Reversibility of the Quad-Edge operations in the Voronoi data structure

Darka Mioc
Department of Geodesy and Geomatics Engineering
University of New Brunswick,
P.O. Box 4400, Fredericton, New Brunswick, Canada E3B 5A3
dmioc@unb.ca

François Anton
Informatics and Mathematical Modelling
Technical University of Denmark
Building 321, 2800 Kgs. Lyngby, Denmark

Christopher M. Gold
University of Glamorgan
Pontypridd, Wales, UK, CF37 1DL
ChristopherGold@Voronoi.Com

Bernard Moulin
Département d'Informatique, Université Laval,
Pavillon Pouliot, Ste Foy, QC G1K 7P4, Canada
moulin@ift.ulaval.ca

Abstract

In Geographic Information Systems the reversibility of map update operations have not been explored yet. In this paper we are using the Voronoi based Quad-edge data structure to define reversible map update operations. The reversibility of the map operations have been formalised at the lowest level, as the basic algorithms for addition, deletion and moving of spatial objects. Having developed reversible map operations on the lowest level, we were able to maintain reversibility of the map updates at higher level as well. The reversibility in GIS can be used for efficient implementation of rollback mechanisms and dynamic map visualisations.

1 Introduction

Reversible computing, in a general sense, means computing using reversible operations, that is, operations that can be easily and exactly reversed, or undone. Furthermore, when reversibility is maintained at the highest levels, in computer architectures, programming languages, and algorithms, it provides opportunities for interesting applications such as bi-directional debuggers, rollback mechanisms for speculative executions in parallel and distributed systems, and error and intrusion detection techniques. The reversibility can be maintained at the lowest level, in the phys-

ical mechanisms of operation of bit-devices and it could be maintained at higher levels as well. Finally, the two types of reversibility (low-level and high-level) are deeply connected, because, as it turns out, achieving the maximum possible computational performance generally requires explicit reversibility not only at the lowest level, but at all levels of computing–in devices, circuits, architectures, languages, and algorithms. In GIS, the reversibility has not been explored sufficiently yet. The reversibility in GIS can be used for efficient implementation of rollback mechanisms and dynamic animations needed in spatial analysis [9].

2 Quad-Edge based Voronoi data structure

The Voronoi diagram for a set of map objects (points and line segments) is the tessellation of space where each map object is assigned an influence zone (or Voronoi region), that is the set of points closer to that object than to any other object (see [6] and Figure 1).

The algorithm used to construct the Voronoi vertices has been described in [1]. The boundaries between the regions of this tessellation form a net (the Voronoi diagram), whose dual graph (the Delaunay triangulation) stores the spatial adjacency (topology) relationships among objects. Within such a dynamic Voronoi spatial data structure, as developed by Gold [3], the map objects (points and/or line segments) are stored as nodes of the dual spatial adjacency (topology)
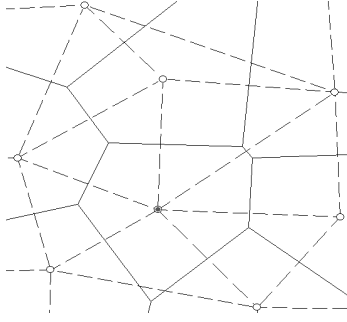
Figure 1. A Voronoi diagram

graph: the Delaunay triangulation. The underlying data structure used is the Quad-Edge data structure [7] (see Figure 2).
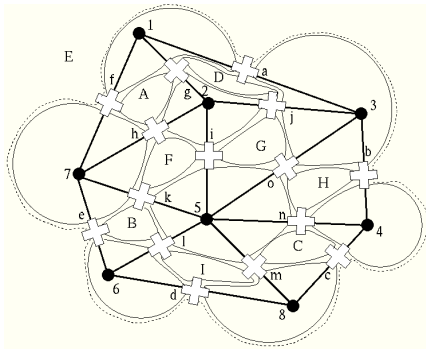

Figure 2. The Quad-Edge data structure for the Voronoi diagram of Figure 1

The Quad-Edge data structure was used for computing the line Voronoi diagram [6], which is the basis of the dynamic Voronoi data structure for points and line segments. The Quad-Edge data structure was introduced by [7] as a primitive topological structure for the representation of any subdivision on a two-dimensional manifold. The Quad-Edge data structure is the implementation of an edge algebra [7], which is the mathematical structure that defines the topology of any pair of dual subdivisions on a two-dimensional manifold. In the context of the application of the Quad-Edge data structure to the computation of Voronoi diagrams, both a primal planar graph (the Voronoi diagram) and its dual graph (the Delaunay triangulation) are stored in the Quad-Edge data structure - see [7].

