Appendix B – PEGGER Avenue™ Code

The following ArcView 3 Avenue code comprises only the parts of PEGGER that deal with route projection, surveying, and other critical aspects of the program. Much of the code behind PEGGER is not directly related to the pegging task and is not included here.

```
' Pegger.PeggingTool.Apply
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98115
'
'              luke@nwgeospatial.com
'
'              October 17th, 2000
'
' Description: respond to user click and initiate pegging process
'
' Calls:       Pegger.PeggingTool.StartRoad(theView,theTheme,thePoint)
'              Pegger.PeggingTool.Peg(thePoint,theLastPoint)
'
' Returns:     nothing
'
'*********************************************************************
theView = av.GetActiveDoc
theTheme = theView.GetEditableTheme
av.ClearMsg
theDict = theTheme.GetObjectTag
theLastPoint = theDict.Get("LastPoint")

If ((System.IsShiftKeyDown) or (theLastPoint = nil)) then
   shiftDown = TRUE
else
   shiftDown = FALSE
end

' Get the point off the display when user clicks
thePoint = theView.GetDisplay.ReturnUserPoint

' If the point they selected is outside of the extent of
' the contour theme then return a warning
theDict = theTheme.GetObjectTag
theContTheme = theDict.Get("ContTheme")
theExtentRect = theContTheme.ReturnExtent
if (theExtentRect.Contains(thePoint).Not) then
   MsgBox.Warning("You cannot peg in a road where no contours exist",
    "PeggingTool.Apply")
   return nil
end
'*********************************************************************
' If the shift key is down then it is a new road
if (shiftDown = TRUE) then
```

```
   av.ShowMsg("Starting new road at:"++thePoint.AsString)
   theStartRoadStatus = av.Run("Pegger.PeggingTool.StartRoad",
                              {theView, theTheme, thePoint})
   return nil
else
   ' Figure out which point to go to
   theGoToPoint = av.Run("Pegger.PeggingTool.Peg",
                        {thePoint, theLastPoint})
end
'*********************************************************************
' If there is no goto point then exit and error
if (theGoToPoint = Nil) then
   return nil
else
   theDict.Set("LastPoint", theGoToPoint)
   theTheme.SetObjectTag(theDict)
end
'*********************************************************************
' Make the line
theGradeLine = PolyLine.Make({{theLastPoint, theGoToPoint}})

' Add the line to the table
if (theTheme <> nil) then
  theFTab = theTheme.GetFTab
  theShapeField = theFTab.FindField("Shape")
  theFTab.BeginTransaction
  rec = theFTab.AddRecord
  theFTab.SetValue(theShapeField, rec, theGradeLine)
  ' Add the grade if setup to do so
  attributeGrades = theDict.Get("AttributeGrades")
  if (attributeGrades) then
    theGrade = theDict.Get("Grade")
    theGradeField = theDict.Get("GradeField")
    theFTab.SetValue(theGradeField, rec, theGrade)
  end
  ' Add the name if setup to do so
  attributeName = theDict.Get("AttributeName")
  if (attributeName) then
    theName = theDict.Get("Name")
    theNameField = theDict.Get("NameField")
    theFTab.SetValue(theNameField, rec, theName)
  end
  theTheme.GetFTab.EndTransaction
  theTheme.GetFTab.GetSelection.ClearAll
  theTheme.GetFTab.GetSelection.Set(rec)
  theTheme.GetFTab.UpdateSelection
end

av.GetProject.SetModified(true)
return nil
```

```
' Pegger.PeggingTool.StartRoad
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98195
'
'              lwrogers@u.washington.edu
'
'              October 17th, 2000
'
' Description: Start a new pegging segment
'
' Calls:       nothing
'
' Returns:     set object tag for the point that was just
'              clicked so that a new road can be started
'
'**********************************************************************
'MsgBox.Info("Starting new road", "PeggingTool.StartRoad")
theView = SELF.Get(0)
theTheme = SELF.Get(1)
thePoint = SELF.Get(2)
theDict = theTheme.GetObjectTag
theContTheme = theDict.Get("ContTheme")
theElevItem = theDict.Get("ElevItem")
theGrade = theDict.Get("Grade")
theInterval = theDict.Get("ContInterval")
theCurrentElev = theDict.Get("CurrentElev")
theVTab = theContTheme.GetFTab
theShapeField = theVTab.FindField("Shape")
theElevField = theVTab.FindField(theElevItem.AsString)
'**********************************************************************
' If we are on a contour then get the elevation from the contour,
' otherwise we will have to use the method below of making polygons

' Clear the selection
theContTheme.ClearSelection
theVTab.UpdateSelection

' Try to select a contour by using the point the user just clicked
if (theContTheme.CanSelect) then
   theContTheme.SelectByPoint(thePoint, #VTAB_SELTYPE_NEW)
end

' See if there is a selection (should only be one if any)
if (theVTab.GetSelection.Count = 1) then
   theSelection = theVTab.GetSelection.GetNextSet(-1)
   theCurrentElevation = theVTab.ReturnValue(theElevField,theSelection)

