# *Chapter Two. Simulating Surface and Subsurface Water Flows*

## 2.1. CONCEPT OF OBJECT-ORIENTED PROGRAMMING

As this study relies heavily on the concept of object-oriented programming, this section briefly reviews the history of object-oriented programming and the definition of object-oriented programming terms that will be used. After this review, the concepts of object-oriented programming will be compared with those of procedural programming languages to identify their differences.

The definitions of the terms given in this section come from the book Object-Oriented Programming (Gunther, 1994), with some modifications based on other references in the field.

Object-oriented programming (OOP) originated with the programming language Simula (Dahl 1966). This language was designed as a tool for simulating physical processes, such as water flow on a river system, that take place in the real world. The notion of objects was probably first introduced in Simula in the 1960s, but it was not formally defined until the early 1980s when the language Smalltalk was developed at the Xerox Research Center in Palo Alto (Goldberg, 1983).

One of reasons why OOP is growing rapidly is that it is simple in concept and resembles physical reality. The principal strength of the object-oriented programming language lies in its ability to handle complexity in a transparent and close-to-nature manner (Razavi, 1995). The major concepts of OOP are object, class, and inheritance. By definition, objects with the same attributes (states) and behavior are grouped into a single class. Subclasses can be generated from an

existing class and these subclasses inherit all the states and behaviors of their parent class. It is the concept of inheritance that gives OOP the ability to handle complex problems and the ability to combine the efforts of multiple programmers and researchers (Wegner 1990). For example, a class created by programmer A can be taken and used by programmer B to generate a new subclass by adding new attributes and methods (element functions) to it. Because the new subclass automatically inherits all the attributes of its super-class created by A, programmer B does not need to know how these attributes are constructed but may concentrate on the construction of the new attributes and methods. This concept of class-inheritance is also a plus for program debugging because if anything ever goes wrong with the new subclass, programmer B needs only to concentrate on the new attributes and methods he or she added to the new subclass (assuming its super-class is properly designed and A is a good programmer). The research efforts of programmers A and programmer B can easily be used by programmer C to construct his subclasses, which will inherit all the attributes and methods developed by programmers B and A. In this way, the research efforts and results of different programmers and researchers can be accumulated to form a complex system. To better understand OOP, some common terms are described below:

Objects: Objects are structures with state and behavior. Objects can cooperate to perform complex tasks and can communicate with each other by means of messages. Objects also have precise interfaces specifying which messages they accept. The closest counterpart of an object in procedural programming is a record or a structure in C or FORTRAN.

Class: Classes describe the properties of objects. Classes in OOP can be viewed as (1) a means of identifying objects with the same properties, which can be used to distinguish objects with different structure and behavior; (2) a

structuring mechanism, similar to a procedure, which improves the readability and maintainability of programs, and (3) a means of creating new members (objects) of the class. The relationship between objects and their classes is many-to-one. Each object belongs to exactly one class while one class can have many objects. For example, a subwatershed can be a polygon object and the collection of all subwatersheds in a river basin form a subwatershed polygon class.

Inheritance: Inheritance refers to the propagation of properties from super classes to subclasses. This property allows OOP to derive new classes from the existing classes. This concept does not exist in procedural programming languages.

Types: Type is a property of variables and expressions, .e.g., integer type, real type, character type, etc. There is no difference between types in procedural and OOP language.

Object References: Object references can be viewed as pointers to objects. They are variables pointing to the memory area where an object is stored. Therefore, they are very similar to a pointer used in a procedural programming language.

Variable: Variables contain object references. A variable can be viewed like a pointer in programming language C. Variables within classes are class variables and variables within objects are instance variables. A class variable is similar to a global variable in a procedural programming language, while an instance variable is similar to the field variable or record component, e.g. type auto in C.

Messages: Messages are the way objects communicate with one another. The invocation of message is called a message send while the object that is to

process the request is called the receiver of the message. Messages are similar to procedural calls in a procedural programming language.

Methods: Methods are the algorithms attached to each object to perform the requests sent by other objects via messages. Methods correspond to procedural declarations in procedural programming languages.

Subclass and Superclass: If class B is derived from class A, then class B is subclass of A, while A is the superclass of B. An object from a subclass inherits all the attributes and behavior from its superclass. For example, two classes: feature attribute table class (FTAB) and value attribute table class (VTAB) exist in the ArcView. A FTAB table is a database table connected to a GIS map and a VTAB table is just a regular database table without direct map connection. Feature attribute table (FTAB) is a subclass of value attribute table (VTAB). Therefore, an FTAB table inherits all the attributes (field types) from VTAB class. What makes FTAB a new class is that an FTAB table has a new SHAPE field, which does not exist in its superclass VTAB.

Dynamic Binding (Late Binding): Dynamic binding is the mechanism that enables an object to decide what action to take and how to act at run time. In contrast to dynamic binding is static binding or early binding, in which all the actions of an object are decided at the time when the program is constructed. For example, a function can be defined as Z=x*y to compute the multiplication of two variables x and y. Dynamic binding make it possible that the type of return value Z depends on the types of x and y. That is, when x and y are of integer type, the return value is also of integer type and when x and y are of float type, the return value is also of float type.

Before ending this section, two additional OOP facts that are important to this study should be pointed out: (1) the logical parallel between classical procedural programming and OOP and (2) the mutual independence of behavior and state of an object.

The differences between procedural programming and OOP lie mainly in how the programs are organized, the modules are called, and the variables are stored and retrieved. As to procedure itself and program construction, the logic applied to classical procedural programming is very similar that used in OOP. Therefore, the programming logic used in procedural programming can be easily applied to OOP.

Although in an object-oriented programming language, state, behavior, and interface are used jointly to define an object, they can each be defined independently from one another. This fact can be used to support the design of a generic GIS database management system. In the context of hydrologic analysis, the state of an object can be described by the attributes of a stream section or a river basin while the behavior can be viewed as the hydrologic processes occurring on a river section or a river basin. Since state and behavior are independent, they can be treated separately with state variables being stored and managed in a GIS database and behaviors being described by various models.

## 2.2. CONCEPTUAL DESIGN OF AN INTEGRATED HYDROLOGIC MODEL

This section describe how the concepts of object-oriented programming will be used to design a map-based surface flow simulation model.

The classes of polygon and line objects are of essential importance to this study because river basins can be represented by polygon objects and rivers can be

represented as line objects. The equation, object=state+behavior will be used to define these two classes of objects.

- **River Basin and Polygon Classes**

For a given object in the polygon object class, its state can be described by area, perimeter, shape, etc. Its behaviors are drawing-itself, coloring-itself, getting-dimension, returning-center etc. Getting-dimension, returning-center, and drawing-itself are the names of element functions (methods) of the objects that perform the tasks of getting the dimension sizes, returning the center point of the polygon object, and drawing and coloring the object. Element functions are the functions that are defined by a class to be associated with an object of the class.

River basin polygons can be viewed as a subclass derived from the polygon object class. Therefore, for a given river basin object, its state can be described by the properties it inherits from the polygon object class plus its own unique state properties, such as soil type, rainfall depth, slope, streams it contains, adjacent basins, hydraulic conductivities $K_x$ and $K_y$, etc. For the same reason, the behavior of a river basin object can also be described by the behavior properties that it inherits from polygon object classes plus the behaviors of all kinds of hydrologic and hydraulic processes, which can be described by different hydrologic and hydraulic models.

