Final Project for GIS Architecture and Algorithms
Course Code AG2411

Implementation of *GIS Tangle1.0* by JAVA

ROYAL INSTITUTE OF TECHNOLOGY

2010/12/06

RUI ZHU

Geodesy and Geo-informatics

Royal Institute of Technology

Table of Contents

Tables and Figures

1. Introduction

In this final project, the author developed a piece of software named GIS Tangle1.0 to implement GIS algorithms for spatial analysis by JAVA language. There are four parts in this software.

Firstly, analysis functions for raster data are accomplished. It includes Local Sum, Focal Variety, Zone Minimum and 3 times 3 Neighbor Hood. Moreover, the author developed two functions of Slope and Aspect as an extension.

Secondly, the team tried to solve the shortest path function for vector analysis. But due to the limited time, we have not display the final results at present.

Thirdly, displaying function is created for all of the new created data in form of ASCII. Furthermore, the picture that is displayed in the software can be easily moved by mouse. Also, user can get the distance between two points by a ruler. And at the same time, clients can renew the picture to the original coordinate in case when the picture is out of the canvas region. Coordinates of the mouse are given accordingly when it is moving in the region of canvas.

Lastly, GIS Tangle1.0 also gives out the developer’s contract information. We created an interface of HELP document file for users. Technology, It can be accomplished as long as we have enough time.

Several classes are created and there is a main function to activate the software. But the structure for this software is not so perfect due to the knowledge limitation of software architecture.

2. Methodology

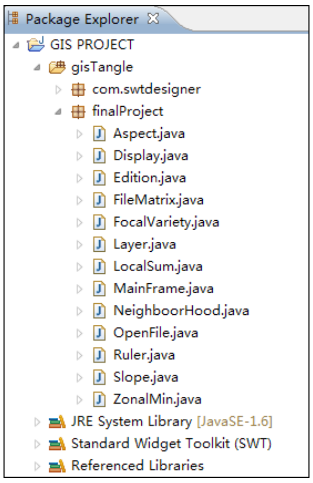
To let the final project proposal come true. Firstly, we installed plug-in of SWT which is perfect for GUI design. Secondly, we created Application Window and Shell as a father window and son windows respectively. Thirdly, we created each class individually by designing interface and creating coding in the background. Lastly, we combine them together locally.

To be more specifically, in this final project report, we give out the methodology of software architecture outline, aspect model, slope model and shortest path model respectively. While function models in the pervious exercise are ignored.

2.1 Software Architecture Outline

Table 1 Structures

Layer 1	Layer 2	Layer 3
	OpenFile Model	OpenFile
	Raster Model	Slope
		Aspect
MainFrame		Local Sum
		Focal Variety
Layer		Zonal Minimum
FileMatrix		3*3 Neighbor Hood
	Vector Model	Shortest Path
	Display Model	Display
		Ruler
	Edition Model	Edition



```
Package Explorer
GIS PROJECT
  gisTangle
    com.swtdesigner
    finalProject
      Aspect.java
      Display.java
      Edition.java
      FileMatrix.java
      FocalVariety.java
      Layer.java
      LocalSum.java
      MainFrame.java
      NeighboorHood.java
      OpenFile.java
      Ruler.java
      Slope.java
      ZonalMin.java
  JRE System Library [JavaSE-1.6]
  Standard Widget Toolkit (SWT)
  Referenced Libraries
```

Figure 1 classes in the project

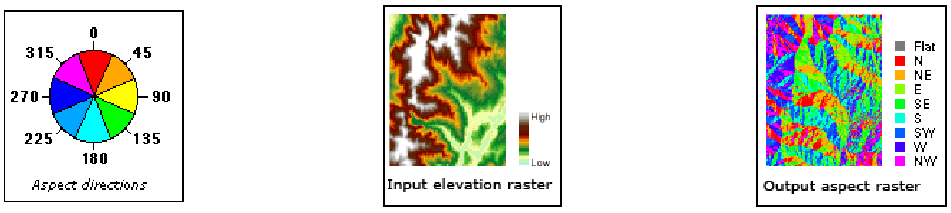
In Fiure1, it is can be seen clearly that MainFrame is used to be the father window and all of the rest classes are son window classes. Package of *com.swtdesigner* is included for background colors of the software. At the same time, *JRE System Library*, *SWT and Libraries* are included for software developing.

