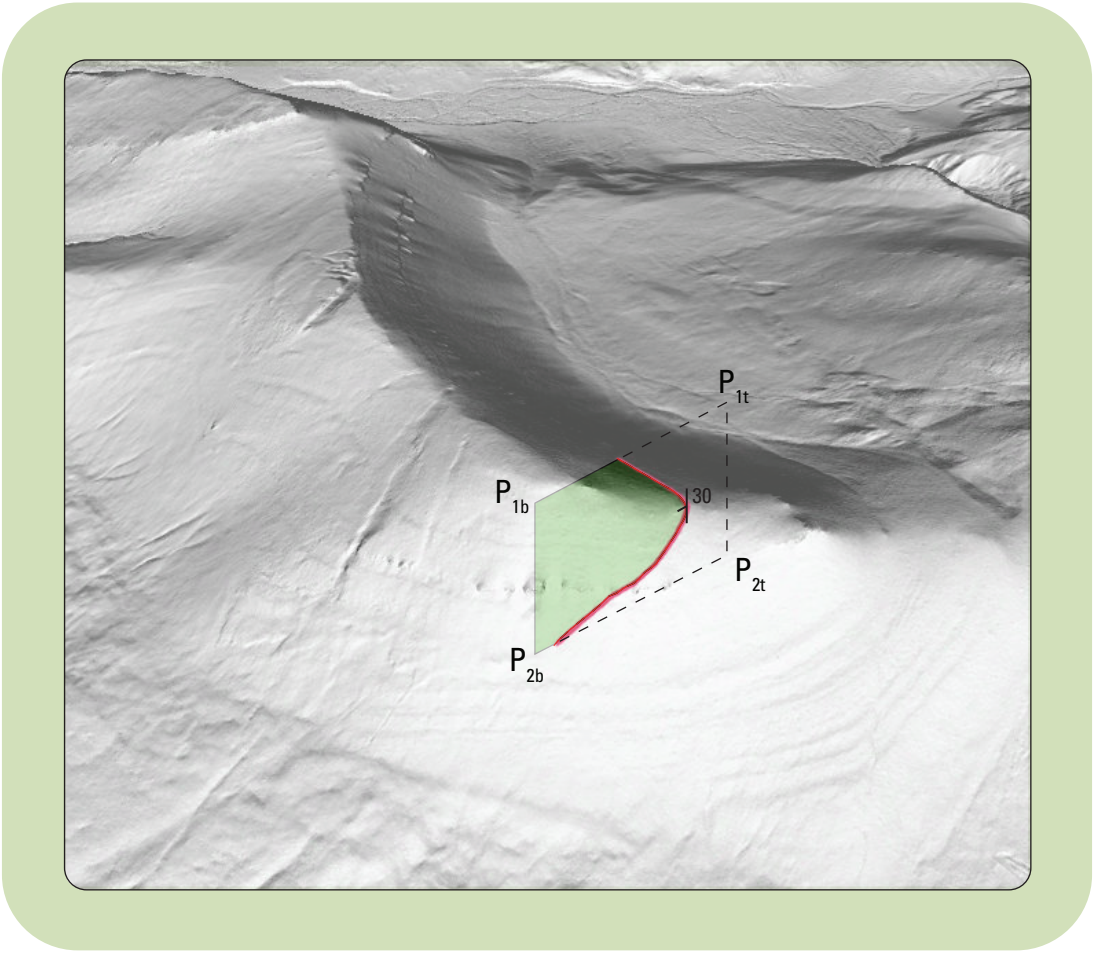
USGS
science for a changing world

Prepared in cooperation with Eastern Washington University

The Surface Trace Tool—Modeling Complex Planar Interactions Using ArcGIS



Open-File Report 2019–1136

U.S. Department of the Interior
U.S. Geological Survey

Cover. Hill-shaded terrain with the surface trace of the intersection between a three-dimensional plane and the ground surface.

The Surface Trace Tool—Modeling Complex Planar Interactions Using ArcGIS

By Drew B. Adams and Heather L. Parks

Prepared in cooperation with Eastern Washington University

Open-File Report 2019–1136

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
DAVID BERNHARDT, Secretary

U.S. Geological Survey
James F. Reilly II, Director

U.S. Geological Survey, Reston, Virginia: 2020

Suggested citation:
Adams, D.B., and Parks, H.L., 2020, The surface trace tool—Modeling complex planar interactions using ArcGIS: U.S. Geological Survey Open-File Report 2019–1136, 14 p., https://doi.org/10.3133/ofr20191136.

ISSN 2331-1258 (online)

Contents

Figures

Conversion Factors

International System of Units to U.S. customary units

Multiply	By	To obtain
	Length	
meter (m)	3.281	foot (ft)
meter (m)	1.094	yard (yd)

Datum

Horizontal coordinate information is referenced to the North American Datum of 1983 (NAD 83), UTM Zone 12N.

The Surface Trace Tool—Modeling Complex Planar Interactions Using ArcGIS

By Drew B. Adams and Heather L. Parks

Introduction

The surface trace tool comprises a Python script written for ArcGIS that will determine the line of intersection between a planar feature and a surface. Specifically, this tool was designed for geologic applications where geologic planar-feature orientations are reported as strike and dip, and the intersecting surface is the ground. The tool output will show how planar geologic layers intersect with topography.

Determining where geologic features crop out on the surface can be used to guide new geologic mapping as well as reviewing existing geologic mapping. This tool was developed to aid in more efficient mapping of an unknown area. These unknown areas may be missing data, either owing to a lack of suitable outcrops or being difficult to traverse, and data about the areas may be extrapolated using this tool and surrounding data to determine where planar features might appear on the ground.

