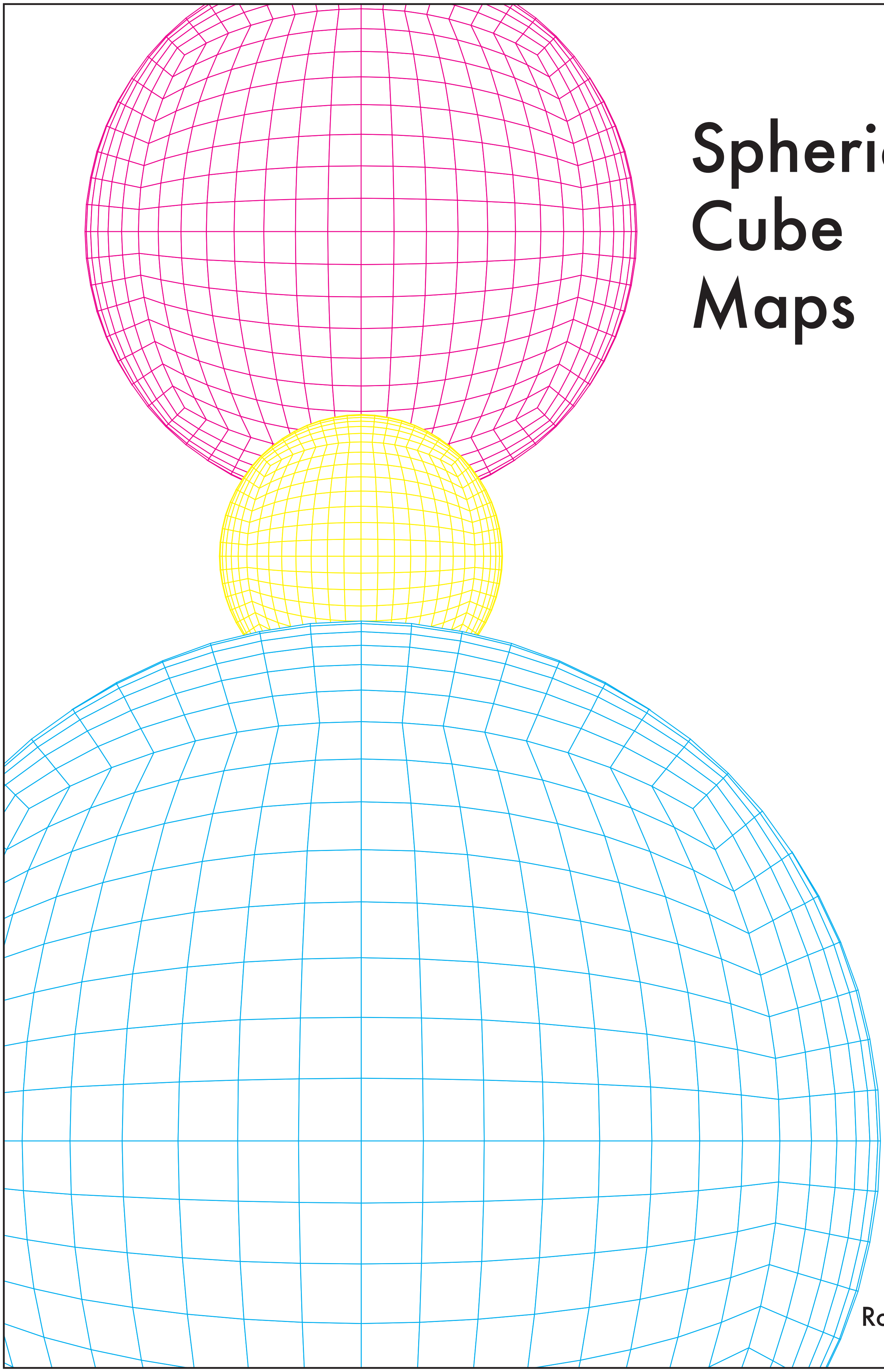


Spherical Cube Maps

Robert Kooima
2012

Contents

1.1 SPHERICAL SAMPLING

The mapping of image data onto the sphere is intuitively similar to a standard OPENGL cube map. Begin by considering an $n \times n$ raster image applied to the $+Z$ face of a cube. For a spherical cube map, the pixel at row r and column c gives a sample at a position (α, β) on the sphere, where

$$\alpha = 90^\circ \frac{c + 1/2}{n} - 45^\circ \qquad \beta = 90^\circ \frac{r + 1/2}{n} - 45^\circ.$$

This corresponds to the 3D vector

$$v = \begin{bmatrix} \sin\alpha \cos\beta \\ -\cos\alpha \sin\beta \\ \cos\alpha \cos\beta \end{bmatrix}.$$

This vector has length $\sqrt{\cos^2 \alpha + \sin^2 \alpha \cos^2 \beta}$, but when normalization is required, one will probably prefer the more familiar divisor $\sqrt{v_x^2 + v_y^2 + v_z^2}$.

Vectors within the remaining cube faces are simple 90° rotations of the definition for $+Z$, trivially implemented in the form of swizzles and negations. For each face, the vector (x', y', z') is defined in terms of (x, y, z) as follows.

	X	$-X$	Y	$-Y$	Z	$-Z$
$x' =$	z	$-z$	x	x	x	$-x$
$y' =$	y	y	z	$-z$	y	y
$z' =$	$-x$	x	$-y$	y	z	$-z$

The cube face orientations given by this swizzle table match the definition of a standard OPENGL linear cube map. It is the non-linear mapping from row and column to 3D vector that puts the “spherical” in “spherical cube map.” While more expensive to compute, this mapping is more amenable to the delivery of high resolution spherical data sets, as it provides a more uniform tesselation of the sphere, and thus a more consistent density of data at every point on its surface.

As depicted by Figure 1.1(a), linear cube map samples stretch in the center of the face and compress toward the edges. While all tessellations of the sphere necessarily demonstrate some degree of similar non-uniformity, the spherical cube map, Figure 1.1(b), is visibly closer to the impossible ideal. Specifically, the very smallest samples of a linear cube map, found at the corners, have only 19% of the area of the largest samples at the cube face centers. That is, samples near the center of a cube map face cover more than *five times* the area of samples at the corners. In contrast, the smallest samples of a spherical cube map face, found at the centers of the face edges,

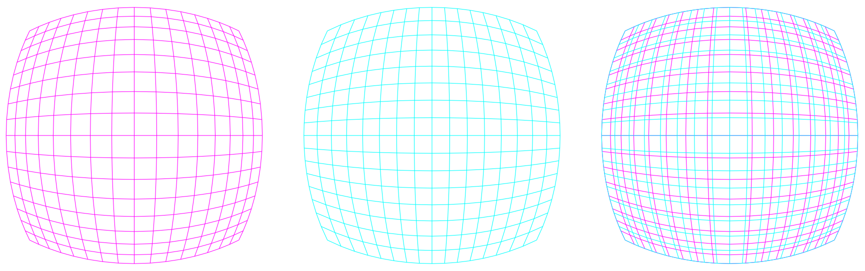

(a) Linear cube map (b) Spherical cube map (c) Overlaid

Figure 1.1: A comparison of linear and spherical cube map sample uniformity

have 70% of the area of the largest samples at the face centers.[1] The corner samples of an SCM are actually *not* the smallest, with 76% the area of the center samples.

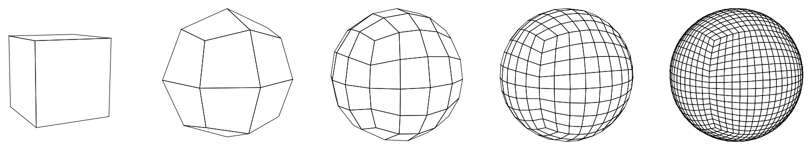
Similarly, the shortest edge of the linear cube map is only 47% of the length of the longest, while the shortest edge of the spherical cube map is 70% of the length of the longest.[2] Notably, the vertical and horizontal edges along the center of the SCM page all have equal lengths, which means that an SCM data set has constant data density along the equator, as well as along the four 90° meridians.

1.2 PAGING

While the SCM mapping does a good job of uniformly representing data on the surface of the sphere, we're not content to simply map six large images onto the six faces of an inflated cube. Large images are slow to load, and the ultimate objective of the SCM data structure is to support real-time viewing with near-instantaneous data access. To render a multi-giga-pixel spherical data set in real-time, an application must calculate the specific subset of the data visible to the user in each frame, as well as the minimum data resolution necessary to fill all pixels of the user's display. Toward that end, the SCM represents a large data set in the form of a number of small pages, at a range of resolutions, organized as a set of six quadtrees.

Cube faces are recursively subdivided, as shown in Figure 1.2, with each four-sided face cut into four child faces, and the newly-created center vertex placed on the surface of the sphere. Each face of the resulting polyhedron gives one spherically-mapped page of image data. A full SCM tree gives all levels of detail, up to and including the native resolution of the source data set. The six pages at depth zero (Figure 1.2(a)) give low-resolution coverage of the entire sphere. The 24 pages at depth one (Figure 1.2(b)) give coverage at twice that density. With each successive increase in depth, the number of pages and the total quantity of image data increases by a factor of four.

