Working with Lunar Reconnaissance Orbiter LROC Narrow Angle Camera (NAC) Data

Introduction to the LROC NAC

The Lunar Reconnaissance Orbiter Camera (LROC) Narrow Angle Cameras (NACs) are a pair of high-resolution pushbroom cameras on the Lunar Reconnaissance Orbiter (LRO). The two cameras (NAC-Left and NAC-Right) sweep out parallel and slightly overlapping ground tracks, and acquire images simultaneously. Typical LROC NAC images range in pixel scale from 0.5–2.0 m/px, and a typical NAC pair is ~10,000 pixels wide (~east-west) by ~52,000 pixels long (~north-south), for normal footprint sizes ranging from ~5x26 km to ~20x100 km.

From September 2009 through December 2011, the LRO spacecraft was in a 50 km altitude circular polar orbit, so most LROC NAC images from that time have a pixel scale of ~0.5 m/px. On 11 December 2011, LRO was moved to a more stable elliptical polar orbit to save fuel (similar to the Commissioning Phase orbit LRO was in prior to 15 September 2009), and images from this elliptical orbit have varying resolutions. The periapsis of this orbit is near the south pole, so images of the northern hemisphere have larger pixel scales (~2.0-1.0 m/px, depending on latitude and year), while images of the southern hemisphere have smaller pixel scales (~1.0-0.4 m/px). The orbit is evolving towards circularity, so this resolution dichotomy gets less pronounced from 2011 to ~2022. On 12 March 2018 the spacecraft's inertial measurement unit was turned off to preserve it for emergencies, significantly reducing the positional accuracy of subsequent images (see Wagner et al. 2024 more details).

To improve signal-to-noise ratio under low-Sun conditions, the NAC can acquire images in “summed” mode, where adjacent pixels have their values combined during readout from the sensor. This reduces the image width from 5064 px to 2532 px, and doubles the pixel scale. Summed images are the norm for polar observations.

Because LROC is a pushbroom camera, the length and width of a pixel on the ground may not match each other. Images are frequently targeted with an exposure time that makes the two dimensions roughly match, making the raw image appear “normal” (albeit frequenty mirrored vertically and/or horizontally due to spacecraft and orbit geometry), but you may run into images with significant down-track stretching or squishing (up-down direction in the raw image). There is generally no additional processing needed to handle this, though long-exposure images of shadowed areas, with pixels several times longer than they are wide, can benefit from shrinking the image in the cross-track direction to a more square pixel aspect ratio before map-projection.

Image IDs: NAC image IDs look like “M1234567890L”, where “M” is the target (M=Moon, E=Earth, C=star Calibration), “1234567890” is a nine- or ten-digit number related to the mission elapsed time when the image was acquired (an extra digit was added in June 2012 to accommodate the extended duration of the mission. Keep in mind that the first digit is always “1” (the spacecraft clock partition ID) regardless of how many digits the ID has, but otherwise the number increases over time), and “L” (or “R”) indicates whether the image is from the NAC-Left or NAC-Right. PDS products append an “E” or “C” to the end of the image ID, depending on whether the product is an Experimental Data Record (EDR) or Calibrated Data Record (CDR).

The Principle Investigator for LROC is Mark Robinson at Intutive Machines (formerly at Arizona State University). For more information on the LROC NACs, see the papers for instrument overview, radiometric calibration, geometric calibration, and cartographic accuracy.

Locating LROC NAC Data

There are a number of ways to find LROC NAC data. The official PDS archive search is at https://data.im-ldi.com/mds.html, which allows searching by a number of parameters. However, a more convenient way to locate LROC data of a specific feature may be to use QuickMap. You can use the Draw Point or Draw Polygon tool to select an area and search for LROC NACs that include that area, and filter the results by lighting and viewing geometry. You can also load results onto the map view to preview images. The results list has links to the PDS archive where you can download the data. See the QuickMap User Guide for more details.

Regardless of how you get to the PDS archive page, you will need to download the EDR file. Do not download CDR files if you intend to do any further processing with them, such as map-projection, as they will be more complicated to use than EDRs, are twice the size, and may not use the most recent calibration parameters. You do not need to download PDS4 label files to use LROC data.

Note: Almost all LROC NAC observations come in pairs (one image each from the left and right cameras) that you must download from separate pages. The PDS archive page should have a link to the matching observation of a pair if it exists.

