PSERC

Satellite Imagery for the Identification of Interference with Overhead Power Lines

Final Project Report

Power Systems Engineering Research Center

A National Science Foundation
Industry/University Cooperative Research Center
since 1996

Power Systems Engineering Research Center

Satellite Imagery for the Identification of Interference with Overhead Power Lines

Final Project Report

Project Team

Yoshihiro Kobayashi
George Karady
Gerald Heydt
Matthias Moeller
Arizona State University

Robert Olsen
Washington State University

PSERC Publication 08-02

January 2008

Information about this project

For information about this project contact:

George G. Karady (project leader)
Salt River Chair Professor
Arizona State University
Department of Electrical Engineering
P.O. BOX 875706
Tempe, AZ 85287-5706
Phone: 480-965-6569
Fax: 480-965-0745
Email: Karady@asu.edu

Power Systems Engineering Research Center

This is a project report from the Power Systems Engineering Research Center (PSERC). PSERC is a multi-university Center conducting research on challenges facing a restructuring electric power industry and educating the next generation of power engineers. More information about PSERC can be found at the Center's website: http://www.pserc.org.

For additional information, contact:

Power Systems Engineering Research Center
Arizona State University
577 Engineering Research Center
Box 878606
Tempe, AZ 85287-8606
Phone: 480-965-1643
FAX: 480-965-0745

Notice Concerning Copyright Material

Acknowledgements

The work described in this report is for research sponsored by the Power Systems Engineering Research Center (PSERC) project “Satellite Imagery for the Identification of Interference with Overhead Power Lines”, project T-37. We express our appreciation for the support provided by PSERC’s industrial members and by the National Science Foundation under grant NSF EEC 0001880 received under the Industry / University Cooperative Research Center program.

The authors thank industry and government collaborators including Clark Love (Forest One), James Crane (Exelon), Charles Priebe (Exelon) and Phil Overholt (U.S. Department of Energy).

Executive Summary

In recent years, renewed emphasis has been placed on vegetation management of transmission right-of-way to avoid tree contacts that could put system reliability at risk. At the same time, new approaches to vegetation management have been sought to be able to achieve the needed tree clearances as cost-effectively as possible. One possible approach is to process satellite images to prioritize tree maintenance work. For this approach to work, new computational tools would be needed to convert satellite image data into useful information for vegetation management scheduling.

This project's objective was to develop such computational tools for determining the location of trees interfering with overhead transmission lines. The input data were derived from satellite images, and the GPS coordinates and altitudes of transmission towers. The tools determine the location of healthy trees that are penetrating a danger zone or safety envelope (e.g., 20 ft radius) surrounding the conductors. Two tools were implemented and tested**:**

- a transmission line scanning computer program and
- a tall tree identification program.

This work is significant because it shows how satellite images that are already commercially available can be used for the large-scale assessment of vegetation encroachment on transmission lines.

Transmission Line Scanning Computer Program

The Transmission Line Scanning Computer Program uses *multi-spectral satellite images* of the transmission line. The input data are GPS coordinates of the transmission line towers, the width of the right-of-way, and the satellite image data. The program is used to scan the satellite image along the right-of-way from one tower to the next following a straight line; spectral analysis of the scanned satellite data is used to identify areas covered by healthy vegetation. Areas with healthy vegetation represent a potential danger for the line because of possible contact between the line and trees. The output of this program is the GPS coordinates of the healthy vegetation. The code was written using the Java (JDK 1.4) programming language with Java Advanced Imaging (JAI) Application Programming Interfaces. The source code is included in the report.

Tall Tree Identification Program

By knowing the location of the healthy trees along transmission line right-of-way, it is possible to use another tool, the Tall Tree Identification Program, to establish where the healthy trees are endangering the transmission line. Danger zones are areas surrounding the transmission line conductors where trees can produce flashover or a ground fault. The Tall Tree Identification Program is designed to identify healthy trees that penetrate the danger zone.

This program uses a *stereo pair of satellite images* of the transmission line. These stereo images are obtained from commercial outlets of satellite images. The input data are 1) the GPS coordinates and altitudes of the towers, and 2) the location of the healthy trees discovered by the Transmission Line Scanning Computer program. The program calculates the dimensions (i.e.,

the width and height) of the danger zone for different voltage levels. The program scans the danger zone pixel by pixel and calculates the elevation of each pixel using stereo matching. The height of each pixel is measured by using the altitudes of the transmission line towers. The comparison of the location and height of the healthy vegetation pixel with the dimensions of the danger zone identifies where trees are penetrating the danger zone. The program outputs are the GPS coordinates of healthy trees in the danger zone and the distance from these trees to the transmission lines.

Case Studies and Next Steps

Case studies were conducted using a transmission line in the San Diego, California area. This line crosses an area with heavy vegetation and pine trees in the vicinity of the San Diego River at Mission Bay Park. Visual observation of the right-of-way along the line verified the proper operation of the Transmission Line Scanning Computer Program in identifying areas of healthy trees. The Tall Tree Identification Program was then used to identify the closest five trees which ranged from 24.3 to 30.1 feet from the transmission line.

The case studies indicate that the developed programs were able to identify trees endangering a transmission line. However, the programs were not fully tested in a utility environment. They are research-grade programs that require additional testing and development. The following is recommended future work to improve the tools:

- Test using the other image files of a different, larger site (such as 60 miles)
- Compare the results from commercial off-the-shelf photogrammetry packages
- Visualize the Digital Elevation Model data in 3D view
- Add query functions to the extracted trees
- Add more control units to allow the users to change the attributes of transmission towers and lines flexibly and interactively.

Table of Contents

Table of Contents (continued)

Table of Contents (continued)

List of Tables

List of Figures

Nomenclature

API	Application Programming Interfaces
ASU	Arizona State University
DEM	Digital Elevation Model
DSM	Digital Surface Model
DoE	U.S. Department of Energy
GIF	Format for digital image
GIS	Geographical Information Systems
GPS	Global Positioning Satellite
GSD	Ground sampled distance
IREQ	Institut de Reserche Electrique du Quebec
JAI	Java Advanced Imaging
JPG	Format for digital image
LIDAR	Light Detection and Ranging
NDVI	Normalized difference vegetation index
PCR	Plant cell ratio
PPR	Plant pigment ratio
PVR	Photosynthetic vigor ratio
RAW	Format for digital image
SRP	Salt River Project
TIFF	Format for digital image
TVA	Tennessee Valley Authority
UTM	Universal Transverse Mercator
WAPA	Western Area Power Administration
WSU	Washington State University

1.0 Introduction

1.1 Project objectives and participants

The objective of the project is to develop software which determines the location of objects or vegetation interfering with overhead transmission lines. The input data are derived from satellite images and the Global Positioning Satellite (GPS) coordinates of the towers. The code determines the location of trees or objects that are penetrating a safety envelope (e.g., 20 ft radius) surrounding the conductors.

The project was initiated in August 2005. The main participants are shown in Table 1.1. Table 1.2 shows the project industry advisors. In addition to the industry advisors, Clark Love of Forest One supplied a good deal of information for the project.

Table 1.1 Project researchers

Researcher	*University*	*Department*	*Project role*
G. Karady	Arizona State University	Electrical Engineering	Lead
Y. Kobayashi	Arizona State University	Architecture	Researcher
G. Heydt	Arizona State University	Electrical Engineering	Researcher
M. Moeller	University of Dortmund	Geography	Advisor
R. Olsen	Washington State University	Electrical Engineering	Researcher

Table 1.2 Project industry advisors

Advisor	*Company*
Kevin Allen	Oncor Electric Delivery
Luc Audette	IREQ
Lane Cope	WAPA
James Crane	Exelon
Mike Ingram	TVA
Dale Krummen	American Electric Power
David Lubkeman	ABB
Philip Overholt	U.S. Department of Energy
Mahendra Patel	PJM
Don Pelley	Salt River Project
Charles Priebe	Exelon
Don Sevcik	CenterPoint Energy

1.2 Project duration

This research project began in September 2005 and ended in October 2007.

1.3 A statement of the problem

The trees and growing vegetation frequently endanger the operation of a high voltage transmission line. A potential scenario is that during a storm, wind drives overgrown tress close to the line. This may cause flashover between the conductor and tree. The effect of this flashover is unforeseeable. In most cases, protective systems de-energize the line to extinguish the arc. After 3 to 6 cycles, re-energization of the line is possible. However, in some cases the short circuit persists and may trigger wide area outages. An example is the August 2003 blackout in the US Northeast and adjacent Canada which resulted in the loss of power for millions of households and significant industrial and commercial financial losses.

Presently, power companies regularly survey the lines by helicopter to locate trees that may endanger the line. One of the most common technologies used is LiDAR which results in airborne images of rights-of-way. In addition, the expected vegetation growth is calculated using appropriate models. The survey and vegetation growth calculations combined result in a tree trimming schedule. This approach is expensive because relatively small areas are surveyed this requiring a considerable number of airborne surveys. Also, sometimes this approach is inaccurate. As an example, inaccuracies may result from an inaccurate prediction of tree growth due to unexpected rainfall.

The United States is regularly (yearly, or in some areas more frequently) surveyed by satellite. The satellite images of large areas, including transmission lines' rights-of-way, can be purchased. The availability of satellite images is motivation to develop a satellite-image-based, low-cost method that identifies the location of trees that endanger transmission line operation.

The objective of this project is to develop software that determines the location of objects or vegetation that interferes with the operation of an overhead transmission line. The input data are the satellite images of the line and the GPS coordinates of the towers. The code determines the location of trees or objects that are penetrating a safety envelope (e.g., a 20 ft radius) surrounding the conductors.

2.0 Literature Review

2.1 The literature of overhead power system transmission

In transmission engineering, major concerns are the vegetation control under transmission lines and the required clearances between the line and vegetation. A literature survey identified a few codes and an Australian standard which deal with the clearance problem. [28-30]. IEEE started the development of a guide on vegetation management in 2005. PSERC published a report in 2007 proposing new advanced vegetation management based on intelligent system monitoring [46].

A further source of information is the description of power companies' vegetation management strategies. Practically every power company has publications dealing with vegetation management near power lines which are published on their WEB pages. A few typical publications are given in [31-32]. The major conclusion of these publications is that most companies clear the vegetation directly under the transmission line to permit visual inspection by and movement of power companies' emergency maintenance vehicles. The vegetation control near the line eliminates the fast growing tall trees without major environmental effect. Pesticides and chemicals are not used because of their adverse environmental effect. Most companies trim the vegetation regularly but not frequently enough. The typical time interval is 2-5 years.

Another successful policy is the selection and planting of slow growing, low-height vegetation under the transmission lines. Several papers [34-44] deal with this method. Most papers were published in agricultural journals. In the last 15 years, practically no papers were published in IEEE transactions on vegetation management under the transmission lines or required clearances. However, the IEEE transactions has published several papers on transmission line survey methods, and one transaction paper [42] deals with the effect of fire on transmission line operation. Reference [35] gives a general overview of the vegetation control method used in the USA and Canada. Papers [34, 36] present an integrated vegetation management strategy, which eliminates power outages caused by vegetation overgrowth. References [37, 41] present vegetation growth prediction methods, which can be used for timing tree pruning in the transmission lines' rights-of-way. Reference [38] is a company report that deals with the environmental effect of vegetation control and discusses related economic problems. Reference [39] proposes the use of herbicides to control vegetation under the transmission lines and discusses the potential environmental effect of herbicides. Paper [44] proposes the planting of stable shrubs under lines.

The conclusion of this literature review is that vegetation overgrowth has produced flashovers which in turn have initiated major black outs in recent years. Thus, the reliable operation of a transmission system requires efficient vegetation control. However, vegetation control is expensive, and power companies delay pruning trees because of this expense and unfavorable public reaction to such actions. The literature shows that the prediction of vegetation growth is inaccurate, which emphasizes the importance of economical, accurate survey techniques of vegetation around transmission lines.

2.2 Remote sensing and photogrammetry

This section shows the literature reviews of two research fields, remote sensing and photogrammetry. In this project, remote sensing is related to the technology of identifying vegetation area, and photogrammetry is related to the technology of determining the height of trees and plans.

Remote sensing is defined as “the measurement or acquisition of information of an object or phenomenon, by a recording device that is not in physical or intimate contact with the object” in WIKIPEDIA [1]. This area of research initiated in 1858 by taking photographs of Paris from a balloon. In World War I, systematic aerial photography was developed for military purposes. Currently, many kinds of measurement devices are used, such as radar, laser, LIDAR, radiometers, photometers, stereographic pairs of aerial photographs, and multispectral images.

On the other hand, Photogrammetry is defined as “a measurement technology in which the three-dimensional coordinates of points on an object are determined by measurements made in two or more photographic images taken from different positions” [1].

2.3 Satellite images

At present there are several main civilian sources for and types of satellite images. For the purpose of extracting ground features in general and tall trees and higher bushes in particular, the ground sample distance (GSD), which is the spatial resolution of imagery, should be very high (e.g., one meter or smaller) [2,3]. Ikonos, QuickBird and OrbView sensors are capable of very high-resolution satellite images and are available on a commercial basis.

According to a satellite image provider, SPACE IMAGING [4], the IKONOS satellite is the world's first commercial satellite to collect panchromatic images with 1 meter GSD and multispectral imagery with 4 meter GSD. Ikonos was launched in September, 1999 and started providing imagery in January, 2000. The images are taken at the satellite altitude of 680 km. The accuracy of an ortho-rectified image is ±1.75 m. Ikonos takes 11 days to return to a location (revisit cycle). Some image examples of the IKONOS sensor are available for free download from web sites, e.g., [4].

QuickBird is a high resolution satellite owned and operated by DigitalGlobe. Using a state-of-the-art Ball’s Global Imaging System 2000 sensor, QuickBird uses remote sensing to a 0.61 meter GSD. It was launched in October, 2001, and acquires images at the satellite altitude of 450 km. The revisit cycle for QuickBird is 1~3.5 days. Information on the QuickBird satellite is available at [5]. According to [6,7], a stereo pair of QuickBird images is readily used to extract the height of buildings. Figure 2.1 shows the concept of generating stereo pair satellite imagines by photographing an area twice using different camera angles.

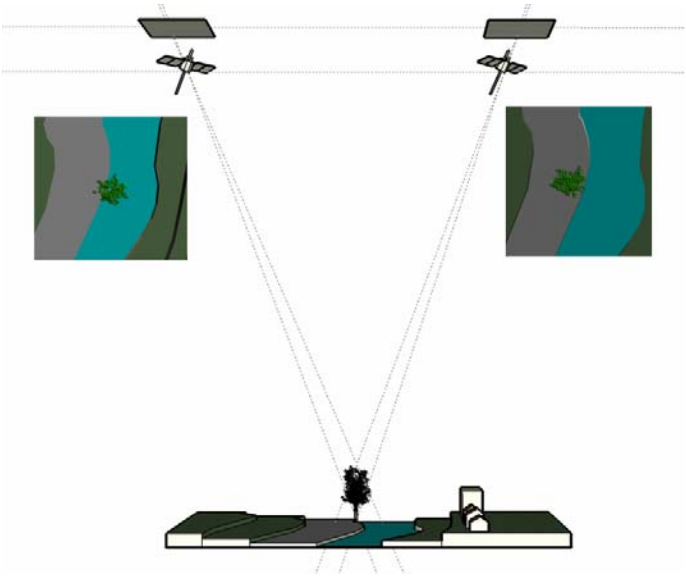


Figure 2.1 Stereo pair of satellite images

OrbView also offers one of the high-resolution satellite images. The original OrbView-1 was launched in 1995, and the mission was complied in 2000. OrbView-2 and OrbView-3 were launched in 1997 and in 2001. The OrbView-3 takes 1 meter panchromatic and 4 meter multispectral images. The next generation, OrbView-5, will be launched in early 2007, and it will offer the highest resolution available to date by simultaneously acquiring 0.41 meter panchromatic and 1.64 meter multispectral imagery [8], which will be the highest-resolution satellite images available to civilians.

2.4 Vegetation identification

In remote sensing, extracting specific vegetation areas in aerial and satellite images is one of the current research interests. As described in [24], the plant pigment ratio (PPR), the photosynthetic vigor ratio (PVR), the plant cell ratio (PCR), and normalized difference vegetation index (NDVI) are tested for identifying vegetation or forest areas.

In contemporary research projects on this topic, many researchers use one of the commercial off-the-shelf remote sensing packages, such as the EARDA IMAGINE Subpixel Classifier, for terrain classification, identifying the location of trees, and distinguishing the trees from low shrubbery and grasses [26].

Reference [27] tries to identify the species of vegetation using multispectral and multitemporal data about canopy texture, leaf density, and spectral reflection.

2.5 Height extraction

A common method for the estimation of vegetation height is the analysis of analog and later digitized aerial photos by means of image stereoscopy. A new technology, which is still under scientific investigation for right-of-way tree trimming identification, is based on LIDAR

(Light Detection and Ranging). For example, Morsdorf makes use of LIDAR technology to extract vegetation structure for fire management [9]. In [10], Clode tries to classify the trees and power lines using LIDAR data, and [11] shows the possibility to extract power lines and towers directly using LIDAR. However, in the LIDAR approach it is necessary to fly over all of the power lines in order to observe trees and plants, which, though it is possible to acquire highly accurate data, is very time and labor intensive. LIDAR technology is based on an active laser pulse signal which is sent out from the airplane towards the earth's surface. The reflected signal is recorded by a special sensor. The distance between the airplane and surface can then be calculated from the time the beam takes from the laser to the ground and back to the sensor. However, LIDAR is still a challenging technology to use for the identification of vegetation. Figure 2.2 shows the concept of acquiring LIDAR data with an airplane.

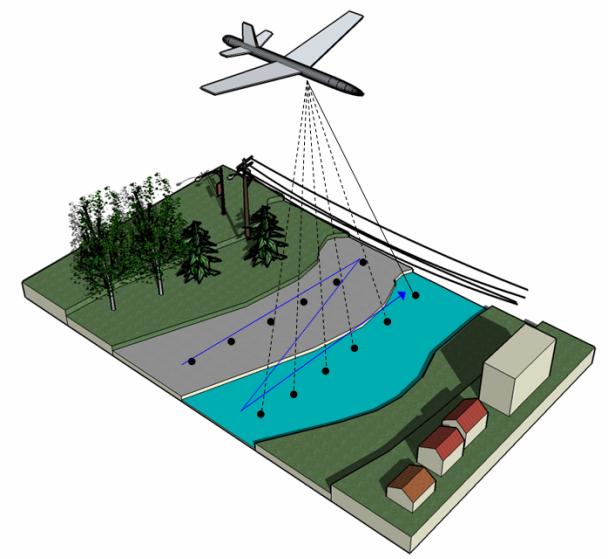


Figure 2.2 Illustration of LIDAR technology

3.0 Calculation of Vegetation Interference from Satellite Images

3.1 Introduction

The power industry, as with most industries, is driven by cost and benefit. Overhead transmissions require maintenance, which includes tree trimming. In the technology of tree trimming, concerns include the identification of trees close to overhead circuits and the resulting time and cost of this identification. It takes a significant amount of time and labor to manually investigate trees under all the power lines by walking or riding the line. Though an approach using LIDAR and aerial photographs is a viable alternative, this approach requires flying over a larger number of circuit miles of overhead lines each year.

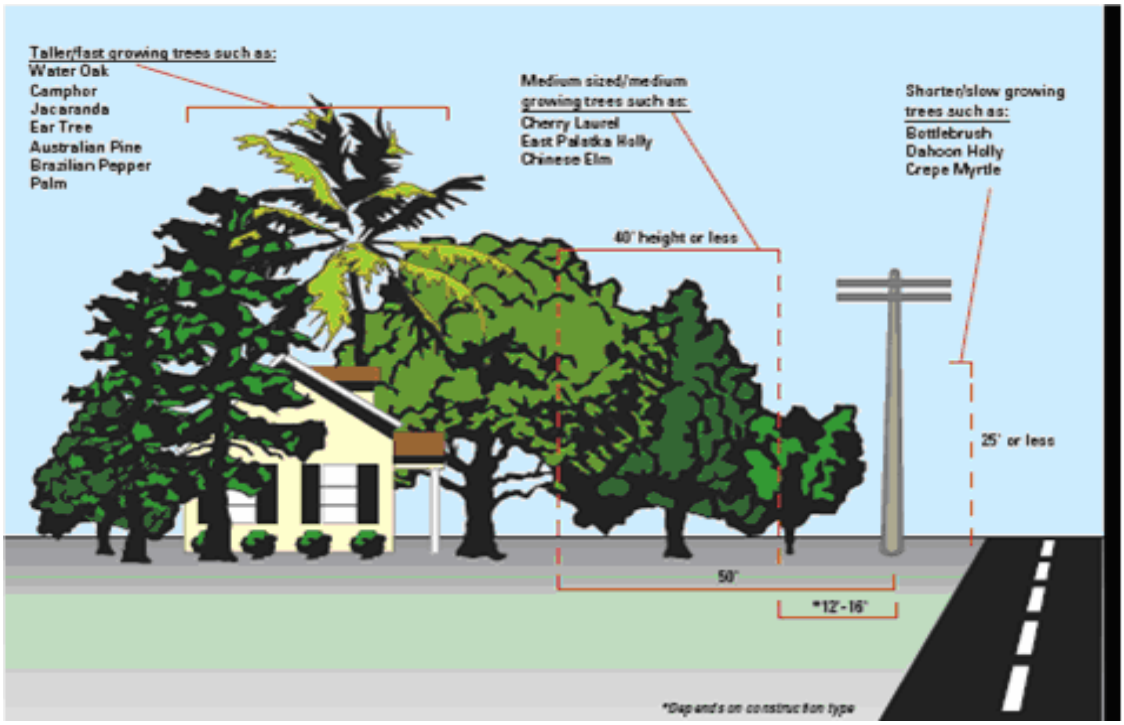


Figure 3.1 Concept of vegetation management on the right-of-way (taken directly from the Western Area Power Administration web site www.wapa.gov)

The other method to extract the height of trees from images is to estimate the height from the radius of a tree's crown. Borgefors and Hyyppä [12,13] showed that the stem diameter correlates with the height and the radius of the crown. Pollock [14] proposed a generic model to represent the size of a tree with a formula, called a generalized ellipsoid of revolution (GER). Straub and Heipke demonstrated an application to extract trees using aerial color infrared images and a dense digital surface model [15]. A series of research projects by Yasuoka [16, 17] showed techniques to estimate the number of trees in an area using both satellite images and LIDAR data.

The objective of this project is to reduce the cost and time of determining the trees interfering with power lines, and it is proposed to use a stereo pair of multispectral satellite images. This approach is suggested for a number of reasons.

1. A pair of high-resolution stereo satellite images is available easily at a low price.
2. New satellite images are taken every few days.
3. It is not necessary to fly or drive the lines, so this method has the potential to substantially reduce costs.
4. This approach may yield more uniform results because the procedure is automated.

3.2 Present practices

Practically every power company has developed a policy for vegetation management around transmission lines. The right-of-way vegetation management policies of Georgia Power, Bonneville Power Administration, Tennessee Valley Power Authority and Hydro Quebec Trans Energy were examined. The basic concept of vegetation control is illustrated in Figure 3.1. This figure shows the typical trees that can be found close to a transmission line. The figure shows that under the line a free area must be kept to allow for inspection and line patrol. The short, slow growing trees or shrubs with expected heights less than 25 ft can be planted close to the line in a distance of more than 12 ft-16 ft. The medium height and medium fast growing trees with an expected height less than 40 ft, must planted more than 50 ft from the line. This prevents flashover if a storm uproots the tree. The distance between the conductor and a fallen tree must remain large enough to prevent flashover. The figure gives the names of the different types of trees that can be found around transmission lines in the USA.

The objective of vegetation management on the right-of-way is to maintain healthy low-growing plant communities (e.g., shrubs, grasses, and native ferns), and cut trees adjacent to the right-of-way determined to be a current or future hazard (due to diseased, damaged or other unstable conditions) to the transmission line. Vegetation management includes mowing, cutting dangerous trees, pruning, applying herbicide, and removing trees. Typically, the right-of-way is mowed once every six years. In addition, herbicide is used to control vegetation, typically fast growing tall trees. The herbicide is applied manually using backpack sprayers, and non-restricted herbicides are used. All of the companies are aware of the potential adverse environmental effect of herbicides and try to minimize their use.

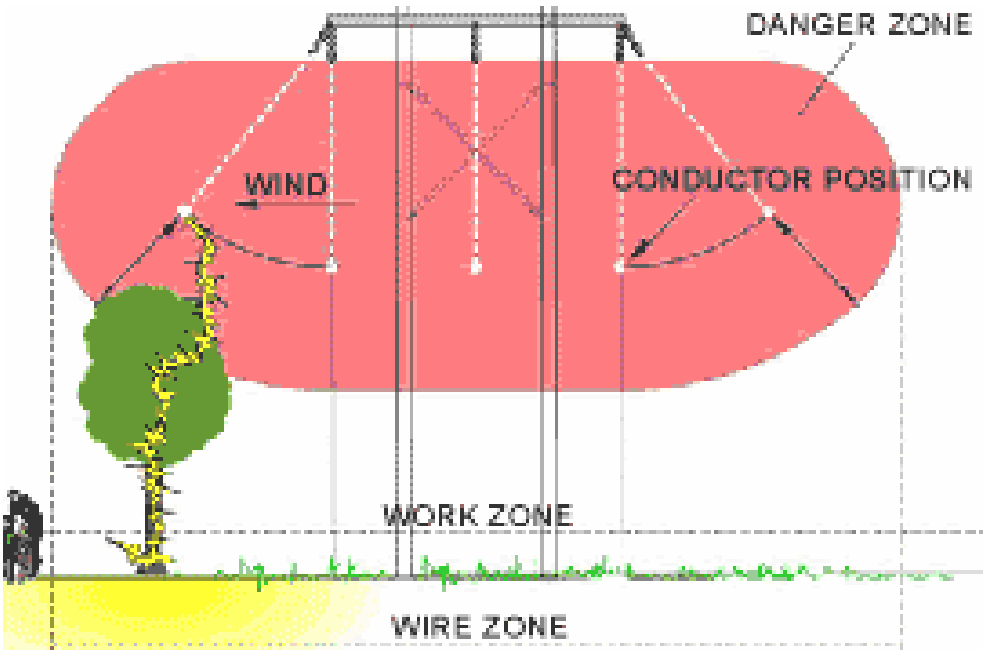


Figure 3.2 Identification of the danger zone around the conductors (taken directly from the Western Area Power Administration web site www.wapa.gov)

The power companies generally patrol all transmission lines once a year by air or by ground patrol to identify "dangerous" trees. The dangerous or problematic trees are those that grow in the easement area and approach the line. The companies regularly prune these trees. Normally, pruning is required every two to fifteen years depending of the type of trees. Some companies may remove trees closer than 15 feet from the easement area. However, the most frequent practice of tree cutting is to prune the tree if the distance between the line and tree is less than a safe value. This leads to the identification of a danger zone around the conductors as shown on Figure 3.2.

Figure 3.2 shows that it is necessary to consider the temperature dependent sag and the wind-caused swing of the conductors. Obviously, this danger zone depends on the tower configuration and voltage-to-ground. The trees penetrating into the danger zone are cut back or removed from the right-of-way or easement area.

NERC VM Standard FAC-003-1 defines the minimum vegetating conductor clearances per IEEE Std. 516-2003 "Guide for Maintenance Methods on Energized Power Lines". In this standard Table 5 gives the Minimum Air Insulation Distances (MAID) without Tools in the Air Gap. Table 3.1 shows the minimum voltage dependent clearances between conductor and vegetation. The table shows that the minimum clearance is 0.75 m or 2.45 ft for a sub-transmission line and 4.48m or 14.66ft for a 500 kV line. Different companies use different minimum clearance values.

Table 3.1 Minimum Air Insulation Distances (MAID) without Tools in the Air Gap.
(Data are copied from Table 5 in IEEE Std. 516-2003)

No	Nominal line-to-line voltage kV	Conductor to vegetation clearance (m/ ft)
1	72.6-121	0.75m/ 2.45ft
2	138-145	0.9m/ 2.94ft
3	161-169	1.05m/3.42ft
4	230-242	1.57m/6.14ft
5	345-362	2.88m/9.44ft
6	500-550	4.48m/14.68ft
7	765-800	6.24m/20.44ft

3.3 Determination of the danger zone

The minimum clearances must be maintained during the worst condition. This suggests that the sag of the line is calculated in the worst condition, which is either the maximum temperature or the maximum ice loading at 15°F. The sag data are available in digital format.

The other worst condition is the maximum swing of the line conductor. Determining this condition requires the identification of the loading zone of the area, which defines the maximum wind load. The loading zone area is found using the National Electrical Safety Code. With these data and the conductor weight, the swing angle is calculated. A sample calculation is shown below.

Transmission line data:

Tower height $H_{tower} := 90\,ft$

Distance between phases $D_{ph\_ph} := 35\,ft$

Distance between conductor and structure, no wind $D_{con\_str} := 15\,ft$

Distance between towers $Span := 1000\ ft$

Arbutus All-Aluminum Conductor

Conductor diameter $d_{cond} := 1.026\ in$

Conductor weight $w := 746 \cdot \frac{lb}{1000\ ft}$

Sag at 15F with ice loading $Sag_{15F} := 48.24\,ft$

The length of the insulator chain is: $L_{chain} := 5.08\ m$

The line is in northern Arizona, which is specified as a Medium loading area by the National Electrical Safety Code NESC. Table 1 of the NESC gives the wind force and ice loading values that must be used for design of transmission lines.

Wind force

$F_{wind\_Med} := 4\frac{lb}{ft^2}$ $\quad Temp := 15F$

Area subjected to the wind is: $\quad A_{wind} := d_{cond} \cdot Span \quad A_{wind} = 7.943m^2$

$F_{wind} := F_{wind\_Med} \cdot A_{wind} \quad F_{wind} = 155.129kg$

Ice load

Ice forms a hollow cylinder on the conductor surface.
The average diameter is:

$D_{inner} := d_{cond} \quad D_{ice} := 0.25in$

$D_{ave\_ice} := \frac{D_{inner} + D_{outer}}{2} \quad D_{outer} := d_{cond} + 2 \cdot D_{ice}$

$D_{outer} = 1.526in \quad Span \cdot D_{ave\_ice} \cdot \pi = 31.035m^2$

Ice weight is: $\quad \rho_{ice} := 57\frac{lb}{ft^3}$

$W_{ice} := Span \cdot \frac{D_{ave\_ice}^2}{4} \cdot \pi \cdot \rho_{ice} \quad W_{ice} = 229.599kg$

Conductor weight $\quad W_{cond} := Span \cdot w \quad W_{cond} = 338.38kg$

Vertical force $\quad F_{Cond\_ice} := W_{cond} + W_{ice} \quad F_{Cond\_ice} = 567.979kg$

The total force and its angle to the vertical line

$F_{con\_ice\_wind} := \sqrt{F_{Cond\_ice}^2 + F_{wind}^2} \quad F_{con\_ice\_wind} = 588.782kg$

$\Phi_o := atan\left(\frac{F_{wind}}{F_{Cond\_ice}}\right) \quad \Phi_o = 15.276deg$

The swing angle determines the position of the conductor in the horizontal direction. Figure 3.3 shows the conductor position. The minimum clearance is a circle around the conductor. This circle determines the danger zone. The vegetation, trees, must not penetrate this zone. This calculation determines the width of the danger zone around the transmission line.

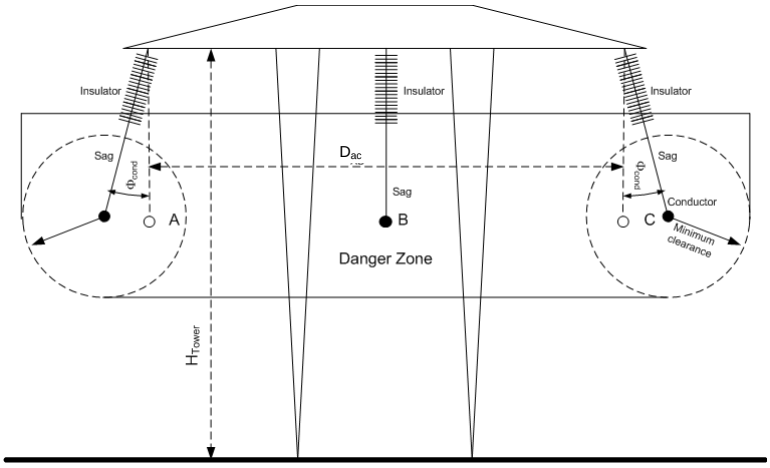


Figure 3.3 Determination of the danger zone

The conductor position also depends on the distance from the tower because at the tower the sag is zero, and it increases with distance. The sag is at a maximum at the midpoint if the adjacent towers are on the same elevation, flat terrain. With the increasing sag the width of the danger zone increases. Accordingly, the danger zone dimensions are at a maximum at the middle of the span and minimum at the tower. The identified danger zone depends on the distance from the tower. Figure 3.4 shows the danger zone at the tower and at the middle of the line.

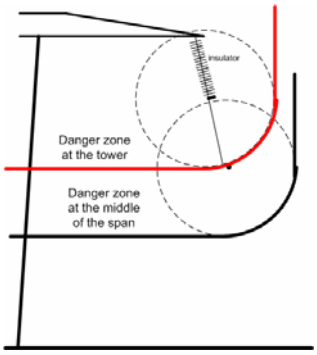


Figure 3.4 An illustration of danger zone dependence from the distance from the tower

Figure 3.4 shows that the width of the danger zone increases slightly. The example shows about a 12% difference between the width of the danger zone at the tower and at the middle of the line

The height of the danger zone also depends on the distance to the tower. The maximum is at the tower and the minimum is at the middle of the line. The example shows that the difference in height is around 58%. The calculation of danger zone widths and heights as a function of distance from the tower is presented below.

3.4 Calculation of danger zone dimensions

The following shows the calculation of danger zone dimensions.

Transmission line data

Tower height	$H_{tower} := 90ft$
Distance between phases	$D_{ph\_ph} := 35ft$
Distance between conductors and structure, no wind	$D_{con\_str} := 15ft$
Distance between towers	$Span := 1000ft$
Maximum value of Sag	$seg_{max} := 30ft$
Maximum clearance	$D_{clearance} := 24ft$
The length of the insulator chain is:	$L_{chain} := 5.08m$
Swing angle	$\Phi_{swing} := 15.276deg$
Distance between phases A and C	$D_{AC} := 23.2ft$

Approximate sag calculation

The sag depends on the distance from the tower. The conductor shape is approximated with a second order parabola. In the sag equation, x is the distance form the tower.

$sag(x) = a \cdot x^2$ Using this equation the maximum sag is: $sag_{max} = a \cdot \left(\frac{Span}{2} \right)^2$

$a := \frac{4 seg_{max}}{Span^2}$ $a = 3.937 \times 10^{-4} \frac{1}{m}$

The equation for the sag is: $sag(x) := seg_{max} - \frac{4 seg_{max}}{Span^2} \cdot \left(\frac{Span}{2} - x \right)^2$

The sag vs. distance (x) from the tower

$x := 0ft, 1ft .. 1000ft$

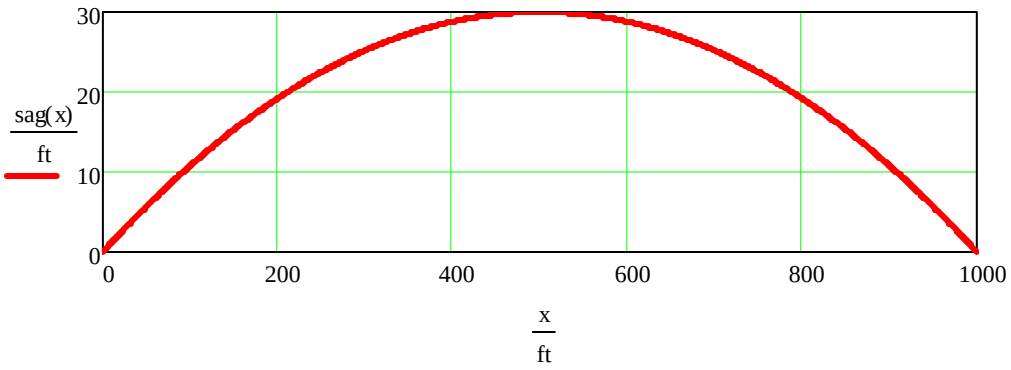


Figure 3.5 Sag vs. distance from the tower

Danger zone width and height calculation

The inspection of Figure 3.3 leads to equations for calculation of the width of the danger zone.

$$W_{Danger}(x) := D_{AC} + 2 \cdot D_{clearance} + 2 \cdot \left(sag(x) + L_{chain}\right) \cdot \sin\left(\Phi_{swing}\right)$$

$$W_{Danger}(0ft) = 126.782\,ft \qquad W_{Danger}\left(\frac{Span}{2}\right) = 142.59\,ft$$

Substitution of the sag equation results in a closed formula for danger zone width calculation

$$W_{Danger}(x) = D_{AC} + 2 \cdot D_{clearance} + 2 \cdot \left[sag_{max} - \frac{4 sag_{max}}{Span^2} \cdot \left(\frac{Span}{2} - x\right)^2 + L_{chain}\right] \cdot \sin\left(\Phi_{swing}\right)$$

Figure 3.6 shows the variation of the danger zone width with the distance from the tower. The figure shows that percentage difference between the maximum and minimum width of the danger zone is only around 12%. This suggests selecting a constant width of the danger zone at a value of 143 ft.

The distance between the bottom of the danger zone and the ground is:

$$H_{danger}(x) := H_{tower} - D_{clearance} - \left(sag(x) + L_{chain}\right)\cdot\cos\left(\Phi_{swing}\right)$$

$$H_{danger}(0) = 49.922\cdot ft \qquad H_{danger}\left(\frac{Span}{2}\right) = 20.982\cdot ft$$

Substitution of sag equation results in closed formula for danger zone height calculation

$$H_{danger}(x) = H_{tower} - D_{clearance} - \left[seg_{max} - \frac{4seg_{max}}{Span^2}\cdot\left(\frac{Span}{2} - x\right)^2 + L_{chain}\right]\cdot\cos\left(\Phi_{swing}\right)$$

Figure 3.7 shows the variation of the danger zone height with the distance from the tower.

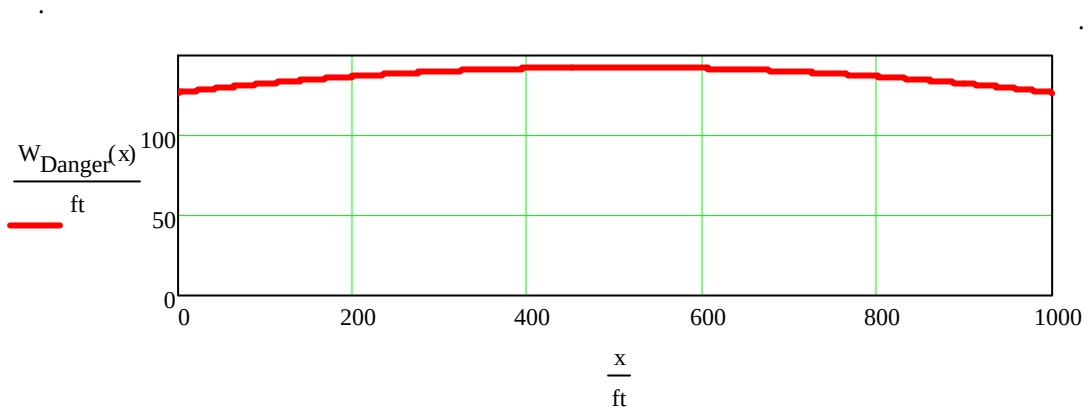


Figure 3.6 Danger zone widths vs. distance from the tower

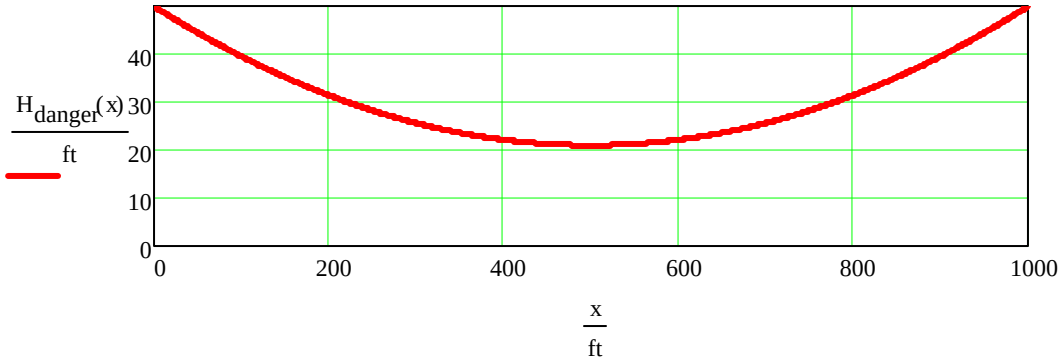


Figure 3.7 Danger zone heights vs. distance from the tower

Trees or vegetation cannot penetrate the danger zone; this includes high trees that are outside the danger zone but close enough that a storm caused uprooting of the tree could result in penetration in the danger zone. Even a falling tree cannot penetrate the danger zone. Figure 3.8 illustrates falling tree penetration in the danger zone.

Accordingly, it is necessary to identify an intermediate danger zone, where a tall falling tree may cause flashover. The approximate width of the intermediate danger zone depends on the area. Typically in Arizona, maximum tree height is around 70 ft; simultaneously a redwood tree in California can be more than 200 ft tall. Local Power Companies must establish the intermediate zone. The intermediate danger zone width must be added to the danger zone width.

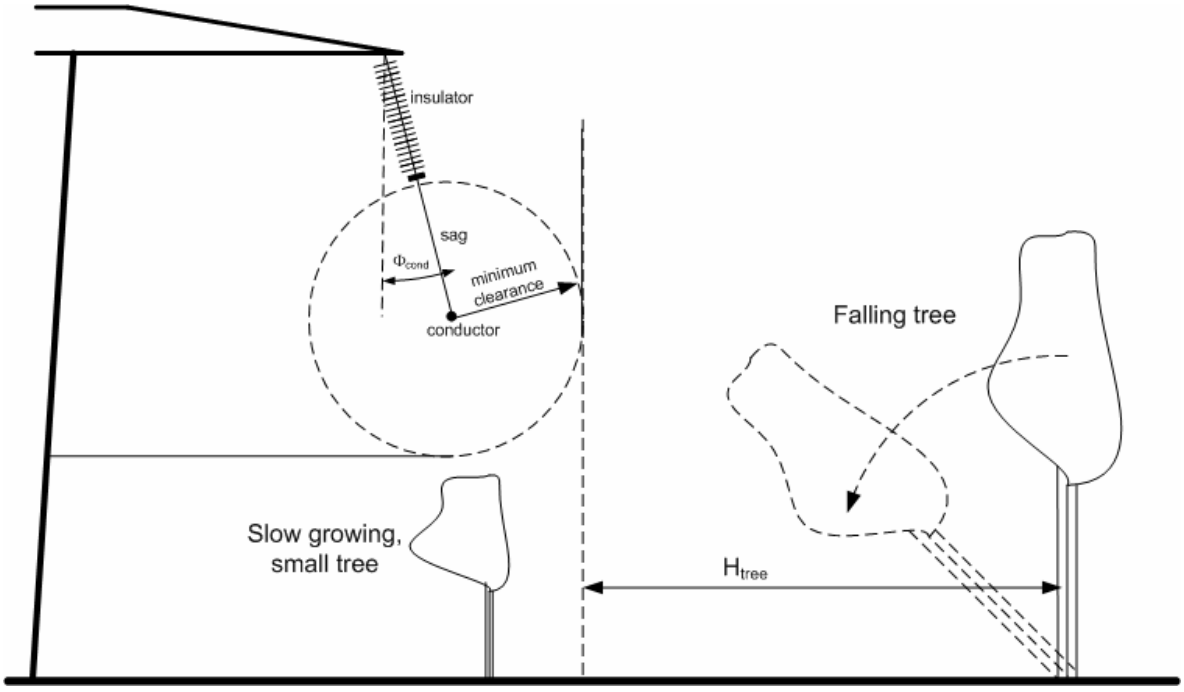


Figure 3.8. Falling tree penetration in the danger zone

3.5 Derivation of an equation describing the danger zone

The calculation above determines the danger zone as the function of the distance from one of the towers, but the satellite tree identification project requires the coordinates of the danger zone and the intermediate danger zone. Figure 3.9 shows the two towers identifying the span and danger zone in an *x, y* coordinate system. The input data are the GPS (UTM) coordinates of the two towers, span length or distance between the towers and the maximum sag. The maximum sag and span are used to calculate the widths and heights of the danger zone and intermediate danger zone. In Figure 3.9 only the danger zone width is used.

If one assumes that the satellite imaging system has identified a tall tree located close to the line, the desired output presents the GPS (UTM) coordinates of the tree. Figure 3.9 shows

that the distance from the tree to the line can be calculated by determining the coordinates of the intersection of two lines: the line between the two towers and the perpendicular line going through the tree point. The calculation steps are listed below.

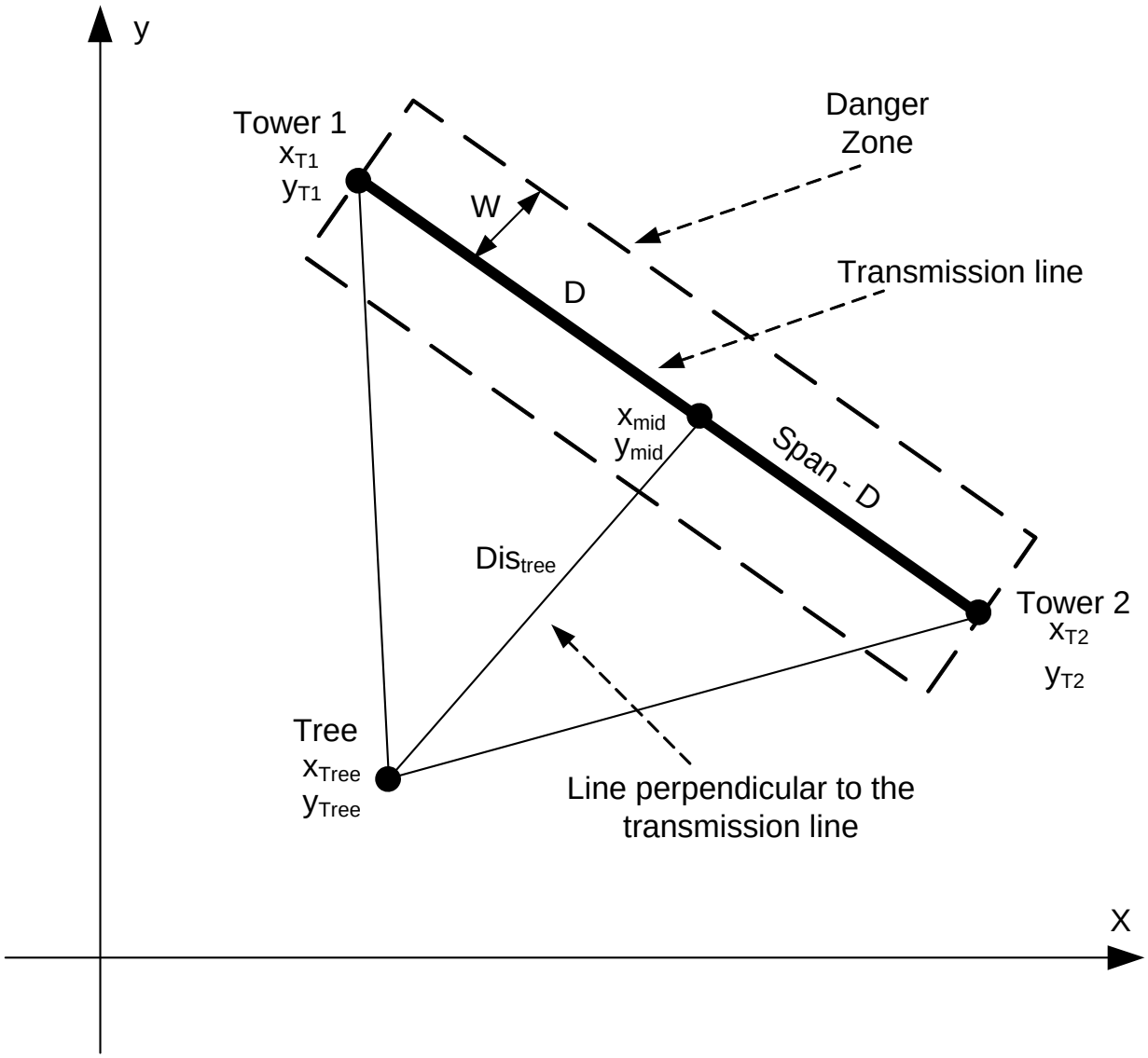


Figure 3.9 A typical transmission line span and its coordinates

3.6 Calculating the distance between the transmission line and tree

Coordinates of the tower and the tree

$x_{T1} := 1197\text{ft}$ $x_{T2} := 2173\text{ft}$ $x_{tree} := 1540\text{ft}$

$y_{T1} := 853\text{ft}$ $y_{T2} := 1071\text{ft}$ $y_{tree} := 676\text{ft}$

Equation of the transmission line

$$m_{line} := \frac{y_{T1} - y_{T2}}{x_{T1} - x_{T2}} \qquad \delta_{line} := \operatorname{atan}\left(m_{line}\right) \qquad \delta_{line} = 12.591\text{deg}$$

$$y - y_{T1} = m_{line}\cdot\left(x - x_{T1}\right)$$

Equation of a line perpendicular to the transmission line and going through the tree coordinates is:

$\delta_{tree} := \delta_{line} - 90\text{deg}$ $\delta_{tree} = -77.409\text{deg}$ $m_{tree} := \tan\left(\delta_{tree}\right)$

$m_{tree} = -4.477$ $y - y_{tree} = m_{.tree}\cdot\left(x - x_{tree}\right)$

The intersection of the two lines is:

$$y_{mid} - y_{T1} = m_{line}\cdot\left(x_{mid} - x_{T1}\right) \qquad y_{mid} - y_{tree} = m_{.tree}\cdot\left(x_{mid} - x_{tree}\right)$$

The subtraction of the equations yields

$$y_{tree} - y_{T1} = \left(m_{line} - m_{tree}\right)\cdot x_{mid} + m_{tree}\cdot x_{tree} - m_{line}\cdot x_{T1}$$

The coordinates of the mid point are:

$$x_{mid} := \frac{\left(y_{tree} - y_{T1}\right) - \left(m_{tree}\cdot x_{tree} - m_{line}\cdot x_{T1}\right)}{m_{line} - m_{tree}} \qquad x_{mid} = 452.946\text{m}$$

$$y_{mid} := m_{tree}\cdot\left(x_{mid} - x_{tree}\right) + y_{tree} \qquad y_{mid} = 279.673\text{m}$$

The distance between the tree and the line is:

$$DIS_{tree} := \sqrt{\left(x_{tree} - x_{mid}\right)^2 + \left(y_{tree} - y_{mid}\right)^2} \qquad DIS_{tree} = 75.442\text{m}$$

$DIS_{.tree}$ must be more than the danger zone width W

The comparision requires the calculation of the variable danger zone width. The first step is the calculation of the distance between the "mid" point and the tower 1. This followed by the calculation of the span.

$$D := \sqrt{\left(x_{T1} - x_{mid}\right)^2 + \left(y_{T1} - y_{mid}\right)^2} \qquad D = 90.272\text{m}$$

$$Span = \sqrt{\left(x_{T1} - x_{T2}\right)^2 + \left(y_{T1} - y_{T2}\right)^2} \qquad Span = 1000\text{ft}$$

The next step is the calculation of the danger zone width and height

$$W_{Dang}(x) := D_{AC} + 2 \cdot D_{clearance} + 2 \cdot \left[\frac{4 \cdot sag_{max}}{Span^2} \cdot \left(\frac{Span}{2} - x \right)^2 + L_{chain} \right] \cdot \sin\left(\Phi_{swing}\right)$$

$$H_{Dang}(x) := H_{tower} - \left[\frac{4 \cdot sag_{max}}{Span^2} \cdot \left(\frac{Span}{2} - x \right)^2 + L_{chain} \right] \cdot \cos\left(\Phi_{swing}\right) - D_{clearance}$$

$$H_{Dang}(D) = 13.75\text{m} \qquad W_{Dang}(D) = 39.444\text{m}$$

The previous calculation shows that the distance between the tree and the line is: D_{tree} = 75.4 m. Consequently, the tree is not in the danger zone. Thus, the presented calculation gives a procedure to identify trees within the danger zone.

