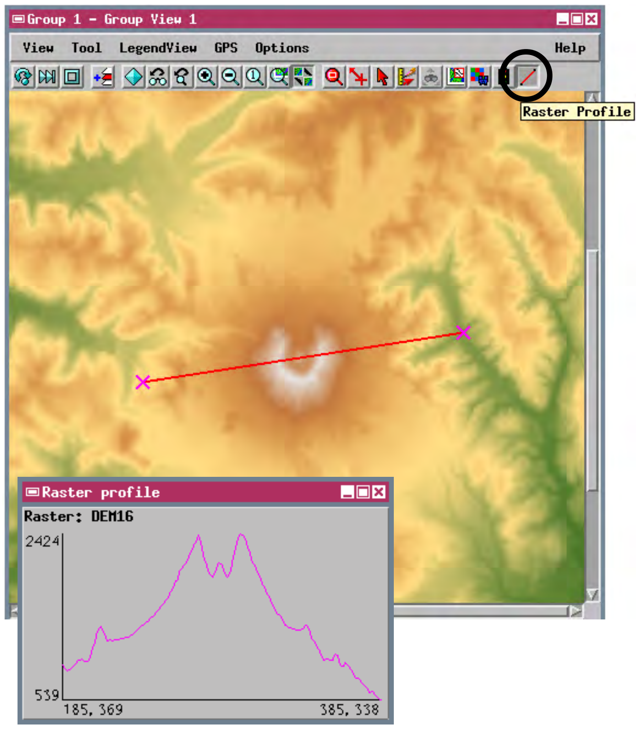
## Sample SML Tool Script

# Raster Profile

The Raster Profile sample script shows how you can set up an SML Tool Script to record and process raster values along a user-drawn line. This script, excerpts of which are shown on the other side of this page, provides a standard two-point line tool that enables the user to draw a straight line anywhere in the View window. The script reads cell values from a raster in the active layer and creates a graph in a separate dialog window showing the corresponding vertical profile of the raster (cell value versus position along the line).

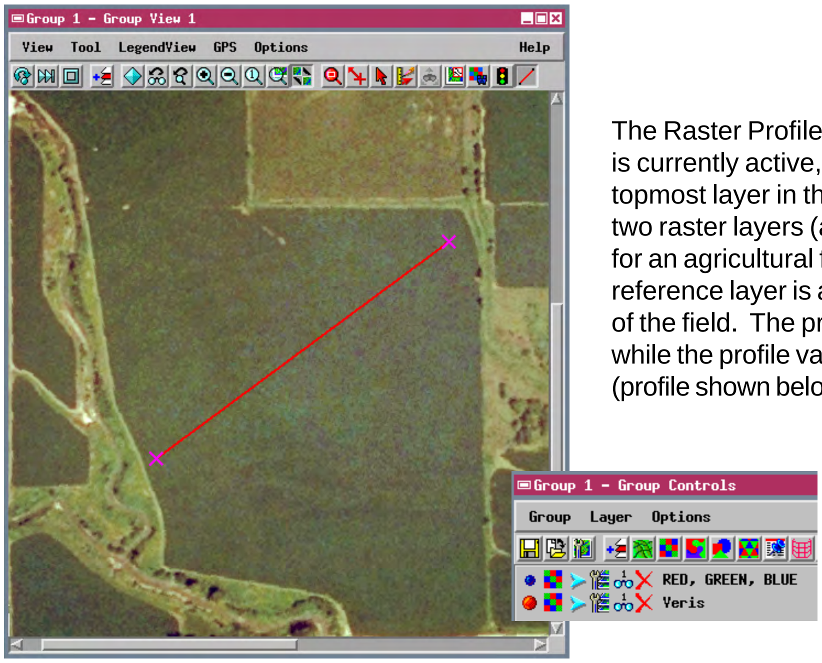
Although the on-screen profile plot is the end result in this example, you could easily modify the script to apply other processing to the profile data. For example, profile positions from a georeferenced raster could be converted from raster coordinates to map coordinates, the profile data could be output to a text file, and an external program could be launched to create a formatted profile plot for printing. You would implement these capabilities by adding appropriate program statements to the procedure that is called when the right mouse button is pressed [cbToolApply()].

The profile drawn by the Raster Profile script is an example of the type of graphic output that you can create in a dialog window using simple SML drawing functions.

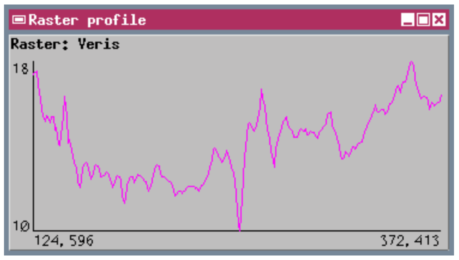


Profile from a digital elevation model raster. The vertical graph axis is labeled with the minimum and maximum cell values along the profile line. The horizontal axis is labeled with the endpoint positions in raster coordinates (column, line).

Many functions are provided to draw lines, unfilled and filled geometric shapes, and text. Related functions allow you to set colors and fonts for the other drawing functions to use.



The Raster Profile script operates on the layer in the View window that is currently active, but the active layer does not have to be the topmost layer in the view. In the View window to the left, the lower of two raster layers (a surface fit to electrical conductivity measurements for an agricultural field) has been set as the active layer. The upper reference layer is an RGB raster set showing a color aerial photograph of the field. The profile line can then be drawn on a familiar image, while the profile values are read from the underlying active raster (profile shown below).



Macro and Tool Scripts can be created using SML in any TNTmips process that uses a View window (Options / Customize from the View window menu bar). These scripts are then available from an icon, which you select or design, on the toolbar. Sample scripts have been prepared to illustrate how you might use these features, which are available only in TNTmips 6.4 or later, to assist with specific tasks you perform on a regular basis. If possible, the full script is printed below for your quick perusal. When a script is too long to fit on one page, key sections are reproduced below. All sample Tool and Macro Scripts illustrated can be found in their entirety on your TNT products CD-ROM in the folder in which you installed TNTmips 6.4. These scripts, among others, can be downloaded from the SML script exchange at www.microimages.com/sml/ftpsmllink/TNT_Products_V6.4_CD.

# Script for Raster Profile (regstats.sml)

callback for drawing area expose; draws text and graph

clear the drawing area and redraw text

draw graph axes

draw labels for graph axes

draw profile

```
proc cbRedraw() {
    if (gc == 0) return;
    ActivateGC(gc);

    SetColorName("gray75");
    FillRect(0, 0, da.width, da.height);
    SetColorName("black");
    DrawInterfaceText(sprintf("Raster: %s", rasterName$), 0, 10);

    if (doGraph == 0) return;

    local string min$, max$;
    local numeric size;
    local class POINT2D graph1, graph2, graph3;

    min$ = sprintf("%d", min);
    max$ = sprintf("%d", max);
    size = strlen(max$);

    graphx[1] = size*6.5+5;
    graphy[1] = 20;
    graphx[2] = graphx[1];
    graphy[2] = da.height - 15;
    graphx[3] = da.width - 5;
    graphy[3] = graphy[2];
    DrawPolyLine(graphx, graphy, 3);

    DrawTextSetFont("stork");
    DrawTextSetHeight(10);

    DrawTextSimple(max$, 0, graphy[1]+10);
    DrawTextSimple(min$, (size - strlen(min$))*6.5, graphy[2]);
    DrawTextSimple(sprintf("%d, %d", startpoint.x, startpoint.y),
        graphx[1], da.height-3);
    str$ = sprintf("%d, %d", endpoint.x, endpoint.y);
    DrawTextSimple(str$, graphx[3] - strlen(str$)*6.5, da.height-3);

    SetColorName("magenta");

    local numeric xscale, yscale, prevnull;
    xscale = (graphx[3] - graphx[2] + 1) / (count - 1);
    yscale = (graphy[2] - graphy[1]) / (max - min);

    if (value[1] < 0 || value[1] >= 0) {
        if (max - min == 0) {
            DrawPoint(graphx[2], graphy[1] + (graphy[2]-graphy[1])*.5);
            }
        else
            DrawPoint(graphx[2], graphy[2]-(value[1]-min)*yscale);
        }
    for i = 2 to count {
        if (value[i] < 0 || value[i] >= 0) {
            if (!(value[i-1] < 0 || value[i-1] >= 0)) {
                if (max - min == 0) {
                    DrawPoint(graphx[2]+(i-1)*xscale, graphy[1] +
                        (graphy[2]-graphy[1])*.5);
                    }
                else
                    DrawPoint(graphx[2]+(i-1)*xscale, graphy[2]-
                        (value[i]-min)*yscale);
                }
            else {
                if (max - min == 0) {
                    DrawTo(graphx[2]+(i-1)*xscale, graphy[1] +
                        (graphy[2]-graphy[1])*.5);
                    }
                else
                    DrawTo(graphx[2]+(i-1)*xscale, graphy[2]-(value[i]-
                        min)*yscale);
                }
            }
        }
    }
```

callback for when user clicks the right mouse button on the line tool (or clicks apply)

get start and end point of tool line

transform to map coordinates

get x and y step values

loop on line to generate point list

if the raster value was retrieved from a null cell or a coordinate outside the raster extents, value[i] will be 'not a number'; therefore testing if it is < 0 or >= 0 determines whether it is a valid value

```
proc cbToolApply() {

    if (checkLayer()) {
        startpoint = tool.start;
        endpoint = tool.end;

        startpoint = TransPoint2D(startpoint,
            ViewGetTransLayerToScreen(View, rasterLayer, 1));
        endpoint = TransPoint2D(endpoint,
            ViewGetTransLayerToScreen(View, rasterLayer, 1));

        local numeric ystep, xstep;
        local class POINT2D cursor;
        cursor = startpoint;
        count = 0;
        ystep = endpoint.y - startpoint.y;
        xstep = endpoint.x - startpoint.x;
        if (xstep == 0 && ystep == 0)
            return;
        if (abs(xstep) > abs(ystep)) {
            ystep = ystep / abs(xstep);
            xstep = xstep / abs(xstep);
            if (ystep == 0) {
                count = abs(endpoint.x - startpoint.x);
                }
            else
                count = abs((endpoint.y - startpoint.y) / ystep);
            }
        else {
            xstep = xstep / abs(ystep);
            ystep = ystep / abs(ystep);
            if (xstep == 0) {
                count = abs(endpoint.y - startpoint.y);
                }
            else

                count = abs((endpoint.x - startpoint.x) / xstep);
            }
        count = count + 1;
        doGraph = 1;

        max = -1000000;
        min = 1000000;

        for i = 1 to count {
            value[i] = targetRaster[round(cursor.y), round(cursor.x)];

            if (value[i] < 0 || value[i] >= 0) {
                if (value[i] > max)
                    max = value[i];
                if (value[i] < min)
                    min = value[i];
                }
            cursor.x = cursor.x + xstep;
            cursor.y = cursor.y + ystep;
            }
        if (max == -1000000 && min == 1000000) {
            max = 0;
            min = 0;
            }
        cbRedraw();
        }
    }

# Called when the close button is pressed.  Closes the dialogs.
proc cbClose() {
    tool.Managed = 0;
    DialogClose(form);
    if (setDefaultWhenClose) {
        setDefaultWhenClose = false;
        View.SetDefaultTool();
        }
    }
```