ELSEVIER

Catena 46 (2001) 159–176

CATENA

www.elsevier.com/locate/catena

A fast, simple and versatile algorithm to fill the depressions of digital elevation models

Olivier Planchon [a,*], Frédéric Darboux [b,c,1]

[a] *Institut de Recherche pour le Développement — IRD, BP 1386, Dakar, Senegal*
[b] *Géosciences – Rennes, Campus de Beaulieu, 35042 Rennes Cedex, France*
[c] *National Soil Erosion Research Laboratory, 1196 SOIL Building, Purdue University, West Lafayette, IN 47907-1196, USA*

Abstract

The usual numerical methods for removing the depressions of a Digital Elevation Model (DEM) gradually fill the depressions and merge the embedded ones. These methods are complex to implement and need large computation time, particularly when the DEM contains a high proportion of random noise. A new method is presented here. It is innovative because, instead of gradually filling the depressions, it first inundates the surface with a thick layer of water and then removes the excess water. The algorithm is simple to understand and to implement, requiring only a few tens of code lines. It is much faster than usual algorithms. Moreover, this method is versatile: depressions can be replaced with a surface either strictly horizontal, or slightly sloping. The first option is used for the calculation of depression storage capacity and the second one for drainage network extraction. The method is fully detailed and a pseudo-code is provided. Its practical computation time, evaluated on generated fractal surfaces, is asymptotically proportional to $N^{1.2}$ where N is the number of grid points. The theoretical computation time is asymptotically proportional to $N^{1.5}$ in all cases, with the exception of some exotic ones with no practical interest. By contrast, existing methods have a computation time asymptotically proportional to N^2. Applications are done for both generated and measured surfaces with 256 cells to 6.2 million cells.

Keywords: Digital elevation model; Soil roughness; Topography; Depression-filling; Storage capacity; Drainage network

1. Introduction

Soil surface roughness is now recognized as a key factor in erosion processes (Favis-Mortlock, 1998; Helming et al., 1998). It is involved in the study of most

* Corresponding author. Fax: +221-820-43-07.
E-mail addresses: Olivier.Planchon@ird.sn (O. Planchon), darboux@ecn.purdue.edu (F. Darboux).
[1] Fax: +1-765-494-5948.

0341-8162/01/$ - see front matter
PII: S0341-8162(01)00164-3

processes occurring during rain events, such as infiltration, runoff and crust formation. Among all of the studied characteristics of soil surface morphology, depression storage capacity (DSC) is the subject of specific research (Huang and Bradford, 1990) and is one of the most used surface properties to characterise microrelief.

Some attempts at direct measurement of DSC were performed by making the surface impervious using bitumen (Langford and Turner, 1972), polyester resin (Gayle and Skaggs, 1978) or a plastic film (Mwendera and Feyen, 1992). With the development of automatic measurement methods of soil roughness, such as laser scanners (Huang et al., 1988; Bertuzzi et al., 1990) and stereometric processing (Bruneau and Gascuel-Odoux, 1990; Helming et al., 1992), nowadays, most DSCs are calculated from Digital Elevation Models ("DEM", i.e. a regular grid of altitudes stored in a digital format).

To measure the DSC of a surface, depressions can be delineated manually on the DEMs (Sneddon and Chapman, 1989; Bruneau and Gascuel-Odoux, 1990), but such a method can only be used if depressions are few. In fact, most storage capacity estimations are computed automatically by using algorithms dedicated to depression delineation.

The algorithms usually referred for soil roughness data analysis are all based on a two-stage principle: (1) to identify local minima and (2) to fill them from the bottom to the top by exploring the neighbourhood of each minima to find their outlets (Moore and Larson, 1979; Ullah and Dickinson, 1979; Onstad, 1984). Practically, these two stages are composed of seven to eight steps. These algorithms are iterative and able to deal with embedded depressions. They essentially differ by the number of neighbours in use (four or eight) and the method of neighbourhood exploration. For practical purposes, they are either directly implemented by authors (Hansen et al., 1999) or slightly modified (Huang and Bradford, 1990).

Depression-filling methods are also routinely used in geomorphology to correct DEMs covering surfaces typically greater than 100 km^2. At this scale, depressions are usually considered as erroneous data and have to be removed before further processing. Numerous methods were proposed (Marks et al., 1984; Jenson and Domingue, 1988; Martz and de Jong, 1988). These authors did not demonstrate that their algorithms converged to the solution for all cases and they did not evaluate the time-complexity of the algorithm (see Section 2 for a definition). This was most probably because DEMs were relatively small in the 1980s. Nowadays, DEMs typically cover several square meters with a millimeter resolution grid (Helming et al., 1998, Darboux et al., 2001), leading to millions of cells. With such DEMs, the time-complexity of depression-filling algorithms is becoming a more important issue.

Among all of the numerical methods, the algorithm described by Jenson and Domingue (1988) seems to be the one with the widest use (Zhang and Montgomery, 1994; Gyasi-Agyei et al., 1995; Tarboton, 1997). It is also the method implemented in the well-known geographic information system ARC/INFO (ESRI, 1999). The algorithm of Jenson and Domingue (1988) gradually fills depressions and so does not really differ from the ones previously cited. These authors give a detailed description of their method. This was not done by Moore and Larson (1979) and Ullah and Dickinson (1979), especially for the sensitive stage where embedded depressions are merged together.

