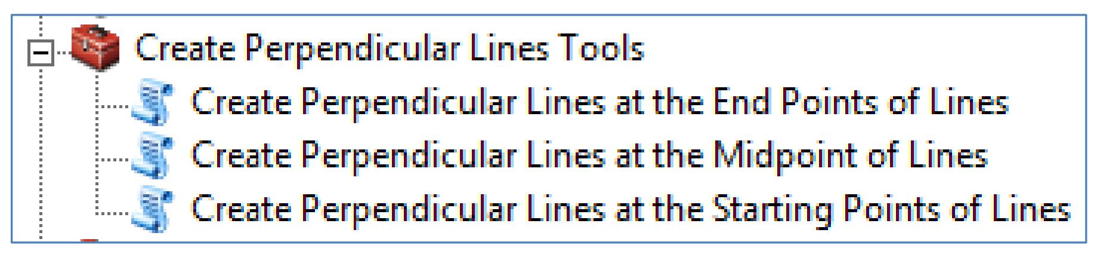
An ArcGIS/Python toolset to automate the creation of lines perpendicular to the lines in a feature class.

by Gerry Gabrisch
GIS Manager, Lummi Indian Business Council
geraldg@lummi-nsn.gov   (360) 384-2372

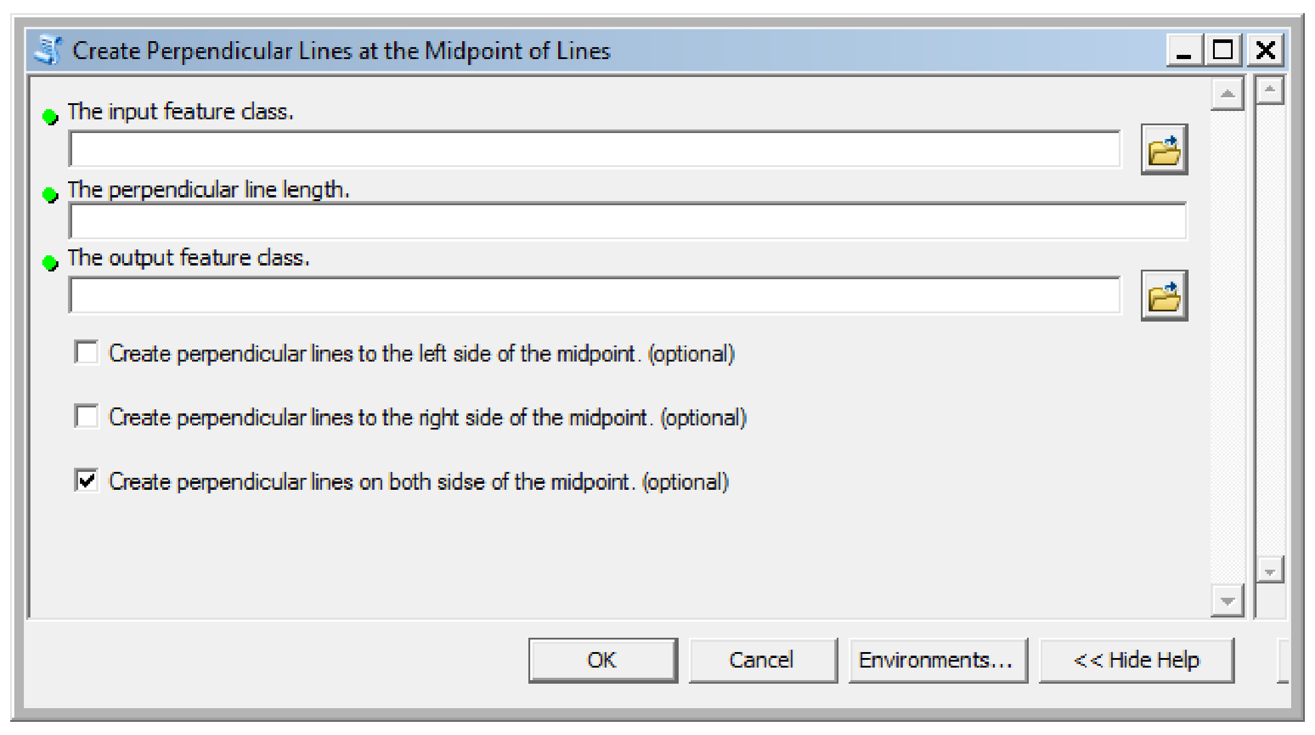
This poster describes a set of Python/ArcGIS-ArcToolbox scripts that create lines perpendicular to each line record in a feature class. Three scripts/tools are included in the *Create Perpendicular Lines* toolbox that allow the user to generate perpendicular lines at different line intersection points and different sides of the lines at user-defined distances. The three tools include *Create Perpendicular Lines at the Midpoint of a Line*, *Create Perpendicular Lines at the End Points of Lines*, and *Create Perpendicular Lines at the Start Point of a Line*. Each script outputs a new feature class in the same projection/coordinate system and datum as the input data.