Tool Usage

This tool requires ArcMap (ArcGIS desktop) and the 3D Analyst extension. In order to use this tool, you will need a triangulated irregular network (TIN) of the surface elevation in the area of study and a point feature class of strike and dip measurement sites with the attributes azimuthal strike, dip, and unique ID. The width of the plane used to calculate the surface trace is specified by the user when running the tool (fig. 1).

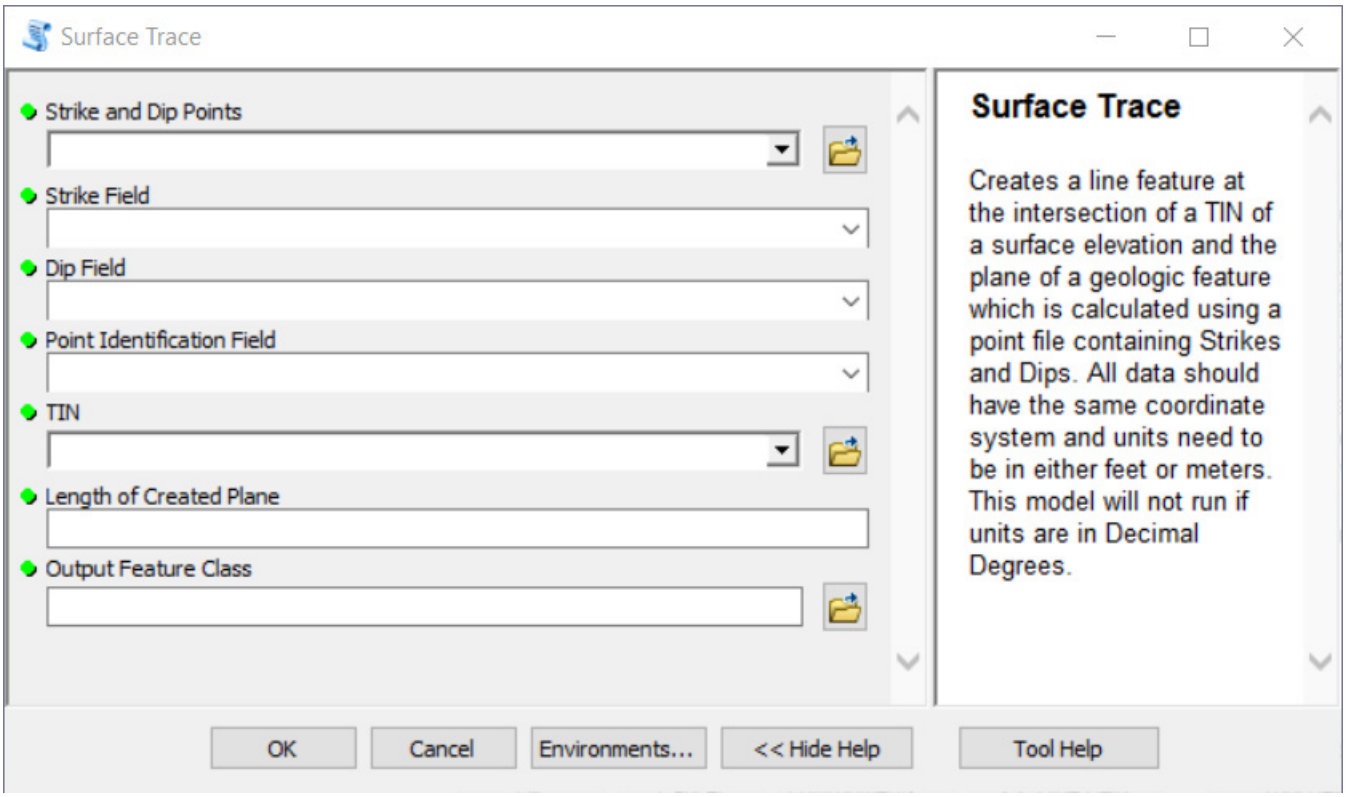


Figure 1. Screenshot showing the SurfaceTrace tool dialog box. Parameters are described in the following list.

- The "Strike Field" values must be in azimuthal (not quadrant) format. The values of strike must be between 0 and 359 degrees, measured clockwise from north at 0 degrees.
- The "Dip Field" values must be between 0 and 90 degrees. Overturned beds should be formatted with a positive dip. This tool can currently take vertical dips as an input but will calculate a plane that has a dip of 89.99 degrees because ArcGIS will not create a vertical planar feature.
- Note that the strike and dip measurements must follow the right-hand rule, where the down-dip direction is 90 degrees clockwise from the reported strike.
- The "Point Identification Field" values must be unique in order to relate the created linear features calculated with the tool with the point where the strike and dip were measured.
- The "Length of Created Plane" can be any value the user chooses. The units are defined by the spatial reference system of the input point feature class.

The tool creates a square plane that is oriented in the same manner as the original measurement. The width of the plane is specified by the user ("Length of Created Plane" input parameter). Because the plane that is created is a square, the "width" describes both the length of the the down-dip direction and the strike direction. This created plane will be used to calculate the surface trace. The length of the calculated trace will be a function of the dip and the complexity of the topographic surface.

The chosen width of the created plane used to calculate the surface trace should reflect the geologic complexity of the area. For less complex areas (laterally continuous planar features), the user may choose to input wider planes (longer traces). However, in areas with more complex structures and features, the width of the plane should be shorter because it is more likely the planar feature will be discontinuous.