Jenson and Domingue (1988) asserted that their method is fast. Indeed, their algorithm is perfect for a DEM representing a large topography with large cells, which was the typical DEM their method was dedicated to. However, it is slow for DEMs with a large random component, as encountered in studies of soil surface roughness. The proportion of pit-cells is indeed much higher at this scale. Furthermore, pit-cells are almost randomly scattered in DEMs representing soil surface, so that the number of depressions is in $\boldsymbol{O}(N)$ (see definition in Section 2) for a given cell size.

To overcome this drawback, Jenson and Domingue (1988) recommended separately treating the single-cell depressions. This speeds-up the algorithm by decreasing the number of depressions remaining to fill, but this does not really change the time-complexity of the algorithm because the number of two-cell depressions is still in $\boldsymbol{O}(N)$. Like other filling methods, the one of Jenson and Domingue (1988) includes a key-stage treating each depression sequentially. This stage consists of merging the embedded depressions. To merge two depressions, the N cells of the DEM need to be scanned in order to re-label the merged drained areas. This stage is the most costly part of the whole procedure, and so, determines the overall time-complexity of the algorithm. Because the number of depressions is in $\boldsymbol{O}(N)$ and the number of cells to scan for each depression is also in $\boldsymbol{O}(N)$, the overall time-complexity of their algorithm is in $\boldsymbol{O}(N^2)$. Such quadratic algorithms are said to be slow and unusable for large data sets (Sedgewick, 1990). Moreover, the algorithm of Jenson and Domingue (1988) is complex to implement: seven basic stages are required, some including complex subroutines.

Furthermore, no published algorithm used to fill depressions is both simple and fast, and is also able to generate surfaces that are either flat (for calculating DSC) or with a determined minimal slope (for removing unwanted depressions in order to build a drainage network). The present article describes such a method. The proposed algorithm involves only two stages. The first stage inundates all the DEM with a thick layer of water. The second stage drains the excess water. Moran and Vézina (1993) suggested such a method in general terms, but it was neither detailed nor translated into an algorithm.

After the definition of the surface properties, an in-depth analysis is made for two multipurpose algorithms, which are the cornerstones of the methods studied in this article: scanning a grid and exploring a tree. Then, the new method is presented and two implementations are proposed for draining the excess water. The first one is very simple to implement and has a time-complexity in $\boldsymbol{O}(N^{1.5})$, better than all the published methods that are in $\boldsymbol{O}(N^2)$. The second one, fully optimised, appears to be close to $\boldsymbol{O}(N)$ for natural surfaces.

2. Definitions and notations

The following definitions and notations are used in the paper.

Z	Spatial distribution of altitude for the initial surface, i.e. the DEM to process.

Wf	Spatial distribution of altitude for the final surface. This is the surface to find.
W	Spatial distribution of altitude for a transient surface converging to Wf.
N, nR, nC	Numbers of grid cells, rows and columns of the DEM, respectively.
c, n	c ('center') is a cell; n ('neighbour') is one of its neighbours.
time-complexity	The time-complexity of an algorithm is a function of N. It represents how the computation cost increases with N. The time-complexity is an intrinsic property of the algorithm and does not depend on the details of implementation (language, compiler, machine, etc.). This requires basing the analysis on very large values of N.
$\boldsymbol{O}(f(N))$	Let's consider a function $g(N)$ to qualify (for example the time-complexity), and another function $f(N)$ qualifying g. The function $g(N)$ is said to be in $\boldsymbol{O}(f(N))$ if there exist two constants c_0 and N_0 such that $g(N) < c_0\ f(N)$ for all $N > N_0$ (Sedgewick, 1990). This mathematical artefact states that, from a certain size, g does not increase faster than f.

3. Basic algorithmic tools

3.1. Dependence graph

Considering a transient surface W, two kinds of cells can be defined: the cells for which the final altitude is already identified (known cells) and the cells for which the final altitude remains to find (unknown cells). The solution to the problem, if it exists, propagates progressively from known cells to unknown cells. Neighbouring cells are linked through dependence links like "have to be calculated before". All these links are part of a graph. We will refer to this graph using the following definitions.

- Spring cell: a cell that does not depend on any other cell. Its final solution is known from the beginning.
- Sink cell: a cell with no other cell depending on it.
- Dependence graph of a cell: the graph linking a cell to its neighbours previously calculated and needed for its own calculation, and so on, from cell to cell up to spring cells. As a standard, the links are oriented from the first calculated cells to the last calculated ones.
- Upward and downward: these qualifiers refer to an hypothetical flow in the graph that will go in the link direction, i.e. from the spring cells, known since the beginning of the procedure, to the sink cells, the last ones to be calculated.
- Seed cell: a cell that is used to generate a dependence graph.
- Distance between an upward cell and a downward cell: distance of the longest path linking the two cells when the graph is travelled following the links.

- Depth of a dependence graph: for a given dependence graph, the largest distance from a spring cell to a seed cell.
- Depth of a cell: depth of the dependence graph when this cell is used as a seed.

A dependence graph characterises the initial data and the order in which these data are evaluated. It characterises the solving method too because an alteration in the method can lead to a different way of propagating the solution through the mesh, and so, to a different dependence graph.

In the following sections, the formalism of graph theory is avoided when possible. Doing so, the demonstration will appear clearer even if less rigorous.

3.2. Iterative scan tool

In order to calculate the cells of the dependence graph in the right order, the simplest tool to implement is an iterative scan of the whole DEM. Jenson and Domingue (1988) made an intensive use of this technique but did not analyse it in detail. We provide in this section the time-complexity analysis of the iterative scan. This leads us to propose an optimised scanning tool.

