Raster Vector Data Analysis in DotSpatial

**Purpose of this tutorial**

Become familiar with the DotSpatial class library and perform the following functions in Visual C# environment.

* Load different Raster data formats. (Accessing the GDAL data provider)
* Perform shape file operations: create a shape file, add attributes, and save them.
* Analyze Raster and Vector data. Based on the Digital Elevation Model (DEM) and polyline shape file, the user can view the elevation distance profile.

## What is a DotSpatial class library?

## DotSpatial aims to provide a free, open source, consistent and dependable set of libraries for the .NET, Silverlight and Mono platforms, enabling developers to easily incorporate spatial data, analysis, and mapping into their applications thereby unleashing the massive potential of GIS in solutions for organizations and communities of all types in a nonrestrictive way. It is being developed by members of the MapWindow geographic information system (GIS) open source team, and the broader OSGeo .NET developer community.

## This tutorial utilizes the following four major steps:

* **Step 1**: Configure the DotSpatial class library in Visual C#
* **Step 2**: Configure the GDAL access in the application
* **Step 3**: Graphical User Interface design
* **Step 4**: Code implementation.

## Step 1: Configure the DotSpatial class library in Visual C#

* Download the DotSpatial class library from the following website. http:// www. DotSpatial.org
* Start a new Windows Forms application project in visual studio.
* Add the following DotSpatial references in the development environment: DotSpatial.Controls.dll, DotSpatial.Data.dll, DotSpatial.Data.Forms.dll, DotSpatial.Projections.dll, DotSpatial.Symbology.dll, DotSpatial.Topology.dll
* Change the target framework in C# development environment, from .Net Framework 4 Client Profile to .Net Framework 4. This property should be under the properties of the application.
* Add the DotSpatial control into the Visual Studio Toolbox. Create a new tab with the name “DotSpatial” on the toolbox and add the DotSpatial controls on it. To add the DotSpatial controls, right click on the DotSpatial tab and select the “choose items” on the context menu; then select the **DotSpatial.Controls.dll** from the downloaded folder.

## Step 2: Configure the Geospatial Data Abstraction Library (GDAL) access in the application

* Copy the Data Extensions folder from the downloaded folder to the current project’s Bin\ Debug folder. The .dlls from this folder is necessary for GDAL data provider access. GDAL is a translator library for raster geospatial data formats and it’s released by the Open Source Geospatial Foundation.



Fig. 1 Data Extensions folder from DotSpatial downloaded unzip folder

## Step 3: Graphical User Interface design.

* Design the main user interface as follows:



Fig.2. Main user interface

Interface design considerations.

* Add three panel controls. Panel control's properties should be as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Properties | Panel 1 | Panel 2 | Panel 3 |
| Name  | pnlOperations | pnlLegend | pnlMap |
| Dock | Top | Left | Fill |

Table.1. Panel controls properties.

Optional step: Panel 1 background color: ActiveCaption

* Add three buttons. Button properties should be as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Properties | Button 1 | Button 2 | Button 3 |
| Name | btnLoadDEM | btnDrawPath | btnViewElevation |
| Text | &Load DEM | &Draw Hiking Path | &View Elevation |

Table.2. Button controls properties.

* Add a label control and set its property as follows: Name: lbltitle Text: Hiking Path Finder.
* Drag a “Legend” control from the DotSpatial tab under toolbox and drop it on the pnlLegend. Legend properties should be set as follows: Name: Legend1 Dock: Fill
* Drag a “Map” control from the DotSpatial tab under toolbox and drop it on pnlMap. Map properties should be set as follows: Name: Map1, Dock: Fill, Legend: Legend1, Back color: Silver, ProjectionModeDefine: Never, ProjectionModeReproject: Never
* Drag an "AppManager" control from DotSpatial tab under toolbox and drop it on the form.

Note: This control is necessary for loading different formats of raster data.