Guibas and Stolfi [7] developed a convenient mathematical structure for representing the topological relationships among edges of a pair of dual subdivisions on a two-dimensional manifold[1]. A subdivision of a manifold M is

---

[1]A two-dimensional manifold is a topological space with the property

[7] a partition S of M into three finite collections of disjoint parts, the vertices (denoted by $\mathcal{VS}$), the edges (denoted by $\mathcal{ES}$) and the faces (denoted by $\mathcal{FS}$) with the following properties:

- Every vertex is a point of M,

- Every edge is a line of M,

- Every face is a disk of M,

- The boundary of every face is a closed path of edges and vertices.

A directed edge of a subdivision P is an edge of P together with a direction along it (see page 80 in [7]). Since directions and orientations can be chosen independently, for every edge of a subdivision there are four directed, oriented edges [7]. For any oriented directed edge e we can define unambiguously its vertex of *origin* $e\,Org$, its *destination,* $e\,Dest$, its *left face,* $e\,Left$, and its *right face,* $e\,Right$. The flipped version $e\,Flip$ of an edge e is the same unoriented edge taken with *opposite orientation* and same direction. The *symmetric* of e, $e\,Sym$ corresponds to the same undirected edge with the *opposite direction* but the same orientation as e.

Edge functions (see Figure 3) allow the traversal of the pair of dual subdivisions. The *next edge with the same origin,* $e\,Onext$ is defined as the one immediately following e (counterclockwise) in the ring of edges out of the origin of e (see Figure 3). The *next counterclockwise edge with the same left face,* denoted by $e\,Lnext$, is defined as the first edge we encounter after e when moving along the boundary of the face $F = e\,Left$ in the counterclockwise sense as determined by the orientation of F.

As shown in the top part of Figure 4, each branch of the Quad-Edge is part of a loop around a Delaunay vertex/Voronoi face, or around a Delaunay triangle/Voronoi vertex. The lower part of Figure 4 shows the corresponding Delaunay/Voronoi structure, where (a,b,c) are Quad-Edges, and (1,2,3) are Delaunay vertices.

2.1 Delaunay/Voronoi Quad edge equivalence

Two subdivisions S and S^* are said to be the *dual* [7] of each other if for every directed and oriented edge e of either subdivision there is another edge $e\,Dual$ (that is defined as the dual of e and isolated by parenthesis in the following expressions) of the other subdivision such that:

- the dual of $e\,Dual$ is e: $(e\,Dual)\,Dual = e$,

- the dual of the symmetric of e is the symmetric of $e\,Dual$: $e\,Sym\,Dual = (e\,Dual)\,Sym$,

---

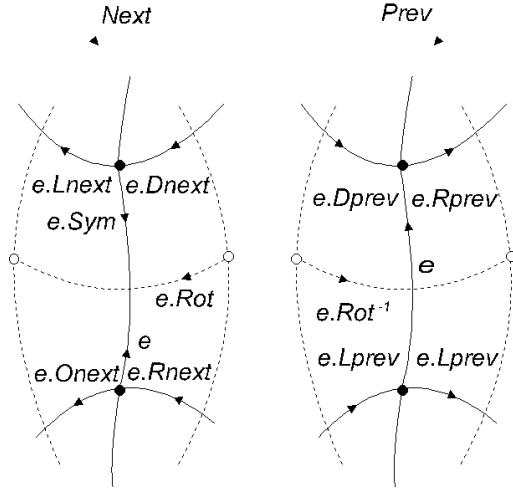that every point has an open neighbourhood which is a disk.

Figure 3. The edge functions (adapted from Guibas and Stolfi [7])

Figure 4. A simple Voronoi diagram and its corresponding Quad-Edge

- the dual of the flipped version of e is the symmetric of the flipped version of $e\,Dual$: $e\,Flip\,Dual = (e\,Dual)\,Flip\,Sym$,

- moving counterclockwise around the left face of e in one subdivision is the same as moving clockwise around the origin of $e\,Dual$ in the other subdivision: $e\,Lnext\,Dual = (e\,Dual)\,Onext^{-1}$.

The dual of an edge e is the edge of the dual subdivision that goes from the (vertex corresponding to the) left face of e to the (vertex corresponding to the) right face of e but taken with orientation opposite to that of e. The definition of the dual of an edge allows to define the operation Rot: the rotated version of an edge e is the dual of e directed from $e\,Right$ to $e\,Left$ and oriented so that moving counterclockwise around the right face of e corresponds to moving counterclockwise around the origin of $e\,Rot$. More concisely, $e\,Rot = e\,Dual\,Flip\,Sym = e\,Flip\,Dual$.