Because these tools rely on feature geometry to perform a number of trigonometric calculations and coordinate system conversions, a projected coordinate system is required. The *Create Perpendicular Lines* tools are subject to error introduced by map projections. Selecting a projection that minimizes the distortion of direction for your specific study area is recommended.

These tools were generated for ArcGIS 10 and operated under an ArcView license, but ArcGIS 9x versions are available on request.


Create Perpendicular Lines Tools
Create Perpendicular Lines at the End Points of Lines
Create Perpendicular Lines at the Midpoint of Lines
Create Perpendicular Lines at the Starting Points of Lines

The *Create Perpendicular Lines at the Midpoint of a Line* user interface is shown to the right. Required input parameters include an input feature class of lines, an output feature class, and the desired length of the perpendicular line (in feature class units of measure). Optional parameters allow the user to create lines that extend to both sides of the midpoint, to either the right side or left side of the midpoint, or any combination. If all three check boxes are selected, three different perpendicular lines are created in the output feature class for each record in the input feature class.



Create Perpendicular Lines at the Midpoint of Lines

The input feature class.

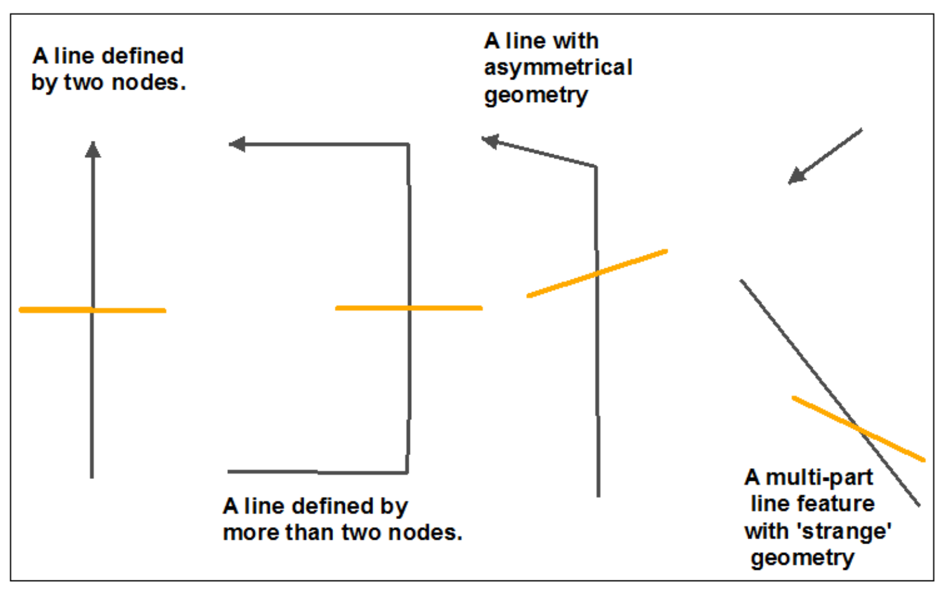
The perpendicular line length.

The output feature class.

