### Processing Steps

The basic GCP Correlation processing flow consists of the following steps:

1. Read the GCPs and L1G image.
2. For each GCP:
   1. Compute the predicted location of the GCP in the L1G image using the GCP map projection coordinates and any specified predicted offset.
   2. Extract an image window from the L1G image at the predicted location.
      1. Make sure the image window contains sufficient non-fill image data.
      2. Make sure the L1G image and GCP image chip are in the same map projection (UTM zone). Reproject (see below) the GCP chip if necessary.
   3. Correlate the GCP image chip with the L1G window to find the optimum match point.
   4. Test the measured correlation and offset against predefined thresholds.
   5. Write out the GCP mensuration results.

The reprojection of the GCPs in the prototype code is done as a separate step through the process called gcpretrieve. This process is a precursor to the actual correlation process. It is also worth noting that the resampling methodology is slightly different between the Landsat heritage code and the prototype. The Landsat methodology used a table of weights that were applied to each chip in order to perform the reprojection.

The GCP correlation procedure was implemented in the heritage ALIAS prototype. Though the correlation process is conceptually simple, it is computationally intensive so the ALIAS implementation was designed to be efficient. This included taking advantage of parallel processing. These processing efficiency measures make the heritage implementation somewhat more complicated than it might otherwise be. The remainder of this processing discussion follows the L8/9 prototype, which was based on the heritage ALIAS implementation, to illustrate how the conceptually simple flow outlined above was mapped to a computationally efficient implementation.

#### Prototype Code

Input to the executable is an ODL file; output is an ASCII file containing measured offsets between the input image file and GCP library. Under the prototype output/input file directory, there is a directory called chips that contains the heritage type GCP data structures and files. Under the prototype output/input directory called the ADD that contains the ODL files needed, the HDF5 input image file, a Perl script that is needed, and the CPF.

The prototype code was compiled with the following options when creating the test data files:

-g -Wall -march=nocona -m32

**Get GCP Correlate Parameters (get\_gcpcorrelate\_parameters)**

This function gets parameters from the ODL parameter file.

**Get GCP Information (get\_gcp\_information)**

This function reads the GCPs from the input GCP library.

**Process GCP (process\_gcp)**

This function processes all of the GCPs by extracting the GCP image chip, extracting the image window, performing the correlation for each point, and then writing the results to the GCP data file.

Notes:

1. The correlation routines want things in sample, line order, so the fit\_offset pairs returned are horizontal (sample) and then vertical (line) offsets. In contrast, the GCP data file contains fit\_offset in line, sample order.
2. To calculate the correlated location, know the following:
   1. The correlate routines return the offset from the reference window (chip) to the search window (L1G), which is also the offset from the nominal reference point to the actual point in the search window.
   2. The integer location of the predicted location roughly corresponds to the integer location of the reference location. We need to report the predicted search line, sample of the reference point, and the offset from the predicted point to the correlated point. To get the correlated location, we start with the integer location of the predicted point because this corresponds to the integer part of the ref point (this is why gcp[num\_used\_gcp].fit\_offset subtracts the fractional part of the predicted location). Then, we add the fractional part of the reference coordinate because this is really the point we are going after. After that, we add the correlation fit\_offset because this tells how the reference point relates to the location in the search window.
3. The calculation for fit\_offset only works correctly because we are assuming the reference and search points are at the center of the window (plus some fractional distance), and the difference in window size is accounted for by nom\_off; if the reference point was not at the middle of the chip, this would have to be adjusted.

*Initialize Parallel Correlator (xxx\_init\_parallel\_correlator)*

This function initializes an instance of the parallel correlator. All of the multiprocessing resources are created, and the memory for the chip buffers and queue structures is allocated.

*Get Correlation Chip Buffers (xxx\_get\_corr\_chip\_buffers)*

This function returns buffers for the search and reference chips that will be submitted to the parallel correlator. Getting buffers from this routine and not submitting them to the parallel correlator will quickly exhaust all of the buffers available. The buffers will be freed when the parallel correlator is closed. When compiled in single-threaded mode, the same set of buffers is used for every pair of chips.

*Close Parallel Correlator (xxx\_close\_parallel\_correlator)*

This function is the routine that needs to be called after all of the chips have been submitted to the correlator. This routine will wait until all of the threads have completed, then destroy this instance of the parallel correlator. The results of the correlation are not valid until this routine returns.

