Gridding with GRASS GIS

Jed Frechette <jdfrec@unm.edu>

6 March 2008

Generateing a 2.5D raster elevation model that can be more easily analyzed using standard GIS applications is necessary for some applications. This tutorial will describe one method for creating such a DEM using GRASS GIS. This tutorial does not attempt to explain all options available, but it should give you a basis for further investigations. Readers are encouraged to consult the GRASS documentation for further details while reading this guide.

The general procedure is to import a point cloud from a text file then aggregate the data in grid cells using univariate statistics. Depending on the density of the point cloud and the resolution of the grid, this may be sufficient, possibly with the addition of some minor hole filling. However, if the point cloud is sparse or the grid cells are small we may add the additional step of approximating the surface using a regularized spline with tension.

For conciseness all examples are given using GRASS's command line interface, however, in many cases it will be easier for new users to specify the commands using their graphical interfaces. The GUIs can be accessed either by selecting the appropriate tool from GRASS's menus or by typing the command name with no arguments, e.g. instead of running `r.in.xyz` followed by a long list of parameters simply run `r.in.xyz` and a GUI will launch allowing you to interactively select the command's parameters.

1 Preparing the data

For this tutorial we will be using a composite point cloud that was built from multiple scans and georeferenced in PolyWorks. The point cloud was export as a simple space separated text file (Figure 1), so this workflow can be applied to data from many sources. For example, if the entire target was captured in a single scan we could export a text file directly from Optech's parser and load it in to GRASS with no intermediate steps[1]. Our input file has no header and its first few lines

[1] This is not quite true, there are a couple issues that need to be addressed if you are using Optech generated text files directly. First, many scans will contain multiple ROIs, which are exported as separate files that need to be combined into a single file prior to import. Second missing values are indicated by assigning values of 0.000 to the x, y, and z coordinates and it would be nice to filter out these null points prior to loading the data in GRASS. Fortunately users of UNIX like operating can address both issues with a single terminal command:

```
cat scan_files*.xyz | grep -v "0.000 0.000 0.000" >slope_pts.txt
```