3.2.1. Time-complexity analysis

At least one cell must be calculated at each iteration; otherwise the method would not converge to a solution. If this calculable cell is unique, it is the deepest (otherwise another cell could be calculated before). By consequence, each pass decreases the depth of the deepest uncalculated cell. This demonstrates: (a) that the iterative method converges in all cases if a solution exists and, (b) that the maximal number of iterations that could be needed to reach the solution is the depth of the deepest cell in the dependence graph. In some exotic cases (like a conically shaped surface with a narrow channel following a downward helix) this depth is in $\boldsymbol{O}(N)$. The time-complexity of this worst-case is therefore in $\boldsymbol{O}(N^2)$. However, this worst-case never appears in practice and can be negated. For usual cases, the depth of the dependence graphs is only proportional to the perimeter of the DEM because the longest path in the graph behaves approximately the same way as the longest slope line. This leads to a time-complexity in $\boldsymbol{O}(N^{1.5})$.

The usual way to scan a mesh is an embedded double loop starting at the upper left, moving right, and then switching to the beginning of the next row. Assuming that a linear segment of the graph follows a column from the bottom to the top of the grid, each iteration will solve a single cell within this segment. Such a case needs a number of iterations equal to the number of cells in that segment. Conversely, a segment following the same column, but oriented from the top to the bottom, would be solved in a single pass.

Eight other scan directions can be identified. They start from one of the four corners and scan the DEM following rows or columns. Alternating these eight possibilities could drastically fasten the iterative methods: any linear segment in the dependence graph will be solved after a maximum of eight iterations whatever its length. The number of iterations will then depend more on the intrinsic complexity of the problem than on the surface size. This is of great interest for large DEMs.

3.2.2. Implementation

A scanning direction is defined with three pairs of values: the initial point (R0, C0), the shift (dR, dC) between neighbour points and the shift (fR, fC) when the DEM border is reached. To implement the alternate scan direction method, a function named Next_Cell(R,C,i) is called. This function alters the coordinates (R,C) considering the ith scanning direction (Table 1). Section 1 (lines 1 to 7) defines the six constant arrays containing the three characteristic couples of each scan direction. Section 2 is the active part of the code. Lines 9 and 10 perform the shift to the neighbour point. If the new coordinates are out of the DEM (line 11), the code of lines 12 and 13 shifts the point to the beginning of a new row or a new column. If these new coordinates are out of the DEM (line 14), this scan in the ith direction is achieved and False is returned.

The following conclusions can be drawn about the iterative technique:

- The iterative technique always converges if a solution exists.
- Assuming that each iteration is in $\boldsymbol{O}(N)$ in time, its time-complexity is in $\boldsymbol{O}(N^2)$ for the worst-case and in $\boldsymbol{O}(N^{1.5})$ for the usual ones.
- Alternating the scan direction after each iteration does not affect the worst-case properties but make it rarer. With this refinement, the iterative technique is certainly close to $\boldsymbol{O}(N)$ for usual cases.

Table 1
Function Next_Cell()
For the given cell (R,C) and the scan direction i, the function Next_Cell() calculates the coordinates of the next cell to consider.

Section 1—Definition of constants

1	Declare R0, C0, dR, dC, fR, fC As constant arrays of eight integers								
2	R0 = (	0,	nR − 1,	0,	nR − 1,	0,	nR − 1,	0,	nR − 1)
3	C0 = (	0,	nC − 1,	nC − 1,	0,	nC − 1,	0,	0,	nC − 1)
4	dR = (	0,	0,	1,	−1,	0,	0,	1,	−1)
5	dC = (	1,	−1,	0,	0,	−1,	1,	0,	0)
6	fR = (	1,	−1,	− nR + 1,	nR − 1,	1,	−1,	− nR + 1,	nR − 1)
7	fC = (	− nC + 1,	nC − 1,	−1,	1,	nC − 1,	− nC + 1,	1,	−1)

Section 2—Function implementation

```
 8  Function Next_Cell (R,C,i)
 9    R = R + dR[i]
10    C = C + dC[i]
11    If R < 0 or C < 0 or R ≥ nR or C ≥ nC Then
12      R = R + fR[i]
13      C = C + fC[i]
14      If R < 0 or C < 0 or R ≥ nR or C ≥ nC Then
15        Return False
16      End If
17    End If
18    Return True
19  End Function
```

3.3. Tree exploration tool

3.3.1. Time-complexity analysis

The tree exploration tool uses the dependence graph, beginning from one or several seed cells and following the dependence links from cell to cell. If the rules forbid exploring the same cell twice, the dependence graph becomes a tree that can be inspected using usual recursive techniques for tree exploration (Sedgewick, 1990): exploring a cell of the tree means visiting this cell and then, exploring the graphs of its neighbouring cells that have not already been visited. The procedure defined to perform this exploration is called Explore. Table 2 shows, in pseudo-code, the template of this procedure. Its time-complexity is in $\boldsymbol{O}(N)$ because a cell is explored only once and its exploration induces a maximum of eight comparisons. A sequence of calls to Explore for the N cells is still in $\boldsymbol{O}(N)$: in the worst-case, the number of external calls is equal to N. Because each internal call is done after the calculation of a cell and only if this cell was not calculated previously, the total number of internal calls is also equal to N in the worst-case. So, whatever the number of successive external calls, a total of only N internal calls will be performed in the worst-case.