4.0 Methodology for Development of Computer Programs

4.1 Introduction

This section shows the methodology for implementing a system to identify trees interfering with overhead power lines from a stereo pair of multispectral satellite images. The framework of the system, the technologies used for implementation and the implementation schedule are explained in Sections 4.2, 4.3, and 4.4, respectively.

4.2 Theoretical framework

The objectives are the development of two computer programs to identify trees endangering transmission line operation. The development is organized into distinct stages:

1. A Transmission Line Scanning Computer program to detect trees and vegetation using commercially available multispectral satellite images.
2. A Tall Tree Identification program, which has two parts:
    a. Development of the Digital Surface Model (DSM) from a pair of stereo images.
    b. Development of a system to identify trees interfering with the lines.

Figure 4.1 shows the diagram of the system framework, and the following list describes the steps that must be performed by the programs identifying tall trees adjacent to the transmission line:

1. Load and display a pair of multispectral stereo satellite images
2. Load the data about transmission lines and towers
3. Calculate the location of the lines and towers on the image
4. Load danger zone data
5. Specify and display the danger zone area along the lines
6. Detect the trees and plants
7. Calculate stereo matching
8. Generate Digital Surface Model
9. Control the threshold value for detecting vegetation
10. Display the high trees close to reaching the power lines inside the envelope area

Java, one of the object-oriented programming languages, and JAI (Java Advanced Imaging), an example of API (Application Programming Interfaces) for image processing in Java, are used for the implementation of the system because they provide advantages for transferring and displaying large scale satellite images through the networks.

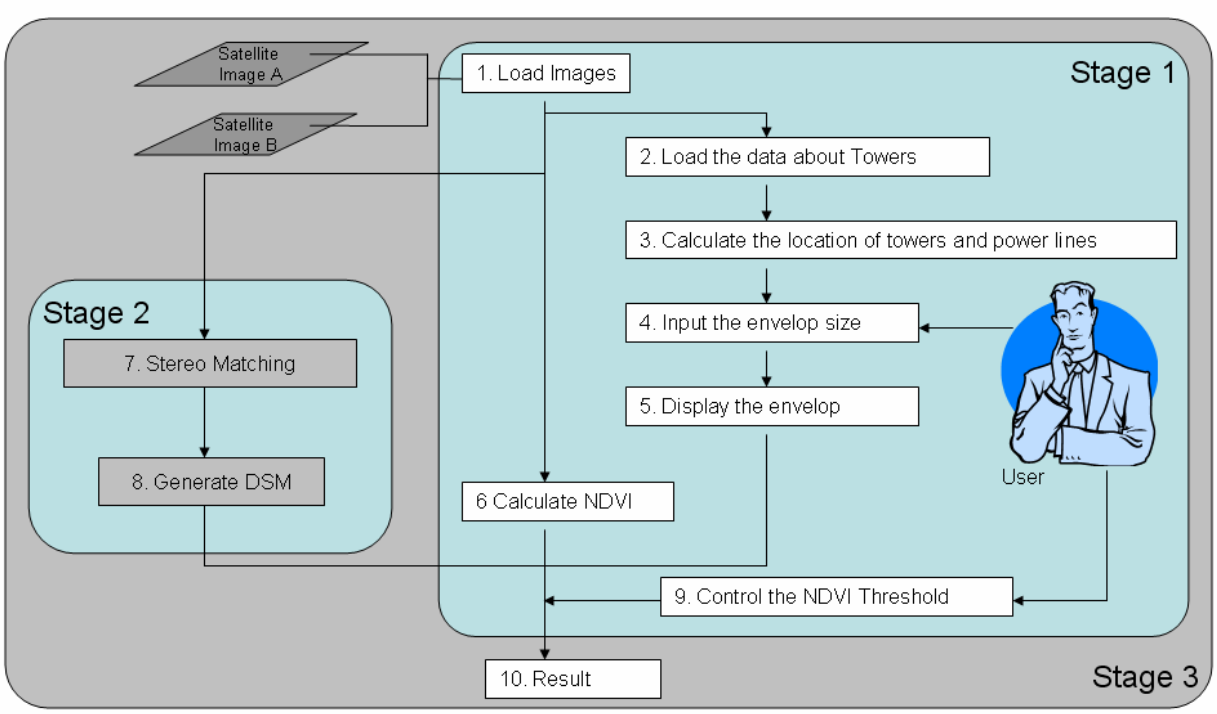


Figure 4.1 Diagram of the system framework

There are many formats for digital images such as JPG, GIF, TIFF, and RAW. Remote sensing techniques especially for the detecting of vegetation commonly use TIFF format (tagged image file format) because it is able to save four bands, blue, green, red, and infrared bands, in one file. In addition to the increased numbers of bands, TIFF files can include additional text information with tags in the format. One of the most common formats in remote sensing is GeoTIFF [18], which is one of the TIFF formats with geographic data. For the detection of vegetation, the ratio of red/infrared is used. The phenomenon is called the red edge because the reflected signal from healthy vegetation shows a steep slope and very strong increase in this portion of the electromagnetic spectrum.

Each of the following sections explains the concept of process in the system.

4.3 Loading and displaying images

This is a description of Process 1which is illustrated in Figure 4.1. A multispectral satellite image file is loaded by selecting the data source in a file browser. A stereo pair of images is also loaded in the same way. The loaded image is not displayed directly but subdivided into an array of tiled images with smaller pixels, such as 256 x 256, in order to display the image in real time. Otherwise, it is not feasible to handle large scale satellite images on regular PCs because of the limitation of memory size. The size of satellite image is usually bigger than that of the computer screen, so only a small part of a loaded image is displayed in the main panel. The visible small area on the main-panel is projected in the sub panel as a boundary box. The user is able to scroll the image and change the visible site by dragging the image with the mouse into the main panel.

4.4 Loading a text file of towers

The following describes how to load a text file of transmission tower information. This is Process 2 depicted in Figure 4.1. The user loads one text file containing the location of transmission towers. In the location file, each tower is indexed with the location information, the longitude and latitude. The index number represents the linkage of transmission lines. For example, the transmission lines are connected between the tower indexed as *N* and the tower indexed as *N*+1. This information is generally available from the plan and profile studies of transmission lines.

4.5 Displaying the towers and lines

The following is a description of how towers are displayed. This is Process 3 in Figure 4.1. After loading the satellite image file and the location file, the towers and lines are displayed on the image by calculating the pixel positions of towers from the longitude and latitude. For the GeoTIFF formatted image file, the pixel positions are calculated by extracting the geographical information in the file. On the other hand, it is necessary for regular TIFF images to define the geographical data manually. The popular public domain, Google Earth [19], may be one method of acquiring the geographical location information. To do this the user must choose three points on the loaded image and, by using the longitude and latitude coordinates, select the same locations in Google Earth. The satellite image use X-Y coordinates with pixels. In order to define the projection system between these two coordinates, the chosen three points are required. The details of this process are explained in Chapter 5. Once the projection system is defined, the geographical location of the towers is expressed using the XY pixel coordinate.

4.6 Defining the danger zone

Processes 4 and 5 in Figure 4.1 relate to defining the danger zone around the transmission line conductors. The region of interest (ROI) is a region marked over an image, and it is defined by the tower positions. The pixels inside a circle with a 30 meter radius around each transmission tower are selected and added to the ROI by default. The enveloped pixels with the same radius along the power lines are added. Depending on the dimensions of the danger zone, the user can change the radius.

4.7 Extracting the vegetation areas

Processes 6 and 9 in Figure 4.1 show the extraction of the vegetation. One important process in Stage 1 is the exact mapping of vegetation. By defining the ROI, only the defined pixels in the image are evaluated (whether the pixel has the strong signal of vegetation or not). The calculation is done by using a Normalized Differenced Vegetation Index (NDVI), which is defined as:

$$NDVI = (NIR - R) / (NIR + R) \quad (4.1)$$

where NDVI is the intensity of the signal in the spectrum corresponding to the amount of healthy vegetation present in the specific pixel, NIR is the near infrared DN, and R is the red DN. The

value range is normalized for programming convenience between -1 and 1. In order to show only the healthy vegetation area properly, the best threshold value needs to be defined. The value is defined manually by the user because it varies from image to image depending on the sensor, the actual conditions when the image was taken such as weather, illumination and shadows, and the sensor features such as focal length. It is highly desired to find an automated way to specify thresholds. However, the practical value for NDVI is approximately 0.1-0.3.

4.8 Stereo matching and DSM generation

Stereo matching in the DSM generation is denominated as Processes 7 and 8 in Figure 4.1. Stereo matching and digital surface model (DSM) generation is implemented in Stage 2. This process includes several steps: first, panchromatic high resolution satellite stereo images have to be loaded as a stereo pair. Then the appropriate sensor model has to be assigned the image data, and both images have to be linked to each other with tie points. After that, each absolute orientation ground control point (gcp), has to be identified with an accurate *x*, *y*, and *z* value in both images. Finally, a 3D model can be calculated, providing *x*, *y*, and *z* values for each pixel. A test DSM using a commercial off-the shelf photogrammetry package, ERDAS IMAGINE, is demonstrated in Chapter 7.

4.9 Showing the results

The depiction of results is represented by Process 10 in Figure 4.1. This step shows the results in 2D and 3D views by integrating the data: NDVI image, DSM, and the danger zone data. The DSM mapped with the NDVI image will be generated, and the danger zone will be displayed on the image. A GIS-based (Geographical Information System) calculation will automatically highlight areas of potential interference by vegetation in the danger zone.

4.10 Technologies utilized to implement the theory

This section introduces some key technologies used in implementing the system to identify trees interfering with overhead power lines.

Java

Java [20] is one of the object-oriented programming languages; it was developed by Sun Microsystems in 1994. It runs in Windows, Mac, UNIX, and Linux environments and has an advantage over other environments in developing network applications. The basic development kits and several immediate development environments (IDE) are available for free.

Java Advanced Imaging (JAI)

Java Advanced Imaging (JAI) is one of the abstract programming interfaces (APIs) for developing image applications. It is also available for free from Sun Microsystems. This API supports various important functions for image processing including file I/O, filtering, and rendering [21].

Multispectral TIFF format images

In this project, multispectral images of TIFF format are used. Digital color images usually have 3 bands, red, green, and blue. but in order to identify the healthy vegetation such as trees and grasses, an infrared band is required as well. The TIFF image format is commonly used in remote sensing and photogrammetry, instead of using the other image formats such as JPG, BMP, and GIF, because it can handle four bands for red, green, blue, and infrared.

UTM map projection

Since the earth is not flat, there are many different map projections. In this project, the Universal Transverse Mercator (UTM) projection is used. It is one of the Gauss-Krüger (ellipsoidal transversal Mercator) projections. The formula to calculate the projection is explained in Chapter 5. First, the meridian arc length is calculated from the latitude φ by using an equidistant latitude function (5.1). Then, the *X*-*Y* coordinate in ellipsoidal cylindrical conformal projection is calculated. A semi-major axis value of 6,378,137 meters and a flattening rate of 1/298.257222101 are used, according to the GRS80 definition [22].

4.11 Implementation schedule

As shown in Figure 4.1, the development is broken into 3 stages. This shows the schedule of development for each stage. Figure 4.2 shows the schedule submitted in the proposal.

Stage 1: Transmission line scanning program
Goal: Develop a tool to extract the healthy vegetation area close to the power lines from multispectral satellite images: Processes 1, 2, 3, 4, 5, 6, 9.

Stage 2: Tall Tree Identification Program
Goal: Develop a tool to generate digital surface models from a stereo pair of multispectral satellite images: Processes 7 and 8.

Stage 3: Tall Tree Identification Program
Goal: Integrate the tools of Stages 1 and 2, and develop a system to identify high trees interfering with overhead power lines: Process 10.

5.0 Transmission line scanning program: Stage 1

5.1 Introduction

This section shows several techniques used in implementing the tool in Java for Stage 1 described in Chapter 4.1. In Stage1, the goal is to develop a tool utilizing the following functions:

- Loading image files
- Displaying the danger zone along power lines on an image
- Identifying the healthy vegetation area

Each function is explained in Sections 5.3, 5.4, and 5.5, respectively.

5.2 Packages and classes

The all Java classes are implemented in a package named "pserc." The following is the list of classes developed for Stage 1:

- PL_ScrollController
- PL_DataTable
- PL_FileLoader
- PL_FilterSlider
- PL_Geo
- PL_ImageCanvas
- PL_MainFrame
- PL_RenderedImageCanvas
- PL_RenderGrid
- PL_ScrollGUI

PL_MainFrame is a main window frame class, and its main function is to execute this program. The GUI of the *PL_MainFrame* has two main parts: the Left Panel and Right Panel.

The Left Panel has three small panels, the *PL_DataTable*, *PL_FilterSlider*, and *PL_RenderGrid*. The *PL_DataTable* is the top panel and shows the geographical information of positions including the corners of a loaded image and transmission towers. The *PL_FilterSlider* is the bottom panel and has 5 sliders for red, green, blue, infrared bands, and NDVI value. The *PL_RenderGrid* is the middle panel, and it shows the visible area in the main image panel, the Right Panel which was introduced in the previous paragraph. It also shows the all towers and lines.

PL_RenderedImageCanvas is the main image panel to show an image in the Right Panel. It extends the *PL_ImageCanvas*, which is the fundamental image panel object.

PL_FileLoader is a class to be used in loading image files. *PL_Geo* is a class to get the geographical information from the table in the *PL_DataTable* object, and it defines the parameters for converting the information of latitude and longitude to a map projection coordinate, Universal Transverse Mercator (UTM).

PL_ScrollController is a Java interface to control scroll events from mouse actions, and the *PL_ScrollGUI* implementing the *PL_ScrollControlle* is used in the *PL_RenderedImageCanvas* to scroll an image.

5.3 Loading image files

Because Java Advanced Imaging (JAI) API supports most of the image formats, the image object is initialized as a planar image object as shown in the PL_FileLoader.

PlanarImage pi = JAI.create("fileload", file.getAbsolutePath())

The most important concern in loading image files is to support a large size image file because this project will require handling a gigabyte image file.

```
/**
   * Make tiled image from image object.
   * @param img PlanarImage input mutlispectral Tiff image.
   * @return RenderedOp 4 bands tiled image.
   */
  protected RenderedOp makeTiledImage(PlanarImage img) {
   ImageLayout tileLayout = new ImageLayout(img);   // set the layout
   tileLayout.setTileWidth(tileWidth);              // set tile width and height
   tileLayout.setTileHeight(tileHeight);
   RenderingHints tileHints = new RenderingH-
ints(JAI.KEY_IMAGE_LAYOUT, tileLayout);
   ParameterBlock pb = new ParameterBlock();
   pb.addSource(img);
   RenderedOp op=JAI.create("format", pb, tileHints); //generate RenderedOp
object with tile info
   return JAI.create("BandSelect", (PlanarImage)op, new int[] {0,1,2,3});
  }
```

Figure 5.1 Java code for making a tiled Image

In order to support such a large imagine file, tiles of a tiled image are drawn one by one instead of drawing the whole image at one time. The following is the function in the PL_RenderedImageCanvas to make a tiled image from a PlanarImage object.

As described in Section 4.2.2, this program accepts the image only with 4 channels. Otherwise, the program fails to make a tiled image. After making a tiled image, each tile is drawn in a paint function. This procedure is explained in Section 5.4.

5.4 Displaying the danger zone area along overhead transmission lines on an image

An image file does not have any geographical information. However, the location information of transmission towers uses geographical coordinates, latitude and longitude. Therefore, some data to define the geographical information of an image is required in advance. In this stage, the user must get the coordinates of three points (left-top, right-top, and left-bottom corners) in the image. The information is obtained from Google Earth [19] and it is hard-coded in PL_Geo class as default. The formula (5.2) is used to convert the coordinates from latitude and longitude to *X*-*Y* in Universal Transverse Mercator (UTM). The formula (5.1) is utilized to get *S*, the meridian arc length, and the arc length is subsequently used in the formula (5.2). The details of the formulas are explained in [22]. Figure 5.2 and Figure 5.3 show the Java codes for implementing the formulas (5.1) and (5.2) respectively.

$$
\left.
\begin{aligned}
&S = a(1-e^2)(A^*\varphi - \frac{B}{2}\sin 2\varphi + \frac{C}{4}\sin 4\varphi - \frac{D}{6}\sin 6\varphi + \frac{E}{8}\sin 8\varphi - \frac{F}{10}\sin 10\varphi + \frac{G}{12}\sin 12\varphi - \frac{H}{14}\sin 14\varphi + \frac{I}{16}\sin 16\varphi \\
&A = 1 + \frac{3}{4}e^2 + \frac{45}{64}e^4 + \frac{175}{256}e^6 + \frac{11025}{16384}e^8 + \frac{43659}{65536}e^{10} + \frac{693693}{1048576}e^{12} + \frac{19324305}{29360128}e^{14} + \frac{4927697775}{7516192768}e^{16} \\
&B = \frac{3}{4}e^2 + \frac{15}{16}e^4 + \frac{525}{512}e^6 + \frac{2205}{2048}e^8 + \frac{72765}{65536}e^{10} + \frac{297297}{262144}e^{12} + \frac{135270135}{117440512}e^{14} + \frac{547521975}{469762048}e^{16} \\
&C = \frac{15}{16}e^4 + \frac{105}{256}e^6 + \frac{2205}{4096}e^8 + \frac{10395}{16384}e^{10} + \frac{1486485}{2097152}e^{12} + \frac{45090045}{58720256}e^{14} + \frac{766530765}{939524096}e^{16} \\
&D = \frac{35}{512}e^6 + \frac{315}{2048}e^8 + \frac{31185}{131072}e^{10} + \frac{165165}{524288}e^{12} + \frac{45090045}{117440512}e^{14} + \frac{209053845}{469762048}e^{16} \\
&E = \frac{315}{16384}e^8 + \frac{3465}{65536}e^{10} + \frac{99099}{1048576}e^{12} + \frac{4099095}{29360128}e^{14} + \frac{348423075}{1879048192}e^{16} \\
&F = \frac{693}{131072}e^{10} + \frac{9009}{524288}e^{12} + \frac{4099095}{117440512}e^{14} + \frac{26801775}{469762048}e^{16} \\
&G = \frac{3003}{2097152}e^{12} + \frac{315315}{58720256}e^{14} + \frac{11486475}{939524096}e^{16} \\
&H = \frac{45045}{117440512}e^{14} + \frac{765765}{469762048}e^{16} \\
&I = \frac{765765}{7516192768}e^{16} \\
&\text{where } a = \text{semimajor axis (6378137m)}, \varphi = \text{latitude}, \\
&e = \text{first eccentricity defined as } \sqrt{2f - f^2},\ f = \text{flattening rate (1/298.257222101).}
\end{aligned}
\right\} \quad (5.1)
$$

$$
\left.
\begin{aligned}
y &= S + \frac{1}{2} N \cdot t \cdot \lambda^2 \cdot \cos^2 \varphi + \frac{1}{24} N \cdot t \cdot (5 - t^2 + 9\eta^2 + 4\eta^4) \cdot \lambda^4 \cdot \cos^4 \varphi \\
&\quad + \frac{1}{720} N \cdot t \cdot (61 - 58t^2 + t^4 + 270\eta^2 - 330t^2\eta^2) \lambda^6 \cos^6 \varphi \\
&\quad + \frac{1}{40320} N \cdot t \cdot (1385 - 3111t^2 + 543t^4 - t^6) \lambda^8 \cos^8 \varphi \\
\\
x &= N \cdot \lambda \cdot \cos \varphi + \frac{1}{6} N \cdot (1 - t^2 + \eta^2) \cdot \lambda^3 \cdot \cos^3 \\
&\quad + \frac{1}{120} N (5 - 18t^2 + t^4 + 14\eta^2 - 58t^2\eta^2) \cdot \lambda^5 \cdot \cos^5 \varphi \\
&\quad + \frac{1}{5040} N (61 - 479t^2 + 179t^4 - t^6) \lambda^7 \cos^7 \varphi \\
&\text{, where } N = \frac{a}{\sqrt{1 - e^2 \sin^2 \varphi}}, e' = \text{second eccentricity defined as } \sqrt{\frac{e^2}{1 - e^2}}, \\
&\lambda = \text{longitude}, t = \tan \varphi, \eta^2 = e'^2 \cos^2 \varphi
\end{aligned}
\right\} \qquad (5.2)
$$

```
  /**
 * Get the meridian arc length by meter from latitude
 * @param lat double Latitude as radian value
 * @return double Meridian Length (meter)
 */
public static double getMeridian(double lat) {
  double a = 6378137.0;                  //semimajor axis of Earth from GRS80
  double f = 1.0 / 298.257222101;        //flattening rate of Earth
  double _F = 298.257222101;
  double e = (Math.sqrt(2.0 * _F - 1.0)) / _F;  // first eccentricity
  double e_ = (Math.sqrt(2.0 * _F - 1.0)) / (_F - 1.0); //second eccentricity

  //Step 1: Calcuate the A, B, C, D, E, F, G, H, I,
  //They are the constants multiplied with e for equidistant latitude function

  double A = 1.0 + (3.0 / 4.0) * Math.pow(e, 2) +
                                    (45.0 / 65.0) * Math.pow(e, 4) + (175.0 / 256.0) * Math.pow(e, 6) +
     (11025.0 / 16384.0) * Math.pow(e, 8) +
     (43659.0 / 65536.0) * Math.pow(e, 10) +
     (693693.0 / 1048576.0) * Math.pow(e, 12) +
     (19324305.0 / 29360128.0) * Math.pow(e, 14) +
     (4927697775.0 / 7516192768.0) * Math.pow(e, 16);
  double B = (3.0 / 4.0) * Math.pow(e, 2) + (15.0 / 16.0) * Math.pow(e, 4) +
     (525.0 / 512.0) * Math.pow(e, 6) +
     (2205.0 / 2048.0) * Math.pow(e, 8) +
     (72765.0 / 65536.0) * Math.pow(e, 10) +
     (297297.0 / 262144.0) * Math.pow(e, 12) +
     (135270135.0 / 117440512.0) * Math.pow(e, 14) +
     (547521975.0 / 469762048.0) * Math.pow(e, 16);
  double C = (15.0 / 64.0) * Math.pow(e, 4) + (105.0 / 256.0) * Math.pow(e, 6) +
     (2205.0 / 4096.0) * Math.pow(e, 8) +
     (10395.0 / 16384.0) * Math.pow(e, 10) +
     (1486485.0 / 2097152.0) * Math.pow(e, 12) +
     (45090045.0 / 58720256.0) * Math.pow(e, 14) +
     (766530765.0 / 939524096.0) * Math.pow(e, 16);
  double D = (35.0 / 512.0) * Math.pow(e, 6) +
     (315.0 / 2048.0) * Math.pow(e, 8) +
     (31185.0 / 131072.0) * Math.pow(e, 10) +
     (165165.0 / 524288.0) * Math.pow(e, 12) +
     (45090045.0 / 117440512.0) * Math.pow(e, 14) +
     (209053845.0 / 469762048.0) * Math.pow(e, 16);
  double E = (315.0 / 16384.0) * Math.pow(e, 8) +
     (3465.0 / 65536.0) * Math.pow(e, 10) +
     (99099.0 / 1048576) * Math.pow(e, 12) +
     (4099095.0 / 29360128.0) * Math.pow(e, 14) +
     (348423075.0 / 1879048192.0) * Math.pow(e, 16);
  double F = (693.0 / 131072.0) * Math.pow(e, 10) +
     (9009.0 / 524288.0) * Math.pow(e, 12) +
     (4099095.0 / 11744052.0) * Math.pow(e, 14) +
     (26801775.0 / 469762048.0) * Math.pow(e, 16);
  double G = (3003.0 / 2097152.0) * Math.pow(e, 12) +
     (315315.0 / 58720256.0) * Math.pow(e, 14) +
     (11486475.0 / 939524096.0) * Math.pow(e, 16);
  double H = (45045.0 / 117440512.0) * Math.pow(e, 14) +
     (765765.0 / 469762048.0) * Math.pow(e, 16);
  double I = (765765.0 / 7516192768.0) * Math.pow(e, 16);

  //Get the valued of all stages of the partial differential coefficients with
  //respect to e.
  double b = a * (1.0 - e * e);
  double B1 = b * A;
  double B2 = b * ( -B / 2.0);
  double B3 = b * (C / 4.0);
  double B4 = b * ( -D / 6.0);
  double B5 = b * (E / 8.0);
  double B6 = b * ( -F / 10.0);
  double B7 = b * (G / 12.0);
  double B8 = b * ( -H / 14.0);
```

```
    double B9 = b * (I / 16.0);

    double d = lat; // latitude

    //equidistant latitude function
    double m = B1 * d + B2 * Math.sin(2.0 * d) + B3 * Math.sin(4.0 * d) +
        B4 * Math.sin(6.0 * d) + B5 * Math.sin(8.0 * d) +
        B6 * Math.sin(10.0 * d) + B7 * Math.sin(12.0 * d) +
        B8 * Math.sin(14.0 * d) + B9 * Math.sin(16.0 * d);

    return m;
    }
```

Figure 5.2 Java Code for calculating the meridian arc length from latitude using formula (5.1)

```
/**
  * Calculate the XY distnace from the original point to the target point
  *
  * @param latS String Original point's latitude as DDMMSS.SS
  * @param lonS String origianl point's longitude as DDMMSS.SS
  * @param _latS String Target point's latitude as DDMMSS.SS
  * @param _lonS String Target point's latitude as DDMMSS.SS
  * @return double[] (double[0], double[1]) represents (X distance, Y distance) in UTM
  */
 public static double[] getXY(String latS, String lonS, String _latS, String _lonS){
  double lat=getRadian(latS);      //convert latitide as Radian
  double lon=getRadian(lonS);      //convert longitude as Radian
  double _lat=getRadian(_latS);    //convert target latitude as Radian
  double _lon=getRadian(_lonS);    //convert target longitude as Radian
  double a=6378137.0;              //semimajor axis of Earth from GRS80
  double f= 1.0/298.257222101;    //flattening rate of Earth
  double _F=298.257222101;
  double e=(Math.sqrt(2.0*_F -1.0)) / _F;         // first eccentricity
  double e_=(Math.sqrt(2.0*_F - 1.0)) / (_F-1.0); // second eccentricity

  double t=Math.tan(lat);
  double delta_lon=lon-_lon;       // offset degree in longitude
  double mu2=e_*e_*Math.cos(lat)*Math.cos(lat);
  double N=a/(Math.sqrt(1.0-e*e*Math.sin(lat)*Math.sin(lat)));
  double m0=0.9999;                //UTM factor from Gauss-Kruger projection
  double cos=Math.cos(lat);
  double[] xy=new double[2];

  //Get the distance in Y direction using Gauss-Kruger projection
  xy[1] = - ( (getMeridian(lat) - getMeridian(_lat))
            +
            1.0 / 2.0 * N * (Math.pow(cos, 2)) * t * (delta_lon * delta_lon)
            +
            1.0 / 24.0 * N * (Math.pow(cos, 4)) * t *
            (5.0 - t * t + 9 * mu2 + 4.0 * mu2 * mu2) *
            Math.pow(delta_lon, 4)
            -
            1.0 / 720.0 * N * (Math.pow(cos, 6)) * t *
            ( -61.0 + 58.0 * t * t - t * t * t * t - 270.0 * mu2 +
             330.0 * t * t * mu2) * Math.pow(delta_lon, 6)
            -
            1.0 / 40320.0 * N * (Math.pow(cos, 8)) * t *
            ( -1385.0 + 311.0 * t * t - 543.0 * t * t * t * t +
             Math.pow(t, 6)) * Math.pow(delta_lon, 8)) * m0;

  //Get the distance in X direction using Gauss-Kruger projection
  xy[0] = (N * cos * delta_lon
          -
          1.0 / 6.0 * N * (Math.pow(cos, 3)) * ( -1.0 + t * t - mu2) *
          Math.pow(delta_lon, 3)
          -
          1.0 / 120.0 * N * (Math.pow(cos, 5)) *
          ( -5.0 + 18.0 * t * t - t * t * t * t - 14.0 * mu2 +
           58.0 * t * t * mu2) * Math.pow(delta_lon, 5)
          -
          1.0 / 5040.0 * N * (Math.pow(cos, 7)) *
          ( -61.0 + 479.0 * t * t - 179.0 * Math.pow(t, 4) + Math.pow(t, 6)) *
          Math.pow(delta_lon, 7)) * m0;

  return xy; // XY distnace by meter
 }
```

Figure 5.3 Java code for calculating the X and Y distance between 2 points from latitude and longitude using formula (5.2)

The formula (5.1) calculates the X and Y distances (meter) from the information on latitude and longitude. Therefore, after the coordinates of 3 points (left-top, left-bottom, and right-top) of an image are defined, any point represented with latitude and longitude can be defined as a position in the image.

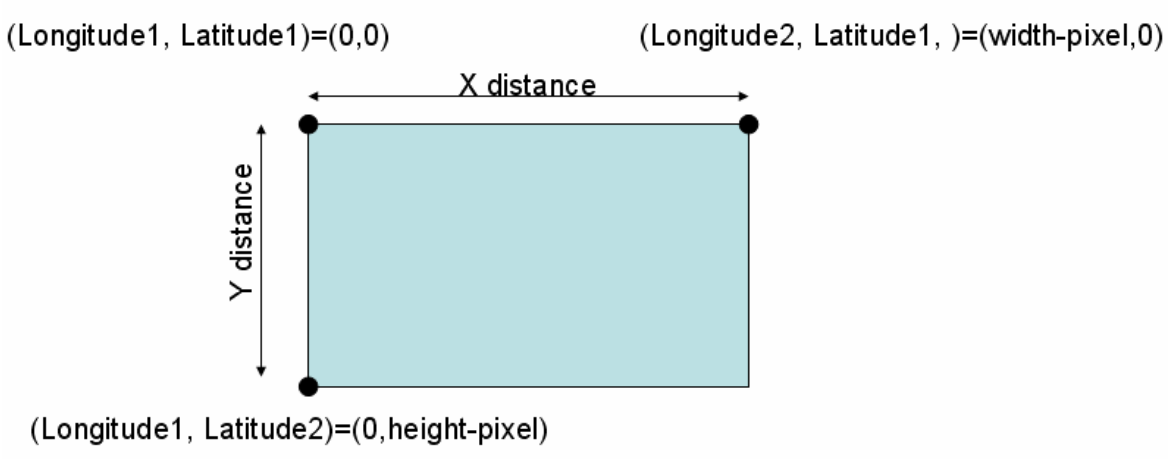


Figure 5.4 Relation of X-Y coordinates and geo-coordinates

For example, if a transmission tower has a coordinate (*Lo*, *La*), then the distance from the left top corner of the image can be calculated using the function in the previous section. The distance is supposed as (*X'*, *Y'*). The position of tower (*x', y')* is defined as:

x'= X'/(X/width_pixel): *X/width_pixel is kept as EW_scale*
y'=Y'/(Y/ height_pixel): *Y/height_pixel is kept as NS_scale*

Figure 5.5 shows the Java code to get the *X-Y* position in the image from latitude and longitude.

```
/**
   * Get the XY pixel position from latitude and longitude.
   * @param _lat String latitude should be represented as DDMMSS.SSS
   * @param _lon String longitude should be represented as DDMMSS.SSS.
   * @return Point Pixel Position in image.
   */
  public static Point pixelPos(String _lat, String _lon){
    double[] xy=getXY(latitude, longitude, _lat, _lon); //get UTM position
    int xpos = (int)(xy[0]/EW_scale);     //convert UTM-xy to X image position
    int ypos = (int)(xy[1]/NS_scale);     //convert UTM-xy to Y image position
    return new Point(xpos, ypos);
  }
```

Figure 5.5 Java code to get XY pixel position in an image from latitude and longitude

In this project, any image is assumed as orth-rectified. In other words, it is assumed that two positions with the same *y*-value have the same latitude, and two points with the same *x*-value have the same longitude. In addition, the latitude or longitude degree in this program is repre-

sented as a string object. The degree, minute and second values, is input as one word. For example, if the degree, minute, and second values are “33,” “45,” and “49.876,” then it is represented as “334549.876.” Figure 5.6 is a function to convert the degree from degree-minutes-second notation to radians.

```
/**
   * Get the radian value from degree as DDMMSS.SSS.
   * @param _d_m_s String input degree.
   * @return double radian value.
   */
  public static double getRadian(String _d_m_s){
   double temp=Double.parseDouble(_d_m_s);
   int _d= (int)(temp/10000.0);   //get Degree value as integer
   temp=temp- ((double)_d*10000.0);
   int _m= (int) (temp/100.0);     //get Minute value as integer
   double _s=temp- ((double)_m*100.0); // get Second value as real number
   return ( (double)_d + (double)_m/60.0 + _s/3600.0 ) *Math.PI/180.0; //convert the degree to ra-
dian
  }
```

Figure 5.6 Java code to convert the value from degree minute-second-format to Radians

5.5 Identifying healthy vegetation areas

For extracting the healthy vegetation area in multispectral images, *NDVI* is used as described in Chapter 4.

$$NDVI = (NIR - R) / (NIR + R). \quad (5.1)$$

NDVI represents the intensity of the signal in the spectrum corresponding to the amount of healthy vegetation present in the specific pixel. The notation NIR is the near infrared signal, and *R* is the red signal. In order to support a large image file, the tiled image is created as explained in Section 5.2. Figure 5.7 shows a sub-block defined in a paint function in a PL_RenderedImage -Canvas. It is best to draw tiles one by one. The tile size is 256 *x* 256. The integer array object, “pixels[],” store the pixel information of a tile. It has 4 bands. The byte array object, “data[],” get the NDVI value, and the green, and blue values from “pixels[].” During the process, if the NDVI value is more than the threshold value defined in the PL_FilterSlider Panel, the pixel is set as white. After getting “data[],” a new buffered image object is initialized using the “data[].” Then, the BufferedImage is drawn in the proper position.

```
// Draw each tile
 for(tj = topIndex; tj <= bottomIndex; tj++) {
  for (ti = leftIndex; ti <= rightIndex; ti++) {

    Raster raster = displayImage.getTile(ti, tj);  // Get the raster information
    int width2 = raster.getWidth();               // get raster's width
    int height2 = raster.getHeight();             // get raster's height
    int bands = raster.getNumBands();             // get raster's band number (Supposed 4)
    int[] pixels = new int[width2 * height2 * bands]; // pixel information of tile image
    byte[] data = new byte[width2 * height2 * 3];  // target pixel data

    raster.getPixels(ti * 256, tj * 256, width2, height2, pixels); // 256 x 256 is the tile size FIXED!!!!

    int index = 0;
    // For each pixel in a tile
    for (int h = 0; h < height2; h++)
      for (int w = 0; w < width2; w++) {
        int red = pixels[index * 4 + 2];    // Get Red band pixel info
        int inf = pixels[index * 4 + 3];    // Get Infrared band pixel info
        float ndvi = (float) ( ( (float) (inf - red)) / ( (float) (red + inf))); // Cacluate NDVI
        data[index * 3 + 2] = (byte) ( (ndvi + 1.0) * 100.0); //Set the NDVI  in Red band
        data[index * 3 + 1] = (byte) pixels[index * 4 + 1];   //Set the original Green in green band
        data[index * 3 + 0] = (byte) pixels[index * 4];       //Set the original Blue in blue band

        int thres=(int)(PL_FilterSlider.threshold*100.0)+100; // Get NDVI Theashold value from PL_FilerSlider
        if(data[index*3+2]>thres){ // If the NDVI value is more than Thesdhold, the pixel is WIHTE.
          data[index * 3 + 2] = (byte) 255; //red
          data[index * 3 + 1] = (byte) 255; //green
          data[index * 3 + 0] = (byte) 255; //blue
        }
        index++;
      }

    // *** ROI Operation *** //
    DataBufferByte dbuffer = new DataBufferByte(data, width2 * height2 * 3);
    //Get SampleModel
    sampleModel = RasterFactory.
       createPixelInterleavedSampleModel(DataBuffer.TYPE_BYTE, width2, height2, 3);
    //Get ColorModel
    colorModel = PlanarImage.createColorModel(sampleModel);

    //Create Buffered image and draw it in the proper tile position
    WritableRaster wr = RasterFactory.createWritableRaster(sampleModel,dbuffer, new Point(0, 0));
    BufferedImage bi = new BufferedImage(colorModel, wr, colorModel.isAlphaPremultiplied(), null);
    int xInTile = displayImage.tileXToX(ti);
    int yInTile = displayImage.tileYToY(tj);
    AffineTransform tx = AffineTransform.getTranslateInstance(xInTile + panX, yInTile + panY);
    g.drawRenderedImage(bi, tx);
   }
 }
```

Figure 5.7 Java code to generate a tile with NDVI, green, and blue bands

6.0 Tool Instruction: Stage 1 for NDVI Visualization

6.1 Introduction

This chapter presents the operation instructions of the developed tool. The intent is to serve as explanatory material in connection with a users' manual which appears as an appendix.

6.2 Setting Java environments

We use a JBuilder X, which is one of the Java immediate developing environment (IDE) packages, and it is available from [24]
http://www.borland.com/downloads/download_jbuilder.html.

In order to run our program, the additional API of Java Advanced Imaging (JAI) is required. JAI is downloadable from
http://java.sun.com/products/java-media/jai/index.jsp

After installing JBuilderX and JAI (1.1.2), the class paths to "jai_codec.jar" and "jai_core.jar" are required. The following image shows the class-path setting in JBuilderX.

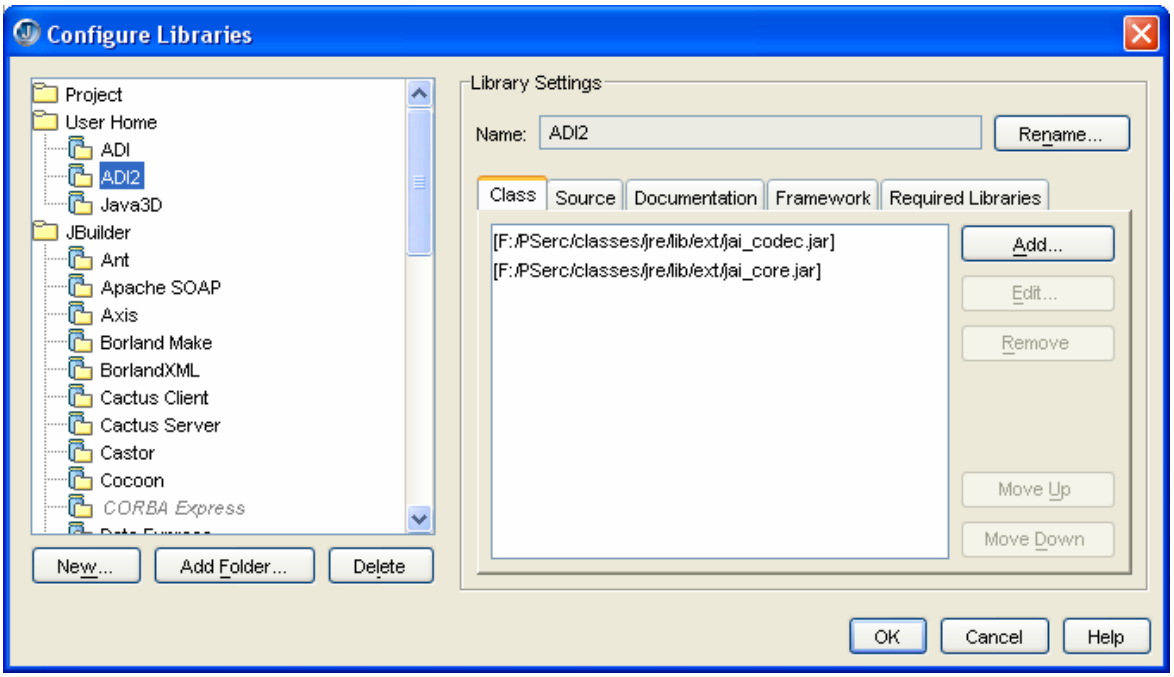


Figure 6.1 CLASSPATH setting in JBUILDER

The last step is to open the "jbuilder.config" file in JBuilder/bin directory and change the amount of memory allocation in order to handle big image files. The following case sets the base starting memory pool as 128 megabytes, and allows for a maximum of 512 megabytes.

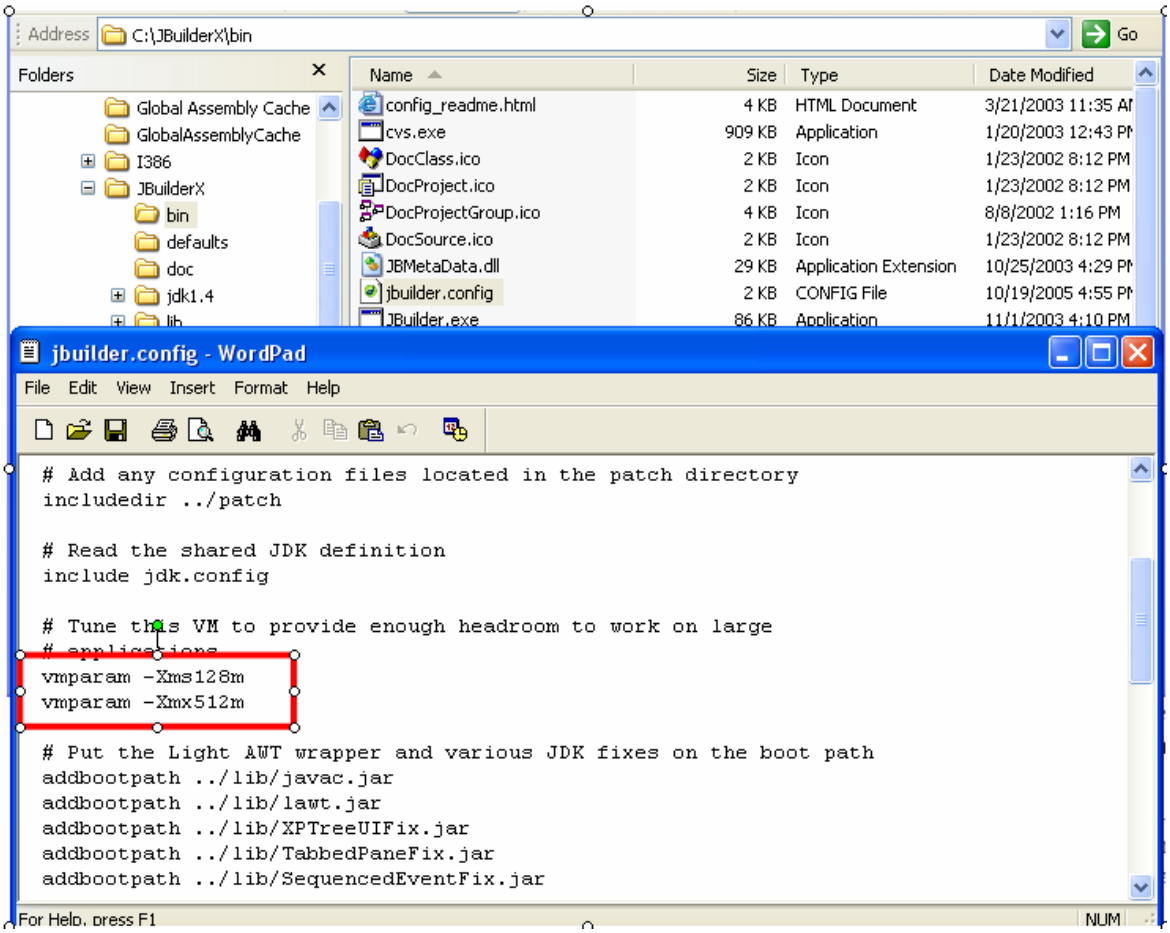


Figure 6.2 Screen shot for memory assignment in JBUILDER

6.3 How to run the program

By opening the project file, “PSerc.jpx” in the PSerc folder and selecting the “Run Project” command, the program starts. Figure 6.3 displays the results.

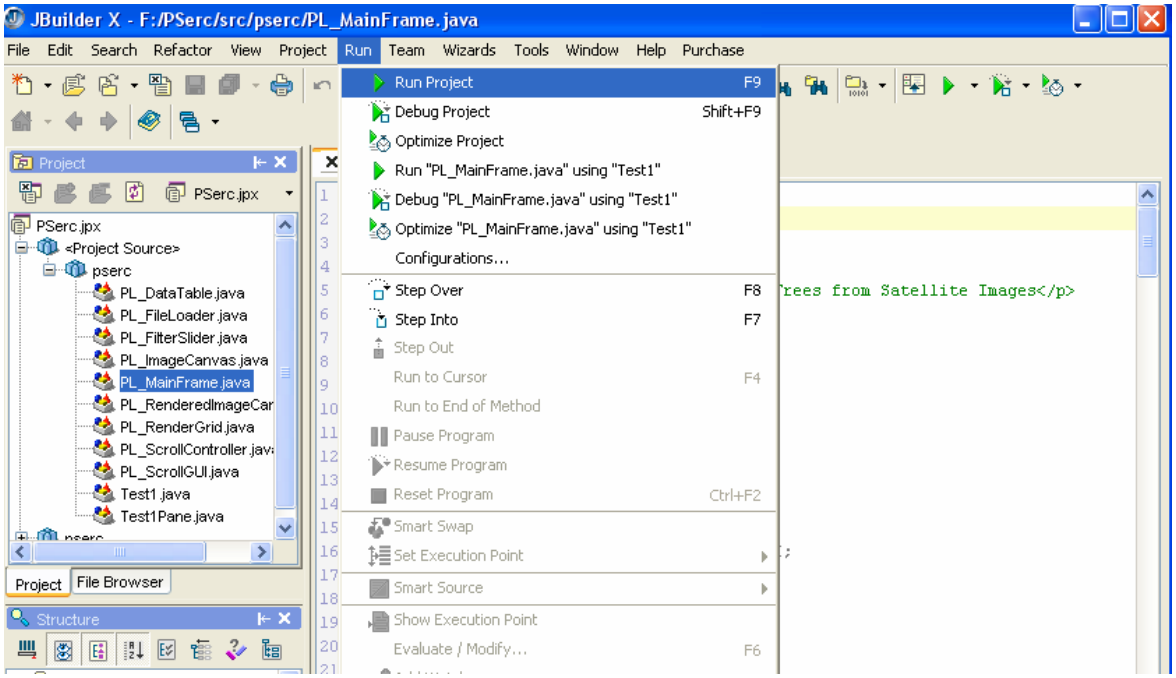


Figure 6.3 Screen shot for Running Projects in JBUILDER

6.4 Loading an image

This program can read only TIFF formatted image files with four bands. The band order is blue-green-red-infrared. By selecting the “Open Image” menu item under the pull down menu “File” -> “Open,” the file loader window pops up. Select the file name in the window then the image is displayed in the main screen. Figures 6.4 and 6.5 show the process.

