**QuickDigitize – Plugin**

**DOCUMENTATION**

**ADD POINT LAYER TOOL**

**About the tool:**

This tool adds a \*.shp (shapefile) of the point geometry.

**How to use:**

* Load Quick Digitize plugin.
* Select the Add Point Layer button.
* In the dialog box specify the name of the layer to be loaded in the QGIS layer panel and click on the push button to get the file browser dialog and select the path and the name of the file as you want to save it.

**Note:**

*A warning message will be shown which is not of concern as we will always be working on WGS 84 for a point layer.*

**How it works:**

The entire working of the tool is contained in two Python files:

**createlayergui.py** : Contains the UI for the dialog box appearing after clicking the button.

**createpointlayertool.py:** Contains the working logic and the saves the point layer.

**createlayergui.py**

**Class Name**

**CreateLayerGui(QDialog, QWidget, Ui\_CreateLayer)**

**Methods:**

**\_\_init\_\_(self, parent, flags): ):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**initGui(self):** Refreshes the settings.

**Point(self):** Adds and sets the name of the layer in the QGIS layer panel and the geometry type to point.

**close\_1(self):** Emits signal to unset tool

**createpointlayertool.py**

**Class Name**

**CreatePointLayerTool**

**Methods:**

**\_\_init\_\_(self, iface, toolBar):**  Initializes variables and connects the signals of the dialog box to the mentioned slots.

**showDialog(self):** Links the dialog box UI to the functionalities of the toolbar.

**select\_output\_file(self):** Saves the file to the selected path from the file browser dialog box.

**close\_func(self):** Calls close\_1 method in class CreateLayerGui from createlayergui.py

**ADD LINE LAYER TOOL**

**About the tool:**

This tool adds a \*.shp (shapefile) of the line geometry.

**How to use:**

* Load Quick Digitize plugin.
* Select the Add Line Layer button.
* In the dialog box specify the name of the layer to be loaded in the QGIS layer panel and click on the push button to get the file browser dialog and select the path and the name of the file as you want to save it.

**Note:**

*A warning message will be shown saying the CRS is defaulted to WGS 84, change it to any projected CRS(WGS- 84/Pseudo Mercator is optimal) while using spline tool to draw lines.*

**How it works:**

The entire working of the tool is contained in two Python files:

**createlinelayergui.py:** Contains the UI for the dialog box appearing after clicking the button.

**createlinelayertool.py:** Contains the working logic and the saves the point layer.

**createlinelayergui.py**

**Class Name**

**CreateLineLayerGui(QDialog, QWidget, Ui\_CreateLayer)**

**Methods:**

**\_\_init\_\_(self, parent, flags): ):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**initGui(self):** Refreshes the settings.

**Line(self):** Adds and sets the name of the layer in the QGIS layer panel and the geometry type to line.

**close\_1(self):** Emits signal to unset tool

**createlinelayertool.py**

**Class Name**

**CreateLineLayerTool**

**Methods:**

**\_\_init\_\_(self, iface, toolBar):**  Initializes variables and connects the signals of the dialog box to the mentioned slots.

**showDialog(self):** Links the dialog box UI to the functionalities of the toolbar.

**select\_output\_file(self):**Saves the file to the selected path from the file browser dialog box.

**close\_func(self):**Calls close\_1 method in class CreateLineLayerGui from createlinelayergui.py

**ADD POLYGON LAYER TOOL**

**About the tool:**

This tool adds a \*.shp (shapefile) of the polygon geometry.

**How to use:**

* Load Quick Digitize plugin.
* Select the Add Polygon Layer button.
* In the dialog box specify the name of the layer to be loaded in the QGIS layer panel and click on the push button to get the file browser dialog and select the path and the name of the file as you want to save it.

**Note:**

*A warning message will be shown which is not of concern as we will always be working on WGS 84 for a polygon layer.*

**How it works:**

The entire working of the tool is contained in two Python files:

**createpolygonlayergui.pym**: Contains the UI for the dialog box appearing after clicking the button.

**createpolygonlayertool.py:** Contains the working logic and the saves the point layer.

**createpolygonlayergui.py**

**Class Name**

**CreatePolygonLayerGui(QDialog, QWidget, Ui\_CreateLayer)**

**Methods:**

**\_\_init\_\_(self, parent, flags): ):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**initGui(self):** Refreshes the settings.

