User's Guide for Texture Shading Program Version 1.2

Purpose and Background Information

This software takes terrain elevation data and performs a transformation I tentatively call "texture shading," which consists primarily of a fractional Laplacian operator. The output is intended to be used by assigning a gray scale or color ramp to the values so as to produce a raster image. In this manner it functions as an alternative to hillshading as a technique for producing shaded relief. For a description of the texture shading method and an overview of the mathematics behind it, see the annotated version of the presentation I gave at NACIS 2010. A copy is available with this software.

Copyright Issues, etc.

Software: The software is published under a permissive (BSD 2-clause) open-source license. You are free to copy, use, modify, and publish the software, including in commercial products. See the files named LICENSE.txt for details.

Output: If any output produced by running the program is used in a published work, I would appreciate being credited for the texture-shading / fractional Laplacian idea, at least until I have published a paper on the concept myself. (My hope is to accomplish that sometime in 2012.) This is merely a request and not a requirement.

Included Files

Folder "Mac": Program "texture" – universal binary for PowerPC or 32/64-bit Intel Mac.

Folder "Windows": Program "texture.exe" – 32-bit Windows program.

Folder "Sample_Data": Sample program inputs, outputs, and resulting image files (see below).

Folder "Source_Code": Program source code files (see below).

File "User_Guide.pdf": This document.

Running from the Command Line

The texture shading program is designed to be run from the command line (i.e., in a "terminal" or command prompt window). It reads a data file of elevation values as input, and it writes a data file of texture-shading data as output. To display the output data will require additional GIS software.

The program accepts input in what is often called GridFloat format. (The GDAL library refers to this as "ESRI .hdr Labelled" or "ESRI BIL" format, with floating-point data type. Integer data types are not supported.) Data in this format consists of at least two required files, with extensions .flt and .hdr, as well as an optional .prj file. These last two are text files.

Elevation data in numerous formats can be converted to GridFloat format using the GDAL library (see www.gdal.org). To produce the files needed for input to the texture shading program, you can use the following command (on a single line):

```
gdal_translate -of EHdr -ot Float32
    <input_file> <gridfloat_file>.flt
```

Texture shading output is produced in a single-band BSQ (band-sequential) 16-bit integer format. This output consists of two files, with extensions .bsq and .hdr. If the input has a corresponding .prj file, one is created for the output as well. Make sure the input and output filenames are different, so that they don't share the same .hdr and .prj files.

The texture shading program has three required parameters. The first I call "gain"; this is the value "alpha" in my NACIS presentation, which is an exponent in the fractional Laplacian equation. Lower gain gives less detail but more sense of the overall, large structures and elevation trends in the terrain; higher gain gives enhanced detail at the expense of an overall "flatter" general appearance. This tradeoff may need to be adjusted individually for each map, but values around 1/2 to 2/3 work well in most cases, with a gain as high as 1.0 sometimes useful. The program allows the gain to be specified either as a decimal or as a fraction (e.g., "0.5" or "1/2")

The other two parameters are the input and output filenames. Although there are actually multiple input files (with extensions .flt,, .hdr, and .prj) and multiple output files (.bsq, .hdr, and .prj), only the .flt and .bsq filenames should be given on the command line. Alternatively, the extensions can be omitted and the program will assume them.

For example, to produce texture-shading data from elevation files elev.flt and elev.hdr using a gain of 2/3, use this command:

```
texture 0.666667 elev.flt shading.bsq
```

or this command:

```
texture 2/3 elev shading
```

Either one will produce two output files: shading.bsq and shading.hdr. If the files already exist, they will be overwritten.

Note: Mac OS or Linux/Unix users will need to add "`./`" (dot slash) to the front of the command name, if the current directory (or other location of the "texture" program) is not in their search path. For example:

```
./texture 2/3 elev shading
```

If there are no errors, the program will print something like this:

```
Terrain texture shading program - version 1.2, built Oct 10 2011
Reading input files...

Input data appears to be in lat/lon (geographic) coordinates.
Using pixel aspect ratio of 0.828 based on latitude range.
Processing 3612 column x 3612 row array using gain of 0.666667...
Processing phase 1...
Processing phase 2...
Processing phase 3...
Processing phase 4...
Writing output files...
DONE.
```

The program does not currently make use of the projection information in the .prj file (if any), so it uses values in the .hdr file to infer the type of coordinate system – geographic (lat/lon angles) or projected (linear distance units). You should verify this information in the printed output.

If you run the program with no parameters (just type "`texture`" or "`./texture`"), it will print a brief explanation of its usage.

