Appendix C: Program Code
C.1. Gross Erosion, Transport Capacity, and Routing Levels

Code

'This code conducts the following

1. Gross erosion based on RUSLE technology and RUSLE-2D
2. Sediment transport capacity based on the approaches of AMEENA and WaTEM/SEDEM
3. Assigns each grid cell a routing level

'This script assumes the following input grids are available

1. Watershed boundary
2. Digital Elevation Model
3. RUSLE C factor
4. RUSLE P factor
5. RUSLE K factor
6. RUSLE R factor
7. NRCS curve number
8. Manning’s roughness coefficient
9. 10 yr 24 hr storm depth

'The code starts here:

theView = av.GetActiveDoc
thePrj = theView.GetProjection

'fill DEM, this part of the code is from ArcView's system scripts

DEM = theView.FindTheme("Ned10pt29").GetGrid

'get the watershed boundary grid
boundary = theView.FindTheme("boundary10m").GetGrid

'get cell size

cellSize = boundary.getcellsize

'estimate cell size in [ha] and [m]

cellArea_ha = ((cellSize ^ 2) / 10000).asgrid * boundary
cellArea_m2 = ((cellSize ^ 2)).asgrid * boundary

'calculate slope percent
SlopePercent = (dem.slope(nil, true)) / 100.asgrid

theView.AddTheme(av.run("MakeGtheme", {SlopePercent, "SlopePercent"})))

'Estimate slope degree
SlopeDegree = dem.slope(nil, false)

'SlopeDegree = SlopeDegree.IsNull.Con(0.asGrid, SlopeDegree)

theView.AddTheme(av.run("MakeGtheme", {SlopeDegree, "SlopeDegree"})))

'Estimate slope radian
SlopeRadian = SlopeDegree * 3.14.asgrid / 180.asgrid

theView.AddTheme(av.run("MakeGtheme", {SlopeRadian, "SlopeRadian"})))

' estimate m coefficient for use in slope length estimation
beta = (SlopeRadian.sin / 0.0896.asgrid) / (3.asgrid * (SlopeRadian.sin).pow(0.8) + 0.56)
mCoef = beta / (1.asgrid + beta)

theView.addTheme(av.run("makeGtheme", {mCoef, "mCoef"})))

' estimate Aspect
Aspect = dem.aspect

Aspect = Aspect.IsNull.Con(0.asGrid, Aspect)

AspectRadian = Aspect * 3.14 / 180.asgrid

'estimate flow direction
d = DEM.flowdirection(false)

'estimate flow accumulation
FlowAccum = d.FlowAccumulation(NIL)
theView.AddTheme(av.run("MakeGtheme",{FlowAccum,"FlowAccum"}))

'populate flow direction of stream cells with 999 (which is not on of the 8 flow
directions) for cells with drainage area of 1.8 ha

d = (cellArea_ha >= 1.8.asgrid).Con(999.asgrid,d)

theView.AddTheme(av.run("MakeGtheme",{d,"d"}))

'estimate drainage area in [m^2]

upslopeArea_m2 = FlowAccum * cellArea_m2

theView.AddTheme(av.run("MakeGtheme",{upslopeArea_m2,"upslopeArea_m2"}))

'for each grid cell, estimate the flow direction of the downstream cell

m = ((d=32) *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{1,0,0},{0,0,0},{0,0,0}}), false)))

+((d=64) *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,1,0},{0,0,0},{0,0,0}}), false)))

+((d=128)*(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,1},{0,0,0},{0,0,0}}), false)))

+((d=16) *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{1,0,0},{0,0,0}}), false)))

+((d=1)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,1},{0,0,0}}), false)))

+((d=8)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,0},{1,0,0}}), false)))

+((d=4)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,0},{0,1,0}}), false)))