**Polygon(self):** Adds and sets the name of the layer in the QGIS layer panel and the geometry type to polygon.

**close\_1(self):** Emits signal to unset tool

**createpolygonlayertool.py**

**Class Name**

**CreatePolygonLayerTool**

**Methods:**

**\_\_init\_\_(self, iface, toolBar):**  Initializes variables and connects the signals of the dialog box to the mentioned slots.

**showDialog(self):** Links the dialog box UI to the functionalities of the toolbar.

**select\_output\_file(self):** Saves the file to the selected path from the file browser dialog box.

**close\_func(self):** Calls close\_1 method in class CreatePolygonLayerGui from createpolygonlayergui.py

**ADD FIELDS TOOL**

**About the tool:**

This tool adds field names to the attribute table of the layer from a .csv file selected by the user.

**How to use:**

* Load Quick Digitize plugin.
* Select layer to whose attribute table the fields are to be added.
* Enable Toggle Edit
* Click on the Add Fields button.
* A dialog box will pop up which asks for the user to enter the input file.
* Select the appropriate .csv file and click OK.
* Open the attribute table to view the added fields.

**How it works:**

The entire working of the tool is contained in three Python files:

**ui\_addfield.py:** Contains the code for the UI of the dialog box

**addfieldsgui.py:** Links the UI components to the working logic of the dialog box

**addfieldstool.py:** Reads the field names from the user input file and adds to attribute table

Finally they are imported to the main program **quick\_digitize.py**.

**ui\_addfield.py**

**Class Name:**

Ui\_AddField(object)

**Methods:**

**setupUi(self, Dialog)**: Adds objects with specified parameters to dialog box

**retranslateUi(self, Dialog)**: Renames the labels

**addfieldsgui.py**

**Class Name:**

AddFieldsGui(QDialog, QWidget, Ui\_AddField)

**Methods:**

**\_\_init\_\_(self, parent, flags):** Initialises variables and connects the signals of the dialog box to the mentioned slots.

**initGui(self):** Refreshes the settings

**select\_input\_file(self):** opens the browse window for user to select the file and accepts the file name.

**close\_1(self):** Emits signal to unset tool

**addfieldstool.py**

**Class Name:**

AddFieldsTool

**Methods:**

**\_\_init\_\_(self, iface, toolBar):** Initialises variables and connects the signals of the dialog box to the mentioned slots.

**toggle(self):** Enables the plugin when layer is editable.

**showDialog(self**): Calls the methods of the AddFieldsGui class through an object

**add\_field(self):** Reads the .csv file and adds the fields to the attribute table

**close\_func(self):** Calls close\_1(self) from AddFieldsGui class

**Note:**

To change any of the field names, Go to the respective \*\*\*\_labels.csv and change the 2nd column and you control the placement of the labels in the setupUi\_\* method in the Ui\_AddAttribute class and the which label which row of the column is displayed in retranslateUi\_\* method of Ui\_AddAttribute class.

**ADD ATTRIBUTE TOOL**

**About the tool:**

This tool adds attributes of a selected feature to the attribute table. The dialog box allows the user to enter parameters such as the Geo- Spatial Data, Geometry type, Class, Sub-Class etc. from drop down boxes and other specifics from line edits.

**How to use:**

* Load Quick Digitize plugin.
* Select layer to whose attribute table the fields are to be added.
* Enable Toggle Edit
* Click on the Select Feature button.
* Select the feature whose attributes are to be added.
* Click on the Add Attributes button.
* The dialog box that pops up will contain the following fields: Geo- Spatial Data, Geometry type, Class, Sub-Class, Code.
* Select the options as required. Code will be assigned automatically.
* According to the Sub-Class selected the other set of fields will be shown. Fill all the fields.
* Press OK when done to add the attributes to the table.

