Raster Vector Data Analysis ~ Hiking Path Finder

**Tutorial (6)**

Purpose of this tutorial: Finding an elevation distance profile for any hiking path on any DEM.

This tutorial will not go into detail on how to add support for using additional formats through GDAL. If you need to know how to do this please see tutorial nine or the Developer Getting Started Guide.

**Step 1**: Download the DotSpatial class library

This step is similar to the Tutorial #1 step 1.

**Step 2:** Add the DotSpatial reference and change the compile option.

Add the following references.

DotSpatial.Data.Forms.dll, DotSpatial.Symbology.dll, DotSpatial.Controls.dll, DotSpatial.Projections.dll, DotSpatial.Data.dll, DotSpatial.Topology.dll

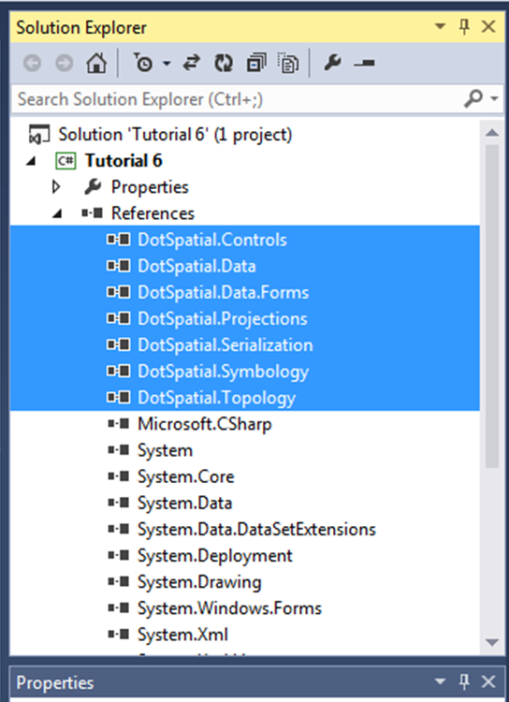


Figure : Required References

**Step 2:** Add the DotSpatial controls on the Visual Studio Toolbox window.

Create a new tab on the toolbox and add the DotSpatial controls on it, the same as in the first tutorial.*.*

**Step 3:** Design the GUI :

GUI has two different parts such as Main interface form and Graph interface form.