*Get Search Line/Sample (get\_search\_line\_samp)*

This function finds the line and sample that corresponds to the given projection y and x. Since the L1G image is positioned (map projection) north up, to find the line (sample), subtract the upper-left projection y(x) value from the GCP projection y(x) value and divide the result by the line(sample) pixel size.

Notes:

The line and sample numbers are 0-relative.

This will not work for a path-oriented image.

*Extract Window (ias\_misc\_extract\_window)*

This function extracts an image window around a specific GCP. From the input image, a window of the specified size will be extracted around the GCP line and sample. If the window is an odd size, the extra line and/or sample will be at the beginning of the imagery. The data in the window representing portions outside of the imagery will be filled with zeros.

The two steps to the extraction are as follows:

1. Data type conversion of whatever the L1G image is to float
2. Setting the calculated window correctly into the buffer (even if the calculated window falls partially outside of the image)

*Check Fill (oli\_check\_fill)*

This function checks the input window for fill data over the specified percentage. This routine is useful to determine if there is too much fill data in a window. If too much fill data exists, then the window might not be good for correlating. Fill data nominally has a value of 0.0.

*Extract Chip ( xxx\_extract\_chip)*

This function reads the specified image chip. The image chip is always assumed to be a flat binary file containing chip\_size[0] x chip\_size[1] BYTE pixels (see note #1).

*Resample chip if necessary (build\_gcp\_lib)*

New logic, derived from the Landsat Product Generation System (LPGS) heritage, will be required here to check the image chip map projection, and if necessary, resample the chip to match the L1G image. This is necessary when working with a global GCP repository containing GCPs extracted from multiple source scenes in multiple UTM zones. The GCPs falling inside of a particular L1G image will frequently (particularly at high northern latitudes) be drawn from source images in neighboring UTM zones. Note that this is not a problem in Antarctica, where a single polar stereographic projection is used. It is also worth noting that the resampling techniques between the LPGS heritage code and the L8/9 prototype are not the same.

Image chip reprojection proceeds as follows:

1. Compute the image chip UL corner coordinates from the GCP UTM coordinates, the GCP image line/sample coordinates, and the image chip pixel size:
   1. UL Corner X = GCP X - GCP sample coordinate \* chip pixel size
   2. UL Corner Y = GCP Y + GCP line coordinate \* chip pixel size

Note that the GCP line/sample coordinates are relative to a zero-origin at the center of the upper-left chip pixel.

1. Project the GCP latitude and longitude to the L1G map projection (UTM zone) using the projection transformation package (see OLI/TIRS LOS Projection ADD) to compute GCP X' and GCP Y' projected coordinates.
2. Compute the desired "new" chip UL corner in the L1G projection using the new GCP X' and GCP Y' coordinates, rounding off to a whole multiple of the pixel size:
   1. UL Corner X' = GCP X' - GCP sample coordinate \* chip pixel size
   2. UL Corner Y' = GCP Y' + GCP line coordinate \* chip pixel size
   3. UL Corner X' = round(UL Corner X'/chip pixel size)\*chip pixel size
   4. UL Corner Y' = round(UL Corner Y'/chip pixel size)\*chip pixel size
3. Compute the "new" GCP line/sample coordinates in the reprojected chip:
   1. GCP sample coordinate' = (GCP X' - UL Corner X')/chip pixel size
   2. GCP line coordinate' = (UL Corner Y' - GCP Y')/chip pixel size
4. For each point in the new chip:

For line = 0 to nlines-1

Compute: Y' = UL Corner Y' - line\*chip pixel size

For samp = 0 to nsamps-1

1. Compute: X' = UL Corner X' + samp\*chip pixel size.
2. Convert (X',Y') to old chip projection (X,Y) using the projection transformation package.
3. Compute: oline = (UL Corner Y - Y)/chip pixel size.
4. Compute: osamp = (X - UL Corner X)/chip pixel size.
5. If the point (oline, osamp) falls inside of the old chip boundary, then interpolate a DN value at that location using bilinear interpolation:
   1. lindex = (int)floor(oline)
   2. sindex = (int)floor(osamp)
   3. fline = oline - lindex
   4. fsamp = osamp - sindex
   5. DN(oline,osamp) =