- **River Section and Line Classes**

For a given object in the line object class, its state can be described by its length, To-Node (Tnode), From-Node (Fnode), Left-Polygon (Lpoly), Right-Polygon (Rpoly), shape, etc. In an ARC/INFO arc coverage, Fnode and Tnode are used to denote starting point and ending point IDs while Lpoly and Rpoly are

used to denote the IDs of polygons to the left and to the right of a line. An object's behaviors may be drawing itself, coloring itself, getting-dimension, returning-center, getting-end-point, getting-start-point, etc. Again, getting-dimension, returning-center, etc., are the names of the element functions of an object that perform the tasks of getting the dimension sizes, returning the center point of the line, drawing and coloring the line object.

A river object belongs to the line object class. Therefore, for a given river object, its state can be described jointly by the properties that it inherits from the line object class plus the behaviors of all kinds of hydrologic or hydraulic processes which are described by different hydrologic and hydraulic models. Figures 2.1 and 2.2 show the examples of classes and objects. Figure 2.2 shows the map of the Guadalupe River Basin created by applying a watershed delineation procedure (Maidment, 1994) to a 3 arc-second digital elevation model (DEM) of the area.

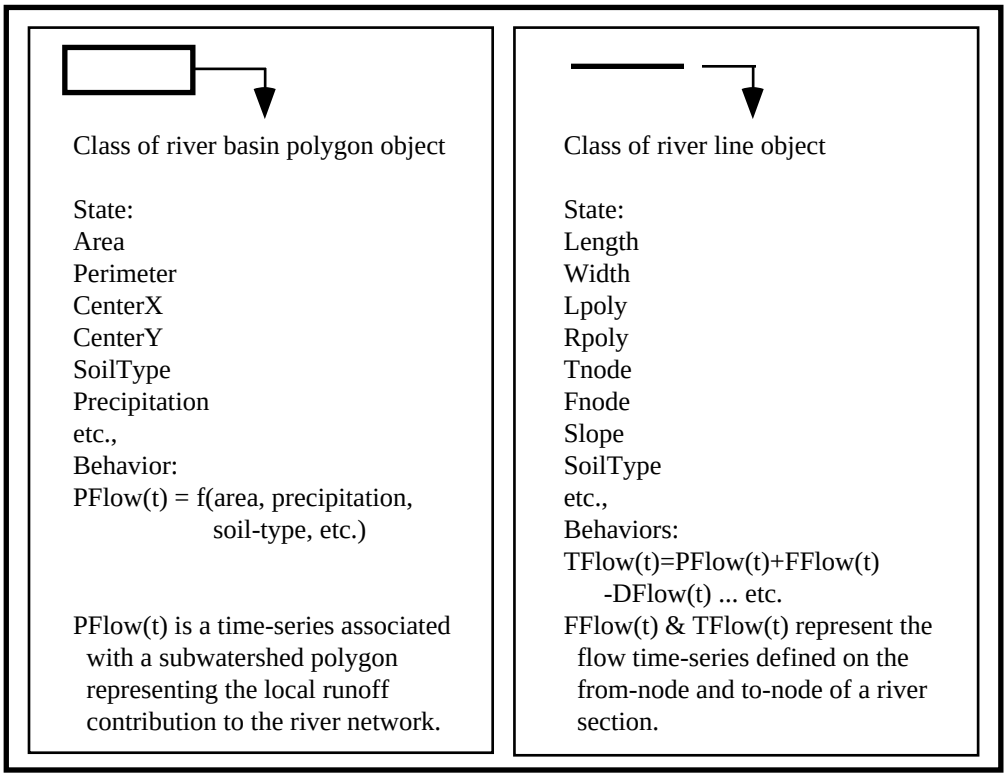


Figure 2.1. State and behavior defined on an object

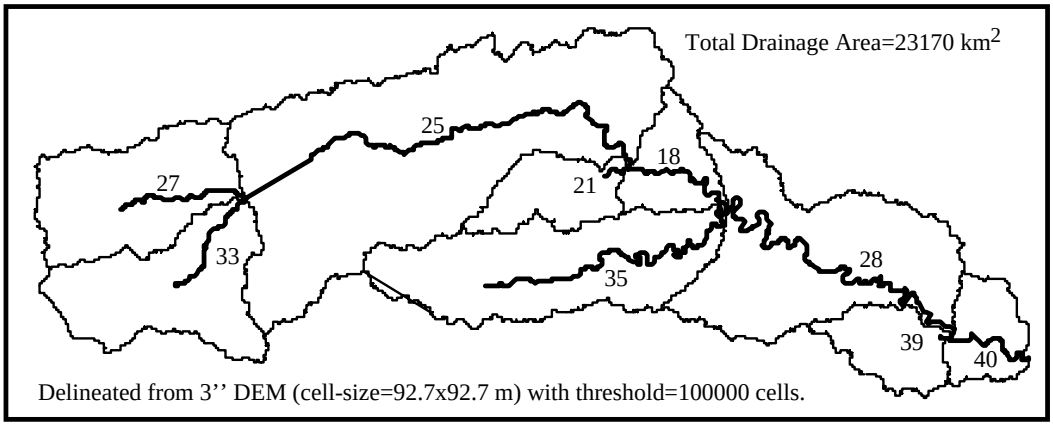


Figure 2.2. Guadalupe River Basin in Central Texas - an example of river line and watershed polygon objects

## 2.3. RELATIONSHIPS BETWEEN MAPS, DATABASES AND PROGRAMS

In order to construct a map-based simulation model, it is important to understand the relations between the maps, relational databases and programs. Figure 2.3 shows how an object is defined and referenced in an object-oriented programming language, in a relational database, and on a map. C++ is used here to illustrate how an object is defined in an object-oriented programming language.

As stated above, an object is defined by the equation: object = state + behavior. The behavior of an object is governed by some equations. Equations are usually translated into element functions. In the same way, states of an object are defined as variables of a class. The program section in Figure 2.3 illustrates how a class is defined and objects generated in C++. In this example program, objects are created in two steps. First, a class is defined and functions declared, and then the instances (objects) of the class are generated. These two steps are analogous to the actions of creating a database structure (template) and adding records to the database. When a GIS map is constructed, a relational database is also created to store the spatially-referenced data sets. Such databases appear as feature attribute tables (FTAB) in the ArcView program. In the database of a GIS map, one field is used to hold the pointers to the geographical features on the map. In a class, states (variables) and behaviors (element functions) can be defined as either private or public. A private variable/function is accessible only by other elements of the same object while a public variable/functions can be called by other objects. The distinction of private and public types of functions and variables provides a mechanism for programs to control the messages (requests) exchanged between objects.