Websites where you can find LROC data:

- LROC PDS Archive Search
- LROC PDS Archive
- Image Atlas, PDS Imaging Node Image Atlas
- ACT-REACT QuickMap for LROC data
- PDS Geosciences Node Lunar Orbital Data Explorer

Processing LROC NAC Data

The standard LROC NAC processing pipeline uses the United States Geologic Survey's Integrated System for Imagers and Spectrometers (ISIS) software, preferably version 7.2.0 or later. If you do not have ISIS installed on your computer, see this page for installation instructions. ISIS is only supported on MacOS and Linux computers or virtual machines, but there have been reports of successful installation on the Linux Subsystem for Windows.

Basic processing summary

1. Download EDR files from your favorite source (see above).
2. `lronac2isis from=M1234567890LE.IMG to=M1234567890L.raw.cub`
3. `spiceinit from=M1234567890L.raw.cub spksmithed=true web=true`
4. `lronaccal from=M1234567890L.raw.cub to=M1234567890L.cal.cub`
5. `lronacecho from=M1234567890L.cal.cub to=M1234567890L.echo.cub`
6. Optionally use `lronacpho` or `photomet` to do a photometric correction (see below for details).

At this point you have a fully-calibrated Level 1 cube, equivalent to a CDR from the PDS, but with embedded position information. Repeat steps 1-6 for all images you want to combine in a single mosaic, then put the paths to those images in a text file "mosaic_level_1.lis".

7. `mosrange fromlist=mosaic_level_1.lis to=mosaic.map projection=equirectangular` (or use `maptemplate`)
8. Optionally use `reduce` to downsample the images (see below for details).
9. `cam2map from=M1234567890L.echo.cub to=M1234567890L.map.cub map=mosaic.map pixres=map \`
   `warpalgorithm=forwardpatch patchsize=50`

Repeat steps 8 and 9 for each image, then put the paths to those images in a text file "mosaic_level_2.lis". You now have a set of map-projected images with matching scales and projections, which can be used alone or mosaicked together:

10. Optionally co-register the images (see below for details).
11. `automos from=mosaic_level_2.lis mos=mosaic.cub`

You can find detailed explanations of each step below, along with additional options and tweaks that may improve your results.

1) Convert EDR to ISIS cube

There is a specific program for converting LROC NAC EDR images to ISIS cubes, `lronac2isis`, which preserves all the keywords needed to properly calibrate the image:

```
lronac2isis from=M1234567890LE.IMG to=M1234567890L.raw.cub
```

2) Attach camera position information

ISIS uses the Navigation and Ancillary Information Facility SPICE system to determine the camera position during the observation, so that the image can be correctly projected onto the surface. This is set up using the `spiceinit` program. Basic operation is simple:

```
spiceinit from=M1234567890L.raw.cub spksmithed=true
```

Note: If you did not choose to download SPICE kernels as part of your ISIS installation, also add `web=true` to get SPICE information from USGS servers. If you are using ISIS 8.0.2 or older, you will also need to set the server URL to connect to the current USGS SPICE server; as of June 2024, add: `url=https://astrogeology.usgs.gov/apis/ale/v0.9.1/spiceserver/`

The `spksmithed=true` part of the basic `spiceinit` call causes ISIS to use the LOLA/GRAIL "smithed" SPKs when available, which improve the position accuracy of LRO by a factor of 5-10 over the default SPKs (add `spkrecon=false` to the command to cause an error if smithed SPKs are not available; smithed SPKs are usually produced with a several-month lag, and might not be available for recent images).

There are also a few more things you can do to further improve the final product:

Use a better shape model

The ISIS default lunar shape model is a 237 m/px (128 px/degree) global LOLA digital elevation model (DEM or DTM), which contains significant interpolated regions near the equator. For an easy-to-use higher-quality DEM, you can download a 100 m/px version of the GLD100+LOLA global DEM from the LROC Popular Downloads page ("WAC GLD 100 Topography"). You can specify this model in `spiceinit` by adding `shape=user model=/path/to/GLD100_file.cub` to the command.

Note: On ISIS versions prior to 4.4.0, you cannot specify a custom shapemodel when using web SPICE.

For images close to a pole (particularly those with large slew angles), it might be better to use LOLA polar DEMs directly ("LDEM" files archived on the LOLA PDS website, or, for significantly higher-quality models of the south pole, this Goddard website). Unlike the GLD100 cube file above, these will need to be converted using `pds2isis`, rescaled to have radius values in meters using `fx`, and then will need `demprep` run on them:

```
pds2isis from=LDEM_80S_20M_FLOAT.LBL to=LDEM_80S_20M_FLOAT.cub

fx f1=LDEM_80S_20M_FLOAT.cub equation="f1*1000" to=LDEM_80S_20M_FLOAT.meters.cub

demprep from=LDEM_80S_20M_FLOAT.meters.cub to=LDEM_80S_20M_FLOAT.demprep.cub
```

Similarly, the SLDEM (download links) is a much higher-quality terrain model (60 m/px resolution) for areas within 60° of the equator. The following code can create a unified SLDEM mosaic from the full-resolution tiles ("SLDEM2015_512"), though be warned that the resulting file is 43 GB:

```
pds2isis from=sldem2015_512_00n_30n_000_045_float.lbl to=SLDEM2015_512_00N_30N_000_045.cub

# Repeat the above command for each tile, and list all the cube paths in sldem.lis

automos fromlist=sldem.lis mosaic=SLDEM_km.cub matchbandbin=false

fx f1=SLDEM_km.cub to=SLDEM_m_rad.cub equation="f1*1000"

demprep from=SLDEM_m_rad.cub to=SLDEM.demprep.cub
```

It can be useful to make a truly global shapemodel using the SLDEM and 60 m/px LOLA polar LDEM tiles (although for polar images, it is better to use shapemodels in polar sterographic projection); to do this, process the SLDEM and LDEM tiles as described above (optionally skipping the `demprep` step), then use `map2map` to reproject the LDEM files into the same equirectangular projection as the SLDEM, and use `automos` to stack all three DEMs into one file (making sure to layer the SLDEM on top of the LDEMs). Finally, run `demprep` on the final 64 GB mosaic.

In small areas, you can use NAC DEMs to get very accurate projection (this is mostly only relevant for images with > ~5° slew angles). Using NAC DEMs as shapemodels requires adding the 1,737,400 m lunar radius to the DEM values, and running `demprep`. Also, keep in mind that for NAC DEMs, it's critical to correct the image's pointing to align with the DEM, for example by using `deltack`:

```
# Set up DEM

pds2isis from=NAC_DTM_APOLLO11_E008N0234.IMG to=NAC_DTM_APOLLO11_E008N0234.cub

fx f1=NAC_DTM_APOLLO11_E008N0234.cub equation="f1+1737400" \
to=NAC_DTM_APOLLO11_E008N0234.radius.cub

demprep from= NAC_DTM_APOLLO11_E008N0234.radius.cub to= NAC_DTM_APOLLO11_E008N0234.demprep.cub

# Process image.  This image is being aligned to the Apollo 11 retroreflector's coordinates
# and elevation as measured from the DTM and corresponding high-resolution orthophoto, which
# may differ slightly from its true position

spiceinit from=M175124932R.raw.cub shape=user model=NAC_DTM_APOLLO11_E008N0234.demprep.cub \
spksmithed=true

deltack from=M175124932R.raw.cub samp1=2505 line1=23805 lat1=0.673433 lon1=23.473113 \
rad1=1735471.9565
```

When using a NAC DEM as the radius source in spiceinit, keep in mind that locations outside the DEM's area will not have defined geometry, so applications like `campt` will not work outside the DEM bounds, `mosrange` will fail for the image, and when map-projecting the image, only those parts that overlap the DEM will be processed.

Note that simply adding the lunar radius to the DEM values with `fx` causes some quantization of the DEM due to limitations of the 32-bit floating-point format. The precision of a given pixel's elevation value will be reduced to ~10-20 cm. This is unlikely to significantly impact projected image quality, but can be partially avoided by converting the DEM to 16-bit format and taking advantage of ISIS's [Base and Multiplier parameters]. The exact parameters to do this, and resulting level of quantization, will vary depending on the minimum and maximum elevations in the DEM, but for the Moon this is an improvement over the `fx` method for DEMs with less than 8,000 m of total relief (all NAC DEMs are expected to fall in this range). Some general code for doing this in Bash follows:

```
# Import and convert to 16-bit

pdsfile=NAC_DTM_APOLLO11_E008N0234.IMG

pdsname=$(basename "$pdsfile" .IMG)

pds2isis from="$pdsfile" to="$pdsname.cub"

read min max <<<"$(stats from="$pdsname.cub" | awk '/ Minimum /{min=$3} / Maximum /{max=$3}
END{print min,max}')"

cubeatt from="$pdsname.cub" to="$pdsname.16bit.cub"+unsignedword+$min:$max

# Add lunar radius by editing the label

base=$(getkey from="$pdsname.16bit.cub" grpname=Pixels keyword=Base)

base=$(bc <<<"$base + 1737400")

editlab from="$pdsname.16bit.cub" grpname=Pixels keyword=Base value=$base

# Finally, run demprep

demprep from="$pdsname.16bit.cub" to="$pdsname.demprep.cub"
```

Light time correction

Technically, ISIS does not quite handle light-time delay to the surface quite correctly, to keep processing times reasonable. For LROC, it assumes there is no delay between light leaving the surface and hitting the detector, which causes a ~0.5-1m downtrack

error for LROC images of the Moon. If you care about this (and are not using web SPICE), you can make a copy of `$ISISDATA/lro/kernels/iak/lro_instrumentAddendum_v04.ti` for each image you process, changing the 0.0 in the following lines to the negative of that image's light time to the surface in seconds:

```
INS-85600_CONSTANT_TIME_OFFSET = ( 0.0 )

INS-85610_CONSTANT_TIME_OFFSET = ( 0.0 )
```

You will then need to specify this new IAK in the `spiceinit` call using `iak=/path/to/custom_iak.ti`. You can calculate the light time to surface by running `spiceinit` on the file once, using `campt` to get the slant distance for the center of the image (distance to surface in km), and dividing that by the speed of light (299792.458 km/s). For example, with a slant distance of 100 km, the light time delay would be 100 / 299792.458 = 0.0003335641 seconds, so you would set the `CONSTANT_TIME_OFFSET`s to `-0.0003335641`.

To properly process one of the handful of LROC NAC images of Earth, you will instead want to turn light-time correction on, by changing the keywords in the following two lines in the copied IAK file from `'NONE'` to `'LT+S'`:

```
INS-85600_LIGHTTIME_CORRECTION = 'NONE'

INS-85610_LIGHTTIME_CORRECTION = 'NONE'
```

You will also need to change the NAC image header to indicate that Earth is the target before running `spiceinit`:

```
editlab from=E1234567890L.raw.cub grpn=Instrument keyw=TargetName Value=Earth
```

DE 440 lunar ephemerides

If you want to use absolutely the most accurate metadata possible, you can use the DE 440 lunar ephemeris instead of the default DE 421. This modification is *not* used by the LROC team, and only improves pointing on a sub-pixel level. Using the DE 440 ephemeris improves spacecraft pointing on average by a few centimeters, and we are including this section mostly out of completeness. To use these kernels, you first need to download them from the NAIF archive, and place them in some central location:

```
curl https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/de440.bsp > de440.bsp

curl https://naif.jpl.nasa.gov/pub/naif/generic_kernels/fk/satellites/moon_de440_220930.tf \
> moon_de440_220930.tf

curl https://naif.jpl.nasa.gov/pub/naif/generic_kernels/pck/moon_pa_de440_200625.bpc \
> moon_pa_de440_200625.bpc
```

You can then specify those files in the `spiceinit` call in place of the DE 421 files they are replacing, along with a few unchanged files that are in the same kernel groups, using ISIS's array syntax for label keywords (note that the use of single-quotes in this command is crucial- the `$lro` and `$base` variables are internal to ISIS, and will not work properly if your command-line shell sees them as variables that it should substitute):

```
spiceinit from=M1234567890L.raw.cub spksmithed=true tspk='(/path/to/moon_pa_de440_200625.bpc,
/path/to/de440.bsp)' pck='($base/kernels/pck/pck00009.tpc, /path/to/moon_de440_220930.tf,
$lro/kernels/pck/moon_assoc_me.tf)' iak=/path/to/custom_iak.ti
```

4) Radiometric calibration

This step converts the pixel values from DN in the EDR into I/F (irradiance) units using `lronaccal`. Note that if you have not downloaded the LRO kernels, you must have run `spiceinit` on the cube to calculate I/F, as the calculation involves the distance to the Sun; without `spiceinit` or downloaded kernels, you can only successfully calibrate to radiance, not I/F.

```
lronaccal from=M1234567890L.raw.cub to=M1234567890L.cal.cub
```

If radiance values are needed instead, the `RadiometricType` keyword will let you calibrate to radiance (in units of W/m^2/sr/µm):

```
lronaccal from=M1234567890L.raw.cub to=M1234567890L.cal.cub RadiometricType=radiance
```

If you are using a version of ISIS older than 7.1.0, you may notice vertical-line artifacts in recent NAC images. This is due to those versions not using time-dependent dark corrections, which correct for uneven degradation of the sensors over time.