   ' Clear the selection
   theContTheme.ClearSelection
   theVTab.UpdateSelection
   ' Lets see if we can snap to the contour for extra precision
   if (thePoint.Snap(theVTab.ReturnValue(theVTab.FindField("Shape"),
      theSelection), 10)) then
      av.ShowMsg("Snapped to the Contour")
   else
      av.ShowMsg("Failed to snap to Contour, location approximate")
   end

   ' Update the CurrentElev item in the dictionary and set the object
   ' tag on the theme
   theDict.Set("CurrentElev", theCurrentElevation)
   theDict.Set("LastPoint", thePoint)
   theTheme.SetObjectTag(theDict)
   av.GetProject.SetModified(true)

   ' We found the elevation so we are done
   'MsgBox.Info("Found elevation using point"+NL+
   '"Elev:"++theCurrentElev.AsString, "PeggingTool.StartRoad")
   return "POINT"
end

if (theVTab.GetSelection.Count = 0) then
   MsgBox.Info("Must be on a contour before you can peg in a grade.",
   "PeggingTool.StartRoad")
   return "ZERO"
elseif (theVTab.GetSelection.Count > 1) then
   MsgBox.Info("More than one contour selected."+NL+
   "Zoom in to eliminate ambiguous selection", "PeggingTool.StartRoad")
   return "ZERO"
end

MsgBox.Error("Start Road Instance Not Handled",
"PeggingTool.StartRoad")
return nil
```

```
' Pegger.PeggingTool.Peg
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98195
'
'              lwrogers@u.washington.edu
'
'              October 17th, 2000
'
' Description: Given the coordinates of the users click and
'              the last point this module will return a new
'              point on grade in the direction of the click
'
' Calls:       nothing
'
' Returns:     theGoToPoint (new point on grade)
'
'*********************************************************************
thePoint = SELF.Get(0)
theLastPoint = SELF.Get(1)
theView = av.GetActiveDoc
theTheme = theView.GetEditableTheme
theDict = theTheme.GetObjectTag
theContTheme = theDict.Get("ContTheme")
theElevItem = theDict.Get("ElevItem")
theGrade = theDict.Get("Grade")
theInterval = theDict.Get("ContInterval")
theCurrentElev = theDict.Get("CurrentElev")
theContFTab = theContTheme.GetFTab
theShapeField = theContFTab.FindField("Shape")
theElevField = theContFTab.FindField(theElevItem.AsString)

'*********************************************************************

' Make the grade circle
' if the grade is 0 then use a 10% grade circle for segment length
if (theGrade = 0) then
   theRadius = theInterval / (10 / 100)
else
   theRadius = theInterval / (theGrade / 100)
end

' If the radius is less than the snap tolerance then no segment
' will be made so send a warning
if ((theTheme.IsSnapping) or (theTheme.IsInteractiveSnapping)) then
   if ((theTheme.GetSnapTolerance > theRadius.abs) or
   (theTheme.GetInteractiveSnapTolerance > theRadius.abs)) then
      MsgBox.Warning("At least one Snap tolerance is greater than "+
                     " the current"+NL+"segment, a line may not be "+
                     "added to the theme until its"+NL+"length is "+
                     "greater than the current snap tolerance."+NL+
                     "The current segment length is:"++
                     theRadius.AsString, "PeggingTool.Peg")
   end
end

theCircle = Circle.Make(theLastPoint, theRadius.Abs)

if (theCircle.IsNull) then
  return nil
end

' Clear the current contour selection
theContTheme.GetFTab.GetSelection.ClearAll

' Get the contour themes vtab bitmap
theBitmap = theContFTab.GetSelection
'**********************************************************************
' Select the contour to intersect the grade circle with
' first select using the grade circle
theContFTab.SelectByShapes ( {theCircle}, #VTAB_SELTYPE_NEW )

if (theContFTab.GetSelection.Count = 0) then
   MsgBox.Warning("A"++theGrade.AsString+"% grade is too steep."+NL+
   "Lessen the grade to continue pegging", "PeggingTool.Peg")
   return nil
end

' Now select using elevation attribute
if (theCurrentElev <> nil) then
   ' build the query string for the contours on either side
   if (theGrade > 0) then
      theNextContour = theCurrentElev + theInterval
   elseif (theGrade < 0) then
      theNextContour = theCurrentElev - theInterval
   elseif (theGrade = 0) then
      theNextContour = theCurrentElev
   end
   theQuery = "(["+theElevItem.AsString+"] ="++
               theNextContour.AsString++")"

   theBitmap = theContFTab.GetSelection
   theContFTab.Query(theQuery, theBitmap, #VTAB_SELTYPE_AND)
   theContFTab.UpdateSelection
   if (theContFTab.GetSelection.Count = 0) then
      MsgBox.Warning("A"++theGrade.AsString+"% grade may be too "+
                     "steep."+NL+"Lessen the grade to continue "+
                     "pegging", "PeggingTool.Peg")
      return nil
   end
else
   MsgBox.Error("The current elevation is null", "PeggingTool.Peg")
   return nil
end
'**********************************************************************
