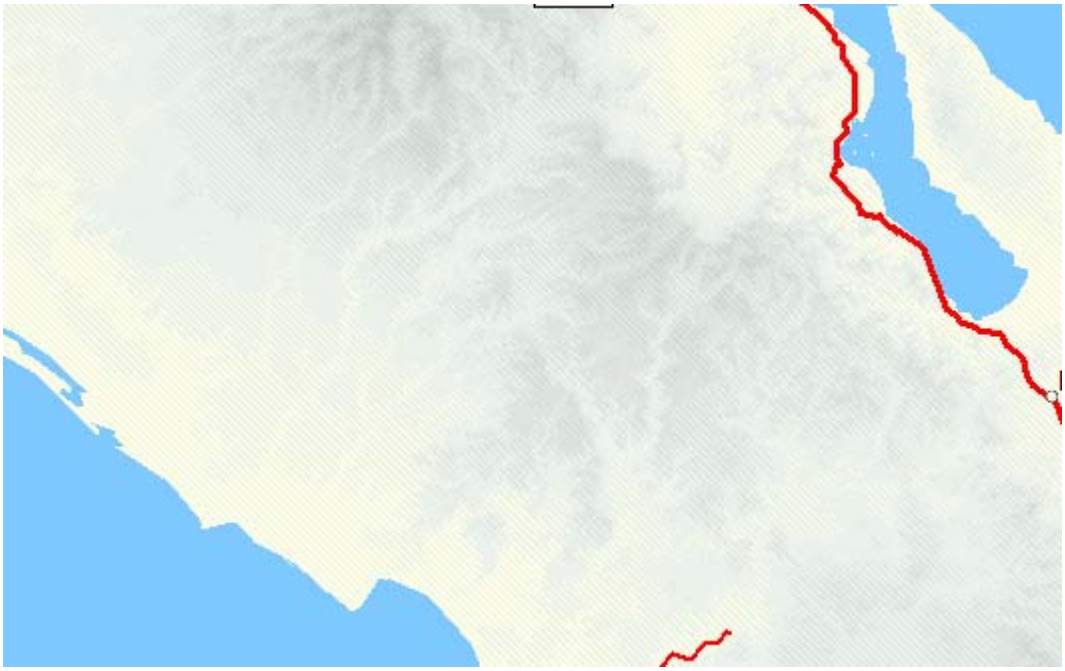
Using custom types to represent elevation data

A common goal for digital mapmakers is to try to have the onscreen results retain the look of a paper map while offering all of advantages of a GPS receiver. I've developed a method for presenting elevation information using colored polygon layers instead of topographic lines. Here's a sample of what can be done with this technique:



Map with elevation layers using custom types rendered in MapSource

If you've studied the possibilities of using custom types, you know that there are 8 levels of draw order specified in your TYP file. Here's a portion of a [_draworder] section:

```
[_drawOrder]
Type=0x01,1       ; Large urban area >200k
Type=0x02,1       ; Small urban area <200k
Type=0x03,1       ; Rural housing area
Type=0x04,1       ; Military base
Type=0x05,1       ; Parking lot
Type=0x06,1       ; Parking garage
Type=0x07,1       ; Airport
Type=0x08,3       ; Shopping center
…(up to 0x54)
[end]
```

The lowest numbers are drawn first, A polygon with a drawOrder of 8 would be drawn above all other polygons, but below all polylines and points.

My goal was to utilize some of the unused polygon types, repurposing them to show elevation in my maps.

The first step is to determine is the range of elevation data we intend to show in our map. This depends entirely upon the data that's available to you. You may have topo layer information every 100 meters up to 3000 meters (as shown in the sample map above), or you may have topo layers from 100' to 500' at 25' intervals. The key is to pick layers at evenly spaced intervals, for example every 100' or every 300'. For my sample map, I followed directions available at http://home.cinci.rr.com/creek/garmin.htm, and downloaded elevation data for the Seattle area.

The area that I'm working with has layers starting at 100' with intervals every 25', up to 500'. That's a total of 16 discrete elevations. 8 representative samples from this data would start at 100', taking every other layer up to 450' for a total of 8 overlaid elevation layers.