3.3.2. Implementation

In some cases, the recursion depth of Explore is in $\boldsymbol{O}(N)$. This could lead to a stack overflow. A simple way to prevent this is to limit the recursion depth. Using an external stack would be a more sophisticated solution (see Sedgewick, 1990 for implementation). However, such a refinement is useless in our case because the tree exploration

Table 2
Algorithmic scheme of the recursive tree exploration method

	Section 1—Definition of constants and external variables
1	Declare depth As Integer = 0
2	Declare MAX_DEPTH As Constant = 2000

	Section 2—Procedure implementation	
3	Procedure Explore(c)	//c is the cell to explore
4	depth = depth + 1	
5	If depth > MAX_DEPTH Then	
6	Go to line 14	
7	End If	
8	For each neighbour n of c (in any order)	
9	If n exists and n was not previously visited	
10	Visit n	
11	Call Explore(n)	
12	End If	
13	End For	
14	depth = depth − 1	
15	End Procedure	

procedure is not intrinsically necessary, but is only useful to accelerate the procedure. "MAX_DEPTH" is a constant initialised to an acceptable value depending on the size of the program's stack (Table 2, line 1). A value of several thousands is most often suitable. The recursion depth is stored in the external variable "depth" (line 2). It is incremented at the beginning of the procedure (line 4) and decremented at its end (line 14). Line 5 checks this value and triggers the end of the tree exploration if the maximal depth is reached (line 6).

Lines 8 to 13 ensure the exploration of the neighbour cells. If the neighbour n exists and was not previously visited (line 9), it is visited (line 10). Then, Explore is internally called for n (line 11). The details of line 10 depend of the specific purpose of the tree exploration (label assignment, modification of altitudes, etc.).

4. The new filling method

4.1. Considerations about surface properties

In order to evaluate the DSC, the surface of the depressions has to be flat. In order to ensure a correct extraction of the drainage network, a depression surface with a slight slope is more practical. The proposed method is able to generate both kinds of surfaces.

A minimal positive difference of altitude ε is defined for each of the eight directions joining a cell to its neighbours. In practice, no more than two values are needed, one for direct directions and the other for diagonal directions. If the values of ε are not equal to zero, the proposed method will correct a DEM by removing depressions and avoiding flat surface development. That way, each cell will have a defined drainage direction and will be connected to the boundary following a strictly decreasing path. If all ε-values are set to zero, depression surfaces will be flat in the final state. Such a value of ε is used to calculate the DSC by subtracting Z to Wf.

Considering an initial surface Z, the final surface Wf is fully defined by the following three properties:

(A) $\mathrm{Wf} \geq Z$ everywhere.

(B) For each cell c of Wf, there is a path that leads to the boundary and that has a descent of ε or more from one cell to the next (ε being taken in accordance with the local direction of the path). Such a path will be referred to as an ε-descending path.

(C) Wf is the lowest surface allowed by properties (A) and (B).

Stated that way, the depression-filling problem remains very general.

4.2. Description of the new filling method

The new method involves two basic stages. First, the surface W is initialised with infinite altitudes except for the boundaries (Table 3). During the second stage, altitudes of the surface W are decreased iteratively, keeping properties (A) and (B) valid (Table 4). Step by step, the surface W will converge to the final surface Wf, meaning that

Table 3
Stage 1: Initialisation of the surface to infinite altitudes

1	For each cell c of the DEM (in any order)
2	If c is on the border Then
3	$W(c) = Z(c)$
4	Else
5	$W(c) =$ a_huge_number
6	End If
7	End For

property (C) will become valid too. Applying two operations to all the neighbours of all the cells ensures this convergence. Operation (1) treats cases where $W(c)$ can be set equal to $Z(c)$ while keeping an ε-descending path to at least one neighbour. If Operation (1) is applied, $W(c)$ reaches its minimal value, so $W(c) = \mathrm{Wf}(c)$. In consequence, the altitude of the cell c will not be modified anymore:

$$Z(c) \geq W(n) + \varepsilon(c,n) \Rightarrow W(c) = Z(c) \qquad \text{(Operation 1)}$$

Operation (2) deals with the opposite case. When $Z(c)$ is lower than $W(n)$, $W(c)$ can be decreased up to $W(n) + \varepsilon(c,n)$:

$$W(c) > W(n) + \varepsilon(c,n) > Z(c) \Rightarrow W(c) = W(n) + \varepsilon(c,n) \qquad \text{(Operation 2)}$$

Line 8 controls if a new iteration between lines 1 and 7 is needed. This leads W to converge to Wf, the final result. The demonstration of this convergence is based on the following reasoning.

The preservation of property (A) does not need to be demonstrated because its preservation is a direct consequence of Operations (1) and (2). So, the surface W can never become lower than the surface Z.

The preservation of property (B) can be demonstrated by recurrence. Let us suppose that the surface W has properties (A) and (B). We show that the algorithm preserves these properties when a cell c is compared with one of its neighbours n (Table 4, lines 3

Table 4
Stage 2: Removal of excess water

1	For each cell c of the DEM (in any order)
2	For each neighbour n of c (in any order)
3	Determine ε for the pair (c,n)
4	If possible, apply operation (1)
5	Else, try to apply operation (2)
6	End For
7	End For
8	If W was modified during this scan, Then
9	Go to line 1
10	End If

to 5). In other words, if properties (A) and (B) exist before Operations (1) and (2), they will persist. This demonstration includes the three following steps:

Step 1 demonstrates that if c is modified after comparison with n, the property (B) at n is not affected: Operations (1) and (2) are executed only if $W(c) > W(n) + \varepsilon(c,n)$. In such a case, there is no ε-descending path from n to c. In consequence, the ε-descending path from n to the boundary cannot include the cell c and a modification of the cell c does not affect the property (B) at n. So, property (B) is preserved at cell n.