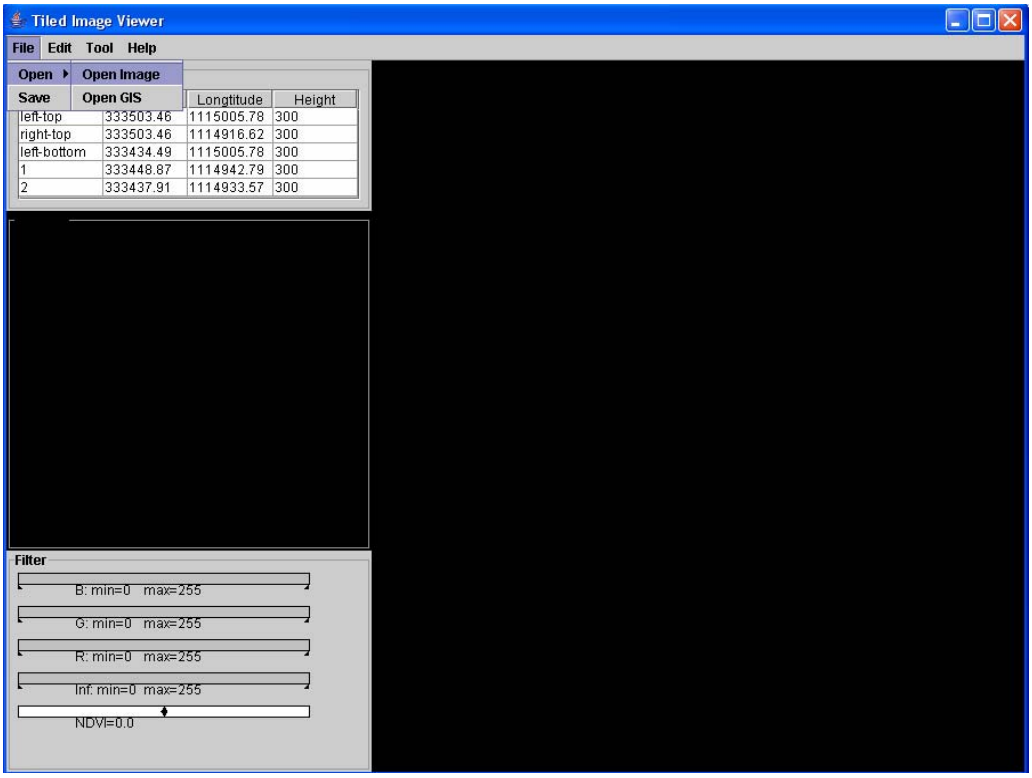


Figure 6.4 Screen shot of the GUI

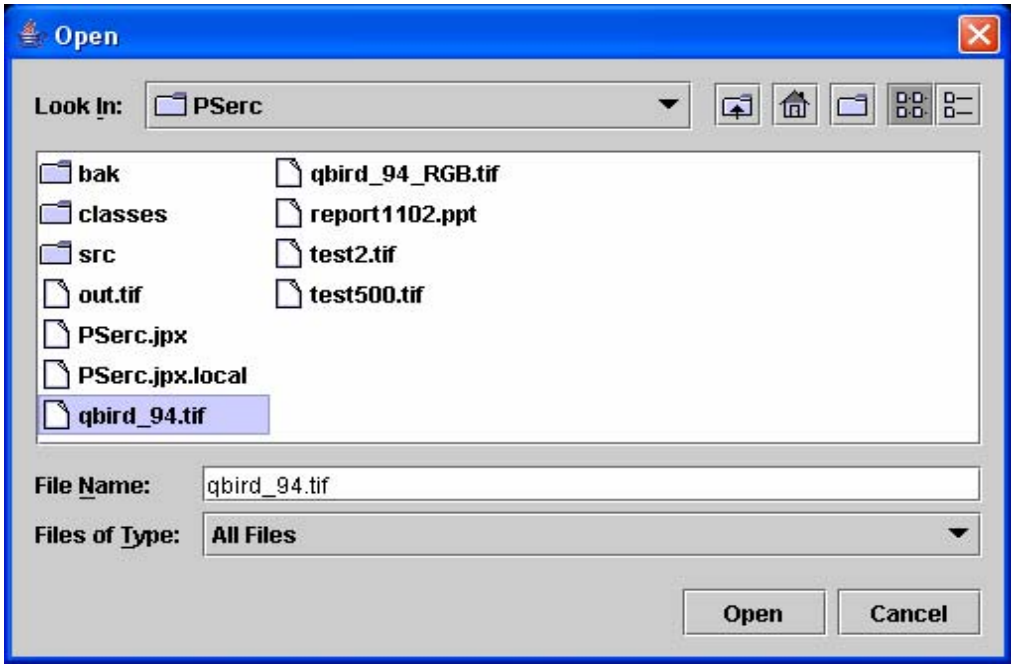


Figure 6.5 Screen shot of the image file loader

Before loading an image, some information should be obtained. TIFF images usually do not have geographical information. However, the location of the transmission tower is repre-

sented with geo-coordinates using latitude and longitude. In order to specify the location and scale of an image, the latitude, and longitude of three corner points (left top, right top, and left bottom points) should be defined. The data can be obtained by using Google Earth [19]. The latitude and longitude information is shown in the bottom status bar.

Figure 6.6 QuickBird satellite image, Scottsdale, AZ

With reference to Figures 6.6 and 6.7, the following is the information on an image used in a test case. The <qbird_94.tif> file is a Quickbird satellite image taken in 1994 over Scottsdale, Arizona. In this test case, the location data of towers is hard-coded.

Left-top:	33º35’ 03.46” N	111º50’05.78”W
Right-top:	33º35’ 03.46” N	111º49’16.62”W
Left-bottom:	33º34’ 34.49” N	111º50’05.78”W

Image size = 2108 *x* 1489 pixels

The power line is depicted from (33º34’ 48.87” N, 111º49’42.79”W) to (33º34’ 37.91” N, 111º49’33.57”W).

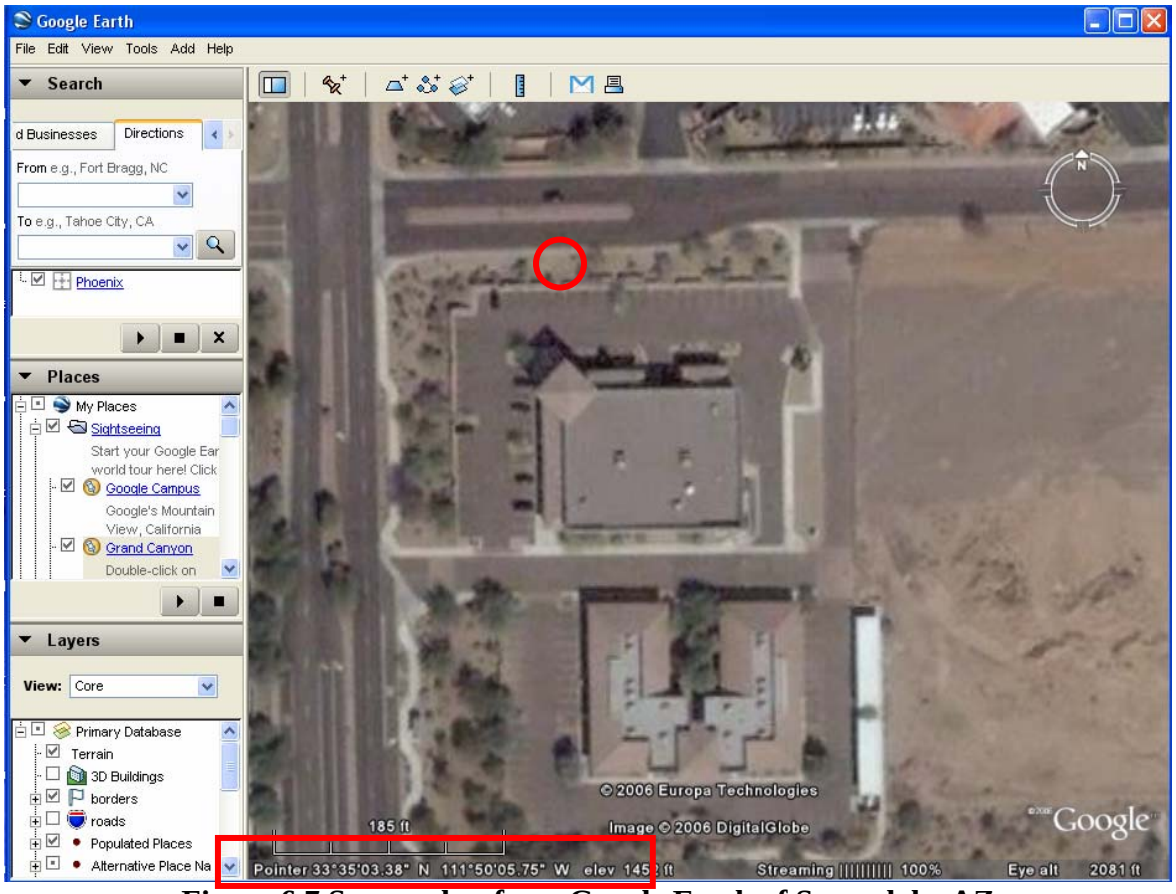


Figure 6.7 Screen shot from Google Earth of Scottsdale, AZ

The red circle shows the left top corner of the image used for testing. The red rectangle shows the latitude and longitude of the position of the red circle.

6.5 Results of a test case in Scottsdale AZ

Once the image is loaded, the boundaries of the envelop area near the power lines are shown in the main panel, which is the panel on the right. The left panel has three components, the geo-coordinate information panel, the grid panel, and the NDVI control panel. Figure 6.8 shows a screen capture of the process.

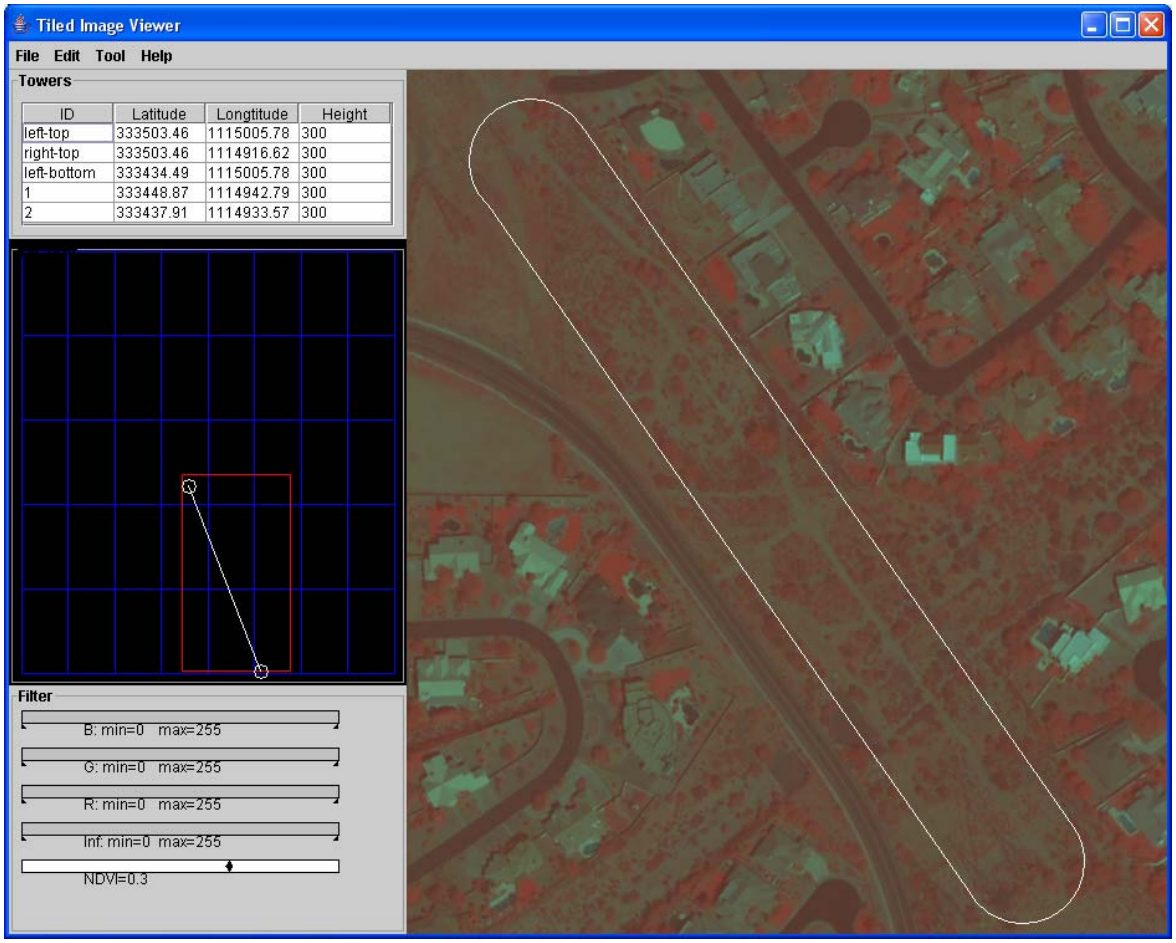


Figure 6.8 Screen shot of the result showing a power line between two transmission towers

PL_DataTabel Panel

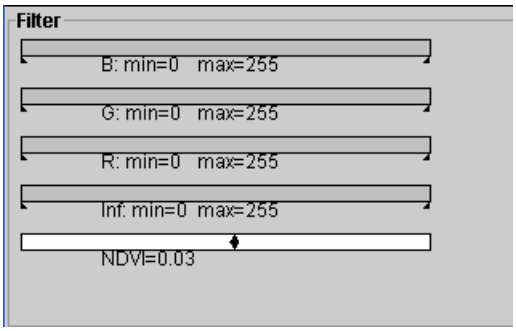
This panel shows the geographical coordinates of the three corner points of the image and the towers. The first three rows are used for the corner points, and the other rows are used for transmission towers. Though the information is hard-coded, the interactive functions allowing the user to add and edit the data will be implemented in the next stage.

PL_RenderGrid Panel

The RenderGrid panel shows the locations of transmission towers and power lines in the image. The red rectangle boundary shows the displaying area in the main right panel.

PL_FilterSlide Panel

This panel has an interactive function to change the threshold values of NDVI. By dragging the mouse close to the black diamond in the NDVI slide bar, the threshold value is updated. The following images show the cases using NDVI=0.03 and NDVI=0.12. The user can investigate the healthy vegetation area by sliding the bar. In this test case, a tree in the middle inside the envelope is considered as a healthy tree as shown in Figure 6.10.



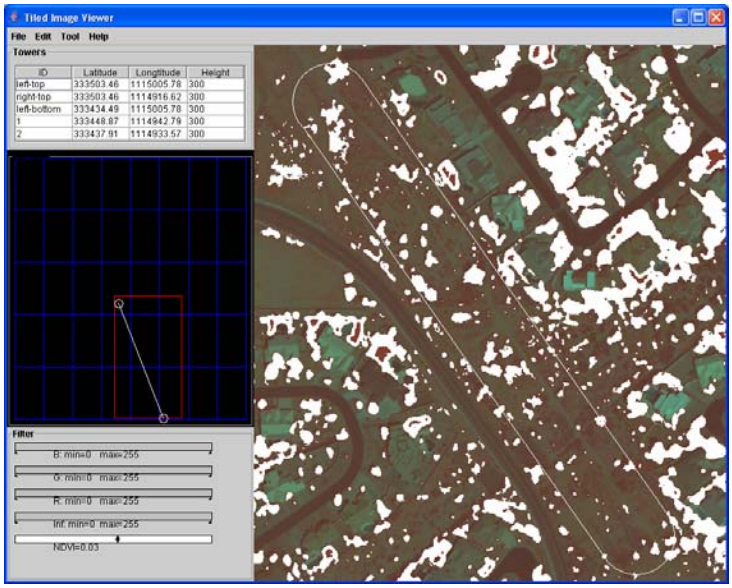
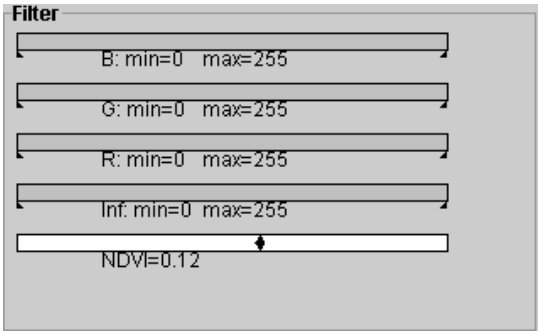


Figure 6.9 Results of the case with a threshold value of NDVI as 0.03



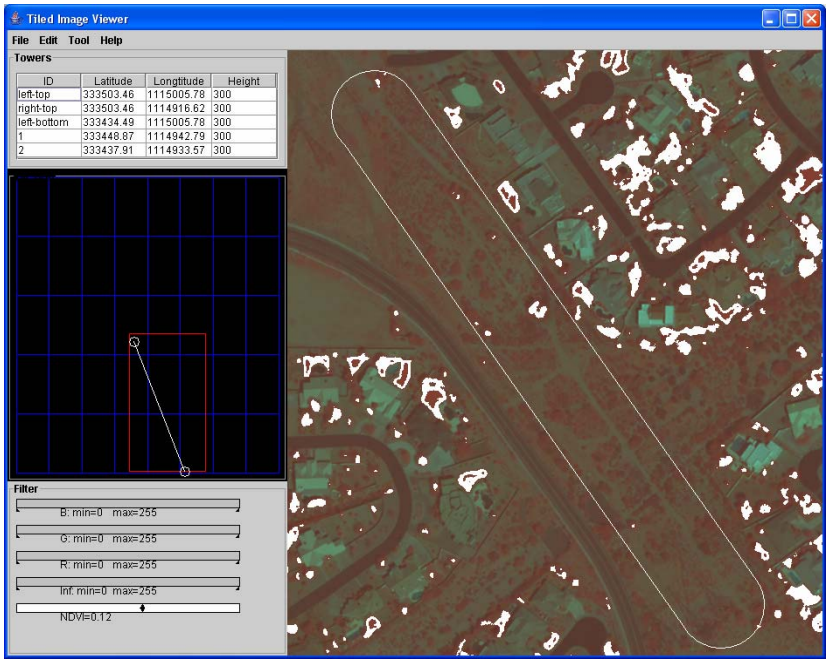


Figure 6.10 Results of the case with a threshold value of NDVI as 0.12

7.0 Digital Surface Model: Pre-Stage 2

7.1 The digital surface model generated in ERDAS IMAGNE

Shown here is the process and results of the first trial of generating a DSM (Digital Surface Model), using 1-meter IKONS stereo images. This is to estimate the quality of DSM using 1-meter 4-bands 8-bits color Epipolar stereo images downloaded from [4]. The stereo images are loaded in the software package for photogrammetry, ERDAS IMAGINE [23]. In order to match the images, additional *X, Y,* and *Z* values are required. The XY data were obtained from Google Earth. By using *the X-Y* (latitude and longitude) data from Google Earth, the *Z* (altitude) is obtained from the elevation data in USGS. The package can generate the height data as a TIF image. Figure 7.1 shows a typical result.

Once the height data is generated, it can be visualized in the regular GIS package such as ArcGIS. The image does not have the regular 8 bit values between 0 and 255, but the height values between –28 and 78, in this case. Figures 7.2 and 7.3 show the visualization.

7.2 Illustration of vegetation extraction

Figures 7.4 and 7.5 illustrate the process of the extraction of vegetation. The regions circled are trees that are identified spectrally and rendered to a three dimensional view in Figure 7.5.

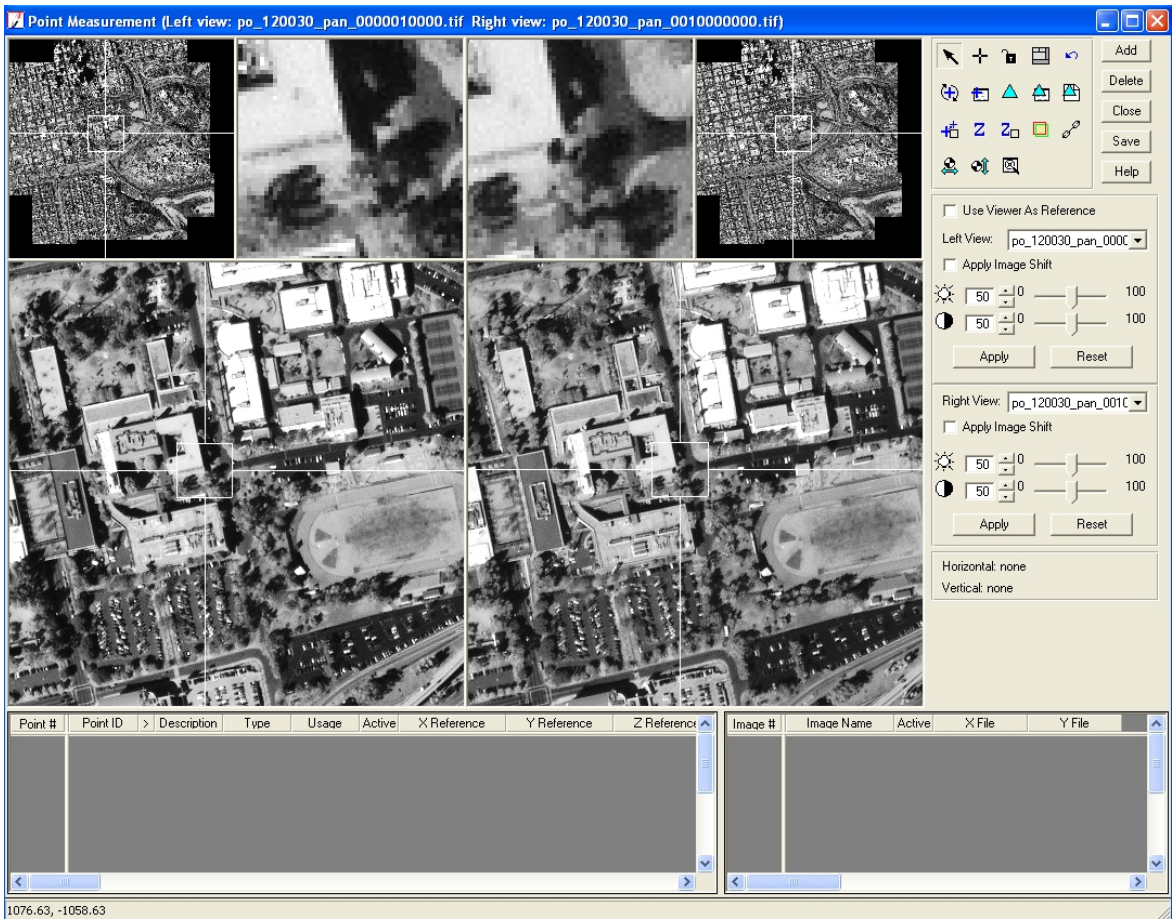


Figure 7.1 A DSM model using IKONOS data

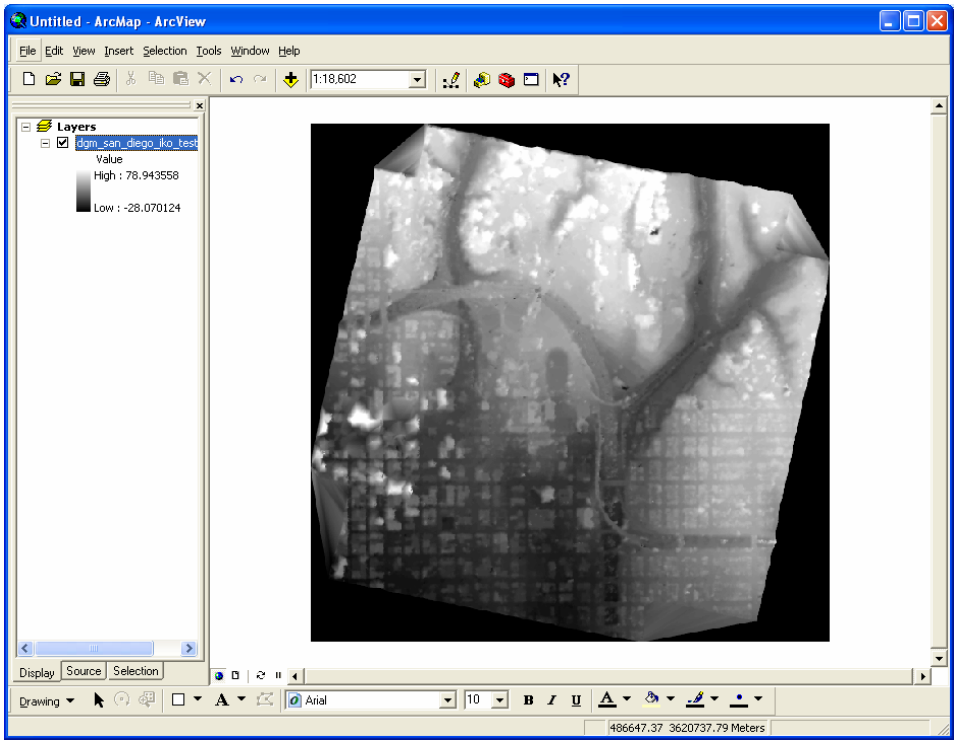


Figure 7.2 A visualization in a conventional GIS package, ArcGIS

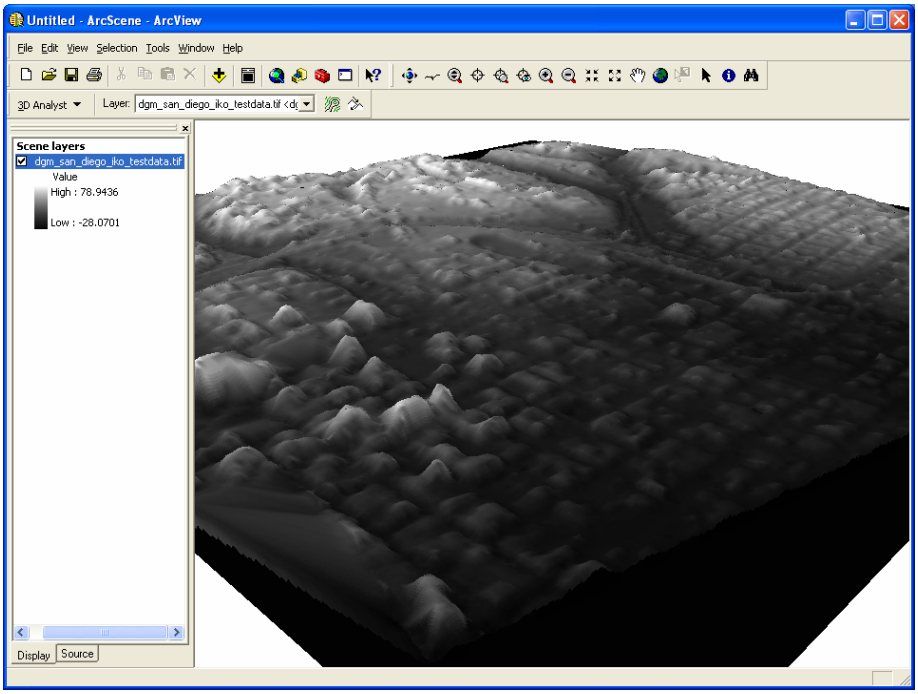


Figure 7.3 Visualization in a conventional GIS package, ArcGIS

Figure 7.4 Results of extracting vegetation using NDVI data

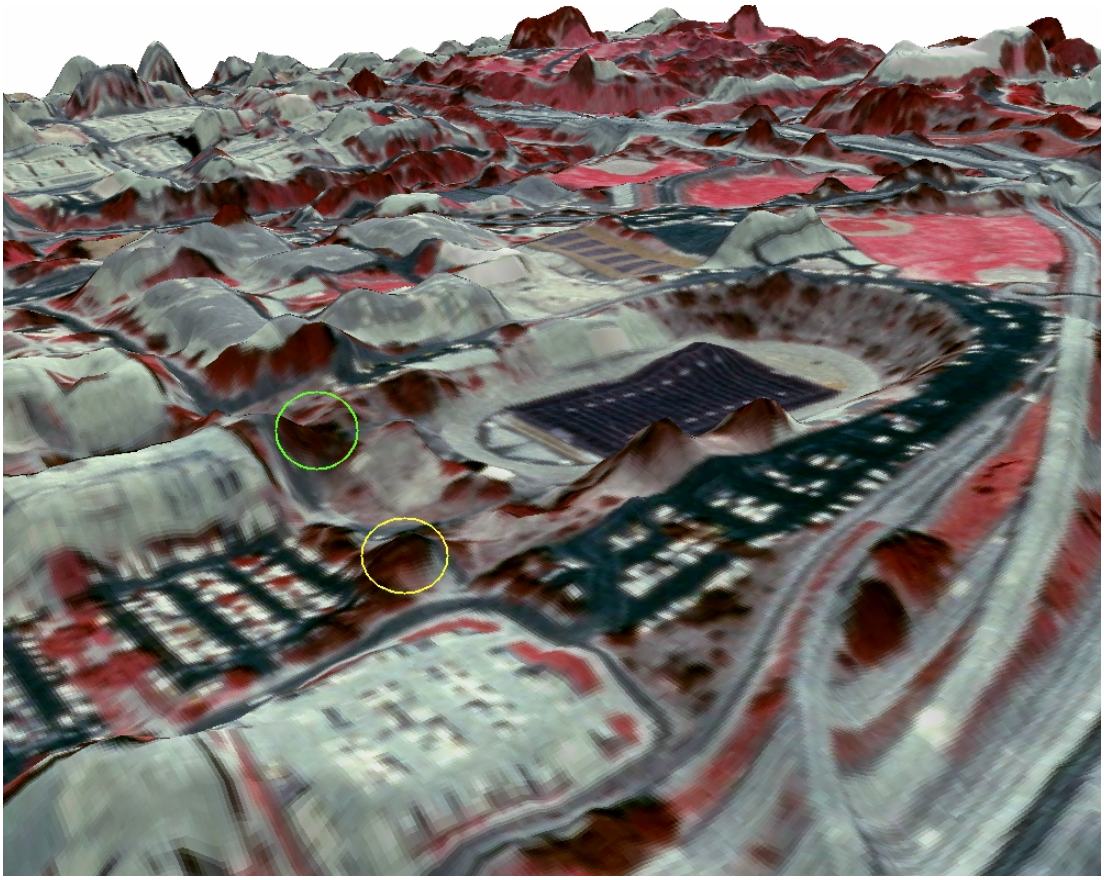


Figure 7.5 3D views of DSM with NDVI data

The more accurate the *X-Y-Z* coordinates used, the more accurate the DSM data which is generated.

8.0 Tool Implementation: Stage 2 for DEM data

8.1 Introduction

This section shows several techniques used in implementing the tool in Java for Stage 2 described in Chapter 4.11. In this stage, the goal is to develop a tool utilizing the following functions:

- Loading a pair of stereo images
- Getting matrices surrounding a pixel in the first image of the stereo pair
- Calculating a cross-correlation value between matrices of each pixel in the first and second images
- Find the maximum cross-correlation point for a pixel in the first image from a list of pixels of a corresponding horizontal line in the second image
- Calculating the distance between each pixel in the first image and the position with the maximum cross-correlation position in the second image
- Generating and saving a DEM image file with a set of the distance values for each pixel

The functions are explained in the following sections.

8.2 Packages and classes

The main class PL_StereoTest class is developed to generate a DEM image file in the same package, "pserc," as the package in Stage 1. Once a PL_StereoTest object is initialized, a DEM file is automatically saved as a TIFF formatted image file with 16-bit float values. The following are the global variables used in the class.

```
Container p;                          // Container of Main Frame
PlanarImage pi, pi2;                  // Stereo pair images
Raster inputRaster, inputRaster2;     // Raster data for stereo pair images
int imageWidth, imageHeight;          // Image size
int maxTileX, maxTileY;               // X and Y Tile maximum indices
RenderedOp op;                        // Output image
int filterSize=4;                     // filterSize 4 = 9x9 matrix for calculate correlation
int hightOffset=-45;                  // search left end (45 pixels)
int hightOffset2=-25;                 // search right end (25 pixels)
String outputFilename;                // Output file name
```

Figure 8.1 Java code of global variable in PL_StereoTest class

8.3 Loading a stereo pair of images

Two images are required to be loaded as a PlanarImage object. The raster data is also extracted from the file as a global variable to save the calculation time.

```
//*** Load stereo images and set their size and raster data
  private void loadImage(){
    pi = JAI.create("fileload", "0000010000.tif"); // load the first image of stereo pair
    pi2= JAI.create("fileload", "0010000000.tif"); // load the seconf image of stereo pair
    outputFilename="dem9x9offset20_2007_08_14.tif"; // Set the output DEM file name
    imageWidth=pi.getWidth();                    // Set the image width
    imageHeight=pi.getHeight();                  // set the image height
    inputRaster = pi.getData();                 // set the raster data of first image
    inputRaster2 = pi2.getData();                // set the raster data of second image
  }
```

Figure 8.2 Java Code for loading a stereo pair of images

8.4 Getting matrices surrounding a pixel in the first image of a stereo pair

The functions explained in Sections 8.4, 8.5, and 8.6 are used in the main constructor function explained in Section 8.7. The getMatrix is a function to extract the pixel values surrounding a pixel at a position of (X, Y) and return the values as a matrix of an integer array. A 9 x 9 matrix is used to generate Dem images in this project.

```
// *** Local function to get a set of pixel values surrounding (xpos, ypos) pixels as a matrix
  private int[] getMatrix(int xPos, int yPos, int imageID){
    int filterMatrix=filterSize*2+1;                //  Matrix size
    int[] matrix=new int[filterMatrix*filterMatrix];   //  Matrix values
    int index=0;
    // for each pixel position repeating matrix such as 81 times for 9x9 and 25 times for 5x5 matrix
    for(int i=yPos-filterSize; i<=(yPos+filterSize); i++){
      for (int j = xPos - filterSize; j <= (xPos + filterSize); j++) {
        if(i < 0 || i>=imageHeight || j<0 || j>=imageWidth) matrix[index]=0;
        else if(imageID==0) matrix[index]=inputRaster.getSample(j, i, 0); // get the pixel value at (j, i) in the first image
        else if(imageID==1) matrix[index]=inputRaster2.getSample(j, i, 0);  // get the pixel value at (j, i) in the second
image
        index++;
      }
    }
    return matrix;
  }
```

Figure 8.3 Java Code for making a matrix surrounding a pixel

8.5 Calculating a cross-correlation value

The main process of stereo matching is to calculate the cross-correlation between two matrices. The cross-correlation, *C*, is calculated as,

$$C(I_1(i_1, j_1), I_2(i_2, j_2)) = \sum (V_1(i_1, j_i) * V_2(i_2, j_2) - U_1 * U_2) / (O_1 * O_2),$$

$C(I_1(i_1, j_1), I_2(i_2, j_2))$: The cross correlation value of the pixel position (i_1, j_1) in the first image, I_1, and the pixel position (i_2, j_2) in the second image, I_2

$V_k(i_k, j_k)$: The Template matrix around the position of (i_k, j_k) in the first I_k

U_k: The mean value of template matrix $V_k(i_k, j_k)$

O_k: The standard deviation of template matrix $V_k(i_k, j_k)$

```
// A local function to get Cross-correlation of _a anb _b matrices
  // This is called by the get_max_corrlation_xposition function
  private float get_corrlation(int[] _a, int[] _b){
   // C( (i1, j1), (i2, j2))= {V1(i1, j1)*V2(i2, j2) - u1*u2)/(o1*o2)
   // V1 (i1, j1) = template matrix surrounding of pixel position (i1, j1)
   // u1, u2 = mean of the template of V1 and V2
   // o1, o2 = root mean sqaure
   int length=_a.length;                          // The length of matrix such as 81 for 9x9 matrix
   float _a_mean=0.0f;  float _b_mean=0.0f;   // The mean value in each matrix
   float _a_SD=0.0f;    float _b_SD=0.0f;       // The standard deviation value in each matrix
   float correlation=0.0f;                        // correlation value

   // calculate the mean value for each matrix
   for(int i=0; i<length; i++){
     _a_mean += (float)_a[i];
     _b_mean += (float)_b[i];}
   _a_mean = _a_mean/length;
   _b_mean = _b_mean/length;

   // calculate the SD and scalar product
   for(int i=0; i<length; i++){
     _a_SD += (_a[i] - _a_mean) *(_a[i] - _a_mean);
     _b_SD += (_b[i] - _b_mean) *(_b[i] - _b_mean);
    correlation += (_a[i] - _a_mean) * (_b[i] - _b_mean);
   }
   // calculate the cross-correlation
   if(_a_SD*_b_SD == 0) correlation =0.0f;
   else correlation =(float) (correlation/Math.sqrt(_a_SD*_b_SD));
   return correlation;
  }
```

Figure 8.4 Java Code for getting a cross-correlation value

8.6 Find the maximum cross-correlation point

For each pixel in the first image, the position with the maximum cross-correlation among a set of positions on the corresponding line in the second image is calculated. The stereo pair of images has an epi-polar line at the same horizontal line position in each image. For example, an epi-polar line in the third row of the first image occurred in the same place in the second image. Therefore, it is sufficient to calculate the cross-correlation values of all points of one line in the second image and find a point with the maximum value. The real position with a maximum cross-correlation value is refined by using the cross-correlation values of neighbor pixels as:

$$p(\text{real position}) = p(\text{max point}) - \frac{\text{C(next point)} - \text{C(previous point)}}{2 * (\text{C(next point)} + \text{C(previous point)} - 2 * \text{C(max point)})},$$

where p(real position) is the position with a real number, p(max point) is the position with an integer number, C(max point) is the maximum cross-correlation value, C(next point) is the cross-correlation value of the next point, and C(previous point) is the value of the previous point. For example, when the maximum cross-correlation point is $x = 100$, the next and previous points are $x = 101$ and $x = 99$, respectively.

Once the real position with the maximum cross-correlation value in the second image is calculated, the distance between the position and the original point in the first image is returned because the height of the position is relative to the distance in stereo matching.

```
// A local function to get maximum corrlation at (xPox, yPos) pixel of a image
  private float get_max_corrlation_xposition(int xPos, int yPos){
    int[] V1=getMatrix(xPos, yPos, 0);    // Get a matrix around (xPos, yPos)
    boolean allZero=true;                 // flag for no-matching case
    for(int i=0; i<V1.length; i++){       // check if the values in matrix is all 0s
      if(V1[i] !=0) allZero=false;
    }
    if(allZero) return 0.0f;              // If all values in Matrix =0, then return 0.
    float pre_correlation = 0.0f;         // Previous orrelation value
    float nxt_correlation = 0.0f;         // Next Correlation value
    float max_correlation = 0.0f;         // Maximum correlation value during the serach
    float current_correlation =0.0f;      // Current correlation value for xPos
    int    max_index=0;
    for(int i=xPos+hightOffset; i<xPos+hightOffset2 ; i++){
      if( i<0 || i>=imageWidth) continue;

      int[] V2=getMatrix(i, yPos, 1);     //Get a matrix of X=i position from the second image
      float correlation = get_corrlation(V1, V2); // *** Call get_corrlation value of V1 and V2 matrices

      // Update the curren, next, previous corrlation values
      if(correlation > max_correlation){
        pre_correlation = current_correlation;
        max_correlation = correlation;
        max_index = i;
        if(i==(imageWidth-1)) nxt_correlation =0.0f;
      }
      else if(i==(max_index+1)) nxt_correlation = correlation;
      current_correlation = correlation;
    }
    // calculate the real position (not integer) using the previous and next correlation value
    float sub_pos = (hightOffset + hightOffset2)/2;
    if((2.0f*(pre_correlation + nxt_correlation - 2.0*max_correlation))==0){}
    else sub_pos = ((float)max_index) - (nxt_correlation - pre_correlation)/(float)(2.0f*(pre_correlation + nxt_correlation -
2.0*max_correlation));
    return sub_pos-(float)xPos;
  }
```

Figure 8.5 Java Code for getting the position with the maximum cross-correlation value

The following is a part of the main function to save the distances, between the position with the maximum cross-correlation point in the second image and the corresponding point in the first image, as a TIFF image files. In order to reduce the calculation time, the tile size is set as 128 x 128 pixels. The distance value is divided by the maximum distance so that the pixel value is saved as a 16 bit floating point value in the range of 0.0 to 1.0. Figure 8.7 shows the DEM image file calculated in this stage. The distance is represented as the gray level in the image. For example, the lighter pixels signify higher elevations or taller objects. The saved image will be used in the next step as DEM data.

```
// For each tiled image
    for(int tj=tiledImage.getMinTileY(); tj<tiledImage.getNumYTiles(); tj++)
      for(int ti=tiledImage.getMinTileX(); ti<tiledImage.getNumXTiles(); ti++){

        float[] imageData=new float[128*128]; // keep the distanct to the point with maximum correlation
        float maxPos = hightOffset2;          // Search right limit for the maximum correlation point
        float minPos = hightOffset;           // Search left limit for the maximum correlation point
        int count = 0;
        for (int j = 0; j < 128; j++) {       // Image is tiled as 128x128 pixels
          for (int i = 0; i < 128; i++) {
            int xIndex = tiledImage.tileXToX(ti)+i;   // X potioin in original image
            int yIndex = tiledImage.tileYToY(tj)+j;   // Y position in original image
            float[] pos = new float[1];               // Local variable to save the distanct to the point with maximum correlation

           // *** MAIN FUNCTION to calcualte MAX-correlation
           pos[0] = (float) get_max_corrlation_xposition(xIndex, yIndex);
            if (maxPos < pos[0]) pos[0] = maxPos; // Set minimum posiiton for calcuaiton error
            if (minPos > pos[0]) pos[0] = minPos; // Set maximum position for calculation error
            imageData[count] = (float) ( (pos[0] - minPos) / (maxPos - minPos));  // distance to maximum correlation point
            count++;
          }
        }
        // Save the array data of distance to maximum correlation points in Output image
        javax.media.jai.DataBufferFloat dbuffer = new javax.media.jai.DataBufferFloat(imageData, 128 * 128);
        Raster raster = RasterFactory.createWritableRaster(sampleModel, dbuffer, Point(128*ti, 128*tj));
        tiledImage.setData(raster);
      }
    JAI.create("filestore", tiledImage, "floatpattern.tif", "TIFF"); // Save the output image as TIFF file
  }
```

Figure 8.6 The Java Code for calculating and saving DEM image files

Figure 8.7 A DEM image

9.0 Tool Implementation: Stage3 Integration

9.1 Introduction

This section shows the several techniques used in implementing the tool in Java for Stage 3 described in Chapter 4.11. The goal is to develop a tool utilizing the following functions:

- Load the multi-spectrum image and DEM image file
- Set the geographical information using reference points
- Set the location of transmission towers and lines
- Extract healthy vegetation pixels with a value more than the NDVI threshold
- Label the regions and extract the boundary polygons
- Get the tree locations
- Visualize the DEM data with two cross-sections

Each function is explained in Sections 9.3 through 9.9, respectively.

9.2 Packages and classes

The all Java classes are implemented in a package named "pserc." The following is the list of classes developed for Stage 3.

- PL_Output
- PL_OutputControlPanel
- PL_OutputHistograph

PL_Ouput is a main window frame class, and it has the main function to execute this tool. The frame has two components, the PL_OutputControlPanel and the PL_OutputHistograph as shown in Figure 9.1.

PL_OutputControlPanel is the panel to allow the user to control several input data. There are 4 sub panels: the NDVI-panel to interact with the NDVI threshold value, the Geo-panel to show the reference points for geographical setting, the tower-panel to show the latitude, longitude, and altitude data of transmission towers, and the tree-panel to show the list of trees with information of location and distance to transmission line.

PL_OutputHistograph is a main image panel to show NDVI data and DEM data. It has the functions to label the regions with healthy NDVI values and generate the polygon and boundary box from each region.

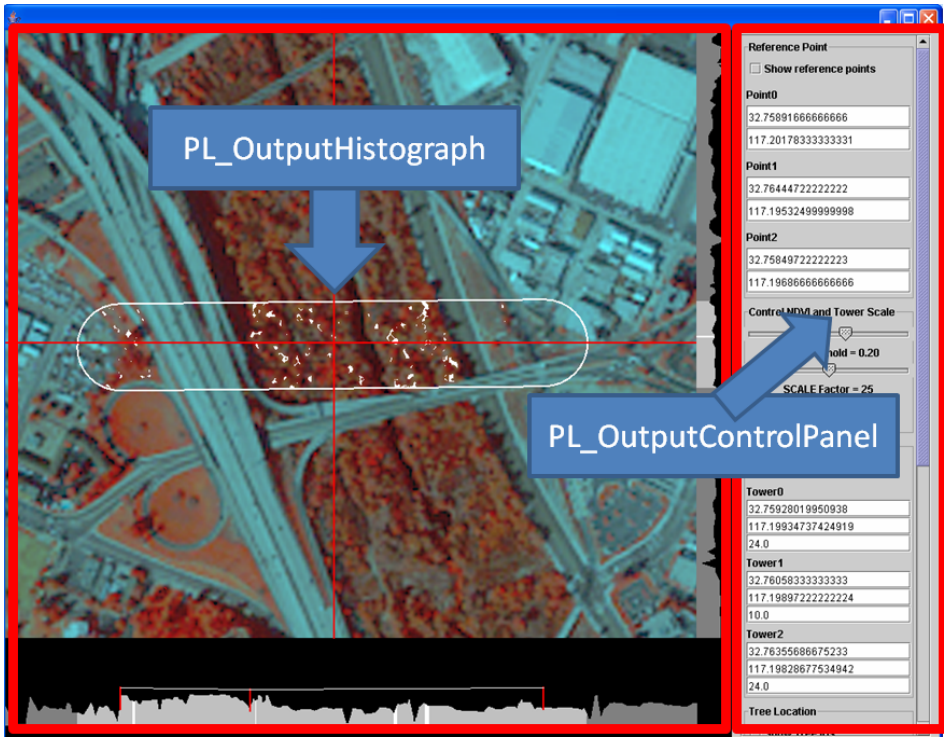


Figure 9.1 A Screen Shot of the PL_Output Frame

9.3 Load the multi spectrum image and DEM image file

The first step is to load the multi spectrum image and DEM image. They are the satellite images of IKONOS provided by a company, GEOEYE. The process to load the multi spectrum image is the same as the one developed in Stage 1. The DEM image file generated by the program of Stage 2 is loaded. The setImage() in the PL_OutputHistograph is the function .

```
  // Sub function called by Constructor to load images
 public void setImage(){
  origImage =JAI.create("fileload", "color4.tif");      // load color image for NDVI image processing
  demImage= JAI.create("fileload", "floatpattern.tif"); // load DEM image for showing cross-section
  panX =0; panY =0;                                     // Set the pan value as (0, 0)
  atx = AffineTransform.getTranslateInstance(0.0, 0.0); // Set the default Translate materix
  RenderedOp op = makeTiledImage(origImage);            // Call this makeTiledImage function for making tiled image
  demRaster=demImage.getData();                         // set DEM raster data
  displayImage = op.createInstance();         // create a display image copied from original image
  sampleModel = displayImage.getSampleModel(); // assign the smaple model from the image
  colorModel = displayImage.getColorModel();   // assign the color model of the image
  getTileInfo(displayImage);                   // get tile information for display image
  fireTilePropertyChange();                    // change the properties of tile
  imageDrawn = false;                          // image drawn is set as false in default
    }
```

Figure 9.2 Java Code for loading the multi-spectrum image and DEM image

9.4 Setting the geographical information using reference points

In this stage, the reference points to set up the geographical information are manually specified because we could not get the data. By using the commercial GPS device and Google Earth, the latitude and longitude was measured as described in Appendix B. Because the images used in this stage did not have proper orientation, a new constructor function was added to the PL_Geo class,

PL_Geo geo = new PL_Geo(lat0, lon0, x0, y0, lat1, lon1, x1, y1, lat2, lon2, x2, y2);

The PL_Geo's constructor needs 12 parameters of 3 reference points to initialize an object. The "lat0" and "lon0" are the latitude and the longitude in the world coordinate, and the "x0" and "y0" are the x and y position of the first reference point in the image. The parameters with the index "1" are for the second reference point, and those with index "2" are for the last reference point. The geographical coordinates are extracted from Google Earth. The conversion from the XY coordinate in the image to the geographical coordinate is implemented using the regular algebra defined as below.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} X_0 & X_1 \\ Y_0 & Y_1 \end{pmatrix}=\begin{pmatrix} latitude_0 & latitude_1 \\ longitude_0 & longitude_1 \end{pmatrix}$$

The 2 x 2 linear transformation matrix is defined in the "createGeoPanel()" function of the PL_OutputControlPanel using the following reference points:

Table 9.1 Information of three reference points in image

	Latitude	Longitude	X	Y
Point 0	32º45'32.10¨	117 º 12'06.42"	46	154
Point 1	32º45'52.01"	117 º 11'43.17"	778	606
Point 2	32º45'30.59"	117 º 11'48.72"	95	607

The latitude and longitude are measured in degrees, minutes, and sounds defined in the "createGeoPanel() function of PL_OutputControlPanel. The *X* and *Y* values are the position of the first image of the stereo pair, and they are defined in the constructor function of PL_OutptuHistograph to allow the user to change and update the position after executing the program.

