

## PolyRES: A Polygon-Based Richards Equation Solver

Prepared by  
R. G. Hills, New Mexico State University  
P. D. Meyer, Pacific Northwest Laboratory  
M. L. Rockhold, Pacific Northwest Laboratory

Pacific Northwest Laboratory  
Richland, WA 99352

Subcontractor:  
Department of Mechanical Engineering  
New Mexico State University  
Las Cruces, NM 88003

Prepared for  
Division of Regulatory Applications  
Office of Nuclear Regulatory Research  
U.S. Nuclear Regulatory Commission  
Washington DC 20555-0001  
NRC JCN L2466



## Abstract

This document describes the theory, implementation, and use of a software package designed to solve the transient, two-dimensional, Richards equation for water flow in unsaturated-saturated soils. This package was specifically designed to model complex geometries with minimal input from the user. The spatial variation of the hydraulic properties can be defined across individual polygon-shaped subdomains, called objects. These objects combine to form a polygon-shaped model domain. Each object can have its own distribution of hydraulic parameters. The resulting model domain and polygon-shaped internal objects are mapped onto a rectangular, finite-volume, computational grid by a preprocessor. This allows the user to specify model geometry independently of the underlying grid and greatly simplifies user input for complex geometries. In addition, this approach significantly reduces the computational requirements since complex geometries are actually modeled on a rectangular grid. This results in well-structured, finite difference-like systems of equations that require minimal storage and are very efficient to solve.

The documentation for this software package includes a user's manual, a detailed description of the underlying theory, and a detailed discussion of program flow. Several example problems are presented that show the use and features of the software package. The water flow predictions for several of these example problems are compared to those of another algorithm (Kirkland et al., 1992) to test for prediction equivalency.

The computer code described in this document is available from the Energy Science and Technology Software Center, P.O. Box 1020, Oak Ridge, TN 37831-1020.



# Contents

Abstract .....	iii
Foreword .....	vii
Acknowledgments .....	ix
1.0 Introduction .....	1
1.1 Overview .....	1
1.2 Approach .....	1
2.0 User's Manual .....	3
2.1 Overview .....	3
2.2 Watermap .....	3
2.2.1 Sample Geometry File for Figure 2.1 .....	5
2.3 Water .....	6
2.3.1 User-Supplied Subroutines for Water Corresponding to Figure 2.1 .....	6
3.0 Theory .....	13
3.1 Overview .....	13
3.2 Mass Balance .....	13
3.3 Boundary Conditions .....	16
3.4 Cells Fully Outside the Boundary .....	18
3.5 Mass Correction .....	19
3.6 Fluxes Between Polygon Objects .....	19
3.7 Summary .....	21
4.0 Preprocessor .....	23
4.1 Overview .....	23
4.2 MAIN .....	23
4.3 BNDFLUX .....	24
4.4 BOUNDINFO .....	24
4.5 FILL .....	25
5.0 Richards Equation Solver .....	27
5.1 Overview .....	27
5.2 MAIN .....	27
5.3 MATRIX .....	28
5.4 UPDATE .....	28
5.5 UPDATEFLUX .....	29
5.6 UPDATEFLUXSEGS .....	29
6.0 Example Problems .....	31
6.1 Example 1: Uniform Soil .....	31
6.1.1 Input File for Example 1 .....	31
6.1.2 User-Supplied Subroutines for Example 1 .....	32

6.2 Example 2: Blocked Soil .....	35
6.2.1 Example 2a: Multiple homogeneous polygon objects .....	36
6.2.1.1 Input File for Example 2a .....	36
6.2.1.2 User Subroutine VG_PROP for Example 2a .....	37
6.2.2 Example 2b: Single heterogeneous polygon object .....	38
6.2.2.1 User Subroutine VG_PROP for Example 2b .....	38
6.3 Example 3: Engineered Cover .....	39
6.3.1 User Subroutine OUTPUT for Example 3 .....	39
7.0 Results .....	43
7.1 Introduction .....	43
7.2 Results and Comparative Analysis for the First Two Test Cases .....	43
7.3 Results for the Engineered Cover .....	43
8.0 Discussion and Conclusions .....	55
9.0 References .....	57

### Figures

2.1.. Example geometry .....	4
3.1.. Polygon objects and computational grid .....	14
3.2.. Two boundary segments .....	18
6.1.. Model domain and van Genuchten parameters for uniform soil example .....	32
6.2.. Model domain and van Genuchten parameters for a blocked soil example .....	35
7-1.. Object map for the blocked soil .....	44
7-2.. Contours of volumetric water content on day 70 for Example 1: Uniform Soil .....	45
7-3.. Contours of volumetric water content on day 70 for Example 2a: Blocked Soil .....	46
7-4.. Contours of volumetric water content on day 70 for Example 2b: Blocked Soil .....	47
7-5.. Object map for the engineered cover .....	49
7-6.. Water movement across segments as output from <b>Water</b> for Example 3: <b>Engineered Cover</b> .....	50
7-7.. Contours of pressure head at 1050 days for Example 3: Engineered Cover .....	51
7-8.. Vector plot of water flux at 1050 days for Example 3: Engineered Cover .....	52
7-9.. Detail of water flux near the corner of the capillary barrier at 1050 days for Example 3: Engineered Cover .....	53
7-10.. Detail of water flux for a single column of nodes above the capillary barrier at x=59.5 ft for Example 3 .....	54

### Tables

7.1.. Prediction and prediction error statistics .....	48
--	----

## Foreword

This technical report, NUREG/CR-6366, was prepared by the New Mexico State University and Pacific Northwest Laboratory<sup>1</sup> under a research project with the Waste Management Branch in the Office of Nuclear Regulatory Research (FIN L2466). The report documents PolyRES<sup>2</sup>, a computer code used to simulate ground-water flow in complex, heterogeneous, unsaturated porous media related to assessment of low-level radioactive waste (LLW) disposal sites and engineered facilities. The unique aspect of the documented code is its use of a polygon-based solution technique to solve the governing Richards equation for partially saturated flow conditions. The report provides a demonstration of the code's capabilities using a range of test case examples. The code was used in the development and testing of an infiltration evaluation methodology (documented in NUREG/CR-6346) for use in assessing performance of a LLW disposal facility. The documented code and simulation examples were independently peer-reviewed by scientists at Sandia National Laboratories.

NUREG/CR-6366 is not a substitute for NRC regulations, and compliance is not required. The approaches and/or methods described in this NUREG/CR are provided for information only. Publication of this report does not necessarily constitute NRC approval or agreement with the information contained herein.

---

1. Pacific Northwest Laboratory is operated for the U.S. Department of Energy by Battelle Memorial Institute under Contract DE-AC06-76RLO 1830.

2. Copies of the PolyRES computer software are available from the Energy Science and Technology Software Center, P.O. Box 1020, Oak Ridge, TN 37831-1020.



## **Acknowledgments**

The authors wish to acknowledge the support of the NRC Office of Nuclear Regulatory Research. The computer code described in this document was developed for the project, Application of an Infiltration Evaluation Methodology to Low-Level Waste Performance Assessment Studies. Tom Nicholson is the NRC technical monitor for this project. The advice and support of Glendon Gee of Pacific Northwest Laboratory, and Ralph Cady and Tom Nicholson of the NRC is greatly appreciated. Thanks are also extended to Jim McCord of Sandia National Laboratory for his review of the code and its documentation.



# 1.0 Introduction

## 1.1 Overview

This document provides a description of a polygon-based Richards equation solver software package. The documentation assumes and the operation of the code requires that the user has a working knowledge of soil physics, numerical methods, and Fortran programming. The solver uses a finite volume approach and can model variably saturated (unsaturated-saturated) transient flow in two dimensions. The package is designed to allow the user to easily model polygon-shaped regions with flux, head, or unit-gradient boundary conditions. The region may be composed of one or more irregularly shaped objects for which the van Genuchten properties (van Genuchten, 1980a, 1980b) may be spatially variable or constant.

The primary goals that guided the development of this software package were 1) complex geometries associated with barrier caps should be very easy for the user to specify and modify, and 2) the execution of the water solver must be fast so it can be used to model large-scale problems over long time scales. The geometry of the problem, including the location of the boundaries, their type, and the shape of internal objects are defined through a simple user-supplied input file. The remaining input data, such as the transient behavior of boundary conditions, are provided through user-supplied subroutines. This provides the user with flexibility as to the type of problems solved.

The computer code described in this document is available from the Energy Science and Technology Software Center, P.O. Box 1020, Oak Ridge, TN 37831-1020.

## 1.2 Approach

The model domain comprises one or more internal polygon-shaped objects. These objects must fully cover the model domain. Each object is defined by the user through the coordinates of the polygon corner points. Each object may correspond to a different soil type, or a soil with different stochastic properties.

To enhance user-friendliness, a preprocessor was developed that maps the geometry of the model domain and the internal objects onto a rectangular computational grid. Mapping the problem to a rectangular grid results in well-structured, banded, symmetric matrices that are quick to assemble, efficient to store, and quick to solve even though the geometry may be very complex. This allows the user to define the geometry of the region being simulated and the geometry of internal objects within that region independently of the computational grid. There is no requirement that the underlying grid align with the problem boundaries or with the sides of the internal objects. The boundary of the model domain is also defined by polygon corner points. Each segment of the boundary polygon can be specified as a head, flux, or unit gradient boundary segment.

The spatial distribution of the van Genuchten parameters (van Genuchten, 1980a, 1980b) over each object are defined through a user-supplied FORTRAN subroutine. This allows the user to specify constant parameters over an object or to define stochastic or other distributions over the object. User-supplied subroutines are also used to define initial conditions, time dependence of the boundary conditions, and output format. These subroutines are linked to the Richards equation solver before execution.

## Introduction

## 2.0 User's Manual

### 2.1 Overview

Program input must be provided by the user to each of two components of the polygon-based solver. These components are the geometric preprocessor **Watermap** and the Richards equation solver **Water**. **Watermap** takes the user-defined polygon-shaped model domain, and the polygon-shaped internal objects, and maps these onto a user-defined rectangular computational grid. **Watermap** then outputs the mapped results in a form readable by the Richards equation solver. Once preprocessing is complete, the Richards equation solver **Water** is compiled, linked to user-supplied subroutines defining van Genuchten properties (van Genuchten, 1980a, 1980b), initial conditions, transient behavior of the boundaries, and the output format. The resulting executable code will then evaluate water contents, heads, and fluxes for the time interval specified by the user.

To run the package, the user should do the following.

1. Create the user-supplied input file describing the geometry of the objects as discussed below.
2. Execute the pre-processor **Watermap**. This routine will ask for the name of the input file. **Watermap** generates two output files, one named by the user that defines geometry information for use by **Water**, and one named INCLUDE.F that is a FORTRAN include file defining array sizes.
3. Compile **Water.f** so that the object code includes the INCLUDE.F information. Failure to complete this step after modifying the input file may result in execution errors.
4. Generate and compile the user-supplied subroutine file. Link this file with the object version of **Water** and execute the results. When executed, **Water** will ask for the name of the output file generated by **Watermap** and for time-step information.

Specific information is provided below.

### 2.2 Watermap

**Watermap** preprocesses the user-supplied geometry file to evaluate finite volume cell information for the Richards equation solver **Water**. Listed in Section 2.2.1 is a sample geometry file that defines the geometry shown in Figure 2.1. This geometry defines an engineered waste isolation cover and is taken from Meyer (1993). Free format is used throughout, and line skips can be placed between the records at the user's discretion.

The underlying finite volume grid is defined first (see Section 2.2.1), with x measured from left to right and y measured downward. Any units can be used as long as the units are consistent throughout the input file and the user-supplied subprograms. The user can use a uniform grid or a variably spaced rectangular grid to increase the resolution locally. In the example provided, the grid in the x-direction contains three regions with 13 equally sized cells from x=0 to x=16.25, 11 equally sized cells from x=16.25 to x=30, and 64 equally sized cells from x=30 to x=90. The y direction grid contains five regions.

The boundary polygon defining the type and location of the boundaries is defined next. In this example, the boundary polygon is defined by 10 linear segments with the x-y coordinate (i.e., x=0, y=23.25) of the first point of the first polygon segment given first. The point x=36, y=16.05 represents the end point of the first segment and the beginning point of the second segment. The beginning point of the last boundary segment is x=0, y=27.852. The last segment is assumed to end at the original point x=0, y=23.25. Note, specification of the boundary polygon must include all points of intersection (e.g., x=36, y=16.05) and not just the polygon vertices. The third and fourth columns represent the boundary type and the internal polygon number adjacent to this line segment. The internal polygon numbers must be provided by the user for each boundary segment so that the solver knows which set of van Genuchten properties to use. For example, internal polygon (object) five lies along segment two of the boundary polygon. A boundary line segment cannot lie along more than one internal polygon. However, segments can be split into colinear segments so that each one is adjacent to only one internal object. The boundary types, which appear in the third column, are represented by the integers 1, 2, and 3, where

- 1 indicates a head boundary,
- 2 indicates a flux boundary, and
- 3 indicates a unit gradient boundary in the vertical direction.

The time dependence for the first two types of boundaries is not defined here but in the user-supplied subroutines. These subroutines are discussed in Section 2.3.

After the boundary polygon segments are defined, the number of internal polygons and the polygon corner points are defined. Internal polygons must cover the entire region defined by the boundary polygon without overlapping. The preprocessor will

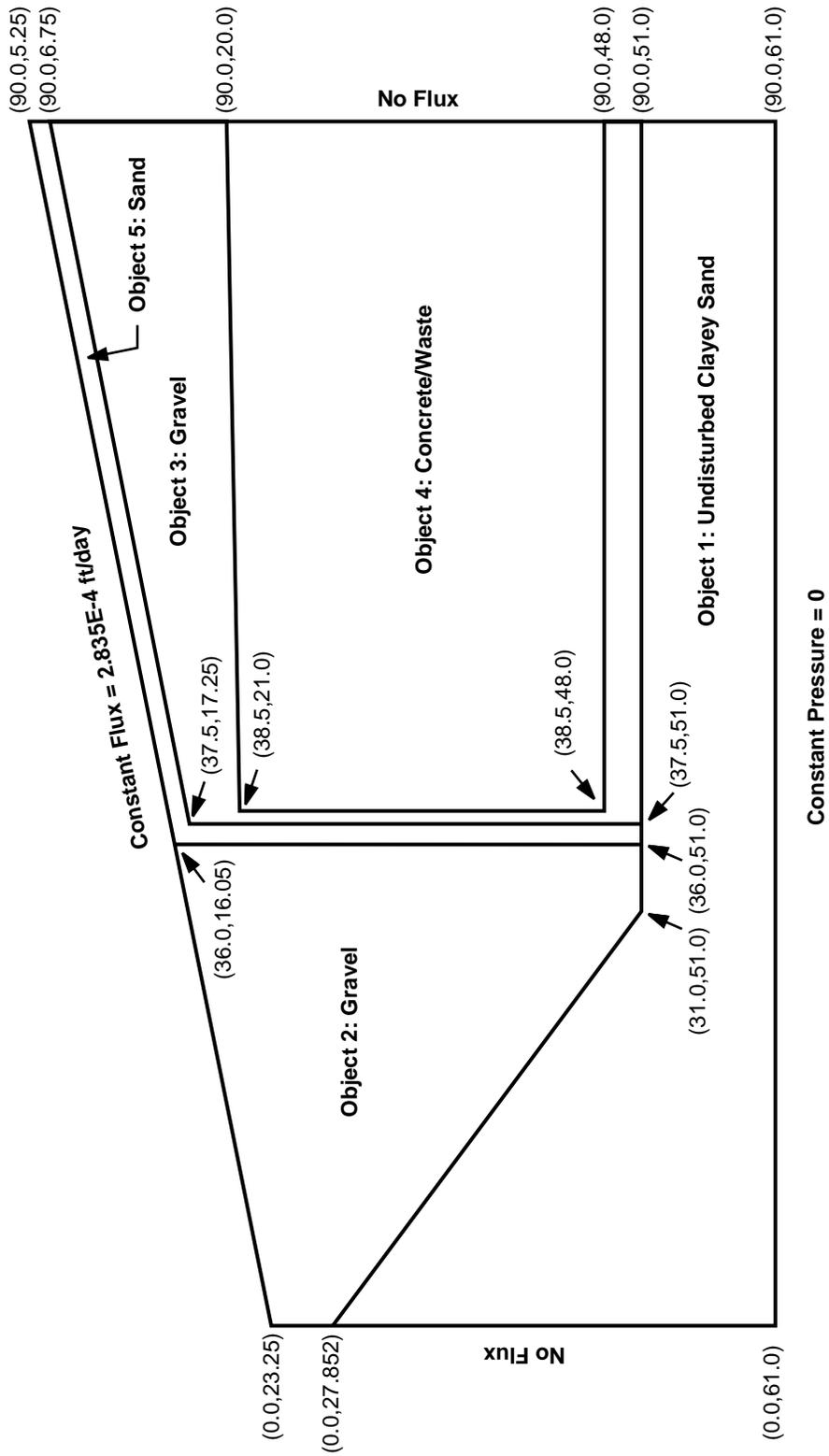


Figure 2.1. Example geometry

check whether the simulation domain is properly covered and will print an error message if it is not. These polygons are used to define the subdomains across which the user may specify different variations in the van Genuchten properties (discussed in the next section). In this example, there are five internal polygons (objects), the first defined by five points (i.e., x-y pairs), the second by five points, the third by eight points, the fourth by four points, and the last by six points. Any record after the last internal polygon record is ignored. Note, numbering of internal polygons is determined by their order of appearance in the input file.

Given the user defined geometry input file, **Watermap** generates two output files. One file is used as input for the Richards equation solver. **Watermap** will ask the user to name this file. The second file is INCLUDE.F, which is a FORTRAN include file that defines the array sizes used by **Water** for this particular geometry.

