Geospatial Scripting

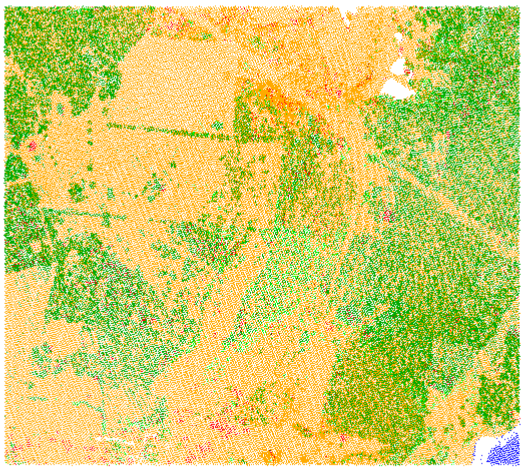
Processing LAS LIDAR Point Files

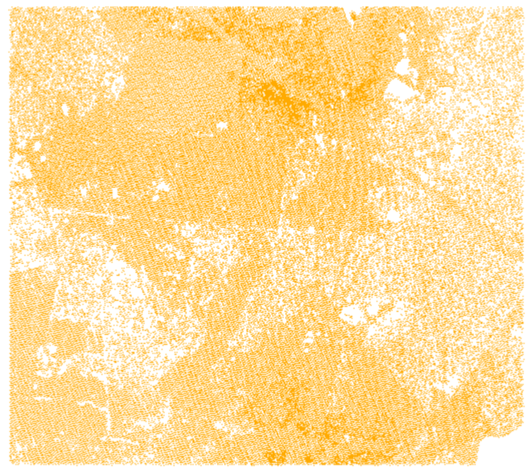
LIDAR point files in the standard LAS file format can be directly displayed and used in this native format in TNTmips Pro. The linked LAS files are represented in TNTmips processes as shape objects. Processing scripts written in the TNT geospatial scripting language (SML) can also access LIDAR point data directly from LAS files, process the points, and create new LAS files to contain the result. The LIDAR points do not have to be imported to an internal TNT geospatial format at any point in this processing. SML's ability to work directly with native LAS files can save a significant amount of processing time that would otherwise be needed to import files that can contain millions of points.

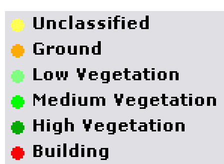
The RVC_SHAPE class in SML is used to represent a linked LAS file. This class includes a MakeLAS() method that is used to create a new LAS file to contain the processing output. This method can create LAS files in the LAS Point Data Record Formats 0, 1, 2, and 3 supported in LAS version 1.2. The input LAS file can be used as a template to set the format of the output file.

LAS files store the spatial coordinates and attributes of each LIDAR point together in a single record within a database table. Each point in a linked LAS file can thus be accessed as a record in a shape database table, and the data can be read, copied, modified, and written to an output LAS file entirely using database constructs (database, table, record, field, ...).

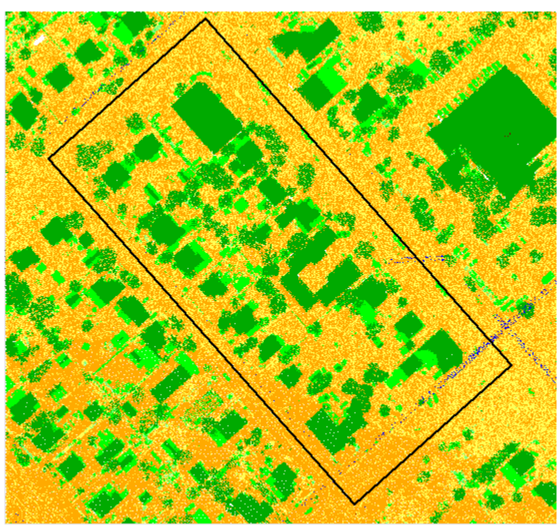
To demonstrate direct processing of LAS files using TNT geospatial scripts, MicroImages has prepared several sample scripts that are excerpted on the reverse of this page and available from the Scripting page at microimages.com (http://www.microimages.com/downloads/scripts.htm). The LAS_GROUND script illustrates how an SML script can use LIDAR point classifications to control processing. It copies only points classified as Ground to a new LAS file. These ground points could then be used as input to generate a bare-earth elevation raster. The LASextractByRegion script uses a region to extract all LIDAR points within the region to a new LAS file.







The illustration above left shows an LAS LIDAR point file with 216,055 points classified into the material categories shown in the legend to the left. Sample script LAS_GROUND was used to extract all points classified as Ground to a new LAS file with 106,466 points (displayed above right).



The illustration above left shows a portion of a large, high spatial-density LAS LIDAR point file of an urban area. The file includes 9,863,071 points, many of which have been classified (using an automated procedure) into ground and vegetation categories; the latter categories also include buildings. The black rectangle outlines a region object enclosing several blocks of buildings. Sample script LASextractByRegion.sml was used to extract all points within this region to a new LAS file with 276,420 points (illustration above right).

Many sample scripts have been prepared to illustrate how you might use the features of the TNT products' scripting language for scripts and queries. These scripts can be downloaded from www.microimages.com/downloads/scripts.htm.

