### OLI Resampling Algorithm

#### Background/Introduction

The OLI resampling method is used to take an L1R image, which has unevenly spaced pixels with respect to the surface of the object imaged, and create a reprojected image where all image pixels are located within an evenly spaced set of grid points, or output space, with respect to the object imaged. This mapping is subject to the errors associated with the interpolation method used to determine the digital numbers associated with the output image.

The geometric resampling grid and the geometric model are used to calculate the mappings between the input and output space. The geometric model contains the individual detector offsets from a nominal location, while the geometric resampling grid contains all other mapping variables. The resampling grid provides a mapping from a 2D input space to a 3D output space, and vice versa. The output space corresponds to x/y/z projection locations, while the input space corresponds to line/sample locations within the L1R. The z component in output space is elevation. If elevation is not to be accounted for during processing, an elevation of zero is used to map output pixels to input pixels.

Due to what can be rather large sample-to-sample offsets within an L1R image, the cubic convolution interpolation option works in a two-step process. A hybrid set of pixels in the sample direction are created using cubic convolution resampling in the line direction. This creates a set of unevenly spaced pixels in the sample direction. The Akima A interpolation method is then used to determine the final digital number for the output image by resampling the hybrid pixels in the sample direction. The nearest-neighbor resampling option simply determines the closest input pixel for the corresponding output pixel.

The OLI resampling algorithm is derived from the corresponding ALI algorithm used in ALIAS. The sensor architecture between the instruments is similar enough that a majority of the ALIAS algorithm can be reused. The baseline geometric modeling approach assumes that the 3D gridding approach used within ALIAS can also be used for OLI. The heritage algorithm will have to be modified to accommodate L8/9 data formats.

#### Dependencies

The OLI resampling algorithm assumes that the Ancillary Data Preprocessing, LOS Model Creation, and Line-of-Sight Projection to Ellipsoid and Terrain algorithms have been executed, and an L1R has been generated. If a DEM is given as input to account for relief, or terrain, displacement the grid must have an adequate number and range (elevation bounds) of z-planes to cover the entire elevation range within the L1R. A geometric model and grid must be available for the L1R. The Ancillary Data Preprocessing (6.1.4), LOS Model Creation, and Line-of-Sight Projection to Ellipsoid and Terrain Algorithm Description Documents (ADDs) contain more information about the data structure and contents of the Geometric Model and Resampling Grid.

#### Inputs

The resampling algorithm and its component sub-algorithms use the inputs listed in the following table. Note that some of these “inputs” are implementation conveniences (e.g., using an ODL parameter file to convey the values of and pointers to the input data).

|  |
| --- |
| **Algorithm Inputs** |
| L1R Image |
| Resampling Grid (see the Line of Sight Projection ADD for contents) |
| Bands to process |
| Terrain correction Flag (yes/no) |
| DEM (if terrain flag set to yes) |
| SCA combine flag (yes/no) |
| Geometric model (see Line of Sight Model Creation ADD for contents) |
| Resampling type (CC,NN) |
| Minimum and maximum DN of output (see note #9) |
| Output data type |
| α (if resampling type is CC) (defaults to -0.5) |
| Fill pixel value |

#### Outputs

|  |
| --- |
| Resampled output image (L1G, L1GT, or L1T) |
|  Resampled image data (band separated, either SCA combined or SCA separated) |
|  Image data metadata fields (see Table 6‑10 and Table 6‑11) |

#### Options

Cubic convolution or nearest-neighbor resampling
Creating an output image with Sensor Chip Assemblies (SCAs) combined or separated

Applying terrain correction, yes or no

#### Procedure

OLI resampling interpolates radiometrically corrected but geometrically raw image data to a map-projected output space. The resampling process uses information stored in the resampling grid, along with focal plane calibration data stored in the geometric model, to map output projection locations to an input location. Since an input location for an output pixel typically lies at a non-integer location, interpolation is used to find the pixel values associated with this non-integer location. OLI resampling is performed on the geometrically raw L1R data using one of two methods: cubic convolution combined with the Akima A method, or nearest neighbor. Note that Modulation Transfer Function Compensation (MTFC) and bilinear resampling are not supported in the baseline algorithm. Due to the lack of inherent band registration and the need to perform subpixel registration to achieve OLI band alignment, cubic convolution, combined with the Akima A interpolation method, will be used to generate the standard L8/9 products. It is also important to have the best subpixel accuracy in the output image during geometric characterization and calibration, so cubic convolution is chosen for interpolation during the characterization and calibration of the OLI instrument. The ALIAS-heritage nearest-neighbor interpolation capability is also provided as an option for special-purpose science products and testing purposes. Since this document focuses on both standard product generation and geometric characterization and calibration, the only interpolation method discussed in detail here is the cubic convolution combined with the Akima A method.