In Table 2, three layers of the structures are represented. The main class which used to be the father window is MainFrame while other classes are used as son windows. The classes of Layer and FileMatrix are non-visual classes, which can be used in every other class.

2.2 Aspect Model

Aspect identifies the down slope direction of the maximum rate of change in value from each cell to its neighbors. It can be thought of as the slope direction. The values of each cell in the output raster indicate the compass direction that the surface faces at that location. It is measured clockwise in degrees from 0 (due north) to 360 (again due north), coming full circle. Flat areas having no down slope direction are given a value of -1. The value of each cell in an aspect dataset indicates the direction the cell's slope faces.

Conceptually, the Aspect tool fits a plane to the z-values of a 3 x 3 cell neighborhood around the processing or center cell. The direction the plane faces is the aspect for the processing cell. The following diagram shows an input elevation dataset and the output aspect raster.



a Aspect directions b Input Elevation Raster c Output Aspect Raster

Figure 2 Aspect Algorithm

A moving 3*3 window visits each cell in the input raster, and for each cell in the center of the window, an aspect value is calculated using an algorithm that incorporates the values of the cell's eight neighbors. The cells are identified as letters a to i, with e representing the cell for which the aspect is being calculated.

a	b	c
d	e	f
g	h	i

The rate of change in the x direction for cell e is calculated with the following algorithm:

$$[dz/dx] = ((c + 2f + i) - (a + 2d + g)) / 8$$

The rate of change in the y direction for cell e is calculated with the following algorithm:

$$[dz/dy] = ((g + 2h + i) - (a + 2b + c)) / 8$$

Taking the rate of change in both the x and y direction for cell e, aspect is calculated using:

$$aspect = 57.29578 * \operatorname{atan2}([dz/dy], -[dz/dx])$$

The aspect value is then converted to compass direction values (0-360 degrees), according to the following rule:

```
if $aspect < 0$
    $cell = 90.0 - aspect$
else if $aspect > 90.0$
    $cell = 360.0 - aspect + 90.0$
else
    $cell = 90.0 - aspect$
```

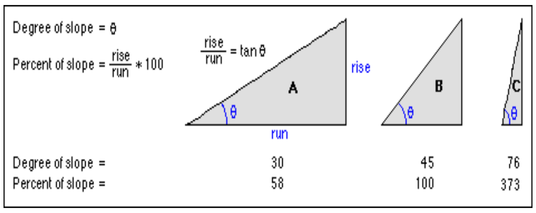
2.3 Slope Model

For each cell, the Slope Model calculates the maximum rate of change in value from that cell to its neighbors. Basically, the maximum change in elevation over the distance between the cell and its eight neighbors identifies the steepest downhill descent from the cell.

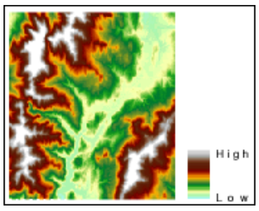
Conceptually, the tool fits a plane to the z-values of a 3 times 3 cell neighborhood around the processing or center cell. The slope value of this plane is calculated using the average maximum technique. The direction the plane faces is the aspect for the processing cell. The lower the slope value is, the flatter the terrain becomes; the higher the slope value is, the steeper the terrain becomes.

If there is a cell location in the neighborhood with a No-Data z-value, the z-value of the center cell will be assigned to the location. At the edge of the raster, at least three cells (outside the raster's extent) will contain No-Data as their z-values. These cells will be assigned the center cell's z-value. The result is a flattening of the 3 x 3 plane fitted to these edge cells, which usually leads to a reduction in the slope.

