

**SHOWFLOW: A PRACTICAL INTERFACE
FOR GROUNDWATER MODELING**

Approved:

Randall J. Charbeneau

Desmond F. Lawler

David R. Maidment

Copyright

© 1990 John Tauxe

DISCLAIMER OF WARRANTY

The ShowFlow software and documentation are provided "as is" without guarantee or warranty of any kind, expressed or implied. Neither the author nor The University of Texas at Austin will be liable for any damages, losses, or claims consequent to use of this software or documentation.

Dedication

This work is dedicated to my parents:

W. Newlon and Margaret H. Tauxe.

**SHOWFLOW: A PRACTICAL INTERFACE
FOR GROUNDWATER MODELING**

by

JOHN DAVID TAUXE, B.A.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

December 1990

ACKNOWLEDGEMENTS

For infinite and enduring patience I wish to acknowledge my wife Katie, who was supportive throughout my many hours at the computer. I also wish to thank Dr. Desmond Lawler and Dr. Randall Charbeneau of the Environmental and Water Resources Engineering Division of the Department of Civil Engineering and Dr. Dale Klein of the School of Engineering of the University of Texas at Austin, whose efforts in locating financial support are greatly appreciated.

For thoughtful review and commentary on this thesis I thank my readers, Dr. David Maidment, Dr. Lawler, and Dr. Charbeneau.

Funding for this research has been provided by Union Carbide Corporation. Part of this work was performed under appointment to the Environmental Restoration and Waste Management Fellowship program administered by Oak Ridge Associated Universities for the U.S. Department of Energy.

submitted November 1990

This version was reformatted in 2004 by the author.

ABSTRACT

SHOWFLOW: A PRACTICAL INTERFACE FOR GROUNDWATER MODELING

by

JOHN DAVID TAUXE, B.A.

Supervising Professor: Randall J. Charbeneau

ShowFlow was created to provide a user-friendly, intuitive environment for researchers and students who use computer modeling software. What traditionally has been a workplace available only to those familiar with command-line based computer systems is now within reach of almost anyone interested in the subject of modeling. In the case of this edition of ShowFlow, the user can easily experiment with simulations using the steady state gaussian plume groundwater pollutant transport model SSGPLUME, though ShowFlow can be rewritten to provide a similar interface for any computer model. Included in this thesis is all the source code for both the ShowFlow application for *Microsoft® Windows™* and the SSGPLUME model, a User's Guide, and a Developer's Guide for converting ShowFlow to run other model programs.

TABLE OF CONTENTS

Chapter 1	
Introduction	1
Chapter 2	
Development of the SSGPLUME Groundwater Contaminant Transport Model .	4
Chapter 3	
Programming Development of ShowFlow	13
Chapter 4	
Application of ShowFlow to SSGPLUME and Other Groundwater Models ...	31
Chapter 5	
Conclusions and Recommendations	43
BIBLIOGRAPHY	45

LIST OF FIGURES

Figure 1. Schematic of the SSGPLUME contaminant transport models.	5
Figure 2. "Parameter File Editor" dialog box.	32
Figure 3. Data entry error message.	35
Figure 4. Graphical representation of the plume.	36
Figure 5. "Display Graphs" dialog box.	37
Figure 6. Standard ShowFlow graphs.	37
Figure 7. Multiple simulations.	38
Figure 8. On-screen comparison of results.	39
Figure 9. Cutting and pasting graphs.	40
Figure 10. ShowFlow and other programs.	40
Figure 11. Interpretation of the X Transect graph.	41
Figure 12. Interpretation of the Y Transects graph.	42
Figure 13. Interpretation of the Concentration Contours graph.	42

ShowFlow: A PRACTICAL INTERFACE FOR GROUNDWATER MODELING

Chapter 1 **Introduction**

Groundwater by its nature is elusive, and its science has been challenged with studying the isolated and indiscernible world of the subsurface. Good data are sparse and expensively obtained. These limitations have led groundwater scientists and engineers to adopt an approach to understanding based on models derived from fundamental principles and verified by data, rather than on a strictly empirical review of existing data. With the continuing development of the physics of fluid flow and solute transport in porous media, the modeling approach has become increasingly sophisticated and fruitful. However, even contemporary models are necessarily grounded in an idealized view of reality, a result of idealized mathematics and incomplete data. Increased understanding of advanced mathematical techniques, including such entities as stochastic variables and fractals, should provide a broader perspective for groundwater modeling.

Complex models require powerful computers, but some of today's most effective models use the ubiquitous desktop computer as a simulation tool. In many cases, the known data and parameters on which modelers base their results do not warrant the use of a computationally intensive simulation. In simulation studies, the sensitivity of results to the variations in parameters is of primary interest. Such sensitivity studies provide the researcher and student alike with valuable insights into the application of physical models to natural systems, and help to strengthen one's intuition about the processes involved.

Many potentially useful models involve sophisticated mathematics but rudimentary computer programming, resulting in a program as practically unapproachable as it is theoretically sophisticated. In part, this is because theoreticians often write their own program code, focussing not so much on the user as on the results, which are often no more than a cumbersome array of numbers. Computer languages such as FORTRAN are still in heavy use, languages which are not conducive to producing a user-friendly interface.

As groundwater science and engineering grow in interest and application, so does the need for programs which can be used and understood by a larger audience. Students, policymakers, and researchers all can benefit from more productive groundwater modeling software. Recently, sophisticated graphical user interfaces (GUIs) have

become easier to produce with the advent of software libraries which promote a standard interface, such as that popularized by Apple Computer and its Macintosh series of machines. This style of GUI, developed by Xerox Palo Alto Research Center, has also been adopted as the basis for the *Microsoft Windows* and *Presentation Manager*TM graphical environments, which operate under DOS and OS/2 operating systems, respectively.

Research Objectives

The principal goal of this research has been to develop a productive interface for the development and use of groundwater models. The most desirable qualities of this interface are that it should

- provide an environment which is easy to use.
- provide for unambiguous and error-free entry of model parameters.
- have the ability to generate useful graphs of model output.
- have the ability to transfer data and graphs to other media and programs.
- be independent of the model program, so that the developer may modify the model separately.
- make efficient use of the computer's resources.
- run on a widely-used platform.
- provide adequate on-line help.
- allow for quick comparison of simulation results.

This thesis presents the *Windows* application **ShowFlow**, designed generally for computer models, and specifically for the groundwater contamination model SSGPLUME, based on work by Smith and Charbeneau (1990). ShowFlow makes SSGPLUME easy and fun to use, with facilities to create and edit files of parameters read by the model, execute the model program itself, and generate descriptive graphs based on data produced by SSGPLUME. The code for ShowFlow is designed in modules with extensive commenting, including notes for converting the code to run other groundwater models. ShowFlow is intended to be a generalized, flexible modeling interface, and can be applied to any computer model, written in any language, which generates output appropriate for graphing. In this way, ShowFlow is not limited to groundwater models, but is applicable to modeling programs in any discipline.

The *Windows* graphical environment and ShowFlow make the development, analysis, and application of groundwater modeling programs a more productive and efficient process. Since the ShowFlow program is distinct from the model it is executing, a modeler can make changes to a developing code and run it from ShowFlow to see the effects. Different versions of the model can be compared easily. Once a model has been developed, the process of examining sensitivities to changes in parameters or of conducting simulations is greatly facilitated by ShowFlow, and its potential as a teaching tool is far greater than that of conventional modeling interfaces.

Chapter 2

Development of the SSGPLUME

Groundwater Contaminant Transport Model

For the purposes of development of the ShowFlow interface, the Steady State Gaussian Plume model SSGPLUME was chosen as the target model program. Its suitability for graphical output and its computational simplicity made it an ideal candidate as ShowFlow's first model. SSGPLUME uses straightforward algebraic solutions of steady state transport equations, requiring little in computational resources, yet is readily applicable to a variety of groundwater contamination scenarios. Having demonstrated its usefulness in running this particular model, ShowFlow can now be modified to run other models as well, as described in Chapter 3: Programming Development of ShowFlow.

The SSGPLUME computer model involves the solution of several algebraic equations for a set of parameters provided in the input (parameter) file created by the ShowFlow program. The basis for SSGPLUME is a deterministic model which predicts single-phase contaminant concentrations using a one-dimensional vertical soil-water solute transport model coupled with a two-dimensional horizontal groundwater contaminant fate and transport model (Huyakorn, et al., 1985). The solution is a three-dimensional surface representing the aqueous phase concentration of a contaminant underlying a specified area — a concentration plume in the saturated zone, extended in the direction of groundwater flow. This surface is represented by a series of two-dimensional graphs, calculated and written to file by SSGPLUME and displayed on the screen by ShowFlow, as illustrated in Chapter 4: Application of ShowFlow to SSGPLUME and Other Groundwater Models.

The contaminant transport model upon which SSGPLUME is based is presented in Smith and Charbeneau (1990), and can be applied to problems of land treatment or disposal of organic compounds. The aqueous phase concentration of an organic constituent is traced as leachate from a zone of contamination in the upper part of the vadose zone, through adsorption and further degradation as it travels vertically through the lower part of the vadose zone, and finally to distribution within the saturated zone below. Mixing of the constituent upon reaching groundwater produces a gaussian concentration distribution beneath the source. The contaminant further disperses as the plume is transported horizontally in the flow direction. Figure 1 illustrates this process.

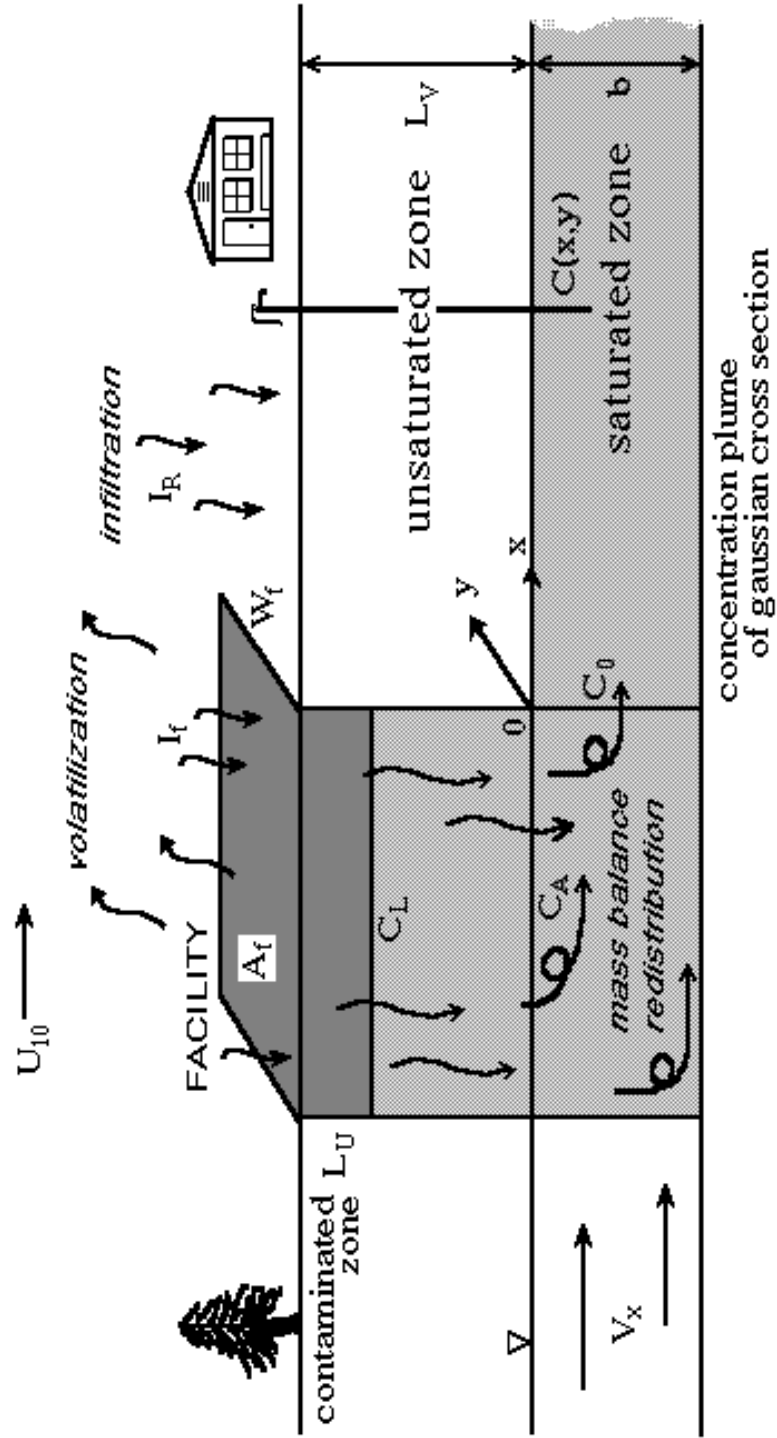


Figure 1. Schematic of the SSGPLUME contaminant model. A contaminated area degrades and leaches a constituent into the vadose zone and groundwater, where it is distributed in a gaussian plume.