### 2.2.1 Sample Geometry File for Figure 2.1

```

3 : Number of cell size regions for x direction
0 16.25 13      : x1 x2 nx - region 1
16.25 30 11    : x1 x2 nx - region 2
30 90 64       : x1 x2 nx - region 3

5 : Number of cell size regions for y direction
5.25 17.25 64  : y1 y2 ny - region 1
17.25 20 11    : y1 y2 ny - region 2
20 23.25 13    : y1 y2 ny - region 3
23.25 24 3     : y1 y2 ny - region 4
24 61 37      : y1 y2 ny - region 5

10 : Number of points for boundary polygon followed by x-y pairs, type, object #
0 23.25 2 2
36 16.05 2 5
90 5.25 2 5
90 6.75 2 3
90 20 2 4
90 48 2 3
90 51 2 1
90 61 1 1
0 61 2 1
0 27.852 2 2

5 : Total number of internal objects (excluding boundary polygon)

5 : Number of points for object 1 followed by x-y pairs
0 27.852
31 51
90 51
90 61
0 61

5 : Number of points for object 2 followed by x-y pairs
0 23.25
36 16.05
36 51
31 51
0 27.852

8 : Number of points for object 3 followed by x-y pairs
37.5 17.25
90 6.75
90 20
38.5 21
38.5 48
90 48

```

## User's Manual

90 51  
37.5 51

4 : Number of points for object 4 followed by x-y pairs  
38.5 21  
90 20  
90 48  
38.5 48

6 : Number of points for object 5 followed by x-y pairs  
36 16.05  
36 51  
37.5 51  
37.5 17.25  
90 6.75  
90 5.25

## 2.3 Water

After **Watermap** generates a preprocessed data file and an INCLUDE.F file, the user must re-compile **Water.f** so that the proper array sizes as defined in INCLUDE.F are incorporated. Failure to do so can lead to execution errors. In addition, the user must supply a file containing the user-supplied FORTRAN subroutines. A compiled version of this file must be linked to **Water** before execution. The required user-supplied subroutines are:

1. **BNDFCN** – This subroutine specifies the time-varying forcing function for each of the boundary segments. The forcing function can represent head, flux, or unit gradient boundary conditions.
2. **H\_INIT** – This subroutine specifies the initial pressure head distribution for each cell.
3. **OUTPUT** – This subroutine generates the output. Pressure head, volumetric water content, and fluxes are available. The user must specify the desired format.
4. **VG\_PROP** – This subroutine specifies the van Genuchten properties (residual and saturated water contents, saturated hydraulic conductivity, and curve fitting parameters  $\alpha$  and  $n$ ) for each object specified in the user-supplied input file. The units of the properties must be consistent with the other user input.

Examples of the user-supplied subroutines are listed in Section 2.3.1. See the comment statements at the beginning of each subroutine for further content information.

During execution, **Water** will ask for the name of the processed geometry file (i.e., the user-named output file from **Watermap**), the name of the output file from **Water** (contains the output as defined by the user in the OUTPUT subroutine listed in Section 2.3.1), and variable time-step information. The user is asked for the initial time step size, the maximum allowable time-step size (to be used by the adaptive time-step algorithm), and the number  $n$  of output times. The user is then asked to specify the  $n$  output times. The program will terminate execution after output for the last output time. The user-supplied subroutine OUTPUT will be called at each output time.

Additional information, such as cumulative water flow through the boundary and object segments, will be printed to the screen for each output time. Segment numbers correspond to the order of the segment definition (i.e., first user-defined segment is segment one and first user defined internal object is object one) in the user-supplied geometry file. The difference between the net flow into the model domain and into all of the objects represents the mass balance error (i.e., the global mass balance error). When a node becomes saturated, mass balance may no longer be conserved for a cell. An estimate of the error due to saturated cells is also printed. The net (i.e., total) water flow through all segments that are not on the boundary is also printed. Since water flowing out of one object flows into the adjacent object, the net flow through each of the corresponding objects' segments should be equal and opposite. The net water flow through all object segments (excluding those on the boundary) should thus be zero. This value is provided as another check on mass balance.

### 2.3.1 User-Supplied Subroutines for Water Corresponding to Figure 2.1

```
SUBROUTINE BNDFCN (TIME,NBS,FORCE)
C
C      This is a user-supplied subroutine defining the forcing
```

```

C      function for each of the NBS boundary segments as a
C      function of time. Note that the forcing function can
C      represent head (type 1), flux (type 2), or unit gradient
C      (type 3). If a unit gradient boundary is specified for a
C      segment (type 3), FORCE for this
C      segment is not used and does not need to be defined.
C
C      The arguments for this subroutine are
C          TIME - Current time (supplied by calling routine)
C                in units consistent with the user-defined
C                input
C          NBS - Number of boundary segments (supplied by
C                the calling routine)
C          FORCE - Value of boundary forcing function in units
C                consistent with the user-defined input
C
C      DIMENSION  FORCE(NBS)
C
C      ***  In this example, the upper boundary (i.e., the first two
C      ***  segments) is a flux boundary (type 2) with the flux specified.
C      ***  The forcing function for the remainder of boundary segments are
C      ***  zero (i.e., zero head for segment 9 and zero flux for the
C      ***  remaining segments).
C
C      FORCE(1) = 2.835E-4
C      FORCE(2) = 2.835E-4
C
C      DO I=3,NBS
C          FORCE(I) = 0.0
C      END DO
C
C      RETURN
C      END
C
C      SUBROUTINE H_INIT(H,NX,NY,XG,YG,XCG,YCG)
C
C      This is a user-supplied subroutine defining the
C      initial pressure distribution.
C
C          H - Initial pressure head defined by user
C              in units consistent with the user
C              input.
C          NX - Number of cells in x direction
C              (supplied by calling routine)
C          NY - Number of cells in y direction
C              (supplied by calling routine)
C          XG - x coordinates of cell sides (vector)
C              (supplied by calling routine)
C          YG - y coordinates of cell sides (vector)
C              (supplied by calling routine)
C          XCG - x coordinates of cell centroids (array)
C              (supplied by calling routine)
C          YCG - y coordinates of cell centroids (array)
C              (supplied by calling routine)
C
C      DIMENSION  H(NX,NY), XG(NX+1), YG(NY+1)
C      DIMENSION  XCG(NX,NY), YCG(NX,NY)
C
C      ***  In this example, a water table is assumed to exist at

```

## User's Manual

```
C *** the lower boundary (i.e., y=YG(NY+1)) with unit gradient
C *** initial pressure above.
C
C      DO J=1,NY
C          DO I=1,NX
C              H(I,J) = -ABS(YG(NY+1) - YCG(I,J))
C          END DO
C      END DO
C
C      RETURN
C      END

SUBROUTINE OUTPUT (NFILEOUT,TIME,H,THETA,FLUXX,FLUXY,NX,NY,
.                XG,YG,XCG,YCG)
C
C      This is a user-supplied subroutine defining the output
C      format. All arguments are input from the main program. The
C      user should not change the values of any of these arguments.
C
C      NFILEOUT - Unit number for output file
C                  (supplied by calling routine)
C      TIME - Current time
C                  (supplied by calling routine)
C      H - Pressure head
C                  (supplied by calling routine)
C      THETA - Volumetric water content
C                  (supplied by calling routine)
C      FLUXX - Water flux through cell sides in x-dir
C                  (supplied by calling routine)
C      FLUXY - Water flux through cell sides in y-dir
C                  (supplied by calling routine)
C      NX - Number of cells in x direction
C                  (supplied by calling routine)
C      NY - Number of cells in y direction
C                  (supplied by calling routine)
C      XG - x coordinates of cell sides (vector)
C                  (supplied by calling routine)
C      YG - y coordinates of cell sides (vector)
C                  (supplied by calling routine)
C      XCG - x coordinates of cell centroids (array)
C                  (supplied by calling routine)
C      YCG - y coordinates of cell centroids (array)
C                  (supplied by calling routine)
C
C      DIMENSION H(NX,NY), THETA(NX,NY), XG(NX+1), YG(NY+1)
C      DIMENSION XCG(NX,NY), YCG(NX,NY), FLUXX(NX-1,NY), FLUXY(NX,NY-1)
C      DIMENSION WORK(200,200)
C
C
C      ***      Change the following to a format of your choice. In the
C      ***      following example, heads at I = 49 from j = 40 to 90
C      ***      are outputted as a function of depth below an upper boundary
C      ***      at Y1 to the screen and to the output file.
C
C
C      I = 49
C
C      WRITE(*,*) 'TIME, X = ', TIME, (XG(I-1)+XG(I))/2.0
C
C      WRITE(*,*) 'Y, Head'
```

```
Y1 = (YG(40)+YG(41))/2.0
```

```
DO J=40,90
  WRITE(*,*) YCG(I,J) - Y1, H(I,J)
END DO
```

```
WRITE(NFILEOUT,*) 'TIME, X = ', TIME, (XG(I-1)+XG(I))/2.0
```

```
WRITE(NFILEOUT,*) 'Y, Head'
```

```
DO J=40,90
  WRITE(NFILEOUT,*) YCG(I,J) - Y1, H(I,J)
END DO
```

```
RETURN
END
```

```
SUBROUTINE VG_PROP (N,IOBJ,X,Y,THEAR,THEAS,ALPHA,AN,AKS)
```

```
C
C      This is a user-supplied subroutine defining the van Genuchten
C      properties for each object in units consistent with the rest
C      of the user input. The subroutine is supplied with the object
C      number and the x, y coordinates of the centroid of each cell which
C      contains that object. The user should return the corresponding
C      vectors of the van Genuchten properties. Notes: 1) While a centroid
C      will never lie outside a boundary, it may lie outside an object
C      when the object intersects a small part of the cell. The user should
C      provide the corresponding van Genuchten properties even though
C      the centroid x, y coordinates may be just outside the object's
C      polygon. The program uses these values to estimate average values
C      for that region at cell interfaces within the object.
C      2) This routine will be called only during the initialization of
C      the solver.
```

```
C
C      N - Number of cells containing the object
C           (supplied by calling routine)
C      IOBJ - Object number
C           (supplied by calling routine)
C      X, Y - Vectors of coordinates of centroid of each cell
C           containing the object
C           (supplied by calling routine)
C      THEAR - Vector of corresponding residual water contents
C           (user defined)
C      THEAS - Vector of corresponding saturated water contents
C           (user defined)
C      ALPHA - Vector of corresponding V-G alpha parameters
C           (user defined)
C      AN - Vector of corresponding V-G n parameters
C           (user defined)
C      AKS - Vector of corresponding saturated hydraulic
C           conductivities (user defined)
```

```
C
C      DIMENSION X(N), Y(N), THEAR(N), THEAS(N), ALPHA(N), AN(N), AKS(N)
```

```
C ***      In the example that follows, five objects are defined with the
C ***      properties taken as constant across each object. Cell centroids
```

## User's Manual

```
C *** are provided if the user wishes to define spatially dependent
C *** properties across an object.
C
C IF (IOBJ .EQ. 1) THEN
C
C *** Clayey sand
C
C DO I=1,N
C   THEAR(I) = 0.21
C   THEAS(I) = 0.30
C   ALPHA(I) = 0.107
C   AN(I) = 3.0
C   AKS(I) = 3.968E-4
C END DO
C
C ELSE IF (IOBJ .EQ. 2) THEN
C
C *** Gravel
C
C DO I=1,N
C   THEAR(I) = 0.014
C   THEAS(I) = 0.51
C   ALPHA(I) = 107.8
C   AN(I) = 2.661
C   AKS(I) = 5.244E3
C END DO
C
C ELSE IF (IOBJ .EQ. 3) THEN
C
C *** Gravel
C
C DO I=1,N
C   THEAR(I) = 0.014
C   THEAS(I) = 0.51
C   ALPHA(I) = 107.8
C   AN(I) = 2.661
C   AKS(I) = 5.244E3
C END DO
C
C ELSE IF (IOBJ .EQ. 4) THEN
C
C *** Concrete/waste
C
C DO I=1,N
C   THEAR(I) = 0.08
C   THEAS(I) = 0.4
C   ALPHA(I) = 0.192
C   AN(I) = 1.082
C   AKS(I) = 2.835E-5
C END DO
C
C ELSE IF (IOBJ .EQ. 5) THEN
C
C *** Sand
C
C DO I=1,N
C   THEAR(I) = 0.045
C   THEAS(I) = 0.37
C   ALPHA(I) = 2.082
C   AN(I) = 2.080
C   AKS(I) = 8.504E+1
```

```
END DO  
  
ELSE  
  
    WRITE(*,*) ' Information for an object number was requested'  
    WRITE(*,*) ' which was not supplied by the user.'  
    STOP ' Execution terminated'  
  
END IF  
  
RETURN  
END
```



## 3.0 Theory

### 3.1 Overview

The two-dimensional pressure-based form of Richards equation can be written as:

$$-C \frac{\partial h}{\partial t} = \frac{\partial}{\partial x} \left( K \frac{\partial h}{\partial x} \right) + \frac{\partial}{\partial y} \left( K \frac{\partial h}{\partial y} - K \right) \quad (3.1)$$

where

$h$  = pressure head

$x$  = spatial coordinate increasing from left to right

$y$  = spatial coordinate increasing from top to bottom

$K$  = hydraulic conductivity, assumed to be isotropic and a unique function of pressure head

$C$  = specific water capacity, defined as  $C = \partial \theta / \partial h$

$\theta$  = volumetric water content, assumed to be a unique function of pressure head.

The water retention and hydraulic conductivity models used in the solution of Equation 3.1 are those of van Genuchten (1980b):

$$S_e = \frac{\theta - \theta_r}{\theta_s - \theta_r} = [1 + (\alpha|h|)^n]^{-m} \quad (3.2)$$

$$K = K_s S_e^{1/2} [1 - (1 - S_e^{1/m})^m]^2 \quad (3.3)$$

where

$S_e$  = effective saturation

$\theta_r$  = residual volumetric water content

$\theta_s$  = saturated volumetric water content

$K_s$  = saturated hydraulic conductivity

$m = 1 - 1/n$

$\alpha, n$  = model fitting parameters.

By accounting for the effect of the irregular shape of objects on the cell-to-cell hydraulic conductivities, a simple finite volume approach can be used to solve Equation 3.1 on a rectangular grid. In this and the following two chapters, we present the underlying theory and implementation of this solution.

### 3.2 Mass Balance

Consider the rectangular grid and the polygon objects shown in Figure 3.1. As stated above,  $x$  is measured to the right, and  $y$  is measured down.  $A_{i,j}^k$  represents the area which object  $k$  covers in cell  $i,j$ . The length of the side between the cells  $i,j$  and  $i+1,j$  covered by a polygon segment of object  $k$  on both sides is  $Lx_{i+1/2,j}^k$ .  $Lx_{i+1/2,j}^{k,k^*}$  is the length of the side of a cell covered by object  $k$  that is adjacent to object  $k^*$  (see Figure 3.1). Considering the lengths  $Lx_{i+1/2,j}^k$  and  $Lx_{i+1/2,j}^{k,k^*}$  separately simplifies the computer code for the evaluation of the object-to-object water movement.

The pressure head gradient anywhere along the side between cell  $i,j$  and  $i+1,j$  is approximated by  $(h_{i+1,j} - h_{i,j}) / \Delta x$ , where  $h_{i,j}$  is the average pressure head in cell  $i,j$ . Using this notation, the water movement per unit time through the side between cell  $i,j$  and  $i+1,j$ , which are both at least partially covered by object  $k$ , is

$$q_{i+1/2,j}^k Lx_{i+1/2,j}^k = -K_{i+1/2,j}^k Lx_{i+1/2,j}^k \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} \quad (3.4)$$

where  $q_{i+1/2,j}^k$  is the flux in object  $k$  across the side between cell  $i,j$  and  $i+1,j$ . In the present work, the unsaturated hydraulic conductivity at a cell interface  $i+1/2$  is evaluated as the arithmetic average of the hydraulic conductivities at the heads associated with the cells  $i$  and  $i+1$  for object  $k$ . The  $\Delta x_{i+1/2}$  represents the  $x$  direction distance between the centroids of those parts

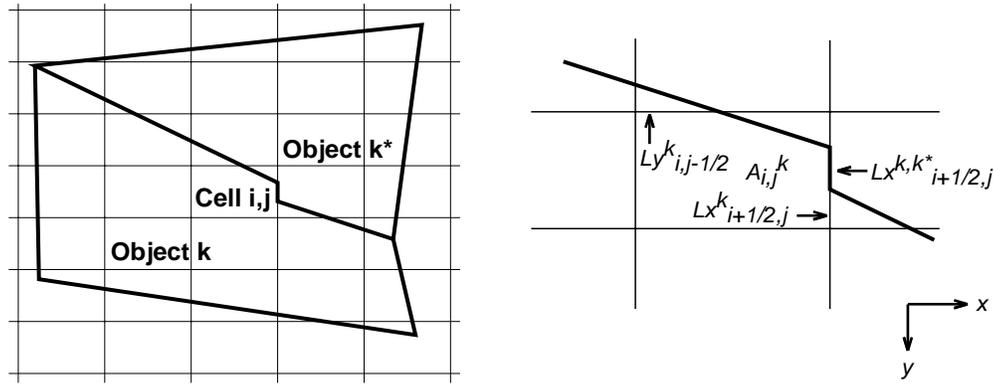


Figure 3.1. Polygon objects and computational grid

of the cells  $i,j$  and cell  $i+1,j$  within the simulation domain. The water movement per unit time between objects  $k$  and  $k^*$  for an object segment that lies along a cell side at  $i+1/2$  is

$$q^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} = -K^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} \quad (3.5)$$

where  $K^{k,k^*}_{i+1/2,j}$  is the arithmetic average of the hydraulic conductivities evaluated at the heads associated with cells  $i$  and  $i+1/2$  for the corresponding objects. The water movement per unit time from cell  $i,j$  to cell  $i+1,j$  for all objects covering this cell side is

$$\begin{aligned} Q_{i+1/2,j} &= -\sum_{k=1}^{n_{objs}} K^k_{i+1/2,j} Lx^k_{i+1/2,j} \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} - \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} \\ &= -\left( \sum_{k=1}^{n_{objs}} K^k_{i+1/2,j} Lx^k_{i+1/2,j} \right) \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} \right) \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} \end{aligned} \quad (3.6)$$

where  $n_{objs}$  is the number of objects covering cell  $i,j$  and  $n_{segs}$  is the number of polygon segments along the cell side. Likewise, the water movement per unit time for all objects covering the interface between cell  $i,j$  and  $i,j+1$  is

$$\begin{aligned} Q_{i,j+1/2} &= \left( \sum_{k=1}^{n_{objs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) - \left( \sum_{k=1}^{n_{objs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \frac{h_{i,j+1} - h_{i,j}}{\Delta y_{j+1/2}} \\ &\quad + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right) - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right) \frac{h_{i,j+1} - h_{i,j}}{\Delta y_{j+1/2}} \end{aligned} \quad (3.7)$$

The first and third terms in Equation 3.7 represent the water movement in the vertical direction under a gravitational gradient of unity. The time rate of change in water in cell  $i,j$  is given by

$$\sum_{k=1}^{n_{objs}} A_{i,j}^k \frac{\partial \theta_{i,j}^k}{\partial t} = \left( \sum_{k=1}^{n_{objs}} A_{i,j}^k C^k_{i,j} \right) \frac{\partial h_{i,j}}{\partial t} \quad (3.8)$$

where  $t$  designates time,  $\theta_{i,j}^k$  is the volumetric water content of object  $k$  in cell  $i,j$ , and  $C^k_{i,j}$  is the specific water capacity of object  $k$  in cell  $i,j$ . Note, the average head of each object in cell  $i,j$  is assumed to be  $h_{i,j}$ . Performing a mass balance for all water fluxes into the  $i,j$  cell leads to

$$\begin{aligned}
\left( \sum_{k=1}^{n_{obs}} A_{i,j}^k C^k_{i,j} \right) \frac{\partial h_{i,j}}{\partial t} = & - \left( \sum_{k=1}^{n_{obs}} K^k_{i-1/2,j} Lx^k_{i-1/2,j} \right) \frac{h_{i,j} - h_{i-1,j}}{\Delta x_{i-1/2}} \\
& - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i-1/2,j} Lx^{k,k^*}_{i-1/2,j} \right) \frac{h_{i,j} - h_{i-1,j}}{\Delta x_{i-1/2}} \\
& + \left( \sum_{k=1}^{n_{obs}} K^k_{i+1/2,j} Lx^k_{i+1/2,j} \right) \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} \\
& + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} \right) \frac{h_{i+1,j} - h_{i,j}}{\Delta x_{i+1/2}} \\
& - \left( \sum_{k=1}^{n_{obs}} K^k_{i,j-1/2} Ly^k_{i,j-1/2} \right) \frac{h_{i,j} - h_{i,j-1}}{\Delta y_{j-1/2}} \\
& - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j-1/2} Ly^{k,k^*}_{i,j-1/2} \right) \frac{h_{i,j} - h_{i,j-1}}{\Delta y_{j-1/2}} \\
& + \left( \sum_{k=1}^{n_{obs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \frac{h_{i,j+1} - h_{i,j}}{\Delta y_{j+1/2}} \\
& + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right) \frac{h_{i,j+1} - h_{i,j}}{\Delta y_{j+1/2}} \\
& + \left( \sum_{k=1}^{n_{obs}} K^k_{i,j-1/2} Ly^k_{i,j-1/2} \right) - \left( \sum_{k=1}^{n_{obs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \\
& + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j-1/2} Ly^{k,k^*}_{i,j-1/2} \right) \\
& - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right)
\end{aligned} \tag{3.9}$$

where  $n_{obs}$  and  $n_{segs}$  may take on different values for each of the 4 cell sides. We discretize Equation 3.9 over time implicitly, evaluating all coefficients at the old time step  $m$ .

