User's Guide for Texture Shading Program Version 1.3
(DRAFT)

Purpose and Background Information

This software takes terrain elevation data and performs a transformation I call "texture shading," which consists primarily of a fractional Laplacian operator. The output is then represented as a grayscale raster image. Texture shading can function as an alternative to traditional hillshading as a technique for producing shaded relief – or the two can be used in combination to enhance standard hillshading. For a description and some examples of the texture shading method, as well as 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; the file is called "Texture_Shading_with_Notes.pdf".

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 in 2014.) This is merely a request and not a requirement.

Included Files

Folder "Mac": Programs "texture" and "texture_image" – universal binaries for PowerPC or 32/64-bit Intel Mac.

Folder "Windows": Programs "texture.exe" and "texture_image.exe" – 32-bit Windows programs.

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.

Changes from Version 1.2

The software and algorithm contain several improvements since version 1.2. The algorithm includes a new nonlinear contrast stretch that improves the look of the result, especially at the tops of ridges. The program now outputs an image file directly. There is

no longer a need to choose the number of standard deviations for a contrast stretch when viewing the image; instead, the built-in contrast stretch is controlled by a new parameter called "vertical enhancement," which behaves more consistently across different terrains than the old method. The software runs faster, and there is no longer a significant speed issue when the height or width of the image is a prime number.

Running the Software

The texture shading software provided here 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 produces a texture-shaded image file as output. This guide assumes a rudimentary understanding of the terminal window.

Texture shading proceeds in two steps. The first step performs the bulk of the computation and generates an intermediate file of texture shading data. The second step adjusts the contrast characteristics for the result and produces a grayscale image file in 16-bit TIFF format, which can be read by most GIS and image processing software. Separating the process into two steps allows you to run the primary computation once and then quickly experiment with different contrast settings, without having to wait for the main computation to repeat each time.

Input data:

The software accepts input in what is often called GridFloat format. (This is also called "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 other formats can be converted to GridFloat format using a variety of software packages, including the free GDAL library from The Open Source Geospatial Foundation (see www.gdal.org). To produce the files needed for input to the texture shading program, you would use the following GDAL command:

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

Step 1 – Compute texture data:

The first computation step is performed by a program simply called "texture," which has three required parameters. For example:

```
texture 0.666667 my_elev.flt my_texture.flt
```

The first parameter is the amount of texture "detail"; this is the value called "alpha" in my NACIS presentation (which is a exponent in the fractional Laplacian equation). Lower values of detail retain more elevation information, giving more sense of the overall, large structures and elevation trends in the terrain, at the expense of fine texture

detail. Higher detail enhances the texture but gives an overall "flatter" general appearance, with elevation changes and large structure less apparent. This tradeoff may need to be adjusted individually for each map, but values around 1/2 to 2/3 work well in many cases, and a detail value as high as 1.0 or more is sometimes useful. The program allows the detail 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, both in GridFloat format. Although there are actually multiple input files (with extensions .flt,, .hdr, and optionally .prj) and likewise multiple output files, only the .flt filenames should be given on the command line. Alternatively, the .flt extensions can be omitted and the program will assume them.

For example, the command shown above would produce texture-shading data from elevation files my_elev.flt and my_elev.hdr using a detail value of 2/3. The command can also be entered like this:

```
texture 2/3 my_elev my_texture
```

Either one will produce two output files: my_texture.flt and my_texture.hdr, which will be used in Step 2. If an input file my_elev.prj exists, a corresponding output file my_texture.prj will also be created. If any of the output files already exist, they will be overwritten without prompting the user.

Note: Mac OS or Linux/Unix users will need to add “`./`” (dot slash) to the front of the command name, unless the current directory (or other location of the “texture” program) is in their search path. For example:

```
./texture 2/3 my_elev my_texture
```

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

```
Terrain texture shading program - version 1.3.1, built Nov 24 2013
Reading input files...

Input data appears to be in lat/lon (geographic) coordinates.
Assuming pixel aspect ratio of 0.828 based on latitude range.
Processing 3612 column x 3612 row array using detail = 0.666667...
Processing phase 1...
Processing phase 2...
Processing phase 3...
Processing phase 4...
Processing phase 5...
Processing phase 6...
Processing phase 7...
Writing output files...
DONE.
```

The program does not interpret the projection information in the .prj file (if one was provided), 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 that this information is correct in the printed output.

Running this step of the processing will take from a few seconds to a few minutes, depending on the size of the input terrain file. On a 64-bit system, the texture shading software is also designed to handle extremely large terrain files (larger than the size of the computer's RAM memory), but in that case the processing will take much longer because it will need to transfer data to and from the hard disk several times.

Step 2 – Select contrast and generate image:

After completing the first processing step, the program "texture_image" is used to generate a texture-shaded image file. It also has three required parameters. For example:

```
texture_image +2.0 my_texture.flt my_image.tif
```

or:

```
./texture_image +2.0 my_texture my_image
```

The first parameter is called "vertical enhancement." Higher numbers increase contrast in the midtones, but may lose detail in the lightest and darkest features. Lower numbers highlight only the sharpest ridges and deepest canyons but reduce contrast overall. As with the "detail" parameter, vertical enhancement may need to be adjusted for each terrain region on a case-by-case basis. In addition, different detail settings for the same terrain will likely require different vertical enhancement values. Typical values for this parameter may be around –2.0 to 5.0. Areas of low relief will generally require higher values than steep, rugged terrain. If the resulting image is mostly in the medium grays, then a higher vertical enhancement is needed.

