International School on LiDAR Technology

Laboratory Manual for LiDAR Data Processing

Bapna Ravish Biswas Susham
Ghosh Suddhasheel Y Surya Aditya

March 31 – April 4, 2008

GEOINFORMATICS DIVISION
DEPARTMENT OF CIVIL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
UP, India – 208 016

Foreword

This laboratory manual has been developed for International School on LiDAR Technology which is held from 31 March to 4 April 2008. One of the major aims of this school is to impart hands-on-training on working with LiDAR data. A duration of 12 hours has been assigned for data processing, which is spread over four days during the school. The laboratory is planned to be conducted at the Computer Centre of IIT Kanpur where each participant would be able to learn on his/her own.

The LiDAR data processing exercises have been designed around the TerraSolid software (Terrascan, Terramatch, Terramodeller and Terraphoto). This manual consists of detailed instructions for LiDAR data processing.

The instructions have been divided into four parts. The first part deals with importing raw LiDAR data and trajectory within Terrascan, creation of projects and different kinds of visualizations. In the second part, LiDAR data are corrected for the inherent errors using the overlap analysis. The corrected data are passed into the classification process which is covered in the third part of the manual. The use of routines and macros is shown to classify LiDAR data into ground points, low points, below surface points, building points etc. At this stage an orthophotograph is also employed to help in the classification process. Finally, the fourth part of laboratory manual shows how to generate vector models for buildings and produce contours.

This manual is a result of the hard work put by several individuals. In particular, I would like to thank the authors of this manual Mr. Suddhasheel Ghosh, Mr. Susham Biswas, Mr. Surya Aditya and Mr. Ravish Bapna. Further, the help provided by Mr. Hannu Korpela and Mr. Tom of TerraSolid is acknowledged. Mr. Manoj Jain helped with setting up of Bentley MS at IIT Kanpur. The data used in laboratory work are provided by Optech Inc. Canada. Prof. Phalguni Gupta, Head, Computer Centre IIT Kanpur permitted use of their facility while Mr. G. Tiwari helped in setting up the software etc. for the laboratory use.

Dr. Bharat Lohani
Associate Professor
Department of Civil Engineering
Indian Institute of Technology Kanpur
And
Coordinator, International School on LiDAR Technology

Contents

© 2008, Department of Civil Engineering, Indian Institute of Technology Kanpur. The contents and the instructions in this manual are protected by copyright.
To refer to this manual, use the following
IITK, 2008, Laboratory Manual for Lidar Data Processing, *International School for LiDAR Technology*, March 31 – April 4, 2008, IIT Kanpur

CONTENTS OF THE FOLDERS FOR LABORATORY

Niagara	• main folder of training data.
dgn	• Microstation Design files • *cut_boundary.dgn* dgn file used for data processing labs 1, 3 and *4* • *lidar_school.dgn* dgn file used in labs 1 and 2
images	• georectified image folder • contains one georectified image *resampled.ecw*
laser	• folder for saving laser blocks
laserraw	• raw laser data sets • *Str_1.las etc.* raw laser data sets in WGS84 with time stamps - total 8 files
macro	• folder for saving macros
mission	• *Niagara.ptc* - point class definitions of TerraScan
sbet	• trajectory information • *sbet_01.out* -text file of trajectories.
temp	• folder for temporary files
trajectory_tscan	• thinned trajectory files in Terrasolid format

LABORATORY NUMBER 1: BASIC TOOLS AND PROJECT CREATION

WHAT WOULD WE LEARN IN THIS LABORATORY

In this laboratory, we would learn to do the following

- Coordinate settings
- Importing trajectories
- Project creation
- Reading, visualization and measurement of data

COORDINATE SETTINGS

Firstly, we have to start **MicroStation** software. For this, go to *All Programs* > *Microstation* > *Microstation*.

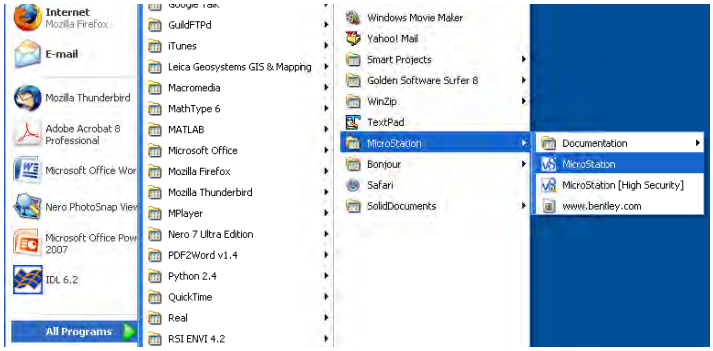


Figure 1: Starting MicroStation

Now browse the design file named **lidar_school.dgn**

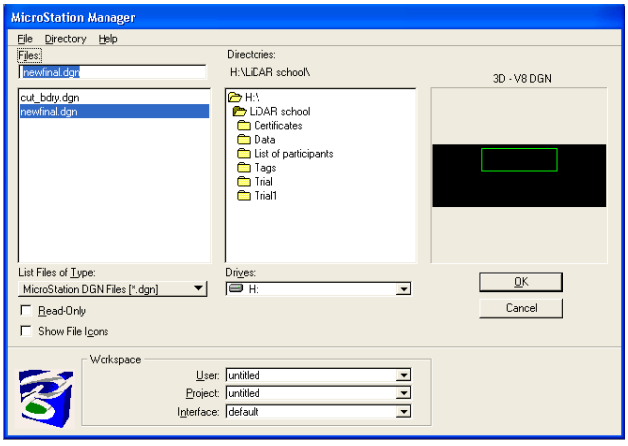


Figure 2: Opening the given design file

Click *OK*.

Click the **fit view**  button present at the bottom of a viewer. We will see a rectangle. This is the designed template.

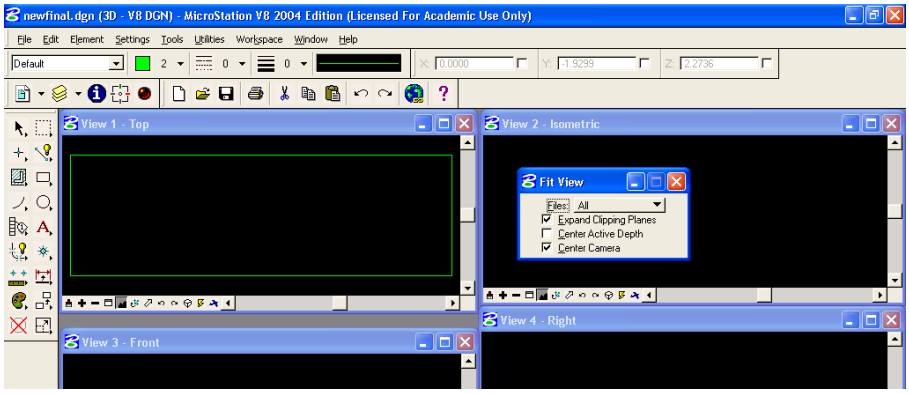


Figure 3: Displaying the rectangle

Now we have to start **TerraScan** module. For this, go to *Utilities* > *MDL applications* on the Microstation menu.

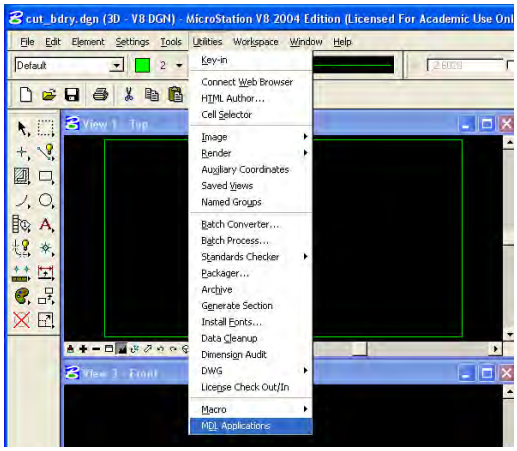


Figure 4: Opening the MDL Applications dialog box

Select **TSCAN** in **Available applications**.

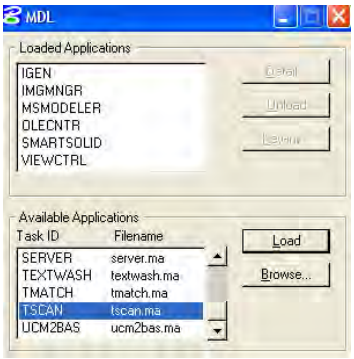


Figure 5: Launching TerraScan

Click *Load*. The **TerraScan** module gets loaded. On the workspace, we see a **TerraScan** menu bar.

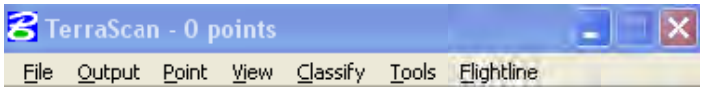


Figure 6: The *TerraScan* menu bar

Close the **MDL** dialog box.

For coordinate settings, go to *Settings* > *Coordinate transformations* > *Builtin projection systems* and check the **UTM WGS North** check box and enter **17** and **35** in Zones field.

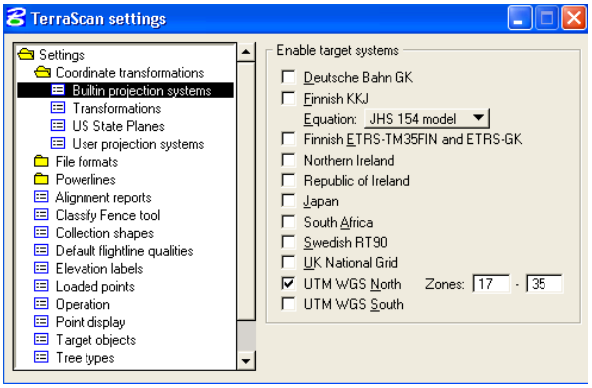


Figure 7: Coordinate Settings

Close **TerraScan Settings** window.

IMPORT TRAJECTORIES

Now we will import trajectories. Browse to **Manage Trajectories** icon.  . This can be accessed either through *Applications* > *TerraScan* > *General* or by clicking and dragging the first icon in the TerraScan toolbox.

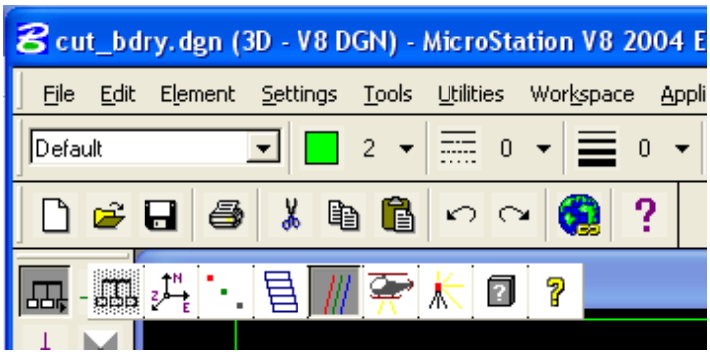


Figure 8: Opening the Trajectories window.

Goto *File* > *Set directory* to set directory for storing trajectory files.

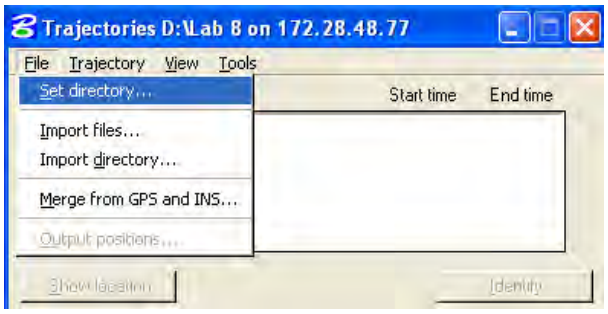


Figure 9: Setting the directory for Trajectories

Browse to a folder of your choice. To import trajectory files click *File* > *Import files* for importing trajectory files.

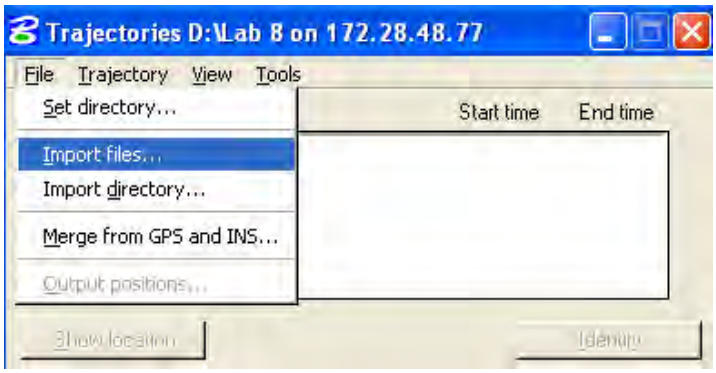


Figure 10: Importing the trajectory files

Select *sbet_01.out* and click *Add*. Click *Done*.

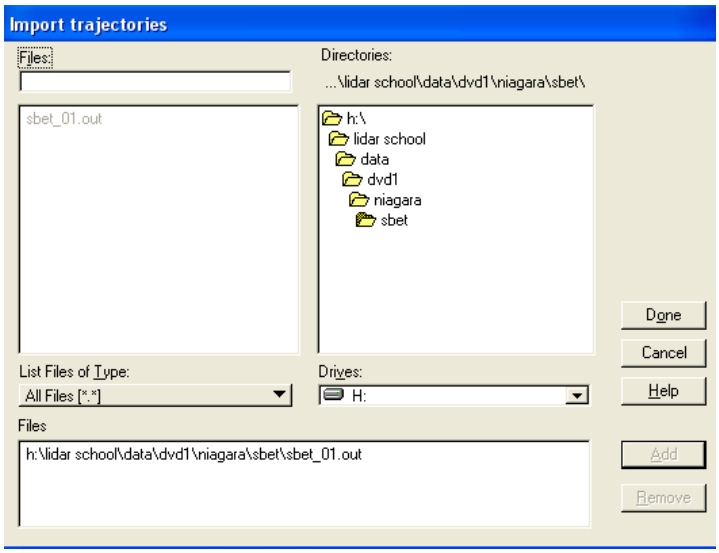


Figure 11: Importing trajectory files

An **Import Trajectories** window opens. Browse to **UTM-17N (81W)** in **WGS84** field. Check the *Thin positions* check box.

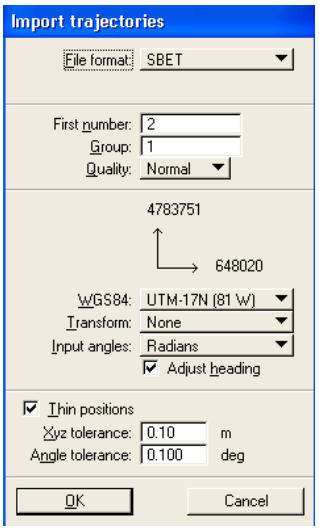


Figure 12: Setting the import parameters for trajectories

Click *OK*. Now select the trajectory file named **403542_409436.trj** in **Trajectory** window and click on *Show location* to view the trajectories.

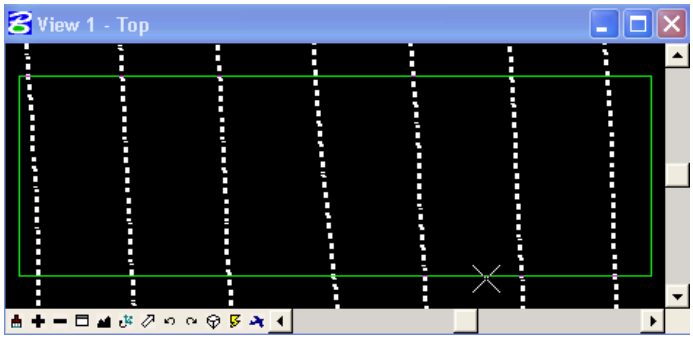


Figure 13: Displaying the location of trajectories

TRAJECTORIES

In the **Trajectories** window, go to *Tools* > *Draw into design*

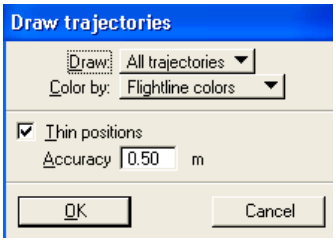


Figure 14: Drawing trajectories

Click OK. This will display the trajectories.

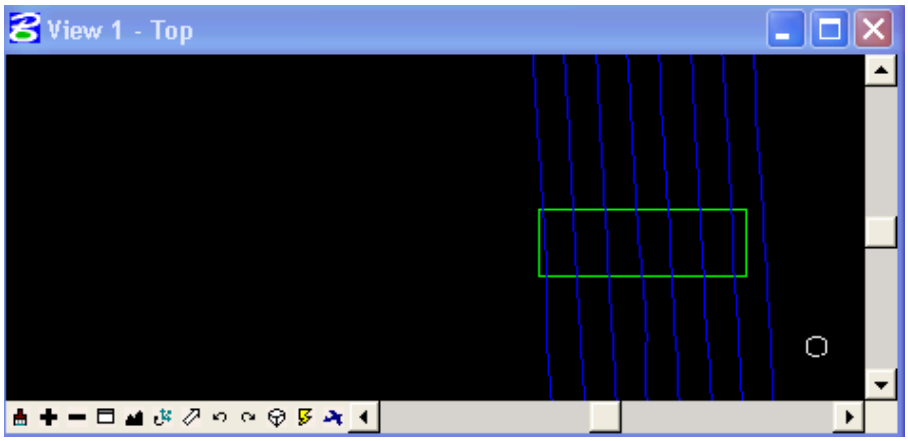


Figure 15: Drawn trajectories

Place a rectangle box so that we can trim the trajectory information.  This rectangle tool is available in the Microstation toolbar.

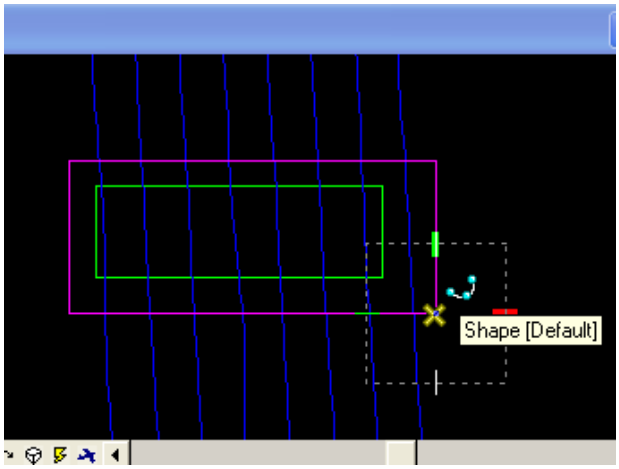


Figure 16: Creating a rectangle

Select the **polygon/rectangle**. In **Trajectories** window, goto *Tools* > *Delete outside polygon*

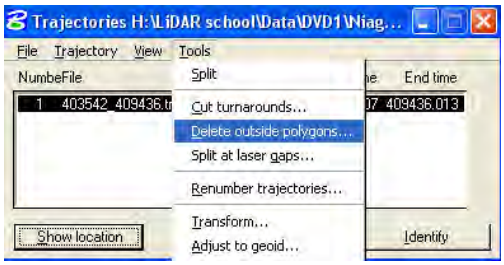


Figure 17: Deleting features outside the polygon

Click *OK*.

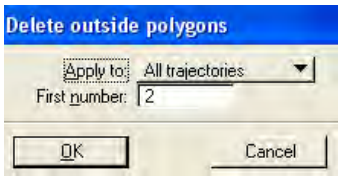


Figure 18: Parameters for deletion

Click OK in the **information** window. Now click on the trajectories and press **delete**.

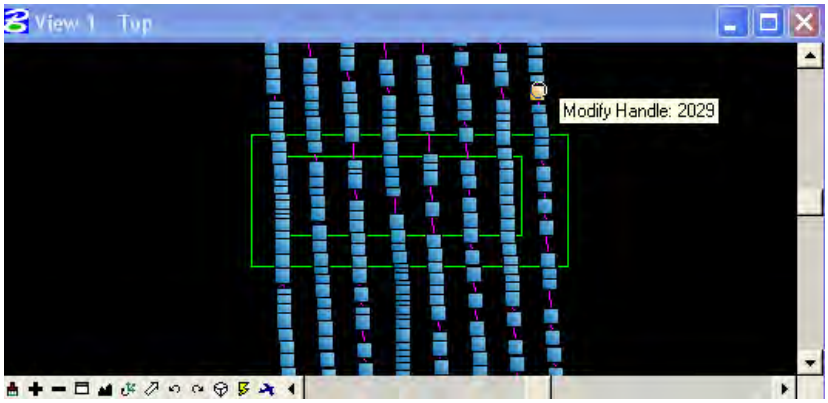


Figure 19: Deletion process

Select the outside polygon and press **delete** button.

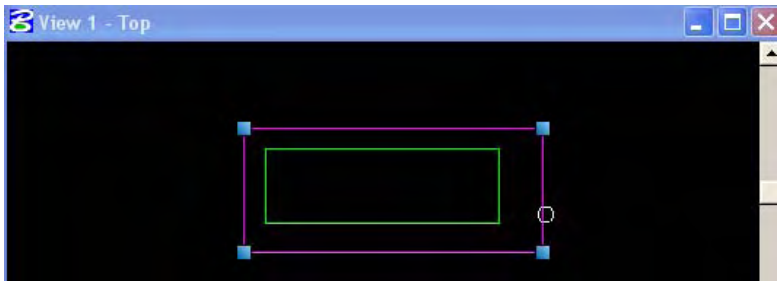


Figure 20: Deletion process outside the polygon

Goto *Tools* > *Draw into design.* Click OK in **Draw trajectories** window to view the trimmed trajectories.

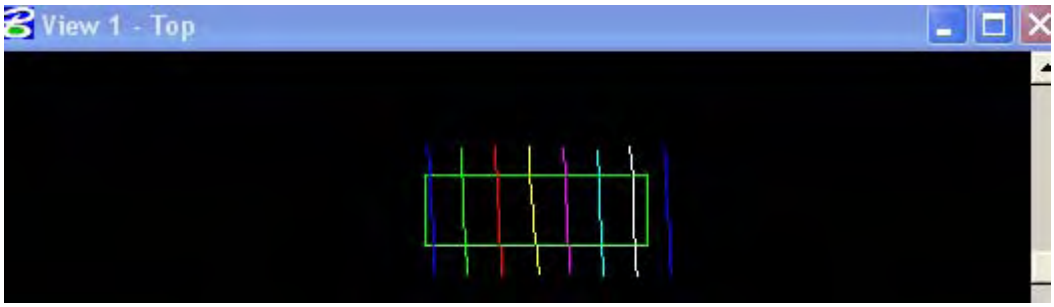


Figure 21: Trimmed trajectories

Press the **undo** button

Figure 22: Polygon

READING DATA

Now we will import data into **TerraScan**. Click *File* > *Read points* in **TerraScan.**

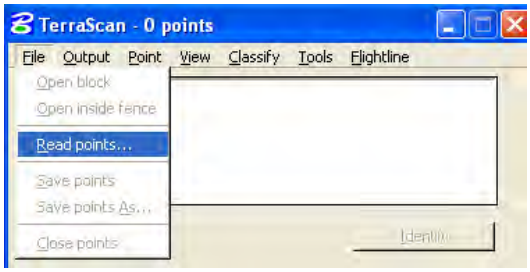


Figure 23: Initiating the process for reading data points

Browse to the folder containing the raw laser data and select all **LAS files**. Click *Add* and then *OK*.

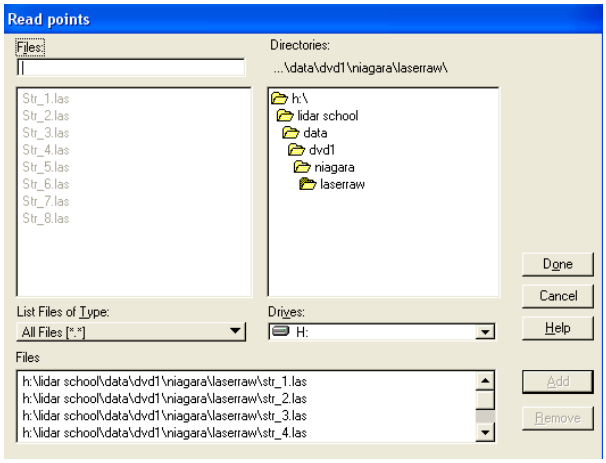


Figure 24: Selecting data point files

In the *Load Points* window, do not apply transformation in **WGS84** field. Also load every 10th point in the data and uncheck the remaining fields.[1]

[1] We load every 10th point from the given data so as to conserve memory for basic processing. Further, the total number of points from all the LAS files is too large.

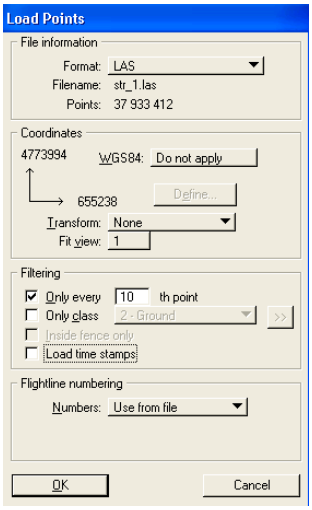


Figure 25: Setting parameters for reading data points

Click *OK*.

CREATING A PROJECT

Now we will create a project. Click **Define Project**  icon. Goto *File* > *New Project*

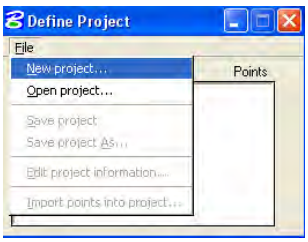


Figure 26: Defining a new project

In **Description**, enter Niagra. In **Storage** field, set the option to *LAS binary*. Check the *Load class list automatically* and select **niagara.ptc** from the given data.

Check the **load trajectories automatically** and browse to the folder containing the trajectory files. Click *OK*.

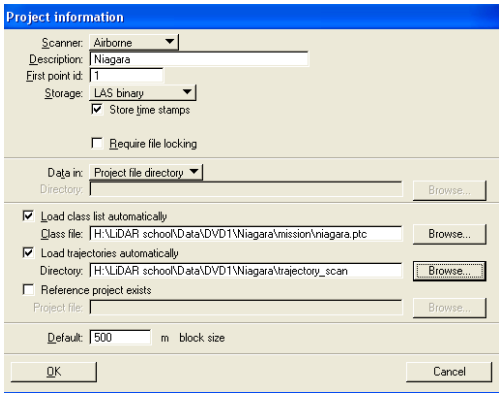


Figure 27: Setting project parameters and related information

For saving project, goto *File>Save Project As.*

Figure 28: Saving the project

Save the project to a desired folder where the file name is **Niagara**. Click **Save**. Now we will add block to the project. For this, select the boundary by clicking on it.

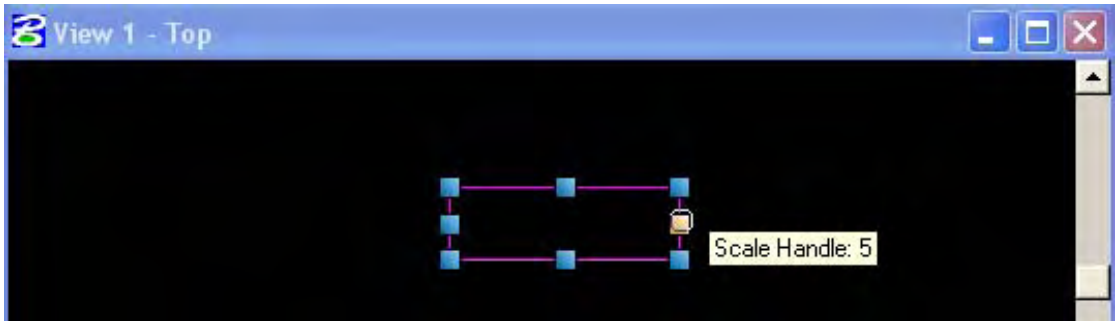


Figure 29: Selecting the boundary for importing data

Goto *Block* > *Add by boundaries* in **Project** window.

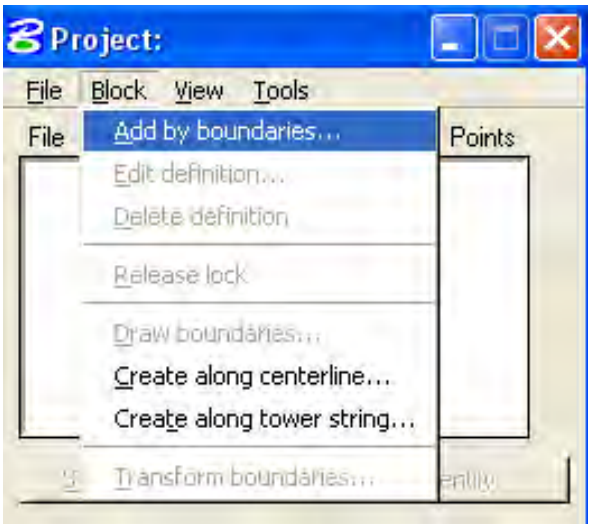


Figure 30: Adding the boundary to the project

In **File prefix** field, enter **Niagara_**

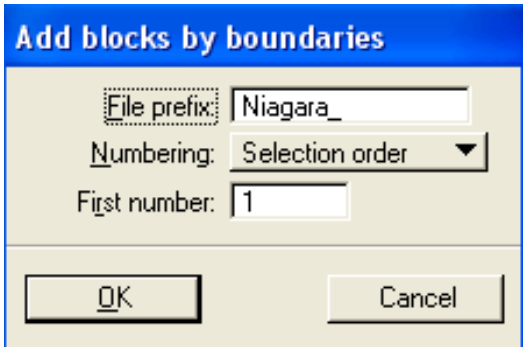


Figure 31: Adding information blocks by boundaries

Click OK. Click OK in the **Information** box. In the **Project** window, go to *File* > *Import points into project* for importing data points into project.

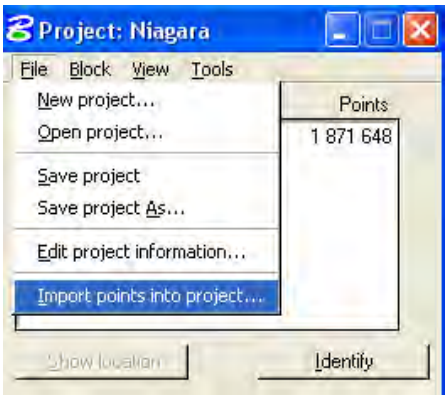


Figure 32: Importing data points into the project

Browse to the folder containing all the raw laser LAS files and select all the **LAS files**. Click *Add* and then click *Done*.

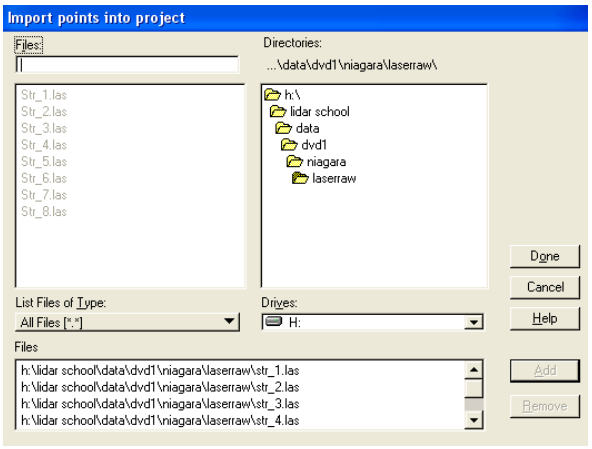


Figure 33: Importing data points – Selection of LAS files

Set **flight numbering** to *increase by file*. This is done to display data corresponding to flightlines.

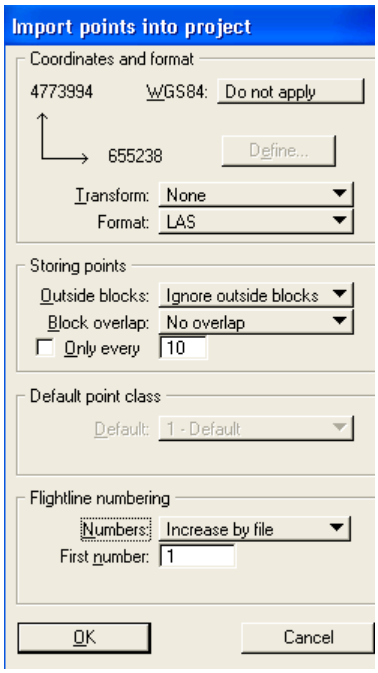


Figure 34: Parameters for importing the point data into the project

Click OK. A report is displayed.

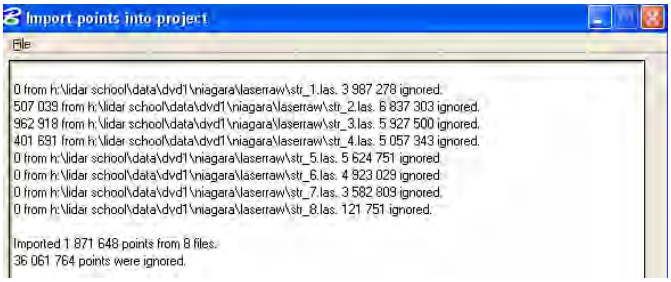


Figure 37: Report of importing data points into project

Close the report. Click *File* > *Open block* in **TerraScan** to view the data points in the project.

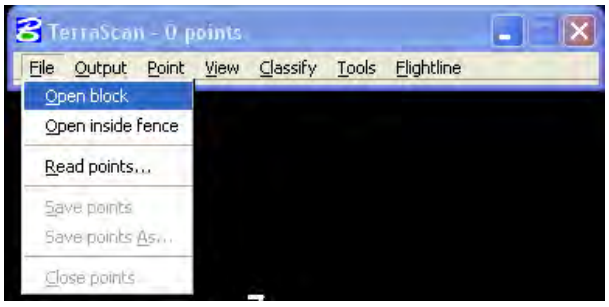


Figure 38: Opening block

Click inside the block to see the point cloud.

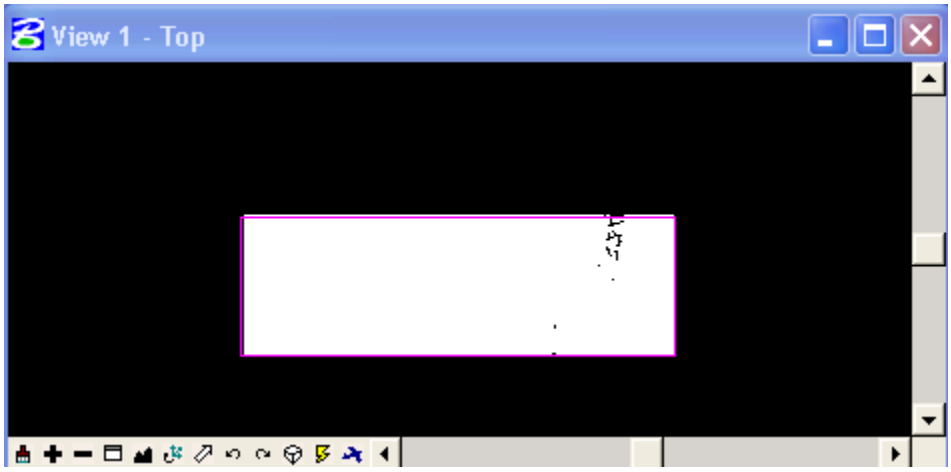


Figure 39: Displaying points into block

DATA DENSITY MEASUREMENT

Now we will measure point density of data. Click the **Measure Point Density** icon. 

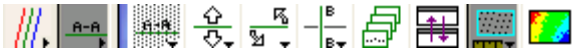


Figure 35: Density measurement icon

In **Measure Point Density** window, enter 1000 in **Width** field and select **Rectangle** in **Sample field**.

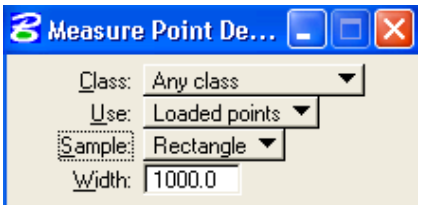


Figure 36: Setting density measurement parameters

Now just by mouse click in the data, you can see the point density at the bottom pane of **Microstation**.

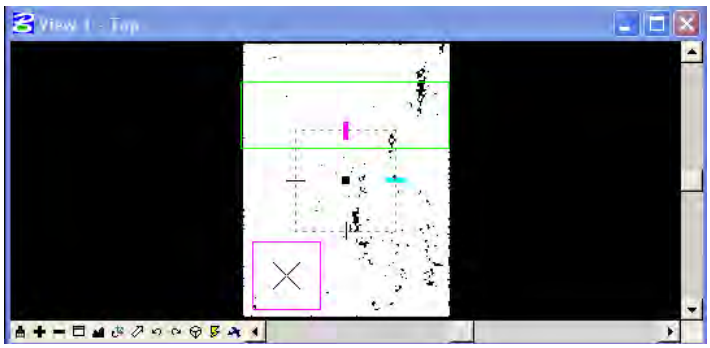


Figure 37: Measurement of point density

Right click the mouse to close the **Point Density measurement** window.

DATA VISUALIZATION

Now we will practice various modes of displaying data. Click *View* > *Display mode*. In **Color** field, select by **Elevation**. Click coloring scheme button.

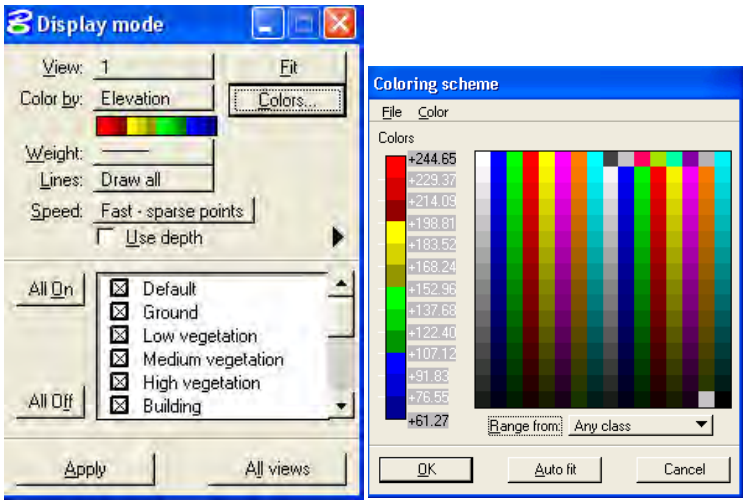


Figure 38: Coloring the point data by elevation

Click *Auto fit* button. Click *OK*. The point cloud is displayed with color scheme corresponding to elevation.

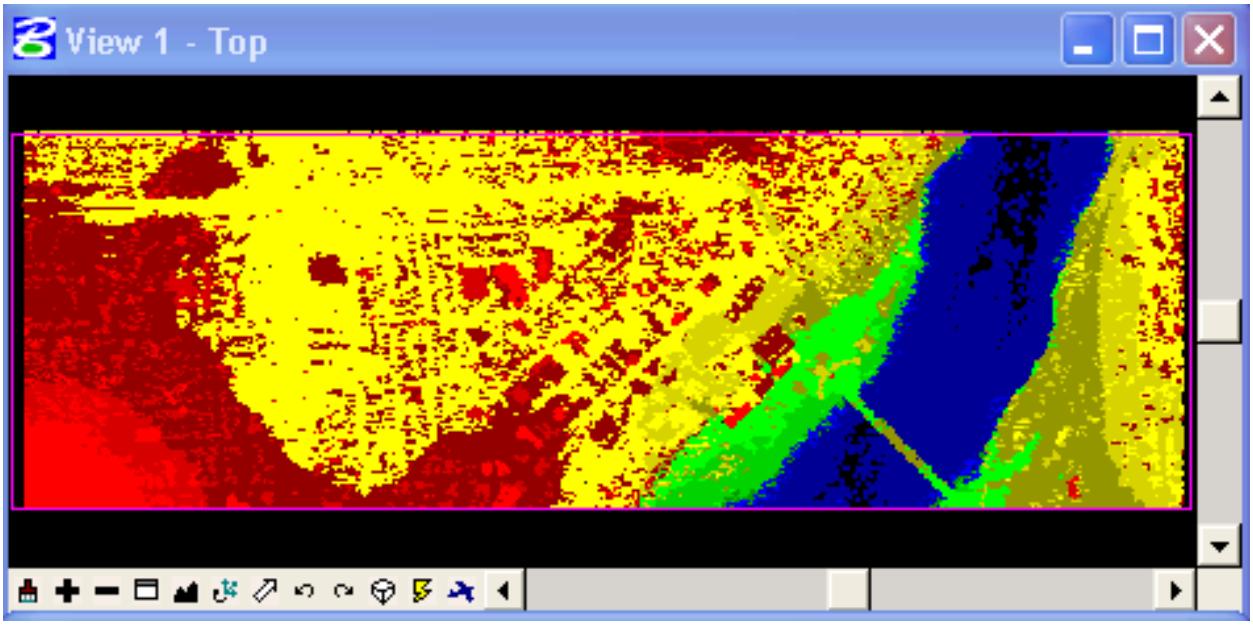


Figure 39: Points displayed and colored by elevation

Now display with color scheme corresponding to intensity

Figure 40: Data points displayed colored by intensity

Now display with color scheme corresponding to flight line.