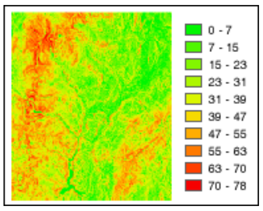
The output slope raster can be calculated in two types of units, degrees or percent (percent rise). The percent rise can be better understood if you consider it as the rise divided by the run, multiplied by 100. Consider triangle B below. When the angle is 45 degrees, the rise is equal to the run, and the percent rise is 100 percent. As the slope angle approaches vertical (90 degrees), as in triangle C, the percent rise begins to approach infinity.


Degree of slope = θ
Percent of slope = rise/run * 100
rise/run = tan θ
rise
run
A
B
C
θ
Degree of slope = 30 45 76
Percent of slope = 58 100 373

a Slope in Degrees Versus Percent



b Input Elevation Raster



c Output Slope Raster

Figure 3 Slope Algorithm

The rates of change (delta) of the surface in the horizontal (dz/dx) and vertical (dz/dy) directions from the center cell determine the slope. The value 57.29578 shown below is a truncated version of the result from 180/pi.The basic algorithm used to calculate the slope is:

slope_radians = ATAN (sqrt((dz/dx)^2 + (dz/dy)^2))

Slope is commonly measured in degrees, which uses the algorithm:

slope_degrees = ATAN (sqrt((dz/dx)^2 + (dz/dy)^2)) * 57.29578

The slope algorithm can also be interpreted as:

slope_degrees = ATAN (rise_run) * 57.29578

where:

rise_run = sqrt((dz/dx)^2 + (dz/dy)^2)

The values of the center cell and its eight neighbors determine the horizontal and vertical deltas. The neighbors are identified as letters from a to i, with e representing the cell for which the aspect is being calculated.

The rate of change in the x direction for cell e is calculated with the following algorithm:

[dz/dx] = ((c + 2f + i) - (a + 2d + g)) / (8 * x_cellsize)

The rate of change in the y direction for cell e is calculated with the following algorithm:

[dz/dy] = ((g + 2h + i) - (a + 2b + c)) / (8 * y_cellsize)

2.4 Display Technology

Two kinds of draw methods are used in the software and they are as follows, respectively:

```
for(int i = 0; i<fm.getNcols(); i++)
        for(int j = 0; j<fm.getNrows() ;j++)
        {
                setpixel = (int)(fm.getMatrix()[i][j]);
                float min = fm.getMin();
                float max = fm.getMax();
                int grayValue = (int)(255 * (setpixel) / (max - min));
                RGB cyanRGB = new RGB(grayValue, grayValue, grayValue);
                Color color = new Color(display, cyanRGB);
                arg0.gc.setBackground(color);
                arg0.gc.fillRectangle(j*res, i*res, res, res);
                arg0.gc.drawRectangle(j*res, i*res, res, res);
        }

for(int i = 0; i<fm.getNrows(); i++)
        for(int j = 0; j<fm.getNcols(); j++)
        {
                setpixel = (int)(fm.getMatrix()[i][j]);
                arg0.gc.setBackground(display.getSystemColor(setpixel));
                arg0.gc.fillRectangle((int)(MainFrame.this.xLeftTop+j*res),
(int)(MainFrame.this.yLeftTop+i*res), res, res);
                arg0.gc.drawRectangle((int)(MainFrame.this.xLeftTop+j*res),
(int)(MainFrame.this.yLeftTop+i*res), res, res);
        }
```

The distance of two points function is as follows:

```
public void showDis(float xs, float xe, float ys, float ye)
        {
                text.setText( Float.toString( (float) Math.sqrt( (xs-xe)*(xs-xe)+(ys-ye)*(ys-ye) ) ) );
        }
```

2.5 Shortest Path Model

In this project Dijkstra shortest path algorithm is used. And whole eclipse project package consists of 4 classes which are Distance, Graph, Path and Vertex. Vertex class includes the nodes. And nodes are city names in this project. Distance consists of the arcs and arc distances between two nodes (Vertex). Graph class covers the Dijkstra Shortest Path Algorithm engine. And finally, the last one is the main class. More specifically, in this project the Dijkstra shortest path algorithm was applied on finding the shortest path from Stockholm to Goteborg practically.

