incidences. Operations such as unions and intersections are easier to implement, are very fast, and parallelize.

1. Introduction

Our question is how to represent 2D shapes such as polygons and embedded planar graphs (aka maps) in GIS. Specifically, what components should be explicit, and what relations between components should be explicit? This is related to the question of what is the most useful definition of a geometric object, such as a polygon. This topic is superficially related to other studies of topology in GIS. For simplicity, we will mostly discuss 2D, although the ideas extend to 3D.

2. Preliminary Definitions and Operations

Consider a polygon, P, in the 2D plane. Intuitively, it has vertices and edges. If we restrict the number of each to be finite, then there will be no problem with infinite sequences, limits and convergence. The mathematical subject of analysis (used by calculus) is irrelevant. However some interesting objects are now prohibited. E.g., we cannot represent a curved object as the limit of a sequence of ever more complex straight-edges objects.

An edge is a connected segment of a straight line. That is, it cannot have two separate parts. Do we want to allow it to be infinite? There are arguments both ways. All Voronoi diagrams have some infinite edges. However, this makes representing edge lengths and polygon areas trickier, since they can be infinite also.

An edge has a vertex at each end, aka each edge is adjacent to two vertices. If we choose to prohibit isolated vertices, then a vertex is adjacent to one or more edges.

Two edges with a common vertex intersect at that vertex. Do we want to allow two edges to intersect elsewhere? Usually not, but allowing that permits star polygons and four new regular, stellated, polyhedra.

The vertices and edges form the *boundary* of P, ∂P. (The most accessible source for the mathematical terms in this paper is Mathworld [Wei16]). P also has an interior, but on which side of ∂P? The conventional answer is the small finite part, not the large infinite part, but sometimes we want the other one.

Next, what operations do we want to perform on legal polygons? The obvious operations on polygons P and Q are *union*, $(P + Q)$, *intersection*, $(P \cdot Q)$, and *exclusive-or*, $(P \oplus Q)$. The *difference* operation, *A-and-not-B*, $(P - Q)$, is a common extra operation. Electrical engineers like the fact that all the other operations can be defined in terms of the difference, and so fewer types of electrical components are needed in circuit design. Allowing the unary operation of *complement*, P' or $\bar{P}$, opens up some possibilities. It is

theoretically desirable to require that the set of all possible polygons be *closed* under these operations. That means that whenever these operations are applied to one or two legal polygons, then the result is always a legal polygon. This implies that: (1) The *empty set*, $\emptyset$, and its complement, the *universal set*, must be legal polygons. Indeed, $\emptyset$ results from intersecting two disjoint polygons. (2) A legal polygon may have multiple separate disjoint components, which may even be nested, like an island in a lake on the mainland.

A broader advantage of having the set of legal polygons closed under union or intersection, and the empty and universal sets being legal polygons, (and since those operations are both associative), is that the set of polygons, together with either operation, forms a mathematical *commutative group* in abstract algebra. Now, all the many theorems that have been proved about groups also apply to polygons under intersection or union.

3. Global Topology of a Polygon

When a polygon's edges form more than one *connected component*, then each component of the edges and vertices is called a *loop*. The loops' containment relations form a forest of trees. E.g., consider the polygon that is the land area of Canada. It has a loop for the mainland shoreline, a second loop for Baffin Island's shore, a third loop for Lake Erie's shore, a fourth loop for Pelee Island in Lake Erie, etc. The fourth loop is inside the third loop, which is inside the first. The second loop is inside none of them, nor are any of them inside it. Conventionally, we would represent these containment relations explicitly. This paper proposes that that is often unnecessary.

4. Local Topology of a Polygon

What relations ought to be stored explicitly depends on what we intend to do. For the above Boolean operations, viz, union, intersection, difference, and complement, a mere *set of oriented edges* suffices. Start by observing that this representation suffices to determine whether a point p is contained in the polygon P. Extend semi-infinite ray up from p and count intersections. To handle geometric degeneracies, such as when the ray runs through a vertex, we recommend Simulation of Simplicity (SoS) [EM90], used with success in, e.g., [MAFL16]. To compute $P + Q$, observe that

1. The output vertices are a subset of the input vertices and all the intersections of input edges. Those input vertices that are output vertices are exactly those that are outside the other polygon.
2. To find the output edges, cut the input edges where they intersect each other. Select those pieces of each polygon that are outside the other polygon.