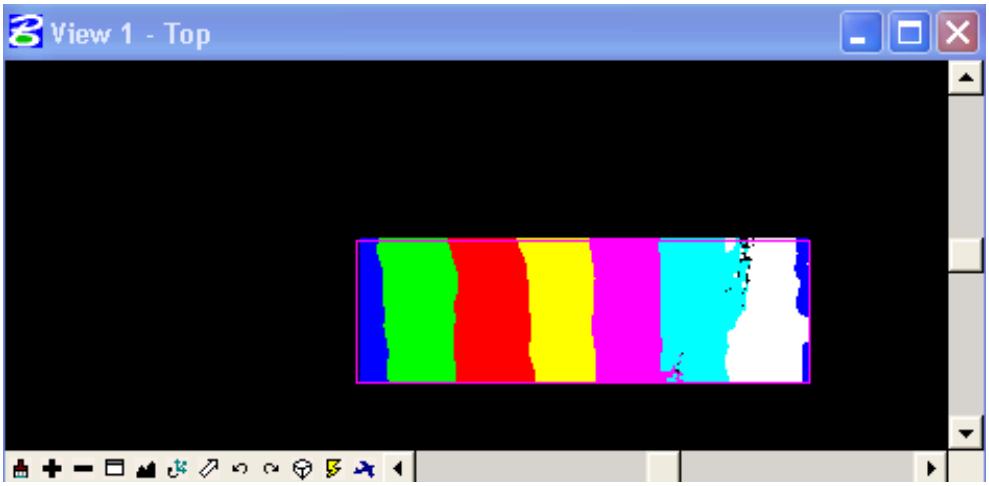


Figure 41: Points colored by flight line

We will now practice the display of various section functions. Goto **Draw section** 

. Draw a line and keep small width of section for display.

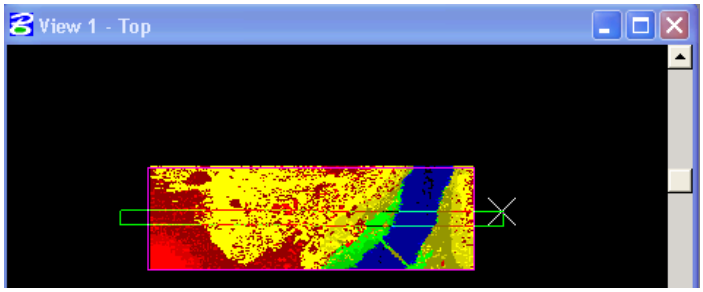


Figure 42: Drawing a section

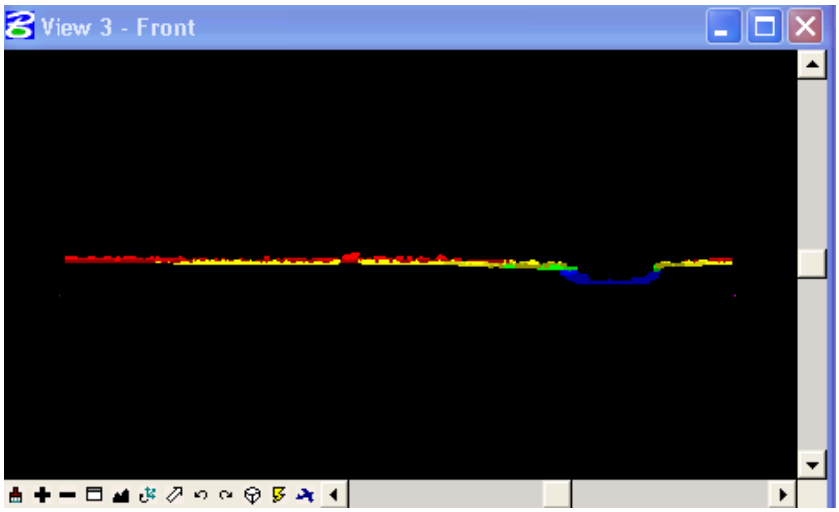


Figure 43: Section plotted

Goto **move section** button  . Draw a section and then select viewer for display. Press left/right click button of mouse to move the section forward or backward.

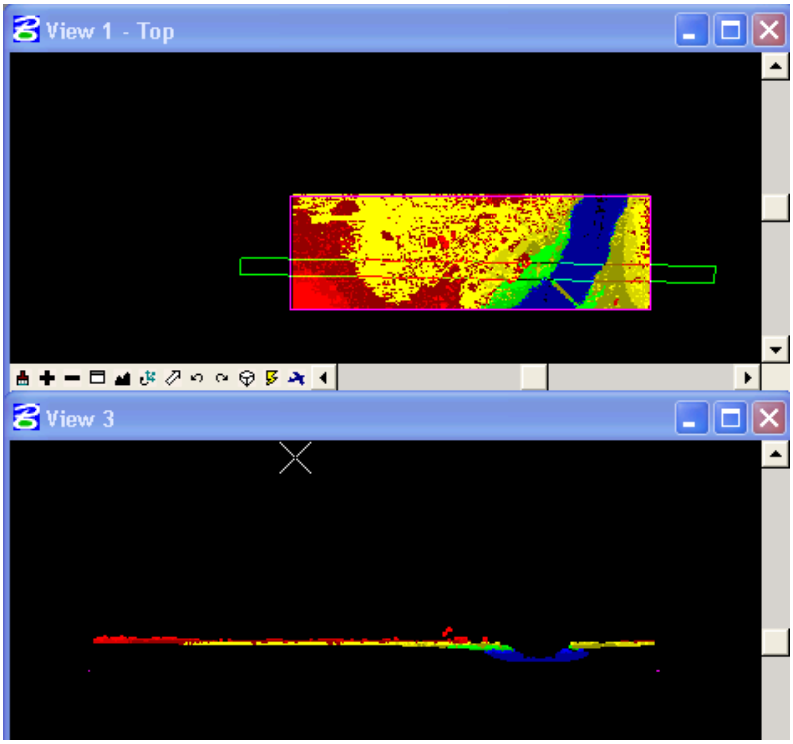


Figure 44: Moving a section

Goto **cut section**  . Place a section with a certain width and then click in a viewer to see the cut section.

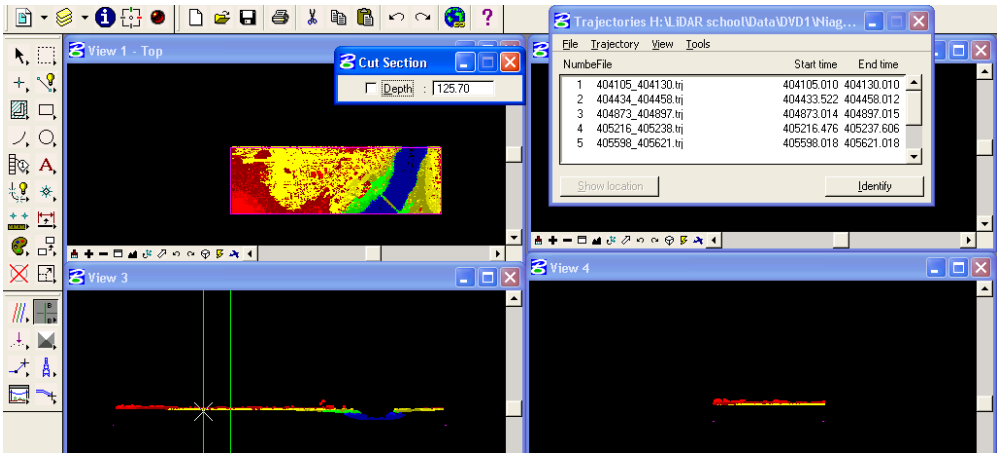


Figure 45: Cutting a section

LABORATORY NUMBER 2: DATA CALIBRATION

WHAT WOULD WE LEARN IN THIS LABORATORY

In this laboratory, we would learn to do the following

- Setting directories and importing trajectories.
- How to run a macro for Deduce line numbering.
- Classify ground points per flight line + low points and points below surface.
- Create a project for matching purposes and Measure match.
- Find match for heading, roll, pitch and mirror scale corrections. Solve these for entire project and save corrections and reports and then applying corrections.
- Find match for Z correction. Solve this for individual flight lines and use entire project save corrections and reports and applying corrections if big DZ differences.
- Find fluctuations for entire project, apply corrections.
- Again measure match for entire project to see the difference.
- Cut overlapping points to produce a more uniform data density and point pattern.

Open Microstation from ***Start***>*All programs*>*Microstation.*

Open the dgn file **lidar_school.dgn** from the main menu of the Microstaion as shown below.
File>Open>*dgn file name*

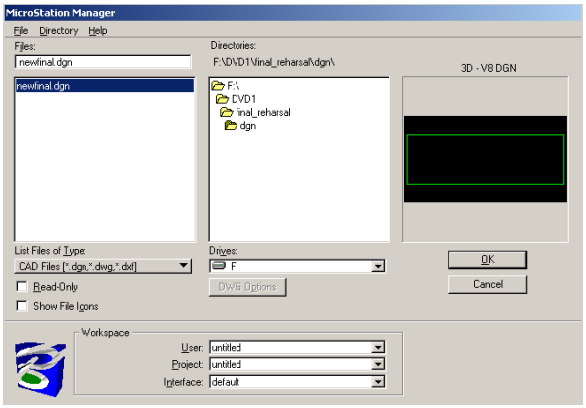


Figure 1: Opening a dgn file

Open Terra scan from main menu of micro station by going to *Utilities*>*MDL Applications*>***Tscan*** and click *load*. In the same way open the Terra Match from **utilities>MDL Applications>Tmatch** and click **load**. TerraScan main window also opens up.

Figure 2: TerraScan and TerraMatch tool bars

Figure 3: TerraScan window

SETTING DIRECTORY AND IMPORTING TRAJECTORIES

By clicking the Manage trajectories tool of Tscan  you can see the manage trajectories window as shown in the Figure.8 with which we can set the directory and import the trajectory information to that specified directory.

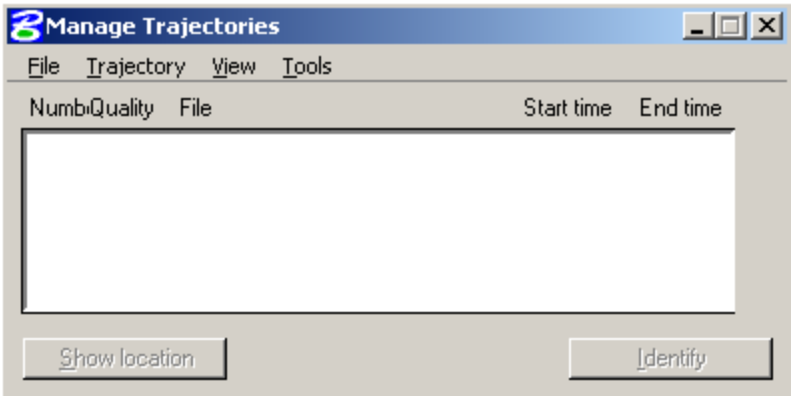


Figure 4: Manage Trajectory window

To set the directory go to File >Set Directory and then browse for the trajectory folder **trajectory_scan.**

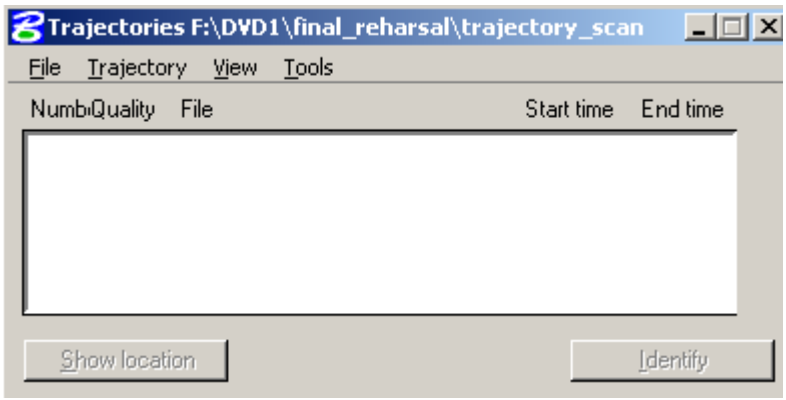


Figure 5: Trajectory window with set directory

To import the trajectory go to *File*>***import trajectories*** then you can see the window as shown in figure below then browse for the file ***sbet_01.out*** which is a trajectory file under sbet folder

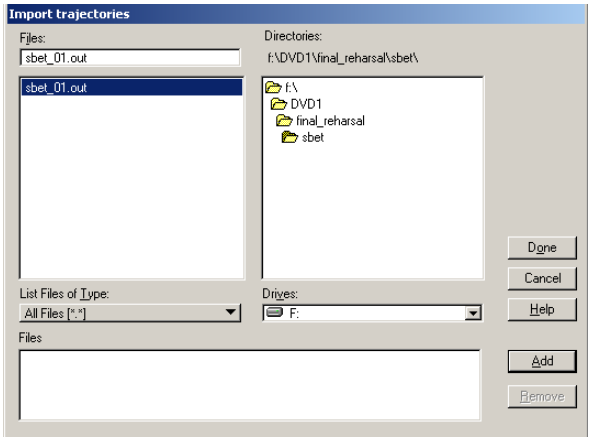


Figure 6: Import trajectories window

Then *click* on the file and click *Add* then you can see the file added in the list then click *Done*.

Now **Import trajectory window** will appear. ***Change*** the **WGS84 to UTM-17N (81W)** and *check* the *Thin positions box* and click *Ok*.

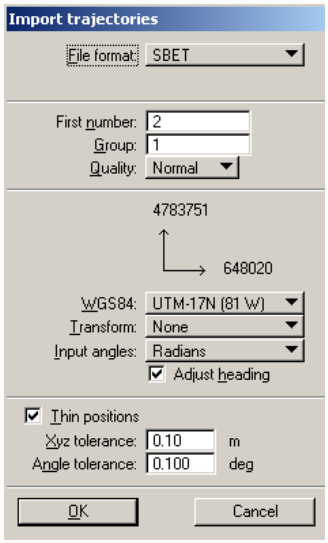


Figure 7: Setting parameters in import trajectories

Then the processing will takes as shown in the figure below

Figure 8: Processing sbet_01.out file

To see the trajectories in the view of the design file go to ***Tools>Draw into design*** as shown in the figure below.

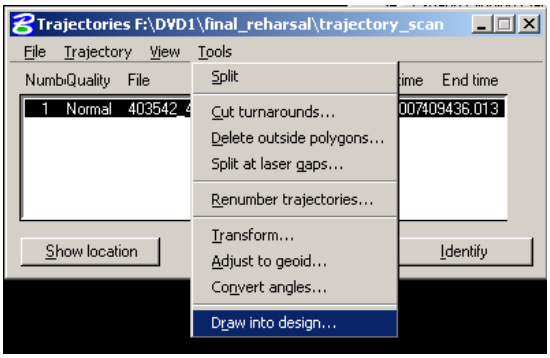


Figure 9: Tools -Draw into design

By clicking in the view you will get trajectories on the design file as shown in the figure below

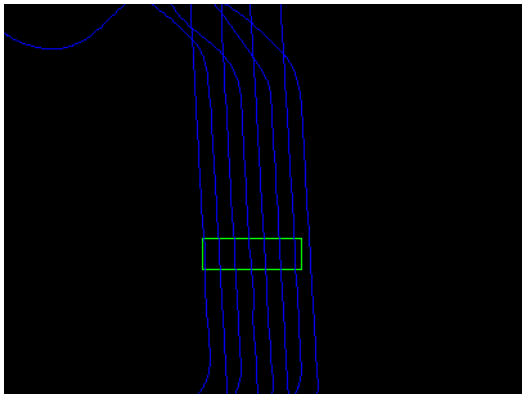


Figure 10: View of trajectories on the design file

Now to draw the trajectories into the design file place roughly a polygon using tools of microstation around the project area to cut useless part of the trajectories. After drawing a rectangle on the project area select that rough polygon as shown in the figure 11 below and then

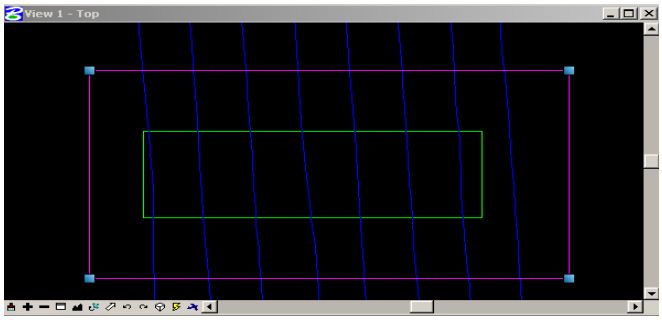


Figure 11: Place rough polygon over the project area

go to the tools option of the trajectories window as shown in figure12 ***Tools>Delete out side polygons*** for deleting the trajectiories outside that polygon and click **Ok**.

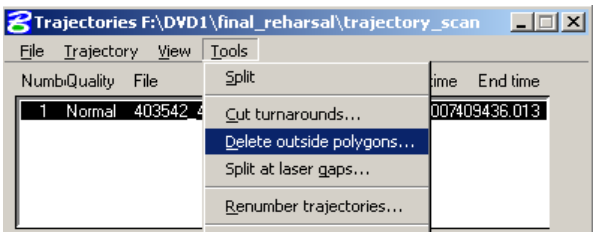


Figure 12: Tools-Delete outside polygons...

Now ***delete previous trajectories*** by *selecting* the trajectories and *clicking delete button* of your *keyboard* and *redraw trajectories* to see the changes in trajectory information as shown in the figure13 by going to ***Tools> Draw into design***

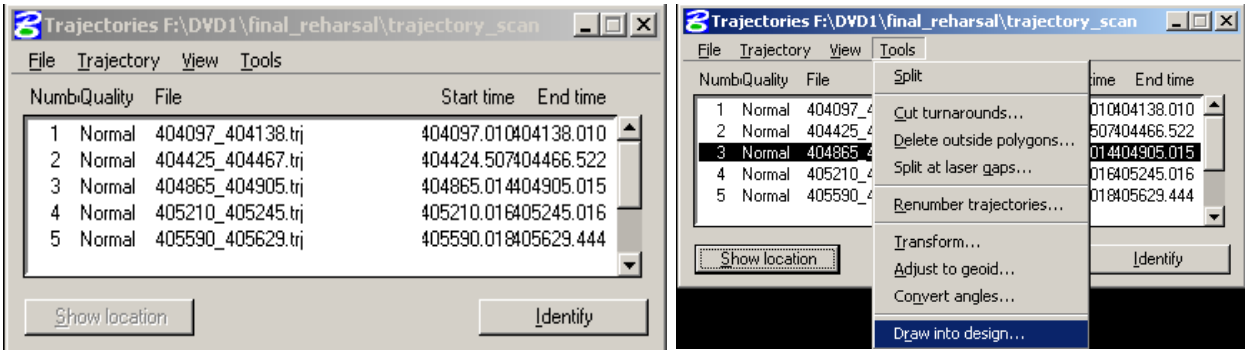


Figure 13: Redrawn trajectories into design file

Now you can see the changed trajectory information in the view as shown in figure 14

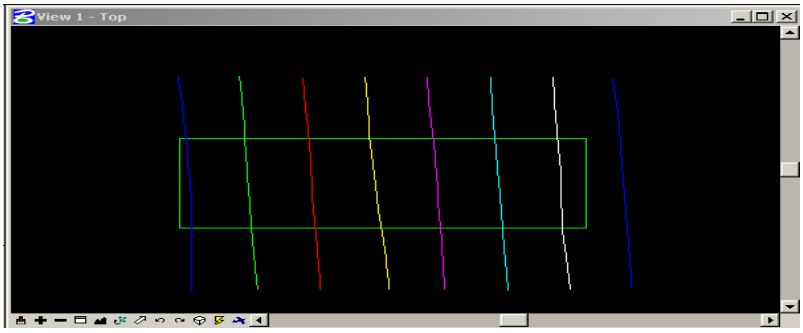
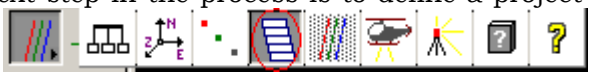


Figure 14: Changed trajectory view with design file

The next step in the process is to define a project by clicking on the icon shown in the Tscan tools. 

Now you will see the define project window as shown in figure 15.

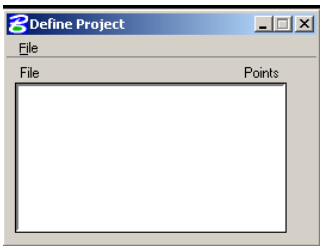


Figure 15: Define project window

For opening a new project go to ***File>New project*** then you will see the project information window where the description of the project has to be made as shown in figure 16. Define a project name as ***Niagara_1000***. And *check the Load class list automatically* and browse for the class file from ***Niagara.ptc*** under ***mission*** folder. Also check the *Load trajectories automatically box* and browse for *trajectories directory* from **trajectory_scan** and click **OK**.

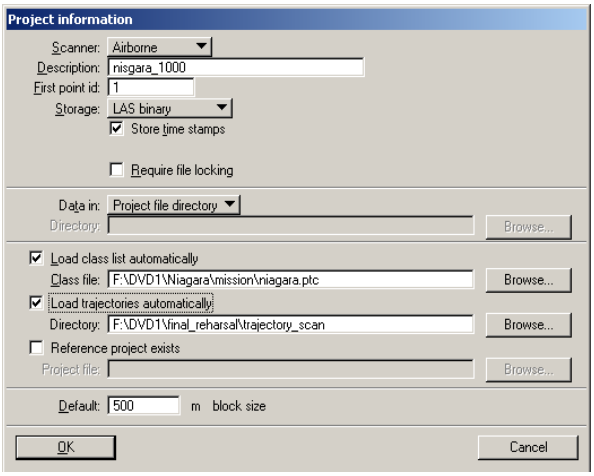


Figure 16: Project information window

Now you can ***save the project***, with the *same name* ***Niagara_1000***.

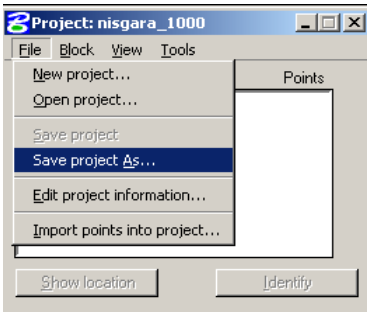


Figure 17: Save project as window

Select the *project polygon* as shown *to* ***import the point data*** into the project.

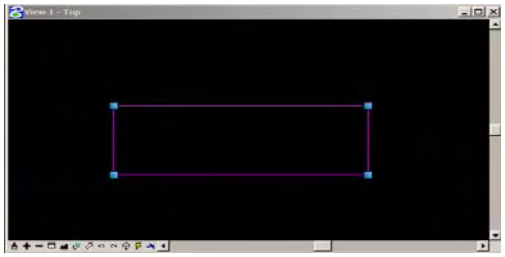


Figure 18: Selection of project polygon

Now select the ***block>Add by boundary*** then you will get the *Add block by boundary window* as in figure below. Give the name in *File prefix* as Niagara and click *Ok*.

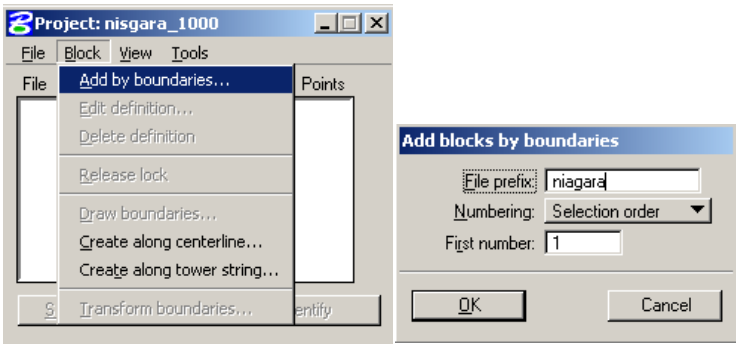


Figure 19: Add blocks by boundaries tools

You can see the block is added in to the active project as shown below

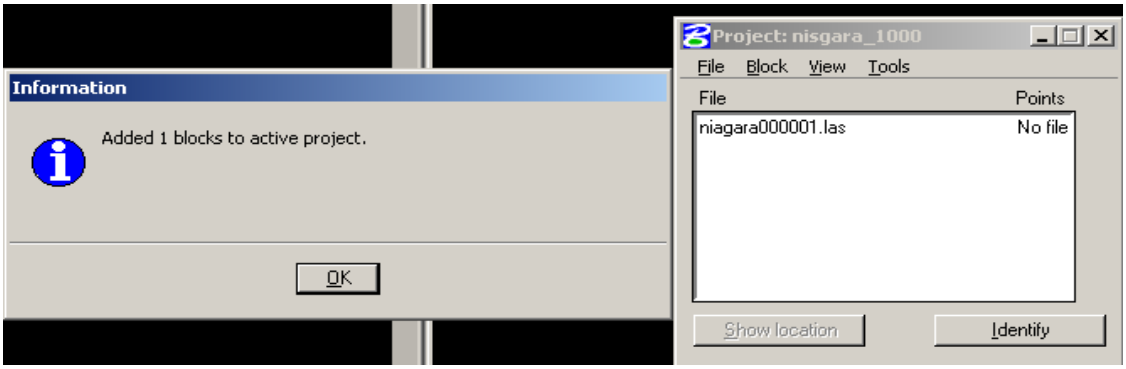


Figure 20: Block added into active project

Now start ***importing the points*** into the project by clicking *File>import* points into project.

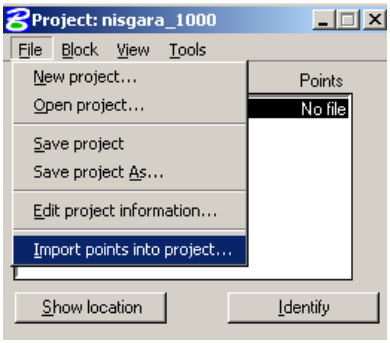


Figure 21: Import points into project

Now you will get a window as shown in figure 22 then browse for the LAS files from **laserraw** under Niagara and select all the 8 LAS files and click **Add and Done**. You will get load points window select *Ok.*

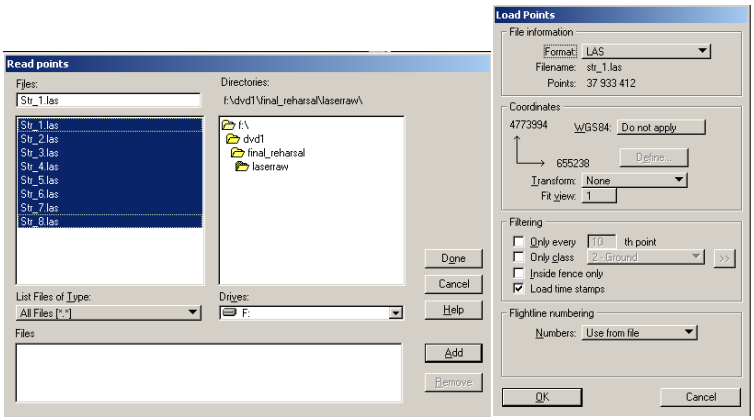


Figure 22: Reading and loading points

Then the points will be read into the project.

Now you will get information of the points that are imported and ignored as shown below.The number of points are shown in define project window as shown below.

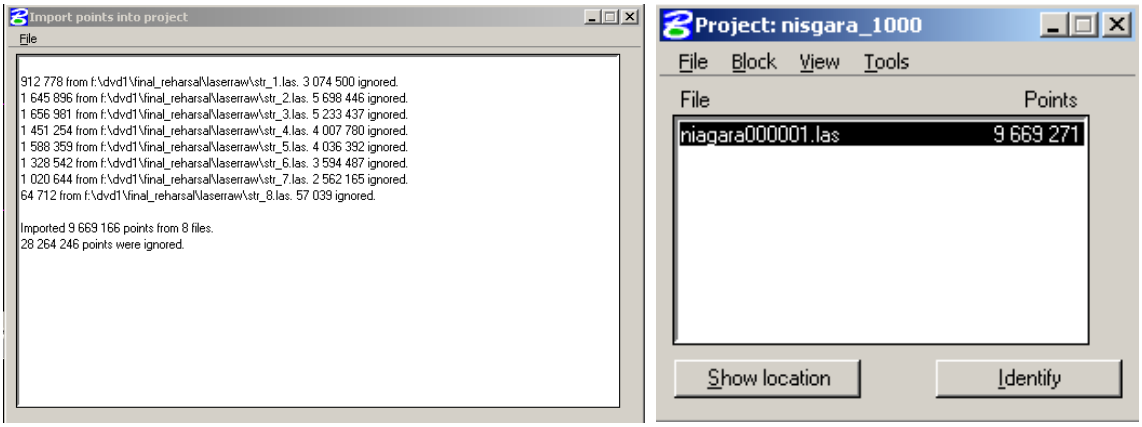


Figure 23: Imported points into the project

RUN A MACRO FOR DEDUCING LINE NUMBER

Now start macro tool to prepare data for different processes, first we **deduce the line numbering** for the laser points by going to *Tools>Macro…* as shown in below

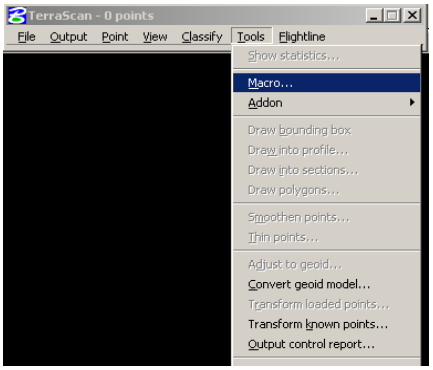
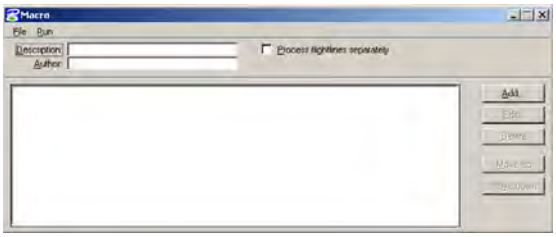


Figure 24: Starting Macro from Tool

Now you will see the macro window as shown below



Click the **Add** button to see the macro step then in Action select the *Deduce line numbers* as shown below then you can see the macro step with action as shown below simply click **Ok.**

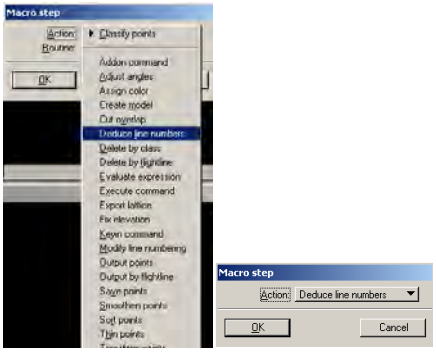


Figure 25: Macro step-Deduce line number

And *check* the box *process flighlines* separately and save it as ***deduce_line_number.mac*** in the **macro** folder.

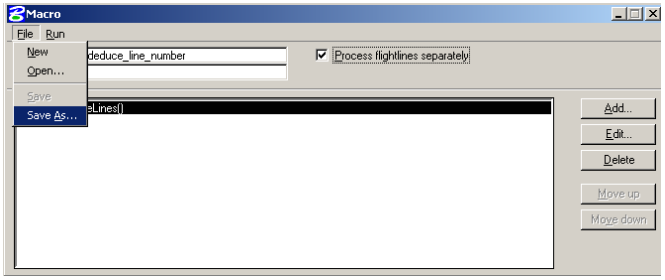


Figure 26: Macro saving

Now to ***process*** the deduce line number macro go to **Tools>Run macro** option of define project window as shown in Figure.36.

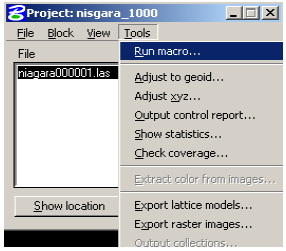


Figure 27: Run Macro tool

The *run macro on block* window will open as shown in the figure 28 simply click **Ok**. Then the macro will process on the block as shown.

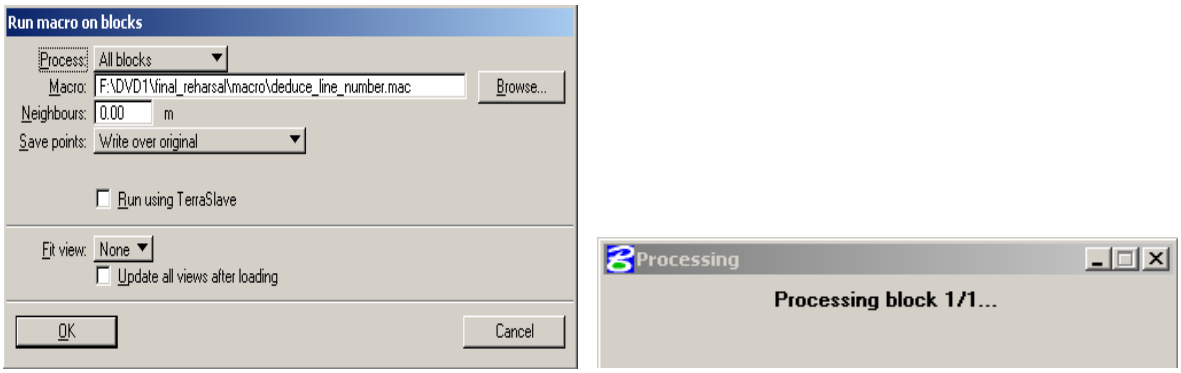


Figure 28: Run macro on block

After the processing or macro execution, the deduce line number report will be see on the screen as shown in below just save that file with *decduce_line_number_macroreport.txt*.

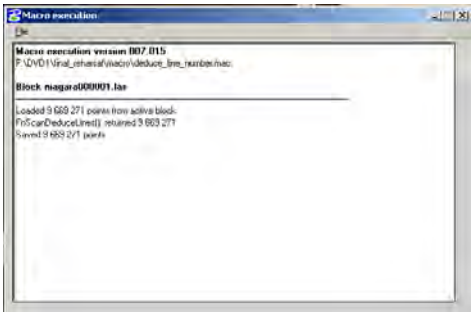


Figure 29: Macro report

Now *open the block* to ***read the points*** into the project by **File>open block** and by clicking inside the polygon as shown in Figure.39.

Figure 30: Opening block into project polygon

Now display the points with flightlines in the view as shown in Figure.40 by clicking on the display mode button and then go to color by flightline option as shown.

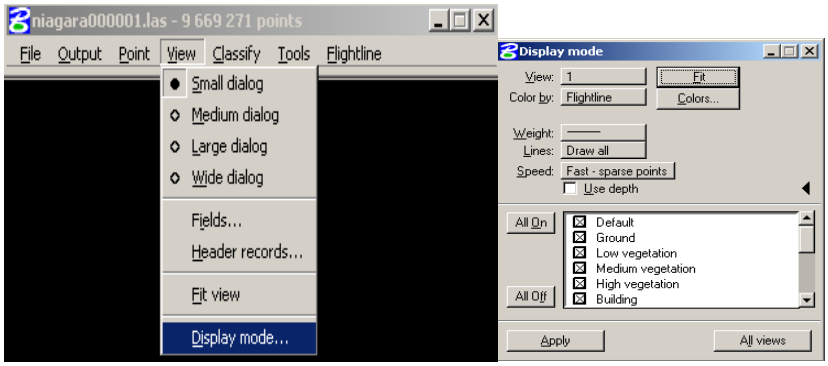


Figure 31: Display mode and flightline option

Now you still see the flight lines as shown below

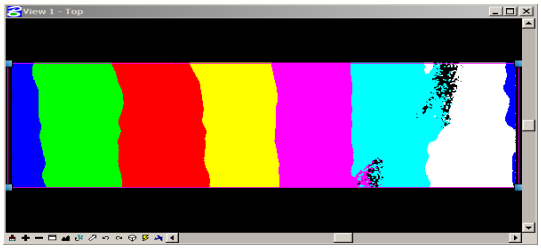


Figure 32: Screen shot of view by flightline

Now ***draw the trajectories*** *into the design* as shown in figure 33 so that the ***trajectories will match with the flightlines***. You can cross check it by clicking on the show location button.

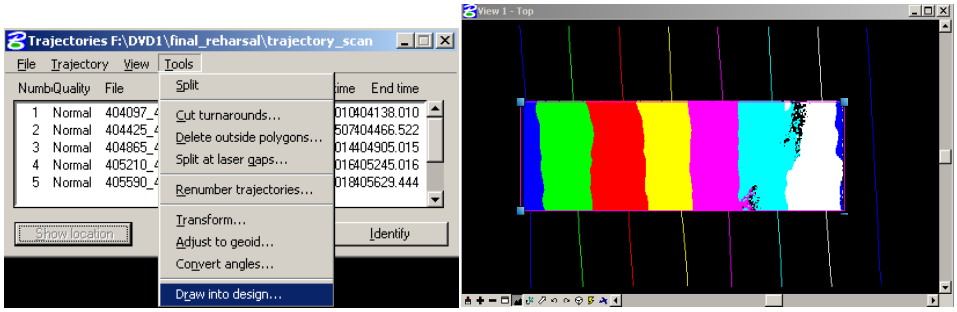


Figure 33: Trajectories matching flightlines.

Now you can *close the points* from the view by clicking on **File>close points**.

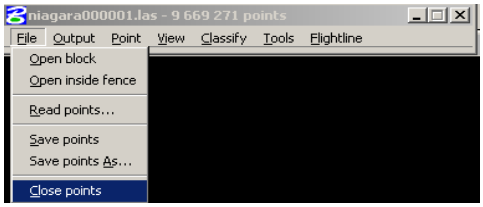


Figure 34: Close points from view

CLASSIFY GROUND POINTS PER FLIGHT LINE + LOW POINTS AND POINTS BELOW SURFACE

Create Macros for *classifying ground per flight line*. Since in data calibration ***Terra match creates Surface model from each flight strip*** and tries to ***find better matching*** for ***adjacent strips.***

Ground classification macro consists of four main steps as you will see in the following Figures.