**Design the main interface form.**

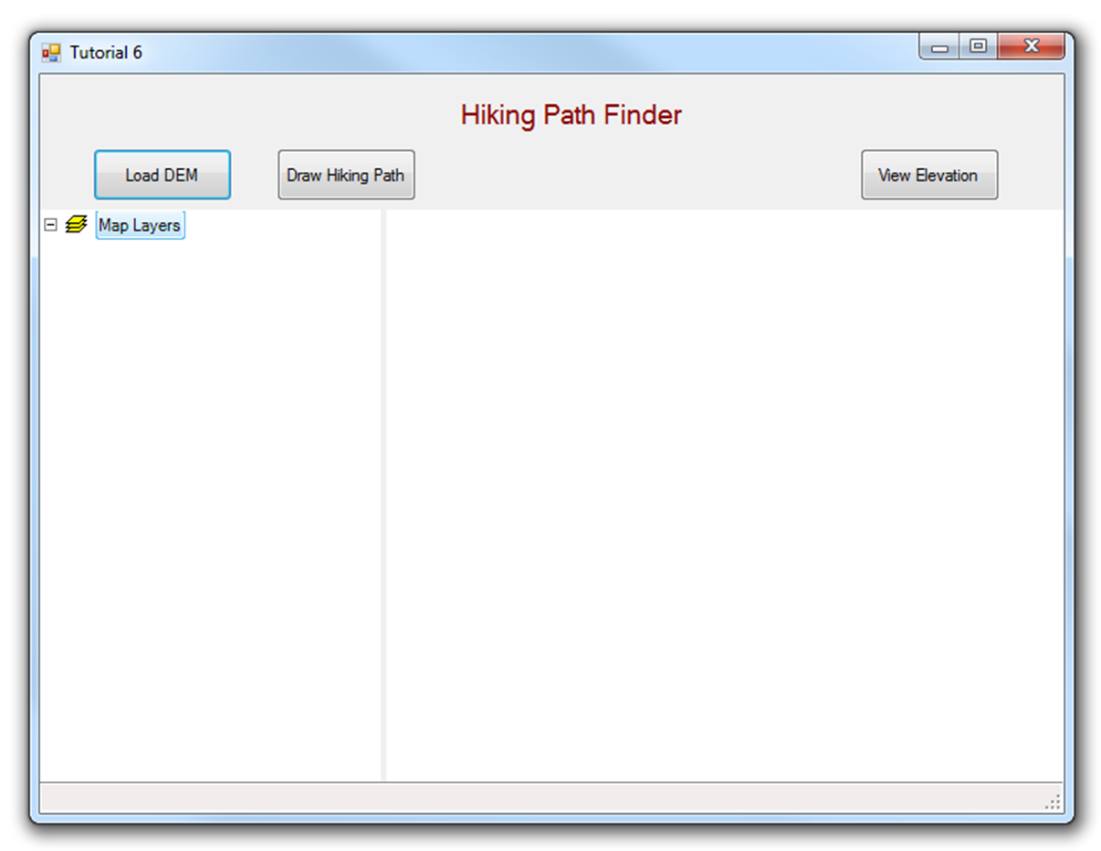


Figure : Final GUI for frmMain

Interface design considerations.

1. Add one panel control and a SpatialDockManger control. Their properties should be as follows:

|  |  |  |
| --- | --- | --- |
| Properties | Panel1 | spatialDockManager1 |
| Name | pnlOperations | sdmLegendMap |
| Dock | Top | Fill |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Properties | Button1 | Button2 | Button3 |
| Name | btnLoadDEM | btnDrawPath | btnViewElevation |
| Text | &Load DEM | &Draw Hiking Path | &View Elevation |

3. Add a label control and set its property as follows:

Name: lbltitle Text: Hiking Path Finder.

4. Drag a Legend control from the DotSpatial tab under toolbox and drop it on the left panel of the SpatialDockManger. Legend properties should be as follows:

Name: legend1 Dock: Fill

5. Drag a Map control from the DotSpatial tab under toolbox and drop it on the right panel of the SpatialDockManger. Map properties should be as follows:

Name: map1, Dock: Fill, Legend: legend1.

6. Set the properties of AppManager1 as follows:

Map: map1 Legend: legend1

**Design the graph interface form.**

1. Download the ZedGraph.dll from the class form website or from the following URL: <http://sourceforge.net/projects/zedgraph/files/>

2. From the tool box, right click on the DotSpatial tab and click browse and select the ZedGraph.dll from the downloaded folder.