2.1 The Unsaturated Zone Model

The unsaturated zone model involves the fate and transport of a single organic constituent subject to degradation, immobilization, volatilization, and leaching under steady state conditions. The aqueous phase concentration leaching from the upper zone, C_L , is given by

$$C_L = \frac{m_L}{k_g K_H \lambda_v L_u B_w I_f} \quad (1)$$

where

C_L	aqueous phase concentration leaching from upper zone	(g/m ³)
m_L	mass release (application or loading rate)	(g/m ² -d)
k_g	mass transfer coefficient to the atmosphere	(m/d)
K_H	Henry's Law constant	(conc _{gas} /conc _{water})
λ_v	effective first-order decay coefficient for upper zone	(d ⁻¹)
L_U	thickness of upper zone of contamination	(m)
B_w	water phase bulk partition coefficient	(V_{water}/V_{total})
I_f	net water infiltration rate through upper zone	(m/d)

The mass transfer coefficient k_g is calculated according to the empirical model suggested by Mackay, et al. (1982):

$$k_g = 39.9 \sqrt{6.1 \lambda_v 0.63 U_{10}} \left(\frac{v_g}{D_g} \right)^{0.67} \quad (2)$$

where

k_g	mass transfer coefficient to the atmosphere	(m/d)
U_{10}	10 meter wind speed	(m/s)
v_g	kinematic viscosity of gas phase	(L ² /T)
D_g	diffusivity of gas phase	(L ² /T)
(v_g/D_g)	Schmidt number	(dimensionless)

and equilibrium conditions and linear partitioning are assumed. The aqueous phase bulk partition coefficient B_w is calculated from

$$B_w = \theta_w \theta_g K_H \theta_o K_o \rho_b K_s \quad (3)$$

where

B_w	water phase bulk partition coefficient	$(V_{\text{water}}/V_{\text{total}})$
θ_w	volumetric water content	$(V_{\text{water}}/V_{\text{total}})$
θ_g	volumetric gas content	$(V_{\text{gas}}/V_{\text{total}})$
K_H	Henry's Law constant	$(\text{conc}_{\text{gas}}/\text{conc}_{\text{water}})$
θ_o	volumetric hydrocarbon content	$(V_{\text{oil}}/V_{\text{total}})$
K_o	hydrocarbon-water partition coefficient	$(\text{conc}_{\text{oil}}/\text{conc}_{\text{water}})$
ρ_b	bulk density of soil	$(\text{mass}_{\text{soil}}/V_{\text{total}})$
K_s	soil-water partition coefficient	$(V_{\text{water}}/\text{mass}_{\text{soil}})$

Leachate enters the lower part of the vadose zone, which extends from the bottom of the upper (contaminated) zone to the top of the saturated zone. In this lower zone, the constituent is subjected to adsorption, degradation, and further leaching under the same conditions and assumptions of local equilibrium and linear partitioning which applied to the upper zone. In the lower zone, however, volatilization is not considered. The aqueous phase concentration reaching the aquifer, C_A , is modeled as

$$C_A = C_L \exp \left[-\lambda_v \left(\frac{B_w (L_v + L_u)}{I_f} \right) \right] \quad (4)$$

where

C_A	concentration of contaminant reaching aquifer	(g/m^3)
C_L	aqueous phase concentration leaching from upper zone	(g/m^3)
λ_v	effective first-order decay coefficient for upper zone	(d^{-1})
B_w	water phase bulk partition coefficient	$(V_{\text{water}}/V_{\text{total}})$
L_v	thickness of vadose zone	(m)
L_u	thickness of upper zone of contamination	(m)
I_f	net water infiltration rate through upper zone	(m/d)

2.2 The Saturated Zone Model

Contaminant fate and transport in the saturated zone is based on the two-dimensional, advection-dispersion equation including a sink term for degradation, a retardation factor for adsorption, and an additional term to account for dilution caused by regional infiltration recharge. Assuming a steady-state, uniform flow field (in the x-

direction) and an isotropic and homogeneous aquifer, the advection-dispersion equation becomes

$$v \frac{\partial C}{\partial t} + D_{xx} \frac{\partial^2 C}{\partial x^2} + D_{yy} \frac{\partial^2 C}{\partial y^2} - \lambda R C - \frac{I_R R C}{b} = 0 \quad (5)$$

where

v	groundwater seepage velocity in the X direction	(m/d)
x	spatial coordinate parallel to groundwater flow direction	(m)
y	spatial coordinate transverse to flow direction	(m)
D_{xx}	longitudinal dispersion coefficient, calculated as longitudinal dispersivity a_L times groundwater velocity v	(m ² /d)
D_{yy}	transverse dispersion coefficient, calculated as transverse dispersivity a_T times groundwater velocity v	(m ² /d)
λ	effective first-order degradation rate constant	(d ⁻¹)
R	retardation factor	(dimensionless)
I_R	net regional infiltration rate	(m/d)
b	saturated thickness of aquifer	(m)
$C(x,y)$	concentration in x-y plane of aquifer	(g/m ³)

The solution to this partial differential equation is constrained by the boundary conditions that the solute concentration profile perpendicular to the flow direction at $x = 0$ follows a gaussian or normal probability distribution, and that the solute concentration equals zero at x and y equal to infinity.

These conditions can be expressed mathematically as

$$C(0,y) = C_0 \exp\left(-\frac{y^2}{2\sigma^2}\right) \quad (6)$$

and

$$C(x,0) = C(x,\infty) = C(x,\infty) = 0$$

where

C_0	maximum concentration at the plume source in the saturated zone	(g/m ³)
-------	---	---------------------

σ standard deviation of the normal probability distribution used to represent the source width (m)

By nondimensionalizing the equations through the introduction of the variables

$$X = \frac{v x}{D_{xx}}, \quad Y = \frac{y}{\sigma}, \quad c = \frac{C}{C_0},$$

$$D = \frac{D_{xx} D_{yy}}{\sigma^2 v^2}, \quad \text{and} \quad \Lambda = \frac{R D_{xx}}{v^2} \left(\lambda \frac{I_r}{b} \right)$$
(7)

Huyakorn has arrived at the dimensionless solution

$$c(X,Y) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} \exp\left(-\frac{w^2 X \sqrt{1 + 4 D w^2 \Lambda}}{2} \right) \cos(w Y) dw. \quad (8)$$

Charbeneau has derived an asymptotic solution, where the integrand drops off rapidly with w . Using the binomial theorem to replace the square root term and integrating produces

$$c(X,Y) = \frac{C(x,y)}{C_0} \frac{\exp\left[\frac{X}{2} \left(1 + \sqrt{1 + 4 \Lambda} - \frac{Y^2}{4 X D \sqrt{1 + 4 \Lambda}} \right) \right]}{\sqrt{1 + \frac{2 X D}{\sqrt{1 + 4 \Lambda}}}}, \quad (9)$$

a dimensionless form which is easily programmed and is an excellent approximation of the more exact analytical solution (Smith and Charbeneau, 1990).

2.3 Coupling of the Models

A mass balance approach is used to equate the constituent flux entering the aquifer (at concentration C_A) and the mass transported in the contaminant plume (with an initial peak concentration of $C_0 \neq C_A$). As illustrated in Figure 1, a mixing zone is established as the control volume. Mass fluxes are calculated and equated as presented in

the Environmental Protection Agency's Composite Landfill Model (EPACML) (Woodward-Clyde, 1989), and can be represented by

$$C_0 = \frac{C_A I_f A_f}{\sqrt{\frac{\pi}{2}} b v n \sigma \left(1 + \sqrt{1 + \frac{4(\lambda \frac{I_R}{b}) R D_{xx}}{v^2}} \right)} \quad (10)$$

where

C_A	concentration of contaminant reaching aquifer	(g/m ³)
I_f	net water infiltration rate through upper zone	(m/d)
A_f	total ground surface area of the contaminant source	(m ²)
b	saturated thickness of aquifer	(m)
v	groundwater seepage velocity in the X direction	(m/d)
n	porosity of the aquifer	($V_{\text{voids}}/V_{\text{total}}$)
σ	standard deviation of the normal probability distribution used to represent the source width	(m)
C_0	maximum concentration at the plume source in the saturated zone	(g/m ³)
λ	effective first-order degradation rate constant	(d ⁻¹)
I_R	net regional infiltration rate	(m/d)
R	retardation factor	(dimensionless)
D_{xx}	longitudinal dispersion coefficient	(m ² /d)

The value of σ is based on the width of the surface source W , and is calculated as $\sigma = W/4$ as long as $C_0 \neq C_A$. The shape of the probability distribution which represents the contaminant plume at x equals zero is constrained so that the maximum concentration C_0 does not exceed the incoming concentration C_A , a physical impossibility. In the case that the application of Equation 10 results in $C_0 > C_A$, the maximum concentration is set equal to that incoming ($C_0 = C_A$) and a new σ is calculated using a variation of Equation 10:

$$C_0 = C_A$$

and

$$\sigma = \frac{I_f A_f}{\sqrt{\frac{\pi}{2}} b v n \left(1 + \sqrt{1 + \frac{4(\lambda \frac{I_R}{b}) R D_{xx}}{v^2}} \right)} \quad (11)$$

The computer model SSGPLUME is essentially a calculation of the aqueous phase concentration $C(x,y)$ over an area extending from the origin (directly below the edge of the surface facility) to user-specified distances x and y from the origin, at a user-specified resolution. The result is a pre-set series of transects through the plume (one along the x -axis and five perpendicular to it) and a concentration contour map over the rectangular area enclosed by the points (X_{\min}, Y_{\min}) (X_{\max}, Y_{\max}) . Chapter 4 covers ShowFlow's representation of this plume "surface".

Concentration contours, presented over the xy -plane, are calculated with y as function of X and various pre-set fractions of the maximum concentration in the aquifer. Equation 12 is a variation of Equation 9 and is solved for y . (Recall that $y = \sigma Y$.)

$$y\left(\frac{C}{C_0}, X\right) = \sigma \left(2\% \frac{4DX}{\sqrt{1\%4\Lambda}} \right) \left[\frac{X}{2} (1 + \sqrt{1\%4\Lambda}) + \ln \left(\frac{C}{C_0} \sqrt{1\% \frac{2DX}{\sqrt{1\%4\Lambda}}} \right) \right] \quad (12)$$

Calculating the concentration is a straightforward application of Equations 1, 2, 3, 4, 7, 9, 10, 11, and 12, given the input parameters and a range of x and y to solve over. The input parameters are expected to be in SI units, though the concentration $C(x,y)$ units are arbitrary. The SSGPLUME program is written in the C language and appears in Appendix C.