A TIN of the ground surface elevation in the area of study is required input for this tool. Raster digital elevation models (DEMs) can be used to create such TINs and are widely available at a variety of scales. A DEM of the area can be converted into a TIN using the 3D Analyst tool "Raster To TIN." When converting a DEM to TIN, the Z Tolerance tool option should be kept as low as possible, as the Z tolerance will determine the maximum allowable difference in between the height of the input raster and the height of the output TIN. A Z Tolerance of one will provide a TIN that will be accurate to the DEM within one unit. However for large DEMs, this Z Tolerance may exceed the maximum number of points available for TINs. If the number of points is exceeded, the Z Tolerance will need to be increased or the DEM divided into several smaller files.

The input DEM should have a resolution that is appropriate for the mapping scale of your project. According to Tobler, "The rule is: divide the denominator of the map scale by 1,000 to get the detectable size in meters. The resolution is one half of this amount" (Tobler, 1988). For example, detailed geologic maps that show strike and dip are commonly compiled on a 1:24,000-scale topographic basemap. For a map at this scale, the DEM should have a resolution of approximately 10 meters (m).

The area of the input DEM should extend laterally out beyond the point locations so that it will incorporate the area of the extension of the created surface trace. In other words, if the DEM extent stops at a point location, the surface trace extension will stop at that point. A DEM that extends past the outermost points to a width of at least the width of the created plane is recommended.

This tool requires a projected coordinate system that uses either feet or meters as the linear unit. The units of the TIN should be the same as the linear units of the coordinate system. *X*, *Y*, and *Z* should be in the same units. If your data are in geographic coordinate system (GCS) or similar format, you will need to reproject the data into a projected coordinate system.

Installation Instructions

1. Copy SurfaceTraceToolbox.pyt to your computer
2. Create a geodatabase that will serve as a scratch workspace (optional)
3. In ArcMap, open ArcToolbox
4. Right click in the ArcToolbox popup window and select "Add Toolbox"
5. Navigate to the toolbox location on your computer, select it, and click open.

Details of the Process

The tool starts by calculating the geometry of a plane passing through a point using the strike and dip measurement. The plane is created by placing four points that are contained within the three-dimensional plane using the strike and dip measurements of the point. In order to place these points using trigonometric functions, the tool converts the angles from geographic measurements (north at 0 degrees) to arithmetic measurements (east at 0 degrees) which is illustrated in figure 2.

After the angle measurement is converted, the tool uses the following equations to determine the coordinates of intermediary points (P_1 and P_2) that are located along

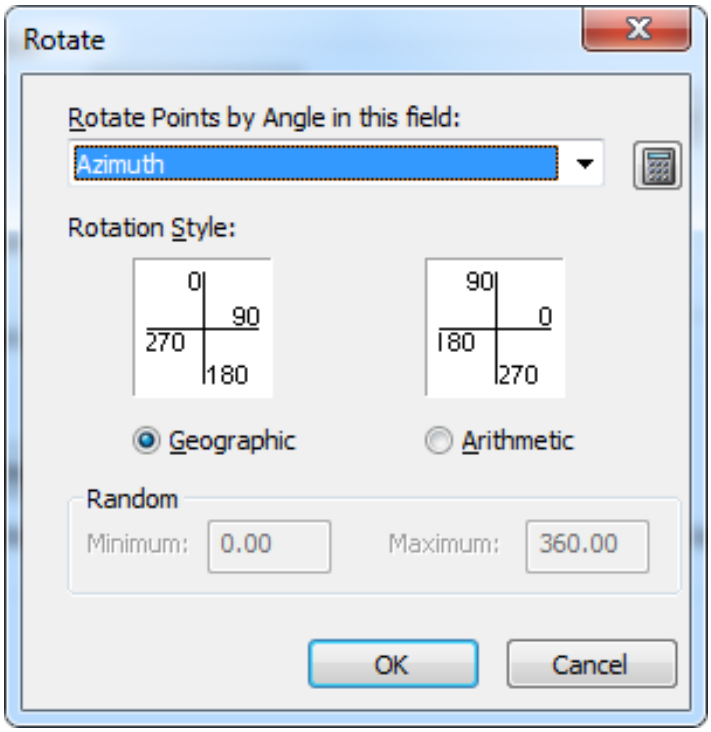


Figure 2. Screenshot showing the Rotate tool dialog box.

the strike of the created plane, where *W* is the width of the plane provided by the user ("Length of Created Plane" input parameter), *a* is the arithimitic angle of the strike, *X* is the original *x* coordinate of the sample point, Δx_1 is the change to the *x* coordinate from the original point to the midpoint, *Y* is the original *y* coordinate of the sample point, Δy_1 is the change to the y coordinate from the original point to the midpoint, and *Z* is the original *z* coordinate of the sample point, which is read from the input TIN.

$$\Delta x_1 = \cos(a)*(W/2)$$

$$\Delta y_1 = \sin(a)*(W/2)$$

$$P_1 = (X+\Delta x_1, Y+\Delta y_1, Z)$$

$$P_2 = (X-\Delta x_1, Y-\Delta y_1, Z)$$

In order to find the direction of the dip of the created plane, the following Python code was used, where S is the strike and d is the dip direction. In order to calculate the direction of the created dipping plane, the right hand rule must be used when recording strike and dip measurements, the bearing of strike is recorded so that the dip direction is 90 degrees clockwise from the strike. This clockwise direction of an angle on the coordinate plane would result in a subtraction of 90 degrees to find the dip direction. If the strike is less than 90, the result would be negative. To counteract this effect, we add this result to 360 to find the dip direction. To simplify this equation, a conditional statement was created such that if the strike is greater than 90, the angle of the dip direction would be the difference between the angle of the strike and 90. Otherwise, if the angle of strike on the coordinate plane is less than 90, the dip direction angle would be the sum of angle of strike and 270.

```
def DipDirection(S):
    if S>=90:
          d=S-90
          return d
    else:
          d=S+270
          return d
```