9.5 Set the location of transmission towers and lines

In Stage3, the geographical coordinates of transmission tower locations are defined by using Google Earth and a GPS device. The following are the latitude, longitude, and altitude for 3 transmission towers.

Table 9.2 Location of three transmission towers

	Latitude (Degree)	Longitude (Degree)	Altitude (feet)
Tower 0	32.75928019950938	117.19934737424919	24.0
Tower 1	32.76058333333333	117.19897222222224	10.0
Tower 2	32.76355686675233	117.19828677534942	24.0

The information is set in the "createTowerPanel()" function of the PL_OutputControlPanel.

9.6 Extract healthy vegetation pixels with value more than the NDVI threshold

The next step is to extract the healthy vegetation pixels whose NDVI value is more than the NDVI threshold defined in the slider component, NDVISlider, in the PL_OutputControlPanel. The NDVI values are calculated using the same formula defined in Section 5.5. The healthy pixels are displayed in white. The process is implemented in the paint-Components() function in PL_OutputHistograph.

9.7 Labeling and Boundary Searching

The pixels with a value more than the NDVI threshold were painted in white, as explained in the previous section. In order to show a list of regions (not a list of pixels) with healthy NDVI values, it is required to make a region from adjacent pixels. This is called "labeling" in image processing, and it is a process to assign a unique index for each region with adjacent pixels from binary data (0s and 1s). The following is the Java code to implement the labeling algorithm.

```
// sub function called by paintComponent function to segment the healthy pixels with unique indices
// Fij[] = 1 or 0, _w= image width, _h=image height, (_ti, _tj) index of tile
private void createRegion8(int[] Fij, int _w, int _h, int _ti, int _tj){

  for(int i=0; i<_h; i++){ // for each pixel
   for(int j=0; j<_w; j++){

    if(Fij[i*_w+j]==0){}     // checking if 0 or 1
    else {
      int[] lp=new int[4];   // create int[4]
      if(i-1<0){ lp[0]=0; lp[1]=0; lp[2]=0;} // if it is out of image
      if(j-1<0){ lp[0]=0; lp[3]=0;}
      if(j+1>_w){lp[2]=0;}
      lp[0]=Fij[(i-1)*_w+(j-1)];    // X    X     X     X       X
      lp[1]=Fij[(i-1)*_w+j];        // X  lp[0] lp[1] lp[2]  X
      lp[2]=Fij[(i-1)*_w+(j+1)];    // X  lp[3]  (j,i)   X     X
      lp[3]=Fij[i*_w+(j-1)];        // X    X     X     X       X
      int L1=0; int L2=0;           // labels for search conditions

      //*** Get the indices of L1 and L2 (L1<L2)
      //except if L1=L2=0, if L2=0, keep the L1=ID, L2=0.
      for(int p=0; p<4; p++){ //*** Keep the indices L1 (smallar ID) L2 (bigger ID)
        if(lp[p]!=0){
          if(L1==0){ L1=lp[p];} // L1 is changed from 0 to lp[p] for first hit
          else if(L2==0 && L1==lp[p]){}// for searching in the same ID area
          else if(L2==0 && L1>lp[p]){  // for second hit but order is wrong
              L2=L1;
              L1=lp[p];
          }
          else if(L2==0 && L1<lp[p]){L2=lp[p];} // for second hit
          else{// L2 !=0 && L1!=0
          }
        }
      }
      //*** Checking Condition of prePixels
      if(L1==0 && L2==0){ // Fij[j,i] is the first point for a new region
        lamda++;
        if(lamda<maxLamda){   //lamda=0~499
          tempT[lamda]=lamda; // lamda = regionID is assigned to tempT and Fij
          Fij[i*_w+j]=lamda;
        }
        else{                 // in the case for updating the array space
          maxLamda+=500;
          int[] tempTT=new int[maxLamda];          // expand the array space
          System.arraycopy(tempT,0,tempTT,0,lamda); // copy the array contents
          tempTT[lamda]=lamda;
          tempT=tempTT;
          Fij[i*_w+j]=lamda;
        }
      }
      else if(L2==0&& L1!=0){ // searching an existing region
        Fij[i*_w+j]=L1;
      }
      else {                // Two areas(L1!=0 && L2!=0) are existing in the lp[]
        Fij[i*_w+j]=L1;     // change the index of Fij from 1 to L1
        for(int r=0; r<=lamda; r++){
          if(tempT[r]==L2) tempT[r]=L1;
        }
      }

    }
   }
  }// end of all pixels

  // seach again to get the starting point for each region
  for(int i=0; i<_h; i++){
   for(int j=0; j<_w; j++){
    if(Fij[i*_w+j]>0){                // Hit a region
       int tempID=Fij[i*_w+j];
       Fij[i*_w+j]=tempT[tempID];     // Assign the regionID to Fij
       Integer segIDInte=new Integer(tempT[tempID]);
       if(!idV.contains(segIDInte)) {
         idV.addElement(segIDInte);   // add IDs to the list
         getSegPolygon(Fij, j, i, _ti, _tj); // call sub function to extract polygon (or boundary box)
  }
```

Figure 9.3 Java Code for labeling the healthy vegetation regions

After labeling the regions with healthy vegetation, it is necessary to define the boundaries of the labeled regions. The boundaries of each region are approximated by a polygon. These polygons are extracted using the technique of boundary searching. There are mainly three kinds of boundary searching, which are edge-based, corner-based, and pixel-based searching. This program applied corner-based searching because the searching algorithm can determine the boundary correctly with only one way tracking. The other two algorithms need both clockwise and counterclockwise tracking. Corner-based searching requires preparing 16 conditions of 2 x 2 binary pixels beforehand. Each condition has one tracking direction among top, bottom, right, and left. The conditions and their tracking directions are shown in Figure 9.2.

The algorithm begins searching from the start point and defines the correct direction by checking which condition is matched among the 16 cases described below. If the direction is changed, the point is added to the list of the boundary polygon. If the searching point comes back to the start point, the searching is stopped. A polygon extracted from each region is saved in a Vector object as a boundary box.

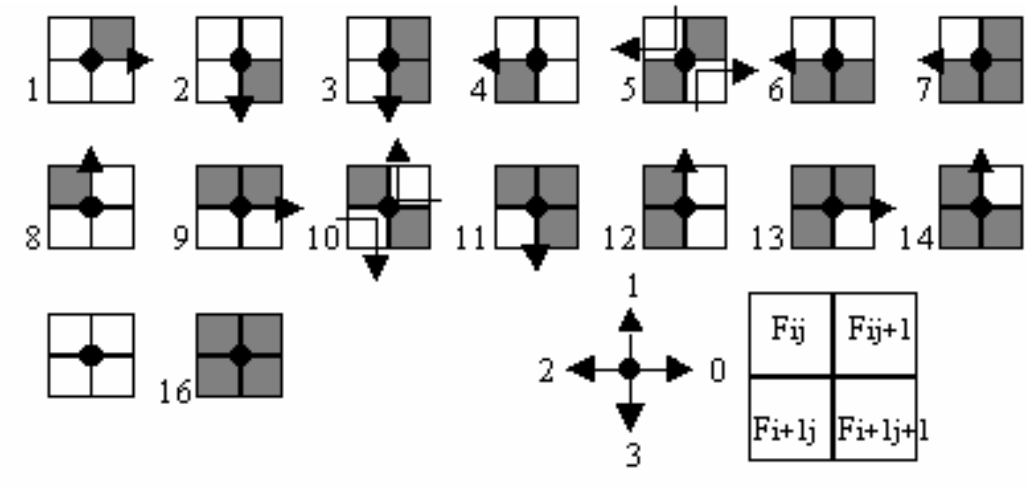


Figure 9.4 Conditions of 2 x 2 pixels and their searching directions

```
// sub function called by createRegion8() function to create a polygon from pixel-region
  // Fij[] = indices of regions. (_x,_y)=staring point, (_ti,_tj)= tile indices
  public void getSegPolygon(int[] Fij, int _x, int _y, int _ti, int _tj){
      Polygon p=new Polygon();    // output polygon for each region
      int _w=256;                 // tiled image has 256x256 pixels
      int j=_x; int i=_y;         // (j, i) is position of starting
      int ID=Fij[_y*_w+_x];       // index of region in Fij[]
      boolean comeback=false;     // used to check if the searching finished
      int     direction = 0;      // direction of searching: 0 (right), 1(up), 2(left), 3(down)
      int count=0;                // keep the counts to avoid infinite loop

      while(!comeback && count<100){                    // X X X    X     X   X X X
        count++;                                        // X X X (LT) (RT) X X X
        int LT = 0;int RT = 0;int LB = 0; int RB = 0;   // X X X (LB) (RB) X X X
        // get the values at the position of LT(left top), RT (right top), LB (left bottom), RB (right bottom)
        if (j - 1 >= 0 && i - 1 >= 0) LT = Fij[ (i - 1) * _w + j - 1];
        if (j <= _w && i - 1 >= 0) RT = Fij[ (i - 1) * _w + j];
        if (j - 1 >= 0 && i < _w) LB = Fij[i * _w + j - 1];
        if (j <= _w && i < _w) RB = Fij[i * _w + j];

        if (LT != ID && RT ==ID && LB !=ID && RB !=ID ) { //case 1: go right
          p.addPoint(_ti*_w+j, _tj*_w+i);  // X O
          direction=0;                     // X X
          j++;
        }
        else if (LT != ID && RT != ID && LB != ID && RB == ID) { //case 2: go down
          p.addPoint(_ti*_w+j, _tj*_w+i);  // X X
          direction=3;                     // X O
          i++;
        }
        else if (LT != ID && RT == ID && LB != ID && RB == ID) { //case 3: go down
          direction = 3;                   // X O
          i++;                             // X O
        }
        else if (LT != ID && RT != ID && LB == ID && RB != ID) { // case 4: go left
          p.addPoint(_ti*_w+j, _tj*_w+i);  // X X
          direction = 2;                   // O X
          j--;
        }
        else if(LT!=ID && RT==ID && LB==ID && RB!=ID){// case 5 : go right if the pre-direction is up
             if(direction == 1){                   // X O
                p.addPoint(_ti*_w+j, _tj*_w+i);   // O X
                direction=0;
                j++;
             }
             else if(direction == 3){              // go left if the pre-direciton is left
                p.addPoint(_ti*_w+j, _tj*_w+i);
                direction=2;
                j--;
             }
             else{System.out.println("*** GetLInkList has case 5 with error.");}
          }

        else if (LT != ID && RT != ID && LB == ID && RB == ID) { //case 6: go left
          direction = 2;     // X X
          j--;               // O O
        }
        else if (LT != ID && RT == ID && LB == ID && RB == ID) { //case 7: go left
          p.addPoint(_ti * _w + j, _tj * _w + i);
          direction = 2;    // X O
          j--;              // O O
        }
        else if (LT == ID && RT != ID && LB != ID && RB != ID) { //case 8 : go top
          p.addPoint(_ti * _w + j, _tj * _w + i);
          direction = 1;    // O X
          i--;              // X X
        }
        else if (LT == ID && RT == ID && LB != ID && RB != ID) { //case 9 : go right
          direction = 0;    // O O
          j++;              // X X
        }
        else if (LT == ID && RT != ID && LB != ID && RB == ID) { //case 10 : go down if pre is right
          if (direction == 0) {                     // O X
            p.addPoint(_ti * _w + j, _tj * _w + i);  // X O
            direction = 3;
            i++;
          }
          else if (direction == 2) {                // go top if pre is left
```

```
          p.addPoint(_ti * _w + j, _tj * _w + i);
          direction = 1;
          i--;
        }
        else {
          System.out.println("*** GetLInkList has case 5 with error.");
        }
      }
      else if (LT == ID && RT == ID && LB != ID && RB == ID) { //case 11 : go down
        p.addPoint(_ti * _w + j, _tj * _w + i);
        direction = 3;       // O O
        i++;                 // X O
      }
      else if (LT == ID && RT != ID && LB == ID && RB != ID) { //case 12: go up
        direction = 1;      // O X
        i--;                // O X
      }
      else if (LT == ID && RT == ID && LB == ID && RB != ID) { //case 13; go right
        p.addPoint(_ti * _w + j, _tj * _w + i);
        direction = 0;     // O O
        j++;               // O X
      }
      else if (LT == ID && RT != ID && LB == ID && RB == ID) { //case 14: go up
        p.addPoint(_ti * _w + j, _tj * _w + i);
        direction = 1;   // O X
        i--;             // O O
      }
      if (i == _y && j == _x) comeback = true; // check if the searching is back to the original position
    }// end of while-loop
    if(p.npoints>1) {  // If there are points more than 1, then add the boundary box to ipV vector.
      Rectangle r=(Rectangle)p.getBounds();
      ipV.addElement(r);
    }
  }
```

Figure 9.5 Java code for boundary searching

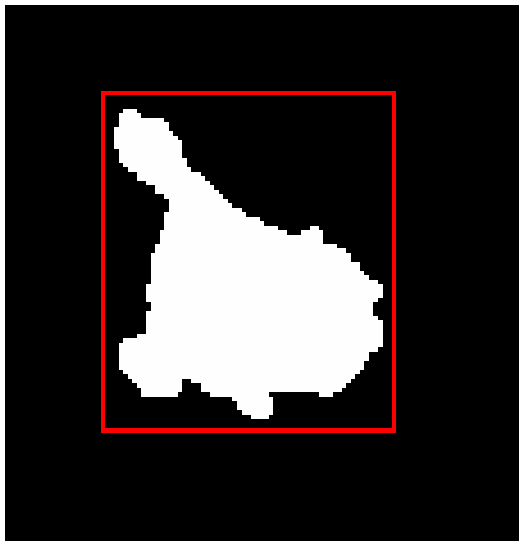


Figure 9.6 Healthy vegetation and boundary box

9.8 Obtaining tree locations

The locations of tree are posted in the text area of the treePanel of the PL_OutputControlPanel. As explained in the precious section, each region of healthy vegetation is converted to a boundary polygon and saved as a boundary box in a Java Vector object. Figure 9.6 shows the healthy vegetation and the boundary box in white and red respectively. The location is define as the center of the boundary box, and the geographical coordinate is calculated

from the XY position in the image by using the “getLAT(X, Y)” function in PL_Geo. Table 9.3 shows the results of 5 trees among 95 total in the case of the NDVI threshold = 0.20.

9.9 Visualizing the DEM data with a two cross-section

The final step is to visualize the output. The paintComponent function in the PL_OutputHistograph is the function to draw the 6 components: 1) pixels, 2) ROI and section lines, 3) cross-sections, 4) transmission lines and towers, 5) boundary boxes of healthy trees, and 6) reference points as shown in Figure 9.7.

Table 9.3 Extracted trees

ID	Latitude	Longitude	Distance to Line
0	32.759251042034705	117.19985426770678	25.782452
1	32.75928400942072	117.1998345690552	27.645329
2	32.75953096171785	117.19973683669522	17.101068
3	32.75918758139942	117.19980425768894	23.711155
4	32.759336868606894	117.1997209176232	27.473486
5	32.759566403240605	117.19962773072264	16.185623
…..			

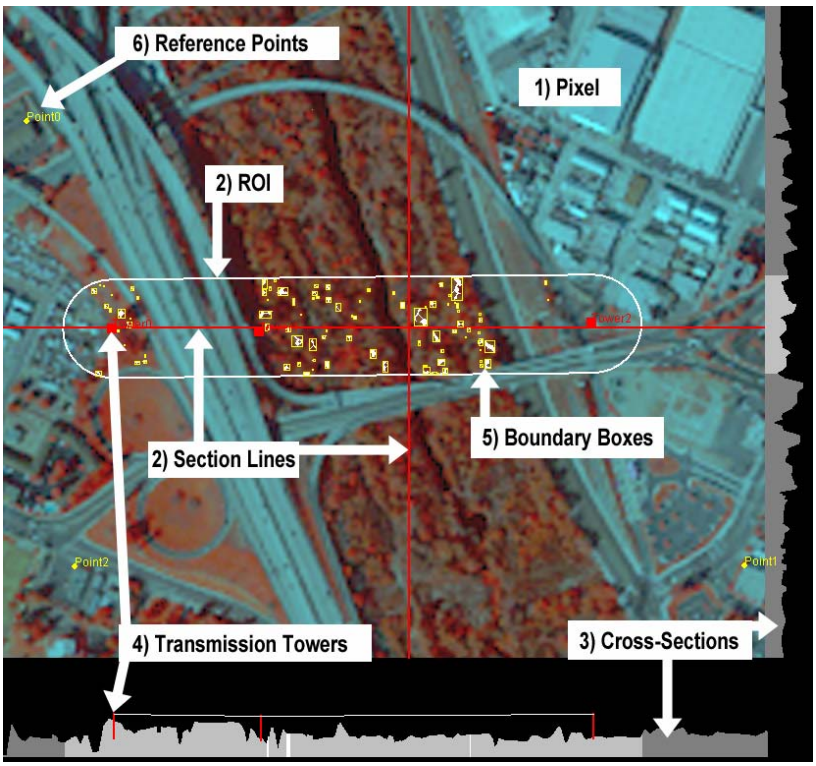


Figure 9.7 Drawing components

1) Pixels: As explained in Section 9.6, the NDVI value is replaced in the red channel the same as Stage 1.

```
for(tj = topIndex; tj <= bottomIndex; tj++) {      // loop for each tile
    for (ti = leftIndex; ti <= rightIndex; ti++) {

      Raster raster = displayImage.getTile(ti, tj);  // Get the raster information
      int width2 = raster.getWidth();               // get raster's width
      int height2 = raster.getHeight();             // get raster's height
      int bands = raster.getNumBands();             // get raster's band number (Supposed 4)
      int[] pixels = new int[width2 * height2 * bands]; // pixel information of tile image
      byte[] data = new byte[width2 * height2 * 3];  // target pixel data
      int[] Fij=new int[width2* height2];           // region ID data
      tempT[0]=0; tempT[1]=0;                       // tempT is used for segmentation of selected pixels
      raster.getPixels(ti * 256, tj * 256, width2, height2, pixels); // 256 x 256 is the tile size FIXED

      int index = 0;
      for (int h = 0; h < height2; h++)             // For each pixel in a tile
        for (int w = 0; w < width2; w++) {

          int red = pixels[index * 4 + 0];    // Get Red band pixel info
          int inf = pixels[index * 4 + 3];    // Get Infrared band pixel info
          float ndvi = (float) ( ( (float) (inf - red)) / ( (float) (red + inf))); // Cacluate NDVI
          byte ndviByte=(byte)( (ndvi + 1.0) * 100.0);  // convet NDVI rane from -1.0~1.0 to 0~200
          data[index * 3 + 2] = (byte) ndviByte;                  //set the NDVI in red band
          data[index * 3 + 1] = (byte) pixels[index * 4 + 1];       //Set the original Green in green band
          data[index * 3 + 0] = (byte) pixels[index * 4 + 2];       //Set the original Blue in blue band
          int thres=slideValue;              // get slide value defined in PL_OutputControlPanel

          // ** Get Healthy Pixels
          int xIndex = displayImage.tileXToX(ti)+w;   // get the real position in a big origianl image
          int yIndex = displayImage.tileYToY(tj)+h;   // get the real position in a big origianl image
          if(ndviByte>thres && Roi.contains(xIndex, yIndex)){
            Fij[index]=1;                          // If the pixel is inside ROI and more than NDVI theshold
          }
          index++;
      } // end for all pixels in a tile

      // cut less than 4 pixels
      int[] Fij2=new int[256*256];  // temporal int[] to convert the dat
      index=0;
      for (int h = 0; h < height2; h++)
        for (int w = 0; w < width2; w++) {
          if(Fij[index]>0 &&  // If the NDVI value is more than Thesdhold, the pixel is WIHTE.
           (w+1)<width2 && (w-1)>=0 && (h+1)<height2 && (h-1)>=0 &&
               Fij[(h+1)*width2+w]>0 &&
               Fij[h*width2+w-1]>0 && Fij[h*width2+w+1]>0 &&
               Fij[(h-1)*width2+w]>0){
            data[index * 3 + 2] = (byte) 255; //red
            data[index * 3 + 1] = (byte) 255; //green
            data[index * 3 + 0] = (byte) 255; //blue
            Fij2[index]=1;
            if(ti*256+w==mouseP.x){  // add the healthy pixel position of vertical cross-section
              int ypos=tj*256+h;
              ndvi380.add(new Integer(ypos));
            }
            if(tj*256+h==mouseP.y){ // add the healthy pixel position of horizontal cross-section
              int xpos=ti*256+w;
              ndvi360.add(new Integer(xpos));
            }
          }
          index++;
        }

      Fij=new int[256*256];
      for(int i=0; i<256*256; i++) Fij[i]=Fij2[i];  // swap the Fij and Fij2
      createRegion8(Fij, width2, height2, ti, tj);  // *** Segmentation to put indices for each region

      // create a data buffer from data[] and draw the new tile image at the proper position
      DataBufferByte dbuffer = new DataBufferByte(data, width2 * height2 * 3);
      sampleModel = RasterFactory.createPixelInterleavedSampleModel(DataBuffer.TYPE_BYTE, width2, height2, 3);
```

```
      colorModel = PlanarImage.createColorModel(sampleModel);
      WritableRaster wr = RasterFactory.createWritableRaster(sampleModel,dbuffer, new Point(0, 0));
      BufferedImage bi = new BufferedImage(colorModel, wr, colorModel.isAlphaPremultiplied(), null);
      int xInTile = displayImage.tileXToX(ti); // get X tile indices
      int yInTile = displayImage.tileYToY(tj); // get Y tile indices
      AffineTransform tx = AffineTransform.getTranslateInstance(xInTile +panX, yInTile + panY);
      g.drawRenderedImage(bi, tx);
     }
    }
```

Figure 9.8 Java code for drawing pixels

2) ROI and section lines: The boundary of ROI is drawn with a thick white line. The section lines are defined from a mouse clicked point, and the horizontal and vertical lines are drawn in red.

```
// Step2: *** Draw ROI and cross-section lines***
    if (Roi == null) return;
    AffineTransform tx = AffineTransform.getTranslateInstance(panX, panY);
    Shape shape = tx.createTransformedShape(Roi.getAsShape());
    g.setStroke(new BasicStroke(2));      //Set the stroke size as 2
    g.setPaint(new Color(255, 255, 255)); //Set as White
    g.draw(shape);                        // draw ROI
    g.setColor(Color.red);                // set as Red
    g.drawLine(0, mouseP.y, 800, mouseP.y); // draw horizontal cross-section line
    g.drawLine(mouseP.x, 0, mouseP.x, 700); // draw vertical cross-section line
```

Figure 9.9 Java code for drawing ROI and section lines

3) Cross-sections: The horizontal cross-section is displayed at the bottom of the site image, and the vertical cross-section is displayed at the right of the image. The sections inside of ROI are drawn in a light gray, and the healthy vegetation points are dawn in white. The outside of ROI is drawn in dark gray. These cross-sections are updated always when the user clicks the different point in the image.

```
// Step3: *** Draw Cross-section ground***
    for(int i=0; i<700; i++){
      if(!Roi.contains(mouseP.x, i)) g.setColor(Color.gray);    // out of ROI
      else if (ndvi380.contains(new Integer(i))) g.setColor(Color.white);  // healthy NDVI
      else g.setColor(Color.lightGray);                        // inside ROI but not healthy
      g.drawLine(800, i, 800 + (int)(xdem[i])  , i);           // draw a section line
    }
    for(int i=0; i<800; i++){
      if(!Roi.contains(i, mouseP.y)) g.setColor(Color.gray);    // out of ROI
      else if (ndvi360.contains(new Integer(i))) g.setColor(Color.white); // healthy NDVI
      else g.setColor(Color.lightGray);                        // inside ROI but not healthy
      g.drawLine(i, 800, i,  800 - (int)ydem[i]);              // draw an horizontal section
    }
```

Figure 9.10 Java code for drawing cross sections

4) Transmission towers and lines: The transmission towers are drawn as red lines on the horizontal cross-section. The locations of towers are also visible when the check box named “Show Tower” in PL_OutputControlPanel is on. The height of tower is changed by sliding the bar named “SCALE Factor” in the PL_OutputControlPanel.

Each transmission line is drawn between the tops of tower as a white line in the horizontal cross-section.

```
// Step4: *** Transmission lines and towers***
   g.setColor(Color.red);
   g.setStroke(new BasicStroke(2));
   g.drawLine(tower0.x, 800-(int)towerZ0 , tower0.x, 800-(int)towerZ0-towerLength); // draw tower 0
   g.drawLine(tower1.x, 800-(int)towerZ1 , tower1.x, 800-(int)towerZ1-towerLength); // draw tower 1
   g.drawLine(tower2.x, 800-(int)towerZ2 , tower2.x, 800-(int)towerZ2-towerLength); // draw tower 2
   g.setColor(Color.white);
   g.setStroke(new BasicStroke(1));
   g.drawLine(tower0.x, 800-(int)towerZ0-towerLength , tower1.x, 800-(int)towerZ1-towerLength); // draw line be-
tween towe0 and 1
   g.drawLine(tower1.x, 800-(int)towerZ1-towerLength , tower2.x, 800-(int)towerZ2-towerLength); // draw line be-
tween tower1 and2
   if(PL_OutputControlPanel.showTower.isSelected()){
     g.setColor(Color.red);                    //set color as Red
     g.setStroke(new BasicStroke(2));
     g.fillRect(tower0.x - 5, tower0.y - 5, 10, 10); // draw tower 0 as 5x5 box
     g.drawString("Tower0", tower0.x, tower0.y);
     g.fillRect(tower1.x - 5, tower1.y - 5, 10, 10);  // draw tower 1 as 5x5 box
     g.drawString("Tower1", tower1.x, tower1.y);
     g.fillRect(tower2.x - 5, tower2.y - 5, 10, 10); // draw tower 2 as 5x5 box
     g.drawString("Tower2", tower2.x, tower2.y);
   }
```

Figure 9.11 Java code for drawing transmission towers and lines

5) Boundary Boxes: The pixels greater than the NDVI threshold are painted in white. To avoid the noise pixels, 4 neighbors filter is applied. The boundary boxes are displayed in yellow.

```
// Step5: *** Draw boundary boxes of healthy trees ***
   g.setColor(Color.yellow);            //set color as yellow
   g.setStroke(new BasicStroke(1));     //Set the stroke size as 2
   PL_OutputControlPanel.treeInfo.setText("ID \t"+"Latitude \t\t"+"Longitude \t\t"+"Distance to Power line \n");
   for(int i=0; i<ipV.size(); i++){      // for each region (polygon)
     Rectangle p=(Rectangle)ipV.elementAt(i);    // extract boundary box from ipV vector object
     if(PL_OutputControlPanel.showTreeBox.isSelected())
       g.drawRect(p.x, p.y, p.width, p.height);    // darw boundary boxes
     if(PL_OutputControlPanel.showTreeID.isSelected())
       g.drawString(""+i, p.x, p.y);              // draw index numbers

     int tX=(p.x+p.width/2); // get center X of boundary box
     int tY=(p.y+p.height/2);// get genter Y of boundary box
     double[] pos=PL_Geo.getLAT(tX, tY);     // convert (x, y) to (latitude, longitude)
     Point tempP =PL_Geo.getPixelPos(pos[0], pos[1]);
```

Figure 9.12 Java code for drawing the boundary box of healthy vegetation

6) Reference Points: The 3 reference points are displayed as a small yellow circle with the names only when the check box named “Show Reference Points” is on in the PL_OutputControlPanel.

```
// Step6: *** Draw Referece points ***
     if(PL_OutputControlPanel.showRef.isSelected()){ // if checkbox is ON
       g.setColor(Color.yellow);
       g.fillOval(ref0.x-3, ref0.y-3, 6, 6);  // draw reference 0
       g.drawString("Point0", ref0.x, ref0.y);
       g.fillOval(ref1.x-3, ref1.y-3, 6, 6);  // draw reference 1
       g.drawString("Point1", ref1.x, ref1.y);
       g.fillOval(ref2.x-3, ref2.y-3, 6, 6);  // draw reference 2
       g.drawString("Point2", ref2.x, ref2.y);
     }
```

Figure 9.13 Java code for drawing reference points

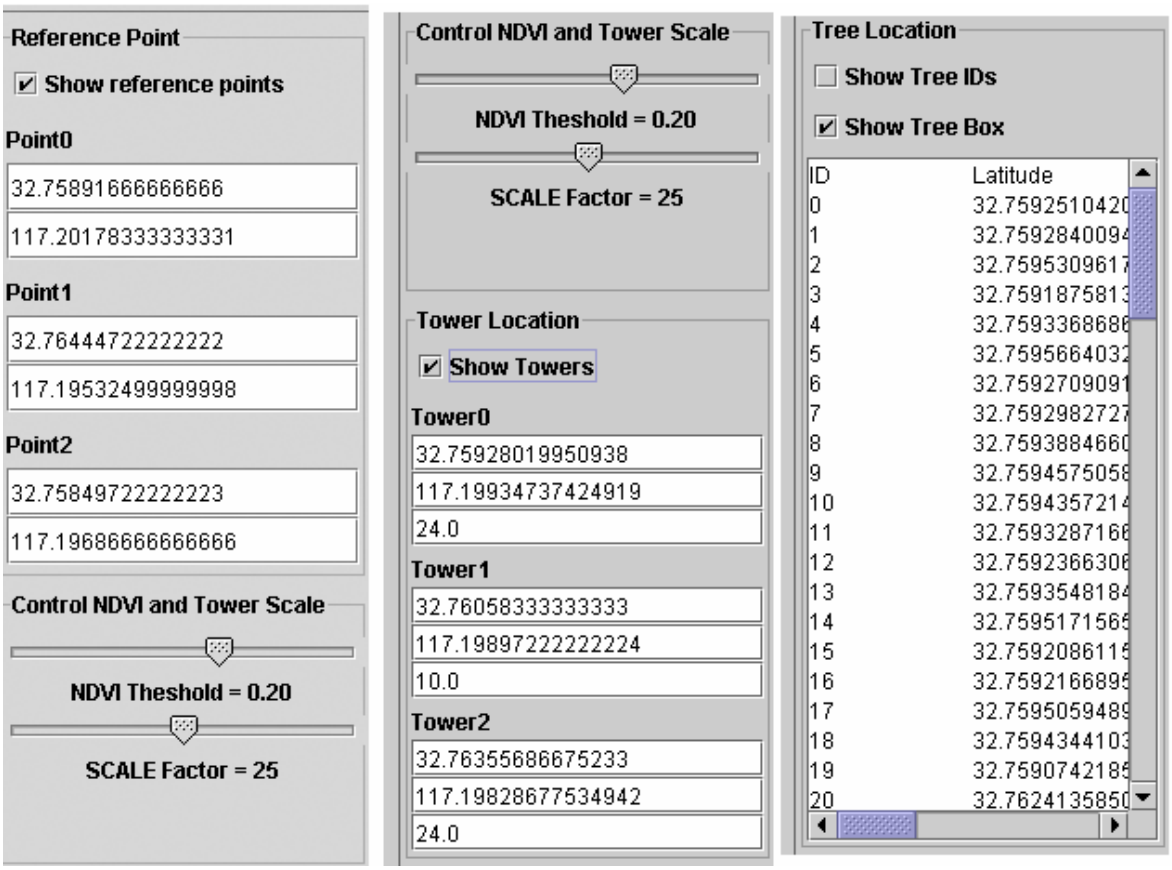


Figure 9.14 The PL OutputControlPane

10.0 Tool Instruction: Visualization

10.1 Prerequisite files

This section shows the instructions for the program developed in Stage 3. The PL_Output has the main function to execute the program.

This tool can be executed in any Java environment. However, there are four required files to run it correctly. Two panchromatic epipolar stereo images, named as “000.tif” and “001.tif,” are required.Correct operation needs a DEM image file and a multispectrum image with red, green, blue, and infrared. The DEM image file should be named “floatpattern.tif,” and the multispectrum image should be named “color4.tif.” The “000.tif” and “001.tif” show the different views because they are a stereo pair. The file, “color4.tif,” should have the same view as “000.tif.” These four image files should be saved in the “PSerc” directory in the window system.

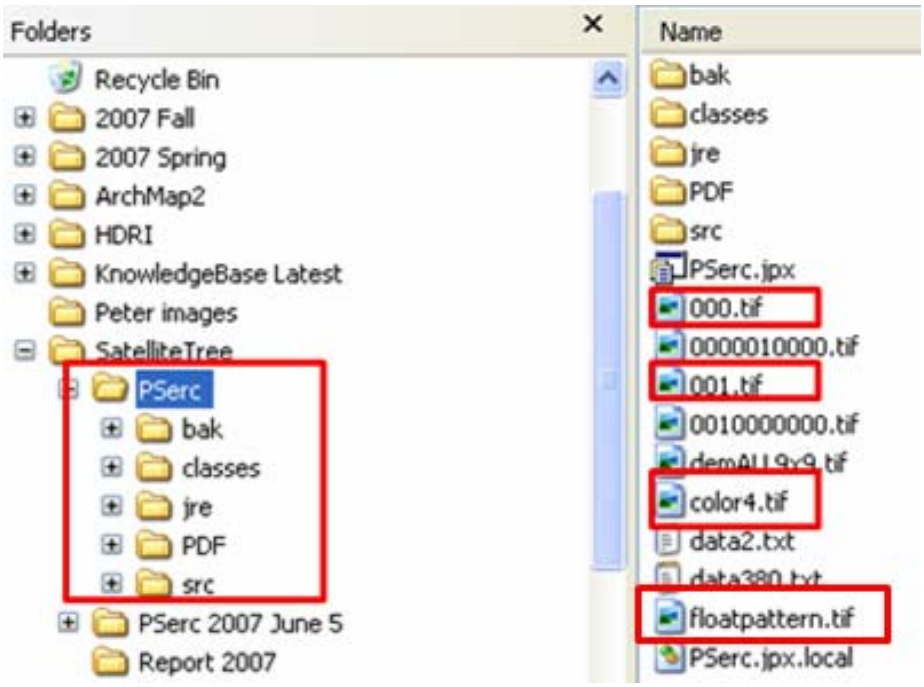


Figure 10.1 Prerequisite files

10.2 Geographical setting

In order to define the projection between the pixel (XY) coordinate and geographical (latitude and longitude) coordinate, the user should acquire the information of three locations in the image before starting the program. The three locations should be ground true points. If the information is not available, the information of latitude and longitude is obtained by using Google Earth. In this project, we used the program to obtain the information as shown in Table 9.1. Theoretically, the three points should be selected near the different corners of the image.

10.3 Tower locations

Table 9.2 presents the tower locations of the transmission line around San Diego

10.4 Running the program

Figure 10.2 shows the initial screen shot of the developed program. The stereo images and multispectrum images are automatically set by reading the image files, which are saved in the proper directory as described in Section 10.1. The three reference points are automatically shown in the first block of the right panel. The default values of the NDVI threshold and scale factor (explained in Sections 9.6 and 9.9) are set as 0.20 and 25, respectively, as shown in the second block in the right panel. The information of the towers is shown in the third block of the right panel.

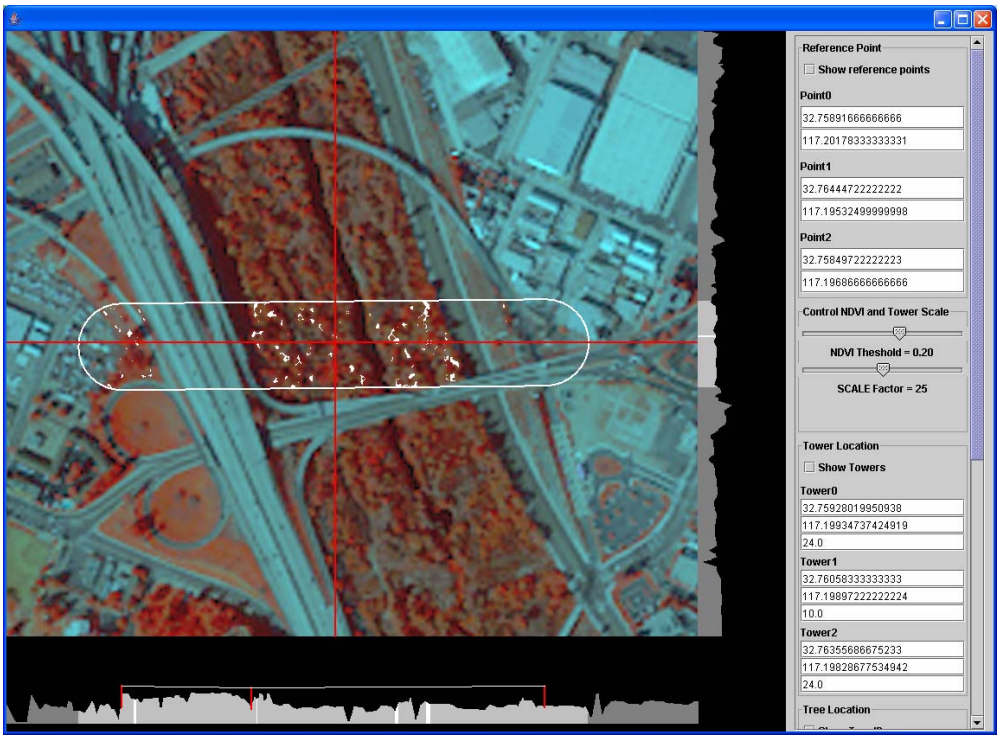


Figure 10.2 The initial screen shot

10.5 Reference point visibility

This program has a function to hide and show the three (3) reference points by checking the box named “Show reference points” in the right panel, as shown in Figure 10.3. Once the checkbox is checked, three reference points are shown on the image. In the same block as the checkbox, the latitude and longitude for each reference point are shown. For each point, the latitude is shown in the first textbox and the longitude is shown in the second textbox. For example, the latitude of the first reference point, “Point0,” is 32.75891666666666 degrees, and the longitude is 117.20178333333331 degrees, as shown in Figure 10.3.

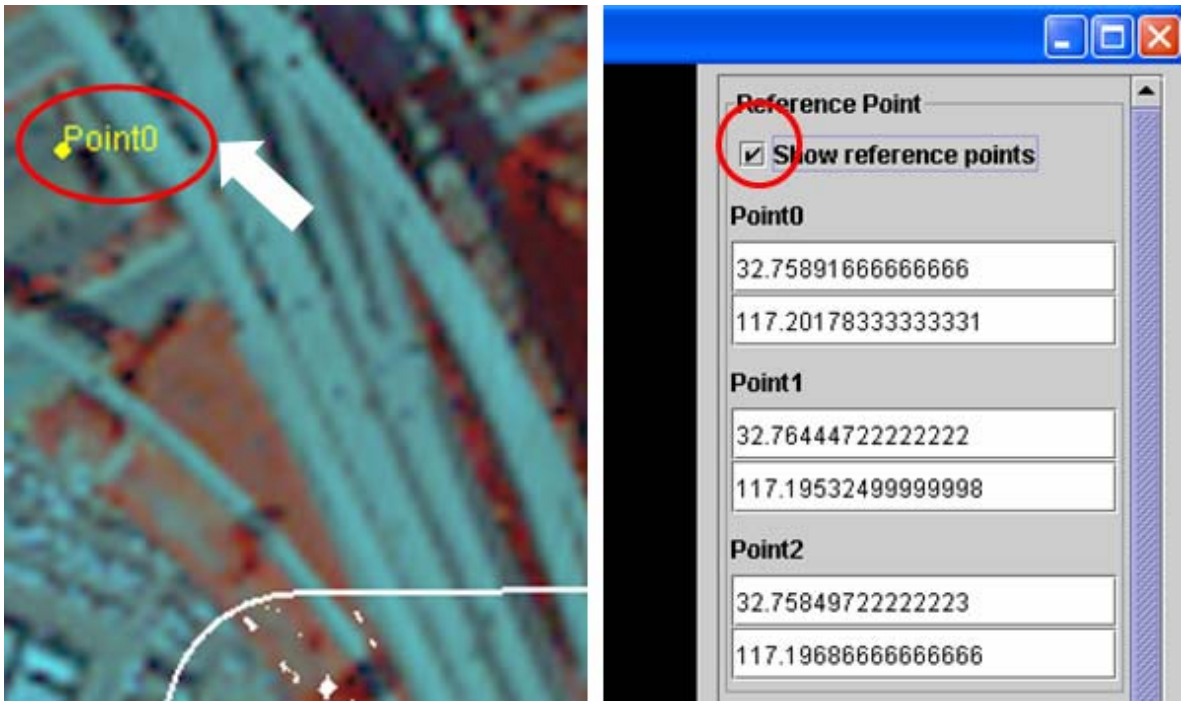


Figure 10.3 Checkbox for visibility of reference points

10.6 Change the NDVI threshold

The second interface is on the NDVI threshold. The user can change the NDVI threshold value by using the slider in the second block (see Figure 10.4). The healthy vegetation area will be updated by changing the value. Figure 10.5 shows the outputs using four different NDVI threshold values. The white pixels represent healthy vegetation points.

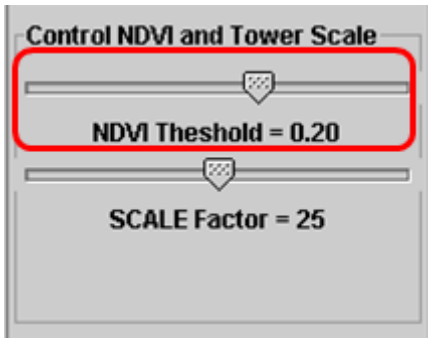


Figure 10.4 GUI of the NDVI threshold slider

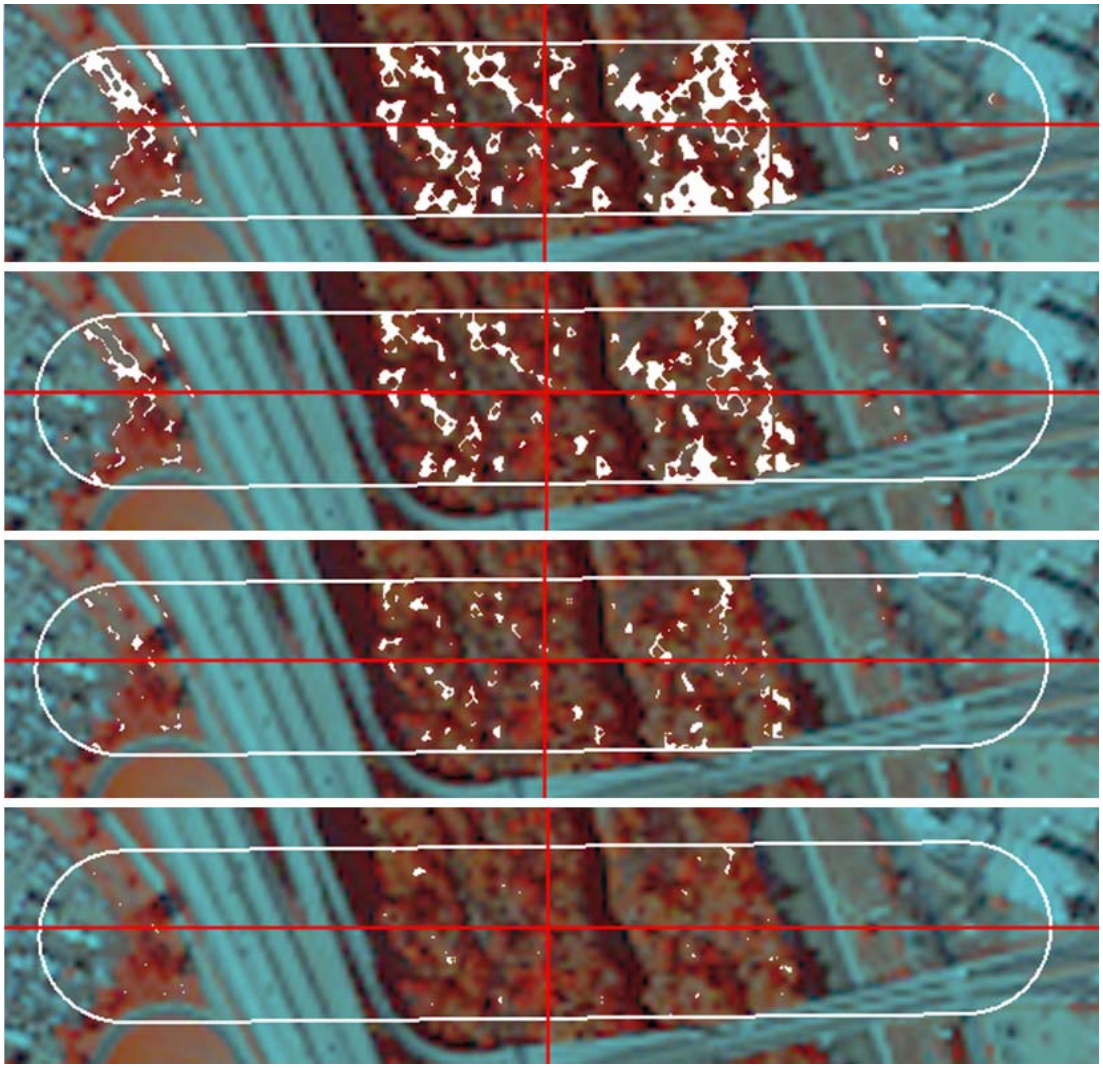


Figure 10.5 Screen shots of different NDVI threshold values: from top to bottom, a) NDVI=0.1, b) NDVI=0.15, c)NDVI=0.2, and d) NDVI=0.25

10.7 Tower location visibility

The program has a function to hide or show the transmission towers by checking the box named “Show Towers” in the third block, as shown in Figure 10.6. When the checkbox is on, all of the transmission towers are shown as red squares on the image, as shown in Figure 10.6. Under the checkbox, the information of each transmission tower is shown. The first textbox shows the latitude degree of a tower’s location, the second shows the longitude degree, and the last one shows the altitude measured by foot. For example, the latitude, longitude, and altitude are 32.75928019950938 degrees, 117.19934737424919 degrees, and 24.0 feet, respectively.

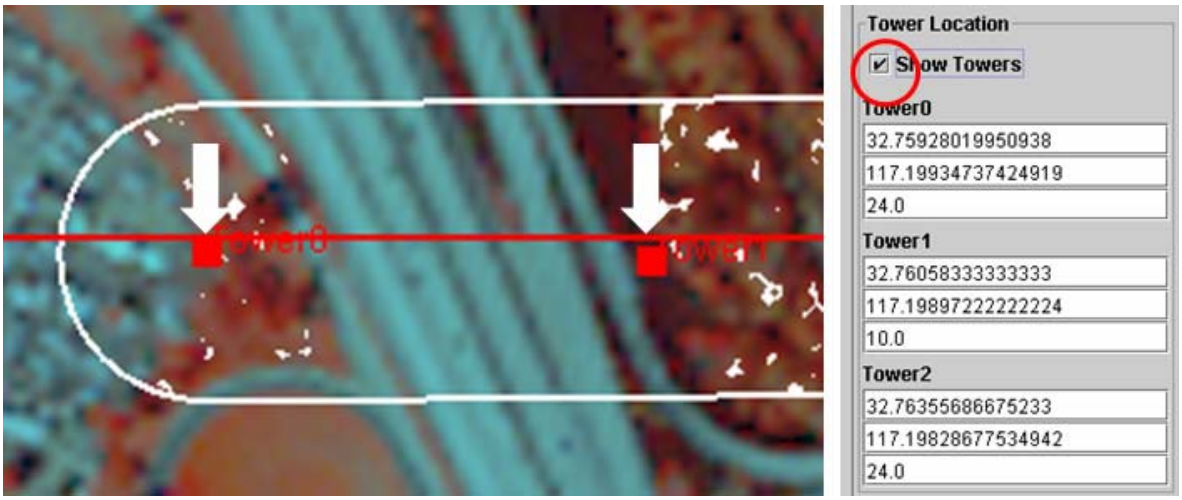


Figure 10.6 Checkbox for visibility of transmission towers

10.8 Obtaining cross sections

When the program starts, an initial cross-section point is set as the center of the danger zone. Two cross-section lines from the point are defined in the X and Y directions of the image and drawn as red lines. The height of each pixel position on the cross-section line is shown in the cross-section diagram (see Figure 10.7). The height data are from the digital elevation model (DEM) image file. The cross-section diagram parallel to the X direction (horizontal cross-section) is shown in the bottom of the image, and the one parallel to the Y direction (vertical cross-section) is on the right side. This program allows the user to change the cross section point by clicking any point in the image. When the user clicks a point in the image, the program collects the data for the cross-section from the DEM image. Figure 10.8 shows cross-sections of three different points.