' Select the proper point to go to
' Intersect the circle with the contour
' Make a list to hold the points
thePointList = List.Make
```

```
for each theSelection in theContFTab.GetSelection
   theContShape = theContFTab.ReturnValue(theShapeField, theSelection)
   theMultiPoint = theContShape.PointIntersection(theCircle)
   thePointList.Merge(theMultiPoint.AsList)
end

' Get the one closest to where we want to go
theDistance = 9999999
for each pnt in thePointList
   if (thePoint.Distance(pnt) < theDistance) then
      theDistance = thePoint.Distance(pnt)
      theGoToPoint = pnt
   end
end
'*********************************************************************
' Set the elevation for the next contour
theDict.Set("CurrentElev", theNextContour)
theTheme.SetObjectTag(theDict)

theView.GetDisplay.Flush
theView.GetGraphics.Empty

av.GetProject.SetModified(true)

return theGoToPoint
```

```
' Pegger.RoadMenu.Dissolve
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98195-2100
'
'              lwrogers@u.washington.edu
'
'              December 11th, 2004
'
' Description: Dissolve roads based on grade or any other item.
'
' Calls:       nothing
'
' Returns:     merges adjacent features together based on a common
'              attribute
'
'**********************************************************************
' Table summary to merge shapes
'Get the theme
theView = av.GetActiveDoc
theTheme = theView.GetActiveThemes.Get(0)
if (theTheme = nil) then
   MsgBox.Warning("Can only dissolve grades on the first active theme"
                  , "RoadMenu.Dissolve")
   return nil
end
if (theTheme.GetFTab.GetShapeClass.GetClassName <> "PolyLine") then
   MsgBox.Warning("Can only dissolve grades on PolyLine features.",
                  "RoadMenu.Dissolve")
   Return nil
end

theFTab = theTheme.GetFTab
'**********************************************************************
'Get the fields
theFieldList = theTheme.GetFTab.GetFields
theValidFields = List.Make
for each theField in theFieldList
  if ((theField.IsTypeNumber) or (theField.IsTypeString)) then
    theValidFields.Add(theField)
  end
end
'**********************************************************************
'Get the dissolve field
theDissolveField = MsgBox.List(theValidFields, "Select the field to "+
                   "dissolve on:", "Select Dissolve Attribute")

if (theDissolveField = nil) then
   return nil
end
'**********************************************************************
'Get the summary fields
```

```
theOtherFields = theValidFields.DeepClone
theDissolveFieldIndex = theValidFields.FindByValue(theDissolveField)
theOtherFields.Remove(theDissolveFieldIndex)

theSummaryList = List.Make
NumTypes = {"by Average","by Sum","by Minimum Value",
            "by Maximum Value",
            "by Standard Deviation","by Variance"}
StrTypes = {"by First","by Last", "by Count"}
for each theField in theOtherFields
  if (theField.IsTypeNumber) then
    for each type in NumTypes
      theSummaryList.Add((theField.AsString++type))
    end
  elseif (theField.IsTypeString) then
    for each type in StrTypes
      theSummaryList.Add((theField.AsString++type))
    end
  end
end

theListToSummarize = MsgBox.MultiListAsString(theSummaryList,
                     "Select additional summary fields if desired:",
                     "Select Dissolve Summaries")
if (theListToSummarize = nil) then
  return nil
end
'**********************************************************************
'Parse the chosen summary fields
theSummaryFieldList = List.Make
theSummaryTypes = List.Make

for each theKey in theListToSummarize
  theTokens = theKey.AsTokens(" ")
  theField = theFTab.FindField(theTokens.Get(0))
  theSummaryFieldList.Add(theField)
  theValue = theTokens.Get(2)
  if (theValue="Sum") then
    theSummary = #VTAB_SUMMARY_SUM
  elseif (theValue="Average") then
    theSummary = #VTAB_SUMMARY_AVG
  elseif (theValue="Minimum") then
    theSummary = #VTAB_SUMMARY_MIN
  elseif (theValue="Maximum") then
    theSummary = #VTAB_SUMMARY_MAX
  elseif (theValue="Standard") then
    theSummary = #VTAB_SUMMARY_STDEV
  elseif (theValue="Variance") then
    theSummary = #VTAB_SUMMARY_VAR
  elseif (theValue="First") then
    theSummary = #VTAB_SUMMARY_FIRST
  elseif (theValue="Last") then
    theSummary = #VTAB_SUMMARY_LAST
  elseif (theValue="Count") then
    theSummary = #VTAB_SUMMARY_COUNT
  end
  theSummaryTypes.Add(theSummary)
```

```
end
'*********************************************************************
'Add the shape field to the summary
theShapeField = theFTab.FindField("Shape")
theSummaryFieldList.Add(theShapeField)
theSummaryTypes.Add(#VTAB_SUMMARY_AVG)
'*********************************************************************
'Select location to store new shapefile
theNewShapefile = FileDialog.Put("Dissolve".AsFileName, "*.shp",
                                 "Save dissolve shapefile as...")