The coordinates of the corner points of the created plane P_{1t}, P_{2t} (top), P_{1b}, P_{2b} (bottom) are located using trigonometric functions, the measured strike, and the measured dip (fig. 3). These four points are the corners of the plane that will be created. The following equations are used to determine their coordinates, where *D* is the dip, Δxy is the measurement that can be broken down into the components of Δx_2 and Δy_2 using trigonometric functions. Δx_2 is the change to the *x* coordinate from the calculated midpoints, Δy_2 is the change in the *y* coordinate from the calculated midpoints, and Δz is the change in elevation from the original sample point.

$$\Delta xy = \cos(D)*W$$

$$\Delta x_2 = \cos(d)*\Delta xy$$

$$\Delta y_2 = \sin(d)*\Delta xy$$

$$\Delta z = \sin(D)*W$$

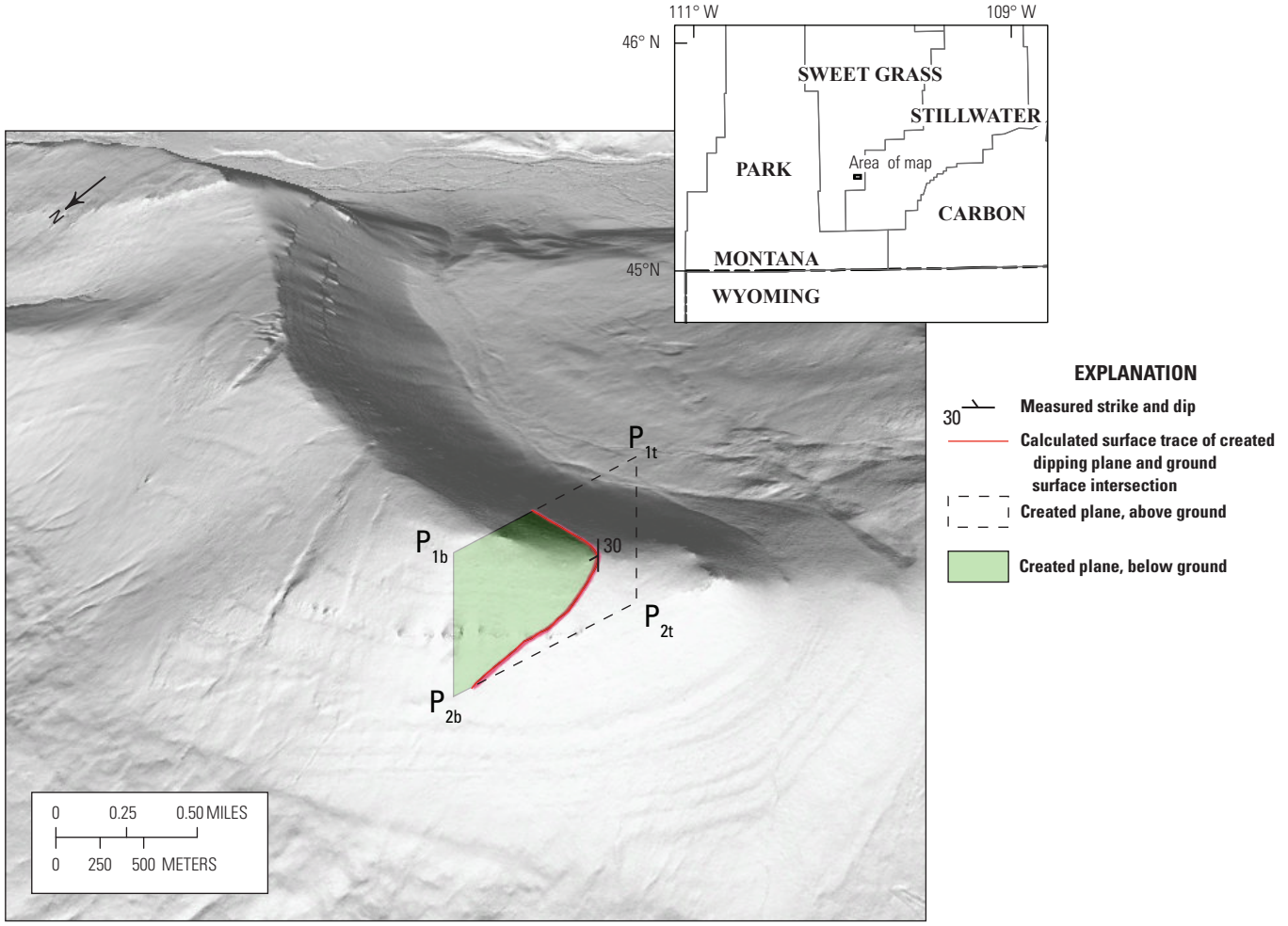
$$P_{1t} = ((X+\Delta x_1)-\Delta x_2, (Y+\Delta y_1)-\Delta y_2, Z+\Delta z)$$

$$P_{2t} = ((X-\Delta x_1)-\Delta x_2, (Y-\Delta y_1)-\Delta y_2, Z+\Delta z)$$

$$P_{1b} = ((X+\Delta x_1)+\Delta x_2, (Y+\Delta y_1)+\Delta y_2, Z-\Delta z)$$

$$P_{2b} = ((X-\Delta x_1)+\Delta x_2, (Y-\Delta y_1)+\Delta y_2, Z-\Delta z)$$

These new three-dimensional (3D) points are used to create a 3D plane (fig. 3). The plane is then converted to a TIN. This TIN of the measured plane is then compared to the TIN



Shaded relief from U.S. Geological Survey National Elevation Dataset 1-meter digital elevation model data.
GCS WGS 84 coordinate system.