* Set the properties of AppManager1 as follows: Map: Map1 Legend: Legend1 Name: appManager1
* Design the Graph user interface: Add a second windows form and design it as follows: Fig. 3 shows the graph form. Set the properties of the form as follows: Name: frmGraph, Text: Elevation-Distance



Fig.3. frmGraph form

* Download the ZedGraph.dll from the class form website or from the following URL: <http://sourceforge.net/projects/zedgraph/files/>
* Add a reference called the ZedGraph.dll from the downloaded folder.
* Add a panel control and set its properties as follows: name : pnlGraph Dock= fill
* Add the Zedgraph control to toolbox: Create a new tab on the toolbox and add the Zedgraph control on it. To add the Zedgraph control, right click on the zedgraph tab and select “chooseitems” on the context menu then select ZedGraph.dll from the downloaded folder.
* Drag the ZedGraph control from the toolbox and drop into the pnlGraph control. Set its properties as follows: Dock :fill

## Step 4: Implementation of code.

First, load a raster file on the map control; then on the map control, based on any drawn polyline, the elevation distance profile on the graph control is visualized. Thus, in the code implementation the following sequence must be maintained.

1. Load raster data.
2. Draw a polyline/ hiking path.
3. Obtain the elevation of the DEM along the hiking path (polyline).
4. Draw a graph based on the calculated elevation along the distance.

Import the following namespaces on the main form. Via the namespaces it is possible to access the different classes and their methods from DotSpatial class libraries.

*using DotSpatial.Controls;*

*using DotSpatial.Data;*

*using DotSpatial.Symbology;*

*using DotSpatial.Topology;*

* + **Load Raster data**: DotSpatial map control has a method called addRasterLayer, which is used to load the different formats of raster files on the map control. The following code on the LoadDEM button click event is used to load any raster files. ZoomToMaxExtent method is used to zoom the shape file on the map control to its maximum extents.

*Private void btnLoadDEM\_Click(object sender, EventArgs e)*

 *{*

 *map1.AddRasterLayer();*

 *map1.ZoomToMaxExtent();*

 *}*

* + **Draw a polyline/ hiking path**: Hiking path is drawn as a polyline on the map control. You can add different points on the polyline by performing a left mouse click on the map control in different locations. The following flowchart shows the procedure of drawing a polyline.

****

Fig.4. Flow chart for creating a polyline

Start drawing a path in the following manner. First, click on the draw path button. Now, start drawing a polyline using a left mouse clicking on the map control.

Create a polyline by left clicking more than once on the map control in different locations and finally a right mouse click to finish the polyline.

***Initialize a line layer:***The first step in creating a polyline is the creation of an instance of the line FeatureSet and an initialization of a line layer. The following code is used to initialize the line layer and instantiate the line feature***.***

*FeatureSet lineF=new FeatureSet(FeatureType.Line);* //*Instantiate the line feature*

*lineF.Projection = map1.Projection; //Set the projection to the line feature as same as map projection.*

*lineLayer = (MapLineLayer lineLayer) map1. Layers.add(lineF)* ***;*** *//Initialize the line layer*

***Mouse click:***After the initialization of the polyline, left mouse click event is used to add the map coordinates to the polyline. The following code is used to get the map coordinates of the map based on the mouse click location.

*Coordinate coord = map1.PixelToProj(e.Location);*

***Add point to line layer****:* After the mouse coordinates are obtained, the following code is used to add the map mouse coordinates into the line feature.

*lineFeature.Coordinates.Add(coord);*

LineGeometry is an instance of the line string, which is used to store the list of line coordinates. Assign the line feature to the IFeature to handle the attribute column values of the polyline shape file.

 *Dim lineFeature As IFeature = lineF.AddFeature(lineGeometry)*

Create an attribute column called ID and store numeric data in the attribute column.

 *lineFeature.DataRow("ID") = lineID // Here lineID is an inter variable.*

***Left mouse click:***Left mouse click to collect the mouse click map coordinates and add them to the polyline feature.

***Right mouse click:*** A right mouse click event is used to refresh the map and save the polyline in the hard disk.

*map1.ResetBuffer();*

*lineF.SaveAs("c:\\MW\\linepath.shp", true);*

**Complete code for creating a hiking path**

Declare the following class level variables.

