CASC2D-SED v 1.0

Reference Manual

A 2-Dimensional Rainfall-Runoff

and Sediment Model

R. Rojas / P. Julien / B. Johnson

Colorado State University

July 2003

TABLE OF CONTENTS

LIST OF FIGURES

Contents   **Page**

LIST OF TABLES

Contents   **Page**

1. MODEL DEVELOPMENT

The CASCade 2 Dimensional SEDiment (CASC2D-SED) model originally began with a two-dimensional overland flow routing algorithm developed and written in APL by Prof. P.Y. Julien at Colorado State University. The overland flow routing module was converted from APL to FORTRAN by Saghafian, then at Colorado State University, with the addition of Green & Ampt infiltration, detention storage and diffusive-wave channel routing (Julien and Saghafian 1991; Saghafian 1992; Julien et al. 1995).

CASC2D was incorporated as part of the GRASS GIS for hydrologic simulations as r.hydro.CASC2D (Saghafian 1993; Saghafian and Ogden 1996). Later on, Ogden (1994) added the implicit channel routing option to CASC2D. Since 1995, CASC2D has been reformulated with the addition of continuous simulation capabilities and other hydrological components such as interception, initial depths, evapotranspiration and redistribution (Ogden 1997b). This last version is incorporated in the Watershed Modeling System (WMS) Interface developed by Brigham Young University (2001) and is known as CASC2D for WMS.

The upland erosion and channel sediment transport module was added by Johnson (1997) based in previous work done by Kilinc (1972) and Kilinc and Richardson (1973) at Colorado State University and was called CASC2D-SED. The sediment transport algorithms developed by Rojas (2002) improve over the version of Johnson et al. (2000) in the following manner: (1) improves simulation of the transition between supply-

limited and capacity-limited sediment transport; (2) allows transport by advection of suspended material even when the transport capacity is negligible; and (3) improves simulation of sedimentation in backwater areas.

2. GOVERNING EQUATIONS

CASC2D has been developed to determine the runoff hydrograph generated from any temporally-spatially varied rainfall event. When using the erosion/sedimentation module of CASC2D, sediment rates can be predicted at any location as well.

For a given rainfall event, once the initial losses have been subtracted from rainfall, water begins to infiltrate. This step requires the adoption of an infiltration scheme that can predict the portion of the rainfall that drains into the ground. The Green & Ampt (1911) infiltration equation accommodates spatial and temporal variabilities due to changes in the rainfall and/or soils properties, and takes into account the accumulated infiltration.

Using a Hortonian overland flow process, when the precipitation rate exceeds the infiltration rate, the excess rainfall will accumulate as surface water and begin to flow. In CASC2D, overland flow is routed into the channels using a diffusive wave approximation in two dimensions. In channels, the water is routed using a 1-D diffusive wave equation.

The erosion/sedimentation rates are calculated as a function of the hydraulic properties of the flow, the physical properties of the soil and the surface characteristics. The modified Kilinc-Richardson equation (Julien 1995) is used in CASC2D-SED to determine the upland sediment transport by grain size from one cell into the next one in

two orthogonal directions. The sediment coming from upland erosion is routed through the channel network into the outlet.

CASC2D solves the equations of conservation of mass, energy and linear momentum to estimate watershed runoff for a given rainfall input. The overland flow routing formulation is based on an explicit 2-dimensional finite difference (FD) technique. An explicit FD technique is used for the routing of 1-dimensional channel flow.

In the next section, the governing equations in the overland flow and channel routing schemes are described. The equations used in estimating precipitation and infiltration rates are also included. Finally, the processes of upland and channel erosion and deposition are described.
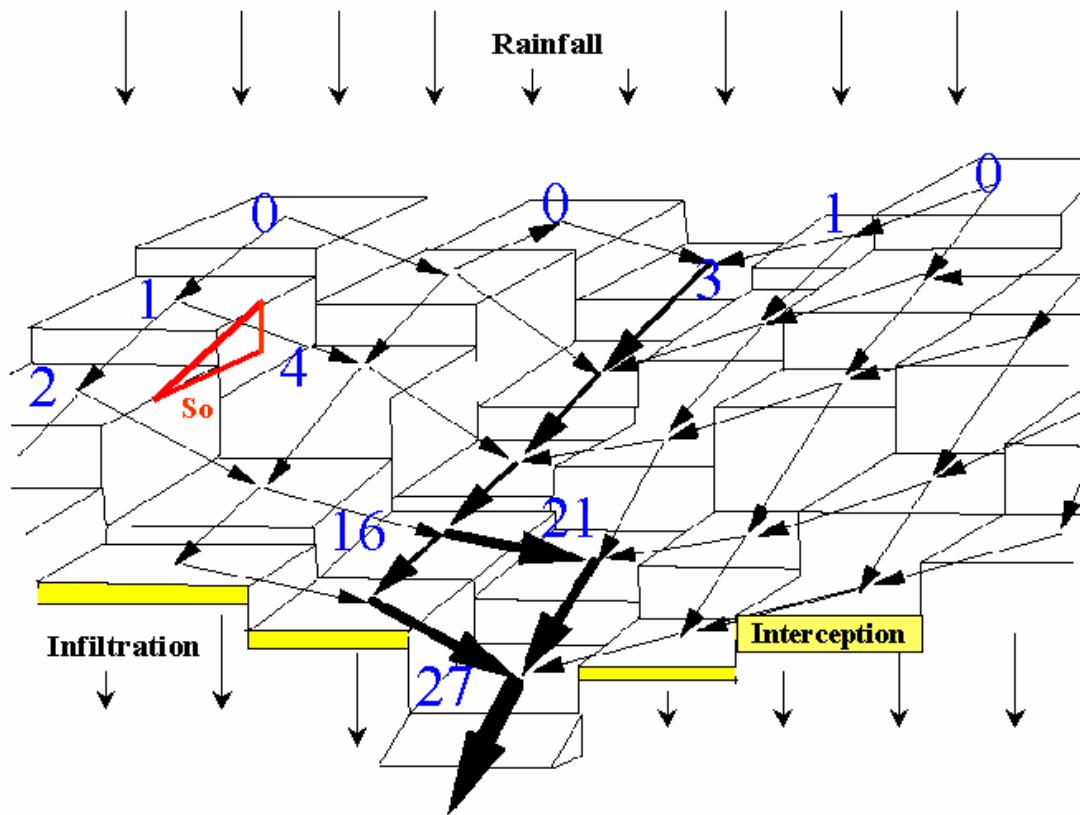

Figure 2-1 Topographical representation of overland flow and channel routing schemes
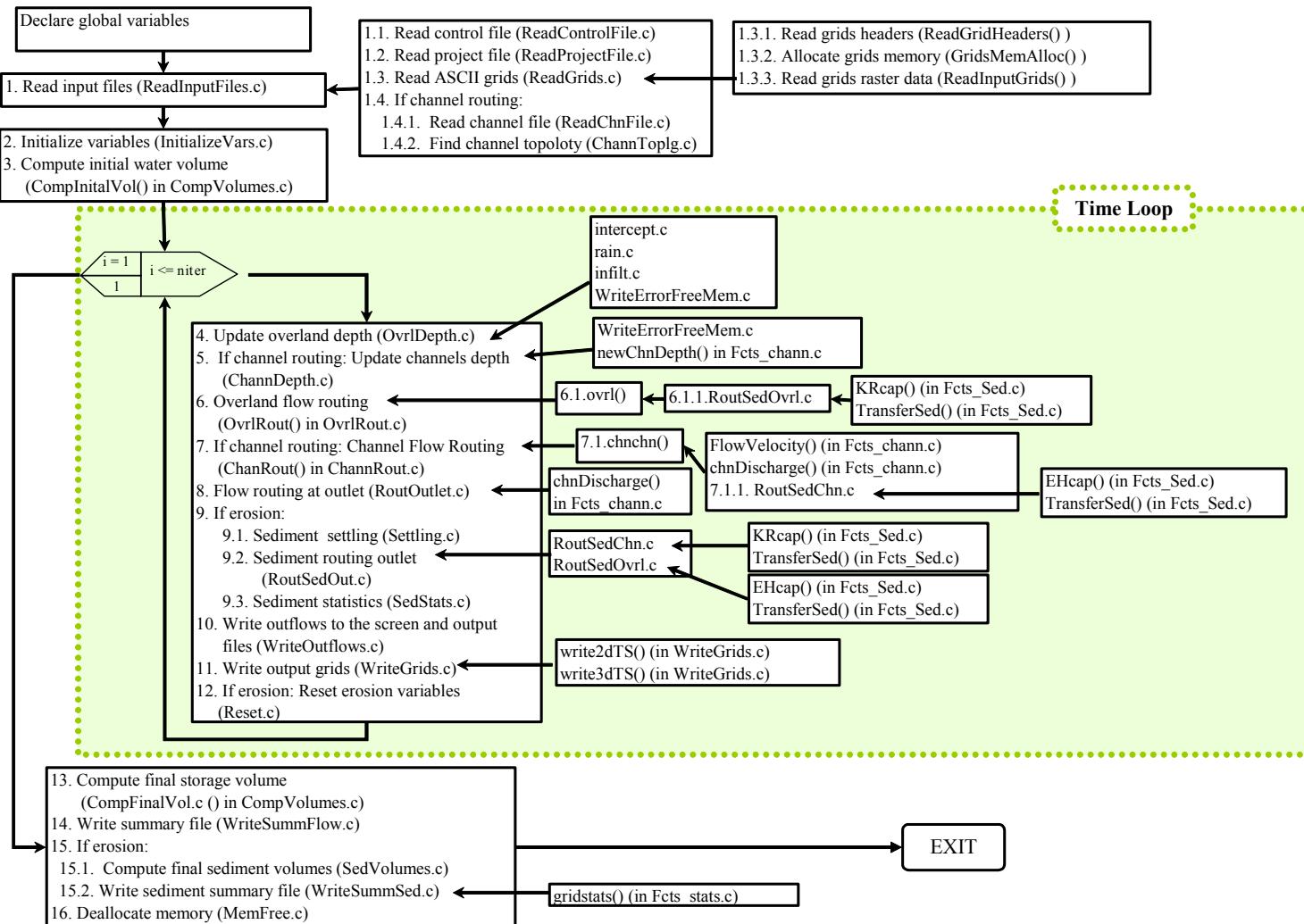
Figure 2-2 CASC2D-SED routines flow chart

2.1   FLOW ROUTING COMPONENTS

2.1.1   Precipitation

The rainfall intensity might be assumed to be uniform. In this case, the rainfall intensity and duration are needed as input to the program. When more than one rain gage is used in the estimation of the precipitation, an interpolation scheme based on the inverse distance squared approximates the distribution of rainfall intensity over the watershed:

$$i^t(j,k) = \frac{\sum_{m=1}^{NRG} \dfrac{i_m^t(jrg,krg)}{d_m^2}}{\sum_{m=1}^{NRG} \dfrac{1}{d_m^2}} \qquad\qquad [1]$$

where,
  $i^t(j,k)$ =   rainfall intensity in element (j,k) at time t
  $i_m^t(jrg,krg)$ = rainfall intensity recorded by the m-th rainfall gage located at (jrg,krg)
  d_m =   distance from element (j,k) to m-th rain gage located at (jrg,krg)
  NRG   =   total number of rain gages

2.1.2   Interception

As rain falls on a vegetated surface, part of it is held on the foliage by surface tension forces. Rather than reaching eventually the ground, this portion of the rainfall evaporates directly and does not take part in ultimate runoff and thus, it is usually termed interception loss (Eagleson 1970). Chow et al. (1988) defined retention the part of the surface storage held for a long period of time and depleted by evaporation.

Because intercepted water does not reach the soil surface, it has no part in infiltration. Accordingly, the interception depth is subtracted from the rainfall before infiltration is calculated. In CASC2D, the rainfall rate is reduced until the interception depth (I) has been satisfied. For a given grid cell inside the basin, if the total rain falling

during the first time increment (dt) is greater than I, the rainfall rate is reduced by I/dt. If the rainfall depth is less than I, the rainfall rate is set to zero and the remainder of the interception is removed from the rainfall in the following time increments. Measured values of interception depth for a number of vegetative covers are found in Woolhiser et al. (1990) and Bras (1990).

2.1.3 Infiltration

The Green & Ampt (1911) equation provides the primary relationship for infiltration within the CASC2D computer model. The Green and Ampt infiltration scheme gained considerable attention partially due to the ever growing trend of physically-based hydrological modeling (Philip 1983). To accurately account for the physical process involved with surface flow, CASC2D uses the Green and Ampt approximation for soil infiltration. Specifically, this relationship is utilized within the model's infiltration scheme to determine the depth and rate of soil infiltration as a component of the resulting overland flow.

| Water |
| Wet soil $\theta = \theta s$ |
| Wetting front |
| Dry soil $\theta = \theta i$ |
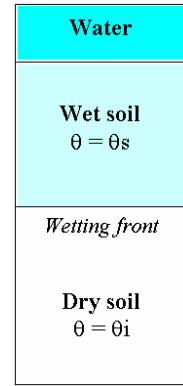
The Green-Ampt model assumes piston flow with a sharp wetting front between the infiltration zone and soil at the initial water content. The wet zone increases in length as infiltration progresses (Bras 1990).

Neglecting the level of ponding on the surface, the general equation showing the Green-Ampt relationship can be expressed as (Bras 1990).

$$f = Ks\left(1 + \frac{H_f M_d}{F}\right)$$ [2]

where:

f   =   infiltration rate
K_s =   saturated hydraulic conductivity
H_f =   capillary pressure head at the wetting front
M_d =   soil moisture deficit = $(\theta_e - \theta_i)$
                θ_e = effective porosity = $(\varphi - \theta_r)$
                    φ = total soil porosity
                    θ_r = residual saturation
                θ_i = soil initial moisture content
F   =   total infiltrated depth

To apply this calculation to the entire watershed area, four separate physical characteristics must be known and provided as input to the model. These characteristics are hydraulic conductivity, capillary pressure, effective soil porosity, and initial soil moisture content. Their numerical values can be obtained from the experimental data by Rawls et al. (1983) depending on the soil texture.

2.1.4   Overland Flow

The governing equations for overland flow with the CASC2D Model are based primarily on the de-Saint Venant Equations of continuity and momentum. Using these formulations, CASC2D was designed around an explicit finite difference, diffusive-wave method to route overland flow. The general form for these equations, as shown in Julien and Saghafian (1991), are commonly expressed in partial differential form as:

Continuity:

$$\frac{\partial h}{\partial t} + \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} = e \qquad [3]$$

Momentum:

x-direction $\dfrac{\partial u}{\partial t} + u\dfrac{\partial u}{\partial x} + v\dfrac{\partial u}{\partial y} = g(S_{ox} - S_{fx} - \dfrac{\partial h}{\partial x})$ [4]

y-direction $\qquad \dfrac{\partial v}{\partial t} + u\dfrac{\partial v}{\partial x} + v\dfrac{\partial v}{\partial y} = g(S_{oy} - S_{fy} - \dfrac{\partial h}{\partial y})$ [5]

where:

h	=	surface flow depth
q_x	=	unit flow rate in the x-direction,
q_y	=	unit flow rate in the y-direction
e	=	excess rainfall (i-f)
i	=	rainfall intensity
f	=	infiltration rate
x,y	=	Cartesian spatial coordinates
t	=	time
$S_{o(x,y)}$	=	bed slopes in the x- and y-direction, respectively
$S_{f(x,y)}$	=	friction slopes in the x- and y-direction, respectively
u,v	=	average velocities in the respective x- and y- directions
g	=	gravitational acceleration

Equations 4 and 5 show the relationship between the net forces per unit mass in each direction and the acceleration of flow in relation to that given direction. Thus, the forces along a given axis are shown on the right side of the equation, while the local and convective acceleration is given by the left-hand side of the equation. The simplified diffusive approximation for Equations 4 and 5 assumes that the net forces acting along the given axis of interest are approximately zero. Thus, the resulting diffusive wave approximation can be descried by the following equations.

$$S_{fx} = S_{ox} - \dfrac{\partial h}{\partial x}$$ [6]

$$S_{fy} = S_{oy} - \dfrac{\partial h}{\partial y}$$ [7]

The key advantage that is provided in using the diffusive form of the momentum equations is the ability to account for backwater effects observed during overland and channel flow events.

Using the three equations given for continuity and momentum, a resistance law can be established. This equation relates flow rate to depth and other given flow parameters such as surface roughness. The defined resistance law can be derived for either the x or y-directions as:

$$q_{x,y} = \alpha_{x,y} h^{\beta}$$ [8]

In this form $\alpha_{x,y}$ and β are flow regime parameters that vary depending on whether turbulent or laminar conditions exist. CASC2D assumes turbulent conditions for the entire watershed and the Manning approximation for $\alpha_{x,y}$ and β are determined to be:

$$\alpha_{x,y} = \frac{S_{f(x,y)}^{1/2}}{n}$$ [9]

$$\beta = \frac{5}{3}$$

Where n is the Manning roughness coefficient, or surface roughness. This coefficient can be estimated from the land use map using the values provided by Woolhiser (1975).

The initial and boundary conditions applied for a plane of length L are respectively:

$$h(x,0) = 0 \qquad 0 \le x \le L$$
$$h(0,t) = 0$$ [10]

2.1.5 Channel Flow

The channel routing scheme employed by CASC2D is capable of processing completely unsteady hydraulic scenarios. This is achieved by the use of a one-

dimensional diffusive channel flow equation (Julien and Saghafian 1991). The governing equations for the channel flow routing process are similar to those for overland flow, with one significant exception to note. The equations used in channel flow routing are defined by a finite width established for a given channel section. The one-dimensional continuity relationship can be expressed by the following equation (Julien and Saghafian 1991):

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = q_l \qquad [11]$$

where:
  A   = channel flow cross-sectional Area
  Q   = total channel discharge
  q_l = lateral inflow rate per unit length (into or out of the channel)

Once again, by assuming the flow within the channel is completely turbulent, the model utilizes Manning's equation to ascertain a value for channel flow equation (Julien and Saghafian 1991).

$$Q = \frac{1}{n} A R^{\frac{2}{3}} S_f^{\frac{1}{2}} \qquad [12]$$

where:
  R   = hydraulic radius
  S_f = friction slope
  n   = Manning roughness coefficient

2.1.6 **Base Flow and Surface Storage**

Base flow is the component of streamflow that can be attributed to ground-water discharge into streams. In CASC2D-SED the base flow is accounted for by defining an initial water depth in channel cells of the input base depth grid. If an initial water depth is defined in overland cells, this may represent lakes or the ponding of water from previous rainfall events.

2.2   S<small>EDIMENT</small> R<small>OUTING</small> C<small>OMPONENTS</small>

2.2.1   <u>Sediment Transport Approximation</u>

Once a soil particle erodes, it becomes part of the flow and is transported downstream. A particle moving past a control must have eroded somewhere in the watershed upstream of the cross section, and it must be transported by the flow from the point of detachment to the point of interest (Einstein 1950). Each of these two requirements may control the sediment rate at the cross section: the availability of sediment in the watershed and the transport capacity of the stream. Typically, the finer material, which is easily carried in large amounts by the flow, has limited availability in the watershed. The coarse material is much more difficult to move by flows, so its rate of movement is limited by the transport capacity of the flow (Haan et al. 1994). Julien (1995) used Figure 2-3 to explain the concept of supply and transport limited capacity.
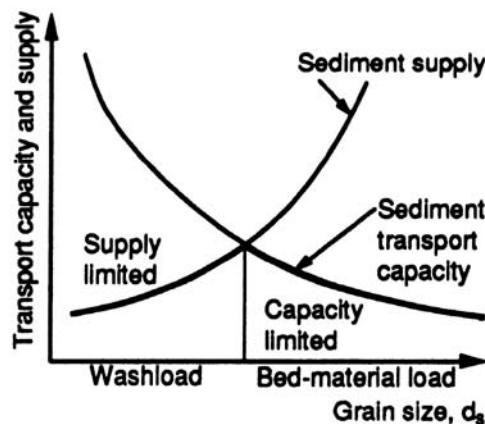
Figure 2-3 Sediment transport capacity and supply curves (Julien 1995)

It is often assumed that the washload travel through a system by stream flow with very little deposition and is carried primarily in suspension. Coarser fractions tend to move as bed-material. In addition, particles moving in suspension at one place may be

moving as bed-material further downstream. The total sediment discharge or total sediment load, consist of both bed-material load and washload. Generally, total sediment load can only be estimated if the washload is estimated by measurements, experiment, or upland sediment yield equation because most sediment transport methods can only determine bed-material load (Haan et al. 1994).

Streamwise velocity of sediment particles always lag behind the velocity of the surrounding fluid. This lag is small for the case of silt particles in suspension and large for the case of sands and gravels (Francis 1973). Thus, it is important to predict the movement of the individual sizes found in the flow (Borah et al. 1982).

CASC2D-SED uses a 2-D sediment flow routing in the overland that simulates the described transport processes for *n* numbers of particle sizes. In the overland, CASC2D-SED uses the modified Kilinc & Richardson (1973) transport capacity equation, which depends on flow discharge, terrain slope and soil and land use characteristics. Small particles such as clay and silt move mostly in suspension while the sand fraction moves as bed-material. This is accomplished by using the particle settling concept: coarser particles brought into suspension rapidly settle while small fractions such as clay try to remain in suspension due to its low settling velocity. Once it is in suspension, the suspended size fraction moves by advection. The advective fluxes describe the transport of sediments imparted by velocity currents (Julien 1995). This implies that the sediment will move with the fluid even for capacity limited conditions. The excess transport capacity is defined as the flow capacity to move the bed-material and produce erosion of the parent material once the transported suspended sediment volume is subtracted from the total transport capacity. First, the excess transport capacity

is used to move the sediment by size fraction according to its percentage in the bed-material (material deposited earlier in the simulation). For each size fraction, the amount of bed-material that will be transported will be limited by the amount that can be transported by the flow by advection or by the excess transport capacity. Then, if there is any remaining transport capacity once the suspended and bed-material have been transported, the soil is eroded proportionally to the percentage of the corresponding size fraction in the parent material. See Figure 2-4 and Figure 2-5 for overland erosion and sediment transport processes schematics and flowchart, respectively.

The volumes of deposited or eroded material translate into a change in the corresponding cell elevation at each time step. At the end of the simulation, the input DEM is modified according to the elevation change in each cell.
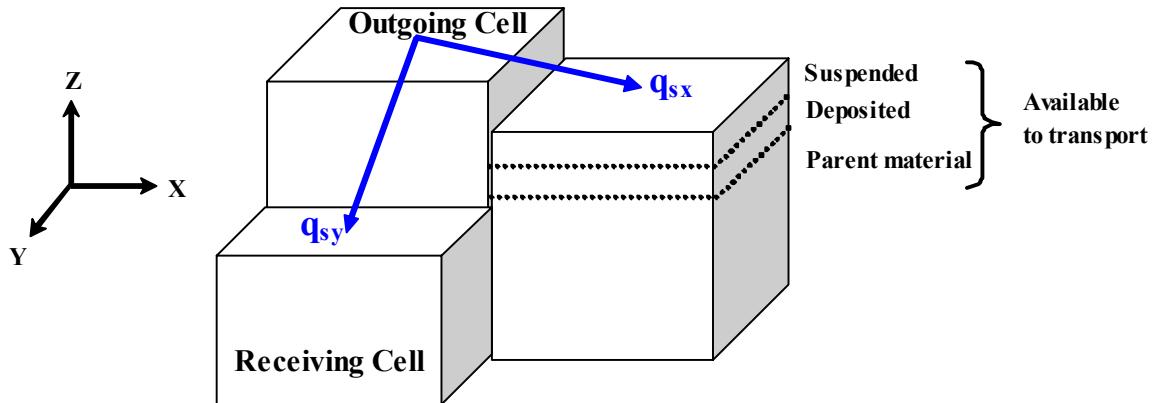
Figure 2-4 Schematic of upland sediment transport (after Johnson 1997)
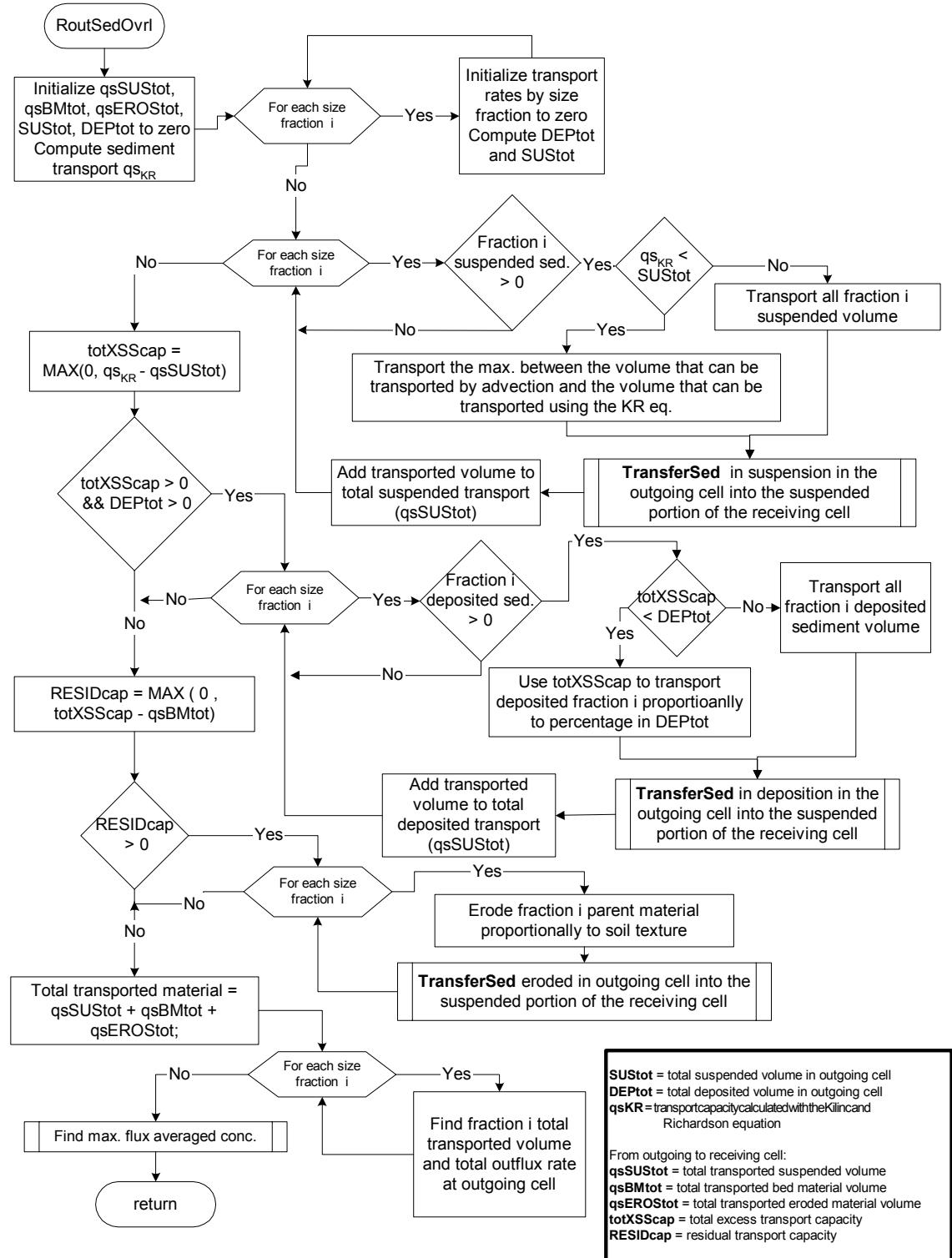
Figure 2-5 Overland erosion and sediment routing flowchart