Figure 3. Map showing hill-shaded terrain with a measured strike and dip point, a three-dimensional plane created by the tool, and the surface trace of the intersection between the created plane and the ground surface, Stillwater Complex, Montana.

of the surface elevation to determine the linear feature where the two planes intersect. In order to find this linear feature, this tool runs a 3D Analyst process called Surface Difference which compares two TINs and returns a polygon feature class that represents the areas above and below a specified TIN. Using the returned data, we can find the intersection of the two planes by extracting the contact between the polygons. This intersect is the resulting "surface trace" which is the calculation of where the measured plane will intersect the ground surface elevation. Identifier data are added into the new feature class so that each surface trace can be associated with the specific strike and dip point on which the tool was run. The data are then compiled into a single feature class. This new feature class's attribute table is joined with the original attribute table based upon the unique ID field, so that the new surface traces can be symbolized and interpreted more efficiently.

Notes on Using the Tool

The results of intermediate calculations produced by this tool are saved in the designated scratch geodatabase and are deleted on completion in order to save space. This tool can be run either over a network or locally, but the process will work faster when working locally. This tool was developed using ArcMap 10.5, and has not been tested using ArcPro. The output and scratch geodatabases should not have any spaces in their file paths as they may cause the tool to crash.

Data Outputs

The output of the tool is a polyline feature class that shows where a plane calculated from a strike and dip measurement at a point intersects the ground surface. The polyline

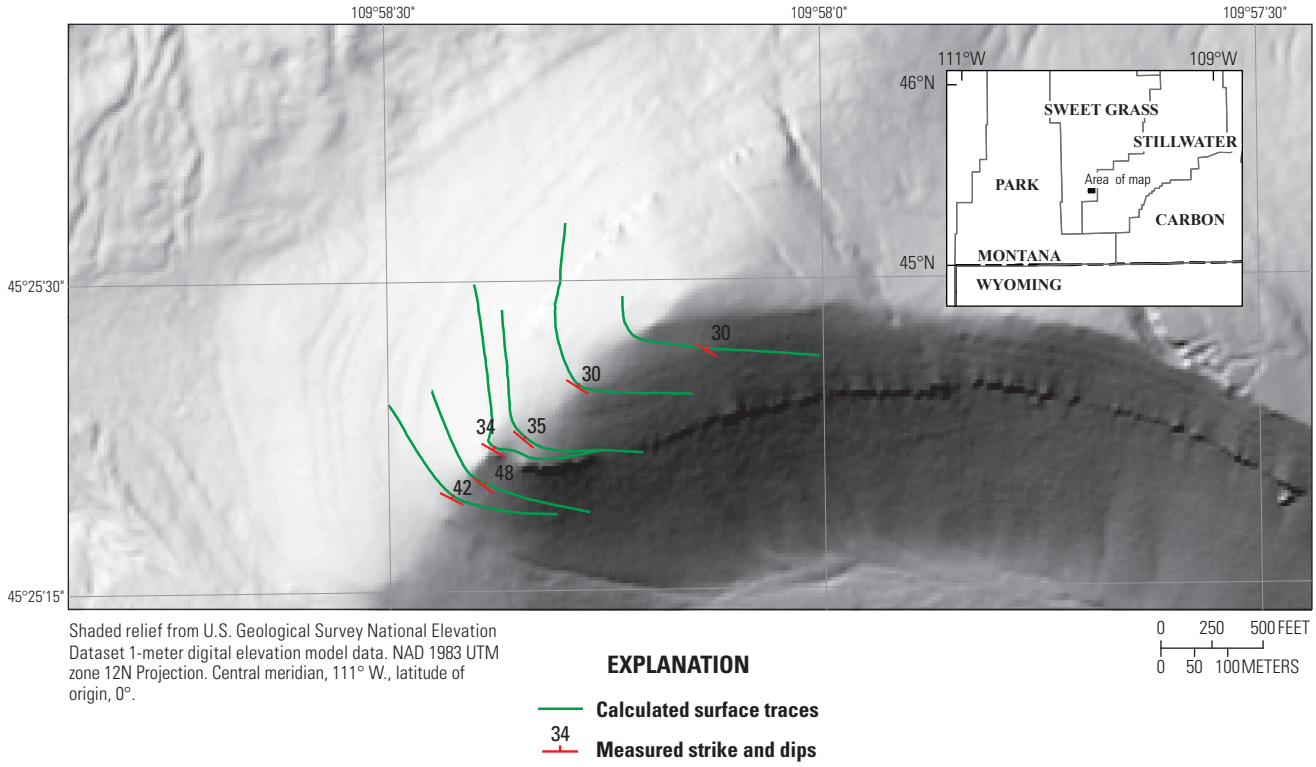


Figure 4. Map showing calculated surface traces on a hillshade image of an area with shallow dipping layering along a ridge, Stillwater Complex, Montana. Strike and dip points from Geraghty (2013) and Parks and Zientek (2019).

feature will have the same attribute table as the original point strike and dip dataset owing to a join of the original attribute table and the output surface trace's attribute table.

An example location was chosen from the vicinity of the Stillwater Complex, Montana, where strike and dip data are available and bedding structures are visible on the DEM. Using the measurement of these planar features, the surface trace tool produces linear features showing where geologic features would crop out on the ground. The example was calculated using a 5-m DEM created from a 1-m lidar bare-earth DEM.

