PERGAMON

Computers & Geosciences 27 (2001) 959–973

COMPUTERS & GEOSCIENCES

An object-oriented framework for distributed hydrologic and geomorphic modeling using triangulated irregular networks

Gregory E. Tucker*, Stephen T. Lancaster[1], Nicole M. Gasparini, Rafael L. Bras, Scott M. Rybarczyk

Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Received 1 January 1999; accepted 12 October 1999

Abstract

We describe a new set of data structures and algorithms for dynamic terrain modeling using a triangulated irregular network (TINs). The framework provides an efficient method for storing, accessing, and updating a Delaunay triangulation and its associated Voronoi diagram. The basic data structure consists of three interconnected data objects: triangles, nodes, and directed edges. Encapsulating each of these geometric elements within a data object makes it possible to essentially decouple the TIN representation from the modeling applications that make use of it. Both the triangulation and its corresponding Voronoi diagram can be rapidly retrieved or updated, making these methods well suited to adaptive remeshing schemes. We develop a set of algorithms for defining drainage networks and identifying closed depressions (e.g., lakes) for hydrologic and geomorphic modeling applications. We also outline simple numerical algorithms for solving network routing and 2D transport equations within the TIN framework. The methods are illustrated with two example applications, a landscape evolution model and a distributed rainfall-runoff model.

Keywords: Geomorphology; Runoff; Erosion; Triangulation

1. Introduction

Continuing advances in computing technology have made terrain-based modeling an attractive approach for numerous geologic and hydrologic applications. Distributed modeling of processes occurring on or near a topographic surface has been widely used, for example, in watershed hydrology (e.g., Garrote and Bras, 1995; Julien et al., 1995; Jackson et al., 1996), landscape evolution (e.g., Willgoose et al., 1991; Howard, 1994; Johnson and Beaumont, 1995; Tucker and Slingerland, 1997; Tucker and Bras, 1998), soil erosion (e.g., Laflen et al., 1997; Mitas and Mitasova, 1998), slope stability analysis (e.g., Montgomery and Dietrich, 1994), and volcanology (e.g., Miyamoto and Sasaki, 1997), among other applications. Although the nature of these models ranges from real-time prediction of runoff in actual drainage basins (e.g., Garrote and Bras, 1995) to hypothetical simulation of mountain range evolution over millions of years (e.g., Tucker and Slingerland, 1996), all of them share the common theme of simulating flow — whether of water, sediment, or magma — across (or near) a topographic surface. The need to model flow over terrain gives rise to three common elements among these otherwise disparate models: (1) division of a terrain surface into a set of discrete, connected elements, (2) application of continuity of mass within each terrain element, and (3) definition of flow pathways and networks across this discretized terrain surface.

*Correspondence address: School of Geography and the Environment, Oxford University, Oxford OX1 3TB, UK.

E-mail address: greg.tucker@geog.ox.ac.uk (G.E. Tucker).

[1] Now at: Department of Geosciences, Oregon State University, Corvallis, OR, USA.

0098-3004/01/$ - see front matter
PII: S0098-3004(00)00134-5

A number of different strategies have been used for terrain discretization, including regular grids, triangulated irregular networks (TINs), sub-watersheds, contour elements (e.g., Moore et al., 1988), and hillslope partitions (e.g., Band, 1989). Among these, only regular grids and TINs are well suited for simulating the dynamics of surface change. The simplicity of regular grids and the increasing availability of grid-based digital elevation models (DEMs) have made fixed grids the framework of choice in most hydrologic models and nearly all geomorphic models (a notable exception being the CASCADE model of Braun and Sambridge (1997)). However, grid-based discretization schemes suffer from a number of distinct disadvantages: (1) landform elements must be represented at a constant spatial resolution, which in practice means the highest resolution required by any feature or process of interest; (2) drainage directions are restricted to 45° increments (though for watershed-scale applications, this limitation may be reduced by using multiple-flow algorithms, e.g., Freeman, 1991; Quinn et al., 1991; Costa-Cabral and Burges, 1994; Tarboton, 1997); (3) under certain circumstances, use of a regular grid introduces anisotropy that can lead to bias in simulated drainage network patterns (Braun and Sambridge, 1997); and (4) use of a fixed grid makes it difficult or impossible to model geologic processes that have a significant horizontal component, such as stream meandering or fault displacement.

The last of these constraints is especially significant in the context of geological models. Although we conventionally speak of "uplift", most crustal deformation processes involve a significant amount of horizontal translation. Previous coupled models have either incorporated only the vertical component of deformation (e.g., Tucker and Slingerland, 1996; Kooi and Beaumont, 1996) or have represented lateral translation by simply offsetting two fixed grids (e.g., Anderson, 1994). Coupled models of deformation, erosion, and sedimentation promise to yield important insights into such issues as the relationship between tectonic behavior and the stratigraphic record, but such models ultimately require the ability to model deformation in three dimensions. Similarly, erosional processes often have a significant horizontal component that is neglected in current models. One of the most important horizontal erosion processes is lateral stream erosion, which by widening a valley can significantly alter the depositional geometry within a floodplain over geologic time.

These and other disadvantages have motivated the development of terrain models based on TINs, which allow for variable spatial resolution, lend themselves naturally to interpolation procedures (e.g., Sambridge et al., 1995), and make dynamic rediscretization a real possibility (e.g., Braun and Sambridge, 1997; Lancaster, 1998). Despite these advantages, however, use of TIN-based dynamic models has not been widespread, in part because of the increased complexity of data structures and algorithm development in a TIN framework. In this paper, we present an efficient set of data structures and algorithms for TIN-based modeling using the Delaunay triangulation criterion. These data structures and algorithms take advantage of the unique capabilities of object-oriented programming languages such as C++ to provide a general framework for (1) storing and rapidly accessing information about mesh connectivity, (2) constructing and updating mesh geometry, (3) computing mass fluxes and maintaining continuity of mass within mesh elements using a finite-difference or finite-volume approach, and (4) establishing drainage pathways across the terrain surface. We present example applications and briefly discuss the use of adaptive meshing to simulate lateral stream channel migration.

2. Delaunay triangulation and Voronoi diagrams

A number of methods for triangulating a set of irregularly spaced points have been proposed. Here, we focus on the commonly used Delaunay triangulation, which offers a number of distinct advantages over other tessellation schemes (Watson and Philip, 1984). Delaunay triangulations and their corresponding Voronoi diagrams are well established in the field of computational geometry (e.g., Guibas and Stolfi, 1985; Sloan, 1987; Knuth, 1992; Sugihara and Iri, 1994; Sambridge et al., 1995; Du, 1996). The Delaunay triangulation of a set of points N_i is a unique triangulation having the property that a circle passing through the three points of any triangle will encompass no other points. A Delaunay triangulation therefore minimizes the maximum interior angles, providing the most "equable" triangulation of a given set of points. From any Delaunay triangulation it is also possible to construct a unique corresponding Voronoi (or Thiessen) diagram, which is the set of polygons formed by connecting the perpendicular bisectors of the triangles (Fig. 1). Although Voronoi polygons have been ignored in most TIN-based models, they have the advantage of providing a natural framework for numerical modeling using finite-volume and finite-difference methods.