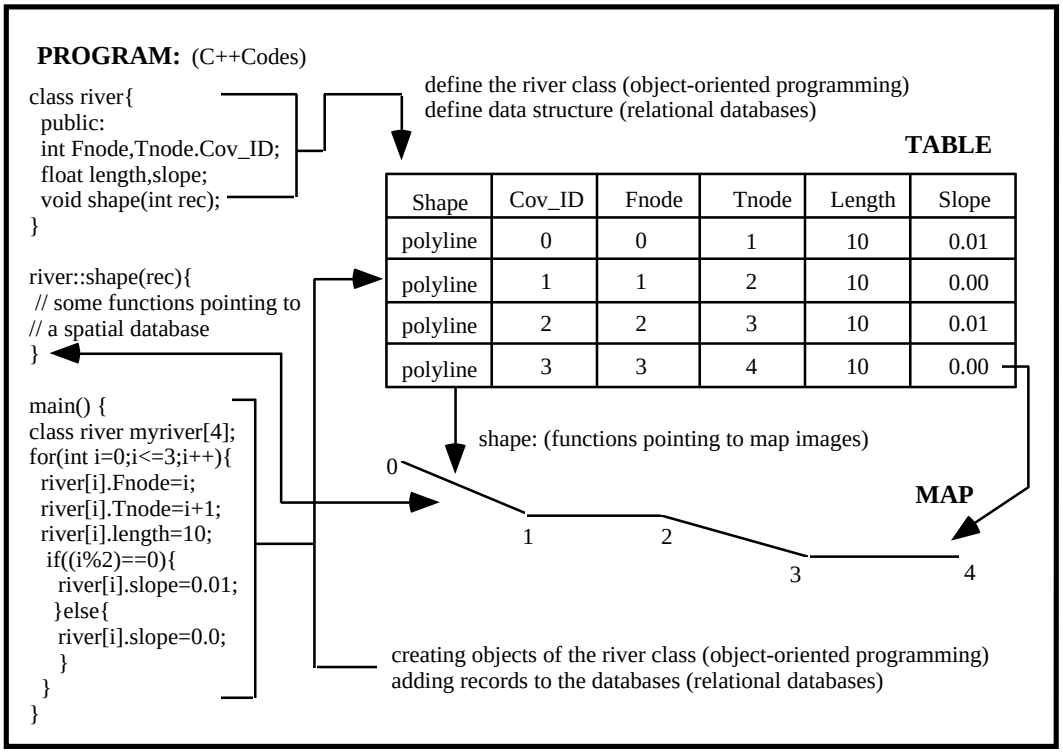


Figure 2.3. Presentation of objects in a program, database and map

## 2.4. GOVERNING EQUATIONS FOR SURFACE AND SUBSURFACE WATER FLOWS

Given a numerical simulation model, there are always two components, which are mathematical equations governing the process and attribute data (extracted from maps and other sources) supporting the equations. In the context of OOP, a mathematical equation describes the behavior while a set of attribute data describes the state of an object.

The following sections review the equations governing the surface and subsurface water movements that will be used for the map-based simulation model design in this study.

### 2.4.1. Equations Related to the Surface Water Flow Simulation

Two types of models are considered for surface water flow simulation: one for stream flow and one for overland flow. Figure 2.4 shows the data flow path on the map-based surface water flow simulation model.

As it can be seen from Figure 2.4, six procedures are used to process and convert the rainfall data sets and produce flow time-series the river section nodes.

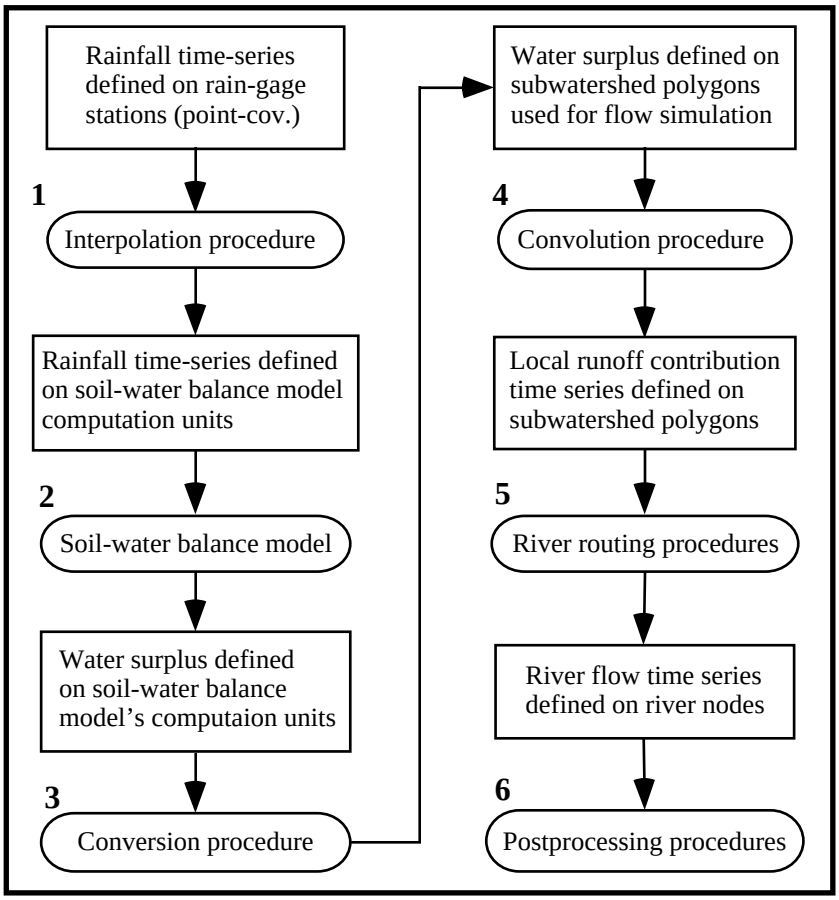


Figure 2.4. Data flow path in the map-based surface water flow simulation model

The input data for this map-based surface water flow simulation model are a set of rainfall time-series defined on the rain-gauge stations on the study area.

These rainfall time-series are interpolated to each of the computation units of a soil-water balance model (procedure 1). The interpolated rainfall time-series are used by the soil-water balance model (procedure 2) to produce soil-water surplus time-series defined on the computational units of the soil-water balance model. If subwatershed polygons are used as the computational units of the soil-water balance model, the water-surplus time-series can be used directly by a convolution procedure (procedure 4) to produce the local runoff contributions. Otherwise, a conversion procedure (procedure 3) has to be applied to convert the water-surplus from a set of time-series defined on the soil-water balance units to another set of time-series defined on the subwatershed polygons before the convolution procedure can be applied. The convolution procedure produces a flow time-series representing the runoff contribution of each subwatershed. The river routing procedures (procedure 5) together with the river network analysis procedure (to be discussed in Chapter Three) are then applied to generate flow time-series defined at the starting and ending points of each river line section in the river network.

In the following sections, the equations governing the soil-water balance computation, water surplus to runoff conversion, river flow routing, and the methods for converting time-series data between different spatial features are discussed.

#### *2.4.1.1. The Soil -Water Balance Model*

A soil-water balance model estimates the soil-water surplus given a precipitation time-series, soil-water holding capacity information, and potential evaporation information. The surplus is defined as water which does not evaporate or remain in soil storage and is available to generate surface and subsurface runoff. Surplus can be estimated using a simple bucket model

(Thornthwaite, 1948, Willmott *et al.*, 1985, Mintz and Serafini, 1993). In the simple bucket model, the basic equations for calculating surplus are:

$$\frac{w(t)}{\Delta t} = \frac{w(t-1)}{\Delta t} + P(t) - E(t) \qquad (2.1a)$$

$$S(t) = \frac{(w(t) - w^*)}{\Delta t};\ w(t) = w^* \quad \text{if } w(t) > w^*$$
$$S(t) = 0;\ w(t) = w(t) \quad \text{if } w(t) \le w^* \qquad (2.1b)$$

where,

$S(t)$ = surplus [$LT^{-1}$],

$P(t)$ = precipitation [$LT^{-1}$],

$E(t)$ = evaporation [$LT^{-1}$],

$w(t)$ = soil moisture storage of the computation unit at time step t [L],

$w^*$ = soil-water holding capacity [L],

$\Delta t$ = computation time step [T].