DN(lindex,sindex)\*(1-fline)\*(1-fsamp) +

DN(lindex+1,sindex)\*fline\*(1-fsamp) +

DN(lindex,sindex+1)\*(1-fline)\*fsamp +

DN(lindex+1,sindex+1)\*fline\*fsamp

Else DN(oline,osamp) = 0

5. Use the reprojected image chip and GCP line/sample coordinates in the GCP correlation procedure.

The build\_gcp\_lib, or gcpretreive process, is separated from the GCPcorrelate process to emulate the GCP retrieval process from the database containing the GCP image chips and their corresponding metadata. This retrieval process also contains the following C modules:

*GCPretrieve -Main driver for GCP retrieval process.*

*get\_gcp\_lib - Reads GCPs according to set of criteria*

*get\_gcp\_information - Wrapper for reading GCPLib information*

*get\_gcp\_proj\_parms - Reads projection information from the image file metadata*

*get\_gcpretrieve\_parameters - Read input ODL parameters*

This code was based on the Landsat ETM+/TM heritage code for GCP retrieval and chip reprojection.

*Put GCP (io\_put\_gcp)*

This function writes all GCP records to the specified output file. This function writes out a set of GCP data records. If the file already exists, it will be overwritten.

***Write GCP (io\_write\_gcp)***

This function writes one record to the GCP data file. The file pointer is left at the end of the current record so sequential calls of xxx\_write\_gcp will sequentially write all of the records in the file. The GCP data file is assumed to be an ASCII file containing one line of text per GCP data record. Each record contains: point\_id, reference\_line, reference\_sample, latitude, longitude, elevation, predicted\_search\_line, predicted\_search\_sample, delta\_y (line), delta\_x (sample), accept/reject\_flag, correlation coefficient, reference band number, search band number, search SCA number (0 for SCA-combined images), chip source (DOQ, GLS).

*Submit Chip to Correlator (xxx\_submit\_chip\_to\_corr)*

The xxx\_parallel\_corr module implements a parallel correlation object. Using the Posix threading interface, up to MAX\_CORR\_THREADS (or the number of processors available - whichever is less) are created to perform correlation. The main thread that creates the parallel correlator is then responsible for "feeding" the parallel correlator chips to correlate. The xxx\_submit\_chip\_to\_corr places the chips into a queue. The correlation threads remove the chips from the queue and perform the correlation. The results of the correlation are not immediately available to the application since xxx\_submit\_chip\_to\_corr returns before the correlation is complete.

Before any of the correlation results are used, the application must call xxx\_close\_parallel\_correlator to make sure all of the chips have been correlated and to destroy the correlation threads.

***Grey Correlator ( xxx\_grey\_corr)***

This function correlates a reference subimage with a search subimage using the pixel grey levels, and evaluates the results.

**Grey Cross Product Same-size (xxx\_grey\_cross\_ss)**

This function computes the unnormalized (raw) sum of pixel-by-pixel cross products between the reference and search images for every combination of horizontal and vertical offsets of the reference relative to the search image for windows of the same size (in one dimension at least).

**Grey Normalization Same-size (xxx\_grey\_norm\_ss)**

This function converts raw cross-product sums to normalized cross-correlation coefficients, using tabulated statistics from previous step (grey\_cross\_ss). This function is much simpler than the one for unequal-sized windows, since all normalizing is done by the space domain same size correlator. All that has to be done is statistics gathering to set up the peak finder.

**Grey Cross Product (xxx\_grey\_cross)**

This function computes the unnormalized (raw) sum of pixel-by-pixel cross-products between the reference and search images for every combination of horizontal and vertical offsets of the reference, relative to the search image. This function works for windows of unequal size.

**Grey Normalization (xxx\_grey\_norm)**

This function converts raw cross-product sums to normalized cross-correlation coefficients, while tabulating statistics needed for subsequent evaluation. This function works for unequal window sizes.

**Grey Evaluation (xxx\_grey\_eval)**

This function evaluates various measures of correlation validity and extracts a subarea of the cross-correlation array centered on the peak.

**Fit Registration (xxx\_fitreg)**

This function fits a quadratic surface to the neighborhood of the correlation peak and from it determines the best-fit registration offsets and their estimated errors.