## Theory

$$\begin{aligned}
 \left( \sum_{k=1}^{n_{objs}} A^k_{i,j} C^k_{i,j}{}^m \right) \frac{h_{i,j}' - h_{i,j}{}^m}{\Delta t} = & - \left( \sum_{k=1}^{n_{objs}} K^k_{i-1/2,j} Lx^k_{i-1/2,j} \right) \frac{h_{i,j}' - h_{i-1,j}'}{\Delta x_{i-1/2}} \\
 & - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i-1/2,j} Lx^{k,k^*}_{i-1/2,j} \right) \frac{h_{i,j}' - h_{i-1,j}'}{\Delta x_{i-1/2}} \\
 & + \left( \sum_{k=1}^{n_{objs}} K^k_{i+1/2,j} Lx^k_{i+1/2,j} \right) \frac{h_{i+1,j}' - h_{i,j}'}{\Delta x_{i+1/2}} \\
 & + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} \right) \frac{h_{i+1,j}' - h_{i,j}'}{\Delta x_{i+1/2}} \\
 & - \left( \sum_{k=1}^{n_{objs}} K^k_{i,j-1/2} Ly^k_{i,j-1/2} \right) \frac{h_{i,j}' - h_{i,j-1}'}{\Delta y_{j-1/2}} \\
 & - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j-1/2} Ly^{k,k^*}_{i,j-1/2} \right) \frac{h_{i,j}' - h_{i,j-1}'}{\Delta y_{j-1/2}} \\
 & + \left( \sum_{k=1}^{n_{objs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \frac{h_{i,j+1}' - h_{i,j}'}{\Delta y_{j+1/2}} \\
 & + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right) \frac{h_{i,j+1}' - h_{i,j}'}{\Delta y_{j+1/2}} \\
 & + \left( \sum_{k=1}^{n_{objs}} K^k_{i,j-1/2} Ly^k_{i,j-1/2} \right) - \left( \sum_{k=1}^{n_{objs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \\
 & + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j-1/2} Ly^{k,k^*}_{i,j-1/2} \right) \\
 & - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right)
 \end{aligned} \tag{3.10}$$

where  $n_{objs}$  and  $n_{segs}$  may take on different values for each of the four cell sides. The symbol ' is used to denote that  $h_{i,j}'$  is a temporary approximation for the head for the end of a time step. A technique analogous to the flux updating technique of Kirkland et al. (1992) will be used to refine this approximation so that the end-time step approximation on head is globally mass conservative (see Kirkland et al., 1992). This is presented in a latter section.

### 3.3 Boundary Conditions

For those cells that contain one or more boundary segments, a source term can be added to Equation 3.10 to account for the flux into the cell through each boundary segment within that cell.

$$\begin{aligned}
\left( \sum_{k=1}^{n_{objs}} A^k_{i,j} C^k_{i,j} \right) \frac{h_{i,j}' - h_{i,j}^m}{\Delta t} = & - \left( \sum_{k=1}^{n_{objs}} K^k_{i-1/2,j} Lx^k_{i-1/2,j} \right) \frac{h_{i,j}' - h_{i-1,j}'}{\Delta x_{i-1/2}} \\
& - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i-1/2,j} Lx^{k,k^*}_{i-1/2,j} \right) \frac{h_{i,j}' - h_{i-1,j}'}{\Delta x_{i-1/2}} \\
& + \left( \sum_{k=1}^{n_{objs}} K^k_{i+1/2,j} Lx^k_{i+1/2,j} \right) \frac{h_{i+1,j}' - h_{i,j}'}{\Delta x_{i+1/2}} \\
& + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i+1/2,j} Lx^{k,k^*}_{i+1/2,j} \right) \frac{h_{i+1,j}' - h_{i,j}'}{\Delta x_{i+1/2}} \\
& - \left( \sum_{k=1}^{n_{objs}} K^k_{i,j-1/2} Ly^k_{i,j-1/2} \right) \frac{h_{i,j}' - h_{i,j-1}'}{\Delta y_{j-1/2}} \\
& - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j-1/2} Ly^{k,k^*}_{i,j-1/2} \right) \frac{h_{i,j}' - h_{i,j-1}'}{\Delta y_{j-1/2}} \\
& + \left( \sum_{k=1}^{n_{objs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \frac{h_{i,j+1}' - h_{i,j}'}{\Delta y_{j+1/2}} \\
& + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right) \frac{h_{i,j+1}' - h_{i,j}'}{\Delta y_{j+1/2}} \\
& + \left( \sum_{k=1}^{n_{objs}} K^k_{i,j-1/2} Ly^k_{i,j-1/2} \right) - \left( \sum_{k=1}^{n_{objs}} K^k_{i,j+1/2} Ly^k_{i,j+1/2} \right) \\
& + \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j-1/2} Ly^{k,k^*}_{i,j-1/2} \right) \\
& - \left( \sum_{k,k^*=1}^{n_{segs}} K^{k,k^*}_{i,j+1/2} Ly^{k,k^*}_{i,j+1/2} \right) \\
& + \left( \sum_{l=1}^{n_{bndsegs}} q_l B_l \right)
\end{aligned} \tag{3.11}$$

where  $n_{objs}$  and  $n_{segs}$  may take on different values for each of the four cell sides,  $q_l$  is the water flux through boundary segment  $l$ , and  $n_{bndsegs}$  is the number of boundary segments within or along cell  $i,j$ . To evaluate the flux through each boundary segment into a cell, consider Figure 3.2. In this figure,  $B_l$  represents the length of each segment within the cell (not the total length of each segment), and  $\mathbf{n}_l$  represents the outward unit normal vector for each segment.  $\Delta z$  represents the smallest distance between the boundary segment and the centroid of the part of the cell within the simulation domain. This information is used for each of the three boundary condition types as follows.

### Head Boundary Conditions

Given that the head for boundary segment  $l$  is given by  $h_l(t)$ , the water flux into the cell  $i,j$  through the boundary segment  $l$ , designated  $q_l$ , is approximated by

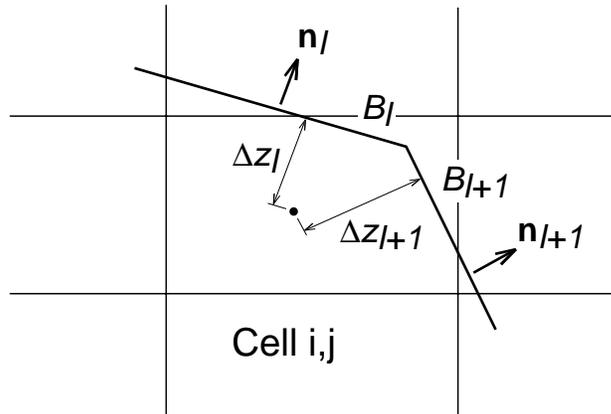


Figure 3.2. Two boundary segments

$$q_l = K_{i,j}^{l,m} \left( \frac{h_l(t^{m+1}) - h_{i,j}'}{\Delta z} - \mathbf{n}_l \cdot \mathbf{j} \right) \quad (3.12)$$

where  $K_{i,j}^{l,m}$  represents the unsaturated hydraulic conductivity for the internal object adjacent to this boundary segment evaluated at  $h_l(t)$ . Gravity is accounted for by the dot product between the outward unit normal  $\mathbf{n}_l$  for segment  $l$  and the downward pointed unit vector  $\mathbf{j}$ .

#### Flux Boundary Conditions

In this case,  $q_l(t)$  is specified explicitly by the user and can be used directly.

$$q_l = q_l(t^{m+1}) \quad (3.13)$$

#### Unit Gradient Boundary Conditions

Given a unit gradient (i.e.,  $h_l = h_{i,j}$  in Equation 3.12) for boundary segment  $l$ , the water flux into the cell  $i,j$  through the boundary segment  $l$  is approximated by

$$q_l = -K_{i,j}^{l,m} \mathbf{n}_l \cdot \mathbf{j} \quad (3.14)$$

### 3.4 Cells Fully Outside the Boundary

If the boundary polygon does not completely cover the computational grid, then there may be cells in the grid that are fully outside the boundary. To maintain the structure of the matrix equations, we include all cells of the rectangular grid in the computation, including those fully outside the boundary polygon. We arbitrarily apply the following condition to these cells:

$$h_{i,j}' = 0 \quad (3.15)$$

While these outside cells are now included in the matrix solution, they have no effect on those cells within the boundary polygon and have no effect on the number of iterations required by the iterative equation solver. They do require that the iterative solver loop through these outside cells. However, the effort required to loop through these cells is small compared to the savings in CPU gained because the matrix is well-structured.

### 3.5 Mass Correction

To enhance the global mass balance of the scheme, we use a flux updating modification. Following Kirkland et. al. (1992), the  $h_{i,j}'$  at the end of the time step are used in Equations 3.6, 3.7, and 3.12 through 3.14 to evaluate the water movement into a cell during that time step. These are then used to re-evaluate the water content in a cell at the end of a time step given the water content at the beginning of the time step.

$$A_{i,j} \frac{\theta_{i,j}^{m+1} - \theta_{i,j}^m}{\Delta t} = Q_{i-1/2,j} - Q_{i+1/2,j} + Q_{i,j-1/2} - Q_{i,j+1/2} + \left( \sum_{l=1}^{n_{\text{bndsegs}}} q_l B_l \right) \quad (3.16)$$

where  $\theta$  is the average volumetric water content for the cell and  $A_{i,j}$  is the area of the cell contained within the boundary. A slightly different approach can be used since the right-hand side of Equation 3.16 is the same as the right-hand side of Equation 3.11. We note that

$$A_{i,j} \frac{\theta_{i,j}^{m+1} - \theta_{i,j}^m}{\Delta t} = \left( \sum_{k=1}^{n_{\text{objs}}} A_{i,j}^k C_{i,j}^{k,m} \right) \frac{h_{i,j}' - h_{i,j}^m}{\Delta t} \quad (3.17)$$

After the  $h_{i,j}'$  are evaluated from the system of equations given by Equations 3.11 through 3.15, Equation 3.17 can be used to evaluate the corresponding average volumetric water content. Evaluating the water content by Equation 3.17 is equivalent to evaluating it by Equation 3.16, and the resulting water content is thus conservative (i.e., a global mass balance has been maintained). Once the average volumetric water content for a cell has been evaluated, the corresponding head at the end of the time step must be re-evaluated. This is done by noting that the sum of water over each object in a cell is equal to the total water in that cell. Using the water retention relations  $\Theta^k(h)$  for each object  $k$  in the cell to evaluate the total water in that cell at the end of the time step gives

$$\sum_{k=1}^{n_{\text{objs}}} A_{i,j}^k \Theta^k(h_{i,j}^{m+1}) = A_{i,j} \theta_{i,j}^{m+1} \quad (3.18)$$

For cells with only one object, the water retention relation can be used to explicitly solve for  $h_{i,j}^{m+1}$ , given the water content. For cells with multiple objects, Equation 3.18 is implicit in  $h_{i,j}^{m+1}$  and must be solved through iteration. Here we use a Newton-Raphson algorithm. Fortunately, the number of cells that contain multiple objects is generally small relative to those that do not, and the resulting computational burden for the multiple object cells is relatively small. Note that  $h_{i,j}^{m+1}$  generally will not be equal to  $h_{i,j}'$ . However, the  $h_{i,j}^{m+1}$  will be consistent with the total water mass in the cell.

The above procedure must be modified slightly if saturated cells are present. Following Kirkland et. al. (1992), we use Equation 3.17 only if that cell, and the cells on either of the four sides, are unsaturated. Otherwise, we set

$$h_{i,j}^{m+1} = h_{i,j}' \quad (3.19)$$

### 3.6 Fluxes Between Polygon Objects

Since net water movement across polygon segments is often of interest to the user, such a feature has been added to the present algorithm. The approach used here to evaluate the water movement across a polygon segment for a time step is to perform a mass balance on those cells containing the segment using the fluxes into and out of the cells. This approach has the advantage that it is consistent with the algorithm used for the Richards equation solver. The alternative approach of using estimates of the normal gradients and hydraulic conductivities across segments requires estimating these gradients and conductivities. This is not a trivial matter and will give fluxes that are not consistent with the water movement as evaluated by the Richards equation solver.

## Theory

Consider the cell  $i,j$  that contains a polygon segment bounding two objects as shown in Figure 3.1. Performing a mass balance for just that part of the cell that contains object  $k$  (see Equation 3.11) and using volumetric water contents for the time storage term rather than heads gives:

$$\begin{aligned}
 A^k_{i,j} \frac{\theta_{i,j}^{m+1} - \theta_{i,j}^m}{\Delta t} &= -K^k_{i-1/2,j} Lx^k_{i-1/2,j} \frac{h_{i,j}' - h_{i-1,j}'}{\Delta x_{i-1/2}} \\
 &\quad + K^k_{i+1/2,j} Lx^k_{i+1/2,j} \frac{h_{i+1,j}' - h_{i,j}'}{\Delta x_{i+1/2}} \\
 &\quad - K^k_{i,j-1/2} Ly^k_{i,j-1/2} \frac{h_{i,j}' - h_{i,j-1}'}{\Delta y_{j-1/2}} \\
 &\quad + K^k_{i,j+1/2} Ly^k_{i,j+1/2} \frac{h_{i,j+1}' - h_{i,j}'}{\Delta y_{j+1/2}} \\
 &\quad + K^k_{i,j-1/2} Ly^k_{i,j-1/2} - K^k_{i,j+1/2} Ly^k_{i,j+1/2} \\
 &\quad + \left( \sum_{l=1}^{n_{bndsegk}} q_l B_l \right) + Q_{segs}^k_{i,j}
 \end{aligned} \tag{3.20}$$

where  $n_{bndsegk}$  is the number of boundary segments that are contained in cell  $i,j$  and bound the object  $k$ .  $Q_{segs}^k_{i,j}$  is the total water that enters the object through all internal polygon segments (segments aligning with the boundary segments are handled by the summation term) that lie within or along the  $i,j$  cell or along its sides. Thus, we can solve for the water flow through the polygon segments that lie within or along the cell sides but not on the boundaries.

$$\begin{aligned}
 Q_{segs}^k_{i,j} &= A^k_{i,j} \frac{\theta_{i,j}^{m+1} - \theta_{i,j}^m}{\Delta t} + K^k_{i-1/2,j} Lx^k_{i-1/2,j} \frac{h_{i,j}' - h_{i-1,j}'}{\Delta x_{i-1/2}} \\
 &\quad - K^k_{i+1/2,j} Lx^k_{i+1/2,j} \frac{h_{i+1,j}' - h_{i,j}'}{\Delta x_{i+1/2}} \\
 &\quad + K^k_{i,j-1/2} Ly^k_{i,j-1/2} \frac{h_{i,j}' - h_{i,j-1}'}{\Delta y_{j-1/2}} \\
 &\quad - K^k_{i,j+1/2} Ly^k_{i,j+1/2} \frac{h_{i,j+1}' - h_{i,j}'}{\Delta y_{j+1/2}} \\
 &\quad - K^k_{i,j-1/2} Ly^k_{i,j-1/2} + K^k_{i,j+1/2} Ly^k_{i,j+1/2} \\
 &\quad - \left( \sum_{l=1}^{n_{bndsegk}} q_l B_l \right)
 \end{aligned} \tag{3.21}$$

Equation 3.21 gives the total water that moves into object  $k$  of cell  $i,j$  through the polygon segments that do not lie on a boundary. Multiple segments may lie in or along the cell. To approximate the water movement through just one segment in or along a cell, we distribute the  $Q_{segs}^k_{i,j}$  among the segments proportionally, based on the segment lengths. Note that this is an approximation.

### 3.7 Summary

The solution procedure for a time step is summarized below:

1. Assemble the system of matrix equations for a time step using Equations 3.11 through 3.15.
2. Solve the resulting set of equations for  $h'$  for each cell.
3. Use Equation 3.17 to evaluate a mass-conservative, volumetric water content for each cell at the end of the time step.
4. Use Equation 3.18 to find the corresponding mass-conservative, end-of-time-step head for those cells that are not saturated or adjacent (along one of the four sides) to a saturated cell, and use Equation 3.19 for those cells that are saturated or next to a saturated cell.