Notation used in the equations:

A_f	total ground surface area of the contaminant source	(m^2)
b	saturated thickness of aquifer	(m)

B_w	water phase bulk partition coefficient	(V_{water}/V_{total})
C_0	maximum concentration at the plume source in the saturated zone	(g/m^3)
C_A	concentration of contaminant reaching aquifer	(g/m^3)
C_L	aqueous phase concentration leaching from upper zone	(g/m^3)
$C(x,y)$	concentration in x-y plane of aquifer	(g/m^3)
$c(X,Y)$	concentration as a function of X and Y	(dimensionless)
D_g	diffusivity of gas phase (same units as v_g)	(L^2/T)
D_{xx}	longitudinal dispersion coefficient	(m^2/d)
D_{yy}	transverse dispersion coefficient	(m^2/d)
I_f	net water infiltration rate at the facility	(m/d)
I_R	net regional infiltration rate	(m/d)
k_g	mass transfer coefficient to the atmosphere	(m/d)
K_H	Henry's Law constant	$(conc_{gas}/conc_{water})$
K_o	hydrocarbon-water partition coefficient	$(conc_{oil}/conc_{water})$
K_s	soil-water partition coefficient	$(V_{water}/mass_{soil})$
L_U	thickness of upper zone of contamination	(m)
L_V	thickness of vadose zone	(m)
m_L	mass release (application or loading rate)	(g/m^2-d)
n	porosity of the aquifer	(V_{voids}/V_{total})
R	retardation factor	(dimensionless)
U_{10}	10 meter wind speed	(m/s)
v	groundwater seepage velocity in the X direction	(m/d)
W	width of the surface source	(m)
x	spatial coordinate parallel to groundwater flow direction	(m)
y	spatial coordinate transverse to flow direction	(m)
θ_w	volumetric water content	(V_{water}/V_{total})
θ_g	volumetric gas content	(V_{gas}/V_{total})
θ_o	volumetric hydrocarbon content	(V_{oil}/V_{total})
λ	effective first-order degradation rate constant	(d^{-1})
λ_v	effective first-order decay coefficient for upper zone	(d^{-1})
v_g	kinematic viscosity of gas phase (same units as D_g)	(L^2/T)
ρ_b	bulk density of soil	$(mass_{soil}/V_{total})$
σ	standard deviation of the normal probability distribution used to represent the source width W	(m)

Note that the units of concentration (g/m^3) and (mg/L) are equivalent.

Chapter 3

Programming Development of ShowFlow

The *Windows* programming environment has been developed by Microsoft Corporation for creating *Windows* applications for DOS computers. The C language was chosen as a programming platform for its versatility and elegance, and the *Microsoft Windows Software Development Kit* (SDK) provides several hundred auxiliary functions and libraries geared specifically toward building a *Windows* interface. The SDK is not for the casual programmer, as *Windows* applications require carefully structured programming and a deeper understanding of the computer than is required by ordinary languages. Despite its intricacies, ShowFlow's modular design and in-code documentation make the conversion from running SSGPLUME to another model straightforward. Such a conversion is the topic of later sections of this chapter.

3.1 Advantages of the *Windows* Interface

Anyone who has used *Windows* can attest to its power and ease of use, but there are other reasons to adopt it as a user interface. With the SDK, Microsoft has promoted a standardized appearance for all *Windows* applications which closely parallels similar interfaces used by Macintosh, SUN, and DEC systems. The look and feel of windows, labels, and menus is the same for all applications, with the beneficial result that a user familiar with one will feel at home with others and will already know how to use it. Not only is the appearance of *Windows* applications standardized, but the methods of storing and displaying information is as well. Bitmaps, texts, and metafiles can be exchanged between applications via built-in *Windows* functions and standard applications. Several *Windows* applications can run simultaneously, but the environment is carefully managed so that the dynamic memory allocations do not interfere with each other, a remarkable accomplishment for the DOS environment.

3.1.1 The Standard *Windows* Interface

All *Windows* applications share several aspects. The standard window has a title bar with a system menu on the left and display mode controls on the right, moveable borders, and a menu bar of available commands. Attributes such as screen colors can be changed on a system (via the *Control Panel*), and all the *Windows* applications running on that system will assume those attributes.

Further standards are recommended by Microsoft in the [Application Style Guide](#) provided with the SDK. For example, any application using files should include a "File"

option in the main menu, with "New", "Open", "Save", and "SaveAs" commands, and similar guidelines are suggested for editing controls. Editing controls further have suggested standard "accelerators", or "hot keys" such as CTRL + INSERT for "Paste". Although this kind of editing does not apply to ShowFlow, accelerator keys are used for almost all other functions. Used in combination with the mouse, accelerators can markedly enhance a user's efficiency.

3.1.2 Exchange of Data

In addition to being attractive and self-instructive, *Windows* provides means to exchange data between applications. One such link is the Dynamic Data Exchange (DDE), which maintains a "live link" between applications so that data may be changed in one application and automatically updated in another. For example, a chart generated in EXCEL is dynamically linked to a spreadsheet of data, and may also be included as a figure in a WORD FOR WINDOWS document. When the spreadsheet data change, so does the EXCEL chart, and so does the WORD figure.

Although ShowFlow does not use the DDE facility, it can export graphs as bitmaps via the *Clipboard*. The Clipboard is a standard *Windows* feature for the transfer of bitmaps, texts, and metafiles between and within applications, with the commands "Cut", "Copy", and "Paste". The item of interest is cut or copied to the Clipboard, and pasted from it, so that the Clipboard acts as a buffer. Any time "Cut" or "Copy" is used, the Clipboard is cleared of any previous data, imposing a one-item limit on storage. All applications have access to the same Clipboard, so that an item copied from one may be pasted into another. Drawings made in PAINTBRUSH may be pasted into a WRITE or WORD document, or text may be included in a drawing. Graphs generated by ShowFlow may be copied to the Clipboard and pasted into PAINTBRUSH where text and other annotations can be added, and then shuttled to a paper in Windows WRITE, for example. All of this can be done without leaving ShowFlow, so that the user can run a model in ShowFlow and write a paper in WRITE with both applications running side-by-side on the same screen. This is the beauty of *Windows*.

3.2 How *Windows* Programs Work

Before attempting a conversion of ShowFlow to run another model, it will be important to understand some of the basics behind *Windows* programs in general. Following this section is The Structure of ShowFlow, which explains how the ShowFlow application works in particular. All of these programs share some unique features: a

message-driven logic and a dynamic memory environment, for example. These topics are presented briefly for the curious reader, but refer the user to [Programming WINDOWS](#) (Petzold, 1988) for a thorough discussion. Grasping these concepts is not prerequisite to modifying ShowFlow, but is necessary to understanding its logic. The reader should also take note of the Typographical Conventions used in this paper, outlined in Appendix A: User's Guide to ShowFlow.

3.2.1 Message Processing

Traditional computer programs run in a more or less linear fashion, following a pre-determined course of events and prompting the user for any needed information. If the user is presented with a range of choices, the program may be called "menu-driven", since the branching is controlled by the user. *Windows* programs have menus of commands as well, but here the similarity stops.

What makes *Windows* codes unique is the concept of messages. Messages are constantly being fed to a part of the program called a Window Procedure, which either processes the message in a decision-making tree or dispatches it to another message-processing procedure. Messages originate from a variety of sources; the system, the keyboard, the mouse, and the program itself can all send messages to *Windows'* central message queue. The inner workings of *Windows* decides which programs should receive a given message and dispatches it accordingly. Each program repeatedly checks the message queue for its messages and processes them. This loop is in the code module ShowFlow.C (this file and other program modules are listed in Appendix B). Each *Windows* program has a subroutine called WinMain which processes incoming messages. After drawing the main window on the screen, WinMain continually executes this message-processing loop:

```
while ( GetMessage( &msg, NULL, 0, 0 ))
{
    if ( !TranslateAccelerator( hWnd, hAccel, &msg ))
    {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
}
```

The function names indicate that this loop simply reads messages, translates them, and dispatches them to other parts of the program.

The sections of the program which actually examine the messages and make decisions are the Window Procedures. Each window, including the main (*parent*)

window, any *child* windows (for example, ShowFlow's graph windows), and dialog boxes all have such a procedure, identified by a function name ending in "Proc". Examples are ShowFlowWndProc, Graph1 WndProc, and EditDlgProc, all found in appropriate modules of the program. Each of these examines messages using the C switch and case construction to determine branching.

As an example, when the user clicks with the mouse on the "Save" command in the ShowFlow window, *Windows* sends a message to WinMain in ShowFlow.C, which forwards the message to ShowFlowWndProc. This procedure tests the value of the message by matching it to the menu identifier IDM_SAVE, and launches the SaveParamFile subroutine. All *Windows* programs process messages similarly.

3.2.2 Memory Management

The dynamic memory management employed by *Windows* is at once a blessing and a curse. Most programming languages rely on static memory, where the storage address for a particular variable does not change. In *Windows*, the location cannot be assumed constant since the memory is frequently reorganized to make room for more data or code segments. This dynamic memory allocation is what makes *Windows* work so well in running multiple applications. When a particular segment of code is needed in starting or continuing an application, *Windows* will load it from the disk. If more room is needed for the segment, *Windows* will discard unused segments and reload them later when needed.

For example, when ShowFlow runs SSGPLUME in a spawned DOS subprocess, it needs to clear a large block of memory for the model program to run in. To do this, the GlobalCompact function is called, which shuffles the existing memory into a more compact form. If necessary, discardable code segments (identified in the ShowFlow.DEF file) will be discarded to make room. When the model program has completed execution, ShowFlow resumes operation, reloading segments as they are needed (for drawing graphs, for example).

Fortunately, *Windows* also keeps track of where everything is located by using *handles*, and when your program refers to a variable *Windows* knows where to find it, but still the programmer is not free from memory worries. Certain operations like file input and output require a static block of memory, which can be allocated with the GlobalAlloc function and freed with the GlobalFree function. Examples of these operations can be found in the ReadFile and WriteFile subroutines in the SFFile.C module. Understanding memory management is one of the most challenging aspects of *Windows* programming.

3.3 The Structure of ShowFlow

3.3.1 Program Modules

ShowFlow is designed in several modules to use as little memory as possible. These modules are written as separate C language source code files and compiled as code segments, each containing code related to a specific task. These segments are identified in the ShowFlow.DEF *module definition file* and are given attributes about how *Windows* can treat them in managing memory. For example, the segment labelled `_FILE` is given the attributes `LOADONCALL`, `MOVEABLE`, and `DISCARDABLE`. This means that the code in the `_FILE` segment is loaded from disk only when it is needed (when some file I/O is done), that it can be moved around in memory, and that it can be discarded if *Windows* needs to make room for something else. (This feature allows several programs to occupy the screen at once.) In contrast, the segment labelled `_TEXT`, which contains the main ShowFlow module ShowFlow.C, is labelled `PRELOAD` and `MOVEABLE`, instructing *Windows* to load the segment when first starting ShowFlow, and to move it if necessary, but does not give permission to discard the `_TEXT` segment.

The program modules are compiled by the ShowFlow.MAK *make file*. The only segment which contains more than one module is the `_FILE` segment, which contains code from SFFile.C and SFFilDlg.C, both necessary to perform the file operations `Open`, `Save`, and `SaveAs`, which involve reading and writing files. Other segments are `_FILE`, `_GRAPH`, `_INIT`, `_PRINT`, and `_RUN`, which are used in the operations of editing the parameter file, generating graphs, program initialization, printing graphs, and running the model program, respectively. The program modules have similar names and are described individually below:

ShowFlow.C This is the central C module, containing the `WinMain` subroutine, `ShowFlowWndProc`, and subroutines related to displaying help files, posting error messages, and other general-purpose tasks. The `_TEXT` segment, which contains the ShowFlow.C code, is not discardable since it contains the principal message-processing loop. Program control starts in `WinMain` which executes the initialization code (calling the subroutines in `SFInit.C`) and then dispatches *Windows* messages to other parts of the program. The central message-processing loop is in `ShowFlowWndProc`, which executes branching as signalled by the user's choice of menu commands. Depending on the command, `ShowFlowWndProc` will call

subroutines in ShowFlow.C, SFEdit.C, SFFile.C, SFGraph.C, SFPrint.C, or SFRun.C.