if (theNewShapefile = nil) then
  return nil
end
'*********************************************************************
'Run the dissolve
dissolveFTab = theFTab.Summarize(theNewShapefile, Shape,
               theDissolveField, theSummaryFieldList, theSummaryTypes)
'*********************************************************************
' Explode multi-part shapes into single-part
theShapeField = dissolveFTab.FindField("Shape")

'Start editing the FTab
dissolveFTab.StartEditingWithRecovery

' Get every selected record after another
For Each record In dissolveFTab
  theLine = dissolveFTab.ReturnValue(theShapeField, record)
  theLine = theLine.ReturnConnected

  ' If the selected feature contains more than one part, explode it
  If (theLine.CountParts > 1) Then
    theLines = theLine.Explode

    ' Take every children...
    For Each aLine In theLines
      newRecord = dissolveFTab.AddRecord
      dissolveFTab.SetValue(theShapeField, newRecord, aLine)

      ' ...and give it the attributes of its ancestor
      For Each aField In dissolveFTab.GetFields
        If (aField.AsString <> "Shape") Then
          dissolveFTab.SetValue(aField, newRecord,
                       dissolveFTab.ReturnValue(aField, record))
        End
      End
    End

    ' Remove the ancestor
    dissolveFTab.RemoveRecord(record)
  End
End

'Stop editing the FTab
dissolveFTab.StopEditingWithRecovery(TRUE)

dissolveFTab.CreateIndex(theShapeField)
theView.AddTheme(FTheme.Make(dissolveFTab))
```

```
    return nil
```

```
' RoadMenu.MergePolyLines
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98115
'
'              luke@nwgeospatial.com
'
'              October 17th, 2000
'
' Description: Merge selected features into single feature
'
' Calls:       nothing
'
' Returns:     nothing
'
'**********************************************************************
theView = av.GetActiveDoc
theTheme = theView.GetEditableTheme
theFTab = theTheme.GetFTab
if (theFTab.GetSelection.Count = 0) then
   MsgBox.Info("No polylines selected.", "RoadMenu.MergePolyLines")
   return nil
end
if(theFTab.GetShapeClass.GetClassName <> "PolyLine") then
  MsgBox.Warning("We can only spline PolyLine features.","")
  return nil
end
'**********************************************************************
' Make the list to hold the points
theList = List.Make

' Beging transaction so we can undo
theFTab.BeginTransaction

For Each selrec in theFTab.GetSelection
  thePolyLine = theFTab.ReturnValue(theFTab.FindField("Shape"),selrec)
  thePoints = thePolyLine.AsList
  for each pnt in thePoints
     theList.Add(pnt)
  end
  theFTab.RemoveRecord(selrec)
end

newpolyline = PolyLine.Make(theList)
newrec = theFTab.AddRecord
theFTab.SetValue(theFTab.FindField("Shape"), newrec, newpolyline)

' End the undo
theFTab.EndTransaction

theFTab.GetSelection.ClearAll
theFTab.GetSelection.Set(newrec)
theFTab.UpdateSelection

theView.Invalidate

return nil
```

```
' Pegger.RoadMenu.Survey
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98115
'
'              lwrogers@u.washington.edu
'
'              March 1st, 2003
'
' Description: Performs a "digital survey" of a selected
'              pegged road or all roads if none are selected.