As before, the other two parameters are the input and output filenames. The input filename should be the .flt file created previously by the "texture" program. The output will have a .tif extension and a companion "world file" with a .tfw extension. Many GIS programs will read the .tfw file along with the TIFF image file and can use this to georeference the image if the map projection is also known. If a .prj file exists for the input file, one will be created for the output file as well.

Very basic help information is available for the "texture" and "texture_image" programs – if you run them with no parameters (e.g., just type "`texture`" or "`./texture`"), each one will print a brief summary 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 texture shading 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 output:

The texture shading output TIFF file should be read and handled as a 16-bit raster image, not as terrain elevation data. It should be displayed with a contrast stretch that includes the whole 16-bit range of 0 to 65535, not just the min/max range of values in the file.

Suggestions for using color:

One way to produce a colored image from the texture shading output would be to replace the shades of gray in the output file with a different color ramp. Keep in mind that the resulting colors will not represent elevation ranges as with hypsometric tints. For example, you could use reds to represent ridges (high values) and blues to represent canyons (low values), with white in between. This would show the same information as the grayscale image, but using color.

A more typical use of the output would be to produce a texture-shaded grayscale image 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 allows somewhat higher values of the "detail" 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 "detail." 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 (i.e., elevation differences) 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 detail and vertical enhancement 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 using either a larger vertical enhancement or higher detail. With this technique a detail value 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 clip it to 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	18,000 × 18,000	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	5° × 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	1½° × 2°	1½° × 2°
3 GB	2° × 2°	2° × 2½°
4 GB	2° × 2°	2° × 4°
6 GB		3° × 4°
8 GB		4° × 4°

Again, larger data sets can be processed (at least on a 64-bit system), but with a substantial increase in running time (hours, instead of minutes).

Miscellaneous Notes

Quality of data: The method benefits from both high horizontal resolution and high vertical precision in the input data. 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 three-dimensional appearance of the terrain. Texture shading provides a related "vertical enhancement" parameter to achieve a similar effect. Multiplying the input elevation data by an exaggeration factor is no longer necessary, since exactly the same result can be achieved by changing the vertical enhancement value. In addition, an input exaggeration factor has less effect on texture shading with low "detail" values, whereas changing the vertical enhancement parameter is designed to have a more consistent effect across different values of detail.

Negative detail: It is possible to supply a negative detail 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 detail value. Unlike most blur filters which are controlled by a size parameter such as a blur radius, the effect here is like trying every choice of blur radius and then producing a weighted average of all those results. The controlling parameter is this case is the "detail," 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 resulting image.

The more negative the detail, the stronger the blurring effect. Useful values range between about 0 and –1 (with –1 giving a very strong blur). Note that unusually high or low (positive or negative) values of vertical enhancement may be needed when negative detail 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. Currently the sample files were all created from an earlier version of texture shading, and the results will not match exactly.

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 detail to a grayscale image (in this case, hillshaded relief of Mt. Rainier). The images use detail values of 0, –1/4, –1/2, –3/4, and –1, respectively.

These image files were 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 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. 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 shading artifacts.

Notes for Software Developers

Besides the command-line interface, the texture shading algorithm can also be called directly from user-developed software. The main 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. These values can then be converted to image pixel values by using the function terrain_image_data() declared in the same file.

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() but before terrain_image_data().

Compiling from Source

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

```
gcc -O2 -funroll-loops -DNOMAIN -c *.c
gcc –O2 –funroll-loops *.o texture.c       -o texture
gcc –O2 –funroll-loops *.o texture_image.c -o texture_image
```

Use these commands on Mac/Linux/Unix. Or if using MinGW on Windows, add ".exe" to the output filenames "texture" and "texture_image".

The programs 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 itself, 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. The interface to the FFT layer is designed to be generic enough to allow replacement of the FFT implementation included here with any other general-purpose FFT library if so desired. The included FFT implementation is also interesting in its own right; it performs discrete cosine transforms (DCT) on data of any size, and handles even prime sizes with reasonable efficiency.

Top Layer:

texture.c (main function for "texture" program)
texture_image.c (main function for "texture_image" program)
read_grid_files.h
read_grid_files.c
write_grid_files.h
write_grid_files.c
WriteGrayscaleTIFF.h
WriteGrayscaleTIFF.c

Middle Layer:

terrain_filter.h (interface to texture shading algorithm)
terrain_filter.c
compatibility.h (helps deal with compiler dependencies)
dct.h (generic interface to FFT code in bottom layer)

Bottom Layer:

compatibility.h (also used by middle layer)
dct_fftpack.c (adapts FFT library to dct.h interface)
fftpack.h (general-purpose DCT and real FFT library)
fftpack.c
transpose_inplace.h (general-purpose matrix transpose)
transpose_inplace.c

Acknowledgments

Many thanks to Tom Patterson and Brett Casebolt for their helpful suggestions and feedback, which have led to an improved texture shading algorithm.

Contact Information

Please feel free to contact me with any comments, questions, or bug reports on the software or the algorithm. I prefer not to publish my email address, but I can be contacted through box.com or through cartotalk.com (both of which require creating a free account). My username on both sites is Leland Brown.

November 24, 2013
Leland Brown