- **Constructing a Precipitation Surface From Rainfall Data**

The precipitation data are usually available in the form of time-series data associated with the locations of rain-gauge stations. These rainfall time-series need to be spatially interpolated to the cells on which equation 2.1 will be applied. There are many algorithms available to perform spatial interpolation, such as the methods of triangulated irregular network (TIN), Kriging, Thiessen polygons, two-dimensional spline, and inverse-distance weighting. Procedures for applying these interpolation methods can be found in numerous publications, e.g. the series

of ARC/INFO User's Guide, (ESRI, 1992). When the method of TIN is used for the interpolation, a TIN is first constructed from the point coverage of rain-gauge stations. The ARC/INFO function TINLATTICE can then be used to interpolate the rainfall values to the centers of soil-water balance computation units.

- **Computing the Evaporation**

Three types of equations are available for potential evaporation estimations (Applied Hydrology, pp82-86) and they are listed below.

(1) Energy method:

$$E_r = \frac{R_n}{l_v \cdot \rho_w} \qquad (2.2a)$$

where,

$E_r$ = the estimated evaporation rate[$LT^{-1}$],

$R_n$ = net radiation flux {200 $W/M^2$}={200 $J/SM^2$},

$l_v$ = latent heat of water vaporization{2441 KJ/Kg},

$\rho_w$ = water density{997 $Kg/M^3$}.

The numbers listed in {} are used to provide a sense of the parameter's normal value range.

(2) Aerodynamic method:

$$E_a = B(e_{as} - e) \qquad (2.2b)$$

where,

$E_a$ = the estimated evaporation rate[mm],

$e_{as}$ = vapor pressure at water surface {3167 Pa at 25ºC},

$e$ = vapor pressure of the air,

$$B = \frac{0.622k^2 \rho_a u_2}{p\rho_w [2\ln(z_2 / z_o)]}$$

k = Von Karman's constant, k = 0.4,

$\rho_a$ = air density, { $\rho_a$ = 1.19 $kg / m^3$ at 25ºC},

p = ambient air pressure, {p = 101.3 kPa at25ºC},

$u_2$ = air velocity at elevation $Z_2$,

$Z_0$ = reference height of boundary.

(3) Combined aerodynamic and energy method:

$$E = \frac{\Delta}{\Delta + \gamma} E_r + \frac{\gamma}{\Delta + \gamma} E_a \qquad (2.3)$$

where,

$\Delta = \frac{de_s}{dT} = \frac{4098 e_s}{(237.3 + T)^2}$ = vapor pressure gradient with temperature,

$\gamma$ = psychometric constant.

In this research, the energy method (Equation 2.2a) is used to estimate the potential evaporation in the simple bucket model.

- **Setting the Model's Initial Conditions**

As can be seen from Equation 2.1, computation of soil-moisture surplus is an iterative procedure, and the initial soil moisture storage w(t=0) is needed

before the computation can start. Since the initial soil moisture storage is typically unknown, the following water balancing procedure is applied to force the net change in soil moisture from the beginning to the end of a specified balancing period to zero, i.e., $w(0) - w(n+1) < \xi$, where n is the number of time steps of the computation period, and $\xi$ is a user specified tolerance ($\xi$ = 0.1 mm is used in the research). Starting with the initial soil moisture being set to the water-holding capacity, budget calculations are made to until t=n+1. w(0) is then set to w(n+1) to start another budget calculation circle until the condition $w(0) - w(n+1) < \xi$ is satisfied.

#### *2.4.1.2. Converting Time-Series between Different Spatial Features*

The soil-water balance model produces a time-series of water surplus defined on the model's computation units. Because the units used for the soil-water balance usually are not the subwatershed polygons used for surface water flow simulation, the time-series of water surplus values needs to be converted so that the values are defined on the subwatersheds. This section describes the procedure for the conversion of a data set defined on one type of spatial features to those defined on another set of spatial features.

To illustrate the procedure, assume P is a set of data defined on In-Coverage and is to be converted so that it is defined on Out-Coverage. The first step of the data converting procedure is to use the INTERSECT function provided by the ARC/INFO to establish the spatial relationships between In-Coverage and Out-Coverage. The INTERSECT operation produces a new Intersect-Coverage. As shown in Figure 2.5, nine components of P on the In-Coverage will become four components defined on Out-Coverage after the conversion. Assume the area on each feature on the In-Coverage to be $A_1$, $A_2$, ..$A_9$, and the areas of map units

on the Intersect-Coverage to be $I_{ij}$, with i representing the In-Coverage ID and j representing the Out-Coverage ID. Let OP and IP represent the components of P defined on the Out-Coverage and defined on the In-Coverage, respectively. The equations used for $OP_1$ can then be written as:

$$OP_1 = \frac{IP_1 \cdot I_{11} + IP_2 \cdot I_{21} + IP_4 \cdot I_{41} + IP_5 \cdot I_{51}}{I_{11} + I_{21} + I_{41} + I_{51}} \qquad (2.4a)$$

if P is an intensive property, and

$$OP_1 = IP_1 \cdot \frac{I_{11}}{A_1} + IP_2 \cdot \frac{I_{21}}{A_2} + IP_4 \cdot \frac{I_{42}}{A_4} + IP_5 \frac{I_{51}}{A_5} \qquad (2.4b)$$

if P is an extensive property.