During resampling, there is a need to know what input pixel goes with a given output pixel. The OLI geometric processing system does not have a “true” inverse model to perform this calculation. Instead, for a given output pixel, the corresponding input pixel is found from the forward and inverse mapping coefficients stored in the resampling grid. There are two scenarios when performing this calculation. The first involves performing resampling for a systematic image, in which case the dimension for z, or elevation, is either zero or a constant value. This involves only a two-dimensional operation in line and sample. The second involves performing resampling for a terrain-corrected image. A terrain-corrected image has the effects of relief removed from the output imagery. When working with a terrain-corrected image, a three-dimensional operation is performed during the inverse mapping, with the dimensions being input (L1R) line, input sample, and elevation (Figure 6‑32). Both procedures of mapping output pixel locations to input pixel locations are discussed below.

Due to the layout of the OLI focal plane, there are along-track offsets between spectral bands within each SCA, along-track offsets between even and odd SCAs, and a reversal of the band ordering in adjacent SCAs. This leads to an along-track offset in the imagery coverage area for a given band between odd and even SCAs, as well as an offset between bands within each SCA. To create a more uniform image coverage within a geometrically corrected output product, the leading and trailing imagery associated with these offsets is trimmed. This trimming is controlled by a set of latitude/longitude bounds for the active image area for each band, contained in the input resampling grid. Trimming is implemented by converting these bounds to a look up table that lists the starting and ending sample location of active (non-fill) data for each line of the output image.



Figure 6‑32. 3D Grid Representation

##### Using the geometric grid to map an output pixel location to an input pixel location.

To find an input line/sample location for an output line/sample location, given that the elevation is zero:

1. Calculate an input line and sample location using the rough polynomial stored in the resampling grid and the current output line and sample location.



Where:

 ra = rough polynomial mapping coefficients for line mapping

 rb = rough polynomial mapping coefficients for sample mapping

 M = Number of sample coefficients in the polynomial

 N = Number of line coefficients in the polynomial

Previous experience when working with the ALI instrument has demonstrated a 1st order polynomial in both the line and sample direction will suffice for the rough polynomial, thus M = N = 1.



There is no evidence to believe that this will not also be the case when working with the OLI instrument.

1. Calculate the grid cell location for the approximate input line and sample location.



Where:

number of lines per cell = size of the grid cell in lines

number of samples per cell = size of the grid cell in samples

Set this grid cell column and row location as the current grid cell column and row location.

1. Using the current grid cell location, check if the correct grid cell has been found.

Use input (current) mapping grid cell coefficients (ai and bi) to map output line and sample to input:

input line = b0 + b1 \* output sample + b2 \* output line + b3 \* output line \* output sample

input sample = a0 + a1 \* output sample + a2 \* output line + a3 \* output line \* output sample

Calculate the grid cell location for this input line and sample location:



If the new grid cell (new row and new column) is the same as the current grid cell (current row and current column):

The correct grid cell has been found, inverse grid mapping coefficients for this grid cell are used to calculate the input line/sample for the current output line/sample.

If the new grid cell (new row and new column) is not the same as the current grid cell (current row and current column):

The new grid cell is chosen as current grid cell, and the 3rd step is repeated until the correct grid cell is found.

This routine or function listed above, of mapping output pixel locations to input pixel locations without taking into account elevation, will be referred to as ols2ils (output space line-sample to input space line-sample mapping). The ols2ils sub-algorithm takes a given output line and sample location and calculates the grid cell column and row location, along with the corresponding input line and sample location for that output location.