**Note:**

1. *Make sure that the fields have been added to the attribute table before adding the attributes. Look at* ***ADD FIELDS TOOL****.*
2. *Make sure that the geometry of the layer selected and the geometry entered in the dialog box is the same.*
3. *Reload the plugin each time you select a layer to add feature attributes to. Look at* ***PLUGIN RELOADER.***

**How it works:**

The entire working of the tool is contained in three Python files:

**ui\_addattribute.py:** Contains the code for the UI of the dialog box

**addattributegui.py:** Links the UI components to the working logic of the dialog box

**addattributetool.py:** Reads the field names from the user input file and adds to attribute table.

Finally they are imported to the main program **quick\_digitize.py**.

**ui\_addattribute.py**

**Class Name:**

**Ui\_AddAttribute(object)**

**Methods:**

**setupUi\_1(self, Dialog)**: Adds objects with specified parameters to dialog box

**retranslateUi\_1(self, Dialog)**: Renames the labels

**setupUi\_2(self, Dialog)**: Adds objects with specified parameters to dialog box

**retranslateUi\_2(self, Dialog)**: Renames the labels

**addattributegui.py**

**Class Name:**

**AddAttributeGui(QDialog, QWidget, Ui\_AddAttribute)**

**Methods:**

**\_\_init\_\_(self, parent, flags):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**initGui(self):** Refreshes the settings and add items to Geo-Spatial and Geometry combo boxes.

**accept(self):** Contains the signals that are emitted when the various combo boxes are triggered and the slots that they connect to.

**append\_1(self):** Appends the data entered in the different fields of the dialog box into a single list which will be added to the attribute table.

**close\_1(self):** Emits signal to unset tool

**creation(self):** Creates the fields and corresponding labels by calling the setup function from Ui\_AddAttribute.

**deletion(self):** Deletes the fields and corresponding labels by calling the setup function from Ui\_AddAttribute.

**PyQt Slots:**

**assign\_value\_1(self, text):** Assigns value to the field, Geometry.

**assign\_value\_2(self, text):** Assigns value to field, Sub-Class.

**assign\_value\_3(self):** Assigns value to field, Class.

**assign\_value\_4(self, text):** Assigns value to field, Geo-Spatial Data.

**add\_attribute\_1(self, string):** Gets the Geometry value to be added to table.

**add\_attribute\_2(self, string):** Gets the Sub-Class value to be added to table.

**add\_attribute\_3(self, string):** Gets the Class value to be added to table.

**add\_attribute\_4(self, string):** Gets Geo-Spatial Data the value to be added to table.

**addattributetool.py**

**Class Name:**

**AddAttributeTool**

**Methods:**

**\_\_init\_\_(self, iface, toolBar):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**toggle(self):** Enables the plugin when layer is editable.

**newattribute(self):** Adds the attribute list imported from AddAttributesGui to the attribute table of the selected layer, for the selected feature.

**close\_func(self):** Calls close\_1(self) from AddAttributesGui class.

**NOTE:**

1. *To change the labels go the respective \*\*\*\*\_labels.csv file and change the first column.*
2. *To add more lineEdits go to Ui\_AddAttribute.py and add to setupUi and add the corresponding label names in retranslateUi. Add the corresponding functionality in AddAttributeGui.py if required.*
3. *Be careful about resizing the dialog to the largest size after making any changes.*
4. *To add more Geo-Spatial Data Categories add to the self.geodat array in the definition of initGui() and add the corresponding data file name as ‘ ‘\*\*\*\*\_class\_subclass.csv’ to self.filelist array.*
5. *To add a different set of fields and labels for a different Geo-Spatial Data set, go to Ui\_AddAttribute and add definitions of methods setupUi\_n and delUi\_n (where ‘n’ is any integer greater than 2). Read labels from files named \*\*\*\*\_labels.csv, named accordingly. Make the necessary changes in conditions in the class AddAttributesGui.*

**RECTANGLE TOOL**

**About the tool:**

This tool creates a rectangle feature with length on the horizontal axis and width on the vertical axis, with a click and drag. The rectangle is also resizable by dragging the extent of the second point.

**How to use:**

* Load Quick Digitize plugin.
* Select Polygon layer to which feature is to be added.
* Enable Toggle Edit.
* Click on the Rectangle Tool button.
* Left click on the desired point using the custom cursor, drag till rectangle of required size is formed and release to form the feature.
* To rotate feature, use the Rotate Feature button or the in-built Rotate feature option in QGIS.
* To add attributes to the feature, use the Add Attributes button.