'              The digital survey creates a "UNIT SURVEY"
'              format .pol file that can be directly imported
'              into the ROADENG road engineering package. This
'              functionality will only be enabled if three
'              conditions exist: Spatial or 3D Analyst must
'              be enabled, a valid surface theme must be
'              in the current view, and only one feature can
'              be selected in the active road theme.
'
' Calls:       Pegger.RoadMenu.Survey.Function.Azimuth
'
' Returns:     None
'
' Dependencies:Spatial Analyst or 3D Analyst
'
'**********************************************************************
theView = av.GetActiveDoc

'Make sure there is only 1 active theme
if (theView.GetActiveThemes.Count <> 1) then
  MsgBox.Warning("There must be only 1 active theme to survey",
                 "RoadMenu.Survey")
  return nil
end

theTheme = theView.GetActiveThemes.Get(0)

'Make sure the theme is a PolyLine
if (theTheme.GetFTab.GetShapeClass.GetClassName <> "PolyLine") then
   MsgBox.Warning("We can only survey PolyLine features.",
                  "RoadMenu.Survey")
   return nil
end

'Make sure there is only 1 selected road
if (theTheme.GetFTab.GetSelection.Count <> 1) then
  if (theTheme.GetFTab.GetSelection.Count > 1) then
    MsgBox.Warning("We can only survey 1 feature at a time",
                   "RoadMenu.Survey")
  elseif (theTheme.GetFTab.GetSelection.Count = 0) then
    MsgBox.Warning("No selected road found to survey",
                   "RoadMenu.Survey")
  end
  return nil
end
'*********************************************************************
' Get the dictionary and the contour theme
theDict = theTheme.GetObjectTag
theSurveyId = theDict.Get("SurveyId")
theSurfaceTheme = theDict.Get("SurfaceTheme")
theSideShotDist = theDict.Get("SideShotDist")
theSideShotNum = theDict.Get("SideShotNum")
densify = theDict.Get("Densify")
theDensity = theDict.Get("DensifyDistance")
drawOnScreen = theDict.Get("DrawOnScreen")
theOutFile = theDict.Get("SurveyFile")
'*********************************************************************
'Set additional parameters from dictionary variables
if (theSurfaceTheme.GetClass.GetClassName = "STheme") then
  theSurface = theSurfaceTheme.GetSurface
  theSurfaceType = "STheme"
else
  theSurface = theSurfaceTheme.GetGrid
  theSurfaceType = "GTheme"
end

theOutFile = theOutFile.AsFileName
theSsDistance = theSideShotDist.AsNumber
theNumSs = theSideShotNum.AsNumber
'*********************************************************************
'Get the selected road
theFTab = theTheme.GetFTab
theShapeField = theFTab.FindField("Shape")
theBitmap = theFTab.GetSelection
theRec = theBitmap.GetNextSet(-1)
thePolyLine = theFTab.ReturnValue(theShapeField, theRec)
'*********************************************************************
'Densify survey points?
if ((densify) and (theDensity.IsNumber)) then
  thePointList = thePolyLine.ReturnDensified(theDensity.AsNumber)
  thePointList = thePointList.AsMultiPoint.AsList
else
  thePointList = thePolyLine.AsMultiPoint.AsList
end
'*********************************************************************
'Make a list to store the survey points
theOutList = List.Make
'*********************************************************************
' Number the side shots
theNumPoints = thePointList.Count
theSsId = theNumPoints
'*********************************************************************
' Initialize the status bar
av.ClearMsg
av.ClearStatus
av.ShowMsg("Surveying " + theSurveyId + "...")