☐ Create perpendicular lines to the left side of the midpoint. (optional)

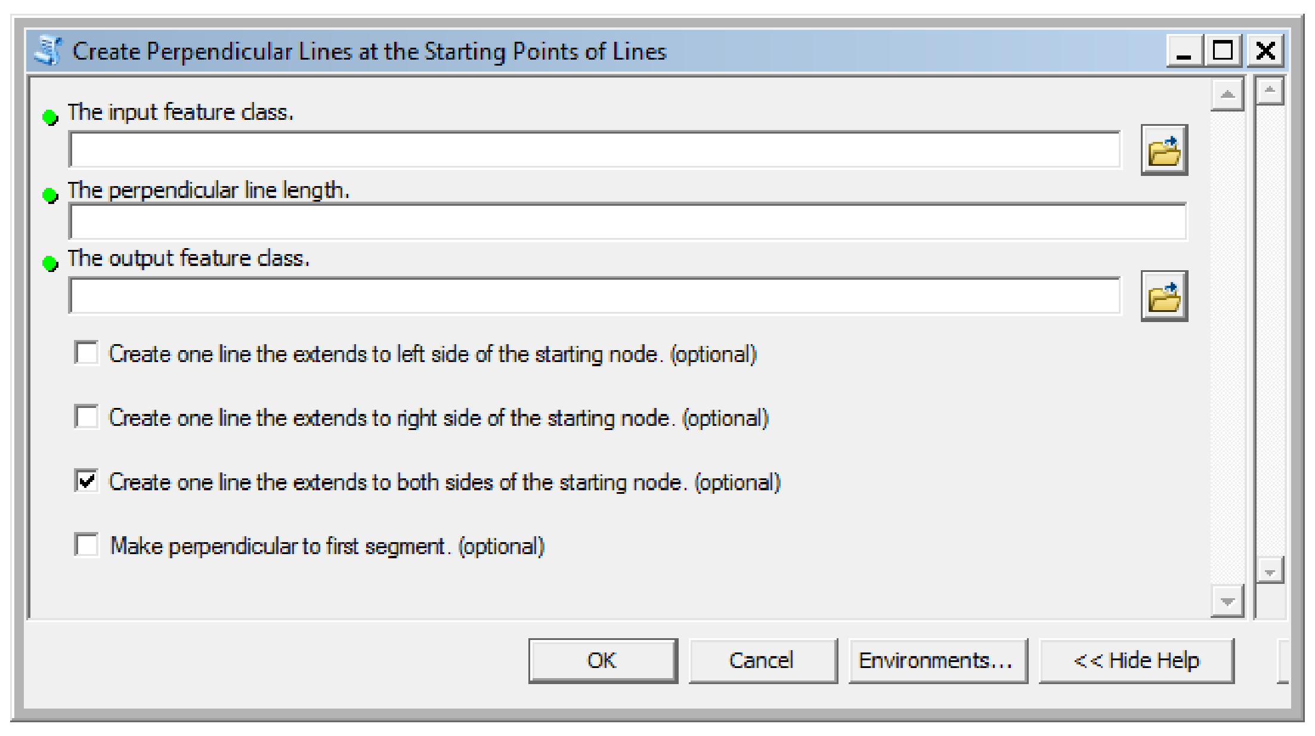
☐ Create perpendicular lines to the right side of the midpoint. (optional)

☑ Create perpendicular lines on both sides of the midpoint. (optional)


A line defined by two nodes.
A line with asymmetrical geometry
A line defined by more than two nodes.
A multi-part line feature with 'strange' geometry

The mid-point tool's output lines are perpendicular to an imaginary line extending from the start vertex to the end vertex. Complex line geometries and multi-part features can return seemingly odd results as shown above in orange.

Perpendicular lines can also be created that intersect the start vertex of a line. The start tool allows the user to create lines that are perpendicular to the first line segment (vertex 0 to vertex 1), or perpendicular to an imaginary line that extends from the start vertex to the end vertex.



Create Perpendicular Lines at the Starting Points of Lines

The input feature class.

The perpendicular line length.

The output feature class.