Sediment by size fraction is routed in the channels using the Engelund and Hansen (1967) transport equation in 1-D for *n* numbers of particle sizes. This formulation depends on hydraulic parameters (hydraulic radius, flow velocity and friction slope) and particle characteristics (specific gravity and particle diameter). For each of those fractions, a transport capacity is calculated using the Engelund and Hansen equation. For a particular size fraction, the excess transport capacity is calculated by subtracting the amount of sediment carried in suspension by advective processes from the transport capacity. The bed-material is transported using the excess transport capacity. The amount of transported bed-material is calculated as the minimum between the amount that can be carried by advective processes and the excess transport capacity. Channels are not allowed to erode in CASC2D-SED and thus, the remaining transport capacity is not used. See Figure 2-6 and Figure 2-7 for overland erosion and sediment transport processes schematics and flowchart, respectively.
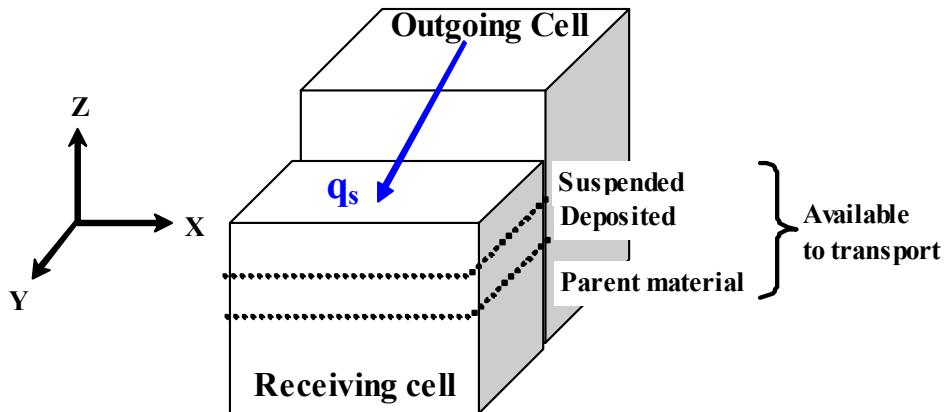
Figure 2-6 Schematic of channel sediment transport
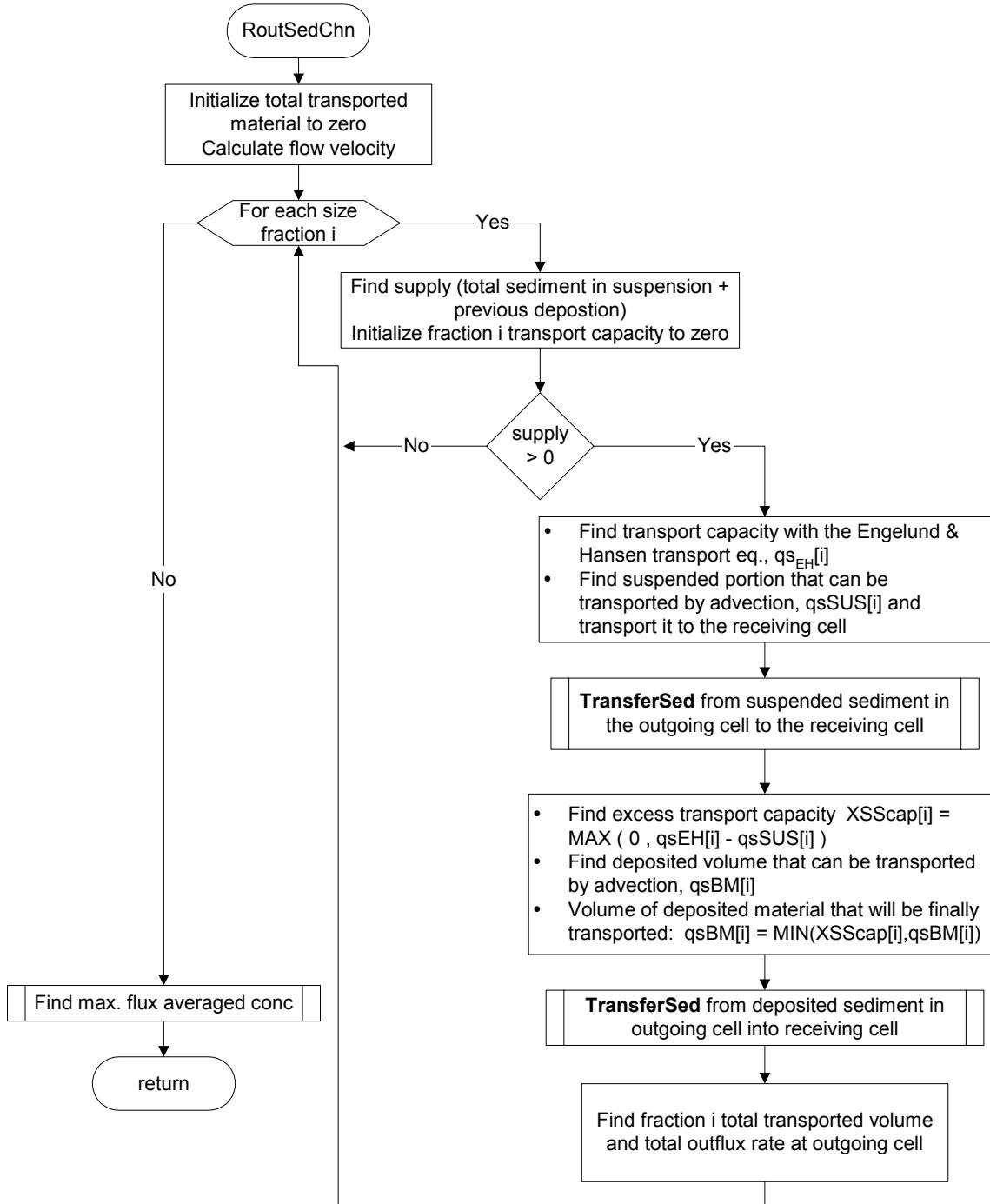
Figure 2-7 Channel sediment routing flowchart

The transported material (from suspension, bed-material or parent material) is transferred from the outgoing cell into the suspended portion of the receiving cell. Once the sediment has been routed for all the overland and channel cells in the watershed in the

x- and y-direction, the sediment in suspension is allowed to settle. For a given sediment size fraction at a given cell, the volume of suspended sediment that will settle depends on the total volume of that fraction in suspension, on the particle settling velocity and the water depth at the given cell.

2.2.2 **Upland Erosion and Sediment Transport**

Julien & Simons (1984) derived a general relationship supported by dimensional analysis that can be written as a power function of slope and discharge. The unit upland sediment discharge from sheet and rill erosion can be written in the form (Julien and Simons 1984):

$$q_s = \alpha \, S_o^{\beta} q^{\gamma} \tag{13}$$

where:
  S_o =   surface slope m/m]
  q =   unit discharge [m^2/s]
  q_s =   sediment unit discharge
  α, β, γ =   coefficients

The values of the exponents β and γ range typically between $1.2 < \beta < 1.9$ and $1.4 < \gamma < 2.4$ (Julien & Simons 1984).

In 1972, Kilinc (1972) studied experimentally and analytically the mechanics of soil erosion from overland flow generated by simulated rainfall. The main objectives of his research were to study the most important factors affecting soil erosion, and to develop a soil loss prediction equation. Experiments were conducted at the rainfall-runoff facilities at the Engineering Research Center of the Colorado State University. Data was collected for sediment concentration, surface velocity of overland flow, water discharge, water temperature, infiltration rate, bulk density of surface soil, slope, intensity

of rainfall, and rill geometry. Some of the conclusions from derived from the study of Kilinc were: a) the Reynolds number and slope were the most important parameters in sediment transport prediction equations; b) the Reynolds number, rainfall excess, and water discharge each had the same significance and influence on sediment transport from overland flow; and c) sediment discharge increases with the square of water discharge and 5/3 power of the slope. This model was comparable to that used by Meyer and Wischmeier (1969).

The Kilinc and Richardson (1973) equation was developed for the estimation of sheet and rill erosion from bare soils. In the more general case of erosion from sheet flow, modifications to the last equation reflect the influence of soil type, vegetation, and practice factor using the USLE factors *K*, *C*, and *P* as (Julien 1995):

$$q_s = 23210 \, S_o^{1.664} \, q^{2.035} \, \frac{K}{0.15} \, C \, P \qquad\qquad [14]$$

where:

qs	=	unit sediment discharge [tons m^{-1} s^{-1}]
So	=	terrain slope [m m^{-1}]
q	=	unit flow discharge [m^2 s^{-1}]
K	=	erodibility factor in the USLE equation [--]
C	=	cropping management factor in the USLE equation [--]
P	=	conservation practice factor in the USLE equation [--]

For a grid of cell size, Δx [m], and for a time interval, Δt [s], the total volume $\forall s_{KR}$ [m^3] of sediment coming from a cell is calculated as:

$$\forall_{S_{KR}} = 58390 * S_o^{1.664} * q^{2.035} * K * C * P * \Delta x * \Delta t \qquad\qquad [15]$$

On the other hand, the rate of mass transport carried by advection for size fraction i, qs_{ADVi} [m^3 s^{-1}] per unit area A [m^2], is obtained from the product of size fraction i

sediment concentration, Ci [m^3 m^{-3}], and the average flow velocity component, V [m s^{-1}] (Julien 1995):

$$\frac{\text{qs}_{\text{ADVi}}}{A} = V * C_i \qquad [16]$$

Simplifying and integrating for a time step Δt [s], the volume (in m^3) of suspended sediment size fraction, i, that can be transported by advection is:

$$SusVol_i * \frac{V * \Delta t}{\Delta x} \qquad [17]$$

where $SusVol_i$ [m^3] is the size fraction i suspended volume. The volume of size fraction i transported from the suspended portion of the source cell, $\forall s_{susi}$, into the receiving cell is found as the maximum value between the amount of sediment that can be carried by advection and the amount calculated using the Kilinc and Richardson equation:

$$\forall s_{\text{SUSi}} = \text{MAX}\left(SusVol_i * \frac{V * \Delta t}{\Delta x} \quad ; \quad \forall s_{\text{KR}} * \frac{SusVol_i}{\sum_{i=1}^{3} SusVol_i} \right) \quad \text{if} \quad \forall s_{\text{KR}} < \sum_{i=1}^{3} SusVol_i \qquad [18]$$

$$\forall s_{\text{SUSi}} = SusVol_i \qquad\qquad\qquad\qquad\qquad \text{otherwise}$$

The transport capacity is then reduced by the amount of sediment in suspension that is carried from the source cell. The total excess capacity (*totXSScap*) of the flow to carry sediments is:

$$\text{totXSScap} = \text{MAX}(0 \quad ; \quad \forall s_{\text{KR}} - \sum_{i=1}^{3} \forall s_{\text{SUSi}}) \qquad [19]$$

This excess capacity is used to transport the sediment from the bed-material in the source cell. The volume of size fraction i transported from the bed-material, $\forall s_{BMi}$, [m^3] is found as:

$$\forall s_{BM\,i} = totXSScap * \frac{BMvol_i}{\sum\limits_{i=1}^{3} BMvol_i} \qquad \text{if } totXSScap < \sum\limits_{i=1}^{3} BMvol_i$$

$$\forall s_{BM\,i} = BMvol_i \qquad\qquad\qquad\qquad \text{otherwise}$$

[20]

where $BMvol_i$ [m^3] is the volume of size fraction i found in the source cell bed-material.

Once the sediment in suspension and the bed-material are transported and provided that there is still remaining capacity left, the soil is eroded proportionally to the percentage of fraction i in the parent material, P_i:

$$\forall s_{EROS\,i} = (totXSScap - \sum\limits_{i=1}^{3} Qs_{BM\,i}) * P_i$$

[21]

where $\forall s_{EROSi}$ [m^3] is the volume of size fraction i eroded from the parent material.

This algorithm provides a better transition between supply-limited and capacity-limited sediment transport calculations.

2.2.3   Channel Sediment Transport

The eroded material in the upland portion of the watershed is transported to the outlet through the channels. Erosion is not allowed to occur in the channels. Thus, transport might be supply limited. Deposition of material is allowed. The Engelund and Hansen (1967) equation is used to calculate the sediment transport capacity in the channels.

Engelund and Hansen (1967) applied Bagnold's stream power concept and the similarity principle to obtain the sediment concentration by weight as follows:

$$Cw_i = 0.05 * \left(\frac{G}{G-1} \right) * \frac{V * S_f}{\sqrt{(G-1) * g * ds_i}} * \sqrt{\frac{R_h * S_f}{(G-1) * ds_i}}$$
[22]

where:
   G = sediment specific gravity
   V = channel depth-averaged velocity [m s^{-1}]
   S_f = channel friction slope [m m^{-1}]
   g = gravitational acceleration [m^2 s^{-1}]
   ds_i = size fraction i diameter [m]
   R_h = channel hydraulic radius [m]

The volume of size fraction i, $\forall_{sEHi}$ [m^3] that can be transported during a time interval, Δt [s], is estimated as:

$$\forall s_{EHi} = \frac{Q * Cw_i * \Delta t}{2.65}$$
[23]

where:
   Q = flow discharge in the channel [m^3 s^{-1}]
   Cwi = sediment concentration by weight of size fraction i

The volume of suspended sediment size fraction i, $\forall_{sSUS}i$ [m^3], that is transported in the channels by advection is:

$$\forall s_{SUSi} = SusVol_i * \frac{V * \Delta t}{\Delta x}$$
[24]

The excess transport capacity for a given fraction i, $XSScap_i$ [m^3], is used to carry size fraction i channel bed-material.

$$XSScap_i = MAX(0 \ ; \ \forall s_{EHi} - \forall s_{SUSi})$$
[25]

On the other hand, the volume of fraction i bed-material that can be transported by advection in the channel is proportional to:

$$BMvol_i * \frac{V * \Delta t}{\Delta x}$$

[26]

where $BMvol_i$ [m^3] is the volume of size fraction i found in the bed-material. The volume of size i bed-material that will be finally transported from the bed-material, $\forall s_{BMi}$, is found to be the minimum between the excess transport capacity and the volume of the bed-material that can be carried by advection for that size fraction:

$$\forall s_{BMi} = MIN\left(XSScap_i \; ; BMvol_i * \frac{V * \Delta t}{\Delta x} \right)$$

[27]

In the case in which there is still remaining transport capacity, this will not be used, as channels are not currently allowed to erode in CASC2D-SED.

2.2.4 Suspended Sediment Settling

A particle falling in turbulent free water moves in response to the difference in the submerged weight of the particle and the drag of the fluid on the particle. At steady state, the forces in equilibrium are described by:

$$CD\left[\frac{\pi d^2}{4}\right]\left[\frac{\rho \omega_s^2}{2}\right] = \frac{\pi d^3}{6}(\rho_s - \rho)g$$

[28]

where:
CD	=	drag coefficient, normally a function of the Reynolds number, Re
d	=	particle diameter
ρ_s	=	particle density
ρ	=	fluid density
ω_s	=	particle settling velocity
g	=	acceleration of gravity

Stokes showed that the drag for spheres depends on the Reynold's number:

$$CD = \frac{24}{Re} \qquad [29]$$

where Re depends on the kinematic viscosity, ν:

$$Re = \frac{\omega_s\, d}{\nu} \qquad [30]$$

For the Stokes' range (i.e. Re < 0.5). Equation 3-30 can be simplified to yield (Haan et al. 1994; Julien 1995)

$$\omega_s = \frac{1}{18}\left[\frac{d^2 g}{\nu}(G-1)\right] \qquad [31]$$

where G is the specific gravity of the particles. The settling velocity of large particles (> 0.2 mm and Re > 0.5), can be estimated with the following relationship (Julien 1995):

$$\omega_s = \frac{8\,\nu}{d}\left\{\left[1+0.0139\,d^3\,\frac{(G-1)\,g}{\nu^2}\right]^{0.5} - 1\right\} \qquad [32]$$

Assuming, medium sediment particle diameter for the sand, silt and clay fractions, the following fall velocities are estimated:

Table 2-1 Particle mean diameter and fall velocity

Size fraction	d [mm]	ws [m/s]
Sand	0.35	0.036
Silt	0.016	2.20E-04
Clay	0.001	8.60E-07

Once the sediment has been routed in overland and channel cells, the suspended sediment portion is allowed to deposit. CASC2D-SED assumes a discrete particle settling type in which particles tend to fall independently of each other as discrete particles. The percentage of suspended sediment fraction i, *PercSett$_i$*, that is allowed to settle depends on the fall velocity corresponding to the median size fraction diameter, ω_i [m/s], elapsed time (i.e. computational time step, Δt [s]) and water depth d [m] in the cell.

$$\text{PercSett}_i = \omega_i \frac{\Delta t}{h} \qquad \text{if} \quad h > \omega_i * \Delta t$$
$$\text{PercSett}_i = 1 \qquad \text{otherwise}$$

[33]

The volume of suspended sediment that settles in a given time increment is subtracted from the suspended sediment portion and added to the deposited one. This algorithm improves upon the trap efficiency algorithm by allowing continuous sedimentation of the suspended load in the backwater areas.

3. NUMERICAL MODELING SCHEME

3.1 EXPLICIT NUMERICAL APPROACH

To understand the method by which the CASC2D model conducts its calculations, it is important to conceptualize the two dimensional spatial nature and resolution of the given watershed data. The explicit numerical solution is achieved by segmenting the entire catchment area into equal square elements, which are then assigned parameters relating to soil infiltration characteristics, roughness coefficients, and soil erosion parameters (Spah 2000). As each parameter is defined, it is assumed to be uniform throughout the cell area with the actual value assigned to a central nodal point.

Figure 3-1 provides a visual illustration of how the grid concept within the CASC2D model is defined for a small nine-cell section of a watershed (Julien and Saghafian 1991).
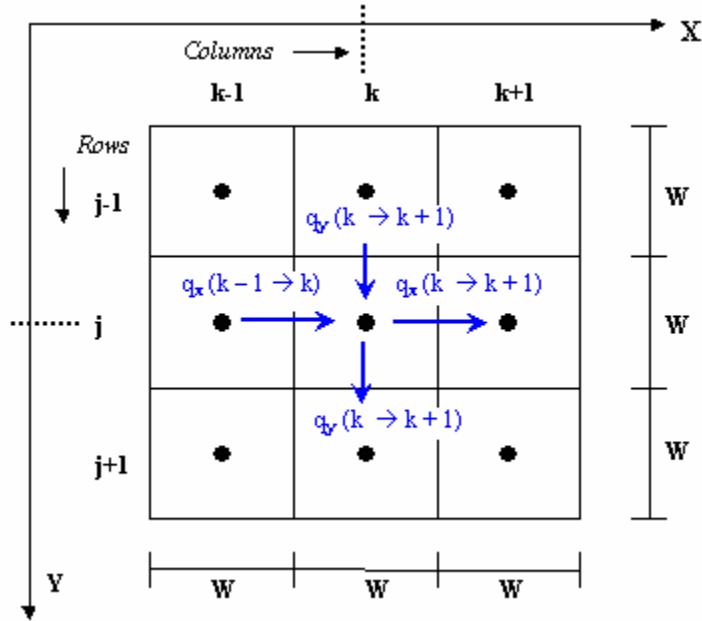
Figure 3-1 A Typical Two Dimensional Model Grid Mesh (Julien and Saghafian 1991).

3.2   THE EXPLICIT SOLUTION FOR INFILTRATION

Initially, when a rainfall event begins, the intensity of the rain event is compared to the infiltration capacity of the soil. The model allows the spatially distributed rainfall volume to infiltrate on a cell by cell basis until a point where the rainfall intensity equals the infiltration capacity of the soil. Once these values are equal, ponding can be observed on the soil surface and overland flow begins (Julien & Saghafian 1991).

As the model progresses beyond the commencement of the rain event, the method used to determine the infiltration depth for each time step is based upon the total current accumulated infiltration depth. The infiltration capacity is then compared to the existing surface depth divided by the established time step (representing the maximum possible rate at which surface water can infiltrate the soil layer). Based upon this comparison, if the infiltration capacity is greater than the maximum infiltration rate, all of the surface

water will be transformed to a local infiltration depth. Conversely, if the capacity is less than the maximum rate, then the value of surface depth divided by the time step is reduced by the capacity rate. This result is then multiplied by the time step to convert it back to a depth value (Saghafian 1992).

The CASC2D model determines the infiltration for each cell at the middle of the given time step. The relation used to calculate this value is (Saghafian 1992):

$$f^{t+\Delta t} = K_s \left[1 + \frac{H_f M_d}{F^t + \frac{\Delta t}{2} f^{t+\Delta t}} \right]$$

[34]

where:

t	=	present time
Δt	=	time step
F	=	accumulated infiltration depth
Md	=	moisture deficit
Hf	=	water pressure head

Since the Green-Ampt equation is implicit with respect to time, a time-explicit solution was proposed by Li et al. (1976) as follows:

$$\Delta F = -\frac{(2F - K\,\Delta t)}{2} + \frac{\left[(2F - K\Delta t)^2 + 8K\,\Delta t\,(\delta + F) \right]^{\frac{1}{2}}}{2}$$

[35]

Calling:

P_1	=	$(K\Delta t - 2F^t)$
P_2	=	$(KF^t + KH_f M_d)$

simplifying this equation, and solving for $f^{t+\Delta t}$ results in the following expression where the superscripts specify the time:

$$f^{t+\Delta t} = \frac{1}{2\Delta t} \left[P_1 + \left(P_1^2 + 8P_2 \Delta t \right)^{1/2} \right]$$

[36]

3.3 THE EXPLICIT SOLUTION FOR OVERLAND FLOW

The fundamental basis for overland flow routing within CASC2D is derived from the conservation of mass principle. For a fluid, this principle can be applied as the continuity relationship as long as the assumption of incompressibility is valid. These principles maintain that as fluid enters a given control volume over a finite period of time, the change in mass of that volume is proportional to the mass entering that volume. When applied to the model, the application of this principle depicts the flow of runoff from one grid cell to another. In this respect, water can be routed throughout the system until equilibrium is achieved. As shown by Saghafian (1992), the application of the first order approximation to the continuity equation for element (j,k) can be described by the following equation.

$$h^{t+\Delta t}(j,k) = h^t(j,k) + r_e \Delta t - \left[\frac{q_x^t(k \to k+1) - q_x^t(k-1 \to k)}{W} + \frac{q_y^t(j \to j+1) - q_y^t(j-1 \to j)}{W} \right] \Delta t \quad [37]$$

where:

$h^{t+\Delta t}(j,k)$	=	flow depth at element (j,k) at time t+Δt
$h^t(j,k)$	=	flow depth at element (j,k) at time t
Δt	=	time step duration
r_e	=	average excess rainfall rate over one time step beginning at time t
$q_x^t(k \to k+1)$	=	unit flow rates in the x-direction at time t, from (j,k) to (j, k+1)
$q_x^t(k-1 \to k)$	=	unit flow rates in the x-direction at time t, from (j, k-1) to (j,k)
$q_y^t(j \to j+1)$	=	unit flow rates in the y-direction at time t, from (j,k) to (j+1, k)
$q_y^t(j-1 \to j)$	=	unit flow rates in the y-direction at time t, from (j-1, k) to (j,k)
W	=	grid size

The unit flow rate terms used in the above equation are determined in the model by a discretized form of Manning's roughness equation, which presumes all flow is within the turbulent regime. The direction of the unit flow rate for any given time and

location is strictly dependent on its relation to the friction slope, S_f, shown in the following equations for the x- and y-direction (Saghafian 1992):

$$S_{fx}^t(k-1 \to k) \cong S_{0x}(k-1 \to k) - \frac{h^t(j,k) - h^t(j,k-1)}{W} \qquad [38]$$

$$S_{fy}^t(j-1 \to j) \cong S_{0y}(j-1 \to j) - \frac{h^t(j,k) - h^t(j-1,k)}{W} \qquad [39]$$

where the bed slope, S_{0x}, is expressed in terms of the cell elevation, E:

$$S_{0x}(k-1,k) = \frac{E(j,k-1) - E(j,k)}{W} \qquad [40]$$

$$S_{0y}(j-1,j) = \frac{E(j-1,k) - E(j,\ k)}{W} \qquad [41]$$

The unit flow rate is approximated within the model for flow between two elements oriented in the x-direction, (j,k-1) and (j,k) as:

$$q_x^t(k-1 \to k) = \frac{1}{n(j,k-1)}\left[h^t(j,k-1)\right]^{5/3}\left[S_{fx}^t(k-1 \to k)\right]^{1/2} \quad if \quad S_{fx}^t(k-1 \to k) \geq 0 \qquad [42]$$

$$q_x^t(k-1 \to k) = \frac{-1}{n(j,k)}\left[h^t(j,k)\right]^{5/3}\left[-S_{fx}^t(k-1 \to k)\right]^{1/2} \quad if \quad S_{fx}^t(k-1 \to k) < 0 \qquad [43]$$

Similarly, the discharge $q_x^t(k \to k+1), q_y^t(j \to j+1)$ and $q_y^t(j-1 \to j)$ are calculated based on the sign of $S_{fx}^t(k \to k+1), S_{fy}^t(j \to j+1)$ and $S_{fy}^t(j-1 \to j)$, respectively. The initial condition for surface depth is assumed to be zero and the boundary condition is accounted for by having no inflow coming from outside of the basin's boundaries.

This entire process of overland flow routing is conducted for a time duration and increment specified by the user.

3.4 THE EXPLICIT SOLUTION FOR CHANNEL FLOW

The process by which CASC2D transports runoff through a defined channel network is based on a one-dimensional, diffusive wave approximation scheme. This scheme is mathematically similar to that used for overland flow computations. As the modeled rainfall event progresses, overland flow is routed through the overland planes until the flow reaches a channel location. This flow is then transformed into a channel flow component and routed through the channel network to the defined outlet location. During the flow routing process, channel losses or gains are neglected.

The channel network defined in a CASC2D project is made up of links, numbered according to the computational order. A channel link is made up of connected grid cells, called the nodes of that particular link. Each node describes the channel reach represented by that node with geometric and hydraulic characteristics.

The channel cross section is assumed to be rectangular. The flow occurs from one cell center to the neighboring cell center in any of the eight flow directions. The channel is assumed to be located in the middle of the grid cell as shown in Figure 3-2.
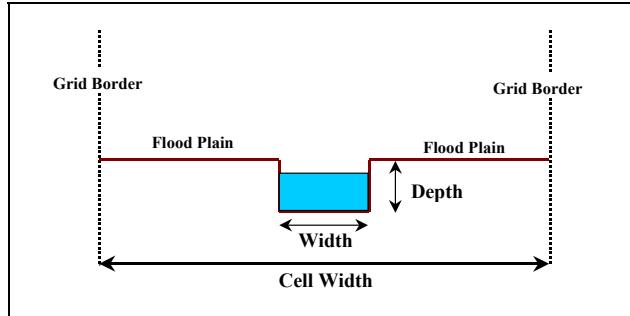
Figure 3-2 Channel cross section.

CASC2D simulations involving channel flow is able depict time varying backwater effects for simulated runoff flows. These backwater effects are continuous as they are carried through all associated channel junctions within the defined stream network (Saghafian 1992).

The computational time step in channels is first equal to the simulation time step. In the case in which the channel routing becomes unstable at a given time step, this time step is cut in half and the flow is routed in the channel network in two smaller steps. If the the channel depth still becomes negative, the time step is divided again by two. This process is repeated until the water can be routed in the channels without being unstable or until the time step becomes unreasonably small (<0.0001).

3.5   THE EXPLICIT SOLUTION FOR UPLAND AND CHANNEL TRANSPORT

Most existing models are based upon the same fundamental physical principles of conservation of mass and momentum (Bennett 1974). In a physical model, the erosion components are normally subsidiary to a hydrologic model (Rojas and Woolhiser 2000).

In CASC2D, the Kilinc (1972) equation used for the estimation of the unit sediment discharge in the overland cells and the Engelund and Hansen equation (1967) in the channel cells.

In the overland cells, the flow, q, is computed at each cell (j,k) in two orthogonal directions to the neighboring cell (jj,kk) – to the right of the cell and to the down side of the cell. In any of those directions, the flux can be:

1. Positive: this is an inflow (q < 0) to that cell and Sf < 0. In this case, the sediment is transported into the current computational cell (jto = j , kto = k) from the source cell (jfrom = jj, kfrom = kk).

2. Negative: this is an outflow (q > 0) from that cell and Sf ≥ 0. In this case, the sediment is transported out of the current computational cell (jfrom = j , kfrom = k) and into the sink cell (jto = jj , kto = kk).

Once the source or outgoing cell (jfrom, kfrom) and the sink or receiving cell (jto, kto) has been determined, the transport capacity is calculated using the Kilinc and Richardson equation (in Page 19):

$$\forall s_{KR} = 23210 \, | \, S_f \, |^{1.664} \, | \, q \, |^{2.035} \, \frac{K(jfrom, kfrom)}{0.15} \, C(jfrom, kfrom) \, P(jfrom, kfrom) \, \Delta x \, \Delta t \qquad [44]$$

This volume of sediment is transported by size fraction from the suspended, deposited or parent material portion in the source cell (jfrom, kfrom) to the suspended portion of the sink cell (jto,kto) according to the scheme described in Section 2.2.2.