First **create a macro** by clicking on **Tools>Macro...** then a macro window will open as shown in the figure 35. Click *Add* button and follow the steps and give the values as shown in Figures from 36 to 40

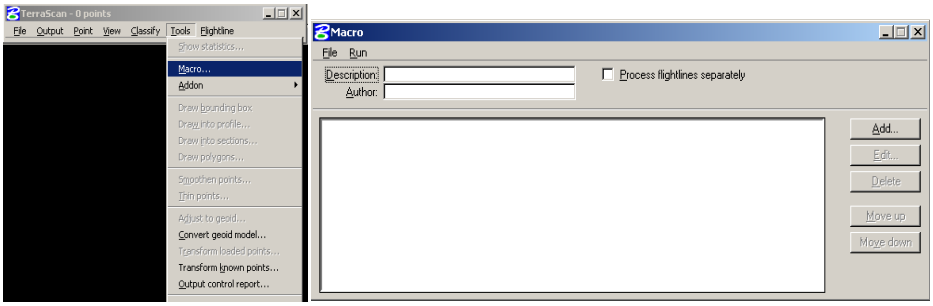


Figure 35: Defining macro from tools

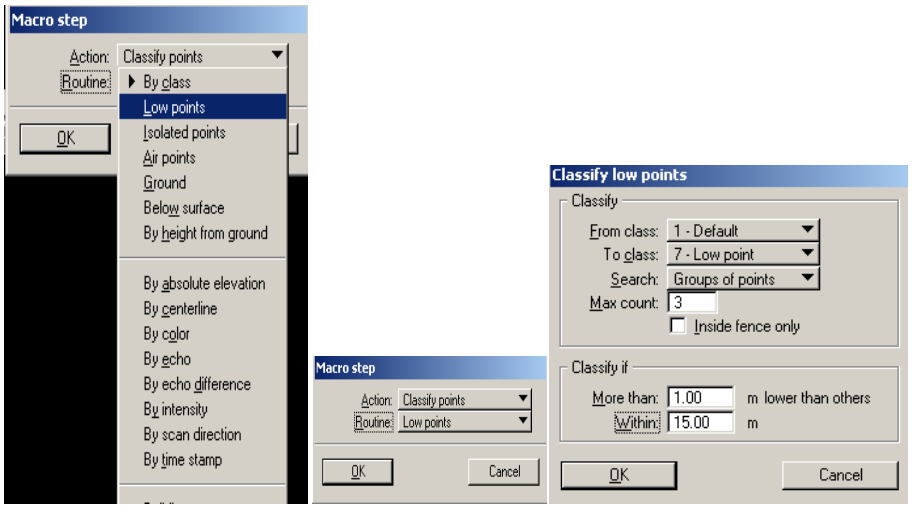


Figure 36: Macro-classify by low points-by group of points (search)

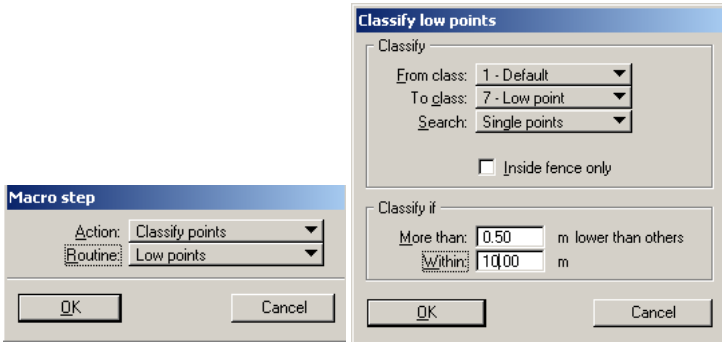


Figure 37: Macro-classify by low points-single point (search)

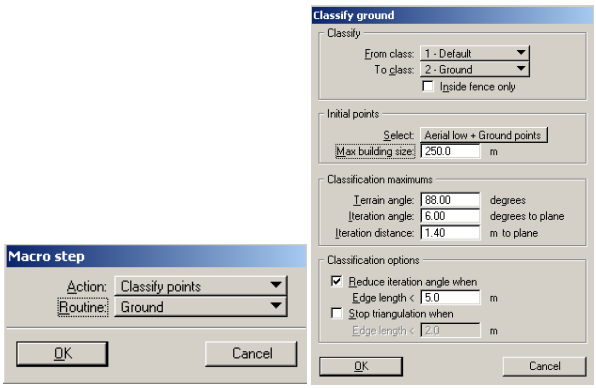


Figure 38: Macro-Classify by ground

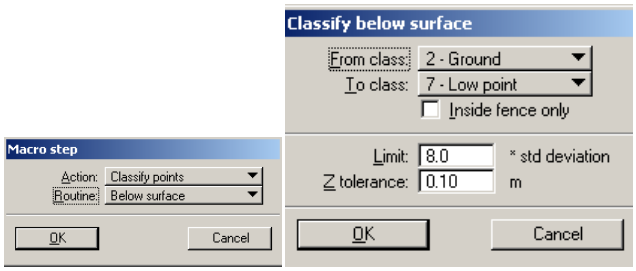


Figure 39: Macro- Classify by below points

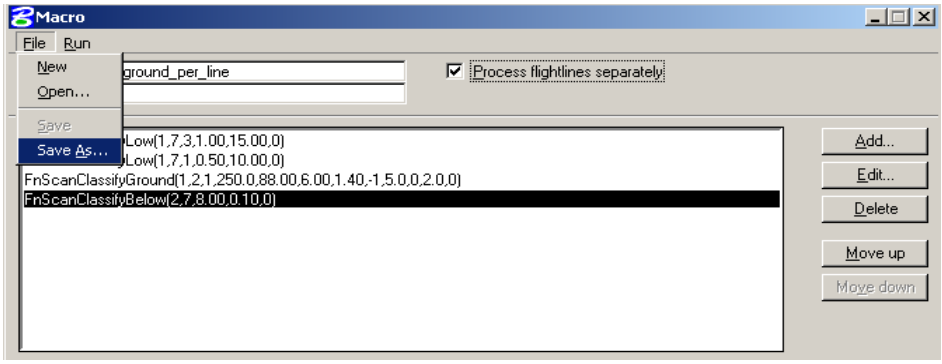


Figure 40: Save as macro of ground _per_line.mac

The save the macro with the file name **ground_per_line.mac** in **macro folder** as shown in figure below.

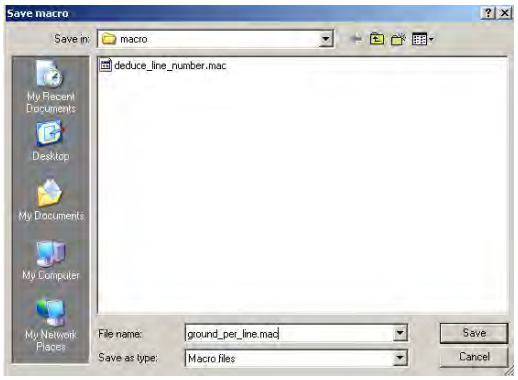


Figure 41: Save as window of macro

Open define project tool, Set big enough value in neighbours field, and *process ground classification*. **Start** ground classification **macro** by **Tools>Run macro…** as shown in figure 42.

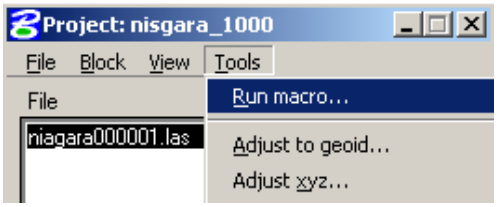


Figure 42: Rum Macro

You can see the ***Run macro on blocks*** window as shown in figure 43 and give the values as shown in the figure and click ***Ok.*** Then the processing will start as shown below.

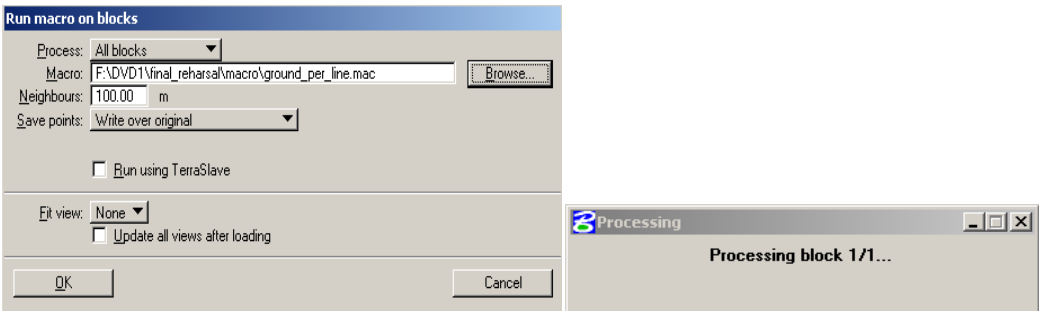


Figure 43: Run macro on blocks and processing of block

The macro will execute and the ***ground_per_line _macro report*** will be printed as shown in the figure 44. *Save the report* with file name *ground_per_line _macroreport.txt*.

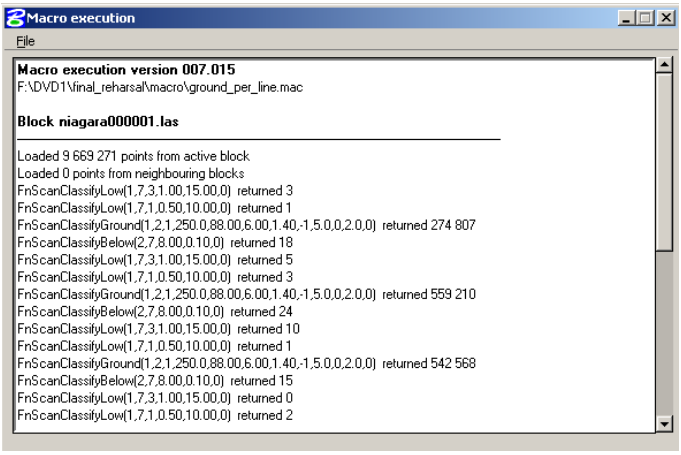


Figure 44: Macro report

CREATE A PROJECT FOR MATCHING PURPOSES AND MEASURE MATCH

You will save the ***project file*** as ***lidar_school*** for further processing of the data i.e. *matching purpose and measure match*. The save as window and giving file name is shown in figure 45.

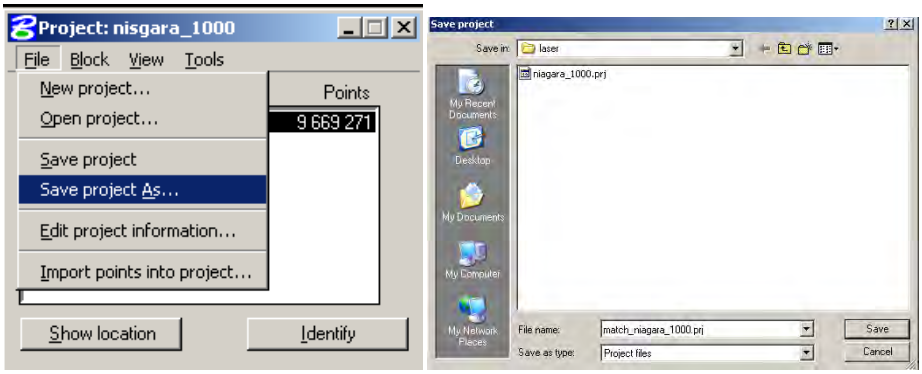


Figure 45: Save as lidar_school.prj file

Now ***open*** *the* ***Terra match*** 

tools for *measure match and applying corrections etc..*

Start measure match tool  , this measures *average difference between adjacent strips*, Make sure that ground classification is completed before starting measure match. Give the values as shown in figure 46 and *click Ok*.

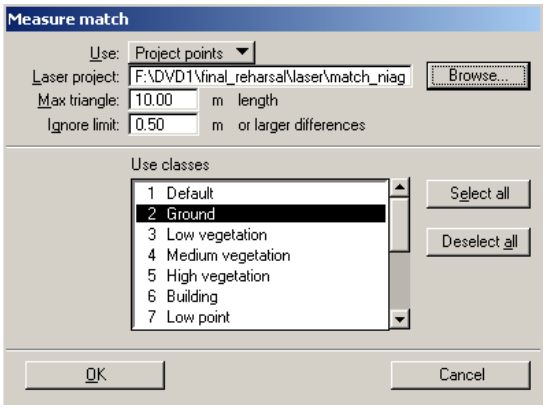


Figure 46: Measure match tool

After the processing of the measure match, the *report will generate* as shown in figure.47. Report shows ***average magnitude*** for whole project and ***difference between strips***, you can *save* report with file name *match_measurereport.txt*.

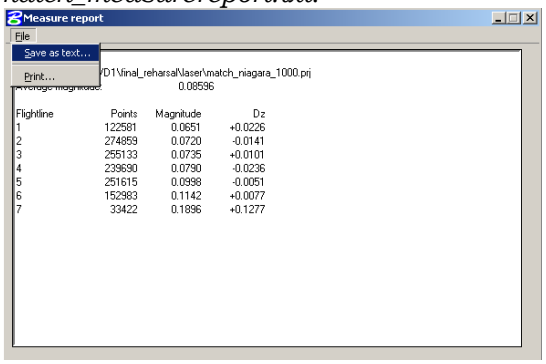


Figure 47: Measure match report

FIND MATCH FOR HEADING, ROLL, PITCH AND MIRROR SCALE CORRECTIONS. SOLVE THESE FOR ENTIRE PROJECT AND SAVE CORRECTIONS AND REPORTS AND THEN APPLYING CORRECTIONS

Start match tool  , on the first round, *solve for* ***heading, roll, pitch and mirror scales*** as shown in figure 48. ***Give*** the values and *check boxes* as shown and click *Ok* then the *iterations* will start as shown.

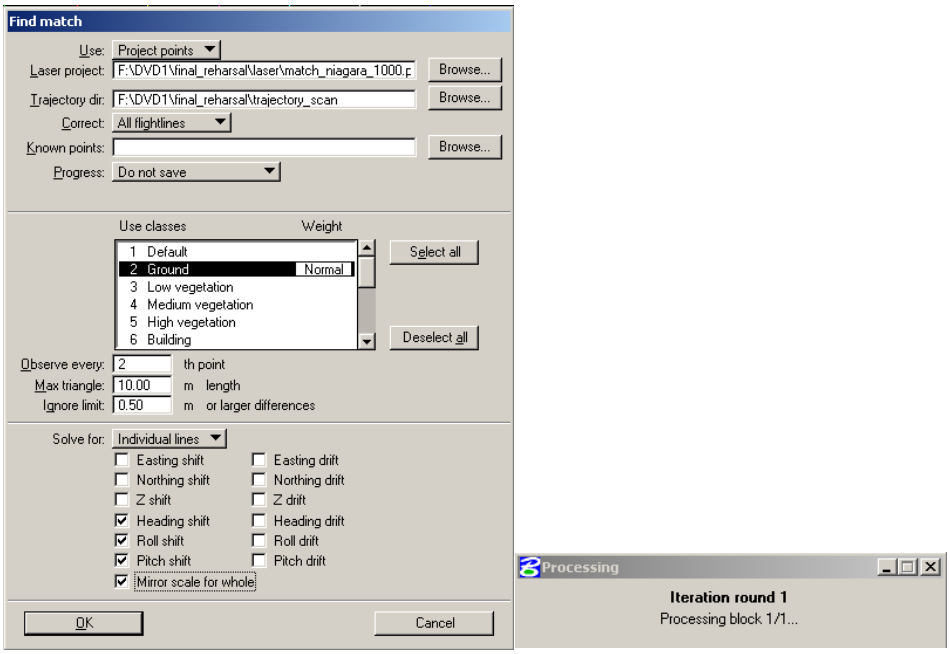


Figure 48: Find match tool and processing

Then the *report will generate* as shown in figure 49 and 50. Simply *save the corrections* by **File>Save corrections..** With the name ***hrpm_cor.tms*** under **laser folder** and also save the *report* as ***hrpm_cor_report.txt.***

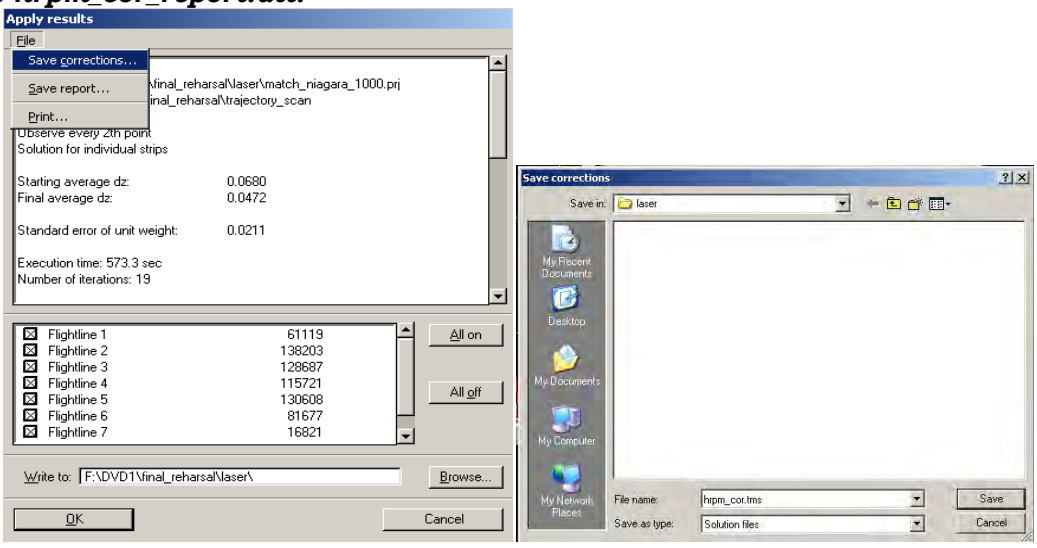


Figure 49: Corrections hrpm_cor.tms

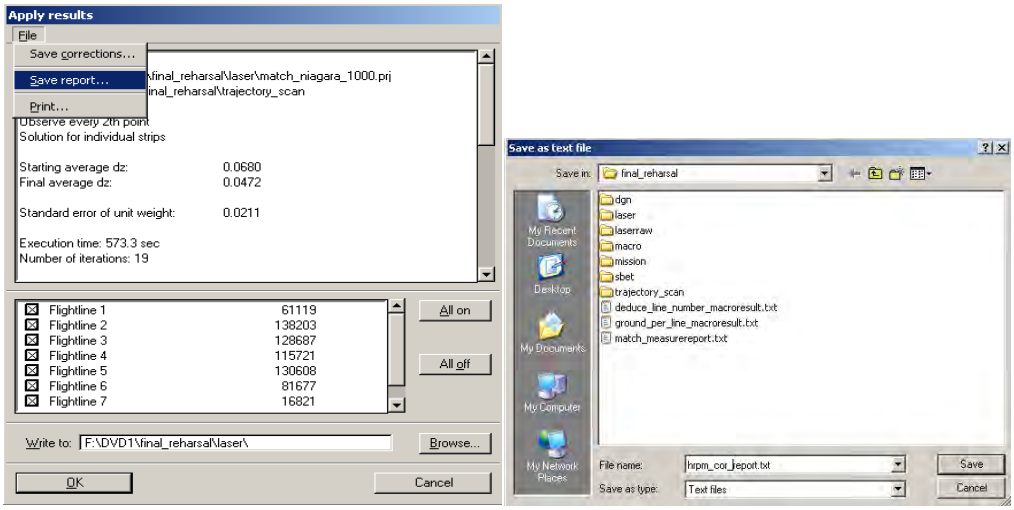


Figure 50: Corrections reports hrpm_cor_report

Save corrections *and report,* ***do not apply*** *corrections yet,* therefore click **cancel** for the window shown in Figure.50.

Start apply corrections tool  , ***select*** the *whole project and load correction file* as shown in figure 51. and click **Ok**.

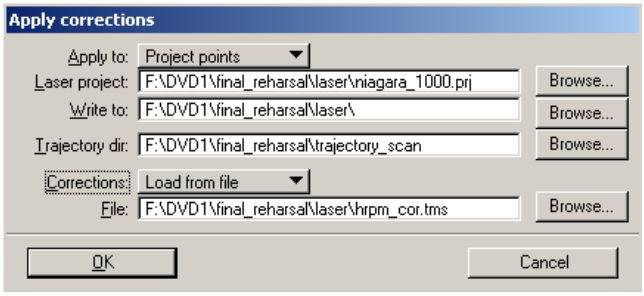


Figure 51: Apply correction to hrpm

Then you will Apply corrections to flightlines window as shown in figure 52 then *Click Ok.*

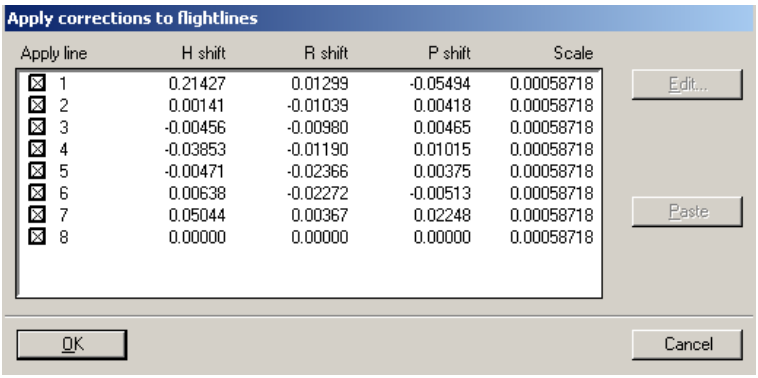


Figure 52: Apply corrections to flightlines

FIND MATCH FOR Z CORRECTION. SOLVE THIS FOR INDIVIDUAL FLIGHT LINES AND USE ENTIRE PROJECT SAVE CORRECTIONS AND REPORTS AND APPLYING CORRECTIONS IF BIG DZ DIFFERENCES.

Now ***start find match tool***  again and now solve for Z only, the values and check boxes should be as shown in figure 53.

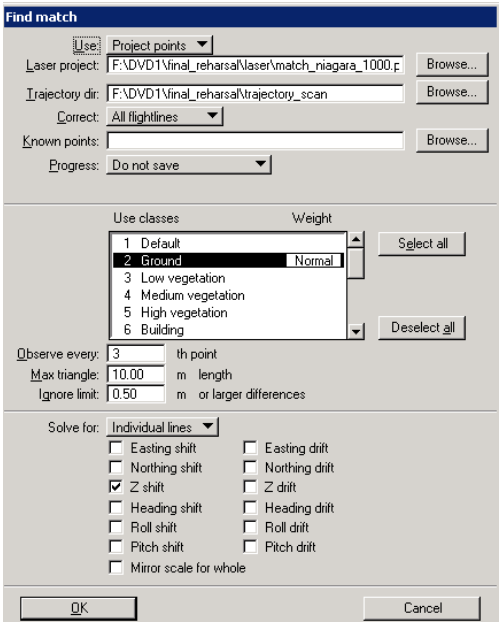


Figure 53: Find match for Z only

Then the report will generate as shown in figure 54 simply ***save the corrections*** by *File>Save* corrections.. With the ***name Z_cor.tms*** **under laser** and also *save the report* as ***Z_cor_report.txt***

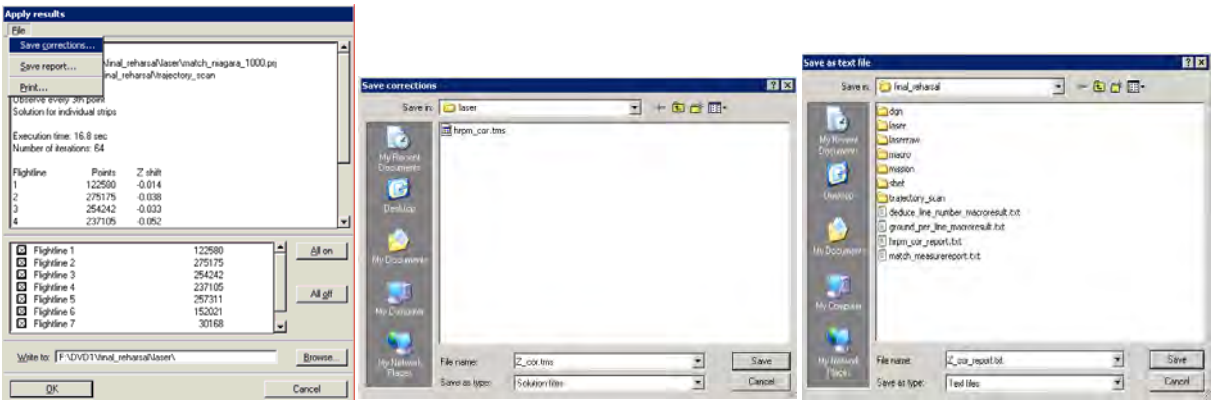


Figure 54: Save corrections of apply results window

You can apply results immediately but now *we practise* how to do if *some strips seem to have more Z error than others.*

Flight line **7** has little bit bigger Z shift, in manage trajectories window we can change the quality of trajectory then in correction computation line 7 has less weight

Save the corrections and *change the quality of trajectory 7* as poor by going to **Trajectory>Edit information...** as shown in figure 55.

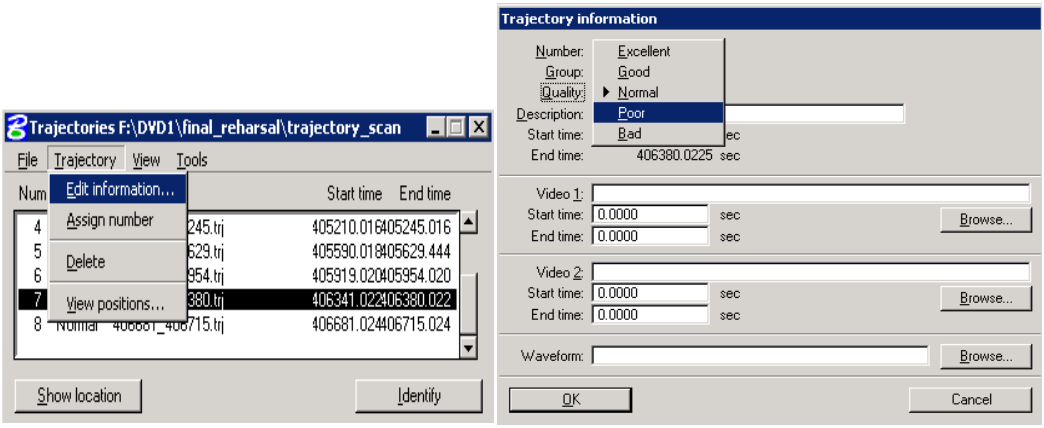


Figure 55: Edit trajectory and quality information

Apply correction  to the project file with Z-cor.tms file as shown in figure 56.

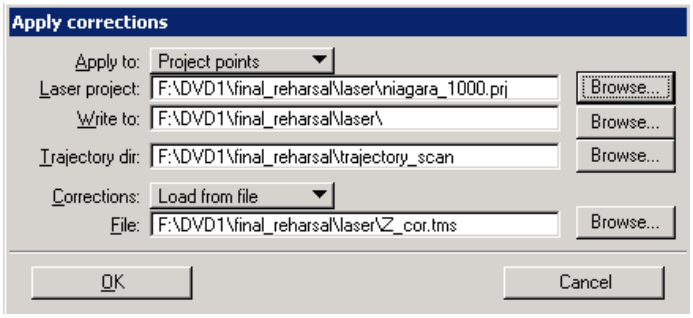


Figure 56: Apply correction for Z-cor.tms

You can *see the corrections* report for the flighlines in the figure 57 and click Ok.

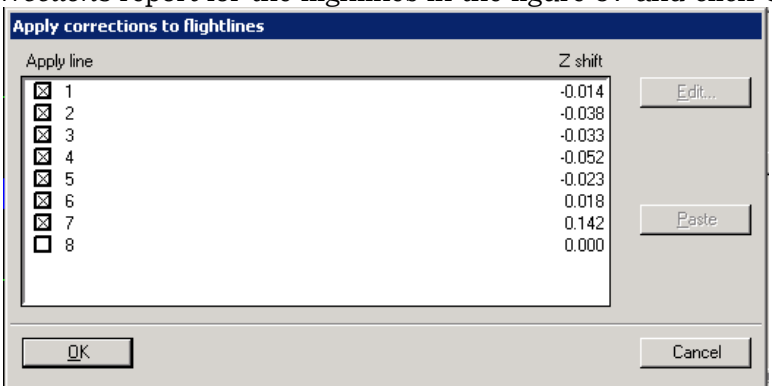


Figure 57: Apply corrections report for Z-cor

FIND FLUCTUATIONS FOR ENTIRE PROJECT, APPLY CORRECTIONS

Start find fluctuations tool  in trajectories are of little bit inaccuracy, in the same place on overlapping area should be same Z from adjacent strips, but because of *inaccurate positioning* in many places the *Z differences will not match*. Software finds these local Z differences and the *yellow bars* present Z differences in 2 seconds interval, look at different flightlines for your convenience. Click the **find fluctuations icon**  and give the values as shown in figure 58.

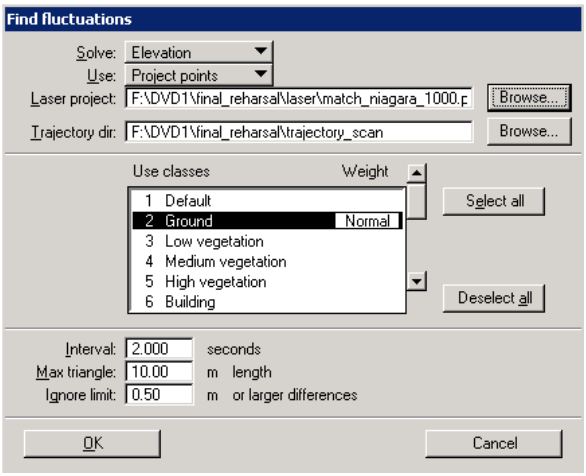


Figure 58: Find fluctuations

Now you can see the *fluctuations in the data points, save the corrections file* with name **fluctuations_*cor.tms*** as shown in figure 59 and 60 under *laser folder.*

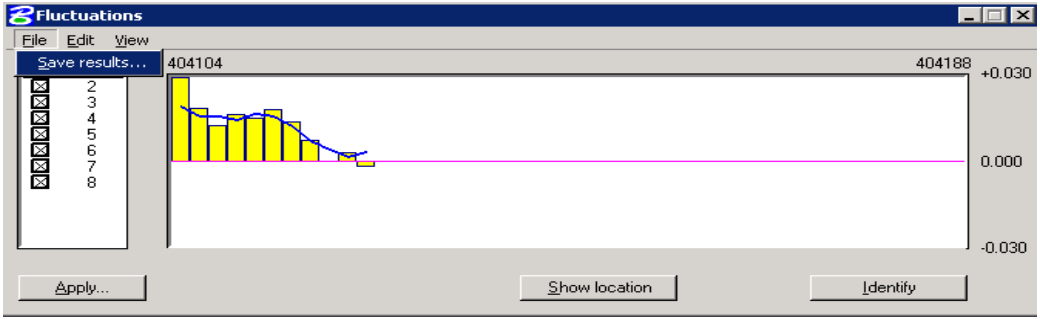


Figure 59: Fluctuations window

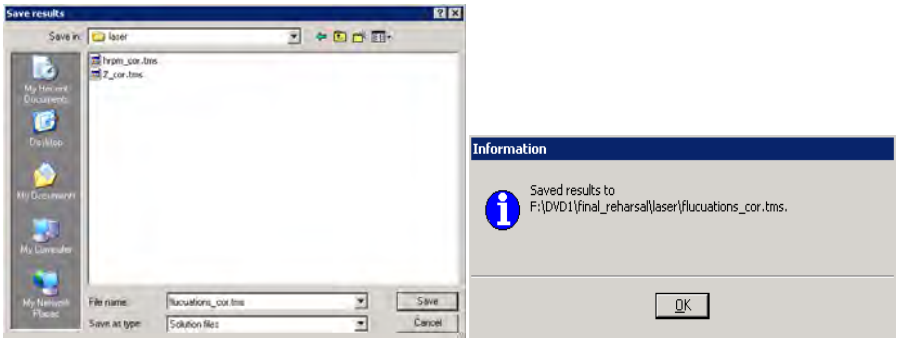


Figure 60: Save as window of fluctuation_cor.tms

You will get the *apply fluctuation correction window* as shown in figure 61 then click *Ok* to *apply corrections*.

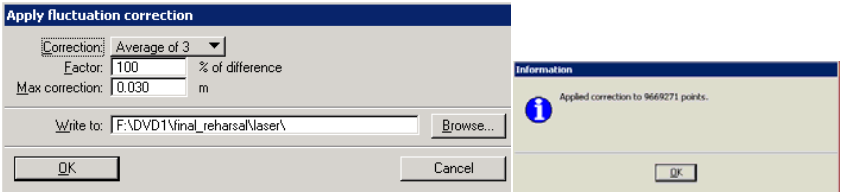


Figure 61: Apply fluctuation correction and information

AGAIN MEASURE MATCH FOR ENTIRE PROJECT TO SEE THE DIFFERENCE

Measure final average magnitude by **measure match** to see the applied corrections as shown in figure 62 and 63

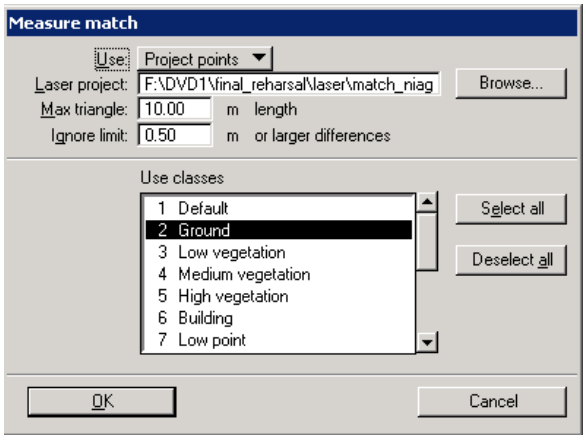


Figure 62: Measure match

You will get the *final report* for the *applied corrections* as in figure 63.

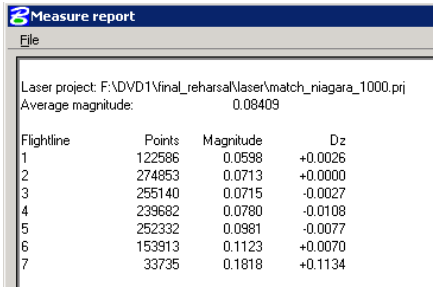


Figure 63: Measure match report

CUT OVERLAPPING POINTS TO PRODUCE A MORE UNIFORM DATA DENSITY AND POINT PATTERN

Open the block and use ***elevation*** and ***flightline coloring***, find **buildings on overlapping** areas and *take section views* from them as shown in figures 64 to 66C

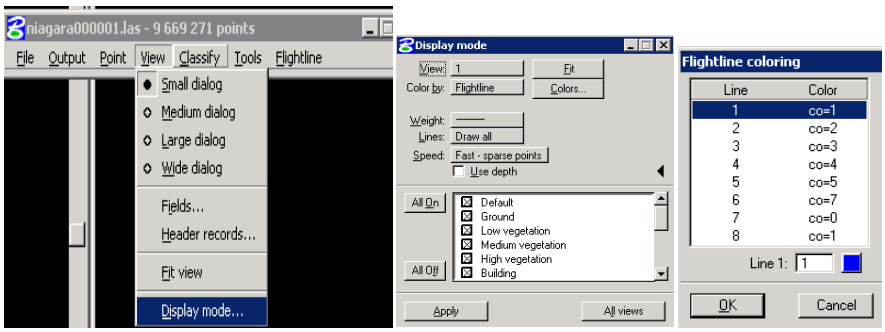


Figure 64: Display by flightline view

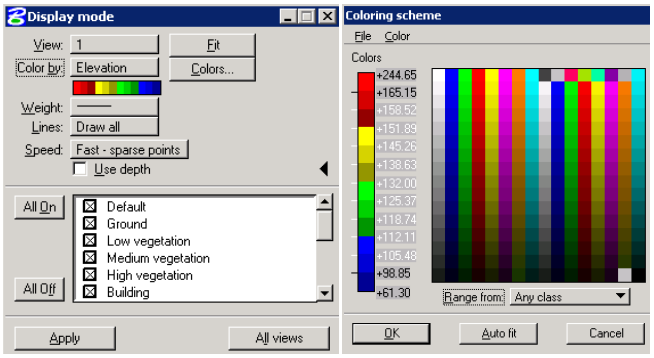


Figure 65: Display by elevation

In the In the figure 66A the top and section view of default points in elevation coloring are shown, and In figure 66B the top and section view of all points in elevation coloring are shown and In figure 66C the top and section view of all points in flightline coloring are shown respectively.

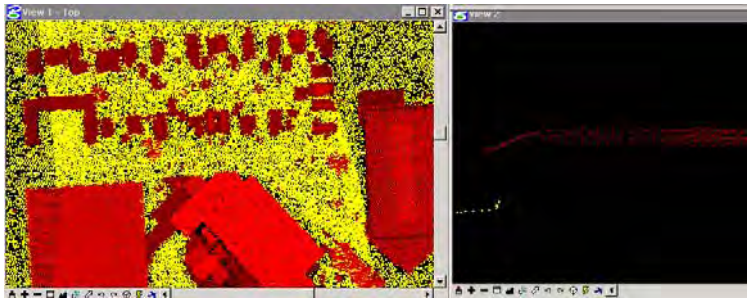


Figure 66A: Top and section view of default points in elevation coloring

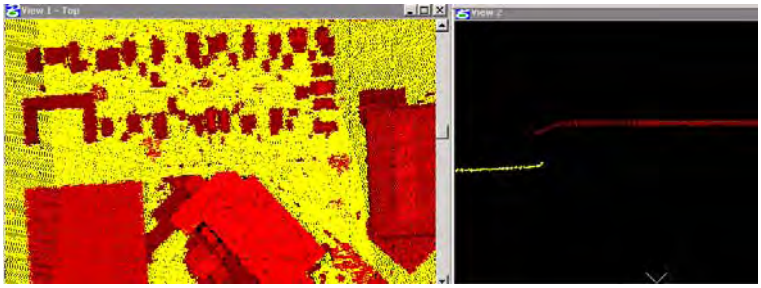


Figure 67: Top and section view of all points in elevation coloring

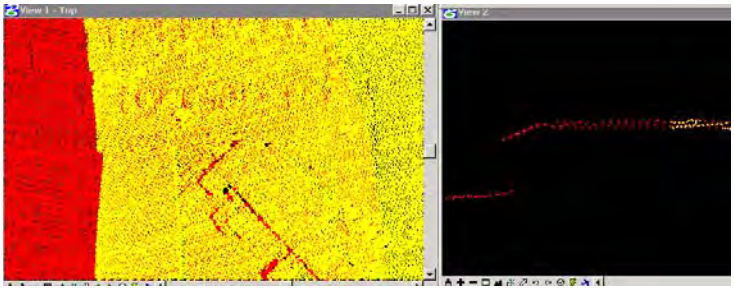


Figure 68: Top and section view of all points in flightline coloring

Run *cut overlap macro* for *cutting the overlapping points* in the block.

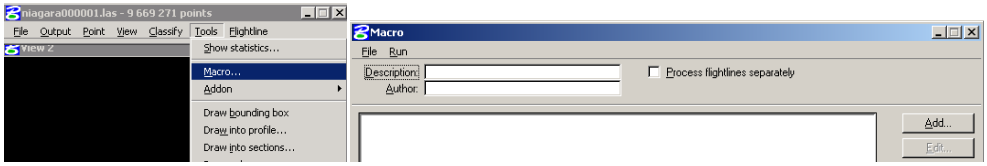


Figure 67: Macro tool

Select the cut overlap from the action as shown in figure 68.

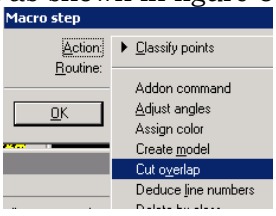


Figure 68: Cut overlap macro

Give the name as ***cut_overlap_10deg*** and *values* in the macro step as shown in the figure 69 and click **Ok**.

