DOI: 10.14714/CP96.1669

Normalizing the Normal Map

Charles Preppernau (he/him)
Esri
Email: geolographer.xyz/contactform

INTRODUCTION

Before becoming a cartographer, I made 3D graphics and animations. My favorite projects were those where I had to achieve my goals by using the tools available to me in ways that were very different from their intended purpose. As a cartographer, I've continued borrowing and "misusing" tools from computer graphics, especially the *normal map*. The more I use normal maps in cartography, the more I feel that they *should* be considered a common tool in both cartographic representation and GIS analyses. Unfortunately, up until recently there has been little mention of them in cartographic communities or scientific literature. I'd like to help popularize their misuse.

Readers may already be familiar with the aspect-slope map (Figure 1), which represents surface orientation using two angular measurements. A normal map is the linear coordinate version of an aspect-slope map; it represents surface orientation using a type of 3D vector called a surface normal.

This article will go into the specifics of what a normal map is, how to make one, and some ways to use them in cartography.

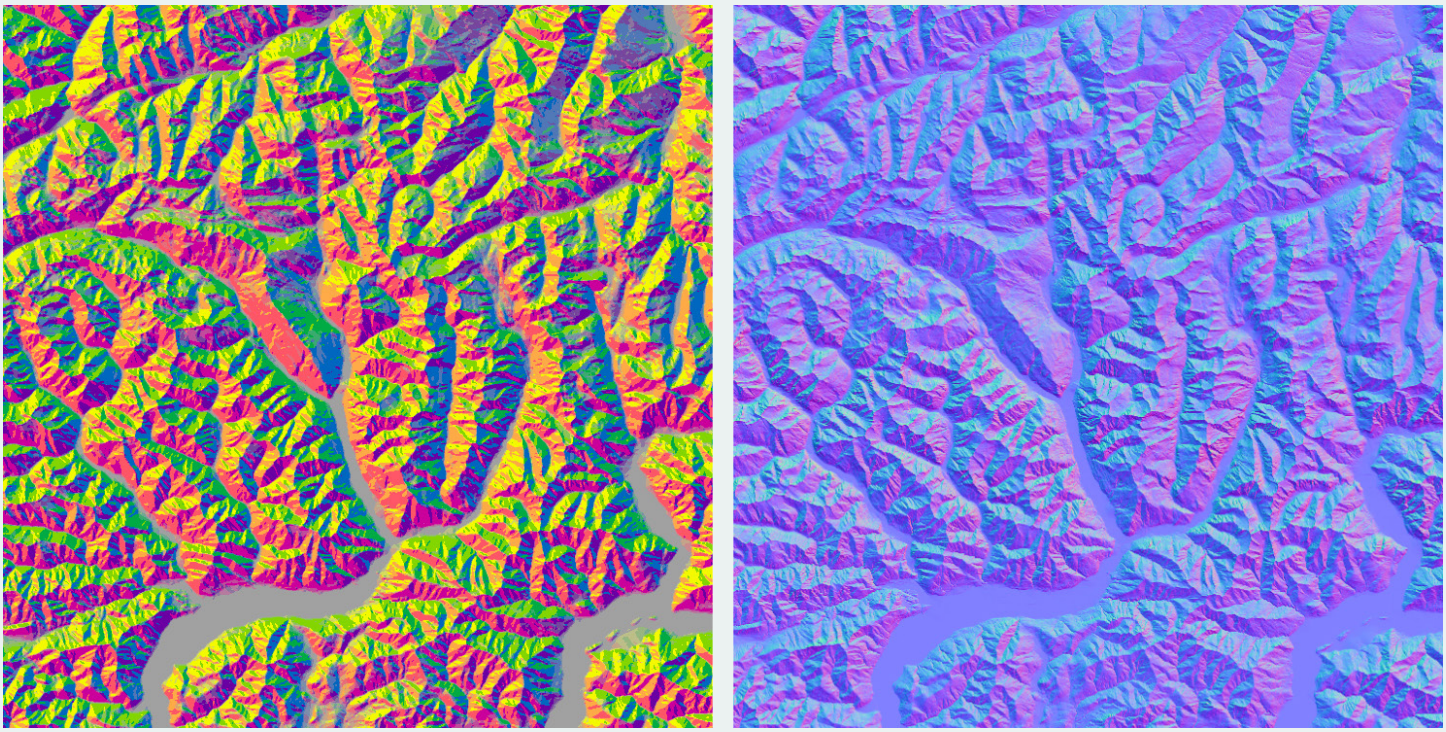

Figure 1. *Two methods for displaying surface orientation.* ***Left:*** *an aspect-slope map.* ***Right:*** *a normal map.*

WHAT IS A NORMAL MAP?

Pick a point on a surface. Imagine a line originating from this point that is one unit long and perpendicular—or normal—to the surface. This line is a vector called a surface normal. A normal map is a raster where the cells represent the normals of a surface.

Vectors have two main properties that will be important for the remainder of this discussion:

- They have a length (also called magnitude) and a direction. In the case of surface normals, the direction is also the direction that the surface faces.
- They have a component for each of their dimensions. A 3D vector like a surface normal has three components: x, y, and z. These components are the distances traversed by the vector along each axis of the coordinate system it is drawn in (Figure 2).

The components of a vector are related to its length by:

$$\sqrt{x^2 + y^2 + z^2} = length \quad \text{(Equation 1)}$$

A normal has a length of one, making it a unit vector. Values for unit vector components have a range of (-1.0, 1.0), and if one component is close to 1.0 or -1.0, the other components will be close to 0. This known range of values makes it easy to remap the components to the range (0, 255) used for RGB color channels:

$$colorValue = 255 \left(\frac{componentValue + 1}{2} \right) \quad \text{(Equation 2)}$$

Usually, x is assigned to red, y to green, and z to blue. So, for a level surface, the normal vector would be (0, 0, 1.0) and the corresponding color is RGB(128, 128, 255) or #8080FF. This is the blue color predominant in many normal maps.