av.SetStatus(0)
'*********************************************************************
```

```
'Loop through the points and get FS, BS and SD information
i = 0
for each pt in thePointList
  ' Update the status bar
  progress = (i/theNumPoints) * 100
  av.SetStatus(progress)
  theOutString = ""
  pt1 = thePointList.Get(i)
  'Draw the survey points on the screen
  if (drawOnScreen) then
    theView.GetGraphics.Add(GraphicShape.Make(pt1))
  end
  x1 = pt1.GetX.SetFormat("d.dd")
  y1 = pt1.GetY.SetFormat("d.dd")
  if (theSurfaceType = "STheme") then
    z1 = theSurface.Elevation(pt1).SetFormat("d.dd")
  else
    z1 = theSurface.CellValue(pt1, Prj.MakeNull).SetFormat("d.dd")
  end
'**********************************************************************
  if (i = 0) then ' Starting point
    ' Get the next point
    pt2 = thePointList.Get(i + 1)
    x2 = pt2.GetX.SetFormat("d.dd")
    y2 = pt2.GetY.SetFormat("d.dd")
    if (theSurfaceType = "STheme") then
      z2 = theSurface.Elevation(pt2).SetFormat("d.dd")
    else
      z2 = theSurface.CellValue(pt2, Prj.MakeNull).SetFormat("d.dd")
    end
    ' Starting Reference
    theOutList.Add(theSurveyId.AsString + ",SR," + x1.AsString + "," +
      y1.AsString + "," + z1.AsString)

    ' Starting sideshots perpendictular to line
    for each n in 1 .. theNumSs
      d = theSsDistance * n
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,x2,y2})
      xSs = x1 + (d * ((theAzimuth + 90).AsRadians.Sin))
      ySs = y1 + (d * ((theAzimuth + 90).AsRadians.Cos))
      ptSs = Point.Make(xSs, ySs)
      if (theSurfaceType = "STheme") then
        zSs = theSurface.Elevation(ptSs).SetFormat("d.dd")
      else
        zSs = theSurface.CellValue(ptSs,
Prj.MakeNull).SetFormat("d.dd")
      end

      'Draw the survey points on the screen
      if (drawOnScreen) then
        theView.GetGraphics.Add(GraphicShape.Make(ptSs))
      end

      'Get the azimuth of the point from the Station
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,xSs,ySs})
      theOutList.Add(theSurveyId.AsString + ",SD," + i.AsString +
                     "," + theSsId.AsString +"," +
                     theAzimuth.AsString + "," + zSs.AsString +
                     "," + d.AsString)

      ' Increment the side shot Id
      theSsId = theSsId + 1

      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,x2,y2})
      xSs = x1 + (d * ((theAzimuth - 90).AsRadians.Sin))
      ySs = y1 + (d * ((theAzimuth - 90).AsRadians.Cos))
      ptSs = Point.Make(xSs, ySs)
      if (theSurfaceType = "STheme") then
        zSs = theSurface.Elevation(ptSs).SetFormat("d.dd")
      else
        zSs = theSurface.CellValue(ptSs,Prj.MakeNull).SetFormat("d.dd")
       end
      'Draw the survey points on the screen?
      if (drawOnScreen) then
        theView.GetGraphics.Add(GraphicShape.Make(ptSs))
      end

      'Get the azimuth of the point from the Station
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,xSs,ySs})

      theOutList.Add(theSurveyId.AsString + ",SD," + i.AsString +
                     "," + theSsId.AsString + "," +
                     theAzimuth.AsString + "," + zSs.AsString +
                     "," + d.AsString)

      ' Increment the side shot Id
      theSsId = theSsId + 1

    end

    'Starting foresight
    theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                        {x1,y1,x2,y2})
    theOutList.Add(theSurveyId.AsString + ",FS," + i.AsString +
                   "," + (i + 1).AsString + "," +
                   theAzimuth.AsString + "," + z1.AsString +
                   "," + pt1.Distance(pt2).AsString)
  '*******************************************************************
  ' Last point so get perpendictular sideshots and no foresight
  elseif (i = (thePointList.Count - 1)) then
    ' Get the last point
    pt0 = thePointList.Get(i - 1)
    x0 = pt0.GetX.SetFormat("d.dd")
    y0 = pt0.GetY.SetFormat("d.dd")
    if (theSurfaceType = "STheme") then
      z0 = theSurface.Elevation(pt0).SetFormat("d.dd")
    else
      z0 = theSurface.CellValue(pt0, Prj.MakeNull).SetFormat("d.dd")
    end
```

```
    ' Backsight
    theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                        {x1,y1,x0,y0})
    theOutList.Add(theSurveyId.AsString + ",BS," + i.AsString +
                   "," + (i - 1).AsString + "," +
                   theAzimuth.AsString + "," + z1.AsString +
                   "," + pt1.Distance(pt0).AsString)

    ' Ending sideshots perpendictular to line
    for each n in 1 .. theNumSs
      d = theSsDistance * n
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,x0,y0})
      xSs = x1 + (d * ((theAzimuth + 90).AsRadians.Sin))
      ySs = y1 + (d * ((theAzimuth + 90).AsRadians.Cos))
      ptSs = Point.Make(xSs, ySs)
      if (theSurfaceType = "STheme") then
        zSs = theSurface.Elevation(ptSs).SetFormat("d.dd")
      else
        zSs = theSurface.CellValue(ptSs,
Prj.MakeNull).SetFormat("d.dd")
      end

      'Draw the survey points on the screen?