3. Mesh elements and data structures

A terrain surface can be represented by a set of nodes N that are connected to form a mesh of triangles using the Delaunay triangulation of N. Each node N_i is associated with a Voronoi polygon of area A_i (Fig. 1). The Voronoi polygon for a node N_i is the region within which any arbitrary point Q would be closer to N_i than to any other node on the mesh. The boundaries between

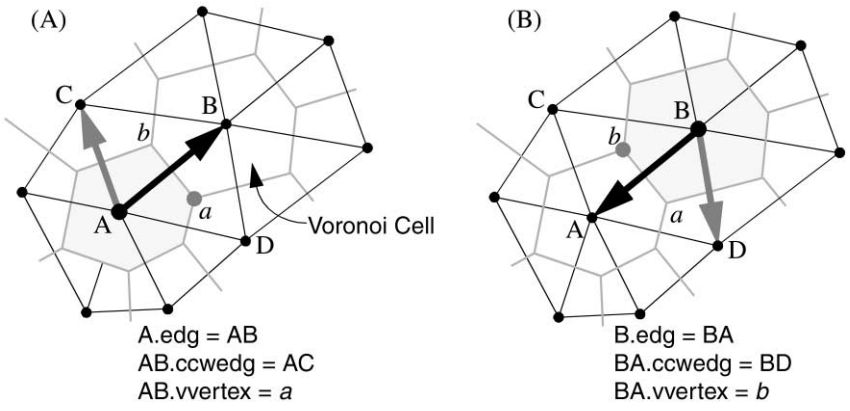

Fig. 1. Illustration of dual edge data structure, showing triangular lattice (black) and corresponding Voronoi diagram (gray). (A) Directed edge AB (black arrow), its counterclockwise neighbor AC (gray arrow), and its right-hand Voronoi vertex *a*. (B) Directed edge BA (black arrow), its counterclockwise neighbor BD (gray arrow), and its right-hand Voronoi vertex *b*.

Voronoi polygons are lines of equal distance between adjacent nodes. The vertices of the Voronoi polygons (here termed *Voronoi vertices*) coincide with the circumcenters of the triangles; in general, each triangle is associated with one and only one Voronoi vertex.

Unlike regular grids, in which each node is connected to either four or eight adjacent neighbors, the number of neighbors connected to a given node in a Delaunay triangulation may in theory be arbitrarily large. Ideally, a data structure should represent this variable connectivity in a way that (1) provides rapid access to adjacent mesh elements without demanding excessive storage space, and (2) is flexible enough to handle dynamic changes in the mesh itself. For dynamic modeling applications, an additional requirement is the need to maximize computational speed. These requirements are satisfied through the use of the following data structure. The "dual edge" structure is adapted from the quad edge data structure of Guibas and Stolfi (1985), and consists of three geometric elements: *nodes, triangles*, and *directed edges*. The data structure is illustrated in Fig. 1 and summarized in pseudo-code form in Fig. 2.

```
Class Node
    x          // x-coordinate
    y          // y-coordinate
    z          // z-coordinate (elevation)
    edg        // pointer to one connected edge
    nnbrs      // number of neighboring nodes
    bnd_code// boundary status code

Class DirectedEdge
    org        // origin node
    dest       // destination node
    ccwedg     // pointer next directed edge counterclockwise
    vvertex_x // x-coordinate of right-hand Voronoi vertex
    vvertex_y // y-coordinate of right-hand Voronoi vertex

Class Triangle
    p(3)       // Pointers to vertex nodes
    t(3)       // Pointers to adjacent triangles (t(1) is opposite p(1), etc.)
    e(3)       // Pointers to clockwise-oriented directed edges
```

Fig. 2. Pseudo-code summary of dual edge data structure, showing data members belonging to Node, DirectedEdge, and Triangle objects.

3.1. Directed edges

Each triangle edge is associated with two *directed edges*, which share the same endpoints but are oriented in opposite directions (Fig. 1). A directed edge is defined by its origin and destination nodes; two directed edges that share the same endpoints are termed *complementary edges*. Each directed edge data object includes a pointer to its origin node, a pointer to its destination node, and a pointer to the directed edge that lies immediately counter-clockwise relative to its origin node (Figs. 2, 3 and Table 1). (Here, we use the term *pointer* to refer to any reference to another data object, whether implemented as a memory address, an array index, or by some other means.) Including a pointer to the counterclockwise edge makes it possible to rapidly access all of the edges and neighboring nodes connected to any given node.

Each directed edge data object also includes the coordinates of the Voronoi vertex associated with the triangle on its right-hand side (clockwise) (Fig. 1). Pseudo-code for the directed-edge data structure is given in Fig. 2, and an example list of edges for a simple mesh (Fig. 3) is given in Table 1. Complementary directed edges are stored pairwise on the list (Table 1), which makes it simple to retrieve the complement of any given edge. In addition, certain operations such as length and slope calculation only need to be performed for one member of each edge pair, with the result simply assigned to the other. Storing directed edges pairwise makes it easy to do this by skipping every other edge on the list. In addition to topologic

information, a directed edge object might also include geometric properties such as length, gradient, and the length of the associated Voronoi polygon edge (or *Voronoi edge*).

3.2. Nodes

Each node data object includes x, y, z coordinates, the number of neighboring nodes, and a pointer to one of its directed edges (that is, any one of the directed edges which originates at the node, here referred to as a *spoke* of that node) (Figs. 2, 3 and Table 2). Note that a pointer to a single spoke is all that is needed to fully describe the connectivity among nodes. Because each spoke points to its counter-clockwise neighbor, a list of spokes and neighboring nodes can easily be constructed for any node using the algorithms described below.

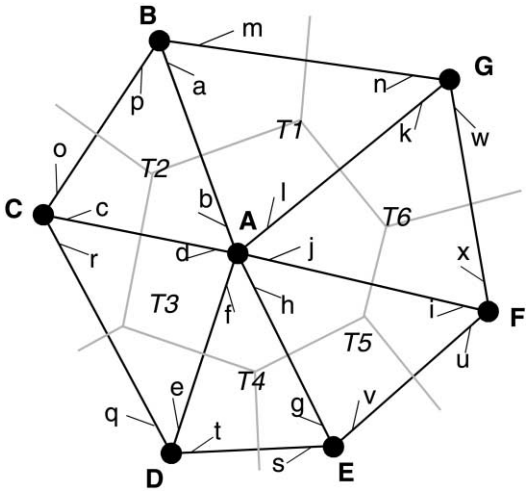

Fig. 3. Sample mesh consisting of seven nodes, six triangles, and 24 directed edges. Capital letters indicate nodes, numbers T1, etc., denote triangles, and small letters indicate directed edges. Half-arrows pointing toward destination node indicate orientation of directed edges.

For finite-volume and finite-difference applications, a node object can also include geometric information such as the projected surface area of its Voronoi polygon (*Voronoi area*) and a flag indicating whether it lies on the boundary or interior of the mesh. An important advantage of an object-oriented approach in this regard is that basic topologic and geometric data and functions can be encapsulated within a base class, with application-specific data and functions added as part of an inherited class. This technique is exploited in the examples discussed below.

3.3. Triangles

Triangle data objects include pointers to the three nodes in the triangle, the three neighboring triangles, and the three directed edges that are oriented clockwise with respect to the triangle. The nodes and neighboring triangles are numbered in such a way that the nth neighboring triangle lies opposite the nth vertex (Fig. 4). Each triangle is also associated with a single Voronoi vertex, which represents the intersection of the three Voronoi polygons belonging to the triangle's three nodes (Fig. 3). Here, however, the coordinates of Voronoi vertices are stored within directed edge objects

Table 2
List of nodes for sample mesh shown in Fig. 3

Node	EDG	No. NBRS	Boundary code
A	*a*	6	0
B	*o*	3	1
C	*q*	3	1
D	*r*	3	1
E	*h*	3	1
F	*w*	3	1
G	*m*	3	1

Table 1
List of directed edges for sample mesh shown in Fig. 3

Directed edge	Origin node	Destination node	CCW edge	RH Voronoi vertex
a	*A*	*B*	*c*	$CC(T1)$
b	*B*	*A*	*n*	$CC(T2)$
c	*A*	*C*	*e*	$CC(T2)$
d	*C*	*A*	*p*	$CC(T3)$
e	*A*	*D*	*g*	$CC(T3)$
f	*D*	*A*	*r*	$CC(T4)$
⋮				
M	*G*	*B*	*l*	$(NULL)$
N	*B*	*G*	*o*	$CC(T1)$
⋮				

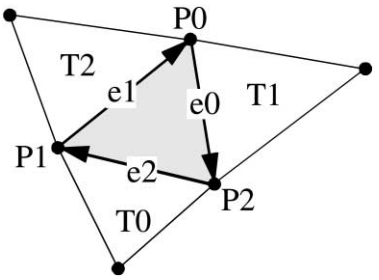

Fig. 4. Illustration of numbering of triangle nodes, adjacent triangles, and clockwise edges in Triangle data objects.