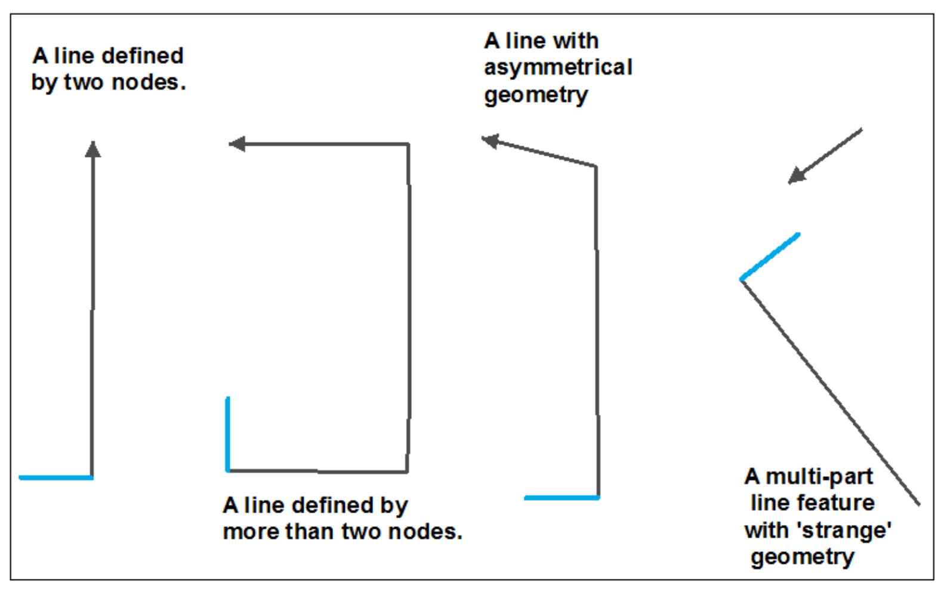
☐ Create one line the extends to left side of the starting node. (optional)

☐ Create one line the extends to right side of the starting node. (optional)

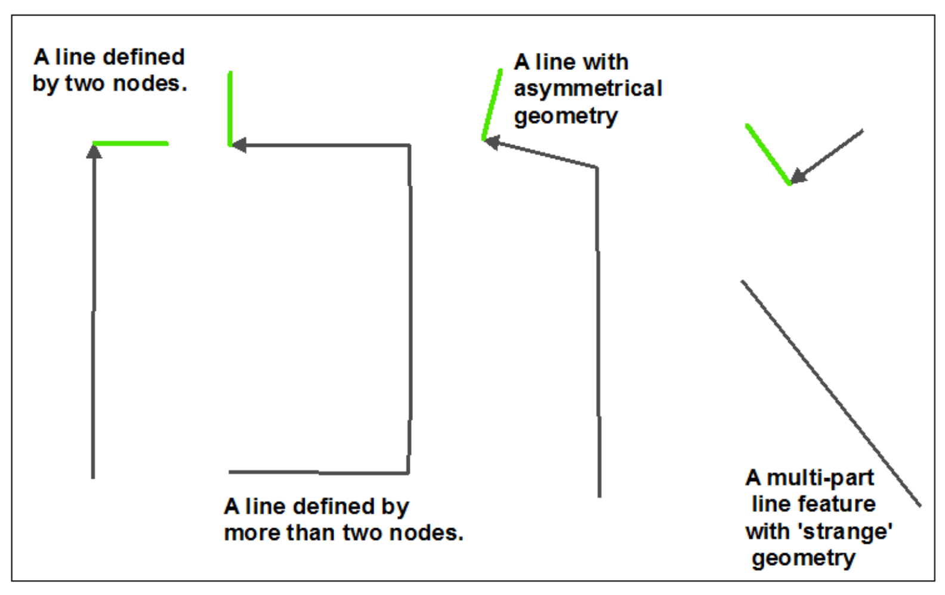
☑ Create one line the extends to both sides of the starting node. (optional)

☐ Make perpendicular to first segment. (optional)