In the channels, the total flow q is computed at each cell (j,k) between that cell and the downstream cell (jj,kk).   In case of a positive flow (inflow), the sediment is transported into the current computational cell (jto = j , kto = k) from the downstream cell (jfrom = jj, kfrom = kk). In case of a negative flow (outflow), the sediment is transported

33

out of the current computational cell (jfrom = j , kfrom = k) and into the sink or downstream cell (jto = jj , kto = kk).

Once the source or outgoing cell (jfrom, kfrom) and the sink or receiving cell (jto, kto) have been determined, the transport capacity is calculated using the Engelund and Hansen equation (in Page 22). This volume of sediment is transported by size fraction from the suspended or deposited portion in the source cell (jfrom, kfrom) to the suspended portion of the sink cell (jto,kto) according to the scheme described in Section 2.2.3).

REFERENCES

Bennett, J. P. (1974). "Concepts of mathematical modeling of sediment yield". *Water Resources Research*, 10(3), 485-492.

Borah, D. K., Alonso, C. V., and Prasad, S. N. (1982). "Routing graded sediment in streams: formulations". *J. Hydraulic Division*, 108(12), 1486-1503.

Bras, R. L. (1990). *Hydrology, an introduction to hydrologic science*, Addison -Wesley.

Brigham Young University (2001). "Environmental Modeling Research Laboratory: WMS", <http://emrl.byu.edu/wms.htm>. Accessed on 2001/7/29.

Chow, V. T., Maidment, D. R., and Mays, L. W. (1988). *Applied hydrology*, McGraw-Hill.

Eagleson, P. S. (1970). *Dynamic hydrology*, McGraw-Hill.

Einstein, H. A. (1950). "The bed-load function for sediment transportation in open channel flows". U.S. Dept. of Agriculture, Washington.

Engelund, F. and Hansen, E. (1967). *A monograph on sediment transport in alluvial streams*, Teknisk Forlag, Copenhagen, Denmark.

Francis, J. R. D. (1973). "Experiments on the motion of solitary grains along the bed of a
water stream". *Procs. Royal Soc. London*, 443-471.

Green, W. H. and Ampt, G. A. (1911). "Studies on soil physics - Part I. The flow of air
and water through soils". *The Journal of Agricultural Science*, 4, 1-24.

Haan, C. T., Barfield, B. J., and Hayes, J. C. (1994). *Design hydrology and
sedimentology for small catchments*, Academic Press.

Johnson, B. E. (1997). "Development of a storm event based two-dimensional upland
erosion model". PhD thesis, Dept. Civil Engr., Colorado State University, Fort
Collins, Colorado.

Johnson, B. E., Julien, P. Y., Molnar, D. K., and Watson, C. C. (2000). "The two-
dimensional-upland erosion model CASC2D-SED". *J. of the AWRA*, 36(1), 31-42.

Julien, P. Y. (1995). *Erosion and Sedimentation*, Cambridge University Press,
Cambridge, New York.

Julien, P. Y., and Saghafian, B. (1991). "CASC2D user's manual - A two dimensional
watershed rainfall-runoff model". *Civil Eng. Report, CER90-91PYJ-BS-12,*
Colorado State University, Fort Collins, Fort Collins, CO.

Julien, P. Y., Saghafian, B., and Ogden, F. L. (1995). "Raster-Based hydrologic modeling

of spatially-varied surface runoff". *Water Resources Bulletin*, 31(3), 523-536.

Julien, P. Y., and Simons, D. B. (1984). "Analysis of sediment transport equations for rainfall erosion". *Civil Eng. report: CER83-84PYJ-DBS52,* Colorado State University, Fort Collins, CO.

Kilinc, M. Y. (1972). "Mechanics of soil erosion from overland flow generated by simulated rainfall". PhD thesis, Colorado State University.

Kilinc, M. Y., and Richardson, E. V. (1973). "Mechanics of soil erosion from overland flow generated by simulated rainfall". *Hydrology Papers No. 63,* Colorado State University, Fort Collins, CO.

Li, R. M., Stevens, M. A., and Simons, D. B. (1976). "Solutions to Green-Ampt infiltration equations". *J. Irrigation and Drainage*, 102(No. IR2), 239-248.

Meyer, L. D. and Wischmeier, W. H. (1969). "Mathematical simulation of the process of soil erosion by water". *Transactions of the ASAE*, 12, 754-758.

Ogden, F. L. (1994). "de-St Venant channel routing in distributed hydrologic modeling". *Procs. Hydraulic Engineering `94, ASCE Hydraulics Specialty Conference*, 492-496.

Ogden, F. L. (1997b). "Primer: Using WMS for CASC2D Data Development". Brigham

Young University, Provo, UT.

Philip, J. R. (1983). "Infiltration in one, two, and three dimensions". *Procs. National Conference on Advances in Infiltration*, American Society of Agricultural Engineers, St. Joseph, Mich., 1-13.

Rawls, W. J., Brakensiek, D. L., and Miller, N. (1983). "Green-Ampt infiltration parameters from soils data". *Journal of Hydraulic Engineering*, 109(1), 62-70.

Rojas, R. (2002). "GIS-based upland erosion modeling, geovisualization and grid size effects on erosion simulations with CASC2D-SED". PhD thesis, Dept. Civil Engr., Colorado State University, Fort Collins, Colorado.

Rojas, R. and Woolhiser, D. A. (2000). "Erosion parameters identifiability in the KINEROS model". *Procs. the debris flow disaster of December 1999 in Venezuela*, Dynamics of Debris Flow section (CD-ROM).

Saghafian, B. (1992). "Hydrologic analysis of watershed response to spatially varied infiltration". PhD thesis, Colorado State University.

Saghafian, B. (1993). "Implementation of a distributed hydrologic model within Geographic Resources Analysis Support System (GRASS)". *Procs. Second International Conference on Integrating Environmental Models and GIS*.

Saghafian, B. and Ogden, F. L. (1996). "r.hydro.CASC2D",

        <http://www.baylor.edu/grass/gdp/html_grass5/html/r.hydro.CASC2D.html>.

        Accessed on 2001/7/29.

Spah, J. A. (2000). "Rainfall runoff and the effects of initial soil moisture associated with

        the Little Washita River Watershed, Oklahoma ". MSc thesis, Colorado State

        University.

Woolhiser, D. A. (1975). "Simulation of unsteady overland flow." *Unsteady flow in open*

        channels, K. Mahmood and V. Yevjevich, eds., Fort Collins, CO.

Woolhiser, D. A., Smith, R. E., Sharif, H. O., and Goodrich, D. C. (1990). "KINEROS, a

        kinematic runoff and erosion model: documentation and user manual". *ARS-77,*

        U.S. Dept. of Agriculture, Agricultural Research Service.

APPENDIX I: PROGRAM INPUTS AND OUTPUTS

Model input and output files are defined in a Project file. The project file name represents the argument of the CASC2D-SED program. The list of input/output files and their definition is found in Table II-1, Table II-2 and Table II-3 below.

Input Files

Required CASC2D inputs and options are:

(1) Simulation control data: computational time step, simulation duration, and number of rows and columns and cell size of the grid.

(2) Precipitation data: precipitation duration and a) if uniform rainfall, the intensity or b) number of raingages, rainfall data time step, and rainfall gages UTM coordinates if distributed rainfall.

(3) Basin outlet characteristics: a) row, column and overland slope of the basin at the outlet and b) link characteristics if channel routing is selected: (link number, total nodes, type, width, depth, side slope, roughness and sinuosity)

(4) Soil parameters: number of different soil types, and a) if infiltration occurs: hydraulic conductivity, suction head, and moisture deficit and b) if erosion option is selected: percentages each soil fraction and the USLE soil erodibility factor.

(5) Land use parameters: number of different soil uses, roughness coefficient and interception depth and, if erosion option is selected, the USLE cover and management practice factors.

The input files are: the <u>simulation control file</u> (specifying parameters defined in the last 5 paragraphs), <u>channel data file</u>, and <u>precipitation file</u> (if distributed rainfall is

used). Input as raster ASCII maps are: DEM, soil index, land use index, watershed mask, and channel links and nodes (if channel routing is selected). The storage depth ASCII grid and base depth ASCII grid are optional grids.

Output Files

CASC2D-SED creates the next output files:

(1) Summary file: contains information on the simulation parameters, channel topology inferred from the link and node maps and the final results of the hydrological and erosion components.

(2) Flow and sediment flow files: containing the water flow and sediment flow at the outlet. Results can optionally be written at other internal locations inside the watershed.

(3) Time-series grids: They are optional and represent simulation values at predetermined times. Output raster maps include: water depth, rainfall rates, accumulated infiltrated depth, total suspended sediment volume, total deposited sediment volume, total sediment flux and concentration. For each size fraction, the amounts of suspended sediment, deposited sediment, sediment flux and sediment concentration are calculated as well. Time series grids can be later used as input in an animated file for the hydrological and sediment dynamics during the simulated event.

The project file is a text file that lists the input and output file names and location. The project file name is used as the argument when CASC2D is executed.

Table II-1 CASC2D-SED input and output text files

		Project name	File variable	Description
Input Text Files		CONTROL	control_file	Controls the simulation.
		PRECIP	rain_file	Precipitation data if gages are present [in/hr]
		CHANNEL	chn_file	Channel characteristics data for each link and node.
Output Text Files		SUMMARY	Summ_file	Simulation summary file
		DISCHARGE_OUT	dis_out_file	Hydrograph at outlet and, if selected, at internal locations [m³/s]
		SEDIMENT_OUT	sed_out_file	Sedigraph at outlet and, if selected, at internal locations [m³/s]

Table II-2 CASC2D-SED input raster maps.

	Project name	File variable	Variable Read	Description [units]
Input Grids	ELEVATION	elev_file	e[j][k]	Grid elevation at cell (j,k) [m]
	SOIL	soil_file	isoil[j][k]	Soil type index at cell (j,k)
	LANDUSE	iman_file	iman[j][k]	Land use / land cover index at cell (j,k)
	MASK	shap_file	ishp[j][k]	Watershed and channel network definition at cell (j,k)
	LINK	link_file	link[j][k]	Link number at cell (j,k)
	NODE	node_file	node[j][k]	Node number at cell (j,k)
	STORAGE_DEPTH	sdep_file	sdep[j][k]	Storage deposition at cell (j,k) [m]
	BASE_DEPTH	bdepth_file	ho[j][k]	Base depth at cell (j,k) [m]

Table II-3 CASC2D-SED output raster maps.

	Project name	File variable	Computed Variable (units)	Time series grid (units)	Description
Time Series Output Grids	RAINFALL	rname	rint[j][k] (m/s)	rint[j][k] * 3600000 (mm/h)	Rainfall rates at cell (j,k)
	INF_DEPTH	vinfname	vinf[j][k] (m)	vinf[j][k] * 1000 (mm)	Infiltrated water at cell (j,k)
	WATER_DEPTH	dname	h[j][k] or hch[j][k] (m)	h[j][k] or hch[j][k] (m)	Depth of water at cell (j,k)
	CONCENTRATION	SedConcName	[1]SedConc[i][j][k] (mg/l)	SedConcp[i][j][k] (mg/l)	Fraction i sediment conc. at cell (j,k)
		tSedConcName	[2]t_SedConc[i][j][k] (mg/l)	t_SedConc[i][j][k] (mg/l)	Total sediment concentration at cell (j,k)
	SUSPENDED	SusSedName	SusSed[i][j][k] (m^3)	SusSed[i][j][k] * 2.65 * 1E6 / $(\Delta x)^2$ (gr/m^2)	Fraction i suspended sed. at cell (j,k)
		tSusSedName	t_SusSed[j][k] (m^3)	t_SusSed[j][k] (m^3)	Total suspended sed. at cell (j,k)
	DEPOSITED	DepSedName	DepSed[i][j][k] (m^3)	DepSed[i][j][k] * 2.65 * 1E6 / $(\Delta x)^2$ (gr/m^2)	Fraction i deposited sed. at cell (j,k)
		tDepSedName	t_DepSed[j][k] (m^3)	t_DepSed[j][k] (m^3)	Total deposited sed. at cell (j,k)
	FLUX	SedFluxName	SedFlux[i][j][k] (m^3/s)	SedFlux[i][j][k] (m^3/s)	Fraction i sediment flux at cell (j,k)
		tSedFluxName	t_SedFlux[j][k] (m^3/s)	t_SedFlux[j][k] (m^3/s)	Total sediment flux at cell (j,k)
	NETEROS	tNetErosName	[3]t_NetEros[j][k] (mm)	t_NetEros[j][k] (mm)	Net erosion at cell (j,k)
	CCU	CCUname	[4]CCU[j][k]	CCU[j][k]	CCU index at cell (j,k)

1

$$\text{SedConc[i][j][k]} = \frac{\text{SusSed[i][j][k]} \, (m^3)}{h * (\Delta x)^2 \, (m^3)} * G * 1E6 \quad (mg/l)$$

2

$$\text{t\_SedConc[i][j][k]} = \frac{\sum_{i=1}^{\text{NumFracts}} \text{SusSed[i][j][k]} \, (m^3)}{h * (\Delta x)^2 \, (m^3)} = \frac{\text{t\_SusSed[j][k]}}{h * (\Delta x)^2} * G * 1E6 \quad (mg/l)$$

3

$$\text{t\_NetEros[j][k]} = \frac{\sum_{i=1}^{\text{NumFracts}} \text{DepSed[i][j][k]} + \sum_{i=1}^{\text{NumFracts}} \text{ErodSed[i][j][k]} \, (m^3)}{(\Delta x)^2 \, (m^2)} = \frac{\text{t\_DepSed[j][k]} + totscourv}{(\Delta x)^2} * 1000 \quad (mm)$$

4

$$\text{CCU[j][k]} = \sum_{i=4}^{6} \frac{\text{SedConc[i][j][k]} \, (mg/l)}{hard[i]} = \sum_{i=4}^{6} \frac{\text{SedConc[i][j][k]} * 1000 \, (\mu g/l)}{hard[i]}$$

APPENDIX II: CASC2D-SED CODE

CASC2D-SED GLOBAL VARIABLES DESCRIPTION

Variable Name	Description and units
amaxcdepth	Simulation absolute maximum channel water depth [m] (for any cell, at any time step)
amaxdepth	Simulation absolute maximum overland water depth [m] (for any cell, at any time step)
amaxrain	Simulation absolute maximum rainfall intensity [m/s] (for any cell, at any time step)
amaxSusCch	Simulation absolute maximum suspended sediment concentration [m^3] in channel cells (for any cell, at any time step)
amaxSusCh	Simulation absolute maximum suspended sediment volume [m^3] in channel cells (for any cell, at any time step)
amaxSusCov	Simulation absolute maximum suspended sediment concentration [m^3] in overland cells (for any cell, at any time step)
amaxSusOv	Simulation absolute maximum suspended sediment volume [m^3] in overland cells (for any cell, at any time step)
amaxvinf	Simulation absolute maximum infiltrated depth [m] (for any cell, at any time step)
amincdepth	Simulation absolute minimum channel water depth [m] (for any cell, at any time step)
amindepth	Simulation absolute minimum overland water depth [m] (for any cell, at any time step)
aminrain	Simulation absolute minimum rainfall intensity [m/s] (for any cell, at any time step)
aminvinf	Simulation absolute minimum infiltrated depth [m] (for any cell, at any time step)
areaQ[j]	Drainage area [hectares] of internal gage J (J = 1…NDIS)
areaQs[j]	Drainage area [hectares] of internal sediment gage J (J = 1…NSED)
bdepth_file_fptr	Grid ASCII file containing the input base depth [m]
CCU	Cumulative Criterion Unit
cfactor[j]	USLE cover factor corresponding to the J land use class
chn_file_fptr	File containing the channel characteristics data for each link and node.
chp[l][n][k]	Channel link L, node N parameter K value L=1..MAXLINKS; N=1…NCHAN_NODE[L] K= 1: channel type, 2: width, 3: depth, 4: side slope, 5: Manning's 'n', 6: sinuosity
crain	Uniform rainfall intensity (IRAIN=0) [m/h](entered as cm/h)
debug	File used for debugging purposes
dep_ch[i]	Size fraction I total deposited sediment volume [m^3] in the channel cells at the end of the simulation
dep_ov[i]	Size fraction I total deposited sediment volume [m^3] in the overland cells at the end of the simulation

Variable Name	Description and units
DepSed[i][j][k]	Deposited volume [m³] of size fraction I at cell (j,k)
dis_out_file_fptr	File containing the hydrograph data at the outlet and, if selected, at internal locations [m³/s]
dname	Prefix of the time series ASCII grids containing the depth of water in channel and overland cells [m]
dqch[j][k]	Channel flow rate [m³/s] at cell (j,k) at each time step
dqov[j][k]	Overland flow rate [m³/s] at cell (j,k) at each time step
ds[i]	Fraction i average sediment fraction diameter [m]
dt	Computational time step [s]
dx	Grid Cell size [m]
e[j][k]	Elevation at cell (j,k) [m]
elapsedTime	Time elapsed since the beginning of the simulation [s]
elconv	Elevation conversion factor (from input data units to meters)
elev_file_fptr	File containing the grid elevation values at (j,k) [m]
ErodSed[i][j][k]	Eroded volume [m³] of size fraction I at cell (j,k)
final_ch_vol	Water volume [m³] remaining in channel cells at the end of the simulation
final_ov_vol	Water volume [m³] remaining in overland cells at the end of the simulation
h[j][k]	Overland depth [m]
h0[j][k]	Initial water depth (or base depth) [m] in the watershed at the beginning of the simulation
hch[j][k]	Channel water depth [m] at channel cell (j,k)
header	String of characters holding the header information in all the ASCII grids used in a CASC2D-SED project
ichn[l][j][k]	Channel L element addresses L=1..MAXLINKS; J = 1..NCHAN_NODE[L]; K = 1:node row and; 2:column location;
icount	Counter. If ICOUNT = NPRN, flow discharge and sediment discharge are written and counter is set back to 1. ICOUNT is incremented by 1 at each time step.
iDepSed	Deposited sediment printout index (1: user wants to print size fraction i deposited sediment time series; ≠ 1: otherwise)
ifcount	Counter that is used as ASCII grid file extension and is incremented by 1 every time the series grids are writen (i.e. IPCOUNT = NPLT)
iman[j][k]	Manning's roughness coefficient at cell at each cell in the basin

Variable Name	Description and units
iman_file_fptr	File containing the land use / land cover index at each cell inside the watershed
indexbdepth	Input base depth option index (1: base depths are defined; ≠ 1: otherwise and initial base depth is assumed to be zero in all the cells inside the watershed)
indexchan	Explicit channel routing option index (1: channel routing is simulated; ≠ 1: otherwise)
indexdis	Internal discharge option index (1:hydrograph at internal locations is written, ≠ 1:otherwise)
indexeros	Erosion option index (1: erosion is simulated; ≠ 1: otherwise)
indexinf	Infiltration option index (1: infiltration is simulated; ≠ 1: otherwise)
indexsdep	Storage depth option index (1: storage depth is simulated; ≠ 1: otherwise)
indexsed	Internal sedigraph option index (1: sedigraph at internal locations is written, ≠ 1: otherwise)
iNetEros	Net erosion printout option index (1: user wants to print size fraction I net erosion time series; ≠ 1: otherwise)
init_ch_vol	Initial water volume [m^3] in channel cells (= 0 if INDEXDEP = 0)
init_ov_vol	Initial water volume [m^3] in overland cells (= 0 if INDEXDEP = 0)
ipcount	Counter. If IPCOUNT = NPLT then time series grids are written and counter set back to 1. IPCOUNT is incremented by 1 at each time step.
iq[j][k]	Location of internal gage J (only if INDEXDIS = 1) J = 1 ..NDIS; K = 1:gage column location; K = 2:gage row location
irain	Rainfall index (1: distributed rainfall; ≠ 1: uniform rain)
ised[j][k]	Location of internal sediment gage J (only if INDEXDIS = 1) J = 1..NSED; K = 1: sediment gage column location; K = 2: sediment gage row location
iSedConc[i]	Sediment concentration printout index (1: user wants to print size fraction I sediment concentration time series; ≠ 1: otherwise)
iSedFlux[i]	Sediment flux printout index (1: user wants to print size fraction I sediment flux time series; ≠ 1: otherwise)
ishp[j][k]	Watershed mask value at cell (j,k)(1: grid cell is within the watershed;0: grid cell is outside of the watershed; 2: channel cell)
isoil[j][k]	Soil type index at cell (j,k)
iSusSed[i]	Suspended sediment printout index (1: user wants to print size fraction I suspended sediment time series; ≠ 1: otherwise)
iter	CASC2D-SED main time loop iteration number
jout	Row number for the outlet cell
kfactor[s]	USLE erodibility factor corresponding to the S soil type
kout	Column number for the outlet cell

Variable Name	Description and units
link[i][j]	Link number at channel cell (j,k)
link_file_ftpr	File containing the link number at (j,k)
maxCCU[j][k]	Simulation absolute maximum CCU index (for any cell, at any time step)
maxlinks	Total number of channel links
maxnodes	Maximum number of nodes found in any of the links in the channel network
ncells	Number of cells defining the watershed
nchan_node[l]	Number of nodes of link L
ncols	Input and output ASCII grids number of columns in the CASC2D-SED project
ndis	Number of locations where discharge is written
NetEros[j][k]	Total net volume of soil (scour+deposited) at cell (j,k)
niter	Total number of time intervals for the runoff simulation
nitrn	Total number of time intervals for the rainfall (= Precipitation duration * 60 / dt)
nman	Number of land use classes found in the overland plane
nodatavalue	No data value in the grid maps (integer type) in the CASC2D-SED project
node[j][k]	Node number at channel cell (j,k)
node_file_fptr	File containing the node number at (j,k)
nplt	Number of time steps to update graphics display
nprn	Number of time steps to write the output
nread	Ratio between raingage rainfall time step and simulation's time step, dt (only if IRAIN = 1)
nrg	Number of rainfall gages (only if IRAIN =1)
nrows	Input and output ASCII grids number of rows in the CASC2D-SED project
nsed	Number of locations where sediment discharge is written
nsoil	Number of soil types found in the overland plane
NumFracts	Maximum number of size fractions found any of the soil types defined in the soil type input grid
pfactor[l]	Soil conservation practice factor corresponding to land use L

Variable Name	Description and units
pfraction[s][f]	Fraction of size fraction F found in soil type S
pinf[j][k]	Infiltration parameter K for soil type J K = 1: hydraulic Conductivity; 2: capillary suction head; 3: soil moisture deficit
pman[l]	Manning's roughness coefficient corresponding to L land use class
project_file	Name of the CASC2D-SED project file (entered as the only argument of casc2dsed.exe)
q[j]	Outflow value at each user defined gage site J [m^3/s]
qout	Flow discharge [m^3/s] at the outlet cell at every time step (= qoutov + qoutch)
qoutov	Flow discharge [m^3/s] at the outlet cell at every time step coming only from the overland
qpeak	Peak flow discharge [m^3/s] at outlet
qsed[g]	Sediment outflow [m^3/s] at each user defined gage site G
rain_file_fptr	File containing the precipitation data [in/hr] (only if IRAIN = 1)
raincount	Counter. When RAINCOUNT = NREAD, rainfall rates at gages will be read from the rainfall file.
ret[j][k]	Interception depth [m] at cell (j,k)
retention[l]	Interception depth [mm] of landuse L (L = 1… NMAN)
rint	Rainfall intensity [m/s] (after conversion from input data)
rname	Prefix of the time series ASCII grids containing the rainfall rates in the basin [in/h]
rrg[g]	Rainfall intensities [m/s] (entered as in/h) at recorded gage G (G=1...NRG) (only if IRAIN = 1)
rtot[j][k]	Total rainfall depth [m] at cell (j,k)
sdep[j][k]	Storage depth [m] at cell (j,k)
sdep_file_fptr	File containing the storage deposition values at each cell in the watershed [m]
sed_out[l]	Size fraction L volume [m^3] that has left the watershed during the simulation
sed_out_file_fptr	File containing the sedigraph at outlet and, if selected, at internal locations [m^3/s]
SedConc[i][j][k]	Size fraction I sediment concentration [mg/l] at cell (j,k) at a given time step
SedFlux[i][j][k]	Size fraction I sediment flux [m^3/s] out of cell (j,k) at a given time step
SedGridsPath	Path in computer where to save the output sediment time series grids
shap_file_fptr	File containing the watershed and channel network definition at each cell in the watershed

Variable Name	Description and units
soil_file_fptr	File containing the soil type index at each cell in the watershed
sout	Outlet channel bed slope [m/m]
sovout	Slope of the outlet cell [m/m]
startTime	Simulation starting time
Summ_file	
Summ_file_fptr	
sus_ch[i]	Size fraction I total suspended sediment volume [m^3] in the channel cells
sus_ov[i]	Size fraction I total suspended sediment volume[m^3] in the overland cells
SusSed[i][j][k]	Suspended volume [m^3] of size fraction I at cell (j,k)
t_DepSed[j][k]	Total deposited sediment [m^3] at cell (j,k)
t_NetEros[j][k]	Total net erosion [mm] at cell (j,k)
t_SedConc[j][k]	Total sediment concentration [mg/l] at cell (j,k)
t_SedFlux[j][k]	Total sediment flux [m^3/s] going out of cell (j,k)
t_SusSed[j][k]	Total suspended sediment [m^3] at cell (j,k)
tot_ch[i]	Fraction I total eroded material (suspension + deposited) remaining in the channel cells at the end of the simulation $= \sum\sum$ (SusSed[I][J][K] + DepSed[I][J][K]) if (j,k) is a channel cell
tot_eroded [i]	Fraction I gross erosion [m^3] during the simulation time $= \sum\sum$ ErodSed[I][J][K]
tot_ov[i]	Fraction I total eroded material (suspension + deposited) remaining on the overland cells at the end of the simulation $= \sum\sum$ SusSed[I][J][K] + DepSed[I][J][K]) if (j,k) is an overland cell
tpeak	Time to peak flow at the outlet [min]
unitsQ	Discharge units index: 1:m^3/s; 2: cfs; 3: mm/h
unitsQs	Sediment discharge units index: 1: m^3/s; 2: tons/ha/day
vin	Total volume of rainfall minus interception [m^3]
vinf[j][k]	Infiltration depth [m] at cell (j,k)
vinfname	Prefix of the time series ASCII grids containing the accumulated infiltration depth [mm] in the basin
vinftot	Total volume [m^3] of infiltrated water

Variable Name	Description and units
vout	Total sediment outflow volume [m^3] at outlet (= qout * dt)
ws[l]	Fall velocity corresponding to medium grain size of fraction L at 20 degrees C
xllcorner	Lower left corner X-coordinate of the grid maps
xrg[j]	X-coordinate of rainfall gage J (J = 1...NRG) (only if IRAIN = 1)
yllcorner	Lower left corner Y-coordinate of the grid maps
yrg[j]	Y-coordinate of rainfall gage J (J = 1...NRG)(only if IRAIN = 1)

```
                    /************************/
                    /*          all.h          */
                    /************************/

                    /************************/
                    /*      INCLUDE FILES      */
                    /************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <time.h>
#include <malloc.h>


                /********************************/
                /*      FUNCTION PROTOTYPES        */
                /********************************/

extern void ReadInputFiles();

extern void  ReadControlFile(char control_file[]);

extern void ReadProjectFile();

extern void ReadGrids();

extern void ReadGridHeaders();

extern void GridsMemAlloc();

extern void ReadInputGrids();

extern void ReadChannFile();

extern void InitializeVars();

extern void ChannToplg();

extern void CompInitialVol();

extern void OvrlDepth( );

extern void intercept(int j, int k);

extern void rain(int j,int k);

extern void infilt(int j, int k);

extern void Settling();

extern void OvrlRout();

extern void ovrl( int j,int k, int jj, int kk);

extern void RoutSedOvrl(float dqq, float sf,int j,int k,
                                        int jj,int kk);

extern void ChannDepth();

extern float newChnDepth(float wch,float dch,float sfactor,
                    int j, int k,float addedVolume);
```