Step 2 demonstrates that property (B) at c is preserved by its own modification: After a modification of c, whether $W(c) \geq W(n) + \varepsilon(c,n)$ (Operation (1)) or $W(c) = W(n) + \varepsilon(c,n)$ (Operation (2)). In both cases, the path from c to n is ε-descending and property (B) is preserved at n (this was demonstrated in Step 1). Conclusion: Operations (1) and (2) preserve property (B) for c.

Step 3 demonstrates that a modification of c resulting from the comparison with n does not affect the property (B) at any other neighbour i: if the path from i to c was ε-descending, it remains identical because the altitude of c is reduced by both Operations (1) and (2). Because the modification of c preserves its own ε-descending path, the neighbour i will also keep the property (B). If the path from i to c was not ε-descending, the modification of c does not affect the ε-descending of i (thanks to the recurrence hypothesis, the ε-descending path of i existed).

In conclusion, the modification of the cell c after comparison with one of its neighbour n does not affect the property (B) at cell c and at its neighbours. Because the whole method is simply a succession of such modifications for all the cells and theirs neighbours, it preserves the property (B).

We now have to demonstrate that the method converges to property (C): W is minimum at the end of the procedure because the procedure stops when, for each cell c with $W(c) > Z(c)$, the difference of altitudes between c and its neighbours is lower or equal to ε (c,n). If $W(c)$ was further decreased, the difference of altitudes would become smaller than the minimal difference allowed with all its neighbours and there would not be an allowed path to the boundary. The cell c will appear as a depression and property (B) would not be valid anymore. To recover property (B), W would need to be decreased on the entire path leading to c, including the cell on the boundary. This is not possible because, on the boundary, W is equal to Z from the beginning and cannot be changed. We have therefore demonstrated that the method converges to property (C).

In conclusion, the method described at the beginning of this section converges in all cases to the final surface defined by the properties (A) to (C). The time-complexity is in $\boldsymbol{O}(N^{1.5})$ in all usual cases.

4.3. Two implementations of stage 2

The first stage only needs to scan the DEM once to set the surface W to its initial state. The second stage can be implemented several ways. Two of them are given in pseudo-code.

4.3.1. Direct implementation of the new filling method

Table 5 gives the pseudo-code of the direct implementation of the new filling method. As demonstrated in a previous section, its time-complexity is in $\boldsymbol{O}(N^{1.5})$. In

Table 5
Direct implementation of stage 2

```
1   something_done = False
2   For each cell c of the DEM (in any order) except its boundary
3    If W(c) > Z(c) Then
4     For each existing neighbour n of c (in any order)
5      If Z(c) ≥ W(n) + ε(c,n) Then                    //operation (1)
6       W(c) = Z(c)
7       something_done = True
8       Go to line 16
9      End If
10     If W(c) > W(n) + ε(c,n) Then                    //operation (2)
11      W(c) = W(n) + ε(c,n)
12      something_done = True
13     End If
14    End For
15   End If
16  End For
17  If something_done = True Then
18   Go to line 1
19  End If
```

consequence, this algorithm is very efficient for large DEMs as compared to other methods that gradually fill small depressions and are in $\boldsymbol{O}(N^2)$. This algorithm is also remarkable because of its conciseness, all the pseudo-code being written in 19 lines.

Table 6
Implementation of Operation (1) with tree exploration

Section 1—Definition of constants and external variables

```
1   Declare MAX_DEPTH As Constant = 2000
2   Declare depth as integer = 0
```

Section 2—Procedure implementation

```
3   Procedure Dry_upward_cell(c)                    //c is a dried cell to explore
4    depth = depth + 1
5    If depth > MAX_DEPTH Then
6     Go to line 16
7    End If
8    For each neighbour n of c (in any order)
9     If n exists And W(n) = a_huge_number (see Table 3, line 5) Then
10     If Z(n) ≥ W(c) + ε(c,n) Then                  //If operation (1) is applicable
11      W(n) = Z(n)                                  //operation (1) (was "Visit n")
12      Call Dry_upward_cell(n)                      //recursive call (was "Explore(n)")
13     End If
14    End If
15   End For
16  depth = depth − 1
17  End Procedure
```

4.3.2. Implementation with improved iterations and tree exploration

During Operation (1), if a cell is dried, then all the paths strictly upward from the initially dried cell could also be dried. In the direct implementation, this is done during the subsequent iterations. To avoid that, Operation (1) can be implemented with a tree exploration each time it is applied. This enables the size of the longest dependence path to be reduced. The pseudo-code of this tree exploration is figured out in Table 6. This is an application of the general algorithm displayed in Table 2.

Table 7 gives the pseudo-code for stage 2 with the following modifications compared with its direct implementation:

- Strictly upward paths from the border are first dried using tree exploration (lines 1 to 3)
- When a cell is dried, all the strictly upward paths are explored without waiting for the next scans (line 13). This reduces the depth of the dependence graph.