Let d be the number of recursive subdivisions applied to the cube. The total number



(a) 0 (b) 1 (c) 2 (d) 3 (e) 4

Figure 1.2: Recursive subdivisions of the cube.

d	w_{512}	h_{512}	w_{360}	h_{360}	*count*
0	2048	1024	1440	720	6
1	4096	2048	2880	1440	30
2	8192	4096	5760	2880	126
3	16,384	8192	11,520	5760	510
4	32,768	16,384	23,040	11,520	2046
5	65,536	32,768	46,080	23,040	8190
6	131,072	65,536	92,160	46,080	32,766
7	262,144	262,144	184,320	92,160	131,070

Table 1.1: Effective resolutions and page counts of SCM trees with various depths and page sizes.

of pages in an SCM tree of depth d is

$$count(d) = 2^{2d+3} - 2.$$

Let n be the size in pixels of each page of image data. The effective sphere map resolution of an SCM tree with page size n and depth d is

$$width(d) = n\,2^{d+2} \qquad height(d) = n\,2^{d+1}.$$

For reference, Table 1.2 shows the effective resolutions and page counts of SCM trees with depths down to 7. Page size $n = 512$ is often used for power-of-two source images, and $n = 360$ works well for geographic and planetary gridded data.

Each page is identified by an index between 0 and $count(d)$ giving the page's position in a breadth-first enumeration of the page tree. As with all basic definitions shown so far, this root configuration coincides with the definition of a standard OpenGL cube map. Pages zero through five map onto the faces of a cube as follows.

$$p_0 = +X \quad p_1 = -X \quad p_2 = +Y \quad p_3 = -Y \quad p_4 = +Z \quad p_5 = -Z$$

The page index uniquely determines all of a page's parameters. Its size, position, and orientation on the sphere, plus its parent, children, and neighbors can all be calculated in constant time given only the page index i. The next several equations build the integer arithmetic giving this capability.

The levels of the SCM tree are enumerated in increasing order. Within each level, the sub-pages of each root page are contiguous. Thus, each level ℓ, $0 \le \ell \le d$ consists of six arrays of $2^\ell \times 2^\ell$ pages. The level of a given page i is

$$level(i) = \frac{\lfloor \log_2(i+2) \rfloor - 1}{2}.$$

This integer binary logarithm is used heavily when computing page index relationship, and an efficient C implementation is given in Appendix A.

The root page of a page i, its earliest ancestor, is

$$root(i) = (i - 2\,(4^{level(i)} - 1)) \;/\; 4^{level(i)}.$$

The pages of the $2^\ell \times 2^\ell$ array for each level and root are enumerated beginning at the top left, going left-to-right and top-to-bottom. The rank of a page in this array is

$$rank(i) = (i - 2\,(4^{level(i)} - 1)) \;\%\; 4^{level(i)}.$$

Note the use of integer division and modulus, plus the useful symmetry in the definitions of *root* and *rank*. The row and column in the array follow directly from this rank.

$$row(i) = rank(i) \;/\; 2^{level(i)}$$

$$col(i) = rank(i) \;\%\; 2^{level(i)}$$

Given these, the SCM spherical mapping defined in Section 1.1 may be applied, and the 3D position vector of each page corner may be determined. To the west and east,

$$\alpha_w = 90^\circ \frac{col(i)}{2^{level(i)}} - 45^\circ \quad \text{and} \quad \alpha_e = 90^\circ \frac{col(i)+1}{2^{level(i)}} - 45^\circ.$$

To the north and south,

$$\beta_n = 90^\circ \frac{row(i)}{2^{level(i)}} - 45^\circ \quad \text{and} \quad \beta_s = 90^\circ \frac{row(i)+1}{2^{level(i)}} - 45^\circ.$$

The vector pointing toward the north-west corner of page i is thus

$$v_{nw} = \begin{bmatrix} \sin\alpha_w \cos\beta_n \\ -\cos\alpha_w \sin\beta_n \\ \cos\alpha_w \cos\beta_n \end{bmatrix},$$

subject to the table of swizzles and negations given in Section 1.1. The remaining corners v_{ne}, v_{sw}, and v_{se} follow accordingly.

Moving on, just as a page's index uniquely determine its *root*, *level*, *row*, and *col*, so too do these four values together uniquely determine an index.

$$index(f, \ell, r, c) = count(\ell - 1) + f\,4^\ell + r\,2^\ell + c.$$

Given this two-way set of definitions, parent, child, and neighbor relationships emerge. The parent is one level higher,

$$\begin{aligned} \mathit{parent}(i) = \mathit{index}(&\mathit{root}(i), \\ &\mathit{level}(i) - 1, \\ &\mathit{row}(i) \ / \ 2, \\ &\mathit{col}(i) \ / \ 2), \end{aligned}$$

and the four children $0 \leq k < 4$ are one level lower,

$$\begin{aligned} \mathit{child}(i,k) = \mathit{index}(&\mathit{root}(i), \\ &\mathit{level}(i) + 1, \\ &2 \cdot \mathit{row}(i) + k \ / \ 2, \\ &2 \cdot \mathit{col}(i) + k \ \% \ 2). \end{aligned}$$

The calculation of neighbor indices is a bit more complicated. In the trivial case, the approach is identical to the calculation of a parent or child. One has only to increment or decrement the row or column argument to $\mathit{index}(f, \ell, r, c)$ to determine any adjacency. However, finding neighbors across cube map face boundaries causes complications due to the varying orientations of the roots. The face swizzle table comes into play and the resulting cases are tedious to derive. They are fully enumerated here for reference and ease of implementation.

$$\mathit{north}(f,\ell,r,c) = \mathit{index}\left(\begin{array}{cccc} f, & \ell, & r-1, & c \\ 2, & \ell, & 2^\ell - c - 1, & 2^\ell - 1 \\ 2, & \ell, & c, & 0 \\ 5, & \ell, & 0, & 2^\ell - c - 1 \\ 4, & \ell, & 2^\ell - 1, & c \\ 2, & \ell, & 2^\ell - 1, & c \\ 2, & \ell, & 0, & 2^\ell - c - 1 \end{array}\right) \begin{array}{l} \text{if } r > 0 \\ \text{if } f = 0 \\ \text{if } f = 1 \\ \text{if } f = 2 \\ \text{if } f = 3 \\ \text{if } f = 4 \\ \text{if } f = 5 \end{array}$$

$$\mathit{south}(f,\ell,r,c) = \mathit{index}\left(\begin{array}{cccc} f, & \ell, & r+1, & c \\ 3, & \ell, & c, & 2^\ell - 1 \\ 3, & \ell, & 2^\ell - c - 1, & 0 \\ 4, & \ell, & 0, & c \\ 5, & \ell, & 2^\ell - 1, & 2^\ell - c - 1 \\ 3, & \ell, & 0, & c \\ 3, & \ell, & 2^\ell - 1, & 2^\ell - c - 1 \end{array}\right) \begin{array}{l} \text{if } r < 2^\ell \\ \text{if } f = 0 \\ \text{if } f = 1 \\ \text{if } f = 2 \\ \text{if } f = 3 \\ \text{if } f = 4 \\ \text{if } f = 5 \end{array}$$

$$west(f,\ell,r,c) = index\begin{pmatrix} f, & \ell, & r, & c-1 \\ 4, & \ell, & r, & 2^\ell-1 \\ 5, & \ell, & r, & 2^\ell-1 \\ 1, & \ell, & 0, & r \\ 1, & \ell, & 2^\ell-1, & 2^\ell-r-1 \\ 1, & \ell, & r, & 2^\ell-1 \\ 0, & \ell, & r, & 2^\ell-1 \end{pmatrix}\begin{matrix} \text{if } c>0 \\ \text{if } f=0 \\ \text{if } f=1 \\ \text{if } f=2 \\ \text{if } f=3 \\ \text{if } f=4 \\ \text{if } f=5 \end{matrix}$$

$$east(f,\ell,r,c) = index\begin{pmatrix} f, & \ell, & r, & c+1 \\ 5, & \ell, & r, & 0 \\ 4, & \ell, & r, & 0 \\ 0, & \ell, & 0, & 2^\ell-r-1 \\ 0, & \ell, & 2^\ell-1, & r \\ 0, & \ell, & r, & 0 \\ 1, & \ell, & r, & 0 \end{pmatrix}\begin{matrix} \text{if } c<2^\ell \\ \text{if } f=0 \\ \text{if } f=1 \\ \text{if } f=2 \\ \text{if } f=3 \\ \text{if } f=4 \\ \text{if } f=5 \end{matrix}$$

As shorthand, let $north(i) = north(root(i), level(i), row(i), col(i))$, etc., and note that our complication has a complication. Due to potential changes in page orientation, diagonal adjacency does not follow directly from nested evaluations of cardinal adjacency. Instead,

$$northwest(i) = \begin{cases} north(west(i)) & \text{if } root(i) = root(west(i)) \\ west(north(i)) & \text{otherwise,} \end{cases}$$

$$northeast(i) = \begin{cases} north(east(i)) & \text{if } root(i) = root(east(i)) \\ east(north(i)) & \text{otherwise,} \end{cases}$$

$$southwest(i) = \begin{cases} south(west(i)) & \text{if } root(i) = root(west(i)) \\ west(south(i)) & \text{otherwise,} \end{cases}$$

$$southeast(i) = \begin{cases} south(east(i)) & \text{if } root(i) = root(east(i)) \\ east(south(i)) & \text{otherwise.} \end{cases}$$

This completes the set of mathematical tools needed to immediately determine all parameters of any SCM page.