Table 2 Coordinates of Cities in Sweden

City Name	Longitude(°)	Latitude(°)
Stockholm	59.330	18.070
Vasteras	59.620	16.540
Eskillstuna	59.370	16.510
Nykoping	58.760	17.020
Orebro	59.280	15.220
Norrkoping	58.600	16.170
Trollhattan	58.290	12.300
Skovde	58.380	13.840
Jonkoping	57.780	14.170
Boras	57.730	12.940
Goteborg	57.720	12.010

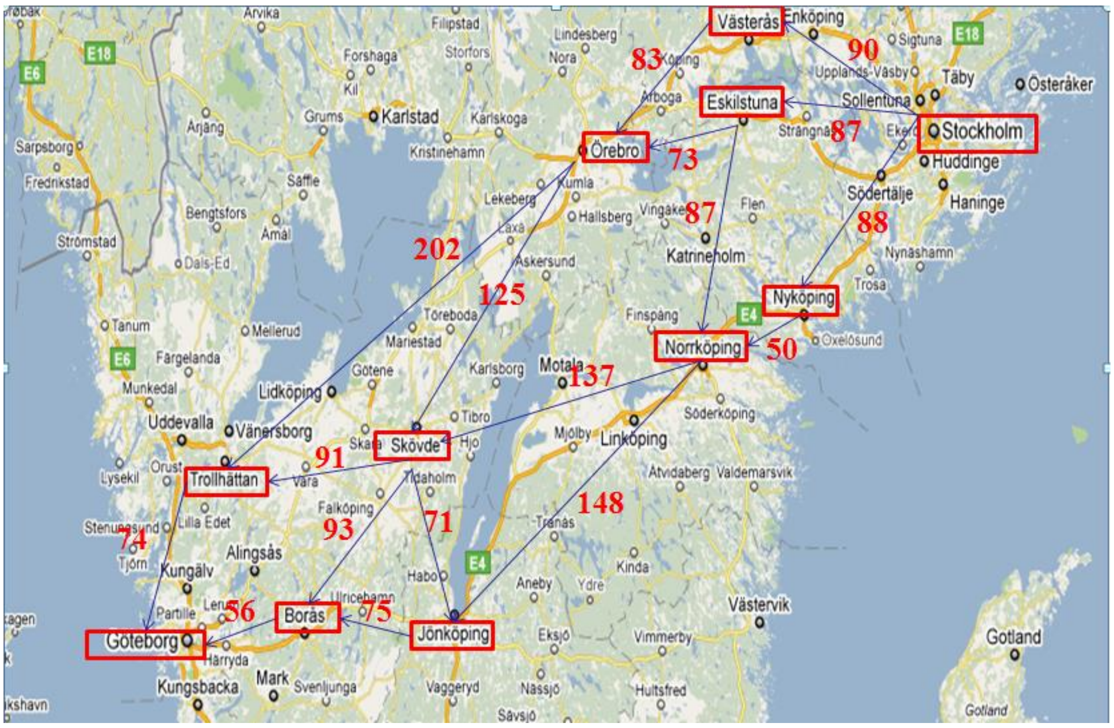


Figure 4 Highway Roads Map Between Stockholm and Goteborg

The picture comes from http://www.maps.google.com/

The figure-4, which is above the paragraph, shows the shortest path tree. And tree consists of 11 nodes (Cities) and 17 arcs and 17 arc distances. Dijkstra engine was created for one to all types. That means that it can calculate and find the shortest path for on destination to other target destinations.