      if (drawOnScreen) then
        theView.GetGraphics.Add(GraphicShape.Make(ptSs))
      end

      'Get the azimuth of the point from the Station
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,xSs,ySs})

      theOutList.Add(theSurveyId.AsString + ",SD," + i.AsString +
                     "," + theSsId.AsString + "," +
                     theAzimuth.AsString + "," + zSs.AsString +
                     "," + d.AsString)

      ' Increment the side shot Id
      theSsId = theSsId + 1

      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,x0,y0})
      xSs = x1 + (d * ((theAzimuth - 90).AsRadians.Sin))
      ySs = y1 + (d * ((theAzimuth - 90).AsRadians.Cos))
      ptSs = Point.Make(xSs, ySs)
      if (theSurfaceType = "STheme") then
        zSs = theSurface.Elevation(ptSs).SetFormat("d.dd")
      else
        zSs = theSurface.CellValue(ptSs,
Prj.MakeNull).SetFormat("d.dd")
      end

      'Draw the survey points on the screen?
      if (drawOnScreen) then
        theView.GetGraphics.Add(GraphicShape.Make(ptSs))
      end
      'Get the azimuth of the point from the Station
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,xSs,ySs})

      theOutList.Add(theSurveyId.AsString + ",SD," + i.AsString +
                     "," + theSsId.AsString + "," +
                     theAzimuth.AsString + "," + zSs.AsString +
                     "," + d.AsString)

      ' Increment the side shot Id
      theSsId = theSsId + 1

    end
  '******************************************************************
  else
    ' Intermediate point so get forsight and backsight
    ' Get the last point
    pt0 = thePointList.Get(i - 1)
    x0 = pt0.GetX.SetFormat("d.dd")
    y0 = pt0.GetY.SetFormat("d.dd")
    if (theSurfaceType = "STheme") then
      z0 = theSurface.Elevation(pt0).SetFormat("d.dd")
    else
      z0 = theSurface.CellValue(pt0, Prj.MakeNull).SetFormat("d.dd")
    end
    ' Get the next point
    pt2 = thePointList.Get(i + 1)
    x2 = pt2.GetX.SetFormat("d.dd")
    y2 = pt2.GetY.SetFormat("d.dd")
    if (theSurfaceType = "STheme") then
      z2 = theSurface.Elevation(pt2).SetFormat("d.dd")
    else
      z2 = theSurface.CellValue(pt2, Prj.MakeNull).SetFormat("d.dd")
    end

    ' Backsight
    ' Get the azimuth backsight
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,x0,y0})
    theOutList.Add(theSurveyId.AsString + ",BS," + i.AsString + "," +
                   (i - 1).AsString + "," + theAzimuth.AsString +
                   "," + z1.AsString + "," +
                   pt1.Distance(pt0).AsString)

    ' Sideshots bisect the angle between backsight and foresight
    for each n in 1 .. theNumSs
      d = theSsDistance * n

      ' Get the X & Y components of the vectors
      vBsx = x0 - x1
      vBsy = y0 - y1
      vFsx = x2 - x1
      vFsy = y2 - y1
      ' Get the magnitudes of the vectors
      vBs = ((vBsx^2) + (vBsy^2)).Sqrt
      vFs = ((vFsx^2) + (vFsy^2)).Sqrt
```

```
      ' Get the components of the unit vectors
      uvBsx = vBsx / vBs
      uvBsy = vBsy / vBs
      uvFsx = vFsx / vFs
      uvFsy = vFsy / vFs
      ' Add the unit vectors to get bisector
      bvX = uvBsx + uvFsx
      bvY = uvBsy + uvFsy
      ' Get the magnitude of the bisector
      vB = ((bvX^2) + (bvY^2)).Sqrt
      ' Get the components of the bisector unit vector
      uvBx = bvX / vB
      uvBy = bvY / vB
      ' Multiply the components by distance
      cvX = uvBx * d
      cvY = uvBy * d

      ' If the magnitude is small then not a perpendicular side shot
      if (vB > (0.0001 * pt0.Distance(pt2))) then
        ' Add the components to x1, y1
        xSs = x1 + cvX
        ySs = y1 + cvY
      else 'it is perpendicular
        theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                            {x1,y1,x0,y0})
        xSs = x1 + (d * ((theAzimuth + 90).AsRadians.Sin))
        ySs = y1 + (d * ((theAzimuth + 90).AsRadians.Cos))
      end

      ptSs = Point.Make(xSs, ySs)
      if (theSurfaceType = "STheme") then
        zSs = theSurface.Elevation(ptSs).SetFormat("d.dd")
      else
        zSs = theSurface.CellValue(ptSs,
Prj.MakeNull).SetFormat("d.dd")
      end

      'Draw the survey points on the screen?