In the cross-section diagram, the short vertical red lines are transmission towers, and the white lines between them are transmission lines. The towers and lines are usually not on the section line so that they are projected to the section plane. Each elevation is painted: in dark gray for outside of the danger zone, in light gray for inside of the danger zone, and in white for healthy vegetation areas inside of the danger zone, as shown in Figure 10.9

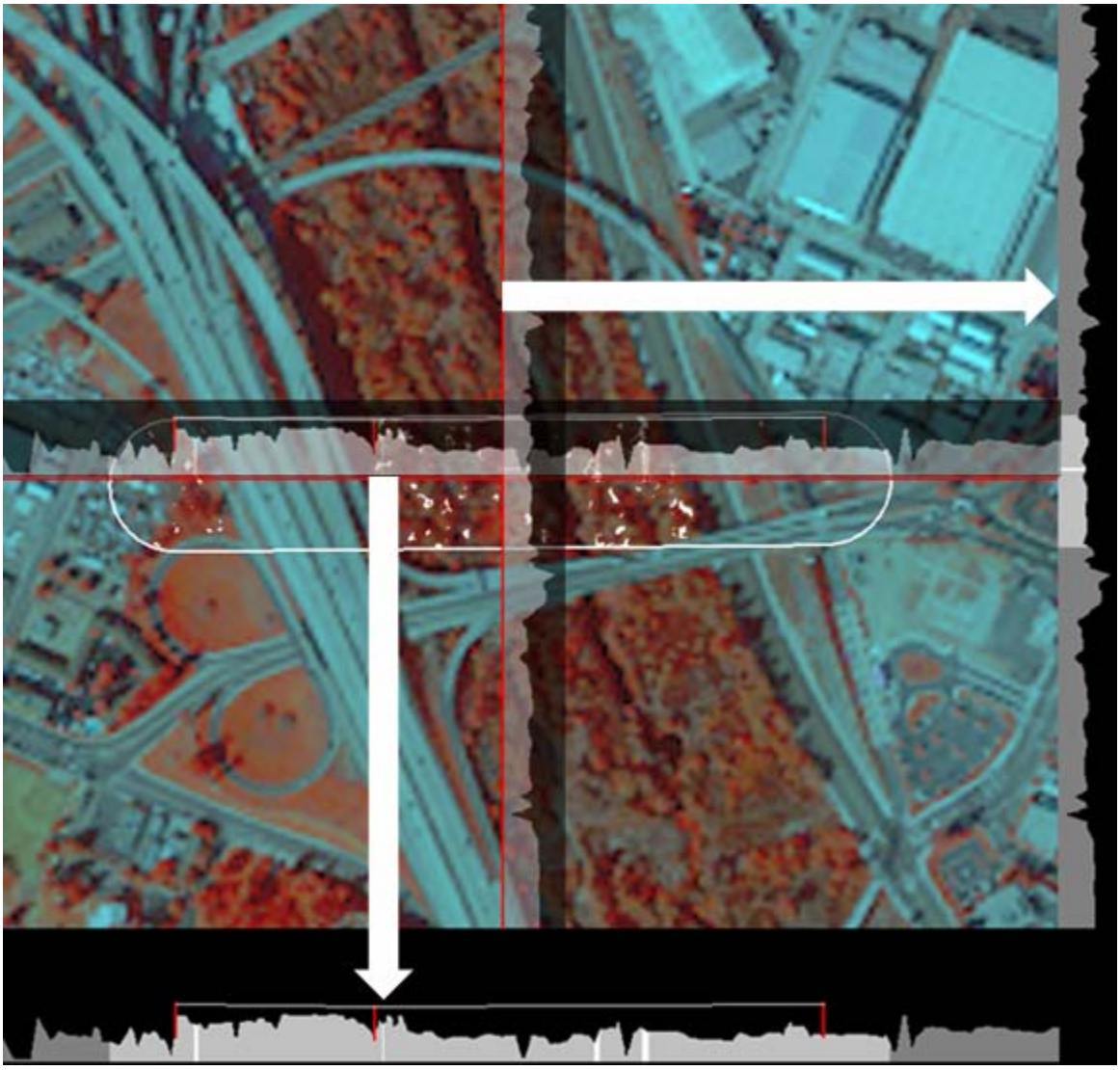


Figure 10.7 Vertical and horizontal cross-sections

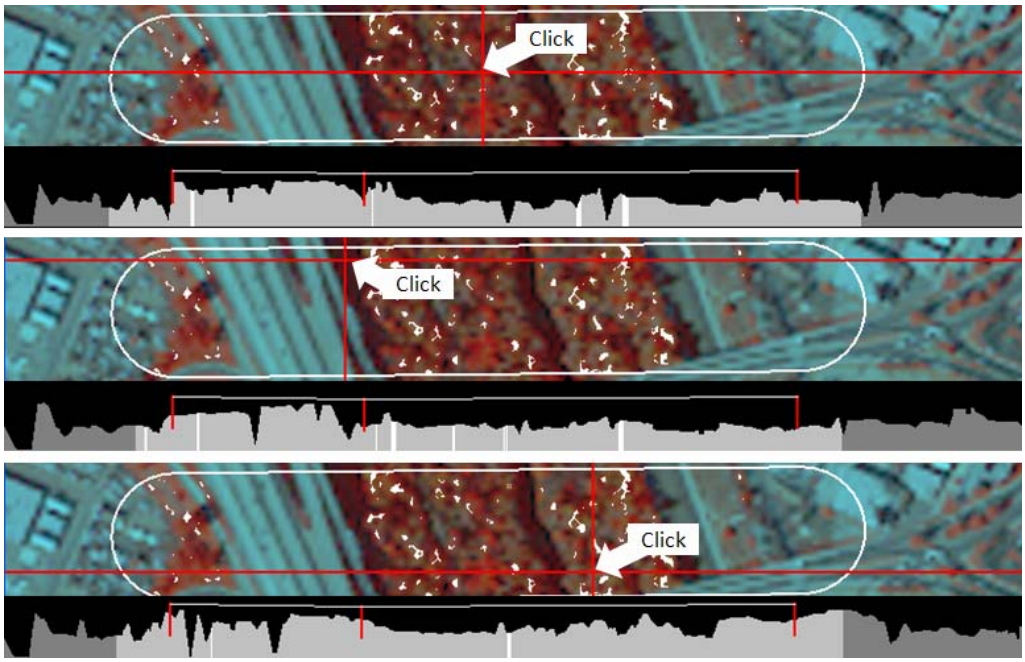


Figure 10.8 Cross-sections of different points

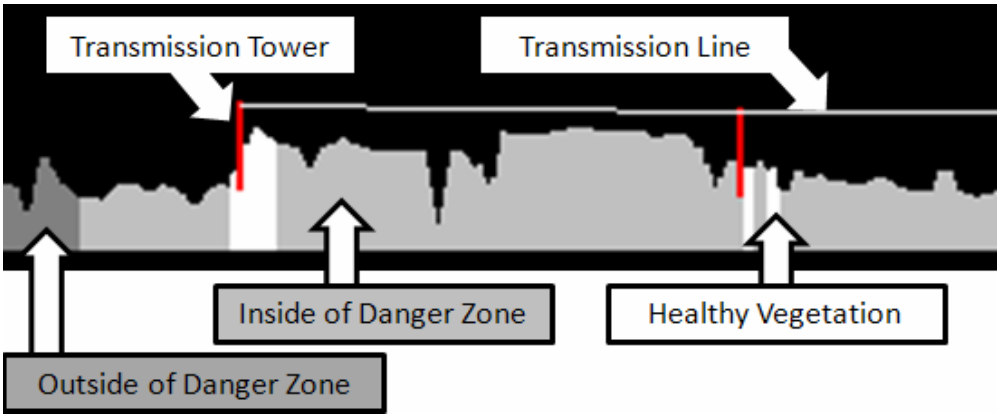


Figure 10.9 Paint rules of the cross-section

10.9 Extracted trees (healthy vegetation sets)

When the image is updated in this program, the program recalculates the location of trees (or healthy vegetation sets), and the list of trees within the location information and distance to the transmission line is updated in the last block of the right panel (see Figure 10.10). Because the text area is so large, Figure 10.11 shows only a partial list of the extracted trees. The user can scroll the vertical and horizontal slide bars to look into the data. Table 10.1 shows the top 13 trees in the list. The textural information can be copied and pasted into the other applications.

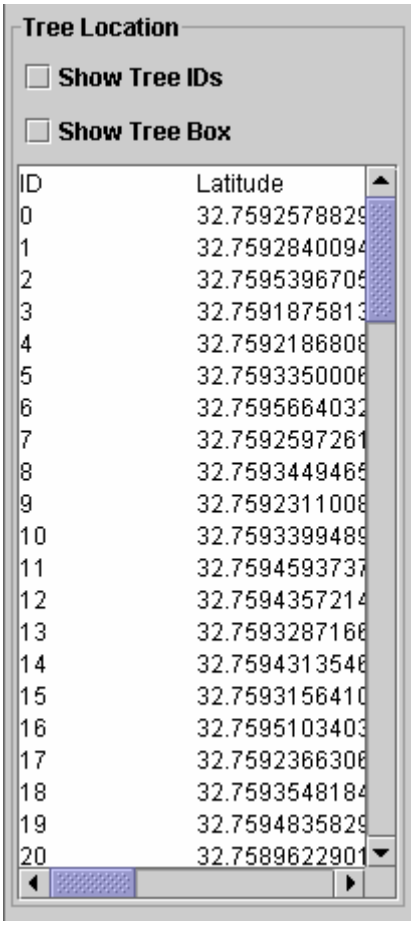


Figure 10.10 List of extracted trees

Table 10.1 Part of extracted trees

ID	Latitude	Longitude	Distance to Lines
0	32.759251042034705	117.19985426770678	21.128582
1	32.75928400942072	117.1998345690552	15.831722
2	32.75953096171785	117.19973683669522	16.415674
3	32.75918758139942	117.19980425768894	28.056217
4	32.759336868606894	117.1997209176232	26.379023
5	32.759566403240605	117.19962773072264	27.535236
6	32.7592622003266	117.1996072612092	18.496902
7	32.75929827272182	117.19955346662981	33.555576
8	32.75937975721643	117.19949891613452	14.917007
9	32.759457505829374	117.19942315017384	15.484547
10	32.75943572141182	117.19935116877471	15.484547
11	32.75932871661798	117.19931252240548	17.171558
12	32.75923663067426	117.19920341145091	35.778946
……..			

In the block there are two checkboxes named “Show Tree IDS” and “Show Tree Box.” If the “Show Tree IDS” is checked, the indices are shown in the image. The index is used in the list of Table 10.1. If the “Show Tree Box” is checked, the boundary boxes of tree are drawn, as shown in Figure 10.11. The right image shows the indices, the middle image shows the boundary boxes, and the right image shows the indices and boundary boxes together.

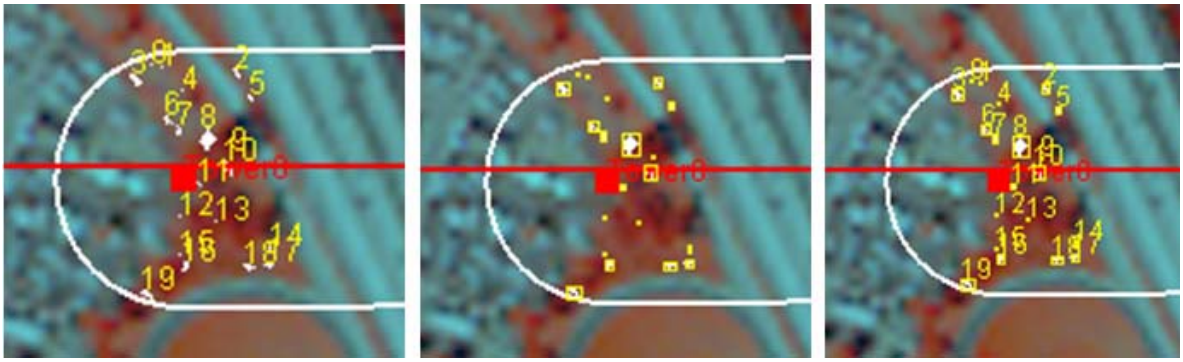


Figure 10.11 Tree IDs and boundary boxes: a) right image, b) middle image, and c) right image

10.10 Scale factor

In the same block as the NDVI threshold slider, there is another slider named “SCALE Factor.” Because of the limited information on the transmission towers, it was difficult to get the information of the towers’ heights. To support this case, the scale factor function is temporary implemented. The user can compare the section height of each point by changing the factor manually. As shown in Figure 10.12, the height of transmission towers is increased by increasing the scale factor from 20 to 40.

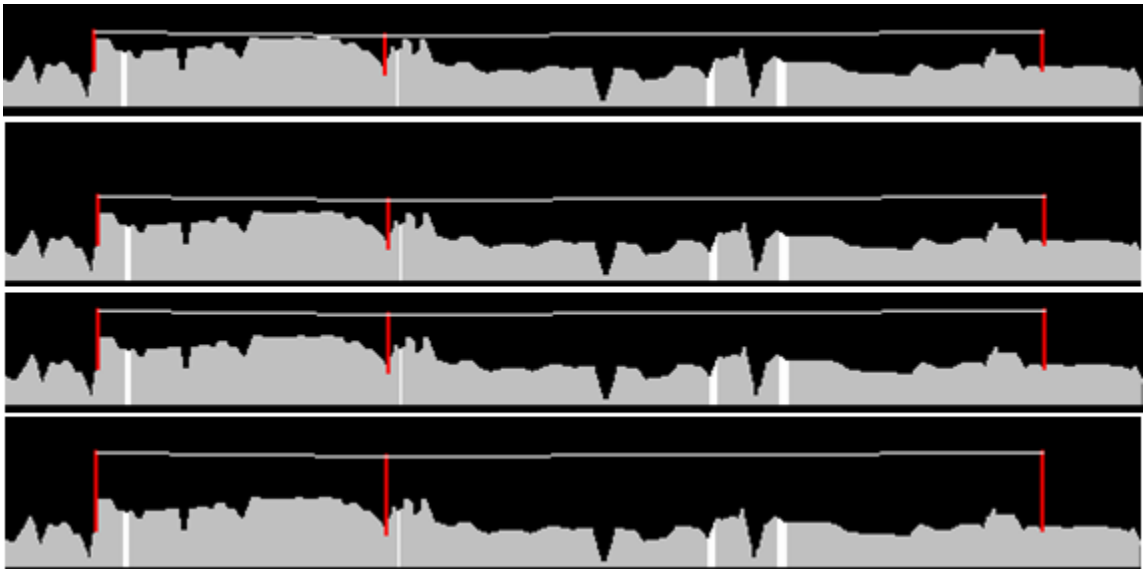


Figure 10.12 Scale factor and tower height, from top to bottom: a) scale factor=20, b) scale factor=25, c) scale factor = 30, and d) scale factor =40

11.0 Case Studies and Consideration

11.1 A problem relating to obtaining a stereo pair of satellite images

GeoEye[45] is one of the biggest companies selling stereo pairs of INOKOS satellite images. They helped a lot on finding stereo pair images with transmission lines and towers. First, they searched in Arizona, but there was no stereo pair with transmission towers and lines whose detail information we had. Second, we looked into California using Google Earth and found one site where the transmission towers and lines are recognizable though we did not have the detail information on the towers. We decided to use the images because we had no other option. We went to the site and measured the height of the towers. The details are attached in Appendix B. The following sections explain the case studies of three different NDVI thresholds using the same images.

11.2 Case 1: The NDVI threshold is set as 0.20

Figure 11.1 shows the result in the case of the NDVI threshold =0.20. The program extracted 95 trees (healthy vegetation sets) as shown in Table 11.1. The top 5 trees closest to the transmission lines are highlighted in yellow in the table. By showing the indices of trees, the top 5 trees are circled in red (See Figure 11.2). By clicking the image and changing the cross-section points, the elevation can be observed in the tool as shown in Figure 11.3.

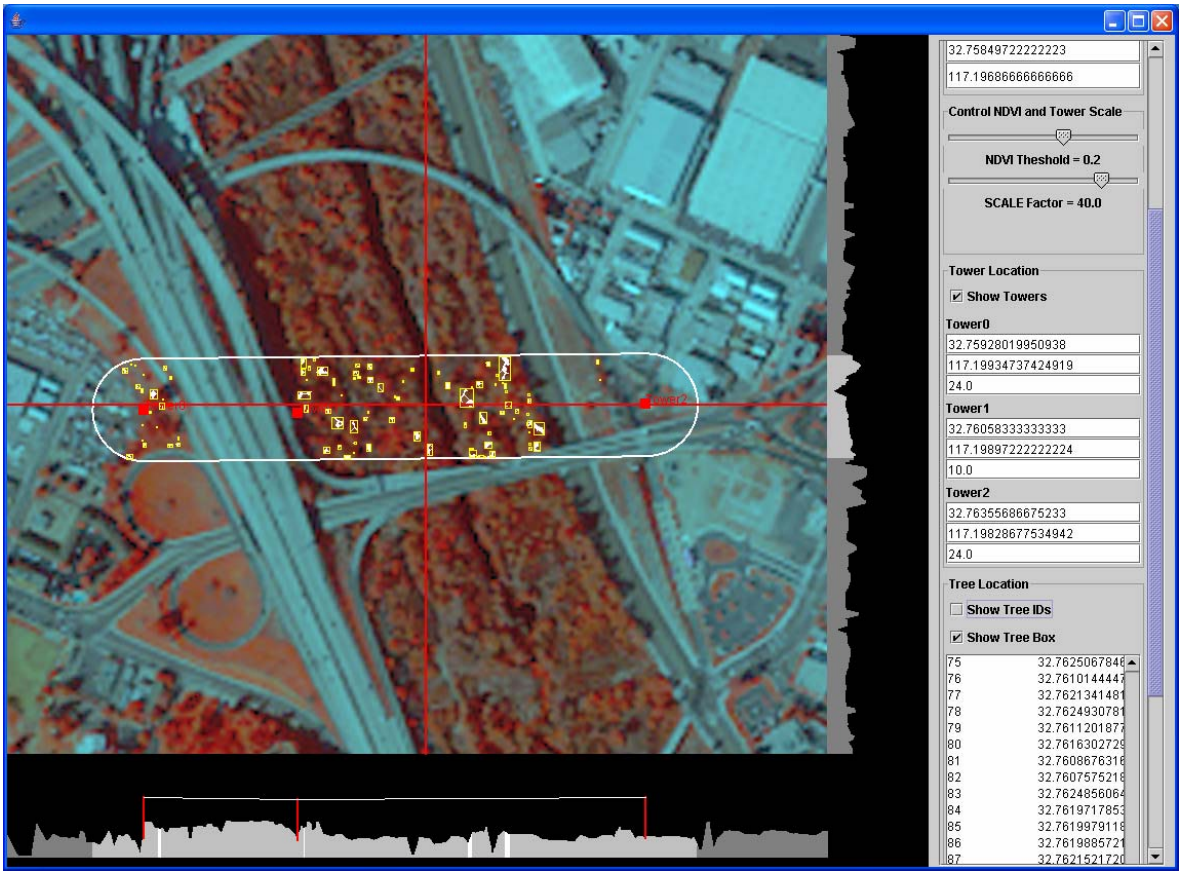


Figure 11.1 Screen shot when the NDVI=0.2

Table 11.1 Extracted tree list when the NDVI=0.2

ID	Latitude	Longitude	Distance to Lines
0	32.759251042034705	117.19985426770678	40.78245
1	32.75928400942072	117.1998345690552	42.64533
2	32.75953096171785	117.19973683669522	32.101067
3	32.75918758139942	117.19980425768894	38.711155
4	32.759336868606894	117.1997209176232	42.473488
5	32.759566403240605	117.19962773072264	31.185623
6	32.75927090915831	117.19960498847946	47.045143
7	32.75929827272182	117.19955346662981	48.571705
8	32.759388466048144	117.1994966434048	26.227222
9	32.759457505829374	117.19942315017384	32.37095
10	32.75943572141182	117.19935116877471	32.37095
11	32.75932871661798	117.19931252240548	24.45303
12	32.75923663067426	117.19920341145091	54.78344
13	32.759354818436556	117.19915037776943	31.116964
14	32.759517156553315	117.19898596538566	30.95321
15	32.75920861156173	117.19904429546051	54.78344
16	32.75921668952098	117.1989867113376	36.429234
17	32.759505948908306	117.1989223189895	30.95321
18	32.75943441031378	117.19892989309457	33.009804
19	32.75907421852783	117.19890184449406	37.527306
20	32.76241358501446	117.1989845631576	34.17917
21	32.760715993703116	117.19948305684636	29.145893
22	32.760817394674454	117.19948988001751	32.63272
23	32.76228171547041	117.1990633577639	34.511993
24	32.761173825902176	117.19934138670577	41.379597
25	32.76127646394188	117.19930350621642	41.122395
26	32.76075890698922	117.19941638180457	24.293316
27	32.76086651797896	117.19935501312011	32.202618
28	32.761621081328684	117.1991913815622	53.909477
29	32.761638498992106	117.19918683610275	57.218025
30	32.760699182235605	117.19938758725212	34.350952
31	32.761617345447014	117.1991701660968	53.909477
32	32.761634763110436	117.19916562063737	57.218025
33	32.76114146471207	117.19926107030366	42.195026
34	32.76184874802156	117.19908758692895	45.383442
35	32.76233954760663	117.19892621813679	45.87314
36	32.761160119443865	117.19921182118372	41.379597
37	32.76112341617618	117.19921030436991	42.195026
38	32.761252180711026	117.1991656056914	41.122395
39	32.76094363571944	117.19922393576624	38.93526
40	32.76071223314489	117.1993065149341	30.106556
41	32.76152712744413	117.19907166287494	39.934822
42	32.76217468600001	117.1988693849478	34.756172
43	32.76190284427611	117.19892923183644	44.49052
44	32.76237001617934	117.19884060035933	36.57298
45	32.76132432550146	117.19905801653265	40.595135
46	32.76230531847569	117.19883529400198	50.505573
47	32.76239053885197	117.19880195897208	34.17917
48	32.762101279464645	117.19886635132018	41.77956
49	32.7620371879569	117.19876102990915	41.723053
50	32.76238804003867	117.1987360398462	35.1852
51	32.760666796368945	117.19915194440313	31.11355
52	32.762574030513825	117.19865421659425	36.89915
53	32.760980314310565	117.19907012613318	36.23942
54	32.76142943767791	117.1989307287223	40.198345
55	32.76145369623221	117.19891330280045	40.315865
56	32.762562822868816	117.19859057019809	36.89915
57	32.76238117447124	117.19859359386173	35.795574
58	32.76065993080151	117.19900949841868	30.620935
59	32.760837212444955	117.19892994789645	37.52764
60	32.76243963747992	117.1985117656278	36.6257
61	32.76117498894088	117.19883070370464	40.6934
62	32.762315845895124	117.19853297611121	35.651165
63	32.76214103838842	117.19852311931243	33.974323

ID	Latitude	Longitude	Distance to Lines
64	32.762529830806244	117.19845494240279	37.079998
65	32.760895044581176	117.19879280826933	36.99898
66	32.761137654800734	117.19877387549764	40.61773
67	32.7613622164844	117.1987041767922	39.41286
68	32.762400435398945	117.19844432968812	36.94704
69	32.76102691412522	117.19871401366302	42.34919
70	32.762530437002155	117.1983549273491	36.880646
71	32.76197991266348	117.19848750158883	41.723053
72	32.76250617844786	117.19837235327097	37.079998
73	32.76155005022384	117.19847763482608	50.72802
74	32.76216527226617	117.1983503669437	34.511993
75	32.76250678464376	117.19827233821728	36.953205
76	32.76101444473528	117.19853974448047	42.195026
77	32.76213414814443	117.19822534688109	34.511993
78	32.76249307818545	117.19814277269523	36.89915
79	32.76112018778419	117.19846776806332	40.6934
80	32.76163027297352	117.1983124615444	40.65263
81	32.76086763166455	117.19853367722524	36.32414
82	32.76075752186151	117.19852912678381	37.142754
83	32.76248560642211	117.19810034176446	36.89915
84	32.761971785351115	117.19823443281798	45.823933
85	32.76199791184625	117.19822761462882	36.9607
86	32.76198857214207	117.19817457596535	39.317696
87	32.762152172003766	117.19812078636797	54.404366
88	32.76098021560434	117.19844882034567	42.34919
89	32.7609186229099	117.19840941806054	40.984272
90	32.762052638973266	117.19812457092951	34.162834
91	32.76208747430011	117.19811548001063	34.275566
92	32.76321729634524	117.19884139114916	40.61274
93	32.763202959014464	117.19865651423392	39.785957
94	32.762599476783386	117.19828143411814	39.86619
95	32.76255030412577	117.1981056481218	38.642048

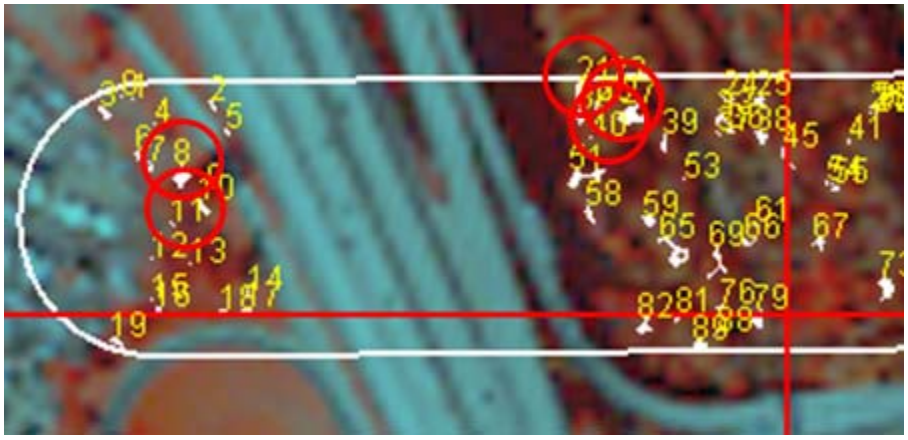


Figure 11.2 Top 5 trees closet to transmission lines in the extracted tree list

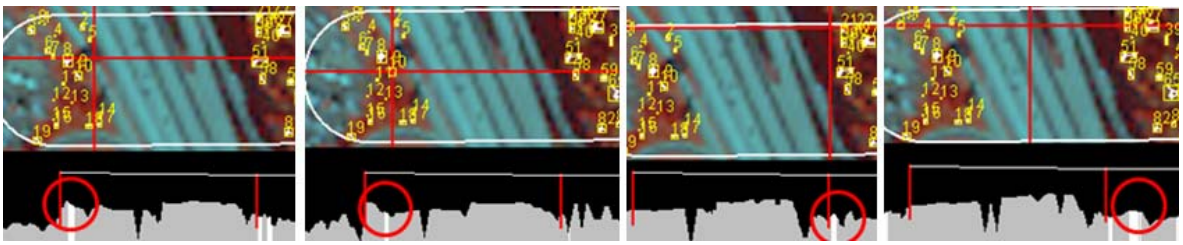


Figure 11.3 Cross-sections for the top five trees in the extracted tree list from left to right: a) no. 8, b) no. 11, c) no. 21, and d) nos. 26 and 40

11.3 Case 2: The NDVI threshold is set as 0.24

This section shows the result the the NDVI=0.24. Figure 11.4 shows the screen shot of the case. The program extracts 37 trees as shown in Table 11.2. The index number is different from the case in Section 11.2.

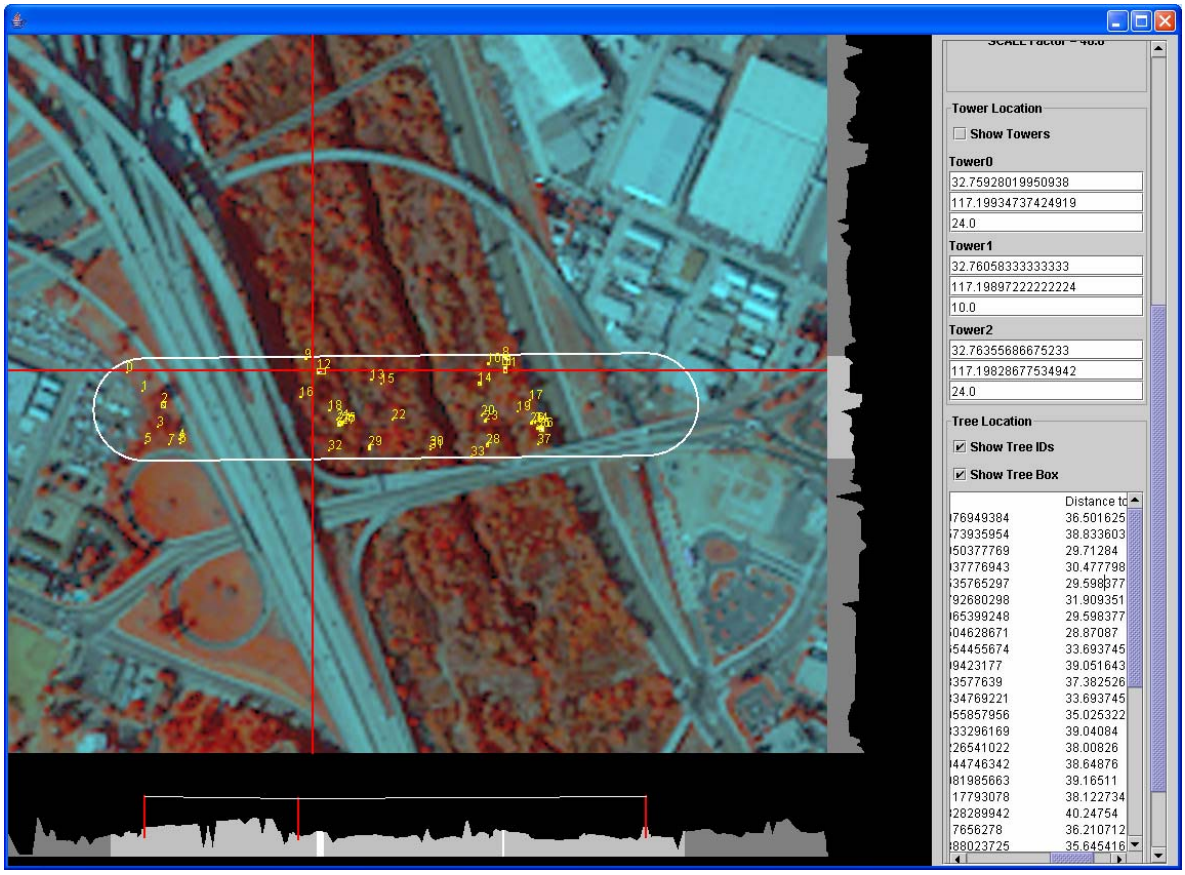


Figure 11.4 Screen shot when the NDVI = 0.24

Table 11.2 Extracted tree list when the NDVI=0.24

ID	Latitude	Longitude	Distance to Lines
0	32.75919255434946	117.19978076949384	36.501625
1	32.75928956389011	117.19955573935954	38.833603
2	32.759446298184365	117.19935950377769	29.71284
3	32.759354818436556	117.19915037776943	30.477798
4	32.759515288612484	117.19897535765297	29.598377
5	32.75922042540265	117.19900792680298	31.909351
6	32.75951652568085	117.19893065399248	29.598377
7	32.75941886059119	117.19894504628671	28.87087
8	32.76243536943201	117.19905654455674	33.693745
9	32.760740883129884	117.1995209423177	39.051643
10	32.76228171547041	117.1990633577639	37.382526
11	32.76240984913279	117.19896334769221	33.693745
12	32.76084910031554	117.19935955857956	35.025322
13	32.76126088954273	117.19916333296169	39.04084
14	32.76216784510913	117.19888226541022	38.00826
15	32.7613317972648	117.19910044746342	38.64876
16	32.760628225160424	117.19913981985663	39.16511
17	32.76256469080965	117.19860117793078	38.122734
18	32.7608477892175	117.19893828289942	40.24754
19	32.76243963747992	117.1985117656278	36.210712
20	32.76212735660667	117.19854888023725	35.645416
21	32.76089193957196	117.19882690419713	44.39688
22	32.7613622164844	117.1987041767922	39.89863
23	32.7621509842885	117.19847614292222	35.975994
24	32.76090001753121	117.19876932007422	42.838875
25	32.76092987990802	117.19878371735045	41.9205
26	32.76253914583387	117.19835265461938	41.42266
27	32.76092614402635	117.19876250188506	41.921906
28	32.76212543931272	117.19822761961082	37.036175
29	32.76111831984336	117.19845716033063	39.21911
30	32.76163587679603	117.19834428474249	38.45661
31	32.76163027297352	117.1983124615444	35.559334
32	32.76076623069322	117.19852685405408	38.287457
33	32.76196555065615	117.19814729822671	35.79366
34	32.76256714026984	117.19835644416291	42.380756
35	32.762582059119964	117.19828597957759	42.91972
36	32.762615026505976	117.19826628092602	43.02298
37	32.76255404000744	117.19812686358718	42.91972

11.4 Case 3: NDVI threshold is set as 0.15

This section shows the result when the NDVI=0.15. Figure 11.5 shows the screen shot of the case. The program extracts 104 trees as shown in Table 11.3.

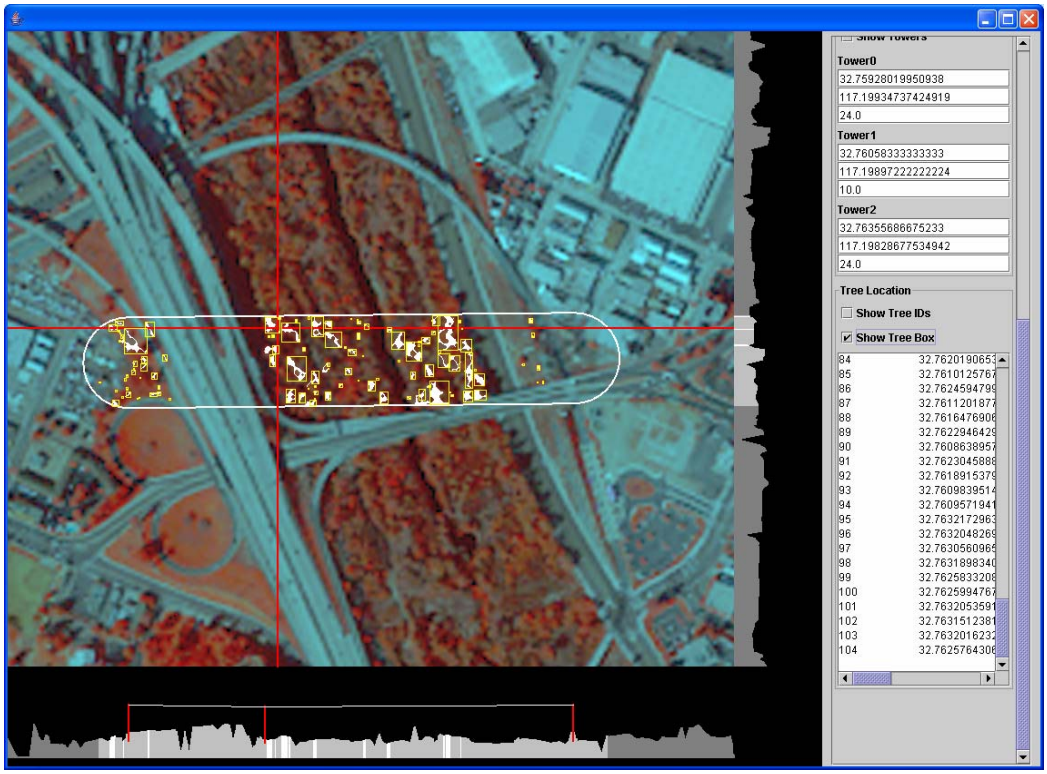


Figure 11.5 Screen shot when the NDVI=0.15

Table 11.3 Extracted tree list when the NDVI=0.15

ID	Latitude	Longitude	Distance to Lines
0	32.75926659175729	117.19983911451465	37.14668
1	32.75954961644965	117.19968758757528	28.99467
2	32.75919629023113	117.19980198495922	36.501625
3	32.7593859919114	117.1995860507258	30.971508
4	32.75924978028978	117.1997436449204	37.047302
5	32.759254753239816	117.19972015672529	37.047302
6	32.7592646991399	117.19967318033507	36.99273
7	32.7592379417723	117.19962468713105	37.047302
8	32.75963105159115	117.19932238418623	28.883392
9	32.75943572141182	117.19935116877471	29.71284
10	32.759339293390525	117.19932085740845	30.926754
11	32.75941020111259	117.19925797191019	30.04827
12	32.759312536022925	117.19927236420442	31.015814
13	32.759305064259586	117.19922993327364	31.015814
14	32.759497895625614	117.19913522955929	28.972927
15	32.75923663067426	117.19920341145091	37.94611
16	32.759354818436556	117.19915037776943	30.477798
17	32.7592117412475	117.19916552597958	38.833603
18	32.75896229013706	117.19919733921368	38.169918
19	32.759517156553315	117.19898596538566	29.517462
20	32.75921047950257	117.1990549031932	32.967175
21	32.759100369699524	117.19905035275177	37.34832
22	32.75920611274844	117.19897837633462	32.381893
23	32.759504080967474	117.1989117112568	29.598377
24	32.75944996003636	117.19891473990243	28.827662
25	32.759148255935656	117.19896018951486	37.121777
26	32.759305014906474	117.19891928037988	30.577312
27	32.75928572930221	117.19891321810664	30.796442
28	32.759597379303	117.19882079210399	29.08852
29	32.75906737763695	117.19891472495647	35.559578
30	32.762311577847214	117.19907775504014	41.152702
31	32.76239864148778	117.19889970129604	33.693745
32	32.76109794523007	117.19942776039915	40.448616
33	32.76115764530712	117.19930122850471	38.956085
34	32.76073464843491	117.19943380772642	38.838875
35	32.76080868584274	117.19949215274724	37.45457
36	32.761496683547975	117.19931260709929	41.81247
37	32.760970417763595	117.19942775541715	45.72044
38	32.76155577742912	117.19928609025854	41.02403
39	32.76253613953089	117.19900805633469	36.180367
40	32.76126588716934	117.19929517121345	39.21911
41	32.76156759127004	117.19924972160102	40.93918
42	32.76089388154247	117.19930349127046	36.306488
43	32.76162792221956	117.19917850109978	39.20672
44	32.76184003918985	117.19908985965868	43.146843
45	32.76253861366763	117.1989186490137	36.285725
46	32.76225930018039	117.19893606497158	37.382526
47	32.761288253106244	117.19911181111202	38.159443
48	32.761904712216946	117.19893983956915	40.709908
49	32.7615308633258	117.19909287834034	40.93918
50	32.76071223314489	117.1993065149341	38.731873
51	32.76067552987721	117.19930499812028	39.409515
52	32.7621659771683	117.19887165767751	37.076923
53	32.76254666695032	117.19870573844392	36.982246
54	32.762101279464645	117.19886635132018	35.79366
55	32.76203034706602	117.19877391037156	29.860455
56	32.762093807701305	117.1988239203894	35.79366

ID	Latitude	Longitude	Distance to Lines
57	32.7607103405275	117.19914058075453	38.450935
58	32.76138465645098	117.1989867960314	39.841084
59	32.760980314310565	117.19907012613318	41.9205
60	32.76143627856879	117.1989178482599	41.81247
61	32.76227419435396	117.19871027393938	34.233295
62	32.762522359042904	117.19841251147201	37.703766
63	32.762299059104166	117.19859283296384	34.515533
64	32.76066179874235	117.19902010615137	38.731873
65	32.761533312785986	117.19884814457247	40.45318
66	32.762397961262195	117.1985337370091	35.56777
67	32.760871416899334	117.19886554558438	45.72044
68	32.76187045840945	117.19869358898747	40.709908
69	32.76118369777259	117.19882843097493	39.186565
70	32.76213917044759	117.19851251157974	35.65512
71	32.76117622600925	117.19878600004415	39.186565
72	32.7613622164844	117.1987041767922	39.89863
73	32.76103562295693	117.1987117409333	39.735664
74	32.76197120383176	117.19848977431855	38.721844
75	32.76162408763167	117.1985359798469	40.138313
76	32.76143933422488	117.19857309943833	40.93918
77	32.761558759055546	117.19847536209636	39.460114
78	32.76218826907554	117.19822231823547	39.580585
79	32.76126265877735	117.19855263490687	39.461567
80	32.76101694354858	117.19860566360636	39.442535
81	32.762105547512554	117.19832157239125	35.543816
82	32.76094353701322	117.19860262997874	40.82066
83	32.760763125684015	117.19856094998188	36.306488
84	32.76201906539134	117.19824428463477	39.64527
85	32.76101257679445	117.19852913674778	39.15861
86	32.762459479926974	117.19810715995362	37.2576
87	32.76112018778419	117.19846776806332	39.186565
88	32.76164769063694	117.19830791608497	40.261955
89	32.76229464299691	117.1982056532115	34.115265
90	32.76086389578288	117.19851246175985	41.777996
91	32.762304588897	117.1981586768213	35.29792
92	32.761891537924875	117.19824427965278	40.93162
93	32.76098395148601	117.19847003581106	39.766853
94	32.76095719411841	117.19842154260704	39.777363
95	32.76321729634524	117.19884139114916	38.31733
96	32.7632048269553	117.19866712196662	38.41857
97	32.763056096590624	117.19833979408493	38.135735
98	32.76318983407551	117.19827160721131	39.50761
99	32.76258332086489	117.19839660236397	42.5644
100	32.762599476783386	117.19828143411814	43.023823
101	32.76320535912154	117.1981011275723	40.336445
102	32.76315123819043	117.19810415621792	38.93403
103	32.76320162323987	117.1980799121069	40.336445
104	32.7625764306209	117.19809882993263	43.067863

11.5 Further discussion and considerations

The images around San Diego, California were used in the case studies because of the limitation of stereo image availability elsewhere (see Section 11.1). Three case studies with three different NDVI threshold values were explained in Sections 11.2, 11.3 and 11. 4. When the NDVI threshold value was set as 0.20, 95 trees (healthy vegetation regions) were extracted. When the NDVI was 0.15, 37 trees were extracted. When the NDVI was 0.24, 104 trees were

extracted. When the NDVI=0.15, too few trees were extracted to recognize. When the NDVI=0.24, too many trees were extracted to identify the locations clearly. Therefore, it was decided to investigate the extracted trees when the NDVI =0.20.

The closest five trees to the transmission lines were selected from the list of all extracted trees and displayed in Table 11.4. The locations are considered to have a priority to be observed if they are endangering the transmission lines.

Table 11.4 Closest five trees to transmission lines

ID in Table 11.1	Latitude	Longitude	Distance to Lines
26	32.76075890698922	117.19941638180457	24.293316
11	32.75932871661798	117.19931252240548	24.45303
8	32.759388466048144	117.1994966434048	26.227222
21	32.760715993703116	117.19948305684636	29.145893
40	32.76071223314489	117.1993065149341	30.106556

12.0 Conclusions, Recommendations, and Future Work

12.1 Conclusions

This report presented a literature review about transmission towers and lines and about remote sensing and photogrammetry on identifying healthy vegetation areas and extracting object heights. We developed a framework for implementing tools to extract trees interfering with overhead power lines. There are 3 stages in this framework:

Stage 1. Transmission line scanning program. This is a tool to extract healthy vegetation areas close to the power lines from multispectral satellite images.

Stage 2. Tall tree identification program. This is a tool to generate digital surface models from a stereo pair of multispectral satellite images.

Stage 3. Integrate the tools of Stages 1 and 2, and develop a system to identify high trees interfering with overhead power lines.

The tool for Stage 1 was implemented and tested in the first year. The interactive function helped on finding the healthy vegetation area. A test for generating a digital surface model (DSM) from a stereo pair of satellite images using one of the commercial off-the-shelf Photogrammetry packages, ERDAS IMAGINE, was demonstrated.

In the second year, the implementations of stages 2 and 3 were completed. One stereo pair of IKONOS satellite images in San Diego, CA provided by GeoEye was used for testing.

Case studies using three different NDVI thresholds were explained and considered. The case studies proved that the developed programs automatically are able to identify trees endangering the transmission line. However, the program was not tested in a utility environment. Thus, this is not a commercial grade program. It requires further testing and development.

12.2 Future work

The following is recommended future work to improve the tool developed:

- Test using the other image files of different larger site such as 60 miles.
- Compare the results from commercial off-the-shelf photogrammetry packages.
- Visualize the DEM data in 3D view.
- Add query functions to the extracted trees
- Add more control units to allow the users to change the attributes of transmission towers and lines flexibly and interactively.

References

[1] WIKIPDEIA, http://en.wikipedia.org/wiki/Main_Page, Aug. 10, 2006.

[2] Ki In Bang, Soo Jeong, Kyung-Ok Kim, Woosug Cho, "Automatic DEM generation using IKONOS stereo imagery," Proc. International Geoscience and Remote Sensing Symposium, vol. 7, no. 21-25, July 2003, pp. 4289 – 4291.

[3] T. Toutin, "Comparison of stereo-extracted DTM from different high-resolution sensors: SPOT-5, EROS-a, IKONOS-II, and QuickBird," IEEE Transactions on Geoscience and Remote Sensing, vol. 42, no. 10, pp. 2121 – 2129.

[4] SPACE IMAGING, http://www.spaceimaging.com/products/ikonos/index.htm, Aug. 10, 2006.

[5] SATELLITE IMAGE CORPORATION, http://www.satimagingcorp.com/gallery-quickbird.html, Aug. 10, 2006.

[6] T. Krauss, M. Lehner, P. Reinartz, U. Stilla, "Comparison of DSM generation methods on IKONOS images," M. Moeller, (editor.), Urban Remote Sensing – Photogrammetrie – Fernerkundung – Geo information special issue, 04/2006.

[7] S. Kocaman, L. Zhang, A. Gruen, D. Poli, "3D city modeling from high-resolution satellite images," ISPRS Ankara Workshop 2006, Ankara, Turkey, Feb. 2006.

[8] ORBIMAGE,http://www.orbimage.com/corp/orbimage_system/satellite.html, Aug 10, 2006

[9] F. Morsdorf, E. Meier, B. Kötz, K. I. Itten, M. Dobbertin, B. Allgöwer, "LIDAR-based geometric reconstruction of boreal type forest stands at single tree level for forest and wildland fire management," Remote Sensing of Environment, vol. 92, no. 3, Aug. 2004, pp 353-362.

[10] Toposys, "Lidar Applications – Pipelines and Trans-mission Lines," http://www.toposys.com/pdf-ext/Engl/AP_Corridor.pdf, Aug 10, 2006.

[11] D. Turton, P. Jonas, "Airborne laser scanning –cost effective spatial data", Map Asia Conference, Kuala Lumpur, Malaysia, October, 2003.

[12] G. Borgefors, T. Brandtberg, F. Walter, "Forest parameter extraction from airborne sensors," International Archives of Remote Sensing and Photogrammetry, vol. 32, part 3-2W5, pp. 151-157.

[13] J. Hyyppä, H. Hyyppä, G. Ruppert, "Automatic derivation of features related to forest stand attributes using laser scanner data," International Archives of Photogrammetry and Remote Sensing, vol. 33, 2002, part B3, pp. 421-428.

[14] R. J. Pollock, “A model-based approach to automatically locating tree crowns in high spatial resolution images,” J. Desachy (ed.), Proc. 1994 SPIE Conference on Image and Signal Processing for Remote Sensing, (SPIE) Vancouver, BC, Canada.

[15] B. M. Straub, C. Heipke, “Automatic extraction of trees 3D city models from images and height data” Proc. of Conference on Automatic Extraction of Man-made Objects From Aerial and Space Images (III), 2001, Zurich, Switzerland.

[16] Yamagish, Y. and Yasuoka, Y, http://yasulab.iis.u-tokyo.ac.jp/2005/IIS_OH/2005_yamagishi.pdf;dated 2005.

[17] M. Teraoka, M. Setojima, Y. Imai, Y. Yasuoka, “Spatial, temporal, and spectral data analysis for environmental and disaster assessment,” (In Japanese), http://yasulab.iis.u-tokyo.ac.jp/2004/IIS_OH/2004_teraoka.pdf; dated 2004.

[18] GeoTIFF, http://www.remotesensing.org/geotiff/geotiff.html, Aug 10, 2006.

[19] Google Earth, http://earth.google.com/, Aug. 10, 2006.

[20] JAVA, http://java.sun.com/, Aug. 10, 2006.

[21] Rodrigues, L. H., 2001. *Building imaging applications with Java technology: using AWT image, Java2D, and Java advanced imaging (JAI),* Addison-Wesley, Boston.

[22] Q. Yang, J. P. Snyder, W. R. Tober, *Map projection transformation: principles and applications.*, Taylor and Francis, Hong Kong, 2000.

[23] ERDAS IMAGINE, http://gi.leica-geosystems.com/LGISub1x33x0.aspx, Aug. 10, 2006.

[24] Z. Islam, G. Metternicht, “Fuzzy approach to mapping tree crowns and species from a forest area using high resolution multispectral data’, Asian Journal of Geoinformatics, vol. 41, no. 1, September 2003.