- SFInit.C This module is used only when ShowFlow first starts up or when a subsequent *instance* is run. Like all *Windows* programs, more than one instance may be run at a time. Since all instances share the same code segments to save memory, information from previous instances must be shared with the new one. This is also handled in the SFInit.C module with the *Windows* GetInstanceData function.
- SFEdit.C The *Parameter File Editor* is run from this module. The EditDlgProc subroutine handles message processing for the EditBox dialog box.
- SFFile.C The tasks of reading and parsing both parameter and data files, and formatting and writing parameter files are performed in this code module. File names are obtained from the user by calling the SFFilDlg.C module.
- SFFilDlg.C The OpenBox, SaveBox, and SaveAsBox dialog boxes are created and processed by this module, which is called by SFFile.C. Both SFFilDlg.C and SFFile.C are compiled into the _FILE code segment since they are always used together.
- SFGraph.C This module executes the creation and message-processing of the graph windows, as well as the composition and "painting" of the graphs. Each graph window has its own message loop in the subroutines Graph*WndProc (where * = 0, 1, 2 ...) and its own suite of datasets to plot. Each time a graph is created, a movable block of memory is assigned to contain the data for the graph, and when the graph window is destroyed, this memory block is freed. The PaintGraph subroutine is called each time a graph window is *ainted*, or redrawn, which happens when a window is uncovered or resized. The appearance of the graph depends on the size of the graph window.

SFPrint.C ShowFlow supports printing of bitmaps on pixel-based printers, including dot matrix and laser printers. (Since plotters are vector-based they cannot print bitmaps.) All of the code necessary to do the printing is in this module, which is called from the Graph*WndProc subroutines when the user chooses the "Print Graph" option.

SFRun.C This code module creates and processes the RunBox dialog box and initiates the spawning of a DOS subprocess in which to execute the model program. A special interrupt function Int21Function4B is called to do the spawning. This procedure is detailed in the *Microsoft Windows Software Development Kit Application Note for Spawning Applications*.

3.3.2 Auxiliary Files

ShowFlow.H This C *header file* is included in all the ShowFlow modules. It defines global program constants and all of the function prototypes.

SFDialog.H An additional header file defining dialog box constants (IDD_*) is #included into the ShowFlow.H header file. These constants are maintained by the dialog box editor DIALOG and so are kept in a separate file.

ShowFlow.DEF This *module definition file* identifies the program segments, exports the *Windows* procedure subroutines, and defines several other environment variables for ShowFlow. This file is called by the linker when assembling the ShowFlow.EXE program.

ShowFlow.MAK The *make file* is used to compile all the elements of ShowFlow, and makes use of the Microsoft C MAKE utility which conveniently recompiles only those files which have changed since the last compiling. C language modules are compiled by the C compiler, and resources by the SDK's resource compiler. The *.OBJ object files and *.RES *resource file* are then linked by the *Windows SDK* LINK4 program following the instructions given in the file ShowFlow.LNK.

ShowFlow.LNK This special *link file* is necessary since there are too many objects to list on one line as normally required by the LINK4 command in the make file. Unfortunately, no comments are allowed in ShowFlow.LNK so it has no in-code documentation.

SSGPLUME.PIF The model program must have a *program information file* (PIF) with the same prefix as the model. This PIF contains information about how much memory is needed to run the model, etc. *Windows* comes with a special PIFEDIT program and instructions on how to construct these files, which are necessary to run DOS applications like SSGPLUME.

3.3.3 Program Resources

ShowFlow.RC The *resource script*, unique to *Windows* programs, identifies ShowFlow's resources, which include the definitions of dialog boxes (from ShowFlow.DLG), the program icon (from ShowFlow.ICO), the menu bar and pull-down menus, accelerators, help text resources (from SFHelp_*.TXT), and string constants. All of these resources are compiled by the SDK's resource compiler from the make file. The resource script and the files it references contain almost all of the text displayed in ShowFlow, so that changing an error message or even translating the entire program into another language requires only altering strings in ShowFlow.RC, ShowFlow.DLG, and SFHelp_*.TXT.

ShowFlow.DLG The *dialog box* definitions are maintained in this separate file by the dialog box editor DIALOG, provided with the SDK. The suite of definitions is then included as a resource in the ShowFlow.RC resource script at compile time.

ShowFlow.ICO The ShowFlow *icon*, which appears in the AboutBox and when the program is minimized (its "iconic" state), is a bitmap created with the SDK's ICONEDIT program (replaced in *Windows 3.0* with SDKPAINT). It is included as a resource in the ShowFlow.RC resource script at compile time.

SFHelp_*.TXT Several help files are also included as resources, with the generalized file names SFHelp_*.TXT, where the * represents a letter identifying the help topic. For example, SFHelp_E.TXT instructs the user in Editing the parameters. These help files are simple text files, limited to a width of SCRBUFWIDTH characters and a length of SCRBUFLINES lines. These constants are defined in ShowFlow.H and represent the dimensions of the screen buffer in which the help files are displayed. Any number of additional help files can be created, but each must be defined in the ShowFlow.RC resource script both as a menuitem (with an associated menu ID IDD_HELP_*) and a TEXT resource. Each menu ID must be assigned a value in ShowFlow.H, and each must have a corresponding "case: IDM_HELP_*" statement in the main message-processing loop ShowFlowWndProc in the ShowFlow.C module.

3.4 System Requirements

Modification of ShowFlow version 1.0 will require the following software:

- *Microsoft Windows version 2.1* or later.
- *Microsoft C Compiler version 5.1* or later.
The C compiler is needed for some standard C libraries and header files, and for the compiler.
- *Microsoft Windows Software Development Kit ver. 2.0*
The SDK includes the libraries for the five hundred-odd *Windows* functions, some alternative header files, and the linking program.
- *Microsoft Windows SDK Application Note for Spawning Applications*
Although this Application Note itself is not necessary for converting ShowFlow, the object file which comes with it is: WSPAWN.OBJ must be available to the linker when compiling the ShowFlow application.
- A text editor will be necessary for editing files.

3.5 Programmer Requirements

Although a general knowledge of the C programming language is advised, minor modifications to ShowFlow will not require it. Modification of ShowFlow does not require advanced skills. Most changes involve simple "boilerplate" substitutions or

extensions of code which is already present, and in-code "CONVERSION NOTES" have been added to identify appropriate sections of the program modules. Specific changes are included in a later section.

3.6 Compatibility Between ShowFlow and the Model

The ShowFlow application works closely with the model program, and so certain compatibilities must exist. Most of the exchange of information is through standard formatted ASCII data files, so the protocol is straightforward. Another restriction, depending on the computer, is the amount of available memory.

3.6.1 System Memory Requirements

Both ShowFlow and *Windows* consume some random access memory (RAM) even after compacting themselves, and a limited amount of space is available for running the model program. Newer machines with more than the standard 640K RAM available will perform better and allow larger models to run.

The best way to find out how much memory is available is to run *Windows* and ShowFlow, and check the amount of RAM. This is displayed in the About box for the Program Manager, (choose the "About Program Manager..." command from the "Help" menu.) The number displayed is the amount of free memory in kilobytes.

3.6.2 Exchange of Data via Files

The structure of data files is important to the linking of ShowFlow and the Model. Examples of each file are included at the end of Appendix C. It is assumed that three such files exist (with identical file name prefixes but different file name extensions) for each simulation:

- The parameter file (extension *.PAR) is written by ShowFlow and read by both ShowFlow and the Model as an input file. The formatting can take any form as long as the reading and writing are compatible, and is done in the subroutines ReadParamFile and SaveParamFile in the SFFile.C module.
- The data file (*.DAT) is produced by the Model, and contains data used in drawing graphs for ShowFlow. Again, the structure may be flexible, though the current requirements of ShowFlow are easily met. Although the code in the subroutine ReadDataFile appears complex, it is designed to

read a generalized data file with these minor constraints: The first record is expected to be a comment or title for all three graphs, and may contain spaces and other characters. Following the comment line are three datasets, delineated by the values in the first column (the first value in each record). These values are associated with the horizontal axis in each graph and should be monotonically increasing for each dataset. ShowFlow recognizes a change of dataset when the first value in a record is less than that of the previous record. All values should be separated by spaces and all records by a carriage return and/or line feed character.

- The text file (*.TXT) is also produced by the SSGPLUME model, and contains formatted data for reading by the user. This file has no required format since it is not read by ShowFlow.

3.6.3 Execution of the Model Program

In general, the model program must be able to run from the command line with no further input from the user. SSGPLUME is flexible, and will prompt the user for input and output file names only if they are not provided on the command line. A typical DOS command line to run SSGPLUME is:

```
SSGPLUME FILENAME.PAR FILENAME.DAT FILENAME.TXT
```

which lists the parameter, data, and text filenames, respectively.

The reason for the necessity of command line arguments is that the spawning process is limited to one command line. This command line is constructed by ShowFlow prior to spawning (see the RunModel subroutine in the module SFRun.C) and is therefore transparent to the user.

3.7 Specific Code Modifications

ShowFlow was designed in as general a way as possible to reduce the work necessary in modifications. For example, program constants used throughout the code are centrally assigned in the ShowFlow.H header file, and most strings displayed by the program are defined in the ShowFlow.RC resource script file. However, these generalities can only do so much, and since future users will undoubtedly want to make changes in ShowFlow to accommodate different model programs, the necessary modifications have been anticipated.

Required program code modifications, which may be made with any ASCII text editor, have been divided into two categories for convenience. The first involves changes which will be necessary for any modification (*General Modifications*), and the second involves additional changes needed only if the number of standard graph windows is changed from three (used by the current ShowFlow) to some other number (*Graph Number Modifications*).

All the changes are noted in the code listings where appropriate, in special boxed comments entitled "CONVERSION NOTES". The easiest way to find them is to search for these words using a text editor. The best instruction about changes will be found in these CONVERSION NOTES, but an outline of modifications is provided here, grouped by program module (file):

3.7.1 General Modifications

- SFEdit.C
 - The testing of values entered by the user from the Parameter File Editor may be enhanced to almost any degree as a method of "idiot-proofing" the model program. Although the model program may also perform these tests, doing so here gives the user the opportunity to correct mistakes immediately with guidance from ShowFlow's error messages.

- SFFile.C
 - The subroutines ReadParamFile and ReadDataFile are both sensitive to what delimiters are used to separate (*parse*) data elements (*tokens*) in each file, which depends on the respective formats of the parameter and data files. Usually, these tokens are parsed by searching for a space, tab, carriage return, or line feed character. Whatever the delimiters are, they must be defined in each of these functions in the string variable: `szDelimiter`.

 - The SaveParamFile subroutine performs the formatting of the parameter file, which is written in the WriteFile subroutine. Any format consistent with the model program may be used, and appropriate changes should be made.

- SFRun.C
 - Changes needed in the RunModel subroutine reflect changes in the command line arguments used to execute the model program. ShowFlow

currently uses three arguments (filenames) in addition to the name of the model program:

```
SSGPLUME FILE.PAR FILE.DAT FILE.TXT
```

If the new model is not executed in this fashion, the command line string must be assembled differently, as outlined in the code comments.