Map Projections and Extents

To work properly, the texture-shading algorithm normally requires that both the shape and area distortion are minimal across the extent of the data region. This limits its use to large- and medium-scale maps, where the earth is fairly flat over the area of the map. As a rough rule of thumb, a map size of about 1,000 km on a side is probably the largest advisable. Larger regions will produce undesirable artifacts in the results, and the software will give a warning message if it encounters this situation. (See the sample files for one example of such artifacts.)

However, the algorithm can be adjusted to account for area distortion (but not shape distortion), so that small-scale maps can be processed correctly if the data is in a specific conformal projection such as Mercator or polar stereographic. The software handles input data in normal-aspect Mercator projection if the option "`-mercator`" is appended to the command line, followed by the latitude limits of the map in decimal degrees. For example:

```
texture 2/3 elev shading -mercator -55.0 15.0
```

If the elevation data is in a different projection, you will have to first convert it to Mercator projection, then run it through the texture shading program, and optionally re-project the output into the desired final projection. The software does not do any projection transformations on the data. Note also that this option handles only the normal Mercator projection and should not be used for data in UTM coordinates.

Use of the Output Data

Reading the data:

The texture shading output should generally be read and handled as raster image data, not as terrain elevation data. Unlike the input elevation data, which is in meaningful units of length like meters or feet, the output values of the texture shading program don't have any direct physical meaning. So don't try to ascribe any significance to the actual numerical values, or their slope, etc. Just think of each value as a number which (after an appropriate contrast stretch) indicates what shade of gray to use for that pixel.

Contrast stretch:

The output file will consist of values covering a large numerical range. To produce a grayscale image, this range of values needs to be fit into the desired range of black to white pixel values (using other software). This is often called a "contrast stretch." The simplest contrast stretch is to assign the lightest and darkest colors to the maximum and minimum data values, respectively. But with the texture shading output, this typically results in an image that is mostly medium gray, with very low contrast.

A better choice is to use a "standard deviation" contrast stretch, which assigns light and dark to the mean plus/minus some multiple of the standard deviation of the data values. The choice of multiple will be dependent on the user's preference and will vary with each map. If the number of standard deviations used is too high, then the image will be mostly in the middle gray, with too little contrast. If the number is too low, the tops of ridges will be saturated at white and look like they've been smashed flat or lopped off. The best choice will consist of making a tradeoff between these two problems. Useful values can be as low as 0.5 standard deviations or as high as 5. A contrast stretch around 3.5 standard deviations often looks good and is a reasonable starting point to try. Be aware that changing the gain will typically also require changing the contrast stretch.

This tradeoff may be somewhat improved in a future version by using a different contrast stretch formula and including this step as part of the program (or as a separate program).

The .bsq output file is always produced such that its mean is 10000.0 and its standard deviation is 1000.0. So if for some reason a standard deviation stretch is not available, you may be able to manually set the limits of the contrast stretch. For example, for a stretch of +/− 3.5 standard deviations, set the limits to 6500 (black) and 13500 (white).

Suggestions for using color:

Instead of using shades of gray, a different color ramp can be applied to the texture-shading output to produce a colored image. Keep in mind that the resulting colors do not represent elevation ranges as with ordinary hypsometric tints. For example, use reds to represent ridges (high values) and blues to represent canyons (low values), with white in between. This shows the same information as the grayscale image, but using color.

Another recommended use of the output is to produce a texture-shaded grayscale image as described above, and overlay it on hypsometric tints produced from the raw elevation values. This is similar to overlaying hillshading on hypsometric tints, except that the shading is replaced by the texture-shaded image, while leaving the colored layer the same. This technique may allow slightly higher values of the "gain" parameter to be useful, since the colors will show the major elevation differences, leaving the texture shading free to focus more on the details. Color schemes that use light colors for the higher elevations work best, since they don't conflict with the use of light shading for peaks and ridges in texture.

On the other hand, texture shading also reduces the need for hypsometric tints to show elevation, because it retains some impression of the absolute elevation in the grayscale image, especially with lower values of "gain." This may provide a unique opportunity to combine it with color showing other types of information – such as land cover, forest canopy, or geologic zones – while still showing both major landforms and terrain relief details using gray shading in the same map.

When adding a color layer to grayscale texture shading, using a "multiply" blend works well (along with a bit of transparency, depending on the desired effect). Or, for more muted colors, simply blend the two layers using transparency alone.