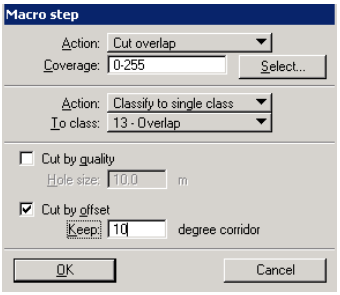


Figure 69: Macro step for cut overlap

And then *save as* the macro as ***cut_overlap_10deg.mac*** inside macro folder as shown in the Figure 70

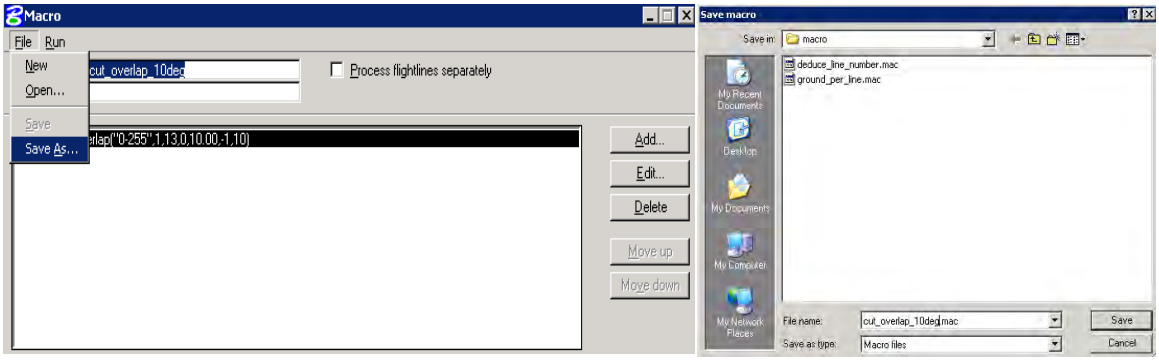


Figure 70: Save as cut_overlap_10deg.mac

Then *run the macro* using the option given in the figure 71 and keep the *values and check* boxes as shown in the figure 72 and click Ok then the processing starts as shown.

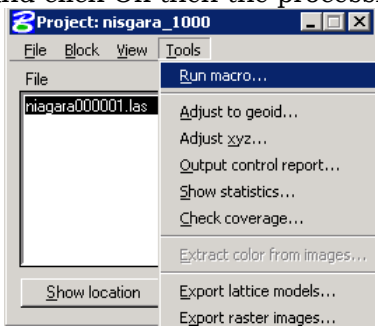


Figure 71: Run macro tool

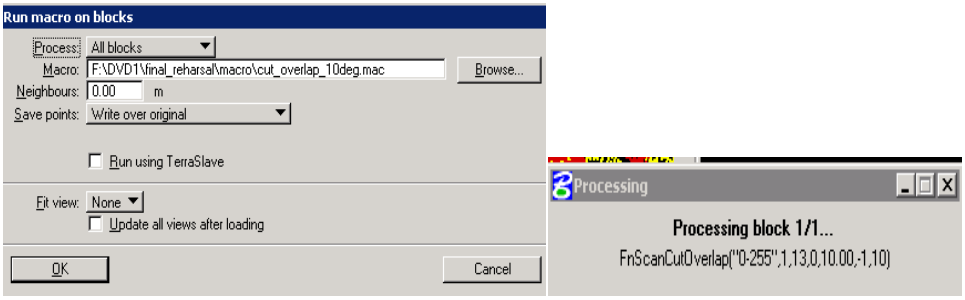


Figure 72: Run macro on blocks and processing

After execution of the macro a *report* will generate *save that file* as *cut_overlap_10deg.txt* which is shown in figure 73.

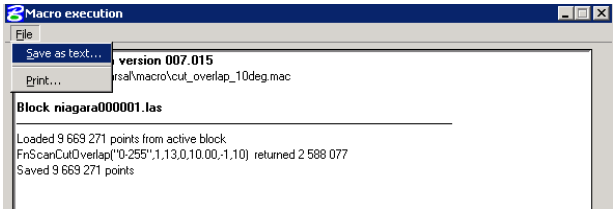


Figure 73: Macro report

After the running a macro on overlapping points again open the block and use elevation and flightline coloring, find buildings on overlapping areas and take section views from them to see the difference as shown in figures74 and 75.

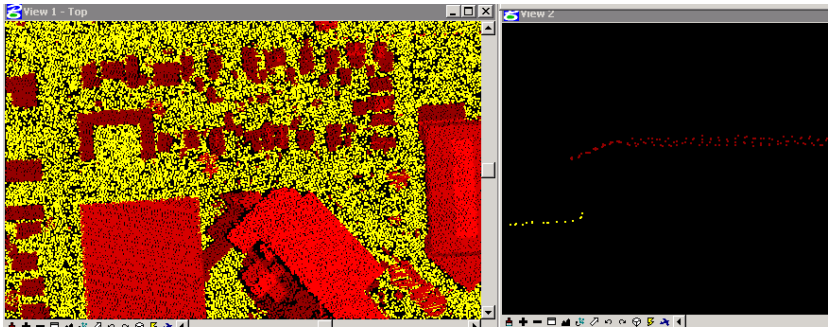


Figure 74: Top and section view all points in elevation coloring

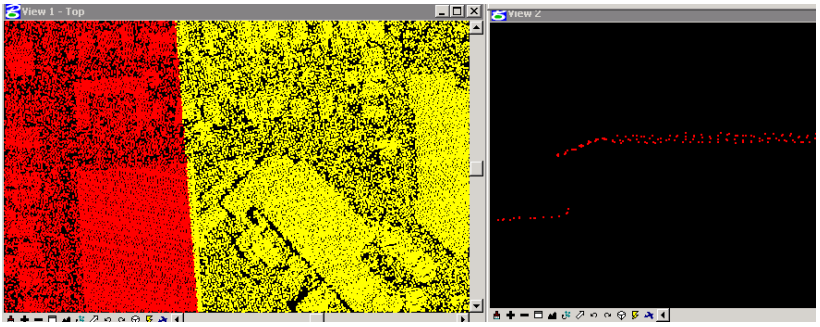


Figure 75: Top and section view all points in flightline coloring

LABORATORY NUMBER 3: DETAILED CLASSIFICATION

INTRODUCTION

Classification using *Terrasolid* software refers to grouping the LiDAR point data into different classes such as below ground, ground, vegetation, building etc.

WHAT WOULD WE LEARN IN THIS LABORATORY

In this laboratory, we would learn to do the following

- Classify points to Below Ground class
  - Classify low points by Absolute Elevation Routine
  - Classify low points by Low Points Routine
  - Classify points by Below Surface Routine
- Classify points to Approximate Ground class
- Correction of Approximate Ground class points to generate accurate ground class points
- Classify points to above surface feature class such as Building, Vegetation etc.
  - Building classification
  - Low vegetation classification
  - Medium vegetation classification
  - High vegetation classification

HOW TO DO IT

We have already learnt how to subset data, apply different corrections to data points. Now we will try to classify points to various ground cover classes. *Terrasolid* software has well defined routines to classify points. But it requires different parameter inputs at the time of processing. Inputs are related to nature of terrain and/or geometry of various land features.

In *Terrasolid* software classification can be carried out using either already defined **Routines**, or **Macros** written using Routines, or **Interactive** schemes aided by Create Editable Model and Orthophotos (georectified image). It is also possible to classify through combination of different techniques. In this laboratory we will try to adopt different technique to get an idea about different procedures.

Since the grouping of points in to below surface class is not complete unless ground is classified. Hence classification of low points is done in two stages i.e. before and after ground classification.

CLASSIFICATION

DATA PREPARATION

We will now proceed to carry out classification from the given data. We have been given the file **cut_bdry.dgn**. This file contains the boundary of the LiDAR data file within which the laser data corrected in lab 2 i.e. **lidar_school.las** would be loaded. Subsequently this data will be used for classification.

Open the '.dgn' file using Microstation.

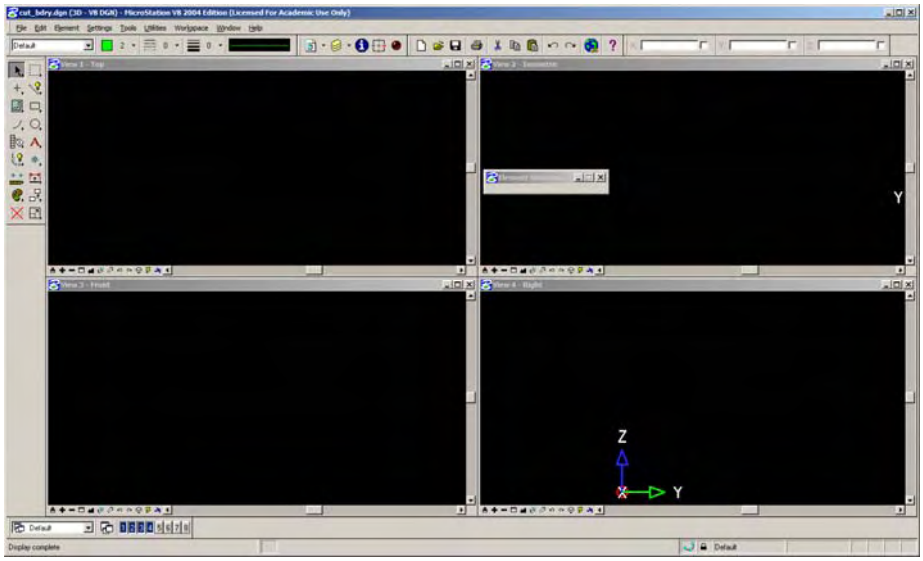


Figure 1: File opened in Microstation

However, the boundary which has been digitized inside this design file is not visible in the window. Thus, in view 1, click the fifth icon in the **view toolbar** which will fit the digitized design to the view.

Figure 2: View Toolbar

For the purpose of this exercise, we would use TerraScan (TSCAN).We would load this through *Tools* > *MDL Applications* in the Microstation menu.

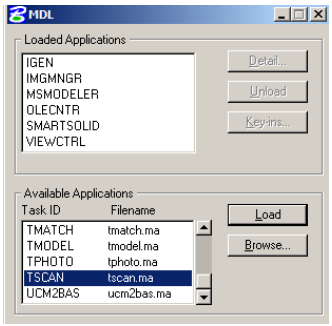


Figure 3: Loading TerraScan using Utilities > MDL Applications

The toolboxes for TSCAN would be loaded in the Microstation environment along with main windows.

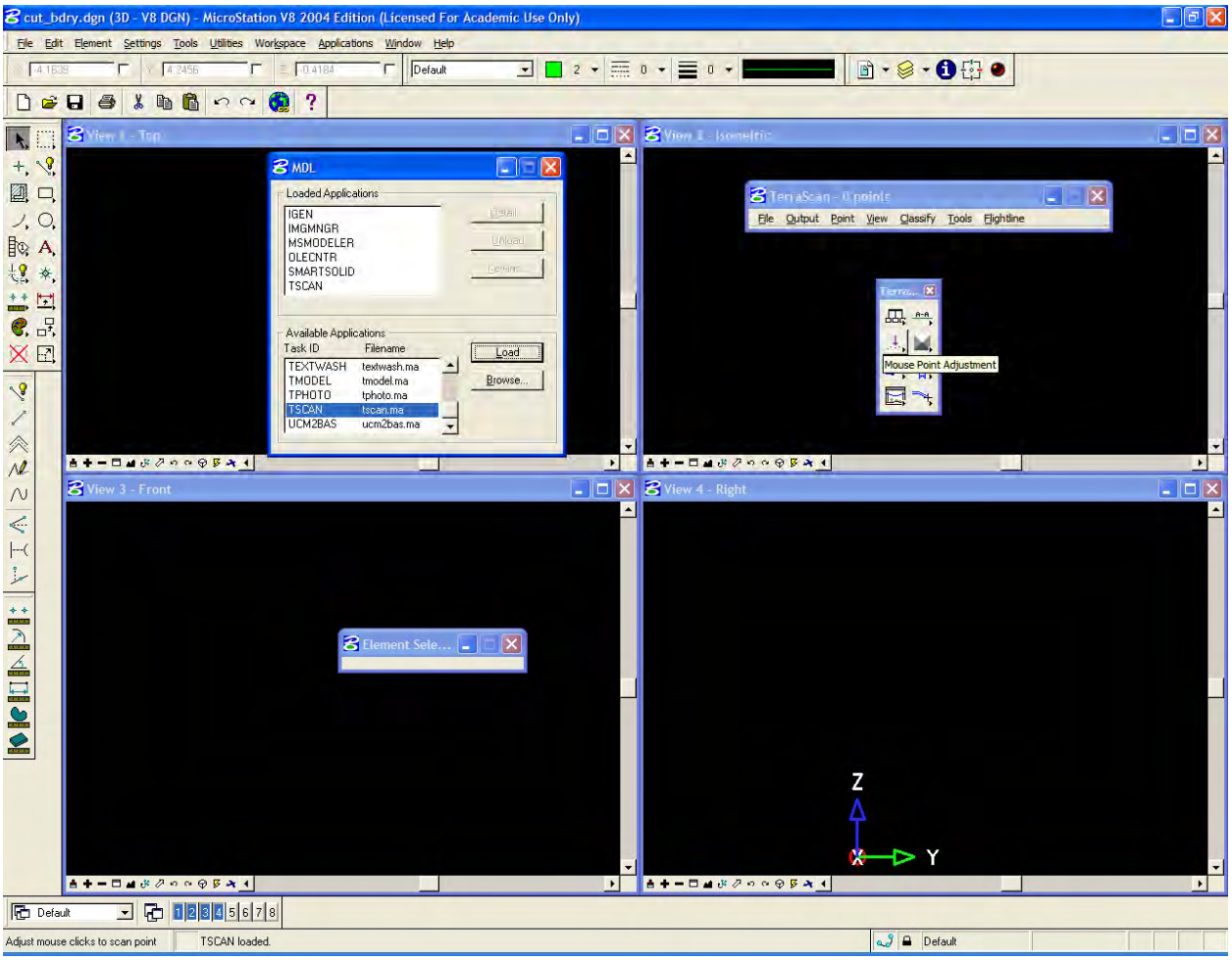


Figure 4: Toolboxes and main windows loaded in the workspace

We will now create a new project using the Terrascan toolbox. To do this, click and drag the first icon on the Terrascan toolbox. A new toolbar would emerge. This toolbar can also be accessed using *Applications* > *TerraScan* > *General*.

Figure 5: General Toolbar in the TerraScan application

The fourth icon in this toolbar manages projects. Click on this icon to open the **Define Project** window.

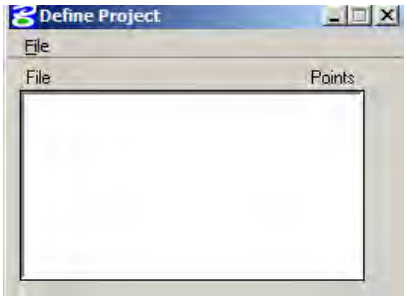


Figure 6: Define Project window

On this window, click File > New Project. In the new **Project information** window, fill in the descriptions as given in the Figure 7.

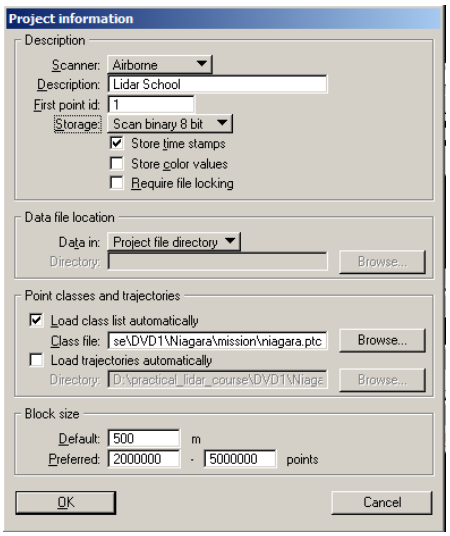


Figure 7: Project information details window

Click *File* > *Save As* and save the project as **classification_project.prj**.

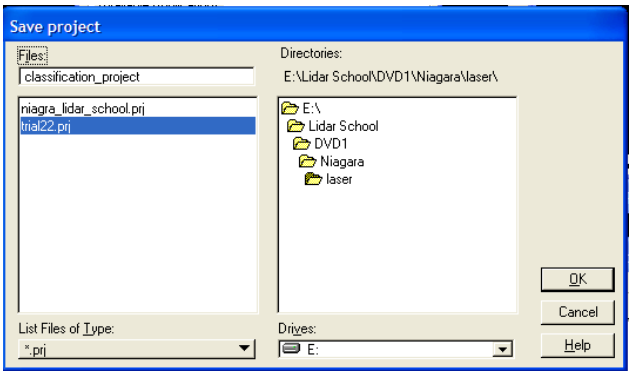


Figure 8: Saving project

Select the block on the window, using  on the Microstation main toolbar. On the **Project Definitions** window, click *Block* > *Add by Boundaries*. In this new window, click OK.

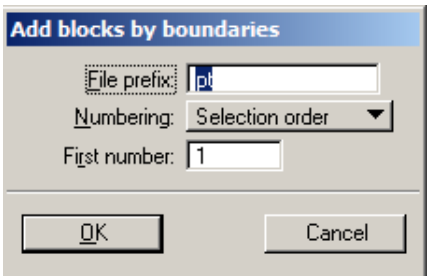


Figure 9: Add blocks by boundaries

A confirmation dialog would show up. Click OK again.

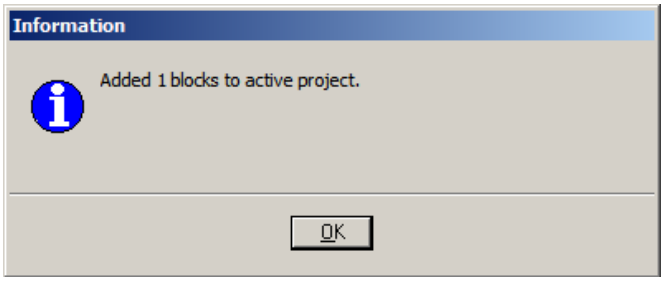


Figure 10: Confirmation window for a block addition.

As we kept our description earlier as **Lidar School,** note that the same text appears in the **Project Definitions** window (Figure 11). In this window, click *File > Import Points to project.*

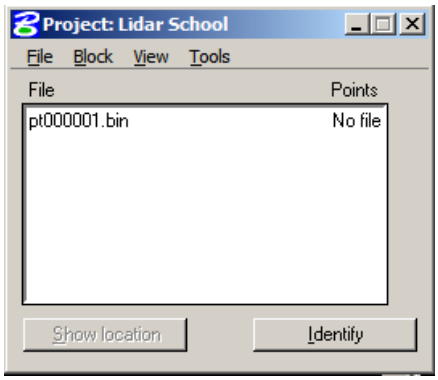


Figure 11: Block imported into the project

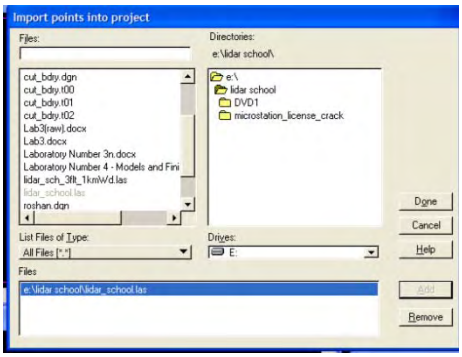


Figure 12: Selecting the LAS file for import

Select the file **lidar_school.las** and click *Add*. Click *Done*. In the new dialog box, click Ok.

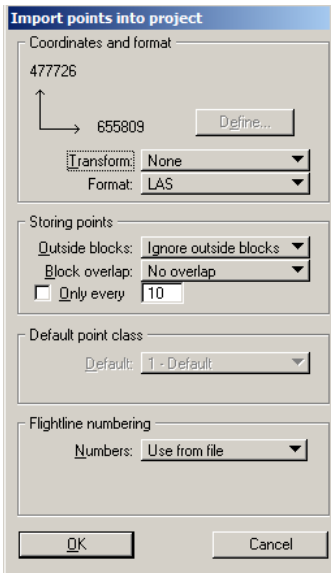


Figure 13: Importing points from the LAS file into the project

The report for the loading of points will now be generated and displayed on the window. Close this new window. In the TerraScan *Main Menu bar,* click *File* > *Open Block* and click inside the block in **View 1** of the Microstation workspace.

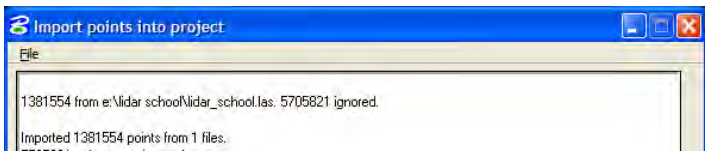


Figure 14: Data loading report

We did rough classification to low and ground points for applying different corrections to data during lab 2.We need to **transfer** all the classified points to **default class** for carrying out systematic classification. From main Terrascan window>*Classify*>*Routine*>*By class*>***From Class*** *Any class* ***To class*** *Default*>*Ok*

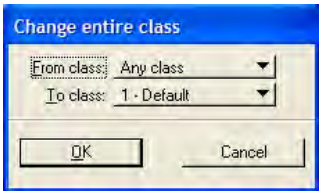


Figure 15: Transferring all the earlier classified points (low and ground class) to default class

In the TerraScan menu bar, click *View* > *Display Mode*. Click *Colors …* and in the new window click *Auto Fit*.

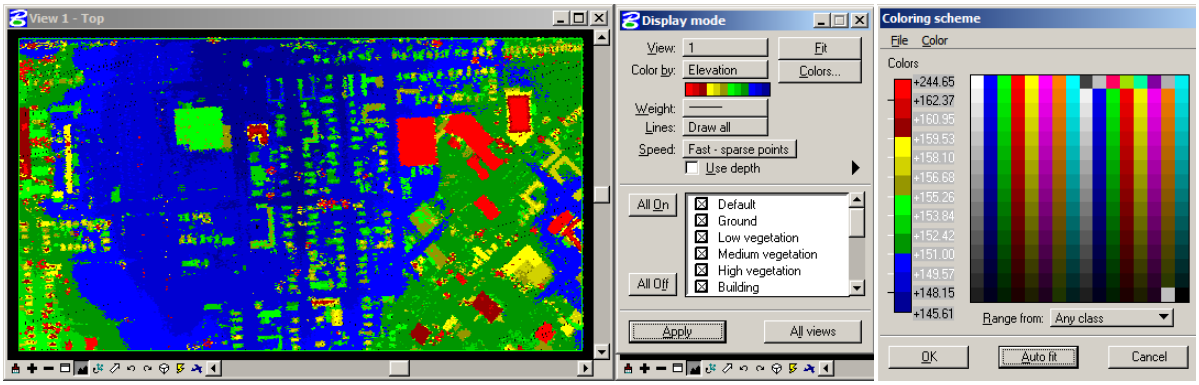


Figure 16: Points displayed based on elevation.

CLASSIFY POINTS TO BELOW GROUND CLASS

We will classify points to low points in below ground classification.

Low point by Absolute Elevation Routine

In order to classify points to below ground class, we need to *detect very low points* and *their elevation.*

Now we will try to detect very low points. Very low points can easily be tracked if we zoom to sectional view of entire area.

Start Draw section tool>*Draw a section* showing all points (depth covering entire working area)> *Display it in view 2.*

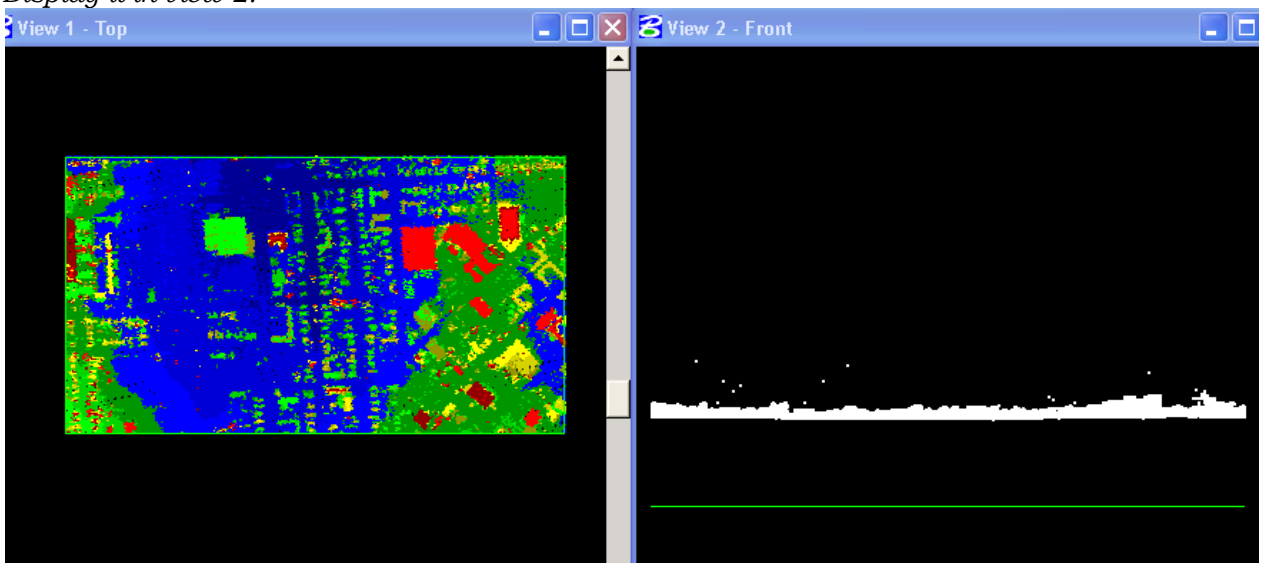


Figure 17: Finding out the low point(s) from cross section

Zoom to view2 (cross section) to find point(s) well below the ground.

Some low points are found. We need to ascertain its elevation for which we need to select Elevation in View Field.

Select *medium dialogue* from view (Terra Scan window)>Check the *option for display* from view i.e., *View>Fields..(*View field palate will appear select required option you want to display)>*Ok*

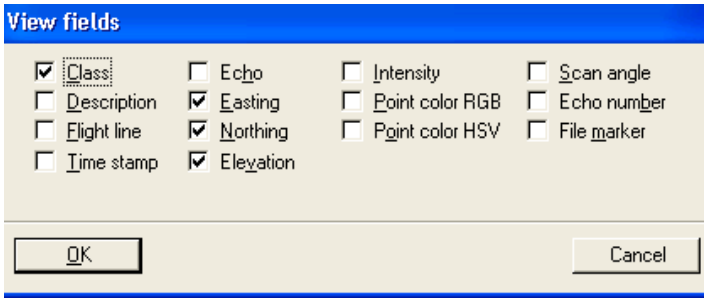


Figure 18: View fields palate

Click **Identify** and then on very low points and note its elevation, say +146.535 to 146.882 (figure 19).Select view small dialogue to return back to previous display mode.

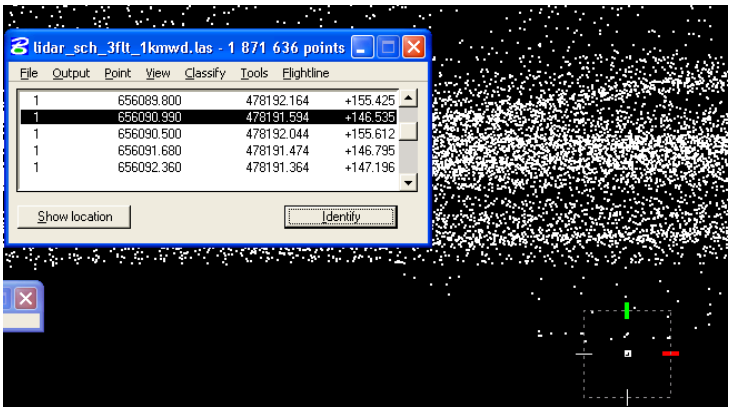


Figure 19: Determination of elevation of low points

Elevation of very low point(s) is/are determined i.e 146m. Now we will **classify low points by Absolute Elevation Routine**

*Select **Classify**>Routine>**By Absolute elevation***

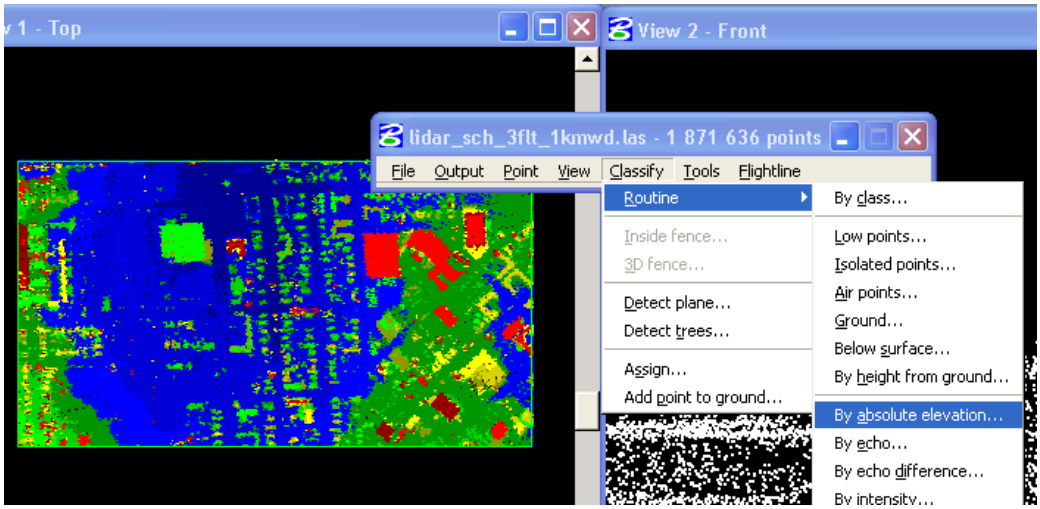


Figure 20: Selection of absolute elevation routine

***From class** any class/default **To class** Low point. **Type** -999 to 146 in Elevation field>click Ok.* (It will classify worst points well below the ground*).*

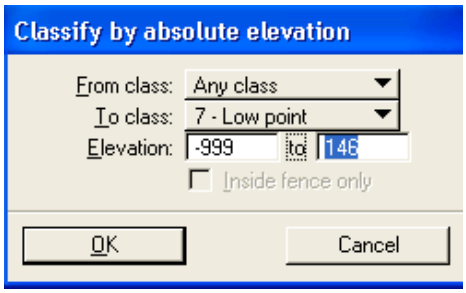


Figure 21: Feeding elevation parameter

Low point by Low Point Routine

Classification of low points can also be done using low points routine. *Select* ***Classify****>Routine>****Low points****>****From*** *default* ***To Class*** *Low points.* ***In search*** *Groups of points,* ***Type*** *3* in *Max count*, ***Type*** 1.00 in *More than* and 10.00 in within>click **Ok**

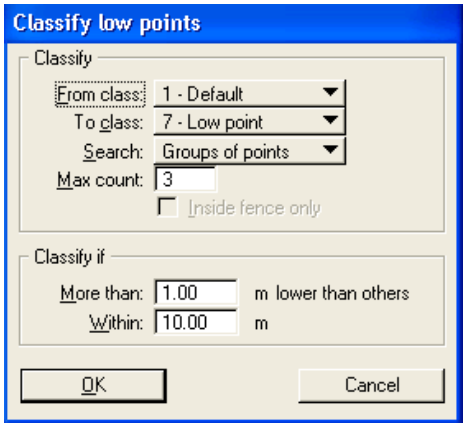


Figure 22: Classification by low point routine (group of point search)

Low point routine is required to be run more than once. ***Classify****>Routine>Low points>****From*** *default* ***To Class*** *Low points,* ***In search*** *Single point ,* ***Type*** *0.5 in More than,* ***Type*** *1.00 in More than and 7.00 in within>click* ***Ok***

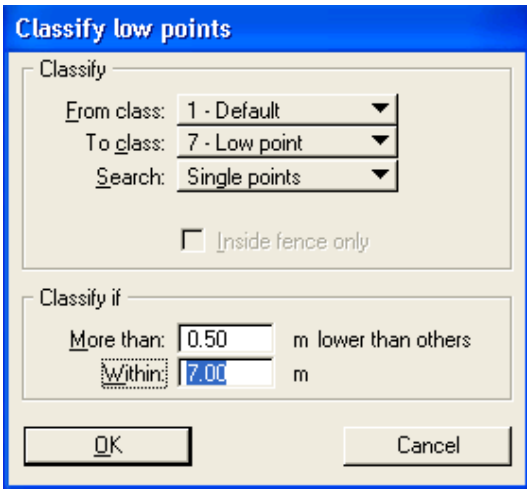


Figure 23: Classification by low point routine (single point search)

Check the performance of Low point classification: *Display mode>view* 1, **color by class**, switch off all class except low point (you may edit color, weight of low class points as well as speed for better display (using *colors>select Low point class>edit its color and wei*ght).

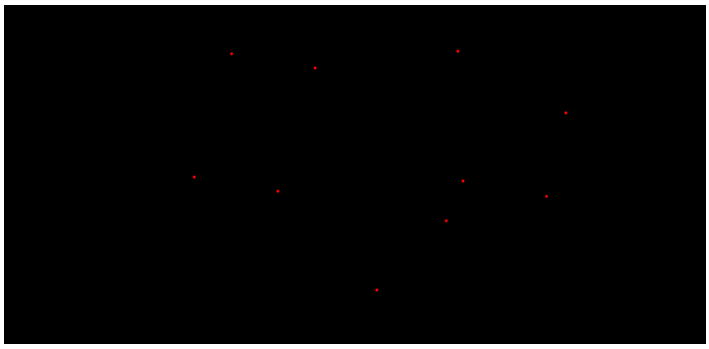


Figure 24: Check the performance of low point routine; only low point class is displayed

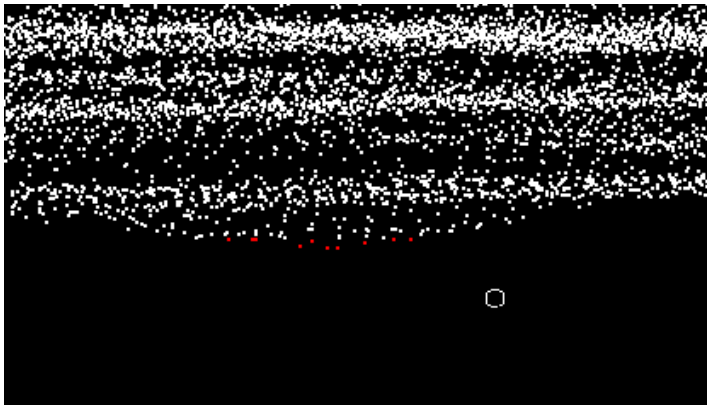


Figure 25: Cross sectional view. Low points are grouped to a separate class, marked in red

CLASSIFY POINTS IN GROUND CLASS

Here we will try to classify points approximately in ground class from default class

Approximate ground class by Ground Routine

We will carry out ground classification by routine. The ground classification routine needs parameter for largest building. We will determine this parameter from building having *largest width*.

Set view 1 to elevation coloring. ***Color*** auto fit, ***Ok>Search and zoom*** into largest building width >Use *distance measurement* tool to measure building width(it ensures at least one of the lasers strikes the ground correspond to every building block area).

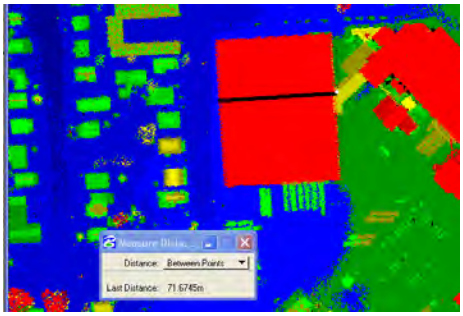


Figure 20: Determination of maximum building size

As the largest distance found to be about 71 m , we will take the parameter to be 75 m and classify ground

Classify>*Routine*>***Ground***

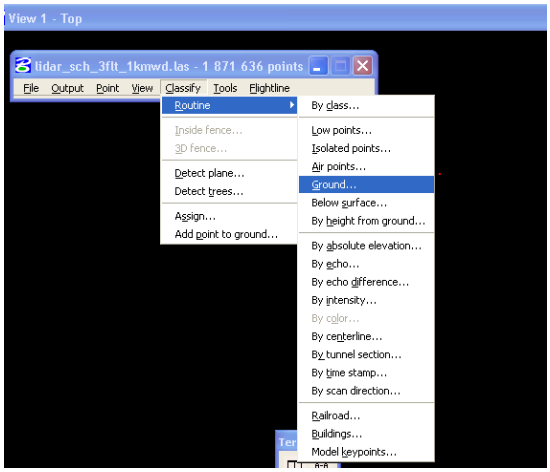


Figure 21 Starting ground classification routine

Type say 75 m in max *building size*, ***Terrain angle*** 88deg, ***Iteration angle*** 6 deg., ***Iteration distance*** 1.40m, ***Check Reduce iteration angle*** when *Edge length*<5.m >***Ok***

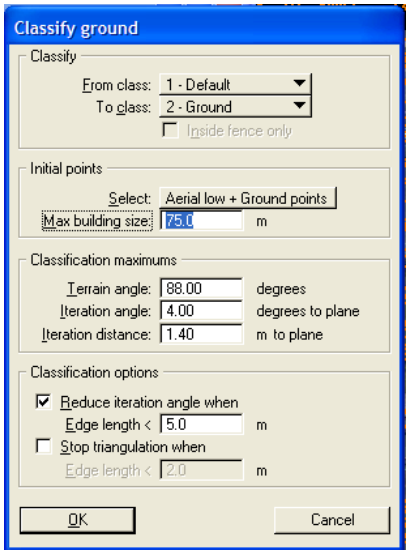


Figure 22: Feeding parameters to ground classification routine

Check the performance by viewing the ground class (Select color by class in view 1using display mode window)>Switch off all the classes except ground class>Apply

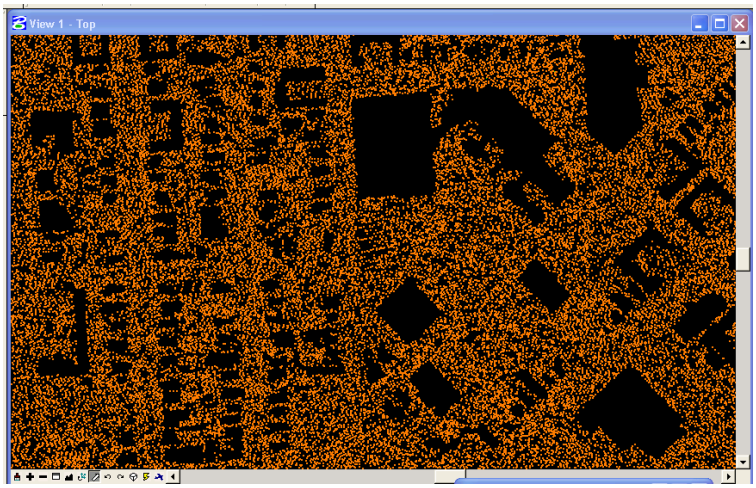


Figure 23: Checking performance of ground classification

Figure 30: Cross section of approximate ground class (in view 2)

Ground classification using above technique will not result pure ground class points. As classified ground points are mixed with some below ground points as well.

CLASSIFY POINTS IN BELOW GROUND CLASS

Low point by Below Surface Routine

The below surface points mixed in ground class will be separated here. At the beginning a temporary class will be created. Low points from this group will be classified in to temporary class using different trial parameters. Finally low points will be separated from approximate ground class using best parameter. And temporary class points are transferred to low class. (one can also classify low points directly from ground points).

Creation of additional class (**Temporary**), which is not listed

Display mode>*Colors*>*Add*>*Insert* the description (name of class), select the color and weight for representation>*Ok*

Now we will classify low points of ground class to temporary class using *below surface routine*

Classify>*Routine*>***Below Surface***>***From class*** *Ground* ***To class*** *Temporary,* ***Type*** *10 in limit*>***Ok***

Figure 31: Feeding below surface classification parameter

To view results: Switch to ***View***>***Display mode***>*Switch off* all other Classes except Temporary;(*To see all the point-Select , Normal-All point in Speed>Apply*)>***Switch to*** *class coloring*>***Draw sections*** to evaluate results.