[25] ERDAS IMAGINE, Subpixel Classifier, http://gi.leica-geosystems.com/-LGISub24x0x118.aspx, Aug. 10, 2006.

[26] Applied Analysis Inc, http://www.discover-aai.com/applications, Aug. 10, 2006.

[27] Remote Sensing Tutorial, http://obelia.jde.aca.mmu.ac.uk/giscons/rstut/tutorial/-chap5/c5p7e.html, Aug. 10, 2006.

[28] Standard FAC-003-0 — Vegetation Management Program. Adopted by NERC Board of Trustees: February 8, 2005 1 of 3;Effective Date: April 1 2005.

[29] Code of Code of Practice for Electric Line Clearance [Vegetable] Victoria Government Gazette No 9169, Thursday 25 November, 1990, Southbank, Victoria, Australia.

[30] Vegetation Management Task Force (VMTF). 1996. Environmental Stewardship Strategy for Electric Utility Rights-of-Way. Edison Electrical Institute, Washington, DC. 9 pp.

[31] PG&E Transmission Line Vegetation Management Project; PGE Transmission line Project 2004.doc.

[32] Trees and Power Lines Right-of-Way Safety. Western Area Power Administration: http://www.wapa.gov.

[33] BPA Vegetation Program. Bonneville Power Administration. DOE / BP-3712; www.bpa.gov.

[34] Richard H. Yahner1 and Russell J. Hutnik: Integrated vegetation management on an Electric transmission right-of-way in Pennsylvania, U.S. Journal of Arboriculture vol. 30, no. 5, September 2004.

[35] Joseph A. Sulak and J. James Kielbaso, "Vegetation management along transmission utility lines in the United States and Canada," Journal of Arboriculture vol. 26, no. 4, July 2000.

[36] Bramble, W.C., and W.R. Byrnes. 1996. Integrated vegetation management of an electric utility right-of way ecosystem. Down to Earth, vol. 51, no. 1, pp. 29–34.

[37] Bramble, W.C., W.R. Byrnes, R.J. Hutnik, and S.A. Liscinsky, "Prediction of cover type on rights-of-way after maintenance treatments," J. Arboric., vol. 17, pp. 38–43, 1996.

[38] DuPont Co., "Brush control that is effective, economical, and environmentally responsible," p. 12, DuPont Vegetation Management, 1996.

[39] Luken, J.O., S.W. Beiting, and R.L. Kumler, "Target/non-target effects of herbicides in power line corridor vegetation," J. Arboric., 1993, vol. 19, pp. 299–302.

[40] Luken, J.O., A.C. Hinton, and D.G. Baker, "Assessment of frequent cutting as a plant-community management technique in power-line corridors," Environ. Manage., vol. 15, pp. 81–388, 1991.

[41] Bramble, W.C., W.R. Byrnes, R.J. Hutnick, and S.A. Liscinsky. 1991. "Prediction of cover type on rights-of-way after maintenance treatments," J. Arboric. vol. 17, pp. 38–43.

[42] Fonseca, J.R., Tan, A.L., Silva, R.P., Monassi, V., Assuncao, L.A.R., Junqueira, W.S., Melo, M.O.C. "Effects of agricultural fires on the performance of overhead transmission lines," IEEE Transactions on Power Delivery, vol. 5, no. 2, pp: 687-694, Apr 1990.

[44] Niering, W.A., and R. Goodwin, "Creation of relatively stable shrub lands with herbicides: Arresting "succession" on rights-of-way and pastureland," Ecology, vol. 55, 1974, pp. 784–795.

[45] GeoEye, http://www.geoeye.com/, Aug. 10, 2006.

[46] "Reliability Based Vegetation Management Through Intelligent System Monitoring" (T-27) is now available for viewing by PSERC members. The document number is 07-31 and it is in the Reports for Members Viewing Only Folder on the PSERC website for the next 90 days. The Executive Summary is publicly available.

Appendix A: The Source Code Utilized

This appendix includes Java source code of the following class definitions in package PSERC:

Stage 1

- PL_ScrollController
- PL_DataTable
- PL_FileLoader
- PL_FilterSlider
- PL_Geo
- PL_ImageCanvas
- PL_MainFrame
- PL_RenderedImageCanvas
- PL_RenderGrid
- PL_ScrollGUI

Stage 2

- PL_StereoTest

Stage 3

- PL_Output
- PL_OutputControlPanel
- PL_OutputHistorgraph

```java
/**
* An interface to control the scroll event especailly for PL_ScrollGUI.
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi.
 * @version 1.0.
 */
package pserc;
import java.awt.*;

public interface PL_ScrollController  {

  /** Gets the panOffset property.
    * @param panOffset the offset by which the currently displayed image is moved
    * from the previous position.
    */
  public void setPanOffset(Point panOffset);

  /** Returns the panOffset property.
    * @return the panOffset.
    */
  public Point getPanOffset();
    /**Starts the scroll and sets the anchor point.
   * @param x the x coordinate of the scroll anchor.
   * @param y the y coordinate of the scroll anchor.
   **/
  public void startScroll(int x, int y);

  /**Scrolls the image.
   * @param x the x coordinate of the current position.
   * @param y the y coordinate of the current position.
   **/
  public void scroll(int x, int y);

  /** Stops scroll.
    */
  public void stopScroll();
}
```

```
593
594
595         // seach again to get the starting point for each region
596         for(int i=0; i<_h; i++){
597          for(int j=0; j<_w; j++){
598            if(Fij[i*_w+j]>0){                  // Hit a region
599                int tempID=Fij[i*_w+j];
600                Fij[i*_w+j]=tempT[tempID];      // Assign the regionID to Fij
601                Integer segIDInte=new Integer(tempT[tempID]);
602                if(!idV.contains(segIDInte)) {
603                  idV.addElement(segIDInte);    // add IDs to the list
604                  getSegPolygon(Fij, j, i, _ti, _tj); // call sub function to extract polygon (or
    boundary box)
605                }
606            }
607          }
608         }
609     }
610      // sub function called by constructor to set up ROI data
611      private void addEnvelop(double rad){
612        Roi=null;
613        double x0=133;  double y0=365;  // first transmission tower location
614        double x1=623;  double y1=360;  // second transmission tower location
615        double xv=x1-x0;  // get the distance X in two towers
616        double yv=y1-y0;  // get the distance Y in two towers
617        double length=Math.sqrt(xv*xv + yv*yv);  // Lenght between two towers
618
619        if(length==0) return;
620        double cos=(xv*1.0)/length; // Dot/Inner/scalar PRODUCT : A(xv, yv)xB(1,0)=cosAB*lengthA*lengthB
621        if(cos>1.0) cos =1.0;
622        if(cos<-1.0) cos=-1.0;
623        double angle=Math.acos(cos); // get angle from cosine value
624        if(yv<0) angle=-angle;       // refine the regative value of angle
625
626        // Create a polygon envelop rectangle
627        Polygon p= new Polygon();
628        p.addPoint((int)x0, (int)(y0-rad));
629        p.addPoint((int)(x0+length), (int)(y0-rad));
630        p.addPoint((int)(x0+length), (int)(y0+rad));
631        p.addPoint((int)x0, (int)(y0+rad));
632
633        // Rotate the envelop polygon
634        AffineTransform  rotation = AffineTransform.getRotateInstance(angle, x0, y0);
635        Shape shape=rotation.createTransformedShape(p);
636        //Add the envelop rectangle to ROI
637        if(Roi==null) Roi=new ROIShape(shape);
638        else Roi=Roi.add(new ROIShape(shape));
639        // Create and Add 2 circle to ROI
640        Ellipse2D ellipse0 = new Ellipse2D.Double((x0-rad), (y0-rad), 2*rad, 2*rad);
641        Ellipse2D ellipse1 = new Ellipse2D.Double((x1-rad), (y1-rad), 2*rad, 2*rad);
642        Roi=Roi.add(new ROIShape(ellipse0));
643        Roi=Roi.add(new ROIShape(ellipse1));
644      }
645  }
```

```
/**
* A Component with a table for any geo-coordinate points.
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */

package pserc;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;

public class PL_DataTable  extends JPanel{
  /**
   * Table Data.
   */
  public  static DefaultTableModel  model;
  /**
   * Table interface on this JPanel object
   */
  public  JTable  table;
  /**
   * Hard coded of 3 points of image corncer and 2 towers.
   */
  String[][] data={                                        // Hard-coded data for test sample
      /*{"left-top", "333503.46", "1115005.78", "300"}, // Left-top corner of image
      {"right-top", "333503.46", "1114916.62", "300"},  // right-top corner of image
      {"left-bottom", "333434.49", "1115005.78", "300"},//left-bottom corner of image
      {"1", "333448.87", "1114942.79", "300"},          // transmission tower position
      {"2", "333437.91", "1114933.57", "300"},          // transmission tower position*/
      {"p0", "324610.82" , "1171214.27" , "20" , "83", "164"},
      {"p1", "324509.96" , "1171132.42" , "139", "812", "959"},
      {"          " , "  ", "  ", "  ", "  ", },
      {"1", "324548.06", "1171154.81" , "24"},
      {"2", "324532.16", "1171155.11" , "24"},
  };
  String[]   attri={"ID", "Latitude", "Longtitude", "Height", "xpos", "ypos"}; // A list of tabel
attributes

  /**
   * Constructur of PL_DataTable.
   */
  public PL_DataTable () {
     model = new DefaultTableModel (data, attri); // create a sample table model
     table = new JTable(model);                   // create a table interface
     JScrollPane  scrPane = new JScrollPane ();   // create a scrollpanel for table
     scrPane.getViewport ().setView(table);       // set the tabel on the scrollpanel
     scrPane.setPreferredSize (new Dimension(300, 100));  //set the scrollpanel's size
     this.add(scrPane);                           // add scrollpanel on this panel
     setBorder(BorderFactory.createTitledBorder ("Towers")); // set the border with a text
  }
}