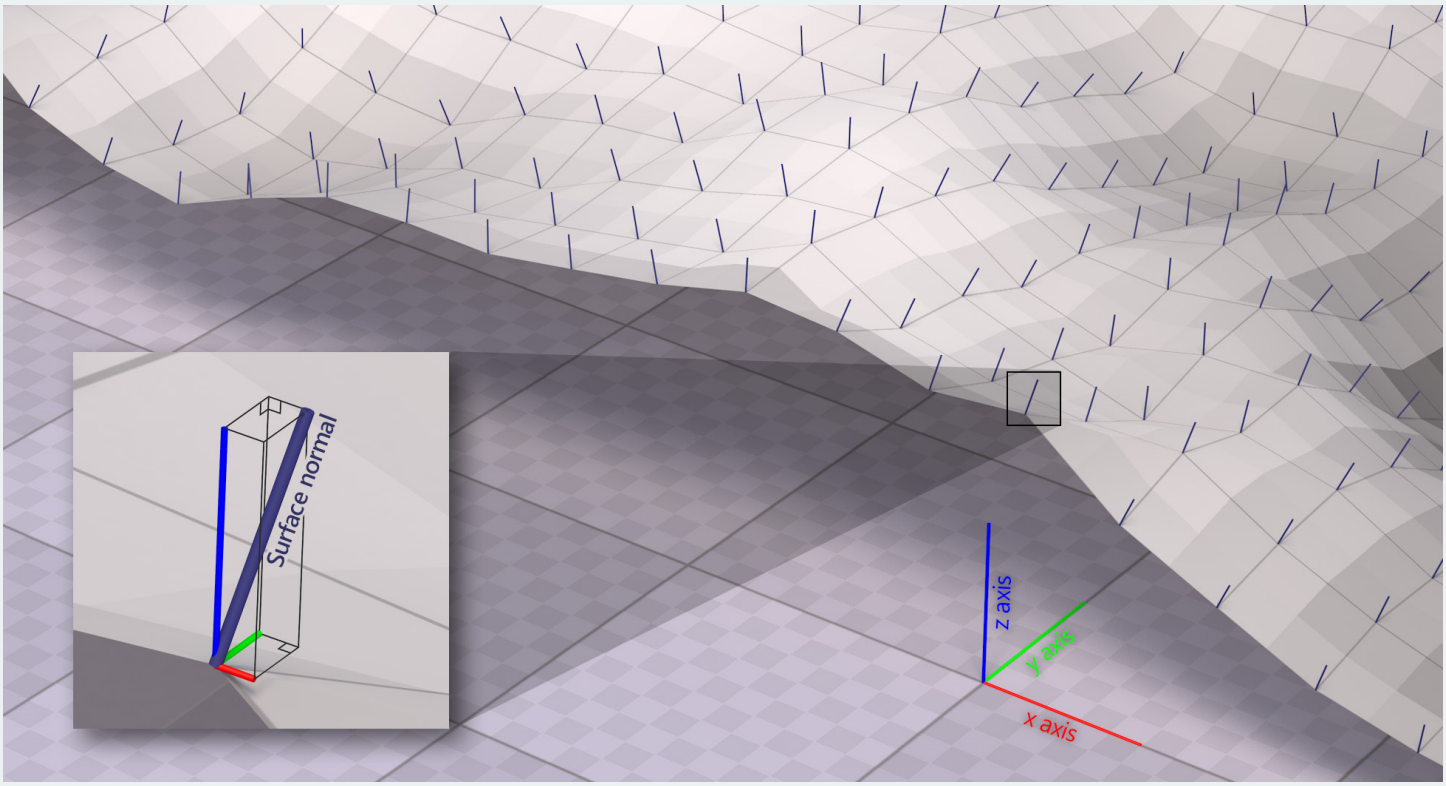

Figure 2. *A 3D surface generated from an elevation raster. Each vertex represents the center of a raster cell, and the normals are drawn from these vertices, perpendicular to the average surface around them.* ***Inset:*** *the distances traveled along each axis by the normal are its components. This one travels a little bit to the east (positive x), a bit more to the north (positive y), and mostly upward (positive z).*

CREATING A NORMAL MAP

Normal maps and the tools used to make them are ubiquitous in graphics apps, but for best results, I recommend creating and working with them in a GIS or with a scripting language like Python. Most of this article assumes that you are using a GIS or coding environment to work with normal maps.

USING GRAPHICS SOFTWARE

Most graphics apps have a function for generating normal maps from grayscale height maps. Those that don't will usually have normal map plugins available. Graphics apps often disagree on whether y or z is the vertical axis, and on which direction is positive along each axis. Assuming your map is oriented with north at the top of the page, the convention I will use for this article is: +x = east, +y = north, and +z = up. If your app does not follow this convention, or you prefer another, you can reorder or invert the color bands (or channels) until they meet your needs.

Some apps have options for generating normal maps in world, object, and tangent space, but these should all be the same for a geospatial elevation map. If not, tangent or world space should be the safest options. If the normal map looks bluish in flat areas, reddish on east slopes, and greenish on north slopes (as in Figure 1), it is similar to those described in this article.

USING A GIS OR PYTHON

Many of the uses for normal maps that I'll discuss later require the ability to perform mathematical operations on rasters. I recommend using a GIS or a programming language like Python to produce and work with normal maps because they make these operations much easier and faster. My two preferred methods for creating a normal map are discussed below.

Option 1: Compute the Components Using Aspect and Slope

Conceptually, the simplest way to make a normal map in a GIS without a specific tool is to first make a slope raster and an aspect raster from your elevation map. Since a normal map is the vector form of the aspect-slope map, it can be obtained by the conversion of these angles to a vector:

$$x = \sin(aspect) \times \sin(slope) \qquad \text{(Equation 3)}$$

$$y = \cos(aspect) \times \sin(slope) \qquad \text{(Equation 4)}$$

$$z = \cos(slope) \qquad \text{(Equation 5)}$$

Once the components are calculated, compositing these raster bands will yield the normal map:

$$\boldsymbol{n} = (x, y, z) \qquad \text{(Equation 6)}$$

Option 2: Compute the Components Using Elevation Gradients

This method is the same one used in Pyramid Shader (terraincartography.com/PyramidShader). It is more direct and probably generates smaller errors than Option 1:

1. **To get the x component, subtract the value of the cell's right neighbor from the value of its left neighbor. Do the same for the top and bottom neighbors to get the y component.** These are elevation gradients (change in elevation, or "rise") along the x and y axes.

2. **The z component for every cell is initially the raster's cell width plus its cell height.** It might seem counterintuitive to use a constant obtained from horizontal distances as the vertical component, but it may be helpful to think of this as the "run" and x and y as the "rise." With both normals and linear functions, zero rise and a nonzero run indicate a level surface or line, while a high rise with the same run indicates a steep surface or line.

   With all three components, the vectors have the correct direction, but the magnitude will vary from cell to cell. It needs to be 1.0 for all cells.

3. **Get the magnitudes of the vectors.** Use Equation 1: $\sqrt{x^2 + y^2 + z^2}$

4. **Divide each component by the magnitudes** to make the vector a unit vector. Note that the value of z will now be small if the value of x or y was large. Or, if it was the only nonzero component, i.e., the surface was level, z will now be 1.0, and the vector will be (0, 0, 1.0).

5. **Write x, y, and z to the red, green, and blue bands of a raster, respectively.**

While this workflow should be doable using the raster tools in most GIS packages, it is best suited for Python or any other programming language. An implementation with NumPy and ArcPy might look like Example 1.

The syntax `H[a:b, c:d]` is a way to get a copy of the array as if the array was shifted one cell in a given direction. This allows you to make four additional arrays containing the original array's left, right, top, and bottom neighbors for each cell, and then subtract those arrays from each other.

PROPERTIES OF A NORMAL MAP

If the normal map is split into its component bands (Figure 3), the bands appear similar to three perpendicular hillshades lit from the positive direction of their respective axes; east, north, and directly above. This is a useful point that I'll come back to when I talk about soft hillshading.

```
import numpy as np
import arcpy

def NormalMap(H, cX, cY):

    # Pad the raster by 1 cell to help deal with raster edges; will be undone in next steps.
    H = np.pad(H, 1, 'edge')

    # x component = left neighbor - right neighbor (also trims width by 1 cell)
    X = (H[1:-1, 0:-2] - H[1:-1, 2:])

    # y component = bottom neighbor - top neighbor (also trims height by 1 cell)
    Y = (H[2:, 1:-1] - H[0:-2, 1:-1])

    # z component = cell width + cell height
    Z = np.ones(X.shape, dtype='float32')
    Z *= cX + cY

    # Get the magnitudes of the 3D vectors
    M = np.sqrt((X ** 2) + (Y ** 2) + (Z ** 2))

    # Divide each component by the magnitude, then stack them into a 3D array
    N = np.stack((X / M, Y / M, Z / M), 2)

    # Make the band the first axis; output axes will be (band, row, column)
    N = np.moveaxis(N, 2, 0)

    return N

def NumpyToRGB(array, corner, cX, cY, srs, destination):

    compX = arcpy.NumPyArrayToRaster(array[0], corner, cX, cY)
    arcpy.DefineProjection_management(compX, srs)

    compY = arcpy.NumPyArrayToRaster(array [1], corner, cX, cY)
    arcpy.DefineProjection_management(compY, srs)

    compZ = arcpy.NumPyArrayToRaster(array [2], corner, cX, cY)
    arcpy.DefineProjection_management(compZ, srs)

    finalComp = arcpy.CompositeBands_management([compX, compY, compZ], destination)
    arcpy.DefineProjection_management(finalComp, srs)

elevationRaster = arcpy.Raster(inputPath)

cX = elevationRaster.meanCellWidth
cY = elevationRaster.meanCellHeight
srs = elevationRaster.spatialReference
corner = arcpy.Point(elevationRaster.extent.XMin, elevationRaster.extent.YMin)

ElevationArray = arcpy.RasterToNumPyArray(elevationRaster)
NormalArray = NormalMap(ElevationArray, cX, cY)
NumpyToRGB(NormalArray, corner, cX, cY, srs, outputPath)
```

Example 1.

The blue color of the most common variant of normal maps is due to the behavior of the z, or blue band. The z-band is usually close to or equal to 1.0 because nearly level ground is more common than slopes, and it is always greater than 0 because these rasters cannot show vertical cliffs or overhanging surfaces. Meanwhile, the x and y bands can be positive or negative, are usually close to 0, and their absolute values are always less than 1.0. Each color on the normal map represents a different orientation. Unlike an aspect raster, it is always the case that the closer the colors (or components) of two normals are to each other, the more similar are their orientations.