As a multi-page raster image, An SCM tree is naturally amenable to storage in the form of a TIFF image file. In this form, SCM data is accessible to a variety of existing applications and tools, most notably the libTIFF image library.

2.1 SCMTIFF

Spherical data sets in a variety of image formats are converted to SCM TIFF files by the `scmtiff` utility. Given the variety of source data preparations, this conversion involves multiple steps, and is often directed by a script or build system. `scmtiff` is a multi-tool that implements all of the different processes needed when traversing the path from raw data to usable SCM. It takes three global options followed by a list of per-process options and a list of input files.

`scmtiff -p` ⟨*process*⟩ [`-o` ⟨*output*⟩] [*options*] ⟨*input*⟩ [...]

`-p` ⟨*process*⟩ Selects the SCM process. Alternatives include `convert`, `combine`, `mipmap`, `border`, `finish`, `normal`, and `sample`. Each of these processes is described here, and examples of complete conversions are given below.

`-o` ⟨*output*⟩ Gives the name of the output file. If left unspecified, the default is `out.tif`, unless otherwise noted.

`-T` Requests that process timing be collected and printed to the terminal upon completion.

An SCM TIFF file produced by the `scmtiff` tool is a standard BigTIFF image, suitable for processing using any BigTIFF-compatible software, including libTIFF version 4 or later, and its related utilities.

Convert

`scmtiff -p convert` [`-o` ⟨*output*⟩] [*options*] ⟨*input*⟩ [...]

The `convert` process is the first and most basic, taking source data input and producing SCM TIFF output. Source data with an equirectangular projection may be provided in JPEG, PNG, or TIFF format, with up to four channels. Data with more complex projection may be provided in PDS format, as attached-label IMG files or detached-label LBL / IMG pairs. Channels may have 8 or 16-bit signed or unsigned integer samples, or 32-bit floating point samples.

If the output file name is not specified using the `-o` option to `scmtiff`, the default name is generated by replacing the file extension of the input file name with `.tif`. This is important when multiple input files are given on a single command line.[1]

`convert` has the most options of any process, as the parameters specified at the first step are carried through into the final data product.

`-n` $\langle n \rangle$ Gives a value for n, the SCM page size. Default is 512.

`-d` $\langle d \rangle$ Gives a value for d, the SCM tree depth. Default is 0. See Section **??** for definitions and examples of how the selection of n and d affect image resolution and file size.

`-b` $\langle b \rangle$ Overrides the number of bits per channel. By default, `scmtiff` uses the bit depth specified by the input image file. If a change in bit depth is desired, this value b is used instead. Values of 8 and 16 select integer samples and 32 selects floating point samples.

`-g` $\langle g \rangle$ Overrides the signedness of integer data. By default, `scmtiff` uses the type specified by the input file, but if a change is desired, 0 specifies unsigned and 1 signed. This option has no effect if floating point samples are selected.

`-N` $\langle n_0 \rangle$`,`$\langle n_1 \rangle$ Specifies a normalization range. Unsigned integer samples have a natural normalized range of $(0,1)$, signed integer samples have the normalized range of $(-1,1)$, and floating point samples have the full range of a 32-bit float. This option remaps each sample onto the range (n_0, n_1). The choice of normalization depends largely on the character of the input and the type of the output. The default normalization retains the natural range of the data.

`-E` $\langle w \rangle$`,`$\langle e \rangle$`,`$\langle s \rangle$`,`$\langle n \rangle$ Specifies a range for equirectangular inputs *other than* PDS. This allows a JPEG, PNG, or TIFF to represent a subset of the sphere. The default is $0°$, $360°$, $-90°$, $90°$.

`-L` $\langle \lambda_c \rangle$`,`$\langle \lambda_0 \rangle$`,`$\langle \lambda_1 \rangle$ Specifies a longitudinal mask. This causes data to appear over a given range of longitudes, fading out at the edges, thus allowing the blended combination of separate source data files. This option chooses a range of degrees centered at λ_c, extending to $\lambda_c \pm \lambda_1$, and fading with cubic drop-off to $\lambda_c \pm \lambda_0$.

`-P` $\langle \phi_c \rangle$`,`$\langle \phi_0 \rangle$`,`$\langle \phi_1 \rangle$ Specifies a latitudinal mask. This option chooses a range of degrees centered at ϕ_c, extending to $\phi_c \pm \phi_1$, and fading with cubic drop-off to $\phi_c \pm \phi_0$. Longitudinal and latitudinal masks may be applied simultaneously.

Combine

`scmtiff -p combine` [`-o` ⟨*output*⟩] [*options*] ⟨*input*⟩ [. . .]

The `combine` process is the second step in cases where a data set is provided in a form spread across multiple data file. It merges multiple converted SCM TIFF input files into a single SCM TIFF output file. It takes just one optional argument.

`-m` ⟨*mode*⟩ Specifies the operator mode used to combine samples. The `max` mode selects the largest sample from each of the named files. The `sum` mode is the default and sums all named files. The `max` mode is often used when merging data sets of differing projection, while the `sum` mode is used when combining data sets that have had a longitudinal or latitudinal mask applied in the `convert` process.

Mipmap

`scmtiff -p mipmap` ⟨*input*⟩

The `mipmap` process generates the intermediate nodes of an SCM tree. In general, the `convert` process generates tree leaves, the `combine` process merges trees, and the `mapmap` process subsamples the result, enabling real-time display of the data at arbitrary resolution. It is important to `combine` before `mipmap`ping, as boundary artifacts may remain apparent in the intermediate pages of combined mipmaps.

The `mipmap` process takes no parameters. In the interest of efficiency, mipmapping occurs *in place*. The output file name option is ignored and subsampled pages are appended to the existing file. The `mipmap` process generates as many subsampled levels as possible, and in the common case, an input with depth d will have $d-1$ levels appended to it.

Border

`scmtiff -p border` [`-o` ⟨*output*⟩] ⟨*input*⟩

In truth, an SCM with page size n is stored in a TIFF file with image width and height $n+2$. The `border` process fills these extra pixels in the outermost rows and columns of each page with pixels from the adjacent rows or columns of the four neighboring pages. This provides each page with the context needed by the graphics hardware to perform linear magnification filtering across page boundaries, effectively allowing many small images to appear as a single very large image.[2] If this step is not performed, artifacts will appear in the output. In practice, the `border` process is expensive, as it must decompress and recompress every page of the SCM.

Finish

`scmtiff -p finish` [*options*] ⟨*input*⟩

The `finish` process calculates and appends SCM metadata for the given SCM TIFF file. This metadata is three-fold: offsets, bounds, and description.

1. A sorted array of page indices and file offsets allows an SCM rendering application to immediately determine exactly where each page appears in the SCM TIFF file. Given an awareness of how points on the sphere map onto indices, the page catalog generated by the `finish` process allows applications to straightforwardly locate the unique SCM TIFF data that maps onto any point on the sphere at any resolution.
2. The minimum and maximum values of each channel of each page allow a displacement-mapping application to determine a tight 3D bounding volume for each page. This enables efficient visibility determination and view culling.
3. An image description documents the provenance of an image, noting its source and copyright holder.

Like mipmapping, the `finish` process occurs *in place*. The output file name option is ignored and the generated data is appended to the input file.

`-t` ⟨*file*⟩ Specifies a text file containing an image description. The contents of this file are read and embedded in the generated TIFF file.

`-l` ⟨ℓ⟩ Specifies a *leaf subdivision depth*. This argument allows the depth of the bounding value cache hierarchy to exceed the depth of the actual page hierarchy. The rationale for this feature is complex, but as we will see in Section 4.1, there is a mismatch between the resolution of a displacement map and the mesh geometry that it displaces. Leaf subdivision accounts for this mismatch. As a rule of thumb, ℓ should give the base-two logarithm of the mesh's down-sampling factor. For example, when finishing a 512×512 displacement map for display on a 128×128 mesh, $\log_2 512/128 = 2$, so specify $\ell = 2$. For most imagery, leaf subdivision is not beneficial, and the default value of zero should be used.

Normal

`scmtiff -p normal [-o` ⟨*output*⟩] [*options*] ⟨*input*⟩

The `normal` process computes a normal map for a given height map. The input may have any number of channels and any format, but only the first channel is used, and the normalization of that channel is modified by the range of radii specified by the command line option. The output will always have 3 channels of 8-bit unsigned samples, scaled and biased to $(0, 1)$ as is standard practice in normal mapping, though the normals are given in *object* space rather than *tangent* space.