53

```
extern void ChannRout();

extern void chnchn(int ic,int l);

float FlowVelocity(float hchan,float wch,float dch,
          float stordep,float mannn,float sf);

float chnDischarge(float hchan,float wch,float dch,
        float stordep,float rmanch,float a,float sf,float sfactor);

extern void RoutSedChn(float dq,float sf,int jfrom,int kfrom,
                       int jto,int kto,float wch,float hchan);

extern float KRcap(float KRcoeff,float discharge,float width,
                   float sf, int jfrom, int kfrom);

extern float EHcap(int SizeFr,float width,float depth,
                   float sf,float discharge);

extern void TransferSed(int SizeFr,float **Source[],
                        float VolToTransfer,
                        int jfrom,int kfrom,int jto,int kto);

extern void RoutOutlet();

extern void RoutSedOut();

extern void WriteOutflow();

extern void SedStats();

extern void WriteGrids(int iter);

extern void write2dTS(char *name,
                float **ParamValue, float UnitsConv);

extern void write3dTS(char *name, int fraction,
                   float **ParamValue[], float UnitsConv);

extern void Reset();

extern void CompFinalVol();

extern void  SedVolumes();

extern void WriteSummFlow();

extern void WriteSummSed();

extern void WriteErrorFreeMem(int row, int col,
                          float OldDepth, float NewDepth);

extern void MemFree();

extern void  RunTime(clock_t finish);

extern double MAX(double one,double theOther);

extern double MIN(double one,double theOther);

void *vMAlloc(int size, int len);
void *mMAlloc(int size, int row, int col);
```

```c
void *tMAlloc(int size, int ht, int wh, int dh);
void vfree(void *v);
void mfree(void *m);
void tfree(void *t);

extern struct gstats gridstats(float **inputgrid);

                /*********************************/
                /*        GLOBAL CONSTANTS       */
                /*********************************/

#define g 9.81      /* Gravitational acceleration [m2/s]          */
#define KRov 58390  /* Kilinc Constant in the overland transp.eq. */
#define G 2.65      /* Specific gravity [adimensional]            */
#define HARD 470    /* Water hardness                             */


                /*********************************/
                /*        GLOBAL VARIABLES       */
                /*********************************/

extern char
      Summ_file[80],Project_file[80], header[300],
      rname[80],vinfname[80],  dname[80],SedGridsPath[100];

extern int
      nrows,ncols,jout,kout,ncells,nman,nsoil,
      indexsdep,indexbdepth,indexinf,indexeros,indexchan,
      nchan_link,nchan_node[100],maxlinks,maxnodes,***ichn,
      iter,niter,nitrn,nprn,nplt,irain,nrg,nread,
      ifcount,icount,ipcount,
      *iSedConc,*iSusSed,*iDepSed,*iSedFlux,*iNetEros,
      NumFracts,indexdis,ndis,**iq,unitsQ,indexsed,nsed,**ised,unitsQs,
      nodatavalue, **link,**node,**isoil, **iman, **ishp;

extern float
      dx,dt,sout,sovout,
      raincount,crain,*xrg,*yrg,*areaQ, *areaQs,
      xllcorner, yllcorner,*pman, *cfactor, *pfactor, *retention,
      elconv,**e,**ec,**ret,**sdep,**h0,**vinf,**pinf,*kfactor,**pfraction,
      vin,vout,vinftot,
      init_ov_vol,init_ch_vol,final_ch_vol,final_ov_vol,
      ***chp,**h,**hch,*rrg,**rint,**rtot,
      **dqov,**dqch,qout,qoutov,qpeak,tpeak,*q,
      *qsed,***SedConc,**t_SedConc,***SusSed, **t_SusSed,
      ***DepSed,**t_DepSed,***ErodSed,***SedFlux,**t_SedFlux,
      **t_NetEros, **CCU,**maxCCU,
      *tot_eroded,*sus_ov,*dep_ov,*tot_ov,
      *sus_ch,*dep_ch,*tot_ch,*sed_out;

extern double
      amaxdepth,amindepth,amaxcdepth,amincdepth, amaxvinf,aminvinf,
      amaxrain,
      aminrain,amaxSusCh,amaxSusOv, amaxSusCch, amaxSusCov,
      *ds,*ws,elapsedTime;

struct gstats {
    double min;
    double max;
    float mean;
    float stdev;};

clock_t startTime;
```

```
/* Input ASCII grids                                              */
extern FILE *shap_file_fptr;
extern FILE *sdep_file_fptr;
extern FILE *bdepth_file_fptr;
extern FILE *elev_file_fptr;
extern FILE *soil_file_fptr;
extern FILE *node_file_fptr;
extern FILE *link_file_fptr;
extern FILE *iman_file_fptr;
/* Input text files                                               */
extern FILE *rain_file_fptr;
extern FILE *chn_file_fptr;
/* Output text files                                              */
extern FILE *Summ_file_fptr;
extern FILE *dis_out_file_fptr;
extern FILE *sed_out_file_fptr;
extern FILE *debug;
```

```
                      /*************************/
                      /*       casc2dsed.c     */
                      /*************************/


      #include "all.h"

             /*********************************/
             /*  GLOBAL VARIABLES DECLARATION */
             /*********************************/

      char   Summ_file[80],Project_file[80], header[300],
             rname[80],vinfname[80],  dname[80],SedGridsPath[100];

      int    nrows,ncols,jout,kout,ncells,nman,nsoil,
             indexsdep,indexbdepth,indexinf,indexeros,indexchan,
             nchan_node[100],maxlinks,maxnodes,***ichn,
             iter,niter,nitrn,nprn,nplt,irain,nrg,nread,
             ifcount,icount,ipcount,
             *iSedConc,*iSusSed,*iDepSed,*iSedFlux,*iNetEros,
             NumFracts,indexdis,ndis,**iq,unitsQ,indexsed,nsed,**ised,unitsQs,
             nodatavalue, **link,**node,**isoil, **iman, **ishp;

      float  dx,dt,sout,sovout,
             raincount,crain,*xrg,*yrg,*areaQ, *areaQs,
             xllcorner, yllcorner,*pman, *cfactor, *pfactor, *retention,
             elconv,**e,**ec,**ret,**sdep,**h0,**vinf,**pinf,*kfactor,**pfraction,
             vin,vout,vinftot,
             init_ov_vol,init_ch_vol,final_ch_vol,final_ov_vol,
             ***chp,**h,**hch,*rrg,**rint,**rtot,
             **dqov,**dqch,qout,qoutov,qpeak,tpeak,*q,
             *qsed,***SedConc,**t_SedConc,***SusSed, **t_SusSed,
             ***DepSed,**t_DepSed,***ErodSed,***SedFlux,**t_SedFlux,
             **t_NetEros, **CCU,**maxCCU,
             *tot_eroded,*sus_ov,*dep_ov,*tot_ov,
             *sus_ch,*dep_ch,*tot_ch,*sed_out;

      double amaxdepth,amindepth,amaxcdepth,amincdepth, amaxvinf,aminvinf,
             amaxrain,
             aminrain,amaxSusCh,amaxSusOv, amaxSusCch, amaxSusCov,
             *ds,*ws,elapsedTime;

      clock_t startTime;


      /* Input ASCII grids                                          */
      FILE *shap_file_fptr = NULL;
      FILE *sdep_file_fptr = NULL;
      FILE *bdepth_file_fptr = NULL;
      FILE *elev_file_fptr = NULL;
      FILE *soil_file_fptr = NULL;
      FILE *node_file_fptr = NULL;
      FILE *link_file_fptr = NULL;
      FILE *iman_file_fptr = NULL;
      /* Input text files                                           */
      FILE *rain_file_fptr = NULL;
      FILE *chn_file_fptr = NULL;
      /* Output text files                                          */
      FILE *Summ_file_fptr = NULL;
      FILE *dis_out_file_fptr = NULL;
      FILE *sed_out_file_fptr = NULL;
      FILE *debug = NULL;
```

```c
      /******************************/
      /*   CASC2D_SED MAIN PROGRAM   */
      /******************************/

int main(argc,argv)
  int argc;
  char *argv[];
  {
    /*int i,j,k;*/
      /********************************************/
      /*          Read and initialize data        */
      /********************************************/

    /* Time CASC2D running time                                   */
    startTime = clock();

    /* Define the Project_file global variable                    */
    strcpy(Project_file,argv[1]);

    /* Reading input files                                        */
    ReadInputFiles();

    /* Initializing Variables                                     */
    InitializeVars();

    /* Computing Initial Storage Volume for Overland and Channels  */
    if(indexsdep == 1 || indexbdepth == 1) CompInitialVol();

      /********************************************/
      /*              Event time loop             */
      /********************************************/

    for(iter=1;iter<=niter;iter++)
    {

      /* Updating overland and channels water depth               */
      OvrlDepth( );
      if(indexchan == 1 && iter > 1)
        ChannDepth();

      /* Overland and channel flow routing                        */
      OvrlRout();
      if(indexchan == 1) ChannRout();

      /* Flow routing at the outlet                               */
      RoutOutlet();

      if (indexeros == 1)
      {
        /* Sediment settling for this water depth and time dt      */
        Settling();
        /* Sediment routing at the outlet                          */
        if(qout != 0) RoutSedOut();
        /* Net erosion, suspended/deposited/scoured volumes,       */
        /* sediment concentrations and absolute max./min values    */
        SedStats();
      }

      /* Write outflow to screen and files                        */
      WriteOutflow();

      /* Write output grids                                       */
```

```
    WriteGrids(iter);

    /* Reset grids to zero                                          */
      if (indexeros == 1) Reset();

  } /* End of Time Loop */

      /*********************************************/
      /* Computes final volumes & writes results   */
      /*********************************************/

  /* Compute final water volumes on the overland and channels    */
  CompFinalVol();

  /* Writing flow summary file                                    */
  WriteSummFlow();

  if(indexeros == 1)
  { /* Compute final sed. volumes on the overland and channels  */
    SedVolumes();
    /* Write sediment summary file                               */
    WriteSummSed();
  }

  /* Deallocate memory */
  MemFree();

  /* Computes casc2d running time                                 */
  RunTime(clock());

} /* End of Main Program */
```

```
                      /**************************/
                      /*      channdepth.c      */
                      /**************************/

#include "all.h"

extern void ChannDepth()
{
  int ic,nn,j,k,jj,r,c,NumSteps,NS;
  float RemTime,dt_chan,
        **depth, wch, dch,sfactor, sdep_ov,inflowVol,vol_ov_in;

  /* Unless the channel becomes unstable the channel time step   */
  /* (DT_CHAN) equals the simulation's time step (DT)            */
  dt_chan = dt;
  NumSteps = 1;

  /* The DEPTH matrix will hold the channel depths in the last dt */
  depth = (float**)mMAlloc(sizeof(float),nrows,ncols);

  /* We come back to this point if a negative depth is found in a */
  /* channel cell.  We repeat the whole process with 1/2 DT_CHAN  */
  NewTry:

  RemTime = dt;  /* Initialize or reset the remaining time to DT  */

  /* Store channel depth in last time step (hch) in new matrix    */
  for(r=1;r<=nrows;r++)
   for(c=1;c<=ncols;c++)
     depth[r][c] = hch[r][c];

  /* Loops for current DT through all the channel cells as many   */
  /* times as DT is divided into.  In the beggining: only once.   */
  /* DT/DT_CHAN: if any channel cell becomes unstable             */

  for(NS=1;NS<=NumSteps;NS++)
  {
    /* Takes care of rounding errors                              */
    if(NS == NumSteps) dt_chan = RemTime;

    /* Double FOR LOOP computes depth in all the channel cells    */

    for(ic=1;ic<=maxlinks;ic++) /* For each link, IC, in network  */
    {
     for(nn=1;nn<=nchan_node[ic];nn++) /* NN:node number in IC     */
     {
       wch = chp[ic][nn][2];     /* Channel width [m]             */
       dch = chp[ic][nn][3];     /* Channel depth [m]             */
       sfactor = chp[ic][nn][6]; /* Sinuosity factor [--]         */

       j = ichn[ic][nn][1];      /* Row number of node nn, link ic */
       k = ichn[ic][nn][2];      /* Col number of node nn, link ic */
       jj = ichn[ic][nn+1][1];   /* Row number of d/s node from nn */

       if(j > 0 && jj >= 0)
       {
         /* Find new channel depth after adding inflow volume     */
         inflowVol = dqch[j][k] * dt_chan;

         /* ..and the volume coming from the overland (vol_ov_in) */
         if(sdep[j][k] > dch)  sdep_ov = sdep[j][k] - dch;
         else                  sdep_ov = 0.0;
```

```
        vol_ov_in = 0;

        if(h[j][k] > sdep_ov)
          vol_ov_in = (h[j][k] - sdep_ov)*dx*dx*dt_chan/dt;

        /* Calculate the new channel depth after adding flows    */
        depth[j][k] = newChnDepth(wch,dch,sfactor,j,k,
                                  (inflowVol + vol_ov_in));

        /* TODO: in case that the water surface in the channel    */
        /* is greater than the water surface in the overland,     */
        /* then, the volume of water needs to be redistributed    */

        /* Negative Depth in the Channel                          */

        if (depth[j][k] < 0.0)
        {
          /* Unless the time step in the channel is really       */
            /* short, keep on cutting DT_CHAN in half             */
            if(dt_chan > 0.0001)
            {
              dt_chan = dt_chan / 2;

              /* Number of times that for the new DT_CHAN the    */
              /* depth in the channel network is updated so that */
              /* the total time equals the simulation's DT       */
              NumSteps = (int)(dt/dt_chan);

              /* Start all over again with the new DT_CHAN       */
              goto NewTry;
            }
            else  /* Channel depth < -1 mm and DT_CHAN < 0.0001   */
              /* Free up computer memory and exit execution      */
              /* The user should be advised to reduce DT         */
              WriteErrorFreeMem(j,k,hch[j][k],depth[j][k]);

        }

    }  /* end of : if(j > 0 && jj >= 0)                          */
   }  /* end of : for(nn=1;nn<=nchan_node[ic];nn++)              */
  } /* end of : for(ic=1;ic<=maxlinks;ic++)                      */

  /* Reduce the simulation time left for this time step, dt      */
  RemTime = RemTime - dt_chan;

} /* end of for(NS=1;NS<=NumSteps;NS++)                          */
```

```
for(ic=1;ic<=maxlinks;ic++) /* For each link, IC, in network  */
  {
   for(nn=1;nn<=nchan_node[ic];nn++) /* NN:node number in IC     */
   {
     j = ichn[ic][nn][1];      /* Row number of node nn, link ic */
     k = ichn[ic][nn][2];      /* Col number of node nn, link ic */
     jj = ichn[ic][nn+1][1];  /* Row number of d/s node from nn */

     if(j > 0 && jj >= 0)
     {
       dqch[j][k] = 0.0;  /* Reset channel cell flow to zero      */

       /* Reset overland depths                               */
       dch = chp[ic][nn][3];     /* Channel depth [m]           */
       if(sdep[j][k] > dch)  h[j][k] = sdep[j][k] - dch;
       else                  h[j][k] = 0.0;

       /* Determining the Min. and the Maximum Channel Depths   */
       amincdepth = MIN(amincdepth,hch[j][k]);
       amaxcdepth = MAX(amaxcdepth,hch[j][k]);
       /* Make global channel depth HCH equal to local DEPTH var  */
       hch[j][k] = depth[j][k];
     }
   }
  }

  mfree(depth);
}
```

```
                /******************************/
                /*        ChannToplg.c        */
                /******************************/


#include "all.h"

extern void ChannToplg()
{
  int maxelements;
  int downstream[50],upstream1[50],upstream2[50],
      ic,l,/*rr,cc,*/ll,rc,i,j,k, p,q,ro,co,outlink,outnode,r,c,row,col;
  float elev;

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}


  /* Determines LINK and NODE numbers of the outlet channel cell  */
  outlink = link[jout][kout];
  outnode = node[jout][kout];

  /* NOTE: in the ICHN matrix, for each of the links, there are    */
  /* a number of elements equal to the maximum number of nodes     */
  /* for all links, plus a last element being the -1 flag (see     */
  /* original CASC2D manual by Saghafian and Julien (1990)         */
  maxelements = maxnodes + 1;   /* Maximum number of nodes + 1       */

  /* Allocates memory for the ICHN matrix                          */
  ichn = (int***)tMAlloc(sizeof(int),maxlinks,maxelements,2);

  /* Initializing ICHN Array.                                      */
  for(i=1;i<=maxlinks;i++)
    for(k=1;k<=2;k++)
      for(j=1;j<=maxelements;j++)
        ichn[i][j][k] = 0;

  /* Determine row and column location for each node in each link  */
  /* and assign these values to the corresponding ICHN elements    */
  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(link[j][k] !=nodatavalue && node[j][k] !=nodatavalue)
      {
        /* Assumes numbering of links and nodes is correct         */
        ichn[link[j][k]][node[j][k]][1] = j;
        ichn[link[j][k]][node[j][k]][2] = k;
      }
    }
  }

  /* Initializing the downstream, upstream1 & upstream2 matrices   */
  /* NOTE that a  given channel link might have two upstream        */
  /* links but only one link downstream                            */
  for(i=1;i<=maxlinks;i++)
  {
    downstream[i] = 0;
    upstream1[i] = 0;
    upstream2[i] = 0;
  }
```

63

```
/* NOTE the next piece of code finds upstream and downstream    */
/* link numbers BUT there might be problems in case of two      */
/* channels running in parallel through contiguous rows or cols.*/
/* Right now, leave it as it is since it does not affect        */
/* computations.                                                */
for(i = 1; i <= maxlinks; i++)
{
  row = ichn[i][1][1];
  col = ichn[i][1][2];
  for (j = -1; j <= 1; j++)
  {
    for (k = -1; k <= 1; k++)
    {
      r = row + j;
      c = col + k;
      if(r>0 && r<=nrows && c>0 && c<=ncols)
      {
        l = link[r][c];
        if(l!=nodatavalue && l!=i)
        {
          if (upstream1[i] == 0)
          {
            upstream1[i] = l;
          }
          else
          {
            /* Upstream link may have two or more nodes next to */
            /* the 1st node of downstream link.Check that one   */
            /* of those nodes has not been already found        */
            if(upstream1[i]  != l) upstream2[i] = l;
          }
          downstream[l] = i;
        }
      }
    }
  }
}


/*Adding the downstream Row and Column to the end of every Link */

for(i=1;i<=maxlinks;i++)
{
  rc = 0;

  for(j=1;j<=maxelements;j++)
  {
    if(i != outlink && ichn[i][j][1] == 0 && rc == 0)
    {
      if(ichn[i][j-1][1] > 0)
      {
        ll = downstream[i];

        ichn[i][j][1] = ichn[ll][1][1];
        ichn[i][j][2] = ichn[ll][1][2];
      }
      rc = 1;
    }
  }
}
```

```c
/* Adding a -1 to the end of every link except for the outlet    */
/* link                                                          */

for(i=1;i<=maxlinks;i++)
{
  for(k=1;k<=2;k++)
  {
    for(j=1;j<=maxelements;j++)
    {
      if(i != outlink && ichn[i][j][k] == 0 && ichn[i][j-1][k] > 0)
        ichn[i][j][k] = -1;
    }
  }
}

/* Writing channel network info and topology to summary file   */

fprintf(Summ_file_fptr,"\nChannel Linkage Information\n");
fprintf(Summ_file_fptr,"--------------------------\n\n");

fprintf(Summ_file_fptr,"Maximum Number of Links = %ld \n",
                                        maxlinks);
fprintf(Summ_file_fptr,"Maximum Number of Nodes = %ld \n\n",
                                        maxnodes);
/*
for(i=1;i<=maxlinks;i++)
{
  for(k=1;k<=2;k++)
  {
    for(j=1;j<=maxelements;j++)
    {
        fprintf(Summ_file_fptr,"%5ld",ichn[i][j][k]);
    }
    fprintf(Summ_file_fptr,"\n");
  }
  fprintf(Summ_file_fptr,"\n");
}
*/

fprintf(Summ_file_fptr,
        " Link     Downstream    Upstream1    Upstream2\n");
fprintf(Summ_file_fptr,
        " ---------------------------------------------\n");

for(i=1;i<=maxlinks;i++)
{
  fprintf(Summ_file_fptr,"%4ld %10ld  %10ld  %10ld \n",
                  i,downstream[i],upstream1[i],upstream2[i]);

  printf("%4ld %10ld  %10ld  %10ld \n",
                  i,downstream[i],upstream1[i],upstream2[i]);
}
```

```
  /* Mask grid value equal to 2 if channel going through it          */
  for(ic=1;ic<=maxlinks;ic++)
  {
    for(l=1;l<=maxelements;l++)
    {
      j=ichn[ic][l][1];
      k=ichn[ic][l][2];

      if(j > 0 && indexchan == 1)
      {
       ishp[j][k] = 2;
      }
    }
  }

  for(p=-1; p<=1; p++)
  {
    for(q=-1; q<=1; q++)
    {
      ro = jout + p;
      co = kout + q;
      if(ro>0 && ro<=nrows && co>0 && co<=ncols && ro!=jout && co!=kout)
        if(ishp[ro][co] == 2 ) elev = e[ro][co];
    }
  }
  sout = (float)(fabs(elev - e[jout][kout] )/ dx);

  fclose(Summ_file_fptr);

}
```

```
                /*******************************/
                /*        ChannRout.c          */
                /*******************************/


#include "all.h"

        /*********************************/
        /*        FUNCT: ChannRout       */
        /*********************************/

extern void ChannRout()
{
  int ic,nn;

  for(ic=1;ic<=maxlinks;ic++)
  {
    for(nn=1;nn<=nchan_node[ic];nn++)
    {
      /*  Note :  When ichn[ic][nn+1][1] is less than zero,      */
      /*          then that  indicates that the channel          */
      /*          routing for the current link is complete.      */
      /*          There is no routing for the outlet cell        */
      /*          (this is done in RoutOutlet.c)                 */

      if(ichn[ic][nn+1][1] > 0)
      {
       chnchn(ic,nn);
      }

    } /* end For loop for the nodes in each link               */
  } /* end For loop for all the links in the network           */
}

        /*********************************/
        /*        FUNCT: chnchn          */
        /*********************************/

extern void chnchn(int ic,  /* Link number in channel network    */
                   int nn)  /* Node number in link number ic     */
{

  float a = 1.0;
  float wch,dch,sslope,rmanch,sfactor,so, sf,dhdx, L,
        dq, stordep, hchan,dq_ov,vel;

  int  j,k,jj,kk,jjj,iic, ijun, ill;

  /* Row and column of link IC and node NN                      */

  j = ichn[ic][nn][1];
  k = ichn[ic][nn][2];

  /* Row and column of downstream node from NN in link IC       */

  jj = ichn[ic][nn+1][1];
  kk = ichn[ic][nn+1][2];

  /* Note : JJJ is a check to see when the end of  a channel link */
  /*        has been reached. JJJ is located 2 nodes d/s from NN  */

  jjj = ichn[ic][nn+2][1];
```

```
/* Channel characteristics :                                    */

wch = chp[ic][nn][2];        /* width                           */
dch = chp[ic][nn][3];         /* depth                          */
sslope = chp[ic][nn][4];    /* side slope                       */
rmanch = chp[ic][nn][5];    /* manning's n                      */
sfactor = chp[ic][nn][6];    /* sinuosity factor                */
stordep = sdep[j][k];          /* Storage depth                 */
hchan = hch[j][k];             /* Channel water depth           */

/* Channel length equals grid size if channel runs N-S or E-W   */
 /* Otherwise, it equals SQRT(2) times the grid cell size        */
L = (float)(sqrt(pow((kk-k),2)+pow((jj-j),2))*dx);

/* Channel slope.                                               */
so = (ec[j][k] - dch - ec[jj][kk] + chp[ic][nn+1][3])/(L*sfactor);

dq_ov = 0.0;

/* If JJJ<0, then the end of the channel link has been reached. */
/* Slope is computed with last node of current link and first   */
/* node of following link                                       */

if(jjj < 0)
{
  for (iic=1;iic<=maxlinks;iic++)
  {
    if(jj == ichn[iic][1][1] && kk == ichn[iic][1][2])
    {
      so = (ec[j][k] - dch - ec[jj][kk] + chp[iic][1][3])/(L*sfactor);
      ijun = iic;
    }
  }
}

dhdx = (hch[jj][kk] - hch[j][k])/(L*sfactor);

sf = (float)(so - dhdx + 1e-30);

/* Nota de Jorge: Sf se deberia quedar con el mismo signo    */
if(fabs(sf) < 1e-20) sf = (float)(1e-20);

if (sf < 0.0)
{
  a = (float)(-1.0*a);

  if (jjj < 0)
  {
    /* Take channel chars. of the 1st node of downstream link   */

    wch = chp[ijun][1][2];
    dch = chp[ijun][1][3];
    sslope = chp[ijun][1][4];
    rmanch = chp[ijun][1][5];
    sfactor = chp[ijun][1][6];
  }
```

```
        else
            {
              /*Take channel chars. of the next node within current link  */

              wch = chp[ic][nn+1][2];
              dch = chp[ic][nn+1][3];
              sslope = chp[ic][nn+1][4];
              rmanch = chp[ic][nn+1][5];
              sfactor = chp[ic][nn+1][6];
            }

           stordep = sdep[jj][kk];
           hchan = hch[jj][kk];
        }

      /* Determining flow velocity (m/s)                                */

      vel = FlowVelocity(hchan,wch,dch,stordep,rmanch,sf);

      /* Determining discharge                                         */

      dq = chnDischarge(hchan,wch,dch,stordep,rmanch,a,sf,sfactor);

      /* Transfer flow from cell (j,k) to (jj,kk)                      */

      dqch[j][k] = dqch[j][k] - dq;
      dqch[jj][kk] = dqch[jj][kk] + dq;

      /* If (j,k) is a printout flow discharge internal location       */

      for (ill=1;ill<=ndis;ill++)
      {
        if(j == iq[ill][1] && k == iq[ill][2]) q[ill] = dq;
      }

      /* Channel sediment routing                                      */

      if(indexeros == 1 && dq != 0.0 && hchan !=0)
      {
        if(a >= 0.0)
        {
          RoutSedChn(dq,sf,j,k,jj,kk,wch,hchan);
        }
        else
        {
          RoutSedChn(-dq,-sf,jj,kk,j,k,wch,hchan);
        }
      }

}     /* End of CHNCHN */
```

```
                    /*******************************/
                    /*       CompVolumes.c         */
                    /*******************************/


#include "all.h"

            /*********************************/
            /* COMPUTE INITIAL STORAGE VOLUME */
            /*********************************/

extern void CompInitialVol()

{
  int j,k;
  float wch, dch, sslope, tw_ch, area_ch, vol_ch;

  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        if(indexchan == 1 && link[j][k] > 0)
        {
          wch = chp[link[j][k]][node[j][k]][2];
          dch = chp[link[j][k]][node[j][k]][3];
          sslope = chp[link[j][k]][node[j][k]][4];

          if(hch[j][k] < dch)
          {
           tw_ch = wch + 2*hch[j][k]*sslope;
           area_ch = (tw_ch + wch)/2 * hch[j][k];
           vol_ch = area_ch * dx * chp[link[j][k]][node[j][k]][6];
          }
          else
          {
           tw_ch = wch + 2*dch*sslope;
           area_ch = (tw_ch + wch)/2 * dch;
           area_ch = area_ch + (hch[j][k] - dch)*tw_ch;
           vol_ch = area_ch * dx * chp[link[j][k]][node[j][k]][6];
          }

          init_ch_vol = init_ch_vol + vol_ch;
          tw_ch = wch + 2*dch*sslope;
          init_ov_vol = init_ov_vol + (dx-tw_ch)*dx*h[j][k];
        }
        else
        {
         init_ov_vol = init_ov_vol + h[j][k]*dx*dx;
        }
      }
    }
  }
}
```

```
                    /*****************************/
                    /* COMPUTE FINAL STORAGE VOLUME */
                    /*****************************/
extern void CompFinalVol()
{
  int j,k;
  float wch, dch, sslope, tw_ch, area_ch, vol_ch;

  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        if(indexchan == 1 && link[j][k] > 0)
        {
          wch = chp[link[j][k]][node[j][k]][2];
          dch = chp[link[j][k]][node[j][k]][3];
          sslope = chp[link[j][k]][node[j][k]][4];

          if(hch[j][k] < dch)
          {
           tw_ch = wch + 2*hch[j][k]*sslope;
           area_ch = (tw_ch + wch)/2 * hch[j][k];
           vol_ch = area_ch * dx * chp[link[j][k]][node[j][k]][6];
          }
          else
          {
           tw_ch = wch + 2*dch*sslope;
           area_ch = (tw_ch + wch)/2 * dch;
           area_ch = area_ch + (hch[j][k] - dch)*tw_ch;
           vol_ch = area_ch * dx * chp[link[j][k]][node[j][k]][6];
          }

          final_ch_vol = final_ch_vol + vol_ch;
          tw_ch = wch + 2*dch*sslope;
          final_ov_vol = final_ov_vol + (dx-tw_ch)*dx*h[j][k];
        }
        else
        {
         final_ov_vol = final_ov_vol + h[j][k]*dx*dx;
        }
      }
    }
  }
}
```

```
                    /*******************************/
                    /*        Fcts_chann.c        */
                    /*******************************/

#include "all.h"

        /**********************************/
        /*         FUNCT: newChnDepth     */
        /**********************************/

extern float newChnDepth(float wch,float dch,float sfactor,
                         int j, int k,float addedVolume)
{
  float area_ch,vol_ch,area_init,vol_init,
    vol_final,newdepth;

  /* Channel area and volume                                    */

  area_ch = wch * dch;
  vol_ch = area_ch * dx * sfactor;

  /* Calculates initial area and volume                         */

  if(hch[j][k] <= dch)
   area_init =  wch * hch[j][k];
  else
   area_init = (hch[j][k] - dch) * dx + area_ch;

  vol_init = area_init * dx * sfactor;

  /* After adding new volume calculates volume                  */

  vol_final = vol_init + addedVolume;

  /* ... and depth corresponding to the final volume            */

  if(vol_final > vol_ch)
   newdepth = dch + (vol_final - vol_ch) / (dx*dx*sfactor);
  else
   newdepth = vol_final / (wch*dx*sfactor);

  return(newdepth);

}
```

```c
/********************************/
/*       FUNCT: FlowVelocity    */
/********************************/

/* Calculates the flow velocity in m/s                          */

float FlowVelocity(float hchan,float wch,float dch,
        float stordep,float mannn,float sf)

/*        _ _ _    _ _    _                                      */
/*       | ~ |___\/____|_ _ _/                                   */
/*  dch  |   |           |    | hchan                           */
/*       |   |_____|_ _ _/                                 */
/*       |_ _|_____|_ _ _| stordep                        */
/*          <--- wch --->     /                                 */

{
  float area, wp, Rh, vel;

  /* Calculates flow area and wetted perimeter (wp)           */

  if(hchan <= dch) /* Flow depth less than channel depth       */
  {
     if(hchan<stordep) /* No flow occurs in case in which the    */
     {                 /* channel depth is less than the storage */
       area = 0;       /* depth.  The flow area is thus, zero    */
       wp = 1; /* simply, initiziale this value so that Rh = 0   */
     }
     else
     {
       area = wch * (hchan-stordep);
       wp = (float)(wch + 2 * (hchan-stordep));
     }
  }
  else  /* Flow depth greater than channel depth              */
  {
     area = wch * (dch - stordep) + dx * (hchan-dch);
     wp = (float)(wch+2*(dch-stordep) + 2*(dx-wch) + 2*(hchan-dch));
  }

  Rh = area / wp;     /* Channel hydraulic radious (m)          */

  /* Flow velocity using Manning's flow resistance equation      */

  vel = (float)(sqrt(fabs(sf))/mannn* pow(Rh,0.6667));

  return(vel);

}
```