Table 3
List of triangles for sample mesh shown in Fig. 3

Triangle	Nodes	Adjacent triangles	Clockwise-oriented edges
T1	B, A, G	T6, −1, T2	n, a, l
T2	A, B, C	−1, T3, T1	c, b, p
T3	C, D, A	T4, T2, −1	d, r, e
T4	D, E, A	T5, T3, −1	f, t, g
T5	E, F, A	T6, T4, −1	h, v, i
T6	F, G, A	T1, T5, −1	j, x, k

rather than with triangles. This represents a time-for-space tradeoff: by storing each Voronoi vertex three times (one for each clockwise edge) it becomes possible to recalculate Voronoi polygon geometry quickly in response to node addition, deletion or movement. The list of triangles for a simple example mesh is given in Table 3. Depending on the application, additional geometric data for a triangle data class might include projected area, gradient, and gradient vector (cf. Palacios and Cuevas, 1986).

3.4. Establishing and updating node connectivity

A list of spokes or neighboring nodes can be easily obtained using the following algorithm, here presented in Pascal-like pseudo-code:

```
Node neighborList(1..thenode.nnbrs)
Edge spokeList(1..thenode.nnbrs)
Current_edge:=thenode.edg
FOR i:=1,thenode.nnbrs DO
  neighborList(i):=current_edge.dest
  spokeList(i):=current_edge
  current_edge:=current_edge.ccwedg
END
```

Generally, neighbor node and spoke list retrieval would be handled separately; they are grouped here for simplicity. Note that creating a list (as opposed to simply performing operations on each spoke in turn) is usually not necessary, though it may be convenient for performance reasons.

3.5. Updating Voronoi geometry

Once a Delaunay triangulation has been established or updated, the associated Voronoi geometry can be updated in a straightforward way. The first step consists of updating and storing the Voronoi vertex coordinates, which can be done by sweeping through the list of triangles as follows:

```
FOR each triangle tri DO
  Find coordinates of triangle circumcenter
  FOR i:=0,2 DO
    tri.e(i).vvertex_x:=x-coordinate of
  circumcenter
    tri.e(i).vvertex_y:=y-coordinate of
  circumcenter
END
```

Note that for triangles with a large aspect ratio, computing the triangle circumcenter can be subject to numerical errors. These errors and strategies for overcoming them are discussed by Sugihara and Iri (1994).

Once the Voronoi vertex coordinates have been stored, the Voronoi polygon for a given node can be retrieved using the following simple algorithm:

```
FOR each node mynode DO
  XYPoint voronoi_poly(1..mynode.nnbrs)
  current_edge:=mynode.edg
  FOR i:=1,mynode.nnbrs DO
    voronoi_poly(i).x:=current_edg.rvpt.x
    voronoi_poly(i).y:=current_edg.rvpt.y
  END
  /* use voronoi_poly to calculate Voronoi area */
END
```

It is also often useful to know the length of a Voronoi polygon edge, which represents the interface between two neighboring nodes. The width of this interface can be used, for example, in numerical solutions to diffusion-like transport equations, as discussed below. In general, each triangle edge is matched by a single Voronoi polygon edge (Guibas and Stolfi, 1985) (Figs. 1 and 3). Voronoi edge lengths are updated using the following algorithm, which exploits the fact that edges are stored pairwise on the list (note that this is a projected length). The algorithm combines updating of

Voronoi edge lengths, triangle edge lengths, and triangle edge slopes:

```
FOR i:=1,3,5,...nedges−1 DO
  dx:=edg(i).rvpt.x-edg(i).ccwedg.rvpt.x
  dy :=edg(i).rvpt.y-edg(i).ccwedg.rvpt.y
  edg(i).vedglen:=sqrt(dx*dx+dy*dy)
  edg(i+1).vedglen:=edg(i).vedglen
  dx:=edg(i).org.x-edg(i).dest.x
  dy:=edg(i).org.y-edg(i).dest.y
  edg(i).length:=sqrt(dx*dx+dy*dy)
  edg(i+1).length:=edg(i).length
  edg(i).slope:=(edg(i).org.z-edg(i).dest.z)/
  edg(i).length
  edg(i+1).slope:=−edg(i).slope
END
```

Note that the endpoints of the Voronoi polygon edge are (1) the "right-hand" Voronoi vertex of the directed edge and (2) the "right-hand" vertex of the directed edge's counter-clockwise neighbor. Once again a time-for-space tradeoff is involved here: edge length, slope, and Voronoi edge length are stored in each member of a complementary edge pair, providing efficient access at the expense of extra storage.

4. Drainage networks and flow routing on a triangulated irregular mesh

An important component of hydrologic and geomorphic models is the routing of surface flow across a terrain surface. Algorithms for flow routing across a regular-grid DEM are well established and are generally fairly straightforward (e.g., O'Callaghan and Mark, 1984; Jenson and Domingue, 1988). The irregular geometry of a TIN surface presents some additional challenges. A number of TIN-based drainage algorithms have been developed for hydrologic modeling (Palacios and Cuevas, 1986; Gandoy and Palacios, 1990; Jones et al., 1990; Nelson et al., 1994). Most of these schemes are "triangle-based" in the sense that they define flow pathways both across and between triangles, using linear interpolation to approximate the terrain surface gradient within each triangle. Algorithms that use this approach are able to delineate precisely a steepest-descent pathway starting from any arbitrary point on the surface and, unlike most grid-based methods, they do not limit drainage directions to 45° increments. However, a notable disadvantage of triangle-based methods is that they are forced to handle flow across and between triangles separately. The most common method for drainage network delineation in TIN-based terrain data consists of finding locations where adjacent triangles slope downward toward their shared edge (e.g., Palacios and Cuevas, 1986; Jones et al., 1990). However, this criterion for channel identification turns out to be somewhat arbitrary, for two reasons. First, it does not account for the physical mechanisms or morphological signatures of channel initiation (e.g., Montgomery and Dietrich, 1989; Dietrich and Dunne, 1993). Second, it is sensitive to inaccuracies in either the data or the linear interpolation: even a slight concavity between adjacent triangles will always be flagged as a "channel", while a flat or weakly convex surface (such as a portion of a floodplain surface) will always be flagged as a "hillslope".

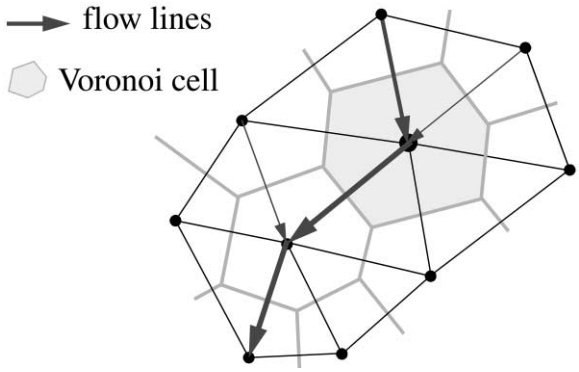

Fig. 5. Illustration of steepest-descent flow routing in TIN framework.

Here, we describe a simple "Voronoi-based" approach to routing that avoids these problems, though at the expense of added simplicity. Following Braun and Sambridge (1997), flow originating at any point within a node's Voronoi polygon is routed downslope along the steepest of the spokes connected to that node (Fig. 5). By this method, the contributing area at node i is equal to the sum of the Voronoi areas of all nodes that flow to i (including i itself). An advantage of this approach is that it lends itself to finite-difference modeling, because each node has a unique watershed and drainage direction assigned to it. The primary disadvantages are: (1) drainage basin boundaries are defined by Voronoi polygons rather than by triangles, and (2) flow pathways and gradients are forced to follow triangle edges. The first limitation can be handled by simply clipping Voronoi polygons along a specified watershed boundary. The second could be eliminated by adopting a more general flow routing procedure (e.g., Tarboton, 1997), though such an extension is beyond the scope of this paper.