Regardless of whether the input height map was bordered or finished, the output normal map will be neither bordered nor finished. In contrast, a mipmapped input height map *will* produce a mipmapped normal map output, and indeed one should *always* normalize mipmaps instead of mipmapping normals, as a mipmapped normal is not guaranteed to have unit length.

`-R` $\langle r_0 \rangle$,$\langle r_1 \rangle$ Specifies the true range of radii represented by the normalized height map input. This allows the true magnitude of the *change* in height to be computed, which gives slope, which gives the normal vector.

Sample

`scmtiff -p sample` [*options*] $\langle$*input*$\rangle$

The `sample` process is not a tool for preprocessing SCM data, but for working with already-processed data. It queries the contents of the SCM TIFF file named on the command line, and produces no new SCM TIFF output. The `sample` process receives queries as latitude-longitude pairs, in degrees, on `stdin`. It reports the results on `stdout`.

`-R` $\langle r_0 \rangle$,$\langle r_1 \rangle$ Specifies the true range of radii represented by the input. This allows the magnitude of the query to be correctly reported, even if the data has been renormalized during the `convert` process.

For example, to query the radius in meters of the landing point of Apollo 16, as given by the merged lunar heightmap prepared in the example of Section 2.3:

```
echo "-8.973 15.500" | scmtiff -p sample -R 1728240,1748170 DTM.tif
1737382.875000
```

The `sample` process does *not* expect a finished SCM and will perform a scan of the SCM TIFF file prior to responding to queries. Thus the very first query may take a few moments to process, while subsequent queries will process immediately. If many queries are to be made, the best practice is to concatenate them in a file.

2.2 SCMVIEW

The `scmview` utility enables interactive inspection and side-by-side comparison of individual pages of SCM TIFF files. Figure 2.1 shows a screenshot of `scmview` in action.

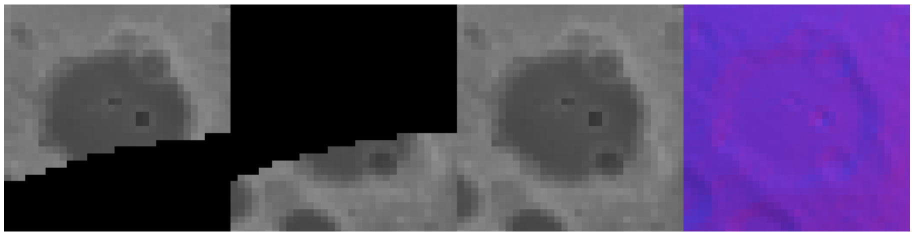


Figure 2.1: The `scmview` utility displaying four of the outputs of the conversion described in Sections 2.3 and 2.3.

There, four SCM files are displayed together, with their pan, zoom, and page number in sync. The user has focused upon Clavius, a region of interest on a lunar height map. The first frame shows a converted equirectangular projection of the map, the second shows a polar projection, and the third demonstrates their seamless merger. This output is taken from the usage example of Section 2.3. The fourth frame shows a normal map derived from the merged height map, discussed in Section 2.3.

The `scmview` utility takes a list of SCM TIFF files on the command line.

`scmview` [⟨*GLUT options*⟩] ⟨*input*⟩ [. . .]

Keyboard and mouse inputs are as follows.

Left Mouse	. . . Click and drag to pan.
Right Mouse	. . . Click and drag to zoom.
Return	. . . Reset the pan and zoom.
Page Up	. . . Go to the next page.
Page Down	. . . Go to the previous page.
Shift Page Up	. . . Scan for the next present page.
Shift Page Down	. . . Scan for the previous present page.
$\begin{array}{ccc}7 & & 9\\ & \times & \\ 1 & & 3\end{array}$	. . . Go to one of the four child pages.
$\begin{array}{ccc} & 8 & \\ 4 & + & 6\\ & 2 & \end{array}$	. . . Go to one of the four neighbor pages.
5	. . . Go to the parent page.
0	. . . Go to the root page.

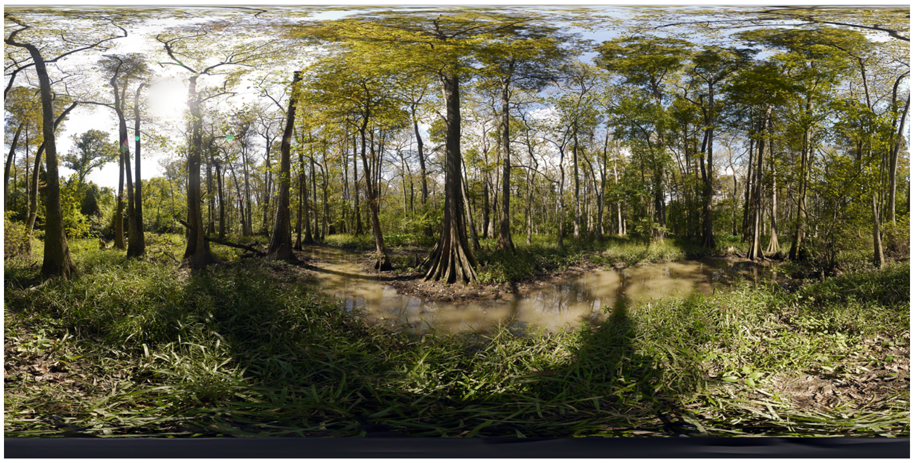


Figure 2.2: `Bluebonnet-0-L.tif`, an equirectangular full-sphere panorama

F1	. . . Normal view mode.
F2	. . . False color view mode.
Escape	. . . Exit.

`scmview` is a GLUT application, and as such it takes the usual set of GLUT command line options. The most useful of these is

`-geometry` $\langle w \rangle$`x`$\langle h \rangle$ Display a window with size (w, h).

2.3 EXAMPLES

A Basic Panorama

The most straightforward SCM TIFF process uses a single image file giving an equirectangular spherical projection, such as that shown in Figure 2.2. This image is $32{,}768 \times 16{,}384$ with 3 channels of 8-bit unsigned samples. The name `Bluebonnet-0-L.tif` reflects that it was captured at the Bluebonnet Swamp Nature Center, it was the first of several captures, and it gives the left channel of a stereoscopic pair.

Table 1.2 indicates that an image with this resolution is well-represented by an SCM with page size $n = 512$ and depth $d = 4$. When finished, the SCM tree will have 2046 pages. The first step is to run the `convert` process to generate the leaves of this tree. The output file name is chosen to reflect the parameters of the SCM as well as the state of its processing. This convention is optional.

```
scmtiff -p convert -n 512 -d 4 -o Bluebonnet-0-L-512-4-M.tif \
                                  Bluebonnet-0-L.tif
```

If the output is examined using `scmview`, pages 510 through 2045 will be present with a resolution of 514×514. Image detail will closely reflect that of the input. There will be mild projection distortion apparent in some pages, though the degree of distortion will be far less than that at the top and bottom of the equirectangular input. All pages will have a border of black pixels around the outside.

The second step is to run the `mipmap` process on this output to generate the intermediate, subsampled pages.

```
scmtiff -p mipmap Bluebonnet-0-L-512-4-M.tif
```

Examination with `scmview` will show that pages 0 through 2045 are now all present. Pages 0 through 5 will show full views along each of the axes, and pages 2 and 3 will show nicely recovered polar views with all of the projection distortion in the input rectified.

The third step fills the border pixels with adjacent page information. This process alters the compressed size of each page, so it cannot be performed in-place. An output file name is required and a new SCM TIFF file is generated.

```
scmtiff -p border -o Bluebonnet-0-L-512-4.tif \
                     Bluebonnet-0-L-512-4-M.tif
```

The output is a complete SCM tree. Figure 2.3 shows the first six pages of output generated from the input in Figure 2.2.

The fourth and final step scans the file, noting the location and bounds of each page, and appending this information to the end of the file. With this, a real-time interactive application, such as the one described in the next chapter, can instantly locate, load, and display any page. A brief description is added to an ASCII text file named `desc.txt`,

```
Bluebonnet Swamp Nature Center - 9 October 2011
Copyright (c) 2011 Robert Kooima
```

and the SCM TIFF is finished.

```
scmtiff -p finish -t desc.txt Bluebonnet-0-L-512-4.tif
```

A Merged Planetary Dataset

The following example is much more involved, merging multiple data sets captured by the Lunar Reconnaissance Orbiter (LRO) to produce a globally-high-quality height map of the moon. Deriving this process requires a close familiarity with the source data, and a great deal of trial and error. However, this does represent real-world data handling and usage of `scmtiff`.

The LRO height map is based primarily on a 100 meter-per-pixel digital terrain model (DTM) derived by stereo reconstruction from imagery captured by the Wide

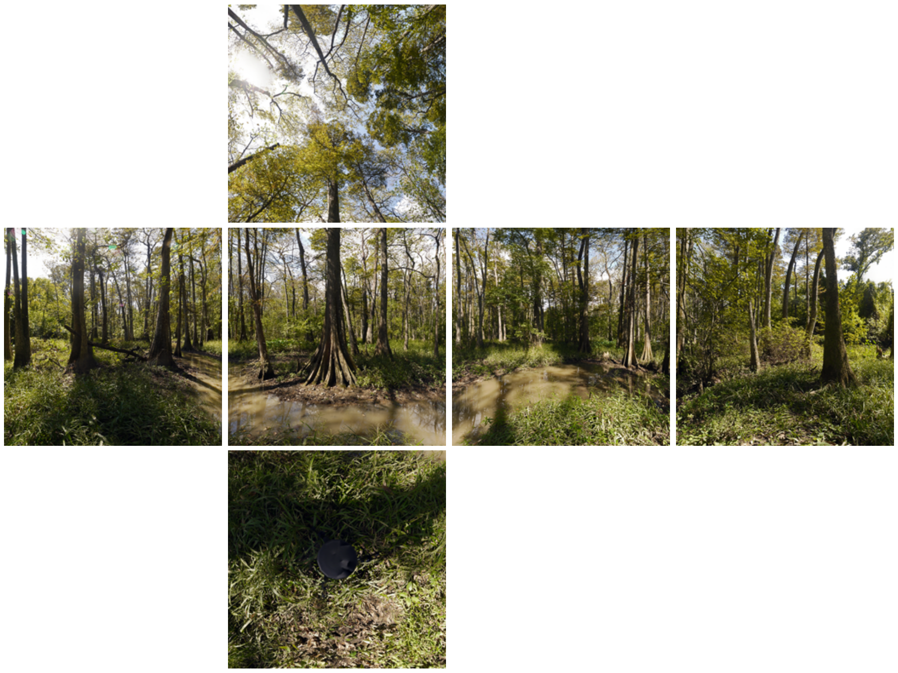


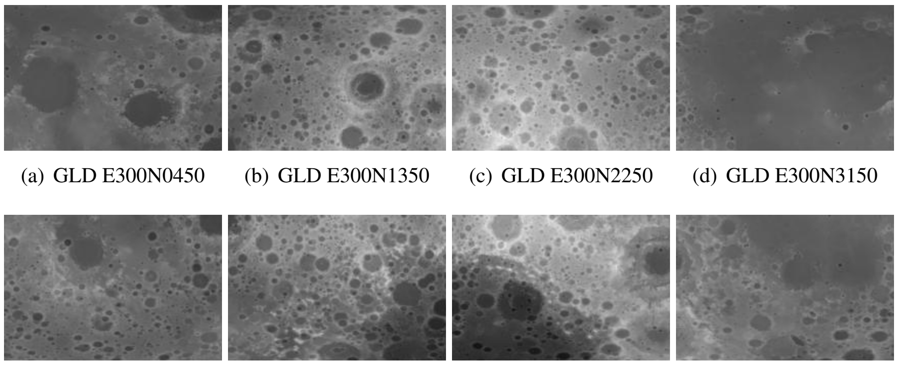
Figure 2.3: The six root pages of the Bluebonnet SCM.

Angle Camera (WAC) of the Lunar Reconnaissance Orbiter Camera (LROC).[3] This data set, known as the Global Lunar 100 Meter DTM (GLD100), gives beautifully clean terrain data over most of the moon, omitting only the poles, where illumination and stereo disparity are insufficient for reconstruction.

In the interest of optimal image representation, GLD100 is provided in several parts, with two different projections. Latitudes below 60° are given by eight equirectangular projections, shown in Figure 2.4. Each of these is 27,291 $\times$ 18,195 pixels in size, totaling 109,164 $\times$ 36,390 pixels. Latitudes above 60° are given by two stereographic polar projections, shown in Figures 2.5(a) and 2.5(b). Latitudes beyond 79° are not represented.

Fortunately, GLD100 is registered with the Lunar Digital Elevation Model (LDEM) captured by the Lunar Orbiter Laser Altimeter (LOLA).[4] Due to the polar orbit of LRO and the incredibly narrow field of view of LOLA, the coverage of LDEM is very sparse at low latitudes, but excellent near the poles. GLD and LDEM compliment one another in this regard, and we can use the latter to fill the gaps in the former. Figures 2.5(c) and 2.5(d) show polar projections of LDEM that align with those of GLD. In general, GLD is preferred to LDEM as the coverage and character of stereo reconstructed data is superior to laser altimetry data.

We need to select a page size and tree depth appropriate for this data set. There are


(a) GLD E300N0450 (b) GLD E300N1350 (c) GLD E300N2250 (d) GLD E300N3150

(e) GLD E300S0450 (f) GLD E300S1350 (g) GLD E300S2250 (h) GLD E300S3150

Figure 2.4: The eight equirectangular projections of GLD100.

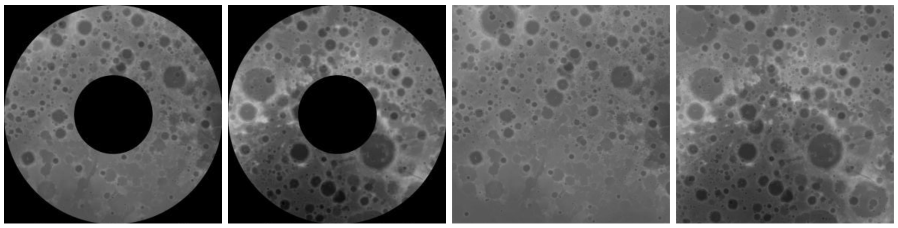

(a) GLD P900N0000 (b) GLD P900S0000 (c) LDEM 60N (d) LDEM 60S

Figure 2.5: The two polar projections of GLD100 and LDEM.

many considerations.

1. To properly represent the source data, we select n and d such that the effective width, $w = n\,2^{d+2}$, closely matches the total horizontal resolution of the input.

2. Large n incurs high I/O latency. RAM-to-VRAM data transfers take time, and if a page upload consumes a significant fraction of a frame period then the frame rate is likely to break up during high I/O load. Given 2012 era hardware, we'll want n around 512.

3. For a given w, small n requires large d. This increases the total number of pages in flight at any given moment, as well as the number of texture image units consumed by the renderer. This is usually not a problem, but depending upon the hardware, one may encounter a limit.[5]

4. The page granularity of a height map impacts the effectiveness of visibility testing, as small pages allow tighter bounding volumes than large pages. This consideration does not apply to color maps or normal maps.

5. n should be an even number to ensure no artifacts arise during the mipmapping process.

Given $w = 109{,}164$, arbitrarily choosing $d = 5$, and solving, we find $n = 852$. That's quite high. Choosing $d = 6$, we find $n = 426$, which is reasonable from most perspectives. SCM parameters $n = 426$ and $d = 6$ will give an effective resolution of $109{,}056 \times 54{,}528$ in 32,766 pages. This is a couple pixels short, but very close to perfect.

It's important to remember that an SCM data product is essentially a temporary cache for visualization. It's not a permanent archive, so source integrity need not be considered precious and we're free to renormalize. This will let us benefit from any smoothing that occurs during resampling and reprojection and take full advantage of the output bit depth. Both GLD and LDEM are signed 16-bit samples ranging from roughly -9160 m to 10,770 m, giving height values relative to the moon's average radius of 1,737,400 m. These values give our normalization range. Of course, normalization maps the data onto $(0, 1)$, so there's no longer any point in using a signed sample, so we'll override the IMG data type and force unsigned output.

Given these parameters, the conversion of the equirectangular inputs of Figure 2.4 is as follows.

```
scmtiff -p convert -n 426 -d 6 -N -9160,10770 -g 0 WAC_GLD100_*.IMG
```

The eight output SCMs will be combined using the `sum` operator. This best accommodates the behavior of edge pixels during resampling and reprojection, as an SCM sample falling only partially within the bounds of an IMG has a coverage coefficient applied. To clarify this point, if the SCM sample *should* be 1.0, but only 20% of the sample falls within the IMG, then the sample value will be 0.2. This does *not* cause edge artifacts because the adjacent IMGs may be expected to cover the remainder of the sample, and their contributions will sum to 1.0. Next we will see a case where this assumption fails, so we must write the summed combination of all equirectangular GLD inputs to a temporary file. We'll call it `DTM-E.tif`.

```
scmtiff -p combine -o DTM-E.tif WAC_GLD100_*.tif
```

The polar GLD inputs of Figures 2.5(a) and 2.5(b) have a different projection so they are not pixel-aligned with the equirectangular inputs. They overlap slightly and the coverages sum to more than 1.0. Careful scrutiny with `scmview`, as in Figure 2.1, can reveal issues like this, but usually they only became apparent as flaws in the output. To resolve the issue, the polar inputs will be combined with the already-merged equatorial inputs using the `max` operator.

But before that can be done, we must look ahead to the contribution of LDEM. As apparent in Figure 2.5, GLD and LDEM overlap *heavily*. Due to the vastly different processes that captured them, they have very different character at full resolution. In an effort to obscure any discontinuity that might emerge at their border, we interpolate from one to the other between 78° and 79° degrees. This is accomplished using a latitudinal fade during the initial conversion from IMG to SCM. Here, the northern annulus of GLD data begins to fade out within 11° of the north pole at 90°, and fades out completely within 12° of 90°.

```
scmtiff -p convert -n 426 -d 6 -N -9160,10770 -g 0 \
        -P 90,12,11 WAC_GLD100_P900N0000.IMG
```

The opposite is true for the northern cap of LDEM. It begins to fade in within 11° of 90° and fades in completely within 12° of 90°.

```
scmtiff -p convert -n 426 -d 6 -N -9160,10770 -g 0 \
        -P 90,11,12 ldem_45n_100m.lbl
```

The situation at the south pole is the same, but centered upon $-90°$.

```
scmtiff -p convert -n 426 -d 6 -N -9160,10770 -g 0 \
        -P -90,12,11 WAC_GLD100_P900S0000.IMG
scmtiff -p convert -n 426 -d 6 -N -9160,10770 -g 0 \
        -P -90,11,12 ldem_45s_100m.lbl
```

When these SCM files are summed, no discontinuity will be apparent. We write this combination of all polar data to a temporary file named `DTM-P.tif`.

```
scmtiff -p combine -o DTM-P.tif WAC_GLD100_P*.tif ldem_45*.tif
```

To complete the merger of the inputs, it remains only to combine the equirectangular and polar data sets using the max operator. As suggested above, a slight overlap between the two exists. It is too thin for interpolation, yet too thick to go unnoticed when summed.

```
scmtiff -p combine -m max -o DTM-M.tif DTM-E.tif DTM-P.tif
```

And with that, the leaves of the SCM are in place. We mipmap, border, and finish just as before to arrive at a usable data product.

```
scmtiff -p mipmap DTM-M.tif
scmtiff -p border -o DTM.tif DTM-M.tif
scmtiff -p finish DTM.tif
```

The six root pages of this SCM are shown in Figure 2.6. Note, in particular, the top and bottom pages, both of which seamlessly combine interpolations of different data sets as well as mergers of distinct projections.

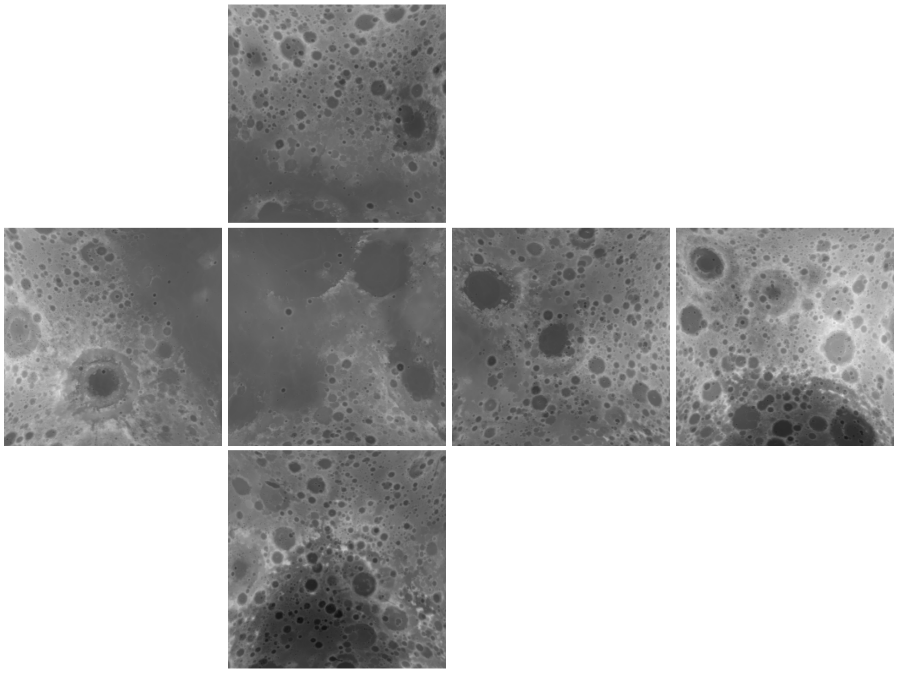


Figure 2.6: The six root pages of the lunar height map SCM.

A Normal Map

The height map generated in the previous section enables a great many visualization tasks and modes, but the highest-quality real-time illumination of it requires the derivation of a normal map. A normal map gives a surface vector for each height map sample. To compute these vectors, both the radius of the sphere and the normalization of the height must be known.

The moon has an average radius of 1,737,400 m. `DTM.tif` was prepared using a normalization of $(-9160, 10{,}770)$. Thus, the interpretation of `DTM.tif` is that a value of 0 maps onto the radius $r_0 = 1{,}737{,}400\,\text{m} - 9160\,\text{m} = 1{,}728{,}240\,\text{m}$ and a value of 1 (represented in the 16-bit unsigned TIFF by `0xFFFF`) maps onto $r_1 = 1{,}737{,}400\,\text{m} + 10{,}770\,\text{m} = 1{,}748{,}170\,\text{m}$. These are the values to provide to the `normal` process.

```
scmtiff -p normal -o DTM-O.tif -R 1728240,1748170 DTM.tif
```

The output of the normal process is neither bordered nor finished, so it is written to a temporary file. By convention, an unbordered normal map has the suffix `O`. Bordering and finishing produce the usable normal map, given the suffix `N`.

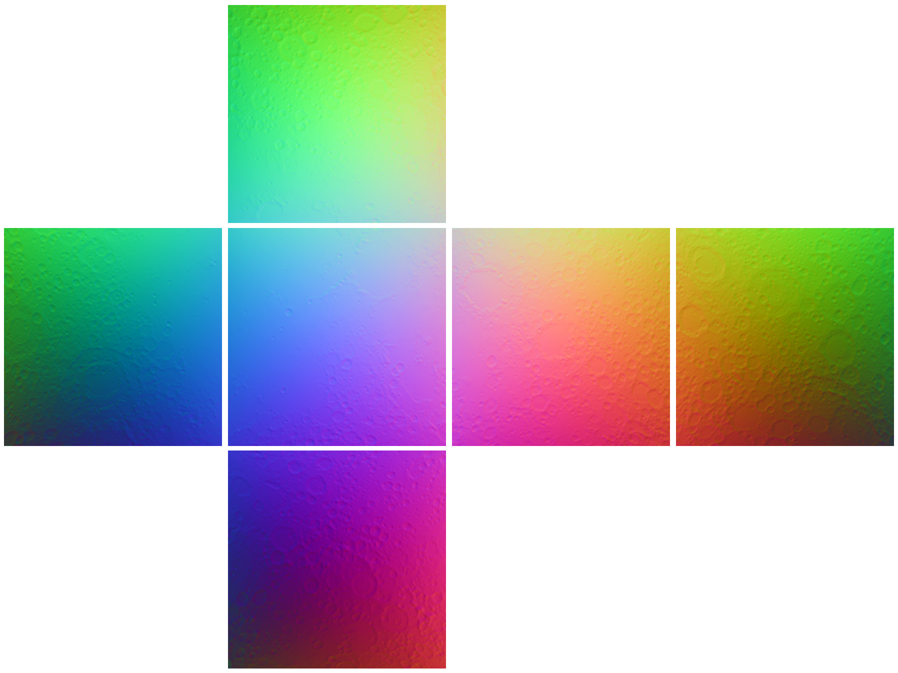


Figure 2.7: The six root pages of the lunar normal map SCM.

```
scmtiff -p border -o DTM-N.tif DTM-O.tif
scmtiff -p finish DTM-N.tif
```

The six root faces of this SCM are shown in Figure 2.7. In normal maps, positive X-facing surfaces appear red, positive Y appears green, positive Z appears blue.

2.4 AUTOMATION

For input images larger than a gigapixel or two, the SCM conversion process will take time. Because this is a multi-step process, it is convenient to automate it and allow it to run in a batch-oriented mode. The most basic way to do so is to write a shell script that issues each of the above command lines in turn. This approach can be modularized using shell variables and loops, and works well in simple cases. However, a more sophisticated approach uses the `make` utility to define the relationships between the inputs and outputs of each step of the conversion and adaptively determine the necessary order of operations automatically. The primary advantages to the use of `make` in this context are that it correctly handles interrupted conversion processes without expending redundant effort, and it intelligently exploits opportunities for process-level parallelism.

The following is an example `Makefile` that converts the equirectangular portion of the WAC GLD100 data set to a bordered, finished height map, and derives from this a bordered, finished normal map.

```
 1  N    = 426
 2  D    = 6
 3  NAME = GLD-$(N)-$(D)
 4
 5  TIFS =  WAC_GLD100_E300N0450_100M.tif \
 6          WAC_GLD100_E300N1350_100M.tif \
 7          WAC_GLD100_E300N2250_100M.tif \
 8          WAC_GLD100_E300N3150_100M.tif \
 9          WAC_GLD100_E300S0450_100M.tif \
10          WAC_GLD100_E300S1350_100M.tif \
11          WAC_GLD100_E300S2250_100M.tif \
12          WAC_GLD100_E300S3150_100M.tif
13
14  all: $(NAME)-N.tif
15
16  %.tif: %.IMG
17          scmtiff -p convert -n$(N) -d$(D) -N-9160,10770 -o $@ $<
18
19  $(NAME)-M.tif: $(TIFS)
20          scmtiff -p combine -o $@ $<
21          scmtiff -p mipmap $@
22
23  $(NAME).tif: $(NAME)-M.tif
24          scmtiff -p border -o$@ $<
25          scmtiff -p finish -t desc.txt $@
26
27  $(NAME)-O.tif: $(NAME).tif
28          scmtiff -p normal -R1728240,1748170 -o$@ $<
29
30  $(NAME)-N.tif: $(NAME)-O.tif
31          scmtiff -p border -o $@ $<
32          scmtiff -p finish -t desc.txt $@
```

The variables `N` and `D` set the page size and tree depth (lines 1–2), which are carried throughout the process, making it easy to reprocess a single input at different resolutions. The variable `NAME` is defined in terms of these values (line 3), as per the SCM naming convention. Following this, the variable `TIFS` gives a list of constituent SCM TIFF files generated by the initial conversion (lines 5–12).

The `Makefile` lists the TIFFs instead of the IMGs to enable the magic that happens next, in the `convert` recipe (lines 16–17). This is an abstract definition of the process of converting an IMG to a TIFF, including its SCM parameters, description, and normalization. When `make` is informed by the `combine` recipe that these TIFFs are required, it knows how to convert them from IMGs.

The `combine` recipe (lines 19–21) produces a single output given the eight inputs, and automatically executes the `mipmap` process on it in-place. By convention, this combined, mipmapped output is identified by the suffix `M`.

The `border` recipe (lines 23–25) re-encodes this `M` file with bordered pages, and finishes it, giving the height map output `GLD-426-6.tif`.

The `normal` recipes (lines 27–32) convert the finished height map to an intermediate normal map with the suffix `O`, which is bordered and finished giving a normal map `GLD-426-6-N.tif`.

In all cases, an output is defined in terms of its input and is annotated with the process necessary to perform the conversion. The build is invoked with a single command.

```
make
```

This command considers all recipes, compares each output with its required input, and determines the minimum number of steps necessary to produce the output. Being aware of all dependencies, it will optionally execute non-interfering processes in parallel. To allow up to four simultaneous IMG conversions,

```
make -j4
```

This `Makefile` example is simplified from the complete planetary dataset example of Section 2.3. It glosses over issues of blending and combining. A full `Makefile` capable of performing the complete lunar map conversion is given in Appendix B.

SCM RENDER LIBRARY

The SCM renderer is implemented as a small C++ library that may be embedded within any OPENGL host application.

3.1 API

The API provides three primary classes: a *cache* class that manages SCM data and implements demand-paging with a least-recently-used ejection policy, a *model* class that renders adaptive spherical geometry to which the cached imagery is applied, and a *label* class that optionally annotates the sphere and its imagery. An application that renders SCM data will require at least one cache object and one model object.

The document lists the *application* API for each class, and includes only those methods of use to rendering applications. These classes do have other public methods, though they are generally reserved for intercommunication.

class scm_cache has the following public methods.

- *scm_cache(int size)*

  Construct a new cache object with the given maximum cache size (in pages).

- *int add_file(const std::string& scm)*

  Add the named SCM TIFF file to the cache an integer descriptor for that file. A cache supports an arbitrary number of simultaneous files, and an application will usually have exactly one cache object per OPENGL context.

- *void update(int time)*

  Update the state of the cache using the given time value. To keep a cache and model in sync, and to properly manage page aging, this argument should given the value returned by *sph_model::tick*.

- *void draw()*

  Draw the entire contents of the cache in thumbnail to a 2D on-screen rectangle.

class scm_model has the following public methods.

- *scm_model(scm_cache& cache, const std::string& vert,*
  const std::string& frag,
  int mesh, double r0, double r1)

  Construct a new adaptive spherical model that renders SCM data in *cache* using the vertex and fragment shaders *vert* and *frag*. The tessellation density is given by *mesh* and the maximum expected range of displaced radii are *r0* and *r1*.

`scm-basic.vert`	...	For static spheres.
`scm-zoom.vert`	...	For zoomable panoramic SCMs.
`scm-displace.vert`	...	Displacement-mapped planetary SCMs.

Table 3.1: SCM viewer vertex shaders

- *int tick()*

  Advance and return the current model time. This value is passed to *scm_cache::update* and helps manage page age and latent-data fade-in.

- *void prep(const double *P, const double *V, int w, int h)*

  Pre-process the model for rendering to a $w \times h$ buffer. P gives the current 4×4 projection matrix and V gives the 4×4 model-view matrix. This function iterates over the SCM hierarchy and determines which pages are in view (given these transformations) and at what resolution they appear (given this buffer size).

- *void draw(const double *P, const double *V, const int *vv, int vc,*
  *const int *fv, int fc,*
  *const int *pv, int pc)*

  Draw the model using projection matrix P and model-view matrix V. The arrays *vv*, *fv*, and *pv* give file descriptors (as returned by *scm_cache::add_file*) to be supplied to the vertex shader, the fragment shader, and the pre-loader, respectively. The values *vc*, *fc*, and *pc* give the length of each of these arrays.

3.2 SCM PATH

This library has just one system-level configuration parameter:

SCMPATH is a shell environment variable akin to the bash executable path. It lists directories where SCM TIFF files may be found. If the application requests that the renderer load a file, but the renderer cannot find that file, then it will search this list of directories. Set this variable in the shell resource file, as need be, separated by colons. For example,

```
export SCMPATH=/share/scm:$HOME/data/scm:.
```

For cluster-driven display systems, try to replicate all SCM TIFFs to local directories on all rendering nodes. This will perform better than data files stored on network shares.

`scm-basic.frag`	...	Basic application of SCM color.
`scm-blend.vert`	...	Blending and animation of SCM sequences.
`scm-colormap.vert`	...	False color mapping of single-channel SCMs.
`scm-lomsee.vert`	...	Lommel-Seeliger shading with color and normal.

Table 3.2: SCM viewer fragment shaders

SCM EXAMPLE APPLICATIONS

Two example SCM rendering applications have been implemented using the Thumb framework, which enables cross-platform portability and scalability from laptops to cluster-driven virtual reality environments. The first of these applications, PANOVIEW, renders SCMfiles inside-out and is suitable for high-resolution spherical panorama rendering. The second, ORBITER, renders SCM files outside-in and is suitable for planetary rendering. While the previous chapter on SCM handling applies to any embedding of the SCM renderer, this chapter specifically describes the configuration and usage of PANOVIEW and ORBITER.

4.1 SPHERE DEFINITION FILE

The sphere definition is an XML file that organizes SCM images into visualization and gathers all of the information required by the renderer.

As Thumb-based applications, PANOVIEW and ORBITER must be able to find these XML files. They appear in the Thumb data hierarchy like any other configuration file. This means they may be placed in a data directory rooted at the current directory, or in the `~/.thumb` hierarchy, or in a data hierarchy given by the `THUMB_RO_PATH` environment variable. Sphere definition files need not be stored along side the SCM TIFF files that they reference, as the `SCMPATH` environment variable still defines the location of SCM TIFFs.

Basic Stereo Panorama

Here is `Bluebonnet-0.xml`, a basic stereoscopic panorama definition.

```
1 <?xml version="1.0"?>
2   <sphere vert="glsl/scm-zoom.vert"
3           frag="glsl/scm-basic.frag"
4           mesh="16" radius="6">
5     <scm channel="0" file="Bluebonnet-0-L-512-4.tif" />
6     <scm channel="1" file="Bluebonnet-0-R-512-4.tif" />
7   </sphere>
```

The file begins with an XML header and contains a single root sphere element with optional attributes and one sub-element for each SCM image.

The `vert` and `frag` attributes (lines 2–3) give the vertex and fragment shaders to be used by the renderer, named relative to the root of the Thumb data hierarchy. Tables 3.1 and 3.2 list the available shaders. This example is a single high-resolution stereoscopic panorama. It should be zoomable, yet it requires no special pixel processing, so the `scm-zoom.vert` vertex shader and the `scm-basic.frag` fragment shader are specified.

The `mesh` attribute (line 4) of the sphere element determines the tessellation density of the geometry mesh used to render each page of data. The example value, 16, indicates that each page of the sphere will be rendered using a 16×16 grid of polygons. This is the default, and is an appropriate value for panoramas.

The `radius` attribute (line 4) determines the radius of the spherical geometry to which the SCM imagery is applied. This value is given in meters, and is most significant in determining the apparent scale of the sphere in a VR display environment.

The `scm` sub-elements (lines 5–6) give the SCM file names. The `channel` attributes indicate which stereoscopic channel each SCM provides.