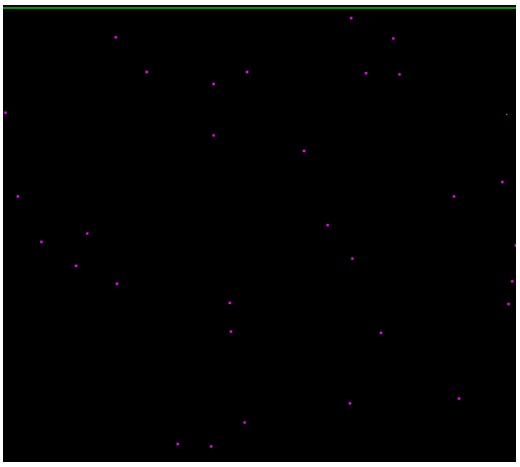


Figure 32: Displaying below surface classified point from ground class

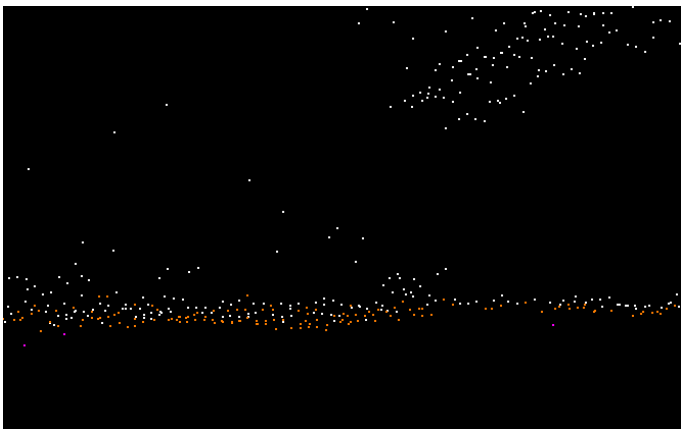


Figure 33: Checking the performance of below surface point classification in cross section

We will have to test above classification with other parameter values as well, to select best parameter. In order to do that we need to undo below surface classification and then classify it with different parameter.

Select point*>*Undo below surface

(Note: Above technique of undoing is very useful to revert back to previous stages at the time of classification**)**

Classify*>*Routine*>*Below Surface*>*From Class *Ground* ***To class*** *Temporary,* ***Type*** *6 in limit->Ok*

To view results: Switch to *View*>***Display mode***>*Switch off* all other Classes except Temporary; (To see all the point-Select , Normal-All point in Speed>Apply)>***Switch to*** *class coloring*>***Draw sections*** to evaluate results.

Similarly we will test with few other limiting values before finally deciding the suitable parameter. Let us say 4 gives us the best results. Set it to 4 and classify low points from ground class using above parameter. After classification of points to temporary class we need to *Transfer Temporary class points* in to Low point class

Classify*>*Routine*>*from class *Temporary T****o class*** *Low point*>**Ok** (Entire class will be shifted to low point class.

CLASSIFY POINTS IN BELOW GROUND AND GROUND CLASS USING MACRO

Select Tools>Macro>Type macro detail (say) Abs_low_ground_below surface in description>add>Classify points (action), Routine (By class)-By Absolute Elevation>Ok;(set range of elevation); >Ok.

Add>Classify points(action),Routine (By class)>Low points>Ok; From class Default To class low point; Search Groups of points; Max count 3; Type 1(More than), 10 (within)>Ok.

Add>Classify points (action), Routine (By class)>Low points>Ok; From class Default To class low point; Search single point; Type 0.5(More than), 7 (within)>Ok.

Add>Classify points (action), Routine (By class)>Ground>Ok; From class Default To class Ground; select settings i.e. Building size size(say 115),Terrain angle, Iteration angle, Iteration distance>Ok

Add>Classify points(action), Routine (By class)>Ground>Ok; From class Ground To class low point; Set settings>ok.

Save as with a name i.e Abs_low_grnd_blosurface.mac>Ok>Close macro window

We can directly run the macro from Tool>Macro>File-Browse the macro you have created >Select it>Run it on loaded points (In case we have block(s) and you want to run it on specific block we will do the following. Start define project tool>Select rows of blocks >Select Tools>Run macro (run macros on blocks)>Brows and select Abs_low_grnd_blosurface.mac>Type (say) 70m neighbours>Click save points at the end>Ok

(We can check that the number of points classified. We can Save the report as .txt file>Close report>Close project window).

CORRECTION TO APPROXIMATE GROUND CLASS POINTS

Ground class points may not still be very accurate. Many non ground points may be shown as ground point and vice versa. This finer classification needs interactive classification using ortho photo and shaded surface.

Interactive Classification using Terra Photo

Set only ground class open in view 1 (using color by class in display mode and switch off all class except ground; Normal all point in speed>apply>Ground class data points will open up)>Now errors in ground classification may be visible if we **zoom into** building places and draw few **cross section** over the area . We will now use ortho photo to reduce error points.

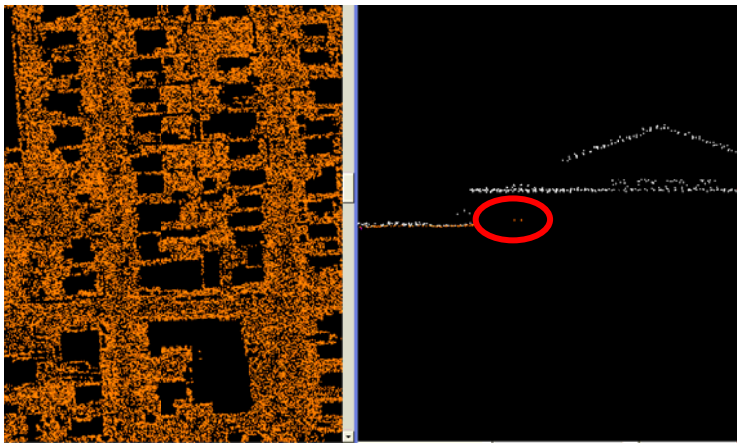


Figure 34: Organize view 1 and 2 to track error points

Displaying ortho photo using terra photo

Organize view 1 and 2 to have same size. *Utilities>MDL Application>**Select TPhoto**>Load>**Start Manage Raster References tool*** from TerraPhoto viewer>*Select file>Attach file* (browse and select your ortho photo i.e. **resampled.ecw**)>Add>Done

Figure 35: Terra Photo viewer

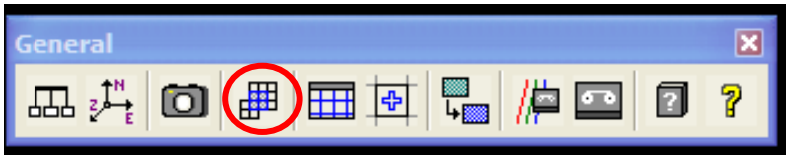


Figure 36: Manage Raster Reference Tool (4th from left)

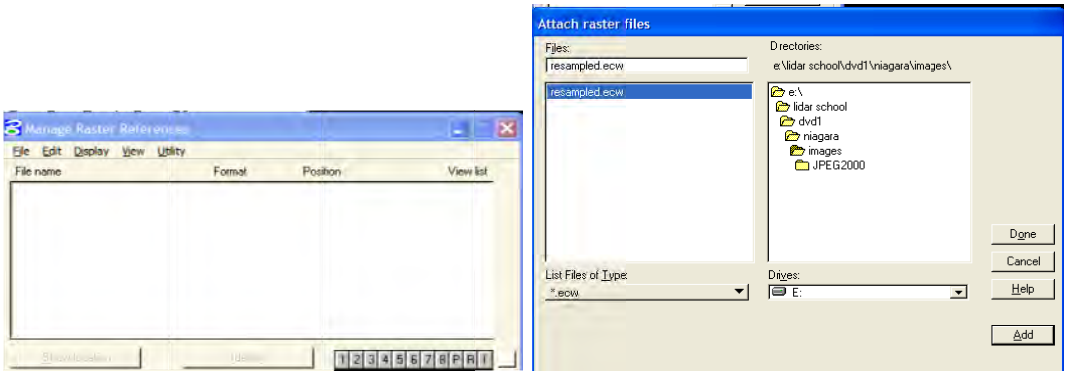


Figure 37: Browse and select your ortho photo file

Reference visibility window will appear; click all off , *Set view 2 on >Ok;*

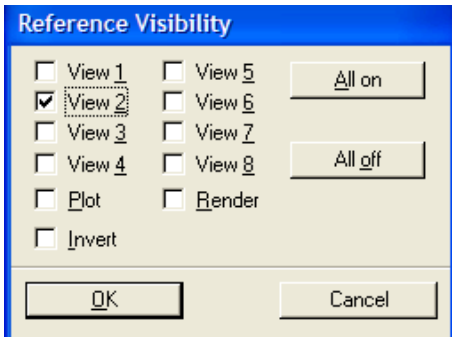


Figure 24: Reference visibility setting window

Close terra photo tool box>***Set view 2*** *to* ***display*** *ground point only* (using Terra Scan display mode options)>***Start synchronize*** *views tool* (using Terra Model Tool –Synchronize view option) >**set view 2** to ***match*** view 1>

Figure 25: Synchronize view tool (topmost right hand colomn)

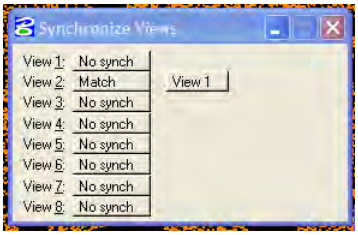


Figure 40: Synchronize views

Update view 1, view 2 will now display the same area .*Ortho photo will now be visible* at the back of view 2 .

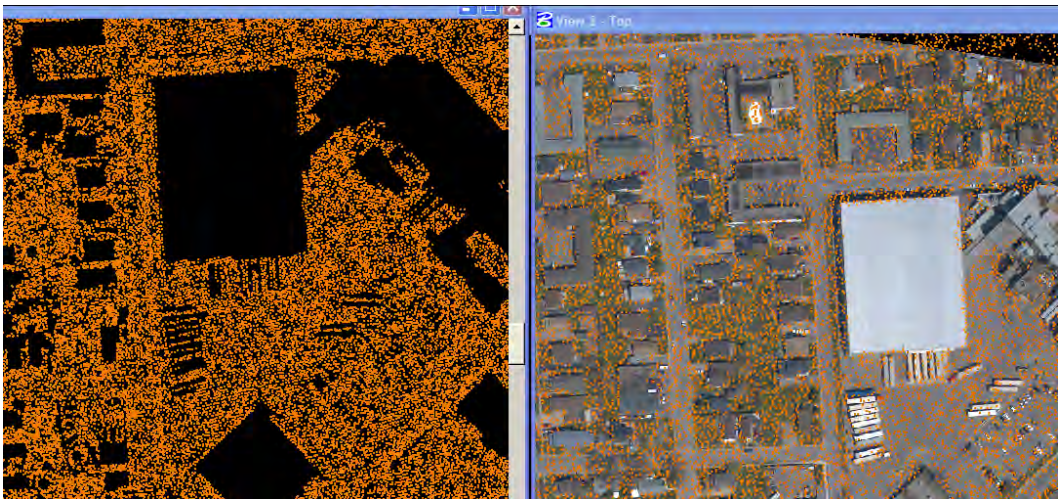


Figure 41: Ortho photo is displayed at the back of view 2

Search and zoom in to problematic points. Error points (i.e. wrongly classified as ground) can easily be found out at building edges and building holes.

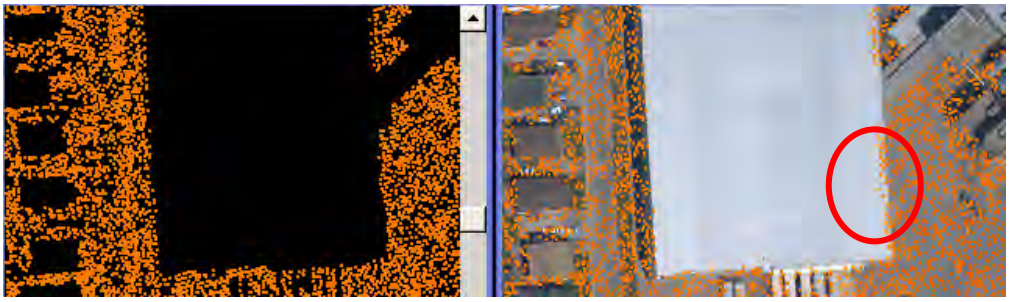


Figure 42: Error points in ground can easily be located

In order to rectify these errors we need to classify them properly to their respective classes.

Rectify the error points by **classification**

Tear out edit laser tool box

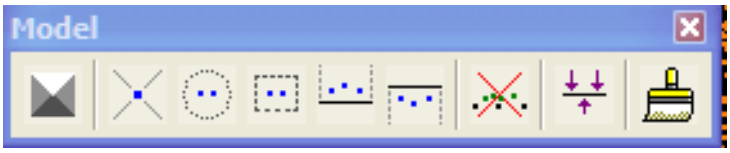


Figure 43: Edit laser tool box

There are different ways to carry out interactive classification using above tool box such as Assign point class, Classify using brush, Classify fence etc. One can use many of these tools to classify error points.

Assign point class (2nd icon from left in figure 43)

Assign Point class classification>***From*** any visible point, ***To class*** say low point/building/water etc; select closest within 2 m

Classify using brush (3rd icon from left in figure 43)> ***From class*** Ground ***To class*** low point /building/water etc; > apply it at different places as per your judgment.

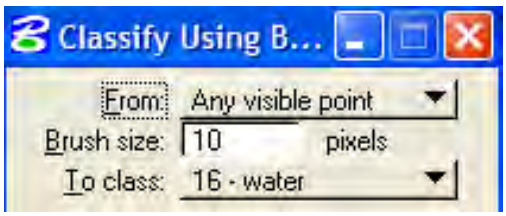


Figure 44: Classify using brush parameter setting

Classify fence(4th icon from left in figure 43)

Classify Fence classify points inside a polygon we draw> ***From class*** Ground ***To class*** low point /building/water etc;> apply it at different places as per your judgment. You may need to draw a few sections to achieve this.

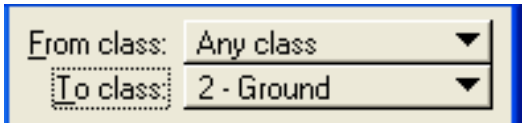


Figure 26: Classify fence parameter setting

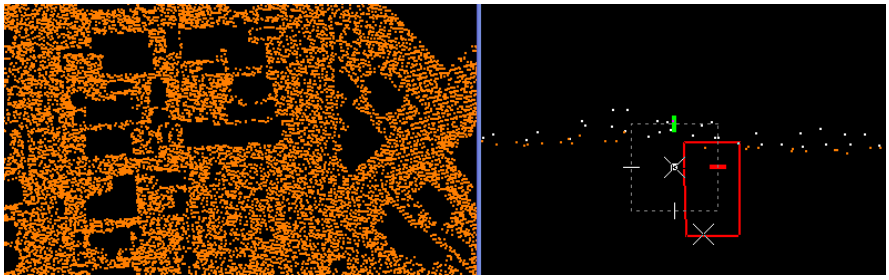


Figure 27: Classify fence; fence is drawn using polygon and then classified

Classify below line (6th icon from left in figure 43)

Classify below line, classifies points below a line in a section view (one may use these tools to classify point inside building> ***From class*** Ground ***To class*** low point/water etc.

Repeat different classification scheme till we are happy with classification.

Classify inside fence

Classification can also be restricted inside any drawn fence. To do that ***Place polygons*** along the approximate line you desire>***Select*** resulting polygons using selection tool>***select classify*** >***Inside fence*** (multiple selected can be classified at a time)>***From class***-any class ***to class*** say water/low point etc>using display option we can ***check the performance*** of this classification.

Interactive Classification using Terra Model

We will create triangulated ground model and Visualize editable ground models as shaded surface. This surface model will be used to improve ground points further.

Start ***create editable model tool***>select ground class only

Figure 28: Create editable tool (first left)

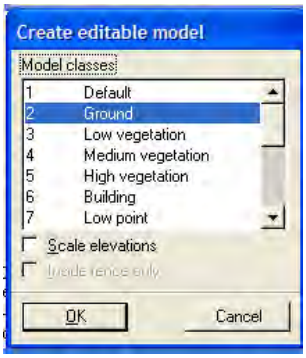


Figure 29: Ground class point to be used for modeling

Start display shaded surface tool (6th icon from left in display surface tools) >***Type*** Surface information Name say ClassLiDAR>make sure only view 1 is on>Ok>*Switch off all classes in view 1* (we are also generating **BEM** of the area in this procedure)

Figure 30: Display surface tools

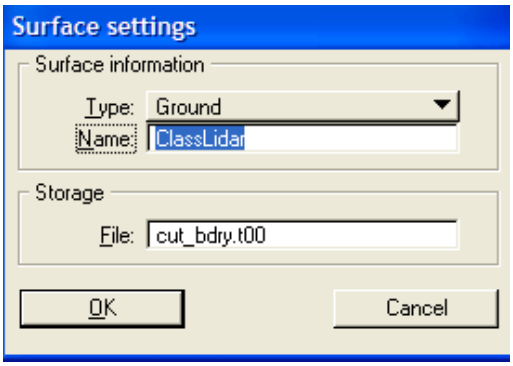


Figure 50: Writing surface setting

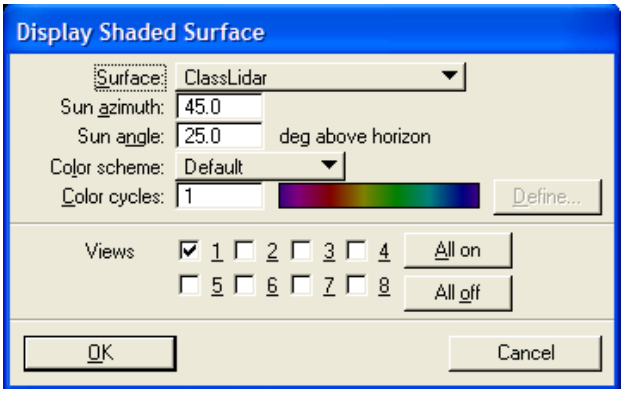


Figure 51: Display shaded surface parameter setting

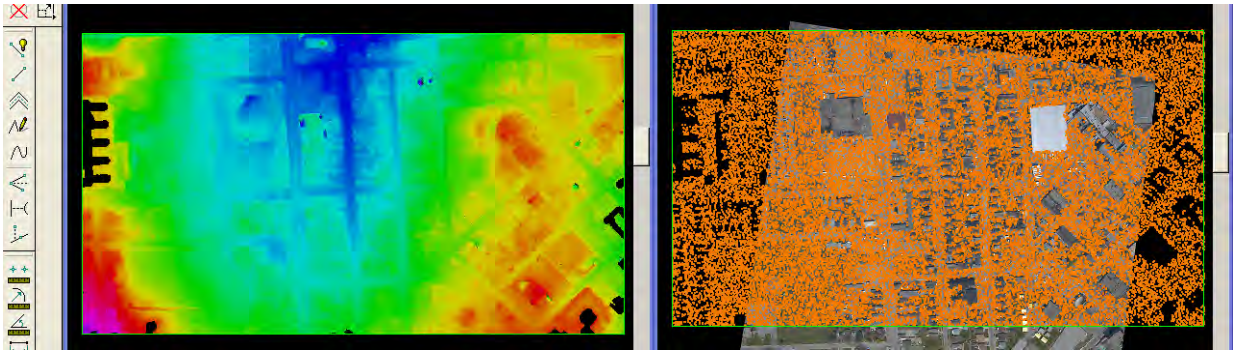


Figure 52: Shaded surface is displayed in view1 and ground class with ortho photo is displayed in view 2

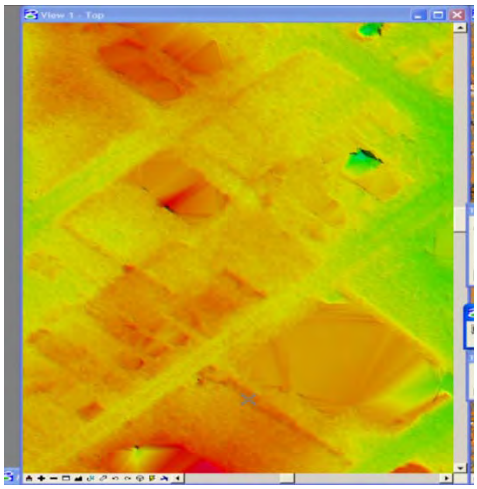


Figure 53: Shaded surface is effectively the BEM of the area

Now we will use shaded surface to rectify our ground class further. In BEM (or shaded surface model) strange looking places are easily visible. This might involve erroneous point. We need to zoom inside such places and rectify such points.

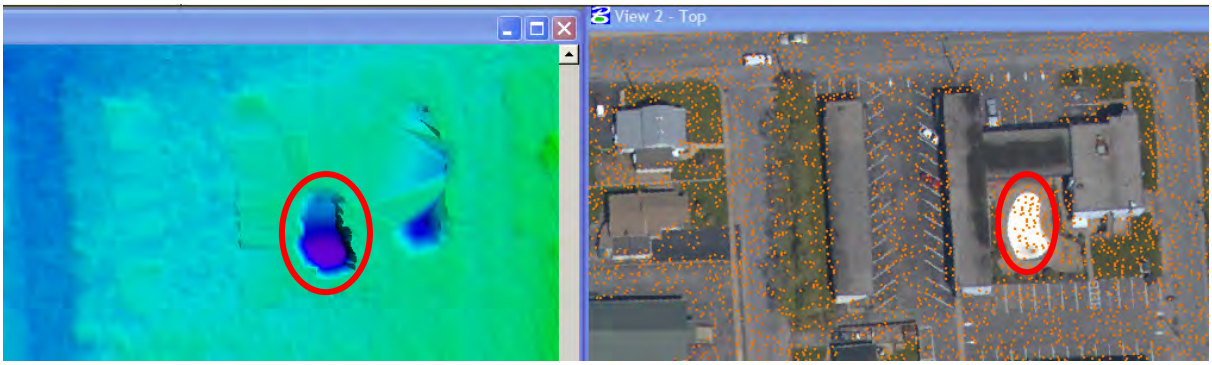


Figure 54: Strange appearing shades in BEM generated as non ground point is classified as ground

Search *strange looking places in model*>**classify** it if required (use different viewing windows and sections with interactive tools i.e. *Classify Point, Classify by Brush* for classification)>We need to continue classification until we are happy> when complete>Select> file save points.

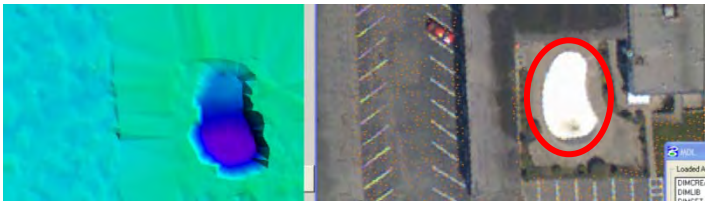


Figure 31: Error points in ground is classified to water class using assign point class classification tool

CLASSIFY ABOVE SURFACE POINTS

Now we will try to classify above surface points.

Classify points in Building class using Building Routine

We will classify points to building class using *Building Routine*

Classify>*Routine*>***From class*** *Default class* ***To class*** *Building; Provide the minimum building size say 25 sqm* >***ok***

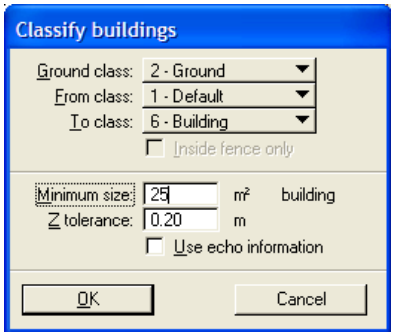


Figure 56: Classify buildings routine

Buildings routine classifies building points which form some kind of a planar surface. This routine requires that you have classified ground points before. After building classification we can *check the performance* of building classification by displaying only buildings class points over ortho photo in view 2 (by switching off all the class except buildings using display mode options) and/or using a sectional view of building class.

Figure 32: Classified building points displayed over ortho photo

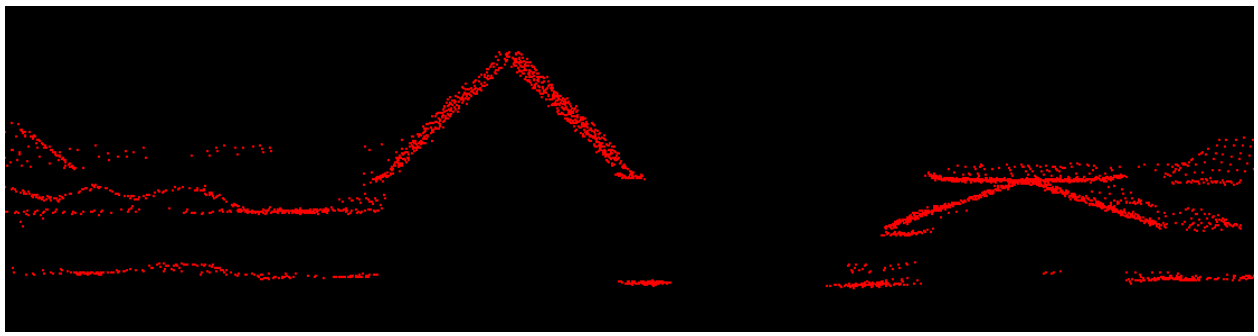


Figure 33: Classified building points in sectional view

Classifying Low, Medium and High Vegetation using Height from Ground Routine in Macro

We will classify different vegetation points using Macro constructed with height from ground routine.

Select tool from *Terrascan main window*>***macro***>***Type vegetation*** *in description*

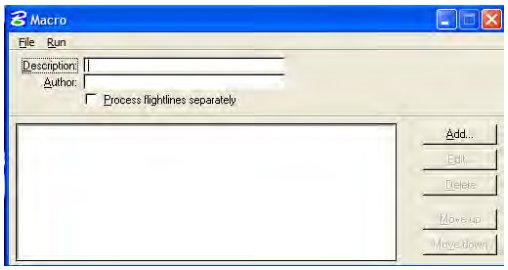


Figure 34: Defining a macro

Select add>Inside Macro step select action-classify points, Routine –By class>ok>***classify by class***, F***rom class*** *Default—****To class*** *Low vegetation >Ok>****Add***>action ***classify point***, *Routine* ***By height from ground***>*Ok>****From class*** *Low vegetation* ***To class*** Medium vegetation, *min height 0.25 , max height 999 Ok>****Add***>action ***classify point***, *Routine* ***By height from ground***>*Ok>****From class*** *Medium vegetation* ***To class*** *High vegetation, min ht 2 , max height 999*

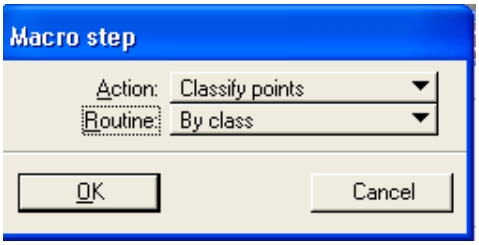


Figure 60: Defining macro steps

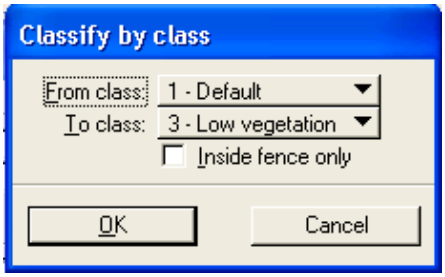


Figure 61: Classify entire default class to low vegetation using classify by class

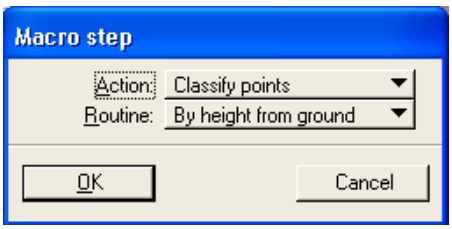


Figure 62: Classify low vegetation points to vegetations of different heights using by height from ground routine

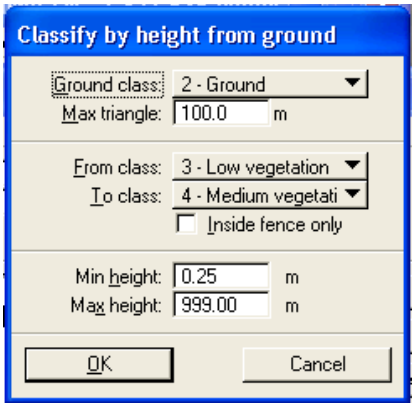


Figure 63: Setting parameters in classify by height from ground for low to medium vegetation

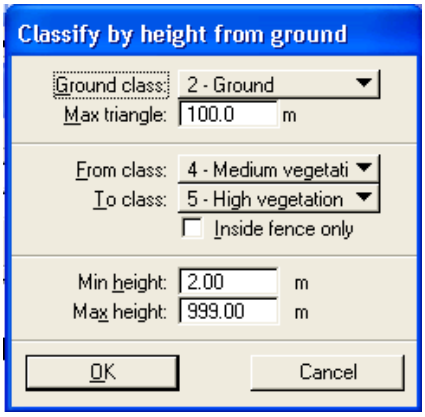


Figure 64: Setting parameters in classify by height from ground for medium to high vegetation

Save the file as ***vegetation. Mac*** macro.

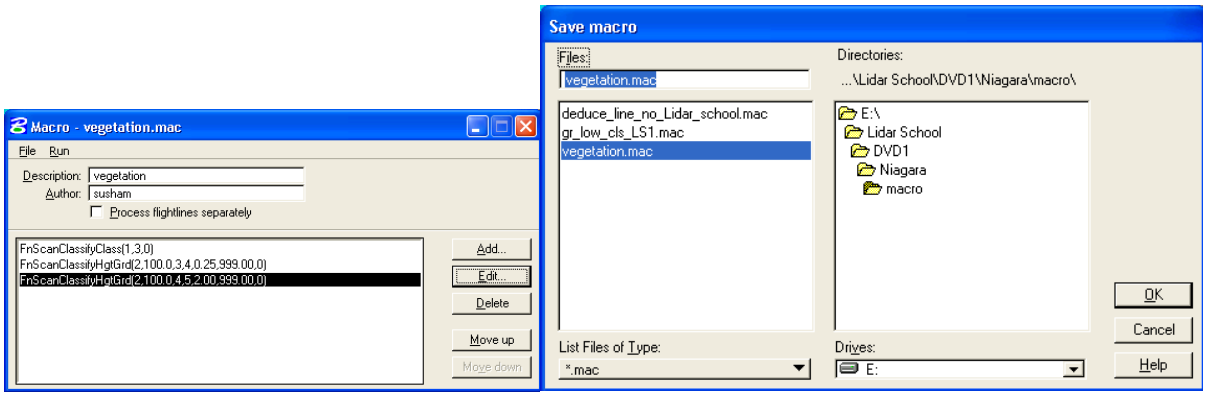


Figure 65: Saving macro as vegetation.mac

Now we will run the macro ***vegetation.mac*** from tool. (it can also be run from macro window itself if it is not been closed)

*Tool>**run macro**>browse vegetation.mac> run it on loaded points >ok*

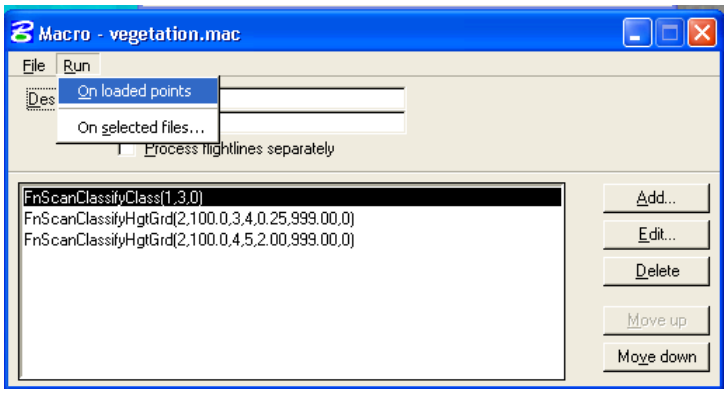


Figure 66: Runing a macro on loaded points

After running the macro it will generate a report. The report should be consistent with the data. So we can Check report and then close it.

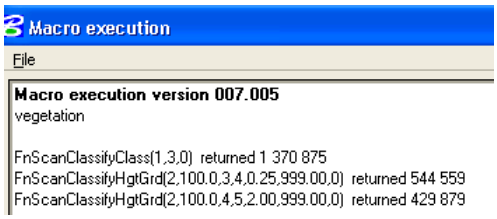


Figure 67: Macro execution report

We can now check the performance of vegetation point classification. A section can be drawn across the terrain for this purpose.

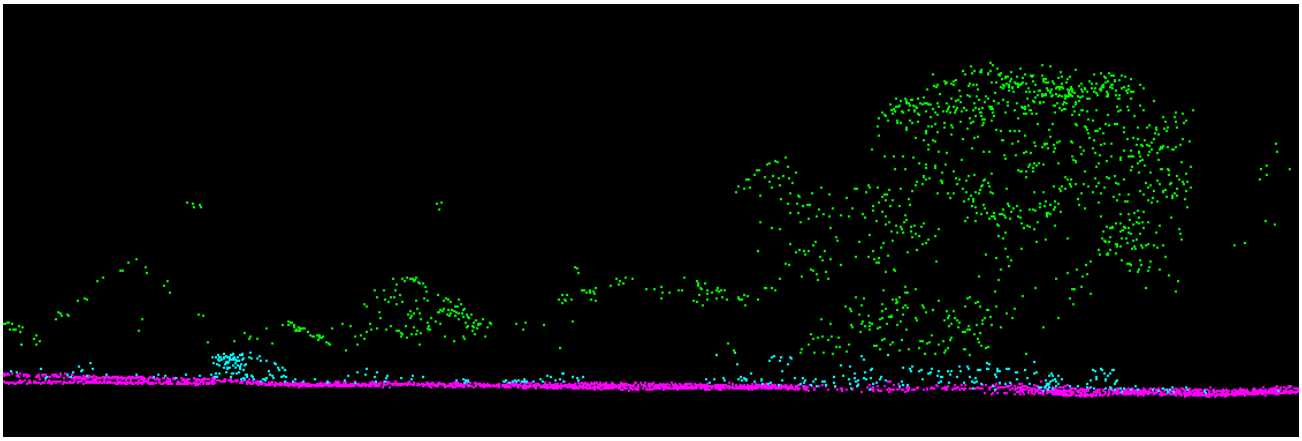


Figure 68: Sectional view of classified vegetation points i.e. low (pink),medium(sky blue),high vegetation(in green color)

Majority of over below ground, ground and over ground points have already been classified. We can check performance of these classifications using a section.

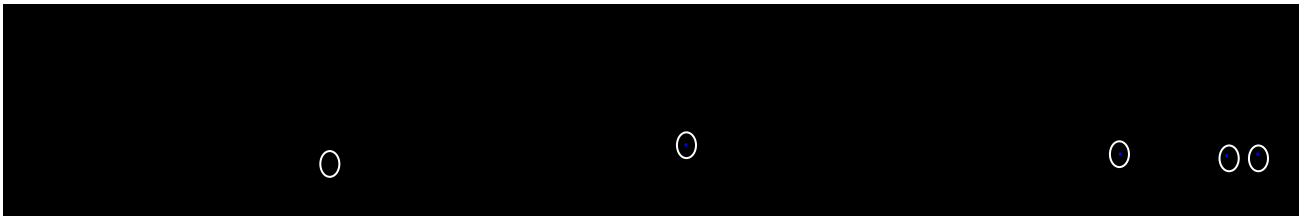


Figure 69: Classified low points in a section

Figure 70: Classified ground points in a section

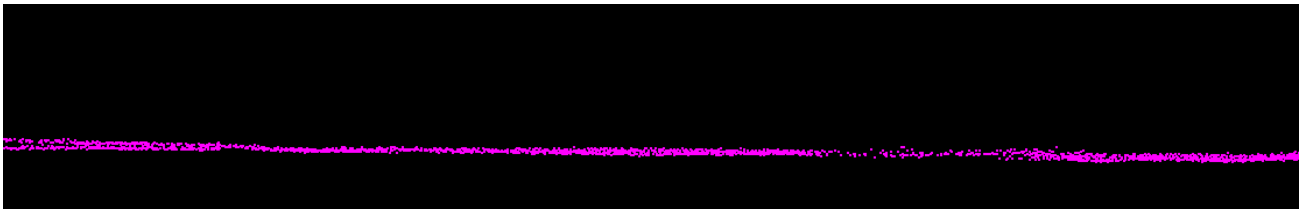


Figure 71: Classified low vegetation points in a section

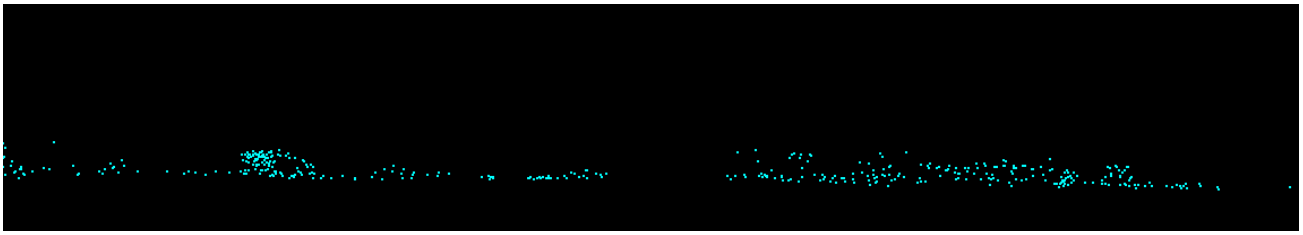


Figure 72: Classified medium vegetation points in a section

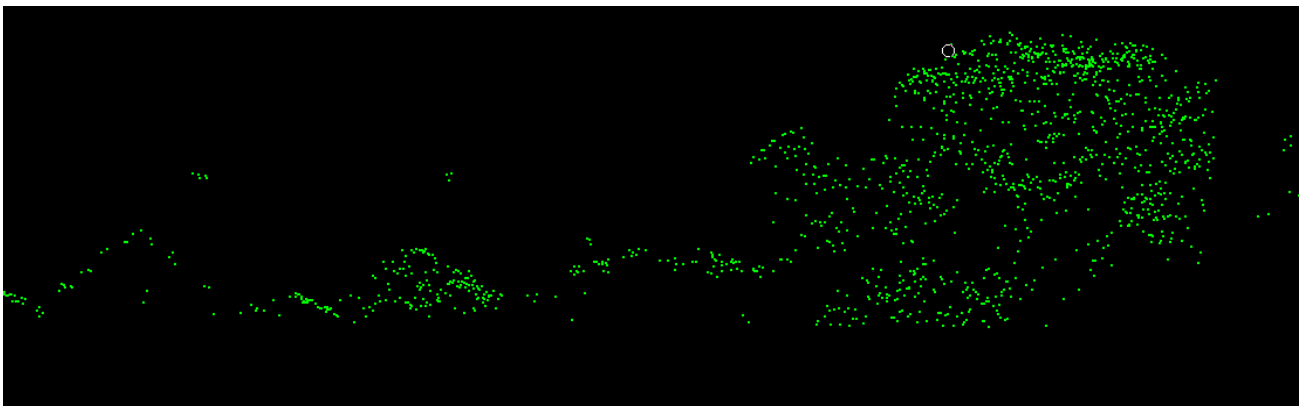


Figure 73: Classified high vegetation points in a section

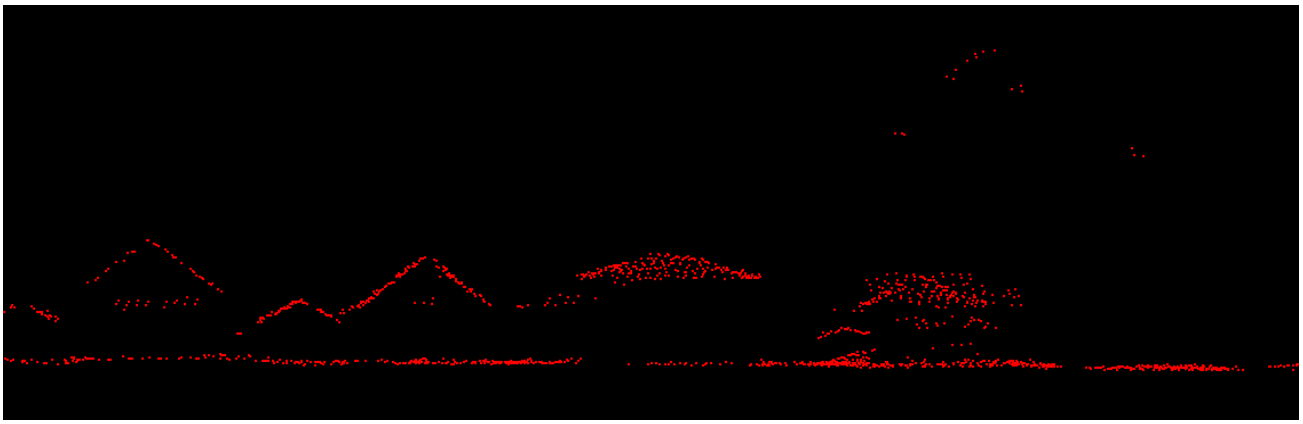


Figure 74: Classified building points in a section

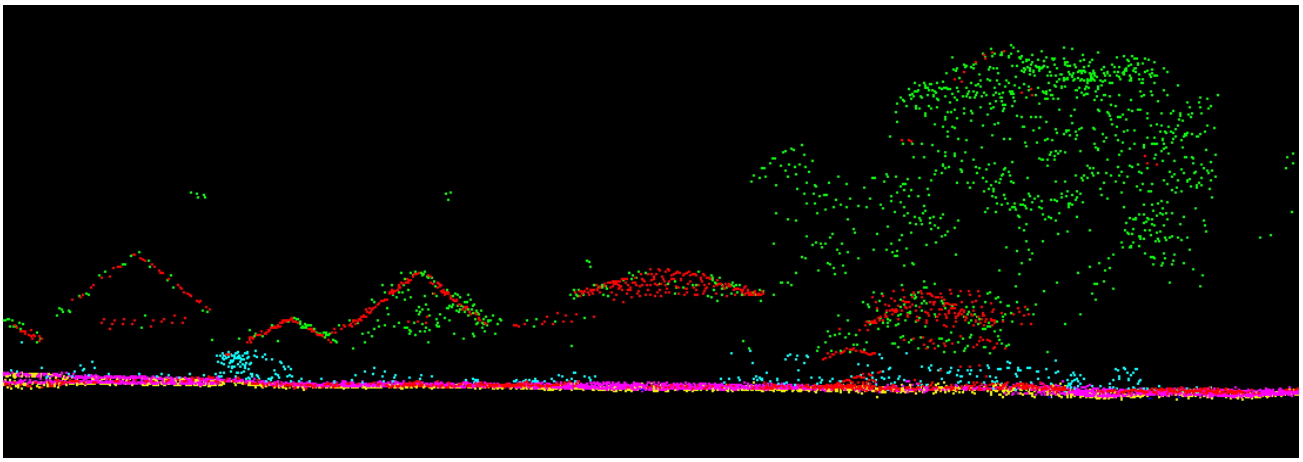


Figure 355: Classified low point, ground, low vegetation, medium vegetation, high vegetation and building points in a section

Vegetation class data points may not be giving us the pure vegetation class it would have data points from utilities such as tower, overhead wire etc along with few building points which could not be classified (particularly, gable roofs) properly during building classification. Hence we can classify further the high vegetation to isolate possible power or transmission line points.