Table 3 Vertex names and Node Numbers used in this java project

Vertex Name	Node Number
Stockholm	0
Vasteras	1
Eskillstuna	2
Nykoping	3
Orebro	4
Norrkoping	5
Trollhattan	6
Skovde	7
Jonkoping	8
Boras	9
Goteborg	10

Stockholm is the 0 number node. And it is the starting node and the all possible shortest paths will be calculated from Stockholm to other cities. Node (Vertex) Numbers are important in order to determine the final destination or target destination. Since the program was created according to node numbers.

Table 4 Vertexes, Edges and the Distances Between These Two Vertexes

Starting Vertex	Node Num	Target Node	Node Num	Distance(km)
Stockholm	0	Vasteras	1	90
Stockholm	0	Eskillstuna	2	87
Stockholm	0	Nykoping	3	88
Vasteras	1	Orebro	4	83
Eskillstuna	2	Orebro	4	73
Eskillstuna	2	Norrkoping	5	87
Nykoping	3	Norrkoping	5	50
Orebro	4	Trollhattan	6	202
Orebro	4	Skovde	7	125
Norrkoping	5	Skovde	7	137
Norrkoping	5	Jonkoping	8	148
Trollhattan	6	Goteborg	10	74
Skovde	7	Trollhattan	6	91
Skovde	7	Boras	9	93
Skovde	7	Jonkoping	8	71
Jonkoping	8	Boras	9	75
Boras	9	Goteborg	10	56

There are 14 possible ways from Stockholm to Goteborg. The program, which the author compiled in the eclipse, can find the possible shortest path correctly. The whole path is presented in the below figure. And all of the possible paths are shown as follows:

Table 5 Possible Paths From Stockholm to Goteborg According to the Given Data

All Possible Paths	km
Stockholm -Vasteras-Orebro-Trollhattan-Goteborg	449
Stockholm- Vasteras- Orebro-Skovde-Boras- Goteborg	447
Stockholm- Vasteras-Orebro-Skovde-Jonkoping-Boras- Goteborg	500
Stockholm- Eskillstuna-Orebro- Trollhattan-Goteborg	436
Stockholm- Eskillstuna – Orebro – Skovde - Boras- Goteborg	434
Stockholm- Eskillstuna -Orebro-Skovde-Jonkoping-Boras- Goteborg	487
Stockholm- Eskillstuna -Norrkoping-Skovde- Boras- Goteborg	460
Stockholm- Eskillstuna -Norrkoping-Skovde-Jonkoping-Boras- Goteborg	513
Stockholm- Eskillstuna - Norrkoping -Skovde- Trollhattan-Goteborg	476
Stockholm- Eskillstuna - Norrkoping - Jonkoping-Boras- Goteborg	453
Stockholm- Nykoping -Norrkoping-Skovde- Boras- Goteborg	424
Stockholm- Nykoping -Norrkoping-Skovde-Jonkoping-Boras- Goteborg	477
Stockholm - Nykoping - Norrkoping -Skovde- Trollhattan-Goteborg	440
Stockholm - Nykoping - Norrkoping -Jonkoping -Boras - Goteborg	417

The Dijkstra`s shortest path algorithm principle is simple and non-complex. The author assigns to every vertex a distance value. And these distance values to zero for our initial vertex and to infinity for all other remaining nodes.

Firstly we mark all vertexes as unvisited and initial vertex is current. For current vertex with all its unvisited neighbors, we calculate their temporary distance (from the initial vertexes to target vertexes). If these distance are less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance. When we become sure that all neighbors of the current vertexes were marked as visited. A visited vertex will not be checked ever again; its last recorded distance is final value and it is the smallest.

If all vertexes have been visited, algorithms can be finished. Otherwise, we should set the unvisited vertex with the smallest distance starting from the initial node as the next "current node" and continue again from step 3.