Table 7
Improved implementation of stage 2

Section 1—Explore all ascending paths from the border

```
1   For each cell c on the border (in any order)
2    Call Dry_upward_cell(c)
3   End For
```

Section 2—Iteratively scan the DEM

```
4   For scans = 1 to 8
5    R = R0(scan); C = C0(scan)
6    something_done = False
7    Do
8     If W(c) > Z(c) Then
9      For each existing neighbour n of c (in any order)
10      If Z(c) ≥ W(n) + ε(c,n) Then                    //operation (1)
11       W(c) = Z(c)
12       something_done = True
13       Call Dry_upward_cell(c)
14       Go to line 22
15      End If
16      If W(c) > W(n) + ε(c,n) Then                    //operation (2)
17       W(c) = W(n) + ε(c,n)
18       something_done = True
19      End If
20     End For
21    End If
22   Loop While Next_Cell(R, C, scan) = True
23   If something_done = False
24    Go to line 28
25   End If
26  End For
27  Go to line 4
28
```

- The iterative method is improved in order to alternate scan directions and so to reduce the depth of the dependence graph (lines 4, 5, 22, 26 and 27). Line 22 refers to the function Next_Cell described in Table 1.

Even with these improvements, the time-complexity of this method remains in $\boldsymbol{O}(N^{1.5})$ because this intrinsic property, defined for the worst-case, is not affected by implementation details. Nevertheless, for the usual cases, this implementation is expected to be significantly faster than the direct method because it reduces the length of dependence paths. It also remains remarkably easy to translate in any language with recursive capability.

5. Applications

5.1. Generated surfaces

The proposed algorithms were tested on generated surfaces with similar statistical properties. This enabled evaluating the efficiency of these algorithms depending on surface size and spatial correlation. For each surface type, a population of 100 samples were generated. Sizes ranged between 16×16 cells to 2048×2048 cells. Tests were performed on a desktop PC (Intel Pentium III, 500 MHz). The algorithms were directly implemented in C without further enhancement. In order to measure the efficiency of the proposed algorithms, the three following variables were computed and averaged on each population:

- The total number of scanned cells. It is equal to the number of iterations multiplied by the number of cells in the considered grid.
- The total number of comparisons between a point and its neighbours. The algorithms that minimise the number of comparisons required to find the final surface are more efficient.
- The execution time. Contrary to the previous variables, this one depends on compiler and computer specifications. The evolution of execution time with grid cell size gives valuable complementary information about algorithm efficiency.

5.1.1. Gaussian white noise

Gaussian white noise surfaces are very rough and contain numerous small depressions. They were generated using the algorithm “gasdev” of Press et al. (1992). Direct implementation of the iterative method gives acceptable results (Fig. 1 and Table 8). The number of scanned cells follows a power law with an exponent close to 1.5, in good agreement with the theoretical analysis. The evolution of the number of comparisons between a current cell and its neighbours follows a power law with an exponent significantly lower than two. The same observation is made for the execution time. As expected, the improved iterative procedure is more efficient than the direct implementa-

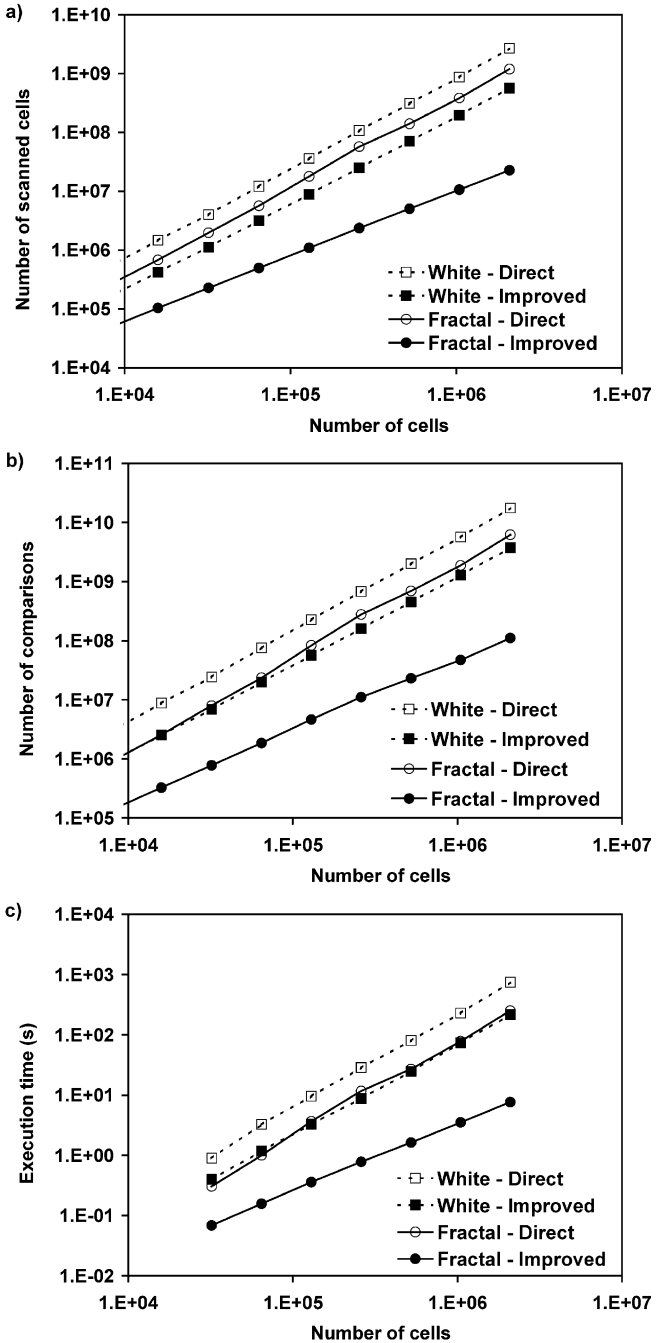

Fig. 1. Evolution of (a) the number of scanned cells, (b) the number of comparisons and (c) the execution time depending on DEM size, surface generation and implementation.