**How it works:**

The entire working of the tool is contained in two Python files:

**rect.py:** Contains the logic of making the rectangle feature.

**recttool.py:** `Links the QGIS canvas to the tool’s working logic.

Finally they are imported to the main program **quick\_digitize.py**.

**rect.py**

**Class Name:**

**RectByExtentTool(QgsMapTool):**

**Methods:**

**\_\_init\_\_(self, canvas):**Initiates the variables and creates the cursor.

**canvasPressEvent(self,event):** Records the coordinates of the clicked point on the screen and converts it into map coordinates.

**canvasMoveEvent(self,event):** Tracks the movement of the cursor on the screen and sets that to be the second point. Accordingly, it calculates the two other points that form the rectangle and displays the rectangle area in-real-time.

**canvasReleaseEvent(self,event):** Checks if all rectangle has been formed and when the cursor is released, sets the polygon as feature in the layer and refreshes all the parameters for the next iteration.

**recttool.py**

**Class Name:**

**RectTool**

**Methods:**

**\_\_init\_\_(self, iface, toolBar):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**rectbyextenttool(self):** Connects to the tool when the button is triggered.

**deactivate(self):** Disconnects from the tool.

**createFeature(self, geom):** Turns on, On the fly transformation of the CRS, creates and adds the feature to the layer.

**ADD STYLE TOOL**

**About the tool:**

This tool adds custom style/symbol to all the features in the layer based on one column classification.

**How to use:**

* Load Quick Digitize plugin.
* Select layer whose features are to be assigned style/symbology categorized based on a particular attribute table column.
* Enable Toggle Edit.
* Double click on the layer name in the Layers Panel to open the Layer Properties dialog box. Go to the Styles option and in the top combo box select Categorized symbol. In the Column drop down, select the field based on which the style is to be assigned.
* Click on the Add Style button.
* A dialog box will pop up which asks for the user to enter the input file.
* Select the appropriate .qml file and click OK.
* Drag the layer to the top in the Layers Panel to view the categorized symbol applied to the features

**Warning:** *This Tool is to be used only after all the features in the layer have been digitized because QGIS does not allow editing after the layer had been assigned symbols based on categories.*

**Note:**

For the sake of testing the working of the tool, style file containing symbology for features based on the parameter “Sub-Class” has been used, following the guidelines of the Design and Standards part of the AMRUT document given by the Town and Country Planning Organisation.

**How it works:**

The entire working of the tool is contained in three Python files:

**ui\_addstyle.py:** Contains the code for the UI of the dialog box

**addstylegui.py:** Links the UI components to the working logic of the dialog box

**addstyletool.py:** Takes user input for the style file and applies style to given layer based on the column used for categorizing.

Finally, they are imported to the main program **quick\_digitize.py**.

**ui\_addstyle.py:**

**Class Name:**

Ui\_AddStyle(object)

**Methods:**

**setupUi(self, Dialog)**: Adds objects with specified parameters to dialog box

**retranslateUi(self, Dialog)**: Renames the labels

**addstylegui.py:**

**Class Name:**

AddStyleGui(QDialog, QWidget, Ui\_AddStyle)

**Methods:**

**\_\_init\_\_(self, parent, flags):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**select\_style\_file(self):** Gets the name of the style file from the browser window and outs it in the lineEdit.

**close\_1(self):** Emits signal to unset tool

**addstyletool.py**

**Class Name:**

AddStyleTool

**Methods:**

**\_\_init\_\_(self, iface, toolBar):** Initializes variables and connects the signals of the dialog box to the mentioned slots.

**toggle(self):** Enables the plugin when layer is editable.

**showDialog(self**): Calls the methods of the AddStyleGui class through an object and gets style file path from select\_style\_file() method of AddStyleGui.

**add\_style(self):** Loads the style file, applies it to layer and repaints layer.

**close\_func(self):** Calls close\_1(self) from AddStyleGui class

**SPLINE TOOL**

**About the tool:**

Spline is a pseudo-curve generating tool to draw curved lines. Try it out a few times before implementing it in an actual file.

**Note:**

Does not actually draw a vector curve but interpolates the curve with multiple line segments so that it appears smooth.