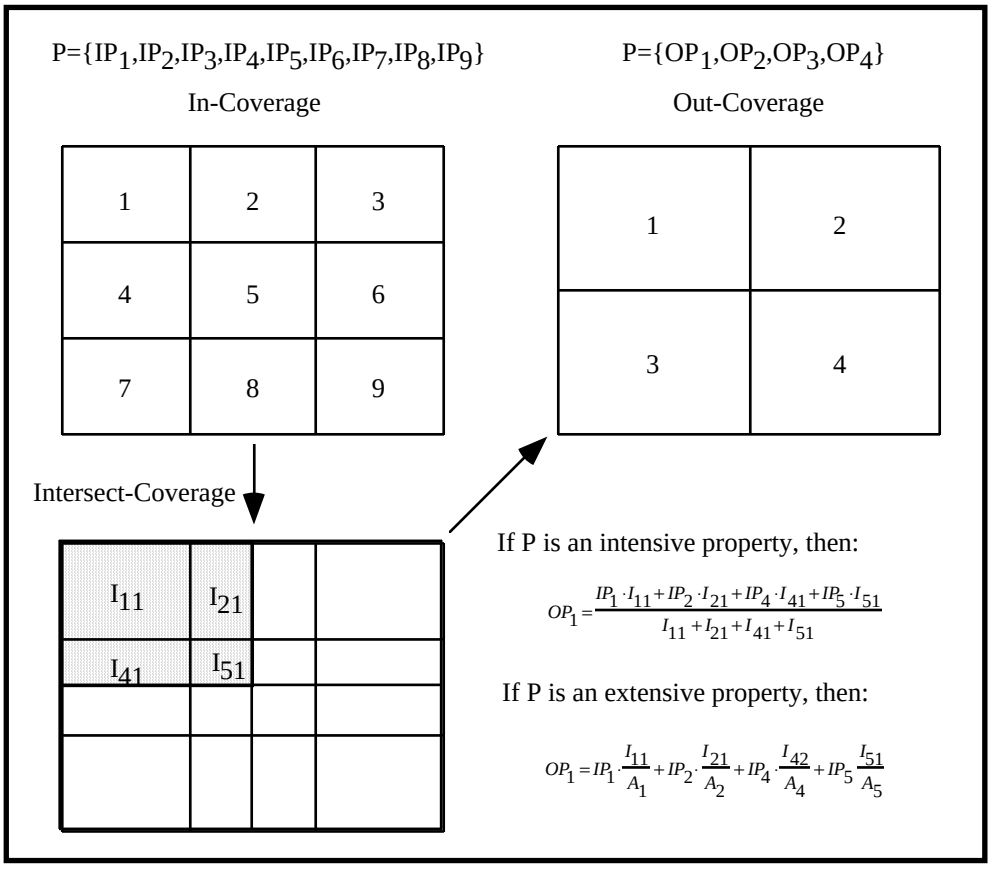


Figure 2.5. Converting data sets between different spatial features

In general, the conversion equations can be written as:

$$OP_j = \sum_i IP_i \cdot I_{ij} \tag{2.5a}$$

if P is an intensive property, and

$$OP_j = \sum_i IP_i \cdot \frac{I_{ij}}{A_i} \tag{2.5b}$$

if P is an extensive property, where,

$OP_j$ = property P defined on unit j at the Out-Coverage,

$IP_i$ = property P defined on unit i at the In-Coverage,

$I_{ij}$ = the area of unit i on the In-Coverage that intersects with unit j on the Out-Coverage,

$A_i$ = the area of unit i on the In-Coverage. To convert time-series data, equation 2.5 needs to be applied to the data at each time step.

#### *2.4.1.3. Convolution Procedure Used To Compute Local Runoff*

The time-series representing the local runoff of a subwatershed to the river network (PFlow(t)) can be calculated from the time-series of water-surplus (SurpF(t)) defined on the subwatershed. In the following text, when referring to a time-series in general, for example, PFLOW, the notation PFlow(t) will be used and when referring the same time-series related to a specific spatial feature, the notation $PFlow_i^t$ will be used; a subscript (i) indicates the spatial feature index and a superscript (t) indicates the time index. Because the water surplus can reach a river section through either overland flow or through subsurface flow, the portion of surplus flow that reaches a river section through overland flow will be referred to as SFlow(t) and the portion that goes into the subsurface before it reaches the river section will be termed as OFlow(t) (Figure 2.6). Based on this assumption, we have:

$$PFlow(t) = SFlow(t) + OFlow(t) \qquad (2.6)$$

The overland flow portion (SFlow(t)) can be computed from SurpF(t) using equation (Olivera and Maidment, 1996):

$$SFlow_i^t = \sum_{k=0}^{\min(t,N)} SurpF_i^{t-k}(1-\alpha_i)\cdot U_i^k \qquad (2.7)$$

where,

$SFlow_i^t$ = local surface water flow contribution ($m^3/s$), of subwatershed i at time step t,

$SurpF_i^{t-k}$ = soil moisture surplus ($m^3/s$) of subwatershed i at time step t-k,

$U_i^k$ = k-th component of the response function of $PFlow_i^t$ on $SurpF_i^t$,

$\alpha_i$ = the fraction of surplus that goes to subsurface, $(0 \le \alpha_i \le 1)$,

N = total number of components in the response function $U_i^k$. The response function of $PFlow_i^t$ on $SurpF_i^t$ used in this study is given below:

$$U_i^k = \frac{1}{2k\sqrt{\pi D_i \frac{kv_i}{T_i}}} \exp\left(-\frac{(1-\frac{kv_i}{T_i})^2}{4D_i\frac{kv_i}{T_i}}\right) \quad k=1,2,3....N \qquad (2.8)$$

where,

k =1,2,3...N, the index of components in the response function,

$D_i$ = dispersion coefficient for subwatershed i, Dispersion coefficient is used to measure the degree of the spreading of overland water flow over time.

$V_i$ = average overland flow velocity for subwatershed i (m/s),

$T_i$ = average overland flow time for subwatershed i (s).

Figure 2.6 is constructed to illustrate how the parameters of Equation 2.8 can be estimated. In Figure 2.6, subwatershed i is composed of a number of cells (elements) and for a given element e, its flow length $l_e$ can be calculated using the GRID module in ARC/INFO. The flow time of water from element e to the outlet of the subwatershed can be estimated by dividing the flow length $l_e$ by the average flow velocity $v_e$, which could estimated from the topology and land cover information of the subwatershed. The average overland flow time for subwatershed $P_i$ is then computed using:

$$T_i = \sum_{e=1}^{N_e} t_e \cdot \frac{A_e}{A_i} \qquad (2.9)$$

where, $A_e$ and $A_i$ are the areas of element e and subwatershed i, respectively.

When all the elements forming the subwatershed have the same size, which is the case when the GRID module is used, Equation 2.9 becomes:

$$T_i = \frac{1}{N_e} \sum_{e=1}^{N_e} t_e \qquad (2.9a)$$

where, $N_e$ = the number of elements in the subwatershed.

Using the Zonalstats function provided by the GRID module in ARC/INFO, the average overland flow length $l_i$ and standard deviation $\sigma_i$ of the flow length for subwatershed $P_i$ can also be calculated. With these two parameters, the dispersion coefficient for the subwatershed $P_i$ can be computed using:

$$D_i = \frac{\sigma_i^2}{2(l_i^2)} \quad (2.10)$$

where,

$\sigma_i$ = the standard deviation of the flow length for subwatershed $P_i$,

$l_i$ = the average overland flow length for subwatershed $P_i$.

The subsurface water flow component of PFlow(t) is considered to be going through an imaginary under-ground-reservoir whose flow can be simulated using a linear-reservoir-model (Equations 2.11 and 2.12):

$$OFlow_i^t = S_i^{t-1} / K_i \qquad (t = 1, 2, 3, ....) \quad (2.11)$$

$$S_i^t = S_i^{t-1} + (SurpF_i^t \cdot \alpha_i - OFlow_i^t) \cdot \Delta t \quad (2.12)$$

where,

$OFlow_i^t$ = PFlow's subsurface component at time step t, on polygon i $[L^3T^{-1}]$,

$S_i^t$ = storage of the underground reservoir at time step t, on polygon i $[L^3]$,

$K_i$ = the linear reservoir constant [T].

After the components simulating surface and subsurface water flows are computed, the local flow contribution of subwatershed i at time step t is computed using Equation 2.6.