After deciding which layers you want to represent in your map, the next step is to perform some transformations on your source data. This portion of the work is the most tedious, but the results are worth it. This assumes that you are comfortable editing Polish format source files and familiar with using GPSMapEdit!

Here are the steps for converting topo lines to polygons:

- Load your source data file with the original topo information.
- Using a text editor, extract all Polyline definitions with the same labels, saving them in separate files. cGPSMapper uses the label of types 0x20, 0x21 and 0x22 to represent the elevation of the topo line. In my sample, I ended up with files named 0100Polygons.mp, 0125Polygons.mp, 0150Polygons.mp., etc..
- Using GPSMapEdit, create closed polylines from the contour lines in each file. In some cases, this means joining separate lines together using the Merge Polylines command. You need to end up with closed polylines. If your topo lines are broken at the edge of your map, you need to add straight line sections to close them.
- After each polyline is closed, Select All and convert all of your polylines to polygons of type 0. We'll change them to their final value in a later step. At this time, you can also extend the polygons up to higher level.
- At the end of this process, you will have a separate file for each elevation layer, with closed polygons instead of topo lines.

Once you've created your elevation polygon files, it's time to create a .img file. I've found that it makes sense to use cGPSMapper's `[File]` directive to include files that I don't need to edit anymore. These topo files are a perfect candidate. Comment out the header in each file and save them in an Include directory. GPSMapEdit simply ignores the `[File]` directive, allowing you to have them compiled without having to look at them while you're editing your basemap. See the cGPSMapper manual section 4.2.4.8 for more information on the `[File]` directive.

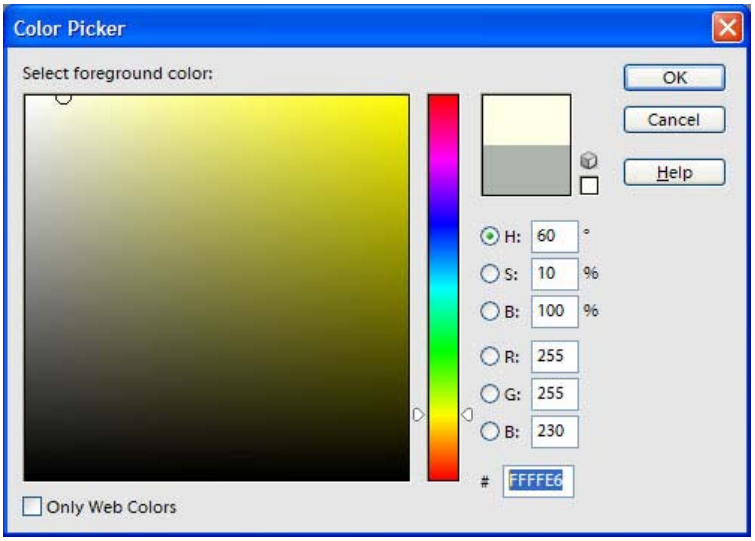
Now we turn our attention to the TYP file and the available spaces in the Polygon types. Here's a list of the 84 standard Garmin polygon types (non-marine) from the cGPSMapper manual:

Type ID	Description	Type ID	Description
0x01	City	0x2b	
0x02	City	0x2c	
0x03	City	0x2d	
0x04	Military	0x2e	
0x05	Parking lot	0x2f	
0x06	Parking garage	0x30	
0x07	Airport	0x31	
0x08	Shopping center	0x32	Sea
0x09	Marina	0x33	
0x0a	University	0x34	
0x0b	Hospital	0x35	
0x0c	Industrial	0x36	
0x0d	Reservation	0x37	
0x0e	Airport runway	0x38	
0x0f		0x39	
0x10		0x3a	
0x11		0x3b	Blue-unknown
0x12		0x3c	Lake
0x13	Man-made area	0x3d	Lake
0x14	National park	0x3e	Lake
0x15	National park	0x3f	Lake
0x16	National park	0x40	Lake
0x17	City park	0x41	Lake
0x18	Golf course	0x42	Lake
0x19	Sport	0x43	Lake
0x1a	Cemetery	0x44	Lake
0x1b		0x45	Blue-unknown
0x1c		0x46	River
0x1d		0x47	River
0x1e	State park	0x48	River
0x1f	State park	0x49	River
0x20		0x4a	Definition area
0x21		0x4b	Background
0x22		0x4c	Intermittent water
0x23		0x4d	Glacier
0x24		0x4e	Orchard
0x25		0x4f	Scrub
0x26		0x50	Woods
0x27		0x51	Wetland
0x28	Ocean	0x52	Tundra
0x29		0x53	Flats
0x2a		0x54	

The highlighted cells are polygon types available for repurposing. For this map, I'm going to use the range of unused polygon types starting at 0x29. My choices may not make sense to you at first, but bear with me and all will become clear.

At first glance, it seems that we're limited to 8 colors, one for each of the 8 rendering layers available to us. The default background color in Mapsource is #FFFFE6. Let's start with that as our base sea level (000') color.

We will want to have a spread of related colors to represent elevation. I'm using Photoshop Elements as my graphics editor. In the Color Picker dialog, I can type in the hex value of the color I'm working with, and PSE will show me the HSB (Hue – Saturation – Brightness) values. Our base value of #FFFFE6 from MapSource has a **H**ue of 60°, **S**aturation of 10% and **B**rightness of 100%. Since this represents sea level, or 0' of elevation, we will use darker values to show higher levels.



Now we need to decide what colors will be mapped to each polygon layer. I'm using a 3% Brightness difference between layers. Photoshop Elements makes it easy to create the other colors in the range by simply changing the Brightness percentage and reading the resulting RGB value. Remember that other map objects are drawn on top of these colors, so we don't want the range of colors too dark.

8-color Elevation Map

Elevation	Polygon layer	Brightness	Color #	Type ID
Sea level (000')	Built-in (Layer 0?)	100%	FFFFE6	
100'	1	97%	F7F7DF	0x2a
125'				
150'	2	94%	F0F0D8	0x2c
175'				
200'	3	91%	E8E8D1	0x2e
225'				
250'	4	88%	E0E0CA	0x30
275'				
300'	5	85%	D9D9C3	0x32
325'				
350'	6	82%	D1D1BC	0x34
375'				
400'	7	79%	C9C9B5	0x36
425'				
450'	8	76%	C2C2AE	0x38

In your custom type definition file, we need to create color assignments for each of the layers we'll be representing. Here's the definition for our 100' and 150' layers:

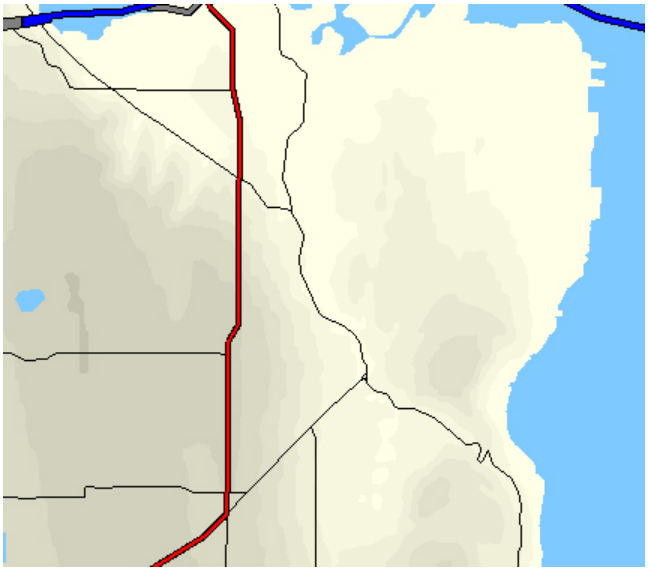
```
[_polygon]
Type=0x2a
String1=0x04,100'
XPM="0 0 2 1"
"1    c #F7F7DF"
"2    c #F7F7DF"
[end]

[_polygon]
Type=0x2c
String1=0x04,150'
XPM="0 0 2 1"
"1    c #F0F0D8"
"2    c #F0F0D8"
[end]
```

Create similar entries for the layers 3-8.