3. Results

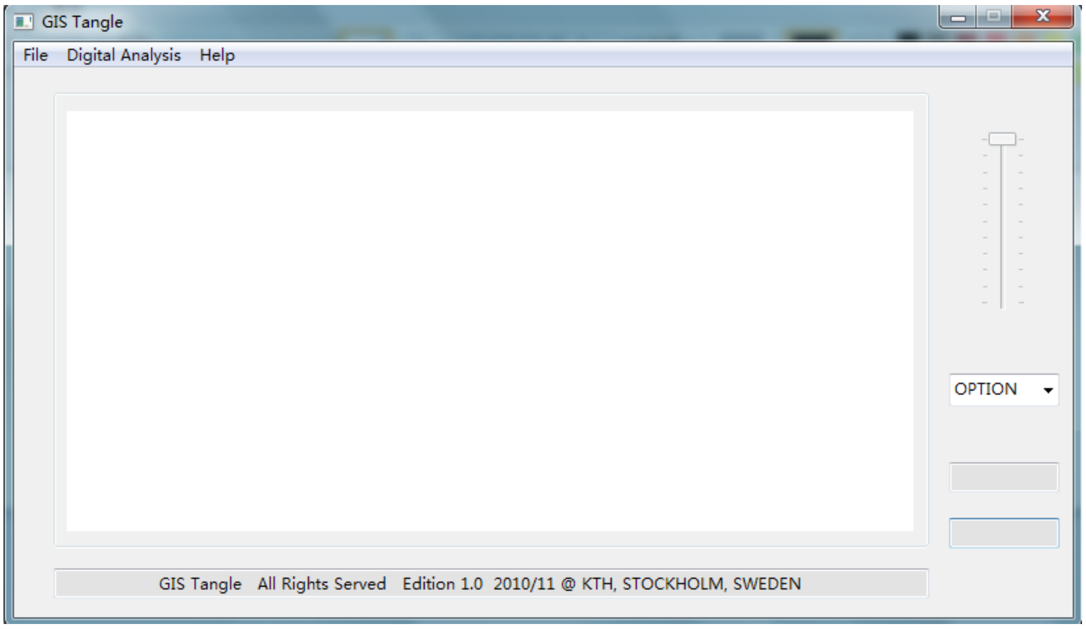

GIS Tangle
File
Digital Analysis
Help
OPTION
GIS Tangle All Rights Served Edition 1.0 2010/11 @ KTH, STOCKHOLM, SWEDEN