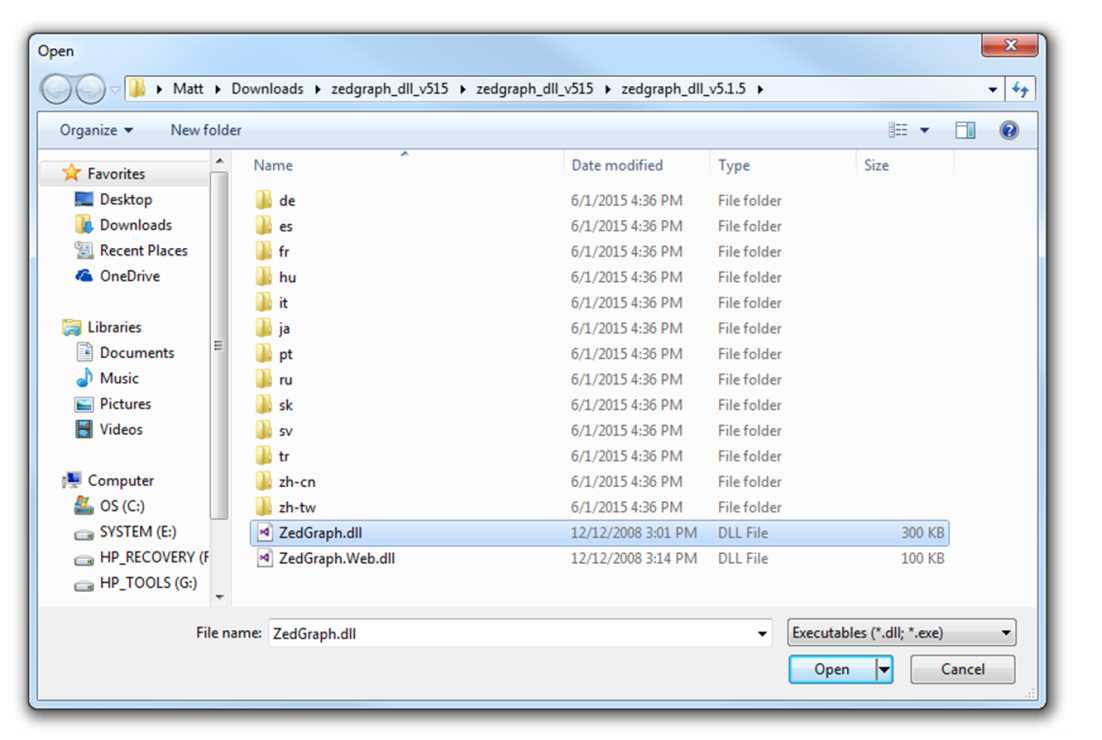


Figure : ZedGraph.dll

3. Add a second form and name it as frmGraph.

4. Add a panel control and set its properties as follows:

Name: pnlGraph Dock= fill

5. Drag the ZedGraph control from the toolbox and drop into the pnlGraph control. Set its properties as follows:

Dock: fill

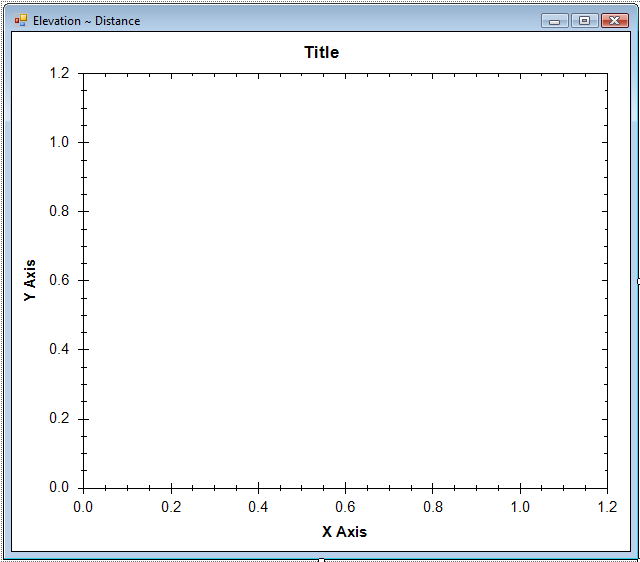


Figure 4: ZedGraph

**Step 5: Code implementation**

Following code is related to frmMain form.

Import the following namespaces:

using DotSpatial.Controls;

using DotSpatial.Data;

using DotSpatial.Symbology;

using DotSpatial.Topology;

Declare the following class level variables:

#region "Class level varibales"

//the line layer

MapLineLayer lineLayer = default(MapLineLayer);

//the line feature set

FeatureSet lineF = new FeatureSet(FeatureType.Line);

int lineID = 0;

//boolean variable for first time mouse click

bool firstClick = false;

//boolean variable for ski path drawing finished

bool hikingpathPathFinished = false;

#endregion

Create a PathPoint class as follows below the frmTutorial6 class:

public class PathPoint

{

public double X;

public double Y;

public double Distance;

public double Elevation;

}

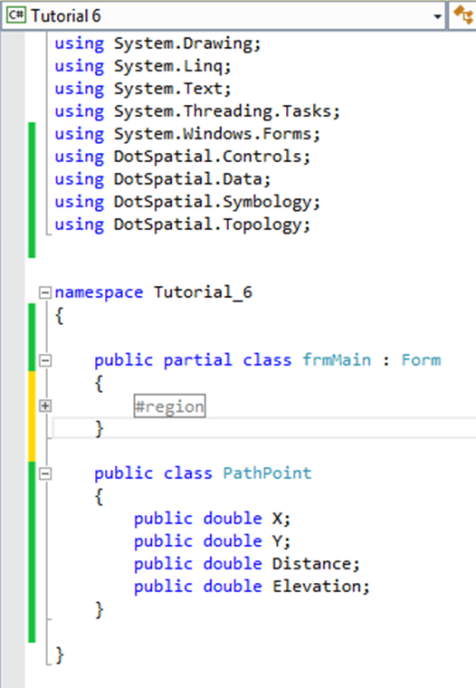


Figure : PathPoint Class

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 the elevation will be calculated.