2.2 Edge algebra

An Edge algebra is the mathematical structure used for representing simultaneously a pair of dual subdivisions [7] (in our use of the Quad-Edge data structure, the Delaunay triangulation and the Voronoi diagram). It captures all the topological properties of a subdivision [7]. The topology of the subdivision is completely determined by its edge algebra, and vice versa. This allows all the edge functions to be expressed using three basic primitives, $Flip$, Rot, and $Onext$ described above [7]. An edge algebra is [7] an abstract algebra $(E, E*, Onext, Flip, Rot)$ where E and E^*

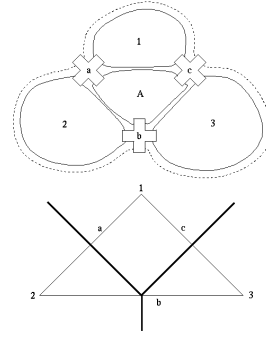are arbitrary finite sets (of edges), and $Onext$, Rot, and $Flip$ are functions on E and E^* satisfying the following properties:

- $e\,Rot^4 = e$;

- $e\,Rot\,Onext\,Rot\,Onext = e$;

- $e\,Rot^2 \neq e$;

- $e \in \mathcal{ES} \Leftrightarrow e\,Rot \in \mathcal{ES}^*$;

- $e \in \mathcal{ES} \Leftrightarrow e\,Onext \in \mathcal{ES}$;

- $e\,Flip^2 = e$;

- $e\,Flip\,Onext\,Flip\,Onext = e$;

- $e\,Flip\,Onext^n \neq e$ for any n;

- $e\,Flip\,Rot\,Flip\,Rot = e$;

- $e \in \mathcal{ES} \Leftrightarrow e\,Flip \in \mathcal{ES}$.

The Quad-Edge traversal operations are based on the edge algebra $(E, E*, Onext, Flip, Rot)$, and their expression as composition of the basic primitives [7]. $Onext$, $Flip$, and Rot will be presented in the following table. Equivalent definitions separated by $=$ signs have been presented sometimes for some operations (in the left column). Similarly, equivalent decompositions separated by $=$ signs have been presented sometimes for some operations (in the right column).

2.3 Basic topological operations in the Quad-edge data structure

The main advantage of the Quad-Edge data structure is that all the construction and modification of planar graphs

Quad-Edge Operation	Decomposition using Edge Algebra
$e\,Dual$	$e\,Flip\,Rot$
$e\,Dual^{-1}$	$e\,Flip\,Rot = e\,Rot^3\,Flip$
$e\,Sym = e\,Sym^{-1}$	$e\,Rot^2$
$e\,Flip^{-1}$	$e\,Flip$
$e\,Rot^{-1}$	$e\,Rot^3$
$e\,Onext^{-1} = e\,Oprev$	$e\,Rot\,Onext\,Rot \quad = \quad e\,Flip\,Onext\,Flip$
$e\,Lnext$	$e\,Rot^{-1}\,Onext\,Rot \quad = \quad e\,Rot^3\,Onext\,Rot$
$e\,Rnext$	$e\,Rot\,Onext\,Rot^{-1} \quad = \quad e\,Rot\,Onext\,Rot^3$
$e\,Dnext$	$e\,Rot^2\,Onext\,Rot^2$
$e\,Lprev = e\,Lnext^{-1}$	$e\,Onext\,Rot^2$
$e\,Rprev = e\,Rnext^{-1}$	$e\,Rot^2\,Onext$
$e\,Dprev = e\,Dnext^{-1}$	$e\,Rot^{-1}\,Onext\,Rot^{-1} \quad = \quad e\,Rot^3\,Onext\,Rot^3$

Table 1. Quad-Edge traversal operations

Operation	Description
e := MakeEdge[]	Creates an edge e to a newly created data structure representing an empty manifold
Splice[a,b]	Joins or separates the two edge rings a Org and b Org, and independently, the two dual edge rings a Left and b Left (see Figure 5)

Table 2. Basic Quad-Edge topological operators

Operation	Description
e := Connect[a,b]	Adds a new edge e connecting the destination of a to the origin of b, in such a way that a Left = e Left = b Left
DeleteEdge[e]	Disconnects the edge e from the rest of the data structure
Swap[e]	Rectifies e in order to respect the empty circumcircle criterion

Table 3. Complex Quad-Edge topological operators

can be done using two basic topological operators (see Table 2), and the complex topological operations built from these two basic topological operators. These complex topological operations are presented in the Table 3. The two basic operators modify the graph locally. Locality of the Quad-Edge operations will be studied in detail in the next subsection.

2.4 Locality of the Quad-Edge data structure

In vector based GIS systems, the maintenance of topology is performed through batch operations, that are global, i.e. by alterating all the objects in the map. In contrast, within the Voronoi spatial data structure, the topology is maintained locally when objects are added and deleted. Indeed, only the neighbours of the object being added/removed may be altered by the topology maintenance. In this section, we will see why the operations on the Quad-Edge data structure have a local scope. In order to prove the locality of the Quad-Edge data structure, we need to prove the locality of its topological operations. There is only one topological operation within the Quad-Edge data structure: the Splice operation. In the next paragraph, we study the scope of the Splice operation.