The computational effort to evaluate the various lengths and areas used in the above equations is significant and should not be done for each time step. Here we use a preprocessor to evaluate these lengths and areas given the geometry defined by the user. These lengths and areas are then provided to the Richards equation solver, which implements steps 1 through 4 above.

The implementation of the preprocessor and the water solver are discussed in the following two chapters.

Theory

## 4.0 Preprocessor

### 4.1 Overview

The preprocessor **Watermap** utilizes the user-supplied geometry file to map the problem geometry onto the computational grid. **Watermap** outputs this information in an ASCII file that is named by the user. **Watermap** also outputs a FORTRAN include file (named INCLUDE.F) that defines the array sizes used in the Richards equation solver, **Water**. The program flow between the main modules of **Watermap** is discussed in this chapter. Since this discussion follows the actual program structure closely, it should be used along with a program listing if the user wishes to further study the implementation of the preprocessor. Listed below is the program flow for the main routine and a discussion of several of the larger, multipurpose subroutines (i.e., those denoted in **bold**).

### 4.2 MAIN

The main routine initializes the arrays and controls basic program flow. The input file is processed in the order that it is read. First, the underlying computational grid is read from the user-supplied geometry file, processed, and written to the output file. Secondly, the boundary polygon is read, processed, and written to the output file. Next, each of the user-defined internal objects is read, processed, and written to the output file. Next, information on multiple objects in single cells is written to the output file. Finally, the FORTRAN include file is created. The specific tasks performed by the **MAIN** routine are listed below in the order of their execution.

- Initialize arrays, open input and output files.
- Read computational mesh information from input file (x-direction first), check for consistency, evaluate grid side locations (XG, YG), and write to the output file.
- Read the corner points for the boundary polygon from the input file.
- CALL **FILL**: Evaluate the area of each cell and the length of cell sides contained within the boundary polygon. The cell areas are used to initialize the coverage check arrays. This boundary information is not used by the Richards equation solver but is used by the preprocessor to check if the internal objects completely cover the simulation domain.
- CALL **BOUNDINFO**: Evaluate boundary condition information for the boundary polygon and write the results to the output file. This information is used for the evaluation of Equations 3.12 through 3.14. This routine also outputs the centroids of all cells and the indices of those cells outside the simulation domain for use by Equation 3.15.
- CALL **ADDCHAR**: Add boundary polygon characters to character output map. This object map is printed at the end of the preprocessing and shows the user which cells are covered by which polygon objects.
- Read the number of internal polygon objects, NOBJ.
- Loop through each internal polygon.
  - Read the internal polygon corner points from the input file and re-orient if necessary. The evaluation of outward normals in **BNDFLUX** requires a consistent orientation.
  - CALL **FILL**: Evaluate the area of each cell and the length of cell sides contained within the internal polygon. This information is used by Equation 3.11.
  - CALL **SUBTRACT**: Subtract the cell areas evaluated from the check arrays and store the results in the check arrays to see if the polygon is fully contained within the boundary. If the polygon overlaps the boundary or a previous internal polygon, then the check array will contain negative values and an error message will be printed to the screen. Only the cell areas (not the cell side lengths) are updated and checked.
  - CALL **ADDCHAR**: Add polygon characters to the appropriate location in the character output map.
  - CALL **ADDOBJ**: Identify those cells with multiple objects and store the appropriate object information in storage arrays. This information will be written to the output file after this loop is completed and is used by the Richards equations solver to evaluate Equation 3.18.
  - CALL **BNDFLUX**: Evaluate cell information to be used for interobject flux calculations.
  - CALL **OBJECTOUT**: Output the remaining information for the present polygon.
- End loop through internal polygon objects.
- CALL **OUTPUT**: Output the character array showing the location of all objects on the computational mesh.
- CALL **COVERED**: Check to see that the entire simulation domain is covered by the internal objects. This is done by testing whether the check array is near zero.
- CALL **OBJECTNROUT**: Outputs the information processed and stored by **ADDOBJ** for use in evaluating Equation 3.18.

## Preprocessor

- Evaluate the array sizes for the water solver and output these as a FORTRAN include file.
- Stop execution of the preprocessor.

### 4.3 BNDFLUX

This subroutine evaluates the information required by the Richards equation solver to evaluate the water transferred between internal objects during the simulation. This routine is called separately for each internal polygon. Specific tasks and their order are listed below.

- Initialize the cell side length arrays to zero.
- Loop through each segment of the polygon.
  - Evaluate whether the polygon segment intersects cells (ISEGTYPE=1), or if it lies along vertical (ISEGTYPE=2) cell sides, or horizontal (ISEGTYPE=3) cell sides.
  - If the segment intersects cells, then
    - Loop through all cells within the loop bounds established in **FILL** for this object.
      - Define the x-y coordinates of the four corner points for a cell.
      - Evaluate the area of the cell and the cell side lengths.
      - CALL SEGMENTS: This routine uses the incoming vectors of x-y (stored in X and Y) coordinates of the polygon segments, evaluates if and where the polygon segments intersect cell sides, and expands the coordinate vectors to include these intersection points. This expanded vector is stored in XS, YS. The expanded polygon has the same shape, but includes collinear points associated with the cell intersections, which greatly simplifies the following operations.
      - Loop through all the segments defined by XS, YS and if the segment lies in the cell but not on the boundary, evaluate and store the segment length of that part of the segment contained within the cell.
    - End loop
  - Else if the segment lies along a vertical cell side, then
    - Loop through all cells within loop bounds along appropriate vertical.line
      - Evaluate which side of the cell and segment area for that part of the segment that lies along the cell.
    - End loop through cells along vertical.line
  - Else if the segment lies along a horizontal cell side, then
    - Loop through all cells within loop bounds along appropriate horizontal.line
      - Evaluate which side of the cell and segment area for that part of the segment that lies along the cell.
    - End loop through cells along horizontal.line
  - End if
  - Write results to the output file for all cells that contained that segment.
- End loop through polygon segments and return to **MAIN**.

### 4.4 BOUNDINFO

This subroutine evaluates boundary condition information for the boundary polygon and writes the results to the output file. This information is used for the evaluation of Equations 3.12 through 3.14. This routine also evaluates and outputs the centroids of all cells and the indices of those cells outside the simulation domain for use by Equation 3.15. Specific tasks and their order are listed below.

- Evaluate default centroids for all cells assuming the polygon covers all cells. Centroids of cells not completely covered by the boundary polygon are corrected below.
- Loop through each cell within the loop limits. Use loop limits evaluated from previous call to **FILL** for the boundary polygon.
  - Define the x-y coordinates of the four corner points for a cell.
  - Evaluate the area of the cell and the cell side lengths.

- **CALL SEGMENTS:** This routine uses the incoming vectors of x-y coordinates (X and Y) of the polygon segments, evaluates if and where the polygon segments intersect cell sides, and expands the coordinate vectors to include these intersection points. This expanded vector is stored in XS, YS. The expanded polygon has the same shape, but includes collinear points associated with the cell intersections which greatly simplifies the following operations.
- Project the coordinates of XS, YS that lie outside the cell onto the cell sides and store the results in XT, YT. The projection is performed such that the new polygon XT, YT represents the intersection of the cell area and the incoming polygon object area.
- If the cell is next to or includes the boundary, then
  - **CALL CENTROIDS** to correct the centroid location of that part of the cell within the polygon XT, YT.
  - **CALL BNDINFO** to evaluate the information required for Equations 3.12 and 3.14 such as the cell area,  $\Delta z$ , and the normal vector  $\mathbf{n}_l$ . BNDINFO loops through each boundary segment XS, YS and evaluates this information for each segment in the cell or each segment along the four cell sides.
- End if
- End loop through cells and return to **MAIN**.

## 4.5 FILL

This subroutine evaluates internal cell information for the incoming polygon object including cell areas and cell side lengths contained within the object. Given the incoming polygon, this routine evaluates loop limits to determine which cells to loop through. The routine loops through each cell, evaluates the common area between the cell and the polygon, and evaluates the corresponding cell side lengths for those cells with non-zero common areas. Specific tasks and their order of performance are listed below:

- Initialize cell areas and the x-direction and y-direction cell side lengths to zero.
- Evaluate the loop limits for the smallest rectangular region that fully contains the polygon object. For small objects, defining loop limits results in very substantial savings in CPU time.
- Loop through each cell within the loop limits.
  - Define the x-y coordinates of the four corner points for the cell.
  - Evaluate the total area of the cell and the cell side lengths.
  - **CALL SEGMENTS:** This routine uses the incoming vectors of X, Y coordinates of the polygon segments, evaluates if and where the polygon segments intersect cell sides, and expands the coordinate vectors to include these intersection points. This expanded vector is stored in XS, YS. The expanded polygon has the same shape, but includes collinear points associated with the cell intersections. The expanded polygon greatly simplifies the following operations.
  - Project the coordinates of XS, YS that lie outside the cell onto the cell sides and store the results in XT, YT. The projection is performed such that the new polygon XT, YT represents the intersection of the cell area and the incoming polygon object area.
  - **CALL AREASUB:** Evaluate the intersection area by evaluating the area contained within XT, YT. Normalize the results by the full cell area and store in AREAC. AREASUB evaluates the area by integrating around the polygon sides. Since AREASUB is orientation dependent, the absolute value of the results is evaluated to insure that the area is positive.
  - If the normalized AREAC just evaluated is unity, then
    - the cell sides will be all included within the incoming polygon. Set the normalized lengths of these cell sides to unity if they do not lie on the boundary polygon XB, YB. The boundary areas are accounted for elsewhere.
  - Else if the normalized AREAC is greater than some small EPS, then
    - the normalized lengths of the cell sides within the incoming polygon may not be unity and must be evaluated. **CALL AREAS** to evaluate the normalized length of each of the four cell sides contained within the extended incoming polygon XS, YS, but not lying on the boundary polygon XB, YB. The boundary areas are accounted for elsewhere.
  - End if
- End loop through cells and return to **MAIN** program. The results from this subroutine will be written to the output file in the **MAIN** program.

## Preprocessor

## 5.0 Richards Equation Solver

### 5.1 Overview

The Richards equation solver **Water** uses the output file from the preprocessor **Watermap** and user-supplied subroutines to simulate the transient Richards equation in two dimensions. During compilation, the INCLUDE.F file generated by **Watermap** is incorporated into the **MAIN** routine of **Water** to define array dimensions for the large arrays. This minimizes storage requirements, which can reduce CPU time on small or multi-user machines. The program flow between the main modules of **Water** is discussed in this chapter. Users should use the following discussion with the program listing to study the program implementation in more detail. Listed below is the program flow for the main routine. This listing also includes brief discussions of the purpose of the called subroutines. Additional discussions of the larger, multi-purpose subroutines (those subroutines denoted in **bold**) follow the discussion of the main routine.

### 5.2 MAIN

The main routine initializes the arrays, inputs the data generated by the preprocessor, and controls basic program flow for the Richards equation solver. The specific tasks performed by the **MAIN** routine and their order are listed below.

- Ask the user for the input file name and time-step information.
- **CALL INPUT**: Read the preprocessed file.
- **CALL VG\_INIT**: Initialize the van Genuchten parameter array. The subroutine **VG\_INIT** loops through the internal polygon objects, calling the user-supplied routine **VG\_PROP** for each object.
- **CALL VG\_INIT\_NR**: Initialize the Newton Raphson van Genuchten parameter array for those cells with multiple objects. **VG\_INIT\_NR** loops through the multiple object cells in the outer loop and through the objects in that cell in the inner loop. The user-supplied routine **VG\_PROP** is called from within the inner loop. This information is used with a Newton Raphson iterative method to evaluate Equation 3.18 for head for those cells with multiple objects. While this information is also scattered throughout the van Genuchten parameter array initialized by **VG\_INIT**, it is also stored here in a format that is computationally more efficient to access.
- **CALL VG\_INIT\_BND**: Initialize the boundary van Genuchten parameter array for those cells that contain a boundary. **VG\_INIT\_BND** loops only through those cells that contain a boundary segment. The user-supplied routine **VG\_PROP** is called from within the loop. These properties are used for flux and unit gradient boundary conditions (i.e., Equations 3.12 and 3.14). While this information is also scattered throughout the van Genuchten parameter array initialized by **VG\_INIT**, it is also stored in a format that is computationally more efficient to access.
- Using  $h=0$  and  $h=-1.0E-6$ , evaluate the corresponding water contents for the cells using the subroutine **H\_TO\_THEA**. These maximum and minimum water contents are used for adaptive time-step purposes only.
- Call the user-supplied **H\_INIT** routine defining the initial head distribution, initialize miscellaneous dynamic variable arrays, and call **H\_TO\_THEA** to evaluate the corresponding initial water contents for each cell.
- Loop through each time step.
  - Define the head array (**HPROP**), which is used to evaluate the van Genuchten properties. Since Picard iteration is not used, **HPROP** is set equal to the head at the beginning of the time step. The program can be easily modified to perform Picard iteration using **HPROP**. However, for the test problems considered when developing this program, Picard iteration did not improve the performance of the algorithm.
  - **CALL MATRIX**: Use Equations 3.11 through 3.15 to assemble the set of linear algebraic equations that defines the approximate head  $h$  at the end of the time step.
  - **CALL CONGRAD**: Solve the resulting symmetric set of equations for  $h'$  using a preconditioned conjugate gradient solver with an incomplete  $LDL^T$  preconditioner.
  - **CALL UPDATE**: Use  $h'$  in Equation 3.17 to update the water contents at the end of the time step and to update the time step size. If the previous time step was found to be too large, return to the top of this loop and re-evaluate the equations for the present time step. Otherwise, continue. Using the updated water contents, use Equations 3.18 and 3.19 to update head for the end of the time step.
  - **CALL UPDATEFLUX**: Update the fluxes between cells.
  - **CALL UPDATEFLUXSEGS**: Update the fluxes through the polygon segments.
  - If the present time is at a user-specified output time, **CALL OUTPUT** (a user-supplied subroutine) and **OUTPUT-FLUX** to output the dynamic variables and the net water movement through the various polygon segments.
- End loop through time steps.

## Richards Equation Solver

- Stop execution of the Richards equation solver.

### 5.3 MATRIX

This routine is the heart of the Richards equation solver. It assembles the matrix equations object by object, flux by flux. Doing so results in a CPU-efficient assembly algorithm that can handle irregularly shaped objects and boundaries. The specific tasks performed by **MATRIX** and their order of execution are listed below.

- CALL BNDFCN: Call the user-supplied subroutine defining the time dependence of the boundary conditions.
- Set the coefficient matrix to zero.
- Loop through each internal polygon object.
  - CALL DYNPROPS to update the coefficients corresponding to mass storage term and the fluxes across cell sides within an object, and across cell sides that coincide with the object's polygon boundary. DYNPROPS uses simple arithmetic averages to obtain a hydraulic conductivity at a cell side given the hydraulic conductivities on either side at the cell nodes.
  - Accumulate the coefficients just evaluated in DYNPROPS into the work arrays CX, COEF(\*,\*,4), and COEF(\*,\*,5) at the appropriate location. After completing this loop, the values accumulated in these three arrays are the summations associated with the mass storage term, the  $x$ -direction flux terms, and the  $y$ -direction flux terms in Equation 3.11, summed over all objects. Since the present formulation results in a coefficient matrix that is symmetric with five non-zero diagonals, only the lower three diagonals are used (i.e., COEF(\*,\*,1), COEF(\*,\*,2), COEF(\*,\*,3)) by the matrix equation solver.
- End loop through the objects.
- Loop through all cells, accumulating the effective mass storage terms into the main diagonal of the coefficient matrix.
- Loop through all  $x$  direction cell sides, accumulating the  $x$  direction effective flux coefficients (including the appropriate  $\Delta x$ ) into the appropriate locations in the main and the  $x$  direction subdiagonal.
- Loop through all  $y$  direction cell sides, accumulating the  $y$  direction effective flux coefficients into the appropriate locations in the main diagonal, the  $y$  direction subdiagonal, and in the right-hand side.
- Accumulate the terms that account for the boundary conditions into the diagonal and right-hand side using Equations 3.12 through 3.14. Store coefficients in the FBXY array to be used by **UPDATEFLUX** and **UPDATEFLUXSEGS** for evaluating the flux through the polygon segments.
- Define the coefficients corresponding to Equation 3.19 for those cells outside the problem domain.
- The coefficient matrix and right-hand side are now complete. Return to the **MAIN** program.

### 5.4 UPDATE

This routine uses the solution of the matrix equation, defined by **MATRIX** for  $h'$ , to update the water contents at the end of the time step and to update the time-step size. Given the end-of-time-step water contents, this routine then uses Equations 3.18 and 3.19 to evaluate the corresponding  $h^{m+1}$  at the end of the time step.

- Accumulate the cell areas of the various objects into a work array to find  $A_{i,j}$  in Equation 3.17. Use Equation 3.17 to find the average cell volumetric water content at the end of the time step.
- Re-evaluate the time-step size. If the time step is reduced, set a flag to repeat this time step at the reduced value when the program returns to **MAIN**.
- Mark the cells that are saturated or are adjacent to saturated cells (see discussion following Equation 3.18).
- Loop through the objects calling THETA\_TO\_H. This routine loops through all cells that contain an object and updates the heads  $h^{m+1}$ , given the water contents for those cells that contain only one object and are not marked as saturated or adjacent to a saturated cell.
- Loop through all cells, calling THETA\_TO\_H\_NR if the cell contains multiple objects and if the cell is not marked as saturated or adjacent to a saturated cell. THETA\_TO\_H\_NR updates the head to  $h^{m+1}$  using Equation 3.18 and Newton-Raphson iteration (allowing a maximum of five iterations).
- Return to the **MAIN** program.

## 5.5 UPDATEFLUX

This routine uses the  $h'$  to evaluate the water fluxes through the cell sides for the present time step. **UPDATEFLUX** does this in order as follows.

- Initialize the cell side flux arrays to zero.
- Loop through all polygon objects.
  - Evaluate the mass flow through each cell side for those sides corresponding to the  $L_x^k$  and  $L_y^k$  lengths of Figure 3.1 and accumulate these into the flux arrays.
  - Evaluate the mass flow through each cell side for those sides corresponding to the  $L_x^{k,k^*}$  and  $L_y^{k,k^*}$  lengths of Figure 3.1 and accumulate these into the flux arrays.
  - Evaluate the flows through each boundary segment into each boundary cell using the coefficients FBXY defined in MATRIX.
  - Convert the cell side mass flows to fluxes based on full cell area and the time-step size.
- End loop through the polygon objects and return to **MAIN**.