The example shows surface traces calculated using low-angle dip planes in an area located along a ridge (fig. 4). The surface traces are semiparallel to linear topographic features that are visible in the figure.

Acknowledgments

This project has been a part of Drew B. Adams interdisciplinary master's degree in Geography and Geology. We thank Michael Zientek and Mark Mihalasky for their ongoing support and encouragement for this project. Joshua Coyan and Joe Edgley tested this tool and provided feedback. Technical reviews were provided by Ralph A. Haugerud and Pamela M Cossette.

References Cited

Geraghty, E., 2013, Geologic map of the Stillwater Complex within the Beartooth Mountains front Laramide triangle zone south-central Montana: Montana Bureau of Mines and Geology Open File Report 645, 21 p., 1 sheet, scale 1:48,000., accessed January 4, 2018, at http://www.mbmg.mtech.edu/mbmgcat/public/ListCitation.asp?pub_id=31631&.

Parks, H.L., and Zientek, M.L., 2019, Stillwater Complex strike and dip database: U.S. Geological Survey data release, https://doi.org/10.5066/P93NUSCJ.

Tobler, Waldo, 1988, Resolution, resampling, and all that, in Mounsey, H., and Tomlinson, R., eds., Building databases for global science: London, Taylor and Francis, p. 129–137.