/// </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 Load DEM button click event:

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

{

map1.AddLayer();

map1.ZoomToMaxExtent();

}

Add the following code in the Drawing hiking path button click event:

private void btnDrawPath\_Click(System.Object sender, System.EventArgs e)

{

//remove any existing path

foreach (IMapLineLayer existingPath in map1.GetLineLayers())

{

map1.Layers.Remove(existingPath);

}

lineF = new FeatureSet(FeatureType.Line);

//ski path is not finished

hikingpathPathFinished = false;

//initialize polyline feature set

map1.Cursor = Cursors.Cross;

//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);

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

lineLayer.Symbolizer = symbol;

lineLayer.LegendText = "Hiking path";

firstClick = true;

}

Add the following code in the Map1\_MouseDown event:

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

{

//if ski 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.

//In List we need not define the size and also

//Here 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

{

//right click - reset first mouse click

firstClick = true;

map1.ResetBuffer();

lineF.SaveAs("c:\\2009 Falls\\linepath.shp", true);

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

map1.Cursor = Cursors.Arrow;

//the ski path is finished

hikingpathPathFinished = true;

}

}

Add the following code in the frmGraph:

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";

}

Add the following code in the frmMain’s btnViewElevation\_Click event:

private void btnViewElevation\_Click(System.Object sender, System.EventArgs e)

{

try

{

//extract the complete elevation

//get the raster layer

IMapRasterLayer rasterLayer = default(IMapRasterLayer);

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

{

MessageBox.Show("Please add a raster layer");

return;

}

//use the first raster layer in the map

rasterLayer = map1.GetRasterLayers()[0];

//get the ski path line layer

IMapLineLayer pathLayer = default(IMapLineLayer);

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

{

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

return;

}

pathLayer = map1.GetLineLayers()[0];

IFeatureSet featureSet = pathLayer.DataSet;

//get the coordinates of the ski 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");