- ShowFlow.H • Several program constants which are model-specific need to be adjusted, such as NPARAMS, MAXDATAPTS, and others defined in this module.
- ShowFlow.RC • If a new icon file has been defined, it should be included here. Icons can be designed with the ICONEDIT program described in section 3.8.2 below.
- Several string constants such as the model name IDS_MODELNAME must be changed as appropriate.
- SFDialog.H • The definition and assignment of dialog box IDs (beginning with IDD_) must reflect the number of parameters required by the model (remember to start counting with zero). These values are assigned to edit control IDs in the EditBox dialog box, and the numeric values must be consecutive. Dialog boxes can be created and edited using the DIALOG program described in section 3.8.1 below.
- ShowFlow.DLG
- The EditBox dialog box (the Parameter File Editor) will need an extensive revision and will probably be the most difficult part of the conversion process. To edit dialog boxes, see the section 3.8.1 below about using the SDK's DIALOG program.
- HELP Files • The help files are ASCII text files which are included in ShowFlow as TEXT resources in the ShowFlow.RC resource script file. The file names are of the form SFHELP_*.TXT where * is an identifying letter. These files may be edited to suit the application, but careful attention must be paid to their size: The text may not exceed SCRBUFWIDTH characters

per line or SCRBUFLINES lines total, since this is the size of the array for the help utility's screen buffer. The constants SCRBUFWIDTH and SCRBUFLINES are defined in ShowFlow.H, and may be changed if necessary. Keep in mind that increasing these values adds to the amount of memory required by ShowFlow.

3.7.2 Graph Number Modifications

These modifications need be done only if you are changing the number of graphs from the current number of three.

- SFInit.C
- The section where graph labels and other strings are loaded into the ShowFlow using the LoadString function must include additional strings associated with additional graphs.
 - Each graph window to be created must be initialized and registered as its own class with the RegisterClass function.
 - String constants must be transferred to subsequent instances of ShowFlow with the function GetInstanceData.
- SFFile.C
- The ReadDataFile subroutine is configured to initialize and read data for a set number of graphs (currently three). The initialization sets all values to zero in consecutive for loops. The reading of data arrays, while complicated looking, is modularized and easily modified. Study the logic and labels and the changes should be apparent.
- SFGraph.C
- In the variable declarations at the top of the code, variables must be added to correspond to each graph generated. In some cases, additional but similar variables are declared; in others the dimensions of an existing variable are changed.
 - Two boilerplate code modifications occur in the subroutine GraphDlgProc, involving the checkbox controls in the GraphBox dialog box.

- The subroutine GraphInit creates the graph windows, and a boilerplate section of code containing the CreateWindow function must be repeated for each.
- An entire subroutine, Graph*WndProc (where * = 0, 1, 2, ...) must be included for each graph window to process messages separately. This subroutine should imitate the existing ones, substituting the index numbers where appropriate. The new subroutine(s) must be declared in ShowFlow.H and exported in ShowFlow.DEF.
- Near the beginning of the GetGraphData subroutine, a case: statement must be repeated within the switch(iGraphType) logic for each graph.
- A similar modification must be made near the beginning of the PaintGraph subroutine.

- ShowFlow.H
- Some program constants are defined for each graph, such as the number of datasets: GR*DATASETS (* = 0, 1, 2, ...)
 - The string constants for graph axis labels (IDS_G*CAPTION, etc.) must be assigned for new graphs. These constants are assigned a numeric value here in ShowFlow.H, a string value in ShowFlow.RC, and are loaded into ShowFlow in SFInit.C.
 - Declaration of the subroutines labelled Graph*WndProc must reflect the number of graphs generated.

- ShowFlow.RC
- The strings used for graph axis labels (IDS_G*CAPTION, etc.) must be defined for new graphs. These constants are assigned a numeric value in ShowFlow.H, a string value here in ShowFlow.RC, and are loaded into ShowFlow in SFInit.C.

- SFDialog.H
- The GraphBox dialog box contains a checkbox and text control for each graph, and IDs must be assigned for each in the SFDialog.H file.

ShowFlow.DLG

- The GraphBox dialog box contains a checkbox and text control for each graph. To edit dialog boxes, see the section below about the SDK's DIALOG program.

ShowFlow.DEF

- This module definition file exports all *Windows* procedures, including the Graph*WndProc (* = 0, 1, 2, ...) declared in ShowFlow.H for each graph. Add one export statement for each graph added.

3.8 Tools provided with the *Software Development Kit*

Several programming aids are provided with the SDK, including a icon editor ICONEDIT (SDKPAINT, in version 3.0), a dialog box editor DIALOG, a memory shuffler SHAKER for testing program integrity, and a memory viewing program HEAPWALK. For the purposes of converting ShowFlow, the first two will be most useful. All are well documented in the Software Development Kit.

3.8.1 DIALOG.EXE

This dialog box editor makes easy work of designing a dialog box, providing an editing screen with special commands for creating various controls, and generating a new ShowFlow.DLG dialog box description file and its associated header file SFDialog.H. To use this facility, run the DIALOG program from *Windows* and open the resource file ShowFlow.RES and header file SFDialog.H. All of ShowFlow's dialog boxes are listed in the "View Dialog..." menu, and choosing one will display it on the screen for editing. Most of ShowFlow's dialog boxes will not require modification, but, at the very least, the ModelBox and EditBox will.

The ModelBox is an "About..." box for the model program, and contains information about its origins. This box is displayed by choosing the "About the Model..." command from ShowFlow's "File" menu. To edit it, choose "MODELBOX" from the "View Dialog..." menu. Click on any controls (mostly text boxes) which you wish to modify, and a dialog box will appear which contains information about the control, including the text displayed in it and its control ID. Modify the text as desired, and make sure that the control ID is "-1", which means that there is no corresponding constant defined in SFDialog.H. This control ID should be used by all text box controls, and if you add any (from the "Control" menuitem) be sure to set it to -1. You will get a

message that another control shares this ID, which is OK for text boxes since they do not accept input. Modifying the About box for ShowFlow ("AboutBox") follows the same procedure.

The EditBox will require much more work, and is probably the most tedious part in converting ShowFlow to run another model. (Before choosing to edit the EditBox it is advised to *Maximize* the DIALOG program to use the entire screen and provide access to the menu bar.) I will assume that the changes involve only a different number of parameters with a new set of labels. If other controls like checkboxes and radiobuttons are to be added, their corresponding processing logic in other parts of the program must also be added. In any case, prerequisite to using the DIALOG program are some modifications of the header file SFDialog.H. These modifications are discussed in the *General Modifications* section above, and entail defining control IDs for the parameters: IDD_PARAM00, IDD_PARAM01, ... IDD_PARAMXX, where XX = number of parameters *minus one*, since they start at index zero. These IDs and their numeric constants must be assigned before they can be referenced in the DIALOG program.

Bring up the EditBox dialog box for editing in the DIALOG program. Inspection will reveal that the edit controls (rectangular boxes for entering data) are identified by their control IDs and are ordered consecutively. The ordering is important, so if fewer parameters are used, delete the extra ones from the end. Likewise, if adding them, add them to the end and label them with consecutive IDs. Adding edit box and text box controls is done by choosing these controls from the "Control" menu in the DIALOG program. After you have added enough edit boxes you may begin changing and adding their text box labels, remembering to assign all text boxes a control ID value of -1. The edit boxes should be assigned the numeric constant corresponding to their control IDs, as defined in SFDialog.H. Edit boxes and text boxes may be resized and moved around on the screen as long as the ordering is preserved, since this ordering is followed in moving through the dialog box with the TAB key. If the width of the edit boxes is changed, the value of MAXPARAMLEN (defined in ShowFlow.H) should be changed to reflect the maximum number of characters which can fit in the box.

Using the DIALOG program will take practice. It is sometimes difficult to position items exactly on the screen. For fine tuning, the ShowFlow.DLG code may be edited directly with a text editor to change positions of controls. This is done by changing the last four numbers in each control description record -- these define the position and size of the control on the screen. This technique is useful in lining up the final arrangement of controls.

3.8.2 ICONEDIT.EXE

The SDK provides a utility for creating small bitmaps for use as program icons. *Windows SDK version 3.0* has replaced ICONEDIT with SDKPAINT. The use of either program is straightforward and self-explanatory. Once the icon has been designed, it is saved in a *.ICO file (ShowFlow.ICO is one), and included as a resource in the ShowFlow program in the ShowFlow.RC resource script file.

3.9 Recompiling ShowFlow

Compiling the ShowFlow *Windows* application is made simple by using Microsoft's MAKE utility and the files ShowFlow.MAK and ShowFlow.LNK, which contain the compiling and linking commands, respectively. All of the modules listed in ShowFlow.MAK must be in the current directory. This includes all of the source code modules as well as the WSPAWN.OBJ spawning code and the WIN87EM.EXE math coprocessor library. Once all of the source codes have been edited, enter the following command from the DOS prompt:

```
MAKE SHOWFLOW.MAK
```

and the MAKE utility will take over.

ShowFlow.MAK can be constructed to compile ShowFlow.EXE for use with or without a math coprocessor, and with or without the debugging aids. For more information, refer to the comments in the ShowFlow.MAK file pertaining to necessary files and compiler switches.

Chapter 4

Application of ShowFlow to SSGPLUME and Other Groundwater Models

The purpose of ShowFlow is to provide an intuitive and productive interface for groundwater modeling, acting as a shell for the SSGPLUME model described in Chapter 2. To this end, ShowFlow performs four basic functions: file management, pre-processing of input data, model execution, and post-processing of data generated by the model. In addition to this, *Windows* provides for the transfer of data between applications. The details of using ShowFlow are covered in Appendix A: User's Guide to ShowFlow. This chapter illustrates the use of ShowFlow and SSGPLUME in simulating groundwater contamination scenarios.

4.1 Specifying Input Data

The Parameter File Editor is used to specify the input parameters for the model, and is invoked with the "Edit Parameters" command. This is a self-explanatory form, with each data item identified by a label and an entry field. Units to be used are provided in the label. A sample form is shown in Figure 2, and each parameter field is explained below. Where appropriate, symbols used in the governing equations described in Chapter 2 have been included for reference.

Parameter File Editor for SSGPLUME			
Run Title:	Current File:	SITE123A.PAR	OK
Site 123A parameters			Cancel
SITE DESCRIPTION:		CONTAMINANT CHARACTERISTICS:	
X minimum point (m)	0	Loading rate (g/m ² /d)	1
X maximum point (m)	30	Water bulk partition coeff	0.1
X resolution (# of points)	100	Henry's Law const (no dim)	0.01
Y minimum point (m)	0	Schmidt number	1.0
Y maximum point (m)	4	(Range 0.2 to 5.0; 1.0 typical)	
Y resolution (# of points)	100	AQUIFER CHARACTERISTICS:	
Width of facility (m)	4	Seepage vel. along X (m/d)	1
Area of facility (m ²)	10	Dispersivity in X (m)	0.5
10-m wind velocity (m/s)	1	Dispersivity in Y (m)	0.1
Water infilt. rate (m/d)	0.05	Retardation factor	2
VADOSE ZONE CHARACTERISTICS:		Degrad'n rate const. (1/d)	0.01
Vadose zone thickness (m)	10	Regional infilt rate (m/d)	0.005
Contam'ed zone thick. (m)	2	Saturated thickness (m)	20
Degrad'n rate const. (1/d)	0.01	Porosity	0.3

Figure 2. "Parameter File Editor" dialog box. This is a simple error-checking form for entry of SSGPLUME parameters.

SITE DESCRIPTION:

X minimum point (m): X_{\min} and

X maximum point (m): X_{\max}

These limits specify the range of area over which SSGPLUME will calculate contaminant concentrations, with X measured in the direction of groundwater flow, with the origin at the edge of the surface facility.