OK
Cancel
Environments...
<< Hide Help

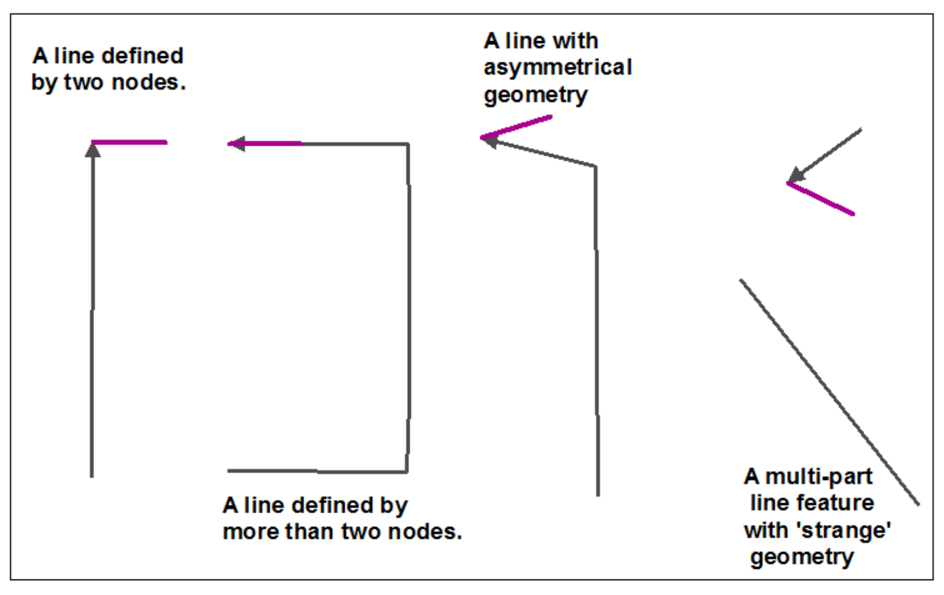

A line defined by two nodes.
A line with asymmetrical geometry
A line defined by more than two nodes.
A multi-part line feature with 'strange' geometry

Above is an example of lines generated to the left of the start vertices and perpendicular to the first line segment of each record (shown in cyan).


A line defined by two nodes.
A line with asymmetrical geometry
A line defined by more than two nodes.
A multi-part line feature with 'strange' geometry

Above is an example of lines generated to the right side of the end vertices and perpendicular to the last line segment of each record (shown in green).

Finally, perpendicular lines can be created that intersect the end vertex of a line. The end tool allows the user to create lines that are perpendicular to the last line segment (end vertex to vertex -2), or perpendicular to an imaginary line that extends from the start vertex to the end vertex.


A line defined by two nodes.
A line with asymmetrical geometry
A line defined by more than two nodes.
A multi-part line feature with 'strange' geometry

Above is an example of lines generated to the right side of the end vertices and perpendicular to imaginary line connecting the start and end vertex (shown in magenta).

Technical Challenges and Beneficial Outcomes

1. ArcGIS-arcpy line geometry properties do not necessarily return the true midpoint of a line, only the center of gravity of a line, or the label point of the line. To overcome the inability to reliably identify a line's midpoint programmatically, I created subroutines to read each feature's geometry and 'walk' up each record segment-by-segment until the true midpoint is identified. These subroutines were easily adapted into other geoprocessing tools (not shown) to place a point at a specific line percentage along a line, and create multiple points as specific distances along a line (for example, river mile points). These line walking subroutines do not require ArcObject programming, operate outside an edit session unlike the *proportion* tool on the ArcGIS COGO toolbar, do not require a linear referencing toolbox, and work with an ArcView license level.

2. These tools convert data between polar coordinates (radians extending counter clockwise where 0 is due east) and degrees travelling clockwise from grid north (zero at north). A number of subroutines were developed that handle the conversions between degrees from the unit circle (counter clockwise) to degrees from grid north (clockwise) which are more intuitive than native Python math module methods.

Future Development

1. These tool currently only return perpendicular lines. Future developments will include lines at any user-defined angle.

2. All perpendicular lines must intersect the midpoint or end vertices. Future developments may include intersections at any point along the line feature.

3. Is there something else you would like to see? If so, pencil your request below:

Applications

1. This tool has supported research at Western Washington University to automatically generate beach elevation profiles perpendicular to the mean higher high water line; the profiles were used as inputs to a two-dimensional beach erosion model. (Grilliot, M.J. 2009. Rising seas and sandy beach transgressions : a study in northern Puget Sound, WA. M.S. Thesis, Western Washington University. Bellingham, WA.)

2. These tools were also used to support the Institute For Watershed Studies *Lake Whatcom Bathymetry and Morphology* report (Mitchell et al. 2010) to generate lines of the greatest width (perpendicular to the geometric centerline) and breadth lines (lines perpendicular to the line of the greatest fetch) of Lake Whatcom in Whatcom County Washington. http://ceratium.ietc.wwu.edu/IWS2/lakestudies/lakewhatcom/online_pdf/bathymetry2010.pdf

3. These tools were also used to support the Lummi Nation Coastal Zone Management Plan to identify average slopes within 200 ft of the mean higher high water line. http://lnnr.lummi-nsn.gov/LummiWebsite/userfiles/1_MHMP_2010FINAL-wAppendices.pdf