}

}

**Output screen shots**

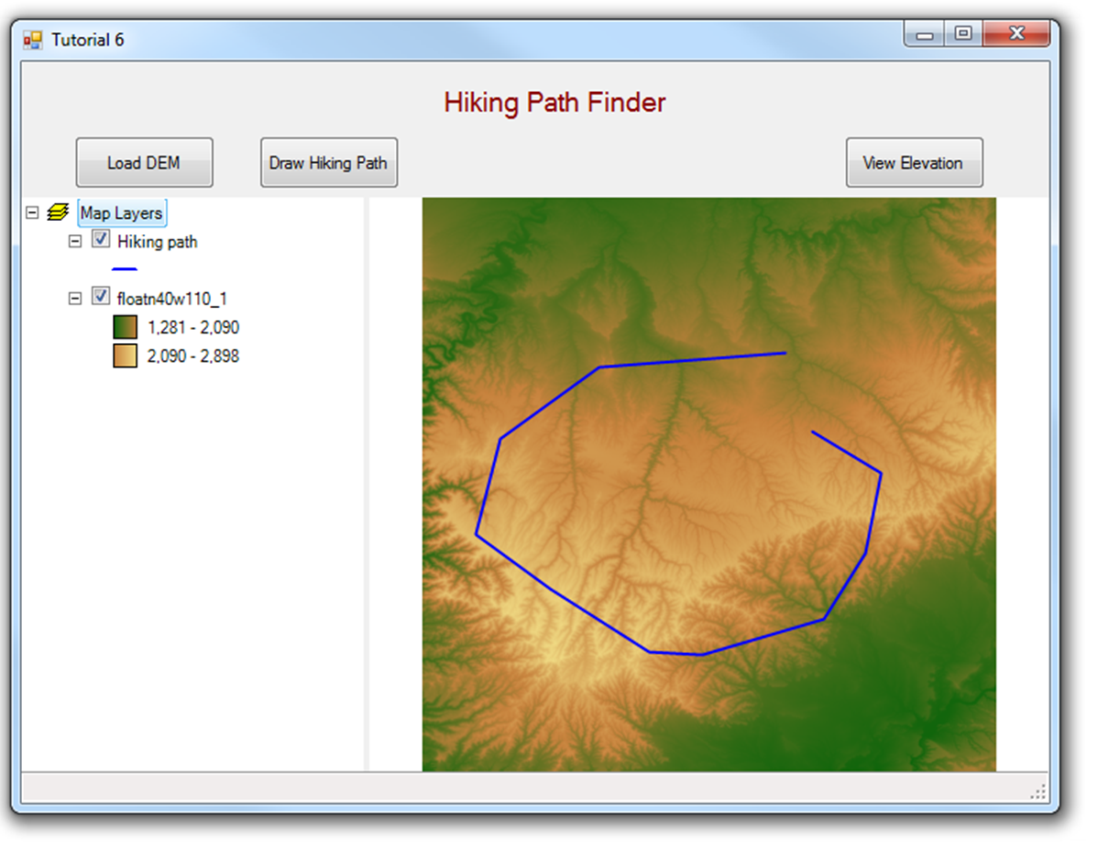


Figure : Hiking Trail

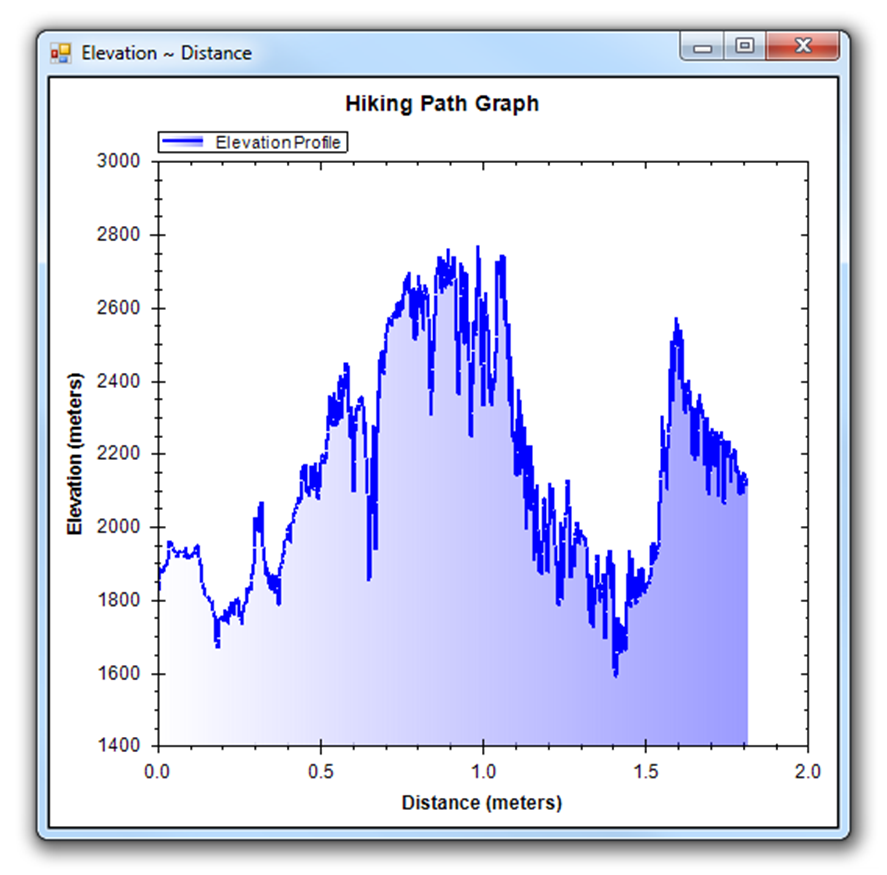


Figure : Hiking Elevation Profile