Program Code


```plaintext
+((d=2) *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular( {{0,0,0},{0,0,0},{0,0,1}})), false)

theView.AddTheme(av.run("MakeGtheme",{m,"m"}))

'estimate flow distance within each grid cell for use in slope length factor
'three distances types based on flow direction

'first, distance equals to half cell width plus half cell diagonal

d1m2 = (d = 1) and (m = 2)
h = d1m2.IsInteger

d1m8 = (d = 1) and (m = 8)
h = d1m8.IsInteger

d1m32 = (d = 1) and (m = 32)
h = d1m32.IsInteger

d1m128 = (d = 1) and (m = 128)
h = d1m128.IsInteger

d2m1 = (d = 2) and (m = 1)
h = d2m1.IsInteger

d2m4 = (d = 2) and (m = 4)
h = d2m4.IsInteger

d2m16 = (d = 2) and (m = 16)
h = d2m16.IsInteger

d2m64 = (d = 2) and (m = 64)
h = d2m64.IsInteger

d4m2 = (d = 4) and (m = 2)
```

*Program Code*
h = d4m2.IsInteger

d4m8 = (d = 4) and (m = 8)

h = d4m8.IsInteger


d4m32 = (d = 4) and (m = 32)

h = d4m32.IsInteger


d4m128 = (d = 4) and (m = 128)

h = d4m128.IsInteger


d8m1 = (d = 8) and (m = 1)

h = d8m1.IsInteger


d8m4 = (d = 8) and (m = 4)

h = d8m4.IsInteger


d8m16 = (d = 8) and (m = 16)

h = d8m16.IsInteger


d8m64 = (d = 8) and (m = 64)

h = d8m64.IsInteger


d16m2 = (d = 16) and (m = 2)

h = d16m2.IsInteger


d16m8 = (d = 16) and (m = 8)

h = d16m8.IsInteger


d16m32 = (d = 16) and (m = 32)

h = d16m32.IsInteger


d16m128 = (d = 16) and (m = 128)

h = d16m128.IsInteger
d32m1 = (d = 32) and (m = 1)
h = d32m1.IsInteger

d32m4 = (d = 32) and (m = 4)
h = d32m4.IsInteger

d32m16 = (d = 32) and (m = 16)
h = d32m16.IsInteger

d32m64 = (d = 32) and (m = 64)
h = d32m64.IsInteger

d64m2 = (d = 64) and (m = 2)
h = d64m2.IsInteger

d64m8 = (d = 64) and (m = 8)
h = d64m8.IsInteger

d64m32 = (d = 64) and (m = 32)
h = d64m32.IsInteger

d64m128 = (d = 64) and (m = 128)
h = d64m128.IsInteger

d128m1 = (d = 128) and (m = 1)
h = d128m1.IsInteger

d128m4 = (d = 128) and (m = 4)
h = d128m4.IsInteger

d128m16 = (d = 128) and (m = 16)
h = d128m16.IsInteger

d128m64 = (d = 128) and (m = 64)
h = d128m64.IsInteger

distance1pt207 = (d1m2 + d1m8 + d1m32 + d1m128 + d2m1 + d2m4 + d2m16 + d2m64
+ d4m2 + d4m8 + d4m32 + d4m128
+ d8m1 + d8m4 + d8m16 + d8m64 + d16m2 + d16m8 + d16m32 + d16m128 + d32m1
+ d32m4 + d32m16 + d32m64
+ d64m2 + d64m8 + d64m32 + d64m128 + d128m1 + d128m4 + d128m16 +
d128m64)
* ((0.5) + ((0.5) * (2.sqrt)))

theView.AddTheme(av.run("MakeGtheme","{distance1pt207,"distance1pt207"}"))) 'second distance equals to cell width

m0 = (m = 0)

h = m0.IsInteger

d1m1 = (d = 1) and (m = 1)

h = d1m1.IsInteger

d1m4 = (d = 1) and (m = 4)

h = d1m4.IsInteger

d1m16 = (d = 1) and (m = 16)

h = d1m16.IsInteger

d1m64 = (d = 1) and (m = 64)

h = d1m64.IsInteger

d4m1 = (d = 4) and (m = 1)

h = d4m1.IsInteger

d4m4 = (d = 4) and (m = 4)
h = d4m4.IsInteger

d4m16 = (d = 4) and (m = 16)

h = d4m16.IsInteger

d4m64 = (d = 4) and (m = 64)

h = d4m64.IsInteger

d16m1 = (d = 16) and (m = 1)