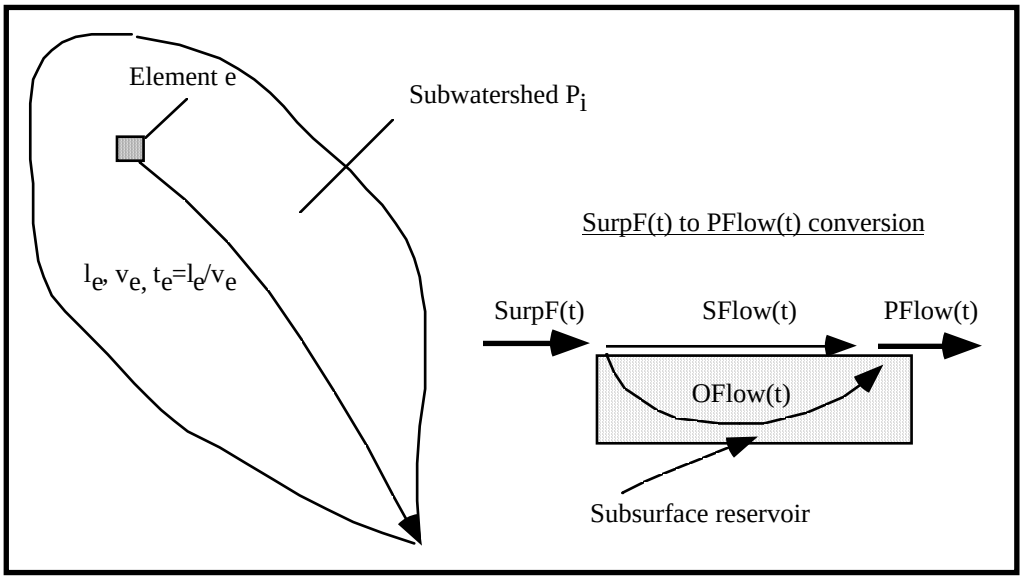


Figure 2.6. Converting SurpF(t) to PFlow(t)

### *2.4.1.4. Flow Routing on a River Section*

The flow in a river section shown in Figure 2.7 can be simulated using the Muskingum or Muskingum-Cunge method (McCarthy, 1938, Cunge,1969, Chow *et al.*, 1987). The Muskingum method is based on the principle of continuity and a relationship between discharge and the temporary storage of excess volumes of water in a river section during the simulation period. The principle can be expressed as:

$$\frac{dS}{dt} = I(t) - Q(t) \qquad (2.13)$$

where,

$S$ = the volume of water in storage in a river section,

*I(t)* = water inflow time-series (hydrograph) of the river section [$L^3T^{-1}$],

*Q(t)* = water outflow time-series of the river section [$L^3T^{-1}$].

In deriving the flow routing formula for the Muskingum method, it is assumed that the storage volume in a river section (Figure 2.7) is composed of two portions: a wedge storage and a prism storage. It is further assumed that the cross-sectional area of the water flow is directly proportional to discharge into the section, the volume of prism storage is K•TFlow(t) and the volume of wedge storage is K•X•(FFlow(t)-TFlow(t)), where K is a proportionality coefficient and X is a weighting factor showing the relative importance of FFlow(t) and TFlow(t). With these assumptions, the total storage of the section can be written as:

$$S(t) = K \cdot TFlow(t) + K \cdot X \cdot (FFlow(t) - TFlow(t)) \qquad (2.14)$$

The formula of Muskingum routing method is derived by expressing the storage change of the section between time step t and t-1 in terms of FFlow(t) and TFlow(t) using Equation 2.14. Muskingum-Cunge method is derived based on the Muskingum method taking into consideration the lateral flow (PFlow(t)). Detailed descriptions of Muskingum-Cunge flow routing method can be found in Applied Hydrology (Chow *et al.*, 1987), Hydrology for Engineers (Linsley *et al*, 1982), and Handbook of Hydrology (Maidment, 1993).

The Muskingum-Cunge flow routing method is described by the following equation:

$$\begin{aligned} TFlow(t) = {} & C_1 \cdot FFlow(t) + C_2 \cdot FFlow(t-1) \\ & + C_3 \cdot TFlow(t-1) + C_4 \end{aligned} \qquad (2.15)$$

where,

TFlow(t) = flow time-series at the To-Node of a river line,

FFlow(t) = flow time-series at the From-Node of a river line,

C1, C2, C3, C4 = coefficients related to river and flow characteristics. These coefficients are computed using equations given below:

$$C_1 = \frac{\Delta t - 2KX}{2K(1-X) + \Delta t} \quad (2.16a)$$

$$C_2 = \frac{\Delta t + 2KX}{2K(1-X) + \Delta t} \quad (2.16b)$$

$$C_3 = \frac{2K(1-X) - \Delta t}{2K(1-X) + \Delta t} \quad (2.16c)$$

$$C_4 = \frac{PFlow_i^t - DFlow_i^t - Loss_i^t}{2K(1-X) + \Delta t} \quad (2.16d)$$

$K = \frac{\Delta x}{\bar{c}}$, K is a storage constant [T], (2.16e)

$$X = \frac{1}{2} - \frac{Avg(TFlow, FFlow)}{2\bar{c}\bar{B}S_e\Delta X} \quad (2.16f)$$

X = a weighting factor showing the relative importance that FFlow and TFlow have on the river section's storage,

$\Delta x$ = the length of the river section, [L]

$\bar{c}$ = kinematic wave velocity [$LT^{-1}$],

$\bar{B}$ = cross-sectional top width associated with average of TFlow and FFlow,

$S_e$ = the energy slope, and $\Delta X$ = length of a river section.

To ensure the stability of the flow routing, $C_3$ needs to be non-negative. From equation 2.16c, it can be seen that in order for $C_3 \geq 0$, we need to have:

$\Delta t \le 2K(1-X)$. The method used to ensure that the time step $\Delta t$ satisfies the non-equality relationship will be discussed in section 3.4.2.

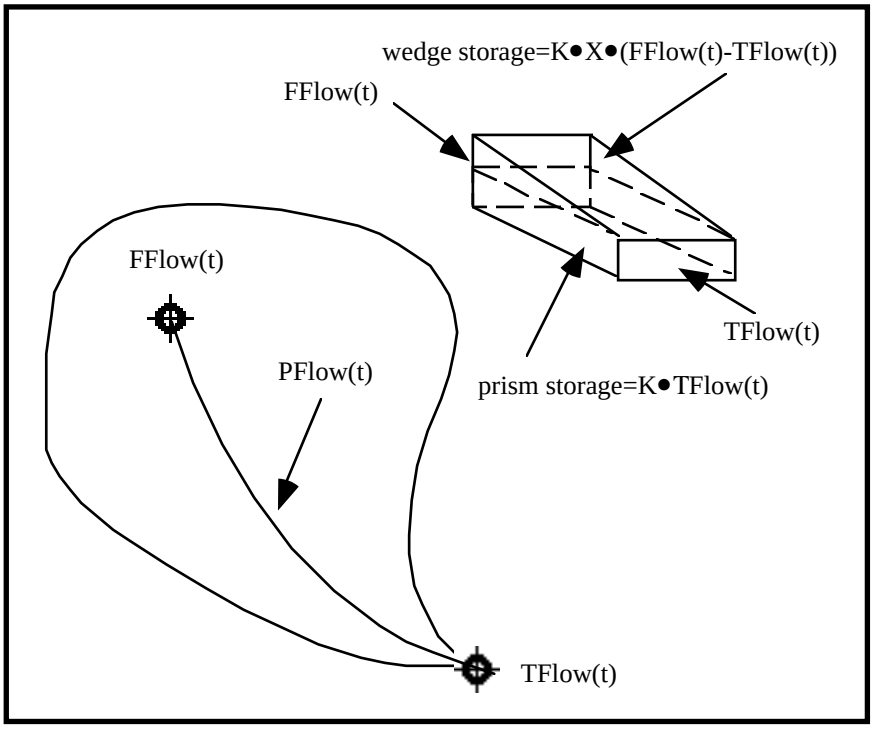


Figure 2.7. Flow routing on a river section

### 2.4.2. Equations Used for Groundwater Flow Simulation

The continuity equation for groundwater flow in three dimensions can be written as (Bear, 1979):

$$-\left(\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}\right)+N(x,y,z,t)=S\frac{\partial h}{\partial t} \qquad (2.17)$$

where,

h = piezometric head of the aquifer [L],

N(x,y,z,t) = a point source (or point sink when N is negative) [$T^{-1}$], at point (x,y,z),