Source code for the midpoint tool

```
try:
    import sys, string, os, arcpy, traceback, math
    def AddAndSubtractRadians(theta):
        return (theta + 1.57079632679, theta - 1.57079632679)
    def Distance(x1, y1, x2, y2):
        '''Cartesian distance formula'''
        return float(math.pow(((math.pow((x2-x1),2)) + (math.pow((y2 - y1),2))),.5))
    def CartesianToPolar(xy1, xy2):
        '''Given coordinate pairs as two lists or tuples, return the polar
        coordinates with theta in radians. Values are in true radians along the
        unit-circle, for example, 3.14 and not -0 like a regular python
        return.'''
        try:
            x1, y1, x2, y2 = float(xy1[0]), float(xy1[1]), float(xy2[0]), float(xy2[1])
            xdistance, ydistance = x2 - x1, y2 - y1
            distance = math.pow(((math.pow((x2 - x1),2)) + (math.pow((y2 - y1),2))),.5)
            if xdistance == 0:
                if y2 > y1:
                    theta = math.pi/2
                else:
                    theta = (3*math.pi)/2
            elif ydistance == 0:
                if x2 > x1:
                    theta = 0
                else:
                    theta = math.pi
            else:
                theta = math.atan(ydistance/xdistance)
                if xdistance > 0 and ydistance < 0:
                    theta = 2*math.pi + theta
                if xdistance < 0 and ydistance > 0:
                    theta = math.pi + theta
                if xdistance < 0 and ydistance < 0:
                    theta = math.pi + theta
            return [distance, theta]
        except:
            print"Error in CartesianToPolar()"
    def PolarToCartesian(polarcoords):
        '''A tuple, or list, of polar values(distance, theta in radians) are
        converted to cartesian coords'''
        r = polarcoords[0]
        theta = polarcoords[1]
        x = r * math.cos(theta)
        y = r * math.sin(theta)
        return [x, y]
    def gPrint(string):
        #print string
        arcpy.AddMessage(string)
    def GetMidpointOfAStraightLine( x1, y1, x2, y2):
        midx = (x1 + x2)/2
        midy = (y1 + y2)/2
        return [midx, midy]
    def GetSegmentLengths(thisrecordsgeometry):
        lengthlist = []
        positiontracker = 0
        totaldistance = 0
        try:
            for item in thisrecordsgeometry:
                x1 = thisrecordsgeometry[positiontracker][0]
                y1 = thisrecordsgeometry[positiontracker][1]
                x2 = thisrecordsgeometry[positiontracker +1][0]
                y2 = thisrecordsgeometry[positiontracker +1][1]
                segmentdistance = Distance(x1, y1, x2, y2)
                totaldistance = totaldistance + segmentdistance
                lengthlist.append(totaldistance)
                positiontracker += 1
        except:
            pass
        return lengthlist
    def stringToBoolean(x):
        if x == 'true':
             x = True
        else:
             x = False
        return x
    gPrint("\n      Runing...Create Perpendicular Lines At The Midpoint of Line Features")
    gPrint("      A Two-Bit Algorithms product.\n      Copyright 2012 Gerry Gabrisch (gerry@gabrisch.us)\n")
    infc = arcpy.GetParameterAsText(0)
    distance = arcpy.GetParameterAsText(1)
    fcname = arcpy.GetParameterAsText(2)
    leftside = arcpy.GetParameterAsText(3)
    rightside = arcpy.GetParameterAsText(4)
    bothsides = arcpy.GetParameterAsText(5)

    leftside = stringToBoolean(leftside)
    rightside = stringToBoolean(rightside)
    bothsides = stringToBoolean(bothsides)

    gPrint("      Cracking Features...")
    #Get the input line files geometry as a python list.
    desc = arcpy.Describe(infc)
    shapefieldname = desc.ShapeFieldName
    rows = arcpy.SearchCursor(infc)
    listofpointgeometry = []
    gPrint("      Finding Midpoints...")
    for row in rows:
        feat = row.getValue(shapefieldname)
        midpointDistance = (feat.length/2)
        partnum = 0
        partcount = feat.partCount
        thisrecordsgeometry = []
        while partnum < partcount:
            part = feat.getPart(partnum)
            pnt = part.next()