#region "Class level variables"

 //the line layer

 MapLineLayer lineLayer;

 //the line feature set

 FeatureSet lineF;

 int lineID= 0;

 //boolean variable for first time mouse click

 Boolean firstClick=false;

 //boolean variable for ski path drawing finished

 Boolean hikingpathPathFinished = false;

#endregion

private void btnDrawPath\_Click(object sender, EventArgs e)

 {

 //remove any existing path

 foreach (IMapLineLayer existingPath in map1.GetLineLayers())

 {

 map1.Layers.Remove(existingPath);

 }

 //hikingpath is not finished

 hikingpathPathFinished = false;

 //change the mouse cursor

 map1.Cursor = Cursors.Cross;

//initialize the polyline featureset

 LineF= new FeatureSet(FeatureType.Line);

 //set projection

 lineF.Projection = map1.Projection;

 //initialize the featureSet attribute table

 DataColumn column = new DataColumn("ID");

 lineF.DataTable.Columns.Add(column);

 //add the featureSet as map layer

 lineLayer = (MapLineLayer)map1.Layers.Add(lineF);

 //implement the symbology

 LineSymbolizer symbol = new LineSymbolizer(Color.Blue, 2);

 lineLayer.Symbolizer = symbol;

 //Set the legend text

 lineLayer.LegendText = "Hiking path";

 //Before clicking the mouse first time

 firstClick = true;

 }

private void map1\_MouseDown(object sender, MouseEventArgs e)

 {

 //if hiking path is fininshed, don't draw any line

 if (hikingpathPathFinished == true)

 return;

 if (e.Button == MouseButtons.Left)

 {

 //left click - fill array of coordinates

 //coordinate of clicked point

 Coordinate coord = map1.PixelToProj(e.Location);

 //first time left click - create empty line feature

 if (firstClick)

 {

 //Create a new List called lineArray.

 //This list will store the Coordinates

 //We are going to store the mouse click coordinates into this array.

 List<Coordinate> lineArray = new List<Coordinate>();

 //Create an instance for LineString class.

 //We need to pass collection of list coordinates

 LineString lineGeometry = new LineString(lineArray);

 //Add the linegeometry to line feature

 IFeature lineFeature = lineF.AddFeature(lineGeometry);

 //add first coordinate to the line feature

 lineFeature.Coordinates.Add(coord);

 //set the line feature attribute

 lineID = lineID + 1;

 lineFeature.DataRow["ID"] = lineID;

 firstClick = false;

 }

 else

 {

 //second or more clicks - add points to the existing feature

 IFeature existingFeature = lineF.Features[lineF.Features.Count - 1];

 existingFeature.Coordinates.Add(coord);

 //refresh the map if line has 2 or more points

 if (existingFeature.Coordinates.Count >= 2)

 {

 lineF.InitializeVertices();

 map1.ResetBuffer();

 }

 }

 }

 Else if (e.Button == MouseButtons.Right)

 {

 //right click - reset first mouse click

 firstClick = true;

 map1.ResetBuffer();

 lineF.SaveAs("c:\\MW\\linepath.shp", true);

 MessageBox.Show("The line shapefile has been saved.");

 map1.Cursor = Cursors.Arrow;

 //the ski path is finished

 hikingpathPathFinished = true;

 }

 }

* + **Get the elevation of the DEM along the polyline:** To calculate the elevation along the polyline, various line segment coordinates will be collected and those divided by a constant number.

The following code is used to access the polyline layer from the map control, obtain the coordinates from the line segments and access the raster layer from the map control.

 *IMapLineLayer pathLayer = default(IMapLineLayer); //declare the maplinelayer*

 *pathLayer = map1.GetLineLayers()[0]; //get the polyline layer from the map control*

*IFeatureSet featureSet = pathLayer.DataSet; //get the data set of the polyline*

*IList<Coordinate> coordinateList = featureSet.Features[0].Coordinates; //get the polyline’s line segment coordinates from the data set.*

Accessing the raster layer from the map control

 *IMapRasterLayer rasterLayer = default(IMapRasterLayer); //Declare an ImapRasterLayer*

 *rasterLayer = map1.GetRasterLayers()[0]; //Get the first raster layer.*

The following flowchart shows how to obtain the elevation based on the line segment’s coordinate and the raster layer.