h = d16m1.IsInteger

d16m4 = (d = 16) and (m = 4)

h = d16m4.IsInteger

d16m16 = (d = 16) and (m = 16)

h = d16m16.IsInteger

d16m64 = (d = 16) and (m = 64)

h = d16m64.IsInteger

d64m1 = (d = 64) and (m = 1)

h = d64m1.IsInteger

d64m4 = (d = 64) and (m = 4)

h = d64m4.IsInteger

d64m16 = (d = 64) and (m = 16)

h = d64m16.IsInteger

d64m64 = (d = 64) and (m = 64)

h = d64m64.IsInteger

distance1pt000 = (d1m1 + d1m4 + d1m16 + d1m64 + d4m1 + d4m4 + d4m16 +
d4m64 + d16m1 + d16m4 + d16m16 + d16m64 + d64m1 + d64m4 + d64m16 + d64m64)
+ m0)
theView.AddTheme(av.run("MakeGtheme", {distance1pt000, "distance1pt000"}))

'second distance equals to cell diagonal

d2m2 = (d = 2) and (m = 2)
h = d2m2.IsInteger

d2m8 = (d = 2) and (m = 8)
h = d2m8.IsInteger

d2m32 = (d = 2) and (m = 32)
h = d2m32.IsInteger

d2m128 = (d = 2) and (m = 128)
h = d2m128.IsInteger

d8m2 = (d = 8) and (m = 2)
h = d8m2.IsInteger

d8m8 = (d = 8) and (m = 8)
h = d8m8.IsInteger

d8m32 = (d = 8) and (m = 32)
h = d8m32.IsInteger

d8m128 = (d = 8) and (m = 128)
h = d8m128.IsInteger

d32m2 = (d = 32) and (m = 2)
h = d32m2.IsInteger

d32m8 = (d = 32) and (m = 8)
h = d32m8.IsInteger
d32m32 = (d = 32) and (m = 32)

h = d32m32.IsInteger

h = d32m128.IsInteger

d128m2 = (d = 128) and (m = 2)

h = d128m2.IsInteger

d128m8 = (d = 128) and (m = 8)

h = d128m8.IsInteger

d128m32 = (d = 128) and (m = 32)

h = d128m32.IsInteger

d128m128 = (d = 128) and (m = 128)

h = d128m128.IsInteger

distance1pt414 = (d2m2 + d2m8 + d2m32 + d2m128 + d8m2 + d8m8 + d8m32 +
    d8m128
+ d32m2 + d32m8 + d32m32 + d32m128 + d128m2 + d128m8 + d128m32 + d128m128
) * 1.414

theView.AddTheme(av.run("MakeGtheme",{distance1pt414,"distance1pt414"}))

distance = (distance1pt207 + distance1pt414 + distance1pt000)

theView.AddTheme(av.run("MakeGtheme",{distance,"distance"}))

'estimate horizontal distance

horizontalDistance = distance / (SlopeRadian.Cos)

theView.AddTheme(av.run("MakeGtheme",{horizontalDistance,"horizontalDistance"}))

'estimate inverse distance
InverseDistance = 1.asGrid / horizontalDistance

theView.AddTheme(av.run("MakeGtheme", [InverseDistance,"InverseDistance"]))

' estimate flow length to downstream end of each grid cell

Lamdai = d.flowlength(InverseDistance, true) + (0.5.asgrid * cellSize.asgrid * boundary)

' need to cap the slope length at 100m

Lamdai = (Lamdai >= 100.asgrid).Con(100.asgrid,Lamdai)

theView.AddTheme(av.run("MakeGtheme", [Lamdai,"Lamdai"]))

LamdaiMinus1 = d.flowlength(InverseDistance, true)

LamdaiMinus1 = (LamdaiMinus1 >= 100.asgrid).Con((100.asgrid - cellSize.asgrid),LamdaiMinus1)

LamdaiMinus1 = LamdaiMinus1.isNull.Con(0.asGrid,LamdaiMinus1)

theView.AddTheme(av.run("MakeGtheme", [LamdaiMinus1,"LamdaiMinus1"]))