4.1. Object hierarchy

The basic mesh element data structures described above and diagrammed in Fig. 2 contain only information relevant to the topology and geometry of the

triangulation. For a given application, of course, additional data and functionality related to processes (e.g., runoff) are also required. A useful advantage of an object-oriented approach is that these process-level attributes can be added in a hierarchical fashion, without actually modifying the basic underlying mesh data structures. In our case, for example, we wish to develop an application that computes surface water flow across a landscape. To create a data structure to implement this application, we can define a new kind of node data class that inherits all the properties of the basic `Node class` (Fig. 2) but adds new pieces of application-specific information:

```
Class WetNode: Node
  flowEdge       // pointer to edge along which runoff
                 flows out of the node
  drainageArea   // drainage area upstream of node
  discharge      // volumetric water flow exiting the
                 node
  flowDepth      // average depth of flow within the
                  node
```

In this example, the colon indicates that the WetNode class inherits properties from the Node class. Each WetNode object contains, in addition to the data associated with a Node object, four additional parameters related to surface runoff. This type of hierarchical design has the advantage of allowing one to create flexible and extensible applications, and it is the method we have used in developing the two example modeling systems discussed below.

4.2. Flow directions and drainage areas

To identify flow directions, the steepest spoke at each node is identified and a pointer to it is stored in `node.flowEdge`. The total contributing area for each node can then be found using the following algorithm:

```
Reset all drainage areas to zero
FOR each node sourceNode DO
  currentNode:=sourceNode
  srcArea:=source node's Voronoi area
  WHILE currentNode is neither a boundary nor a
  pit DO
    currentNode.drainageArea:=currentNode.
    drainageArea+srcArea
   currentNode:=currentNode.flowEdge.dest
```

This "Stream Trace" algorithm can also be used to compute accumulated flow at any point, if steady-state flow is assumed.

4.3. Resolving drainage from closed depressions

Digital elevation data, whether grid- or TIN-based, typically contain "pits" (closed depressions) which usually arise from errors in the data, poor data resolution, or both. Various methods have been developed to remove pits from grid-based terrain data prior to hydrologic analysis (e.g., O'Callaghan and Mark, 1984; Jenson and Domingue, 1988). Pits can also form in dynamic landscape models, either as the result of arbitrary initial conditions or in response to simulated surface deformation (e.g., Howard, 1994; Tucker and Slingerland, 1996; Braun and Sambridge, 1997). Below, we outline an algorithm for identifying flooded regions — depressions in which surface water would tend to pond — and resolving outlets from those regions. Unlike the commonly used "pit filling" procedures described by O'Callaghan and Mark (1984) and Jenson and Domingue (1988), the algorithm does not alter the underlying topography. It is capable of identifying clusters of nodes that form lakes, and therefore has potential application for environmental modeling in regions in which lakes are common (e.g., Mackay and Band, 1998).

During the process of identifying flow directions, any node lacking a downhill pathway is flagged as a pit. Once all pits have been identified, the Lake Fill algorithm (Appendix A) is invoked for each one. The algorithm begins by creating a list of flooded nodes, which initially contains only the starting pit. The lowest node on the perimeter of the flooded area ("lake") is identified, and is tested to determine whether it can drain downslope toward a node that is not already on the list. If there is no drainage outlet, this "low node" is added to the list of flooded nodes and the process repeats, continuing until an outlet is found. If a node is encountered that is part of separate lake (i.e., one identified during a previous iteration), it is also added to the list (in other words, initially separate lakes can merge).

Once an outlet has been identified, all of the nodes on the list are flagged as lake nodes, indicating, for example, that they should be handled separately in computing runoff and sediment routing. To maintain the continuity of drainage networks, flow directions for lake nodes can be resolved by iteratively tracing a flow path upstream from the lake outlet, seeking the shortest path to the outlet for each node (Appendix B). Once flow directions have been resolved, the lake list is cleared and the next pit is processed.

An example of a lake computed using the Lake Fill algorithm is shown in Fig. 6. The algorithm is robust enough to handle any arbitrary initial condition, and is useful for modeling natural or artificial reservoirs or, in geological applications, for modeling rising baselevel.

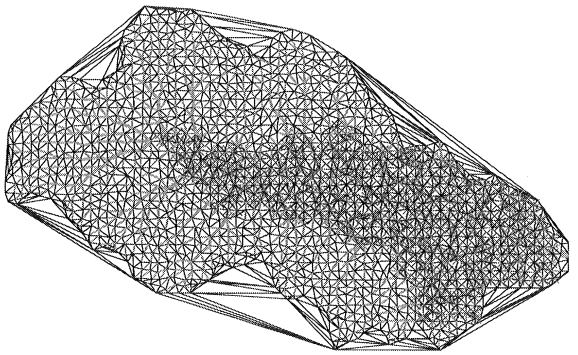

Fig. 6. Example of lake formation in CHILD model. Here, we are looking down onto TIN representation of watershed, with outlet to right. Gray lines indicate flow paths (stream lines). Gray asterisks denote lake nodes. Lake has formed in response to "digital dam" that was created by artificially raising elevation of nodes near the catchment outlet. Lake outlet is indicated by thick gray line at right-hand edge.

4.4. Ordering nodes by network position

It is often necessary to sort nodes in upstream-to-downstream order (or the reverse) before solving water or sediment routing equations. This is the case, for example, in kinematic-wave runoff models (e.g., Gandoy and Palacios, 1990) and in many landscape evolution models (e.g., Slingerland et al., 1993; Tucker and Slingerland, 1994; Braun and Sambridge, 1997). In the examples presented later, node ordering is performed using the "cascade" algorithm of Braun and Sambridge (1997), which is efficient and can be easily generalized to handle multiple flow pathways.

5. Numerical algorithms

For problems involving continuity of mass calculations (which includes most geologic and hydrologic models), mass continuity at each node can be applied by treating each Voronoi polygon as a finite-volume cell (e.g., Peyret and Taylor, 1983; Versteeg and Malalasekera, 1995). In general, the continuity of mass equation for a Voronoi cell may be written as

$$\frac{\mathrm{d}V_i}{\mathrm{d}t} = \sum_{j=1}^{N_i} Q_{ji}, \tag{1}$$

where V_i represents volume or mass stored at node i, N_i is the number of neighbor nodes connected to node i, and Q_{ji} is the total flux from node j to node i (negative if the net flux is from i to j). Efficient numerical implementation of Eq. (1), however, depends on how the flux terms are defined.

5.1. One-dimensional network transport

Transport of water and sediment within drainage networks is often modeled as a quasi-one-dimensional problem, with the network essentially treated as a cascade of one-dimensional stream segments. In modeling landscape evolution, an equation for dynamic changes in surface elevation due to fluvial erosion or deposition can be written as

$$\frac{\mathrm{d}z_i}{\mathrm{d}t} = \frac{\left(\sum_{j=1}^{n} Q_{sj}\right) - Q_{si}}{(1-v)\Lambda_i}, \tag{2}$$

where z_i is the elevation at node i, t the time, n the number of nodes that flow directly to i, Q_s the sediment flux, v the sediment porosity, and Λ_i the Voronoi area of node i (Braun and Sambridge, 1997; Tucker et al., 1997). The system of ordinary differential equations described by Eq. (2) can be solved using simple forward differencing in time, via matrix methods, or by any similar scheme. Solving Eq. (2) in upstream-to-downstream order ensures that the total incoming sediment flux at any point is always known.