      if (drawOnScreen) then
        theView.GetGraphics.Add(GraphicShape.Make(ptSs))
      end

      ' Get the azimuth of the point from the Station
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,xSs,ySs})

      theOutList.Add(theSurveyId.AsString + ",SD," + i.AsString +
                     "," + theSsId.AsString + "," +
                     theAzimuth.AsString + "," + zSs.AsString +
                     "," + d.AsString)

      ' Increment the side shot Id
      theSsId = theSsId + 1

     ' If the magnitude is small then not a perpendicular side shot
      if (vB > (0.0001 * pt0.Distance(pt2))) then
        ' Subtract the components from x1, y1
        xSs = x1 - cvX
        ySs = y1 - cvY
      else 'it is perpendicular
        theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                            {x1,y1,x0,y0})
        xSs = x1 - (d * ((theAzimuth + 90).AsRadians.Sin))
        ySs = y1 - (d * ((theAzimuth + 90).AsRadians.Cos))
      end

      ptSs = Point.Make(xSs, ySs)
      if (theSurfaceType = "STheme") then
        zSs = theSurface.Elevation(ptSs).SetFormat("d.dd")
      else
        zSs = theSurface.CellValue(ptSs,
Prj.MakeNull).SetFormat("d.dd")
      end

      'Draw the survey points on the screen?
      if (drawOnScreen) then
        theView.GetGraphics.Add(GraphicShape.Make(ptSs))
      end

      ' Get the azimuth of the point from the Station
      theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                          {x1,y1,xSs,ySs})

      theOutList.Add(theSurveyId.AsString + ",SD," + i.AsString +
                     "," + theSsId.AsString + "," +
                     theAzimuth.AsString + "," + zSs.AsString +
                     "," + d.AsString)

      ' Increment the side shot Id
      theSsId = theSsId + 1

    end

    ' Foresight
    theAzimuth = av.run("Pegger.RoadMenu.Survey.Function.Azimuth",
                        {x1,y1,x2,y2})
    theOutList.Add(theSurveyId.AsString + ",FS," + i.AsString +
                   "," + (i + 1).AsString + "," +
                   theAzimuth.AsString + "," + z1.AsString +
                   "," + pt1.Distance(pt2).AsString)
  end

  i = i + 1
end
'**********************************************************************
'Make output file by writing list to theOutFile
theLineFile = LineFile.Make(theOutFile, #FILE_PERM_WRITE)
theLineFile.Write(theOutList, theOutList.Count)
theLineFile.Close
'**********************************************************************
'Clear the status bar
av.ClearMsg
av.ClearStatus
```

```
    return nil
```

```
' Pegger.RoadMenu.Survey.Function.Azimuth
'
' Created By:  Luke Rogers
'              In pursuit of a Masters of Science
'              Forest Engineering
'              College of Forest Resources
'              University of Washington
'              Box 352100
'              Seattle, WA 98195
'
'              lwrogers@u.washington.edu
'
'              December 11th, 2004
'
' Description: Calculates the angle created by two points.
'              Based on original work by David F. Kimball.
'
' Calls:       Nothing
'
' Returns:     Returns the angle (azimuth) formed by the two
'              points as a number from 0 - 360.
'
'**********************************************************************
'----------------------------------------------------------------------
'the script must be passed a list containing either 2 pts or 4 numbers
'Sample script call:    av.Run("ReturnLineSegmentAzimuth",{1,1,6,9})
'                 or     av.Run("ReturnLineSegmentAzimuth",{1@1,6@9})
'----------------------------------------------------------------------
if (SELF.Count = 4) then
  x1 = SELF.Get(0)                 'xcoord of first (origin) point
  y1 = SELF.Get(1)                 'ycoord of first (origin) point
  x2 = SELF.Get(2)                 'xcoord of second point
  y2 = SELF.Get(3)                 'ycoord of second point
  p1 = Point.Make(x1,y1)
  p2 = Point.Make(x2,y2)
elseif (SELF.Count = 2) then
  p1 = SELF.Get(0)                 'first (origin) point
  p2 = SELF.Get(1)                 'second point
  x1 = p1.GetX
  y1 = p1.GetY
  x2 = p2.GetX
  y2 = p2.GetY
else
  return nil
end
'----------------------------------------------------------------------
'calculate the angle using simple trig:
'----------------------------------------------------------------------
h = p1.Distance(p2)                'h  = the distance between the pts
dX = x2 - x1                       'dX = difference in xcoords
dY = y2 - y1                       'dY = difference in ycoords
a = 90 - ((dX / h).ACos.AsDegrees) 'a  = the angle made by the points
if (dX < 0) then
  if (dY < 0) then
    a = 180 + a.Negate
  else
    a = 360 + a
  end
else
  if (dY < 0) then
    a = 180 - a
  end
end
'----------------------------------------------------------------------
'return the angle as a number between 0 and 360:
'----------------------------------------------------------------------
return a
```