            pntcount = 0
            while pnt:
                thetuple = [pnt.X, pnt.Y]
                thisrecordsgeometry.append(thetuple)
                pnt = part.next()
                pntcount += 1
            partnum += 1
        startnode = [thisrecordsgeometry[0][0], thisrecordsgeometry[0][1]]
        endnode = [thisrecordsgeometry[-1][0], thisrecordsgeometry[-1][1]]
        #If the line is straight, then do this simple calculation to get its midpoint....
        if len(thisrecordsgeometry) == 2:
            midpoint = GetMidpointOfAStraightLine( thisrecordsgeometry[0][0], thisrecordsgeometry[0][1],thisrecordsgeometry[1][0], thisrecordsgeometry[1][1])
        #If the line is made of more than two verticies, sum up the line segments until the center point is located...
        else:
            listofsegmentlengths = GetSegmentLengths(thisrecordsgeometry)
            nodecounter = 1
            for item in listofsegmentlengths:
                if item < midpointDistance:
                    nodecounter += 1
                else:
                    distanceback = item - midpointDistance
                    break
            polar = CartesianToPolar(thisrecordsgeometry[nodecounter], thisrecordsgeometry[nodecounter -1])
            polardistancetomidpoint = [distanceback, polar[1]]
            midpointascartesiantemp = PolarToCartesian(polardistancetomidpoint)
            midx = thisrecordsgeometry[nodecounter][0] + midpointascartesiantemp[0]
            midy = thisrecordsgeometry[nodecounter] [1] + midpointascartesiantemp[1]
            midpoint = [midx,midy]
        listofpointgeometry.append([startnode,midpoint,endnode])
    #Create the data models to store the new geometery....
    featureList = []
    array = arcpy.Array()
    pnt = arcpy.Point()
    gPrint("      Defining Perpendicular Lines...")
    for pt in listofpointgeometry:
        startx = pt[0][0]
        starty = pt[0][1]
        midx = pt[1][0]
        midy = pt[1][1]
        endx = pt[2][0]
        endy = pt[2][1]
        #use the start point and end point to get a theta
        polarcoor = CartesianToPolar((startx,starty), (endx,endy))
        #Add and subtract the 90 degrees in radians from the line...
        ends = AddAndSubtractRadians(polarcoor[1])
        firstend = PolarToCartesian((float(distance),float(ends[0])))
        secondend = PolarToCartesian((float(distance),float(ends[1])))
        firstx2 = midx + firstend[0]
        firsty2 = midy + firstend[1]
        secondx2 = midx + secondend[0]
        secondy2 = midy + secondend[1]
        if bothsides:
            pnt.X, pnt.Y = firstx2 , firsty2
            array.add(pnt)
            pnt.X, pnt.Y = secondx2 , secondy2
            array.add(pnt)
            polyline = arcpy.Polyline(array)
            array.removeAll()
            featureList.append(polyline)
        if leftside:
            pnt.X, pnt.Y = midx , midy
            array.add(pnt)
            pnt.X, pnt.Y = firstx2 , firsty2
            array.add(pnt)
            polyline = arcpy.Polyline(array)
            array.removeAll()
            featureList.append(polyline)
        if rightside:
            pnt.X, pnt.Y = pnt.X, pnt.Y = midx , midy
            array.add(pnt)
            pnt.X, pnt.Y = secondx2 , secondy2
            array.add(pnt)
            polyline = arcpy.Polyline(array)
            array.removeAll()
            featureList.append(polyline)
    gPrint("      Creating New Feature Class...")
    arcpy.CopyFeatures_management(featureList, fcname)
    spatialRef = arcpy.Describe(infc).spatialReference
    arcpy.DefineProjection_management(fcname, spatialRef)
    gPrint ("      Done")
except arcpy.ExecuteError:
    msgs = arcpy.GetMessages(2)
    arcpy.AddError(msgs)
    gPrint(msgs)
except:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback info:\n" + tbinfo + "\nError Info:\n" + str(sys.exc_info()[1])
    msgs = "ArcPy ERRORS:\n" + arcpy.GetMessages(2) + "\n"
    arcpy.AddError(pymsg)
    arcpy.AddError(msgs)
    gPrint(pymsg + "\n")
    gPrint (msgs)
```

Download the tool for here:

More scripts at this website:

Custom tool development?
Contact information here: