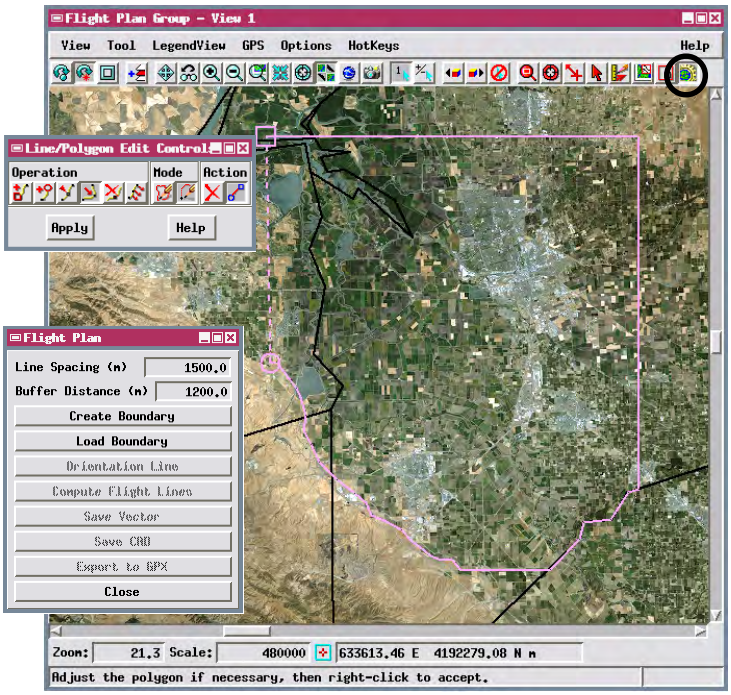
Sample Geospatial Script

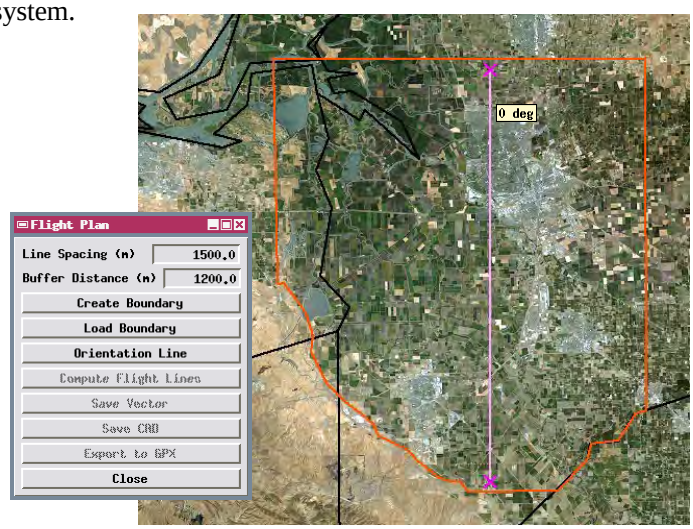
Flight Planning

MicroImages has created a sample geospatial tool script that provides an interactive automated procedure for laying out a pattern of parallel, equally-spaced flight lines for aerial operations. This tool script can be added to 2D View windows in the TNTmips Display process and TNTview, and can be saved with a layout for use in an atlas in TNTatlas.

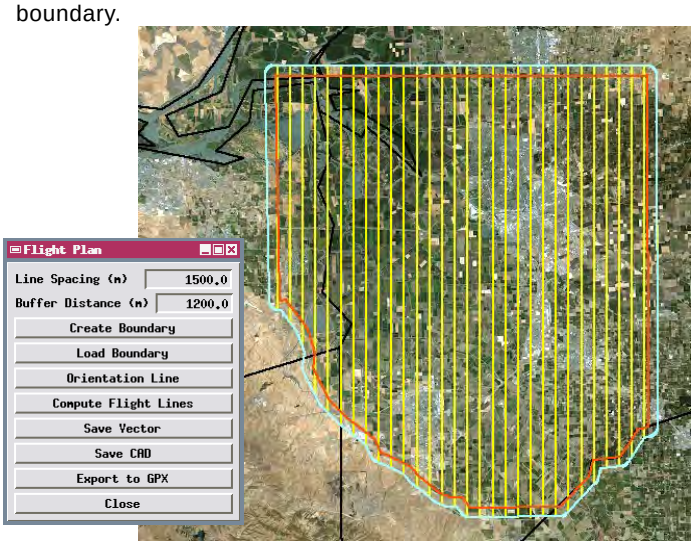
The Flight Planning tool script (excerpted on the reverse side of this page) creates a set of parallel flight lines that completely cover the designated target area with the specified direction and line spacing. The script provides interactive tools for drawing the target polygon in the View over any desired reference geospatial data (such as a map or satellite image) and for drawing a reference line to indicate the flight line direction. These tools are activated by controls on a custom dialog window created and opened by the script. You also have the option of loading the area boundary from an existing vector, CAD, or region object. The dialog provides fields for entering the flight line spacing and a buffer distance to extend the flight lines around the target area polygon. After these parameters and the boundary polygon and reference direction line are set, pressing the Compute Flight Lines button generates a buffer zone polygon around the target area boundary and the parallel flight lines are created and clipped to this buffer polygon. You can save the resulting flight lines, target area boundary, and buffer boundary in separate vector or CAD objects. The flight lines can also be exported to GPS Exchange (GPX) format so the line data can be imported by and used to control a GPS-based navigation system.



After you enter the flight line spacing and buffer distance values in the Flight Plan dialog window, press the Create Boundary button to activate a polyline tool that you can use to draw the outline of the flight area in the View. A Line/Polygon Edit Controls window opens automatically with the tool to allow you to make adjustments to the shape of the polygon if needed. When you have correctly placed the boundary polygon, right-click or press the Apply button on the Edit Controls window to accept and use it.



Pressing the Orientation Line button activates a line tool that allows you to draw a straight reference line in the view to indicate the flight line direction (its compass direction is shown automatically by a pop-in ToolTip). Drag either end of the line tool to adjust its position and direction, then right-click to accept the line. This line is used as the reference line for creating the set of parallel flight lines at the designated line spacing. The end points of this orientation line can be anywhere inside or outside of the desired flight area, as all of the flight lines the script creates are automatically extended to or clipped at the specified buffer distance around the area boundary.



Pressing the Compute Flight Lines button creates the buffer zone (cyan in this illustration) around the boundary and the set of flight lines (yellow). You can then save the flight lines, boundary, and buffer boundary as separate vector or CAD objects, and export the flight lines to a GPS Exchange (GPX) file for use in navigation.

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 Flight Planning Tool Script (FlightPlan.sml)

```
proc PolyLineActivate() {
   LineTool.Managed = 0;
   PolyLineTool.Managed = 1;
   PolyLineTool.HasPosition = 0;
   View.SetMessage("Draw polygon in view outlining the area boundary.");
   }
```

Procedure called when Boundary button on dialog is pressed; activates polyline tool.

```
proc OnPolyLineToolApply() {
   class POLYLINE polyline;
   polyline = PolyLineTool.GetPolygon();
   polyline.AppendVertex(polyline.GetVertex(0));

   class TRANSPARM ScreenToView = ViewGetTransViewToScreen(View, 1);
   class TRANSPARM ViewToMap = ViewGetTransMapToView(View, crs, 1);

   polyline.ConvertForward(ScreenToView);
   polyline.ConvertForward(ViewToMap);

   VectorAddPolyLine(vectorPoly, polyline);
   PolyLineTool.HasPosition = 0;

   LayerDestroy(vectorPolyLayer);
   vectorPolyLayer = GroupQuickAddVectorVar(group, vectorPoly);
   ViewRedraw(View);

   orientationBtn.SetEnabled(1);
   View.SetMessage("Press Orientation Line button to set direction of flight lines.");
   }
```

Procedure called when user right-clicks to accept the field boundary polygon.

```
proc LineActivate() {
   PolyLineTool.Managed = 0;
   LineTool.Managed = 1;
   LineTool.HasPosition = 0;
   View.SetMessage("Drag a line in the view in the desired flight line direction.");
   }
```

Procedure called when Orientation button on dialog is pressed; activates line tool.

```
proc OnLineToolApply () {
   array numeric xpoints[5];
   array numeric ypoints[5];

   ClearVector(vectorLine);
   ClearVector(vectorPoint);

   class TRANSPARM ScreenToView = ViewGetTransViewToScreen(View, 1);
   class TRANSPARM ViewToMap = ViewGetTransMapToView(View, crs, 1);

   coordstart = TransPoint2D(TransPoint2D(LineTool.start, ScreenToView, 0),
      ViewToMap, 0);
   coordend = TransPoint2D(TransPoint2D(LineTool.end, ScreenToView, 0),
      ViewToMap, 0);

   xpoints[1] = coordstart.x;  xpoints[2] = coordend.x;
   ypoints[1] = coordstart.y;  ypoints[2] = coordend.y;

   VectorAddLine(vectorLine, 2, xpoints, ypoints);
   LineTool.HasPosition = 0;
   VectorAddPoint(vectorPoint, coordstart.x, coordstart.y);
   VectorAddPoint(vectorPoint, coordend.x, coordend.y);

   LayerDestroy(vectorLineLayer);
   LayerDestroy(vectorPointLayer);
   vectorLineLayer = GroupQuickAddVectorVar(group, vectorLine);
   vectorPointLayer = GroupQuickAddVectorVar(group, vectorPoint);
   ViewRedraw(View);

   computeBtn.SetEnabled(1);
   View.SetMessage("Press the Compute Flight Lines button to create flight lines.");
   }
```

Procedure called when user right-clicks to accept the orientation line.

clear everything but boundary polygon and buffer polygon

add line

add line to view

```
proc OnExport() {
   class SR_COORDREFSYS latloncrs;
   latloncrs.Assign("Geographic2D_WGS84_Deg");
   class REGION2D layerregion = group.ActiveLayer.MapRegion;
   layerregion.ConvertTo(latloncrs);
   class RECT extents = layerregion.Extents;

   class TRANSPARM MapToGeog;
   MapToGeog.InputCoordRefSys = group.ActiveLayer.MapRegion.CoordRefSys;
   MapToGeog.OutputCoordRefSys = latloncrs;

   class FILE fOut;
   fOut = GetOutputTextFile("c:/default.txt", "Select Text file", "gpx");

   fwritestring(fOut, "<?xml version=\"1.0\"?>\n");
   fwritestring(fOut, "<gpx\n");
   fwritestring(fOut, " version=\"1.0\"\n");
   fwritestring(fOut, " creator=\"MicroImages - http://www.microimages.com\"\n");
   fwritestring(fOut, " xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
      instance\"\n");
   fwritestring(fOut, " xmlns=\"http://www.topografix.com/GPX/1/0\"\n");
   fwritestring(fOut, " xsi:schemaLocation=\"http://www.topografix.com/GPX/1/0
      http://www.topografix.com/GPX/1/0/gpx.xsd\">\n");

   if (extents.x1 < extents.x2) {
      minlon = extents.x1;       maxlon = extents.x2;
      }
   else {
      minlon = extents.x2;       maxlon = extents.x1;
      }
   if (extents.y1 < extents.y2) {
      minlat = extents.y1;       maxlat = extents.y2;
      }
   else {
      minlat = extents.y2;       maxlat = extents.y1;
      }
   class STRING bounds$ = sprintf("<bounds minlat=\"%.6f\" minlon=\"%.6f\"
         maxlat=\"%.6f\" maxlon=\"%.6f\"/>\n", minlat, minlon, maxlat, maxlon);
   fwritestring(fOut, bounds$);

   class STRINGLIST pointlist;                class STRING point$, name$;
   class POINT2D pt;  class GEOREF georef = GetGeorefObject(vectorInitLine);

   for i = 1 to NumVectorLines(vectorInitLine) {
      for j = 1 to NumVectorPoints(vectorPoint) {
         pt.x = vectorPoint.point[j].Internal.x;
         pt.y = vectorPoint.point[j].Internal.y;

         if (FindClosestLine(vectorInitLine, pt.x, pt.y, georef, 0.0001) == i) {
            point$ = sprintf("<wpt lat=\"%.6f\" lon=\"%.6f\">\n",
                  MapToGeog.ConvertPoint2DFwd(pt).y,
                  MapToGeog.ConvertPoint2DFwd(pt).x);
            pointlist.AddToEnd(point$);
            }
         }
   pointlist.Sort();
   for k = 1 to (pointlist.GetNumItems()) {
      fwritestring(fOut, pointlist[k-1]);
      name$ = sprintf(" <name>LINE%i #%i</name>\n", i, k);
      fwritestring(fOut, name$);
      fwritestring(fOut, "</wpt>\n");
      }
   pointlist.Clear();
   }
   fwritestring(fOut, "</gpx>\n");
   fclose(fOut);
   }
```

write GPX header

determine bounding lat/lon

step through each line

for all points

find points that lie on current line and add these points

output points in pointlist