Transmission line points detection

Isolated points in high vegetation may belong to transmission (power) line. We will run Isolated Points routine over high vegetation to determine it.

Classify>Routine>**Isolated Points..**

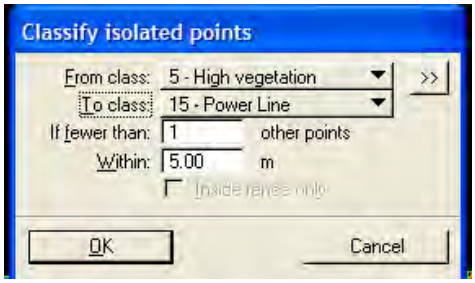


Figure 76: Runing isolated points routine over high vegetation class

Since we do not have power line class in default list we have to create it (from Display mode>colors>add..).

We can check the performance by displaying only power line points using display option

Figure 77: Power line points are detected and displayed over ortho photo in view 2

VIEWING PROJECT STATISTICS

We can now evaluate statistically the performance of entire classification using tool as below.

Terra scan main window> ***Tool*** >***show statistics***

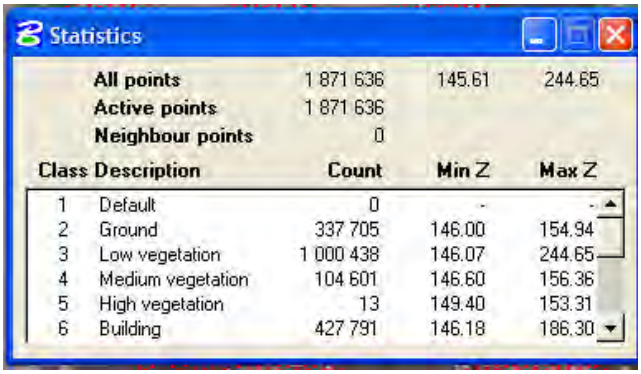


Figure 78: Statistics of classification

Saving classified data points

We need to save all the classified points to use it for future use such as Building modeling (to be covered in Laboratory 4)

Terra Scan main window>*File*>***Save points As***.. ***lidar_school_classified***

After saving your data points you can close data points, displayed surface, ortho photo etc.

FURTHER READING

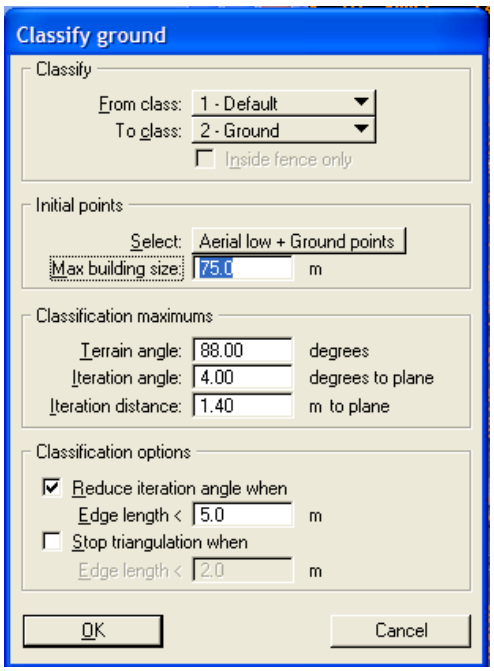
Ground

Ground routine classifies ground points by iteratively building a triangulated surface model. The routine starts by selecting some local low points as sure hits on the ground. You control initial point selection with the **Max building size** parameter. If maximum building size is 60.0 m, the application can assume that any 60 by 60 m area will have at least one hit on the ground (provided there are points around different parts of the area) and that the lowest point is a ground hit.

The routine builds an initial model from selected low points. Triangles in this initial model are mostly below the ground with only the vertices touching ground. The routine then starts molding the model upwards by iteratively adding new laser points to it. Each added point makes the model follow ground surface more closely.

Iteration parameters determine how close a point must be to a triangle plane so that the point can be accepted to the model. **Iteration angle** is the maximum angle between point, its projection on triangle plane and closest triangle vertex. **Iteration distance** parameter makes sure that the iteration does not make big jumps upwards when triangles are large. This helps to keep low buildings out of the model.

The smaller the **Iteration angle**, the less eager the routine is to follow changes in the point cloud (small undulations in terrain or hits on low vegetation). Use a small angle (close to 4.0) in flat terrain and a bigger angle (close to 10.0) in hilly terrain.



Setting:	Effect:
From class	Source class from which to classify points.
To class	Target class to put points into.
Select	Selection of initial ground points. Use Current ground **points when you want to continue iteration in a fenced,** previously classified area.
Max building size	Size of largest buildings.
Terrain angle	Steepest allowed slope in ground terrain.
Iteration angle	Maximum angle between point, its projection on

	triangle plane and closest triangle vertex. Normally between 4.0 and 10.0 degrees.
Iteration distance	Maximum distance from point to triangle plane during iteration. Normally between 0.5 and 1.5 m.
Reduce iteration angle when	If on, reduce eagerness to add new points to ground inside a triangle when every edge of triangle is shorter than Edge **length. Helps to avoid adding unnecessary point density** into ground model and reduces memory requirement.
Stop triangulation when	If on, quit processing a triangle when every edge in triangle is shorter than Edge length. Helps to avoid adding unnecessary point density into ground model and reduces memory requirement.

Building Classification

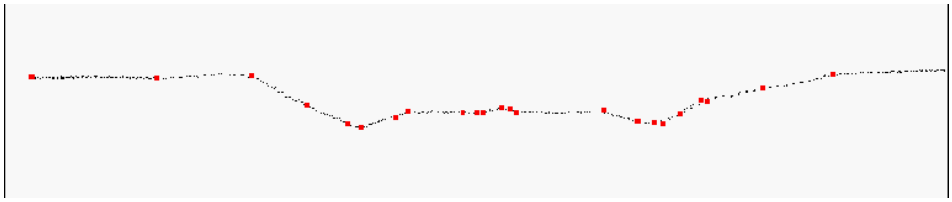
Parameters

Setting:	Effect:
Ground class	Class into which ground points have been classified before.
From class	Source class from which to search building points.
To class	Target class to put points into.
Minimum size	Smallest building plane size as square meters.
Maximum angle	Steepest roof angle.
Elevation tolerance	Elevation accuracy of laser points.
Maximum gap	Maximum empty gap between points belonging to same roof. Should be smaller than smallest gap between adjacent buildings.

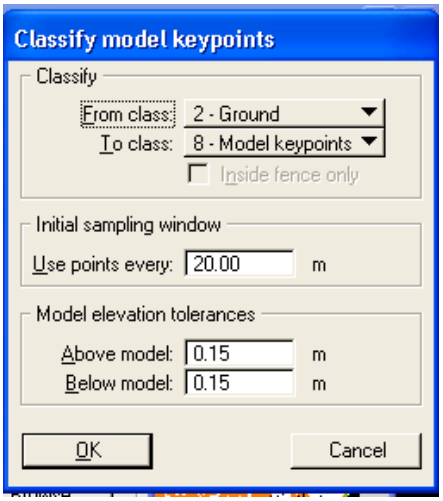
Model keypoints

Model keypoints routine classifies laser points which are needed to create a triangulated surface model of a given accuracy. This routine is normally used to create a thinned data set from points classified as ground hits.

You control the accuracy with elevation tolerance settings **Above model** and **Below model**. These settings determine the maximum allowed elevation differences from a ground laser point to a triangulated model. **Above model** determines how much laser points can be above the model. **Below model** determines how much laser points can be below the model. The application will try to find a relatively small set of points (= keypoints) which would create a triangulated model of given accuracy. These keypoints will be classified into another class.



All other ground points are within the given elevation differences from a model that the keypoints would produce when triangulated. Some of the ground points are above the model, some ground points are below. This classification is an iterative process similar to ground classification. The process starts by searching for initial points inside rectangular regions of a given size. The lowest and the highest source point inside each rectangle is classified as keypoint and those are used to create an initial triangulated model. During each iteration loop the routine searches for source points which are too far above or below the current model. If such points are found, the furthest points are classified and added to the model. The **Use points every** setting provides a method for ensuring a minimum point density in the final model even in flat places. For example, if you want to have at least a point every 10 meters, you should set the **Use points every** setting to 10.0.



Setting:	Effect:
From class	Source class from which to search keypoints.
To class	Target class to put keypoints into.
Use points every	Rectangle size for searching initial points. The highest and the lowest point inside each rectangle will be classified.
Above model	Maximum allowed elevation difference to the triangulated model from a source point above the model.
Below model	Maximum allowed elevation difference to the triangulated model from a source point below the model.

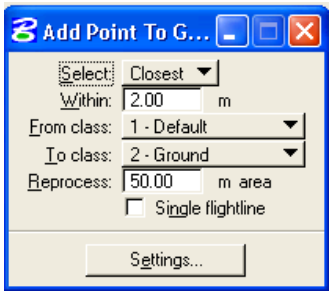
Intensity Routine Classification

By intensity routine classifies points which have an intensity (strength of return) value within a given range. The type of surface material hit by the laser beam affects the intensity. Metal, white paint and green leaves typically give high intensity returns. This can be used to classify points which are possible hits on railroad rails; the metal surface gives a high intensity return and the surroundings often give a low intensity return.

Adding points to ground

It is generally advisable to run **Ground** classification using an **Iteration angle** which is too small rather than too big. It is easier to add points to ground than to remove points from it. Ground terrain may have some small formations which you want to reprocess locally. One way

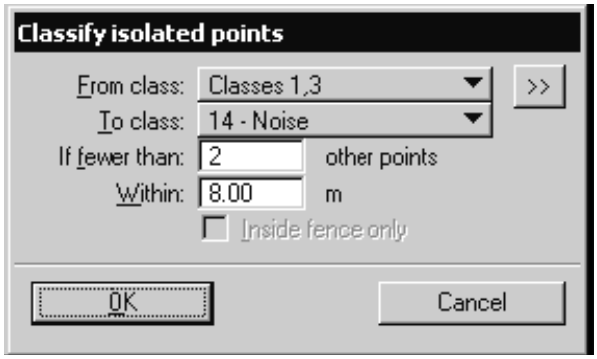
to do this is using **Add point to round** menu command. It lets you identify an unclassified point which belongs to ground, adds that to ground model and reprocesses the surroundings.



Setting:	Effect:
Select	Which point to choose around mouse click: • Closest – Point closest to mouse click. • Highest – Highest point within search circle. •Lowest – Lowest point within search circle.
Within	Search radius around mouse click.
From class	Source class from which to take a point.
To class	Ground class to add point into.
Reprocess	Size of the area to reprocess around the added point.

Isolated points

Isolated points routine classifies points which do not have very many other points within a 3D search radius. This routine is useful for finding isolated points up in the air or below the ground. When possibly classifying one point, this routine will find how many neighbouring points there are within a gived 3D search radius. It will classify the point if it does not have enough neighbours.



Setting:	Effect:
From class	Source class or classes from which to classify points.
To class	Target class to put points into.
If fewer than	Number of other points that must be found within a search radius for the point not to be classified. Normally 1 - 5.
Within	3D search radius. Normally 2.0 - 10.0 m.

LABORATORY NUMBER 4: BUILDING MODELS AND CONTOURS

WHAT WOULD WE LEARN IN THIS LABORATORY

In this laboratory, we would learn to do the following

- Generation and correction of building models using aerial ortho photos
- Generation of contours

RECAPITULATION

In the last laboratory, we covered a detailed classification of low points, ground, vegetation and buildings. The process of classification was based on various aspects of the given data as well as some background knowledge about LiDAR data. In this exercise, we will take the classified data as an input for further processing and designing building models and contours.

CREATING BUILDING MODELS

We will now proceed to create Building models from the given data. To do this, we will take the help of a georectified image. For saving space, this image has been converted to the ECW format.

We have been given the file **cut_bdry.dgn**. This file contains the boundary of the LiDAR data file within which the laser data would be loaded. Make a copy of this file and open it using Microstation.

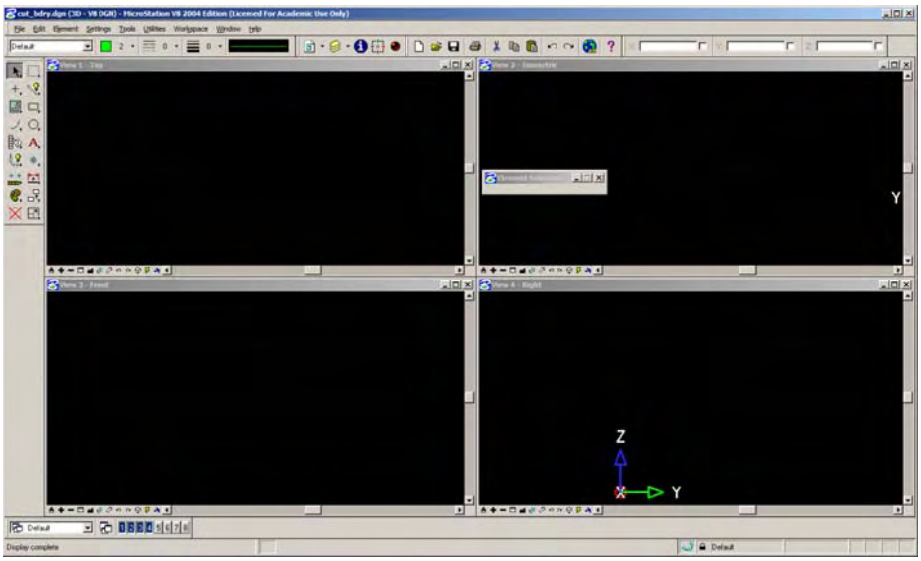


Figure 1: File opened in Microstation

However, the boundary which has been digitized inside this design file is not visible in the window. Thus, in view 1, click the fifth icon in the **view toolbar** which will fit the digitized design to the view.

Figure 2: View Toolbar

For the purpose of this exercise, we would use TerraScan (TSCAN) and TerraPhoto (TPHOTO) and we would load this through *Tools* > *MDL Applications* in the Microstation menu.

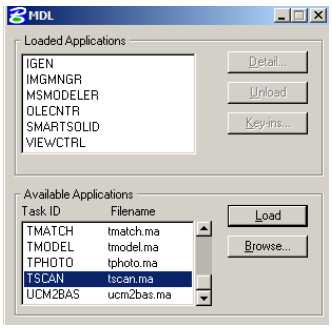


Figure 3: Loading TerraScan and TerraPhoto using Utilities > MDL Applications

The toolboxes for TSCAN and TPHOTO would be loaded in the Microstation environment. Further, the main toolbox for both the application would be also loaded.

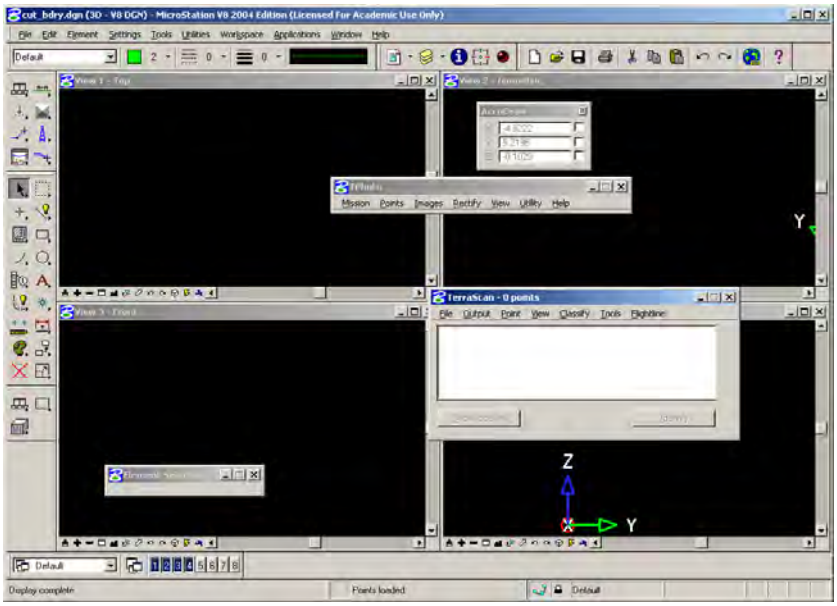


Figure 4: Toolboxes and main windows loaded in the workspace

We will now create a new project using the Terrascan toolbox. To do this, click and drag the first icon on the Terrascan toolbox. A new toolbar would emerge. This toolbar can also be accessed using *Applications* > *TerraScan* > *General*.

Figure 5: General Toolbar in the TerraScan application

The fourth icon in this toolbar manages projects. Click on this icon to open the **Define Project** window.

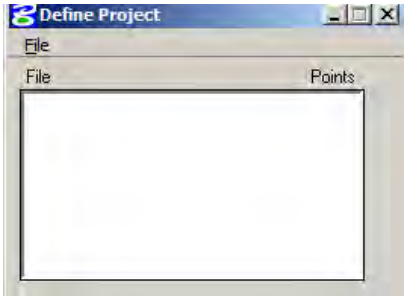


Figure 6: Define Project window

On this window, click File > New Project. In the new **Project information** window, fill in the descriptions as given in the Figure 7.

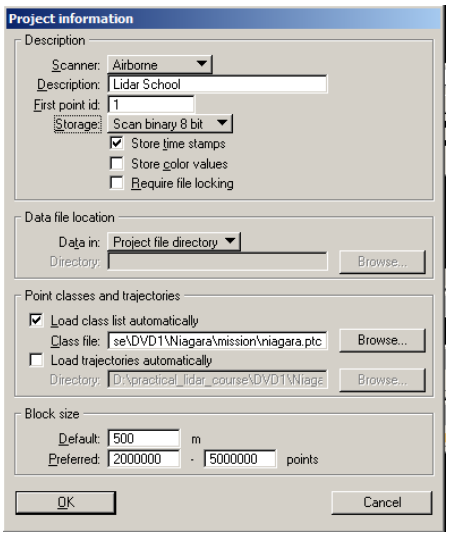


Figure 7: Project information details window

Click *File* > *Save As* and save the project as **building_project.prj**. Select the block on the window, using  on the Microstation main toolbar. On the **Project Definitions** window, click *Block* > *Add by Boundaries*. In this new window, click OK.

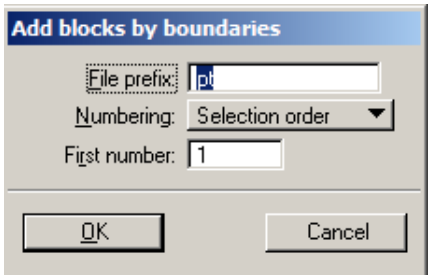


Figure 8: Add blocks by boundaries

A confirmation dialog would show up. Click OK again.

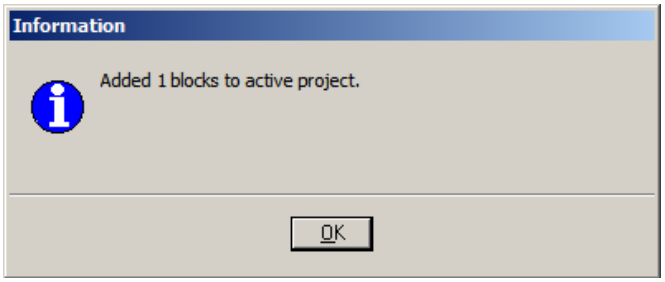


Figure 9: Confirmation window for a block addition.

As we kept our description earlier as **Lidar School,** note that the same text appears in the **Project Definitions** window (Figure 10). In this window, click *File* > *Import Points to project.*

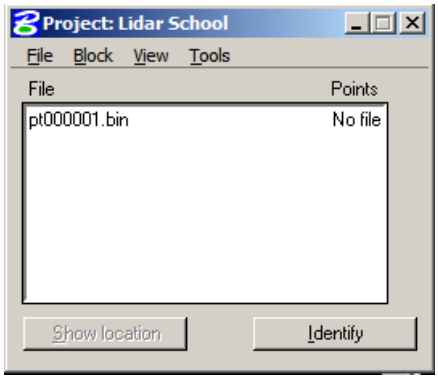


Figure 10: Block imported into the project

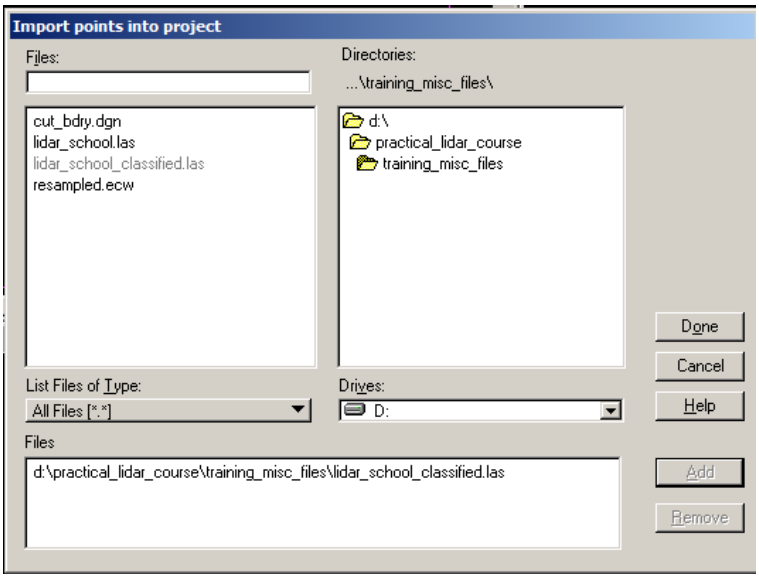


Figure 11: Selecting the LAS file for import

Select the file **lidar_school_classified.las** and click *Add*. Click *Done*. In the new dialog box, click OK.

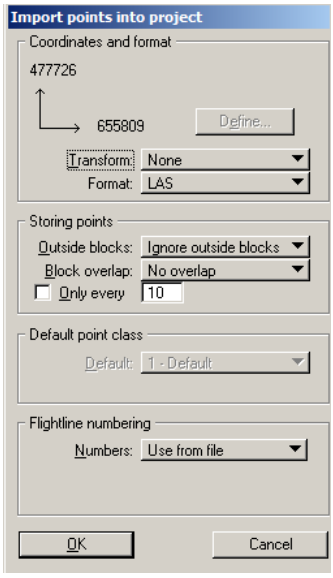


Figure 12: Importing points from the LAS file into the project

The report for the loading of points will now be generated and displayed on the window. Close this new window. In the TerraScan *Main Menu bar,* click *File* > *Open Block* and click inside the block in **View 1** of the Microstation workspace.

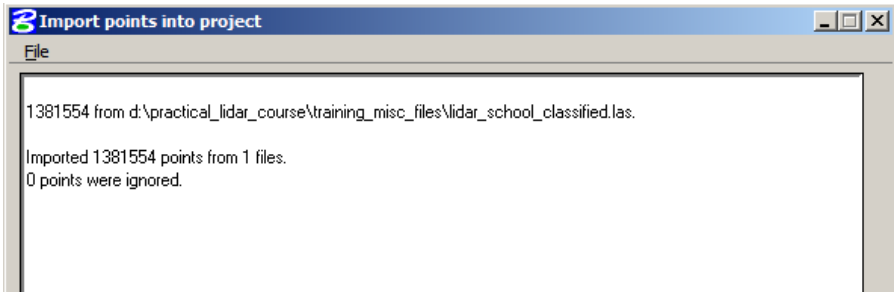


Figure 13: Data loading report

In the TerraScan menu bar, click *View* > *Display Mode*. Click *Colors …* and in the new window click *Auto Fit*.

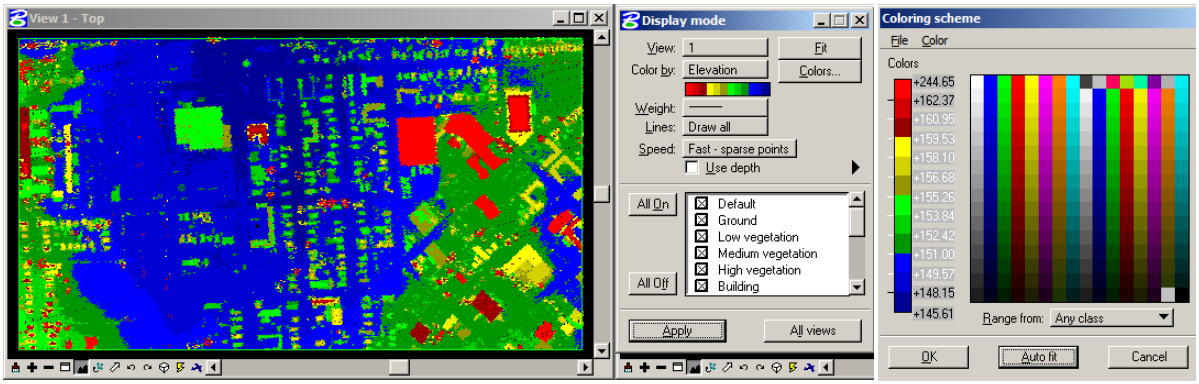


Figure 14: Points displayed based on elevation.

In the TerraScan menu bar, click View > Display mode. In the dialog box, set *Color by* class and check off all the classes except the ground class.

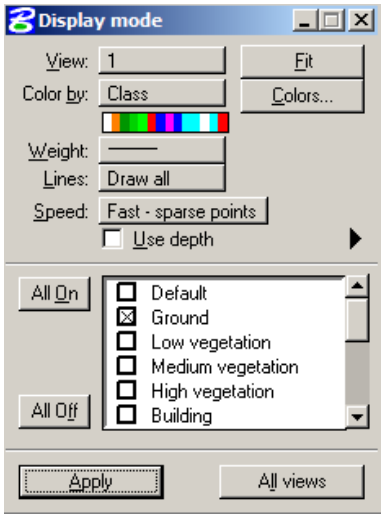


Figure 15: Coloring by class

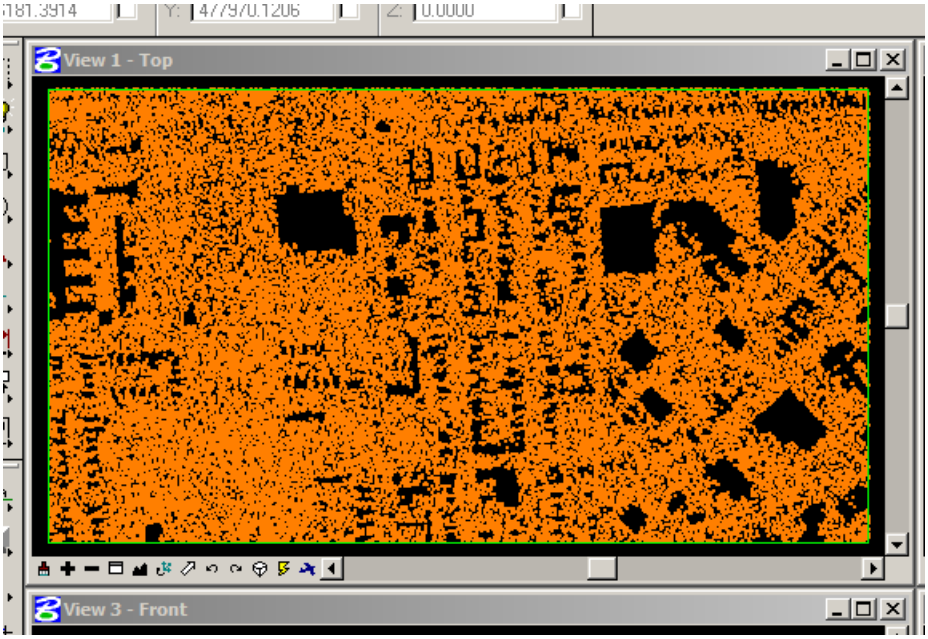


Figure 16: Ground classes displayed on the workspace

On the Microstation Main Menu, click *Applications* > *TerraPhoto* > *General*. The General toolbar for Terraphoto appears in the workspace.

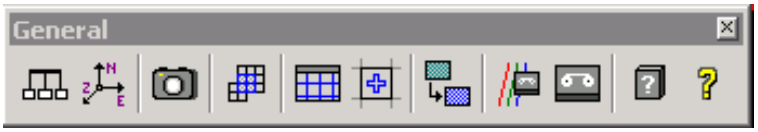


Figure 17: General toolbar for TerraPhoto

The fourth icon  on this toolbar, manages the rasters for display. Click on this icon. A new window opens up.

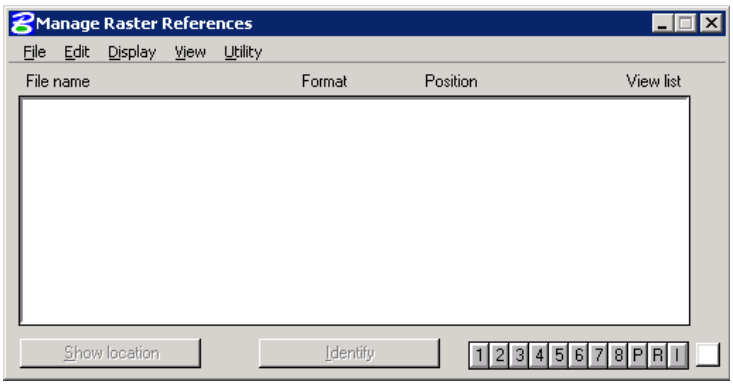


Figure 18: Manage Raster Window

Click *File* > *Attach rasters*. Select the file **resampled.ecw** from the given data. Click *Add* and then click *Done*. Set the **reference visibility** only for **View 1** and click *OK*.

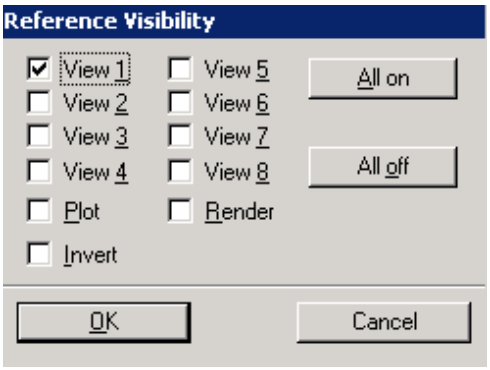


Figure 19: Setting reference visibility

The georectified photo is now loaded at the background for viewing. Since we switched off all the classes except the ground, we will see the buildings through the point cloud.

Figure 20: Image shown at the background

Click on *Applications* > *TerraScan* > *Draw* and the **Draw** toolbar appears in the Microstation workspace.

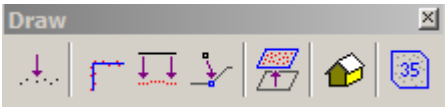


Figure 21: Draw toolbar in TerraScan

This module actually reclassifies the building again. Therefore, to avoid confusion, we shall use a new class named **Building_2**. To create this new class, click *View* > *Display Mode* on the TerraScan menu, and click *Colors …* In the dialog box, click *Add*. Select the color that you desire to have for the new class and then click *OK*. Close this dialog box.

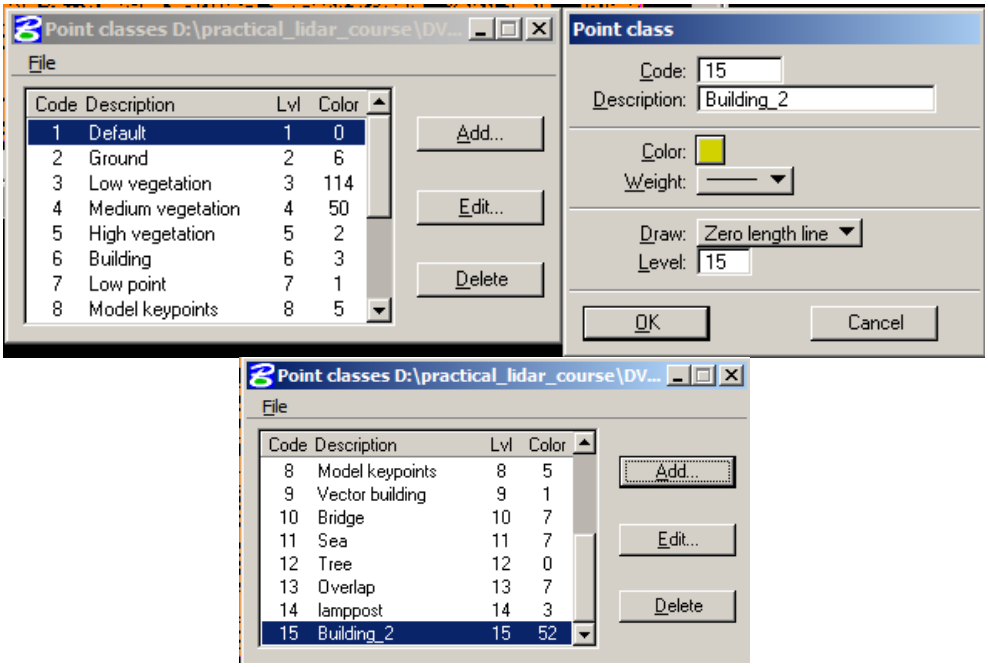


Figure 22: Adding a new class to the list of existing classes

Click on the **House shaped icon** to start modeling buildings.

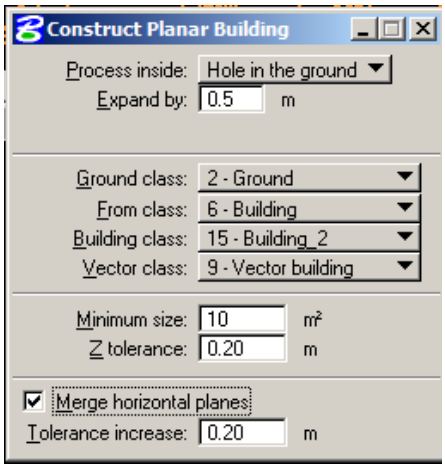


Figure 23: Setting parameters for modeling buildings

For this laboratory, we shall concentrate on a single building. The building is shown in Figure 24. Click on the roof of this building.

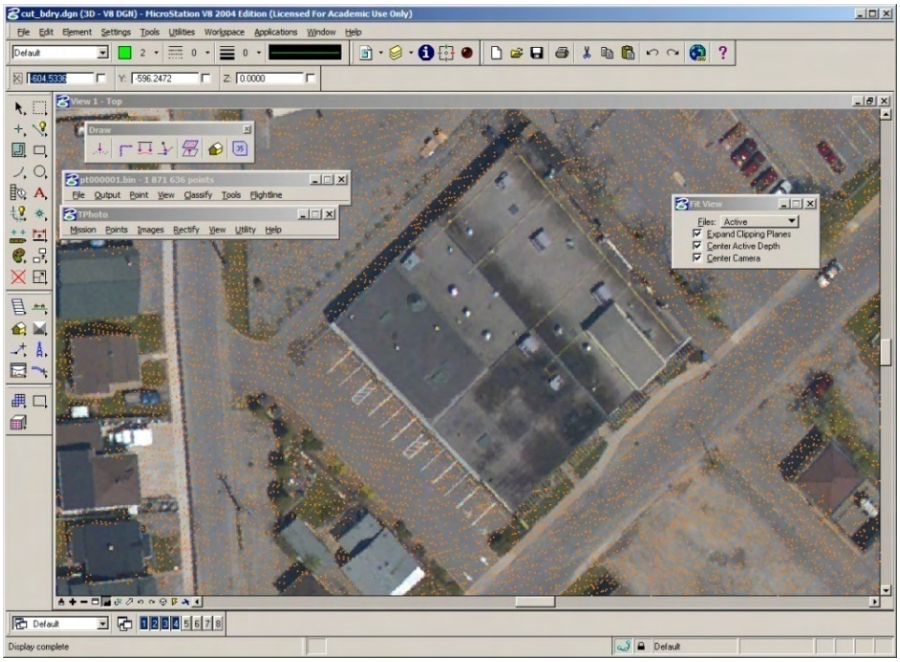


Figure 24: One building for the laboratory

By clicking on the roof of the building, many polygons are generated which actually denote planar regions identified by the algorithm in TerraScan. These planar regions may need to be merged, aligned or split as the need arises based on the ground truth, a priori knowledge or cross section visualization with the help of TerraScan tools.