## 5.6 UPDATEFLUXSEGS

This routine uses the cell side fluxes and the heads to estimate the total water movement through each polygon segment (excluding those along the boundary) and each boundary segment through the present time. **UPDATEFLUXSEGS** does this using Equation. 3.21 in order as follows.

- If the first call to **UPDATEFLUXSEGS**, then
  - Loop through cells for which there are multiple objects and CALL H\_TO\_THEA\_UP to evaluate the volumetric water contents at the beginning of the time step for each object in the cell.
- Else
  - Set water contents for the beginning of the time step to those obtained at the end of the previous time step.
- End if
- Loop through cells for which there are multiple objects and CALL H\_TO\_THEA\_UP to evaluate the volumetric water contents at the end of the time step for each object in the cell. The old and new water contents for those cells with one object are already available.
- Loop through all internal polygon objects
  - Set the residual matrix RESID to zero. When complete, RESID in **UPDATEFLUXSEGS** corresponds to  $Q_{segs}^{k,i,j}$  in Equation 3.21.
  - Add the contribution of the transient term in Equation 3.21 for cells with multiple objects to RESID. The total water added into a polygon object is also updated.
  - Add the contribution of fluxes associated with the  $L_x^k$  and  $L_y^k$  lengths in Figure 3.1 into RESID.
  - Add the contribution of boundary fluxes into RESID. The total water through each boundary segment is also updated.
- End the loop through all internal polygon objects.
- RESID for each cell now contains the water that passed through all the polygon segments into that cell. Partition RESID to each polygon segment according to the relative length of each segment in that cell (this is an approximation).
- Return to **MAIN**.

## Richards Equation Solver

## 6.0 Example Problems

Three test problems are presented. The first two are used to test the polygon-based algorithm against the flux updating algorithm of Kirkland et al. (1992). The first test problem assumes a simple uniform soil on a rectangular simulation domain with water flux applied over a subregion on the top boundary and no flux conditions applied elsewhere.

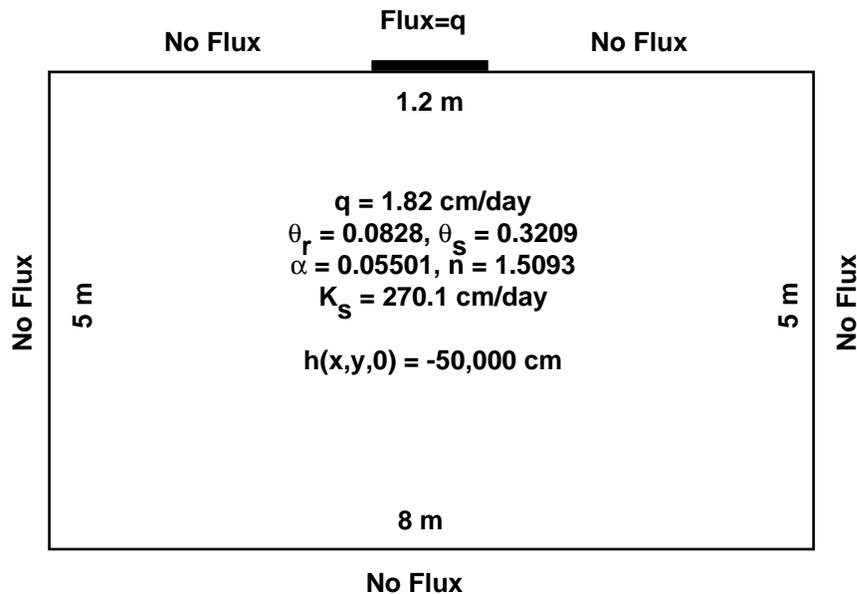
The same domain and similar boundary conditions are used for the second test problem. In this case, the domain contains a block of sandy loam surrounded by clayey loam. The flux updating algorithm models this region as a simple block of sandy loam within the clayey loam. Two simulations of this problem are performed by the polygon-based algorithm. In one, the domain is modeled as one polygon object containing a spatial distribution of soil properties (i.e., block of sandy loam surrounded by clayey loam). In the other, the rectangular block of sandy loam is bisected by one of the diagonals to form two triangular objects. In addition, the clayey loam is divided into three polygon objects. The resulting domain is covered by five polygon objects, with the properties of either sandy or clayey loam. This case tests the ability of the polygon based algorithm to predict flow between non-trivial objects and yet allows the finite difference based, flux updating algorithm to be used for comparison.

In the final problem, the complex geometry associated with a barrier cover for waste isolation is modeled. This example is based on the barrier cover presented in Meyer (1993) and is not presented as a test case. It is presented to show additional features of the polygon-based code such as head boundary conditions, a sloped upper boundary, and a complex arrangement of internal polygons.

### 6.1 Example 1: Uniform Soil

The first example problem is based on the Plot 2b experiment at the Las Cruces Trench Site and uses the uniform soil model for that site (Wierenga et al., 1991). This problem is illustrated in Figure 6.1. A constant flux of 1.82 cm/day is applied over a 1.2 m length of the top boundary. The residual volumetric water content,  $\theta_r$ , is 0.0828; saturated water content,  $\theta_s$ , is 0.3209. The soil hydraulic properties are modeled using the van Genuchten (1980b) relationship with parameters,  $\alpha=0.05501 \text{ cm}^{-1}$  and  $n=1.5093$  (the code assumes the third parameter of the van Genuchten model,  $m=1-1/n$ ). The saturated hydraulic conductivity,  $K_s$ , is 270.1 cm/day. The initial pressure head is -50,000 cm throughout the domain.

The geometric input file used to solve this problem is listed in Section 6.1.1. The assumed spatial and temporal units are centimeters and days. Note from the input file that the problem is solved over the domain  $x=-400 \text{ cm}$  to  $400 \text{ cm}$  and  $y=0$  to  $500 \text{ cm}$ . The finite volume cells are each 10 cm by 10 cm. All boundary segments are of the flux type and are bounded by only one



**Figure 6.1. Model domain and van Genuchten parameters for uniform soil example. Simulation domain is 8-m wide by 5-m deep. Water flux is applied over a 1.2-m wide strip centered on the top boundary.**

## Example Problems

internal object (polygon object #1). All fluxes are zero except for that applied over the boundary segment (segment #2), which starts at (-60,0) and ends at (60,0). Since the interior is modeled by only one object, the same polygon points are used to define the internal polygon object. The domain must be fully covered by the internal polygon objects. The corresponding user-supplied subroutines are listed in Section 6.1.2. The spatial, temporal, head, and flux units used in Section 6.1.2 are cm, days, cm, and cm/day. A consistent set of units must be used.

### 6.1.1 Input File for Example 1

```
1 : number of x regions
-400 400 80 : x1 x2 nx - region 1

1 : number of y regions
0 500 50 : y1 y2 ny - region 1

6 : Number of points for boundary polygon followed by x-y pairs, type, adjacent object #
-400 0 2 1
-60 0 2 1
60 0 2 1
400 0 2 1
400 500 2 1
-400 500 2 1

1 : Total number of internal objects (excluding boundary object)

6 : Number of points for object 1 followed by x-y pairs
-400 0
-60 0
60 0
400 0
400 500
-400 500
```

### 6.1.2 User-Supplied Subroutines for Example 1

```
C
C ***** User-supplied subprograms for water solver *****
C

      SUBROUTINE BNDFCN (TIME,NBS,FORCE)
C
C      This is a user-supplied subroutine defining the forcing
C      function for each of the NBS boundary segments as a
C      function of time. The forcing functions can
C      represent head or flux. In this case, they all represent
C      flux.
C
      DIMENSION FORCE(NBS)

      DO I=1,NBS
        FORCE(I) = 0.0
      END DO

      FORCE(2) = 1.82

      RETURN
      END
```



## Example Problems

```
WRITE(NFILEOUT,*) (XCG(I,1),I=1,NX)
WRITE(NFILEOUT,*) THETA
```

```
RETURN
END
```

```
SUBROUTINE VG_PROP (N,IOBJ,X,Y,THEAR,THEAS,ALPHA,AN,AKS)
```

```
C
C      This is a user-supplied subroutine defining the van Genuchten
C      properties for each object. The user is supplied with the object
C      number and the x, y coordinates of the centroid of each cell that
C      includes that object. The user should then return the corresponding
C      vectors of the van Genuchten properties. Notes: 1) While a centroid
C      will never lie outside a boundary, it may lie outside an object
C      even though the object intersects part of the cell. The user should
C      still provide the corresponding van Genuchten properties even though
C      the centroid x, y coordinates may be just outside the object's
C      polygon. The program uses these values to estimate average values
C      for that region at the cell interface covered by the object.
C      2) This routine will be called (actually, multiple calls for
C      sub regions within an object) only during the initialization of
C      the problem.
```

```
C
C          N - Number of cells containing the object (input)
C          IOBJ - Object number (input)
C          X, Y - Vector of coordinates of centroid of each cell
C                containing the object (input)
C      THEAR - Vector of corresponding residual water contents
C      THEAS - Vector of corresponding saturated water contents
C      ALPHA - Vector of corresponding V-G alpha parameters
C          AN - Vector of corresponding V-G n parameters
C          AKS - Vector of corresponding saturated hydraulic
C                conductivities
```

```
C
C      DIMENSION X(N), Y(N), THEAR(N), THEAS(N), ALPHA(N), AN(N), AKS(N)
```

```
C
C      ***      In the example that follows, one object is defined with the
C      ***      properties constant across the object (hence, there is no need to
C      ***      use the cell centroids X, Y).
```

```
C
C      IF (IOBJ .EQ. 1) THEN
```

```
C      ***      Uniform soil
```

```
C
C      DO I=1,N
C          THEAR(I) = 0.0828
C          THEAS(I) = 0.3209
C          ALPHA(I) = 0.05501
C          AN(I) = 1.5093
C          AKS(I) = 270.1
C      END DO
```

```
C
C      ELSE
```

```
C          WRITE(*,*) ' Information for an object number was requested'
C          WRITE(*,*) ' which was not supplied by the user.'
C          STOP ' Execution terminated'
```

END IF

RETURN  
END

## 6.2 Example 2: Blocked Soil

The second example problem uses the geometry of the first example, but incorporates a block of sandy loam surrounded by clay loam. Boundary conditions are identical to the first example with the exception of the value of the input flux, which is increased to 5.0 cm/day. The hydraulic properties used for this example are from Kirkland et al. (1992). The hydraulic parameters for the clay loam are  $\theta_r = 0.1060$ ,  $\theta_s = 0.4686$ ,  $\alpha = 0.0104 \text{ cm}^{-1}$ ,  $n = 1.3954$ , and  $K_s = 13.1 \text{ cm/day}$ . (Symbols are defined in Section 6.1.) The corresponding parameters for the sandy loam are  $\theta_r = 0.0286$ ,  $\theta_s = 0.3658$ ,  $\alpha = 0.0280 \text{ cm}^{-1}$ ,  $n = 2.2390$ , and  $K_s = 541.0 \text{ cm/day}$ . The initial pressure head is -50,000 cm throughout the domain.

This example was simulated two ways. Example 2a uses multiple polygon objects with homogeneous properties. Example 2b uses a single polygon object with heterogeneous properties specified in the user-supplied subroutine VG\_PROP.

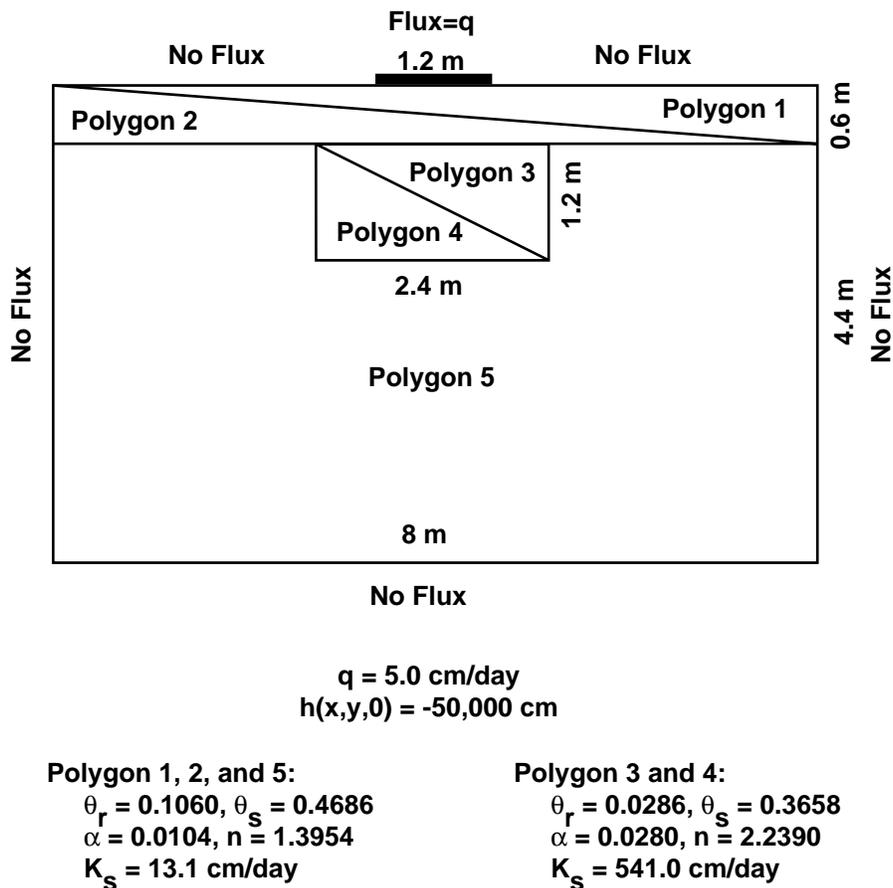


Figure 6.2. Model domain and van Genuchten parameters for a blocked soil example. Simulation domain is 8 m wide by 5 m deep. Water flux is applied over a 1.2-m wide strip centered on the top boundary. Polygons 3 and 4 form a rectangle 2.4-m wide by 1.2-m deep with the top centered 0.6 m below the irrigated area. Polygons shown are for Example 2a. Example 2b uses a single polygon with heterogeneous properties.

## Example Problems

### 6.2.1 Example 2a: Multiple homogeneous polygon objects

Example 2a is illustrated in Figure 6.2. Note that several of the polygons combine to form equivalent rectangular blocks. This allows the flux updating algorithm of Kirkland et. al. (1992) to be used to test the polygon-based algorithm. The corresponding geometric input file is listed in Section 6.2.1.1. The assumed spatial and temporal units are cm and days. Thus, the head and flux units used are cm and cm/day. A consistent set of units must be used. The entire model domain (as defined by the boundary polygon) must be completely covered by internal polygons. The VG\_PROP subroutine used to define the van Genuchten properties for the five polygon objects is listed in Section 6.2.1.2. The remaining subroutines are the same as those used for the uniform soil example discussed previously with the following exception: the input flux is changed from 1.82 cm/day to 5.0 cm/day in subroutine BNDFCN.

#### 6.2.1.1 Input File for Example 2a

```
1      :      Number of delta x regions
-400 400 80      : x1 x2 nx - region 1

1      :      Number of delta y regions
0 500 50 : y1 y2 ny - region 1

8 : Number of points for boundary polygon followed by x-y pairs, type, adjacent object #
-400 0 2 1
-60 0 2 1
60 0 2 1
400 0 2 1
400 60 2 5
400 500 2 5
-400 500 2 5
-400 60 2 2

5 : Total number of internal polygons

5 : Number of points for polygon 1 followed by x-y pairs
-400 0
-60 0
60 0
400 0
400 60

5 : Number of points for polygon 2 followed by x-y pairs
400 60
120 60
-120 60
-400 60
-400 0

3 : Number of points for polygon 3 followed by x-y pairs
-120 60
120 60
120 180

3 : Number of points for polygon 4 followed by x-y pairs
-120 60
120 180
-120 180

8 : Number of points for polygon 5 followed by x-y pairs
-400 60
-120 60
-120 180
120 180
```

```

120 60
400 60
400 500
-400 500

```

### 6.2.1.2 User Subroutine VG\_PROP for Example 2a

```

SUBROUTINE VG_PROP (N,IOBJ,X,Y,THEAR,THEAS,ALPHA,AN,AKS)
C
C      This is a user-supplied subroutine defining the van Genuchten
C      properties for each object. The user is supplied with the object
C      number and the x, y coordinates of the centroid of each cell that
C      includes that object. The user should then return the corresponding
C      vectors of the van Genuchten properties. Notes: 1) While a centroid
C      will never lie outside a boundary, it may lie outside an object
C      even though the object intersects part of the cell. The user should
C      still provide the corresponding van Genuchten properties even though
C      the centroid x, y coordinates may be just outside the object's
C      polygon. The program uses these values to estimate average values
C      for that region at the cell interface covered by the object.
C      2) This routine will be called (actually, multiple calls for
C      sub regions within an object) only during the initialization of
C      the problem.
C
C      N - Number of cells containing the object (input)
C      IOBJ - Object number (input)
C      X, Y - Vector of coordinates of centroid of each cell
C             containing the object (input)
C      THEAR - Vector of corresponding residual water contents
C      THEAS - Vector of corresponding saturated water contents
C      ALPHA - Vector of corresponding V-G alpha parameters
C      AN - Vector of corresponding V-G n parameters
C      AKS - Vector of corresponding saturated hydraulic
C            conductivities
C
C
C      DIMENSION X(N), Y(N), THEAR(N), THEAS(N), ALPHA(N), AN(N), AKS(N)
C
C      ***      In the example that follows, five objects are defined with the
C      ***      properties constant across each object (hence, no need to use
C      ***      the cell centroids X, Y).
C
C      IF (IOBJ .EQ. 1 .OR. IOBJ .EQ. 2 .OR. IOBJ .EQ. 5) THEN
C
C      ***      Clayey loam
C
C      DO I=1,N
C          THEAR(I) = 0.1060
C          THEAS(I) = 0.4686
C          ALPHA(I) = 0.0104
C          AN(I) = 1.3954
C          AKS(I) = 13.1
C      END DO
C
C      ELSE IF (IOBJ .EQ. 3 .OR. IOBJ .EQ. 4) THEN
C
C      ***      Sandy loam
C

```

## Example Problems

```
DO I=1,N
  THEAR(I) = 0.0286
  THEAS(I) = 0.3658
  ALPHA(I) = 0.0280
  AN(I) = 2.2390
  AKS(I) = 541.0
END DO

ELSE

  WRITE(*,*) ' Information for an object number was requested'
  WRITE(*,*) ' which was not supplied by the user.'
  STOP ' Execution terminated'

END IF

RETURN
END
```

### 6.2.2 Example 2b: Single heterogeneous polygon object

The domain of Example 2 can also be defined in a much simpler fashion than described above for Example 2a. The same domain can be defined using one internal object with spatially variable properties. In this case, Example 2b, the geometry file is identical to that shown for the Example 1. The VG\_PROP subroutine for Example 2b is given in Section 6.2.2.1. Note that this greatly simplifies the geometry file without complicating the user-supplied van Genuchten subroutine. When the geometry of the internal objects are simple, this method can be used as an alternative to defining multiple objects.