$S = \bar{\rho} g(\alpha + \beta)$ = specific storage [$L^{-1}$],

$\alpha = \frac{1}{1-n}\frac{dn}{dp}$ = the matrix compressibility [$M^{-1}LT^2$],

$\beta = \frac{1}{\rho}\frac{d\rho}{dp}$ = the water compressibility [$M^{-1}LT^2$],

n = the porosity of the aquifer,

$q_x$, $q_y$, $q_z$ = the components of the specific discharge vector $\vec{q}$ [$LT^{-1}$] in x, y, z directions. $\vec{q}$ can be computed using Darcy's law:

$$\vec{q} = -K\, grad\, h = -K\nabla h \qquad (2.18)$$

where,

K = the hydraulic conductivity of the aquifer [$LT^{-1}$],

$\nabla = (\frac{\partial}{\partial x}\vec{i} + \frac{\partial}{\partial y}\vec{j} + \frac{\partial}{\partial z}\vec{k})$ is gradient operator.

The continuity equation for groundwater flow in a phreatic aquifer in two dimensions (Figure 2.8) can be written as:

$$-(\frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y}) + R - P = S\frac{\partial h}{\partial t} \qquad (2.19a)$$

or

$$\frac{\partial}{\partial x}(K_x h\frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(K_y h\frac{\partial h}{\partial y}) + R - P = S\frac{\partial h}{\partial t} \qquad (2.19b)$$

where,

$K_x$ and $K_y$ are the hydraulic conductivity in x and y directions,

$Q_x = -K_x h\frac{\partial h}{\partial x}$ = the discharge per unit width of the aquifer in x direction $[L^2T^{-1}]$,

$Q_y = -K_y h\frac{\partial h}{\partial y}$ = the discharge per unit width of the aquifer in x direction $[L^2T^{-1}]$,

R = R(x,y,t) = recharge to the aquifer $[LT^{-1}]$,

P = P(x,y,t) = pumpage from the aquifer $[LT^{-1}]$

S = specific storage.

Equations 2.19a and 2.19b can be derived by integrating Equation 2.17 over the Z dimension while taking into consideration (1) the Dupuit horizontal flow assumption, (2) Leibnitz rule and (3) the boundary condition at the phreatic surface. Detailed derivation procedure can be found in the book Hydraulics of Groundwater (Bear, 1979). A brief description of Leibnitz rule and the boundary condition at the phreatic surface is given here.

For an aquifer of thickness $B = Z_2(x,y,t) - Z_1(x,y,t)$ and given a scalar $h(x,y,z,t)$ defined on the aquifer, according to Leibnitz rule, we have:

$$\frac{\partial}{\partial t}\int_{Z_1(x,y,t)}^{Z_2(x,y,t)} h dz = \frac{\partial}{\partial t}(B\bar{h}) = \int_{Z_1}^{Z_2}\frac{\partial h}{\partial t}dz + h\Big|_{Z_2}\frac{\partial Z_2}{\partial t} - h\Big|_{Z_1}\frac{\partial Z_1}{\partial t} \qquad (2.20)$$

where, $\bar{h} = \frac{1}{B}\int_{Z_1}^{Z_2} h dz$.

Assuming a phreatic surface with accretion (Figure 2.8) is moving at a velocity of $V_s$, the continuity requirement yields the following equation:

$$(\vec{q} - \vec{N}) \cdot \vec{n} = n_e V_s \cdot \vec{n} \qquad (2.21)$$

where,

$n_e$ = effective porosity,

$\vec{n}$ = the normal direction of the phreatic surface,

N = the rate of accretion [$LT^{-1}$].

Equation 2.21 states that the phreatic surface should move at a velocity such that the rate of water storage variation under the surface equals the rate of water exchange across the surface.

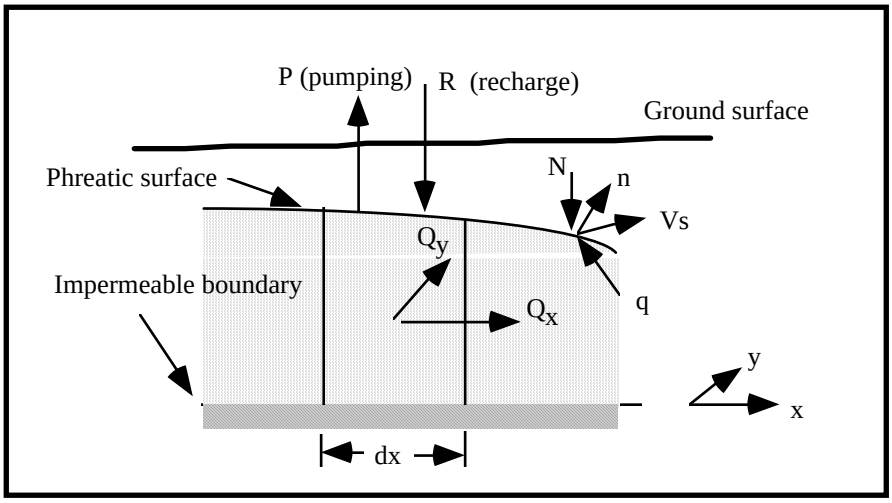


Figure 2.8. Water flow in a phreatic aquifer

The continuity equation for water flow in a confined, inhomogeneous anisotropic aquifer in two dimensions can be written as:

$$\frac{\partial}{\partial x}(T_x \frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(T_y \frac{\partial h}{\partial y}) + R - P = S_S \frac{\partial h}{\partial t} \qquad (2.22)$$

where,

h = h(x,y,t) = piezometric head of the aquifer at point x,y [L],

$T_x = T_x(x,y)$ = aquifer transimisivity in x direction [$L^2T^{-1}$],

$T_y = T_y(x,y)$ = aquifer transimisivity in y direction [$L^2T^{-1}$],

$S_S$ = aquifer storativity. If the aquifer is inhomogeneous and isotropic, we have $T_x = T_y = T(x,y)$. If the aquifer is homogeneous and isotropic, we have $T_x = T_y = T$.

Equation 2.22 can also be derived by integrating equation 2.17 over the aquifer thickness while taking into consideration of Leibnitz rule and aquifer's top and bottom boundary conditions.

To solve Equation 2.19 (or Equation 2.22) numerically, the first step is to discretize the region of interest, i.e. to replace the continuous region for which a solution is desired by an array of points. These points are usually the center points of grid cells or corner points of a grid. Then either a finite element or a finite difference method is applied to these points to convert the differential equation into a set of linear equations. This set of linear equations are then solved by some appropriate solver. Detailed discussions on the subject of solving the equations using finite element and finite difference methods can be found in the textbooks by Becker *et al*. (1981), Remson *et al*. (1971), Desai (1979), Wang and Anderson (1982), Huyakorn and Pinder (1983), and numerous articles, e.g., Pinder and Gray (1977).