APPLICATIONS FOR NORMAL MAPS

Normal maps are seen most commonly in interactive 3D graphics, where rendering times are of high importance. Typically, a high-resolution mesh is used to create a normal map, which is draped or wrapped onto a simplified version of the mesh in a manner similar to a texture. The renderer reads the values of the normal map instead of the actual surface normals of the mesh when performing lighting calculations (Figure 4). Having a low-density mesh that responds to light like a high-density mesh greatly improves rendering time with minimal impact on quality.

This works because the two most important factors for modeling the interaction of light with a surface are: the orientation of incoming light rays, and the orientation (or normal) of the reflecting surface.

This is also true for cartographic relief representation; in many cases elevation rasters are an abstraction of the

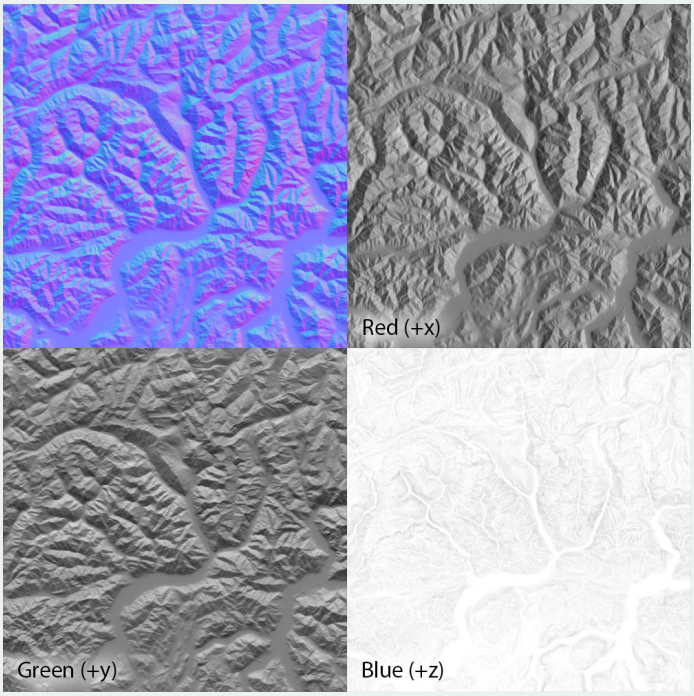

Figure 3. *A normal map and its three component bands. For each band, white is equal to 1.0, medium gray to 0, and black to -1.0.*

information we use for terrain shading. Now I'll discuss some benefits of working directly with normal maps in cartography, starting with basic hillshading.

SOFT HILLSHADING

A common hillshading technique is based on a model of light diffusion described by Lambert (1760), and is called Lambert shading. It uses the following procedure:

1. Compute the surface normal at each cell of an elevation raster.

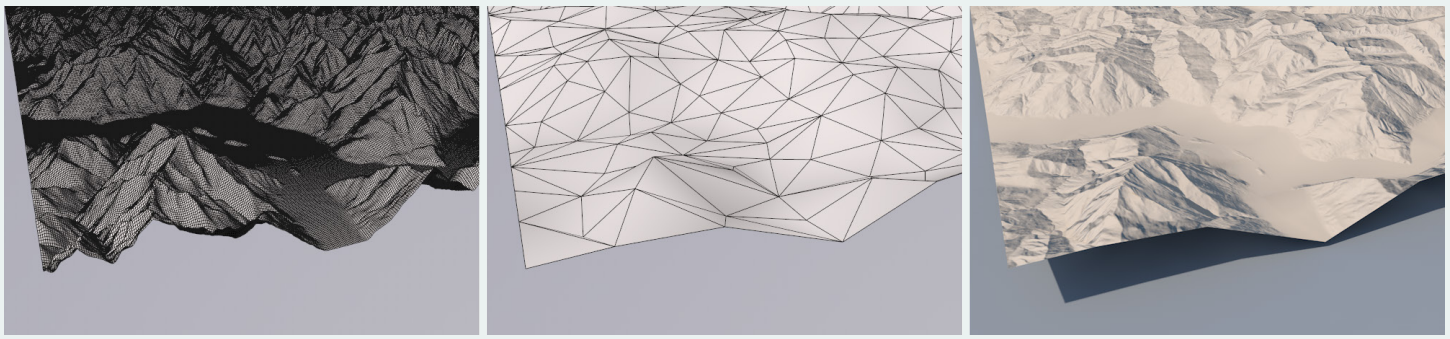

Figure 4. *A dense mesh (**left**) may not draw fast enough for an interactive display, while a simplified version (**center**) may draw quickly but lack the desired detail. A normal map generated from the full-resolution mesh can be used to override the simplified geometry's normals during shading (**right**), giving the benefits of both.*

2. Take the dot product between that normal and the direction of the light source. This produces a raster with values between -1.0 and 1.0.

3. Set all negative values to zero and multiply all values by the output format's value for white (in the case of an 8-bit raster, this is usually 255).

Previously I mentioned that the individual bands of a normal map are similar to hillshades lit from their respective axes. This seems a bit of a stretch if we compare the two (Figure 5), but it turns out that if the Lambert method stopped before step 3, they would be identical. That final step essentially took everything equal to or darker than medium gray in the right image and clipped it to black, adjusting the contrast accordingly. This included the level surfaces, which were perpendicular to the light vector.

Even with standard lighting, it is common for a cartographer to reduce contrast or add transparency to a Lambert hillshade. Since this work essentially undoes the last step of the Lambert method, and since the low clip on negative values isn't reversible, it makes more sense to perform the shading yourself and skip step 3. If it turns out that the contrast or clip are necessary, you can apply them yourself.

Step 1 is complete upon the creation of a normal map. For step 2, you'll calculate the dot product between the normal map and a constant unit vector representing the lighting direction. Not all apps have a dot product function out of the box, but you can compute it by multiplying the respective components of the two vectors together, then summing the results:

$$\boldsymbol{n} \cdot \boldsymbol{l} = \boldsymbol{n}_x \boldsymbol{l}_x + \boldsymbol{n}_y \boldsymbol{l}_y + \boldsymbol{n}_z \boldsymbol{l}_z \qquad \text{(Equation 7)}$$

Using a single cell of a terrain surface with a westward slope of 36.87° as an example, the normal for that cell would be (-0.6, 0, 0.8). Typical hillshade lighting, from a 315° azimuth and elevated by 45°, corresponds to the vector (-0.5, 0.5, 0.71).

$\boldsymbol{n}$ = (-0.6, 0, 0.8)
$\boldsymbol{l}$ = (-0.5, 0.5, 0.71)
$\boldsymbol{n} \cdot \boldsymbol{l}$ = (-0.6 × -0.5) + (0 × 0.5) + (0.8 × 0.71)
= 0.87 (light gray on a scale from -1.0 to 1.0)