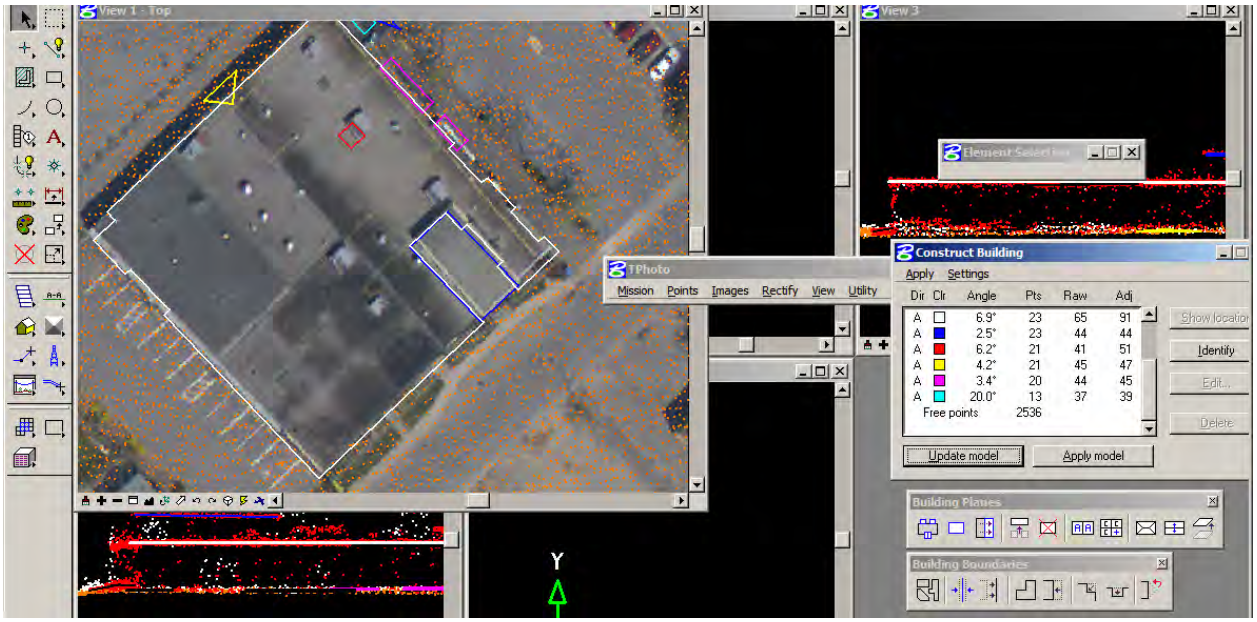


Figure 25: Initial building model generated by TerraScan.

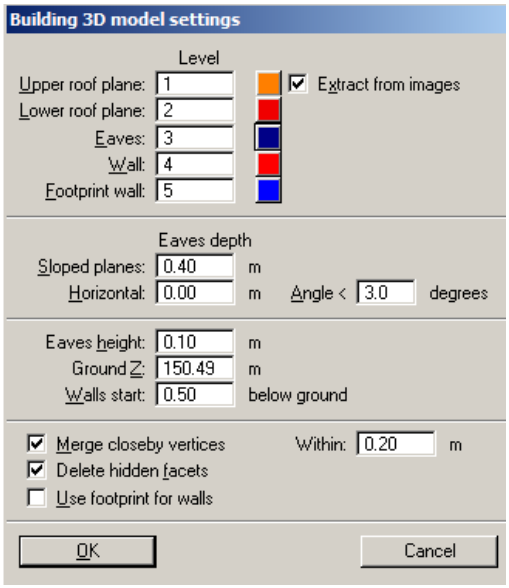


Figure 26: Setting the parameters of the 3D model

Now there are two new toolbars on the screen namely **Building Planes** and **Building Boundaries**. The specific functions of a few of them would be explained during the hands-on laboratory exercise.[1]

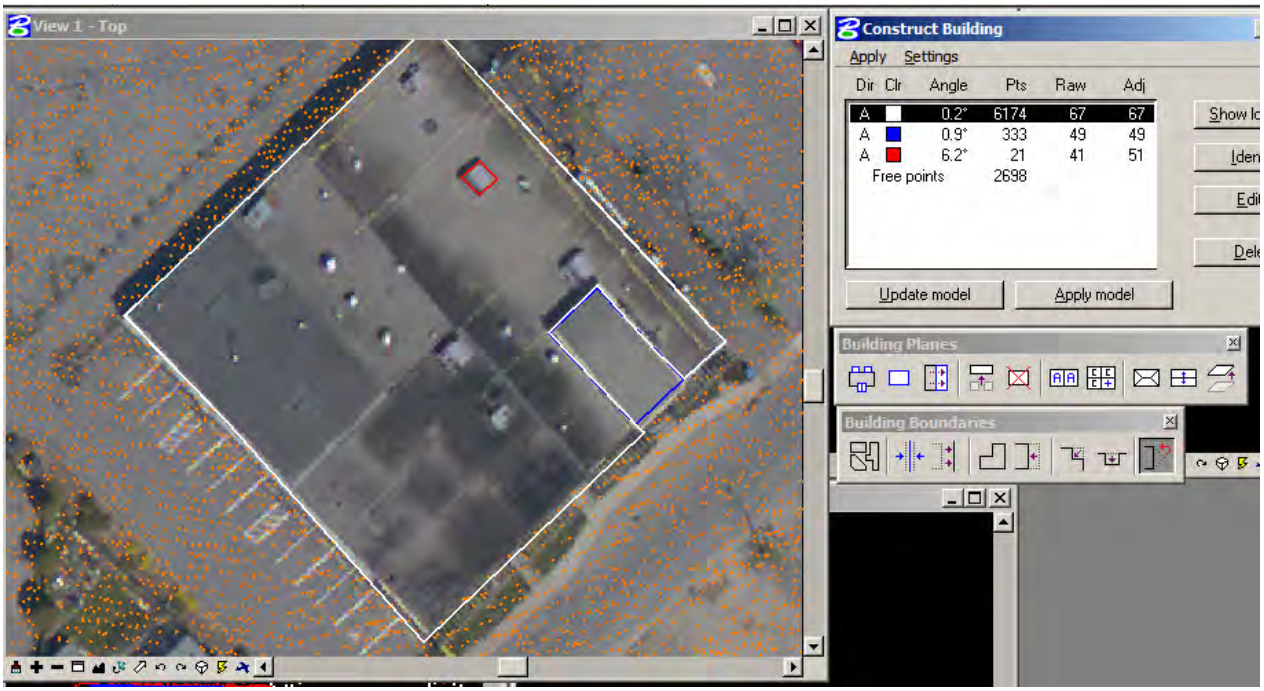


Figure 27: Building model nearly finalized with the help of Planes and Boundaries toolbars

Now click on *Apply Model*. Once this is done, the building gets walls. We would now like to see the buildings that have been formed, in 2.5D.

In the **View 1** window, click on the  icon.

[1] The description of the tools have been given at the end of this chapter

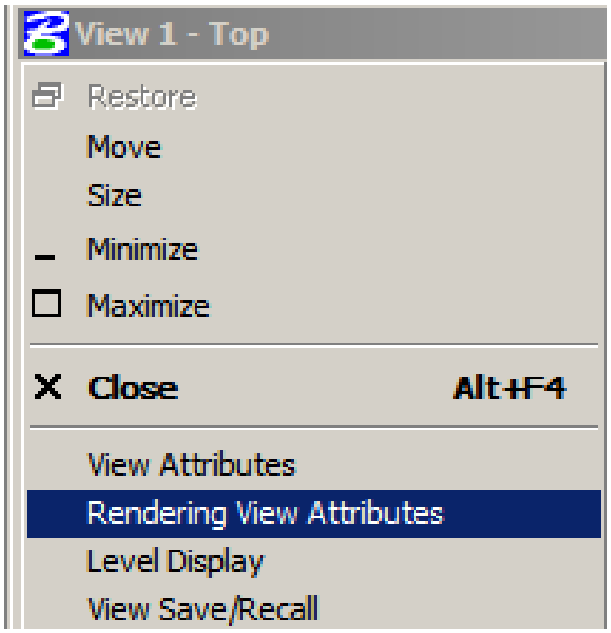


Figure 28: Rendering viewing attributes

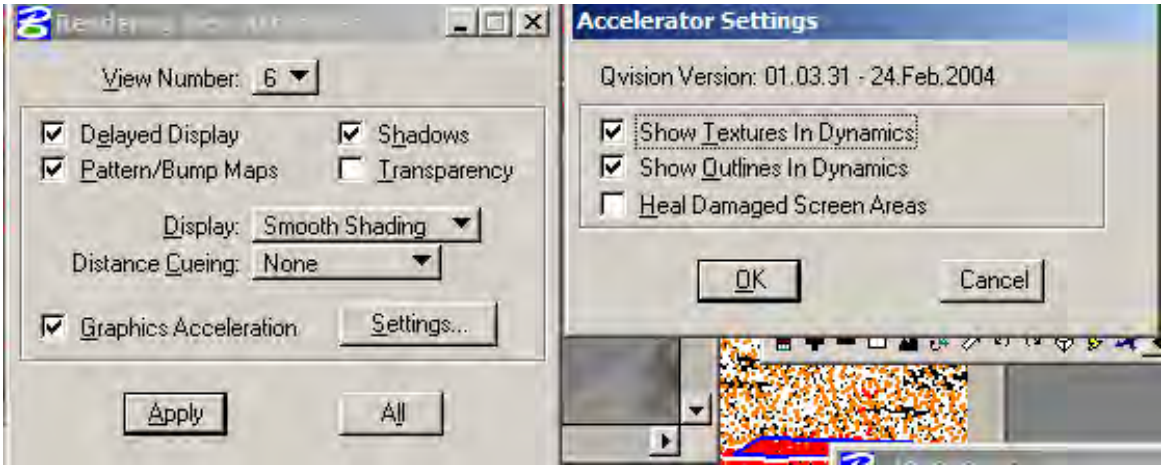


Figure 29: Rendering parameters being set

The model is finally generated in view number 6.

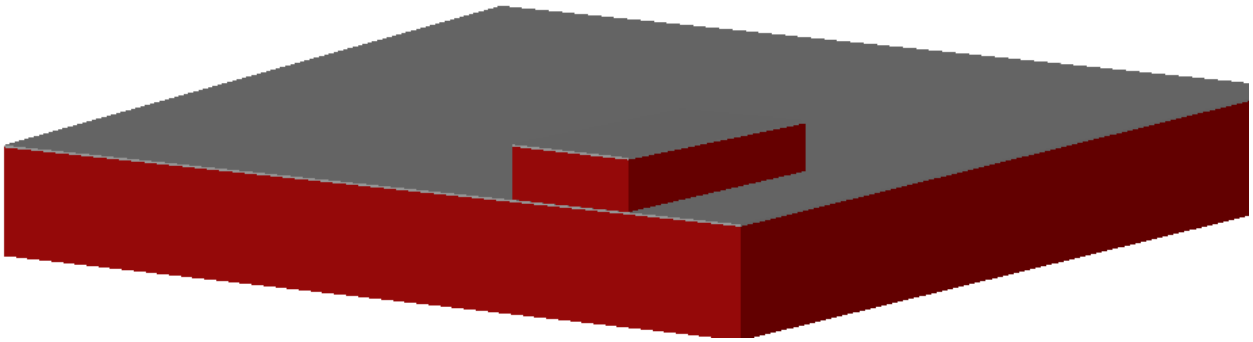


Figure 30: Building model generated by the TerraScan

CONTOUR PRODUCTION

We have been given a design file **cut_boundary.dgn.** Make a copy of the design file in your personal folder and open the DGN file, Click Tools > MDL Applications. In this new dialog box, load TSCAN and TMODEL utilities. This will set the toolbar and the main window of TSCAN and TMODEL on the screen.

Create a new project as mentioned earlier in this chapter and import the points from the file **lidar_school_classified.las** into the project. Read the points inside the block using the TerraScan main menu.

We can see by clicking *Tools* > *Show statistics*, that all of the points have been classified.

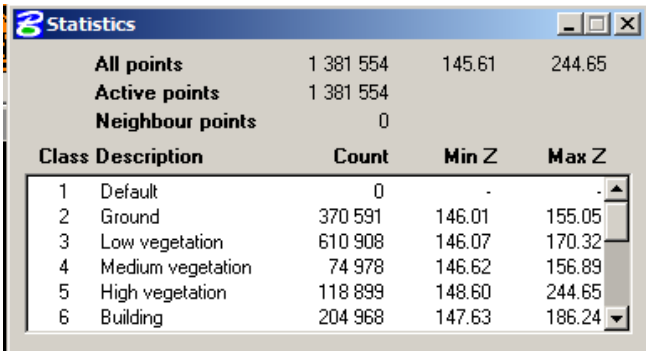


Figure 31: Statistics box

Close the statistics box. We now create **model keypoints** on our ground class. To do this, click *Classify* > *Routine* > *Model keypoints*. In the sampling window box, set *use points every* 10.00 m and set the model elevation tolerances to 0.12 m.

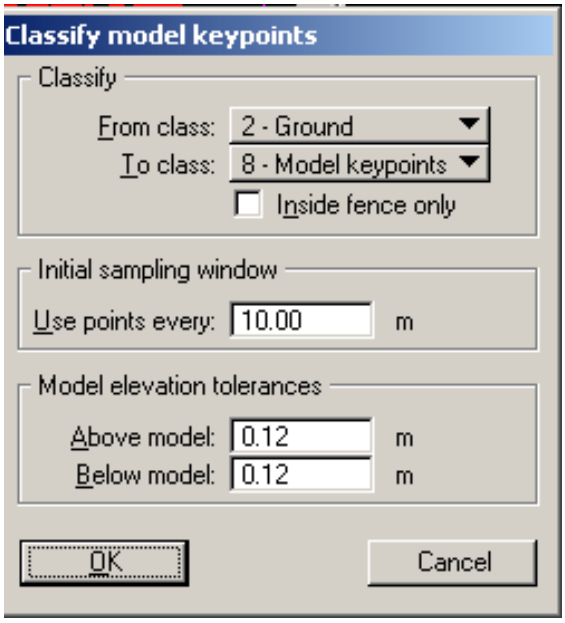


Figure 32: Setting parameters for model keypoints

Click OK. Once these points are classified we will save these points for a later use. On the TerraScan menu, click *File* > *Save Points As*. Select the *Model Keypoints* class and the output type to be Scan binary 8 bit lines.

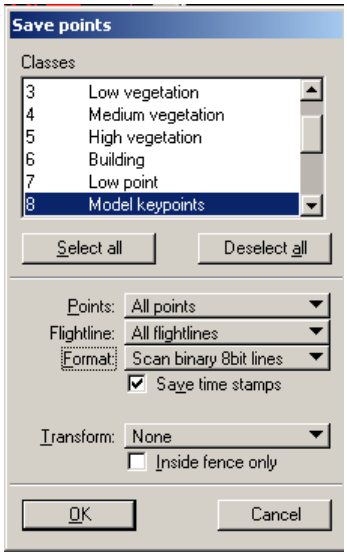


Figure 33: Saving points with TerraScan

Save the file as **mdl_kp_12.bin**. This file would be used later in creating the contours.

Let us now focus our attention on the TerraModel toolbox.

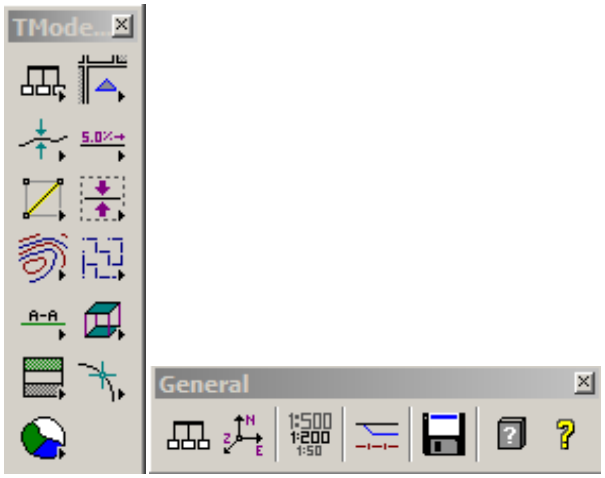


Figure 34: The TerraModel toolbar and the General Toolbar of TMODEL

Click Applications > TerraModeller > General. In the toolbar, which appears, we will click on the *Surface Tool*  icon. The *TerraModeler - Surfaces* window opens up.

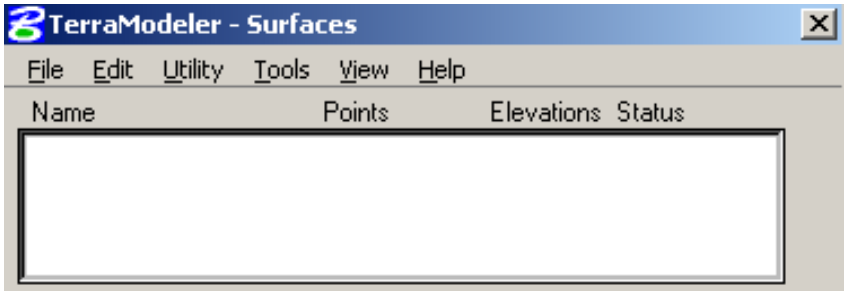


Figure 35: Surfaces window for TerraModeler

Click *File* > *Import* > *XYZ binary file*. Select the **mdl_kp_12.bin** file. Click *Add* and then click *Done*. A code usage window would open up. Use the model keypoints as random points.

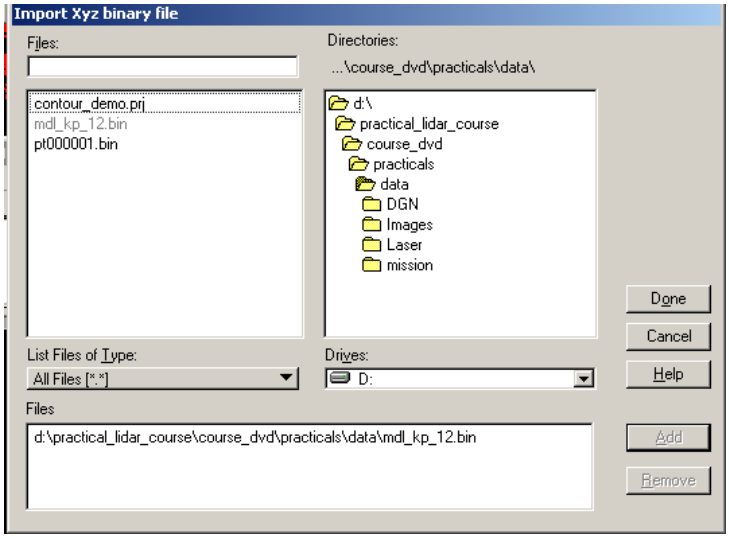


Figure 36: Adding data for TerraModeler

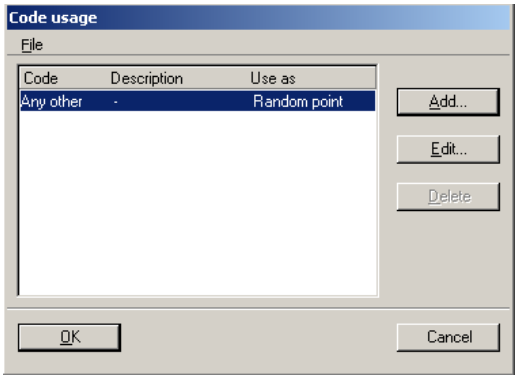


Figure 37: Code usage definitions window

Click OK. A **triangulate surface** dialog will pop-up. In this dialog, click OK.

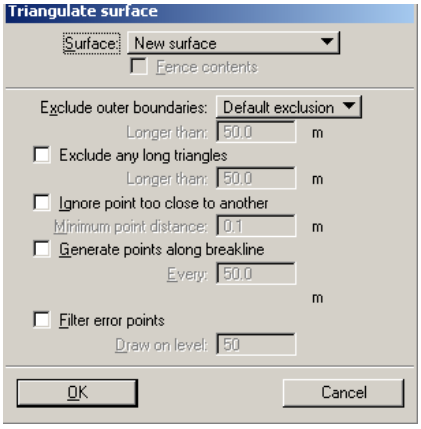


Figure 38: Setting parameters for the triangulated surface to be generated

In the surface settings dialog, enter the name of the ground surface as **mdl_key_12** and the storage name as **mdl_key_12.tin**.

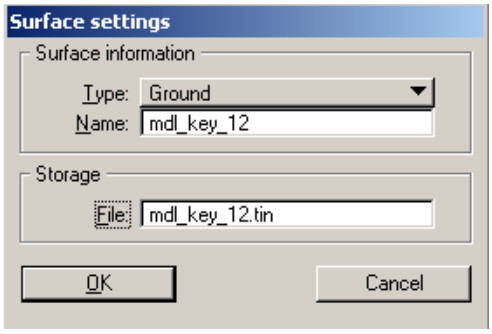


Figure 39: Naming the output file

Click on *Ok* and close the TerraModeller window. Now click on  icon in the TerraModeller toolbox.

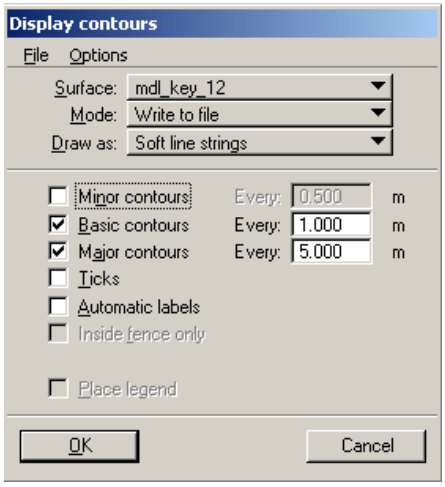


Figure 40: Options for displaying contours

Click *Options* > *Contours* …The display options for contours opens up.

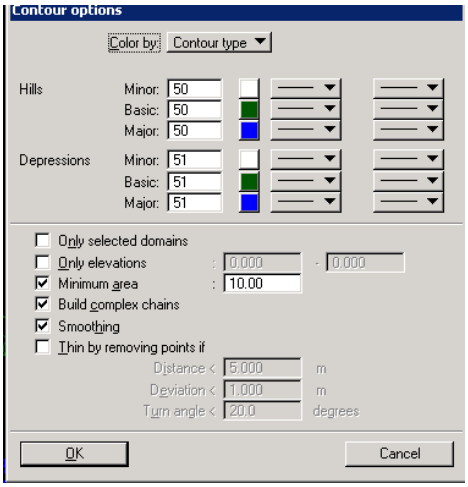


Figure 41: Setting layers and colors for storing the contours in the DGN file.

Set the display levels[2] and colors for the contours. Click Ok. Click OK again. The contours would now be calculated and displayed.

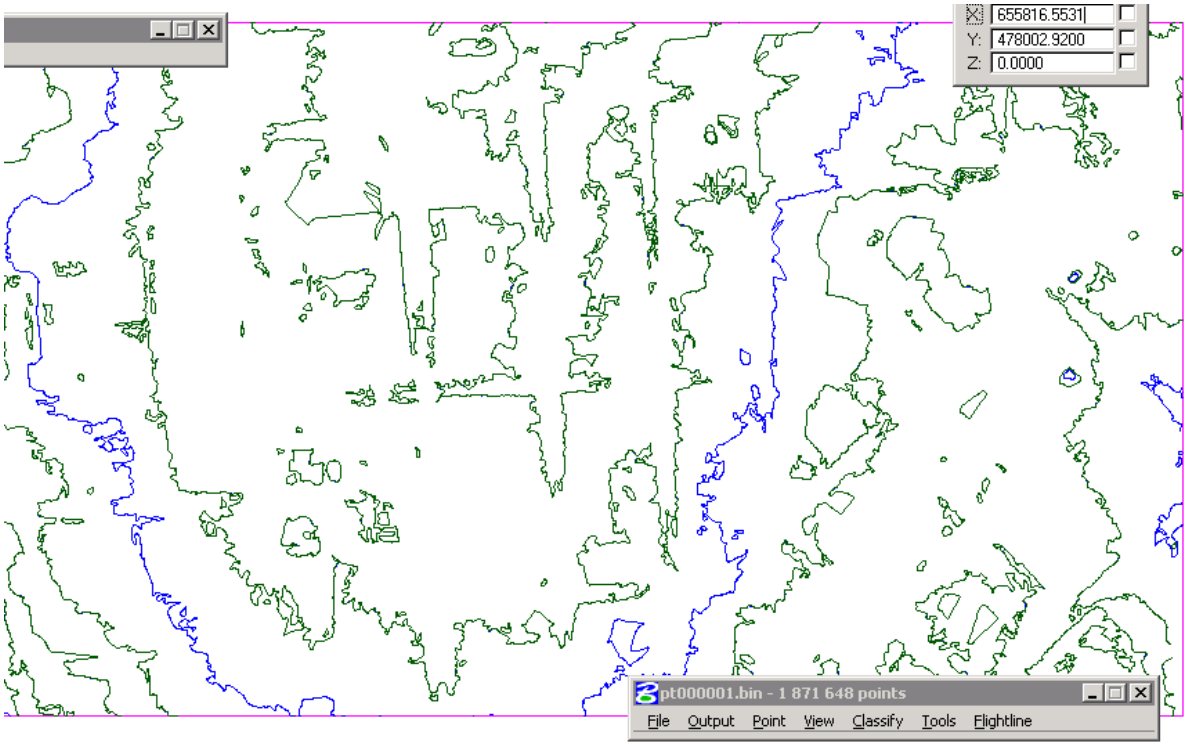


Figure 42: Contours displayed in the window

ADVANCED TASK: CREATING SMOOTHER CONTOURS

We see that in the earlier task, the contours are not very smooth. Without going into much detail, we present the methodology of producing smoother contours.

In the above exercise, we have already classified the points. In this exercise we will smoothen the ground points first by clicking *Tools* > *Smoothen Points* on the TerraScan menu.

In the dialog box which pops up, we set the parameters for smoothing and set the fixed class to any dummy class.

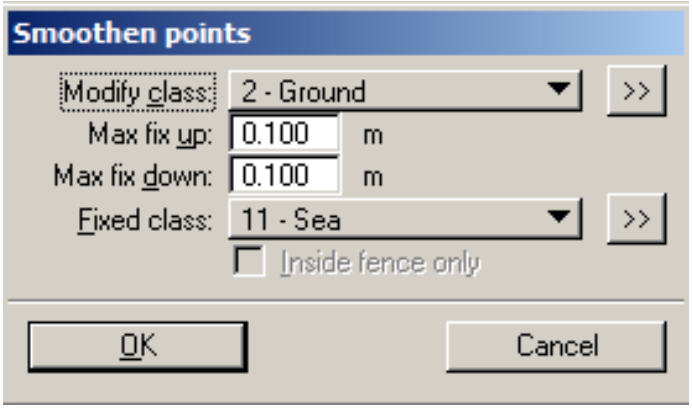


Figure 43: Entering the criteria for smoothening the points.

[2] In Microstation, various shapes and polyhedrals can be arranged and managed in different layers. These layers are referred to as **Levels** in the jargon of Bentley-Microstation.

Click OK. Now click *Classify* > *Routine* > *Model keypoints* and in the new dialog box, we collect samples from a larger area.

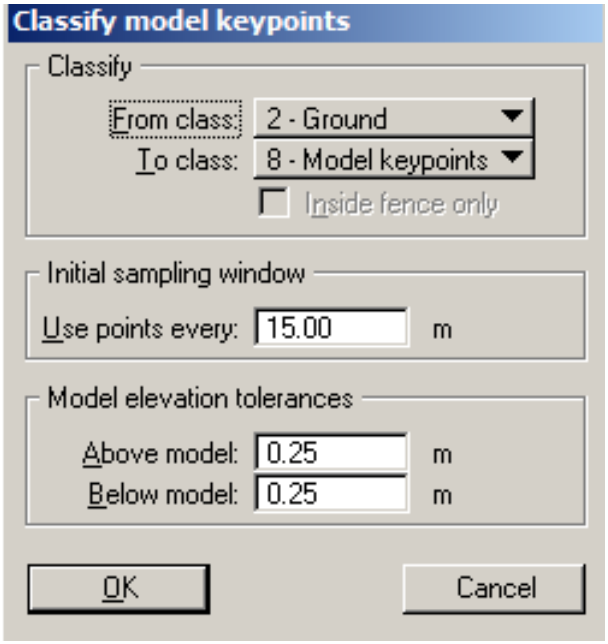


Figure 44: Model keypoints criteria for smoother contours

Click OK. Now click on *File* > *Save Points As*. Save the Model Keypoints class in a new file **mdl_smooth_15_25.bin**. This file would be loaded later to generate the surface model and contours.

Now click on the icon  to open the TerraModel surfaces window. Now import the points from the **mdl_smooth_15_25.bin**. And then generate the surface from these points considering them as random points. Name the new surface model as **mdl_smooth_15_25.tin**.

Now click on the icon to open the **Display contours** dialog.

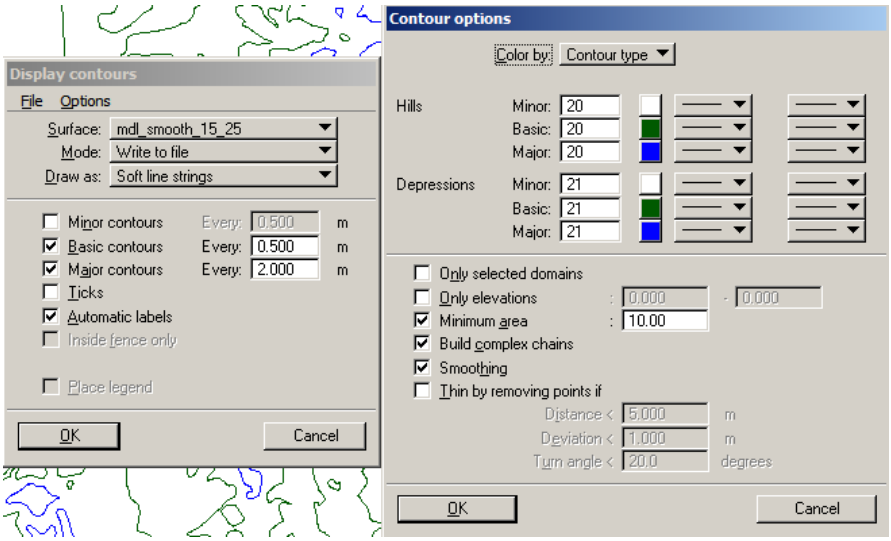


Figure 45: Setting contour display options

Click OK. Click OK again. The contours are now generated.

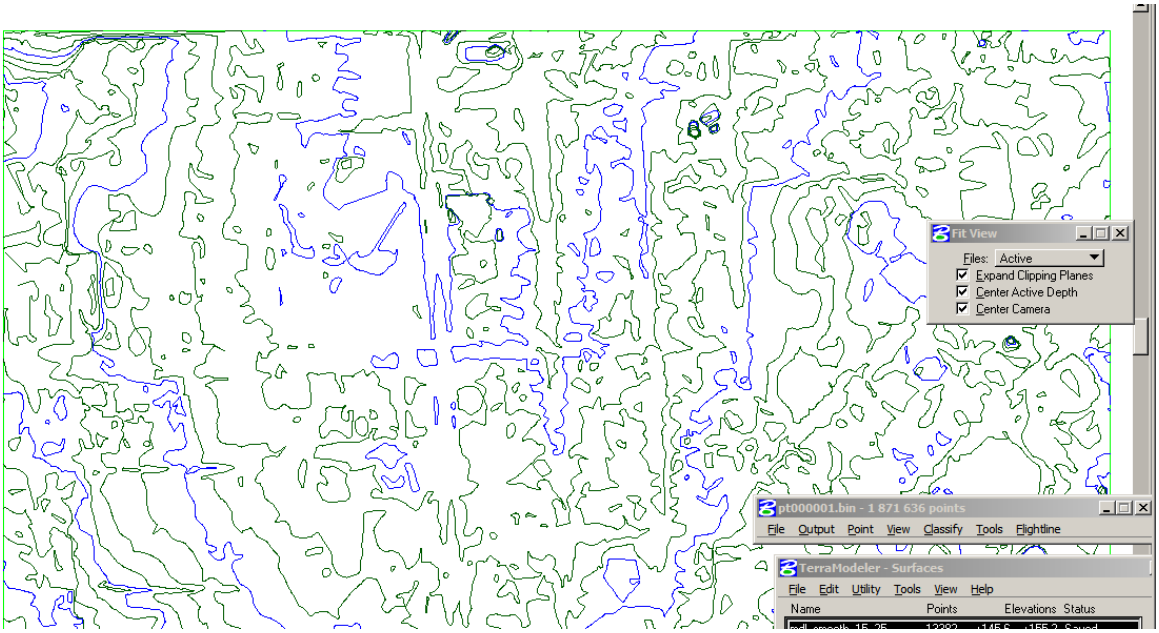


Figure 46: Smoothened contours generated

EXERCISES

(a) Generate and edit a few more building models using the Building Plane and Building Boundary Tools.
(b) Try the same exercise for generating contours using macros.

APPENDIX 4-I: DESCRIPTION OF BUILDING VECTORIZATION TOOLS

BUILDING PLANES

Tools	Description
	Finds more planes in detail.
	Adds a building plane to the set of already detected planes
	Mirrors a building plane
	Merge two building planes
	Delete a building plane
	Create a direction group for planes
	Add a plane to a group
	Assign plane block

	Assign plane symmetry
	Adjust building height

BUILDING BOUNDARIES

Tools	Description
	Define the type of the plane boundary: Polygon, Rectangle, Rectangular
	Auto align plane boundaries
	Manually align a boundary segment
	Place a boundary shape
	Modify a boundary shape
	Cut a boundary corner
	Cut a boundary segment
	Delete a boundary vertex

Annexure

ASPRS LIDAR Data Exchange Format Standard

Version 1.0

May 9, 2003

FORMAT DEFINITION:

Files conforming to the ASPRS LIDAR data exchange format standard are named with an LAS extension. The LAS file is intended to contain LIDAR point data records. The data will generally be put into this format from software (provided by LIDAR hardware vendors) which combines GPS, IMU, and laser pulse range data to produce X, Y, and Z point data. The intention of the data format is to provide an open format which allows different LIDAR vendors to output data into a format which a variety of LIDAR software vendors can use. Software that creates the LAS file will be referred to as "generating software", and software that reads and writes to the LAS file will be referred to as "user software" within this specification.

The format contains binary data consisting of a header block, variable length records, and point data. All data is in little-endian format. The header block consists of a public block followed by variable length records. The public block contains generic data such as point numbers and coordinate bounds. The variable length records contain variable types of data including projection information, metadata, and user application data.

PUBLIC HEADER BLOCK
VARIABLE LENGTH RECORDS
POINT DATA

DATA TYPES:

char (1 byte)
unsigned char (1 byte)
short (2 bytes)
unsigned short (2 bytes)
long (4 bytes)
unsigned long (4 bytes)
double (8 byte IEEE floating point format)

PUBLIC HEADER BLOCK:

Item	Format	Size	Required
File Signature ("LASF")	char[4]	4 bytes	*
Reserved	unsigned long	4 bytes	
GUID data 1	unsigned long	4 bytes	
GUID data 2	unsigned short	2 byte	
GUID data 3	unsigned short	2 byte	
GUID data 4	unsigned char[8]	8 bytes	
Version Major	unsigned char	1 byte	*
Version Minor	unsigned char	1 byte	*
System Identifier	char[32]	32 bytes	*
Generating Software	char[32]	32 bytes	*
Flight Date Julian	unsigned short	2 bytes	
Year	unsigned short	2 bytes	
Header Size	unsigned short	2 bytes	*
Offset to data	unsigned long	4 bytes	*
Number of variable length records	unsigned long	4 bytes	*
Point Data Format ID (0-99 for spec)	unsigned char	1 byte	*
Point Data Record Length	unsigned short	2 bytes	*
Number of point records	unsigned long	4 bytes	*
Number of points by return	unsigned long[5]	20 bytes	*
X scale factor	double	8 bytes	*

Y scale factor	double	8 bytes	*
Z scale factor	double	8 bytes	*
X offset	double	8 bytes	*
Y offset	double	8 bytes	*
Z offset	double	8 bytes	*
Max X	double	8 bytes	*
Min X	double	8 bytes	*
Max Y	double	8 bytes	*
Min Y	double	8 bytes	*
Max Z	double	8 bytes	*
Min Z	double	8 bytes	*

All data that is not required and not filled with data must be set to zeros.

File Signature: The file signature must contain the four characters "LASF", and it is required by the ASPRS standard. These four characters should be checked by user software to determine valid LAS files.

Reserved: This data field is reserved, and must contain zeros unless otherwise specified by subsequent versions of the ASPRS LIDAR data exchange format standard.

GUID data: The Globally Unique Identifier (GUID) data fields are not required by the ASPRS standard (GUID data 1, GUID data 2, GUID data 3, and GUID data 4). The GUID data fields are intended to provide space for uniquely identifying each file created. The GUIDs provide a method of tracking individual files regardless of file names or directory structure.

Version Number: The version number is required by the ASPRS standard. The version number consists of a major and minor field. The major and minor fields combine to form the number that indicates the format number of the current specification itself. For example, specification number 1.0 would be 1 in the major field and 0 in the minor field.

System ID: The system identifier field is populated by the generating software application. This field provides a mechanism for specifying the actual hardware system which gathered the LIDAR data. The system identifier itself should be provided by the hardware system vendor and populated by the generating software. If the character data is less than 32 characters, the remaining data must be null.

Generating Software: The "generating software" field is also populated by the generating software. This information is ASCII data describing the generating software itself. This field provides a mechanism for specifying which generating software package and version was used during LAS file creation. If the character data is less than 16 characters, the remaining data must be null.

Flight Date Julian: The julian day of the year that the data was collected. This field should be populated by the generating software.

Year: The year in which the data was collected. This field should be populated by the generating software.

Header Size: The header size field is the actual header block itself. This number is required, and must be provided by the generating software. All software should read this header size number, and regard that number as the header size in bytes. In the event that data is added to the

header, the header size field must be updated with the new header size. Generating software is the only type of software that is allowed by the specification to add data to the general header. The variable length records should be used whenever possible to add data to the header, and not to extend the general header. However, this specification allows for a general header to be larger than the specified data structure. Therefore, the header size field must at all times contain the exact size of the header itself. In the event a generating software package adds data to the general header, this data must be placed at the end of the specified general header structure.

Offset to data: The offset to data field is the actual number of bytes from the beginning of the file to the data itself including the two bytes for the data start signature. This data offset must be updated if any software adds data from the general header or adds/removes data from the variable length records. It is recommended that additional space be allocated to the variable length header space, so that additional variable length data can be added after the initial data is written. This space prevents the need to rewrite the file every time a variable length record is added.

Number of variable length records: This field contains the current number of variable length records. This number must be updated if the number of variable length records changes at any time.

Point Data Format ID: The point data format ID corresponds to the point data record format type.

Number of point records: This field contains the total number of point records within the file.

Number of points by return: This field contains an array of the total point records per return. The first unsigned long value will be the total number of records from the first return, and the second contains the total number for return two, and so forth up to five returns.

X, Y, and Z scale factors: The scale factor fields contain a double floating point value that is used to scale the corresponding X, Y, and Z long values within the point records. The corresponding X, Y, and Z scale factor must be multiplied by the X, Y, or Z point record value to get the actual X, Y, or Z coordinate. For example, if the X, Y, and Z coordinates are intended to have two decimal point values, then each scale factor will contain the number 0.01.

X, Y, and Z offset: The offset fields should be used to set the overall offset for the point records. In general these numbers will be zero, but for certain cases the resolution of the point data may not be large enough for a given projection system. However, it should always be assumed that these numbers are used. So to scale a given X from the point record, take the point record X multiplied by the X scale factor, and then add the X offset.

$X_{coordinate} = (X_{record} * X_{scale)} + X_{offset}$
$Y_{coordinate} = (Y_{record} * Y_{scale)} + Y_{offset}$
$Z_{coordinate} = (Z_{record} * Z_{scale)} + Z_{offset}$

Max and Min X, Y, Z: The max and min data fields are the actual file coordinate extents of the LAS point file data.

The projection information for the point data is required for all data. The projection information will be placed in the variable length records. Placing the projection information within the variable length records allows for any projection to be defined including custom projections. The GeoTiff specification http://www.remotesensing.org/geotiff/geotiff.html is the model for representing the projection information, and the format will be explicitly defined by this specification.

VARIABLE LENGTH RECORDS:

Item	Format	Size	Required
Record Signature (0xAABB)	unsigned short	2 bytes	*
User ID	char[16]	16 bytes	*
Record ID	unsigned short	2 bytes	*
Record Length After Header	unsigned short	2 bytes	*
Description	char[32]	32 bytes	

Record Signature: The record signature is a two byte data field that must contain 0xAABB. Note that there may be other data within the file that has 0xAABB as its data.