$Splice[a, b]$ constructs a new edge algebra $A' = (E, E^*, Onext', Rot, Flip)$ from an existing edge algebra $A = (E, E^*, Onext, Rot, Flip)$. The only difference between A and A' is their $Onext$ edge function. $Onext'$ differs from $Onext$ in the following ways:

- interchange the values of the next edge with the same origin of a with the next edge with the same origin of b:
  - the edge immediately following a with the same origin in A' is the edge immediately following b with the same origin in A (see Figure 5): $a\,Onext' = b\,Onext$,
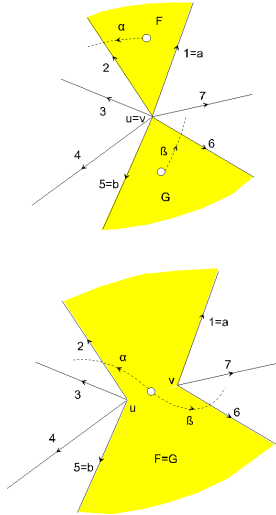 - the edge immediately following b with the same origin in A' is the edge immediately follow-

Figure 5. The Splice topological operator

Figure 6. A modified Quad-edge

ing a with the same origin in A (see Figure 5):
$b\,Onext' = a\,Onext$;

- interchange the values of the next edge with same origin of $\alpha = a\,Onext\,Rot$ (see Figure 5) with the next edge with same origin of $\beta = b\,Onext\,Rot$ (see Figure 5):

    - the edge immediately following α with same origin in A' is the edge immediately following β with same origin in A: $\alpha\,Onext' = \beta\,Onext$,
    - the edge immediately following β with the same origin in A' is the edge immediately following α with the same origin in A: $\beta\,Onext' = \alpha\,Onext$;

- for each change of the value of the next edge with the same origin of some edge e (i.e. $e\,Onext' = f$), re-define the next edge with same origin of the flipped version of f ($f\,Flip\,Onext'$) to be the flipped version of e:

    - the edge immediately following $b\,Onext\,Flip$ with same origin in A' is the flipped version of a in A: $(b\,Onext\,Flip)\,Onext' = a\,Flip$,
    - the edge immediately following $a\,Onext\,Flip$ with the same origin in A' is the flipped version of b in A: $(a\,Onext\,Flip)\,Onext' = b\,Flip$,
    - the edge immediately following $\beta\,Onext\,Flip$ with same origin in A' is the flipped version of α in A: $(\beta\,Onext\,Flip)\,Onext' = \alpha\,Flip$,
    - the edge immediately following $\alpha\,Onext\,Flip$ with same origin in A' is the flipped version of β in A: $(\alpha\,Onext\,Flip)\,Onext' = \beta\,Flip$.

Now, we can conclude that the scope of the Splice operation is limited to the edges a, b, α, and β, and the flipped version of the next edges with the same origin of a, b, α, and β ($a\,Onext\,Flip$, $b\,Onext\,Flip$, $\alpha\,Onext\,Flip$, $\beta\,Onext\,Flip$). We conclude that the Splice operation has a local scope, and therefore, that the Quad-Edge data structure has a local scope.

This property of the Quad-edge data structure imposes the following definition of edge modifications due to the operations on the data structure:

- newly created Quad-edges, when a new point is inserted into the structure (one Voronoi region is created for the newly inserted point, and the neighbouring regions are modified);

- deleted Quad-edges, where the edges belonging to the deleted point are deleted (the deletion of the point or line segment in the Voronoi diagram removes its belonging Voronoi cell, and consequently the deletion and the modification of the edges occur);

- modified Quad-edges, under stolen area interpolation (see [5]), and triangle switches. Modified edges are edges with the same ID as before, only one or two vertices are changed (see Figure 6).

2.5 Reversibility of the Quad-edge operations:

Within vector based GIS systems, the operations of maintenance of topology are not reversible. The topology of the entire map is computed by batch operations. The only way to revert to a previous state of the entire map is to store the map before and after each set of batch operation. There is no possibility to revert to a previous local state (i.e. to reverse the topology operation on a region of a map). In this section, we will see that the set of operations on the Quad-Edge data structure is equal to its closure

under inversion. This is what we mean by reversibility of the Quad-Edge operations. From the reversibility of the operations on the Quad-Edge data structure, we will prove in a later section that the set of the operations on the Voronoi data structure is also closed by inversion. In order to prove the closure of the set of operations on the Quad-Edge data structure, we need to prove that the inverse of each one of the operations on the Quad-Edge data structure pertains to the set of operations on the Quad-Edge data structure. Before doing this, we prove the reversibility of the operations on the edge algebra, on which the Quad-Edge data structure is based.