**How to use:**

* Load Quick Digitize plugin.
* Select layer to which the curved lines are to be added.
* Enable Toggle Edit
* Click on the Spline button.
* Make sure the layer is a LineLayer and it is in a projected CRS or else you will only make straight lines with no bends.(WGS- 84/Pseudo Mercator will work just fine)
* Select the layer and make sure it is in the edit mode.
* Select the tool from the tool bar and you are good to go.
* Left click on the map to start making the feature, make another click at the top of the curvature and then move the cursor left or right according to required bend and left click again to make outline of the feature, Right click to make the feature as with the normal feature creation.
* If you are not able to find the correct fit for the curve, Go to \*\*Vector\*\*->\*\*Digitize Spline\*\*->\*\* Settings\*\*(click on the \*\*rightmost and lowermost\*\*).You can change the tolerance and tightness to suite your needs.\* \*\*Higher tolerance\*\* means\*\* lesser number of interpolated points\*\* , \*\*lower\*\* the value to get a \*\*smoother curve\*\* and increase it to lessen the number of line segments required to make the curve. \*\*Lowering the tightness\*\* value will make the curve \*\*stick to the straight lines\*\* made from the marked points , to get a more bloated curve increase the value of tightness.
* To add attributes use the

> Add Attribute Button

**How it works:**

The entire working of the tool is contained in five Python files:

**spline.py**-> Contains the logic and the math involved with using the bezier curves.

**splinetool.py**->Contains linking the logic to a button in a toolbar and other GUI functionalities to program and to the settings dialog box.

**settingsdialog.py**->Links the settings dialog box UI to the functionality to the program.

**ui\_settingsdialog.py**->Contains the code for the UI.

**utils.py**->Has the Default tolerance and Default tightness.

Finally, they are imported to the main program **quick\_digitize.py**

**spline.py**

**Class Name:**

Spline(QgsMapTool):

**Methods:**

\* **\_\_init\_\_(self, iface)**: Initiates the plugin with the variables.

\* **canvasPressEvent(self,event):**Gets the point from QGIS and marks it on the map.

\* **resetPoints(self):**Empties the point array.

\* **createFeature(self):**Makes the feature and add geometry to the feature and enables on the fly reprojection if not already enabled.

\* **canvasMoveEvent(self,event):**Whenever the cursor is moved it get the point and draws the outline of the feature on the map.

\* **activate(self):**Is called from the splinetool.py to activate the spline plugin.

\* **resetRubberBand(self):** The resets the red outline.

\* **setRubberBandPoints(self,points):**Draws the outline.

\* **deactivate(self):**Called in splinetool.py to deactivate the plugin.

\* **interpolate(self, points):**Given the tightness and tolerance calls hermite() to interpolate the position of the points.

\* **hermite(self, points, tolerance, tightness):**Uses Bezier curves to calculate the points in the curve.

\* **simplifyPoints( self, points, tolerance):**Uses the tolerance to reduce the number of nodes in the curve.

\* **pointScalar( self, p, k):**Multiplies a point P1(x,y) with a variable k to give P(kx,ky).

\* **pointsAdd( self, p1, p2):**Adds two points P1(x1,y1) and P2(x2,y2) to give P(x1+x2,y1+y2).

\* **pointsTangentScaled(self, p1, p2, k ):**Gets the slope of the line joining two points P1(x1,y1) and P2(x2,y2).

\* **pointsDist(self, a, b):**Gets the distance between two points P1(x1,y1) and P2(x2,y2).

**splinetool.py**

**Class Name:**

SplineTool()

**Methods:**

\* **\_\_init\_\_(self, iface, toolBar):**Creates a button on the toolbar and links its behaviour to appropriate actions.

\* **digitize(self):**Activates the spline logic.

\* **deactivate(self):**Stops the working of the spline tool.

\* **toggle(self):**Is used to make a button non selectable if the layer is not in edit mode.

\* **settingsChanged(self):**Refreshes the tightness and tolerance value if changed in the settings dialog box and applied.

**settingsdialog.py**

**Class Name:**

SettingsDialog( QDialog, Ui\_SettingsDialog ):

**Methods:**

\* **\_\_init\_\_( self, parent = None ):**Initiates the logic and connects the signals and the slots.

\* **ok(self):**applies the settings and closes.