#### 6.2.2.1 User Subroutine VG\_PROP for Example 2b

```
SUBROUTINE VG_PROP (N,IOBJ,X,Y,THEAR,THEAS,ALPHA,AN,AKS)
C
C   This is a user-supplied subroutine defining the van Genuchten
C   properties for each object. The user is supplied with the object
C   number and the x, y coordinates of the centroid of each cell that
C   includes that object. The user should then return the corresponding
C   vectors of the van Genuchten properties. Notes: 1) While a centroid
C   will never lie outside a boundary, it may lie outside an object
C   even though the object intersects part of the cell. The user should
C   still provide the corresponding van Genuchten properties even though
C   the centroid x, y coordinates may be just outside the object's
C   polygon. The program uses these values to estimate average values
C   for that region at the cell interface covered by the object.
C   2) This routine will be called (actually, multiple calls for
C   sub regions within an object) only during the initialization of
C   the problem.
C
C       N - Number of cells containing the object (input)
C       IOBJ - Object number (input)
C       X, Y - Vector of coordinates of centroid of each cell
C             containing the object (input)
C       THEAR - Vector of corresponding residual water contents
C       THEAS - Vector of corresponding saturated water contents
C       ALPHA - Vector of corresponding V-G alpha parameters
C       AN - Vector of corresponding V-G n parameters
C       AKS - Vector of corresponding saturated hydraulic
C            conductivities
C
C   DIMENSION X(N), Y(N), THEAR(N), THEAS(N), ALPHA(N), AN(N), AKS(N)
C
```

```

C
C ***      In the example that follows, one object is defined with the
C ***      properties given as a function of location in the object.
C
C
      DO I=1,N
      IF (X(I) .GE. -120. .AND. X(I) .LE. 120. .AND.
      .   Y(I) .GE. 60. .AND. Y(I) .LE. 180.) THEN
C
C ***      Sandy loam
C
      THEAR(I) = 0.0286
      THEAS(I) = 0.3658
      ALPHA(I) = 0.0280
      AN(I) = 2.2390
      AKS(I) = 541.0

      ELSE
C
C ***      Clayey loam
C
      THEAR(I) = 0.1060
      THEAS(I) = 0.4686
      ALPHA(I) = 0.0104
      AN(I) = 1.3954
      AKS(I) = 13.1

      END IF
      END DO

      RETURN
      END

```

### 6.3 Example 3: Engineered Cover

The last example is the engineered barrier cover already discussed in Chapter 2. This example problem is taken from Meyer (1993). The spatial and temporal units for all variables in this problem are in feet and days. We use these units rather than SI units to demonstrate that other units can be used. The output results will thus have spatial units in feet and days. The geometry for this cover is defined in Figure 2.1 with the geometry file presented in Section 2.2.1 and the user-supplied subroutines (with one exception) presented in Section 2.3.1. The user-supplied subroutine VG\_PROP contains the hydraulic properties for each of the materials shown in Figure 2.1. A constant flux of  $2.835 \times 10^{-4}$  ft/day was applied along the entire top boundary. The lower boundary was held at a constant pressure head of zero. Vertical boundaries are no flow boundaries; the right represents a symmetry condition and the left boundary is assumed to be far enough from the concrete vault that its influence there is negligible (Meyer, 1993). The initial condition given in subroutine H\_INIT is a unit gradient pressure (hanging water column).

We used the user-supplied OUTPUT subroutine given in Section 6.3.1 rather than that given in Section 2.3.1. The user-specified output file is used to output the head distribution in a format suitable for import to the scientific visualization package Transform (Spyglass, Inc., Champaign, IL). Two additional files are opened to output the  $x$  and  $y$  components of flux at the cell centers. Since the fluxes evaluated by the algorithm are those normal to the cell sides, simple averaging was used to define flux components at the cell centers. These fluxes are then normalized by 10 times the flux applied at the surface. This scales the vectors for graphical output. Only one output time (day 1050) was used when running this program. Thus this subroutine is called only once.

#### 6.3.1 User Subroutine OUTPUT for Example 3

```

      SUBROUTINE OUTPUT (NFILEOUT, TIME, H, THETA, FLUXX, FLUXY, NX, NY,
      .                  XG, YG, XCG, YCG)
C
C      This is a user-supplied subroutine defining the output

```

## Example Problems

```
C      format. All arguments are input from the main program. The
C      user should not change the values of any of these arguments.
C
C      NFILEOUT - Unit number for output file
C      TIME - Current time
C      H - Pressure head
C      THETA - Volumetric water content
C      FLUXX - Water flux through cell sides in x-dir
C      FLUXY - Water flux through cell sides in y-dir
C      NX - Number of cells in x direction
C      NY - Number of cells in y direction
C      XG - x coordinates of cell sides (vector)
C      YG - y coordinates of cell sides (vector)
C      XCG - x coordinates of area centroids (array)
C      YCG - y coordinates of area centroids (array)
C
C      DIMENSION H(NX,NY), THETA(NX,NY), XG(NX+1), YG(NY+1)
C      DIMENSION XCG(NX,NY), YCG(NX,NY), FLUXX(NX-1,NY), FLUXY(NX,NY-1)
C      DIMENSION WORK(200,200)
C
C
C ***      Output head distribution to user-defined output file in a
C ***      format suitable for import into the Spyglass visualization
C ***      software package Transform (Spyglass, Inc., Champaign, IL).
C ***      The cell center points (i.e., average of the locations of the
C ***      cell sides) rather than area centroids are used since they
C ***      lie on a rectangular grid pattern. The centroids may not be
C ***      at the center of those cells through which a boundary
C ***      segment passes (upper boundary for this case).
C ***
C
C
C      ZERO = 0.0
C
C      WRITE(NFILEOUT,*) NY, NX
C      WRITE(NFILEOUT,*) ZERO, ZERO
C      WRITE(NFILEOUT,*) ((YG(J+1)+YG(J))/2.0,J=1,NY)
C      WRITE(NFILEOUT,*) ((XG(I+1)+XG(I))/2.0,I=1,NX)
C      WRITE(NFILEOUT,*) H
C
C
C ***      Use a work array to evaluate and output the x and y components
C ***      of the fluxes at the center of each cell in Spyglass format.
C ***      Since FLUXX and FLUY Y are the fluxes at the cell sides,
C ***      simple averages are used to estimate the fluxes at the cell
C ***      centers. Only one output time will be specified when running
C ***      Water.f. As a result, the new files opened below will be
C ***      opened only once during execution.
C
C
C      DO J=1,NY
C        DO I=1,NX
C          WORK(I,J) = 0.0
C        END DO
C      END DO
C
C      DO J=2,NY-1
C        DO I=2,NX-1
C          WORK(I,J) = (FLUXX(I,J) + FLUXX(I-1,J)) / 2.0
C        END DO
C      END DO
C
```

```

C ***      Normalize by 10 times the flux applied at the surface
C ***      (see subroutine BNDFCN).
C
DO J=2,NY-1
  DO I=2,NX-1
    WORK(I,J) = WORK(I,J) / (10.0 * 2.835E-4)
  END DO
END DO

OPEN (UNIT=15,STATUS='UNKNOWN',FILE='Xdir')
  WRITE(15,*) NY, NX
  WRITE(15,*) ZERO, ZERO
  WRITE(15,*) ((YG(J+1)+YG(J))/2.0,J=1,NY)
  WRITE(15,*) ((XG(I+1)+XG(I))/2.0,I=1,NX)
  WRITE(15,*) ((WORK(I,J),I=1,NX),J=1,NY)
CLOSE (UNIT=15)

DO J=2,NY-1
  DO I=2,NX-1
    WORK(I,J) = (FLUXY(I,J) + FLUXY(I,J-1)) / 2.0
  END DO
END DO
C
C ***      Normalize by 10 times the flux applied at the surface
C ***      (see subroutine BNDFCN).
C
DO J=2,NY-1
  DO I=2,NX-1
    WORK(I,J) = WORK(I,J) / (10.0 * 2.835E-4)
  END DO
END DO

OPEN (UNIT=15,STATUS='UNKNOWN',FILE='Ydir')
  WRITE(15,*) NY, NX
  WRITE(15,*) ZERO, ZERO
  WRITE(15,*) ((YG(J+1)+YG(J))/2.0,J=1,NY)
  WRITE(15,*) ((XG(I+1)+XG(I))/2.0,I=1,NX)
  WRITE(15,*) ((WORK(I,J),I=1,NX),J=1,NY)
CLOSE (UNIT=15)

RETURN
END

```

## Example Problems

## 7.0 Results

### 7.1 Introduction

Here we compare the predictions of the polygon-based Richards equation solver to those of the flux updating algorithm of Kirkland et al. (1992). The same spatial discretization ( $\Delta x = \Delta y = 10$  cm) was used by both algorithms for Examples 1 and 2 (see previous chapter), but different temporal step sizes were used. This is because the two algorithms use different adaptive time-step strategies. Since the polygon-based algorithm is designed to be used as a production code, it uses a more conservative adaptive time-step strategy that results in smaller average time steps. In this chapter,  $\theta_{\text{poly}}$  refers to the volumetric water content obtained from the polygon-based algorithm, and  $\theta_{\text{flux}}$  refers to the volumetric water content obtained from the flux updating algorithm of Kirkland et al. (1992). CPU times were intentionally not tabulated for comparisons since the flux updating algorithm is a stripped down, special purpose solver whereas the polygon-based algorithm is a more general purpose solver that can be applied to complex geometries. In general, CPU times for the polygon-based solver were of the same order of magnitude as the flux updating solver.<sup>1</sup> Finally, a contour plot of head and vector plots of flux are presented for an engineered cover [Example 3, see the previous chapter and Meyer (1993)]. This example is presented to show features of the algorithm not represented in the first two examples. The example also shows the effect that a well-engineered capillary break can have on the flow through an engineered cover.

### 7.2 Results and Comparative Analysis for the First Two Test Cases

The preprocessor **Watermap** outputs the mapping of the objects onto the underlying cells. This map is shown in Figure 7.1 for the blocked soil modeled, which is defined by five internal polygon objects. The other maps for this problem are not shown since they contain only one internal object. Each cell is represented by the object number mapping onto that cell. Only the last digit of the object numbers is shown. Cells that contain multiple objects are denoted by an '\*'. As the output map illustrates, this test case contains many cells that are occupied by more than one object.

Contour plots for the first two test problems are shown in Figures 7.2 through 7.4. Figure 7.2 shows contours of volumetric water contents  $\theta_{\text{poly}}$  and  $\theta_{\text{flux}}$  on day 70 for the uniform soil. Figures 7.3 and 7.4 show the volumetric water contents on day 50 for the blocked soil using two polygon-based methods (see the discussion in the previous chapter) to specify the soil properties. The results of the polygon-based algorithm and that of the flux updating algorithm for each test problem show no perceivable difference. There appears to be no distortion of the results due to the triangular-shaped polygons.

To further define the differences between the predictions of the two algorithms, Table 7.1 presents several statistics for the volumetric water content of the flux updating algorithm,  $\theta_{\text{flux}}$ , and for the difference between the algorithm predictions,  $\theta_{\text{poly}} - \theta_{\text{flux}}$ , for the population of nodal values of water content on the 80 by 50 cell grid. As the results of the table indicate, there is little difference between the predictions of the two algorithms. The largest difference is 0.001 for the uniform soil test case. The mean errors are essentially zero, and the standard deviations of the errors are small relative to the standard deviations of the flux updating algorithm predictions. The differences between the algorithm predictions are remarkably small considering the differences in the setup of the matrix equations and the different adaptive time-step strategies for the two algorithms.

### 7.3 Results for the Engineered Cover

The object map generated by **Watermap** for the engineered cover problem is shown in Figure 7.5. The map is shown in a small font so that the full map can appear on one page. Cells that are fully outside the boundary polygon are denoted by a '-'. Note that, as required, the internal polygons cover the entire domain defined by the boundary polygon. In this test problem, water is applied at a uniform flux normal to the inclined upper boundary. The lower boundary is assumed to be at the water table ( $h=0$ ), with the lateral boundaries acting as no-flux boundaries. The initial head distribution is defined by a unit gradient in the pressure head down to the water table.

The Richards equation solver **Water** prints time, time-step size, and the number of iterations for the equation solver for each time step. A summary of water movement through the various polygon segments is also printed to the screen for each output time. This part of the screen output is shown in Figure 7.6 for day 1050. In this output, water flow is the total water that has

---

1. In an independent review of the PolyRES code, Dr. J. T. McCord ran Example 2 using the VS2DT code (Healy, 1990) on a 486 DX2/50 IBM compatible computer and obtained results essentially identical to those presented here. The PolyRES code was approximately 1.9 times faster than VS2DT for this problem (J. T. McCord, Sandia National Laboratories, September 11, 1995). In a benchmarking study of infiltration modeling, McCord and Goodrich (1994) found VS2DT to be 2 to 10 times faster than comparable codes. These results suggest that PolyRES is a competitive code with respect to computational efficiency.



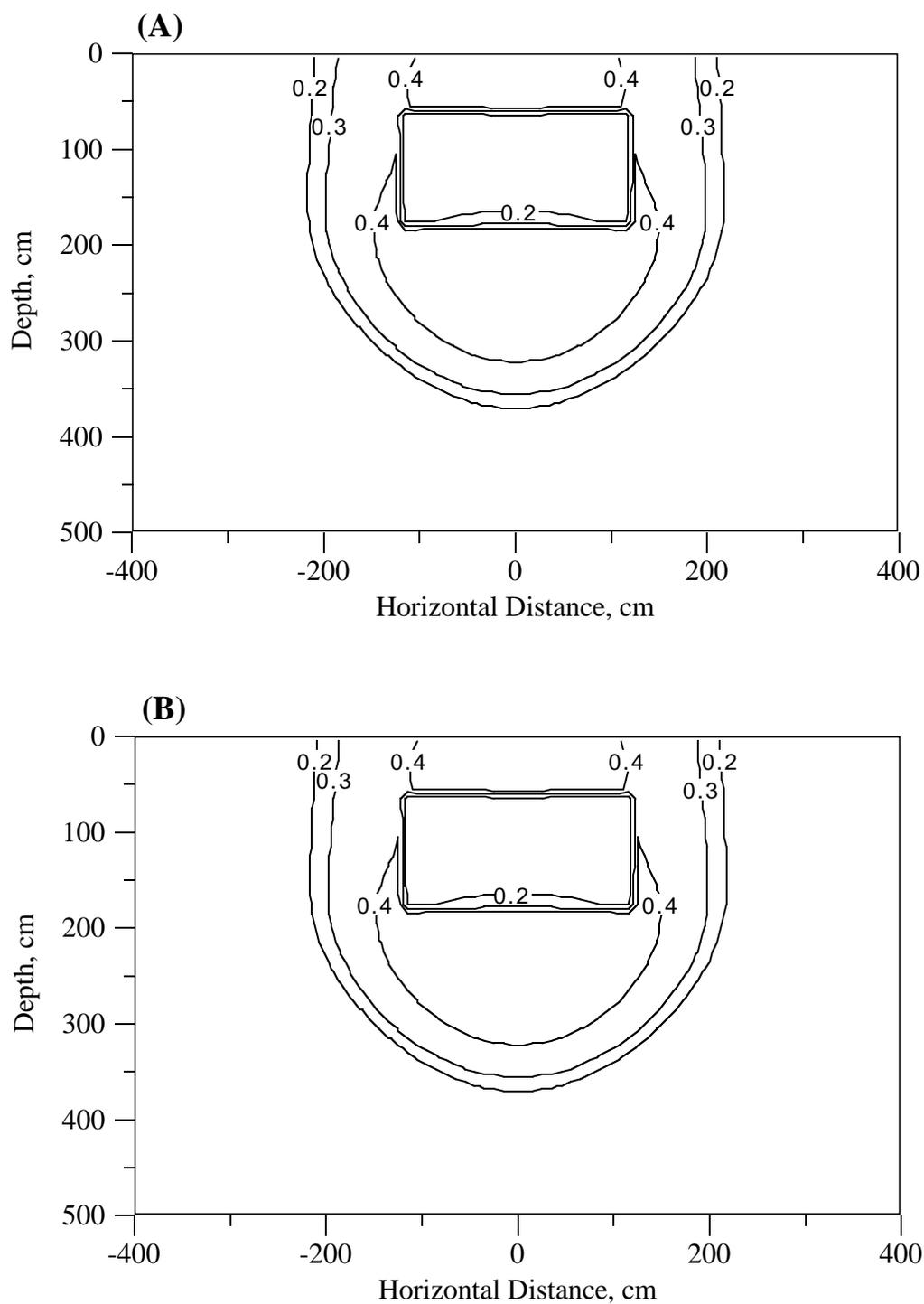
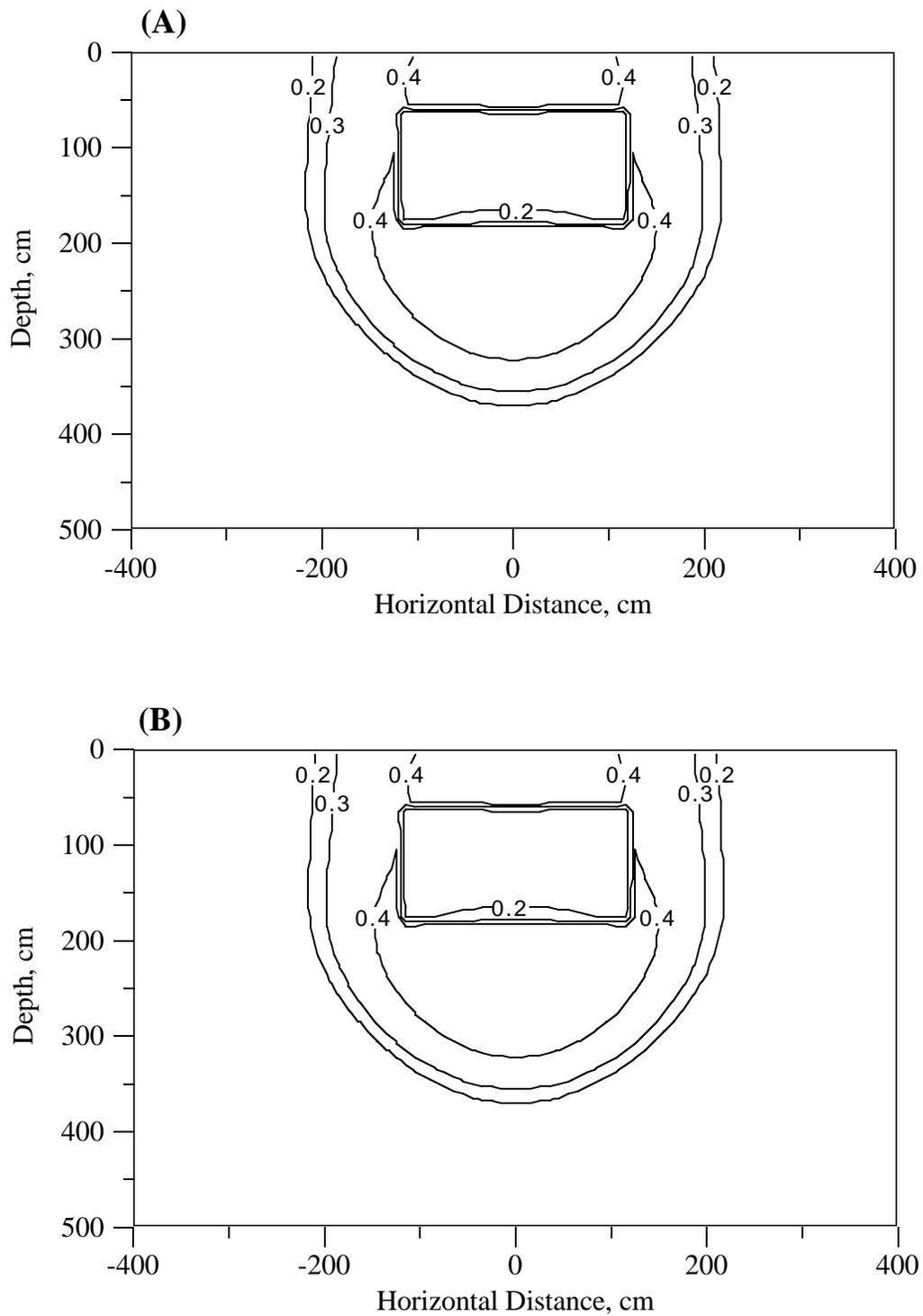


Figure 7.3. Contours of volumetric water content on day 70 for Example 2a: Blocked Soil. (A) Flux updating algorithm, (B) Polygon-based algorithm with multiple internal objects.