' estimate slope RUSLE slope length factor

' cell downstream - flowlength of current which translates into distance to upper edge of the segment - distance to lower edge

slopelengthL = ((((Lamdai).Pow(mCoef+1) - (LamdaiMinus1).Pow(mCoef+1)) / ((22.1.asgrid ^ mCoef) * (Lamdai - LamdaiMinus1))))

theView.AddTheme(av.run("MakeGtheme", [slopelengthL,"slopelengthL"]))

' estimate slope steepness factor

'slope steepness for cells with slope less than 9

slopelessthn9 = (SlopeDegree < ((5.1428).AsGrid))

slopelessthn9 = slopelessthn9.isNull.Con(0.asGrid,slopelessthn9)

'slope steepness for cells with slope less than 9
slopmorethn9=(SlopeDegree >= ((5.1428).AsGrid))
slopmorethn9 = slopmorethn9.isNull.Con(0.asGrid,slopmorethn9)
SlopeSteepnessLess9 = (10.8.asgrid * SlopeRadian.sin + 0.03.asgrid) * slopelessthn9
SlopeSteepnessMore9 = (16.8.asgrid * SlopeRadian.sin - 0.5.asgrid) * slopmorethn9
SlopeSteepnessS = SlopeSteepnessLess9 + SlopeSteepnessMore9
'commented out on 2-23-06
theView.AddTheme(av.run("MakeGtheme",{SlopeSteepnessS,"SlopeSteepnessS"}))
estimate combined topographic factor
LS = slopelengthL * SlopeSteepnessS
theView.AddTheme(av.run("MakeGtheme",{LS,"LS"}))
'LS according to a similar formulae but from desmet and govers comments on mitasova's paper (page 607)
'Ldesmet = (((upslopeArea_m2 + (cellSize^2)).pow(mCoef+1)) -
upslopeArea_m2.pow(mCoef+1)) / ((cellSize.asgrid ^ (mCoef+2)) * (Aspect.cos +
Aspect.sin).pow(mCoef) * (22.13.asgrid ^ mCoef))
'having the aspect in the equation results in some negative values in the denominator and therefore, underined cells
x1 = (d = 1) * boundary
h = x1.IsInteger
x2 = (d = 2) * boundary
h = x2.IsInteger
x4 = (d = 4) * boundary
h = x4.IsInteger
x8 = (d = 8) * boundary
h = x8.IsInteger

x16 = (d = 16) * boundary
h = x16.IsInteger

x32 = (d = 32) * boundary
h = x32.IsInteger

x64 = (d = 64) * boundary
h = x64.IsInteger

x128 = (d = 128) * boundary
h = x128.IsInteger

x256 = (d = 256) * boundary
h = x256.IsInteger

x1pt0 = (x1 + x4 + x16 + x64) * 1.asgrid
x1pt41 = (x2 + x8 + x32 + x128) * 2.sqrt.asgrid
x = (x1pt0 + x1pt41) * cellsize.asgrid

theView.addTheme(av.run("makeGtheme",{x,"x"}))

Ldesmet = (((upslopeArea_m2 + (cellSize^2)).pow(mCoef+1)) -
upslopeArea_m2.pow(mCoef+1)) / ((cellSize.asgrid ^ (mCoef+2)) * (22.13.asgrid ^
mCoef) * x.pow(mCoef))

theView.AddTheme(av.run("MakeGtheme",{Ldesmet,"Ldesmet"}))

LShybrid = SlopeSteepnessS * Ldesmet

theView.AddTheme(av.run("MakeGtheme",{LShybrid,"LShybrid"}))

'get RUSLE R factor
R = theView.FindTheme("rfactor10m").GetGrid

'convert to SI if input is US

RSI = R * 17.02.asgrid

'get RUSLE K factor

K = theView.FindTheme("kfactor10m").GetGrid

'convert to SI if input is US

KSI = K * 0.1317.asgrid

'get RUSLE C factor

C = theView.FindTheme("cfactor10m").GetGrid

'get RUSLE P factor

P = theView.FindTheme("pfactor10m").GetGrid

' Estimate RUSLE 2D total soil erosion rate [t/ha]