#### Input and Output File Details

The details of the fields contained in the input GCP library file (Table 6‑1) and the output measured GCP file (Table 6‑2) are as follows:

|  |  |
| --- | --- |
| **Field** | **Description** |
| Header Text | Zero or more lines of ASCII text, each line beginning with the "#" symbol to designate it as a header comment, describing GCP library contents. |
| Data Start Marker | "BEGIN" - static text to indicate beginning of data area |
| Number of GCPs | Integer number (N) of GCPs to follow (new) |
| GCP Record Fields: | One set per GCP |
| GCP Number | Sequence number of GCP in this package (1 to N) |
| GCP ID | Unique ID for GCP of the form: ppprrrnnnn  where: ppprrr = WRS path/row GCP was taken from  nnnn = chip sequence number |
| GCP Image Chip Line Coordinate | GCP location within image chip - line coordinate (fractional pixel). |
| GCP Image Chip Sample Coordinate | GCP location within image chip - sample coordinate (fractional pixel). |
| GCP Latitude | GCP latitude in degrees. |
| GCP Longitude | GCP longitude in degrees. |
| GCP X | GCP projected X coordinate in meters. |
| GCP Y | GCP projected Y coordinate in meters. |
| GCP Height | GCP WGS84 ellipsoid height in meters. |
| Image Chip GSD | Chip pixel size in meters. |
| Image Chip Lines | Number of lines in the image chip. |
| Image Chip Samples | Number of samples in the image chip. |
| GCP Source | Source of GCP, either "DOQ," "GLS," or "TM6" |
| GCP Type | Control/validation point flag, either "CONTROL" or "VALIDATION" |
| Image Chip Projection | UTM or PS |
| Image Chip Zone | UTM zone number (1-60). Use 0 for PS. |
| Image Chip Date | yyyymmdd = year/month/day of the GCP source |
| Image Chip | Link to chip image data (could be in file named with GCP ID) |

Table 6‑1. Input GCP Library Contents

|  |  |
| --- | --- |
| **Field** | **Description** |
| GCP Record Fields: | One set per GCP |
| Point ID | GCP ID (see Table 6‑1) |
| GCP chip line location | Line location of the GCP within the chip |
| GCP chip sample location | Sample location of the GCP within the chip |
| GCP latitude | GCP WGS84 latitude in degrees |
| GCP longitude | GCP WGS84 longitude in degrees |
| GCP height | GCP WGS84 ellipsoid height in meters |
| Predicted GCP image line | Predicted line location of the GCP in the L1G image |
| Predicted GCP image sample | Predicted sample location of the GCP in the L1G image |
| GCP image line offset | Measured line offset from the predicted location |
| GCP image sample offset | Measured sample offset from the predicted location |
| Correlation success flag | Flag 0 = correlation failure, 1 = success |
| Correlation coefficient | Measured correlation coefficient (new) |
| Search band number | L1G band number used |
| Search SCA number | L1G SCA where the GCP was found |
| Chip source | GCP source (DOQ or GLS or TM6) |

Table 6‑2. Output GCP Mensuration File Contents

#### Maturity

Though much of the ALI model correction algorithm will be reusable, there are several areas where changes are expected:

* The heritage process takes WRS path/row as input and accesses a static GCP Library file set indexed by path/row. For the L8/9 implementation, it is assumed that GCP storage and retrieval is managed externally and that this process will be provided with a set of GCPs applicable to the image area. The prototype uses a pre-processing step involving a Perl script and a C executable called gcpretrieve to mimic the database retrieval and chip reprojection steps. These steps are in the prototype and verification sections.
* The computed correlation coefficient is added as an output to make it available for subsequent outlier filtering, if necessary.
* At high latitudes, scenes will frequently straddle multiple UTM zones. The control point chips falling in a given scene may thus be in more than one projection, leading to difficulties in correlating the chips (due to the rotation between the chip projection and the mensuration image projection). This problem has been solved in the Landsat processing system (LPGS) by including logic in gcpcorrelate that resamples the GCP chips, if necessary, to match the mensuration image projection. This logic is not part of the ALIAS heritage code, but will be needed to support global L8/9 product generation using the GLS ground control. The reprojection of the chips has been addressed and prototyped. This is no longer an issue with the prototype code.
* The baseline plan for L8/9 GCP correlation is to use a terrain-corrected (rather than the heritage systematically corrected) image for GCP mensuration. This is a departure from the ALIAS (and Landsat) heritage, motivated by the implications of processing off-nadir images.
  + The predicted GCP locations used to establish the search area in the mensuration image are computed without reference to terrain-displacement effects. For nadir images, the terrain-induced cross-track offsets introduced between these “flat Earth” predictions and the actual GCP locations are small enough to fit inside of the normal GCP search window: a maximum elevation of 8000m corresponds to ~1200m of parallax at the edge of the swath, which is a displacement of ~40 pixels (at the 30m GLS control resolution). A 64x64 GCP chip and a 128x128 search area can accommodate offsets up to +/-31 pixels, whereas a 256x256 search area can accommodate offsets up to +/-95 pixels.
  + For off-nadir images, the terrain sensitivity is roughly tripled, so offsets up to 120 pixels would have to be accounted for. Since increasing the search area substantially increases processing time and increases the likelihood of a false GCP match, and because L8/9 pointing should be sufficiently accurate to generate predicted GCP locations that are within a pixel or two most of the time, it would be better to account for terrain offsets when predicting GCP locations to minimize the search area. One way to do this is to calculate the magnitude of this effect and include it in the computation of the predicted GCP locations. The problem with this is that the LOS model correction algorithm uses the measured offset between predicted and measured GCP locations to derive model corrections, so any adjustment to the predicted location based on a computed terrain offset will have to be accounted for in the precision-correction algorithm. Thus, complicated offset prediction and offset removal logic would need to be added to both the GCP correlation and LOS model correction algorithms.
  + A better way to accomplish the same objective is to perform the mensuration on a terrain-corrected image, where the terrain offsets have been accounted for explicitly in the image-generation process. This approach is preferable to attempting to correct for terrain point-by-point in the mensuration of a systematically corrected image, but it does have some drawbacks:
    - Using a terrain-corrected mensuration image will require using the DEM as another input to the LOS model correction algorithm in order to compensate for any difference between the GCP elevation and the corresponding DEM elevation at the point where the GCP match was found in the mensuration image. This adds complexity to an already complex algorithm.
    - Misregistration between the systematic image and the DEM can cause the terrain-correction process to inject high-frequency image distortion. This would probably not be a huge problem, given the expected accuracy of L8/9 pointing and ephemeris knowledge. In areas of significant relief, even a slightly misregistered DEM may provide a mensuration image that is more similar to the GCP chips (which are terrain corrected) than a systematic image would be.

#### Notes

The following are additional background assumptions and notes:

* The heritage GLS, TM6, DOQ image chips are stored as 8-bit (BYTE) arrays, whereas the L8/9 imagery will be 16-bit (or float). The correlation is performed on floating-point data so both the image and the chips are converted to float on input. Thus, the image and chip data types need not match.
* The correlation result fit method defines the algorithm used to estimate the correlation peak location to subpixel accuracy. Only the quadratic surface-fitting method described in this ADD is supported in the baseline algorithm.
* Though the normal baseline for measuring control points is to use an SCA-separated terrain corrected image, this algorithm should also function with a combined-SCA image so that it can be used to measure test-point GCPs in L1T product images to support the geometric accuracy characterization algorithm.
* The GCPs in the GCP repository (part of the Infrastructure Element) should be flagged as either “control” points, to be used for LOS model correction, or “validation” points, to be used for geometric accuracy characterization. The utility that extracts control points from this repository should be able to extract either control set. The “control” set would contain the majority of the points. The “validation” flag would only be used in areas where more than some minimum threshold number of GCPs are available. The Cal/Val team would set these flags when the GCP repository was loaded and could adjust them thereafter, if necessary. Criteria for selecting validation points would be based upon considerations such as the following:
  + The total number of available GCPs in the scene must exceed some minimum (e.g., 100).
  + Points that fall on the boundary (or, more precisely, the convex hull) of the GCP set would not be validation point candidates.
  + Points that are within some maximum distance (e.g., 25 km) of another GCP would be validation point candidates.
  + The goal would be to develop an automated validation point identification algorithm that would operate somewhat like an outlier rejection algorithm: identify the best validation point candidate based on a set of criteria, remove it from the control point list, and iterate until no additional validation points are identified.
* Scenes with poor geolocation accuracy can lead to the actual GCP L1G image locations being sufficiently far from their predicted locations so as to make it impractical to expand the GCP search window to the extent necessary to find the GCPs. An optional parameter to specify an a priori predicted offset provides a more reliable way to find and correctly correlate the GCPs in this situation. This can occur early in the mission, before the first on-orbit sensor alignment calibration, or during an anomaly investigation.