Table 8
Regression parameters for a power law $y = a \times x^b$

Conditions		Scanned cells		Comparisons		Execution time	
Surface	Implementation	b	R^2	b	R^2	b	R^2
White noise	direct	1.53	0.9999	1.57	0.9999	1.58	0.9992
	improved	1.50	0.9998	1.57	0.9996	1.50	0.9998
Fractal	direct	1.42	0.9995	1.47	0.9998	1.58	0.9972
	improved	1.14	0.9997	1.27	0.9985	1.12	0.9996

tion. It allows decreasing the number of scanned cells, the number of comparisons and the execution time.

5.1.2. Fractal surfaces

Surfaces with a fractal dimension equal to 2.2 are a good approximation of the morphology of Earth surface at topographic scale (Voss, 1988). Such surfaces were generated using the two-dimensional inverse Fourier method (Saupe, 1988; Press et al., 1992). These surfaces are spatially correlated and display large and embedded depressions. In fact, they display many more depressions than natural topographic surfaces. Therefore, they are an efficient test for depression-filling algorithms.

The direct implementation gives better results on these surfaces than on the Gaussian white noise surfaces, but the improved implementation is still the most efficient (Fig. 1 and Table 8). A large contrast is observed between direct and improved implementations. Because of the presence of a spatial correlation on such surfaces, the likelihood for a dried cell to have several neighbour cells with higher altitudes is larger than in the white noise case. This significantly increases the efficiency of the recursive procedure. Thanks to the combination of the recursive procedure and the switch of scan directions, the exponents are close to or lower than 1.2 for the three computed variables. This result is in good agreement with the theoretical analysis, which predicted that the exponent for natural surfaces should be lower than 1.5 and close to 1.

5.2. Measured surfaces

DEMs of soil surface were acquired using a laser scanner (Flanagan et al., 1995). Roughness measurements were performed on a soil box of 2.5×2.5 m^2. Grid size resolution was equal to 1 mm in both horizontal directions. A scan was performed on the initial soil surface and then after each rainfall application. The depression storage capacity decreases with the added rainfall amount. More details on the experimental design are given in Darboux et al. (2001).

All the DEMs had a similar size, around 2500×2500 cells. Such large DEMs are a good opportunity to test the efficiency of the two implementations of the new filling method (Table 9). The number of scanned cells, the number of comparisons and the execution time do not depend upon the depression storage capacity of the surface or the number of depressions. Such a result was expected because the algorithm does not fill

Table 9
Algorithm efficiency tests for a sequence of microtopographic Digital Elevation Models
Each DEM represents a surface area of 2.5×2.5 m with a 1-mm resolution grid.

	Before rain	After 1st rain	After 2nd rain	After 3rd rain	After 4th rain
Surface characteristics					
General slope (%)	5	5	5	5	5
Standard deviation (mm)	10.2	9.7	9.2	8.5	8.0
Semivariogram sill (mm^2)	89	65	58	42	34
Semivariogram range (mm)	47	52	48	47	49
Depression storage capacity (mm)	1.61	0.85	0.45	0.13	0.07
Puddle surface area (m^2/m^2)	0.34	0.29	0.23	0.16	0.14
Number of local minima	2.0×10^5	1.8×10^5	2.0×10^5	2.2×10^5	2.4×10^5
Number of filled puddles	1.2×10^5	0.9×10^5	1.3×10^5	1.8×10^5	2.0×10^5
Algorithm results					
Number of scanned cells					
Direct implementation	5.3×10^9	6.0×10^9	6.4×10^9	6.2×10^9	5.4×10^9
Improved implementation	9.7×10^7	7.9×10^7	1.2×10^8	9.8×10^7	7.9×10^7
Number of comparisons					
Direct implementation	4.3×10^{10}	5.1×10^{10}	5.2×10^{10}	4.9×10^{10}	3.6×10^{10}
Improved implementation	6.2×10^8	4.7×10^8	5.3×10^8	3.6×10^8	2.9×10^8
Execution time (s)					
Direct implementation	1589	1842	1927	1990	1432
Improved implementation	39.2	31.6	39.6	32.7	25.4

progressively the depressions. For all the surfaces and variables, the improved implementation is clearly the most efficient. While 1000 iterations (6×10^9 scanned cells) were needed using the direct implementation, only 15 iterations (10^7 scanned cells) were necessary to reach the final surface with the improved implementation. It leads to a tremendous contrast in execution time: half an hour with the direct implementation and only 30 s with the improved one.

6. Discussion and conclusions

Published algorithms for depression-filling have a time-complexity in $\boldsymbol{O}(N^2)$ in the worst-case and are close to this limit for DEMs containing a large amount of random noise. Therefore, these numerical methods are not efficient when dealing with large DEMs of soil roughness.

The method presented in this article is based on a radically different approach that first adds a thick layer of water over all the DEM and then drains excess water. Two algorithms were implemented. The first one needs a very short computer code and is in $\boldsymbol{O}(N^{1.5})$ for all studied cases. The second algorithm is more sophisticated but still very simple to implement. It is faster than the direct implementation, especially for surfaces close to natural ones. For this later case, its time-complexity is in $\boldsymbol{O}(N^{1.2})$. That means execution time is multiplied by five when the number of cells is multiplied by four. Such moderate increase of the execution time has to be compared with the multiplication by

eight for the direct implementation in $O(N^{1.5})$ and by 16 for algorithms in $O(N^2)$ considering the same variation in cell numbers. In consequence, depressions of a DEM with 2500 rows and columns can be filled in 30 s on a desktop PC.