5.2. Two-dimensional diffusive transport

Diffusive transport processes typically require a fully two-dimensional solution and must therefore be handled in a different manner. Examples of such quasi-diffusive transport processes include groundwater flow, hillslope sediment transport, and lava flow. In modeling 2D mass transport on an irregular mesh using a finite-volume approach, the width of the interface between adjacent nodes must be known in order to compute the total mass exchange. Consider, for example, sediment transport by hillslope processes such as soil creep, which is frequently modeled as a linear (e.g., Culling, 1960) or nonlinear (e.g., Howard, 1994; Roering et al., 1999) function of surface gradient. In its linear form, sediment transport per unit contour width, q_s, is given by

$$q_s = -k_d \frac{\partial z}{\partial \mathbf{x}}, \tag{3}$$

where k_d is a diffusivity constant and $\mathbf{x}$ is a vector oriented in the downslope direction. To solve this equation in a TIN framework, the slope width between two adjacent nodes can be approximated by the width of their shared Voronoi cell edge, λ_{ij} (Fig. 7). Combining this with continuity of mass (Eq. (1)), the rate of elevation change at a node due to diffusive transport is approximated numerically by

$$\frac{\mathrm{d}z_i}{\mathrm{d}t} = -\frac{k_d}{\Lambda_i} \sum_{j=1}^{n} \lambda_{ij} \frac{(z_i - z_j)}{L_{ij}}, \tag{4}$$

where n is the number of nodes adjacent to node i, L_{ij} is the length of the triangle edge connecting nodes i and j,

and λ_{ij} is the width of the shared Voronoi cell edge between nodes i and j. In the CHILD model (Tucker et al., 1997), this system of equations is solved using a simple forward-difference method with an adaptive time-stepping scheme. Note that because diffusive mass exchange takes place along triangle edges, the equation can be solved efficiently by first computing the mass exchange along each triangle edge (or equivalently, across each Voronoi polygon face), then updating the node elevations accordingly. This numerical algorithm performs well when compared with analytical solutions (Fig. 8).

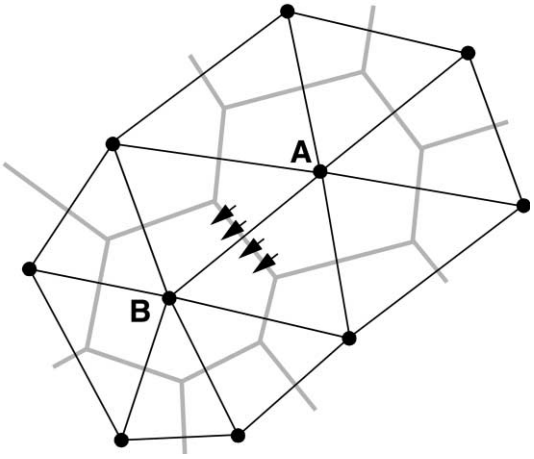

Fig. 7. Illustration of mass flux across Voronoi polygon face between two adjacent nodes. Mass flux per unit contour width is calculated based on gradient of triangle edge connecting A and B, and total flux is obtained by multiplying by width of shared Voronoi polygon edge.

Similarly, 2D groundwater flow can be approximated numerically using an expression of the following form:

$$\frac{\mathrm{d}h_i}{\mathrm{d}t} = -\frac{1}{A_i}\sum_{j=1}^{n}\lambda_{ij}\tfrac{1}{2}(T(h_i)+T(h_j))\frac{(h_i-h_j)}{L_{ij}}, \tag{5}$$

where h_i is the groundwater table elevation at node i and $T(h_i)$ is transmissivity (in this example, transmissivity is assumed to vary with water table height, with the average transmissivity between nodes i and j used to compute the flux between them). Fig. 9 compares the equilibrium water table height as computed from a simplified version of Eq. (5) with an analytical solution, assuming a linear hillslope geometry.

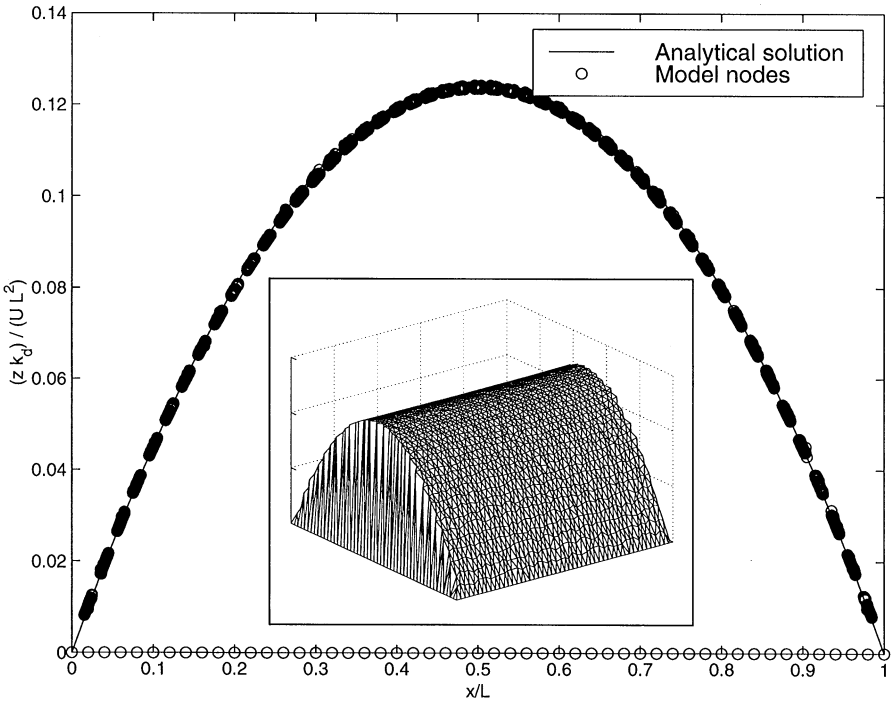

Fig. 8. Numerical solution to hillslope diffusion equation (Eq. (4)) under constant rate of baselevel fall on 5000-triangle TIN (inset), compared with analytical solution. TIN represents rectangular domain with two fixed boundaries. Open circles show nodes in TIN viewed from side. Solid line shows analytical solution in 1D. (Circles along x-axis are boundary points; y-axis is nondimensional elevation with elevation, z, scaled by hillslope length, L, rate of baselevel fall at boundaries, U, and diffusivity constant, k_d.)

6. Examples

We have applied these data structures and algorithms in simulation models of long-term landscape evolution (Tucker et al., 1997) and of catchment rainfall-runoff. Fig. 10 shows a simulated drainage basin responding to a period of rapid baselevel lowering at the outlet. Basin evolution is driven by a sequence of randomly generated storms, with fluvial erosion and sediment transport modeled on the basis of surface gradient and total discharge at each point (Tucker and Bras, 2000).

Fig. 11 illustrates a hydrologic application of the same concepts. The figure shows simulated depth to the water table in a small catchment in Kansas in response to spatially uniform recharge. Water table depth at each point is computed using Eq. (5), assuming an exponential decrease in saturated hydraulic conductivity with depth. In both of these examples, the TIN framework is implemented in C++, with each data element (nodes, triangles, and directed edges) encapsulated within a class. The hydrologic model and the landscape evolution model (Fig. 10) share the same code for mesh handling and drainage network delineation, which highlights the advantages of a modular, object-oriented approach in terms of code reusability.

The data structures and algorithms we have outlined are also well suited to applications involving dynamic remeshing in response to changing surface morphology. Fig. 12 illustrates output from a model that combines vertical erosion/deposition with lateral erosion associated with river meandering. Such adaptive remeshing strategies make it possible to explore a range of geologic problems that were heretofore inaccessible to modeling, such as horizontal tectonic deformation. Strategies for adaptive remeshing are discussed by Braun and Sambridge (1997) and Lancaster (1998).

7. Summary and conclusions

We have presented a set of data structures and associated algorithms that facilitate TIN-based terrain modeling, with an emphasis on geologic and hydrologic applications. The framework is particularly well suited to applications involving dynamic changes in surface morphology and, more generally, to applications in which nodes (as opposed to triangles) are used as the basic computational elements. The data structure provides an efficient means of storing both a Delaunay triangulation and its corresponding Voronoi diagram.