X resolution (# of points)

This is the number of points in the X direction of the solution grid. More points will result in smoother graphs, but also in larger files. The maximum number of points is defined in ShowFlow.H as MAXDATAPTS and is currently set at 100. Experiment with this number to find a minimum number of data points which still produces useable graphs. When graphic resolution is important, the data point resolution may be changed to the maximum value of 100.

Y minimum point (m): Y_{\min} and

Y maximum point (m): Y_{\max}

These limits specify the range of area over which SSGPLUME will calculate contaminant concentrations, with Y measured perpendicular to the direction of groundwater flow, with the origin at the edge of the surface facility.

Y resolution (# of points)

See notes above for X resolution.

Width of facility (m): W

Area of facility (m^2): A_f

The areal size of the facility or disposal site is defined by its width transverse to groundwater flow direction and its area, assuming a rectangular shape.

10-m wind velocity (m/s): U_{10}

This is the wind velocity at the site, measured in meters per second at a height of 10 meters from the ground surface.

Water infiltration rate at facility (m/d): I_f

I_f is the NET infiltration rate of water at the site.

VADOSE ZONE CHARACTERISTICS:

Vadose zone thickness (m): L_v

This is the thickness of the entire vadose (unsaturated) zone.

Contaminated zone thickness (m): L_U

This is the thickness of the upper part of the vadose zone which has been directly contaminated.

Degradation rate constant in vadose zone (d^{-1}): λ_v

This is the first-order degradation rate constant for the constituent in the vadose zone. A different constant is defined for the saturated zone.

CONTAMINANT CHARACTERISTICS:

Loading rate / area (g/m^2-d): m_L

The mass loading rate is calculated as mass flux per unit area. The units employed here determine those of the final groundwater concentration. If m_L is expressed in g/m^2-d , the resultant concentrations will be in g/m^3 (mg/L). If concentrations units

of g/L are desired, then m_L should be given in kg/m²-d, though the graph labels will still read "mg/L".

Bulk water partition coefficient: B_w

This is the volume of pollutant in the aqueous phase divided by the total volume.

Henry's Law constant: K_H

This Henry's Law constant is calculated as the pollutant concentration in the gaseous phase divided by that in the water phase.

Schmidt number: (ν_g/D_g)

The Schmidt number is the dimensionless ratio of the kinematic viscosity ν_g and the diffusivity D_g in water of the gaseous phase of the constituent. D_g and ν_g may be expressed in any identical units.

AQUIFER CHARACTERISTICS:

Seepage velocity along X (m/d): v

This is the groundwater longitudinal seepage (not Darcy) velocity.

Dispersivity in X (longitudinal) (m): a_L

The dispersivity is used to calculate the dispersion coefficient D_{xx} as $D_{xx} = a_L v$.

Dispersivity in Y (transverse) (m): a_T

The dispersivity is used to calculate the dispersion coefficient D_{yy} as $D_{yy} = a_T v$.

Retardation: R

Retardation is used to quantify processes which inhibit migration of the pollutant through the porous medium. R is a scalar variable, so that a value of $R = 1$ implies no retardation, and values $R > 1$ imply retardation.

Degradation rate constant in the aquifer (d⁻¹): λ

This is the first-order degradation rate constant for the constituent in the saturated zone. A different constant, λ_v , is defined for the vadose zone.

Regional infiltration rate (m/d): I_R

The NET regional infiltration rate (groundwater recharge rate) is used to calculate dilution of the contaminant plume by recharge water.

Saturated thickness (m): b

This is the average saturated thickness of the aquifer.

Porosity: n

This is the porosity of the saturated aquifer.

When the Parameter File Editor is first displayed, the text in the "Run Title" edit box is pre-selected for editing. The reverse video means that it will be replaced by any text entered on the keyboard, or can be deleted entirely with the DELETE key. The Run Title is preselected since it will usually be changed.

Other editing fields are accessed by stepping through them with the TAB and SHIFT + TAB keys or simply clicking on them with the mouse cursor. Each field may be edited by using the DELETE, BACKSPACE, or DIRECTION keys, or by positioning the cursor or selecting text by dragging the "I beam" cursor over it. The numeric fields are limited to a few characters, but by using scientific notation ("1.2e10" or "2.54E-5") all practical values should be available. Since SSGPLUME is written in the C language rather than FORTRAN, values may be entered in any format, with or without decimal points.

If the user wishes to abandon the current changes, the "Cancel" button (or ESC key) will close the Editing window and ignore the changes.

When the user selects "OK" (or ENTER) to accept the changes made in the Parameter File Editor, ShowFlow scans the fields for errors. Error messages (Figure 3) are generated if any data fields are deemed invalid according to the tests run in the SFEdit.C module. These tests scan all

numeric fields for illegal (non-numeric) characters, and test several field values for range limitations. (Legal characters for the fields are contained in the program constant IDS_OKCHARS defined in the resource script ShowFlow.RC.) Range limitations vary, but may test for values to be positive, to be between zero and one, or to be non-zero, for example. Any field which contains an illegal character or an out-of-range value will produce a Message Box containing information about the problem. After the user chooses "OK" (or ENTER) acknowledging the message, the cursor is placed on the field in question to facilitate correction. If no errors are detected, control is returned to the main ShowFlow window and the parameters are ready to save in a new file with the "Save As..." command.

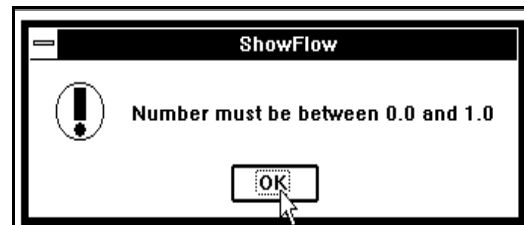


Figure 3. Data entry error message. The Parameter File Editor displays error messages if any field contains invalid data.

4.2 Displaying Results

After the model has been run, the "Graph Results" function of ShowFlow produces three pre-set graphs showing the shape of the contaminant concentration plume calculated by the SSGPLUME model (Figure 4).

ShowFlow's Representation of the Contaminant Plume Generated by the Model SSGPlume

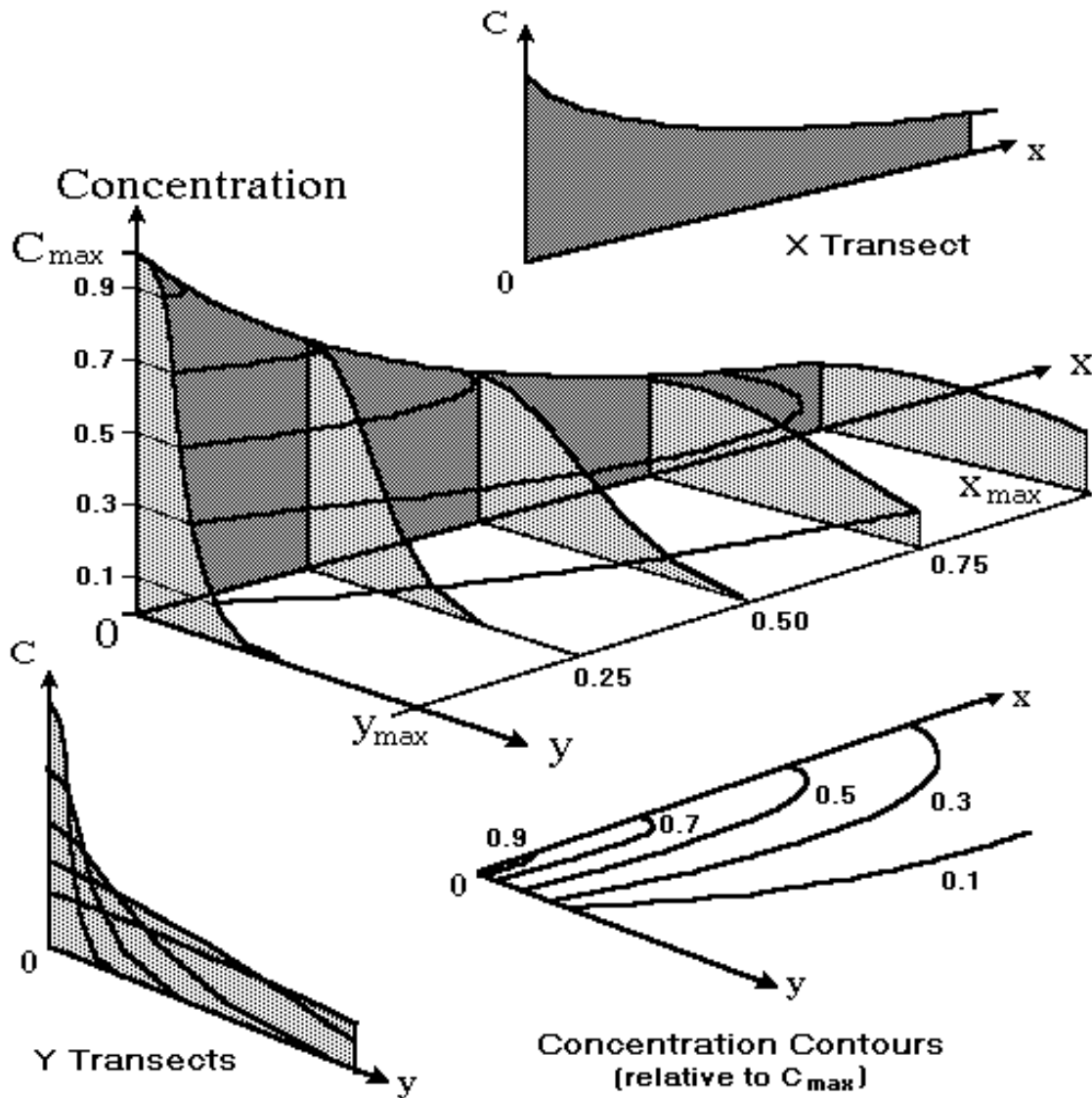


Figure 4. Graphical representation of the plume. ShowFlow uses three two-dimensional graphs to represent the shape of the contaminant plume as generated by SSGPlume.

The three graphs are (1) a transect through the plume along the X-axis, (2) five transects parallel to the Y-axis at 0.00, 0.25, 0.50, 0.75, and 1.00 times the maximum value of X, and (3) a concentration contour map with contours at 0.1, 0.3, 0.5, 0.7, and 0.9 times the maximum concentration of the plume in the saturated zone. The mathematics behind these calculations is discussed in Chapter 2: Development of the SSGPLUME Ground-water Contaminant Transport Model.

Choosing "Graph Results" produces a dialog box which allows the user to select which graphs are to be generated or to cancel the procedure as shown in Figure 5. Selecting "OK" causes ShowFlow to draw the three graphs in small "child windows" below the "parent" ShowFlow main window (Figure 6). These graph windows can be resized or moved about the screen, and the flexible images within them can be copied to the Clipboard or sent directly to the printer for hardcopy. As a

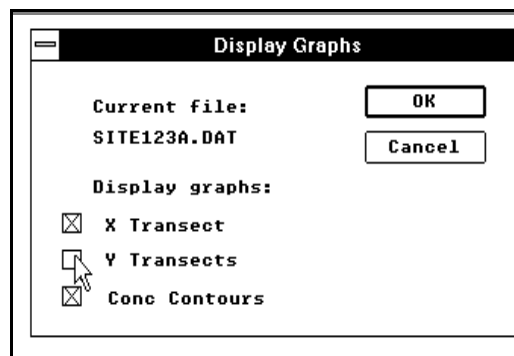


Figure 5. "Display Graphs" dialog box. This allows the user to choose which graphs to generate.