Appendix 1 Annotated code.

```
#Start of code. Annotations are noted with pound symbols.
import arcpy


class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name of the
        .pyt file)."""
        self.label = "SurfaceTraceToolbox"
        self.alias = "Surface Trace"
        self.name = "SurfaceTraceToolbox"

        #List of tool classes associated with this toolbox.
        self.tools = [SurfaceTrace]


class SurfaceTrace(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "Surface Trace"
        self.description = "Using a point feature class, the tool plots a line feature
class at the intersection of a measured plane and the surface elevation"
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""
        param0 = arcpy.Parameter(displayName = "Strike and Dip Points",
                 name= "strike_dip",
                 datatype = "GPFeatureLayer",
                 parameterType = "Required",
                 direction= "Input")
        param0.filter.list = ['POINT']
        param1 = arcpy.Parameter(displayName = "Strike Field",
                 name = "Strike",
                 datatype = "Field",
                 parameterType = "Required",
                 direction = "Input")
        param2 = arcpy.Parameter(displayName = "Dip Field",
                 name = "Dip",
                 datatype = "Field",
                 parameterType = "Required",
                 direction= "Input")
        param3 = arcpy.Parameter(displayName = "Point Identification Field",
                 name = "ID",
                 datatype = "Field",
                 parameterType = "Required",
                 direction = "Input")
        param4 = arcpy.Parameter(displayName = "TIN",
                 name = "Tin",
                 datatype = "GPTinLayer",
                 parameterType = "Required",
                 direction = "Input")
        param5 = arcpy.Parameter(displayName = "Length of Created Plane",
                 name = "width",
```

```
                datatype = "GPLong",
                parameterType = "Required",
                direction = "Input")
        param6 = arcpy.Parameter(displayName = "Output Feature Class",
                name = "output",
                datatype = "DEFeatureClass",
                parameterType = "Required",
                direction = "Output")

        parameters = [param0, param1, param2, param3, param4, param5, param6]
        parameters[1].parameterDependencies = [parameters[0].name]
        parameters[2].parameterDependencies = [parameters[0].name]
        parameters[3].parameterDependencies = [parameters[0].name]
        return parameters

    def isLicensed(self):
        """Set whether tool is licensed to execute."""
        if arcpy.CheckExtension("3D") == "Available" and arcpy.
CheckExtension("Spatial") == "Available":
            arcpy.CheckOutExtension("3D")
            arcpy.CheckOutExtension("Spatial")
        else:
            arcpy.AddMessage("Either 3D or Spatial Analysis License Missing")
        return True

    def updateParameters(self, parameters):
        """Modify the values and properties of parameters before internal
        validation is performed.  This method is called whenever a parameter
        has been changed."""

        return

    def updateMessages(self, parameters):
        """Modify the messages created by internal validation for each tool
        parameter.  This method is called after internal validation."""
        return

    def execute(self, parameters, messages):
        #Importing libraries into Python in order to run the tool.
        import arcpy
        import math
        import arcpy.mapping



        #Checking out extensions and setting environments.
        arcpy.env.overwriteOutput = True


        #Variables defined by the user.
        FeatureClass =  parameters[0].valueAsText
        Strike = parameters[1].valueAsText
        Dip = parameters[2].valueAsText
        ID = str(parameters[3].valueAsText)
        TINsurface = parameters[4].valueAsText
        wid = parameters[5].valueAsText
```

```
        output =  parameters[6].valueAsText


        #Formatting the ID field.
        fields = arcpy.ListFields(FeatureClass)
        for field in fields:
            if field.name == ID:
                IDtype= field.type
                IDlength = field.length
        #Variables defined by ArcMap.
        coordinatesystem = arcpy.Describe(FeatureClass).spatialReference
        width = float(wid)
        Output_RMS_file = ""
        vectorWorkspace = arcpy.env.scratchGDB
        rasterWorkspace = arcpy.env.workspace
        counter = 1

        #Creating lists to hold X, Y, and Z coordinates of points containing
strike and dip.
        xcoordinates= []
        ycoordinates= []
        zcoordinates= []

        #Placing all the X values (X coordinates) in a list to be used by the tool.
        rows = arcpy.da.SearchCursor(FeatureClass, "SHAPE@X")
        for row in rows:
            xcoordinates.append(float(row[0]))

        #Placing all the Y values (Y coordinates) in a list to be used by the tool.
        rows = arcpy.da.SearchCursor(FeatureClass, "SHAPE@Y")
        for row in rows:
            ycoordinates.append(float(row[0]))

        #Adding Z information to the points and placing Z values (Z coordinates) in a
list to be used by the tool.
        arcpy.AddSurfaceInformation_3d(FeatureClass, TINsurface, "Z", "BILINEAR", "",
"1", "0", "NO_FILTER")
        rows = arcpy.da.SearchCursor(FeatureClass, "Z")
        for row in rows:
            zcoordinates.append(float(row[0]))

        #Adding strikes to a list to be used by the tool.
        rows = arcpy.da.UpdateCursor(FeatureClass, Strike)
        strikelist = []
        for row in rows:
            strikelist.append(row[0])

        #Error catching to make sure the input data are valid.
        for strike in strikelist:
            if strike< 0 or strike > 360:
                arcpy.AddError("Strike in attribute table is out of bounds, end-
ing script")
                sys.exit(0)

        #Adding dips to a list to be used by the tool.
        rows = arcpy.da.UpdateCursor(FeatureClass, Dip)
```

```
        diplist = []
        for row in rows:
            if row[0]==90:
                diplist.append(float(89.99))
            else:
                diplist.append(float(row[0]))

        #Catching errors to make sure the input data are valid.
        for dip in diplist:
            if dip < 0 or dip > 90:
                arcpy.AddError("Dip in attribute table is out of bounds, end-
ing script")
                sys.exit(0)

        #Adding IDs to a list to be used by the tool.
        rows = arcpy.da.UpdateCursor(FeatureClass, ID)
        idlist = []
        for row in rows:
            idlist.append(row[0])



        #Progress message.
        arcpy.AddMessage("Finished cycling through lists")

        #Checking to make sure that there are unique ID numbers for all the strike and
dip points.
        def uniqueValues(X):
            s = set()
            for x in X:
                if x in s: return False
                s.add(x)
            return True
        unique = uniqueValues(idlist)
        if unique:
            arcpy.AddMessage("All ID values are unique")
        else:
            arcpy.AddError("ID values are not unique")
            sys.exit(0)




        #Creating a list to hold individual surface traces.
        traces = []

        #Starting the iterative process of creating surface traces.
        for x, y, z, dip, strike, Id in zip(xcoordinates, ycoordinates, zcoordinates,
diplist, strikelist, idlist):

            #Starting calculations of points within the same plane of the strike
and dip.

            #Secondary error catching to make sure that the script doesn't crash.
            if dip < 0 or dip > 90:
```

```
                message = "Dip out of bounds for point "+ str(Id)
                arcpy.AddError(message)
                counter += 1
            elif strike < 0 or strike > 360:
                message ="Strike out of bounds for point "+ str(Id)
                arcpy.AddError(message)
                counter +=1

        #Beginning of determining the placement of points in a plane.
        #In order to change the orientation of the angles from azimuth to coordinate
plane we need to use the following conversion.
            else:
                def conversion(strike):
                    #Use 90-strike due to the coordinate plane starting at where 90
would be in azimuth, and going the opposite the way.
                    dif = 90 - strike
                    if dif < 0:
                        orientation = 360 + dif
                        return orientation
                    else:
                        return dif

                #Code to determine the placement of points using trig functions.