Figure 5 Interface of the MainFrame Class

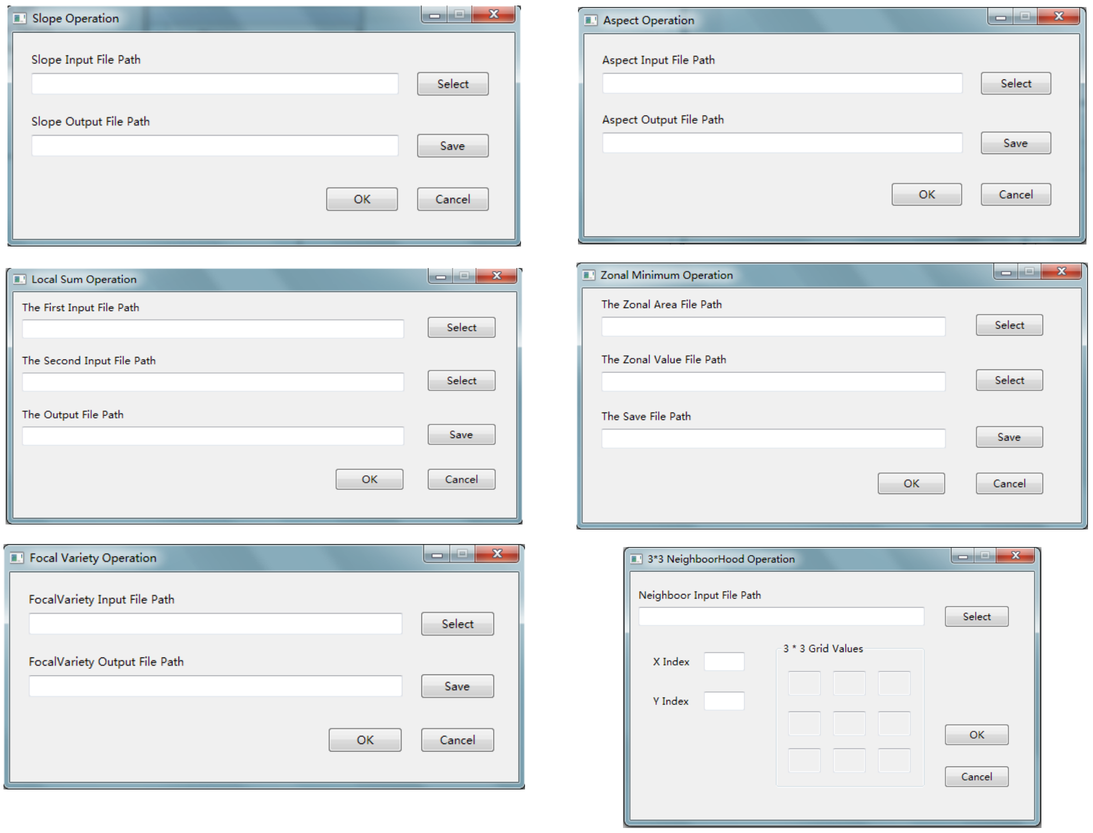
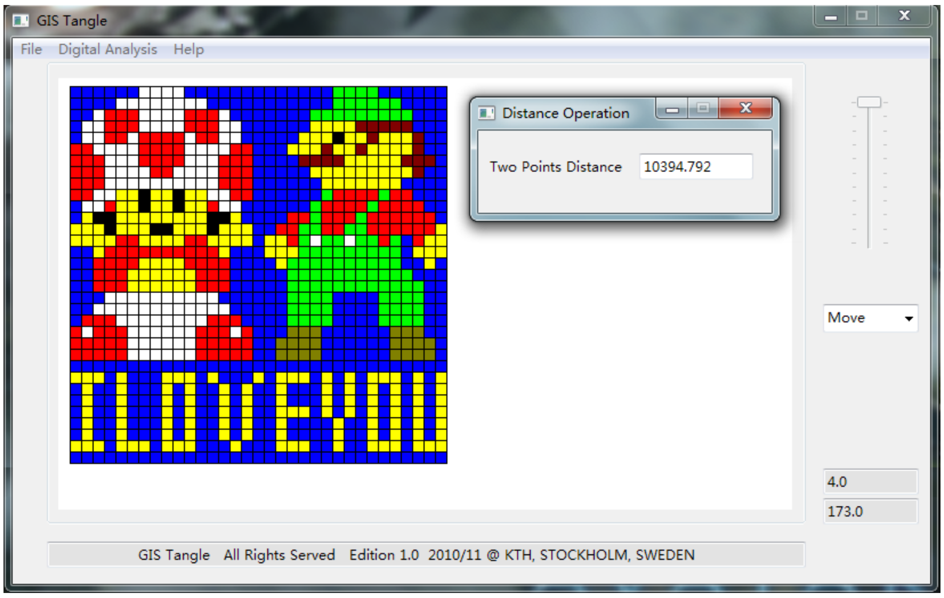