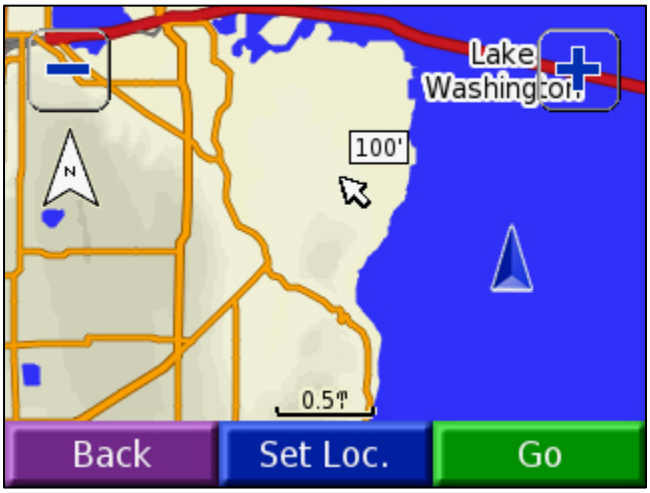
Now we need make sure that the layers are drawn with the proper priority. In the `[_drawOrder]` section, assign these values as shown:

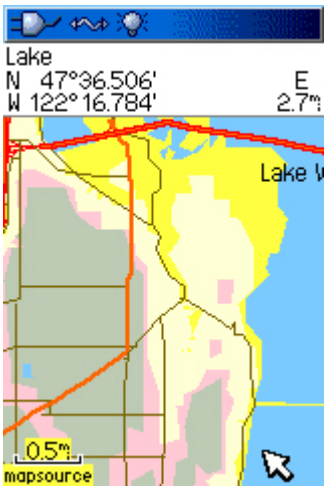
```
Type=0x2a,1       ; 100'
Type=0x2c,2       ; 150'
Type=0x2e,3       ; 200'
Type=0x30,4       ; 250'
Type=0x32,5       ; 300'
Type=0x34,6       ; 350'
Type=0x36,7       ; 400'
Type=0x38,8       ; 450'
```

Compile your custom type file, and then you can see the results of your work in MapSource:



To download this map and custom type definition to a nüvi, use SendMap 2.0 to merge your .img file and the compiled .typ file into a GMAPSUPP.IMG file, and copy it into the Garmin subdirectory on your nüvi. For the 60CSx, simply click 'Upload Maps to GPS'. Disconnect your GPSr from your PC, and then you can preview the map on the device.

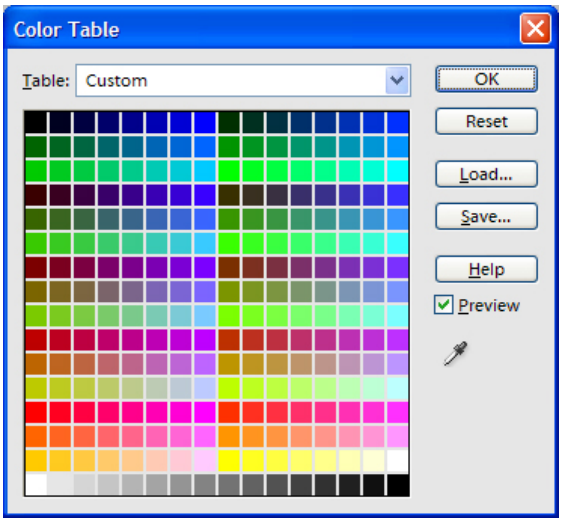




There are a few notable differences. Let's look at the nüvi first. Nüvi's default road types are different than MapSource. The default water color is different on the nüvi, and the default terrain color is different. Each of these can be addressed so that the resulting images look nearly identical.