                orient = conversion(strike)
                radians = math.radians(orient)
                cosine = math.cos(radians)
                sine = math.sin(radians)


                #Calculating change in X coordinates of new points along the line
of strike.
                xmid1= (width/2)*cosine
                xmid2 = xmid1
                if xmid2 > 0:
                    xmid2 =  xmid2*-1
                elif xmid2 < 0:
                    xmid2 =  abs(xmid2)
                else:
                    xmid2 = 0

                #Calculating change in Y coordinates of new points along the line
of strike.
                ymid1 = ((width)/2)*sine
                ymid2 = ymid1
                if ymid2 > 0:
                    ymid2 =  ymid2*-1
                elif ymid2 < 0:
                    ymid2 =  abs(ymid2)
                else:
                    ymid2 = 0


                #Calculating the coordinates of the new points along the line
of strike.
                xmid1 += x
                xmid2 += x
```

```
                ymid1 += y
                ymid2 += y
                zmid = z



                #Creating a function that determines the angle at which the plane is
dipping, which is normal to the line of strike.
                def dipdir(orient):
                    if orient>= 90:
                        #Use -90 instead of +90 for right hand rule since coordinate
plane is reversed from azimuth.
                        dipdirection= orient - 90
                        return dipdirection
                    else:
                        dipdirection = orient +270
                        return dipdirection

                orient = conversion(strike)
                dipdirection = dipdir(orient)

                #Determining the change in the Z coordinates.
                deltaz = (math.sin(math.radians(dip)))*(width)

                #Determining the overall change in X and Y coordinates (angled).
                deltaxy = (math.cos(math.radians(dip)))*(width)

                #Determining the component of angular change to change in X and Y
coordinates.
                deltax = (math.cos(math.radians(dipdirection)))*deltaxy
                deltay = (math.sin(math.radians(dipdirection)))*deltaxy

                #Determining coordinates for corners of the plane. The higher eleva-
tion points X and Y subtract the delta since they are opposite of the dip direction.
                xtop1 = xmid1 - deltax
                xtop2 = xmid2 - deltax
                xbot1 = xmid1 + deltax
                xbot2 = xmid2 + deltax
                ytop1 = ymid1 - deltay
                ytop2 = ymid2 - deltay
                ybot1 = ymid1 + deltay
                ybot2 = ymid2 + deltay
                ztop = zmid + deltaz
                zbot = zmid - deltaz



                #Setting a list of coordinates X, Y, and Z calculated by the above
script to be used to create a plane.
                xyz = [(xtop1,ytop1,ztop), (xtop2,ytop2,ztop), (xbot1,ybot1,zbot),
(xbot2, ybot2, zbot)]

                #Setting intermediary variables.
                testPoints = vectorWorkspace
                testname = "points" + str(Id)

                #Process: Create Feature Class.
```

```
                #Creating a point feature class to hold the calculated points that are
the corners of the designated plane.
                arcpy.CreateFeatureclass_management(testPoints, testname, "POINT",
has_m = "DISABLED", has_z = "ENABLED", spatial_reference = coordinatesystem)

                fc = testPoints + "\\" +testname
                cursor = arcpy.da.InsertCursor(fc, ["SHAPE@XYZ"])
                for row in xyz:
                    cursor.insertRow([row])

                #Checking to see if the output dataset was a shapefile or a fea-
ture class.
                checkName = output[-4:]
                shapename =""
                if checkName== ".shp":
                    shapename = output[:-4]+ str(Id)+".shp"
                else:
                    shapename = output+str(Id)


                #Setting intermediary file paths that will be deleted later.
                tin = vectorWorkspace+"\\tin"+ str(Id)
                if arcpy.Exists(tin):
                    arcpy.Delete_management(tin)
                surfacePoly = vectorWorkspace+ "\\surface" +str(Id)
                if arcpy.Exists(surfacePoly):
                    arcpy.Delete_management(surfacePoly)
                surfaceLine = vectorWorkspace+"\\surfaceline"+str(Id)
                if arcpy.Exists(surfaceLine):
                    arcpy.Delete_management(surfaceLine)
                above = "above"+str(Id)
                abovepath = vectorWorkspace + "\\above"+str(Id)
                if arcpy.Exists(abovepath):
                    arcpy.Delete_management(abovepath)
                below = "below"+str(Id)
                belowpath= vectorWorkspace+"\\below"+str(Id)
                if arcpy.Exists(belowpath):
                    arcpy.Delete_management(belowpath)
                intersect = vectorWorkspace+"\\intersect"+str(Id)
                if arcpy.Exists(intersect):
                    arcpy.Delete_management(intersect)

                arcpy.AddZInformation_3d (fc, "Z")

                arcpy.CreateTin_3d(out_tin= tin, spatial_reference= coordinatesystem,
in_features= fc, constrained_delaunay="DELAUNAY")

                #Process: Surface Difference.
                #Finding the areas above and below using the surface elevation and the
created plane.
                arcpy.SurfaceDifference_3d(tin, TINsurface, surfacePoly)


                #Converting polygon to polyline.
                #In order to extract the contact of the areas "above and below" the
surface elevation, the areas need to be converted to polylines.
```

```
                arcpy.PolygonToLine_management(surfacePoly, surfaceLine,
"IGNORE_NEIGHBORS")

                #Allowing for query of the correct features.
                delimitedField = arcpy.AddFieldDelimiters(surfaceLine, "Code")

                #Setting the query to extract two different features.
                abovecode = delimitedField+" = 1"
                belowcode = delimitedField+" = -1"


                #Creating two new feature classes that are the "above" and "below"
polygons created by the surface difference.
                arcpy.FeatureClassToFeatureClass_conversion(surfaceLine, vectorWork-
space, above, abovecode)
                arcpy.FeatureClassToFeatureClass_conversion(surfaceLine, vectorWork-
space, below, belowcode)


                #Combining the two datasets into a list to run the intersect tool.
                abovebelow = [abovepath, belowpath]

                #Running an intersect between the above and below to find the overlap-
ping surface trace.
                arcpy.Intersect_analysis(abovebelow, intersect, "ONLY_FID")

                #Adding plane ID information to the surface trace so it can be identi-
fied after merging.
                arcpy.AddField_management(intersect, "PlaneID", IDtype, IDlength)
                rows = arcpy.UpdateCursor(intersect)
                for row in rows:
                    row.setValue("PlaneID", Id)
                    rows.updateRow(row)


                #Putting individual surface traces into a list to be merged into one
feature class.
                traces.append(intersect)

                #Deleting intermediaries.
                arcpy.Delete_management(fc)
                arcpy.Delete_management(tin)
                arcpy.Delete_management(surfacePoly)
                arcpy.Delete_management(surfaceLine)
                arcpy.Delete_management(abovepath)
                arcpy.Delete_management(belowpath)

                #Progress messages.
                message = "finished iteration", counter, "of", len(idlist)
                arcpy.AddMessage(message)
                counter += 1

        mergeName =  vectorWorkspace +"\\merge"
        if arcpy.Exists(mergeName):
            arcpy.Delete_management(mergeName)
        #Merging traces together.
```

```
        arcpy.Merge_management(traces, mergeName)

        #Ensuring that surface traces that have the same ID will be a single feature.
        arcpy.Dissolve_management(mergeName, output, "PlaneID", multi_part =
"MULTI_PART")

        #Joining original feature class attribute table to the new surface traces.
        arcpy.JoinField_management(output, "PlaneID", FeatureClass, ID)

        #Deleting individual feature classes.
        for fc in traces:
            arcpy.Delete_management(fc)
        arcpy.Delete_management(mergeName)
        return
```

Menlo Park Publishing Service Center, California
Manuscript approved for publication December 3, 2019
Edited by Regan Austin
Layout and design by Kimber Petersen

ISSN 2331-1258 (online)
https://doi.org/10.3133/ofr20191136