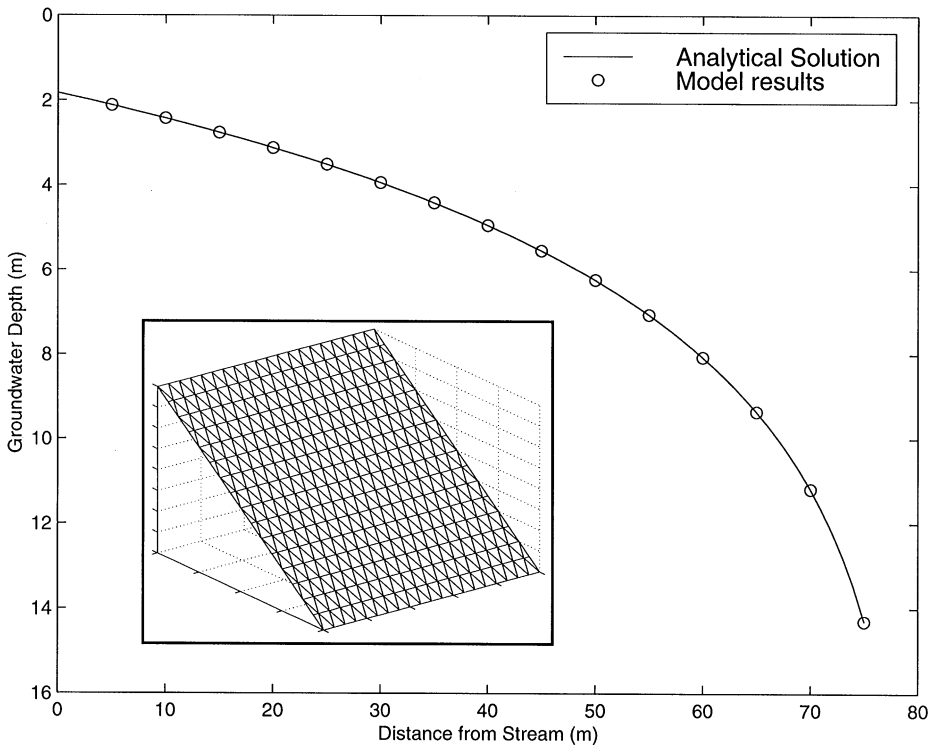

Fig. 9. Numerical solution to simplified 2D groundwater flow equation (Eq. (5)) on straight hillslope (inset), compared with 1D analytical solution. Symbols show nodes in TIN viewed from side. Solid line shows analytical solution in 1D.

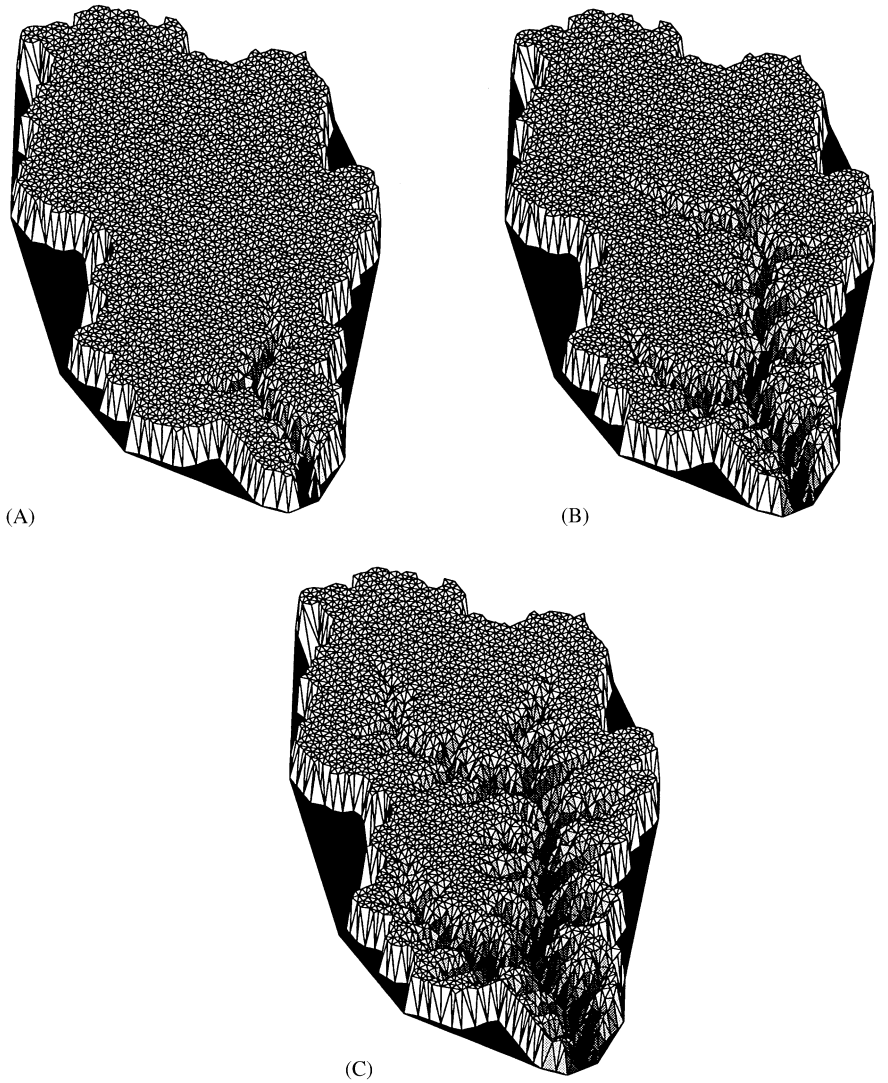

Fig. 10. Example landscape evolution simulation using CHILD model. Here, gully erosion into nearly flat plateau has been stimulated by sharp drop in elevation at outlet point.

By using Voronoi polygons rather than triangles as the computational elements, the problem of distinguishing between flow within triangles and flow between triangles is avoided. Voronoi polygons also provide a natural basis for solving diffusion-like equations numerically, using Voronoi polygon edges to approximation the effective contour width between each pair of adjacent nodes.

The object-based data structure simplifies bookkeeping. Each node points to a single directed edge (or spoke), which then points to its counterclockwise neighbor, thus fully describing the connectivity between nodes and triangle edges. The data structure also contains sufficient information to provide a complete representation of Voronoi geometry.

Based on this framework, we describe a simple flow-routing method. The algorithm is analogous to the most commonly used grid-based flow routing method, and does not account for flow divergence on convex landscape elements (but could easily be generalized to do so). We also present a new algorithm for identifying closed depressions within a drainage basin, and resolving drainage for those depressions without altering the topography.

The dual edge data structure is object-oriented in the sense that data associated with each of the three mesh elements (nodes, triangles, and directed edges) are grouped together within three data classes. While in principle this data structure could be represented using traditional procedural programming (for example, as a

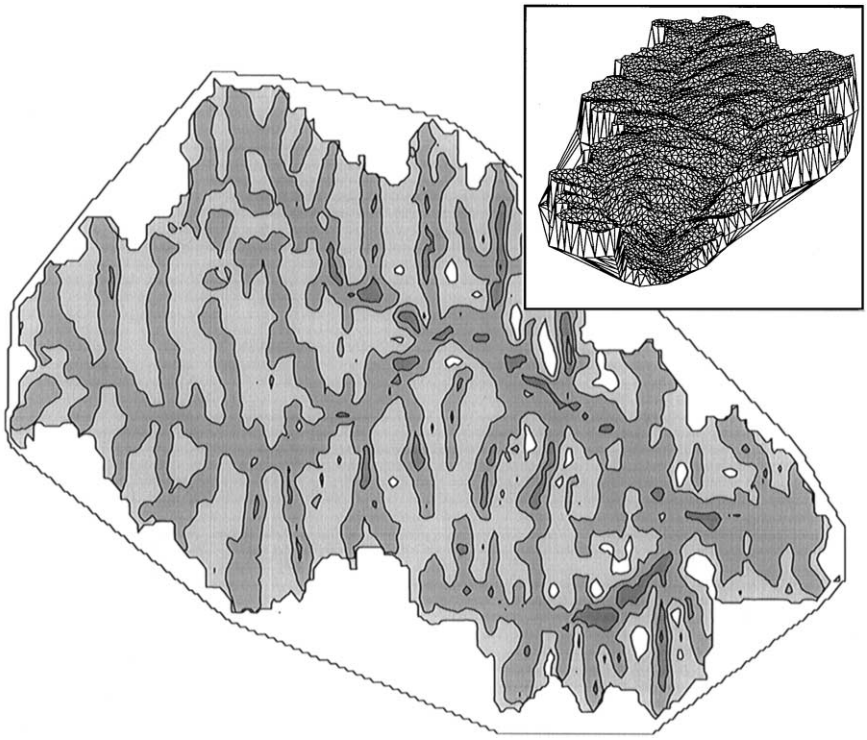

Fig. 11. Contour plot showing simulated depth to water table under constant recharge, Forsyth Creek catchment, Fort Riley, Kansas, using TRIBS model. Darker colors correspond to shallower water table. Inset shows catchment topography represented by TIN mesh.