Excerpts of LAS_GROUND.sml

get input LAS shape object

```
class RVC_SHAPE lasIn;
class RVC_OBJITEM objItemIn;
DlgGetObject("Select input LAS shape object:", "Shape", objItemIn,
"ExistingOnly");
lasIn.Open(objItemIn, "Read");
```

get default georeference from input LAS file

```
class RVC_GEOREFERENCE georef;
lasIn.GetDefaultGeoref(georef);
```

open input shape database and main table (table number = 0) for read

```
class RVC_DBASE_SHAPE dbIn;
dbIn.OpenAsSubobject(lasIn, "Read");

class RVC_DBTABLE tableIn;
tableIn.Open(dbIn, 0, "Read");
```

get filepath for output LAS file

```
class FILEPATH path = GetOutputFileName("output.las",
        "Select LAS file to make:", "las");
```

make output LAS file for ground points; use method that takes the RVC_DBTABLE class instance for the existing LAS file to set the same point data record type for the new LAS file

```
class RVC_SHAPE lasOut;
lasOut.MakeLAS(path, georef.GetCoordRefSys(), tableIn);
```

open shape database and main table for write

```
class RVC_DBASE_SHAPE dbOut;
dbOut.OpenAsSubobject(lasOut, "Write");
class RVC_DBTABLE tableOut;
tableOut.Open(dbOut, 0, "Write");
```

record class instances for reading, copying, and writing records

```
class RVC_DBTABLE_RECORD recordIn(tableIn);
class RVC_DBTABLE_RECORD recordOut(tableOut);
class RVC_RECORDNUM recordNum;
```

container for record number

loop through LIDAR point records to find points classified as ground

```
for i = 1 to tableIn.GetNumRecords()
   {
   recordNum.Number = i;
   tableIn.Read(recordNum, recordIn);
```

read record from input LAS

check value in Classification field, copy only ground points

```
   if (recordIn.GetValue("Classification") == 2)
      {
      recordIn.CopyTo(recordOut);
```

copy field values from record in input to a new record for the output

```
      tableOut.AddRecord(recordOut);
      }
   }
```

write the new record to output LAS file

Excerpts of LASextractByRegion.sml

```
class RVC_SHAPE lasIn;
```

input LAS file linked as a shape object

```
class RVC_OBJITEM objItemIn;

class SR_COORDREFSYS crs;
```

Coordinate reference system of input LAS file

```
class RECT3D lasExtents;
```

extents of the input LAS file in its CRS

```
class REGION Reg;
```

region object selected

procedure to select the region object and check that its extents overlap those of the LAS file

```
proc selectRegion () {
   GetInputRegion(Reg);
   Reg.ConvertTo(crs);

   if (lasExtents.Overlaps(Reg.Extents) == 0) {
      PopupMessage("Region selected does not overlap LAS file extents;
         please select another ");
      selectRegion();
      }
   }
```

get input LAS shape object

```
DlgGetObject("Select input LAS shape object:",
      "Shape", objItemIn, "ExistingOnly");
lasIn.Open(objItemIn, "Read");
```

get default georeference from input LAS file

```
class RVC_GEOREFERENCE georef;
lasIn.GetDefaultGeoref(georef);
crs = georef.GetCoordRefSys();
```

get the extents of the LAS shape object for comparison with region

```
lasIn.GetExtents(lasExtents);
```

open input shape database and main table (table number = 0) for read

```
class RVC_DBASE_SHAPE dbIn;
dbIn.OpenAsSubobject(lasIn, "Read");

class RVC_DBTABLE tableIn;
tableIn.Open(dbIn, 0, "Read");
```

call user-defined procedure to select and check the extraction region

```
selectRegion();
```

get filepath for output LAS file

```
class FILEPATH path = GetOutputFileName("output.las",
      "Select LAS file to make:", "las");
```

make output LAS file for extracted points; use method that takes the RVC_DBTABLE class instance for the existing LAS file to set the same point data record type for the new LAS file

```
class RVC_SHAPE lasOut;
lasOut.MakeLAS(path, crs, tableIn);

class RVC_GEOREFERENCE georefOut;
lasOut.GetDefaultGeoref(georefOut);
printf("Input CRS: %s\n", georefOut.GetCoordRefSys().Name);
```

open shape database and main table for write

```
class RVC_DBASE_SHAPE dbOut;
dbOut.OpenAsSubobject(lasOut, "Write");

class RVC_DBTABLE tableOut;
tableOut.Open(dbOut, 0, "Write");
```

record class instances for reading, copying, and writing records

```
class RVC_DBTABLE_RECORD recordIn(tableIn);
class RVC_DBTABLE_RECORD recordOut(tableOut);
class RVC_RECORDNUM recordNum;
```

container for record number

```
class STATUSCONTEXT status;
class STATUSDIALOG statusDLG;
statusDLG.Create();
status = statusDLG.CreateContext();
status.BarInit(tableIn.GetNumRecords(), 0);
status.Message = "Processing LIDAR points...";
class POINT2D pt;
```

loop through the LIDAR point records to find points inside region

```
for i = 1 to tableIn.GetNumRecords()
   {
   status.BarUpdate(i, tableIn.GetNumRecords(), 0);
   recordNum.Number = i;
   tableIn.Read(recordNum, recordIn);
```

read record from input LAS to record container

```
   pt.x = recordIn.GetValue("X");
   pt.y = recordIn.GetValue("Y");
```

get map position of current point

check that these map coordinates are inside the extraction region

```
   if (Reg.IsPointInside(pt) )
      {
      recordIn.CopyTo(recordOut);
```

copy field values from record in input to a new record for the output

```
      tableOut.AddRecord(recordOut);
      }
   }
statusDLG.Destroy();
```

write the new record to output LAS file