5) Echo-correction

Due to the way the LROC NAC detectors were set up to be able to read out pixels fast enough to image the Moon's surface at 0.5 m pixel scale, there is an artifact in moderate and high signal pixels where ~30% of each pixel's brightness is echoed two pixels later along the detector. The **lronacecho** program fixes that artifact, although for some images, particularly those with a high Sun elevation, this may result in a significantly noisier-looking image:

```
lronacecho from=M1234567890L.cal.cub to=M1234567890L.echo.cub
```

This process introduces its own minor artifact on one edge of the image, which, along with a very minor edge-of-image darkening that `lronaccal` does not remove, can be trimmed off with the **trim** command. The ideal amounts to trim differ depending on whether you are processing a NAC-Left or -Right image:

```
# NAC-L
trim from=M1234567890L.echo.cub to=M1234567890L.tr.cub left=46 right=26
```

```
# NAC-R
trim from=M1234567890R.echo.cub to=M1234567890R.tr.cub left=26 right=46
```

If the image is summed (in the EDR label, look for "`CROSSTRACK_SUMMING = 2`", or check if the image width is 2532 pixels), divide those numbers by two: the trim amounts should be 23 and 13.

6) (Optional) Photometric correction

When mosaicking images taken under non-matching lighting conditions, it can be important to correct the images' brightnesses to a common set of photometric angles. This is particularly useful for Featured Mosaic image sequences, where LRO slews on several successive orbits to build up a continuous swath of NAC images with nearly-identical incidence, but varying emission angles.

The lronacpho application (new in ISIS 7.1) is usually the recommended method of photometrically normalizing LROC NAC images to remove the effects of incidence and emission angle changes, if the image has a phase angle (angle between vectors to the Sun and camera) between 15° and 65°:

```
lronacpho from=M1234567890L.tr.cub to=M1234567890L.pho.cub phopar=nacpho.pvl
```

Where nacpho.pvl contains the following:

```
Object = NormalizationModel
  Group = Algorithm
    Name = LROC_Empirical
    PhotoModel = LROC_Empirical
    Incref=30.0
    Emaref=0.0
    Pharef=30.0
  EndGroup
EndObject

Object = PhotometricModel
  Units = Degrees
  Group = Algorithm
    Name = LROC_Empirical
```

```
    FilterName = “Broadband”
    BandBinCenter = 600.0
    A0 = -2.9811422
    A1 = -0.0112862
    A2 = -0.8084603
    A3 =  1.3248888
  EndGroup
EndObject
```

For images with large incidence angles (>60°), the LROC team has found that **photomet** produces better results (in keeping with the phase angle recommendations for `lronacpho`). Additionally, photomet was used for many older controlled mosaics and the uncontrolled polar mosaics, which predate the development of `lronacpho` and its photometric model. Use photomet as follows:

```
photomet from=M1234567890L.tr.cub to=M1234567890L.pho.cub frompvl=basicpho.pvl
```

Where basicpho.pvl contains the following:

```
Object = NormalizationModel
  Group = Algorithm
    Name = Mixed
    Incmat = 87.0
    Albedo = 0.05
    Incref = 0.0
    Thresh = 10E30
  EndGroup
EndObject

Object = PhotometricModel
  Group = Algorithm
    Name = HapkeHen
    Wh = 0.278906221
    Hh = 0.0438
    B0 = 1.0
    Theta = 10.0
    HG1 = 0.229315607
    HG2 = 0.0
  EndGroup
EndObject
```

Map-projection and mosaicking

7) Create a map file

The LROC team typically uses the equirectangular projection type, with center latitude and longitude near the center of the mosaic region (our PDS products typically use the center of the region rounded to 0.1 degree). For small polar products above 70 degrees latitude, we use a polar stereographic projection with the same center latitude/longitude conventions. For products covering an entire pole, or that are intended to be used in conjunction with such products, we use polar stereographic with a center latitude of 90 or -90 (depending on the pole), and a center longitude of 0.