73

```
      /*********************************/
      /*        FUNCT: chnDischarge    */
      /*********************************/

/* Calculates the outflow in m3/s from a cell */

float chnDischarge(float hchan,float wch,float dch,
         float stordep,float rmanch,float a,float sf,float sfactor)
{
  float area, wp, dQ, vol_ch_avail;

  /* Calculates flow area and wetted perimeter                 */

  if(hchan <= dch) /* Flow depth less than channel depth        */
  {
     if(hchan<stordep) /* No flow occurs in case in which the    */
     {                 /* channel depth is less than the storage */
       area = 0;       /* depth.  The flow area is thus, zero    */
       wp = 1; /* simply, initiziale this value so that Rh = 0   */
     }
     else
     {
       area = wch * (hchan-stordep);
       wp = (float)(wch + 2 * (hchan-stordep));
     }
  }
  else  /* Flow depth greater than channel depth                */
  {
     area = wch * (dch - stordep) + dx * (hchan-dch);
     wp = (float)(wch+2*(dch-stordep) + 2*(dx-wch) + 2*(hchan-dch));
  }

  dQ = (float)(a*(sqrt(fabs(sf))/rmanch)*
                       (pow(area,1.6667))/(pow(wp,0.6667)));

  /* Limit the outflow by availability                         */

  vol_ch_avail = area * dx * sfactor;

  if(dQ*dt > vol_ch_avail) dQ = vol_ch_avail/dt;

  return(dQ);

}
```

```
                      /******************************/
                      /*       Fcts_Memory.c         */
                      /******************************/

/*  A Package of Dynamical Memory Allocation-Related Functions   */
/*                                                               */
/* NAMING CONVENTION:                                            */
/*                                                               */
/* VECTOR: is a 1-D array. Ex. v = {1 2 3}                       */
/*   v     Each element has a size = sizeof(int)                 */
/*         dim1 = 3                                              */
/*         Ex. v[1] = 2                                          */
/*                                                               */
/* MATRIX: is a 2-D array. Ex.     |1. 2.|                       */
/*   m                        m =  |3. 4.|                       */
/*                                 |5. 6.|                       */
/*         Each element has a size = sizeof(float)               */
/*         dim1 = 3                                              */
/*         dim2 = 2                                              */
/*         Ex. m[2][0] = 5.                                      */
/*                                                               */
/* TENSOR: is a 3-D array.  Ex.                                  */
/*    t                                                          */
/*           |a b c d|        |e f g h|       |i j k l|          */
/*         t[0]=|m n o p|  t[1]=|q r s t|  t[2]=|u v w y|        */
/*                                                               */
/*         Each element has a size = sizeof(char)               */
/*         dim1 = 3                                              */
/*         dim2 = 2                                              */
/*         dim3 = 4                                              */
/*         Ex. t[1][1][3] = t                                    */
/*                                                               */
/* USAGE:                                                        */
/*    Declare variable:  Ex. int  **isoil (2-D array)           */
/*    Allocate memory:                                           */
/*         Ex. isoil = (int**)mMAlloc(sizeof(int),nrows,ncols)  */
/*    Deallocate memory:                                         */
/*         Ex. mfree(isoil)                                      */
/*                                                               */
/* NOTE 1:                                                       */
/* In this program, the element 0 of an array is ignored for    */
/* clarity purposes.  We normally think of the first element    */
/* in a vector as element 1, not element 0.                     */
/* Ignoring the Zero-th element and starting in element 1 means */
/* that we have to allocate memory for an extra element         */
/*                                                               */
/* NOTE 2:                                                       */
/* By defining SIZE as an argument when allocating memory, gives */
/* us the freedom of allocating memory for any variable type    */
/*                                                               */
/* Rosalia Rojas -- Nov. 11, 2002                                */

#include "all.h"
```

```
          /*********************************/
          /*         FUNCT:  vMAlloc        */
          /*********************************/
/* (v)ector (M)emory (Alloc)ation                              */
/* Dynamical memory allocator for a vector or 1-D array        */

void *vMAlloc(int size, /* size of an element         */
              int dim1 /* Number of elements in vector */)
{

  char *v;

  if (!(v = (char *) malloc((size_t) (dim1+1) * size))) return NULL;

  return v;
}



          /*********************************/
          /*         FUNCT:  mMAlloc        */
          /*********************************/
/* (m)atrix (M)emory (Alloc)ation                              */
/* Dynamical memory allocator for a matrix or 2-D array        */


void *mMAlloc(int size, /* size of an element         */
              int dim1,  /* Number of rows in matrix  */
              int dim2 /* Number of columns in matrix */)
{
  char **m;
  int i;

  if (!(m = (char **) malloc((size_t) (dim1+1) * sizeof(char *)))) return NULL;
  if (!(m[0] = (char *) malloc((size_t) (dim1+1) * (dim2+1) * size)))
    {
      free(m);
      return NULL;
    }

  for (i = 1; i <= dim1; i++) m[i] = m[i - 1] + dim2 * size;

  return m;
}
```

```
                /*********************************/
                /*        FUNCT:  tMalloc        */
                /*********************************/
/* (t)ensor (M)emory (Alloc)ation                              */
/* Dynamical memory allocator for a tensor or 3-D array        */

void *tMAlloc(int size, /* size of an element        */
              int dim1,  /* First dimension in matrix  */
              int dim2,  /* Second dimension in matrix */
              int dim3 /* Third dimension in matrix    */)
{
  char ***t;
  int i, j;

  if (!(t = (char ***) malloc((size_t) (dim1+1) * sizeof(char **)))) return
NULL;
  if (!(t[0] = (char **) malloc((size_t) (dim1+1) * (dim2+1) * sizeof(char
*))))
     {
       free(t);
       return NULL;
     }
  if (!(t[0][0] = (char *) malloc((size_t) (dim1+1) * (dim2+1) * (dim3+1) *
size)))
     {
       free(t);
       free(t[0]);
       return NULL;
     }

  for (j = 1; j <= dim2; j++)
    t[0][j] = t[0][j - 1] + dim3 * size;

  for (i = 1; i <= dim1; i++)
    {
      t[i] = t[i - 1] + dim2;

      for (j = 0; j <= dim2; j++)
        t[i][j] = t[i - 1][j] + dim2 * dim3 * size;
    }

  return t;
}
```

```
                  /********************************/
                  /*          FUNCT:  vfree        */
                  /********************************/

/* 'Free()' for vector()                                        */

void vfree(void *v)
{
  char *ptr = v;

  free(ptr);

  return;
}

                  /********************************/
                  /*          FUNCT: mfree         */
                  /********************************/
/*  'Free()' for matrix()                                       */

void mfree(void *m)
{
  char **ptr = m;

  free(ptr[0]);
  free(ptr);

  return;
}

                  /********************************/
                  /*          FUNCT: tfree         */
                  /********************************/
/* 'Free()' for tensor()                                        */

void tfree(void *t)
{
  char ***ptr = t;

  free(ptr[0][0]);
  free(ptr[0]);
  free(ptr);

  return;
}
```

```
/*****************************/
/*          Fcts_misc.c      */
/*****************************/

#include "all.h"

    /********************************/
    /*              FUNCT: max      */
    /********************************/

/* Finds the maximum between two floating numbers */

extern double MAX(double one,double theOther)

{
  double maximum;

  if (one >= theOther)
    maximum = one;
  else
    maximum = theOther;

  return(maximum);
}

    /********************************/
    /*              FUNCT: min      */
    /********************************/

/* Finds the minimum between two floating numbers */

extern double MIN(double one,double theOther)

{
  double minimum;

  if (one <= theOther)
    minimum = one;
  else
    minimum = theOther;

  return(minimum);
}
```

```
                    /*****************************/
                    /*        Fcts_sed.c         */
                    /*****************************/


#include "all.h"


        /********************************/
        /*          FUNCT: EHcap        */
        /********************************/


/* Determines the sediment transport capacity based on the        */
/* Engelund and Hansen transport equation. Result is in m3        */

extern float EHcap(int SizeFr,float width,float depth,
                   float sf,float discharge)
{
  float Cw, Rh,V, EHcapacity;

  /* Hydraulic radius                                             */
  Rh = (width*depth)/(width + 2 * depth);

  /* Flow velocity                                                */
  V = (float)(discharge/(width*depth));

  /* Concentration by weight by size fraction                     */
  Cw = (float)(0.05*(G/(G-1)) *
              fabs(V*sf)/sqrt((G-1)*g*ds[SizeFr]) *
              sqrt(Rh*fabs(sf)/((G-1)*ds[SizeFr]) ) ) ;

  /* Engelund and Hansen transport capacity  (in m3)             */
  EHcapacity = (float)(discharge * Cw / 2.65)*dt ;

  return(EHcapacity);

}
        /********************************/
        /*          FUNCT: KRcap        */
        /********************************/


/* Determines the sediment transport capacity based on the        */
/* Kilinc and Richardson transport equation. Result is in m3      */

extern float KRcap(float KRcoeff,float discharge,float width,
                   float sf, int jfrom, int kfrom)
{
  int SoilType, LandUse;
  float KRcapacity, UnitDischarge,Cusle,Kusle,Pusle;

  UnitDischarge = (float)(fabs(discharge/width));

  SoilType = isoil[jfrom][kfrom];
  LandUse = iman[jfrom][kfrom];

  Cusle = cfactor[LandUse];
  Kusle = kfactor[SoilType];
  Pusle = pfactor[LandUse];
```

```c
  KRcapacity = (float)(KRcoeff * pow(fabs(sf),1.66) *
                pow(UnitDischarge,2.035) * Cusle * Kusle * Pusle
                * width * dt);

  return(KRcapacity);

}


      /***********************************/
      /*         FUNCT: TransferSed      */
      /***********************************/

/* Transfers a certain volume of sediment, VolToTransfer, from    */
/* an outgoing cell (jfrom,kfrom) to receiving cell (jto,kto).The */
/* source, can be the suspended,deposited or parent material in    */
/* outgoing cell.  The transferred volume of sed. will be added    */
/* in all cases onto the suspended portion of the receiving cell. */
/* If the outgoing cell is the outlet cell, the VolToTransfer is  */
/* not added into a receiving cell but to the variable that holds */
/* the total volume of sediment leaving the basin by size fraction*/

extern void TransferSed(int SizeFr,float ***Source,
                        float VolToTransfer,
                        int jfrom,int kfrom,int jto,int kto)
{
  /* For each size fraction, substracts the sediment volume that  */
  /* has been transported from the suspended, deposited or parent */
  /* material of the outgoing cell                                */
  Source[SizeFr][jfrom][kfrom] -= VolToTransfer;


  if(jfrom==jout && kfrom==kout)
  {
    /* If outgoing cell is the outlet cell, keeps track by size   */
    /* fraction of the volume of sediment that left the basin     */
    sed_out[SizeFr] += VolToTransfer;
  }
  else
  {
    /* if outgoing cell is not the outlet cell, it adds by size   */
    /* fraction the transported sediment to the suspended portion */
    /* of the receiving cell                                      */
    SusSed[SizeFr][jto][kto] += VolToTransfer;
  }
}
```

```
            /*******************************/
            /*          Fcts_stats.c        */
            /*******************************/


#include "all.h"


      /********************************/
      /*          FUNCT: gridstats    */
      /********************************/


/* Determines the mean and standard deviation of a grid            */

extern struct gstats gridstats(float **inputgrid)

{
  struct gstats statistics;
  float s;
  int j,k;

  statistics.min = 10E300;
  statistics.max = -10E300;
  statistics.mean = 0;
  s = 0;

  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        statistics.min = MIN(statistics.min,inputgrid[j][k]);
        statistics.max = MAX(statistics.max,inputgrid[j][k]);
        statistics.mean += inputgrid[j][k] / ncells;
      }
    }
  }
  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        s +=
        (float)(pow(inputgrid[j][k]-statistics.mean,2)/(ncells-1));
      }
    }
  }

  statistics.stdev = (float)(sqrt(s));

  return(statistics);
}
```

```
/*******************************/
/*          infilt.c           */
/*******************************/


#include "all.h"

extern void infilt(int j, int k)
{

  int SoilType;

  float hydcon, cs, smd, p1, p2, rinf;

  SoilType = isoil[j][k];
   hydcon = pinf[SoilType][1];
  cs = pinf[SoilType][2];
  smd = pinf[SoilType][3];

  /* Infiltration rate                                      */

  p1 = (float)(hydcon*dt - 2.0*vinf[j][k]);
  p2 = hydcon*(vinf[j][k] + cs*smd);
  rinf = (float)((p1 + sqrt(pow(p1,2.0) + 8.0*p2*dt))/(2.0*dt));

  /* When the Rate of Infiltration is Greater Than HOV/DT,  */
  /* then all the water on the overland cell is assumed to  */
  /* be  infiltrated and the overland depth is reduced to zero.    */

  if (h[j][k]/dt <= rinf)
  {
    rinf = h[j][k]/dt;
    h[j][k] = 0.0;
  }
  else
  {
    h[j][k] = h[j][k] - rinf*dt;
  }

  /* Accumulated volume of infiltrated water (in m)         */

  vinf[j][k] = vinf[j][k] + rinf*dt;
}
```

```
                      /*******************************/
                      /*        InitializeVars.c       */
                      /*******************************/


#include "all.h"

extern void InitializeVars()
{
  int SizeFr,j,k;
  float InitDepth;

  /* Grid values                                              */
  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        /* retention depth was entered in mm.                 */
        ret[j][k] = (float)(retention[iman[j][k]] * 0.001);
        dqov[j][k] = 0.0;
        dqch[j][k] = 0.0;
        if(indexinf == 1) vinf[j][k] = 0.0;
        rtot[j][k] = 0.0;
        rint[j][k] = 0.0;

        /* If base depths are defined,they are read as a grid    */
        /* Otherwise, initialize them to zero                    */
        if(indexbdepth != 1) h0[j][k] = 0.0 ;

        /* If storage depths are defined,they are read as a grid  */
        /* Otherwise, initialize them to zero                    */
        if(indexsdep   != 1) sdep[j][k] = 0.0 ;

        if(indexeros == 1)
        {
          t_SedFlux[j][k] = 0.0;
          for (SizeFr=1;SizeFr<=NumFracts;SizeFr++)
          {
            DepSed[SizeFr][j][k] = 0.0;
            ErodSed [SizeFr][j][k] = 0.0;
            SusSed[SizeFr][j][k] = 0.0;
            SedFlux[SizeFr][j][k] = 0.0;
          }

          /* Case in which metals are present                 */
          if(NumFracts > 3) maxCCU[j][k] = 0.0 ;

        }
      }
    }
  }

  /* Total volumes                                            */
  vin = 0.0;
  vout = 0.0;
  vinftot = 0.0;
```

```c
/* Counters                                                      */
raincount = 0;
icount = 1;
ipcount = 1;
ifcount = 1;

/* Variables that hold min. and max. values                     */
amaxdepth = 9e-30;
amindepth = 9e30;
amaxcdepth = 9e-30;
amincdepth = 9e30;
amaxvinf = 9e-30;
aminvinf = 9e30;
amaxrain = 9e-30;
aminrain = 9e30;
amaxSusOv = 9e-30;
amaxSusCh = 9e-30;
amaxSusCov = 9e-30;
amaxSusCch = 9e-30;

/* Sediment volumes by size fraction (sand/silt/clay)           */
if(indexeros == 1)
{
  /* Allocate here memory for the next vectors                  */
  tot_eroded = (float*)vMAlloc(sizeof(float),NumFracts);
  sus_ov     = (float*)vMAlloc(sizeof(float),NumFracts);
  dep_ov     = (float*)vMAlloc(sizeof(float),NumFracts);
  tot_ov     = (float*)vMAlloc(sizeof(float),NumFracts);
  sus_ch     = (float*)vMAlloc(sizeof(float),NumFracts);
  dep_ch     = (float*)vMAlloc(sizeof(float),NumFracts);
  tot_ch     = (float*)vMAlloc(sizeof(float),NumFracts);
  sed_out    = (float*)vMAlloc(sizeof(float),NumFracts);

  for (SizeFr=1;SizeFr<=NumFracts;SizeFr++)
  {
    tot_eroded[SizeFr] = 0.0;  /* Eroded  during the simulation     */
    sus_ov[SizeFr] = 0.0;     /* Suspended remaining on overl.      */
    dep_ov[SizeFr] = 0.0;     /* Deposited remaining on overl.      */
    tot_ov[SizeFr] = 0.0;     /* Susp.+ Dep. remaining on overl.    */
    sus_ch[SizeFr] = 0.0;     /* Suspended remaining in channels    */
    dep_ch[SizeFr] = 0.0;     /* Deposited remaining in channels    */
    tot_ch[SizeFr] = 0.0;     /* Susp.+ Dep. remaining in channels  */
    sed_out[SizeFr] = 0.0;
  }

}

/* Water volumes at the begining and end of the  simulation     */
init_ov_vol = 0.0;
init_ch_vol = 0.0;
final_ov_vol = 0.0;
final_ch_vol = 0.0;
```

```
/* Initializing channel and overland water depths                      */
for(j=1;j<=nrows;j++)
{
  for(k=1;k<=ncols;k++)
  {
    /* Initial Depth = storage depth + base depth (in meters)    */
    InitDepth = sdep[j][k] + h0[j][k];

    if(indexchan == 1 && link[j][k] > 0)
    {
      hch[j][k] = InitDepth;

      if( InitDepth > chp[link[j][k]][node[j][k]][3])
      {
        h[j][k] = InitDepth - chp[link[j][k]][node[j][k]][3];
      }
      else
      {
        h[j][k] = 0.0;
      }
    }
    else
    {
      h[j][k] = InitDepth;
    }
  }
}

/* After computing total initial depths (h0+sdep) we no longer  */
/* need **h0. Storage depths, **sdep, will be used thouhgout    */
/* the program and its memory is freed at the end of the run    */
mfree(h0);
}
```

```
/******************************/
/*          intercept.c          */
/******************************/


#include "all.h"

extern void intercept(int j, int k)
{

  if (rint[j][k] * dt >= ret[j][k])
  {
    rint[j][k] = rint[j][k] - ret[j][k] / dt;
    ret[j][k] = 0;
  }
  else
  {
    ret[j][k] = ret[j][k] - rint[j][k] * dt;
    rint[j][k] = 0;
  }

}
```

```c
/******************************/
/*          MemFree.c         */
/******************************/


#include "all.h"

extern void MemFree()
{
  vfree(pman);
  vfree(retention);
  vfree(cfactor);
  vfree(pfactor);

  if(indexdis == 1)
  {
    mfree(iq);
    vfree(areaQ);
  }

  /* Deallocate memory for input arrays                        */
  mfree(ishp);
  mfree(isoil);
  mfree(iman);
  mfree(e);
  mfree(ret);
  if(indexinf == 1) mfree(pinf);

  /* Deallocate memory for distributed rainfall related params.  */
  if(irain == 1)
  {
    vfree(xrg);
    vfree(yrg);
    vfree(rrg);
  }

  /* Deallocate memory for the channel related parameters        */
  if(indexchan == 1)
  {
    mfree(link);
    mfree(node);
    tfree(chp);
    tfree(ichn);
    mfree(hch);
    mfree(ec);
  }

  /* Deallocate memory for output arrays                       */

  mfree(sdep);
  if(indexinf == 1)  mfree(vinf);
  mfree(h);
  mfree(dqov);
  mfree(dqch);
  mfree(rtot);
  mfree(rint);
  vfree(q);
```

```c
  if(indexeros == 1)
  {
    vfree(ds);                    /* Deallocate memory for 1-D arrays  */
    vfree(ws);
    vfree(kfactor);
    vfree(iSedConc);
    vfree(iSusSed);
    vfree(iDepSed);
    vfree(iSedFlux);
    vfree(iNetEros);
    vfree(tot_eroded);
    vfree(sus_ov);
    vfree(dep_ov);
    vfree(tot_ov);
    vfree(sus_ch);
    vfree(dep_ch);
    vfree(tot_ch);
    vfree(sed_out);
    if(indexsed == 1) vfree(areaQs);
    vfree(qsed);

    mfree(pfraction);          /* Deallocate memory for 2-D arrays  */
    if(indexsed == 1) mfree(ised);
    mfree(t_SedConc);
    mfree(t_SusSed);
    mfree(t_DepSed);
    mfree(t_SedFlux);
    mfree(t_NetEros);
    if(NumFracts > 3)
    {
      mfree(CCU);
      mfree(maxCCU);
    }

    tfree(SedConc);              /* Deallocate memory for 3-D arrays  */
    tfree(SusSed);
    tfree(DepSed);
    tfree(ErodSed);
    tfree(SedFlux);
  }

}
```

```
                /*****************************/
                /*        OvrlDepth.c        */
                /*****************************/


#include "all.h"


extern void OvrlDepth( )
{

  int j,k,l,rindex,icall;
  float hov;


  /*****************************/
  /*    Read rainfall intensity  */
  /*****************************/

  icall = 0;
  rindex = 1;

  if(iter > nitrn) rindex = 0;
  if(iter <= nitrn && irain == 1)
  {
    icall=1;

    raincount = raincount + 1;

    if(raincount == nread+1) raincount = 1;

    if(raincount == 1)
    {
      for(l=1;l<=nrg;l++)
      {
        fscanf(rain_file_fptr,"%f ",&rrg[l]);
      }
    }
  }
```

```
/********************************************/
/*    Updating overland depth (water balance) */
/********************************************/

for(j=1;j<=nrows;j++)
{
  for(k=1;k<=ncols;k++)
  {
    if(ishp[j][k] != nodatavalue)
    {
      /* Calculate rainfall intensity in each watershed cell     */
      if(irain == 0)   /* Uniform Rainfall */
      {
        /* Convert uniform rainfall intensity: mm/h -> m/s      */

        rint[j][k] = (float)(crain/3600.0/1000.0);

        /* Rainfall rate reduced until intercept. is satisfied  */

        if (ret[j][k] != 0)   intercept(j,k);
      }
      else      /* Spatially Distributed Rainfall */
      {
        if(icall == 1)
        {
          rain(j,k);
        }
      }

      if(iter > nitrn) rint[j][k] = 0.0;

      /* Determining the Min. and Max. Rainfall Intensity        */

      if(rint[j][k] <= aminrain) aminrain=rint[j][k];
      if(rint[j][k] >= amaxrain) amaxrain=rint[j][k];

      /* Determines total rainfall depth                         */

      rtot[j][k] = rtot[j][k] + rint[j][k]*dt;

      /* Computing the Overland Depth due to the Overland Flow  */
      hov = dqov[j][k] * dt / (dx*dx);

      /* Computing the Total Overland Depth due to the        */
      /* overland flow, previous overland depth, plus rainfall */
      /* minus    interception                                */

      hov = hov + h[j][k] + rindex * rint[j][k] * dt;

      /* When HOV < 0, then a negative depth situation occurs. */

      if(hov < 0.0)
      {
        if(hov > -0.0001) hov = 0;
        else
          WriteErrorFreeMem(j,k,h[j][k],hov);
      }

      h[j][k] = hov;

      /* Calling the Infiltration Subroutine. HOV will be     */
      /* modified by subtracting  the infiltration losses.    */
      /* NOTE : If the infiltration volume that can be  lost  */
```

```
      /* for DT is greater than the amount of surface        */
      /* volume present, then HOV will be reduced to zero.    */
      /* Determine the maximum and minimum infiltration depth */

      if(indexinf == 1)
      {
        infilt(j,k);
        aminvinf = MIN(vinf[j][k],aminvinf);
        amaxvinf = MAX(vinf[j][k],amaxvinf);
      }


      /* Determining the Minimum and Maximum Overland Depth   */

      amindepth = MIN(h[j][k], amindepth);
      amaxdepth = MAX(h[j][k], amaxdepth);

      /* Once used in the water balance calculation, this     */
      /* variable  is reset to zero. It will be recalculated  */
      /* in the next time step in the ov. flow routing        */

      dqov[j][k] = 0.0;

      if(iter == niter)  /* Last computation step */
      {
        /* Total Volume of Rainfall minus interception;       */

        vin = vin + rtot[j][k] * dx * dx;

        /* Total Volume of Infiltration                       */

        if(indexinf == 1) vinftot = vinftot + vinf[j][k] * dx * dx;
      }
    }
  }
 }
}
```

```c
/*******************************/
/*            OvrlRout.c          */
/*******************************/

/*********************************/
/*          FUNCT: OvrlRout        */
/*********************************/

#include "all.h"

extern void OvrlRout()
{

  int j,k,jj,kk,l;

  for(j=1;j<=nrows;j++)
  {
   for(k=1;k<=ncols;k++)
   {
     if(ishp[j][k] != nodatavalue)
     {
       for(l=-1;l<=0;l++)
       {

         if (!(iter % 2))   /* This IF ELSE statatements simply   */
         {                  /* alternate the computational        */
           jj=j-l;          /* direction between one time step    */
           kk=k+l+1;        /* and the next one. For an odd       */
         }                  /* number of time step the flow is    */
         else               /* computed first in the x-direction  */
         {                  /* and then in the y-direction.  For  */
           jj=j+l+1;        /* an even number of time step, the   */
           kk=k-l;          /* opposite is true                   */
         }


         if(jj <= nrows && kk <=ncols && ishp[jj][kk] != nodatavalue)
         {
           ovrl(j,k,jj,kk);
         }
       }
     }
   }
  }
}
```

```
      /*********************************/
      /*          FUNCT: Ovrl          */
      /*********************************/

extern void ovrl( int j,int k, int jj, int kk)
{
  int jfrom,kfrom,jto,kto;

  float a=1.0;

  float vel = 0.0;

  float so,sf,dhdx,hh,rman,alfa,dqq,stordepth;

  so = (e[j][k] - e[jj][kk])/dx;

  dhdx = (h[jj][kk] - h[j][k])/dx;

  sf = so - dhdx +(float)(1e-30);

  hh = h[j][k];  /* Water depth including storage depth        */

  rman = pman[iman[j][k]];

  if(indexchan == 1 && link[j][k] > 0)
  {
    if(sdep[j][k] > chp[link[j][k]][node[j][k]][3])
    {
      /* Channel routing and current cell is a channel cell       */
      stordepth = sdep[j][k] - chp[link[j][k]][node[j][k]][3];
    }
    else
    {
      stordepth = 0.0;
    }
  }
  else
  {
    stordepth = sdep[j][k];
  }

  /* Negative Sf => reverse flow, take depth, Manning n and      */
  /* storage depth values from (jj,kk) cell                      */
  if(sf < 0)
  {
    hh = h[jj][kk];
    rman = pman[iman[jj][kk]];

    if(indexchan == 1 && link[jj][kk] > 0)
    {
      if(sdep[jj][kk] > chp[link[jj][kk]][node[jj][kk]][3])
      {
        stordepth =
          sdep[jj][kk] - chp[link[jj][kk]][node[jj][kk]][3];
      }
      else
      {
        stordepth = 0.0;
      }
    }
    else
    {
      stordepth = sdep[jj][kk];
```