RUSLE2D = LShybrid * P * KSI * RSI * C

theView.AddTheme(av.run("MakeGtheme", [RUSLE2D,"RUSLE2D"]))

'estimate erosion amount [t]

erosionAmountRUSLE2D = cellArea_ha * RUSLE2D

theView.AddTheme(av.run("MakeGtheme", [erosionAmountRUSLE2D,"erosionAmountRUSLE2D"]))

' Estimate RUSLE total soil erosion rate [t/ha]

RUSLE = LS * RSI * KSI * C * P

theView.AddTheme(av.run("MakeGtheme", [RUSLE,"RUSLE"]))

'estimate erosion amount [t]
erosionAmountRUSLE = cellArea_ha * RUSLE

theView.AddTheme(av.run("MakeGtheme", {erosionAmountRUSLE, "erosionAmountRUSLE"}))

'estimate transport capacity using AMEENA method with a = 1 and WaTEM/SEDEM
with Ktc = 1 (all three will be calibrated later)

'but first estimating the flow

'get 10yr24hr precipitation

Precipitation_10yr_24hr = theView.FindTheme("10yr24hr10m").getGrid

'get curve number

CN = theView.FindTheme("cn10m").getGrid

'get Manning's roughness coefficient

Mannings = theView.FindTheme("manning10m").getGrid

'estimate storage

S_mm = (25400.AsGrid / CN) - 254.AsGrid

theView.AddTheme(av.run("MakeGtheme", {S_mm, "S_mm"}))

'estimate runoff depth

Q_m = ((Precipitation_10yr_24hr - (0.2.AsGrid * S_mm)).Pow(2) / (Precipitation_10yr_24hr + (0.8.AsGrid * S_mm))) / 1000.AsGrid

theView.AddTheme(av.run("MakeGtheme", {Q_m, "Q_m"}))

'ridge cells

ridgeCells = ((d.flowaccumulation(nil)) = 0.asgrid)

Cumulative_Q_m = d.flowaccumulation(Q_m) + (ridgeCells * Q_m)
Program Code

```
theView.AddTheme(av.run("MakeGtheme", {Cumulative_Q_m, "Cumulative_Q_m"}));

'estimate runoff volume
Cumulative_Q_m_cube = Cumulative_Q_m * cellArea_m2
theView.AddTheme(av.run("MakeGtheme", {Cumulative_Q_m_cube, "Cumulative_Q_m_cube"}));

'estimate runoff volume per
Cumulative_q_m_square = Cumulative_Q_m_cube / (x)
theView.AddTheme(av.run("MakeGtheme", {Cumulative_q_m_square, "Cumulative_q_m_square"}));

'estimate Alsmadi's transport capacity
'the power for flow and slope were set to 1.4 which is the median of the transport
capacity equations
TransportCapacityAlsmadi = (1.AsGrid / (Mannings.Pow(1.5))) * Cumulative_q_m_square.pow(1.4) * SlopePercent.pow(1.4)
theView.AddTheme(av.run("MakeGtheme", {TransportCapacityAlsmadi, "TransportCapacityAlsmadi"}));

'transport capacity Van Rompaey Tc = Ktc R K (LS - aSir) where Ktc is transport
capacity coefficient obtained by calibration
'the division by 10000 is a conversion factor
'based on RUSLE2D LS factor
'set Ktc to a constant
Ktc = boundary * 1.asgrid
```
TransportCapacityVanRompaeyRUSLE2D = (Ktc * RSI * KSI * (LShybrid - (0.6.asgrid
* 6.86.asgrid * (SlopePercent).pow(0.8))))/ 10000.asgrid
theView.AddTheme(av.run("MakeGtheme",{TransportCapacityVanRompaeyRUSLE2D,
"TransportCapacityVanRompaeyRUSLE2D"}))