There are two good ways to create a map file for a project. If you know the exact bounds or centerpoint of the area you’re interested in, use **`maptemplate`**. If you just want the best settings to mosaic a set of NACs together (especially for just processing a single pair), use **`mosrange`**.

Note: If you want to match an existing map-projected cube, you can specify that cube as the map file, rather than creating a separate map file. If you do this, you can use the `matchmap=true` argument to make the new cube have identical pixel bounds to the map.

maptemplate

`maptemplate` is the standard program for making a map definition file for ISIS. If you are making multiple products of a specific area, this is the program to use. Unless you are writing a script, it's usually easiest to just run `maptemplate` with no options, and use the graphical interface to select the options you want.

mosrange

If you are just map-projecting one NAC pair or a small mosaic, and don't care about aligning/mosaicking with other products, it can be more convenient to use `mosrange`, which automatically generates a map projection definition with a suitable center, resolution, and bounds for a set of images. First, list all of the calibrated cubes in a text file (called `images.lis` in the below example), then run `mosrange`:

```
mosrange fromlist=images.lis projection=equirectangular to=mosaic.map
```

Considerations for use with GIS software

All published LRO products use longitude values 0-360. However, ESRI programs (such as ArcMap) work better with longitude values in the range -180 to +180. If you intend to use your map-projected products in Arc, you should add `londom=180` to the command you use to generate a map file (both `maptemplate` and `mosrange` accept this argument; in the `maptemplate` graphical interface, the option is called "LONDOM: -180 to 180 degree longitude values" under "Target Parameters").

8) (Optional) Reduction

If you will be downsampling your images significantly while map-projecting them (that is, the map-projected pixels will be more than two times larger in X or Y than the raw images' pixels), it is beneficial to use **`reduce`** to downsample the unprojected image to a scale closer to the final product before map projection. This will both decrease processing time and increase image quality, as without the reduction step the map-projection algorithm will not properly average all the pixels that should go into each output pixel, and the final image will appear much noisier than it should. For the absolute highest quality of downsampled image, you should map-project it at full scale and then downsample with `reduce`, but using `reduce` first is much faster, and only slightly noisier.

The optimal reduction factor is the largest integer factor that will leave the unprojected image higher-resolution than the output map. Because NACs may have non-square pixels, the scaling factor may be different in X and Y. For each dimension, the scaling factor should be `ceiling( output_resolution / (input_resolution*2) )`:

```
s_factor=$(bc <<<"$outputRes / ( $crosstrack * 2 ) + 1")

l_factor=$(bc <<<"$outputRes / ( $downtrack * 2 ) + 1")

reduce from=M1234567890L.tr.cub to=M1234567890L.tr.reduced.cub sscale=$s_factor lscale=$l_factor
```

Unfortunately, the ISIS **`campt`** program does not report accurate values for the line-direction pixels scales for raw NAC images. A better source for the line-direction pixel resolutions is the INDEX.TAB or CUMINDEX.TAB files with metadata for all LROC images, included in the INDEX subfolder of each **LROC PDS Archive** release. However, it can be inconvenient to deal with this 2+ GB text file, and these values are calculated with a spherical shape model and also do not correctly account for off-nadir viewing angles. To get the best measure of the input pixel size, use multiple `campt` calls to measure the surface position at multiple points, and calculate the pixel scales from that (this does not account for changes in scale from topography within the image, but will still be a marked improvement from the other methods). The following bash function will print out the sample-direction, line-direction, and average pixel scale at the center of the image using that method, assuming the center of the image is covered by the shape model (this function takes ~5 seconds/image):

```
function get_image_resolution() {
  local _im="$1"
  local _line=$(bc <<<"$(getkey from="$_im" grpn=Dimensions keyw=Lines) / 2")
  local _sample=$(bc <<<"$(getkey from="$_im" grpn=Dimensions keyw=Samples) / 2")
  local _tmp=$(mktemp)

  campt from="$_im" line="$_line" sample="$_sample" to="$_camptfile" append=false &>/dev/null
  local _xyz1=$(getkey from="$_tmp" grpn=GroundPoint keyw=BodyFixedCoordinate | sed 's/,/ /g')
```

```
  campt from="$_im" line="$((_line + 1))" sample="$_sample" to="$_tmp" append=false >/dev/null
  local _xyz_dt=$(getkey from="$_tmp" grpn=GroundPoint keyw=BodyFixedCoordinate | sed 's/,/ /g')

  campt from="$_im" line="$_line" sample="$((_sample + 1))" to="$_tmp" append=false >/dev/null
  local _xyz_ct=$(getkey from="$_tmp" grpn=GroundPoint keyw=BodyFixedCoordinate | sed 's/,/ /g')
  rm -f "$_tmp"

  local _downtrack=$(echo "$_xyz1 $_xyz_dt" | awk '{print sqrt( ($1-$4)^2 + ($2-$5)^2 + ($3-
$6)^2 ) * 1000}')
  local _crosstrack=$(echo "$_xyz1 $_xyz_ct" | awk '{print sqrt( ($1-$4)^2 + ($2-$5)^2 + ($3-
$6)^2 ) * 1000}')

  local _average=$(bc -l <<<"($_crosstrack + $_downtrack)/2")

  echo $_crosstrack $_downtrack $_average
}
read crosstrack downtrack average <<<"$(get_image_resolution M1234567890L.tr.cub)"
```