Multi-image Panorama

This example, `Taliesin-Path.xml`, is a more complex stereoscopic panorama giving a series of images for each channel. The `scm` elements are wrapped in `frame` groups and organized into a sequence. The `frag` attribute of the `sphere` element specifies the `scm-blend.frag` fragment shader, which fades smoothly from one frame to the next. The result is a panoramic virtual reality walk down a garden path at Taliesin.

```
<?xml version="1.0"?>
  <sphere vert="glsl/sph-zoomer.vert"
          frag="glsl/sph-blend.frag">
    <frame>
      <scm channel="0" file="Taliesin-Path-A-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-A-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-B-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-B-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-C-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-C-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-D-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-D-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-E-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-E-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-F-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-F-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-G-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-G-R-512-3.tif" />
    </frame>
    <frame>
      <scm channel="0" file="Taliesin-Path-H-L-512-3.tif" />
      <scm channel="1" file="Taliesin-Path-H-R-512-3.tif" />
    </frame>
  </sphere>
```

Displacement-mapped Illuminated Planet

The following example brings together multiple disparate data sets to produce a displacement-mapped sphere with real-time illumination and diffuse color, suitable for viewing with `orbiter`. It demonstrates a number of additional attributes.

```
<?xml version="1.0"?>
  <sphere mesh="128" r0="0.994728" r1="1.0062" radius="20.0"
            vert="glsl/scm-displace.vert"
            frag="glsl/scm-lomsee.frag">
    <scm shader="vert" file="DTM-426-6.tif" />
    <scm shader="frag" file="DTM-426-6-N.tif" />
    <scm shader="frag" file="clementine-512-3.tif" />
  </sphere>
```

The `mesh` resolution has been increased to 128 from its default 16. This is because a height map SCM is applied to the sphere geometry by the `scm-displace.vert` vertex shader. The geometry must have a dense tessellation to allow a close mapping from height map samples to 3D vertices. Related to this, the `r0` and `r1` attributes give the magnitude of the displacement applied to that geometry, relative to a radius of 1. These are the same values used when generating the normal map in Section 2.3. `r0` = 1,728,240/1,737,300 and `r1` = 1,748,170/1,737,400.

The `scm` elements have an additional attribute, `shader`, which allows an SCM to be targeted toward a specific shader. In this case, the `scm-displace.vert` shader is granted access to the height map. The default shader target for `scm` elements is "frag," which is given explicitly here for symmetry. This rendering uses the `scm-lomsee.frag` fragment shader which requires *two* images, separately applied to the per-fragment normal and diffuse color coefficients of the Lommel-Seeliger lighting model. Order matters here, and this shader expects the normal map to be listed first.

It's worth noting that not all of the data sets used in this visualization have the same page size and depth. Equality in not required, and the adaptive renderer will defer to the lowest page size to ensure that an appropriate level of detail is achieved for all data. Likewise, the deepest SCM depth *will* be fully utilized, and shallower SCMs will undergo linear magnification filtering to match.

4.2 PANOVIEW USAGE

PANOVIEW and ORBITER are configured and run like any other Thumb application. The following keyboard commands are defined.

F1 Toggle the sphere definition file selection dialog. This dialog allows the user to navigate the Thumb data hierarchy and select a visualization definition for viewing.

F2 Toggle the cache visualization overlay. This allows the set of all resident pages to be viewed in thumbnail.

F3 Toggle the model label, if any.

F4 Toggle the wire-frame view of the sphere geometry.

In addition, there is one option in the Thumb configuration file `conf.xml` that impacts the behavior and performance of these applications:

```
<option name="scm_cache_size">128</option>
```

This option gives the value passed to the `scm_cache` constructor, selecting the maximum number of pages that may be loaded at any given moment. A 512×512 page of RGB data consumes 1 MB of VRAM and this cache size option should be set accordingly. A larger value gives better performance, but too large a value will result in catastrophically bad performance. Knowledge of your hardare's available VRAM will help to determine the optimal value.

4.3 TROUBLESHOOTING

This is a list of issues to be aware of, should trouble arise when configuring or using PANOVIEW and ORBITER.

- If the panorama definition XML files are not visible in the sphere definition file selector, then be sure they are located within the Thumb data hierarchy, or add their location to the Thumb data hierarchy by including the path in the `THUMB_RO_PATH` environment variable.
- If the panorama definition loads but does not display an image, be sure the path to the SCM TIFF files appears in the `SCMPATH` environment variable.
- If a multi-image panorama jumps from one panorama to the next instead of fading, be sure the blend fragment program is referenced by the definition.

- If performance is sluggish, be sure that panorama image files are not being accessed from a network share, and that the `conf.xml` setting of `scm_cache_size` is not too high.

The following C Language function computes the 64-bit integer binary logarithm in $O(\log n)$ operations. It takes and returns a signed value, instead of unsigned, as this is the most appropriate C type for SCM page indices.

```c
static inline long long log2i(long long n)
{
    unsigned long long v = (unsigned long long) n;
    unsigned long long r;
    unsigned long long s;

    r = (v > 0xFFFFFFFFULL) << 5; v >>= r;
    s = (v > 0xFFFFULL    ) << 4; v >>= s; r |= s;
    s = (v > 0xFFULL      ) << 3; v >>= s; r |= s;
    s = (v > 0xFULL       ) << 2; v >>= s; r |= s;
    s = (v > 0x3ULL       ) << 1; v >>= s; r |= s;

    return (long long) (r | (v >> 1));
}
```

LRO DTM MAKEFILE

```
N=426
D=6

NAME = DTM-$(N)-$(D)
FORM = -n $(N) -d $(D)
NORM = -N -9160,10770 -g 0
RADI = -R 1728240,1748170
TEXT = -t desc.txt

GLDE =\
        WAC_GLD100_E300N0450_100M.tif \
        WAC_GLD100_E300N1350_100M.tif \
        WAC_GLD100_E300N2250_100M.tif \
        WAC_GLD100_E300N3150_100M.tif \
        WAC_GLD100_E300S0450_100M.tif \
        WAC_GLD100_E300S1350_100M.tif \
        WAC_GLD100_E300S2250_100M.tif \
        WAC_GLD100_E300S3150_100M.tif
GLDP =\
        WAC_GLD100_P900N0000_100M.tif \
        WAC_GLD100_P900S0000_100M.tif
LOLA =\
        ldem_45n_100m.tif \
        ldem_45s_100m.tif

# Convert a raw normal map to a bordered and finished normal map.

$(NAME)-N.tif: $(NAME)-O.tif
        scmtiff -T -p border -o $@ $<
        scmtiff -T -p finish $(TEXT) $@

# Convert a height map into a raw normal map.

$(NAME)-O.tif: $(NAME).tif
        scmtiff -T -p normal $(RADI) -o $@ $<

# Border and finish.

$(NAME).tif: $(NAME)-M.tif
        scmtiff -T -p border -o $@ $<
        scmtiff -T -p finish $(TEXT) $@

# Combine the two projections and mipmap the result.
```

```

$(NAME)-M.tif: $(NAME)-E.tif $(NAME)-P.tif
	scmtiff -T -p combine -m avg -o $@ $^
	scmtiff -T -p mipmap $@

# Sum the equirectangular projections.

$(NAME)-E.tif: $(GLDE)
	scmtiff -T -p combine -m max -o $@ $^

# Sum the polar projections.

$(NAME)-P.tif: $(GLDP) $(LOLA)
	scmtiff -T -p combine -o $@ $^

# Convert all PDS files to TIFF.

WAC_GLD100_E%.tif: WAC_GLD100_E%.IMG
	scmtiff -T -p convert $(FORM) $(NORM) -o $@ $<

WAC_GLD100_P900N0000_100M.tif: WAC_GLD100_P900N0000_100M.IMG
	scmtiff -T -p convert $(FORM) $(NORM) -o $@ -P  90,12,11 $<

WAC_GLD100_P900S0000_100M.tif: WAC_GLD100_P900S0000_100M.IMG
	scmtiff -T -p convert $(FORM) $(NORM) -o $@ -P -90,12,11 $<

ldem_45n_100m.tif: ldem_45n_100m.lbl
	scmtiff -T -p convert $(FORM) $(NORM) -o $@ -P  90,11,12 $<

ldem_45s_100m.tif: ldem_45s_100m.lbl
	scmtiff -T -p convert $(FORM) $(NORM) -o $@ -P -90,11,12 $<

clean:
	rm -f $(NAME).tif $(GLDE) $(GLDP) $(LOLA)
```

NOTES

CHAPTER 1 SPHERICAL CUBE MAP

1. In an unexpected result, as the resolution of an SCM page tends toward infinity, the ratio of the solid angle of a center pixel (the largest pixel) to the solid angle of an edge pixel (the smallest pixel) tends toward exactly $\sqrt{2}/2$.
2. Compounding the surprise, the ratio of the length of the shortest edge to the length of the longest edge also tends toward $\sqrt{2}/2$.

CHAPTER 2 SCM PRE-PROCESSING

1. Yes, this process is dumb enough to overwrite a source data file that already has the `.tif` extension.
2. The replicated borders provide only a single pixel of context. This enables linear magnification filtering, but does *not* enable anisotropic filtering.
3. `http://wms.lroc.asu.edu/lroc/global_product/100_mpp_DEM`
4. `http://pds-geosciences.wustl.edu/missions/lro/lola.htm`
5. The real limit is the total depth of all simultaneously loaded data sets, which must be less than GL_MAX_TEXTURE_IMAGE_UNITS. On a 2012-era Mac, this limit is 16. On a 2012-era NVIDIA-equipped Linux or Windows PC, this limit is 56 or 64.