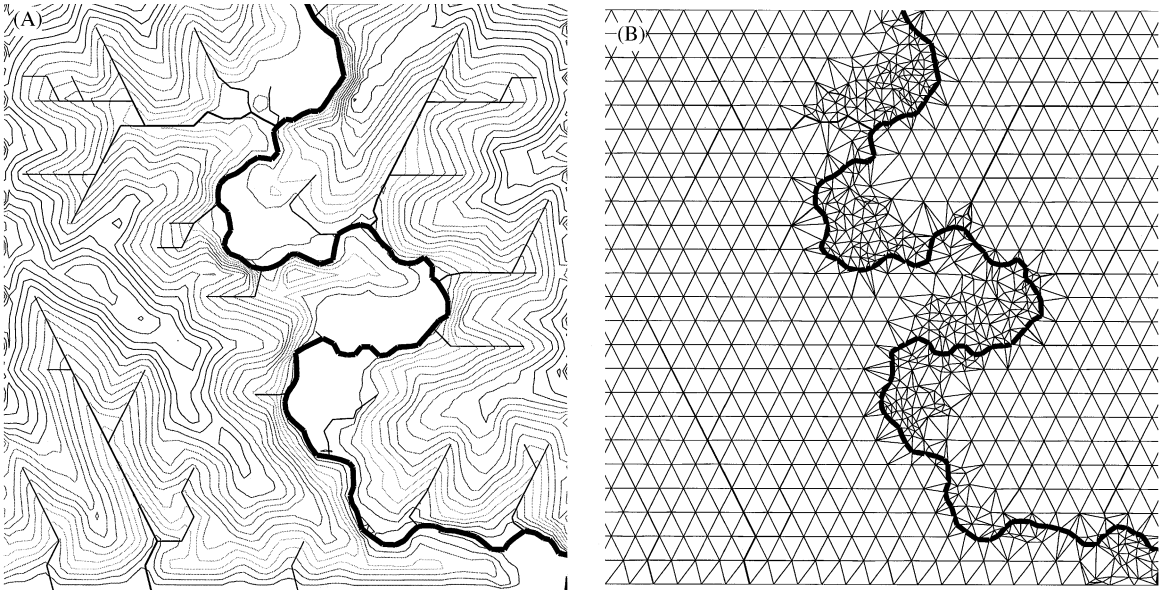

Fig. 12. Simulation of 3D floodplain development by lateral stream migration (meandering). Adaptive meshing strategy is used to dynamically move, add, and delete points in response to lateral movement along main channel. (A) Contour map of simulation. (B) Triangulation, showing densification of mesh in response to stream migration. Straight channel segments are artifacts of low mesh resolution outside of meander belt.

series of arrays), an object-oriented implementation offers several advantages. By grouping together data and functionality for each terrain element, it becomes possible to isolate the mesh implementation from the calculations that are performed on the mesh. This type of strategy enhances modularity and portability, and has the potential to reduce software development time. Furthermore, the inheritance capabilities of object-oriented programming languages such as C++ make it possible for different applications to inherit basic TIN functionality while adding application-specific capabilities as needed.

Acknowledgements

This work was supported by the US Army Corps of Engineers Construction Engineering Research Center (DACA88-95-C-0017), by the Army Research Office (DAAH04-95-1-0181), and by the National Aeronautics and Space Administration. We are grateful to Judy Ehlen and Russell Harmon for convening the GeoComputation conference that led to this volume, to Don Knuth for steering us toward relevant computational geometry literature, to Jean Braun for discussion of finite-volume methods, and to an anonymous reviewer for helpful comments.

Appendix A. Pseudo-code for Lake Fill algorithm

The Lake Fill algorithm uses a flag to indicate whether a node is a normal self-draining node (Unflooded), a pit for which an outlet has yet to be resolved (Pit), a flooded node that is part of the "lake" currently being computed (CurrentLake), or a flooded node that is part of a lake identified on a previous iteration (Flooded). The algorithm assumes that initially all nodes have already been flagged as either Unflooded or Pit by the drainage direction finding routine.

```
FOR each node pitNode flagged as a pit DO
  Place pitNode on "lake list" (list of flooded nodes) as 1st item
  foundOutlet:=FALSE
  DO
    // Start by finding the lowest node on the perimeter of the lake
    lowestNode:=first node on lake list
    lowestElev:=arbitrarily large number
    FOR each node curnode on lake list DO
      FOR each neighboring node nbr DO
        IF nbr not flooded & not a closed boundary THEN
          IF nbr lower than lowestElev THEN
            lowestNode:=nbr
            lowestElev:=nbr's elevation
          ENDIF
        ELSE IF nbr is flagged as a Pit or a Flooded node THEN
          Add nbr to lake list and flag it as CurrentLake
        ENDIF
      ENDFOR
    ENDFOR
    // Check to see whether the lowest node is a valid outlet
    IF lowestNode is an open boundary THEN foundOutlet:=TRUE
    ELSE
      IF lowestNode has a neighbor that is (a) lower than itself, (b) Is not flagged as CurrentLake, (c) is not a
      closed boundary, and (d) is not part of a previous lake that has an outlet higher than lowestNode, THEN
        lowestNode is the outlet — set its drainage path accordingly
        foundOutlet:=TRUE
      ELSE
        Add lowestNode to the lake list and flag it as CurrentLake
      ENDIF
    ENDIF
  WHILE NOT foundOutlet
  // Resolve drainage directions for lake nodes by (a) finding the shortest
  // path to the outlet for each or (b) pointing each directly toward the outlet.
  // A lake ID number can also be assigned to each node for further handling
  // (e.g., application of a lake-routing hydrologic model)
ENDFOR
```

Appendix B. Algorithm for resolving flow directions within a closed depression

The algorithm shown below resolves drainage directions within a closed depression after an outlet point has been identified. The algorithm attempts to find the shortest path toward the outlet by iteratively scanning the neighbors of each node. A node is assigned to flow toward the closest of any neighbors whose drainage has already been resolved. The process continues until a drainage direction has been assigned to every node. The net effect of this procedure is that drainage is resolved sequentially in an "upstream" direction, starting from the depression's outlet. (Note that this simple algorithm is intended only to ensure that flow paths are routed across depressions in a reasonable, consistent, and computationally efficient manner; it is not a foolproof solution to the shortest-path problem).

```
Flag outlet node as OutletFound
REPEAT
  done:=TRUE // assume done until proven otherwise
  FOR each node on lake list DO
    IF node isn't flagged OutletFound, look at its neighbors:
      done:=FALSE
      minDist:=arbitrarily large number
      FOR each neighboring node nbr DO
        IF nbr flagged as OutletFound
          dist:=nbr.distToOutlet + dist from node to nbr
          IF dist < minDist DO
            minDist:=dist
            Assign node to flow toward nbr
          ENDIF
          Flag node as TentativeOutletFound
        ENDIF
      ENDFOR
    ENDIF
  ENDFOR
  FOR each node on lake list DO
    IF node flagged as TentativeOutletFound, flag it OutletFound
  ENDFOR
UNTIL done
```

References

Anderson, R.S., 1994. Evolution of the Santa Cruz Mountains, California, through tectonic growth and geomorphic decay. Journal of Geophysical Research 99, 20161–20179.