9) Map-projection

Use **cam2map**:

```
cam2map from=M1234567890L.pho.cub to=M1234567890L.map.cub map=mosaic.map \
warpalgorithm=forwardpatch patchsize=50
```

To make this process take a reasonable amount of time, you need to specify the patch size parameter, otherwise ISIS will do a slow radius calculation for every pixel. The patch size should be the larger of (DEM pixel scale / raw image pixel scale) or (output image pixel scale / raw image pixel scale). Usually, it will be the first one. Example: Assume that you are projecting an image with a native pixel scale of 0.5 m/px, using the 100 m/px GLD100 as the DEM, and creating a file with a resolution of 1 m/px. Patch size should be the larger of: `100 / 0.5 = 200` (DEM pixel scale / raw image pixel scale) or `1 / 0.5 = 2` (Output pixel scale / raw image pixel scale) Thus, you would add the following arguments to the `cam2map` command:

```
 warpalgorithm=forwardpatch patchsize=200
```

Keep in mind that this formula may not always produce good results for oblique images (emission/slew angle >45 degrees). If the resulting image has unexpected null patches (or you just want to be conservative), reduce the patchsize parameter by at least a factor of four. This will increase processing time, however.

If you don't want to do the detailed calculation for each image, 50 is generally a safe value for images that are non-oblique enough to be useful after map-projection, if using a global terrain model, but will likely be slower than the ideal value.

10) (optional): Coregistration

Usually, there will be little if any visible seam between the left and right NACs of a pair. However, in some cases the temperature-based model of the varying offset between the two does not fully correct the alignment. In these cases, it's appropriate to use **coreg** to align the two images, and then rmosaic them. You can check the alignment before mosaicking by loading the two images in **qview**, linking them, using the 'Find' tool to mark the same lat/lon coordinates in both images, and looking for offsets.

This takes a few steps. First, you need to get a subsection of one of the images, with exactly the same map bounds and dimensions as the other image, and then run `coreg` on those two images (see the end of the section for `nac_pair_align.def`):

```
map2map from=M1234567890R.map.cub to=M1234567890R.Lbounds.cub map=M1234567890L.map.cub \
matchmap=true

coreg from=M1234567890R.Lbounds.cub match=M1234567890L.cub deffile=nac_pair_align.def
```

In the output of `coreg`, you will find `SampleAverage` and `LineAverage` values. You will need these to offset the image that you are moving using **`translate`**:

```
translate from=M1234567890R.map.cub to=M1234567890R.map.shifted.cub strans=$sampleaverage \
ltrans=$lineaverage
```

You should now be able to seamlessly mosaic M1234567890L.map.cub and M1234567890R.map.shifted.cub together.

Note: Just running `translate` will likely cut a few pixels off of the edge of the image being moved. If you care about this, you can use the pad program to add some space (equal to the line and sample averages) to the appropriate sides of the image before running `translate`. Alternatively, instead of using `translate`, you can use editlab to change to UpperLeftCornerX and UpperLeftCornerY keys in the Mapping group to shift the image by a given number of meters without editing any pixels, though this does require converting the offsets from pixels to meters.