Fig.5. Extract elevation procedure

***Get map coordinates****:* First, obtain the various line segments of the polyline. Second, from each line segment, obtain their coordinates and convert them to map coordinates.

 *IList<Coordinate> coordinateList = featureSet.Features[0].Coordinates; //Get the coordinates of the polyline’s point feature.*

 *Coordinate startCoord = coordinateList[i]; //get the start point coordinate from the polyline’s line segment*

 *Coordinate endCoord = coordinateList[i + 1]; //get the end point coordinate from the polyline’s line segment*

 *Coordinate coordinate = new Coordinate(curX, curY); //Get the map coordinate based on the map’s x, y coordinate*

***Calculate raster cell coordinates:***Based on each point’s map coordinate, the coordinate in each raster cell is calculated as the following manner:

*RcIndex rowColumn = raster.DataSet.Bounds.ProjToCell(coordinate); //Get the raster column index based on the coordinate*

***Extract the elevation from the raster cell***.Based on the raster’s row and column value, the following code is used to get the elevation value from the raster data layer.

 *curElevation = raster.DataSet.Value[rowColumn.Row, rowColumn.Column];*

***Calculate distance from the following points:*** Based on the following points X, Y values, the Pythagorean Theorem is used to find the distance.

Steps for calculating the total distance: Given 2 points (PSTART, PEND) and a step ∆.

1. Calculate the Distance (PSTART, PEND)

PSTART = X Start, Y Start

PEND= X End, Y End

1. Calculate the constant Xdiff, Ydiff

Constant Xdiff = ( [Xend – Xstart] / N )

Constant Ydiff = ( [Yend – Ystart] / N )

N is the number of steps.

1. X i = X (i-1) + Constant Xdiff
2. Y i = Y (i-1) + Constant Ydiff
3. Distance di = (X i 2 + Y i 2 ) ^ (½)
4. Add the distance di to the distance array as follows:

di =+ di

**Complete code for extracting elevation of a raster layer along polyline shape file**

private void btnViewElevation\_Click(object sender, EventArgs e)

 {

 try

 {

 //extract the complete elevation

 //get the raster layer

 IMapRasterLayer rasterLayer = default(IMapRasterLayer);

 if (map1.GetRasterLayers().Count() == 0)

 {

 MessageBox.Show("Please load the DEM");

 return;

 }

 //use the first raster layer in the map

 rasterLayer = map1.GetRasterLayers()[0];

 //get the hiking path line layer

 IMapLineLayer pathLayer = default(IMapLineLayer);

 if (map1.GetLineLayers().Count() == 0)

 {

 MessageBox.Show("Please add the hiking path");

 return;

 }

 pathLayer = map1.GetLineLayers()[0];

 IFeatureSet featureSet = pathLayer.DataSet;

 //get the coordinates of the hiking path. this is the first feature of

 //the feature set.

 IList<Coordinate> coordinateList = featureSet.Features[0].Coordinates;

 //get elevation of all segments of the path

 List<PathPoint> fullPathList = new List<PathPoint>();

 for (int i = 0; i < coordinateList.Count - 1; i++)

 {

 //for each line segment

 Coordinate startCoord = coordinateList[i];

 Coordinate endCoord = coordinateList[i + 1];

 List<PathPoint> segmentPointList = ExtractElevation(startCoord.X, startCoord.Y, endCoord.X, endCoord.Y, rasterLayer);

 //add list of points from this line segment to the complete list

 fullPathList.AddRange(segmentPointList);

 }

 //calculate the distance

 double distanceFromStart = 0;

 for (int i = 1; i <= fullPathList.Count - 1; i++)

 {

 //distance between two neighbouring points

 double x1 = fullPathList[i - 1].X;

 double y1 = fullPathList[i - 1].Y;

 double x2 = fullPathList[i].X;

 double y2 = fullPathList[i].Y;

 double distance12 = Math.Sqrt(((x2 - x1) \* (x2 - x1)) + ((y2 - y1) \* (y2 - y1)));

 distanceFromStart += distance12;

 fullPathList[i].Distance = distanceFromStart;

 }

 frmGraph graphForm = new frmGraph(fullPathList);

 graphForm.Show();

 }

 catch (Exception ex)

 {

 MessageBox.Show("Error calculating elevation. the whole path should be inside the DEM area" + ex.Message);

 }

 }

Create a Path Point class below the frmTutorial class as follows. This class is used to store the collection of elevation and distance.

 public class PathPoint

 {

 public Double X;

 public Double Y;

 public Double Distance;

 public Double Elevation;

 }

Create an ExtractElevation function as follows: This function is used to get the elevation from the DEM along with the line segment.