Both the input and output are a set of edges, together with the vertices at their ends. No more global topology is needed.

Indeed, not even the complete edges are always needed. Suppose that our desired operation is to compute properties of the polygon such as area and edge length. Then, representing the polygon as a set of the *vertex-adjacencies* suffices. For each such adjacency, store this triple: the vertex's position, the direction that the edge leaves the vertex, and which side of the edge is the inside of the polygon. Each vertex and each edge induces two such triples, but they are not explicitly associated in this representation. This representation is so local that it does not even have complete vertices or edges, although that information could be computed. More details, including the formulae, are in [NF91, Fra04].

To intersect two polygons represented by sets of their edges, computing the above vertex-edge incidence representation is easier than computing the output edges, since we do not need to sort the intersections along each edge to cut it into segments. This provides an efficiency if we want the area of the union or intersection, but not the union or intersection itself.

5. Unions of Multiple Polygons

The above representations also greatly simplify the problem of finding the area of the union of many, perhaps millions of, polygons. The usual algorithm to unite N polygons is to start by uniting pairs, then to unite the pair polygons, and so on, building up a computation tree of height $\Theta(\lg N)$. With the above representation, the output can be computed as follows:

1. Find all the intersections of any two input edges.
2. Test all those points, together with all the input vertices, to see which are outside all the input polygons.
3. Each point that passes is an output vertex. Determine the two resulting vertex-edge incidences, and apply the area formula.

For a wide distribution of inputs, the expected time is *linear* in the number of input edges.

6. Planar Graphs (Maps)

The above ideas are even more useful when operating on embedded planar graphs, aka maps or meshes. The notation is not standardized, so we will use the terms *vertices*, also known as nodes and points, *edges*, and *polygons* aka faces. Typical operations include determining which polygon contains a test point, and intersecting, aka overlaying, two maps to produce a new map.

The easy way to locate which polygon contains point p is to test p against every polygon in turn. However, with this representation, we can run a ray up from p until it hits the first edge, and from that know which polygon contains p. The expected time is *constant* per query point, after expected linear processing time. We have implemented this in 3D [MAFL16].

Overlaying two maps is also easy with this representation; see [MAFL15]. The expected time, as usual, is linear in the input maps' sizes.

7. Parallel Algorithms and Functional Programming

The recent availability of consumer-grade massively parallel GPUs has fostered a theoretical interest in representations amenable to parallel algorithms. E.g., in May 2016, Nvidia introduced the GeForce GTX 1080 card listing at USD$600, with 3840 CUDA cores, which can process floating point numbers at 1 TFLOPS. Simple flat representations parallelize much better than complicated tree-based representations. We have demonstrated this in [MAFL15, MAFL16], often seeing speedups of $10\times$.

The representations presented above also compatible with functional programming. This has various theoretical advantages, such as allowing languages like Haskell.

8. Summary and Future Plans

Less is better (up to a point) has many advantages when representing GIS geometry. We must fight the urge to continually discover new topological relations. These ideas extend to

3D and are useful in other geometric domains, like Computer Aided Design and Computational Fluid Dynamics. We are currently demonstrating that with new implementations in those domains.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1117277.

References

[EM90] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM T. Graphics*, 9(1):66–104, January 1990.

[Fra04] W. Randolph Franklin. Analysis of mass properties of the union of millions of polyhedra. In M. L. Lucian and M. Neamtu, editors, *Geometric Modeling and Computing: Seattle 2003*, pages 189–202. Nashboro Press, Brentwood TN, 2004.

[MAFL15] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. Fast exact parallel map overlay using a two-level uniform grid. In *4th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial)*, Bellevue WA USA, 3 Nov 2015.

[MAFL16] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. PinMesh – Fast and exact 3D point location queries using a uniform grid. *Computer & Graphics Journal, special issue on Shape Modeling International 2016*, 2016. (to appear).

[NF91] Chandrasekhar Narayanaswami and Wm Randolph Franklin. Determination of mass properties of polygonal CSG objects in parallel. In Joshua Turner, editor, *Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 279–288. ACM/SIGGRAPH, 5–7 June 1991.

[Wei16] Eric W. Weisstein. MathWorld–a Wolfram web resource. `http://mathworld.wolfram.com/`, retrieved 11 May 2016.