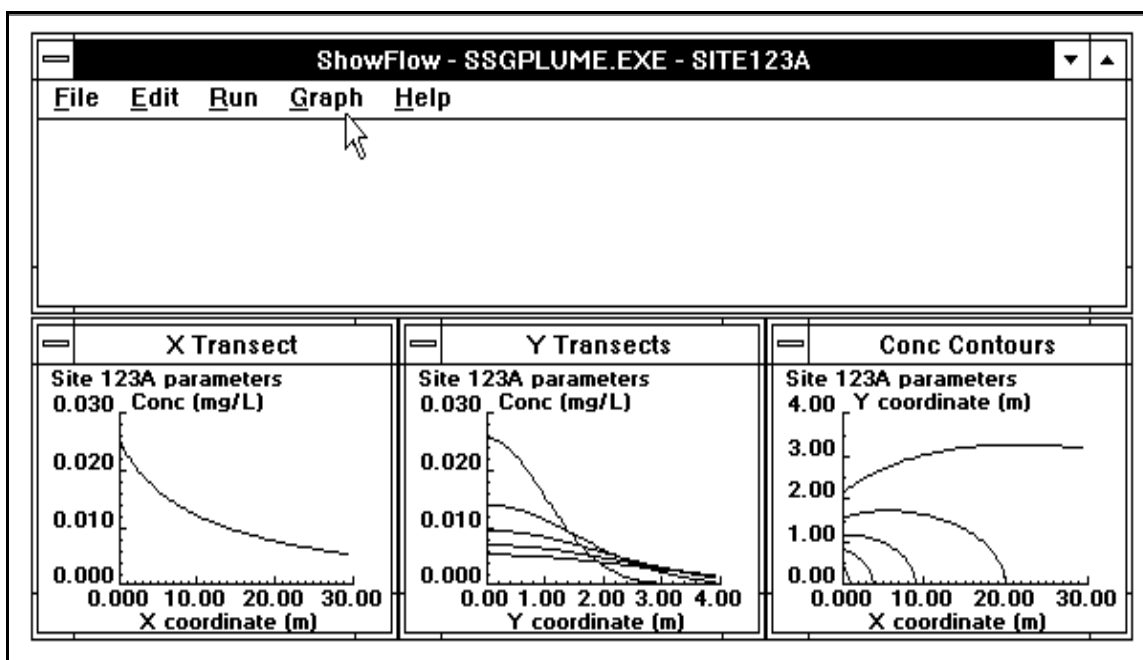


Figure 6. Standard ShowFlow graphs. ShowFlow represents the contaminant plume using three two-dimensional graphs.

graph window is changed in size, the tick marks and labels are repositioned so that labels do not overlap and tick marks are not too close together. Each window is tied to a unique data set, and so takes up one or two kilobytes (KB) of memory, but ordinarily there should be sufficient resources to display dozens of graphs simultaneously.

Each time the "Graph Results" procedure is run, these graphs are produced in the

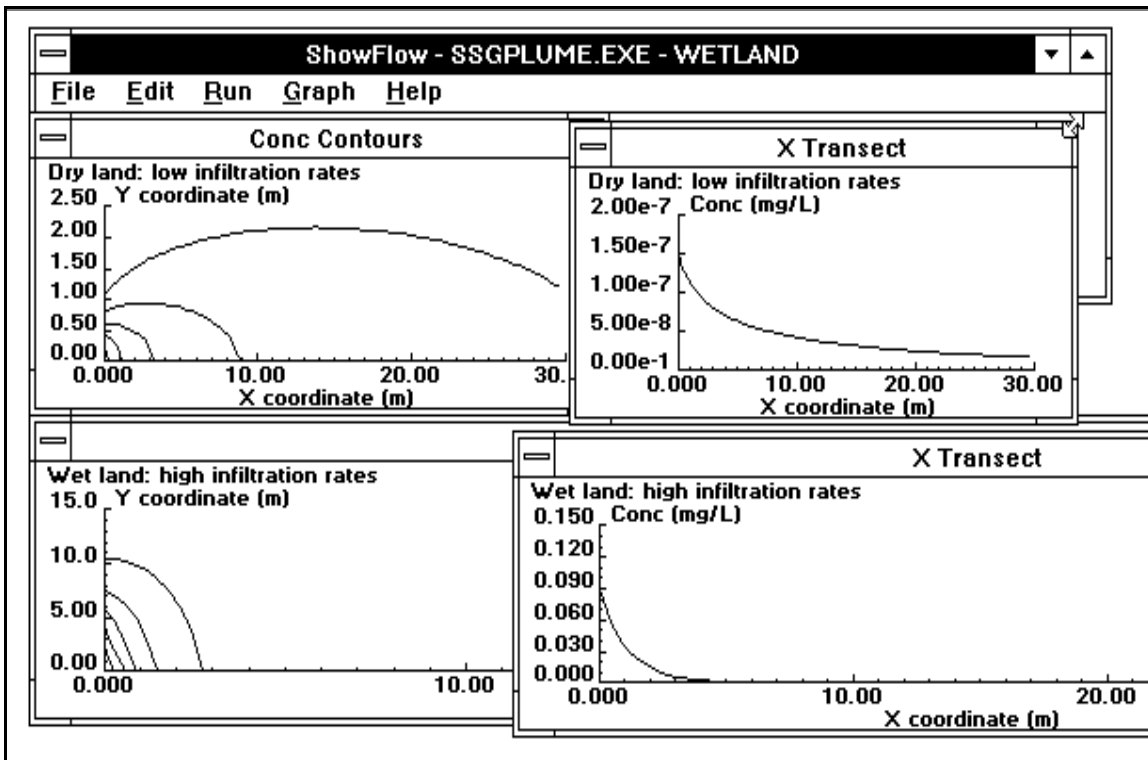


Figure 7. On-screen comparison of results. This is an easy and efficient way to examine the results of various simulations.

same locations on the screen, so unless they are moved they will be covered by the next set of graphs. To help organize the screen, the user may find it convenient to make the windows smaller and arrange them on the screen so that only their unique graph titles are visible (Figure 7). Note that for smaller graph windows only the title is drawn, since the graphs would be too small to be of use. This saves on time needed to "re-paint" the graph windows.

Graphs from different modeling scenarios can be compared side-by-side by manipulating their sizes and position on the screen. Figure 8 shows an example demonstrating

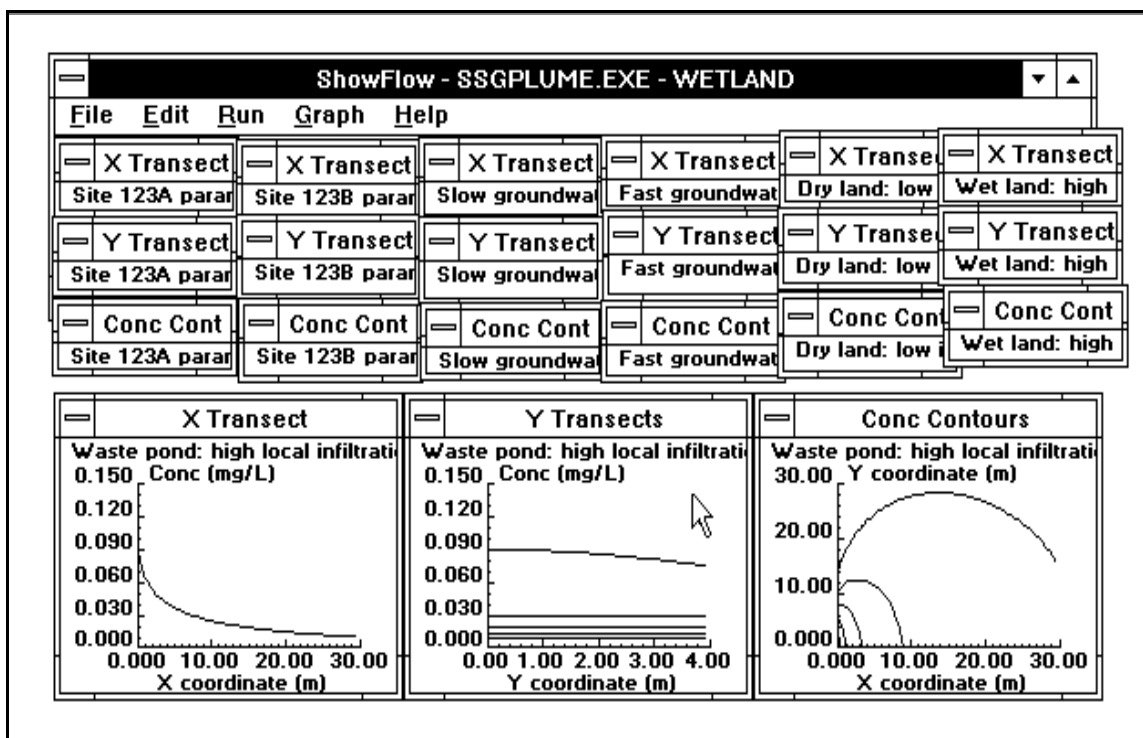


Figure 8. Multiple simulations. Graphs from several simulations can be stored on the screen for careful examination.

the different plume shapes generated by SSGPLUME given wet and dry conditions. In this example it is important to remember that the concentration contours are measured relative to the peak concentration, determined from the transect graphs. The contours show only the shape of the plume.

Capturing the graphs displayed by ShowFlow can be done with the "Copy Graph" and "Print Graph" commands as described in Appendix A: User's Guide to ShowFlow. "Print Graph" generates a printer hardcopy which is as close as possible to the actual screen size of the graph. "Copy Graph" gives a copy of the graph to the Clipboard, from which it can be saved to a *.CLP file or pasted into a PAINTBRUSH drawing (Figure 9), or as a figure in a document in any of several *Windows* word processors (Figure 10). The flexibility of ShowFlow's graphs and its ability to share them with other applications is a direct result of programming ShowFlow in the *Windows* graphical environment.

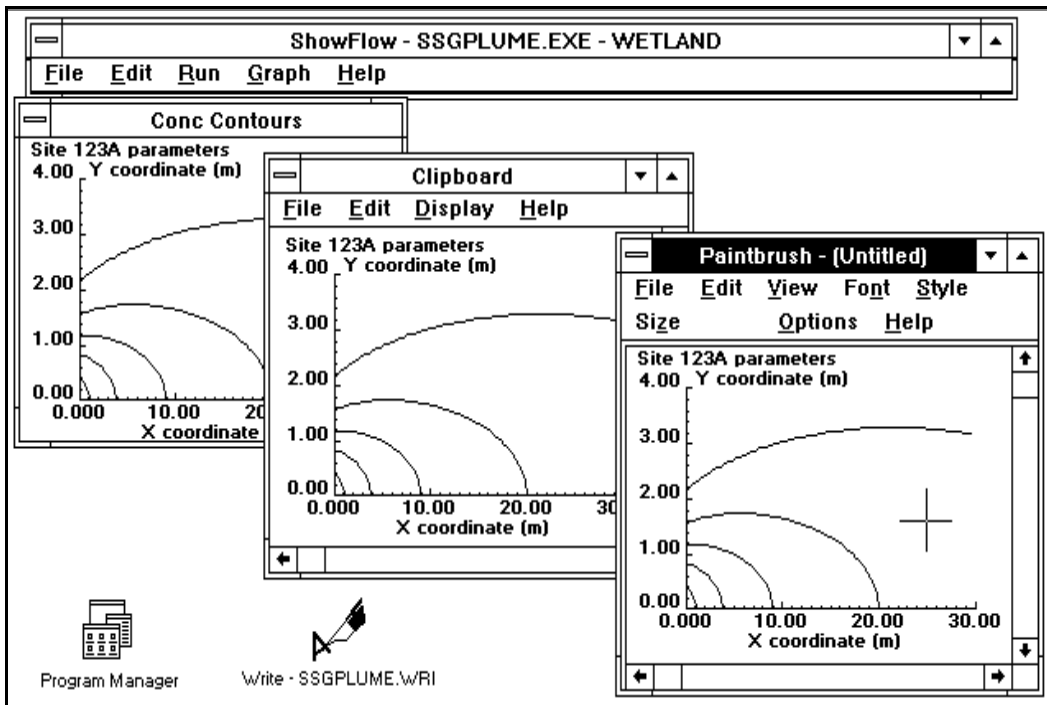


Figure 9. Cutting and pasting graphs. ShowFlow makes use of the *Windows* Clipboard to copy and paste a graph into PAINTBRUSH.