94

```
        }
    }

    /* There is flow between (i,j) and (ii,jj) only if the water    */
    /* depth is greater than the storage depth and Sf is not zero   */
    if(hh > stordepth && sf != 0)
    {
      alfa = (float)((pow(fabs(sf),0.5))/rman);

      /*  Note : The variable "a" represents the sign of the        */
      /*  Friction Slope (Sf)  Computing Overland Flow              */

      if(sf >= 0) a = 1.0;

      if(sf < 0) a = -1.0;

      dqq = (float)(a*dx*alfa*pow((hh-stordepth),1.667));

      dqov[j][k] = dqov[j][k] - dqq;

      dqov[jj][kk] = dqov[jj][kk] + dqq;

      /* Compute overland sediment flow and erosion/deposition      */

      if(indexeros == 1 && dqq != 0.0)
      {
        /* Checking which is the receiving/outgoing cell */
        if(a > 0)          /* (J,K) to (JJ,KK) */
        {
          jfrom = j;
          kfrom = k;
          jto = jj;
          kto = kk;
        }
        else                  /* (JJ,KK) to (J,K) */
        {
          jfrom = jj;
          kfrom = kk;
          jto = j;
          kto = k;
        }
        /* Sediment is not routed FROM a channel cell, only from an */
        /* overland cell. Sediment can be routed TO a channel cell  */
        if (ishp[jfrom][kfrom] == 1)
        {
          RouteSedOvrl(dqq,sf,jfrom,kfrom,jto,kto);
        }
      }

    }  /* End of HH >= STORDEPTH */

}   /* End of OVRL */
```

```
                /******************************/
                /*          rain.c            */
                /******************************/


#include "all.h"

extern void rain(int j,int k)
{
  float totdist = 0.0;
  float totrain = 0.0;

  float xc,yc,dist,xul,yul;
  int l;

  rint[j][k] = -999.0;

  xul = xllcorner;
  yul = yllcorner + nrows * dx;

  if(nrg == 1)
  {
    rint[j][k] = rrg[1];
  }
  else
  {
    for(l=1;l<=nrg;l++)
    {
      yc = (float)(yul - j * dx + dx / 2.0);
      xc = (float)(xul + k * dx - dx / 2.0);

      dist = (float)(sqrt(pow((yc-yrg[l]),2.0) +
                                    pow((xc-xrg[l]),2.0)));
      if(dist < 1e-5)
      {
        rint[j][k] = rrg[l];
      }
      else
      {
        totdist = (float)(totdist + 1.0/pow(dist,2.0));
        totrain = (float)(totrain + rrg[l]/pow(dist,2.0));
      }
    }
  }

  if(rint[j][k] == -999.0)
  {
    rint[j][k] = totrain / totdist;
  }

  /* Changing Units from inches/hour to meters/second */

  rint[j][k] = (float)(rint[j][k] * 0.0254 / 3600.);

  /* Rainfall rate is reduced until interception is satisfied */

  if (ret[j][k] != 0)
  {
    intercept(j,k);
  }

}
```

```
                /******************************/
                /*       ReadChannFile.c       */
                /******************************/


#include "all.h"

extern void ReadChannFile()
{
  char ch;
  int j,k,nchan_link;

  /* Opening SUMMARY file to write channel parameters            */

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}

  /* Allocate memory for the CHP matrix   */
  chp = (float***) tMAlloc(sizeof(float),maxlinks,maxnodes,6);

  printf("Reading Channel Data \n");

  do{
   ch=fscanf(chn_file_fptr,"%ld",&nchan_link);
   ch=fscanf(chn_file_fptr,"%ld",&nchan_node[nchan_link]);

   for(k=1;k<=nchan_node[nchan_link];k++)
   {
     for(j=1;j<=6;j++)
     {
       ch=fscanf(chn_file_fptr,"%f ",&chp[nchan_link][k][j]);
     }
   }

  } while(ch!=EOF);

  /* Writing Channel Information to Output Summary File

  fprintf(Summ_file_fptr,"\n Channel Input Data \n");
  fprintf(Summ_file_fptr," ------------------ \n");
  for(i=1;i<=maxlinks;i++)
  {
    for(k=1;k<=nchan_node[i];k++)
    {
      for(j=1;j<=6;j++)
      {
        fprintf(Summ_file_fptr,"CHP[%ld][%ld][%ld] = %f ",
                                   i,k,j,chp[i][k][j]);
      }
      fprintf(Summ_file_fptr,"\n");
    }
  }
*/

  printf("Successfully Read Channel Data \n");
  fclose(chn_file_fptr);
  fclose(Summ_file_fptr);

}
```

```
              /*****************************/
              /*      ReadControlFile.c        */
              /*****************************/


#include "all.h"


extern void ReadControlFile(char control_file[])
{
  int i,j,k,r;
  FILE *control_file_fptr = NULL;

  /* Opens control file to read and the summary file to write */
  if((control_file_fptr=fopen(control_file,"r"))==NULL)
  {printf("Can't open control File : %s \n",control_file);
  exit(EXIT_FAILURE);}

  printf("\nReading CONTROL file \n");

  fprintf(Summ_file_fptr,"\nCONTROL Input Data \n");
  fprintf(Summ_file_fptr,   "================= \n");
  fprintf(Summ_file_fptr,   "Control file : %s\n", control_file);

  fscanf(control_file_fptr,"%f %ld %ld %ld %ld",
         &dt,&niter,&nitrn,&nprn,&nplt);

  fprintf(Summ_file_fptr,
  "DT = %2.2f sec NITER= %ld NITRN = %ld NPRN = %ld NPLT = %ld\n\n",
                              dt,niter,nitrn,nprn,nplt);

  fprintf(Summ_file_fptr,"Overland outlet cell: \n");
  fscanf(control_file_fptr,"%ld %ld %f",&jout,&kout,&sovout);
  fprintf(Summ_file_fptr,"JOUT = %ld KOUT = %ld SOVOUT = %g \n\n",
                                          jout,kout,sovout);

  fscanf(control_file_fptr,"%i",&indexchan);
  fprintf(Summ_file_fptr,"INDEXCHAN= %i \n",indexchan);

  fscanf(control_file_fptr,"%f",&elconv);
  fprintf(Summ_file_fptr,"ELCONV = %.2f\n",elconv);

  fprintf(Summ_file_fptr,"\nRainfall Data \n");
  fprintf(Summ_file_fptr,   "============= \n");

  fscanf(control_file_fptr,"%ld ",&irain);
  fprintf(Summ_file_fptr,"IRAIN = %ld \n",irain);

  if(irain == 0)
  {
   fscanf(control_file_fptr,"%f ",&crain);
   fprintf(Summ_file_fptr,"CRAIN = %.3f (mm/h)\n",crain);
  }
  else
  {
   fscanf(control_file_fptr,"%ld %ld ",&nrg,&nread);
   fprintf(Summ_file_fptr,"NRG = %ld NREAD = %ld \n",nrg,nread);

   xrg = (float*) vMAlloc(sizeof(float),nrg);
   yrg = (float*) vMAlloc(sizeof(float),nrg);
   rrg = (float*) vMAlloc(sizeof(float),nrg);
```

```c
  for(r=1;r<=nrg;r++)
  {
    fscanf(control_file_fptr,"%f %f ",&xrg[r],&yrg[r]);
    fprintf(Summ_file_fptr,"XRG[%ld] = %8.6f\tYRG[%ld] = %8.6f \n",
                                            r,xrg[r],r,yrg[r]);
  }
}

fprintf(Summ_file_fptr,"\nLand Use Parameters \n");
fprintf(Summ_file_fptr,   "=================== \n");

fscanf(control_file_fptr," %i %i %i",&nman,&indexsdep,&indexbdepth);
fprintf(Summ_file_fptr,"NMAN = %i INDEXSDEP = %i INDEXBDEPTH = %i\n\n",
                                    nman,indexsdep,indexbdepth);

/* Allocate memory for the variables derived from the landuse    */
pman = (float*) vMAlloc(sizeof(float),nman);
retention = (float*) vMAlloc(sizeof(float),nman);
cfactor = (float*) vMAlloc(sizeof(float),nman);
pfactor = (float*) vMAlloc(sizeof(float),nman);


fprintf(Summ_file_fptr,"\tIndex\tManning    Intercept.\tCusle\tPusle\n");
fprintf(Summ_file_fptr,"\t-----\t-- n --    -- [mm] --\t-----\t-----\n");

for(i=1;i<=nman;i++)
{
 fscanf(control_file_fptr,"%i ",&k);
 fscanf(control_file_fptr,"%f %f %f %f",
        &pman[k], &retention[k],&cfactor[k],&pfactor[k]);
 fprintf(Summ_file_fptr,
    "\t%3ld%11.3f\t%10.3f\t%5.3f\t%5.3f\n",
                    k,pman[k],retention[k],cfactor[k],pfactor[k]);
}

fprintf(Summ_file_fptr,"\nSoil type Parameters \n");
fprintf(Summ_file_fptr,   "==================== \n");

fscanf(control_file_fptr,"%i ",&nsoil);
fprintf(Summ_file_fptr,"\nNSOIL= %i",nsoil);

fscanf(control_file_fptr,"%i ",&indexinf);
fprintf(Summ_file_fptr,"\nINDEXINF= %i",indexinf);

if(indexinf == 1)
{
  pinf= (float**)mMAlloc(sizeof(float),nsoil,3);

  fprintf(Summ_file_fptr,
    "\n      Index    Ks        G        Md    \n");
  fprintf(Summ_file_fptr,
      "       -----   [cm/h]    [cm]     -----  \n");
```

```c
  /* Read infiltration parameter values from the CONTROL file */
  for(i=1;i<=nsoil;i++)
  {
    fscanf(control_file_fptr,"%i ",&k);
    fprintf(Summ_file_fptr,"%10i",k);
    for(j=1;j<=3;j++)
    {
      fscanf(control_file_fptr,"%f ",&pinf[k][j]);
      fprintf(Summ_file_fptr,"%10.3f",pinf[k][j]);
      /* Converts Ks: cm/h -> m/s                               */
      if(j ==1) pinf[k][j] = (float)(pinf[k][j] * 0.01 / 3600) ;
      /* Converts G:  cm -> m                                   */
      if(j ==2) pinf[k][j] = (float)(pinf[k][j] * 0.01);
    }
    fprintf(Summ_file_fptr,"\n");
  }

}

fscanf(control_file_fptr,"%i %i",&indexeros,&NumFracts);
fprintf(Summ_file_fptr,"\nINDEXEROS= %i NOFRACTS= %i\n",indexeros,NumFracts);

if (indexeros == 1)
{
  /* Characteristics for each size fraction:
*/
  ds = (double*)vMAlloc(sizeof(double),NumFracts);  /* ... particle size
*/
  ws = (double*)vMAlloc(sizeof(double),NumFracts);  /* ... particle fall
velocity */

  /* Erosion parameters for each soil type
*/
  kfactor = (float*)vMAlloc(sizeof(float),nsoil);   /* ... erodibility factor
*/
  pfraction = (float**)mMAlloc(sizeof(float),nsoil,NumFracts); /* ..%
fraction    */

  for(i=1;i<=NumFracts;i++)  /* Reads in mean particle sizes [m]          */
  {
    fscanf(control_file_fptr,"%lf ",&ds[i]);
    fprintf(Summ_file_fptr,"ds[%i] = %10g  ",i,ds[i]);
  }
  fprintf(Summ_file_fptr,"\n");

  for(i=1;i<=NumFracts;i++)  /* Reads in particle fall velocity [m/s]      */
  {
    fscanf(control_file_fptr,"%lf ",&ws[i]);
    fprintf(Summ_file_fptr,"ws[%i] = %10g  ",i,ws[i]);
  }
  fprintf(Summ_file_fptr,"\n\n");


  fprintf(Summ_file_fptr,
      "       Index    Kusle    %%Fract1   %%Fract2   %%Fract3 (etc) \n");
  fprintf(Summ_file_fptr,
      "       -----    -----    -----    -----     -----  \n");
```

```c
 /* Read erosion parameter values from the CONTROL file */
  for(i=1;i<=nsoil;i++)
  { /* Soil number and Kusle     */
    fscanf(control_file_fptr,"%i %g ",&k, &kfactor[i]);
    fprintf(Summ_file_fptr," %10i  %10.3g ",k, kfactor[i]);
    for(j=1;j<=NumFracts;j++)
    {
      fscanf(control_file_fptr,"%g ",&pfraction[k][j]);
      fprintf(Summ_file_fptr,"%10.3g",pfraction[k][j]);
    }
    fprintf(Summ_file_fptr,"\n");
  }

  /* Allocate memory for the vectors holding the flags for printing */
  /* the sediment variables time-series grids and initialize them   */
  iSedConc = (int*)vMAlloc(sizeof(int),NumFracts+1);
  iSusSed  = (int*)vMAlloc(sizeof(int),NumFracts+1);
  iDepSed  = (int*)vMAlloc(sizeof(int),NumFracts+1);
  iSedFlux = (int*)vMAlloc(sizeof(int),NumFracts+1);
  iNetEros = (int*)vMAlloc(sizeof(int),NumFracts+1);
}


fprintf(Summ_file_fptr,"\nInternal Gages Data \n");
fprintf(Summ_file_fptr,   "==================== \n");

fscanf(control_file_fptr,"%ld %ld %i",&indexdis,&ndis, &unitsQ);
fprintf(Summ_file_fptr,
                  "INDEXDIS = %ld  NDIS = %ld  Q_units = %i \n",
                                      indexdis,ndis, unitsQ);

if(indexdis == 1)
{
  fprintf(Summ_file_fptr,"\n        Gage      Row      Column      Area\n");
  fprintf(Summ_file_fptr,"        ----      ---      ------     [has]\n");
  iq = (int**)mMAlloc(sizeof(int),ndis,2);
  areaQ = (float*) vMAlloc(sizeof(float),ndis);

  for (i=1;i<=ndis;i++)
  {
   fprintf(Summ_file_fptr,"        %2i",i);
   for (j=1;j<=2;j++)
   {
     fscanf(control_file_fptr,"%ld ",&iq[i][j]);
     fprintf(Summ_file_fptr,"%10i",iq[i][j]);
   }
   fscanf(control_file_fptr,"%f", &areaQ[i]);
   fprintf(Summ_file_fptr,"%14.3f\n", areaQ[i]);
  }
}


fprintf(Summ_file_fptr,"\nInternal Sediment Gages Data \n");
fprintf(Summ_file_fptr,   "============================= \n");

fscanf(control_file_fptr,"%ld %ld %i",&indexsed,&nsed, &unitsQs);
fprintf(Summ_file_fptr,
                  "INDEXSED = %ld  NSED = %ld Qs_Units = %i\n",
                                      indexsed,nsed, unitsQs);
```

```
   if(indexeros == 1 && indexsed == 1)
   {
     fprintf(Summ_file_fptr,"\n        Gage      Row      Column      Area\n");
     fprintf(Summ_file_fptr,"        ----      ---      ------      [has]\n");
     ised = (int**)mMAlloc(sizeof(int),nsed,2);
     areaQs = (float*) vMAlloc(sizeof(float),nsed);

     for(i=1;i<=nsed;i++)
     {
      fprintf(Summ_file_fptr,"       %2i",i);
      for (j=1;j<=2;j++)
      {
        fscanf(control_file_fptr,"%ld ",&ised[i][j]);
        fprintf(Summ_file_fptr,"%10i",ised[i][j]);
      }
      fscanf(control_file_fptr,"%f", &areaQs[i]);
      fprintf(Summ_file_fptr,"%14.3f\n",areaQs[i]);
     }
   }

   /* Allocate memory for the vector that holds the flow        */
   /* dicharge and sediment discharge at each of those locations  */
   q =  (float*) vMAlloc(sizeof(float),ndis);
   qsed = (float*) vMAlloc(sizeof(float),nsed);

   /* Closing CONTROL and SUMMARY file                          */
   fclose(control_file_fptr);
   fclose(Summ_file_fptr);

   printf("Successfully Read All CONTROL Information\n\n");
}
```

```
                 /******************************/
                 /*        ReadGrids.c         */
                 /******************************/


#include "all.h"

extern void ReadGrids()
{
  /* Reads the input grids headers                           */
  ReadGridHeaders();

  /* Number of rows and columns in the grids have been read   */
  /* from header files.  Now we are able to dynamically       */
  /* allocating memory to the input & output grids            */
  GridsMemAlloc();

  /* Reading Input Grids                                      */
  ReadInputGrids();
}

      /********************************/
      /*     FCT: ReadGridHeaders     */
      /********************************/

extern void ReadGridHeaders()
{
  /* Variables that will hold the ASCII grids header information  */
  char nrows_label[15], ncols_label[15], xllcorner_label[15],
    yllcorner_label[15], cellsize_label[15], nodatavalue_label[15],
    h_label[7][80];
  int  head,i;
  float h_value[7];

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}

  header[0] = '\0';

  /* Read the shape ASCII grid header to get project info        */
  /* the "header" variable will be used to hold the header info   */

  fprintf(Summ_file_fptr,"\nASCII grid files header  \n");
  fprintf(Summ_file_fptr,"------------------------ \n");
  fscanf(shap_file_fptr,"%s %i",&ncols_label, &ncols);
  fscanf(shap_file_fptr,"%s %i",&nrows_label, &nrows);
  fscanf(shap_file_fptr,"%s %f",&xllcorner_label, &xllcorner);
  fscanf(shap_file_fptr,"%s %f",&yllcorner_label, &yllcorner);
  fscanf(shap_file_fptr,"%s %f",&cellsize_label, &dx);
  fscanf(shap_file_fptr,"%s %i",&nodatavalue_label, &nodatavalue);

  /* Do not use the xxxx_label variables because I want to       */
  /* make sure that the user entered them in the right order.     */

  head  = sprintf( header,        "ncols\t\t%i\n", ncols );
  head += sprintf( header + head, "nrows\t\t%i\n", nrows  );
  head += sprintf( header + head, "xllcorner\t%.2f\n", xllcorner);
  head += sprintf( header + head, "yllcorner\t%.2f\n", yllcorner);
  head += sprintf( header + head, "cellsize\t%.2f\n", dx);
  head += sprintf( header + head, "NODATA_value\t%i\n",
                                           nodatavalue);
```

```c
    fprintf(Summ_file_fptr,header);

/* Need to read the rest of the input grids headers              */
for (i=1;i<=6;i++)
 {
   if(elev_file_fptr)
     fscanf(elev_file_fptr,"%s %f",&h_label[i], &h_value[i]);
   else
   {
     printf(" Elevation grid doesn't exist. Pease, define one\n");
     exit(EXIT_FAILURE);
   }

   if(nsoil > 1)
   {
     if(soil_file_fptr)
       fscanf(soil_file_fptr,"%s %f",&h_label[i], &h_value[i]);
     else
     {
       printf("\n\nYou have defined more than one soil type");
       printf("\nYou need to define a soil type grid\n\n");
       exit(EXIT_FAILURE);
     }
   }

   if(nman > 1)
   {
     if(iman_file_fptr)
       fscanf(iman_file_fptr,"%s %f",&h_label[i], &h_value[i]);
     else
     {
       printf("\n\nYou have defined more than one land use type");
       printf("\nYou need to define a land use type grid\n\n");
       exit(EXIT_FAILURE);
     }
   }

   if(indexchan == 1)
   {
     if(node_file_fptr)
       fscanf(node_file_fptr,"%s %f",&h_label[i], &h_value[i]);
     else
     {
       printf("\n\nYou have selected the channel routing option");
       printf("\nYou need to define a node number grid\n\n");
       exit(EXIT_FAILURE);
     }

     if(link_file_fptr)
       fscanf(link_file_fptr,"%s %f",&h_label[i], &h_value[i]);
     else
     {
       printf("\n\nYou have selected the channel routing option");
       printf("\nYou need to define a link number grid\n\n");
       exit(EXIT_FAILURE);
     }

   }
```