Adding contour lines may also be helpful to enhance the elevation information in texture-shaded maps, though I have not yet explored this. Likewise, texture shading might help make contour maps easier to interpret.

Combining with hillshading:

A promising idea is to combine a texture shaded image with traditional hillshading, so as to get the benefits of both methods. To do this, after adjusting the gain and contrast stretch appropriately as described above, create a separate grayscale image of standard hillshaded relief, and then blend the two images by using a transparency around 50% on the upper layer. Since this process softens the contrast, it may be useful to enhance the texture shading by either using a smaller number of standard deviations in the contrast stretch or using a higher gain value. With this technique a gain of 1.0 is not unreasonable in many cases.

In order to add a color layer, you may want to first create a new image from the blended grayscale images, and then combine this result with the color layer. This allows more options for combining the gray and color layers.

Important Limitations

Integer data: The method may not work well on elevation data which has been rounded to integers (e.g., whole meters or feet). This creates terracing artifacts in the data, and the texture shading transformation enhances these. This effect is inherent to the method, so it is not easily fixed. The problem will be most obvious in data with high horizontal resolution and/or gradually-sloping terrain. (SRTM, ASTER, and DTED are common sources of integer-rounded elevation data.) Whenever possible, use floating-point elevation data which has not been rounded to integers.

No void points: In order for the algorithm to work, it needs a complete rectangle of data with no void points. In order to use texture shading, any voids must be filled in using interpolation or some other method. The command-line program assumes any "NODATA" values it reads are ocean and assigns a zero elevation to them. If there are true voids in the data that are not at sea level, you won't get good results unless you either fill in the voids or clip the region to a rectangle that doesn't include any voids, before using the program. Likewise, if your data is not rectangular in shape, you will need to extract a rectangular region on which to perform the texture shading.

Tiled maps: The algorithm currently cannot support the assembling of seamless maps from separately-processed tiles. In order for adjacent regions of texture shading to match, all the data must be processed together as a single, large array. To accomplish this, the data may need to be reformatted between individual tiles and a monolithic array using other software. This limitation is not easy to resolve – simply blending the edges of adjacent or even overlapping tiles is not a sufficient solution. (However, I now have an idea that would allow tiles to be processed separately, which I hope to have available eventually in a version of this software.)

Data Size

In order to reduce the impact of the tiling limitation, significant effort has been put into allowing the software to process very large arrays in an efficient manner. Essentially any array that fits in the available RAM can be processed in-place by the program (see below). For even larger arrays, swapping to disk is inevitable; but some optimization is realized for these as well – though the increase in processing time can still be dramatic.

The table below gives some approximate maximum sizes of arrays that can be processed in memory for different computer/compiler architectures:

RAM size	32-bit	64-bit
2 GB	16,000 × 16,000	20,000 × 20,000
3 GB	23,000 × 23,000	26,000 × 26,000
4 GB	23,000 × 23,000	31,000 × 31,000
6 GB		38,000 × 38,000
8 GB		45,000 × 45,000

As an example, for NED 1-arc second data processed in its native resolution and geographic projection, the following region sizes can probably be handled in memory, depending on the memory layout of the particular computer:

RAM size	32-bit	64-bit
2 GB	4° × 5°	5° × 6°
3 GB	5° × 8°	5° × 10°
4 GB	5° × 8°	7° × 10°
6 GB		10° × 11°
8 GB		10° × 15°

And these are corresponding examples for data at 1/3-arc second resolution:

RAM size	32-bit	64-bit
2 GB	1° × 2°	1½° × 2°
3 GB	1½° × 3° or 2° × 2°	1½° × 3½° or 2° × 2½°
4 GB	1½° × 3° or 2° × 2°	2° × 4°
6 GB		3° × 4°
8 GB		3° × 5½° or 4° × 4°

Again, larger data sets can be processed, but with a substantial increase in running time (hours, instead of minutes).

Speed Issues and Array Sizes

The current FFT implementation used by the program depends on breaking down the numbers of rows and columns into their prime factors, and suffers significant performance degradation if those factors are large. This effect is most significant when the number of rows and/or columns is itself a prime number. Thus, array sizes like 1800, 3600, or 10,800 rows/columns are very efficient, whereas 3601 is not as fast (3601 = 13×277), and 1801 or 3602 will be much slower since 1801 is prime.