There is another tool for determining offsets between images, `findfeatures`, which uses computer vision algorithms to find matching points between two images even if the areas do not fully overlap or have arbitrary offsets. However, this application is best used on areas much smaller than a full NAC image to keep processing time and memory usage reasonable, and extracting suitable x/y offset values from its output is outside the scope of this document. For NAC left/right pairs, `coreg` is the better application to use to measure offsets. For most NAC pairs, the following `nac_pair_align.def` file is appropriate:

```
Object = AutoRegistration
  Group = Algorithm
    Name             = MaximumCorrelation
    Tolerance        = 0.7
    SubpixelAccuracy = True
  End_Group

  Group = PatternChip
    Samples = 20
    Lines   = 20
  End_Group

  # If you determine that the vertical or horizontal offset is >50 pixels, adjust the SearchChip
  # size to be at least `20+offset*2`, where 20 is the PatternChip size.
  Group = SearchChip
    Samples = 120
    Lines   = 120
  End_Group

  Group = SurfaceModel
    DistanceTolerance = 1.5
    WindowSize        = 5
  End_Group
End_Object
End
```

11) Mosaicking

Once you have map-projected and optionally aligned the individual left/right images of a pair, you may want to mosaic them together into a single image file. The easiest way to do this is using automos:

```
ls M1234567890L.map.cub M1234567890R.map.cub > M1234567890.lis

automos fromlist=M1234567890.lis mos=M1234567890.mosaic.cub
```

Converting to other formats

GeoTiff

GeoTIFF is a common format for working with map-projected data on Earth, and often has better support than the more obscure ISIS cube format. To convert an ISIS cube to GeoTIFF, you will need to install the GDAL tools, and use `gdal_translate`. Basic usage requires specifying GeoTIFF ("`GTiff`") as the output type, a suitable bit depth ("`Byte`" for 8-bit data, or "`UInt16`" for 16-bit), and using the `-scale` flag. The following command will stretch the min/max I/F values in the input (which range 0.0-1.0) to the full range of the output format (0-255 for 8-bit images):

```
gdal_translate -ot Byte -of GTiff -a_nodata 0 -scale mosaic.cub mosaic.tif
```

In cases where the image has some extremely bright pixels (particularly images of human or robotic landers), the default stretch may result in a very dark image. In this case, you will need to manually specify the input and output data ranges. (Tip: You can use the ISIS `percent` program to get the DN value for a certain percentile of an image: `percent from=mosaic.cub percent=0.1`. 0.1% to 99.9% usually gives a good-looking output image.) The following command stretches values form 0.0 to 0.35:

```
gdal_translate -ot Byte -of GTiff -a_nodata 0 -scale 0.0 0.35 0 255 mosaic.cub mosaic.tif
```

If you specify a custom stretch, and want NULL pixels in the ISIS cube to be preserved as unique values (DN 0) in an 8- or 16-bit GeoTIFF, you will need to limit the range of pixel values in ISIS using `fx` before running `gdal_translate`. Otherwise, valid data values below the minimum specified input DN (such as shadowed craters) will be incorrectly output as NoData (DN 0) instead of the minimum valid data value of DN 1.

```
minDN=$(percent from=mosaic.cub percent=0.1 | awk '/Value/{print $3}')

maxDN=$(percent from=mosaic.cub percent=99.9 | awk '/Value/{print $3}')

fx f1=mosaic.cub to=mosaic.clip.cub equation="f1 * (f1 >= $minDN) * (f1 <= $maxDN) + $minDN *
(f1 < $minDN) + $maxDN * (f1 > $maxDN)"

gdal_translate -ot Byte -of GTiff -a_nodata 0 -scale $minDN $maxDN 1 255 mosaic.clip.cub \
mosaic.8bit.tif

# Or, for a 16-bit output file:

gdal_translate -ot UInt16 -of GTiff -a_nodata 0 -scale $minDN $maxDN 1 65535 mosaic.clip.cub \
mosaic.16bit.tif
```

Recent versions of GDAL can convert GTiffs (and many other formats) into ISIS cubes, using the `-of isis3` flag. If ISIS includes an importer for a specific format, that will generally be better, but this covers a lot of additional formats, and allows the use of ISIS commands to edit GeoTIFFs.

PNG, etc

For common image formats used for making figures, the `isis2std` program works very well. The default stretch of 0.5-99.5% is usually too high-contrast for the high dynamic range of LROC NAC images, so it's often good to use a wider range:

```
isis2std from=mosaic.cub to=mosaic.png minpercent=0.1 maxpercent=99.9
```

Version and permanent URL of this document

This is version 1.1.3 of this guide, last updated 15 December 2025. The most recent version of this document, along with citation information, can always be found at https://doi.org/10.5281/zenodo.10055768.