104

```c
      if(indexsdep == 1)
      {
        if(sdep_file_fptr)
          fscanf(sdep_file_fptr,"%s %f",&h_label[i], &h_value[i]);
        else
        {
          printf("\n\nYou have selected the retention depth option");
          printf("\nYou need to define a retention depth grid\n\n");
          exit(EXIT_FAILURE);
        }
      }

      if(indexbdepth == 1)
      {
        if(bdepth_file_fptr)
          fscanf(bdepth_file_fptr,"%s %f",&h_label[i], &h_value[i]);
        else
        {
          printf("\n\nYou have selected the initial base depth option");
          printf("\nYou need to define the base depth grid\n\n");
          exit(EXIT_FAILURE);
        }
      }
    }

  fclose(Summ_file_fptr);
}


      /*********************************/
      /*      FCT: GridsMemAlloc        */
      /*********************************/

extern void GridsMemAlloc()
{
  printf("starting grids memory allocation\n");
  /* Allocate memory for input matrices                          */
  ishp  = (int**)mMAlloc(sizeof(int),nrows,ncols);
  isoil = (int**)mMAlloc(sizeof(int),nrows,ncols);
  iman  = (int**)mMAlloc(sizeof(int),nrows,ncols);
  e   = (float**)mMAlloc(sizeof(float),nrows,ncols);
  ret = (float**)mMAlloc(sizeof(float),nrows,ncols);

  if(indexchan == 1)
  {
    link = (int**)mMAlloc(sizeof(int),nrows,ncols);
    node = (int**)mMAlloc(sizeof(int),nrows,ncols);
    hch = (float**)mMAlloc(sizeof(float),nrows,ncols);


  /* Allocate memory for the EC (elevation in channels) matrix    */
  /* NOTE: EC[J][K] = E[J][K] at the beggining of the simulation  */
  /* This might change if the elevation of channel cells is       */
  /* defined in the CHANNEL parameter file -- along with the      */
  /* geometric and hydraulic characteristic corresponding to      */
  /* every node in a channel link                                 */
  ec = (float**)mMAlloc(sizeof(float),nrows,ncols);

  }

  sdep = (float**)mMAlloc(sizeof(float),nrows,ncols);
  h0 = (float**)mMAlloc(sizeof(float),nrows,ncols);
```

```c
  /* Allocate memory for output matrices                          */

  if(indexinf == 1)  vinf = (float**)mMAlloc(sizeof(float),nrows,ncols);

  h = (float**)mMAlloc(sizeof(float),nrows,ncols);

  if(indexeros == 1)
  {
    t_SedConc = (float**)mMAlloc(sizeof(float),nrows,ncols);
    t_SusSed  = (float**)mMAlloc(sizeof(float),nrows,ncols);
    t_DepSed  = (float**)mMAlloc(sizeof(float),nrows,ncols);
    t_SedFlux = (float**)mMAlloc(sizeof(float),nrows,ncols);
    t_NetEros = (float**)mMAlloc(sizeof(float),nrows,ncols);

    if(NumFracts > 3) /* Case in which metals are present        */
    {
      CCU = (float**)mMAlloc(sizeof(float),nrows,ncols);
      maxCCU = (float**)mMAlloc(sizeof(float),nrows,ncols);
    }

    /* Allocate memory for 3-D matrices                          */
    SedConc = (float***)tMAlloc(sizeof(float),NumFracts,nrows,ncols);
    SusSed = (float***)tMAlloc(sizeof(float),NumFracts,nrows,ncols);
    DepSed = (float***)tMAlloc(sizeof(float),NumFracts,nrows,ncols);
    ErodSed = (float***)tMAlloc(sizeof(float),NumFracts,nrows,ncols);
    SedFlux = (float***)tMAlloc(sizeof(float),NumFracts,nrows,ncols);
  }

  dqov = (float**)mMAlloc(sizeof(float),nrows,ncols);
  dqch = (float**)mMAlloc(sizeof(float),nrows,ncols);
  rtot = (float**)mMAlloc(sizeof(float),nrows,ncols);
  rint = (float**)mMAlloc(sizeof(float),nrows,ncols);

  printf("\nGrids memory allocation ends\n");
}
        /*********************************/
        /*         ReadInputGrids        */
        /*********************************/

extern void ReadInputGrids()
{
  int  j,k;

  ncells = 0;
  maxlinks = 0;
  maxnodes = 0;

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}
```

```
for(j=1;j<=nrows;j++)
{
  for(k=1;k<=ncols;k++)
  {
    fscanf(shap_file_fptr,"%ld ",&ishp[j][k]);

    if(ishp[j][k] != nodatavalue) ncells++;

    fscanf(elev_file_fptr,"%f ",&e[j][k]);
    e[j][k] = e[j][k] / elconv;

    /* If there is a channel in the current (j,k) cell,        */
    /* initialize the EC values to the E values before the     */
    /* simulation starts.                                      */
    /* NOTE: this won't be needed when the channel elevations  */
    /* are specified in the CHANNEL file                       */
    if (link[j][k] != nodatavalue) ec[j][k] = e[j][k];

    if(nsoil > 0)
    {
      if(nsoil == 1)
      {
        isoil[j][k] = 1;
      }
      else
      {
        fscanf(soil_file_fptr,"%ld ",&isoil[j][k]);
      }
    }

    if(nman == 1)
    {
      iman[j][k] = 1;
    }
    else
    {
      fscanf(iman_file_fptr,"%ld ",&iman[j][k]);
    }

    if(indexchan == 1)
    {
      fscanf(node_file_fptr,"%ld",&node[j][k]);
      fscanf(link_file_fptr,"%ld",&link[j][k]);
      if(ishp[j][k] != nodatavalue)
        maxlinks = (int)MAX(maxlinks,link[j][k]);
      if(ishp[j][k] != nodatavalue)
        maxnodes = (int)MAX(maxnodes,node[j][k]);

    }

    if(indexsdep == 1) fscanf(sdep_file_fptr,"%f",&sdep[j][k]);
    if(indexbdepth == 1) fscanf(bdepth_file_fptr,"%f",&h0[j][k]);
  }
}
```

```c
   /* NOTE: because the last node of one link and the first node   */
   /* of the downstream link is the same, the number of nodes of   */
   /* one link is the last numbered node in the link plus one      */
   /* Thus, the link with the maximum number of nodes will have    */
   /* an extra element as well                                     */
   /* NOTE: THIS IS NOT TRUE IF THE LINK WITH THE MAX. NUMBER OF    */
   /* NODES IS THE OUTER LINK. THIS AFFECTS THE PROGRAM BY         */
   /* ALLOCATING MEMORY FOR AN EXTRA ROW                          */
   maxnodes = maxnodes + 1;


   /* Input ASCII grids .... read
   fprintf(Summ_file_fptr,
                    "\nThe following Grids have been read: \n");
   fprintf(Summ_file_fptr," ----------------------------- \n");
   fprintf(Summ_file_fptr,"Elevation Grid: %s \n", elev_file);
   fprintf(Summ_file_fptr,"Shape Grid: \t%s \n", shap_file);
   fprintf(Summ_file_fptr,"Soils Grid: \t%s \n", soil_file);
   fprintf(Summ_file_fptr,"Landuse Grid: \t%s \n", iman_file);
   fprintf(Summ_file_fptr,"Node Grid: \t%s \n", node_file);
   fprintf(Summ_file_fptr,"Link Grid: \t%s \n\n", link_file);  */
   printf("Successfully Read Input Grid Data \n");
   fprintf(Summ_file_fptr,"Number of cells in basin: %i\n", ncells);
   fprintf(Summ_file_fptr,"Basin area: %.3f [ha]\n", ncells*dx*dx/10000);

   fclose(elev_file_fptr);
   fclose(shap_file_fptr);
   if(nsoil > 1)  fclose(soil_file_fptr);
   if(nman > 1) fclose(iman_file_fptr);
   if(indexchan == 1)
   {
     fclose(node_file_fptr);
     fclose(link_file_fptr);
   }
   if(indexsdep == 1) fclose(sdep_file_fptr);
   if(indexbdepth == 1) fclose(bdepth_file_fptr);

   fclose(Summ_file_fptr);
}
```

```c
                /*******************************/
                /*      ReadInputFiles.c       */
                /*******************************/

#include "all.h"

extern void ReadInputFiles()
{
  /* Local variables declaration */
  int ch ;
  char file_id[100],control_file[100];
  FILE *project_file_fptr = NULL;

  /* Opens the project file for reading */

  printf("Project File    %s \n",Project_file);

  if((project_file_fptr=fopen(Project_file,"r"))==NULL)
  {printf("Can't open Project File : %s \n",Project_file);
   exit(EXIT_FAILURE);}

  /* Reading the control and summary file names and opening the */
  /* summary file to write the parameter values                 */

  do{
  ch=fscanf(project_file_fptr,"%s",&file_id);
  if(ch!=EOF)
  {
    if(strcmp(file_id,"CONTROL") == 0)
    {
      ch=fscanf(project_file_fptr,"%s",&control_file);
      printf("CONTROL         %s \n",control_file);
    }
    if(strcmp(file_id,"SUMMARY") == 0)
    {
      ch=fscanf(project_file_fptr,"%s",&Summ_file);
      printf("SUMMARY         %s \n",Summ_file);
      if((Summ_file_fptr=fopen(Summ_file,"w"))==NULL)
      {printf("Can't open Output PRN File : %s \n",Summ_file);
       exit(EXIT_FAILURE);}
    }
   } /* End Of: if(ch!=EOF) */
  }while(ch != EOF);

  /* Closing the Project file */
  fclose(project_file_fptr);

  /* Reading in the CONTROL Information                         */
  ReadControlFile(control_file);

  /* Keeps on reading the project file  */
  ReadProjectFile();

  /* Reading and allocating memory for grids              */
  ReadGrids();

  /* Reads channel parameters and creates network topology      */
  if(indexchan == 1)
  {
    ReadChannFile();
    ChannToplg();
  }
}
```

```
                /*****************************/
                /*     ReadProjectFile.c     */
                /*****************************/


#include "all.h"

extern void ReadProjectFile()
{
  char   file_id[80],rain_file[80],elev_file[80],shap_file[80],soil_file[80],
       iman_file[80],sdep_file[80],bdepth_file[80],node_file[80],link_file[80],
       dis_out_file[80],sed_out_file[80],chn_file[80];

  int ch, SizeFr;
  FILE *project_file_fptr = NULL;

  /* Initialize the output grid names to the null element          */
  dname[0] = '\0';
  rname[0] = '\0';
  vinfname[0] = '\0';

  if((project_file_fptr=fopen(Project_file,"r"))==NULL)
  {printf("Can't open Project File : %s \n",Project_file);
   exit(EXIT_FAILURE);}

  /* Reading in the Data Filenames and Opening Files              */

  do{

  ch=fscanf(project_file_fptr,"%s",&file_id);

  if(ch!=EOF)
  {
    if(strcmp(file_id,"PRECIP") == 0 && irain == 1)
    {
      ch=fscanf(project_file_fptr,"%s",&rain_file);
      if((rain_file_fptr=fopen(rain_file,"r"))==NULL)
      {printf("Can't open Rain File : %s \n",rain_file);
       exit(EXIT_FAILURE);}
      printf("PRECIP            %s \n",rain_file);
    }
    if(strcmp(file_id,"CHANNEL") == 0)
    {
      ch=fscanf(project_file_fptr,"%s",&chn_file);
      if((chn_file_fptr=fopen(chn_file,"r"))==NULL)
        {printf("Can't open Channel File : %s \n",chn_file);
        exit(EXIT_FAILURE);}
      printf("CHANNEL          %s \n",chn_file);
    }
    if(strcmp(file_id,"ELEVATION") == 0)
    {
      ch=fscanf(project_file_fptr,"%s",&elev_file);
      if((elev_file_fptr=fopen(elev_file,"r"))==NULL)
      {printf("Can't open Elevation File : %s \n",elev_file);
       exit(EXIT_FAILURE);}
      printf("ELEVATION        %s \n",&elev_file);
    }
```

```c
if(strcmp(file_id,"SOIL") == 0)
{
  ch=fscanf(project_file_fptr,"%s",&soil_file);
  if((soil_file_fptr=fopen(soil_file,"r"))==NULL)
  {printf("Can't open Soil File : %s \n",soil_file);
  exit(EXIT_FAILURE);}
  printf("SOIL             %s \n",soil_file);
}
if(strcmp(file_id,"LANDUSE") == 0)
{
  ch=fscanf(project_file_fptr,"%s",&iman_file);
  if((iman_file_fptr=fopen(iman_file,"r"))==NULL)
  {printf("Can't open Landuse File : %s \n",iman_file);
  exit(EXIT_FAILURE);}
  printf("LANDUSE          %s \n",&iman_file);
}
if(strcmp(file_id,"NODE") == 0  && indexchan == 1)
{
  ch=fscanf(project_file_fptr,"%s",&node_file);
  if((node_file_fptr=fopen(node_file,"r"))==NULL)
  {printf("Can't open Node File : %s \n",node_file);
  exit(EXIT_FAILURE);}
  printf("NODE         %s \n",node_file);
}
if(strcmp(file_id,"LINK") == 0 && indexchan == 1)
{
  ch=fscanf(project_file_fptr,"%s",&link_file);
  if((link_file_fptr=fopen(link_file,"r"))==NULL)
  {printf("Can't open Link File : %s \n",link_file);
  exit(EXIT_FAILURE);}
  printf("LINK         %s \n",link_file);
}
if(strcmp(file_id,"MASK") == 0)
{
  ch=fscanf(project_file_fptr,"%s",&shap_file);
  if((shap_file_fptr=fopen(shap_file,"r"))==NULL)
  {printf("Can't open Shape File : %s \n",shap_file);
  exit(EXIT_FAILURE);}
  printf("MASK         %s \n",shap_file);
}
if(strcmp(file_id,"STORAGE_DEPTH") == 0 && indexsdep == 1)
{
  ch=fscanf(project_file_fptr,"%s",&sdep_file);
  if((sdep_file_fptr=fopen(sdep_file,"r"))==NULL)
  {printf("Can't open Storage Depth File : %s \n",sdep_file);
  exit(EXIT_FAILURE);}
  printf("STORAGE_DEPTH            %s \n",sdep_file);
}
if(strcmp(file_id,"BASE_DEPTH") == 0 && indexbdepth == 1)
{
  ch=fscanf(project_file_fptr,"%s",&bdepth_file);
  if((bdepth_file_fptr=fopen(bdepth_file,"r"))==NULL)
  {printf("Can't open Base Depth File : %s \n",bdepth_file);
  exit(EXIT_FAILURE);}
  printf("BASE_DEPTH            %s \n",bdepth_file);
}
```

```c
if(strcmp(file_id,"DISCHARGE_OUT") == 0)
{
  ch=fscanf(project_file_fptr,"%s",&dis_out_file);
  if((dis_out_file_fptr=fopen(dis_out_file,"w"))==NULL)
  {printf("Can't open Discharge File : %s \n",dis_out_file);
  exit(EXIT_FAILURE);}
  printf("DISCHARGE_OUT   %s \n",dis_out_file);
}
if(strcmp(file_id,"SEDIMENT_OUT") == 0 && indexeros == 1)
{
  ch=fscanf(project_file_fptr,"%s",&sed_out_file);
  if((sed_out_file_fptr=fopen(sed_out_file,"w"))==NULL)
  {printf("Can't open Sediment File : %s \n",sed_out_file);
  exit(EXIT_FAILURE);}
  printf("SEDIMENT_OUT    %s \n",sed_out_file);
}

/* Reading in the time-series grid names                    */
if(strcmp(file_id,"WATER_DEPTH") == 0)
{
  ch=fscanf(project_file_fptr,"%s",&dname);
  printf("WATER_DEPTH          %s \n",dname);
}
if(strcmp(file_id,"RAINFALL") == 0  && irain == 1)
{
  ch=fscanf(project_file_fptr,"%s",&rname);
  printf("RNAME          %s \n",rname);
}
if(strcmp(file_id,"INF_DEPTH") == 0 && indexinf == 1)
{
  ch=fscanf(project_file_fptr,"%s",&vinfname);
  printf("INF_DEPTH        %s \n",vinfname);
}

/* Reading the sediment time-series grids flags          */
if(strcmp(file_id,"OutputPath:") == 0 && indexeros == 1)
{
  ch=fscanf(project_file_fptr,"%s",&SedGridsPath);
  printf("SEDGRIDSPATH    %s \n",SedGridsPath);
}

if(strcmp(file_id,"CONCENTRATION") == 0 && indexeros == 1)
{
  printf("CONCENTRATION   ");
  for(SizeFr=1;SizeFr<=NumFracts+1;SizeFr++)
  {
    ch=fscanf(project_file_fptr,"%i",&iSedConc[SizeFr]);
    printf("%i    ",iSedConc[SizeFr]);
  }
  printf("\n");
}
if(strcmp(file_id,"SUSPENDED") == 0 && indexeros == 1)
{
  printf("SUSPENDED       ");
  for(SizeFr=1;SizeFr<=NumFracts+1;SizeFr++)
  {
    fscanf(project_file_fptr,"%i",&iSusSed[SizeFr]);
    printf("%i    ",iSusSed[SizeFr]);
  }
  printf("\n");
}
```

112

```c
    if(strcmp(file_id,"DEPOSITED") == 0 && indexeros == 1)
    {
      printf("DEPOSITED       ");
      for(SizeFr=1;SizeFr<=NumFracts+1;SizeFr++)
      {
        fscanf(project_file_fptr,"%i",&iDepSed[SizeFr]);
        printf("%i     ",iDepSed[SizeFr]);
      }
      printf("\n");
    }
    if(strcmp(file_id,"FLUX") == 0 && indexeros == 1)
    {
      printf("FLUX            ");
      for(SizeFr=1;SizeFr<=NumFracts+1;SizeFr++)
      {
        fscanf(project_file_fptr,"%i",&iSedFlux[SizeFr]);
        printf("%i     ",iSedFlux[SizeFr]);
      }
      printf("\n");
    }
    if(strcmp(file_id,"NETEROSION") == 0 && indexeros == 1)
    { /* In reality, we are only interested in the last element of*/
      /* this vector since we only allow printing the net erosion */
      /* We leave this open in case that we want to print the net */
      /* erosion by size fraction down the road.                  */
      printf("NETEROSION      ");
      for(SizeFr=1;SizeFr<=NumFracts+1;SizeFr++)
      {
        fscanf(project_file_fptr,"%i",&iNetEros[SizeFr]);
        printf("%i     ",iNetEros[SizeFr]);
      }
      printf("\n");
    }

  } /* End Of: if(ch!=EOF) */

  }while(ch != EOF);

  if((debug =fopen("debug.txt","w"))==NULL)
  {printf("Can't open debug File \n");
  exit(EXIT_FAILURE);}


  printf("Successfully Read all data from the Project File\n\n");
  fclose(project_file_fptr);

}
```

```
                 /******************************/
                 /*         Reset.c          */
                 /******************************/

/* For each time step and cell resets the following variables to  */
/* zero:                                                          */
/*    t_SedFlux: Total sediment flux out of each cell             */
/*    SedFlux: Total volume of sediment out of each cell by       */
/*             size fraction                                      */

#include "all.h"

extern void Reset()
{
  int j,k,SizeFr;

  for(j=1; j<=nrows; j++)
  {
    for(k=1; k<=ncols; k++)
    {
      t_SusSed[j][k] = 0.0;
      t_DepSed[j][k] = 0.0;
      t_SedFlux[j][k] = 0.0;

      for (SizeFr=1;SizeFr<=NumFracts;SizeFr++)
      {
        SedFlux[SizeFr][j][k] = 0.0;
      }
    }
  }
}
```

```
                    /******************************/
                    /*        RoutOutlet.c        */
                    /******************************/


#include "all.h"

extern void RoutOutlet()
{
  int ill, outlink, outnode;
  float alfa,qoutch,wchout,dchout,rmanout,sfactorout;

  /* Initializes overland and channel discharges at the outlet    */
  qoutov = 0.0;
  qoutch = 0.0;

  /* Determines LINK and NODE numbers of the outlet channel cell  */
  outlink = link[jout][kout];
  outnode = node[jout][kout];
  /* Channel characteristics at the outlet                        */
  wchout = chp[outlink][outnode][2];
  dchout = chp[outlink][outnode][3];
  rmanout = chp[outlink][outnode][5];
  sfactorout = chp[outlink][outnode][6];


  /* FIRST:calculate the flow going out from the overl. portion    */

  alfa = (float)(sqrt(sovout)/pman[iman[jout][kout]]);

  /* Discharge from overland flow.  NOTE: because the water from   */
  /* this part of the outlet overland cell was already "poured"    */
  /* into the channel when updating the channel depth             */
  /* (channDepth.c) qoutov = 0 if the channel routing is selected  */

  if(h[jout][kout] > sdep[jout][kout])
  {
    qoutov =
      (float)(dx*alfa*pow((h[jout][kout]-sdep[jout][kout]),1.667));
  }

  /* Overland water depth at outlet cell is reduced after taking  */
  /* the outflow out of the cell                                  */

  h[jout][kout] = (float)(h[jout][kout] - qoutov*dt/(pow(dx,2.0)));

  /* SECOND:calculate the flow going out from the channel portion  */

  if(indexchan == 1 && hch[jout][kout] > sdep[jout][kout])
  {
    qoutch = chnDischarge(hch[jout][kout],wchout,dchout,
                          sdep[jout][kout],rmanout,1,sout,sfactorout);

    dqch[jout][kout] = dqch[jout][kout] - qoutch;
  }

  /* The total outflow at the basin's outlet is given by adding    */
  /* the outflow from the overland & channel portion of the cell  */

  qout = qoutov + qoutch;
```

```
    /* Keeping Track of the Total Outflow Volume                    */

    vout = vout + qout*dt;

    /* Checking to see if the Peak Flow has been reached            */

    if(qout > qpeak)
    {
     qpeak = qout;
     tpeak = (float)( iter*dt/60.0);
    }

    /* Populating the Output Flows at the Watershed Outlet          */
    if(indexdis == 1)
    {
      for(ill = 1;ill <= ndis; ill++)
      {
       if(jout == iq[ill][1] && kout == iq[ill][2])
       {
         q[ill] = qout;
       }
      }
    }
}
```

```
                /******************************/
                /*        RoutSedChn.c        */
                /******************************/


#include "all.h"

        /*********************************/
        /*        FUNCT: RoutSedChn      */
        /*********************************/

extern void RoutSedChn(float dq,float sf,int jfrom,int kfrom,
                       int jto,int kto,float wch,float hchan)
{
  int SizeFr;
  float TotQs,V,supply,*qs,adv_factor,
        *EHvol,*SUSvol, *XSScap, *BMvol;

  qs = (float*)vMAlloc(sizeof(float),NumFracts);
  EHvol= (float*)vMAlloc(sizeof(float),NumFracts);
  SUSvol = (float*)vMAlloc(sizeof(float),NumFracts);
  XSScap = (float*)vMAlloc(sizeof(float),NumFracts);
  BMvol = (float*)vMAlloc(sizeof(float),NumFracts);

  /* Total sediment volume out of outgoing cell is initialized  */
  TotQs = 0.0;

  /* Flow velocity (m/s)                                        */
  V = (float)(dq/(wch*hchan));


  for (SizeFr=1; SizeFr<=NumFracts;SizeFr++)
  {
    /* Sediment supply in the outgoing cell if found in suspension*/
    /* (**SusSed) and/or in deposition (**DepSed)                 */
    supply = SusSed[SizeFr][jfrom][kfrom] + DepSed[SizeFr][jfrom][kfrom];

    /* initialize qs to 0 in case that supply = 0  */
    qs[SizeFr] = 0;

    /* If there is sediment supply in the outgoing cell          */
    if (supply != 0)
    {
      /* Transport capacity [m3] for size fraction SizeFr  using    */
      /* the Engelund and Hansen equation                          */
      EHvol[SizeFr] = EHcap(SizeFr,wch,hchan,sf,dq);

      /* Suspended sediment transport capacity [m3] by advection  */
      /* Limit this transport by availability                     */

      adv_factor = (float)(MIN(V*dt/dx,1));

      SUSvol[SizeFr] = SusSed[SizeFr][jfrom][kfrom] * adv_factor;

      /* Transfers the volume SUSvol[SizeFr] of sediment size     */
      /* fraction, SizeFr in suspension in the outgoing cell      */
      /* to the suspended portion of the receiving cell           */

      TransferSed(SizeFr,SusSed,SUSvol[SizeFr],jfrom,kfrom,jto,kto);

      /* Excess capacity [m3] to move seds. from the bed-material  */

      XSScap[SizeFr] = (float)(MAX(0, EHvol[SizeFr] - SUSvol[SizeFr]));
```