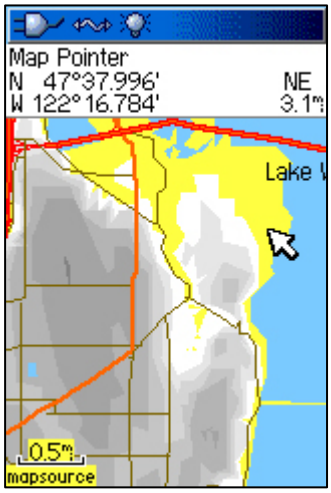
Now let's look at the 60CSx. Whoops. Not quite the same as MapSource and the nüvi! Obviously it's possible to create custom types for the 60CSx, but there's something pretty different about how colors are displayed.

The 60CSx is limited to a palette of 256 colors, and that palette is a *fixed* palette, meaning that whatever colors you assign to your custom types will be translated into that fixed palette. Here's what the 60CSx palette looks like:



Any RGB colors specified in your custom type definitions will be mapped into their closest palette equivalents. Here's a table showing the available values of R, G and B that can be combined, plus a 16-level grayscale:

Red	Green	Blue	Grayscale
00	00	00	000000
39	30	20	101010
7B	65	41	202020
BD	95	6A	313131
FF	CA	8B	414141
	FF	B4	525252
		D5	626262
		FF	737373
			838383
			949494
			A4A4A4
			B4B4B4
			C5C5C5
			D5D5D5
			E6E6E6
			FFFFFF



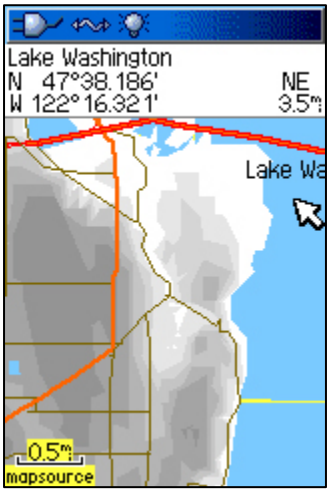
The first step might be to try using the grayscale, since it offers the smoothest color range available from this palette. Redefining the colors to match the grayscale colors yields this image, which looks better in the elevations but fails with the yellow terrain. This image highlights a few changes that need to be made:

- We need to have control over the representation of the lowest terrain layer of the map so that we can fit it into our color scheme.
- The grayscale range is too dark, creating distraction from the roads and other map elements.

The next step is to create another topo layer representing 000' of elevation. Look at your map, identifying the edges of the lowest terrain level. It's normally either a water boundary or the rectangular edge of the map. Using GPSMapEdit, select the water boundary, copy and paste it. Right click the copy, then Modify | Kind | Convert to Polyline. Using the Edit Nodes mode, right click to Split Polyline where the water edge meets the land mass. Join your segments together so that you have closed polylines representing your land masses. Right click to convert the closed polylines back to a polygon. For our example map, we'll make it a type 0x28.

At this point, you may be concerned about performance. If you think about how the standard maps are rendered, this lowest level of terrain is not rendered at all. Everything else is simply rendered on top of it, and what's left over is the ground. We're adding an

extra layer that's not there in a standard map. How will that impact size and performance? That's a legitimate concern, and we'll address it later in this document.



Recompile your project with the 0' elevation layer included. We'll reassign the colors in the .MPT file so that 000' of elevation uses #FFFFFF in the grayscale range, resulting in this image.

Definitely an improvement, but there's still the issue of the grayscale range being too dark, and in creating an elevation level below the topo layers, we had to give up one of our 8 levels, reducing the detail quality of the map.

At this point, we've demonstrated that it's possible to create elevation layering, but we're limited to 8 layers, and the differences between the various displays require lots of extra work. We need another approach, and it would be nice if the new approach would be usable in MapSource, the nüvi family, and the 60CSx and its cousins. We're going to solve all of these problems at once.

If you're familiar with graphics techniques, you may be familiar with the term 'dithering'. Dithering was a technique used on displays with limited color rendering capability to create additional apparent colors. Say for example that you have a 2-color display, black and white, and you want to create gray. By creating patterns of alternating black and white, the eye reads gray. Let's use XPM to explain, since we'll be using it in our .MPT file in just a moment.