Results



**Figure 7.4. Contours of volumetric water content on day 70 for Example 2b: Blocked Soil. (A) Flux updating algorithm, (B) Polygon-based algorithm with a single internal object with spatially varying properties.**

**Table 7-1. Prediction and prediction error statistics. The statistics are based on the volumetric water contents (cm<sup>3</sup>/cm<sup>3</sup>) at 4000 points (i.e., each point in a 80 by 50 cell grid) for each simulation.  $\theta_{\text{flux}}$  represents the predictions of the flux updating algorithm,  $\theta_{\text{poly1}}$  represents the predictions of the polygon-based algorithm for the five object soil, and  $\theta_{\text{poly2}}$  represents the predictions of the polygon-based algorithm for the one object soil with spatially variable properties across the object**

Statistic	Uniform Soil		Blocked Soil		
	$\theta_{\text{flux}}$	$\theta_{\text{poly}}-\theta_{\text{flux}}$	$\theta_{\text{flux}}$	$\theta_{\text{poly1}}-\theta_{\text{flux}}$	$\theta_{\text{poly2}}-\theta_{\text{flux}}$
Minimum	0.0870	-0.0010	0.1127	-0.0003	-0.0003
Maximum	0.2310	0.0005	0.4629	0.0002	0.0002
Mean	0.1252	6.3499E-07	0.2038	2.1000E-06	2.1000E-06
Std. Deviation	0.0521	1.1786E-04	0.1108	3.9572E-05	3.9572E-05

moved across a segment or segments since the beginning of the simulation. For example, approximately 10.9 ft.<sup>2</sup> of water (the spatial and temporal units used to define the variable in the input files were feet and days) has moved across boundary segment one and 16.4 ft.<sup>2</sup> has moved across boundary segment two during the 1050 days of simulation. Approximately 3.3 ft.<sup>2</sup> of water has moved out of the model domain into the water table through the constant head boundary (h=0) defined by segment eight. The remaining boundary segments are defined as zero flux segments.

The net water flow through each internal polygon object segment is also listed in Figure 7.6. As expected, water flows into objects through some of the segments (denoted by positive values) and out of the objects through other segments (denoted by negative values). The net water flow into the simulation domain was calculated to be 24.05302 ft.<sup>2</sup>. This is evaluated by summing the net water flow through all of the boundary segments into the domain (i.e., the values listed at the top of Figure 7.6). The net water flow into the simulation domain is also evaluated by calculating the total change in water content in all of the internal polygon objects. This value is shown as 24.02780 ft.<sup>2</sup>. The difference between these numbers is the global mass balance error for the 1050 day simulation, approximately 0.1 percent. This error is larger than would be expected from the flux updating algorithm of Kirkland et al. (1992). Although the polygon-based code uses the same mass conservative algorithm, the polygon-based algorithm apparently introduces additional mass balance errors.

Also shown in Figure 7.6 is an estimated mass balance error due to the saturated nodes. Since a node may become saturated during a time step, it is possible to predict super-saturated nodes for a time step. Note that in this case, the estimated error due to this effect was found to be zero. Finally, the water flow through all internal polygon segments for all objects is summed. Those segment that lie along a boundary are not included. Since water flowing across a segment from object *a* to object *b* is defined as a negative flow for object *a* and as an equal but positive flow for object *b*, this sum should be zero. As the results indicate, a net flow was evaluated to be approximately -0.0017, which is less than 0.1 percent of the water added to the system through the boundary. This indicates that the fluxes across polygon segments are consistent, globally, to less than 0.1 percent.

Contour and vector plots generated from the **Water** output file (format defined by the user-supplied subroutine OUTPUT) are shown in Figures 7.7 through 7.10. Figure 7.7 shows contours of the head distribution for day 1050 of a simulation. Note, the concrete/waste region (see Figure 2.1) is essentially unaffected by the surface flow. The water appears to move down the slope of the capillary barrier and pool (i.e., reach saturated conditions) at the bottom center of the simulation domain. For this engineered cover to succeed over long periods of time, a drain must be provided in this area.

Figure 7.8 presents a vector plot for the flux distribution across all cells. The fluxes are normalized by ten times the flux added to the surface. A flux equal to that applied at the surface will thus appear as a vector of length 0.1 ft. as defined by the coordinate axis shown in the figure. Only those normalized fluxes that are greater than 0.05 are shown (all of the fluxes are available in the output file). This corresponds to fluxes that are half of the surface flux or greater. Note that water moves down slope along the capillary barrier, draining through the vertical sand column between  $x=36$  ft. and  $x=37.5$  ft. The water then moves into the clayey sand below 51 ft. and out through the bottom of the simulation domain into the water table.

Figure 7.9 shows a close-up view for the flux field near the coordinates (35 ft., 20 ft.). The flux where the flow turns the corner into the vertical sand column and drains downward is considerably larger than the flux applied at the surface. This is expected, since all of the water that was applied to the right of this region can flow through this region if the capillary barrier is effective. A close-up view of the flux field for a single vertical column of nodes above the capillary barrier at  $x=59.5$  ft. is shown in Figure 7.10. The flux of the water increases as one moves closer to the capillary barrier for all but the lowest cell. This cell con-

Results

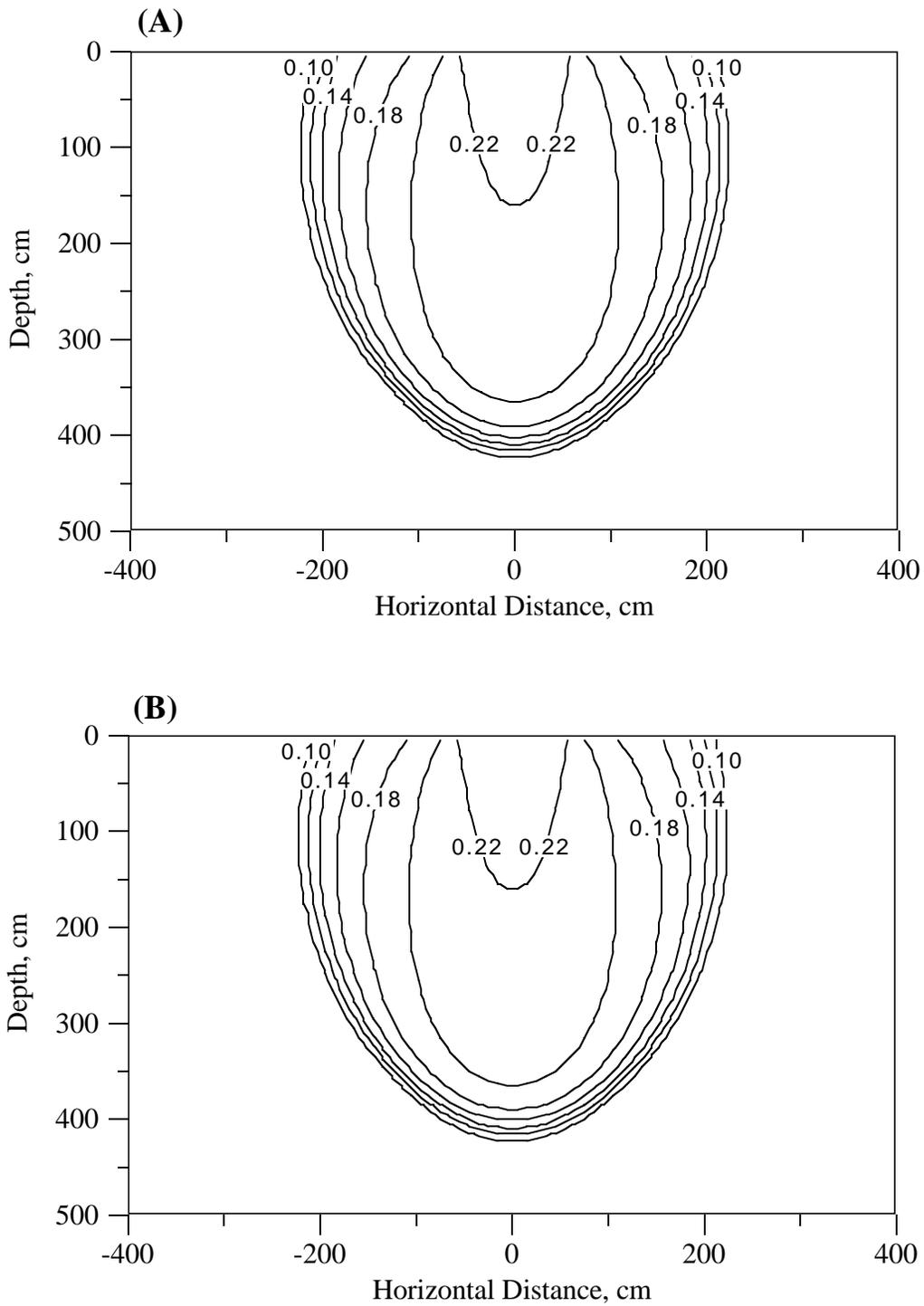


Figure 7.2. Contours of volumetric water content on day 70 for Example 1: Uniform Soil. (A) Flux updating algorithm, (B) Polygon-based algorithm.

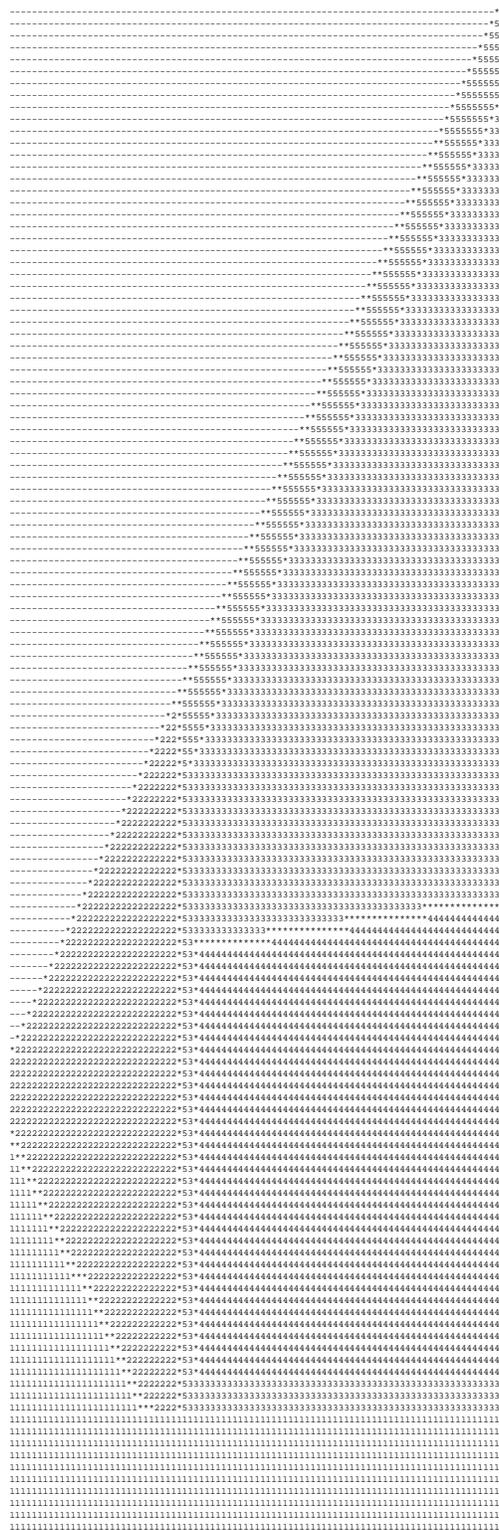


Figure 7.5. Object map for the engineered cover

## Results

tains some of the gravel below the capillary barrier and thus has a lower effective flux. It is interesting to note that the longest vector is approximately five units long, which corresponds to approximately 50 times the flux applied to the surface.

Net water flow through boundary segments into domain (Seg #, Water flow):

1	10.92751
2	16.39207
3	0.000000E+00
4	0.000000E+00
5	0.000000E+00
6	0.000000E+00
7	0.000000E+00
8	-3.266566
9	0.000000E+00
10	0.000000E+00

Net water flow through object segments into objects (Obj #, Seg #, Water flow):

1	1	6.333026
1	2	6.288396
1	3	0.000000E+00
1	4	0.000000E+00
1	5	0.000000E+00
2	1	0.000000E+00
2	2	1.977063
2	3	-0.9863616
2	4	-6.363867
2	5	0.000000E+00
3	1	5.7973131E-04
3	2	0.000000E+00
3	3	-9.6463690E-09
3	4	-6.8008649E-07
3	5	-8.1205899E-06
3	6	0.000000E+00
3	7	-1.024803
3	8	2.230026
4	1	6.7100331E-07
4	2	0.000000E+00
4	3	4.5936486E-09
4	4	7.9837207E-08
5	1	-2.038961
5	2	-3.764146
5	3	-2.668345
5	4	-3.3491568E-04
5	5	0.000000E+00
5	6	0.000000E+00

Net water flow through all boundaries into domain: 24.05302

Net water flow into all objects: 24.02780

Net water not accounted for due to saturation: 0.000000E+00

Net water flow through all object segments: -1.7734226E-02

**Figure 7.6.** Water movement across segments as output from **Water** for Example 3: **Engineered Cover**

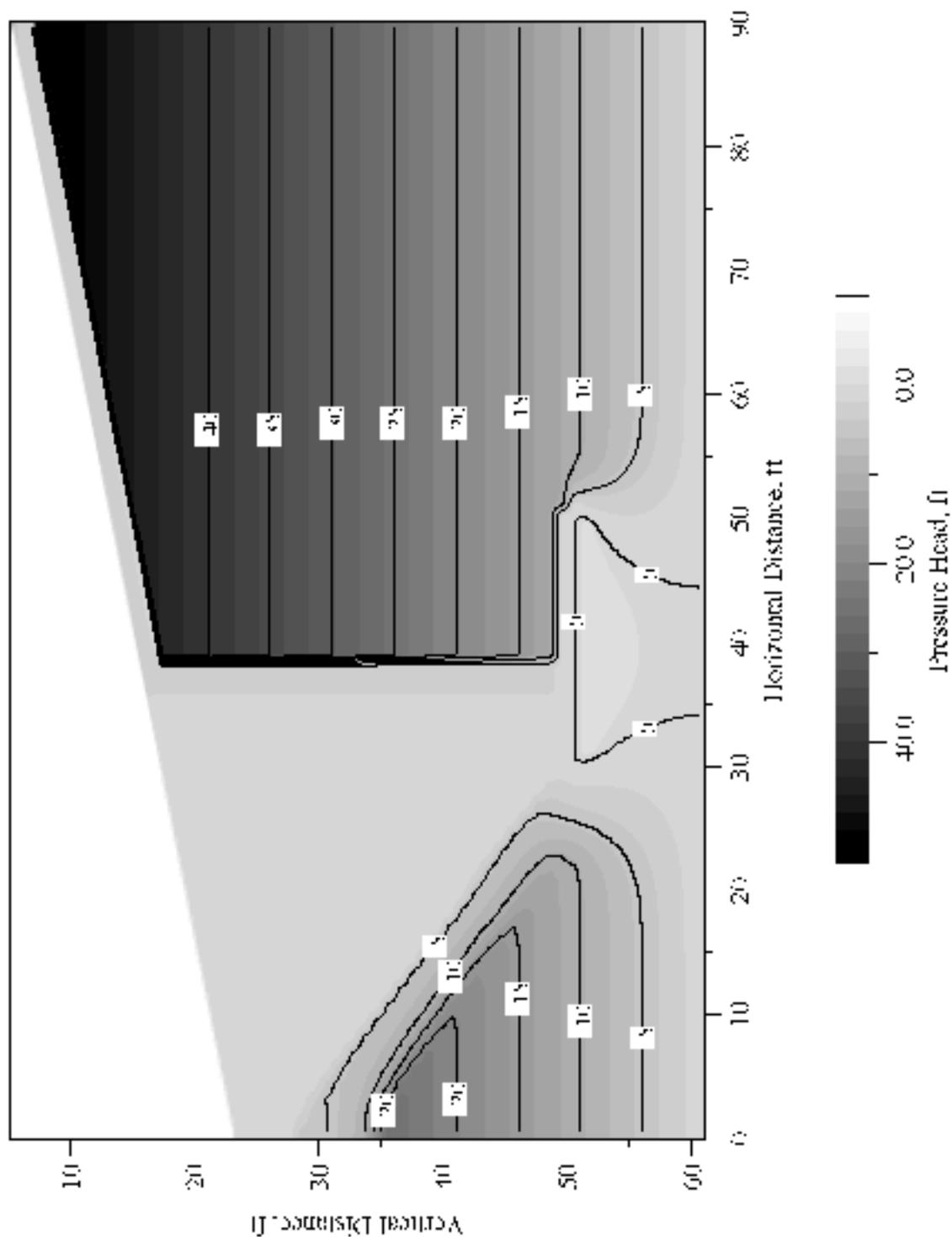


Figure 7.7. Contours of pressure head at 1050 days for Example 3: Engineered Cover