In order to prove the reversibility of the operations on an edge algebra $A = (E, E^*, Onext, Rot, Flip)$, we need to prove the reversibility of the primitive edge functions $Onext$, Rot, and $Flip$.

The $Onext$ edge function maps an edge of the primal E to an edge of the primal E, or an edge of the dual E^* to an edge of the dual E^*:

$Onext$:

$E \longrightarrow E$

$E^* \longrightarrow E^*$

$e \longrightarrow e.Onext.$

In both cases, the image of e is the edge immediately following e with same origin.

The reverse of $Onext$ is also an edge function: it is $Oprev$, and its decomposition using edge algebra primitive edge functions is $RotOnextRot$:

$Onext^{-1} = RotOnextRot$:

$E \longrightarrow E$

$E^* \longrightarrow E^*$

$e \longrightarrow e.RotOnextRot = e.Onext^{-1}$

The $Flip$ edge function maps an edge of the primal E to an edge of the primal E, or an edge of the dual E^* to an edge of the dual E^*:

$Flip$:

$E \longrightarrow E$

$E^* \longrightarrow E^*$

$e \longrightarrow e.Flip.$

In both cases, the image of e is the flipped version of e (i.e. the edge connecting the same vertices as e, with the same direction as e, but with opposite orientation).

The reverse of $Flip$ is also an edge function: it is $Flip$ itself: $Flip$ is an involution ($Flip^2 = id$):

$Flip^{-1} = Flip$

$E \longrightarrow E$

$E^* \longrightarrow E^*$

$e \longrightarrow e.Flip^{-1} = e.Flip$

The Rot edge function maps an edge of the primal E to an edge of the dual E^*, or an edge of the dual E^* to an edge of the primal E^*:

Rot:

$E \longrightarrow E^*$

$E^* \longrightarrow E$

$e \longrightarrow e.Rot$

The reverse of Rot is also an edge function: it is Rot^3, and its decomposition using edge algebra primitive edge functions is $RotRotRot$:

$Rot^{-1} = Rot^3$:

$E^* \longrightarrow E$

$E \longrightarrow E^*$

$e \longrightarrow e.Rot^3 = eRotRotRot$

The reversibility of the other edge functions results from application of the reversion of the composition of applications. Let f and g denote two edge functions, then $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$. Let us apply it to some edge function: the reverse of the $Lprev$ edge function whose decomposition into primitive edge functions is $Onext\,Rot^2$, is $Lprev^{-1} = (Onext\,Rot^2)^{-1} = (Rot^2)^{-1}(Onext)^{-1} = Rot^2\,Rot\,Onext\,Rot = Rot^3\,Onext\,Rot$, which is an edge function that can be also written as composition of the primitive edge functions $Onext$, Rot, and $Flip$. The same reasoning can be applied to any edge functions in order to prove its reversibility.

Let us examine now the reversibility of the quad-edge topological operations. The quad-edge topological operations are:

e:=MakeEdge[] creates a new data structure representing a subdivision of the sphere, where apart from orientation and direction, e is the only edge of the subdivision, and e is not a loop [7]. Its reverse would be DestroyEdge[e], destroying a data structure representing a subdivision of the sphere, where apart from orientation and direction, e is the only edge of the subdivision, and e is not a loop. Therefore, the edge e must have been disconnected from all the edges connected to it before calling the DestroyEdge operation.

Splice[a,b] is self reversible: $(Splice[a, b])^{-1} = (Splice[a, b])$. These operations can be written in the terms of edge algebra[2].

In the next section the further formalization of the operations within the Voronoi diagram will be presented.

---

[2]In section 3.1 of [7], it is shown that the topology of a subdivision is completely determined by its edge algebra.

3 The atomic actions on the dynamic Voronoi data structure

These map state changes are produced by map commands [3], that are composed of atomic actions. Each atomic action in the map command executes the geometric algorithm for addition, deletion or change of map objects and corresponding Voronoi cells.

The *atomic actions* are:

- the *Split* action inserts a new point into the structure by splitting the nearest point from the pointed location into two points.

- the *Merge* action deletes the selected point by merging it with its nearest neighbour.

- the *Switch* action is performed when a point moves and a topological event occurs (i.e. the moving point enters or exits a circle circumscribed to a Delaunay triangle, switching[3] the common boundary of two adjacent triangles.

In the following tables the Quad-edge implementations of the atomic actions *Switch* in the Voronoi spatial data structure are given.

- the *Link* action adds a line segment[4] between the points obtained after a Split action. The Link action must occur after a Split action, and adds a line segment between the point selected for splitting and the newly created point.