'********** THIS PART CALCULATES THE ROUTING LEVEL AND
DOWNSTREAM ROUTING LEVEL IN INTEGERS

DoubleInverseDistance = InverseDistance * 2.asgrid
RoutingLevel = d.flowlength(DoubleInverseDistance, true) + (0.5.asgrid * cellSize.asgrid
* boundary)
theView.AddTheme(av.run("MakeGtheme",{Lamdai,"Lamdai"}))
'estimate flow length to the the buttom of each cell
RoutingLevelMinus1 = d.flowlength(DoubleInverseDistance, true) + (cellSize.asgrid * boundary)
RoutingLevelMinus1 = RoutingLevelMinus1.isNull.Con(0.asgrid,RoutingLevelMinus1)
theView.AddTheme(av.run("MakeGtheme",{LamdaiMinus1,"LamdaiMinus1"}))
'convert routing level to integer
FlowLengthInteger = ((RoutingLevel - RoutingLevel.Floor) > (RoutingLevel
L = FlowLengthInteger.isNull.Con(0.asgrid,FlowLengthInteger)
L = L.int
theView.AddTheme(av.run("MakeGtheme",{L,"L"}))
'estimate routing level of downstream cell
DFL = ((d=32)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{1,0,0},{0,0,0},{0,0,0}}), false)))+((d=64)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,1,0},{0,0,0},{0,0,0}}), false)))+((d=128)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,0,1},{0,0,0},{0,0,0}}), false)))+((d=16)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,0,0},{1,0,0},{0,0,0}}), false)))+((d=1)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,0,0},{0,0,1},{0,0,0}}), false)))+((d=8)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,0,0},{0,0,0},{1,0,0}}), false)))+((d=4)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,0,0},{0,0,0},{0,1,0}}), false)))+((d=2)*(L.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{0,0,0},{0,0,0},{0,0,1}}), false)));
theView.AddTheme(av.run("MakeGtheme",{DFL,"DFL"}))
DFL = DFL.int
theView.AddTheme(av.run("MakeGtheme",{DFL,"DFL"}))
'identify cells that flow into channel cells
m = ((d=32) *(d.FocalStats(#GRID_STATYPE_SUM, NbrHood.MakeIrregular({{1,0,0},{0,0,0},{0,0,0}}), false)));
Program Code

```
+((d=64) *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,1,0},{0,0,0},{0,0,0}}), false)))
+((d=128)*(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,1},{0,0,0},{0,0,0}}), false)))
+((d=16) *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{1,0,0},{0,0,0}}), false)))
+((d=1)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,1},{0,0,0}}), false)))
+((d=8)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,0},{1,0,0}}), false)))
+((d=4)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,0},{0,1,0}}), false)))
+((d=2)  *(d.FocalStats(#GRID_STATYPE_SUM,
NbrHood.MakeIrregular({{0,0,0},{0,0,0},{0,0,1}}), false)))

cellsFlowingIntoStreams = (m = 999).con(1.asgrid,0.asgrid)
'theView.AddTheme(av.run("MakeGtheme",{cellsFlowingIntoStreams,"cellsFlowingInto Streams"}))
```
C.2. Routing Algorithm

The routing algorithm assumes that the following grids have already been generated and are available in the View:

1. Watershed boundary
2. Flow direction
3. Gross erosion
4. Routing level
5. Downstream routing level

```plaintext
theView = av.GetActiveDoc
thePrj = theView.GetProjection

'R get erosion amount grid
RUSLE = theView.FindTheme("erosionAmountRUSLE").GetGrid

'get transport capacity grid
TraCapa = (theView.FindTheme("TransportCapacityAlsmadi").GetGrid)

'get watershed boundary grid
boundary = theView.FindTheme("80minclusive2").GetGrid

'get flow direction grid
FlowDirection = theView.FindTheme("d").GetGrid

'create a duplicate of flow direction for later use inside the loop
FlowDirectionTotal = theView.FindTheme("d").GetGrid

'get the routing level of downstream cells
routingLevelDownstreams = theView.FindTheme("DFL").GetGrid

get the routing level
routingLevelLocal = theView.FindTheme("L").GetGrid
```

Program Code
'this line is used to change the a factor or the Ktc factor for purpose of calibration

TraCapa = TraCapa * 1.3

'obtain the max and min of routing levels for use in routing loop

routingLevelLocalStats = routingLevelLocal.GetStatistics
routingLevelLocalMax   = routingLevelLocalStats.Get(1)
routingLevelLocalMin   = routingLevelLocalStats.Get(0)