Because solving the groundwater flow equation in form of the equation 2.19 or 2.22 is complicated and computational intensive, models based on this

type of equations such as Modflow (McDonald and Harbaugh, 1988) and GWSim4 (TDWR, 1974) are usually self-contained. Therefore, it is difficult to fully integrate this type of groundwater model with a geographic information system (GIS).

To avoid this difficulty, a map-based groundwater simulation model will be constructed. The map-based model is constructed on a polygon and the polygon's boundary line coverages. This model simulates groundwater flow by alternatively applying the continuity equation to the polygon objects and the momentum equation to the polygon boundary line objects. This section describes the concept of the map-based groundwater simulation model. A detailed model constructing and programming procedure are discussed in Chapter Four.

Figure 2.9 illustrates the concept of the map-based groundwater simulation model. The continuity equation (discretized in time) derived from Equation 2.19 or Equation 2.22 for a polygon object in Figure 2.9 can be written as:

$$\Delta t^t \cdot \left[A_i \cdot \left(R_i^t - P_i^t - Q_i^t\right)\right] + \sum_j V_{ij}^t = A_i \cdot S_i \cdot (h_i^t - h_i^{t-1}) \qquad (2.23)$$

where,

$\Delta t^t$ = time interval at time step t,

$A_i$ = area of cell i,

$R_i^t$, $P_i^t$, and $Q_i^t$ = recharge, pumpage, and discharge of the aquifer under cell i at time step t, respectively, [$LT^{-1}$],

$V_{ij}^t$ = volume of water that enters cell i through boundary j at time step t [$L^3$], The computation of $V_{ij}^t$ is based on the momentum equation

(Darcy's Law) and a line integration technique to be discussed in Chapter Four.

$S_i$ = the storativity (for a confined aquifer) or the specific storage (for a phreatic aquifer, of cell i [$L^0T^0$],

$h_i^t$ = water level of cell i at the end of time step t [L],

$h_i^{t-1}$ = water level of cell i at the end of time step t-1 [L].

Also in Figure 2.9, the momentum equation in the form of Darcy's law can be applied to each boundary line of the polygon objects and for a given boundary line object, the momentum equation can be written as:

$$\vec{q}_{ij}^{t} = -k_{ij}\frac{dh}{ds}\vec{s} \approx -k_{ij}\frac{h_j^{t-1} - h_i^{t-1}}{\Delta s_{ij}}\vec{s} \qquad (2.24)$$

where,

$k_{ij}$ = hydraulic conductivity between polygons i and j [$LT^{-1}$],

$\vec{q}_{ij}^{t}$ = flux of water [$LT^{-1}$] across the boundary line between polygons i & j at time step t,

$\vec{s}$ = a unit vector pointing from the center of polygon i to that of polygon j,

$\Delta s_{ij}$ = distance between the centers of polygons i and j [L],

$h_j^{t-1}$, $h_i^{t-1}$ = water levels of polygons j and i at time step t-1 [L].

The logic of the map-based model is simple. Given the initial head levels of the polygons, Equation 2.24 is applied to each boundary line object to compute groundwater flow across each boundary line object. As a result of this computation, the water volumes ($V_{ij}^t$) transported in and out of each polygon

object in the first time step are obtained. With $V_{ij}^{t}$ known, the continuity equation (Equation 2.23) is then applied to each polygon object to calculate the water level at the end of first time step. This procedure of alternatively applying the momentum equation to line objects and the continuity equation to polygon objects will be repeated until the end of the simulation period.

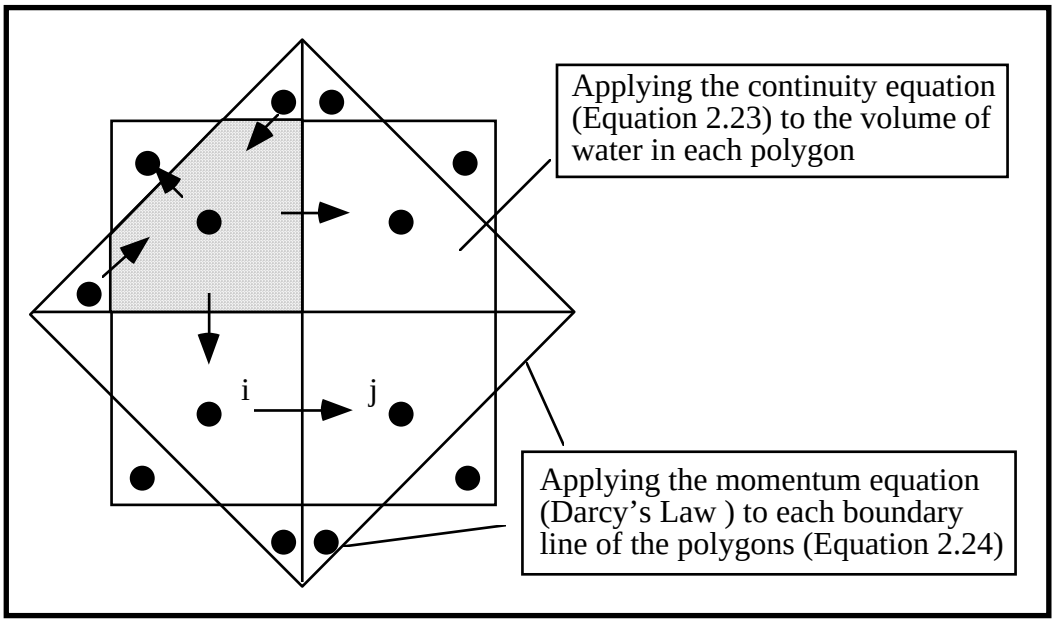


Figure 2.9. The conceptual design of a map-based groundwater model

## 2.5. CHAPTER SUMMARY

As it can be seen from this section, the surface water flow processes (overland flow, river flow etc.) can be simulated one region at a time when the water exchange between subwatersheds can be represented by a set of time-series data. For this reason, surface water flow processes for a subwatershed can be simulated by making a sequence of functional calls in a certain order.

Also, it can be seen from the equations presented above that the data supporting these equations can be categorized into two types, static and dynamic. Static data do not change throughout the whole simulation period, while dynamic data vary from one time step to another. The dynamic data type itself can be further divided into two types, predetermined and run-time determined. Predetermined dynamic data, such as rainfall time-series, are known before running the model. Run-time determined dynamic data, such as the groundwater flow velocity field used to compute Courant number ( $Co = \frac{V \cdot \Delta t}{\Delta x}$ ) for a given time step, are not known until the simulation of previous time steps is completed. The reason for making this data classification is that different types of data may require different data structures for efficient data storage and retrieval. Detailed discussions about the treatments of different data types are presented in Chapter Three.

The map-based groundwater simulation model is constructed by applying the finite difference form of the continuity equation (Equation 2.23) to the water volumes of polygon objects and the finite difference form of the momentum equation (Equation 2.24) to the polygon boundary line objects. Because the spatial features are grouped into line and polygon coverages in ARC/INFO, separately applying these two equations to these two types of objects greatly simplifies the solution procedures.