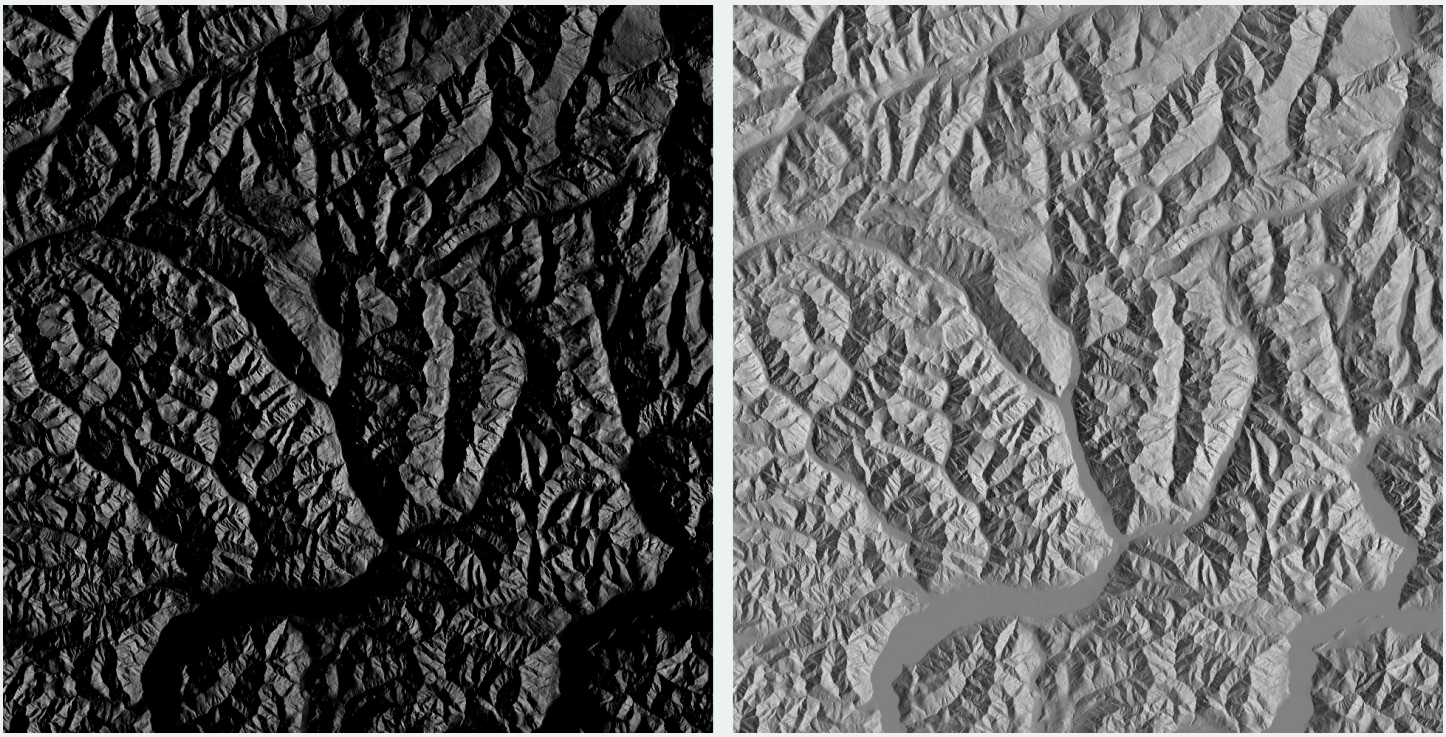

***Figure 5. Left**: A Lambert hillshade lit from the eastern horizon. **Right:** The normal map's x-band, or the same Lambert hillshade before the final step of the algorithm was performed.*

Figure 6 shows an example of this method, which I refer to as soft hillshading, and illustrates the relationship between surface orientation, lighting angle, and shading. The illustration makes it clear that this is not a realistic model of direct lighting. However, shading a terrain exclusively with direct lighting is itself unrealistic, since indirect light is cast on the terrain by the sky and from surrounding illuminated surfaces. In any case, as cartographers, we often favor clarity over realism.

APPROXIMATING MANUAL RELIEF WITH NORMAL MAPS

Manual shaded reliefs have characteristics that make them both very useful for terrain visualization and very difficult for automation (Marston and Jenny 2015; Hurni 2008). I focus on three of these characteristics here:

- Major landforms have greater visual weight than small landforms, reducing visual clutter and noise.
- Lighting and shading are adjusted locally to show features of equal prominence at roughly equal contrast, regardless of the map's lighting direction.
- Greater contrast is given to the crests of peaks and ridges, and less is given to valley floors, regardless of their absolute elevation.

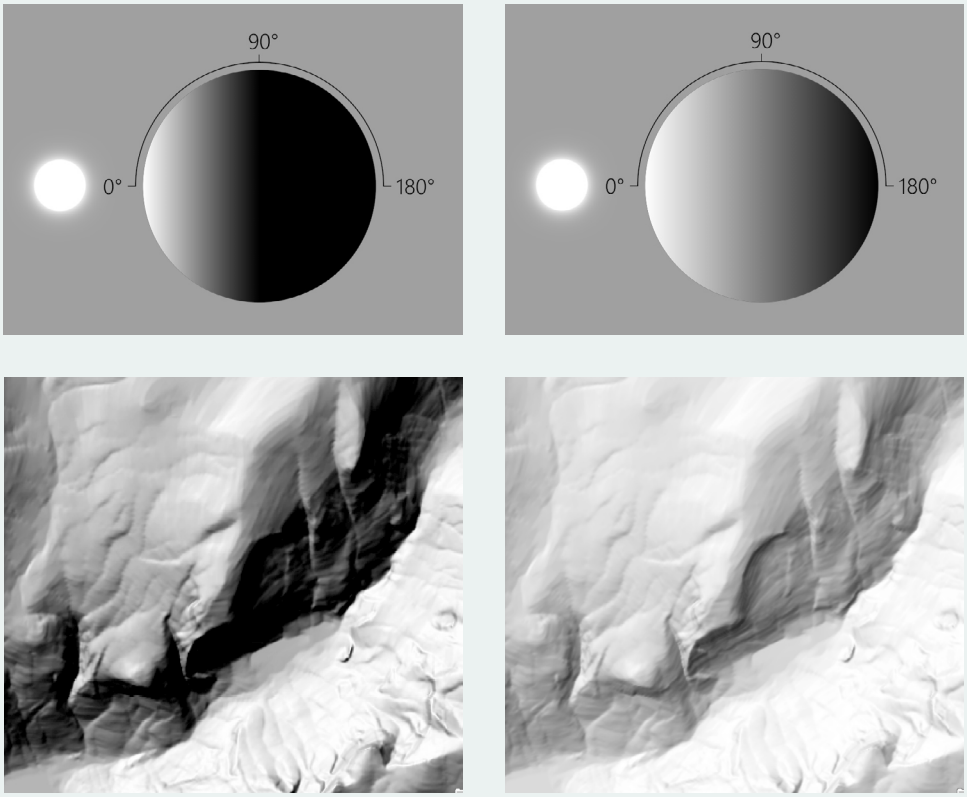

Figure 6. Left: *Lambert shading will only illuminate surfaces that face toward the light source at least partially.* ***Right:*** *With the same lighting angle, soft hillshading preserves all the information in the surface, has a contrast profile that is less harsh, and requires less correction by the cartographer.*

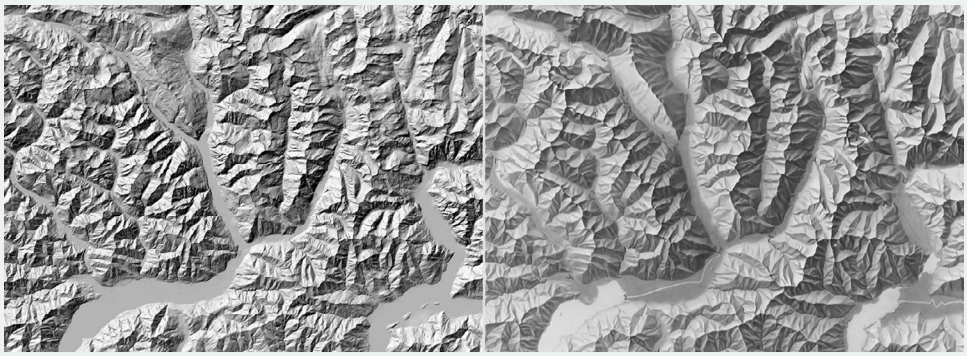

Figure 7. *A standard analytical shaded relief* ***(left)*** *compared with a manual shaded relief by Imhof and Leuzinger (1963;* shadedreliefarchive.com/Graubuenden_SW.html*;* ***right****). Despite deviating from what more physically accurate lighting would portray (or because it deviates from it), this style is better able to communicate a mental map of the relative significance of topographic features.*

These characteristics are beyond the capabilities of standard hillshading (Figure 7), and many researchers, myself included, continue to explore methods for automated relief shading that more closely approximate manual reliefs. My own explorations have mostly involved the use of normal maps, and I have found that normal maps can be applied to each of the characteristics listed above.

Terrain Generalization in Orientation Space

In Lambert hillshading, all slopes of a certain orientation are given the same shade, whether that slope occurs over a contiguous area of one square meter, or thousands of square meters. Thus, in Lambert shading, it is common for small, insignificant topographic features to visually

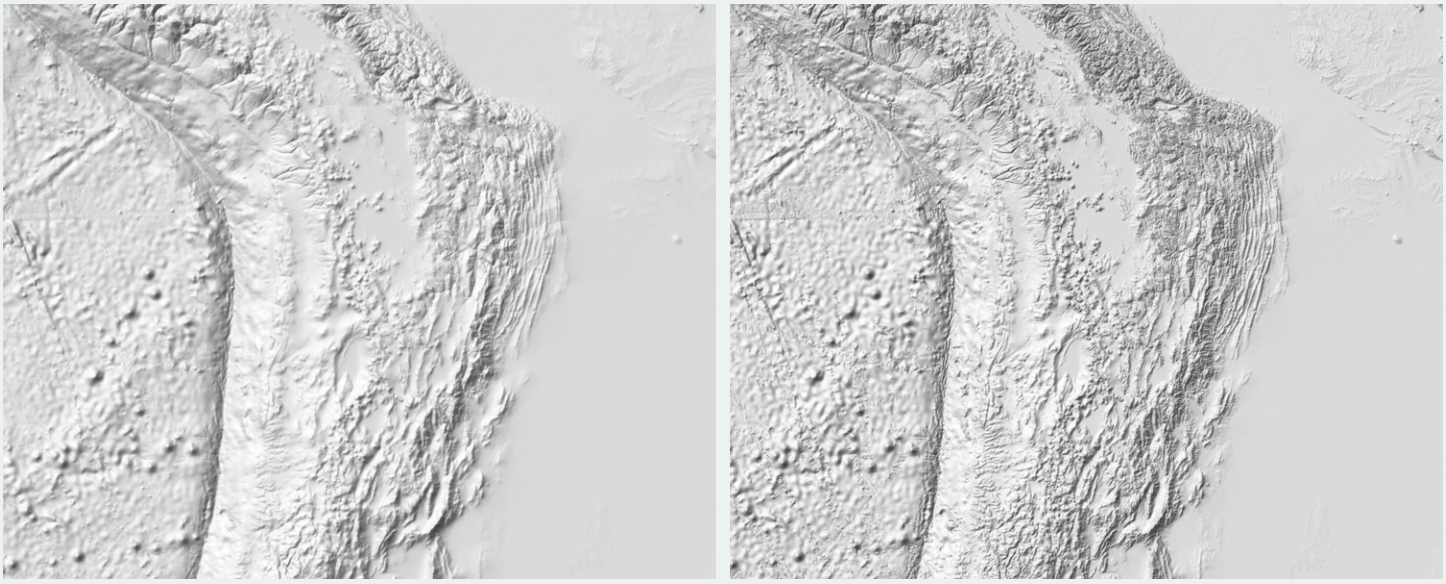

Figure 8. *A comparison of an Andes hillshade based off generalized (**left**) and ungeneralized (**right**) elevation data. The area northeast of center is an especially good demonstration of how Lambert shading with an ungeneralized elevation raster can obscure large features.*

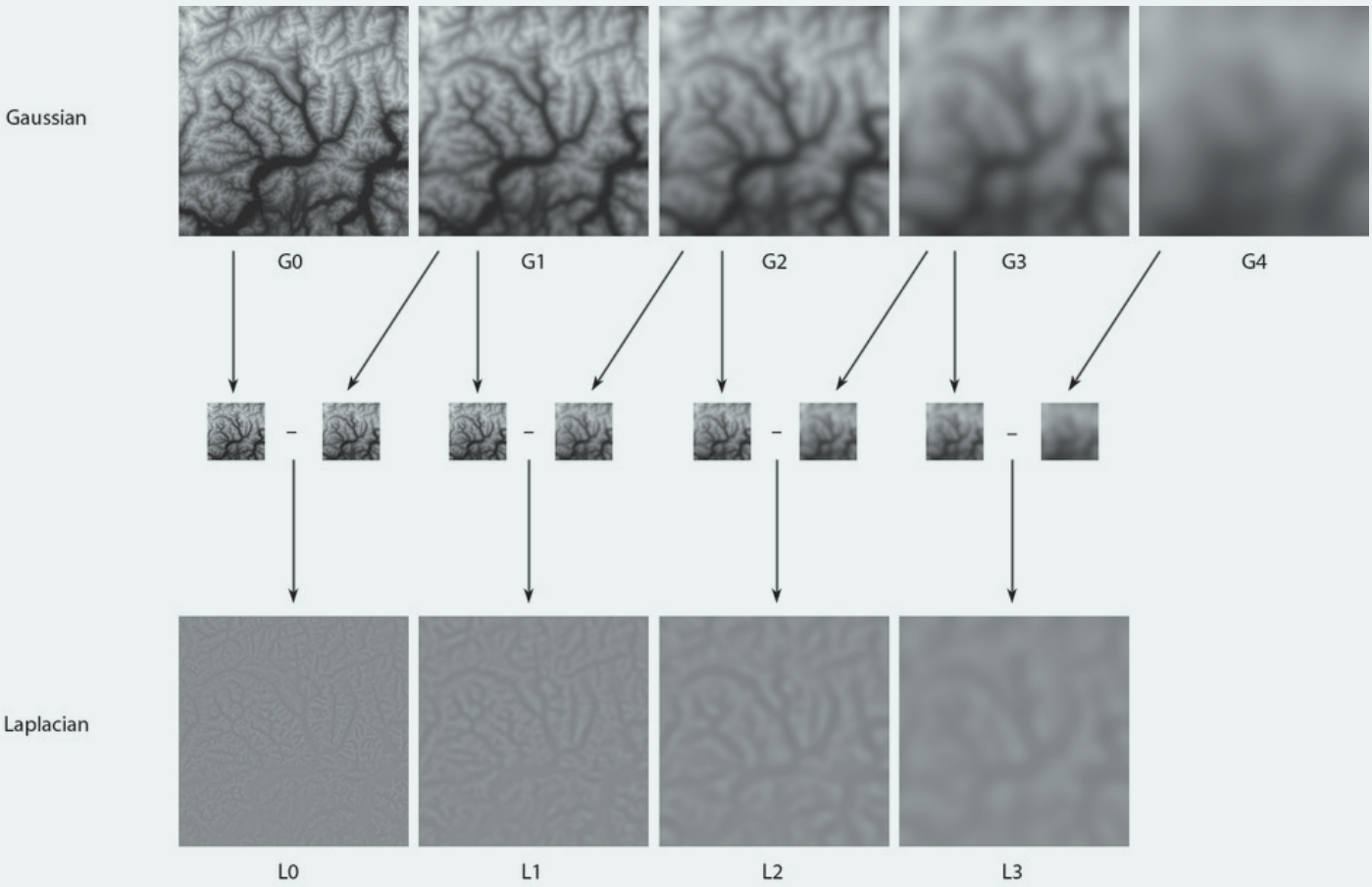

Figure 9. *Construction of a Laplacian pyramid. Starting with an elevation raster (represented by G0), run a series of Gaussian filters, each with twice the radius of the previous. The resulting set of blurred elevation rasters is a Gaussian pyramid (top row). To produce the levels of the Laplacian pyramid (bottom row), subtract each level of the Gaussian pyramid from the previous level. The sum of all Laplacian levels plus the largest Gaussian level is the original elevation raster, so the elevation raster can be generalized by assigning different weights to these levels.*

overwhelm the large features that they are a part of. This is especially true on small-scale maps (Figure 8).

Some readers may be familiar with Pyramid Shader, a project I worked on as a member of the Oregon State University Cartography and Visualization Group (**terraincartography.com/PyramidShader**). This Java application uses Laplacian pyramids to isolate different frequencies, or scales, of detail in an elevation raster (Figure 9). Higher (smaller) frequencies are given a lesser weight than lower (larger) frequencies, so the shading influence of small features is more proportional to their size. The isolated levels of detail are then recombined to produce a generalized elevation raster for hillshading. I describe Pyramid Shader's method in greater detail on my blog: **geolographer.xyz/blog/2017/2/27/an-introduction-to-pyramid-shader**.