```

```
/**
 * A Componet used in loading an image file.
 *
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */

package pserc;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import javax.swing.*;
import javax.media.jai.*;

public class PL_FileLoader implements ActionListener{
  /**
   * Parent Component to generate this file chooser.
   */
  public PL_MainFrame  f;
  /**
   * Constructor of Image File Loader.
   * @param _f PL_MainFrame is the parent component to generate this instance.
   */
  public PL_FileLoader(PL_MainFrame  _f) {f=_f;}
  /**
   * Overrided function to control the event in this image file loader.
   * @param e ActionEvent.
   */
  public void actionPerformed(ActionEvent  e){
    JFileChooser  fileChooser  = new JFileChooser(".");  //Create an instance of JFileChooser
    int ret = fileChooser.showOpenDialog(null);          //Get the list of files.
    File file = fileChooser.getSelectedFile();           //Specify the selected file in the chooser
    if (ret != JFileChooser.APPROVE_OPTION  || file == null)return; //If there is a selected file
    PlanarImage  pi = JAI.create("fileload", file.getAbsolutePath()); //Create an instance of Planer
Image
    f.viewer.setImage(pi);                               //Set the selected image in the main-panel on
the main frame.
    f.validate();
  }
}
```

```
140         g.drawRect(minmax[2*i].x, minmax[2*i].y, minmax[2*i+1].x-minmax[2*i].x, 10);
141         Polygon p=new Polygon();
142         p.addPoint(minmax[2*i].x, minmax[2*i].y+10);
143         p.addPoint(minmax[2*i].x+5, minmax[2*i].y+15);
144         p.addPoint(minmax[2*i].x, minmax[2*i].y+15);
145         g.fillPolygon(p);
146         p=new Polygon();
147         p.addPoint(minmax[2*i+1].x, minmax[2*i+1].y+10);
148         p.addPoint(minmax[2*i+1].x-5, minmax[2*i+1].y+15);
149         p.addPoint(minmax[2*i+1].x, minmax[2*i+1].y+15);
150         g.fillPolygon(p);
151       }
152
153       //Draw NDVI slider
154       g.setColor(Color.white);
155       g.fillRect(xoff, minmax[6].y+30, 255, 10);
156       g.setColor(Color.black);
157       g.drawRect(xoff, minmax[6].y+30, 255, 10);
158       double threshold=(ndviPos.x - xoff - 128)/128.0; // -128~128
159       threshold=((int)(threshold*100.0))/100.0;
160
161       //Draw NDVI threashold
162       g.drawString("NDVI="+threshold, 50 + xoff,minmax[6].y + 50);
163       Polygon pp=new Polygon();
164       pp.addPoint(ndviPos.x, ndviPos.y);
165       pp.addPoint(ndviPos.x+3, ndviPos.y+5);
166       pp.addPoint(ndviPos.x, ndviPos.y+10);
167       pp.addPoint(ndviPos.x-3, ndviPos.y+5);
168       g.fillPolygon(pp);
169
170     }
171
172   }
173
```

```java
/**
* A Componet with Red, Gree, Blue, and NDVI sliders for pixel manipulation.
*
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */

package pserc;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class PL_FilterSlider  extends JPanel {

  /**
   * min and max points for R,G,B,and Inf silders.
   */
  public static Point[] minmax;
  /**
   * min and max points for NDVI slider.
   */
  public  Point   ndviPos;
  /**
   * Global threshold of NDVI.
   */
  public static double threshold=0.2;

  /**
   * Label for sliders
   */
  String[] names={"B: min", "max", "G: min", "max","R: min", "max","Inf: min", "max"};

  /**
   * offset of the first slider position in this panel.
   */
  int xoff=10, yoff=20;

  /**
   *  Parent component to generate this instance
   */
  PL_MainFrame  f;
  /**
   * Flag used for specifying a slider mouse-picked.
   */
  private int selectID=-1;

  /**
   * Original Constructor.
   * @param _f PL_MainFrame pointer to parent component.
   */
  public PL_FilterSlider(PL_MainFrame  _f) {
    f=_f;
    minmax=new Point[8];                          //create 4 min and 4 max points
    for(int i=0; i<4; i++){
      minmax[2*i]=new Point(xoff+0, yoff+i*30);     //min
      minmax[2*i+1]=new Point(xoff+255, yoff+i*30); //max
    }
    ndviPos=new Point(xoff+128, yoff+120);        //create min and max points for NDVI
    setPreferredSize(new Dimension(300, 200));

    addAction();
    setBorder(BorderFactory.createTitledBorder("Filter")); // Show the title
  }
  /**
   * Add mouse motion and mouse actions.
   */
```

```
71    private void addAction(){
72      this.addMouseListener(new MouseAdapter(){
73        /**
74         * Identify the selected slider in mouse pressed.
75         * @param e MouseEvent.
76         */
77        public void mousePressed(MouseEvent e){
78          selectID=-1;
79          int mx=e.getX(); int my=e.getY();
80          for(int i=0; i<8 ;i++){ // Check if one of the R,G,B,IF slider is selected
81            if(Math.abs(mx-minmax[i].x) <10 && Math.abs(my-minmax[i].y)<10){
82              selectID=i;
83              return;
84            }
85          }
86          if(Math.abs(mx-ndviPos.x) <10 && Math.abs(my-ndviPos.y)<10) //Chech if NDVI slider is
    selected
87              selectID=9;
88        }
89        /**
90         * Set the selectedID as -1 in mouse released.
91         * @param e MouseEvent.
92         */
93        public void mouseReleased(MouseEvent e){
94          selectID=-1;
95        }
96      });
97
98      this.addMouseMotionListener(new MouseMotionAdapter(){
99        /**
100        * Change the min/max positions in slider in mouse dragging.
101        * @param e MouseEvent.
102        */
103       public void mouseDragged(MouseEvent e){
104         if(selectID == -1) return;
105         if(selectID == 9){ // Condition of the case where NDVI is selcted.
106           ndviPos.x=e.getX();
107           if(ndviPos.x-xoff<0) ndviPos.x=xoff;  //update the position
108           if(ndviPos.x-xoff>255) ndviPos.x=255+xoff;
109           threshold=(ndviPos.x-xoff-128)/128.0;
110           repaint();
111           f.viewer.repaint();
112           return;
113         }
114         // Else if one of R, G, B, IF sliders is slected.
115         minmax[selectID].x =e.getX();
116         if(minmax[selectID].x-xoff<0) minmax[selectID].x=xoff;
117         if(minmax[selectID].x-xoff>255) minmax[selectID].x=255+xoff;
118         repaint();
119
120       }
121     });
122   }
123   /**
124    * Overrided paint function to darw R, G, B, IF, NDVI sliders.
125    * @param g Graphics.
126    */
127   public void paintComponent(Graphics g){
128     super.paintComponent(g);
129     //Draw R, G, B, IF sliders
130     for(int i=0; i<4; i++){
131       g.setColor(Color.white);
132       g.fillRect(xoff, minmax[2*i].y, 255, 10);
133       g.setColor(Color.black);
134       g.drawRect(xoff, minmax[2*i].y, 255, 10);
135       g.drawString(names[2*i]+"="+(minmax[2*i].x-xoff), 50+xoff,  minmax[2*i].y+20);
136       g.drawString(names[2*i+1]+"="+(minmax[2*i+1].x-xoff), 50+xoff+60,  minmax[2*i].y+20);
137       g.setColor(Color.lightGray);
138       g.fillRect(minmax[2*i].x, minmax[2*i].y, minmax[2*i+1].x-minmax[2*i].x, 10);
139       g.setColor(Color.black);
```

```java
/**
 * A Object to store the geographical information of image.
 *
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */

package pserc;
import java.awt.*;

public class PL_Geo {

  /**
   * Latitude of Left-top corner of image.
   */
  public static String latitude;
  /**
   * Longitude of Left-top corner of images.
   */
  public static String longitude;
  /**
   * East-West length (meter) per pixel.
   */
  public static double EW_scale=1.0;
  /**
   * default North-South length (meter) per pixel.
   */
  public static double NS_scale=1.0;  //Scale factor of North-South Direciton
  public static int refX=0;           //X coordinate of a reference point
  public static int refY=0;           //Y coordinate of a reference point
  public static double refLat=0.0;    // Latitude of a reference point
  public static double refLon=0.0;    // Longitude of a reference point
  public static double a=1.0; // a, b, c, d are matrix elements
  public static double b=1.0;
  public static double c=1.0;
  public static double d=1.0;
  /**
   * Constructor of Liner Transformation used by PL_Output (Stage 3)
   * The algorithm is expalin at the botoom of this function
   *
   * @param lat0 double, latitude of the first point
   * @param lon0 double, longitude of the first point
   * @param _x0 int, x coordinate of the first point
   * @param _y0 int, y coordinate of the first point
   * @param lat1 double, latitude of the second point
   * @param lon1 double, longitude of the second point
   * @param _x1 int, x coordinate of the second point
   * @param _y1 int, y coordinate of the second point
   * @param lat2 double, latitude of the third point
   * @param lon2 double, longitude of the third point
   * @param _x2 int, x coordinate of the thrid point
   * @param _y2 int, y coordinate of the thrid point
   */
  public PL_Geo(double lat0, double lon0, int _x0, int _y0,
                double lat1, double lon1, int _x1, int _y1,
                double lat2, double lon2, int _x2, int _y2){
    refLat=lat0;
    refLon=lon0;
    refX=_x0;
    refY=_y0;
    double x0=_x1-_x0;
    double y0=_y1-_y0;
    double x1=_x2-_x0;
    double y1=_y2-_y0;
    double delta=x0*y1-x1*y0;
```

```
71        double _lat0=lat1-lat0;
72        double _lon0=lon1-lon0;
73        double _lat1=lat2-lat0;
74        double _lon1=lon2-lon0;
75        System.out.println("delta="+delta);
76        if(delta ==0.0) return;
77        a = 1.0/delta*(_lat0*y1 - _lat1*y0);
78        b = 1.0/delta*(_lat1*x0 - _lat0*x1);
79        c = 1.0/delta*(_lon0*y1 - _lon1*y0);
80        d = 1.0/delta*(_lon1*x0 - _lon0*x1);
81      }
82      // ********** Inverse Matrix
83
84      //  |a  b| |x0 x1|  |Lat0 Lat1|
85      //  |c  d|*|y0 y1| =|Lon0 Lon1|
86
87      //  |a  b| |Lat0 Lat1|                  |y1  -x1|
88      //  |c  d|=|Lon0 Lon1| * 1/(x0*y1-x1*y0) |-y0  x0|
89
90      //  |a  b|                |Lat0 Lat1|  |y1  -x1|
91      //  |c  d|= 1/(x0*y1-x1*y0)|Lon0 Lon1|  |-y0  x0|
92
93      //  |a  b|                |Lat0*y1 - Lat1*y0 Lat1*x0-Lat0*x1 |
94      //  |c  d|= 1/(x0*y1-x1*y0)|Lon0*y1 - Lon1*y0 Lon1*x0-Lon0*x1 |
95
96      /**
97       * To get the XY pixel position from latitude and longitude
98       * @param lat double, latiutde of a point
99       * @param lon double, longitude of a point
100      * @return Point, return XY coordinate
101      */
102     public static Point getPixelPos(double lat, double lon){
103       double _lat=lat-refLat;
104       double _lon=lon-refLon;
105       double delta=a*d-b*c; // delta for calculating the inverse matrix
106       if(delta==0) return new Point(0,0);
107       double newX=1.0/delta*(d*_lat-b*_lon);
108       double newY=1.0/delta*(-c*_lat + a*_lon);
109       return new Point((int)newX+refX, (int)newY+refY);
110     }
111
112     /**
113      * To get the Latitude and Longitude from XY pixel coordinate
114      *
115      * @param x int, X position in an image
116      * @param y int, Y position in an image
117      * @return double[], [0]=latitude and [1] = longitude
118      */
119     public static double[] getLAT(int x, int y){
120       double _x=x-refX;
121       double _y=y-refY;
122       double[] geoPos=new double[2];
123       geoPos[0]=a*_x+b*_y +refLat;
124       geoPos[1]=c*_x+d*_y +refLon;
125       return geoPos;
126     }
127
128     /**
129      * Constructor using in Stage 1
130      *
131      * @param _lat0 String
132      * @param _lon0 String
133      * @param x0 int
134      * @param y0 int
135      * @param _lat1 String
136      * @param _lon1 String
137      * @param x1 int
138      * @param y1 int
139      */
140     public PL_Geo(String _lat0, String _lon0, int x0, int y0,
```

```
141                   String _lat1, String _lon1, int x1, int y1){
142       double[] disXY=getXY(_lat0, _lon0, _lat1, _lon1);
143       latitude=_lat0;
144       longitude=_lon0;
145       EW_scale=disXY[0]/( x1-x0 );
146       NS_scale=disXY[1]/( y1-y0 );
147       refX=x0;
148       refY=y0;
149     }
150
151     /**
152      *
153      * Origianl Constructor using the information of three point,
154      * 1) left-top, 2) right-top, 3) left-bottom, and image pixel size;
155      * imageX and imageY.
156      *
157      * @param left_top_lat String latitude of left top of image.
158      * @param left_top_lon String longitude of left top of image.
159      * @param right_top_lat String latitude of right top of image.
160      * @param right_top_lon String longitude of right top of image.
161      * @param left_bottom_lat String latitude of left-bottom of image.
162      * @param left_bottom_lon String longitude of left-bottom of image.
163      * @param imageX int pixels of image width.
164      * @param imageY int pixels of image height.
165      */
166     public PL_Geo(String left_top_lat, String left_top_lon,
167                   String right_top_lat, String right_top_lon,
168                   String left_bottom_lat, String left_bottom_lon,
169                   int imageX, int imageY) {
170       // Step1: get XY cordinate in UTM projection
171       latitude=left_top_lat;
172       longitude=left_top_lon;
173       double[] east_west=getXY(left_top_lat,left_top_lon, right_top_lat,right_top_lon);
174       double[] south_north=getXY(left_top_lat,left_top_lon,left_bottom_lat,left_bottom_lon);
175
176       // Step2: get East_West length of image
177       EW_scale=east_west[0]/imageX;
178
179       // Step3: get North_South length of image
180       NS_scale=south_north[1]/imageY;
181     }
182     /**
183      * Default Constructor using the inforamtion of PL_DataTable.
184      */
185     public PL_Geo(){
186       // this should be called after table and image are set.
187       String _lat0=(String)PL_DataTable.model.getValueAt(0, 1); //left-top latitude
188       String _lon0=(String)PL_DataTable.model.getValueAt(0, 2); //left-top longitude
189       String _lat1=(String)PL_DataTable.model.getValueAt(1, 1); //right-top latitude
190       String _lon1=(String)PL_DataTable.model.getValueAt(1, 2); //right-top longitude
191       String _posX0=(String)PL_DataTable.model.getValueAt(0, 4); //left-bottom latitude
192       String _posY0=(String)PL_DataTable.model.getValueAt(0, 5); //left-bottom longitude
193       String _posX1=(String)PL_DataTable.model.getValueAt(1, 4); //left-bottom latitude
194       String _posY1=(String)PL_DataTable.model.getValueAt(1, 5); //left-bottom longitude
195      // Step1: get XY cordinate in UTM projection
196      latitude=_lat0;
197      longitude=_lon0;
198      double[] dis=getXY(_lat0,_lon0, _lat1,_lon1);
199      int posX=Integer.parseInt(_posX0);
200      int posY=Integer.parseInt(_posY0);
201      int posX1=Integer.parseInt(_posX1);
202      int posY1=Integer.parseInt(_posY1);
203
204      // Step2: get East_West length of image
205      EW_scale=dis[0]/(double)(posX1-posX);
206
207      // Step3: get North_South length of image
208      NS_scale=dis[1]/(double)(posY1-posY);
209      refX=posX;
210      refY=posY;
```

```
211
212     }
213     /**
214      * Get the XY pixel position from latitude and longitude.
215      * @param _lat String latitude should be represented as DDMMSS.SSS
216      * @param _lon String longitude should be represented as DDMMSS.SSS.
217      * @return Point Pixel Position in image.
218      */
219     public static Point pixelPos(String _lat, String _lon){
220       double[] xy=getXY(latitude, longitude, _lat, _lon); //get UTM position
221       int xpos = (int)(xy[0]/EW_scale) +refX;     //convert UTM-xy to X image position
222       int ypos = (int)(xy[1]/NS_scale) +refY;     //convert UTM-xy to Y image position
223       return new Point(xpos, ypos);
224     }
225     /**
226      * Get the degree value from degree as DDMMSS.SSS.
227      * @param _d_m_s String input degree.
228      * @return double degree value. (without minutes or second)
229      */
230       public static double getDegreeValue(String _d_m_s){
231        double temp=Double.parseDouble(_d_m_s);
232        int _d= (int)(temp/10000.0);
233        temp=temp- ((double)_d*10000.0);
234        int _m= (int) (temp/100.0);
235        double _s=temp- ((double)_m*100.0);
236        double degree=( (double)_d + (double)_m/60.0 + _s/3600.0 );
237        System.out.println("Degree="+degree);
238        return  degree;
239      }
240     /**
241      * Get the radian value from degree as DDMMSS.SSS.
242      * @param _d_m_s String input degree.
243      * @return double radian value.
244      */
245     public static double getRadian(String _d_m_s){
246      double temp=Double.parseDouble(_d_m_s);
247      int _d= (int)(temp/10000.0);
248      temp=temp- ((double)_d*10000.0);
249      int _m= (int) (temp/100.0);
250      double _s=temp- ((double)_m*100.0);
251      double degree=( (double)_d + (double)_m/60.0 + _s/3600.0 );
252      System.out.println("Degree="+degree);
253      return  degree*Math.PI/180.0;
254    }
255
256    /**
257     * Get the Degree, Minutes, and Secound as a String
258     * @param lat double
259     * @return double
260     */
261     public static void getDegree(double d){
262       int degree= (int)d;
263       int minutes= (int)((d-degree)*60);
264       double second = (d-degree-(double)minutes/60.0)*3600.0d;
265
266       double newD=degree+(double)minutes/60.0+second/3600.0;
267       System.out.println("Degree="+degree+":"+minutes+":"+second+", orig="+d+":conveted"+newD);
268     }
269
270    /**
271     * Get the meridian length by meter from latitude.
272     * @param lat double Latitude as radian value.
273     * @return double Meridian Length (meter).
274     */
275    public static double getMeridian(double lat) {
276      double a = 6378137.0;                    //semimajor axis of Earth from GRS80
277      double f = 1.0 / 298.257222101 ;         //flattening rate of Earth
278      double _F = 298.257222101 ;
279      double e = (Math.sqrt(2.0 * _F - 1.0)) / _F;  // first eccentricity
280      double e_ = (Math.sqrt(2.0 * _F - 1.0)) / (_F - 1.0); //second eccentricity
```

```
281
282     //Step 1: Calcuate the A, B, C, D, E, F, G, H, I,
283     //They are the constants multiplied with e for equidistant latitude function
284
285     double A = 1.0 + (3.0 / 4.0) * Math.pow(e, 2) +
286         (45.0 / 65.0) * Math.pow(e, 4) + (175.0 / 256.0) * Math.pow(e, 6) +
287         (11025.0 / 16384.0) * Math.pow(e, 8) +
288         (43659.0 / 65536.0) * Math.pow(e, 10) +
289         (693693.0 / 1048576.0) * Math.pow(e, 12) +
290         (19324305.0 / 29360128.0) * Math.pow(e, 14) +
291         (4927697775.0 / 7516192768.0) * Math.pow(e, 16);
292     double B = (3.0 / 4.0) * Math.pow(e, 2) + (15.0 / 16.0) * Math.pow(e, 4) +
293         (525.0 / 512.0) * Math.pow(e, 6) +
294         (2205.0 / 2048.0) * Math.pow(e, 8) +
295         (72765.0 / 65536.0) * Math.pow(e, 10) +
296         (297297.0 / 262144.0) * Math.pow(e, 12) +
297         (135270135.0 / 117440512.0) * Math.pow(e, 14) +
298         (547521975.0 / 469762048.0) * Math.pow(e, 16);
299     double C = (15.0 / 64.0) * Math.pow(e, 4) + (105.0 / 256.0) * Math.pow(e, 6) +
300         (2205.0 / 4096.0) * Math.pow(e, 8) +
301         (10395.0 / 16384.0) * Math.pow(e, 10) +
302         (1486485.0 / 2097152.0) * Math.pow(e, 12) +
303         (45090045.0 / 58720256.0) * Math.pow(e, 14) +
304         (766530765.0 / 939524096.0) * Math.pow(e, 16);
305     double D = (35.0 / 512.0) * Math.pow(e, 6) +
306         (315.0 / 2048.0) * Math.pow(e, 8) +
307         (31185.0 / 131072.0) * Math.pow(e, 10) +
308         (165165.0 / 524288.0) * Math.pow(e, 12) +
309         (45090045.0 / 117440512.0) * Math.pow(e, 14) +
310         (209053845.0 / 469762048.0) * Math.pow(e, 16);
311     double E = (315.0 / 16384.0) * Math.pow(e, 8) +
312         (3465.0 / 65536.0) * Math.pow(e, 10) +
313         (99099.0 / 1048576) * Math.pow(e, 12) +
314         (4099095.0 / 29360128.0) * Math.pow(e, 14) +
315         (348423075.0 / 1879048192.0) * Math.pow(e, 16);
316     double F = (693.0 / 131072.0) * Math.pow(e, 10) +
317         (9009.0 / 524288.0) * Math.pow(e, 12) +
318         (4099095.0 / 11744052.0) * Math.pow(e, 14) +
319         (26801775.0 / 469762048.0) * Math.pow(e, 16);
320     double G = (3003.0 / 2097152.0) * Math.pow(e, 12) +
321         (315315.0 / 58720256.0) * Math.pow(e, 14) +
322         (11486475.0 / 939524096.0) * Math.pow(e, 16);
323     double H = (45045.0 / 117440512.0) * Math.pow(e, 14) +
324         (765765.0 / 469762048.0) * Math.pow(e, 16);
325     double I = (765765.0 / 7516192768.0) * Math.pow(e, 16);
326
327     //Get the valued of all stages of the partial differential coefficients with
328     //respect to e.
329     double b = a * (1.0 - e * e);
330     double B1 = b * A;
331     double B2 = b * ( -B / 2.0);
332     double B3 = b * (C / 4.0);
333     double B4 = b * ( -D / 6.0);
334     double B5 = b * (E / 8.0);
335     double B6 = b * ( -F / 10.0);
336     double B7 = b * (G / 12.0);
337     double B8 = b * ( -H / 14.0);
338     double B9 = b * (I / 16.0);
339
340     double d = lat; // latitude
341
342     //equidistant latitude function
343     double m = B1 * d + B2 * Math.sin(2.0 * d) + B3 * Math.sin(4.0 * d) +
344         B4 * Math.sin(6.0 * d) + B5 * Math.sin(8.0 * d) +
345         B6 * Math.sin(10.0 * d) + B7 * Math.sin(12.0 * d) +
346         B8 * Math.sin(14.0 * d) + B9 * Math.sin(16.0 * d);
347
348     return m;
349    }
```

```
351     /**
352      * Calculate the XY distnace from the original point to the target point.
353      *
354      * @param latS String Original point's latitude as DDMMSS.SS.
355      * @param lonS String origianl point's longitude as DDMMSS.SS.
356      * @param _latS String Target point's latitude as DDMMSS.SS.
357      * @param _lonS String Target point's latitude as DDMMSS.SS.
358      * @return double[] (double[0], double[1]) represents (X distance, Y distance) in UTM.
359      */
360     public static double[] getXY(String latS, String lonS, String _latS, String _lonS){
361       double lat=getRadian(latS);       //convert latitide as Radian
362       double lon=getRadian(lonS);       //convert longitude as Radian
363       double _lat=getRadian(_latS);     //convert target latitude as Radian
364       double _lon=getRadian(_lonS);     //convert target longitude as Radian
365       double a=6378137.0;              //semimajor axis of Earth from GRS80
366       double f= 1.0/298.257222101;     //flattening rate of Earth
367       double _F=298.257222101;
368       double e=(Math.sqrt(2.0*_F -1.0)) / _F;          // first eccentricity
369       double e_=(Math.sqrt(2.0*_F - 1.0)) / (_F-1.0); // second eccentricity
370
371       double t=Math.tan(lat);
372       double delta_lon=lon-_lon;        // offset degree in longitude
373       double mu2=e_*e_*Math.cos(lat)*Math.cos(lat);
374       double N=a/(Math.sqrt(1.0-e*e*Math.sin(lat)*Math.sin(lat)));
375       double m0=0.9999;                 //UTM factor from Gauss-Kruger projection
376       double cos=Math.cos(lat);
377       double[] xy=new double[2];
378
379       //Get the distance in Y direction using Gauss-Kruger projection
380       xy[1] = - ( (getMeridian(lat) - getMeridian(_lat))
381                  +
382                  1.0 / 2.0 * N * (Math.pow(cos, 2)) * t * (delta_lon * delta_lon)
383                  +
384                  1.0 / 24.0 * N * (Math.pow(cos, 4)) * t *
385                  (5.0 - t * t + 9 * mu2 + 4.0 * mu2 * mu2) *
386                  Math.pow(delta_lon, 4)
387                  -
388                  1.0 / 720.0 * N * (Math.pow(cos, 6)) * t *
389                  ( -61.0 + 58.0 * t * t - t * t * t * t - 270.0 * mu2 +
390                   330.0 * t * t * mu2) * Math.pow(delta_lon, 6)
391                  -
392                  1.0 / 40320.0 * N * (Math.pow(cos, 8)) * t *
393                  ( -1385.0 + 311.0 * t * t - 543.0 * t * t * t * t * t +
394                   Math.pow(t, 6)) * Math.pow(delta_lon, 8)) * m0;
395
396      //Get the distance in X direction using Gauss-Kruger projection
397      xy[0] = (N * cos * delta_lon
398                -
399                1.0 / 6.0 * N * (Math.pow(cos, 3)) * ( -1.0 + t * t - mu2) *
400                Math.pow(delta_lon, 3)
401                -
402                1.0 / 120.0 * N * (Math.pow(cos, 5)) *
403                ( -5.0 + 18.0 * t * t - t * t * t * t * t - 14.0 * mu2 +
404                 58.0 * t * t * t * mu2) * Math.pow(delta_lon, 5)
405                -
406                1.0 / 5040.0 * N * (Math.pow(cos, 7)) *
407                ( -61.0 + 479.0 * t * t - 179.0 * Math.pow(t, 4) + Math.pow(t, 6)) *
408                Math.pow(delta_lon, 7)) * m0;
409
410       return xy; // XY distnace by meter
411     }
412   }
```

```
/**
 * A Base Component extended by PL_RenderedImageCanvas object.
 * PL_ScrollController is implemented as mouse event interface.
 *
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */

package pserc;
import javax.swing.*;
import java.awt.*;
import javax.media.jai.*;
import java.awt.geom.*;
public class PL_ImageCanvas  extends JComponent  implements  PL_ScrollController {
  /**
   * Maximum image width.
   */
  public final static int MAX_WIDTH=2048;
  /**
   * Maximum image height.
   */
  public final static int MAX_HEIGHT=2048;
  /**
   *  origianl image.
   */
  transient protected PlanarImage  image;
  /**
   * Metrix for transformation.
   */
  protected AffineTransform  atx=new AffineTransform();
  /**
   * width and height of image.
   */
  protected int width, height;
  /**
   * condition if the image is drawn or not.
   */
  protected boolean  imageDrawn=false;
  /**
   * scroll amount in X and Y.
   */
  protected int panX=0, panY=0;
  /**
   * scroll Anchor point used in mouse pressed.
   */
  protected Point scrollAnchor  = new Point(0,0);
  /**
   * condition if the scroll is active or not.
   */
  protected boolean  scrollOn=true;
  /**
   * view port position in scroll panel.
   */
  protected Point vpPos=new Point(0,0);
  /**
   *  offset of mouse position in pan.
   */
  protected Point panOffset=new Point(0,0);

  /**
   * Default Constructor.
   */
  public PL_ImageCanvas(){}

  /**
   * Origianl Constructor
```

```
71      * @param img PlanarImage a image file.
72      */
73     public PL_ImageCanvas (PlanarImage  img){setImage(img);}
74
75     /**
76      * Set the image on the panel.
77      * @param img PlanarImage a image file to be set.
78      */
79     public void setImage(PlanarImage  img){
80       reset();
81       image=img;
82       int wid=img.getWidth(); int ht=img.getHeight();
83       width = (wid > MAX_WIDTH) ? MAX_WIDTH : wid;
84       height= (ht >MAX_HEIGHT) ? MAX_HEIGHT : ht;
85       setPreferredSize(new Dimension(width, height));
86       imageDrawn = false;
87       repaint();
88     }
89
90     /**
91      * Get Pan offset in X and Y.
92      * @return Point Offset Amount in X and Y.
93      */
94     public Point getPanOffset(){return panOffset;}
95
96     /**
97      * Set offset values for pan action.
98      * @param panOffset Point offset point in panning.
99      */
100    public void setPanOffset(Point panOffset){
101      firePropertyChange("PanOffset", this.panOffset, panOffset);
102      this.panOffset=panOffset;
103      panX=panOffset.x;
104      panY=panOffset.y;
105    }
106
107    /**
108     * Set the condition for scrolling event.
109     * @param onOff boolean If this is true, the scroll events are active. Otherwise, not.
110     */
111    public void setScrollOn(boolean onOff){scrollOn=onOff; panX=0; panY=0; vpPos=new Point(0,0);}
112
113    /**
114     * Get the condition of scroll-on or off.
115     * @return boolean IF scroll is active, return TURE. Otherwise, FALSE.
116     */
117    public boolean getScrollOn(){return scrollOn;}
118
119    /**
120     * Called when the scoll is stareted.
121     * @param x int Mouse X position.
122     * @param y int Mouse Y position.
123     */
124    public void startScroll(int x, int y){
125      scrollAnchor.x=x-panX;
126      scrollAnchor.y=y-panY;
127      setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));
128      repaint();
129    }
130
131    /**
132     * Called when the scoll is done.
133     * @param x int Current Mouse X position.
134     * @param y int Current Mouse Y Position.
135     */
136    public void scroll(int x, int y){
137      if(x<0||y<0) return;
138      int panX=x-scrollAnchor.x;  //get Panning value in X
139      int panY=y-scrollAnchor.y;  //get Panning value in Y
140      setPanOffset(new Point(panX, panY));   //Set Offset
```

```
141       setViewportPosition (new Point(-panX, -panY)); //Set View Port
142       repaint();
143     }
144
145     /**
146      * Stop the scroll.
147      */
148     public void stopScroll(){setCursor(Cursor.getDefaultCursor());}
149
150     /**
151      * Pan the screen.
152      * @param x double pan value in X direction.
153      * @param y double pan value in Y direction.
154      */
155     public void pan(double x, double y){this.setPanOffset(new Point((int)x, (int)y));repaint();}
156
157     /**
158      * Get the view port postion in scroll panel.
159      * @return Point.
160      */
161     public Point getViewportPosition(){return vpPos;}
162
163     /**
164      * Set the view port position in scroll panel.
165      * @param vpPos Point.
166      */
167     public void setViewportPosition(Point vpPos){
168       firePropertyChange("viewportPosition", this.vpPos, vpPos);
169       this.vpPos=vpPos;
170     }
171
172     /**
173      * Set Transformation Materix.
174      * @param at AffineTransform.
175      */
176     public void setTransform(AffineTransform  at){atx=at;}
177
178     /**
179      * Get Transformation Materix.
180      * @return AffineTransform.
181      */
182     public AffineTransform  getTransform(){return atx;}
183
184     /**
185      * Reset the AffineTransform and Pan value as (0, 0).
186      */
187     public void reset(){atx=new AffineTransform(); panX=0; panY=0;}
188
189     /**
190      * Draw the image in the scrolled position on Scroll panel.
191      * @param gc Graphics.
192      */
193     public void paintComponent(Graphics  gc){
194       Graphics2D  g=(Graphics2D)gc;
195       Rectangle  rect=this.getBounds();
196       if(width!=rect.width || height != rect.height){
197         double magx=rect.width/(double)width;
198         double magy=rect.height/(double)height;
199         atx.setToScale(magx, magy);
200       }
201       g.setColor(Color.black);
202       g.fillRect(0,0, rect.width, rect.height);
203       atx=AffineTransform.getTranslateInstance(panX, panY);
204       if(image !=null) g.drawRenderedImage(image, atx);
205       imageDrawn=true;
206     }
207   }
208
```

```
/**
* Main Frame Component for this program.
*
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */
package pserc;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.beans.*;

public class PL_MainFrame  extends JFrame{

  /**
   * Main Function to be called in running this program.
   * @param arg String[].
   */
  public static void main (String[] arg){
    PL_MainFrame  f=new PL_MainFrame ();
 }
 /**
  * Container of this frame.
  */
 public Container p;
 /**
  * Top menu bar.
  */
 public MenuBar menuBar;
 /**
  * Main Image panel.
  */
 public PL_RenderedImageCanvas  viewer;
 /**
  * Grid Panel.
  */
 public PL_RenderGrid  renderGrid;
 /**
  *  Data table Panel for Georeferenced points.
  */
 public PL_DataTable  table;
 /**
  * Slider Panel for control NDVI.
  */
 public PL_FilterSlider  slide;
 /**
  * Scroll action for Image Panel
  */
 public PL_ScrollGUI  scroll;
 /**
  * Text filed to show the width and height
  */
 public JTextField  twidth, theight;

 /**
  * Default Constructor to initialize panels and add events.
  */
 public PL_MainFrame () {
    // Frame setting
    super("Vegetation Extraction Viewer" );          // Set the title of this program
    addMenuBar2 ();                          // add menu bar
    Dimension  dim=Toolkit.getDefaultToolkit ().getScreenSize ();
    int width=(int)(dim.width*3/4.0);
    int height=(int)(dim.height*3/4.0);
    p = getContentPane ();
```

```
71      p.setLayout(new BorderLayout());       // set the layout manager
72
73      //Create Components put on this frame
74      viewer=new PL_RenderedImageCanvas();            // Initialize the left main panel
75      viewer.setPreferredSize(new Dimension (512,512));
76      renderGrid=new PL_RenderGrid();                 // initialize the Grid panel
77      renderGrid.setBackground(Color.gray);
78      renderGrid.setPreferredSize(new Dimension (300,200));
79      renderGrid.setBorder(BorderFactory.createTitledBorder("Tile Grid"));
80      twidth = new JTextField(5);
81      theight = new JTextField(5);
82      theight.setText(Integer.toString(viewer.getTileHeight()));
83      twidth.setText(Integer.toString(viewer.getTileWidth()));
84      JPanel tp = new JPanel();                       // Left column panel
85      tp.setLayout(new BorderLayout());
86      table = new PL_DataTable();                     // Initialize Data Table Panel
87      slide = new PL_FilterSlider(this);              // Initialize Filter Slider Panel
88
89      // Add components
90      tp.add(renderGrid, BorderLayout.CENTER);
91      tp.add(table, BorderLayout.NORTH);
92      tp.add(slide, BorderLayout.SOUTH);
93      p.add(tp, BorderLayout.WEST);
94      p.add(viewer, BorderLayout.CENTER);
95
96      // Add Event contorl
97      addEventAdapters();                             // Add Event Adapter
98      scroll=new PL_ScrollGUI(viewer);                // Initialize Scroll GUI
99      viewer.addMouseListener(scroll);                // Add mouse events
100     viewer.addMouseMotionListener(scroll);
101     viewer.repaint();
102     renderGrid.setTileDimension(viewer.getTileWidth(), viewer.getTileHeight());
103     updateRenderGrid();
104
105     this.setSize(900, 700);                       // Default size of frame
106     this.setVisible(true);
107     this.validate();
108     this.addWindowListener(new WindowAdapter() {  //Set the window close event
109       public void windowClosing(WindowEvent e) {
110         System.exit(0);
111       }
112     });
113   }
114
115   /**
116    * Add Menu items in top menu bar.
117    */
118   protected void addMenuBar2(){
119     JMenuBar menuBar=new JMenuBar();
120     this.setJMenuBar(menuBar);
121
122     //Initialize the menu items
123     JMenu file=new JMenu("File");     // Menu for File item
124     JMenu edit=new JMenu("Edit");     // Menu for Edit item
125     JMenu tool=new JMenu("Tool");     // Menu for Tool item
126     JMenu help=new JMenu("Help");     // Menu for Help item
127     JMenu open=new JMenu("Open");     // Meue for Open item
128     JMenuItem openImage=new JMenuItem("Open Image"); //item for opening image files under Open item
129     JMenuItem openGIS=new JMenuItem("Open GIS");     //item for opening GIS files uder Open item
130     JMenuItem save=new JMenuItem("Save");     //item for saving files under File item
131     JMenuItem test=new JMenuItem("Test");     //item for testing
132
133     //Add menu items hierarchy
134     open.add(openImage);
135     open.add(openGIS);
136     file.add(open);
137     file.add(save);
138     tool.add(test);
139     menuBar.add(file);
140     menuBar.add(edit);
```

```
141       menuBar.add(tool);
142       menuBar.add(help);
143
144       //add image file loader event to the Open-Image item.
145       PL_FileLoader loader=new PL_FileLoader(this);
146       openImage.addActionListener(loader);
147     }
148
149     /**
150      * Update the PL_RenderGrid Panel.
151      */
152     protected void updateRenderGrid(){
153         if((viewer == null)|| (renderGrid == null)) return;
154         // Set the tile information
155         int twl = viewer.getTileWidth();
156         int thl = viewer.getTileHeight();
157         int xIndl = viewer.getMaxTileIndexX();
158         int yIndl = viewer.getMaxTileIndexY();
159
160         //Set the grid information
161         renderGrid.setTileDimension(twl,thl);
162         renderGrid.setTileIndices(xIndl, yIndl);
163         int wid = renderGrid.getViewportDimension().width;
164         int ht = renderGrid.getViewportDimension().height;
165         double scaleX = viewer.getSize().width/(double)viewer.getImageWidth();
166         double scaleY = viewer.getSize().height/(double)viewer.getImageHeight();
167         double scaledVpX = wid*scaleX;
168         double scaledVpY = ht*scaleY;
169
170         //Set the viewport size
171         renderGrid.setViewportSize(new Dimension((int)scaledVpX,(int)scaledVpY));
172         renderGrid.repaint();
173     }
174
175     /**
176      * Add Events for Tiles and Properties.
177      */
178     protected void addEventAdapters() {
179         addTileParamsEventAdapters();
180         addPropertyChangeEventAdapters();
181      }
182
183      /**
184       * Override the event function for setting tile paramters.
185       */
186      protected void addTileParamsEventAdapters(){
187        // for getting the width information of tiles
188        twidth.addActionListener(
189            new ActionListener(){
190               public void actionPerformed(ActionEvent e) {
191                  try {
192                     String str = ((JTextField) e.getSource()).getText();
193                     int wid = Integer.parseInt(str);
194                     viewer.setTileWidth(wid);
195                     str = theight.getText();
196                     int ht = Integer.parseInt(str);
197                     viewer.setTileHeight(ht);
198                     if(renderGrid != null) updateRenderGrid();
199                  } catch (Exception e2){}
200               }
201             } );
202         // for getting the height inforamtion of tiles
203         theight.addActionListener(
204            new ActionListener(){
205               public void actionPerformed(ActionEvent e) {
206                  try {
207                     String str = ((JTextField) e.getSource()).getText();
208                     int ht = Integer.parseInt(str);
209                     viewer.setTileHeight(ht);
210                     str = twidth.getText();
```

```
                    int wid = Integer.parseInt(str);
                    viewer.setTileWidth(wid);
                    if(renderGrid != null) updateRenderGrid();
                  } catch (Exception e1){}
                }
            });
    }
    /**
     * Add event functions to change the properties for Tiles in
     * PL_RenderedImageCanvas and PL_RenderGrid.
     */
    protected void addPropertyChangeEventAdapters (){
      /**
       * Change the properties of PL_RenderedImageCanvas.
       */
      viewer.addPropertyChangeListener (
           new PropertyChangeListener () {
              public void propertyChange(PropertyChangeEvent e) {
                 if(e.getPropertyName().equals("viewportPosition")){
                    twidth.setText(Integer.toString(viewer.getTileWidth()));
                    theight.setText(Integer.toString(viewer.getTileHeight()));
                    if(renderGrid != null) updateRenderGrid();
                    Point p =(Point)e.getNewValue();
                    renderGrid.setViewportPosition(p);
                 }
                 if(e.getPropertyName().equals("maxTileIndexX")){
                    renderGrid.setMaxTileIndexX(((Integer)e.getNewValue()).intValue());
                 }
                 if(e.getPropertyName().equals("maxTileIndexY")){
                    renderGrid.setMaxTileIndexY(((Integer)e.getNewValue()).intValue());
                 }
                 if(e.getPropertyName().equals("tileWidth")){
                    renderGrid.setTileWidth(((Integer)e.getNewValue()).intValue());
                 }
                if(e.getPropertyName().equals("tileHeight")){
                    renderGrid.setTileHeight(((Integer)e.getNewValue()).intValue());
                 }
                 if(e.getPropertyName().equals("transform")){
                    int wid = renderGrid.getViewportDimension().width;
                    int ht = renderGrid.getViewportDimension().height;
                    double scaleX = wid/(double)viewer.getSize().width;
                    double scaleY = ht/(double)viewer.getSize().height;
                    AffineTransform atx =  (AffineTransform)e.getNewValue();
                    renderGrid.setTransform(atx);
                 }
              }
           }
      );
      /**
       * Change the properties of PL_RenderGrid.
       */
      renderGrid.addPropertyChangeListener (
           new PropertyChangeListener () {
              public void propertyChange(PropertyChangeEvent e) {
                 if(e.getPropertyName().equals("viewportPosition")){
                    double scaleX, scaleY;
                    int wid = renderGrid.getViewportDimension().width;
                    int ht = renderGrid.getViewportDimension().height;
                    scaleX = viewer.getImageWidth()/wid;
                    scaleY = viewer.getImageHeight()/ht;
                    if(e.getNewValue() instanceof Point){
                       Point p =(Point)e.getNewValue();
                       viewer.pan(-p.x*scaleX, -p.y*scaleY); //Pan the viewer
                    }
                 }
              }
           }
      );
    }
}
```

```
/**
 * A Main Panel to show an image in the left column of Main Frame.
 *
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */
package pserc;
import java.awt.*;
import java.awt.image.*;
import java.awt.geom.*;
import java.awt.image.renderable.*;
import javax.media.jai.*;
import java.util.Vector;

public class PL_RenderedImageCanvas  extends PL_ImageCanvas {
  /**
   * Default viewer size.
   */
  protected int viewerWidth  = 480, viewerHeight  = 400;
  /**
   * Images to display and orignal.
   */
  transient protected PlanarImage  displayImage , origImage ;
  /**
   * Tile size (fixed).
   */
  protected int tileWidth  = 256, tileHeight  = 256;
  /**
   * SampleModel for generating new image
   */
  transient protected SampleModel  sampleModel ;
  /**
   * ColorModel for generating new image.
   */
  protected ColorModel  colorModel ;
  /**
   * maximum index of tile.
   */
  protected int maxTileIndexX , maxTileIndexY ;
  /**
   * maximum cordinate of tile.
   */
  protected int maxTileCordX , maxTileCordY ;
  /**
   * minimum index of tile.
   */
  protected int minTileIndexX , minTileIndexY ;
  /**
   *  minimum cordinate of tile.
   */
  protected int minTileCordX , minTileCordY ;
  /**
   * tile offset.
   */
  protected int tileGridXOffset , tileGridYOffset ;
  /**
   * image size.
   */
  public static int imageWidth  =0, imageHeight  =0;
  /**
   * cache for tiled image.
   */
  protected TileCache  tc;
  /**
   * Region of Interest (ROI) object to collect the envelop area.
   */
```

```
71    public static ROI Roi=null;
72    /**
73     * Store the Roi shapes for drawing.
74     */
75    public Vector shapes=new Vector();
76    /**
77     * Geographical utility object.
78     */
79    public static PL_Geo geo=null;
80
81    /**
82     * Default Constructor.
83     */
84    public PL_RenderedImageCanvas () { }
85
86    /**
87     * Original Constructor.
88     * @param img PlanarImage Image to be put on the panel.
89     */
90    public PL_RenderedImageCanvas (PlanarImage  img){
91     this();
92     setImage(img); // set the input image object on the panel
93    }
94
95    /**
96     * Set an image object on this Panel.
97     *
98     * @param img PlanarImage to be set.
99     */
100   public void setImage(PlanarImage  img){
101    origImage  = img;              // Make a pointer to the original image
102    panX =0; panY =0;              // Set the pan value as (0, 0)
103    atx = AffineTransform.getTranslateInstance (0.0, 0.0); // Set the default Translate materix
104    RenderedOp  op = makeTiledImage(img);          // Call this makeTiledImage function for making tiled
    image
105
106    displayImage  = op.createInstance();           // create a display image copied from original image
107    sampleModel  = displayImage.getSampleModel(); // assign the smaple model from the image
108    colorModel  = displayImage.getColorModel();    // assign the color model of the image
109    getTileInfo(displayImage);                     // get tile information for display image
110    fireTilePropertyChange();                      // change the properties of tile
111    imageDrawn  = false;                           // image drawn is set as false in default
112    repaint();
113    geo=new PL_Geo();                              // Initialize Geographical information
114   }
115
116   /**
117    * Get display image.
118    * @return PlanarImage displayed.
119    */
120   public PlanarImage  getDisplayImage(){return displayImage;}
121
122   /**
123    * Make tiled image from image object.
124    * @param img PlanarImage input mutlispectral Tiff image.
125    * @return RenderedOp 4 bands tiled image.
126    */
127   protected RenderedOp  makeTiledImage(PlanarImage  img) {
128    ImageLayout  tileLayout  = new ImageLayout(img);    // set the layout
129    tileLayout.setTileWidth(tileWidth);                 // set tile width and height
130    tileLayout.setTileHeight(tileHeight);
131    RenderingHints  tileHints  = new RenderingHints(JAI.KEY_IMAGE_LAYOUT, tileLayout);
132    ParameterBlock  pb = new ParameterBlock();
133    pb.addSource(img);
134    RenderedOp  op=JAI.create("format", pb, tileHints); //generate RenderedOp object with tile info
135    return JAI.create("BandSelect", (PlanarImage)op, new int[] {0,1,2,3});
136   }
137
138
139   /**
```

```
140     * Add ROI shapes into the global "Roi".
141     * @param x0 double for first Transmission tower X position.
142     * @param y0 double for first Transmission tower Y position.
143     * @param x1 double for second Transmission tower X position.
144     * @param y1 double for second Transmission tower Y position.
145     * @param rad double for radius.
146     */
147    public static void addEnvelop ( double x0, double y0, double x1, double y1, double rad){
148      double xv=x1-x0;  // get the distance X in two towers
149      double yv=y1-y0;  // get the distance Y in two towers
150      double length=Math.sqrt(xv*xv + yv*yv);  // Lenght between two towers
151
152      if(length==0) return;
153      double cos=(xv*1.0)/length; // DOT PRODUCT : A(xv, yv)xB(1,0)=cosAB*lengthA*lengthB
154      if(cos>1.0) cos =1.0;
155      if(cos<-1.0) cos=-1.0;
156      double angle=Math.acos(cos);
157      if(yv<0) angle=-angle;
158
159      // Create a polygon envelop rectangle
160      Polygon p= new Polygon();
161      p.addPoint((int)x0, (int)(y0-rad));
162      p.addPoint((int)(x0+length), (int)(y0-rad));
163      p.addPoint((int)(x0+length), (int)(y0+rad));
164      p.addPoint((int)x0, (int)(y0+rad));
165
166      // Rotate the envelop polygon
167      AffineTransform rotation = AffineTransform.getRotateInstance(angle, x0, y0);
168      Shape shape=rotation.createTransformedShape(p);
169      //Add the envelop rectangle to ROI
170      if(Roi==null) Roi=new ROIShape(shape);
171      else Roi=Roi.add(new ROIShape(shape));
172      // Create and Add 2 circle to ROI
173      Ellipse2D ellipse0 = new Ellipse2D.Double((x0-rad), (y0-rad), 2*rad, 2*rad);
174      Ellipse2D ellipse1 = new Ellipse2D.Double((x1-rad), (y1-rad), 2*rad, 2*rad);
175      Roi=Roi.add(new ROIShape(ellipse0));
176      Roi=Roi.add(new ROIShape(ellipse1));
177    }
178
179    /**
180     * Get ans Store the tile information of image object.
181     * @param img PlanarImage dispaly image.
182     */
183    protected void getTileInfo(PlanarImage img) {
184      imageWidth = img.getWidth();        // image width
185      imageHeight = img.getHeight();      // image height
186      tileWidth = img.getTileWidth();     // tile width
187      tileHeight = img.getTileHeight();   // tile height
188      maxTileIndexX = img.getMinTileX()+img.getNumXTiles()-1; // maximum index of tile
189      maxTileIndexY = img.getMinTileY()+img.getNumYTiles()-1;
190      maxTileCordX = img.getMaxX();       // maximum tile position
191      maxTileCordY = img.getMaxY();
192      minTileIndexX = img.getMinTileX();  // minimum index of tile
193      minTileIndexY = img.getMinTileY();
194      minTileCordX = img.getMinX();       // minimum tile position
195      minTileCordY = img.getMinY();
196      tileGridXOffset = img.getTileGridXOffset(); // offset of tile
197      tileGridYOffset = img.getTileGridYOffset();
198    }
199
200    /**
201     * Set tile width.
202     * @param tw int.
203     */
204    public void setTileWidth(int tw){
205      tileWidth = tw;
206      setImage(displayImage);
207    }
208    /**
209     * Get tile width.
```

```
210      * @return int.
211      */
212     public int getTileWidth(){ return tileWidth;}
213
214     /**
215      * set Tile height.
216      * @param th int.
217      */
218     public void setTileHeight(int th){
219      tileHeight = th;
220      setImage(displayImage);
221     }
222
223     /**
224      * Get Tile height.
225      * @return int.
226      */
227     public int getTileHeight(){ return tileHeight;}
228
229     /**
230      * Get maximum tile index in X direction.
231      * @return int.
232      */
233     public int getMaxTileIndexX(){return maxTileIndexX;}
234
235     /**
236      * Get maximum tile index in Y direction.
237      * @return int.
238      */
239     public int getMaxTileIndexY(){return maxTileIndexY;}
240
241     /**
242      * Get image width.
243      * @return int.
244      */
245     public int getImageWidth(){return imageWidth;}
246
247     /**
248      * Get image height.
249      * @return int.
250      */
251     public int getImageHeight(){return imageHeight;}
252
253     /**
254      * Change the tile properties.
255      */
256     protected void firePropertyChange() {
257      firePropertyChange("maxTileIndexX", null, new Integer(maxTileIndexX));
258      firePropertyChange("maxTileIndexY", null, new Integer(maxTileIndexY));
259      firePropertyChange("tileWidth", null, new Integer(tileWidth));
260      firePropertyChange("tileHeight", null, new Integer(tileWidth));
261      firePropertyChange("transform", null, atx);
262     }
263     /**
264      * Override paint funciton to display each tile one by one.
265      * @param gc Graphics.
266      */
267     public void paintComponent(Graphics gc){
268      Graphics2D g = (Graphics2D)gc;              // Convert from Graphcs to Graphcs2D
269      Rectangle rect = this.getBounds();          // get the boundary rectangle of image
270      if((viewerWidth != rect.width) || (viewerHeight != rect.height)){viewerWidth = rect.width;viewerHe
    ight = rect.height;}
271      g.setColor(Color.black);
272      g.fillRect(0, 0, viewerWidth, viewerHeight); // fill the rectngle in Black
273      if(displayImage == null) return;
274      int ti =0, tj = 0;                          // tile index of X and Y directions
275      Rectangle bounds = new Rectangle(0, 0, rect.width, rect.height);
276      bounds.translate(-panX, -panY);             // move the boundary to pan position.
277
278      // Get the visiible tile indices
```

```
279     int leftIndex = displayImage.XToTileX(bounds.x);
280     if(leftIndex < minTileIndexX) leftIndex = minTileIndexX;
281     if(leftIndex > maxTileIndexX) leftIndex = maxTileIndexX;
282     int rightIndex = displayImage.XToTileX(bounds.x + bounds.width - 1);
283     if(rightIndex < minTileIndexX) rightIndex = minTileIndexX;
284     if(rightIndex > maxTileIndexX) rightIndex = maxTileIndexX;
285     int topIndex = displayImage.YToTileY(bounds.y);
286     if(topIndex < minTileIndexY) topIndex = minTileIndexY;
287     if(topIndex > maxTileIndexY) topIndex = maxTileIndexY;
288     int bottomIndex = displayImage.YToTileY(bounds.y + bounds.height - 1);
289     if(bottomIndex < minTileIndexY) bottomIndex = minTileIndexY;
290     if(bottomIndex > maxTileIndexY) bottomIndex = maxTileIndexY;
291
292     // Draw each tile
293     for(tj = topIndex; tj <= bottomIndex; tj++) {
294       for (ti = leftIndex; ti <= rightIndex; ti++) {
295
296         Raster raster = displayImage.getTile(ti, tj);  // Get the raster information
297         int width2 = raster.getWidth();                // get raster's width
298         int height2 = raster.getHeight();              // get raster's height
299         int bands = raster.getNumBands();              // get raster's band number (Supposed 4)
300         int[] pixels = new int[width2 * height2 * bands]; // pixel information of tile image
301         byte[] data = new byte[width2 * height2 * 3];  // target pixel data
302
303         raster.getPixels(ti * 256, tj * 256, width2, height2, pixels); // 256 x 256 is the tile size
    FIXED!!!!
304
305         int index = 0;
306         // For each pixel in a tile
307         for (int h = 0; h < height2; h++)
308           for (int w = 0; w < width2; w++) {
309             int red = pixels[index * 4 + 0];     // Get Red band pixel info
310             int inf = pixels[index * 4 + 3];     // Get Infrared band pixel info
311             float ndvi = (float) ( ( (float) (inf - red)) / ( (float) (red + inf))); // Cacluate NDVI
312             byte ndviByte=(byte)( (ndvi + 1.0) * 100.0);
313             data[index * 3 + 2] = (byte) ndviByte;
314             data[index * 3 + 1] = (byte) pixels[index * 4 + 1];         //Set the original Green in
    green band
315             data[index * 3 + 0] = (byte) pixels[index * 4 + 2];        //Set the original Blue in blue
    band
316
317             int thres=(int)(PL_FilterSlider.threshold*100.0)+100; // Get NDVI Theashold value from
    PL_FilerSlider
318             if(ndviByte>thres){ // If the NDVI value is more than Thesdhold, the pixel is WIHTE.
319               data[index * 3 + 2] = (byte) 255; //red
320               data[index * 3 + 1] = (byte) 255; //green
321               data[index * 3 + 0] = (byte) 255; //blue
322             }
323             index++;
324           }
325
326         // *** ROI Operation *** //
327         DataBufferByte dbuffer = new DataBufferByte(data, width2 * height2 * 3);
328         //Get SampleModel
329         sampleModel = RasterFactory.
330             createPixelInterleavedSampleModel (DataBuffer.TYPE_BYTE, width2, height2, 3);
331         //Get ColorModel
332         colorModel = PlanarImage.createColorModel(sampleModel);
333
334         //Create Buffered image and draw it in the proper tile position
335         WritableRaster wr = RasterFactory.createWritableRaster(sampleModel,dbuffer, new Point(0, 0));
336         BufferedImage bi = new BufferedImage(colorModel, wr, colorModel.isAlphaPremultiplied(), null);
337         int xInTile = displayImage.tileXToX(ti);
338         int yInTile = displayImage.tileYToY(tj);
339         AffineTransform tx = AffineTransform.getTranslateInstance(xInTile + panX, yInTile + panY);
340         g.drawRenderedImage(bi, tx);
341       }
342     }
343     imageDrawn = true;
344     // draw the Roi shapes boundary
```

```
345     if (Roi == null)return;
346     AffineTransform  tx = AffineTransform .getTranslateInstance (panX, panY);
347     Shape shape = tx.createTransformedShape (Roi.getAsShape ());
348     g.setStroke(new BasicStroke(2));       //Set the stroke size as 2
349     g.setPaint(new Color(255, 255, 255)); //Set as White
350     g.draw(shape);
351    }
352  }
```

```
/**
 * A Component to show the rendered area in the large image area.
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi.
 * @version 1.0.
 */
package pserc;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.geom.*;

public class PL_RenderGrid  extends JPanel {
  /**
   * maimum tile index in X and Y.
   */
  protected int maxTileIndexX, maxTileIndexY;
  /**
   * tile width and height.
   */
  protected int tileWidth, tileHeight;
  /**
   * view port position.
   */
  protected Point vpPos = new Point(0,0);
  /**
   * used for visilble area.
   */
  protected Rectangle currentShape;
  /**
   * pan value in X and Y.
   */
  protected int panX =0, panY =0;
  /**
   * height and width of this panel.
   */
  protected int width, height;
  /**
   * condition if it is drawn or not.
   */
  protected boolean dragOn = false;
  /**
   * scorll anchor position.
   */
  private Point scrollAnchor  = new Point(0,0);
  /**
   * transform matrix.
   */
  protected AffineTransform atx = new AffineTransform();
  /**
   * view prot size.
   */
  protected Dimension vpSize;

  /**
   * Default Constructor.
   */
  public PL_RenderGrid() {
     enableEvents(AWTEvent.MOUSE_MOTION_EVENT_MASK | AWTEvent.MOUSE_EVENT_MASK);
  }

  /**
   * get the viewe port size.
   * @return Dimension.
   */
  public Dimension getViewportDimension(){
     return new Dimension(width, height);
```

```
71      }
72
73      /**
74       * set tile numbers in X and Y.
75       * @param maxXIndex int.
76       * @param maxYIndex int.
77       */
78      public void setTileIndices (int maxXIndex, int maxYIndex){
79         this.maxTileIndexX  = maxXIndex;
80         this.maxTileIndexY  = maxYIndex;
81         repaint();
82      }
83      /**
84       * Set tile size.
85       * @param width int.
86       * @param height int.
87       */
88      public void setTileDimension (int width, int height){
89         tileWidth = width; tileHeight = height;
90         repaint();
91      }
92      /**
93       * Set tile width.
94       * @param tw int.
95       */
96      public void setTileWidth (int tw){
97         tileWidth = tw; repaint();
98      }
99      /**
100      * get Tile width.
101      * @return int.
102      */
103     public int getTileWidth(){ return tileWidth;}
104
105     /**
106      * set tile height.
107      * @param th int.
108      */
109     public void setTileHeight (int th){
110        tileHeight = th; repaint();
111     }
112     /**
113      * get tile height.
114      * @return int.
115      */
116     public int getTileHeight(){ return tileHeight;}
117     /**
118      * set maimum tile index in X direction.
119      * @param tix int.
120      */
121     public void setMaxTileIndexX (int tix){
122        maxTileIndexX = tix; repaint();
123     }
124     /**
125      * get maximum tile index in X direction.
126      * @return int.
127      */
128     public int getMaxTileIndexX (){return maxTileIndexX;}
129     /**
130      * set maximum tile index in Y direction.
131      * @param tiy int.
132      */
133     public void setMaxTileIndexY (int tiy){
134        maxTileIndexY = tiy; repaint();
135     }
136     /**
137      * get maimxum tile index in Y direction.
138      * @return int.
139      */
140     public int getMaxTileIndexY (){return maxTileIndexY;}
```

```
141     /**
142      * set transform matrix.
143      * @param at AffineTransform.
144      */
145     public void setTransform(AffineTransform  at) {atx = at;}
146     /**
147      * set view prot size.
148      * @param size Dimension.
149      */
150     public void setViewportSize (Dimension  size){vpSize = size;}
151
152     /**
153      * Override and implement mouse events.
154      * @param e MouseEvent.
155      */
156     public void  processMouseEvent (MouseEvent  e){
157        switch(e.getID()){
158            case MouseEvent.MOUSE_PRESSED :  // In mouse pressed
159               setCursor(new Cursor(Cursor.MOVE_CURSOR));
160               int x = e.getX();
161               int y = e.getY();
162               // if the mouse is insize the shape, set the dragOn as ture
163               if(currentShape.contains(x,y)) {  dragOn = true;}
164               break;
165            case MouseEvent.MOUSE_CLICKED :
166               break;
167            case MouseEvent.MOUSE_RELEASED : // In mouse released
168               setCursor(Cursor.getDefaultCursor());
169               dragOn = false; //Set the dragOn as false
170               break;
171        }
172     }
173     /**
174      * Override and implement mouse motion events.
175      * @param e MouseEvent.
176      */
177     public void processMouseMotionEvent (MouseEvent  e){
178         switch(e.getID()){
179            case MouseEvent.MOUSE_DRAGGED :
180               if(dragOn) scroll(e.getX(), e.getY()); // if dragOn is true, call scroll the screen.
181               break;
182        }
183     }
184     /**
185      * Stop the scroll event.
186      */
187     public void stopScroll(){ setCursor(Cursor.getDefaultCursor());}
188     /**
189      * Start the scroll event.
190      * @param x int.
191      * @param y int.
192      */
193     public void startScroll(int x, int y){
194        if(dragOn) return;
195        scrollAnchor.x = x- panX;
196        scrollAnchor.y = y- panY;
197        repaint();
198     }
199     /**
200      * Scroll the panel by changing the panX and PanY.
201      * @param x int.
202      * @param y int.
203      */
204     public void scroll(int x, int y){
205        panX = x-scrollAnchor.x;
206        panY = y-scrollAnchor.y;
207        setPan(new Point(panX, panY));
208        repaint();
209     }
210     /**
```

```
211      * Set the pan value.
212      * @param pos Point.
213      */
214     public void setPan(Point pos){
215        firePropertyChange ("viewportPosition" ,this.vpPos, pos);
216        int x = (int)( pos.x * (maxTileIndexX *tileWidth)/(width-20));
217        int y = (int)( pos.y * (maxTileIndexY *tileHeight)/(height-20));
218        this.vpPos = new Point(x,y);
219        repaint();
220     }
221     /**
222      * Set view port position.
223      */
224     public void setViewportPosition (Point vpPos){
225        this.vpPos = vpPos; repaint();
226     }
227     /**
228      * Override paint function to draw grid lines and visible rectangle.
229      * @param gc Graphics.
230      */
231     public void paintComponent (Graphics gc) {
232        Graphics2D g = (Graphics2D)gc;              // Convert Graphcs to Graphcs2D objects
233        Rectangle bounds = this.getBounds();       // Get the boundry box for this panel
234        g.setColor(Color.black);                   // Set the black brash
235        width = bounds.width-20;                   // width is set 20 minus from the box
236        height = bounds.height-20;                 // height is set 20 minus from the box
237        double gridWidth =  (width)/(double)maxTileIndexX;  //set grid width
238        double gridHeight = (height)/(double)maxTileIndexY; //set grid hieght
239        g.fillRect(0,0,bounds.width, bounds.height);        // fill the background
240        if ((maxTileIndexX == 0)|| (maxTileIndexY == 0))  return;
241        g.setColor(Color.blue);                    // set the color as blue
242        double vertStartX = 10.0; double vertStartY = 10.0;
243        double vertEndX = 10.0;
244        double vertEndY = (double)maxTileIndexY *gridHeight +vertStartY;
245        // *** Draw horizontal lines for grid
246        for(int i= 0; i<maxTileIndexX; i++) {
247            g.drawLine((int)vertStartX, (int)vertStartY, (int)vertEndX, (int)vertEndY);
248            vertStartX += gridWidth;
249            vertEndX   += gridWidth;
250        }
251        g.drawLine((int)vertStartX, (int)vertStartY, (int)vertEndX, (int)vertEndY);
252        double horizStartX = 10.0; double  horizStartY = 10.0;
253        double horizEndX = maxTileIndexX *gridWidth +horizStartX;
254        double horizEndY = 10.0;
255        // STEP 1 *** Draw vertical lines for grid
256        for(int i=0; i<maxTileIndexY; i++) {
257            g.drawLine((int)horizStartX, (int)horizStartY,(int)horizEndX, (int)horizEndY);
258            horizEndY += gridHeight;
259            horizStartY   += gridHeight;
260        }
261        g.drawLine((int)horizStartX, (int)horizStartY, (int)horizEndX, (int)horizEndY);
262
263        // STEP 2 *** Draw visible red box
264        g.setColor(Color.red);
265        vertStartX = 10.0; vertStartY = 10.0;
266        int vpWid, vpHt;
267        if(vpSize == null){ vpWid = (int)gridWidth; vpHt = (int)gridHeight;}
268        else {vpWid = vpSize.width;vpHt = vpSize.height;}
269        if(dragOn) {
270           currentShape  = new Rectangle(-(int)vertStartX+panX,
271                                         -(int)vertStartY+panY,vpWid, vpHt);
272         }else {
273           int x = (int)(vertStartX +
274                    vpPos.x * ((width)/(double)(maxTileIndexX *tileWidth)));
275           int y = (int)(vertStartY +
276                    vpPos.y * ((height)/(double)(maxTileIndexY *tileHeight)));
277           currentShape  = new Rectangle(x,y,vpWid,vpHt);
278        }
279        g.draw(currentShape);
```

```
281       // STEP 3 *** draw Tower and lines
282        if(PL_RenderedImageCanvas .geo==null) return;
283        boolean roi_exist=false;
284        if(PL_RenderedImageCanvas .Roi != null) roi_exist=true;
285        // Draw a circle and line from the forth row in PL_DataTabel.
286        //    The first 3 rows are used for lefttop, righttop, and leftbottom points of image
287        for(int i=3; i<PL_DataTable .model.getRowCount()-1; i++){
288          Point p=PL_Geo.pixelPos((String)PL_DataTable .model.getValueAt(i, 1),
289                                  (String)PL_DataTable .model.getValueAt(i, 2));  // Get the current
    tower
290          Point p2=PL_Geo.pixelPos((String)PL_DataTable .model.getValueAt(i+1, 1),
291                                  (String)PL_DataTable .model.getValueAt(i+1, 2)); // Get the next tower
292
293          double radius=30.0/PL_Geo.EW_scale; //************radius = 30 meter  ****08062006
294          if(roi_exist==false) PL_RenderedImageCanvas .addEnvelop(p.x, p.y, p2.x, p2.y, radius);// add
    ROI shapes from the towers
295
296          // Draw a small circle for tower and line for power line
297          g.setColor(Color.white);   // Set the color as white
298          int sx=(int)((double)p.x/(double)256*gridWidth);   //256 is the tile size
299          int sy=(int)((double)p.y/(double)256*gridHeight);
300          int ex=(int)((double)p2.x/(double)256*gridWidth);
301          int ey=(int)((double)p2.y/(double)256*gridHeight);
302          g.drawOval(sx-5, sy-5, 11,11);  // draw a circle with radias 11
303          g.drawOval(ex-5, ey-5, 11,11);
304          g.drawLine(sx, sy, ex,ey);      // draw a line between two towers
305        }
306    }
307  }
```

```
1  /**
2  * A ScrollGUI to control the scroll action from mouse event and mouse motion event.
3   * <p>Title: PSerc Project</p>
4   * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
5   * <p>Copyright: Copyright (c) 2005</p>
6   * <p>Company: ASU</p>
7   * @author Yoshihiro Kobayashi.
8   * @version 1.0.
9   */
10 package pserc;
11 import java.awt.event.*;
12 import javax.swing.*;
13
14 public class PL_ScrollGUI implements  MouseListener,MouseMotionListener {
15   /**
16    * Target object to be implmented this interface.
17    */
18   protected PL_ScrollController  scrollController;
19   /**
20    * Condition if scroll is on or off.
21    */
22   protected boolean scrollOn = true;
23   /**
24    * Condition if mouse pressed or not.
25    */
26   protected boolean mousePressed = false;
27
28     /**
29      * Origianl Constructor with PL_ScrollController interface.
30      * @param c PL_ScrollController.
31      */
32     public PL_ScrollGUI(PL_ScrollController  c){scrollController  = c;}
33
34     /**
35      * Set the scroll condition flag.
36      * @param onOff boolean.
37      */
38     public  void setScrollOn(boolean onOff){ scrollOn = onOff;}
39
40     /**
41      * Initialize the scroll condition flag as true.
42      */
43     public void init() { setScrollOn(true);}
44
45     /**
46      * get Scroll condition.
47      * @return boolean.
48      */
49     public  boolean getScrollOn(){ return scrollOn; }
50     /**
51      * Start the scroll.
52      */
53     public void startScroll(){ scrollOn = true;}
54
55     /**
56      * Override the mouse pressed event.
57      * @param e MouseEvent.
58      */
59     public void mousePressed(MouseEvent  e) {
60        if(!scrollOn) return;
61        if(mousePressed) return;
62        mousePressed  = true;                // Set the mouse pressed flag as true
63        if(SwingUtilities.isLeftMouseButton(e)){
64           scrollController.startScroll(e.getX(), e.getY()); //start the scrolling
65        }
66     }
67     /**
68      * Override the mouse released event.
69      * @param e MouseEvent.
70      */
```

```
71      public void mouseReleased(MouseEvent e) {
72         scrollController.stopScroll(); //stop the scrolling
73         mousePressed = false; // set the mouse pressed flag as false
74      }
75      /**
76       * Override the mouse clicked event.
77       * @param e MouseEvent.
78       */
79      public void mouseClicked(MouseEvent e){}
80      /**
81       * Override the mouse entered event.
82       * @param e MouseEvent.
83       */
84      public void mouseEntered(MouseEvent e){}
85      /**
86       * Override the mosue exited event.
87       * @param e MouseEvent.
88       */
89      public void mouseExited(MouseEvent e){}
90
91      /**
92       * Overridet the mouse dragged event.
93       * @param e MouseEvent.
94       */
95      public void mouseDragged(MouseEvent e){
96         if(SwingUtilities.isLeftMouseButton(e)){
97            if(scrollOn) scrollController.scroll(e.getX(), e.getY()); //scroll the position
98         }
99      }
100     /**
101      * Override the mouseMoved function.
102      * @param e MouseEvent.
103      */
104     public void mouseMoved(MouseEvent e){}
105     /**
106      * Reset the scroll conditions.
107      */
108     public void reset(){
109        if(scrollController != null)scrollController.stopScroll(); //Stop the scroll
110        setScrollOn(false); // Set the ScrollOn is false
111     }
112  }
```

```
/**
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */
package pserc;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.media.jai.JAI;
import javax.media.jai.RenderedOp;
import javax.media.jai.PlanarImage;
import javax.media.jai.TiledImage;
import java.awt.image.SampleModel;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;
import java.awt.image.*;
import java.awt.image.renderable.*;
import javax.media.jai.*;
import java.io.*;

public class PL_StereoTest extends JFrame{

  Container p;                      // Container of Main Frame
  PlanarImage pi, pi2;              // Stereo pair images
  Raster inputRaster, inputRaster2; // Raster data for stereo pair images
  int imageWidth, imageHeight;      // Image size
  int maxTileX, maxTileY;           // X and Y Tile maximum indices
  RenderedOp op;                    // Output image

  int filterSize=4;                  // filterSize 4 = 9x9 matrix for calculate correlation
  int hightOffset=-45;               // search left end (45 pixels)
  int hightOffset2=-25;              // search right end (25 pixels)
  String outputFilename;             // Output file name

  //*** Main function to generate and save DEM file from a stereo pair images
  public static void main(String[] args){
    PL_StereoTest f=new PL_StereoTest();
  }

  //*** Load stereo images and set their size and raster data
  private void loadImage(){
    pi = JAI.create("fileload", "0000010000.tif"); // load the first image of stereo pair
    pi2= JAI.create("fileload", "0010000000.tif"); // load the seconf image of stereo pair
    outputFilename ="dem9x9offset20_2007_08_14.tif" ; // Set the output DEM file name
    imageWidth=pi.getWidth();                         // Set the image width
    imageHeight=pi.getHeight();                       // set the image height
    inputRaster = pi.getData();                       // set the raster data of first image
    inputRaster2 = pi2.getData();                     // set the raster data of second image
  }
 //*** Set tiled images to reduce calculation time
  private void makeTiledImage(){
    ImageLayout tileLayout = new ImageLayout(pi);// set the layout
    tileLayout.setTileWidth(128);                // set tile width and height
    tileLayout.setTileHeight(128);
    RenderingHints tileHints = new RenderingHints(JAI.KEY_IMAGE_LAYOUT, tileLayout);
    ParameterBlock pb = new ParameterBlock();
    pb.addSource(pi);
    op=JAI.create("format", pb, tileHints); //generate RenderedOp object with tile info
    maxTileX=op.getNumXTiles();             //set the maximum indices of X and Y
    maxTileY=op.getNumYTiles();
  }
  //*** Temporary test function. It is not used anymore ***//
  public static void createTEXT(String filename){
    try{
      FileReader reader=new FileReader(filename);
      StreamTokenizer st = new StreamTokenizer(reader);
```

```
71        st.eolIsSignificant (true);     st.wordChars (' ','_');
72
73        FileOutputStream  file = new FileOutputStream ("Degree"+filename);
74        DataOutputStream  stream = new DataOutputStream (file);
75
76        double D = 0.d;
77        double M = 0.d;
78        double S=0.0d;
79        double DL = 0.d;
80        double ML = 0.d;
81        double SL =0.0d;
82
83        stream.writeBytes ("LA"+"\t"+"LO"+"\r\n");
84        while (true) {
85              st.nextToken ();
86              if (st.ttype==StreamTokenizer .TT_EOF) break;
87              else if(st.ttype==StreamTokenizer .TT_EOL){
88                if(st.nextToken ()==StreamTokenizer .TT_NUMBER) D=st.nval;
89                if(st.nextToken ()==StreamTokenizer .TT_NUMBER) M=st.nval;
90                if(st.nextToken ()==StreamTokenizer .TT_NUMBER) S=st.nval;
91                if(st.nextToken ()==StreamTokenizer .TT_NUMBER) DL=st.nval;
92                if(st.nextToken ()==StreamTokenizer .TT_NUMBER) ML=st.nval;
93                if(st.nextToken ()==StreamTokenizer .TT_NUMBER) SL=st.nval;
94                System.out.println(D+":"+M+":"+S+",  -"+DL+":"+ML+":"+SL);
95                double  La=(double)D+(double)M/60.0+(double)S/3600.0;
96                double  Lo=-((double)DL+(double)ML/60.0+(double)SL/3600.0);
97                stream.writeBytes (La+"\t"+Lo+"\r\n");
98              }
99        }
100       reader.close();
101       stream.close();
102     }catch(IOException  e){System.out.println("Reading DXF plines file failed" );}
103
104   }
105   //*** Main Constructor ***//
106   public PL_StereoTest () {
107     super();                        // call parent class constructor
108     p=this.getContentPane ();       // Set the container of Main window
109
110     this.addWindowListener (new WindowAdapter () { // Close and Stop function
111       public void windowClosing (WindowEvent  e) {
112         System.exit(0);
113       }
114     });
115
116     //Step1: Import two images
117     loadImage ();
118
119     //Step2: Create a writable buffer for new pixels
120     WritableRaster  outputRaster  = inputRaster.createCompatibleWritableRaster ();
121
122     //step3: Create a tiledImage for display
123     makeTiledImage ();
124
125     //Step4: for each tiled image to calculate the maxium correlation
126     SampleModel  sampleModel =RasterFactory.createBandedSampleModel (DataBuffer.TYPE_FLOAT, 128, 128, 1
    );
127     ColorModel  colorModel =PlanarImage.createColorModel (sampleModel);
128     TiledImage  tiledImage =new TiledImage (0, 0, imageWidth, imageHeight, 0,0, sampleModel, colorModel
    );
129
130     // For each tiled image
131     for(int tj=tiledImage.getMinTileY (); tj<tiledImage.getNumYTiles (); tj++)
132       for(int ti=tiledImage.getMinTileX (); ti<tiledImage.getNumXTiles (); ti++){
133
134         float[] imageData=new float[128*128]; // keep the distanct to the point with maximum
    correlation
135         float maxPos = hightOffset2;          // Search right limit for the maximum correlation point
136         float minPos = hightOffset;           // Search left limit for the maximum correlation point
137         int count = 0;
```

```
138          for (int j = 0; j < 128; j++) {        // Image is tiled as 128x128 pixels
139            for (int i = 0; i < 128; i++) {
140              int xIndex = tiledImage.tileXToX(ti)+i;    // X potioin in original image
141              int yIndex = tiledImage.tileYToY(tj)+j;    // Y position in original image
142              float[] pos = new float[1];                // Local variable to save the distanct to the
     point with maximum correlation
143              pos[0] = (float) get_max_corrlation_xposition (xIndex, yIndex); // *** MAIN FUNCTION to
     calcualte MAX-correlation
144              if (maxPos < pos[0]) pos[0] = maxPos; // Set minimum posiiton for calcuaiton error
145              if (minPos > pos[0]) pos[0] = minPos; // Set maximum position for calculation error
146              imageData[count] = (float) ( (pos[0] - minPos) / (maxPos - minPos));  // normalized
     distance to maximum correlation point
147              count++;
148            }
149          }
150          // Save the array data of distance to maximum correlation points in Output image
151          javax.media.jai.DataBufferFloat  dbuffer = new javax.media.jai.
152              DataBufferFloat (imageData, 128 * 128);
153          Raster raster = RasterFactory.createWritableRaster (sampleModel, dbuffer,
154              new Point(128*ti, 128*tj));
155          tiledImage.setData(raster);
156        }
157      JAI.create("filestore", tiledImage, "floatpattern.tif", "TIFF"); // Save the output image as TIFF
     file
158    }
159
160    // *** Local function to get a set of pixel values surrounding (xpos, ypos) pixels as a matrix
161    private int[] getMatrix(int xPos, int yPos, int imageID){
162      int filterMatrix=filterSize*2+1;                    //  Matrix size
163      int[] matrix=new int[filterMatrix*filterMatrix];    //  Matrix values
164      int index=0;
165      // for each pixel position repeating matrix such as 81 times for 9x9 and 25 times for 5x5 matrix
166      for(int i=yPos-filterSize; i<=(yPos+filterSize); i++){
167        for (int j = xPos - filterSize; j <= (xPos + filterSize); j++) {
168          if(i < 0 || i>=imageHeight  || j<0 || j>=imageWidth) matrix[index]=0;
169          else if(imageID==0) matrix[index]=inputRaster.getSample(j, i, 0);
170          else if(imageID==1) matrix[index]=inputRaster2.getSample(j, i, 0);
171          index++;
172        }
173      }
174      return matrix;
175    }
176    // A local function to get Cross-correlation of _a anb _b matrices
177    // This is called by the get_max_corrlation_xposition function
178    private float get_corrlation(int[] _a, int[] _b){
179     // C( (i1, j1), (i2, j2))= {V1(i1, j1)*V2(i2, j2) - u1*u2)/(o1*o2)
180     // V1 (i1, j1) = template matrix surrounding of pixel position (i1, j1)
181     // u1, u2 = mean of the template of V1 and V2
182     // o1, o2 = root mean sqaure
183     int length=_a.length;                      // The length of matrix such as 81 for 9x9 matrix
184     float _a_mean=0.0f;  float _b_mean=0.0f;  // The mean value in each matrix
185     float _a_SD=0.0f;    float _b_SD=0.0f;    // The standard diviation value in each matrix
186     float correlation=0.0f;                    // correlation value
187     // calculate the mean value for each matrix
188     for(int i=0; i<length; i++){
189       _a_mean += (float)_a[i];
190       _b_mean += (float)_b[i];
191     }
192     _a_mean = _a_mean/length;
193     _b_mean = _b_mean/length;
194
195     // calculate the SD and scalar product
196     for(int i=0; i<length; i++){
197       _a_SD += (_a[i] - _a_mean) *(_a[i] - _a_mean);
198       _b_SD += (_b[i] - _b_mean) *(_b[i] - _b_mean);
199       correlation += (_a[i] - _a_mean) * (_b[i] - _b_mean);
200     }
201     // calculate the cross-correlation
202     if(_a_SD*_b_SD == 0) correlation =0.0f;
203     else correlation =(float) (correlation/Math.sqrt(_a_SD*_b_SD));
```

```
204      return correlation;
205    }
206
207     // A local function to get maximum corrlation at (xPox, yPos) pixel of a image
208     private float get_max_corrlation_xposition (int xPos, int yPos){
209       int[] V1=getMatrix(xPos, yPos, 0);     // Get a matrix around (xPos, yPos)
210       boolean allZero=true;                  // flag for no-matching case
211       for(int i=0; i<V1.length; i++){        // check if the values in matrix is all 0s
212         if(V1[i] !=0) allZero=false;
213       }
214       if(allZero) return 0.0f;               // If all values in Matrix =0, then return 0.
215       float pre_correlation  = 0.0f;         // Previous orrelation value
216       float nxt_correlation  = 0.0f;         // Next Correlation value
217       float max_correlation  = 0.0f;         // Maximum correlation value during the serach
218       float current_correlation  =0.0f;      // Current correlation value for xPos
219       int    max_index=0;
220       for(int i=xPos+hightOffset; i<xPos+hightOffset2  ; i++){
221         if( i<0 || i>=imageWidth) continue;
222
223         int[] V2=getMatrix(i, yPos, 1);      //Get a matrix of X=i position from the second image
224         float correlation = get_corrlation(V1, V2); // *** Call get_corrlation value of V1 and V2
    matrices
225
226         // Update the curren, next, previous corrlation values
227         if(correlation  > max_correlation ){
228           pre_correlation  = current_correlation ;
229           max_correlation  = correlation;
230           max_index = i;
231           if(i==(imageWidth-1)) nxt_correlation  =0.0f;
232         }
233         else if(i==(max_index+1)) nxt_correlation  = correlation;
234         current_correlation  = correlation;
235       }
236       // calculate the real position (not integer) using the previous and next correlation value
237       float sub_pos = (hightOffset  + hightOffset2 )/2;
238       if((2.0f*(pre_correlation  + nxt_correlation  - 2.0*max_correlation ))==0){}
239       else sub_pos = ((float)max_index) - (nxt_correlation  - pre_correlation )/(float)(2.0f
    *(pre_correlation  + nxt_correlation  - 2.0*max_correlation ));
240       return sub_pos-(float)xPos;
241     }
242   }
```

```
1  /**
2   * <p>Title: PSerc Project</p>
3   * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
4   * <p>Copyright: Copyright (c) 2005</p>
5   * <p>Company: ASU</p>
6   * @author Yoshihiro Kobayashi
7   * @version 1.0
8   */
9  package pserc;
10 import javax.swing.*;
11 import java.awt.*;
12 import java.awt.event.*;
13
14 public class PL_Output extends JFrame{
15
16   Container p;                          // Container of window
17   public JPanel controlP=new JPanel();  // Main Panel
18
19   public static void main(String[] args){
20     PL_Output f=new PL_Output();
21   }
22   public PL_Output(){
23     super();                                    // call parent's constructor
24     p=this.getContentPane();                    // get container
25     p.setLayout(new BorderLayout());            // set border layout design
26     setSize(1000, 800);                         // set size of window
27     this.setVisible(true);
28     this.addWindowListener(new WindowAdapter() {// add close function
29       public void windowClosing(WindowEvent e) {
30         System.exit(0);
31       }
32     });
33     PL_OutputControlPanel  cP=new PL_OutputControlPanel(this); // create Control Panel
34     PL_OutputHistograph  hist=new PL_OutputHistograph();       // create Histograph panel
35     p.add(hist, BorderLayout.CENTER);           // add the panels to container
36     p.add(cP, BorderLayout.EAST);
37   }
38 }
```

```
/**
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */
package pserc;
import javax.swing.*;
import java.awt.*;
import javax.swing.event.*;

public class PL_OutputControlPanel  extends JPanel{

  JPanel geoPanel =new JPanel(); //  Specify the PL_Geo coordinate
  JPanel NDVIPanel=new JPanel(); //  Control NDVI threashold value
  JPanel towerPanel=new JPanel(); // put the location of towers
  JPanel treePanel=new JPanel(); //  information of tree locations
  JScrollPane sp1;  JPanel eastPanel=new JPanel();

  // GUI components
  public static JCheckBox showRef=new JCheckBox("Show reference points"); //checkbox for 3 refrence points
  public static JCheckBox showTower=new JCheckBox("Show Towers");      // checkbox for visibility of transmission towers
  public static JCheckBox showTreeID=new JCheckBox("Show Tree IDs");   // checkbox for visibility of tree ID number
  public static JCheckBox showTreeBox=new JCheckBox("Show Tree Box");  // checkbox for visibility of tree location
  public static JSlider NDVISlider=new JSlider(0, 200);          //slider to control NDVI theashold
  public static JSlider ScaleSlider=new JSlider(0, 200);         //slider to control scale of height
  public static JTextArea treeInfo=new JTextArea();              //component to put tree info
  public static JLabel  NDVI=new JLabel("NDVI Theshold = 0.20"); //label of NDVI
  public static JLabel  SCALE=new JLabel("SCALE Factor = 25");   //label of scale of height
  public static JTextField tower0_T, tower0_TT, tower0_TTT, // towerI_TT (latitude)
      tower1_T, tower1_TT, tower1_TTT,                      // towerI_TT (longitude)
      tower2_T, tower2_TT, tower2_TTT;                      // towerI_TTT (distance to lines)

  PL_Output frame; //pointer to parent component

  // *** Override the size set function *** //
  public Dimension getPreferredSize(){
    sp1.setPreferredSize(new Dimension(220, frame.getHeight()-40));
    return new Dimension(240, frame.getHeight());
  }
  // *** Main Constructor *** //
  public PL_OutputControlPanel(PL_Output _f) {
    this.setPreferredSize(new Dimension(240, 1000));
    frame=_f;

    // Add 4 major components to the scroll panel
    sp1=new JScrollPane(eastPanel);
    sp1.setPreferredSize(new Dimension(220, frame.getHeight()));
    eastPanel.setPreferredSize(new Dimension(200, 1300));
    eastPanel.add(geoPanel);
    eastPanel.add(NDVIPanel);
    eastPanel.add(towerPanel);
    eastPanel.add(treePanel);
    // Add titles to each sub panel
    geoPanel.setBorder(BorderFactory.createTitledBorder("Reference Point"));
    NDVIPanel.setBorder(BorderFactory.createTitledBorder("Control NDVI and Tower Scale"));
    towerPanel.setBorder(BorderFactory.createTitledBorder("Tower Location"));
    treePanel.setBorder(BorderFactory.createTitledBorder("Tree Location"));
    this.add(sp1);

    createGeoPanel(); // sub funciton 1 to add geographical panel
    createNDVISlider(); // sub function 2 to add NDVI control unit
    createTowerPanel(); // sub function 3 to add Tower location data
    createTreePanel();  // sub function 4 to add tree locations
```

```
67
68      }
69      // *** sub function called by constructor to create tree informaiton panel ***//
70      public  void createTreePanel (){
71        treePanel.setLayout(new BorderLayout(5,5));           // set borderlayout with (5,5) gaps
72        JPanel checkPanel=new JPanel(new GridLayout(2,1,3,3));// set the layout (2x1)
73        checkPanel.add(showTreeID); //add checkboc for treeID
74        checkPanel.add(showTreeBox);//add checkbox for tree boundary boxes
75        treePanel.add(checkPanel, BorderLayout.NORTH); //checkboxes are added to treepanel
76
77        JScrollPane treeInfoScroll = new JScrollPane(treeInfo); // create scroll panel for tree info
78        treePanel.add(treeInfoScroll, BorderLayout.CENTER);  // add the scroll panel to tree panel
79        treePanel.setPreferredSize(new Dimension(200, 450)); // set the size of tree panel
80
81      }
82      // *** sub function called by constructor to create tower information ***//
83      private void createTowerPanel (){
84        towerPanel.setLayout(new BorderLayout(5,5));         //set borderlayout with (5,5)gaps
85        towerPanel.add(showTower, BorderLayout.NORTH);       //add the checkbox for tower visibility
86        towerPanel.setPreferredSize(new Dimension(200, 300)); //set the size of panel
87
88        JPanel towerP=new JPanel(new GridLayout(3, 1, 2, 2)); // create a sub panel
89        JPanel tower0=new JPanel(new GridLayout(4, 1, 0, 0)); // create a sub-sub panel
90        JLabel tower0_L=new JLabel("Tower0");                 // create a label for tower index
91        tower0_T=new JTextField(""+32.75928019950938 );       // latitude of tower location
92        tower0_TT=new JTextField(""+117.19934737424919 );     // longitude of tower location
93        tower0_TTT=new JTextField("24.0");                    // altitude of tower location
94        tower0.add(tower0_L);                                 // add text fields above
95        tower0.add(tower0_T);
96        tower0.add(tower0_TT);
97        tower0.add(tower0_TTT);
98        towerP.add(tower0);
99
100       JPanel tower1=new JPanel(new GridLayout(4, 1, 0, 0)); // create a sub-sub panel
101       JLabel tower1_L=new JLabel("Tower1");                 // create a label for tower index
102       tower1_T=new JTextField(""+32.76058333333333 );       // latitude of tower location
103       tower1_TT=new JTextField(""+117.19897222222224 );     // longitude of tower location
104       tower1_TTT=new JTextField("10.0");                    // altitude of tower location
105       tower1.add(tower1_L);                                 // add text fields above
106       tower1.add(tower1_T);
107       tower1.add(tower1_TT);
108       tower1.add(tower1_TTT);
109       towerP.add(tower1);
110
111       JPanel tower2=new JPanel(new GridLayout(4, 1, 0, 0)); // create a sub-sub panel
112       JLabel tower2_L=new JLabel("Tower2");                 // create a label for tower index
113       tower2_T=new JTextField(""+32.76355686675233 );       // latitude of tower location
114       tower2_TT=new JTextField(""+117.19828677534942 );     // longitude of tower location
115       tower2_TTT=new JTextField("24.0");                    // altitude of tower location
116       tower2.add(tower2_L);                                 // add text fields above
117       tower2.add(tower2_T);
118       tower2.add(tower2_TT);
119       tower2.add(tower2_TTT);
120       towerP.add(tower2);
121       towerPanel.add(towerP, BorderLayout.CENTER);
122
123     }
124     // *** sub function called by constructor to create NDVI control unit ***//
125     private void createNDVISlider (){
126       NDVIPanel.setPreferredSize(new Dimension(200, 150)); //set the size
127       NDVIPanel.add(NDVISlider);                           //add the slider for NDVI theshold
128       NDVISlider.setValue(120);                            // set initial value as 120
129       NDVIPanel.add(NDVI);                                 //add NDVI label
130       NDVIPanel.add(ScaleSlider);                          //add the slider for scaling height
131       NDVIPanel.add(SCALE);                                //add the SCALE label
132
133       NDVISlider.addChangeListener(new ChangeListener(){ //Add action function to Slider
134         public void stateChanged(ChangeEvent e){
135           JSlider s=(JSlider)e.getSource();
```

```
136           float value=(float)((s.getValue()-100.0)/100.0); //calculate the NDVI theshold value from
    slider position
137           NDVI.setText("NDVI Theshold = " +value);
138         }
139       });
140       ScaleSlider.addChangeListener(new ChangeListener(){// Add action function to update NDVI value
141         public void stateChanged(ChangeEvent  e){
142           JSlider s=(JSlider)e.getSource();
143           float value=(float)(s.getValue()/4.0);          // calculate the scale factor from slider
    position
144           SCALE.setText("SCALE Factor = " +value);
145         }
146       });
147
148   }
149     // *** sub function called by constructor to create Geographical refrence points ***//
150     private void createGeoPanel(){
151       double LATITUDE0=PL_Geo.getDegreeValue("324532.10");  // latitude of point 0
152       double LONGITUDE0=PL_Geo.getDegreeValue("1171206.42");// longitutde of point 0
153       double LATITUDE1=PL_Geo.getDegreeValue("324552.01");  // latitude of point 1
154       double LONGITUDE1=PL_Geo.getDegreeValue("1171143.17");// longitutde of point 1
155       double LATITUDE2=PL_Geo.getDegreeValue("324530.59");  // latitude of point 2
156       double LONGITUDE2=PL_Geo.getDegreeValue("1171148.72");// longitutde of point 2
157
158       geoPanel.setLayout(new BorderLayout(5,5));      // set the properties of panel
159       geoPanel.add(showRef, BorderLayout.NORTH);
160       geoPanel.setPreferredSize(new Dimension(200, 300));
161
162       JPanel refP=new JPanel(new GridLayout(3, 1, 2, 2)); //set layout
163       JPanel ref0=new JPanel(new GridLayout(3, 1, 0, 0)); // set sub layout
164       JLabel ref0_L=new JLabel("Point0");                 // add label
165       JTextField ref0_T=new JTextField(""+LATITUDE0);     // put latitude value
166       JTextField ref0_TT=new JTextField(""+LONGITUDE0);   // put longitude value
167       ref0.add(ref0_L);ref0.add(ref0_T);                  // add components
168       ref0.add(ref0_TT);refP.add(ref0);
169
170       JPanel ref1=new JPanel(new GridLayout(3, 1, 0, 0)); // set sub layout
171       JLabel ref1_L=new JLabel("Point1");                 // add label
172       JTextField ref1_T=new JTextField(""+LATITUDE1);     // put latitude value
173       JTextField ref1_TT=new JTextField(""+LONGITUDE1);   // put longitude value
174       ref1.add(ref1_L);ref1.add(ref1_T);                  // add components
175       ref1.add(ref1_TT);refP.add(ref1);
176
177       JPanel ref2=new JPanel(new GridLayout(3, 1, 0, 0)); // set sub layout
178       JLabel ref2_L=new JLabel("Point2");                 // add label
179       JTextField ref2_T=new JTextField(""+LATITUDE2);     // put latitude value
180       JTextField ref2_TT=new JTextField(""+LONGITUDE2);   // put longitude value
181       ref2.add(ref2_L);ref2.add(ref2_T);                  // add components
182       ref2.add(ref2_TT);refP.add(ref2);
183
184       geoPanel.add(refP, BorderLayout.CENTER);        // add three panels to the main one
185     }
186   }
```

```
/**
 * <p>Title: PSerc Project</p>
 * <p>Description: Detect High Dangerous Trees from Satellite Images</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: ASU</p>
 * @author Yoshihiro Kobayashi
 * @version 1.0
 */
package pserc;
import java.awt.*;
import java.awt.event.*;
import javax.media.jai.JAI;
import javax.media.jai.RenderedOp;
import javax.media.jai.PlanarImage;
import java.awt.image.SampleModel;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;
import java.awt.geom.AffineTransform;
import java.awt.image.*;
import java.awt.geom.*;
import java.awt.image.renderable.*;
import javax.media.jai.*;
import java.util.*;
import java.io.*;

public class PL_OutputHistograph  extends PL_ImageCanvas {

  protected int viewerWidth  = 480, viewerHeight  = 400;               //Default viewer size.
  transient protected PlanarImage displayImage, origImage, demImage; //Images to display and orignal.
  protected int tileWidth = 256, tileHeight = 256;                     //Tile size (fixed).
  transient protected SampleModel  sampleModel;                        //SampleModel for generating new
image
  protected ColorModel  colorModel;                                    //ColorModel for generating new
image.
  protected int maxTileIndexX, maxTileIndexY;                          // maximum index of tile.
  protected int maxTileCordX, maxTileCordY;                            //maximum cordinate of tile.
  protected int minTileIndexX, minTileIndexY;                          //minimum index of tile.
  protected int minTileCordX, minTileCordY;                            //minimum cordinate of tile.
  protected int tileGridXOffset, tileGridYOffset;                      // tile offset.
  public static int imageWidth =0, imageHeight =0;                     //image size.
  protected TileCache tc;                                              //cache for tiled image.
  public static ROI Roi=null;                                          //Region of Interest (ROI)
object to collect the envelop area.
  public Vector shapes=new Vector();                                   //Store the Roi shapes for
drawing.
  public static PL_Geo geo=null;                                       //Geographical utility object.
  public Point tower0, tower1, tower2;                                 //tower locations in image
  public float towerZ0, towerZ1, towerZ2;                              // tower's height

  Raster demRaster;                                                    // raster of DEM data
  int lamda=1;                                                         // keep the current index of
regions
  int maxLamda=500;                                                    // maximum index of regions
  int[] tempT=new int[maxLamda];                                       // keep the indices for
segmentation
  Vector idV=new Vector();                                             // keep the index of regions
  Vector ipV=new Vector();                                             // region's initial point
  Point ref0, ref1, ref2;                                              // geographical reference points
in image
  Point mouseP=new Point(380, 360);                                    // mouse poition in image
  float[] xdem, ydem;                                                  // data for cross-section
vertically & horizontally
  Vector ndvi360 = new Vector();                                       // keep the healthy points in
cross section
  Vector ndvi380 = new Vector();                                       // keep the healthy points in
cross section


  public PL_OutputHistograph () {
    // set up the power tower positions
```

```
61        double lat0=PL_Geo.getDegreeValue("324532.10");  // default geogrphical reference pionts in the world
62        double lon0=PL_Geo.getDegreeValue("1171206.42");
63        double lat1=PL_Geo.getDegreeValue("324552.01");
64        double lon1=PL_Geo.getDegreeValue("1171143.17");
65        double lat2=PL_Geo.getDegreeValue("324530.59");
66        double lon2=PL_Geo.getDegreeValue("1171148.72");
67        int x0=46;int y0=154;                              // default geographical reference points in image
68        int x1=778;int y1=606;
69        int x2=95;int y2=607;
70        ref0 =new Point(x0, y0);                           // point objects for 3 reference points
71        ref1 =new Point(x1, y1);
72        ref2 =new Point(x2, y2);
73
74        geo = new PL_Geo(lat0, lon0, x0, y0, lat1, lon1, x1, y1, lat2, lon2, x2, y2); //Set geogrphcial information
75        // get the pixel position of towers from latitude and longitude in textfileds
76        tower0=PL_Geo.getPixelPos(Double.parseDouble(PL_OutputControlPanel.tower0_T.getText()),
77                                  Double.parseDouble(PL_OutputControlPanel.tower0_TT.getText()));
78        tower1=PL_Geo.getPixelPos(Double.parseDouble(PL_OutputControlPanel.tower1_T.getText()),
79                                  Double.parseDouble(PL_OutputControlPanel.tower1_TT.getText()));
80        tower2=PL_Geo.getPixelPos(Double.parseDouble(PL_OutputControlPanel.tower2_T.getText()),
81                                  Double.parseDouble(PL_OutputControlPanel.tower2_TT.getText()));
82        setImage();        // sub function to set DEM data
83        addEnvelop(50);    // sub function to set dangerous zone
84
85
86      this.addMouseListener(new MouseAdapter(){    // add mouse function to get the point for cross-section
87        public void mousePressed(MouseEvent e){
88          mouseP=new Point(e.getX(), e.getY());
89        }
90        public void mouseReleased(MouseEvent e){
91          repaint();
92        }
93      });
94
95
96    }
97    // This is a temporary test-function to get DEM data from text information
98    public void read380(){
99      String filename="data380.txt";          // horizontal DEM data file
100     String filename2="data2.txt";           // vertical DEM data file
101     xdem=new float[700]; ydem=new float[800];
102     try{       // Regular Read function of text file with numbers
103       FileReader reader=new FileReader(filename);
104       StreamTokenizer st = new StreamTokenizer(reader);
105       st.eolIsSignificant(true);     st.wordChars(' ','_');
106       int count=0;
107       while (true) {
108             st.nextToken();
109             if (st.ttype==StreamTokenizer.TT_EOF)
```

```
124              if (st.ttype==StreamTokenizer.TT_EOF) break;
125              else if(st.ttype==StreamTokenizer.TT_NUMBER){
126                 ydem[count]=(float)st.nval;
127                 count++;
128              }
129        }
130        reader.close();
131      }catch(IOException e){System.out.println("Reading DXF plines file failed" );}
132      normalizeDem();    // normalize the elevation data to reduce noisy values
133    }
134
135    // Sub function called when DEM data is created to reduce noisy values
136    private void normalizeDem(){
137      float[] newDem=new float[700]; // The vertical DEM has 700 point for test image
138      newDem[0]=xdem[0]; newDem[1]=xdem[1];     // create the temporary array for copying the contents
139      newDem[698]=xdem[698];newDem[699]=xdem[699];
140      for(int i=2; i<698; i++){       // gaussian filter is applied
141         newDem[i]=(float)((xdem[i-2]+xdem[i-1]+xdem[i]+xdem[i+1]+xdem[i+2])/5.0);
142      }
143      xdem=new float[700];xdem=newDem; // Swap the arrays
144
145      float[] newDem2=new float[800]; // The horizontal DEM has 800 points for test image
146      newDem2[0]=ydem[0];newDem2[1]=ydem[1];  // crate the temporary array for copying the contents
147      newDem2[798]=ydem[798];newDem2[799]=ydem[799];
148      for(int i=2; i<798; i++){       // gaussian fileter is applied
149         newDem2[i]=(float)((ydem[i-2]+ydem[i-1]+ydem[i]+ydem[i+1]+ydem[i+2])/5.0);
150      }
151      ydem=new float[800];ydem=newDem2; // swap the arrays
152    }
153    // Sub function called by Constructor to load images
154    public void setImage(){
155     origImage =JAI.create("fileload", "color4.tif");       // load color image for NDVI image
    processing
156     demImage= JAI.create("fileload", "floatpattern.tif"); // load DEM image for showing cross-section
157     panX =0; panY =0;                                      // Set the pan value as (0, 0)
158     atx = AffineTransform.getTranslateInstance(0.0, 0.0); // Set the default Translate materix
159     RenderedOp op = makeTiledImage(origImage);             // Call this makeTiledImage function for
    making tiled image
160     demRaster=demImage.getData();                          // set DEM raster data
161     displayImage = op.createInstance();           // create a display image copied from original image
162     sampleModel = displayImage.getSampleModel(); // assign the smaple model from the image
163     colorModel = displayImage.getColorModel();    // assign the color model of the image
164     getTileInfo(displayImage);                    // get tile information for display image
165     fireTilePropertyChange();                     // change the properties of tile
166     imageDrawn = false;                           // image drawn is set as false in default
167     getDEM();  // sub-function to set xdem[] and ydem[] data from DEM raster data
168    }
169    // sub-funciton called in setImage() to set cross-section data from DEM raster image
170    private void getDEM(){
171      xdem=new float[700]; ydem=new float[800];    // initialize the data
172      for(int i=0; i<700; i++) xdem[i]=(float) (40.0*demRaster.getSampleFloat(mouseP.x, i, 0)); // get
    vertical cross-section
173      for(int i=0; i<800; i++) ydem[i]=(float) (40.0*demRaster.getSampleFloat(i, mouseP.y, 0)); // get
    horizontal cross-section
174
175      towerZ0=(float) (40.0*demRaster.getSampleFloat(tower0.x, tower0.y, 0)); // z-value of tower 0
176      towerZ1=(float) (40.0*demRaster.getSampleFloat(tower1.x, tower1.y, 0)); // z-value of tower 1
177      towerZ2=(float) (40.0*demRaster.getSampleFloat(tower2.x, tower2.y, 0)); // z-value of tower 2
178      normalizeDem();     // call sub fucntion to reduce noises
179
180    }
181    // Change the properties of tiled image
182    protected void fireTilePropertyChange() {
183     firePropertyChange("maxTileIndexX", null, new Integer(maxTileIndexX));
184     firePropertyChange("maxTileIndexY", null, new Integer(maxTileIndexY));
185     firePropertyChange("tileWidth", null, new Integer(tileWidth));
186     firePropertyChange("tileHeight", null, new Integer(tileWidth));
187     firePropertyChange("transform", null, atx);
188    }
189    // utility funciton to get the information of tiled image
```

```
190     protected void getTileInfo(PlanarImage img) {
191       imageWidth = img.getWidth();        // image width
192       imageHeight = img.getHeight();      // image height
193       tileWidth = img.getTileWidth();     // tile width
194       tileHeight = img.getTileHeight();   // tile height
195       maxTileIndexX = img.getMinTileX() + img.getNumXTiles() - 1; // maximum index of tile
196       maxTileIndexY = img.getMinTileY() + img.getNumYTiles() - 1;
197       maxTileCordX = img.getMaxX();       // maximum tile position
198       maxTileCordY = img.getMaxY();
199       minTileIndexX = img.getMinTileX(); // minimum index of tile
200       minTileIndexY = img.getMinTileY();
201       minTileCordX = img.getMinX();       // minimum tile position
202       minTileCordY = img.getMinY();
203       tileGridXOffset = img.getTileGridXOffset(); // offset of tile
204       tileGridYOffset = img.getTileGridYOffset();
205     }
206     // utility function to get display image
207      public PlanarImage getDisplayImage(){return displayImage;}
208
209      // sub-function to create a tiled image from an big one image
210      protected RenderedOp makeTiledImage(PlanarImage img) {
211      ImageLayout tileLayout = new ImageLayout(img);    // set the layout
212      tileLayout.setTileWidth(tileWidth);               // set tile width and height
213      tileLayout.setTileHeight(tileHeight);
214      RenderingHints tileHints = new RenderingHints(JAI.KEY_IMAGE_LAYOUT, tileLayout);
215      ParameterBlock pb = new ParameterBlock();
216      pb.addSource(img);
217      RenderedOp op=JAI.create("format", pb, tileHints); //generate RenderedOp object with tile info
218      return JAI.create("BandSelect", (PlanarImage)op, new int[] {0,1,2,3});
219     }
220
221     // sub function called by createRegion8() function to create a polygon from pixel-region
222     // Fij[] = indices of regions. (_x,_y)=staring point, (_ti,_tj)= tile indices
223     public void getSegPolygon(int[] Fij, int _x, int _y, int _ti, int _tj){
224         Polygon p=new Polygon();     // output polygon for each region
225         int _w=256;                  // tiled image has 256x256 pixels
226         int j=_x; int i=_y;          // (j, i) is position of starting
227         int ID=Fij[_y*_w+_x];        // index of region in Fij[]
228         boolean comeback=false;      // used to check if the searching finished
229         int     direction = 0;       // direction of searching: 0 (right), 1(up), 2(left), 3(down)
230         int count=0;                 // keep the counts to avoid infinite loop
231
232         while(!comeback && count<100){                    // X X X  X    X   X X X
233           count++;                                        // X X X (LT) (RT) X X X
234           int LT = 0;int RT = 0;int LB = 0; int RB = 0;   // X X X (LB) (RB) X X X
235           // get the values at the position of LT(left top), RT (right top), LB (left bottom), RB
    (right bottom)
236           if (j - 1 >= 0 && i - 1 >= 0) LT = Fij[ (i - 1) * _w + j - 1];
237           if (j <= _w && i - 1 >= 0) RT = Fij[ (i - 1) * _w + j];
238           if (j - 1 >= 0 && i < _w) LB = Fij[i * _w + j - 1];
239           if (j <= _w && i < _w) RB = Fij[i * _w + j];
240
241           if (LT != ID && RT ==ID && LB !=ID && RB !=ID ) { //case 1: go right
242             p.addPoint(_ti*_w+j, _tj*_w+i);    // X O
243             direction=0;                       // X X
244             j++;
245           }
246           else if (LT != ID && RT != ID && LB != ID && RB == ID) { //case 2: go down
247             p.addPoint(_ti*_w+j, _tj*_w+i);    // X X
248             direction=3;                       // X O
249             i++;
250           }
251           else if (LT != ID && RT == ID && LB != ID && RB == ID) { //case 3: go down
252             direction = 3;                     // X O
253             i++;                               // X O
254           }
255           else if (LT != ID && RT != ID && LB == ID && RB != ID) { // case 4: go left
256             p.addPoint(_ti*_w+j, _tj*_w+i);    // X X
257             direction = 2;                     // O X
```

```
259          }
260          else if(LT!=ID && RT==ID && LB==ID && RB!=ID){// case 5 : go right if the pre-direction is up
261                 if(direction == 1){                    // X O
262                     p.addPoint(_ti*_w+j, _tj*_w+i);    // O X
263                     direction=0;
264                     j++;
265                 }
266                 else if(direction == 3){                        // go left if the pre-direciton is
     left
267                     p.addPoint(_ti*_w+j, _tj*_w+i);
268                     direction=2;
269                     j--;
270                 }
271                 else{System.out.println("*** GetLInkList has case 5 with error." );}
272             }
273
274          else if (LT != ID && RT != ID && LB == ID && RB == ID) { //case 6: go left
275            direction = 2;      // X X
276            j--;                // O O
277          }
278          else if (LT != ID && RT == ID && LB == ID && RB == ID) { //case 7: go left
279            p.addPoint(_ti * _w + j, _tj * _w + i);
280            direction = 2;     // X O
281            j--;               // O O
282          }
283          else if (LT == ID && RT != ID && LB != ID && RB != ID) { //case 8 : go top
284            p.addPoint(_ti * _w + j, _tj * _w + i);
285            direction = 1;     // O X
286            i--;               // X X
287          }
288          else if (LT == ID && RT == ID && LB != ID && RB != ID) { //case 9 : go right
289            direction = 0;     // O O
290            j++;               // X X
291          }
292          else if (LT == ID && RT != ID && LB != ID && RB == ID) { //case 10 : go down if pre is right
293            if (direction == 0) {                          // O X
294              p.addPoint(_ti * _w + j, _tj * _w + i);  // X O
295              direction = 3;
296              i++;
297            }
298            else if (direction == 2) {                              // go top if pre is left
299              p.addPoint(_ti * _w + j, _tj * _w + i);
300              direction = 1;
301              i--;
302            }
303            else {
304              System.out.println("*** GetLInkList has case 5 with error." );
305            }
306          }
307          else if (LT == ID && RT == ID && LB != ID && RB == ID) { //case 11 : go down
308            p.addPoint(_ti * _w + j, _tj * _w + i);
309            direction = 3;        // O O
310            i++;                  // X O
311          }
312          else if (LT == ID && RT != ID && LB == ID && RB != ID) { //case 12: go up
313            direction = 1;       // O X
314            i--;                 // O X
315          }
316          else if (LT == ID && RT == ID && LB == ID && RB != ID) { //case 13; go right
317            p.addPoint(_ti * _w + j, _tj * _w + i);
318            direction = 0;      // O O
319            j++;                // O X
320          }
321          else if (LT == ID && RT != ID && LB == ID && RB == ID) { //case 14: go up
322            p.addPoint(_ti * _w + j, _tj * _w + i);
323            direction = 1;    // O X
324            i--;              // O O
325          }
326          if (i == _y && j == _x) comeback = true; // check if the searching is back to the original
     position
```

```
327       }// end of while-loop
328       if(p.npoints>1) {  // If there are points more than 1, then add the boundary box to ipV vector.
329         Rectangle  r=(Rectangle)p.getBounds();
330         ipV.addElement(r);
331       }
332     }
333     // Paint funciton
334     public void paintComponent(Graphics gc){
335       Graphics2D g = (Graphics2D)gc;            // Convert from Graphcs to Graphcs2D
336       Rectangle rect = this.getBounds();        // get the boundary rectangle of image
337       if((viewerWidth != rect.width) || (viewerHeight != rect.height)){
338         viewerWidth = rect.width;viewerHeight = rect.height;
339       }
340       g.setColor(Color.black);
341       g.fillRect(0, 0, viewerWidth, viewerHeight); // fill the rectngle in Black
342       if(displayImage == null) return;          // if it is painted, then return
343       int ti =0, tj = 0;                        // tile index of X and Y directions
344       Rectangle bounds = new Rectangle(0, 0, rect.width, rect.height);
345       bounds.translate(-panX, -panY);           // move the boundary to pan position.
346
347       // Get the visilble tile indices of left top, right top, left bottom, and right bottom.
348       int leftIndex = displayImage.XToTileX(bounds.x);
349       if(leftIndex < minTileIndexX) leftIndex = minTileIndexX;
350       if(leftIndex > maxTileIndexX) leftIndex = maxTileIndexX;
351       int rightIndex = displayImage.XToTileX(bounds.x + bounds.width - 1);
352       if(rightIndex < minTileIndexX) rightIndex = minTileIndexX;
353       if(rightIndex > maxTileIndexX) rightIndex = maxTileIndexX;
354       int topIndex = displayImage.YToTileY(bounds.y);
355       if(topIndex < minTileIndexY) topIndex = minTileIndexY;
356       if(topIndex > maxTileIndexY) topIndex = maxTileIndexY;
357       int bottomIndex = displayImage.YToTileY(bounds.y + bounds.height - 1);
358       if(bottomIndex < minTileIndexY) bottomIndex = minTileIndexY;
359       if(bottomIndex > maxTileIndexY) bottomIndex = maxTileIndexY;
360
361       // Step1: ** Update and redraw each tile **
362       idV=new Vector(); ipV=new Vector();          // initialize the vector objects
363       ndvi380=new Vector(); ndvi360=new Vector();
364       getDEM();      // call sub-function to reset cross section data
365       lamda=0;       // initialize the index of current healthy vegetation region
366       int slideValue=PL_OutputControlPanel.NDVISlider.getValue();  // get the NDVI slide value
367       int towerLength=(int)(PL_OutputControlPanel.ScaleSlider.getValue()/4.0); // get the Scale factor
368
369       for(tj = topIndex; tj <= bottomIndex; tj++) {      // loop for each tile
370         for (ti = leftIndex; ti <= rightIndex; ti++) {
371
372           Raster raster = displayImage.getTile(ti, tj);  // Get the raster information
373           int width2 = raster.getWidth();                // get raster's width
374           int height2 = raster.getHeight();              // get raster's height
375           int bands = raster.getNumBands();              // get raster's band number (Supposed 4)
376           int[] pixels = new int[width2 * height2 * bands]; // pixel information of tile image
377           byte[] data = new byte[width2 * height2 * 3];  // target pixel data
378           int[] Fij=new int[width2* height2];            // region ID data
379           tempT[0]=0; tempT[1]=0;                        // tempT is used for segmentation of selected
    pixels
380           raster.getPixels(ti * 256, tj * 256, width2, height2, pixels); // 256 x 256 is the tile size
    FIXED
381
382           int index = 0;
383           for (int h = 0; h < height2; h++)              // For each pixel in a tile
384             for (int w = 0; w < width2; w++) {
385
386               int red = pixels[index * 4 + 0];     // Get Red band pixel info
387               int inf = pixels[index * 4 + 3];     // Get Infrared band pixel info
388               float ndvi = (float) ( ( (float) (inf - red)) / ( (float) (red + inf))); // Cacluate NDVI
389               byte ndviByte=(byte)( (ndvi + 1.0) * 100.0);  // convet NDVI rane from -1.0~1.0 to 0~200
390               data[index * 3 + 2] = (byte) ndviByte;                    //set the NDVI in red band
391               data[index * 3 + 1] = (byte) pixels[index * 4 + 1];       //Set the original Green in
    green band
392               data[index * 3 + 0] = (byte) pixels[index * 4 + 2];       //Set the original Blue in blue
    band
```

```
393             int thres=slideValue;              // get slide value defined in PL_OutputControlPanel
394
395             // ** Get Healthy Pixels
396             int xIndex = displayImage.tileXToX(ti)+w;    // get the real position in a big origianl
    image
397             int yIndex = displayImage.tileYToY(tj)+h;    // get the real position in a big origianl
    image
398             if(ndviByte>thres && Roi.contains(xIndex, yIndex)){
399               Fij[index]=1;                              // If the pixel is inside ROI and more than
    NDVI theshold
400             }
401             index++;
402         } // end for all pixels in a tile
403
404         // cut less than 4 pixels
405         int[] Fij2=new int[256*256];  // temporal int[] to convert the dat
406         index=0;
407         for (int h = 0; h < height2; h++)
408           for (int w = 0; w < width2; w++) {
409             if(Fij[index]>0 &&  // If the NDVI value is more than Thesdhold, the pixel is WIHTE.
410              (w+1)<width2 && (w-1)>=0 && (h+1)<height2 && (h-1)>=0 &&
411                   Fij[(h+1)*width2+w]>0 &&
412                   Fij[h*width2+w-1]>0 && Fij[h*width2+w+1]>0 &&
413                   Fij[(h-1)*width2+w]>0){
414               data[index * 3 + 2] = (byte) 255; //red
415               data[index * 3 + 1] = (byte) 255; //green
416               data[index * 3 + 0] = (byte) 255; //blue
417               Fij2[index]=1;
418               if(ti*256+w==mouseP.x){  // add the healthy pixel position of vertical cross-section
419                 int ypos=tj*256+h;
420                 ndvi380.add(new Integer(ypos));
421               }
422               if(tj*256+h==mouseP.y){ // add the healthy pixel position of horizontal cross-section
423                 int xpos=ti*256+w;
424                 ndvi360.add(new Integer(xpos));
425               }
426             }
427             index++;
428           }
429
430         Fij=new int[256*256];
431         for(int i=0; i<256*256; i++) Fij[i]=Fij2[i];  // swap the Fij and Fij2
432         createRegion8(Fij, width2, height2, ti, tj);  // *** Segmentation to put indices for each
    region
433
434         // create a data buffer from data[] and draw the new tile image at the proper position
435         DataBufferByte dbuffer = new DataBufferByte(data, width2 * height2 * 3);
436         sampleModel = RasterFactory.createPixelInterleavedSampleModel(DataBuffer.TYPE_BYTE, width2,
    height2, 3);
437         colorModel = PlanarImage.createColorModel(sampleModel);
438         WritableRaster wr = RasterFactory.createWritableRaster(sampleModel,dbuffer, new Point(0, 0));
439         BufferedImage bi = new BufferedImage(colorModel, wr, colorModel.isAlphaPremultiplied(), null);
440         int xInTile = displayImage.tileXToX(ti); // get X tile indices
441         int yInTile = displayImage.tileYToY(tj); // get Y tile indices
442         AffineTransform tx = AffineTransform.getTranslateInstance(xInTile +panX, yInTile + panY);
443         g.drawRenderedImage(bi, tx);
444       }
445     }
446     imageDrawn = true; // imageDrawn is set as true to avoid redraw
447
448     // Step2: *** Draw ROI and cross-section lines***
449     if (Roi == null) return;
450     AffineTransform tx = AffineTransform.getTranslateInstance(panX, panY);
451     Shape shape = tx.createTransformedShape(Roi.getAsShape());
452     g.setStroke(new BasicStroke(2));       //Set the stroke size as 2
453     g.setPaint(new Color(255, 255, 255)); //Set as White
454     g.draw(shape);                         // draw ROI
455     g.setColor(Color.red);                 // set as Red
456     g.drawLine(0, mouseP.y, 800, mouseP.y); // draw horizontal cross-section line
457     g.drawLine(mouseP.x, 0, mouseP.x, 700); // draw vertical cross-section line
```

```
458
459       // Step3: *** Draw Cross-section ground***
460       for(int i=0; i<700; i++){
461         if(!Roi.contains(mouseP.x, i)) g.setColor(Color.gray);     // out of ROI
462         else if (ndvi380.contains(new Integer(i))) g.setColor(Color.white);  // healthy NDVI
463         else g.setColor(Color.lightGray);                          // inside ROI but not healthy
464         g.drawLine(800, i, 800 + (int)(xdem[i])   , i);            // draw a section line
465       }
466       for(int i=0; i<800; i++){
467         if(!Roi.contains(i, mouseP.y)) g.setColor(Color.gray);     // out of ROI
468         else if (ndvi360.contains(new Integer(i))) g.setColor(Color.white); // healthy NDVI
469         else g.setColor(Color.lightGray);                          // inside ROI but not healthy
470         g.drawLine(i, 800, i,  800 - (int)ydem[i]);                // draw an horizontal section
471       }
472
473       // Step4: *** Transmission lines and towers***
474       g.setColor(Color.red);
475       g.setStroke(new BasicStroke(2));
476       g.drawLine(tower0.x, 800-(int)towerZ0 , tower0.x, 800-(int)towerZ0-towerLength); // draw tower 0
477       g.drawLine(tower1.x, 800-(int)towerZ1 , tower1.x, 800-(int)towerZ1-towerLength); // draw tower 1
478       g.drawLine(tower2.x, 800-(int)towerZ2 , tower2.x, 800-(int)towerZ2-towerLength); // draw tower 2
479       g.setColor(Color.white);
480       g.setStroke(new BasicStroke(1));
481       g.drawLine(tower0.x, 800-(int)towerZ0-towerLength , tower1.x, 800-(int)towerZ1-towerLength); //
    draw line between towe0 and 1
482       g.drawLine(tower1.x, 800-(int)towerZ1-towerLength , tower2.x, 800-(int)towerZ2-towerLength); //
    draw line between tower1 and2
483       if(PL_OutputControlPanel.showTower.isSelected()){
484          g.setColor(Color.red);                    //set color as Red
485          g.setStroke(new BasicStroke(2));
486          g.fillRect(tower0.x - 5, tower0.y - 5, 10, 10); // draw tower 0 as 5x5 box
487          g.drawString("Tower0", tower0.x, tower0.y);
488          g.fillRect(tower1.x - 5, tower1.y - 5, 10, 10);  // draw tower 1 as 5x5 box
489          g.drawString("Tower1", tower1.x, tower1.y);
490          g.fillRect(tower2.x - 5, tower2.y - 5, 10, 10); // draw tower 2 as 5x5 box
491          g.drawString("Tower2", tower2.x, tower2.y);
492       }
493
494       // Step5: *** Draw boundary boxes of healthy trees ***
495       g.setColor(Color.yellow);              //set color as yellow
496       g.setStroke(new BasicStroke(1));       //Set the stroke size as 2
497       PL_OutputControlPanel.treeInfo.setText("ID \t"+"Latitude \t\t" +"Longitude \t\t" +"Distance to
    Power line \n");
498       for(int i=0; i<ipV.size(); i++){       // for each region (polygon)
499         Rectangle p=(Rectangle)ipV.elementAt(i);      // extract boundary box from ipV vector object
500         if(PL_OutputControlPanel.showTreeBox.isSelected())
501           g.drawRect(p.x, p.y, p.width, p.height);    // darw boundary boxes
502         if(PL_OutputControlPanel.showTreeID.isSelected())
503           g.drawString(""+i, p.x, p.y);               // draw index numbers
504
505         int tX=(p.x+p.width/2); // get center X of boundary box
506         int tY=(p.y+p.height/2);// get genter Y of boundary box
507         double[] pos=PL_Geo.getLAT(tX, tY);     // convert (x, y) to (latitude, longitude)
508         Point tempP =PL_Geo.getPixelPos(pos[0], pos[1]);
509
510         // **** Here need the formula to get the distance to powerline ** //
511         float demH=ydem[tempP.x];           // get z-value of tree position
512         float dis=towerZ0+towerLength-demH; // distanct to powerlines
513         PL_OutputControlPanel.treeInfo.append(i+"\t"+pos[0]+" \t"+pos[1]+" \t" +dis+ "\n"); //add
    textural info into textArea of PL_OutputControl Panel
514       }
515
516
517
518      // Step6: *** Draw Referece points ***
519       if(PL_OutputControlPanel.showRef.isSelected()){ // if checkbox is ON
520         g.setColor(Color.yellow);
521         g.fillOval(ref0.x-3, ref0.y-3, 6, 6);  // draw reference 0
522         g.drawString("Point0", ref0.x, ref0.y);
523         g.fillOval(ref1.x-3, ref1.y-3, 6, 6);  // draw reference 1
```

```
524           g.drawString("Point1", ref1.x, ref1.y);
525           g.fillOval(ref2.x-3, ref2.y-3, 6, 6);  // draw reference 2
526           g.drawString("Point2", ref2.x, ref2.y);
527         }
528
529     }
530
531     // sub function called by paintComponent function to segment the healthy pixels with unique
    indices
532     // Fij[] = 1 or 0, _w= image width, _h=image height, (_ti, _tj) index of tile
533     private void createRegion8(int[] Fij, int _w, int _h, int _ti, int _tj){
534
535        for(int i=0; i<_h; i++){ // for each pixel
536         for(int j=0; j<_w; j++){
537
538           if(Fij[i*_w+j]==0){}      // checking if 0 or 1
539           else {
540              int[] lp=new int[4];   // create int[4]
541              if(i-1<0){ lp[0]=0; lp[1]=0; lp[2]=0;} // if it is out of image
542              if(j-1<0){ lp[0]=0; lp[3]=0;}
543              if(j+1>_w){lp[2]=0;}
544              lp[0]=Fij[(i-1)*_w+(j-1)];     // X    X    X     X     X
545              lp[1]=Fij[(i-1)*_w+j];         // X lp[0] lp[1] lp[2]   X
546              lp[2]=Fij[(i-1)*_w+(j+1)];     // X lp[3]  (j,i)  X     X
547              lp[3]=Fij[i*_w+(j-1)];         // X    X    X     X     X
548              int L1=0; int L2=0;      // labels for search conditions
549
550              //*** Get the indices of L1 and L2 (L1<L2)
551              //except if L1=L2=0, if L2=0, keep the L1=ID, L2=0.
552              for(int p=0; p<4; p++){ //*** Keep the indices L1 (smallar ID) L2 (bigger ID)
553                  if(lp[p]!=0){
554                     if(L1==0){ L1=lp[p];} // L1 is changed from 0 to lp[p] for first hit
555                     else if(L2==0 && L1==lp[p]){}// for searching in the same ID area
556                     else if(L2==0 && L1>lp[p]){  // for second hit but order is wrong
557                          L2=L1;
558                          L1=lp[p];
559                     }
560                     else if(L2==0 && L1<lp[p]){L2=lp[p];} // for second hit
561                     else{// L2 !=0 && L1!=0
562                     }
563                  }
564              }
565              //*** Checking Condition of prePixels
566              if(L1==0 && L2==0){ // Fij[j,i] is the first point for a new region
567                  lamda++;
568                  if(lamda<maxLamda){   //lamda=0~499
569                     tempT[lamda]=lamda; // lamda = regionID is assigned to tempT and Fij
570                     Fij[i*_w+j]=lamda;
571                  }
572                  else{                 // in the case for updating the array space
573                     maxLamda+=500;
574                     int[] tempTT=new int[maxLamda];           // expand the array space
575                     System.arraycopy(tempT,0,tempTT,0,lamda); // copy the array contents
576                     tempTT[lamda]=lamda;
577                     tempT=tempTT;
578                     Fij[i*_w+j]=lamda;
579                  }
580              }
581              else if(L2==0&& L1!=0){ // searching an existing region
582                  Fij[i*_w+j]=L1;
583              }
584              else {                  // Two areas(L1!=0 && L2!=0) are existing in the lp[]
585                  Fij[i*_w+j]=L1;     // change the index of Fij from 1 to L1
586                  for(int r=0; r<=lamda; r++){
587                     if(tempT[r]==L2) tempT[r]=L1;
588                  }
589              }
590           }
591         }
592        }// end of all pixels
```