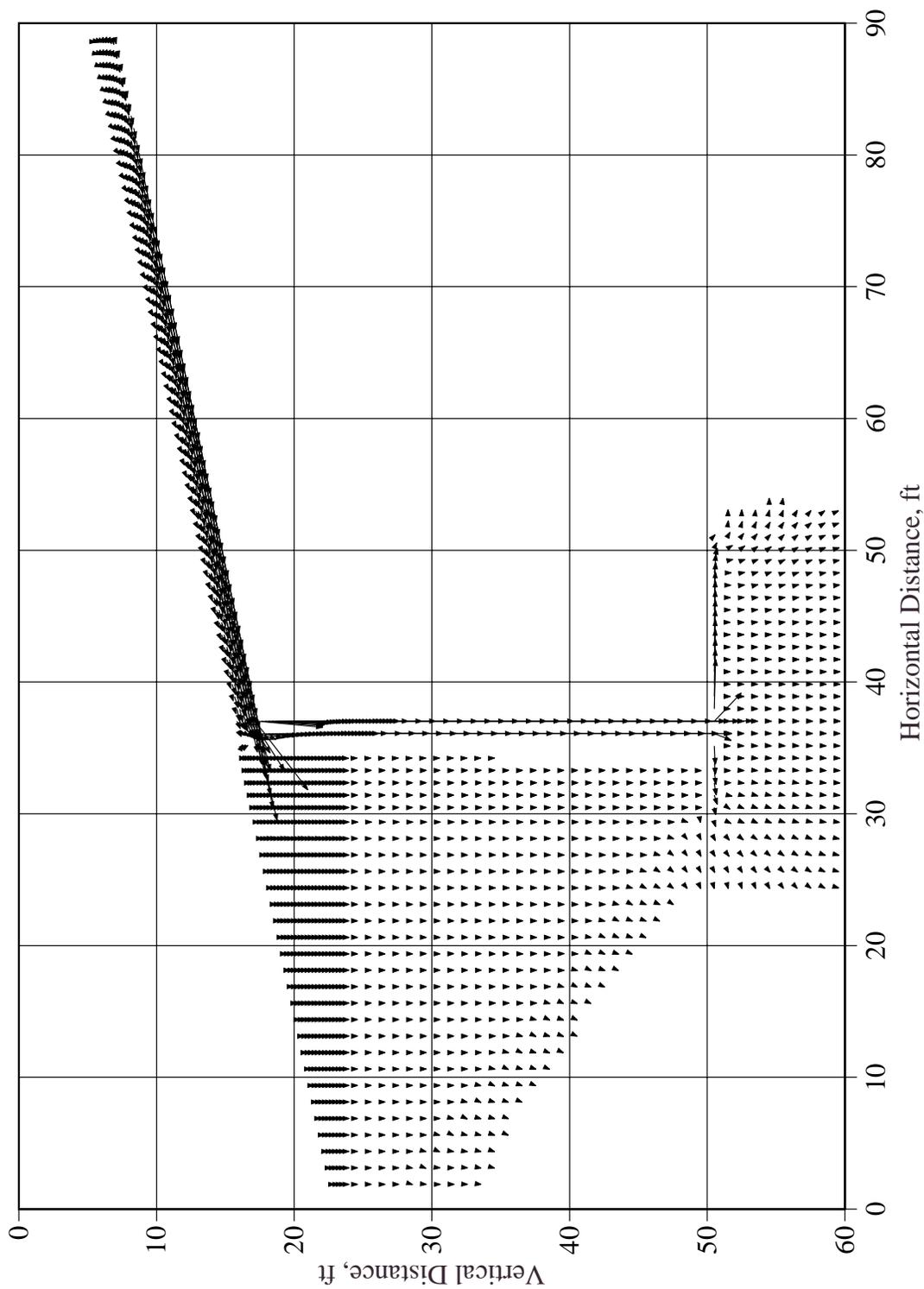
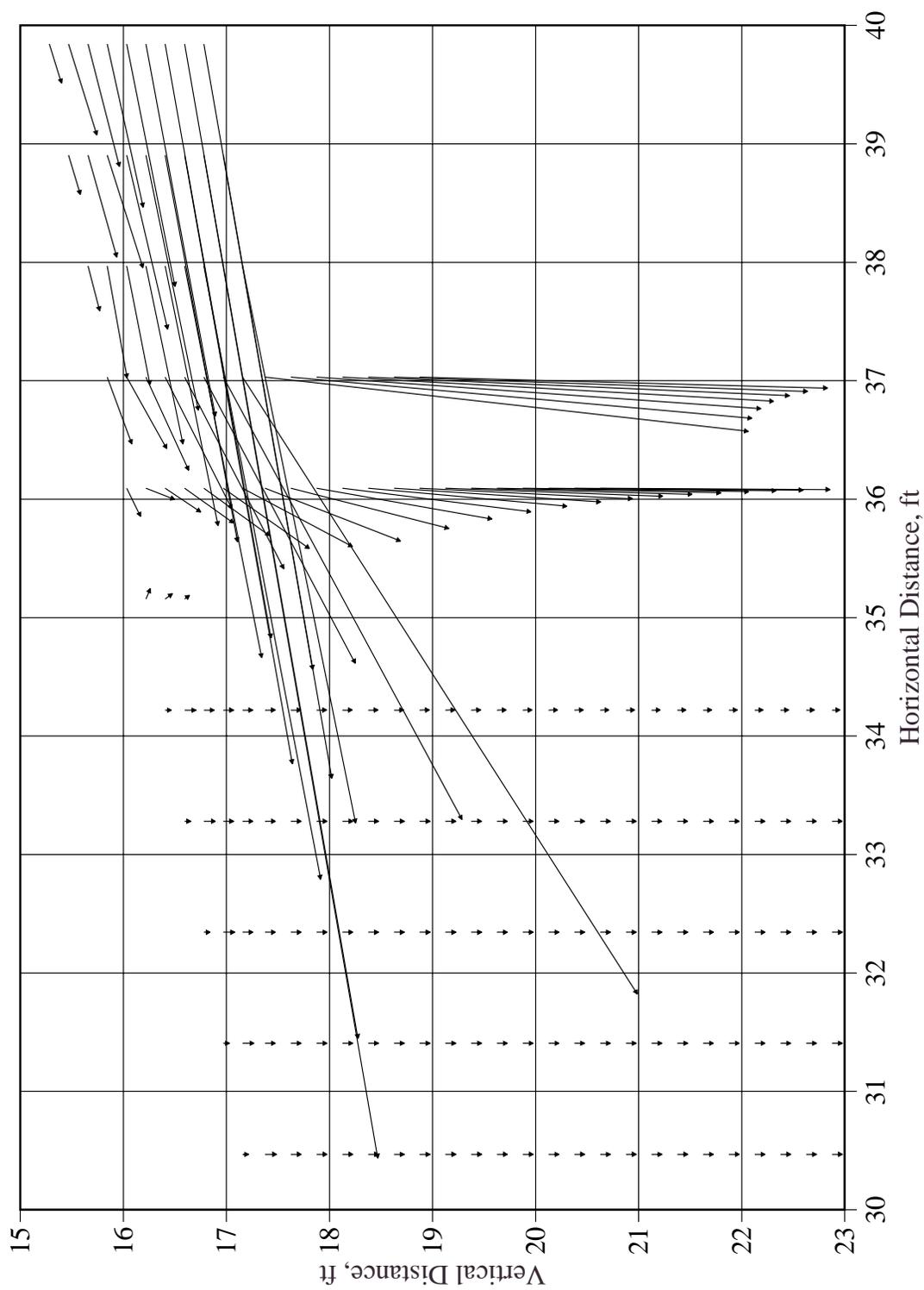


Figure 7.8. Vector plot of water flux at 1050 days for Example 3: Engineered Cover. A vector of length 1.0 ft. indicates a flux 10 times the flux applied at the upper boundary. Only normalized fluxes greater than 0.05 ft. are shown.



**Figure 7.9.** Detail of water flux near the corner of the capillary barrier at 1050 days for Example 3: Engineered Cover. A vector of length 1.0 ft. indicates a flux 10 times the flux applied at the upper boundary. Only normalized fluxes greater than 0.05 ft. are shown.

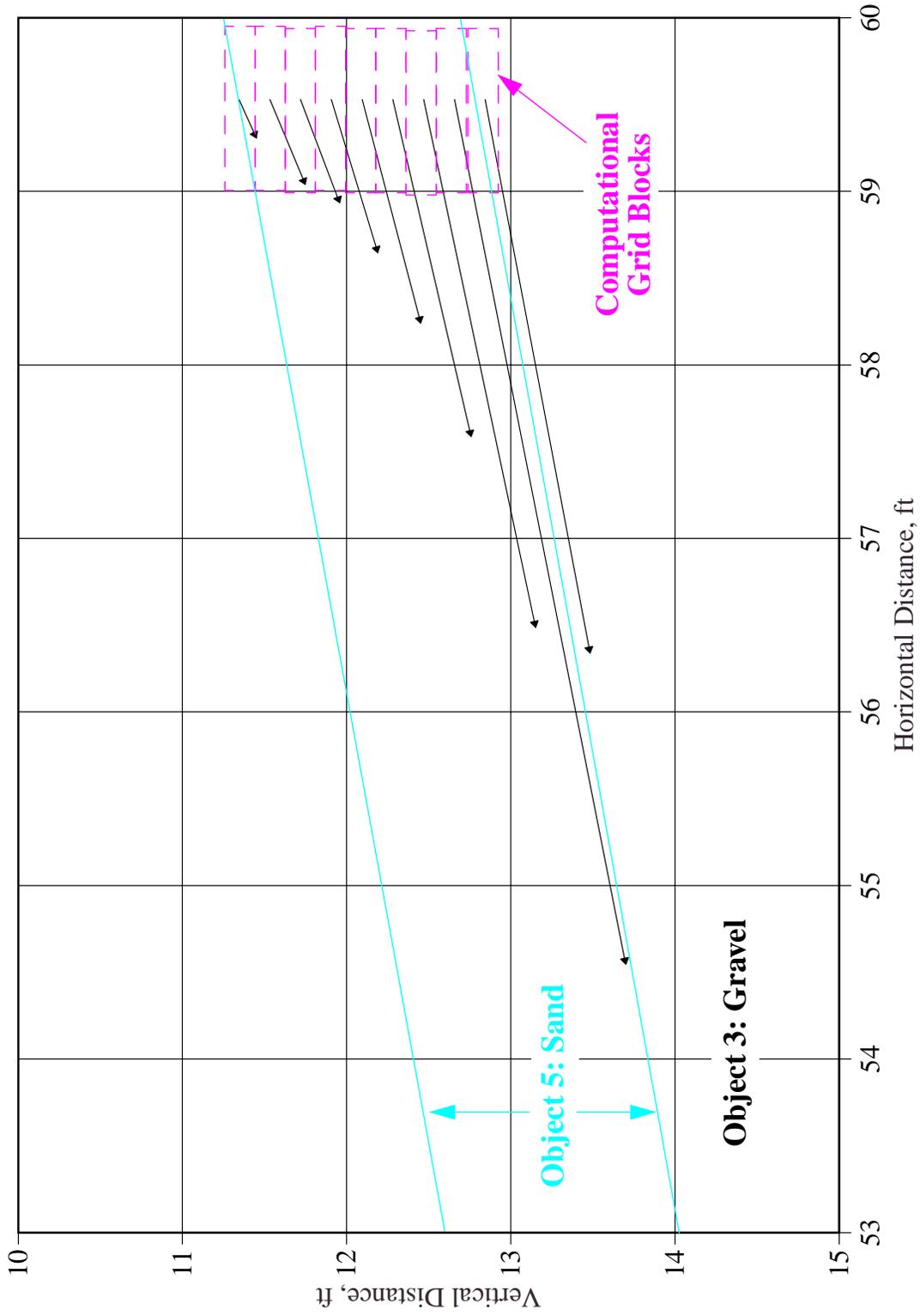


Figure 7.10. Detail of water flux for a single column of nodes above the capillary barrier at  $x=59.5$  ft. for Example 3: Engineered Cover. Time is 1050 days. A vector of length 1.0 ft. indicates a flux 10 times the flux applied at the upper boundary.

## 8.0 Discussion and Conclusions

The objective of this work was to develop the flux updating Richards equation algorithm of Kirkland et. al. (1992) into a user-friendly software package suitable for modeling water flow in variably saturated soils. The flux updating algorithm was chosen because it has been shown to be one of the fastest available (Kirkland et al., 1992) for modeling water flow through unsaturated/saturated soils and because it is especially well-suited for modeling infiltration into dry soils characteristic of arid environments.

The primary goals that guided the development of the software package were 1) complex geometries associated with barrier caps should be very easy for the user to specify and modify, and 2) execution of the water solver must be very fast so that it can be used to model large-scale problems over long time scales. The intended users of the package are scientists and engineers who have a basic familiarity with the Richards equation and with FORTRAN. The geometry of the problem, including the location and type of boundaries and the shape of internal objects, are defined through a very simple input file. The remaining input data, such as the transient behavior of the boundary conditions, are provided through user-supplied subroutines. This provides the user with more flexibility as to the type of problems solved.

A user-friendly preprocessor was developed that allows the user to define the hydraulic property fields in terms of geometry objects (polygons) independent of the computational grid. The user specifies the number of polygon object corner points and the location of each point for each object. The user also defines a boundary polygon. Each segment in the boundary polygon can represent a different type of boundary (head, flux, or unit-gradient).

A preprocessor maps the polygons onto a rectangular finite volume grid defining the computational domain. Intersections between the polygon sides and the grid cells are evaluated and used to define weighted volume and area parameters. These parameters are subsequently used by the water solver to evaluate effective hydraulic properties for the finite volume cells.

The Richards equation solver uses the mapping information generated by the preprocessor and the user supplied subroutines defining the distribution of the van Genuchten parameters (van Genuchten, 1980a, 1980b) across the objects, initial conditions, and time dependence of the boundary conditions. This solver is based on the flux updating algorithm of Kirkland et al. (1992), but is completely restructured to efficiently handle complex geometries.

The primary advantages of this approach are

1. The user can specify polygon objects that are independent of the underlying grid. The user is not required to define an underlying grid using finite volumes or elements that align with the object boundaries. Modifying or adding an object does not require modification to the grid.
2. The use of rectangular cells with a rectangular finite difference algorithm greatly enhances computational efficiency and reduces the memory requirements of the water flow algorithm. Computational efficiency is of primary concern for algorithms to be used for long-term predictions and for Monte-Carlo analysis.

The polygon-based Richards equation solver was tested by comparing model predictions to those obtained from a research code using the flux updating algorithm of Kirkland et. al. (1992). This research code has been used extensively for model validation (Hills et al., 1994) and has been compared to other algorithms (Kirkland et al., 1992). The comparisons between the polygon-based code developed here and the flux updating algorithm show that both codes give similar results. Contour plots of predictions of water content for the two codes were indistinguishable. The largest volumetric water content difference observed at the end of the simulations across all nodes was  $0.0001 \text{ cm}^3/\text{cm}^3$ . This difference is partially due to the different adaptive time-step algorithms used by the two codes, which resulted in different time-step sizes.

To our knowledge, the methodology used here to map complex geometries onto simple computational grids is unique in vadose zone hydrology and has much potential. This method explicitly accounts for discontinuities in the hydraulic properties when a computational cell is bisected by two soil types. The method assumes continuity in head but accounts for discontinuity in hydraulic properties and in volumetric water content across such a bisection. Because of this, there is no need for the underlying computation grid to align with discontinuities in hydraulic properties. This simplifies the incorporation of adaptive grids, where the grids adapt over time in response to changing conditions in the computational cells. Finally, the algorithm can be readily extended to include transport since the geometry and water flow information required for transport is already evaluated by the algorithm.

## Discussion and Conclusions

## 9.0 References

- Healy, R. W., "Simulation of Solute Transport in Variably Saturated Media with Supplemental Information on Modifications to the USGS's Computer Program VS2D," *Water Resources Investigations Report 90-4025*, U.S. Geological Survey, Denver, CO, 1990.
- Hills, R. G. and P. J. Wierenga, with contributions from S. Luis, D. Mclaughlin, M. Rockhold, J. Xiang, B. Scanlon, and G. Wittmeyer. "INTRAVAL Phase II Model Testing at the Las Cruces Trench Site." *NUREG/CR-6063*, U.S. Nuclear Regulatory Commission, Washington, DC., 1994.
- Kirkland, M. R., R. G. Hills, and P. J. Wierenga. "Algorithms for Solving Richards' Equation for Variably Saturated Soils." *Water Resources Research*, Vol. 28, No. 8, pp. 2049-2058, 1992.
- McCord, J. T. and M. T. Goodrich, "Benchmark Testing and Independent Verification of the VS2DT Computer Code," *SAND91-1526*, Sandia National Laboratories, Albuquerque, NM, 1994.
- Meyer, P. D. "Performance Assessment of a Hypothetical Low-Level Waste Facility: Application of an Infiltration Evaluation Methodology." *NUREG/CR-6114*, Vol. 1, U.S. Nuclear Regulatory Commission, Washington, DC., 1993.
- van Genuchten, M. Th. "Calculating the Unsaturated Hydraulic Conductivity with a New Closed-Form Analytical Model." Report 78-WR-08, Dept. of Civil Engineering, Princeton University, 63 pp., 1980a.
- van Genuchten, M. Th. "A Closed-Form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils." *Soil Sci. Soc. Am. J.*, Vol. 44, pp. 892-898, 1980b.
- Wierenga, P. J., R. G. Hills, and D. B. Hudson. "The Las Cruces Trench Site: Characterization, Experimental Results, and One Dimensional Flow Predictions." *Water Resources Research*, Vol. 27, No. 10, 1991.