Since working on Pyramid Shader, I've explored using normal maps instead of elevation rasters, and median filters instead of Gaussian filters (Figure 10). Like the Gaussian filter, the median filter tends to smooth out regions of similar color. Unlike the Gaussian filter, median filters preserve edges where colors abruptly change. On an elevation raster, edges correspond to sudden changes in elevation, which usually represent cliffs. On a normal map, edges correspond to abrupt changes in orientation, which include cliffs, ridgelines, stream channels, edges of floodplains, and other major topographical features that a cartographer will likely want to retain. In other words, median filters on a normal map remove the details we don't want and preserve the details we do want.

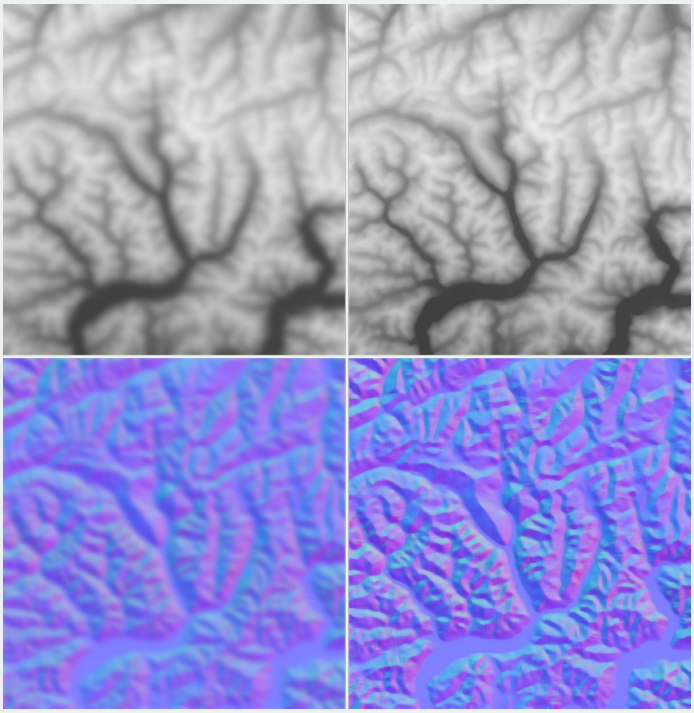

Figure 10. *Comparison of Gaussian (**left**) and median (**right**) filters on an elevation raster (**top**) and a normal map (**bottom**). The normal median (lower right) has the best results in terms of eliminating noise while preserving scale-specific features of cartographic interest, and thus is used in the modified pyramid generalization algorithm.*

Substituting normal maps for elevation rasters, and median filters for gaussian filters, it is possible to build a pseudo-Laplacian pyramid in a similar manner to Pyramid Shader (Figure 11). The median filters can take much longer to compute than the Gaussian filters, but they are worth the wait.

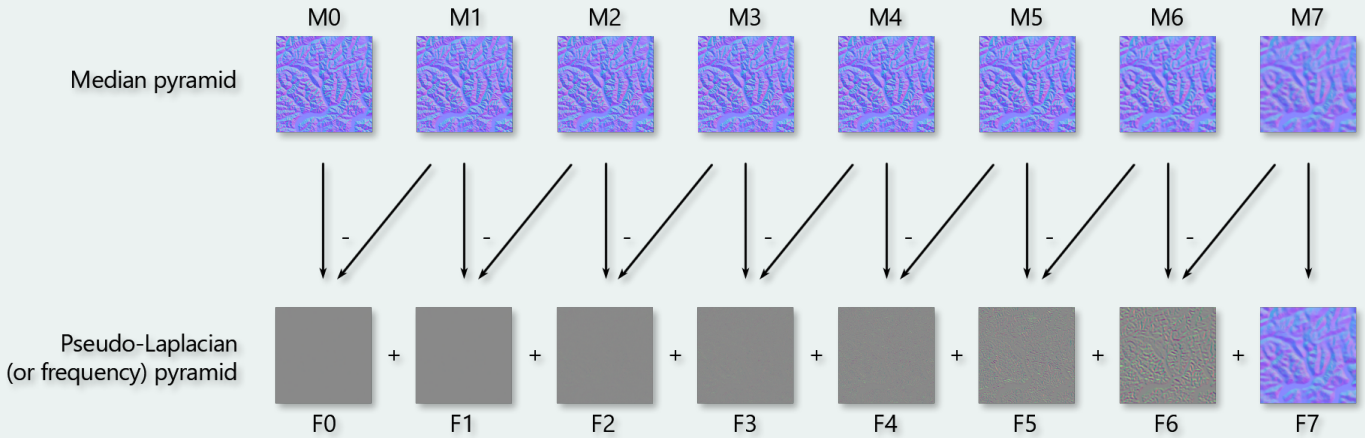

Figure 11. *A modification of Pyramid Shader's approach. Build a median pyramid from a normal map, and then use the differences between those medians to build a pseudo-Laplacian pyramid.*

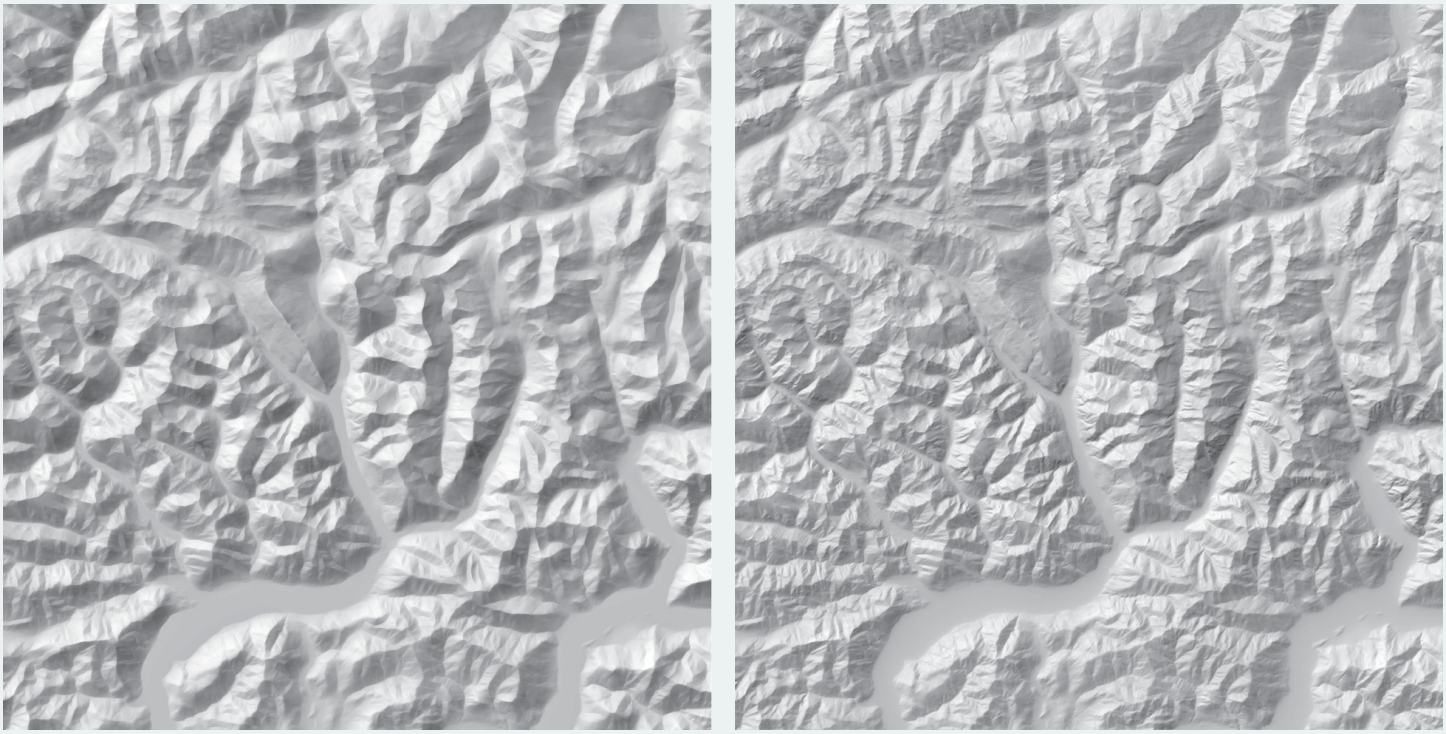

Figure 12. *Normal median generalization (**left**) with exponential weights allows for very strong generalization of small features without blurring large features. In Pyramid Shader (**right**) the same terrain, with the same number of pyramid levels, cannot have its weights reduced further than what is shown here without visibly blurring the ridgelines and valley edges of large-scale features.*

Once the frequency pyramid is generated, weights can be assigned to each level of the pyramid, and then they and the largest median level are summed together to form the pyramid-generalized normal map.