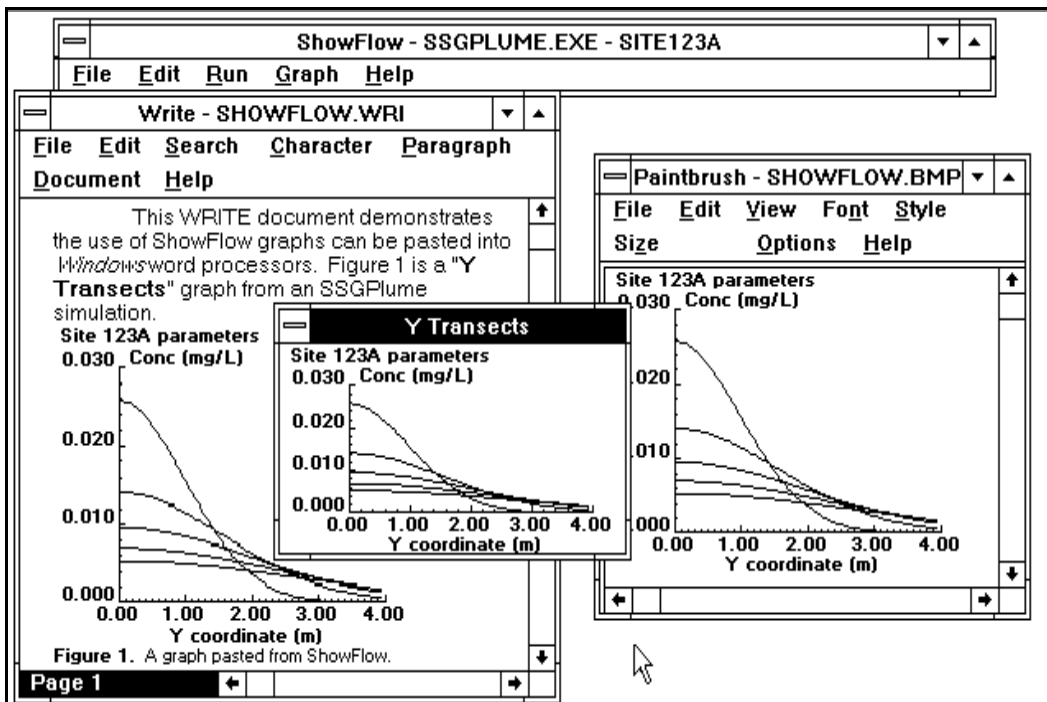


Figure 10. ShowFlow and other programs. *Windows* allows for ShowFlow to run simultaneously with PAINTBRUSH and WRITE, so that figures and documents may be created during a modelling session.

4.3 Interpretation of Results

The concentration value at a particular location can be determined by examining the graph. For example, for locations directly downgradient of the source, concentrations can be read directly from the **X Transect** graph shown in Figure 11. In this example, the concentration at $y = 0$, $x = 15$ m, is 0.20 mg/L (or g/m^3).

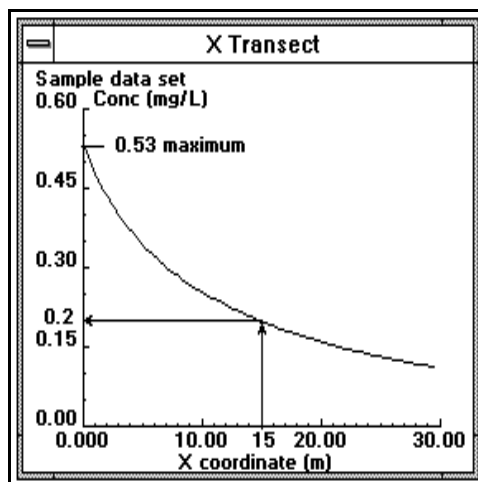


Figure 11. Interpretation of the **X Transect** graph. This plots contaminant concentration versus x for $y = 0$.

For off-axis locations ($y \neq 0$), the user must interpolate using the **Y Transects** and/or **Concentration Contours** graphs, shown in Figures 12 and 13. For example, if the location of interest is at $x = 15$ meters and $y = 1.5$ meters, there are two ways to find the contaminant concentration.

Using the **Y Transects** graph, note that since the maximum value of x is 30 m, the middle line represents the **Y transect** at $x = 0.5(30 \text{ m}) = 15 \text{ m}$. This is the desired x position. From the graph, it is apparent that for this transect and $y = 1.5 \text{ m}$, the concentration is about 0.15 mg/L.

Alternatively, using the **Concentration Contours** in Figure 13, the point (15 m, 1.5 m) lies just outside the 30% contour, at about 28% of the maximum concentration. This maximum occurs at the origin and can be read from either the **X Transect** or **Y Transects** graphs as 0.53 mg/L. 28% of this maximum is $0.28(0.53 \text{ mg/L}) = 0.15 \text{ mg/L}$.

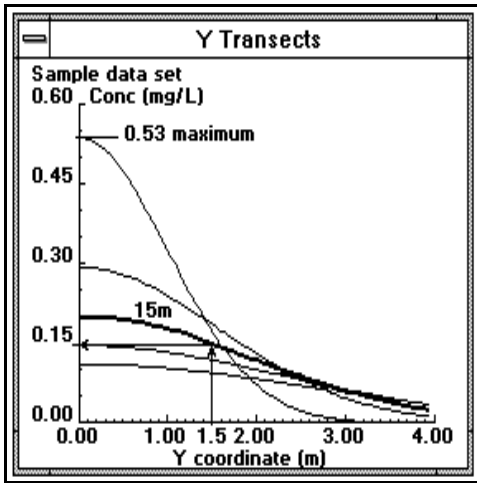


Figure 12. Interpretation of the **Y Transects** graph. This plots contaminant concentration versus y at five evenly-spaced intervals along x.

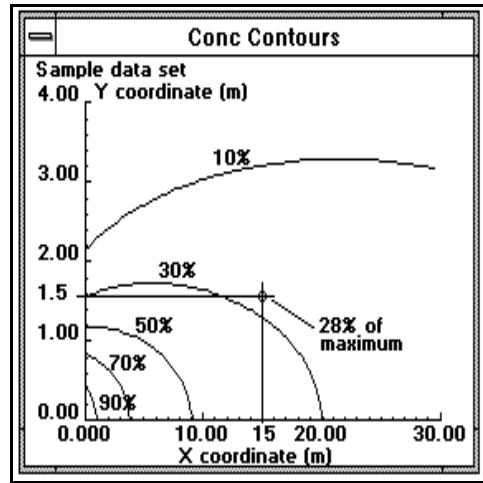


Figure 13. Interpretation of the **Concentration Contours** graph. This plots contaminant concentration over the xy plane, with contour intervals at 10, 30, 50, 70, and 90 percent of the maximum concentration.

Chapter 5

Conclusions and Recommendations

Conclusions

The principal goal of this research was to develop a productive interface for the development and use of groundwater models. As outlined in the introduction, the most desirable qualities of this interface are that it should

- provide an environment which is easy to use.
- provide for unambiguous and error-free entry of model parameters.
- have the ability to generate useful graphs of model output.
- have the ability to transfer data and graphs to other media and programs.
- be independent of the model program, so that the developer may modify the model separately.
- make efficient use of the computer's resources.
- run on a widely-used platform.
- provide adequate on-line help.
- allow for quick comparison of simulation results.

As demonstrated in the preceding chapters and in the following User's Guide to ShowFlow each of these objectives has been fully achieved.

Recommendations

The ShowFlow idea has a bright future. *Windows* is rapidly becoming the primary platform for new applications for IBM-type machines, and ShowFlow, though relatively elementary in comparison, is in good company with sophisticated word processors, spreadsheets, and analytical instrument interfaces. As the availability of useful *Windows* programs increases, so will the number of *Windows* users and hence the number of users who will feel immediately at home with ShowFlow's interface.

The limitation of available memory has long been a problem for many software developers, and every attempt was made to keep ShowFlow trimmed down to minimum memory consumption.

However, as machines become upgraded with additional extended memory for use by *Windows*, the old memory barriers will fall. There are several enhancements to ShowFlow which could be implemented as memory consumption becomes a less serious limitation:

- ShowFlow could be modified to run several different models from the same interface, each with its own Parameter File Editor screen.
- The graphing utility could incorporate some user preferences for font style and colors.
- An interactive graph utility would be useful and fun, providing the user with a way to choose the positions of transect graphs, or involving rotatable projections of three-dimensional plume surfaces.

The children of ShowFlow may incorporate these and other enhancements, but such programs will be memory-intensive and could not be used on most of today's installed base of IBM-style computers.

BIBLIOGRAPHY

- Andrews, N., Running Windows: The Microsoft Guide to Windows 2.0, Windows/286, and Windows/386, 2nd ed., Microsoft Corporation, 1988
- Andrews, N., Windows: The Official Guide to Microsoft's Operating Environment, Microsoft Corporation, 1986
- Huyakorn, P.S., M.J. Unga, E.D. Sudicky, L.A. Mulkey, and T.D. Wadworth, RCRA Hazardous Waste Identification and Land Disposal Restrictions Groundwater Screening Procedure, Draft Report, United States Environmental Protection Agency, 1985
- Mackay, D.M., W.Y. Shiu, A. Bobra, J. Billington, E. Chau, A. Yuen, C. Ng, and F. Szeto, Volatilization of Organic Pollutants from Water, Environmental Research Laboratory Report No. EPA 600/3-82-019, United States Environmental Protection Agency, 1982
- Microsoft C Version 5.1 Optimizing Compiler: *Run-Time Library Reference*, Microsoft Corporation, 1984-1987
- Microsoft C Version 5.1 Optimizing Compiler: *User's Guide, Language Reference*, Microsoft Corporation, 1984-1987
- Microsoft Windows Software Development Kit Version 2.0: *Programmer's Learning Guide*, Microsoft Corporation, 1984-1987
- Microsoft Windows Software Development Kit Version 2.0: *Programmer's Reference*, Microsoft Corporation, 1984-1987
- Microsoft Windows Software Development Kit Version 2.0: *Programming Tools, Application Style Guide, and Microsoft Windows Extensions*, Microsoft Corporation, 1984-1987
- Microsoft Windows: Questions and Answers, Microsoft Corporation, 1984-1987
- Microsoft Windows User's Guide Version 3.0, Microsoft Corporation, 1985-1990
- Myers, B. and C. Doner, Graphics Programming Under WINDOWS, SYBEX Inc., 1988

Petzold, C., Programming WINDOWS: The Microsoft Guide to Programming for the MS-DOS Presentation Manager: Windows 2.0 and Windows/386, Microsoft Press, 1988

Purdum, J., C Programming Guide, 2nd Edition, Que Corporation, 1985

Purdum, J., and T. C. Leslie, C Standard Library, Que Corporation, 1987

Smith, V.J., and R.J. Charbeneau, Probabilistic Soil Contamination Exposure Assessment Procedures, ASCE Journal of Environmental Engineering, 116(6), pp. 1143-1163, December, 1990

United States Government Printing Office Hazardous Waste Management System: Land Disposal Restrictions; Proposed Rule, Federal Register 51 Number 9, January 14, 1986

Woodward-Clyde Consultants, Oakland, CA Background Document for EPA's Composite Landfill Model (EPACML), Unpublished draft, 1989