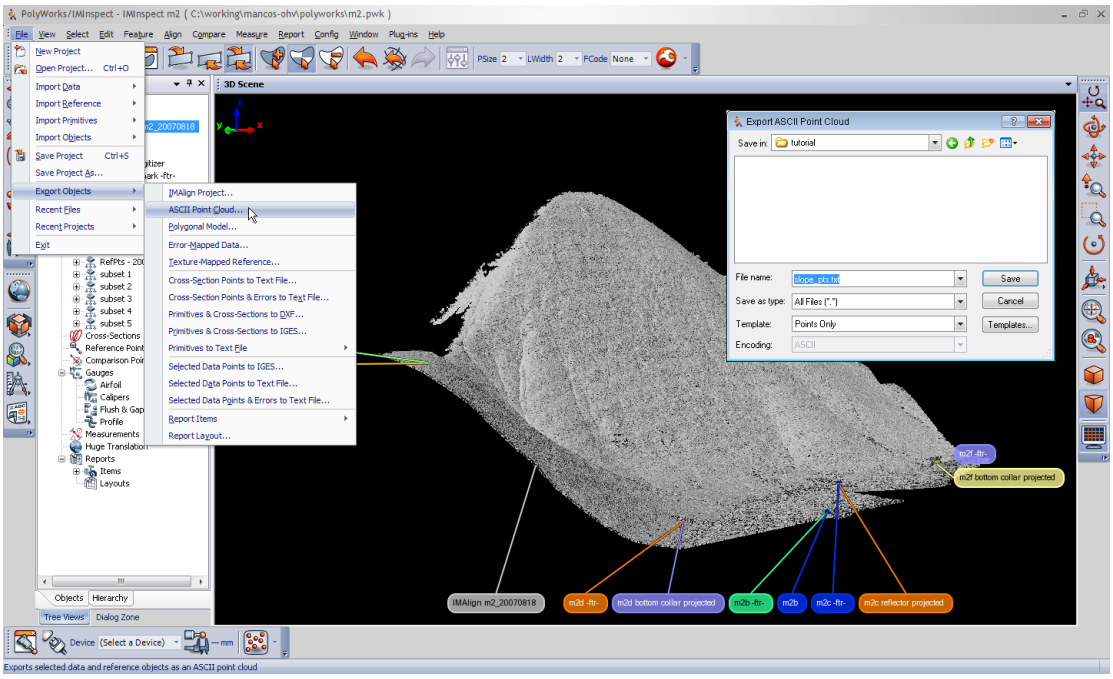

Figure 1: Once the initial processing in PolyWorks is complete the data is exported as text file. A variety of data for each point can be included in this file by selecting a different template, however, for this exercise we are only interested in topography so we will use the default "Points Only" template.

are:

```
247509.339423 4278310.697355 1745.803980
247509.348719 4278310.664623 1745.812589
247509.339915 4278310.705417 1745.803036
247509.337834 4278310.705513 1745.804889
```

This data set has been georeferenced to NAD83 / UTM zone 13N so the first column gives eastings (x), the second northings (y), and the third elevation (z) in meters. The order of the columns and the separator used is not important as these can be specified during the import process.

2 Setting up GRASS

The projection, units, extents and resolution, of the DEMs we are about to create will be determined by the GRASS project used to create them, therefore we need to build a new project location with

The output of this command, `slope_pts.txt`, is a text file produced by combining all of the valid points in files with names matching the pattern `scan_files*.xyz`.

Figure 2: To create a location based on a standard projection look up the EPSG code and specify the name for the new location.

appropriate settings. If your data is georeferenced the easiest way to do this is to either use an existing GIS coverage with the correct projection to define a new project or specify an EPSG code. We know that our example data is in NAD83 / UTM zone 13N so it is a simple matter to look up the EPSG projection code on http://spatialreference.org and use it to create a new location (Figure 2). If you are working with data in scanner coordinates you will need to define your own projection. Once the new location is created select the default PERMANENT mapset or create a new mapset to store your data and enter GRASS.

We will be using GRASS's `r.in.xyz` tool to load and aggregate the data. The resolution and bounds of the raster map generated by `r.in.xyz` are controlled by GRASS's current region settings. To display the current region settings run `g.region` with the print flag:

```
g.region -p
```

In all likelihood we will need to adjust the default settings to something more reasonable before continuing. To begin with we need to determine what the bounds of our point cloud are. We could manually inspect the point cloud, however, `r.in.xyz` is able to calculate the bounds of our data for us so lets allow it to do the work.

```
r.in.xyz -g input=slope_pts.txt output=no_map_created fs=' '
```

This command will not import any data it simply loads the data into memory, calculates the bounds, and prints them to the screen. Note that we need to specify a name for the output map even though no such map will be created. The `fs` parameter specifies the separator between columns, in our case a space. Which data columns correspond to x, y, and z values may also need to be specified. In this case the default settings of `x=1`, `y=2`, and `z=3` are in agreement with our file structure so we don't need to make any changes. Another possible way to import this data would be as a topographic map of a vertical surface with z representing a horizontal distance. In this case

we might query the bounds of the data set using:

```
r.in.xyz -g input=slope_pts.txt output=no_map_created fs=' ' x=1 y=3 z...
    ...=2
```

The output of `r.in.xyz -g` is formatted so that it can be used as parameters for `g.region`:

```
g.region n=4278371.968620 s=4278308.874365 e=247556.186244 w...
    ...=247472.796181 b=1744.971427 t=1783.413267
```

In addition to setting the bounds we also need to set the resolution. The point cloud we are using for this tutorial is very dense and we are interested in micro topographic features such as rills and incipient gullies so we will try to keep the size of our grid cells as small as possible. Lets begin by setting it to 2 cm, slightly above the nominal accuracy of the scanner used to collect the data:

```
g.region res=0.02 -a
```

The resolution chosen in this step is not critical as we will have an opportunity to adjust it later. Finally verify that your region settings are correct by passing the `-p` parameter to `g.region` and we are ready to import some data.