Pyramid Shader currently uses a linear system to assign weights, but my preferred method is to use an exponential function so that the coarsest frequency level has a weight of one, and each finer frequency level's weight is $1/b$ of the weight of the previous level. If, for example, you set $b = 2$, every finer level of detail has half the weight of the one before it:

$$w = 1 \,/\, b^{(n-1)-l} \qquad \text{(Equation 8)}$$

Where w is the weight applied to a level, b is a user-selected base for the exponent, n is the number of levels, and l is the number of the current level. Here, level numbers start at 0, not 1.

Exponential weighting causes fine levels to be more sensitive to generalization than coarse levels, which allows the relative generalization of larger features to be kept to a minimum. The use of normals, medians, and exponential weights allows the cartographer to generalize small features further and preserve the sharpness of large features more easily than with Pyramid Shader's linear Laplacian method (Figure 12).

Variable Lighting Direction

As discussed in the soft hillshading section, the dot product takes two vectors as inputs. One of these inputs was variable, and the other was constant. However, there is no reason why they can't both be variable. By using a variable light vector for hillshading, a cartographer can emulate the local lighting adjustments in manual hillshading. What would this light vector raster look like?

First, since it will represent a unit vector, this raster will have the same value limits as the normal map, and it will satisfy Equation 1. Unlike with the normal map, the z-band can be negative, which would mean the light is coming from below the ground (probably an uncommon case, but I encourage you to experiment with it). So, the light vector raster can be any color you'd see in a normal map, plus the colors with negative z values.

Second, a dot product between two unit vectors is 1 when the vectors are equal and -1 when they are opposite. So, the most brightly lit areas will be where the normal map

and light map are equal, and the darkest areas will be where they are opposite.

Recall that the standard cartographic light vector is (-0.5, 0.5, 0.70711), or #3FBFD9. This is a dull cyan. The areas where lighting is not modified will be this color on the light vector raster. Areas where lighting should be adjusted will be a different color. Good lighting direction choices would be from the west (-0.70711, 0, 0.70711), which is the color #257FD9, or from the southwest (-0.5, -0.5, 0.70711), which is the color #3F3FD9. You can obtain the color for these or any other light vector by using Equation 2 through Equation 5.

To approximate manual hillshading, the lighting should change only for major topographical features according to their generalized aspect, which can be obtained from a smoothed normal map. In the previous section, I covered using median filters to smooth normal maps at different scales. For the following example, I'll use the highest of those median levels. Again, I recommend using median normals rather than Gaussian normals.

1. **Multiply the z-band of the largest median level by 0.1.** Large values for the z-band can reduce the quality of the final result, but z must be non-zero for proper handling of level surfaces, so it is simply reduced here.

2. **Divide the result of step 1 by its magnitudes.** Use Equation 1 ($\sqrt{x^2 + y^2 + z^2}$). With the z-band reduced, this new unit vector raster is essentially an aspect map using vector values, which will be called the xy raster.

3. **Compute the dot product between the xy raster and a horizontal vector perpendicular to the azimuth of your main light.** There will be two vectors that fit this description, but either choice will lead to the same result in the next step. In this example, my main light is from the northwest, and I chose the vector pointing to the southwest horizon (-0.70711, -0.70711, 0).

4. **Take the absolute value of the result of step 3.** Surfaces facing directly toward or directly away from the horizontal vector in step 3 should have their lighting adjusted in the same way, and conveniently have the same absolute value. The result of this step is a mask representing the ratio with which to apply the adjusted light vector vs. the main light vector. If you want to narrow the range of aspects where lighting is adjusted, multiply this mask by itself before moving on.

5. **Create the initial light vector raster with the expression (adjustedVector × maskLayer) + (mainVector × (1 - maskLayer)).** This is a weighted sum of your chosen light vectors, using the mask's value (or its complement) as the weight.

6. **Divide the light vector by its magnitudes as in step 2.** You now have a variable light vector raster, where the light source rotates smoothly between your main and adjusted lighting angles depending on color.

7. **Compute the dot product of your regular or generalized normal map and the light vector raster.** I strongly recommend using a generalized normal map from the previous section for this step.

The output of this process is a soft hillshade (Figure 13) where, as large features face more to the southwest or northeast, the light direction rotates toward your chosen adjusted vector.

Another, simpler option might now be apparent from Figure 13; you could paint your light vectors using your

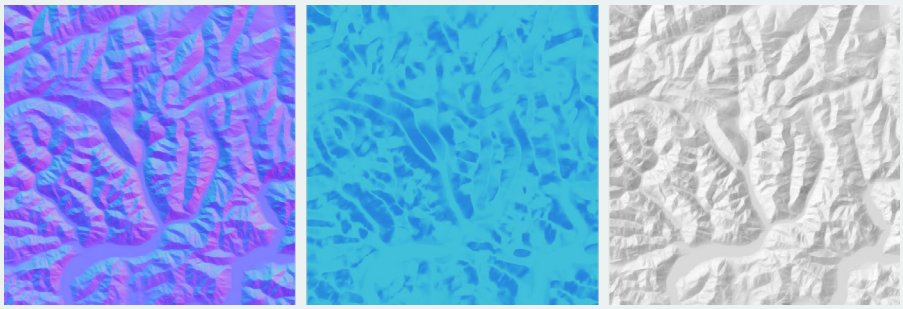

Figure 13. *The dot product between a generalized normal map (**left**) and a light vector raster derived from one of the pyramid levels used to make that normal map (**middle**) is a soft hillshade with a variable light source (**right**). Note that the brightest parts of the hillshade are where the normal map and light vector colors are most similar. In the light vector raster, cyan corresponds to standard lighting from the northwest and blue corresponds to lighting from the southwest, both elevated 45°.*

app of choice. All you need are the colors corresponding to the unit vectors, which you can get using a spreadsheet and Equation 2 through Equation 5, or by constructing a reference image such as Figure 14. Due to the potential for color reproduction issues, I recommend building your own or first verifying that the colors are correct using the equations. While painting may produce colors that are not quite unit vectors, they should usually be close enough to get decent results.

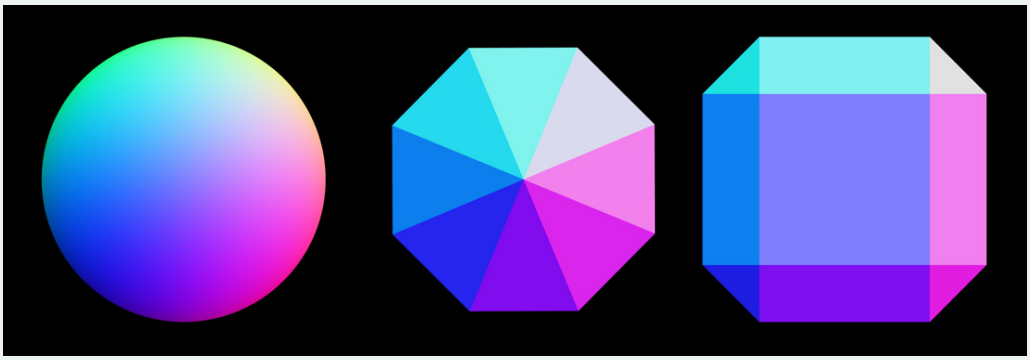

Figure 14. *A reference for unit vector colors on, left to right, a sphere, a faceted cone with a slope of 45°, and a beveled cube.*

Variable lighting vectors can be useful for applications other than manual hillshading; they could be used to make the lighting change by latitude on a worldwide hillshade, to simulate changing sun position in an animated or interactive map, or to highlight regions on a map as if they're lit from multiple light sources if your platform doesn't otherwise support them. I'm sure there are many other uses that you could come up with.