Moreover, the presented method is versatile. It can be used for both DSC calculation, which needs depression surfaces to be exactly flat, and for drainage network computation, which needs to keep a small slope for each cell of the DEM. Such surfaces are a basic need for hydrologic modelling at topographic scales even if there is no standard, published and recognized method to calculate them.

References

Bertuzzi, P., Caussignac, J.M., Stengel, P., Morel, G., Lorendeau, J.Y., Pelloux, G., 1990. An automated, noncontact laser profile meter for measuring soil roughness in situ. Soil Science 149 (3), 169–178.

Bruneau, P., Gascuel-Odoux, C., 1990. A morphological assessment of soil microtopography using a digital elevation model on one square metre plots. Catena 17, 315–325.

Darboux, F., Davy, P., Gascuel-Odoux, C., Huang, C., 2001. Evolution of soil surface roughness and flowpath connectivity in overland flow experiments. In: Auzet, V., Poesen, J., Valentin, C. (Eds.), Soil Pattern as a Key Factor of Water and/or Wind Erosion. Catena, pp. 125–139.

ESRI, 1999. Technical Documentation of ARC, version 8.0.1. Environmental Systems Research Institute, Redlands, CA, USA.

Favis-Mortlock, D., 1998. A self-organizing dynamic systems approach to the simulation of rill initiation and development on hillslopes. Computers & Geosciences 24 (4), 353–372.

Flanagan, D.C., Huang, C., Norton, L.D., Parker, S.C., 1995. Laser scanner for erosion plot measurements. Transactions of the ASAE 38 (3), 703–710.

Gayle, G.A., Skaggs, R.W., 1978. Surface storage on bedded cultivated lands. Transactions of the ASAE 21 (1) 101–104, 109.

Gyasi-Agyei, Y., Willgoose, G., de Troch, F.P., 1995. Effects of vertical resolution and map scale of digital elevation models on geomorphological parameters used in hydrology. Hydrological Processes 9, 363–382.

Hansen, B., Schjønning, P., Sibbesen, E., 1999. Roughness indices for estimation of depression storage capacity of tilled soil surfaces. Soil & Tillage Research 52, 103–111.

Helming, K., Jeschke, W., Storl, J., 1992. Surface reconstruction and change detection for agricultural purposes by close range photogrammetry. In: Fritz, L.W., Lucas, J.R. (Eds.), International Archives of Photogrammetry and Remote Sensing. International Society for Photogrammetry and Remote Sensing. Committee of the XVII International Congress for Photogrammetry and Remote Sensing, Washington, DC, USA, pp. 610–617.

Helming, K., Römkens, M.J.M., Prasad, S.N., 1998. Surface roughness related processes of runoff and soil loss: a flume study. Soil Science Society of America Journal 62, 243–250.

Huang, C., Bradford, J.M., 1990. Portable laser scanner for measuring soil surface roughness. Soil Science Society of America Journal 54, 1402–1406.

Huang, C., White, I., Thwaite, E.G., Bendeli, A., 1988. A noncontact laser system for measuring soil surface topography. Soil Science Society of America Journal 52, 350–355.

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 (11), 1593–1600.

Langford, K.J., Turner, A.K., 1972. Effects of rain and depression storage on overland flow. Transactions of the Institution of Engineers, Australia 14 (2), 137–141.

Marks, D., Dozier, J., Frew, J., 1984. Automated basin delineation from digital elevation data. Geo-Processing 2, 299–311.

Martz, L.W., de Jong, E., 1988. CATCH: a fortran program for measuring catchment area from digital elevation models. Computers & Geosciences 14 (5), 627–640.

Moore, I.D., Larson, C.L., 1979. Estimating micro-relief surface storage from point data. Transactions of the ASAE 22 (5), 1073–1077.

Moran, C.J., Vézina, G., 1993. Visualizing soil surfaces and crop residues. IEEE Computer Graphics and Applications 13 (2), 40–47.

Mwendera, E.J., Feyen, J., 1992. Estimation of depression storage and Manning's resistance coefficient from random roughness measurements. Geoderma 52, 235–250.

Onstad, C.A., 1984. Depressional storage on tilled soil surfaces. Transactions of the ASAE 27 (3), 729–732.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1992. Numerical Recipes in C: The Art of Scientific Computing. Cambridge Univ. Press, New York, NY, USA.

Saupe, D., 1988. Algorithms for random fractals. In: Peitgen, H.-O., Saupe, D. (Eds.), The Science of Fractal Images, Springer, New York, NY, USA, Chap. 2, pp. 71–113, 126–136.

Sedgewick, R., 1990. Algorithms in C. Addison Wesley, Reading, MS, USA.

Sneddon, J., Chapman, T.G., 1989. Measurement and analysis of depression storage on a hillslope. Hydrological Processes 3, 1–13.

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

Ullah, W., Dickinson, W.T., 1979. Quantitative description of depression storage using a digital surface model. Journal of Hydrology 42, 63–75.

Voss, R.F., 1988. Fractals in nature: from characterization to simulation. In: Peitgen, H.-O., Saupe, D. (Eds.), The Science of Fractal Images. Springer, New York, NY, USA, pp. 21–70, Chap. 1.

Zhang, W., Montgomery, D.R., 1994. Digital elevation model grid size, landscape representation, and hydrologic simulations. Water Resources Research 30 (4), 1019–1028.