3 Import ASCII and generate DEM

Now that setup is complete we can proceed to importing the data and generating a DEM. First we should generate a map containing the number of points that will contribute to each grid cell, in order to evaluate the appropriateness of our grid resolution.

```
r.in.xyz input=slope_pts.txt output=n_grd fs=' ' method=n
```

Note that, in addition to specifying the statistical method used for calculating cell values, we have removed the `-g` parameter and specified a real name for the output map. Next evaluate the grid resolution by plotting and examining the generated map (Figure 3). Due to the variability in point densities the following custom color table was applied using `r.colors`.

```
nv black
0 black
1 43 131 186
15 171 221 164
30 255 255 191
45 253 174 97
60 215 25 28
100% 215 25 28
```

The range of values shown on the legend was also limited to 0-100. Although there are numerous cells that contain no data points a resolution of 2 cm should provide enough data points for interpolating a a detailed surface using a regularized spline with tension. Alternatively we could increase the region's resolution, to 10 cm for example, in order to generate a map requiring little or or no interpolation.

Although there is little vegetation on this slope we do want to try and filter out the sparse grass that is present, as well as the tripod mounted targets placed in the scene. A simple way to do that is

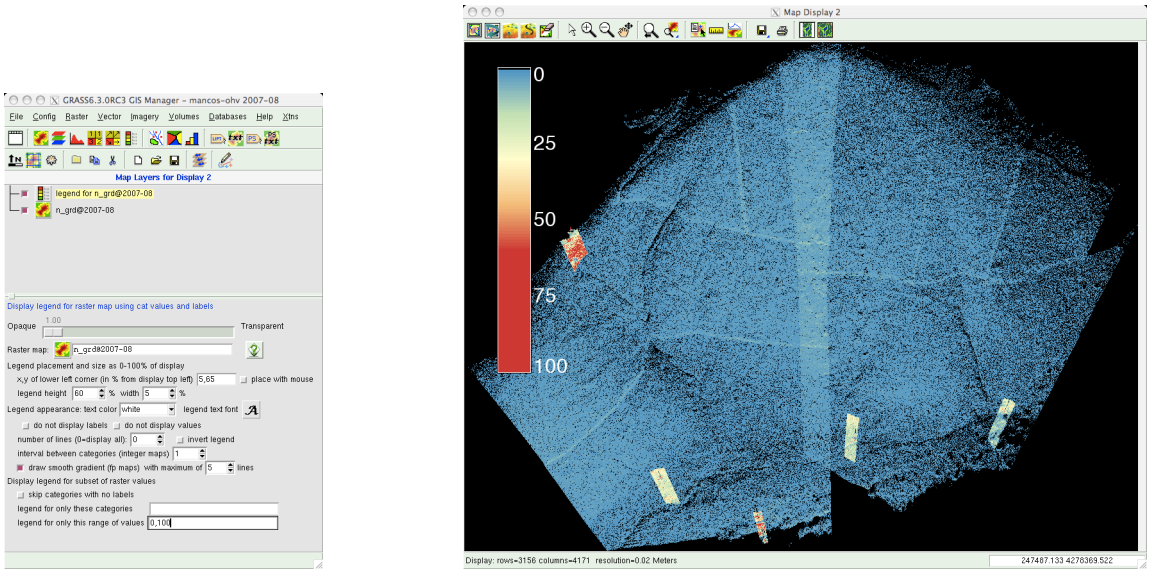

Figure 3: Layers showing the number of points in each grid cell and the associated legend were added to the GIS Manager. Regions with high point densities are where ROIs overlap and the highest densities are achieved where detailed scans of georeferenced targets were collected.

to use the lowest elevation point in each grid cell to set the elevation of the cell. This can be done by setting the `method=min` parameter of `r.in.xyz`.

```
r.in.xyz input=slope_pts.txt output=min_grd fs=' ' method=min
```

This will give us a topographic map that we can use as the basis for an interpolated surface. Next we need to convert the raster map generated by `r.in.xyz` back to a vector point map that can be used as input for the interpolation function.

```
r.to.vect -z feature=point in=min_grd out=min_pt
```