117

```c
      /* Bed-material transport capacity [m3] by advection       */

      BMvol[SizeFr] = DepSed[SizeFr][jfrom][kfrom] * adv_factor;

      /* Volume of bed-material transported  out of (jfrom,kfrom)  */

      BMvol[SizeFr] = (float)(MIN(XSScap[SizeFr],BMvol[SizeFr]));

      /* Transfers the volume BMvol[SizeFr] of sediment size     */
      /* fraction, SizeFr in deposition in the outgoing cell     */
      /* to the suspended portion of the receiving cell          */

      TransferSed(SizeFr,DepSed,BMvol[SizeFr],jfrom,kfrom,jto,kto);


      /* Total volume of transported sediment from outgoing cell  */

      qs[SizeFr] = SUSvol[SizeFr] + BMvol[SizeFr];

      /* Updates total sediment volume leaving the outgoing cell  */

      t_SedFlux[jfrom][kfrom] += qs[SizeFr]/dt;

      /* Keeps track of sediment flux out of the outgoing cell    */

      SedFlux[SizeFr][jfrom][kfrom] += qs[SizeFr] / dt;

      /* Total sediment volume leaving the outgoing cell         */

      TotQs += qs[SizeFr];
    }
  }

  vfree(qs);
  vfree(EHvol);
  vfree(SUSvol);
  vfree(XSScap);
  vfree(BMvol );

}
```

```c
/*****************************/
/*         RoutSedOut.c         */
/*****************************/


#include "all.h"

extern void RoutSedOut()
{
  float wchout;

  /* Channel width at the outlet                              */
  wchout = chp[link[jout][kout]][node[jout][kout]][2];

  if(ishp[jout][kout] == 2 )  /* Case of a channel cell           */
  {
    if(hch[jout][kout] != 0)
    {
      RoutSedChn(qout,sout,jout,kout,1,1,wchout,hch[jout][kout]);
    }
  }

  else /* case of an overland cell                             */
  {
    if(h[jout][kout] != 0)
    {
      RoutSedOvrl(qoutov,sovout,jout,kout,1,1);
    }
  }
}
```

```
                    /*******************************/
                    /*        RoutSedOvrl.c        */
                    /*******************************/


#include "all.h"

extern void RoutSedOvrl(float dqq, float sf,int jfrom,int kfrom,
                                                int jto,int kto)
{
  float V, *qsSUS,*qsADV,qsSUStot,SUStot, DEPtot,*qsBM,
        totXSScap,*capacity,RESIDcap, qsBMtot,
        *qsEROS,qsEROStot,*qs,qsKR;

  int SizeFr,SoilType;

  qsSUS = (float*)vMAlloc(sizeof(float),NumFracts);
  qsADV = (float*)vMAlloc(sizeof(float),NumFracts);
  qsBM = (float*)vMAlloc(sizeof(float),NumFracts);
  capacity = (float*)vMAlloc(sizeof(float),NumFracts);
  qsEROS = (float*)vMAlloc(sizeof(float),NumFracts);
  qs = (float*)vMAlloc(sizeof(float),NumFracts);


  /* SizeFr: Soil size fraction (1... NumFracts)              */
  /* Outgoing cell : (jfrom, kfrom)                           */
  /* Receiving cell: (jto, kto)                               */

  /* Flow velocity [m/s]                                      */
  V = (float)(fabs(dqq)/(dx * h[jfrom][kfrom]));

  /* Initialize sediment volumes to zero                      */
  qsSUStot = 0;      /* Total transported suspended volume    */
  qsBMtot = 0;       /* Total transported bed material volume  */
  qsEROStot = 0;    /* Total transported eroded material volume */
  SUStot = 0;        /* Total susp. volume in outgoing cell   */
  DEPtot = 0;        /* Total deposited volume in outgoing cell  */

  /* Finds soil type for the outgoing overland cell           */
  SoilType = isoil[jfrom][kfrom];

  /* Transport Capacity using the Kilinc-Richardson equation   */
  qsKR = KRcap(KRov,dqq,dx,sf,jfrom,kfrom);

  /* Initialize transport volumes by size fraction to zero     */
  for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
  {                      /* For size fraction SizeFr:          */
    qsSUS[SizeFr] = 0;  /* transported volume from suspension  */
    qsBM[SizeFr] = 0 ;  /* transported volume from deposition  */
    qsEROS[SizeFr] = 0; /* transported volume from parent material*/
    qs[SizeFr] = 0;     /* total transported volume            */
    /* Outgoing cell total suspended and deposited sediment    */
    SUStot += SusSed[SizeFr][jfrom][kfrom];
    DEPtot += DepSed[SizeFr][jfrom][kfrom];

  }
```

```c
for(SizeFr=1;SizeFr<=NumFracts;SizeFr++) /* For each size fract.*/
{
  /* Is this size fraction present in suspended portion ?      */
  if(SusSed[SizeFr][jfrom][kfrom] > 0)
  {
    if(qsKR < SUStot) /* Transport capacity is < total suspended*/
    {
      /* Volume that can be transported using the KR equation  */
      capacity[SizeFr] = qsKR * SusSed[SizeFr][jfrom][kfrom]/SUStot;
      /* Volume that can be transported by advective processes */
      qsADV[SizeFr] = SusSed[SizeFr][jfrom][kfrom] * V * dt / dx;
      /* Transport the maximum between the last two quantities */
      qsSUS[SizeFr] = (float)(MAX(capacity[SizeFr],qsADV[SizeFr]));
    }
    else /* If transport capacity is > total suspended sediment */
    {
      /* Transport all the suspended sediment for this size fr. */
      qsSUS[SizeFr] = SusSed[SizeFr][jfrom][kfrom];
    }

    /* Transfers the volume qsSUS[SizeFr] of sediment size      */
    /* fraction, SizeFr, in suspension in the outgoing cell     */
    /* to the suspended portion of the receiving cell           */
    TransferSed(SizeFr,SusSed,qsSUS[SizeFr],jfrom,kfrom,jto,kto);

    /* Keeps track of the total volume of sediment coming from  */
    /* the suspended material portion of the outgoing cell      */
    qsSUStot += qsSUS[SizeFr];
  }
}

/* Reduces the transport capacity by the volume that has already*/
/* been transported from the outgoing cell to get the total     */
/* excess capacity                                              */
totXSScap = (float)(MAX(0,qsKR - qsSUStot));

/* If there is an excess transport capacity, and provided that  */
/* the outgoing cell has previously deposited material, we use  */
/* this capacity to put this sediment in suspension and move it */
/* to the suspended portion of the receiving cell               */
if(totXSScap > 0 && DEPtot > 0)
{
  for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
  {
  /* Is this size fraction present in deposited portion ?       */
    if(DepSed[SizeFr][jfrom][kfrom] > 0)
    {
      /* Excess transport capacity is < total deposited sediment*/
      if(totXSScap < DEPtot)
      {
        /* Volume that can be transported for this size         */
        /* fraction is proportional to its percentage in the    */
        /* total deposited sediment                             */
        qsBM[SizeFr] =
              totXSScap * DepSed[SizeFr][jfrom][kfrom] / DEPtot;
      }
      else /* If transport capacity is > total deposited sed.   */
      {
        /* Transport all the deposited sed. for this size fr.   */
        qsBM[SizeFr] = DepSed[SizeFr][jfrom][kfrom];
      }
```

```c
        /* Transfers the volume qsBM[SizeFr] of sediment size    */
        /* fraction, SizeFr, in deposition in the outgoing cell   */
        /* to the suspended portion of the receiving cell         */
          TransferSed(SizeFr,DepSed,qsBM[SizeFr],jfrom,kfrom,jto,kto);

        /* Keeps track of the total volume of sediment leaving    */
        /* the deposited material portion of the outgoing cell    */
        qsBMtot += qsBM[SizeFr];
        }
    }
  }

  /* Reduces the excess transport capacity by the total volume    */
  /* of sediment, qsBMtot,that has alrady been transported        */
  /* from the deposited portion of the outgoing cell              */
  RESIDcap = (float)(MAX(0,totXSScap - qsBMtot));

  /* Any residual transport capacity is used to erode the parent  */
  /* material.  Erosion by size fraction is proportional to its   */
  /* percentage in the parent material                            */
  if(RESIDcap >0)
  {
    for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
    {
      /* Volume of eroded parent material corresponding to size   */
      /* fraction SizeFr                                          */
      qsEROS[SizeFr] = RESIDcap * pfraction[SoilType][SizeFr];

      /* Keeps track of the total volume of sediment coming from  */
      /* the parent material portion of the outgoing cell         */
      qsEROStot += qsEROS[SizeFr];

      /* Transfers the volume qsEROS[SizeFr] of sediment size     */
      /* fraction, SizeFr from the outgoing cell parent material  */
      /* to the suspended portion of the receiving cell           */
      TransferSed(SizeFr,ErodSed,qsEROS[SizeFr],jfrom,kfrom,jto,kto);
    }
  }

  /* Keeps track of the total sediment volume leaving the         */
  /* outgoing cell (coming from all 3 sources)                    */
  t_SedFlux[jfrom][kfrom] += qsSUStot + qsBMtot + qsEROStot;

  for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
  {
    /* Total sediment volume leaving the outgoing cell by size    */
    /* fraction                                                   */
    qs[SizeFr] = qsSUS[SizeFr] + qsBM[SizeFr] + qsEROS[SizeFr];

    /* Keeps track of the sediment flux [m3/s] by size fraction   */
    /* out of the outgoing cell                                   */
    SedFlux[SizeFr][jfrom][kfrom] += qs[SizeFr] / dt;
  }

  vfree(qsSUS);
  vfree(qsADV);
  vfree(qsBM);
  vfree(capacity);
  vfree(qsEROS);
  vfree(qs);

}
```

```
/*****************************/
/*          RunTime.c        */
/*****************************/


#include "all.h"

extern void  RunTime(clock_t finish)
{

/*  struct tm when;
  time_t now;*/
  double elapsedTime;

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}

  fprintf(Summ_file_fptr,
    "\nProgram stops at simulation minute:%10.2f\n",(iter-1)*dt/60);

  elapsedTime = (double)(finish - startTime)/CLOCKS_PER_SEC;

  fprintf(Summ_file_fptr,
    "\nCASC2D RUNNING TIME:%10.2f minutes\n", elapsedTime/60);

  /*when = *localtime(&now);

  fprintf(Summ_file_fptr,
    "\nCurrent time is: %s \n", asctime(&when) );*/

  fclose(Summ_file_fptr);
}
```

```c
/******************************/
/*          SedStats.c        */
/******************************/


#include "all.h"

extern void SedStats()
{
  int j,k, SizeFr;
  float totscourv, hard_Cd, hard_Cu, hard_Zn;

  for(j = 1;j <= nrows; j++)
  {
    for(k = 1;k <= ncols; k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        totscourv = 0.0;

        for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
        {
          /* Total suspended sed. volume (m3)                */
          t_SusSed[j][k] += SusSed[SizeFr][j][k];

          /* Total deposited sed. volume (m3)                */
          t_DepSed[j][k] += DepSed[SizeFr][j][k];

          /* Total scoured sediment volume (m3)             */
          totscourv += ErodSed[SizeFr][j][k];
        }

        /* Net (deposited - eroded) sediment volume (mm)       */

        t_NetEros[j][k] = (t_DepSed[j][k] + totscourv) * 1000 / (dx*dx);

        /* Elevation changes -> modify the elevation grid (in m)  */
        /* NOTE: The DEM at the end of the simulation might have  */
        /* changed.  We need a print out of the new elevation map */

        if(ishp[j][k] == 1)
          e[j][k]  += (float)(t_NetEros[j][k] * 0.001);
        else
          ec[j][k] += (float)(t_NetEros[j][k] * 0.001);

        /* Absolute max. flux conc., susp. volume and suspended    */
        /* conc. at any time step and at any overland cell        */

        if(ishp[j][k] == 1)
        {
          amaxSusOv = MAX(t_SusSed[j][k],amaxSusOv);

          /* Limiting the mim. water depth to 0.5mm will reduce    */
          /* very high concentration calculations                 */
          /* Total suspended concentration in [mg/l]              */
```

```
         if(h[j][k] > 5E-4)
         {
           for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
           {
             /* Suspended sediment concentration [mg/l]          */
             SedConc[SizeFr][j][k] = (float)
               (SusSed[SizeFr][j][k] * G *1E6 / (h[j][k]*dx*dx) );
           }

           t_SedConc[j][k] = (float)(t_SusSed[j][k]* G*1E6
                                          / (h[j][k]*dx*dx) );
           amaxSusCov = MAX(amaxSusCov,t_SedConc[j][k]);
         }
         else
         {
           for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
           {
             /* Suspended sediment concentration [mg/l]          */
             SedConc[SizeFr][j][k] = 0.0;
           }
           t_SedConc[j][k] = 0.0;
         }
       }

       /* Absolute max. flux conc., susp. volume and suspended    */
       /* conc. at any time step and at any channel  cell         */
       /* Total suspended concentration in [mg/l]                 */

       if(ishp[j][k] == 2)
       {
         amaxSusCh = MAX(amaxSusCh,t_SusSed[j][k]);

         if(hch[j][k] > 5E-4)
         {
           for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
           {
             /* Suspended sediment concentration [mg/l]          */
             SedConc[SizeFr][j][k] = (float)(SusSed[SizeFr][j][k] * G *1E6 /
               (hch[j][k]*dx*chp[link[j][k]][node[j][k]][2]) );
           }
           t_SedConc[j][k] = (float)(t_SusSed[j][k] * G*1E6 /
             (hch[j][k]*dx*chp[link[j][k]][node[j][k]][2]) );

           amaxSusCch = MAX(amaxSusCch,t_SedConc[j][k]);
         }
         else
         {
           for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
           {
             /* Suspended sediment concentration [mg/l]          */
             SedConc[SizeFr][j][k] = 0.0;
           }
           t_SedConc[j][k] = 0;
         }
       }
     }
   }
 }
}
```

```c
/*******************************/
/*       SedVolumes.c        */
/*******************************/

/* Calculates at the end of the simulation the volume of       */
/* suspended, deposited and (suspended+deposited) sediment      */
/* by size fraction remaining in the overland or the channels   */
/* All the units are in m3                                      */

#include "all.h"

extern void  SedVolumes()
{
  int j,k,SizeFr;

  for(j=1;j<=nrows;j++)
  {
    for(k=1;k<=ncols;k++)
    {
      if(ishp[j][k] != nodatavalue)
      {
        /* Total eroded sediment by size fraction              */
        for(SizeFr =1;SizeFr<=NumFracts;SizeFr++)
         tot_eroded[SizeFr] += ErodSed[SizeFr][j][k];

        if(ishp[j][k] == 1)  /* for the overland cells          */
        {
          for(SizeFr =1;SizeFr<=NumFracts;SizeFr++)
          {
            /* Total suspended volume by size fraction          */
            sus_ov[SizeFr] += SusSed[SizeFr][j][k];

            /* Total deposited volume by size fraction          */
            dep_ov[SizeFr] += DepSed[SizeFr][j][k];

            /* Total suspended and deposited volume by size fract.*/
            tot_ov[SizeFr] += SusSed[SizeFr][j][k] +
                            DepSed[SizeFr][j][k];
          }
        }

        if(ishp[j][k] == 2)  /* for the channel cells           */
        {
          for(SizeFr =1;SizeFr<=NumFracts;SizeFr++)
          {
            /* Total suspended volume by size fraction          */
            sus_ch[SizeFr] += SusSed[SizeFr][j][k];

            /* Total deposited volume by size fraction          */
            dep_ch[SizeFr] += DepSed[SizeFr][j][k];

            /* Total suspended and deposited volume by size fract.*/
            tot_ch[SizeFr] += SusSed[SizeFr][j][k] +
                            DepSed[SizeFr][j][k];
          }
        }
      }
    }
  }
}
```

```
                    /*******************************/
                    /*          Settling.c         */
                    /*******************************/

/* At each time step and by size fraction, calculates the volume  */
/* of sediment in suspension that deposits.  This volume is sub-   */
/* tracted from the suspended portion and added to the deposited   */
/* portion                                                         */

#include "all.h"

extern void Settling()
{
  int j,k,SizeFr;
  float depth, *settlPerc, *settl;

  settlPerc = (float*)vMAlloc(sizeof(float),NumFracts);
  settl = (float*)vMAlloc(sizeof(float),NumFracts);

  for(j=1;j<=nrows;j++)
  {
   for(k=1;k<=ncols;k++)
   {
     if(ishp[j][k] != nodatavalue)
     {
        if (ishp[j][k] == 1) depth = h[j][k];
        if (ishp[j][k] == 2) depth = hch[j][k];

        if(depth > 0)
        {
           for (SizeFr=1;SizeFr<=NumFracts;SizeFr++)
           {
             /* Settling is calculated for  each size fraction    */

             if(depth > ws[SizeFr]*dt)
             {
               settlPerc[SizeFr] = (float)(ws[SizeFr] * dt / depth);
             }
             else
             {
               settlPerc[SizeFr] = 1;
             }

             settl[SizeFr] = SusSed[SizeFr][j][k] * settlPerc[SizeFr];
             DepSed[SizeFr][j][k] += settl[SizeFr];
             SusSed[SizeFr][j][k] -= settl[SizeFr];
          }
        }
     }
    }
   }
  }
  vfree(settlPerc);
  vfree(settl);

}
```

```
/******************************/
/*   WriteErrorFreeMem.c       */
/******************************/

/* This function
/* 1. Appends an error message to the "ErrorMsg.txt" file in case */
/*    that the program stops because of a negative depth in cell  */
/*    This is useful when a batch of projects are run consecutiv. */
/* 2. Frees allocated memory by the program                       */
/* 3. Stops program execution                                     */

#include "all.h"

extern void WriteErrorFreeMem(int row, int col,
                              float OldDepth, float NewDepth)
{
  FILE *ErrorFile_fptr;

  /* Opens text file to append the error message                 */
  if((ErrorFile_fptr=fopen("ErrorMsg.txt","a"))==NULL)
  {printf("Can't open the error file ErrorMsg.txt\n");
  exit(EXIT_FAILURE);}

  fprintf(ErrorFile_fptr,
    "Program stopped running project file %s \n\n",Project_file);

  if(ishp[row][col] == 1) /* The cell is in the overland        */
    fprintf(ErrorFile_fptr,
      "A negative Depth has ocurred in an overland cell \n");
  else                    /* The cell is in the channel network  */
  {
    fprintf(ErrorFile_fptr,
      "A negative Depth has ocurred in a channel cell \n");
    fprintf(ErrorFile_fptr,"Time Step: %f \n",dt);
  }

   fprintf(ErrorFile_fptr,"Simulation time = %g min. \n",
                                        iter*dt/60);

  fprintf(ErrorFile_fptr,"Grid cell row: %i column: %i \n",row,col);

  fprintf(ErrorFile_fptr,
      "Water depth in the last time step was: %g m.\n",OldDepth);

  fprintf(ErrorFile_fptr,
      "Calculated depth in current time step is: %g m.\n",NewDepth);

  fprintf(ErrorFile_fptr,"Rainfall Intensity at [%ld][%ld] = %g \n",
                                      row,col,rint[row][col]);

  /* Free all allocated memory during program execution          */
  MemFree();

  /* Program stops execution with code 0                         */
  exit(0);

  fclose(ErrorFile_fptr);
}
```

```
                 /*****************************/
                 /*         WriteGrids.c      */
                 /*****************************/


#include "all.h"

extern void WriteGrids(int iter)
{
  int j1,k1,SizeFr;
  float **WatDep;
  char SedConcName[110],tSedConcName[110],SusSedName[110],tSusSedName[110],
    DepSedName[110],tDepSedName[110],SedFluxName[110],tSedFluxName[110],
    tNetErosName[110], CCUname[110], maxCCUname[110];

  sprintf(SedConcName,"%sSedConc_",SedGridsPath);
  sprintf(tSedConcName,"%sSedConc_t",SedGridsPath);
  sprintf(SusSedName,"%sSusSed_",SedGridsPath);
  sprintf(tSusSedName,"%sSusSed_t",SedGridsPath);
  sprintf(DepSedName,"%sDepSed_",SedGridsPath);
  sprintf(tDepSedName,"%sDepSed_t",SedGridsPath);
  sprintf(SedFluxName,"%sSedFlux_",SedGridsPath);
  sprintf(tSedFluxName,"%sSedFlux_t",SedGridsPath);
  sprintf(tNetErosName,"%sNetEros_t",SedGridsPath);
  sprintf(CCUname,"%sCCU",SedGridsPath);
  sprintf(maxCCUname,"%smaxCCU",SedGridsPath);

  WatDep = (float**)mMAlloc(sizeof(float),nrows,ncols);

  if(ipcount == nplt)
  {
    printf("Writing Output Grids, IFCOUNT = %ld  Time = %.2f \n",
                                          ifcount,iter*dt/60.);

    /* WatDep: water depth in basin's overland or channel cells    */

    for(j1=1; j1<=nrows; j1++)
    {
      for(k1=1; k1<=ncols; k1++)
      {
        if (ishp[j1][k1] != nodatavalue)
        {
          if(ishp[j1][k1] == 1)
          {
            WatDep[j1][k1] = h[j1][k1];
          }
          else
          {
            WatDep[j1][k1] = hch[j1][k1];
          }
        }
      }
    }

    /* Writing time series grids                                   */

    /* Water depth in meters                                       */
    if(dname[0] != '\0') write2dTS(dname, WatDep,1);

    /* Rainfall intensity in mm/h                                  */
    if(rname[0] != '\0') write2dTS(rname, rint,3600000);
```

```c
  /* Infiltrated depth in mm                                        */
  if(indexinf == 1 && vinfname[0] != '\0') write2dTS(vinfname,vinf,1000);

  if(indexeros == 1)
  {
    /* Total suspended sediment concentration in mg/l             */
    if(iSedConc[NumFracts+1] == 1) write2dTS(tSedConcName,t_SedConc,1);

    /* Total suspended sediment in m3                             */
    if(iSusSed[NumFracts+1] == 1) write2dTS(tSusSedName,t_SusSed,1);

    /* Total deposited sediment in m3                             */
    if(iDepSed[NumFracts+1] == 1) write2dTS(tDepSedName,t_DepSed,1);

    /* Total sediment flux out in m3/s                           */
    if(iSedFlux[NumFracts+1] == 1) write2dTS(tSedFluxName,t_SedFlux,1);

    /* Net Erosion in mm                                         */
    if(iNetEros[NumFracts+1] == 1) write2dTS(tNetErosName,t_NetEros,1);

    if(NumFracts > 3) write2dTS(CCUname,CCU,1);

    for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
    {
      /* Sediment concentration by size fraction in mg/l         */
      if(iSedConc[SizeFr]==1)
        write3dTS(SedConcName,SizeFr,SedConc,1);
      /* Suspended sediment by size fraction in gr/m2            */
      if(iSusSed[SizeFr]==1)
        write3dTS(SusSedName,SizeFr,SusSed,(float)(G*1E6/(dx*dx)));
      /* Deposited sediment by size fraction in gr/m2            */
      if(iDepSed[SizeFr]==1)
        write3dTS(DepSedName,SizeFr,DepSed,(float)(G*1E6/(dx*dx)));
      /* Sediment flux by size fraction in m3/s                  */
      if(iSedFlux[SizeFr]==1)
        write3dTS(SedFluxName,SizeFr,SedFlux,1);
    }

  }

  /* Incrementing counters                                        */
  ifcount++;
  if (iter == 1) ipcount++;
  if (iter != 1) ipcount = 1;
  }
  else
  {
    ipcount++;
  }

  /* At the end of the simulation prints out the grid with the   */
  /* maximum values of CCU                                       */
  if(indexeros == 1 && NumFracts > 3 && iter == niter )
    write2dTS(maxCCUname,maxCCU,1);

  /* Freeing memory                                              */

  mfree(WatDep);

}
```

```c
/*********************************/
/*         FUNCT: write2dTS      */
/*********************************/

extern void write2dTS(char name[],
               float **ParamValue, float UnitsConv)
{
  int j1,k1;

  char wr_name[100] ;

  FILE *wr_name_h = NULL;

  wr_name[0] = '\0';

  strncpy(wr_name,name,21);
  sprintf(wr_name,"%s.%ld",name,ifcount);
  if((wr_name_h=fopen(wr_name,"w"))==NULL)
  {
    printf("Can't open time-series file : %s \n",wr_name);
    exit(EXIT_FAILURE);
  }
  fprintf(wr_name_h,"%s",header);

  for(j1=1; j1<=nrows; j1++)
  {
    for(k1=1; k1<=ncols; k1++)
    {
      if (ishp[j1][k1] != nodatavalue)
      {
        fprintf(wr_name_h,"%10.5f ",ParamValue[j1][k1]*UnitsConv);
      }
      else
        fprintf(wr_name_h,"%i ",nodatavalue);
    }
    fprintf(wr_name_h,"\n");
  }

  fclose(wr_name_h);

  return;

}
```

```c
/*********************************/
/*        FUNCT: write3dTS       */
/*********************************/

extern void write3dTS(char *name, int fraction,
                      float **ParamValue[], float UnitsConv)
{
  int j1,k1;

  char wr_name[100] ;

  FILE *wr_name_h = NULL;

  wr_name[0] = '\0';

  strncpy(wr_name,name,21);
  sprintf(wr_name,"%s%i.%i",name,fraction,ifcount);

  if((wr_name_h=fopen(wr_name,"w"))==NULL)
  {
    printf("Can't open time-series file : %s \n",wr_name);
    exit(EXIT_FAILURE);
  }
  fprintf(wr_name_h,"%s",header);

  for(j1=1; j1<=nrows; j1++)
  {
    for(k1=1; k1<=ncols; k1++)
    {
      if (ishp[j1][k1] != nodatavalue)
      {
        fprintf(wr_name_h,"%g ",ParamValue[fraction][j1][k1]*UnitsConv);
      }
      else
        fprintf(wr_name_h,"%i ",nodatavalue);
    }
    fprintf(wr_name_h,"\n");
  }

  fclose(wr_name_h);

  return;

}
```

132

```
                      /*****************************/
                      /*      WriteOutflows.c       */
                      /*****************************/


#include "all.h"

extern void WriteOutflow()
{
  int ill,ils,jsed,ksed,i, SizeFr;

  /*****************************/
  /*   write results to screen  */
  /*****************************/
  if(indexchan == 1)
  {
    printf(
    "Time (Min) = %7.2f\tQ (m3/s)= %7.2f\tChan. Depth(m) = %7.3f \n"
                        ,iter*dt/60.0, qout,hch[jout][kout]);
  }
  else
  {
    printf(
    "Time (Min) = %7.2f\tQ (m3/s) = %7.2f\tOver. Depth(m) = %7.3f\n"
                        ,iter*dt/60.0,qout,h[jout][kout]);
  }

  /*****************************************************/
  /*   write outflow and sediment flow to output files  */
  /*****************************************************/

  /* The first time, write the headers of the Q and Qs files */
  if(iter==1)
  {
    fprintf(dis_out_file_fptr," Time_(min) ");
    if(indexeros == 1)fprintf(sed_out_file_fptr,"Time_(min) ");
    if(indexdis == 1)
    {
      for (i=1;i<=ndis;i++)
      {
        fprintf(dis_out_file_fptr,"Row%iCol%i ",iq[i][1],iq[i][2]);
        if(indexeros == 1 && indexsed == 1)
            fprintf(sed_out_file_fptr,"Row%iCol%i ",
                                          ised[i][1],ised[i][2]);
      }
    }
    fprintf(dis_out_file_fptr,"\n");
    if(indexeros == 1)fprintf(sed_out_file_fptr,"\n");
  }

  if(icount == nprn)
  {
    /* Write outflow discharge at selected locations            */

    fprintf(dis_out_file_fptr,"%.2f",iter*dt/60.0);

    for(ill=1; ill<=ndis; ill++)
    {
      switch(unitsQ)
      {
          case 1: /* Discharge in m3/s */
            fprintf(dis_out_file_fptr,"%15.3f ",q[ill]);
            break;
```

133

```
            case 2: /* Discharge in cfs */
              fprintf(dis_out_file_fptr,"%15.3f ",
                                  q[ill]* pow(3.28084,3));
              break;

            case 3: /* Discharge in mm/h */
              fprintf(dis_out_file_fptr,"%15.3f ",
                              q[ill] / areaQ[ill] * 360);
              break;
        }
      }

    fprintf(dis_out_file_fptr,"\n");

    /* Writing outflow sediment Data at selected locations      */

    if(indexeros == 1 && indexsed ==1 )
    {

      fprintf(sed_out_file_fptr,"%.2f",iter*dt/60.0);

      for(ils=1; ils<=nsed; ils++)
      {
          jsed = ised[ils][1];
          ksed = ised[ils][2];

          qsed[ils] = t_SedFlux[jsed][ksed];

          switch(unitsQs)
          {
           case 1: /* Sediment discharge in m3/s */
             fprintf(sed_out_file_fptr,"%15.5f ", qsed[ils]);
             break;
           case 2: /* Sediment discharge in tons/ha/day */
             fprintf(sed_out_file_fptr,"%15.5f ",
                       qsed[ils] / areaQs[ils] * G * 86400);
             break;
          }
      }
      fprintf(sed_out_file_fptr,"\n");

      /* Print sediment discharge at outlet by size fraction */
      fprintf(debug,"%.2f \t",iter*dt/60);
      for(SizeFr=1;SizeFr<=NumFracts;SizeFr++)
        fprintf(debug,"%g \t", SedFlux[SizeFr][jout][kout]);
      fprintf(debug,"%g \t", qout);
      fprintf(debug,"%g", CCU[jout][kout]);
      fprintf(debug,"\n");

    }

      icount = 1;
  }

  else
  {
    icount = icount + 1;
  }

}
```

```
              /******************************/
              /*      WriteSummFlow.c        */
              /******************************/


#include "all.h"

extern void WriteSummFlow()
{

  double PercentFlow,InitialVol,FinalVol,InVolume,OutVolume;

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}

  if(indexinf == 0)
  {
    aminvinf = 0;
    amaxvinf = 0;
  }
  if(indexchan == 0)
  {
    amincdepth = 0;
    amaxcdepth = 0;
  }


  InitialVol = init_ch_vol   +   init_ov_vol;
  FinalVol   = final_ch_vol  +   final_ov_vol;
  InVolume   = InitialVol + vin;
  OutVolume = vinftot + vout + FinalVol;

  PercentFlow =  100.0 * (InVolume - OutVolume) / InVolume;

  /* ... summary of Output Flow */

  fprintf(Summ_file_fptr,
                         "\nSUMMARY OF FLOW OUTPUT\n");
  fprintf(Summ_file_fptr,"======================\n\n");

  fprintf(Summ_file_fptr,
    "Peak Discharge (m3/s)....................................=");
  fprintf(Summ_file_fptr,"%15.2f \n",  qpeak);

  fprintf(Summ_file_fptr,
    "Time to Peak (min).......................................=");
  fprintf(Summ_file_fptr,"%15.2f \n",tpeak);

  fprintf(Summ_file_fptr,
    "Initial Surface Volume (m3)..............................=");
  fprintf(Summ_file_fptr,"%15.2f \n",  InitialVol);

  fprintf(Summ_file_fptr,
    "Volume of Rainfall - retention, V_in (m3)................=");
  fprintf(Summ_file_fptr,"%15.2f \n",  vin);

  fprintf(Summ_file_fptr,
    "Volume leaving the Watershed, V_out (m3).................=");
  fprintf(Summ_file_fptr,"%15.2f \n",vout);

  fprintf(Summ_file_fptr,
    "Percentage of V_out to V_in..............................=");
```

```
      fprintf(Summ_file_fptr,"%15.2f \n",(vout/vin)*100.0);


      fprintf(Summ_file_fptr,
        "Final Surface Volume, V_final (m3).........................=");
      fprintf(Summ_file_fptr,"%15.2f \n",  FinalVol);

      fprintf(Summ_file_fptr,
        "Percentage of V_final to V_in..............................=");
      fprintf(Summ_file_fptr,"%15.2f \n",(FinalVol/vin)*100.0);

      fprintf(Summ_file_fptr,
        "Volume Infiltrated, V_inf (m3).............................=");
      fprintf(Summ_file_fptr,"%15.2f \n",vinftot);

      fprintf(Summ_file_fptr,
        "Percentage of V_inf to V_in................................=");
      fprintf(Summ_file_fptr,"%15.2f \n\n",  (vinftot/vin)*100.0);

      fprintf(Summ_file_fptr,
        "Percent Mass Balance Error ............................... =");
      fprintf(Summ_file_fptr,"%15.2f\n",PercentFlow);



      fprintf(Summ_file_fptr,
        "\nHYDROLOGICAL VARIABLES MINIMUM AND MAXIMUM VALUES\n");
      fprintf(Summ_file_fptr,
        "================================================\n\n");

      fprintf(Summ_file_fptr,
        "Min. Rainfall Intensity (mm/hr)............................=");
      fprintf(Summ_file_fptr,"%15.2f \n",  aminrain*3600.*1000);

      fprintf(Summ_file_fptr,
        "Max. Rainfall Intensity (mm/hr)............................=");
      fprintf(Summ_file_fptr,"%15.2f \n",  amaxrain*3600.*1000);

      fprintf(Summ_file_fptr,
        "Min. Infiltration Depth (mm)...............................=");
      fprintf(Summ_file_fptr,"%15.2f \n",  aminvinf*1000);

      fprintf(Summ_file_fptr,
        "Max. Infiltration Depth (mm)...............................=");
      fprintf(Summ_file_fptr,"%15.2f \n",  amaxvinf*1000);

      fprintf(Summ_file_fptr,
        "Min. Overland Depth (m)....................................=");
      fprintf(Summ_file_fptr,"%15.2f \n",amindepth);

      fprintf(Summ_file_fptr,
        "Max. Overland Depth (m)....................................=");
      fprintf(Summ_file_fptr,"%15.2f \n",amaxdepth);

      fprintf(Summ_file_fptr,
        "Min. Channel Depth (m).....................................=");
      fprintf(Summ_file_fptr,"%15.2f \n",  amincdepth);

      fprintf(Summ_file_fptr,
        "Max. Channel Depth (m).....................................=");
      fprintf(Summ_file_fptr,"%15.2f \n\n",amaxcdepth);

      fclose(Summ_file_fptr);
}
```

136

```
              /*******************************/
              /*       WriteSummSed.c        */
              /*******************************/


#include "all.h"

extern void WriteSummSed()
{
  int SizeFr;

  double PercentSed,TotRemOvr,TotRemChn,TotEroded,TotLeaving,
         TotSusRemOvr,TotDepRemOvr,TotSusRemChn,TotDepRemChn;

  struct gstats t_NetEros_stats;

  if((Summ_file_fptr=fopen(Summ_file,"a"))==NULL)
  {printf("Can't open Output PRN File : %s \n",Summ_file);
  exit(EXIT_FAILURE);}

  t_NetEros_stats = gridstats(t_NetEros);

  TotEroded    = 0.0;
  TotSusRemOvr = 0.0;
  TotDepRemOvr = 0.0;
  TotRemOvr    = 0.0;
  TotSusRemChn = 0.0;
  TotDepRemChn = 0.0;
  TotRemChn    = 0.0;
  TotLeaving   = 0.0;

  for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
  {
    TotEroded    += fabs(tot_eroded[SizeFr]);
    TotSusRemOvr += sus_ov[SizeFr];
    TotDepRemOvr += dep_ov[SizeFr];
    TotRemOvr    += tot_ov[SizeFr];
    TotSusRemChn += sus_ch[SizeFr];
    TotDepRemChn += dep_ch[SizeFr];
    TotRemChn    += tot_ch[SizeFr];
    TotLeaving   += sed_out[SizeFr];
  }

  PercentSed = 100.0 * (TotLeaving + TotRemOvr + TotRemChn
                        - TotEroded) / TotEroded ;

  /* ... summary of output sediment                          */

  fprintf(Summ_file_fptr,
    "\nSUMMARY OF SEDIMENT OUTPUT (by size fraction and total)\n");
  fprintf(Summ_file_fptr,
    "Volumes are in cubic meters\n");
  fprintf(Summ_file_fptr,
    "===============================================\n\n");

  /* Total eroded sediment by size fraction                  */
  fprintf(Summ_file_fptr,"Total_Erod......>");
  for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
    fprintf(Summ_file_fptr,"%15.2f  ",fabs(tot_eroded[SizeFr]));
  fprintf(Summ_file_fptr,"...total: %g \n\n",TotEroded);
```

```c
  fprintf(Summ_file_fptr,
          "Volumes of Eroded Sediment Remaining on the Overland:\n");

 fprintf(Summ_file_fptr,
   "---------------------------------------------------\n");

 fprintf(Summ_file_fptr,"In Suspension...>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(sus_ov[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotSusRemOvr);

 fprintf(Summ_file_fptr,"Deposited.......>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(dep_ov[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotDepRemOvr);


 fprintf(Summ_file_fptr,"Total..........>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(tot_ov[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotRemOvr);


  fprintf(Summ_file_fptr,
          "Volumes of Eroded Sediment Remaining in the Channels:\n");
  fprintf(Summ_file_fptr,
          "---------------------------------------------------\n");

 fprintf(Summ_file_fptr,"In Suspension...>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(sus_ch[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotSusRemChn);

 fprintf(Summ_file_fptr,"Deposited.......>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(dep_ch[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotDepRemChn);


 fprintf(Summ_file_fptr,"Total..........>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(tot_ch[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotRemChn);


 fprintf(Summ_file_fptr,
   "Volumes of Eroded Sediment Leaving the Watershed:\n");
 fprintf(Summ_file_fptr,
   "---------------------------------------------------\n");


 fprintf(Summ_file_fptr,"Total..........>");
 for(SizeFr = 1; SizeFr<=NumFracts; SizeFr++)
   fprintf(Summ_file_fptr,"%15.2f   ",fabs(sed_out[SizeFr]));
 fprintf(Summ_file_fptr,"...total: %g \n\n",TotLeaving);


 fprintf(Summ_file_fptr,
   "\nPercent Sediment Mass Balance Error = %10.2f \n\n", PercentSed);

 /* ... Minimum and Maximum Values for Various Variables          */
```

```
    fprintf(Summ_file_fptr,
            "\nSEDIMENT VARIABLES MINIMUM AND MAXIMUM VALUES\n");
    fprintf(Summ_file_fptr,
            "=========================================\n\n");

    fprintf(Summ_file_fptr,
      "Max. Suspended Sediment in Overland (mm)..................=");
    fprintf(Summ_file_fptr,"\t%g \n", amaxSusOv*1000/(dx*dx));

    fprintf(Summ_file_fptr,
      "Max. Suspended Sediment in Channels (mm)..................=");
    fprintf(Summ_file_fptr,"\t%g \n", amaxSusCh*1000/(dx*dx));

    fprintf(Summ_file_fptr,
      "Max. Concentration on Overland (mg/l).....................=");
    fprintf(Summ_file_fptr,"\t%g \n", amaxSusCov);

    fprintf(Summ_file_fptr,
      "Max. Concentration in Channels (mg/l).....................=");
    fprintf(Summ_file_fptr,"\t%g \n", amaxSusCch);

    fprintf(Summ_file_fptr,
      "Total Net Volume (mm) at the End of the Simulation: \n");

    fprintf(Summ_file_fptr,
      "      Minimum Value ......................................=");
    fprintf(Summ_file_fptr,"%15.3f \n", t_NetEros_stats.min );

    fprintf(Summ_file_fptr,
      "      Maximum Value ......................................=");
    fprintf(Summ_file_fptr,"%15.3f \n",t_NetEros_stats.max );

    fprintf(Summ_file_fptr,
      "      Mean ...............................................=");
    fprintf(Summ_file_fptr,"%15.3f \n",t_NetEros_stats.mean );

    fprintf(Summ_file_fptr,
      "      Standard Deviation .................................=");
    fprintf(Summ_file_fptr,"%15.3f \n",t_NetEros_stats.stdev );

    fclose(Summ_file_fptr);
}
```