MsgBox.Info( routingLevelLocalMax.AsString, "")
MsgBox.Info( routingLevelLocalMin.AsString, "")

'obtain the VTable containing all unique values within the routing level grid

theVTab = routingLevelLocal.GetVTab

routingLevel = theVTab.FindField( "Value" )

' copy each row in the VTab to list for use in the loop

routingLevelsList = list.make

iMax = 0

for each i in theVTab

'create the next row and add values to the list

    routingLevelsList.add (theVTab.ReturnValue( routingLevel , i))

        iMax = iMax + 1

end

'initialize delivery and deposition

Delivery = 0.asgrid

Deposition = 0.asgrid

'find erosion in ridge cells
E = RUSLE * (routingLevelLocal = routingLevelLocalMin)

'find transport capacity in ridge cells

Tc = TraCapa * (routingLevelLocal = routingLevelLocalMin)

'estimate delivery from and deposition within ridge cells

DelPar = (E > Tc).Con(Tc, E)

DepPar = (E > Tc).Con(E - Tc, 0.asgrid)

'start the routing loop

for each i in (routingLevelLocalMin + 1)..(iMax - 1)

'get flow direction within the two levels of routing

FlowDirPar = ((routingLevelLocal = routingLevelsList.get(i)) +
(routingLevelDownstreams =
routingLevelsList.get(i))).Con(FlowDirection, 1.asgrid/0.asgrid)

'get get gross erosion within the two levels of routing

EPartial = ((routingLevelLocal = routingLevelsList.get(i)) * RUSLE) +
FlowDirPar.flowaccumulation(Delivery * (routingLevelDownstreams =
routingLevelsList.get(i)))

EPartial = EPartial.isNull.Con(0.asgrid, EPartial)

DepPar = Nil

'get get transport capacity within the two levels of routing

TcPartial = (routingLevelLocal = routingLevelsList.get(i)) * TraCapa

DelPar = Nil

'remove temporary grids
av.purgeobjects

'estimate delivery and deposition within current routing level and add to delivery and deposition grids

\[
\text{DelPar} = (\text{EPartial} > \text{TcPartial}).\text{Con}(\text{TcPartial}, \text{EPartial}) \ast (\text{routingLevelLocal} = \text{routingLevelsList.get(i)})
\]

\[
\text{Delivery} = \text{DelPar} + \text{Delivery}
\]

\[
\text{Deposition} = (((\text{EPartial} > \text{TcPartial}).\text{Con}(\text{EPartial} - \text{TcPartial}, 0.\text{asgrid}))) + \text{Deposition}
\]

'Deposition

'remove the temporary grids

\[
\text{FlowDirPar} = \text{Nil}
\]

\[
\text{TcPartial} = \text{Nil}
\]

\[
\text{EPartial} = \text{Nil}
\]

av.purgeobjects

'reforce estimation of grid

\[
\text{b} = \text{Delivery}.\text{IsInteger}
\]

\[
\text{b} = \text{Deposition}.\text{IsInteger}
\]

end

'remove the temporary grid so the directory does not jam. This one is outside the loop to delete the last partial delivery calculation

\[
\text{DelPar} = \text{Nil}
\]

av.purgeobjects

'add deposition and delivery grids to View
theView.AddTheme(av.run("MakeGtheme", {deposition,"Deposition RUSLE TransportCapacityAlsmadi \* 1.3"})))

theView.AddTheme(av.run("MakeGtheme", {delivery,"Delivery RUSLE TransportCapacityAlsmadi \* 1.3"})))

'this script saves the project

theProject = av.GetProject

theFileName = theProject.GetFileName

if (theFileName = nil) then
  av.Run("Project.SaveAs", nil)
else
  if (av.Run("Project.CheckForEdits", nil).Not) then
    return nil
  end
  if (theProject.Save) then
    av.ShowMsg("Project saved to " + theFileName.GetBaseName + ","")
    if (System.GetOS = #SYSTEM_OS_MAC) then
      Script.Make("MacClass.SetDocInfo(SELF, Project)").DoIt(theFileName)
    end
  end
end

finished = "Run Completed!"

MsgBox.Info( finished.AsString, "")