\* **apply(self):**Changes the values of tolerance and tightness.

\* **cancel(self):** closes the dialog box.

\* **defaults(self):**Sets the values to the default value.

**ui\_settingsdialog.py**

**Class Name:**

Ui\_SettingsDialog(object):

**Methods:**

**setupUi(self, Dialog)**: Adds objects with specified parameters to dialog box

**retranslateUi(self, Dialog)**: Renames the labels.

**utils.py**

Contains the default Tolerance and tightness value.

**ROTATE OBJECT TOOL**

**About the tool:**

The tool creates a new rotated polygon in the same layer. It however, does not rotate the selected polygon. Use it when you need to keep the original polygon as well as require a new rotated polygon.

**Note:**

* **Works only in WGS 84 CRS , so change the CRS before using this button else it won’t work.**
* If there is no requirement of a new polygon and just the selected polygon needs to be rotated, use the Rotate tool in Advanced Digitizing Toolbar.

**How to use:**

* Load Quick Digitize plugin.
* Select Polygon layer in which feature is to be rotated and added.
* Enable Toggle Edit.
* Click on the Select an Object and a Vertex button.
* Select the polygon to be rotated and the vertex about which it is to be rotated.
* Click on the Rotate Object button.
* A dialog box will pop up, values ranging from -360 to 360 can be entered in the “Rotation Angle” box.
* Click OK and then close the box with the “x” button.

**How it works:**

The entire working of the tool is contained in five Python files:

**rotateobjecttool.py:** Takes user input for the rotation value and adds the rotated feature to given layer.

**rotateobjectgui.py:** Links the UI components to the working logic of the dialog box.

**ui\_rotateobject.py:** Contains the code for the UI of the dialog box

**vertexandobjectfindertool.py:** Contains the code for selecting the object and vertex.

**utils.py:** Has the rotate, distance and addtoLayer functions required.

Finally, they are imported to the main program **quick\_digitize.py**

**rotateobjecttool.py**

**Class Name:**

RotateObjectTool

**Methods:**

**\_\_init\_\_(self, iface, toolBar):** Creates a button on the toolbar and links its behaviour to appropriate actions.

**toggle\_1(self):** Is used to make a button non selectable if the layer is not in edit mode.

**selectvertexandobject(self):**Calls the selectvertexandobject function.

**storeVertexAndObject(self, result):**Stores the point and the object in assigned array.

**showDialog(self):**Creates a warning if not enough arguments are given or shows continues to show the dialog box if all the conditions are met.

**rotateObject(self, angle):**Callsthe rotate function in the utils.

**unsetTool(self):**Emits an Un set tool signal.

**deactivate(self):**Releases the point and unchecks the plugin button.

**rotateobjectgui.py**

**Class Name:**

RotateObjectGui(QDialog, Ui\_RotateObject)

**Methods:**

**\_\_init\_\_(self, parent, flags):** Initiates the logic and connects the signals and the slots.

**initGui(self):**Creates the instances.

**accept(self):**Getsthe value in the dialog box when ok is clicked.

**close(self):**Emits an Unset signal.

**ui\_rotateobject.py:**

**Class Name:**

Ui\_RotateObject(object)

**Methods:**

**setupUi(self, Dialog)**: Adds objects with specified parameters to dialog box

**retranslateUi(self, Dialog)**: Renames the labels.

**vertexandobjectfindertool.py**

**Class Name:**

VertexAndObjectFinderTool(QgsMapTool)

**Methods:**

**\_\_init\_\_(self,canvas):** Creates a button on the toolbar and links its behaviour to appropriate actions and a custom cursor is set.

**canvasPressEvent(self, event):**First gets the selected object from the canvas and then the point which is also the centre of rotation.

**createRubberBand( self, isPolygon ):**Creates an outline around and over the object and the point.

**showSettingsWarning(self):**Displays an warning if the vertex could not snap due to lower snapping properties of the project.

**activate(self):**Activates the custom cursor.

**deactivate(self):**Releases the stored polygon and removes the outline.

**utils.py**

**Class Name:**

-

**Methods:**

**rotate(geom, point, angle):** Given the geometry (polygon, line, point), a point(centre of rotation) and an angle, it gives a rotated feature of the same geometry.