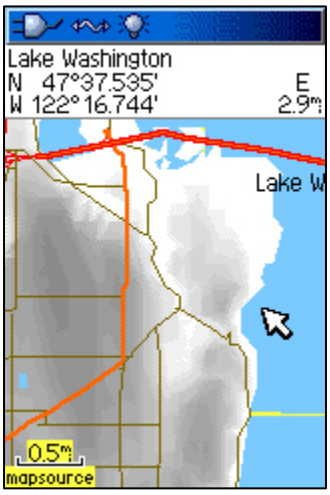
```
[_polygon]
Type=0x2a
String1=0x04,100'
XPM="8 8 2 1"
"      c None"
"*     c #e6e6e6"
"* * * * "
" * * * *"
"* * * * "
" * * * *"
"* * * * "
" * * * *"
"* * * * "
" * * * *"
[end]
```

Assuming that we have a white background (#ffffff), this pattern will alternate pixels of #e6e6e6 with transparent pixels, which will show #ffffff from below, creating a dithered color that appears to be halfway between the two. Depending on the dot size of the display, you may be able to see this pattern as diagonally striped lines, but the effect will still be an area of an apparently different color. Now suppose that we want to add the next color in our spread. We use the same dither pattern, except we shift it by one line.

```
[_polygon]
Type=0x2b
String1=0x04,125'
XPM="8 8 2 1"
"      c None"
"*     c #e6e6e6"
" * * * *"
"* * * * "
" * * * *"
"* * * * "
" * * * *"
"* * * * "
" * * * *"
"* * * * "
[end]
```

Notice that the first line of this bitmap starts with a space instead of a star. Because this pattern represents a higher elevation than the previous pattern, it will always be shown in combination with (i.e., above) the previous pattern. The actual displayed result when both patterns are rendered will be a solid area of #e6e6e6 pixels. So we use the same color assignment for both patterns. When only the first pattern is rendered, we get a "half color". When both patterns are rendered, we get the "full color".

An additional bonus is that because the two patterns don't overlap – in other words, their pixels are drawn in each other's transparent spaces – the two patterns can be rendered together on the same level. This means that instead of 8 colors, we now have a possible 16 colors.

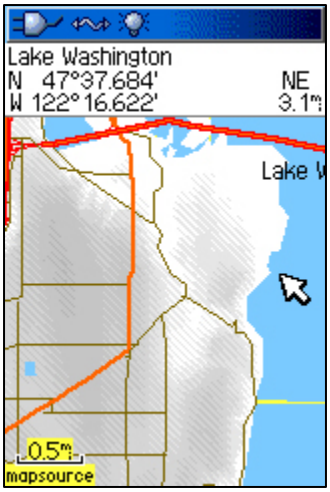


Before we go any further, I'm going to reserve the lowest layer for the sea level terrain. Rather than render it as a pattern, I think it looks better to render it on its own solid color layer, replacing the default yellow on the 60CSx. That leaves us with 7 layers (2-8), each with two apparent colors for a total of 15. This screen grab shows the result.

This is definitely a smoother representation of the elevation layers, but it's still too dark, since we have to move halfway down the grayscale to get all 15 colors displayed.
By now maybe you're thinking "If we can create two colors with dithering and get twice the colors, how about creating four colors and four times the color range?"

And that's exactly the next step. The table below represents four distinct but interlocking dither patterns that can be combined on a single layer.

[_polygon] Type=0x2a String1=0x04,100’ XPM=”8 8 2 1” “ c None” “* c #e6e6e6” "* * " " * * " " * * " " * *" "* * " " * * " " * * " " * *" [end]	[_polygon] Type=0x2b String1=0x04,125’ XPM=”8 8 2 1” “ c None” “* c #e6e6e6” " * * " " * * " " * *" "* * " " * * " " * * " " * *" "* * " [end]	[_polygon] Type=0x2c String1=0x04,150’ XPM=”8 8 2 1” “ c None” “* c #e6e6e6” " * * " " * *" "* * " " * * " " * * " " * *" "* * " " * * " [end]	[_polygon] Type=0x2d String1=0x04,175’ XPM=”8 8 2 1” “ c None” “* c #e6e6e6” " * *" "* * " " * * " " * * " " * *" "* * " " * * " " * * " [end]
25% color	50% color	75% color	100% color