Feature Contrast

The last feature of manual hillshading I'll cover here is the sharpening of contrast on ridgelines. This procedure is best done as part of the pseudo-Laplacian pyramid generalization discussed previously. It is labor-intensive if not done with a script, so if you are not using Python, I recommend only performing this operation for the largest pyramid levels. The steps are:

1. **Run a high-pass filter on the pyramid level with a radius matching the radius of the median that was used on it.**

2. **Multiply the high-pass filtered raster by a mask.** The creation of the mask is described below. Optionally, you can also apply an additional multiplier here if you want even more sharpening.

3. **Add the result to the median pyramid level.**

4. **Divide the result by its magnitudes and proceed with the generalization as above.** Use Equation 1 to obtain the magnitudes. This step may not be necessary, but in some cases not performing it may cause the weighted sum in the generalization process to create unexpected results.

The creation of the mask is the most complicated part of this process, but it is necessary because the high-pass filter will sharpen flat areas near slopes. I recommend starting with either a local hypsometric (LH) raster as described by Huffman and Patterson (2013), or the difference raster from the same paper if the LH raster contains NoData cells. In either case, use the same radius as the median filter for your pyramid level. Once that is done:

1. **Divide the LH raster by its maximum value, then take the maximum between the result and 0.** This clips all negative values and ensures the highest value is 1.

2. **Take the square root of the raster.** Since all values are between 0 and 1.0, this will increase the middle values in the raster without any change to the minimum or maximum, similar to a curve operation in Photoshop where the center of the curve is moved upward.

This should yield a raster where convex areas are close to 1 and concave areas are 0, on a scale roughly matching that of your median normal map and the high-pass filter you ran on it.

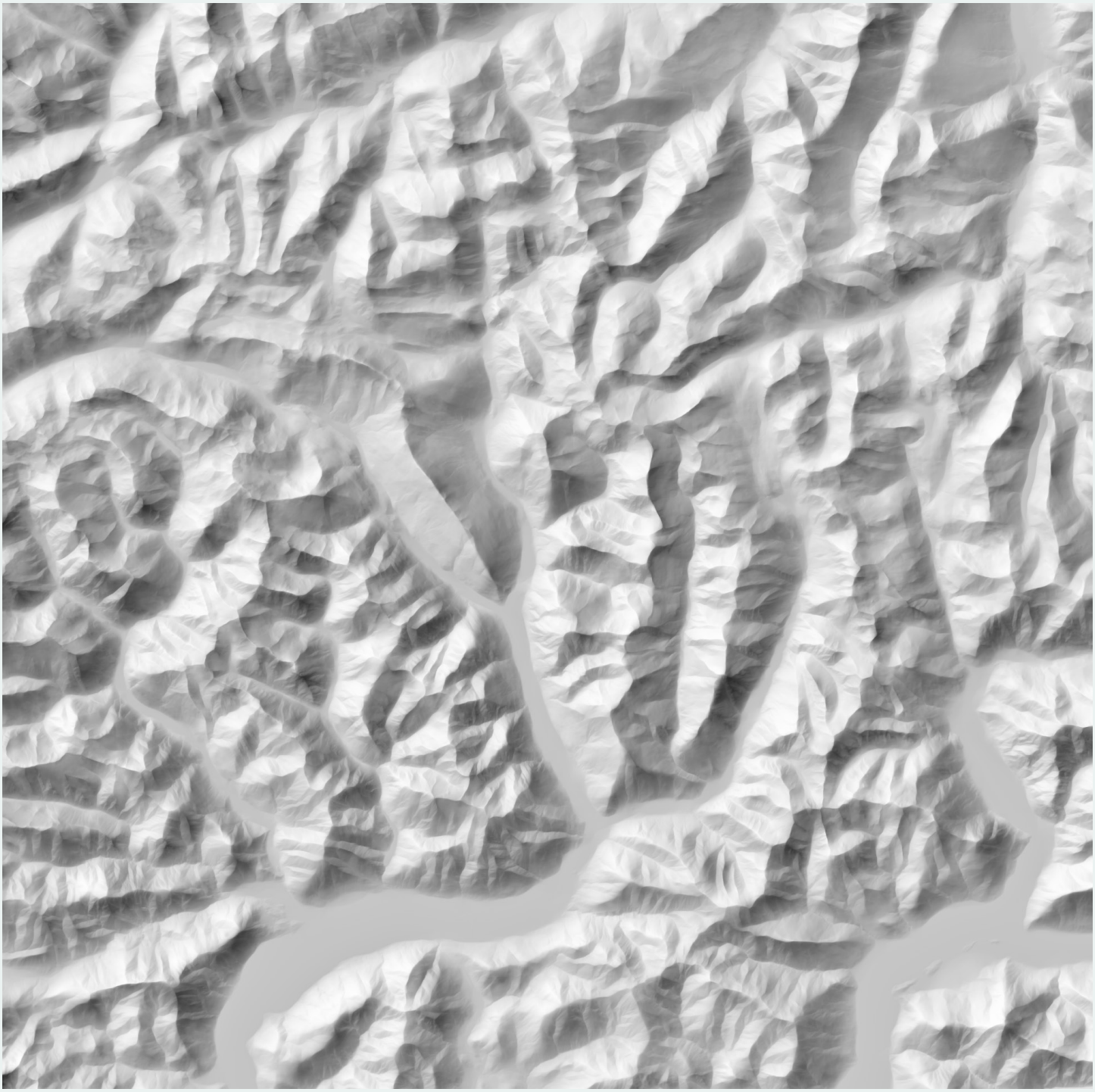

Figure 15. *A hillshade incorporating all the applications for normal maps discussed in this article. In addition, the mask for ridge sharpening was also used to lower contrast in the valleys.*

CONCLUSION

Many of the applications I discussed here can be very labor-intensive and are more practical as scripted tools. Pyramid Shader has a tool for creating normal maps and soft hillshades, and I am working to finish an ArcGIS Python toolbox called Relief Toolbox that contains those, in addition to the rest of the applications described in this article. It will be available at **links.esri.com/ReliefToolbox**.

There are more cartographic applications for normal maps that I haven't covered here. Even though I've used some of

them professionally, I'm still exploring how to make them work consistently and practically before discussing them in a practical cartography context. In summary, further research is recommended.

Still, I hope this at least serves as an introduction and encourages cartographers and toolmakers to explore the uses of normal maps in cartography. Again, if you've ever made a hillshade before, you've essentially used normal maps; possibly without being aware of it. I think normal maps should be at least as commonly seen and talked about in cartography and GIS as they are in computer graphics.

ACKNOWLEDGEMENTS

I WOULD LIKE TO THANK Bernhard Jenny, Brooke Marston, and Bojan Šavrič for, over the course of eight years, providing feedback and advice on the research and experimentation that led me to these methods. Thanks also to Jane Darbyshire for editing and additional feedback.

The Graubünden/Ticino elevation data used for most of the terrain examples in this article was compiled by Jonathan de Ferranti and is available at **viewfinderpanoramas.org**. The elevation data for Figure 6 was obtained from the US Geological Survey at **nationalmap.gov/elevation.html**. The data for Figure 8 was taken from the GEBCO_2014 Grid, version 20150318, available at **www.gebco.net**.

REFERENCES

Huffman, Daniel P., and Tom Patterson. 2013. "The Design of Gray Earth: A Monochrome Terrain Dataset of the World." *Cartographic Perspectives* 74: 61–70. **https://doi.org/10.14714/cp74.580**.

Hurni, Lorenz. 2008. "Cartographic Mountain Relief Presentation." In *Mountain Mapping and Visualisation: Proceedings of the 6th ICA Mountain Cartography Workshop*, edited by Lorenz Hurni and Karel Kriz, 11–15. Zürich: ETH Zürich.

Lambert, Johann Heinrich. 1760. *Photometria Sive de Mensura et Gradibus Luminis, Colorum et Umbrae*. Augsburg: Klett.

Marston, Brooke E., and Bernhard Jenny. 2015. "Improving the Representation of Major Landforms in Analytical Relief Shading." *International Journal of Geographical Information Science* 29 (7): 1144–1165. **https://doi.org/10.1080/13658816.2015.1009911**.