Band, L.E., 1989. Spatial aggregation of complex terrain. Geographical Analysis 21, 279–293.

Braun, J., Sambridge, M., 1997. Modelling landscape evolution on geological time scales: a new method based on irregular spatial discretization. Basin Research 9, 27–52.

Costa-Cabral, M., Burges, S.J., 1994. Digital Elevation Model Networks (DEMON): a model of flow over hillslopes for computation of contributing and dispersal areas. Water Resources Research 30 (6), 1681–1692.

Culling, W.E.H., 1960. Analytical theory of erosion. Journal of Geology 68, 336–344.

Dietrich, W.E., Dunne, T., 1993. The channel head. In: Beven, K., Kirkby, M.J. (Eds.), Channel Network Hydrology. Wiley, New York, pp. 175–219.

Du, C., 1996. An algorithm for automatic Delaunay triangulation of arbitrary planar domains. Advances in Engineering Software 27, 21–26.

Freeman, T.G., 1991. Calculating catchment area with divergent flow based on a regular grid. Computers and Geosciences 17, 413–422.

Gandoy-Bernasconi, W., Palacios-Velez, O., 1990. Automatic cascade numbering of unit elements in distributed hydrological models. Journal of Hydrology 112, 375–393.

Garrote, L., Bras, R.L., 1995. A distributed model for real-time flood forecasting using digital elevation models. Journal of Hydrology 167, 279–306.

Guibas, L., Stolfi, J., 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics 4 (2), 74–123.

Howard, A.D., 1994. A detchment limited model of drainage basin evolution. Water Resources Research 30, 2261–2285.

Jackson, T.H., Tarboton, D.G., Cooley, K.R., 1996. A spatially-distributed hydrologic model for a small arid

mountain watershed. Working Paper WP-96-HWR-DGT/002, Utah Water Research Laboratory, Logan, UT.

Jenson, S.K., Domingue, J.O., 1988. Extracting topographic structure from digital elevation data for geographic information system analysis. Photogrammetric Engineering and Remote Sensing 54, 1593–1600.

Johnson, D.D., Beaumont, C., 1995. Preliminary results from a planform kinematic model of orogen evolution, surface processes and the development of clastic foreland basin stratigraphy. In: Dorobek, S.L., Ross, G.M. (Eds.), Stratigraphic Evolution of Foreland Basins. SEPM Special Publication, Vol. 52, pp. 3–24.

Jones, N.L., Wright, S.G., Maidment, D.R., 1990. Watershed delineation with triangle-based terrain models. Journal of Hydraulic Engineering 116, 1232–1251.

Julien, P.Y., Saghafian, B., Ogden, F.L., 1995. Raster-based hydrologic modeling of spatially-varied surface runoff. Water Resources Bulletin 31 (3), 523–536.

Knuth, D.E., 1992. Axioms and Hulls. Lecture Notes in Computer Science, Vol. 606. Springer, New York, 109pp.

Kooi, H., Beaumont, C., 1996. Large-scale geomorphology: classical concepts reconciled and integrated with contemporary ideas via a surface processes model. Journal of Geophysical Research 101 (B2), 3361–3386.

Laflen, J.M., Elliot, W.J., Flanagan, D.C., Meyer, C.R., Nearing, M.A., 1997. WEPP-predicting water erosion using a process-based model. Journal of Soil and Water Conservation 52, 96–102.

Lancaster, S.L., 1998. A nonlinear river meander model and its incorporation in a landscape evolution model. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 177pp.

Mackay, D.S., Band, L.E., 1998. Extraction and representation of nested catchment areas from digital elevation models in lake-dominated topography. Water Resources Research 34, 897–901.

Mitas, L., Mitasova, H., 1998. Distributed soil erosion simulation for effective erosion prevention. Water Resources Research 34, 505–516.

Miyamoto, H., Sasaki, S., 1997. Simulating lava flows by an improved cellular automata method. Computers & Geosciences 23, 283–292.

Montgomery, D.R., Dietrich, W.E., 1989. Source areas, drainage density, and channel initiation. Water Resources Research 25 (8), 1907–1918.

Montgomery, D.R., Dietrich, W.E., 1994. A physically-based model for the topographic control on shallow landsliding. Water Resources Research 30 (4), 1153–1171.

Moore, I.D., O'Loughlin, E.M., Burch, G.J., 1988. A contour-based topographic model for hydrological and ecological applications. Earth Surface Processes and Landforms 13, 305–320.

Nelson, E.J., Jones, N.L., Miller, A.W., 1994. Algorithm for precise drainage-basin delineation. Journal of Hydraulic Engineering 120, 298–312.

O'Callaghan, J.F., Mark, D.M., 1984. The extraction of drainage networks from digital elevation data. Computer Vision, Graphics, and Image Processing 28, 323–344.

Palacios-Velez, O.L., Cuevas-Renaud, B., 1986. Automated river-course, ridge and basin delineation from digital elevation data. Journal of Hydrology 86, 299–314.

Peyret, R., Taylor, T.D., 1983, Computational Methods for Fluid Flow. Springer, New York, 358pp.

Quinn, P., Beven, K.J., Chevallier, P., Planchon, O., 1991. The prediction of hillslope flow paths for distributed hydrological modeling using digital terrain models. Hydrological Processes 5, 59–80.

Roering, J.J., Kirchner, J.W., Dietrich, W.E., 1999. Evidence for nonlinear, diffusive sediment transport on hillslopes and implications for landscape morphology. Water Resources Research 35 (3), 853–870.

Sambridge, M., Braun, J., McQueen, H., 1995. Geophysical parameterization and interpolation of irregular data using natural neighbors. Geophysical Journal International 122, 837–857.

Slingerland, R.L., Harbaugh, J.W., Furlong, K.P., 1993. Simulating Clastic Sedimentary Basins. Prentice-Hall, New York, 220pp.

Sloan, S.W., 1987. A fast algorithm for constructing Delaunay triangulations in the plane. Advances in Engineering Software 9 (1), 34–55.

Sugihara, K., Iri, M., 1994. A robust topology-oriented incremental algorithm for Voronoi diagrams. International Journal of Computational Geometry and Applications 4 (2), 179–228.

Tarboton, D.G., 1997. A new method for the determination of flow directions and contributing areas in grid digital elevation models. Water Resources Research 33 (2), 309–319.

Tucker, G.E., Bras, R.L., 1998. Hillslope processes, drainage density, and landscape morphology. Water Resources Research 34, 2751–2764.

Tucker, G.E., Bras, R.L., 2000. A stochastic approach to modeling the role of rainfall variability in drainage basin evolution. Water Resources Research 36 (7), 1953–1964.

Tucker, G.E., Gasparini, N.M, Lancaster, S.L., Bras, R.L., 1997. An integrated hillslope and channel evolution model as an investigation and prediction tool. Technical Report, prepared for U.S. Army Corps of Engineers, 130pp.

Tucker, G.E., Slingerland, R.L., 1994. Erosional dynamics, flexural isostasy, and long-lived escarpments: a numerical modeling study. Journal of Geophysical Research 99, 12229–12243.

Tucker, G.E., Slingerland, R.L., 1996. Predicting sediment flux from fold and thrust belts. Basin Research 8, 329–349.

Tucker, G.E., Slingerland, R.L., 1997. Drainage basin response to climate change. Water Resources Research 33 (8), 2031–2047.

Versteeg, H.K., Malalasekera, W., 1995. An Introduction To Computational Fluid Dynamics; The Finite Volume Method. Longman, New York, 257pp.

Watson, D.F., Philip, G.M., 1984. Systematic triangulations. Computer Vision, Graphics, and Image Processing 26, 217–223.

Willgoose, G.R., Bras, R.L., Rodriguez-Iturbe, I., 1991. A physically based coupled network growth and hillslope evolution model, 1, theory. Water Resources Research 27, 1671–1684.