The round trip from points to raster back to points may seem unnecessary but it performs a couple of useful functions. First, as already mentioned, it helps to remove some vegetation and unwanted vertical features. Second it greatly reduces the number of points that will be passed to the interpolation function. For this data set only 6.27 million data points were used for interpolation, compared to the 15.7 million in the original point cloud. Given that the interpolation is the most computationally expensive part of this whole process the time savings achieved by this data reduction is significant. On the other hand there are plenty of reasons why you would want to interpolate a surface directly from the raw point cloud, or perhaps a point cloud that has been preprocessed in another application. That workflow is actually much simpler as all that is needed prior to interpolation is to import the points.

```
v.in.ascii -btz input=slope_pts.txt output=min_pt
```

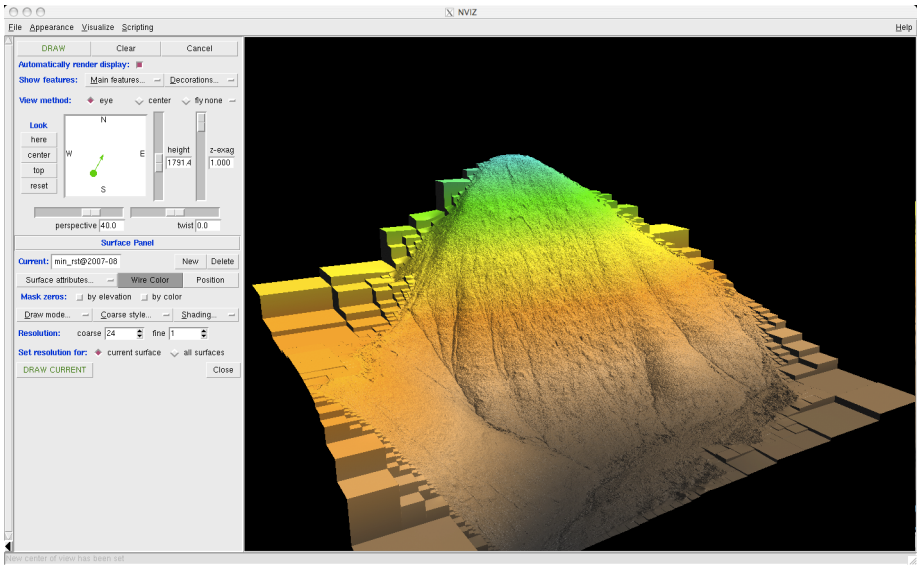

Figure 4: GRASS includes NVIZ, a rudimentary 3D visualization application, that allows us to inspect the interpolated surface generated from our point cloud data. Additional processing is necessary to remove the large facets near the edges of the data set where no data was available.

Regardless of how the vector map `min_pt` is created we can interpolate a raster map from it using a regularized spline with tension.

```
v.surf.rst layer=0 in=min_pt elev=min_rst
```

Due to the highly variable point density this command will issue numerous warnings about taking too long to find points for interpolation, warnings that can be safely ignored. Even on a fast computer interpolating a surface from several million points is likely to take several hours so now would be a good time to go for lunch.

The tool `v.surf.rst` accepts various parameters for adjusting the properties of the spline, generating derivative maps, and performing cross validation. In a real project these options should be understood and utilized in order to maximize the fidelity of the generated surface, however, for the purposes of this tutorial we can assume that default values yielded a reasonable approximation of the surface (Figure 4).

4 Export GeoTIFF

Although GRASS contains many useful analysis tools it will inevitably be necessary to use the raster map we created in other applications. Currently GeoTIFFs are the most reliable format for data transfer. Use the following command to create a GeoTIFF that is compatible with ESRI products.

```
r.out.gdal input=min_rst output=min_rst.tif format=GTiff createopt=...
    ...INTERLEAVE=PIXEL
```

Note that `r.out.gdal` exports data based on the current region settings so the region should be set to coincide with the input map if it has been adjusted and you still want to export the entire map. This can be done automatically with `g.region`.

```
g.region rast=min_rst
```