 /// <summary>

 /// This function is used to get the elevation.

 /// Based on the given line segment's start and endpoint, 100 points will be divided and based on the points elevation will be claculated.

 /// </summary>

 /// <param name="startX">Line segement's start X point</param>

 /// <param name="startY">Line segement's start Y point</param>

 /// <param name="endX">Line segement's end X point</param>

 /// <param name="endY">Line segement's end Y point</param>

 /// <param name="raster">Raster DEM</param>

 /// <returns>List of elevation</returns>

 /// <remarks></remarks>

 public List<PathPoint> ExtractElevation(double startX, double startY, double endX, double endY, IMapRasterLayer raster)

 {

 double curX = startX;

 double curY = startY;

 double curElevation = 0;

 List<PathPoint> pathPointList = new List<PathPoint>();

 int numberofpoints = 100;

 double constXdif = ((endX - startX) / numberofpoints);

 double constYdif = ((endY - startY) / numberofpoints);

 for (int i = 0; i <= numberofpoints; i++)

 {

 PathPoint newPathPoint = new PathPoint();

 if ((i == 0))

 {

 curX = startX;

 curY = startY;

 }

 else

 {

 curX = curX + constXdif;

 curY = curY + constYdif;

 }

 Coordinate coordinate = new Coordinate(curX, curY);

 RcIndex rowColumn = raster.DataSet.Bounds.ProjToCell(coordinate);

 curElevation = raster.DataSet.Value[rowColumn.Row, rowColumn.Column];

 //set the properties of new PathPoint

 newPathPoint.X = curX;

 newPathPoint.Y = curY;

 newPathPoint.Elevation = curElevation;

 pathPointList.Add(newPathPoint);

 }

 return pathPointList;

 }

* + Add the following code in the frmGraph.cs form. Based on the elevation and distance, Zed graph control is used to draw an elevation profile graph. Zed graph control is an open source graph control. The following line of code is used to add the X, Y, caption to the zed graph control.

 public frmGraph(List<PathPoint> pathList)

 {

 InitializeComponent();

 //populate the graph

 //create the distance and elevation arrays..

 double[] distanceArray = new double[pathList.Count];

 double[] elevationArray = new double[pathList.Count];

 for (int i = 0; i <= pathList.Count - 1; i++)

 {

 distanceArray[i] = pathList[i].Distance;

 elevationArray[i] = pathList[i].Elevation;

 }

 zedGraphControl1.GraphPane.CurveList.Clear();

ZedGraph.LineItem myCurve = zedGraphControl1.GraphPane.AddCurve("Elevation Profile", distanceArray, elevationArray, Color.Blue);

 myCurve.Line.Width = 2f;

 myCurve.Symbol.Type = ZedGraph.SymbolType.None;

 myCurve.Line.Fill.Color = Color.LightBlue;

 myCurve.Line.Fill.Color = Color.FromArgb(100, 0, 0, 255);

 myCurve.Line.Fill.IsVisible = true;

 zedGraphControl1.GraphPane.XAxis.Title.Text = "Distance (meters)";

 zedGraphControl1.GraphPane.YAxis.Title.Text = "Elevation (meters)";

 //refresh the graph

 zedGraphControl1.AxisChange();

 //set the graph title

 zedGraphControl1.GraphPane.Title.Text = "Hiking Path Graph";

 }

**Output Screen shots of the project**: Figure 6, 7 shows the elevation distance profile.



Fig.6. Main form – It has a DEM and a polyline



Fig.7. Elevation distance profile along the polyline