For example, here's a comparison of approximate running times for different array sizes (tested on a 3 GHz, 64-bit Mac). Notice how a few extra rows or columns can sometimes make a very large difference in the speed of the program:

Array size	Running time
1800 × 1800	2 seconds
1800 × 1801	20 seconds
1801 × 1801	30 seconds
18,000 × 18,000	4 minutes
18,010 × 18,010	45 minutes (because 1801 is prime)
18,002 × 18,002	3 hours (because 9001 is prime)

It's possible to significantly improve on this behavior by using an enhanced FFT algorithm. I expect to make this upgrade in a future version of the program. In the meantime, you can avoid this problem by making sure your array sizes are round numbers (e.g., multiples of 100), at least for large arrays.

Miscellaneous Notes

Quality of data: The method benefits from both high horizontal resolution and high vertical precision. Thus, it works especially well with data derived from LIDAR. At the other extreme, maps with low pixel density or lacking in terrain detail don't tend to look very interesting with texture shading.

Vertical exaggeration: For regular shaded relief (hillshading), a common technique is to first apply a vertical exaggeration factor to enhance the terrain before computing the hillshade. This gives an extra parameter to play with besides the illumination direction and elevation angle. For texture shading, applying a vertical exaggeration to the input will have no effect on the output; the resulting image should come out identical as long as the same type of contrast stretch is used. So the gain and contrast stretch are the basic parameters available to affect the result.

Negative gain: It is possible to supply a negative gain value to the program, in which case it has an effect similar to a blur filter on images. For example, grayscale image data can be converted to GridFloat format and then passed to the program with a negative gain. Unlike most blur filters which are controlled by a size parameter such as a blur radius, the effect here is like considering every choice of blur radius and then producing a weighted average of all those results. The controlling parameter is this case is the gain, which determines how much relative emphasis is given to smaller blurs versus larger ones. Since this balance is consistent across scales, the blur should appear similar over a wide range of zoom levels on the image.

The more negative the gain, the stronger the blurring effect. Useful values range between about 0 and –1 (with –1 giving a very strong blur). A simple min/max contrast stretch is usually appropriate to display the output when negative gain is used.

Error and Warning Messages

If the program encounters any problems with the data it will give messages saying "ERROR" or "WARNING" and briefly explaining the issue. Most unexpected conditions will cause error messages and the program will exit; the problem will have to be corrected in order to continue. Warnings will not quit the program, but they should be taken seriously, as they generally indicate something is wrong.

Sample Files

Included are the following sample input and output files, as well as sample image files showing what the output should look like once it is projected and converted to an image. These can be used to check your workflow to ensure the data is being processed correctly.

rainier_elev.flt, rainier_elev.hdr, rainier_elev.prj:

> NED 1-arc second elevation data for the region 46.5° to 47° N, 122° to 121.5° W (which includes Mt. Rainier) in geographic coordinates, courtesy of the U.S. Geological Survey (USGS).

rainier_tex.bsq, rainier_tex.hdr, rainier_tex.prj:

> Output from running this command:
>
> ```
> texture 2/3 rainier_elev rainier_tex
> ```

rainier_tex.png

> Projected image file created from rainier_tex files, using a contrast stretch of 4.5 standard deviations. (The image uses a Mercator projection, with approximately 31 meters/pixel at this latitude.)

rainier_color.png

> An example of the result obtained by combining the shading from rainier_tex.gif with hypsometric tints that are based on the original rainier_elev files.

rainier_combined.png

> An example of blending texture shading with traditional hillshading, and then overlaying the result on hypsometric tints.

rainier_redblue.png

> A different approach to displaying texture shading – using red for ridges and blue for canyons, overlaid on standard hillshading.

helens_elev.flt, helens_elev.hdr, helens_elev.prj:

Elevation data at 1/3-arc second resolution, which was derived by subsampling of USGS NED 1/9-arc second data in the area of Mt. St. Helens.

helens_combined.png:

An example map created by applying hypsometric tints to standard hillshading for the Mt. St. Helens region, and then blending that with a texture shading image using a only a transparency effect.

global_elev.flt, global_elev.hdr, global_elev.prj, global_COPYRIGHT.txt:

Terrain and bathymetry data covering 75°S to 75°N latitude in Mercator projection, with central meridian of 10°E. Derived from Scripps Institution of Oceanography's 1-minute Global Topography V14.1. To make a more reasonable file size, the data has also been subsampled by choosing the center pixel out of each 5x5 square. Note that this data set is copyrighted, but its use is permitted for educational, research, and non-profit purposes (see global_COPYRIGHT.txt).

global_merc.bsq, global_merc.hdr, global_merc.prj:

Output from running this command (single line):

```
texture 2/3 global_elev global_merc
     -mercator -74.999 74.999
```

global_merc.gif

Image file in Mercator projection created from global_merc files, using a contrast stretch of 1.5 standard deviations.

global_INCORRECT.gif

Image file resulting from processing the global_elev files without using the "`-mercator`" option, for comparison with global_merc.gif. Notice that the incorrect file shows lower contrast in the high latitudes and higher contrast near the equator, whereas the correct file shows more uniform contrast.

global_color.gif

Example image file obtained by combining the shading from global_merc.gif with hypsometric tints based on the original global_elev files.

blur-0.00.png, blur-0.25.png, blur-0.50.png, blur-0.75.png, blur-1.00.png

> Images illustrating the blur effect created by applying texture shading with negative gain to a grayscale image (in this case, hillshaded relief of Mt. Rainier). The images use a gain of 0, –1/4, –1/2, –3/4, and –1, respectively.

These image files have been produced using Global Mapper Version 13.

Future Enhancements Contemplated

Roughly in order of current priority:

1. Allow the user to specify whether data is cell-registered or grid-registered. (Currently all data is treated as cell-registered – i.e., pixel edges are assumed to be aligned with grid boundaries.) This is a minor change. It will help the processing speed for odd array sizes, and may eventually be useful in solving the problem of splicing adjacent tiles. But otherwise, the differences will not be noticeable to the user. This change also involves completing the set of DCT functions, so they will be more widely useful as a standalone library.

2. Fix slow processing time for certain array sizes. This will be accomplished by implementing Bluestein's FFT algorithm in the DCT functions. The result will be much better performance for the worst cases, and more consistent speed overall.

3. Improve contrast appearance: Adjust output values so as to saturate gradually as they approach user-specified contrast limits. This will allow ridge tops to avoid being solid white and looking flat, without needing to reduce the contrast in the middle grays. I expect this to give a modest improvement to the look of the resulting maps.

4. Add an image preview program, to make it easy to see what the texture shading output might look like visually, without having to run a full GIS program.

5. Provide input/output of GeoTIFF files. This will allow more complete georeferencing information to be available to the texture shading program and also allow this information to be included in its output, which may give more ease or flexibility to its subsequent use in other software.

6. Provide the ability to divide a large set of elevation data into tiles that can be processed individually and then reassembled into a seamless texture-shaded map. This is a significant change that will take me some time to work out.

Known Issues

Geographic units: The program won't properly process data in geographic projection unless the latitude/longitude units are in decimal degrees. With arc seconds, minutes,

radians, etc., the projection type will not be recognized correctly, which will result in anisotropic artifacts.

Usage from within Your Own Software

Besides the command-line interface, the texture shading algorithm can also be called directly from user-developed software. The computation is performed by the function terrain_filter(), which is declared in the file terrain_filter.h. If you have (or wish to write) source code capable of reading terrain elevation data into a two-dimensional floating-point array, then a call to terrain_filter will replace the elevation data in that array with corresponding texture-shading values. Further processing will then be needed to make use of this data, such as by producing an image. Note that this interface may change as additional features are added.

For small-scale maps, functions fix_mercator() and fix_polar_stereographic() are also provided in terrain_filter.h to do the post-processing necessary to correct for the scale variations in these projections. These functions should be called after terrain_filter().

Compiling from Source

Using the GCC compiler, a recommended set of options for building the texture shading program is as follows:

```
gcc -O2 -funroll-loops *.c -o texture
```
[on Mac/Linux/Unix]

or

```
gcc -O2 -funroll-loops *.c -o texture.exe
```
[on Windows using MinGW]

It should compile with other C compilers as well, with minimal or no modifications, though this has not been tested.

Source Files

The source code can be grouped into layers, with a middle layer that implements the texture shading algorithm, a top layer that provides the command-line interface and takes care of the file I/O, and a bottom layer that does the FFT computations needed.

Top Layer:

texture.c (contains function “main”)
read_grid_files.h
read_grid_files.c
write_grid_files.h
write_grid_files.c

Middle Layer:

terrain_filter.h (interface to texture shading algorithm)
terrain_filter.c
compatibility.h (helps deal with compiler dependencies)
transpose_inplace.h (general-purpose matrix transpose)
transpose_inplace.c
dct.h (generic interface to bottom layer)

Bottom Layer:

dct_fftpack.c (adapts FFT library to dct.h interface)
compatibility.h (also used by middle layer)
fftpack.h
fftpack.c

Please feel free to contact me with any comments, questions, or bug reports on the software or the algorithm.

October 12, 2011
Leland Brown