Figure 6 Functions Models in the Software



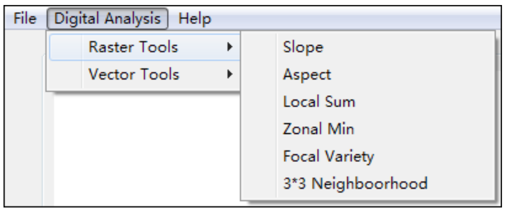
Two Points Distance 10394.792

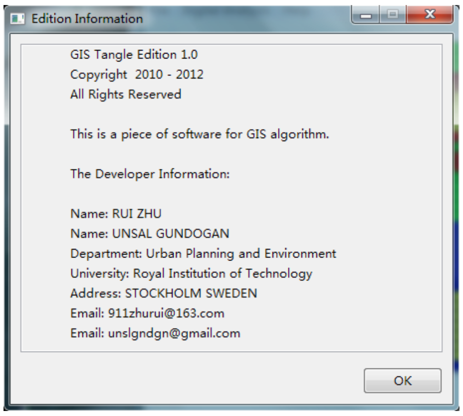
4.0

173.0

GIS Tangle All Rights Served Edition 1.0 2010/11 @ KTH, STOCKHOLM, SWEDEN

Figure 7 Picture Display and Distance Operation





GIS Tangle Edition 1.0
Copyright 2010 - 2012
All Rights Reserved

This is a piece of software for GIS algorithm.

The Developer Information:

Name: RUI ZHU
Name: UNSAL GUNDOGAN
Department: Urban Planning and Environment
University: Royal Institution of Technology
Address: STOCKHOLM SWEDEN
Email: 911zhurui@163.com
Email: unslgndgn@gmail.com

Figure 8 Models of Menu Bar and Edition Information

Stockholm

Stockholm	90	Vasteras
Stockholm	87	Eskillstuna
Stockholm	88	Nykoping
Eskillstuna	160	Orebro
Nykoping	138	Norrkoping
Orebro	362	Trollhattan
Norrkoping	275	Skovde
Norrkoping	286	Jonkoping
Jonkoping	361	Boras
Boras	417	Goteborg

```
System.out.println(theGraph.GetVListelement(10));
        Distance node = theGraph.GetSpathelement(10);

System.out.println(theGraph.GetVListelement(node.Vert));
        node = theGraph.GetSpathelement(node.Vert);
System.out.println();
```

Figure 9 Results of DijkstraAlgorithm(1st Output)

```
417
Stockholm
Nykoping
Norrkoping
Jonkoping
Boras
Goteborg
```

```
/*System.out.println(theGraph.GetVListelement(10));
        Distance node = theGraph.GetSpathelement(10);
        System.out.println(theGraph.GetVListelement(node.Vert));
        node = theGraph.GetSpathelement(node.Vert);
System.out.println();*/
```

Figure 10 Results of DijkstraAlgorithm (2nd Output)

4. Analysis and Discussion

Compare the results with our expectations; all of what have been created is reasonable. The method is correct and the final results for the data and drawing pictures are half-professional.

1. 4.1 Datasets are used appropriate for the analysis. Data comes from ASCII file which are transformed from the professional GIS software of ArcGIS Desktop 10.0. For shortest path analysis, we get the data from Google Map which has been illustrated previously and it is reliable.
2. Results are reliable when considering the datasets and methods used. Firstly, datasets are reasonable and a lot of relative materials have been referred to get the details of algorithms which have been implemented in the software. Secondly, we make this conclusion by comparing the results getting from ArcGIS.
3. It would have been more appropriate if time is enough. In this case, more contents can be implemented. And more importantly, the structure can be more efficient and the locally outlines can be more clearly by using more classes and different layers. But anyway, it is reasonable to get present situation in this short time.
4. The results for Dijkstra shortest path algorithm is exactly correct with respect to Google map. Therefore, it can be applied for every highway network system.
5. The finding input file for Dijkstra was difficult for us but the first reason was the lack of time because first part took too much time.

5. Reference

Burrough, P. A., and McDonell, R. A., 1998. Principles of Geographical Information Systems (Oxford University Press, New York), 190 pp.

Kingdom of Sweden (2010). Retrieved 1 December, 2010, from French Database of Geographic Coordinate Information site: *http://www.tageo.com/index-e-sw-cities-SE.htm*

Sweden Open Street Map, Retrieved 1 December, 2010, from *http://www.maps.google.com/*