User ID: The user ID field is ASCII character data that identifies the user which created the variable length record. It is possible to many variable length records from different sources with different user Ids. If the character data is less than 16 characters, the remaining data must be null. The user ID must be registered with the managing body of the ASPRS standard or its official designee. The management of these Ids insures that no two individuals accidentally use the same ID. The specification will initially use two Ids. One for globally specified records (LASF_Spec), and another for projection types (LASF_Projection).

Record ID: The record ID is dependent upon the User ID. There can be 0 to 65535 record Ids for every User ID. The ASPRS standard will manage its own record Ids (User Ids owned by the specification), otherwise record Ids will be managed by the owner of the given User ID. So each User ID is allowed to assign 0 to 65535 record Ids as they wish. Publicizing the meaning of a given record ID will be left to the owner of the given User ID. Unknown User ID/Record ID combinations should be ignored.

Record Length after Header: The record length is the number of bytes for the record after the end of the standard part of the header.

Description: Optional null terminated text description of the data. Any remaining characters not used must be null.

Point Data Start Signature:
Two bytes after the last variable length record, and before the point data
0xCCDD

POINT DATA RECORD FORMAT 0:

Item	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits	3 bits	*
Number of Returns (given pulse)	3 bits	3 bits	*
Scan Direction Flag	1 bit	1 bit	*
Edge of Flight Line	1 bit	1 bit	*
Classification	unsigned char	1 byte	
Scan Angle Rank (-90 to +90) – Left side	char	1 byte	*
File Marker	unsigned char	1 byte	
User Bit Field	unsigned short	2 bytes	

X, Y, and Z: The X, Y, and Z values are stored as long integers. The corresponding X scale, Y scale, and Z scale values from the public header block change these long integers to their true floating point values. The corresponding offset values can also be used for projections with very large numbers.

Intensity: The intensity value is the integer representation of the pulse return magnitude. This value is optional and system specific.

Return Number: The return number is the pulse return number for a given output pulse. A given output laser pulse can have many returns, and they must be marked in sequence of return. The first return will have a return number of one, the second a return number of two, and so on up to five returns.

Number of Returns (given pulse): The number of returns is the total number of returns for a given pulse. So a laser data point may be return two (return number) with a total number of five returns.

Scan Direction Flag: The scan direction flag denotes the direction at which the scanner mirror was traveling at the time of the output pulse. A bit value of 1 is a positive scan direction, and a bit value of 0 is a negative scan direction.

Edge of Flight Line: The edge of flight line data bit has a value of 1 only when the point is at the end of a scan. It is the last point on a given scan line before it changes direction.

Classification: The classification field is a number to signify a given classification during filter processing. The ASPRS standard has a public list of classifications which shall be used when mixing vendor specific user software.

Scan Angle Rank: The scan angle rank is a signed one-byte number. The scan angle rank is the angle at which the laser point was output from the laser system including the roll of the aircraft. The scan angle is within 1 degree of accuracy from +90 to –90 degrees. The scan angle is an angle based on 0 degrees being NADIR, and –90 degrees to the left side of the aircraft in the direction of flight.

File Marker: The file marker is an optional field that should be used in conjunction with the variable length records. The file marker allows for the LAS flight-line based files to be combined into single files with more than one flight-line. The actual original flight-line should be tracked in the variable length records, so that individual flight lines can be separated from a given combined file.

User Bit Field: A bit field that is to be used at the users discretion.

Other point data formats must be derived from the "Point Data 0" structure with the additional data added thereafter.

POINT DATA RECORD FORMAT 1:

Item	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits	3 bits	*
Number of Returns (given pulse)	3 bits	3 bits	*

Scan Direction Flag	1 bit	1 bit	*
Edge of Flight Line	1 bit	1 bit	*
Classification	unsigned char	1 byte	
Scan Angle Rank (-90 to +90) – Left side	unsigned char	1 byte	*
File Marker	unsigned char	1 byte	
User Bit Field	unsigned short	2 bytes	
GPS Time	double	8 bytes	*

GPS Time: The GPS time is the double floating point time tag value at which the point was acquired.

DEFINED VARIABLE LENGTH RECORDS: Georeferencing Information

Georeferencing for the ASPRS standard format will use the same robust mechanism that was developed for the GeoTIFF standard. The variable length header records section will contain the same data that would be contained in the GeoTIFF key tags of a TIFF file. With this approach, any vendor that has existing code to interpret the coordinate system information from GeoTIFF tags can simply feed the software with the information taken from the LAS file header. Since LAS is not a raster format and each point contains its own absolute location information, only 3 of the 6 GeoTIFF tags are necessary. The ModelTiePointTag (33922), ModelPixelScaleTag (33550), and ModelTransformationTag (34264) records can be excluded. The GeoKeyDirectoryTag (34735), GeoDoubleParamsTag (34736), and GeoASCIIParamsTag (34737) records will be used.

Only the GeoKeyDirectoryTag record is required. The GeoDoubleParamsTag and GeoASCIIParamsTag records may or may not be present, depending on the content of the GeoKeyDirectoryTag record.

GeoKeyDirectoryTag Record
User ID: LASF_Projection
Record ID: 34735

This record contains the key values that define the coordinate system. A complete description can be found in the GeoTIFF format specification. Here is a summary from a programmatic point of view for someone interested in implemention.

The GeoKeyDirectoryTag is defined as just an array of unsigned short values. But, programmatically, the data can be seen as something like this:

```
struct sGeoKeys
{
   unsigned short wKeyDirectoryVersion;
   unsigned short wKeyRevision;
   unsigned short wMinorRevision;
   unsigned short wNumberOfKeys;
   struct sKeyEntry
   {
      unsigned short wKeyID;
      unsigned short wTIFFTagLocation;
      unsigned short wCount;
      unsigned short wValue_Offset;
   } pKey[1];
```

};

Where:
wKeyDirectoryVersion = 1; // Always
wKeyRevision = 1; // Always
wMinorRevision = 0; // Always
wNumberOfKeys // Number of sets of 4 unsigned shorts to follow

For each set of 4 unsigned shorts:

wKeyID	Defined key ID for each piece of GeoTIFF data. IDs contained in the GeoTIFF specification.
wTIFFTagLocation	Indicates where the data for this key is located: 0 means data is in the wValue_Offset field as an unsigned short 34736 means the data is located at index wValue_Offset of the GeoDoubleParamsTag record. 34767 means the data is located at index wValue_Offset of the GeoAsciiParamsTag record.
wCount	Number of characters in string for values of GeoAsciiParamsTag , otherwise is 1
wValue_Offset	Contents vary depending on value for wTIFFTagLocation above

GeoDoubleParamsTag Record
User ID: LASF_Projection
Record ID: 34736
This record is simply an array of doubles that contain values referenced by tag sets in the GeoKeyDirectoryTag record.

GeoASCIIParamsTag Record
User ID: LASF_Projection
Record ID: 34737
This record is simply an array of ascii data. It contains many strings separated by null terminator characters which are referenced by position from data in the GeoKeyDirectoryTag record.

Classification lookup
User ID: LASF_Spec
Record ID: 0
Length: 256 recs X 16 byte struct len
struct CLASSIFICATION
{
    unsigned char ClassNumber;
    char Description[15];
};

Header lookup for flight-lines
User ID: LASF_Spec
Record ID: 1
Length: 256 recs X struct len

```
Struct FLIGHTLINE
{
        unsigned char FileMarkerNumber;
        char Filename[256];
};
```

Histogram
User ID: LASF_Spec
Record ID: 2

Text area description
User ID: LASF_Spec
Record ID: 3

Hough transform

From Wikipedia, the free encyclopedia

The **Hough transform** (pronounced /'hʌf/) is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses. The Hough transform as it is universally used today was invented by Richard Duda and Peter Hart in 1972, who called it a "generalized Hough transform" after the related 1962 patent of Paul Hough. The transform was popularized in the computer vision community by Dana H. Ballard through a 1981 journal article titled "Generalizing the Hough transform to detect arbitrary shapes".

Contents

Theory

In automated analysis of digital images, a subproblem often arises of detecting simple shapes, such as straight lines, circles or ellipses. In many cases an edge detector
can be used as a pre-processing stage to obtain image points or image pixels that are on the desired curve in the image space. Due to imperfections in either the image data or the edge detector, however, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line/circle/ellipse and the noisy edge points as they are obtained from the edge detector. For these reasons, it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects.

The simplest case of Hough transform is the linear transform for detecting straight lines. In the image space, the straight line can be described as $y = mx + b$ and can be graphically plotted for each pair of image points (x,y). In the Hough transform, a main idea is to consider the characteristics of the straight line not as image points x or y, but in terms of its parameters, here the slope parameter m and the intercept parameter b. Based on that fact, the straight line $y = mx + b$ can be represented as a point (b, m) in the parameter space (b vs. m graph.)

If one uses the slope and intercept parameters m and b, however, one faces the problem that vertical lines give rise to unbounded values of the parameters m and b [1] [2]. For computational reasons, it is therefore better to parameterize the lines in the Hough transform with two other parameters, commonly referred to as *r* and θ (*theta*). The parameter *r* represents the distance between the line and the origin, while θ is the angle of the vector from the origin to this closest point (see Coordinates). Using this parametrization, the equation of the line can be written as

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right),$$

which can be rearranged to $r = x\cos\theta + y\sin\theta$.

It is therefore possible to associate to each line of the image, a couple (r,θ) which is unique if $\theta \in [0, \pi]$ and $r \in \mathbf{R}$, or if $\theta \in [0, 2\pi]$ and $r \geq 0$. The (r,θ) plane is sometimes referred to as *Hough space* for the set of straight lines in two dimensions. This representation makes the Hough transform to be conceptually very close to the two-dimensional Radon transform.

It is well known that an infinite number of lines can go through a single point of the plane. If that point has coordinates (x_0,y_0) in the image plane, all the lines that go through it obey the following equation:

$$r(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

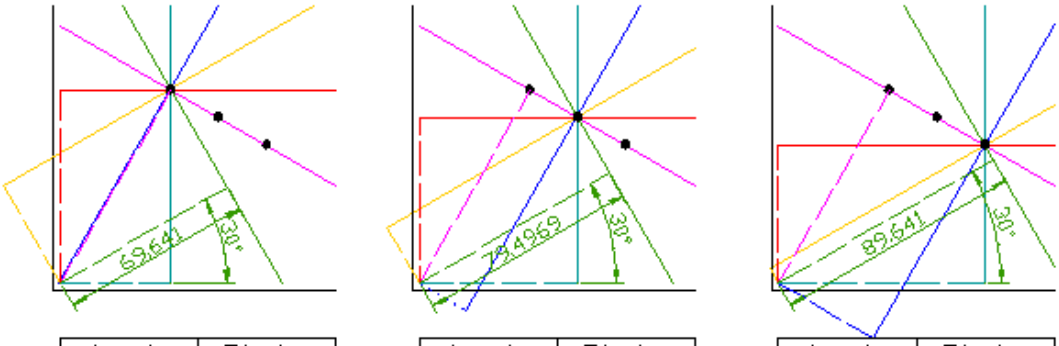
This corresponds to a sinusoidal curve in the (r,θ) plane, which is unique to that point. If the curves corresponding to two points are superimposed, the location (in the *Hough space*) where they cross correspond to lines (in the original image space) that pass through both points. More generally, a set of points that form a straight line will produce sinusoids which cross at the parameters for that line. Thus, the problem of detecting colinear points can be converted to the problem of finding concurrent curves.

Implementation

The Hough transform algorithm uses an array, called accumulator, to detect the existence of a line y = mx + b. The dimension
of the accumulator is equal to the number of unknown parameters of the Hough transform problem. For example, the linear Hough transform problem has two unknown parameters: m and b. The two dimensions of the accumulator array would correspond to quantized values for m and b. For each pixel and its neighborhood, the Hough transform algorithm determines if there is enough evidence of an edge at that pixel. If so, it will calculate the parameters of that line, and then look for the accumulator's bin that the parameters fall into, and increase the value of that bin. By finding the bins with the highest values, typically by looking for local maxima in the accumulator space, the most likely lines can be extracted, and their (approximate) geometric definitions read off. The simplest way of finding these *peaks* is by applying some form of threshold, but different techniques may yield better results in different circumstances - determining which lines are found as well as how many. Since the lines returned do not contain any length information, it is often next necessary to find which parts of the image match up with which lines. Moreover, due to imperfection errors in the edge detection step, there will usually be errors in the accumulator space, which may make it non-trivial to find the appropriate peaks, and thus the appropriate lines.

Example

Consider three data points, shown here as black dots.

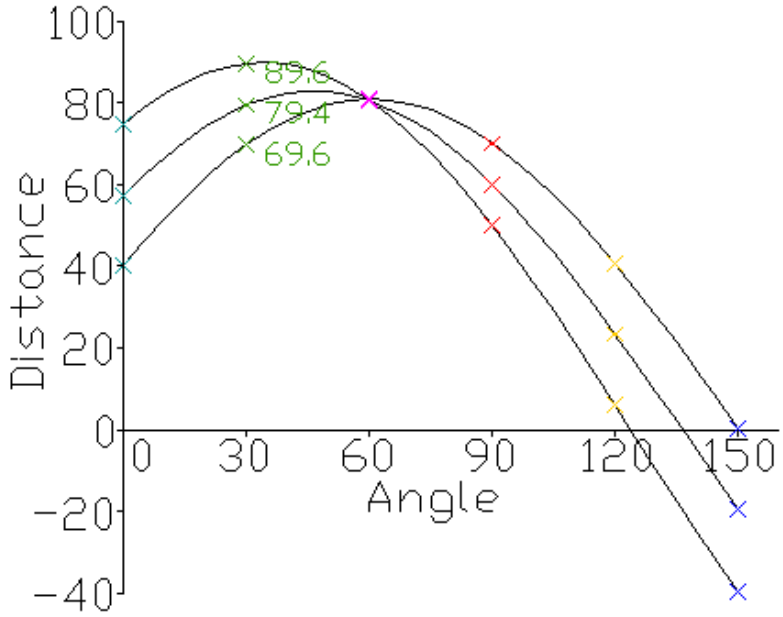


Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5

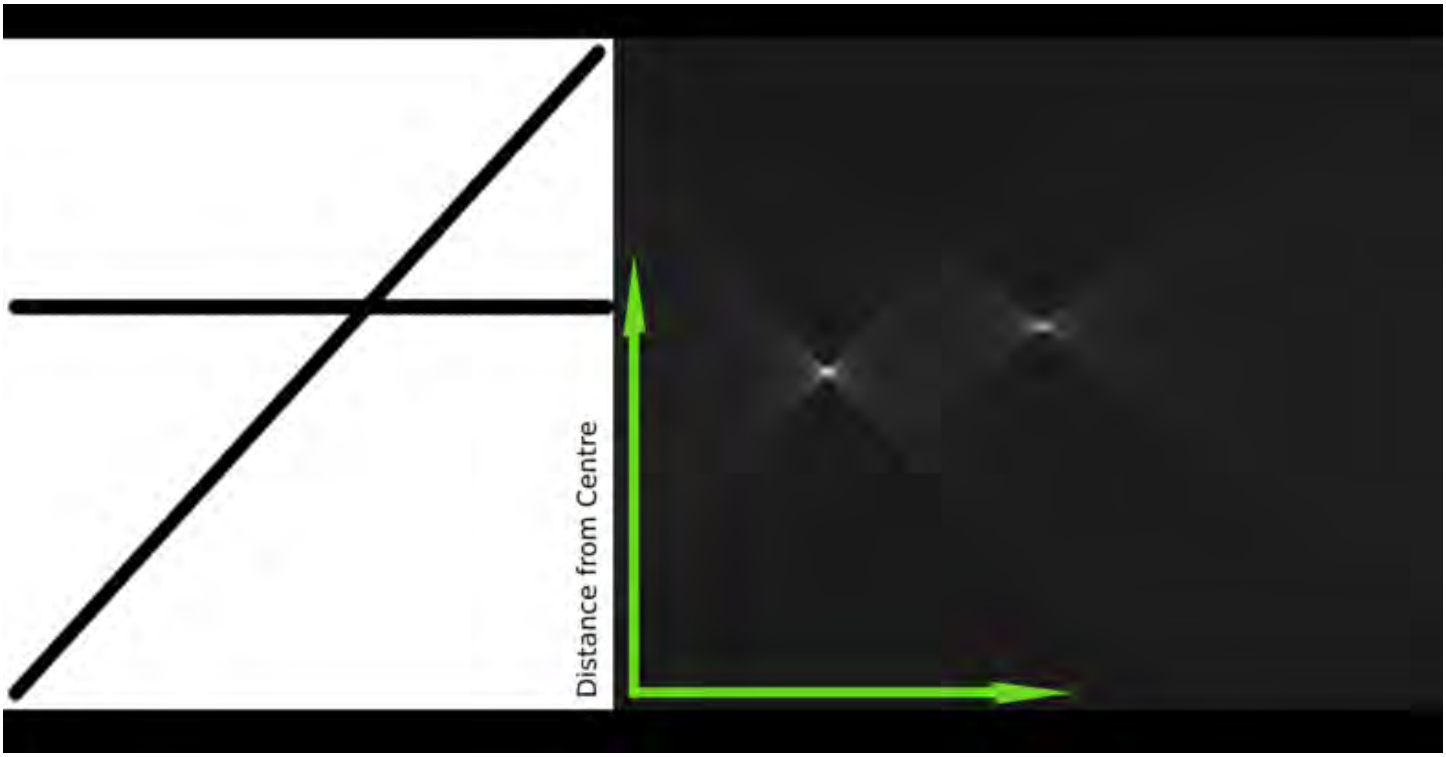
Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6

- For each data point, a number of lines are plotted going through it, all at different angles. These are shown here as solid lines.
- For each solid line a line is plotted which is perpendicular to it and which intersects the origin. These are shown as dashed lines.
- The length and angle of each dashed line is measured. In the diagram above, the results are shown in tables.
- This is repeated for each data point.
- A graph of length against angle, known as a Hough space graph, is then created.



The point where the lines intersect gives a distance and angle. This distance and angle indicate the line which bisects the points being tested. In the graph shown the lines intersect at the purple point; this corresponds to the solid purple line in the diagrams above, which bisects the three points.

The following is a different example showing the results of a Hough transform on a raster image containing two thick lines.



The results of this transform were stored in a matrix. Cell value represents the number of curves through any point. Higher cell values are rendered brighter. The two distinctly bright spots are the intersections of the curves of the two lines. From these spots' positions, angle and distance from image center of the two lines in the input image can be determined.

Variations and extensions

Using the gradient direction to reduce the number of votes

An improvement suggested by O'Gorman and Clowes can be used to detect lines if one takes into account that the local gradient of the image intensity will necessarily be orthogonal to the edge. Since edge detection generally involves computing the intensity gradient
magnitude, the gradient direction is often found as a side effect. If a given point of coordinates (x,y) happens to indeed be on a line, then the local direction of the gradient gives the θ parameter corresponding to said line, and the r parameter is then immediately obtained. In fact, the real gradient direction is only estimated with a given amount of accuracy (approximately ±20°), which means that the sinusoid must be traced around the estimated angle, ±20°. This however reduces the computation time and has the interesting effect of reducing the number of useless votes, thus enhancing the visibility of the spikes corresponding to real lines in the image.

Hough transform of curves, and Generalised Hough transform

Although the version of the transform described above applies only to finding straight lines, a similar transform can be used for finding any shape which can be represented by a set of parameters. A circle, for instance, can be transformed into a set of three parameters, representing its center and radius, so that the Hough space becomes three dimensional. Arbitrary ellipses and curves can also be found this way, as can any shape easily expressed as a set of parameters. For more complicated shapes, the Generalised Hough transform
is used, which allows a feature to vote for a particular position, orientation and/or scaling of the shape using a predefined look-up table.

Using weighted features

One common variation detail. That is, finding the bins with the highest count in one stage can be used to constrain the

range of values searched in the next.

Limitations

The Hough Transform is only efficient if a high number of votes fall in the right bin, so that the bin can be easily detected amid the background noise. This means that the bin must not be too small, or else some votes will fall in the neighboring bins, thus reducing the visibility of the main bin.

Also, when the number of parameters is large (that is, when we are using the Hough Transform with typically more than three parameters), the average number of votes cast in a single bin is very low, and those bins corresponding to a real figure in the image do not necessarily appear to have a much higher number of votes than their neighbors. Thus, the Hough Transform must be used with great care to detect anything other than lines or circles.

Finally, much of the efficiency of the Hough Transform is dependent on the quality of the input data: the edges must be detected well for the Hough Transform to be efficient. Use of the Hough Transform on noisy images is a very delicate matter and generally, a denoising stage must be used before. In the case where the image is corrupted by speckle, as is the case in radar images, the Radon transform
is sometimes preferred to detect lines, since it has the nice effect of attenuating the noise through summation.

History

It was initially invented for machine analysis of bubble chamber photographs.[3]

The Hough transform was patented as U.S. Patent 3,069,654 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=3069654) in 1962 with the name "Method and Means for Recognizing Complex Patterns". This patent uses a slope-intercept parametrization for straight lines, which awkwardly leads to an unbounded transform space since the slope can go to infinity.

The rho-theta parametrization universally used today was first described in

> Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Comm. ACM, Vol. 15*, pp. 11–15 (January, 1972).

although of course was already standard for the Radon transform since at least the 1930s.

O'Gorman and Clowes' variation is described in

> Frank O'Gorman, MB Clowes: Finding Picture Edges Through Collinearity of Feature Points. IEEE Trans. Computers 25(4): 449-456 (1976)

References

1. ^ Use of the Hough Transformation to Detect Lines and Curves in Pictures (http://www.ai.sri.com/pubs/files/tn036-duda71.pdf) .
2. ^ Hough Transform (http://planetmath.org/encyclopedia/HoughTransform.html) .
3. ^ P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

External links

- http://www.rob.cs.tu-bs.de/content/04-teaching/06-interactive/Hough.html - Java Applet + Source for learning the Hough transformation in slope-intercept form
- http://www.rob.cs.tu-bs.de/content/04-teaching/06-interactive/HNF.html - Java Applet + Source for learning the Hough-Transformation in normal form
- http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

- http://imaging.gmse.net/articledeskew.html - Deskew images using Hough transform (Visual Basic source code)

Tarsha-Kurdi, F., Landes, T., Grussenmeyer, P., 2007a. Hough-transform and extended RANSAC algorithms for automatic detection of 3d building roof planes from Lidar data. ISPRS Proceedings. Workshop Laser scanning. Espoo, Finland, September 12-14, 2007.

See also

- Generalised Hough Transform
- Radon Transform
- Fourier Transform

Retrieved from "http://en.wikipedia.org/wiki/Hough_transform"
Categories: Image processing | Computer vision
Hidden categories: Cleanup from November 2007 | All pages needing cleanup

- This page was last modified on 13 February 2008, at 04:00.
- All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.

RANSAC

From Wikipedia, the free encyclopedia

RANSAC is an abbreviation for "RANdom SAmple Consensus". It is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. The algorithm was first published by Fischler and Bolles in 1981.

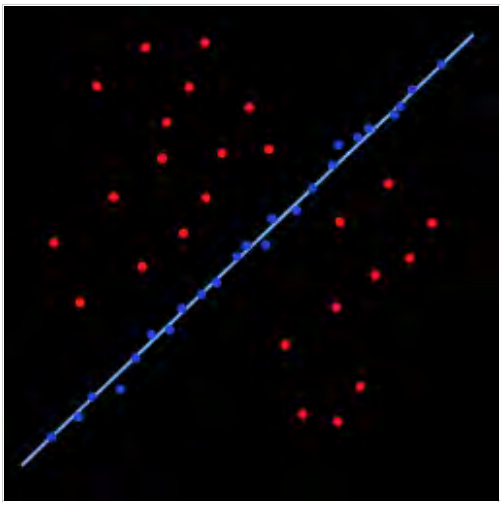
A basic assumption is that the data consists of "inliers", i.e., data points which can be explained by some set of model parameters, and "outliers" which are data points that do not fit the model. In addition to this, the data points can be subject to noise. The outliers can come, e.g., from extreme values of the noise or from erroneous measurements or incorrect hypotheses about the interpretation of data. RANSAC also assumes that, given a (usually small) set of inliers, there exists a procedure which can estimate the parameters of a model that optimally explains or fits this data.

Contents

Example

A simple example is fitting of a 2D line to set of observations. Assuming that this set contains both inliers, i.e., points which approximately can be fitted to a line, and outliers, points which cannot be fitted to this line, a simple least squares method for line fitting will in general produce a line with a bad fit to the inliers. The reason is that it is optimally fitted to all points, including the outliers. RANSAC, on the other hand, can produce a model which is only computed from the inliers, provided that the probability of choosing only inliers in the selection of data points is sufficiently high. There is no guarantee for this situation, however, and there are a number of algorithm parameters which must be carefully chosen to keep the level of probability reasonably high.

A data set with many outliers for which a line has to be fitted.



Fitted line with RANSAC, outliers have no influence on the result.

Overview

The input to the RANSAC algorithm is a set of observed data values, a parameterized model which can explain or be fitted to the observations, and some confidence parameters.

RANSAC achieves its goal by iteratively selecting a random subset of the original data points. These points are *hypothetical inliers* and this hypothesis is then tested as follows:

1. A model is fitted to the hypothetical inliers, i.e. all free parameters of the model are reconstructed from the point set.
2. All other data points are then tested against the fitted model and, if a point fits well to the estimated model, also considered as a hypothetical inlier.
3. The estimated model is reasonably good if sufficiently many points have been classified as hypothetical inliers.
4. The model is reestimated from all hypothetical inliers, because it has only been estimated from the initial set of hypothetical inliers.
5. Finally, the model is evaluated by estimating the error of the inliers relative to the model.

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. In the latter case, we keep the refined model if its error is lower than the last saved model.

The algorithm

The generic RANSAC algorithm works as follows:

```
input:
    data - a set of observed data points
    model - a model that can be fitted to data points
    n - the minimum number of data values required to fit the model
    k - the maximum number of iterations allowed in the algorithm
    t - a threshold value for determining when a data point fits a model
    d - the number of close data values required to assert that a model fits well to data
output:
    best_model - model parameters which best fit the data (or nil if no good model is found)
    best_consensus_set - data point from which this model has been estimated
    best_error - the error of this model relative to the data points

iterations := 0
best_model := nil
best_consensus_set := nil
best_error := infinity
while iterations < k
    maybe_inliers := n randomly selected values from data
    maybe_model := model parameters fitted to maybe_inliers
    consensus_set := maybe_inliers

    for every point in data not in maybe_inliers
        if point fits maybe_model with an error smaller than t
            add point to consensus_set

    if the number of elements in consensus_set is > d
        (this implies that we may have found a good model,
        now test how good it is)
        better_model := model parameters fitted to all points in consensus_set
        this_error := a measure of how well better_model fits these points
        if this_err < best_err
            (we have found a model which is better than any of the previous ones,
            keep it until a better one is found)
            best_model := better_model
            best_consensus_set := consensus_set
            best_error := this_error

    increment iterations

return best_model, best_consensus_set, best_error
```

Possible variants of the RANSAC algorithm includes

- Break the main loop if a sufficiently good model has been found, that is, one with sufficiently small error. May save some computation time at the expense of an additional parameter.
- Compute `this_error` directly from `maybe_model` without re-estimating a model from the consensus set. May save some time at the expense of comparing errors related to models which are estimated from a small number of points and therefore more sensitive to noise.

The parameters

The values of parameters *t* and *d* have to be determined from specific requirements related to the application and the data set, possibly based on experimental evaluation. The parameter *k* (the number of iterations), however, can be determined from a theoretical result. Let *p* be the probability that the RANSAC algorithm in some iteration selects only inliers from the input data set when it chooses the *n* points from which the model parameters are estimated. When this happens, the resulting model is likely to be useful so *p* gives the probability that the algorithm produces a useful result. Let *w* be the probability of choosing an inlier each time a single point is selected, that is,

w = number of inliers in data / number of points in data

A common case is that *w* is not well known beforehand, but some rough value can be given. Assuming that the *n* points needed for estimating a model are selected independently, w^n is the probability that all *n* points are inliers

and $1 - w^n$ is the probability that at least one of the n
points is an outlier, a case which implies that a bad model will be estimated from this point set. That probability to the power of k is the probability that the algorithm never selects a set of n points which all are inliers and this must be the same as $1 - p$. Consequently,

$$1 - p = (1 - w^n)^k$$

which, after taking the logarithm of both sides, leads to

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}$$

It should be noted that this result assumes that the n
data points are selected independently, that is, a point which has been selected once is replaced and can be selected again in the same iteration. This is often not a reasonable approach and the derived value for k should be taken as an upper limit in the case that the points are selected without replacement. For example, in the case of finding a line which fits the data set illustrated in the above figure, the RANSAC algorithm typically chooses 2 points in each iteration and computes `maybe_model`
as the line between the points and it is then critical that the two points are distinct.

To gain additional confidence, the standard deviation or multiples thereof can be added to k. The standard deviation of k is defined as

$$SD(x) = \frac{\sqrt{1 - w^n}}{w^n}$$

Advantages and disadvantages

An advantage of RANSAC is its ability to do robust estimation of the model parameters, i.e., it can estimate the parameters with a high degree of accuracy even when outliers are present in the data set. A disadvantage of RANSAC is that there is no upper bound on the time it takes to compute these parameters. If an upper time bound is used, the solution obtained may not be the most optimal one.

RANSAC can only estimate one model for a particular data set. As for any one-model approach when two (or more) models exist, RANSAC may fail to find either one.

Applications

The RANSAC algorithm is often used in computer vision, e.g., to simultaneously solve the correspondence problem and estimate the fundamental matrix related to a pair of stereo cameras.

References

- Martin A. Fischler and Robert C. Bolles (June 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". *Comm. of the ACM* 24: 381–395. doi:10.1145/358669.358692.
- David A. Forsyth and Jean Ponce (2003). *Computer Vision, a modern approach*. Prentice Hall. ISBN ISBN 0-13-085198-1.
- Richard Hartley and Andrew Zisserman (2003). *Multiple View Geometry in Computer Vision*, 2nd edition,

Cambridge University Press.
- P.H.S. Torr, and D.W. Murray (1997). "The Development and Comparison of Robust Methods for Estimating the Fundamental Matrix". *International Journal of Computer Vision* 24: 271–300. doi:10.1023/A:1007927408552.

External links

- RANSAC Toolbox for Matlab (http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=18555&objectType=file) . A research (and didactic) oriented toolbox to explore the RANSAC algorithm in Matlab. It is highly configurable and contains the routines to robustly fit homographies and lines.

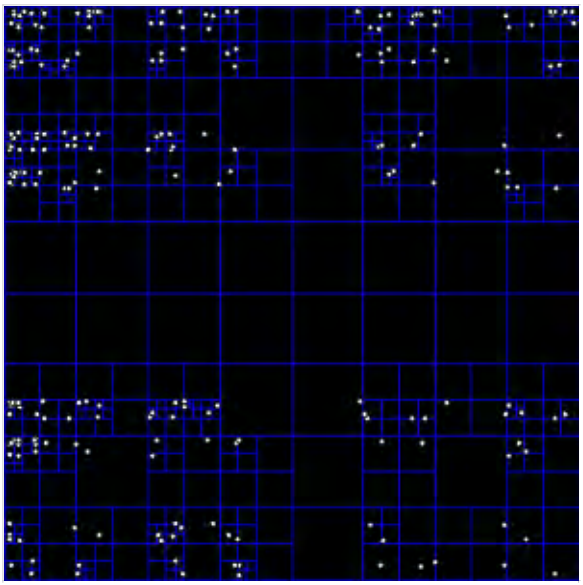
Retrieved from "http://en.wikipedia.org/wiki/RANSAC"
Categories: Algorithms | Geometry in computer vision

- This page was last modified on 19 February 2008, at 22:31.
- All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.

Quadtree

From Wikipedia, the free encyclopedia

A **quadtree** is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have arbitrary shapes. This data structure was named a quadtree by Raphael Finkel and J.L. Bentley in 1974. A similar partitioning is also known as a *Q-tree*. All forms of Quadtrees share some common features:



A point quadtree

- They decompose space into adaptable cells
- Each cell (or bucket) has a maximum capacity. When maximum capacity is reached, the bucket splits
- The tree directory follows the spatial decomposition of the Quadtree

Contents

Types

Quadtrees may be classified according to the type of data they represent, including areas, points, lines and curves.

Quadtrees may also be classified by whether the shape of the tree is independent of the order data is processed. Some common types of quadtrees are:

The region quadtree

The region quadtree represents a partition of space in two dimensions by decomposing the region into four equal quadrants, subquadrants, and so on with each leaf node containing data corresponding to a specific subregion. Each node in the tree either has exactly four children, or has no children (a leaf node). The region quadtree is not strictly a 'tree' - as the positions of subdivisions are independent of the data. They are more precisely called 'tries'.

A region quadtree with a depth of n may be used to represent an image consisting of $2^n * 2^n$ pixels, where each pixel value is 0 or 1. The root node represents the entire image region. If the pixels in any region are not entirely 0s or 1s, it is subdivided. In this application, each leaf node represents a block of pixels that are all 0s or all 1s.

A region quadtree may also be used as a variable resolution representation of a data field. For example, the temperatures in an area may be stored as a quadtree, with each leaf node storing the average temperature over the subregion it represents.

If a region quadtree is used to represent a set of point data (such as the latitude and longitude of a set of cities), regions are subdivided until each leaf contains at most a single point.

Point quadtree

The point quadtree is an adaptation of a binary tree
used to represent two dimensional point data. It shares the features of all quadtrees but is a true tree as the centre of a subdivision is always on a point. The tree shape depends on the order data is processed. It is often very efficient in comparing two dimensional ordered data points, usually operating in O(log n) time.

Node structure for a point quadtree

A node of a point quadtree is similar to a node of a binary tree, with the major difference being that it has four pointers (one for each quadrant) instead of two ("left" and "right") as in an ordinary binary tree. Also a key is usually decomposed into two parts, referring to x and y coordinates. Therefore a node contains following information:

- 4 Pointers: quad[‘NW’], quad[‘NE’], quad[‘SW’], and quad[‘SE’]
- point; which in turn contains:
  - key; usually expressed as x, y coordinates
  - value; for example a name

Edge quadtree

Edge quadtrees are specifically used to store lines rather than points. Curves are approximated by subdividing cells to a very fine resolution. This can result in extremely unbalanced trees which may defeat the object of indexing.

Some common uses of quadtrees

- Image representation
- Spatial indexing
- Efficient collision detection in two dimensions
- View frustum culling of terrain data
- Storing sparse data, such as a formatting information for a spreadsheet or for some matrix calculations

- Solution of multidimensional fields (computational fluid dynamics, electromagnetism)

Quadtrees are the two-dimensional analog of octrees.

Caveats

If geometric subdividing fails to reduce the item count for each quadrant, (e.g., for overlapping data,) QuadTree subpartitioning fails, and the capacity must be breached for the algorithm to continue. For example, if the maximum capacity for a quadrant is 8, and there are 9 points at (0, 0), subpartitioning would produce three empty quadrants, and one containing the original 9 points, and so on. Because the tree must allow more than 8 points in such a quadrant, QuadTrees can approach O(N) complexity for data sets with arbitrary geometry (e.g., maps or graphs).

References

- Raphael Finkel and J.L. Bentley (1974). "Quad Trees: A Data Structure for Retrieval on Composite Keys". *Acta Informatica* 4 (1): 1-9.
- Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf (2000). *Computational Geometry*, 2nd revised edition, Springer-Verlag. ISBN 3-540-65620-0. Chapter 14: Quadtrees: pp.291–306.

See also

- Octree
- Binary space partitioning
- Kd-tree
- R-tree
- UB-tree
- Spatial index

External links

- A discussion of the Quadtree and an application (http://www.cs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html)
- Considerable discussion and demonstrations of Spatial Indexing (http://homepages.ge.ucl.ac.uk/~mhaklay/java.htm)

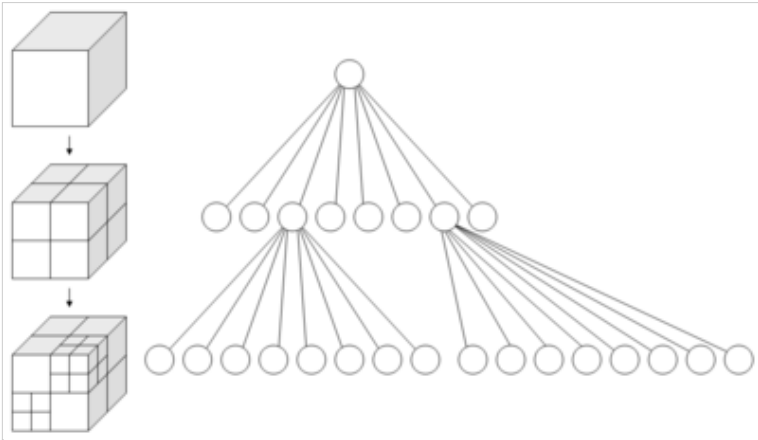
Retrieved from "http://en.wikipedia.org/wiki/Quadtree"
Categories: Trees (structure)

- This page was last modified on 7 December 2007, at 20:42.
- All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.

Octree

From Wikipedia, the free encyclopedia

An **octree** is a tree data structure in which each internal node has up to eight children. Octrees are most often used to partition a three dimensional space by recursively subdividing it into eight octants. Octrees are the three-dimensional analog of quadtrees. The name is formed from *oct* + *tree*, and normally written "*octree*", not "*octtree*".



Left: Recursive subdivision of a cube into octants. Right: The corresponding octree.

Contents

Octrees for spatial representation

Each node in an octree subdivides the space it represents into eight octants. In a point region (PR) octree, the node stores an explicit 3-dimensional point, which is the "center" of the subdivision for that node; the point defines one of the corners for each of the eight children. In an MX octree, the subdivision point is implicitly the center of the space the node represents. The root node of a PR octree can represent infinite space; the root node of an MX octree must represent a finite bounded space so that the implicit centers are well-defined. When used in this way, octrees are a special case of k-dimensional trees, in which the coordinate system is three dimensional.

Common uses of octrees

- Spatial indexing
- Efficient collision detection in three dimensions
- View frustum culling
- Fast Multipole Method

Application to color quantization

The octree color quantization
algorithm, invented by Gervautz and Purgathofer in 1988, encodes image color data as an octree up to nine levels deep. Octrees are used because $2^3 = 8$ and there are three color components in the RGB system. The node index to branch out from at the top level is determined by a formula that uses the most significant bits of the red, green, and blue color components, e.g. 4r + 2g + b. The next lower level uses the next bit significance, and so on. Less significant bits are sometimes ignored to reduce the tree size.

The algorithm is highly memory efficient because the tree's size can be limited. The bottom level of the octree consists of leaf nodes that accrue color data not represented in the tree; these nodes initially contain single bits. If much more than the desired number of palette colors are entered into the octree, its size can be continually reduced by seeking out a bottom-level node and averaging its bit data up into a leaf node, pruning part of the tree. Once sampling is complete, exploring all routes in the tree down to the leaf nodes, taking note of the bits along the way, will yield approximately the required number of colors.

See also

- kd-tree
- Bounding Interval Hierarchy
- Klee's measure problem
- Linear octrees
- Balanced octrees
- Sauerbraten, a 3D game engine which geometry is enterly based on octrees

External links

- Octree Quantization in Microsoft Systems Journal (http://www.microsoft.com/msj/archive/S3F1.aspx)
- Color Quantization using Octrees in Dr. Dobb's (http://www.ddj.com/184409805)
- Color Quantization using Octrees in Dr. Dobb's Source Code (ftp://66.77.27.238/sourcecode/ddj/1996/9601.zip)
- Octree Overview (http://web.cs.wpi.edu/~matt/courses/cs563/talks/color_quant/CQoctree.html)
- C++ implementation (GPL license) (http://nomis80.org/code/octree.html)
- Parallel Octrees for Finite Element Applications (http://sc07.supercomputing.org/schedule/pdf/pap117.pdf)

Retrieved from "http://en.wikipedia.org/wiki/Octree"
Categories: Trees (structure) | Computer graphics data structures | Computer science stubs

- This page was last modified on 23 March 2008, at 15:43.
- All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.