- the *Unlink* action removes the selected line segment. The Unlink action must occur before a Merge action, and removes the line segment between the selected point and its nearest object.

These actions compose the set of atomic actions of the dynamic spatial Voronoi data structure [8].

3.1 The Quad-edge implementation of the atomic actions

The Quad-edge implementations of the atomic actions *Split* and *Merge* in the Voronoi spatial data structure are given in Tables 4 and 5.

The Quad-edge implementation of the atomic action *Switch* is shown in Table 6. On Figure 7 we can see the topological event caused by "swap" atomic operation.

---

[3]The *Switch* action will be used in the construction of the *Move* action. The *Move* (topological event) action moves the selected point from its current position to a new position or until the next topological event.

[4]A line segment is composed of two half-line segments, whose Voronoi regions are on each side of the line segment, having the line segment as a common boundary.

Atomic operation	Quad-Edge implementation

Split	e:=Locate[X];
	base:=MakeEdge[];
	base.Dest:=X;
	Splice[base,e];
	base:=Con.[e,base.Sym];
	e:=base.Oprev;
	base:=Con.[e,base.Sym]

Table 4. Split operation

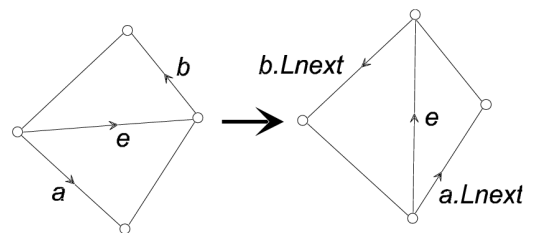| Atomic operation	Quad-Edge implementation

Merge	e:=Locate[X];
	e.org=X;
	e:=e.Sym;
	DeleteE.[e.Onext.Onext];
	DeleteE.[e.Onext];
	DeleteE.[e]

Table 5. Merge operation

Figure 7. The topological event caused by "swap" atomic operation

Atomic operation	Quad-Edge implementation
Topological event (Move)	Swap[e] where e is the "suspect" edge (see Figure 7). A suspect edge is an edge that is no longer valid, because the Delaunay triangulation does not obey the empty circumcircle criterion

Table 6. Topological event operation

The Quad-edge implementations of the atomic actions *Link* and *Unlink* in the Voronoi spatial data structure are given in Tables 7 and 8.

4 The map update commands

The *atomic actions* are the basis upon which *map commands* have been built. All the map update commands [3], [10] of this dynamic Voronoi data structure are complex operations composed of atomic actions. The composition of atomic actions into map commands is provided by syntactic rules.

The *map commands* are composed of atomic operations, and the exact decomposition of map commands into sequences of atomic actions is given in Table 9. The atomic operations are denoted by the symbols (*N, S, M, L* and *U*).

For example, the map command *"Move a Point"* corresponds to the sequence of movements of the point from its initial position to its destination through all the intersections of its trajectory with circumcircles, and the corresponding triangles switches (*"N"*) in the Voronoi data structure. The map command *"Move a Point"* is possible in this Voronoi data structure because the Voronoi data structure is kinematic : one point may move at a time, and this point is called the *"moving point"* [4, 11]. In fact, all the operations on this kinematic Voronoi data structure use this concept of the moving point.

For example, when a point is to be created at some location, the nearest point from that location is split into two (*S* term in the decomposition of *"Add a Point"* operation SN^t), and then the newly created point is moved as far as its final destination (N^t term in the decomposition of *"Add a Point"* operation SN^t). In fact, the triangle switch operation incorporates the movement of the moving point to the intersection of the trajectory of the moving point with the circumcircle that induced the triangle switch. In Table 9, the exponents denote how many times

Atomic operation	Quad-Edge implementation

Link	e:=Locate[X];
	e.Org;
	e:=e.Sym;
	While e.Dest:
	e:=e.Onext;
	a:=e.Onext;
	b:=e.Oprev;
	c:=b.Lnext;
	d:=a.Rprev;
	DeleteE.[e];
	f:=MakeE.[];
	f.dest :=(X+Y)/2;
	Splice[b,f];
	e:=MakeE.[];
	e.org:=f.dest;
	Splice[f,e.Sym];
	g:=Con.[e.sym,a.Sym];
	i:=Con.[d.sym,g.Sym];
	h:=Con.[b,f.Sym];
	j:=Con.[c,h.Sym];
	k:=Con.[f,e]