Appendix B: Elevations Along a 69 kV Sub transmission Power Line Right-of-Way in San Diego, CA (June 11, 2007, G. T. Heydt)

1. Introduction

This is a report on the elevation above mean sea level (MSL) along a power line right-of-way in San Diego, CA. This is a 69 kV line. The line is the crossing of Interstate 8 at Old Town in San Diego. Three structures support the circuit as shown in Figure B.1.

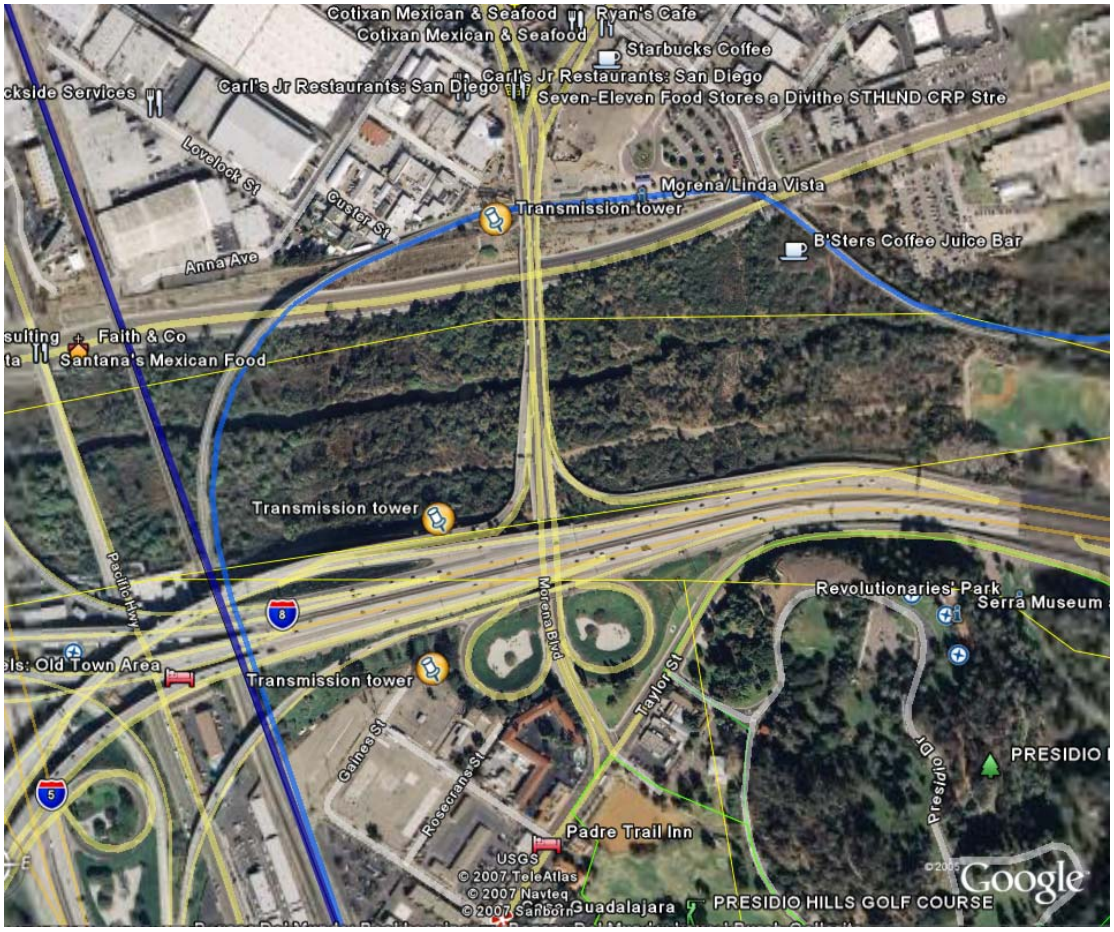


Figure B.1 69 kV transmission line in San Diego

The bases of the estimates of the elevation above MSL of the supporting structures are:

- A visit to the site including visual estimates
- The USGS topographical map "7.5 Minute Series: La Jolla, CA," dated 1996
- Google Earth

2. Estimates of surface elevation

The tidal difference in the Pacific Ocean off the coast of San Diego is about 4 feet. This gives an estimate of the location of the MSL. The San Diego River, which approximately bisects the stated right-of-way, does not have any tidal difference. The USGS map of the area shows the

ultimate river banks for the river as 10 feet MSL. It is estimated that the river itself is approximately 3.1 miles from the Pacific Ocean at the Morena Boulevard crossing. At one foot per mile river slope, the river surface is estimated as 3 feet above mean high tide or 7 feet MSL. This agrees (approximately) with the USGS map. If the center structure tower base is 3 feet above the river surface, this gives a surface elevation of 10 feet.

The 10 and 20 foot contours on the USGS map (made in 1996) give an indication of the surface elevation at the power line tower bases. These are estimated on the basis of map contours and visitation to the site and are summarized in Table B.1. Figure B.2 shows the location of several features.

Table B.1 Tower base elevations, estimated from several sources

	Tower base elevation (feet, MSL)	Comments
North tower	24	Tower is near to Morena Blvd. embankment. Clears street railway.
Center tower	10	South of San Diego River.
South tower	24	South tower is adjacent to on ramp for I-8 freeway which passes through the 10 foot contour on the USGS map.

3. Estimate of the right-of-way elevation

Figure B-3 shows an estimate of the surface elevation along the power line right-of-way.

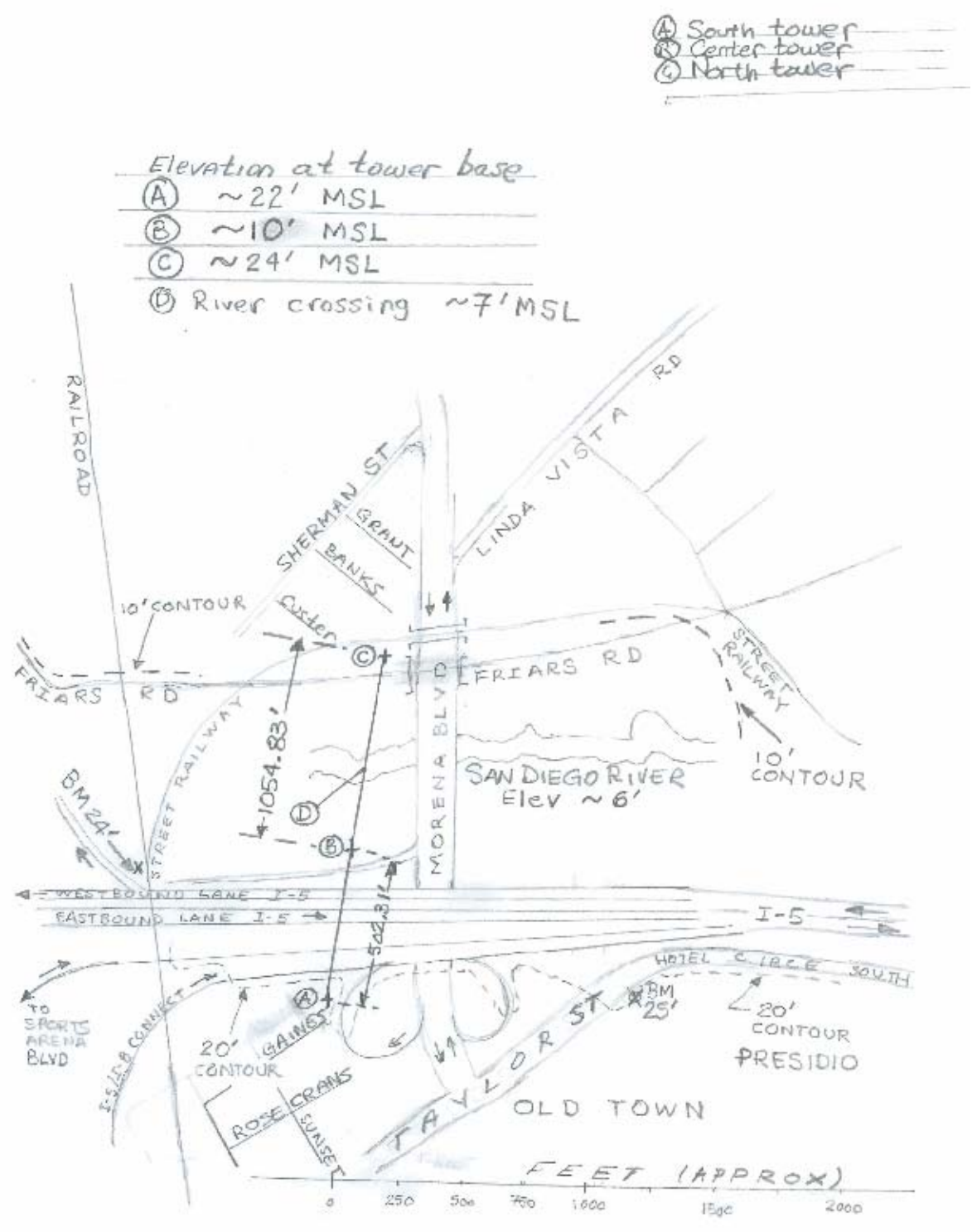


Figure B.2 Sketch of the right-of-way and adjacent features

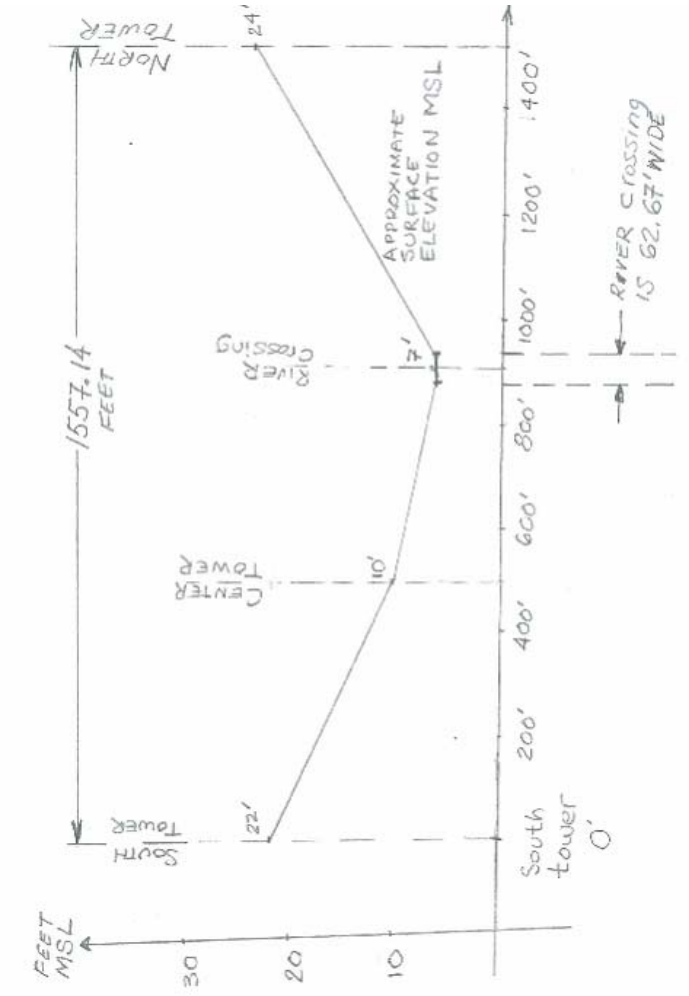


Figure B.3 Surface elevation along right-of-way