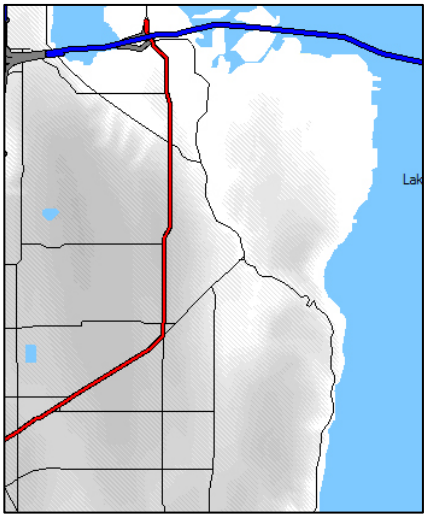


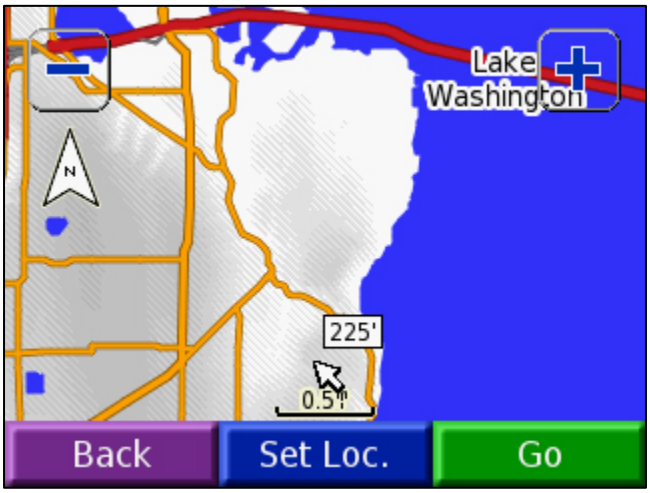
All of these patterns can be rendered on the same level, because their pixels interlock. With our reserved base layer for 000’, we have 7x4=28 + 1 = 29 total elevation layers possible. And, we have addressed the “too dark” problem by creating 4 steps for each grayscale color – 25%, 50%, 75% and 100%.

You can definitely see the dithering, but the effect is not distracting. This image is rendered using only 17 of the possible 29 colors.

Now we have a technique that can be used with MapSource, the nüvi family, and the 60CSx and its cousins.

So how does this same image look in MapSource and on the nüvi? Apart from the differences in road rendering and color of the water, the elevation layering is virtually identical.





For map developers supporting the 60CSx and its compatible cousins, you can experiment with using other color ranges. Blue has the most potential with 8 levels, but the steps are twice as large as the grayscale palette. When you look at the results on the 60CSx, you will see that it actually looks quite a bit better than the screen grabs I'm showing here. The great thing about custom type files is that it takes very little time to try different color schemes.

For map developers supporting the nuvi family, you are not at all limited by color choices. Take a look at a variety of printed atlas legends to see how elevation is rendered, and try some different color schemes. This map has elevation detail every 100 meters up to 3000 meters:

MapSource – 32 color dither

nüvi – 32 color dither

Finally, a brief discussion of performance.

Obviously, we are adding a lot of data to a map to represent elevation. Whether or not this is an acceptable tradeoff is something that you as a mapmaker will have to decide. MapSource takes noticeably longer to render these maps when you zoom or scroll. Performance on the two GPSr's that I've tested this technique with is actually not too bad. But again, this is subjective, and it will depend entirely upon your dataset and expectations. Additionally, you can expect to experience longer compile times with the additional polygon definitions in your map.

I do have one recommendation for using elevation layers which will improve performance and shorten compile time. Consider turning off the elevation information at lower levels in your map. In other words, if you are zoomed in to street level, you probably don't need to see the elevation information at the same time. By eliminating the layer information from the lower levels you can improve performance.

Thanks to Stan and Konstantin for making all of this possible!