Table 7. Link operation

Atomic operation	Quad-Edge implementation

Unlink	e:=Locate[X];
	e.Org;
	e:=e.Sym;
	While e.Onext.Dest e.Dest:
	e:=e.Onext;
	f:=e.Onext;
	g:=e.Lnext;
	h:=g.Oprev;
	i:=g.Rprev;
	j:=f.Lnext;
	k:=e.Rprev;
	a:=e.Sym.Lnext;
	b:=k.Lnext;
	DeleteE.[e];
	DeleteE.[f];
	DeleteE.[g];
	DeleteE.[h];
	DeleteE.[i];
	DeleteE.[j];
	DeleteE.[k];
	Connect[b,a]

Table 8. Unlink operation

the operation is executed repeatedly, e.g. N^t denotes N executed t times, where t denotes the number of topological events. Whenever more than one connected sequence of topological events is executed in a map command, such as in "*Add a Line*" command *(SN^{t_1} SLN^{t_2} ($SLN^{t_{2i+1}}$ $MSLN^{t_{2i+2}}$))*, the total number of topological events is broken down into the number of topological events in the first connected sequence *(N^{t_1})*, the number of topological events in the second connected sequence *(N^{t_2})*, and so on. The parameter i denotes the number of times the line segment being added intersects existing line segments. This type of intersection with an existing line segment is called collision. The terms in parentheses are repeated for each intersection with an existing line (i.e. each collision).

We will now briefly explain the decomposition of each map command. We have already seen the description of "*Move a Point*" and "*Add a Point*" map commands. Map command "*Delete a Point*" is exactly the reverse of "*Add a Point*" map command: the point to be deleted is moved to the location of the nearest point (N^t), and then it is merged with this nearest point (M). The remaining map commands involve the addition or removal of one or more new line segments. For all these map commands, the decomposition includes a fixed sequence of atomic actions that is executed only once (the sequence outside the parenthesis), and a sequence that is executed at each collision (replicating sequence). In the case of case"*Add a Line*" and all the join map commands, the replicating sequence has always the same pattern in terms of atomic operations (($SLN^{t_{2i+1}}MSLN^{t_{2i+2}}$), although the actual indices may vary). This corresponds to the splitting of the existing line ($SLN^{t_{2i+1}}$), the merging of the newly created point (by the S atomic action in this last sequence) with the extremity of the line segment being added (M), and the continuation of the new line segment after collision ($SLN^{t_{2i+2}}$). In the case of "*Delete a Line*" and all the unjoin map commands, the replicating sequence has always the same pattern in terms of atomic operations (($N^{t_{2i+2}}UMSN^{t_{2i+1}}UM$), although the actual indices may vary). This is exactly the reverse of the previous replicating sequence (the replication sequence for "*Add a Line*" and all the join map commands).

Now, we will explain the fixed sequence for all these map commands. In order to add a line with "*Add a Line*" map command, the nearest point from the starting extremity location has to be split into two (S), then it has to be moved to the starting extremity location (N^{t_1}). Then, the ending extremity has to be created by splitting the starting extremity into two (S). At this point the two extremities must be linked (L) in order to form a line segment. Finally, the ending extremity has to be moved (N^{t_2}) to its expected location. The fixed sequence for "*Delete a Line*"

Map construction command	Decomposition (the terms in parentheses appear at each line-line collision, index i = collision, $1 \le i \le c$, c=#$collisions$; t, t_x denote of topological events)
Move a Point	N^t
Add a Point	SN^t
Delete a Point	$N^t M$
Add a Line	$SN^{t_1} SLN^{t_2} (SLN^{t_{2i+1}}) M$ $(SLN^{t_{2i+2}}) N^{t_2} UMN^{t_1} M$
Delete a Line	$(N^{t_{2i+2}} UMSN^{t_{2i+1}} UM)$ $N^{t_2} UMN^{t_1} M$
Join 2 Points	$SLN^{t_1} (SLN^{t_{2i}} MSLN^{t_{2i+1}}) M$
Unjoin 2 Points	$(N^{t_{2i+1}} UMSN^{t_{2i}} UM) N^{t_1} UM$
Join pt Line	$SLN^{t_1} (SLN^{t_{2i+1}} MSLN^{t_{2i+2}})$ $SLN^{t_2} M$
Unjoin Pt Line	$SN^{t_2} UM (N^{t_{2i+2}} UMSN^{t_{2i+1}} UM)$ $N^{t_1} UM$
Join 2 Lines	$SLN^{t_1} SLN^{t_2} (SLN^{t_{2i+2}}) M$ $(SLN^{t_{2i+3}}) SLN^{t_3} M$
Unjoin 2 Lines	$SN^{t_3} UM (N^{t_{2i+3}} UMSN^{t_{2i+2}} UM)$ $N^{t_2} UMN^{t_1} UM$

Table 9. The map commands and their decomposition into atomic actions

is exactly the reverse of the preceding sequence. In order to join two points with "*Join two points*" map command, the first point must be split into two (S) in order to create the ending extremity of the line segment that starts at the first point. Then, these two points must be linked (L) in order to form a line segment. Then, the ending extremity must be moved (N^{t_1}) to the location of the second point (including eventually the replicating sequence in case of collisions). Finally, the ending extremity must be merged with the second point (M). The fixed sequence for "*Unjoin two points*" map command is exactly the reverse of the fixed sequence for "*Join two points*" map command. The fixed sequences of the remaining map commands follow immediately from the fixed sequence of "*Join two points*" map command. Indeed, the other join map commands fixed sequence involve several sequences corresponding to the same atomic actions as the SLN^{t_1} sequence already encountered in the fixed sequence of "*Join two points*" map command. The unjoin map commands are the exact reverse of their join counterpart.

5 Reversibility of the map commands in the dynamic Voronoi data structure

For each map command, the reverse map command is composed of reverse atomic actions in exactly the reverse

Atomic action	Reverse atomic action
Split	Merge
Switch	Switch is self-reversible
Link	Unlink

Table 10. The reversibility of the atomic actions

Map construction command	Reverse map construction command
Move a Point	Self-reversible
Add a Point	Delete a Point
Add a Line	Delete a Line
Join 2 Points	Unjoin 2 Points
Join Pt & Line	Unjoin Pt & Line
Join 2 Lines	Unjoin 2 Lines

Table 11. The reversibility of the map commands

order. Due to the local scope of its spatio-temporal topology, all the atomic actions of the dynamic Voronoi spatio-temporal model are reversible. Indeed, each atomic action has its reverse atomic action shown in the Table 10.

The consequence of the property of reversibility of the atomic actions inside the Voronoi dynamic data structure is that a sequence of atomic actions applied in a map update command can be reconstructed from the predecessor and successor map states. This proves in another way that the atomic actions are reversible: the input can be deduced from the output; or, in other words, computation happens without any loss of information [2].

The resulting complex operations (map commands) are reversible and Table 11), as long as their decomposition into atomic actions is exactly known (including the numbers of topological events and the number of line-line collisions).

6 Conclusions

In this paper we presented formalisation of the reversible operations needed for constructing a Voronoi diagram for points and line segments using the Quad-Edge data structure. These reversible operations are formalized at the lowest level, as the basic algorithms for addition, deletion and moving of spatial objects in the Quad-Edge data structure; defined as the atomic actions. Furthermore, we managed to preserve the reversibility of the map commands that are composed of these atomic actions.

The applications of reversible computations in GIS could significantly improve transaction management and rollback functionality.

References

[1] Anton, F. and Gold, C. M., 1997, An iterative algorithm for the determination of Voronoi vertices in polygonal and non-polygonal domains, *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG'97)*, Kingston, Canada, pp. 257-262.

[2] Frank, M., Knight, T., Margolus, N., 1998, Reversibility in optimally scalable computer architectures, *The First International Conference on Unconventional Models of Computation*, January 1998.

[3] Gold, C. M., 1992, An object-based dynamic spatial data model, and its applications in the development of a user-friendly digitizing system, *Proceedings of the Fifth International Symposium on Spatial Data Handling*, Charleston, pp. 495-504.

[4] Gold., C. M., 1994, Three approaches to automated topology, and how computational geometry helps, *Proceedings of the Sixth International Seminar on Spatial Data Handling*, Edinburgh, Scotland, pp. 145-158.

[5] Gold, C. M., Remmele, P. R., Roos, T., 1995 Voronoi Diagrams of Line Segments Made Easy, *Proceedings of the Seventh Canadian Conference in Computational Geometry, (CCCG'95)*, Québec, Canada, pp. 223-228.

[6] Gold C. M, and Dakowicz M., 2006, Kinetic Voronoi/Delaunay Drawing Tools, *ISVD*, pp. 76-84.

[7] Guibas, L., and Stolfi, J., 1985, Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, *ACM Transactions on Graphics*, Vol. **4**, No. 2, pp. 74-123.

[8] Mioc, D., Anton, F., Gold, C. and Moulin, B., 1998, Spatio-temporal change representation and map updates in a dynamic Voronoi data structure, *Proceedings of the Eight International Symposium on Spatial Data Handling*, Vancouver, Canada, pp. 441-452.

[9] Mioc, D., Anton F., Gold C. M. and Moulin B., 1999, Time-travel visualization of changes in a dynamic Voronoi data structure, *Cartography and GIS*, Vol. **26**, No. 2, pp. 99-108.

[10] Mioc, D., Anton F., Gold C. M. and Moulin B., 2006, Map updates in a dynamic Voronoi data structure, *ISVD*, pp. 264-269.

[11] Roos, T., 1991, Dynamic Voronoi diagrams, *Ph.D. Thesis, University of Würzburg*, Germany.