To find an input line/sample location for an output line/sample location, given that the elevation is not zero:

1. Find the z planes that the elevation associated with the output pixel falls between.



 Where:

 elevation = elevation associated with current output location (from DEM)

 elevation increment = elevation increment between z planes stored in grid

 zelev=0 = zero z plane, the index of the zero elevation z-plane

 The output line/sample falls between z plane and z plane+1.

1. Call ols2ils for z plane and z plane+1. This yields (input sample0, input line0), and (input sample1, input line1).
2. Interpolate between z plane and z plane + 1 to find input line and sample location for elevation.

Calculate elevations for z plane and z plane + 1:

elev0 = elevation increment \* ( z plane - zero z plane )

elev1 = elev0 + elevation increment

Calculate weights for ols2ils results:



input sample = input sample0 \* w0 + input sample1 \* w1

input line = input line0 \* w0 + input line1 \* w1

Where:

input sample0 = input sample for z plane

input sample1 = input sample for z plane + 1

input line0 = input line for z plane

input line1 = input line for z plane + 1

This routine or function listed above, which performs the three-dimensional output space line-sample to input space line-sample mapping, is referred to as 3d\_ols2ils.

##### Resampling Methodology

The along- and cross- track detector offsets are applied during resampling. These include both the dynamic odd and even terrain-dependent relief and parallax effects that were calculated during the resampling grid generation, and the individual detector selection shift that are stored in the OLI geometric model. The nature of these geometric effects due to the individual detector characteristics is such that, in input space, they are evenly spaced in the line direction, but unevenly spaced in the sample direction. This is because as you move along raw imagery in the line direction, the detector number does not change. Since the detector number does not change along the line direction in raw input space, the along-track detector offset, stored within the geometric model, does not change. These geometric effects, due to these detector offsets, are slowly varying in time, staying essentially constant within the area that resampling is performed. Therefore, the along-track geometric effect, and essentially spacing in the line direction, can be treated as a constant over this area. The same logic helps explain why the across-track detector offset is not constant in the sample direction, since each sample comes from a different detector. This creates unevenly spaced samples in raw input space. Figure 6‑33 shows an example of a detector layout and its associated offset. The squares in Figure 6‑33 represent a location of an input pixel, taking into account the detector offsets. The circle with the cross hairs in Figure 6‑33 represents the true input location for the current output pixel. At this point, an interpolated value is needed to represent the current output pixel.



Figure 6‑33. Example Detector Layout

Detector offsets are handled in the resampler by first applying a resampling kernel in the line direction that assumes evenly spaced detectors. Cubic convolution interpolation is used in the line direction; this will align a set of pixels in the sample direction. Once the pixels are aligned in the sample direction, at uneven spacing, the Akima A interpolation is used to find the final output pixel value.

Cubic convolution interpolation uses a set of piecewise cubic spline interpolating polynomials. The polynomials have the following form:



Four points, centered on the point to be interpolated, are used in interpolation. The weights for each point are generated from *f(x)*. The α in the cubic convolution function is a variable parameter that affects the edge slope of the function. For standard processing, a value of -0.5 is used. Figure 6‑34 shows an example of what the cubic convolution function looks like, and the corresponding weights for a phase shift of zero (marked as Xs).

Figure 6‑34. Cubic Convolution Function

As stated previously, for the OLI resampler, the cubic convolution resampling process produces a set of hybrid points that are aligned in the line direction. This is done by resampling several sets of L1R pixels in the line direction using the cubic convolution kernel; each time cubic convolution is performed, one hybrid pixel is produced. The set of hybrid points produced from the cubic convolution process is not evenly spaced in the sample direction. Figure 6‑35 illustrates a set of hybrid samples that has been aligned in the line direction using the cubic convolution process.



Figure 6‑35. Hybrid Pixels for Detector Offsets

The Akima A method for interpolation is used for interpolating the hybrid pixels created from the cubic convolution process. This method of interpolation does not require the samples used to be evenly spaced. The Akima A method uses a third order polynomial for interpolation. The interpolating polynomial is defined by the coordinates and the slopes of the two points that are on either side of the point to be interpolated. The slopes of the adjacent points are determined as follows:

If five points are defined as 1, 2, 3, 4, and 5, then the slope at point 3, t, is defined as follows:



 Where:

m1 = slope of line segment defined by points 1 and 2

m2 = slope of line segment defined by points 2 and 3

m3 = slope of line segment defined by points 3 and 4

m4 = slope of line segment defined by points 4 and 5

The Akima A method of interpolation is based on the values (*y*) and slopes (*t*) on either side of the point that is to be interpolated. The interpolating polynomial for a point x between xi and xi+1 is then defined as follows:



 Where:

*x* = sample location of point to be interpolated

*x*i = location of point to the left of x

*x*i+1 = location of point to the right of x

*yi* = DN value for the input point at *x*i

*y* = interpolated DN value for an output line and sample location

This methodology must be adjusted somewhat to account for higher frequency image distortion effects than those that can be captured by the conventional resampling grid. To model such effects, the L8/9 attitude data stream is separated in to low-frequency and high-frequency segments, with the low-frequency portion being used for the OLI line-of-sight projection operations that build the resampling grid. The high-frequency data are interpolated to match the OLI panchromatic band line sampling times and stored in the OLI LOS model in a jitter table for application as an extra correction at image resampling time. The OLI Line-of-Sight Model Creation ADD (6.2.1) describes the process of separating the attitude data stream by frequency.

Sensitivity coefficients that relate these high-frequency roll-pitch-yaw jitter terms to the equivalent input image space line and sample offset effects are stored in the OLI LOS grid. This makes it possible to look up the roll-pitch-yaw jitter for each image line being resampled, and convert the jitter values to compensating input line/sample corrections that are used to refine the image interpolation location coordinates. The OLI Line-of-Sight Projection/Grid Generation ADD (see 6.2.1) describes the generation of these sensitivity coefficients. The process by which the jitter table from the OLI model and jitter sensitivity coefficients from the OLI grid are used during image resampling is shown schematically in Figure 6‑36. The items in green in the figure are new structures added to support jitter correction.

Since the jitter effects vary by image line, the time delay between even and odd (or deselected) detectors will lead to slightly different jitter effects in adjacent image samples, as depicted in Figure 6‑37. The figure shows six time samples (t0 through t5) for six adjacent detectors. Note that the input line location returned by the grid is adjusted differently for the even and odd detectors due to their timing offset. Including the effects of detector deselect, the interpolated line location for the hybrid pixels could be different for each detector. The current approach does not account for sample-to-sample variations in jitter for each detector, applying the jitter correction only at the output location. This preserves the uniform along-track sampling assumption required to apply the cubic convolution kernel. Also note that while the interpolation location is adjusted relative to the input pixel locations in the line direction, the detector sample locations are adjusted relative to the interpolation location in the sample direction. The jitter-adjusted resampling procedure is explained in more detail below.



Figure 6‑36. OLI LOS Model and OLI LOS Grid Jitter Correction Data Flow

****

Figure 6‑37. Jitter Effects in Image Resampling

##### Building The SCA-trimmed Look Up Table (LUT).

Allocate the SCA-trim LUT. There is a starting and ending sample location of active or valid imagery stored for each line of output in the SCA-trimming look up table.

 LUT = malloc( 2 \* nl )

 Where nl = number of lines in output imagery

Given the set of geographic corner coordinates, read from the input grid file, that represent valid imagery for a given band:

1. Map four corners to the output projection coordinates.
2. Map four output projection coordinates to line and sample coordinates.
3. Set up polygon definition from four coordinates:

<px0,py0> = <sample upper left, line upper left>

<px1,py1> = <sample upper right, line upper right>

<px2,py2> = <sample lower right, line lower right>

<px3,py3> = <sample lower left, line lower left>

<px4,py4> = <sample upper left, line upper left>

1. Set up sample locations for each line that is outside of active imagery:

osamp1 = -1.0

osamp2 = output number of samples

for nn = 0 to 3

 if px[nn] < osamp1 then osamp1 = px[nn] – 1.0

 if px[nn] > osamp2 then osamp2 = px[nn] + 1.0

1. Initialize LUT values to fill for all output lines:

For nn = 0 to (2 \* number of output lines)

 LUT[nn] = 0

1. For nn = 0 to number of output lines (nn and current line are synonymous).

6.1. Define line by sample locations calculated from 4 and current line

 <x0,y0> = <osamp1, nn>

 <x1,y1> = <osamp2, nn>

6.2. Determine the intersection between the sides of the polygon defined in 3 and line defined in 6.1

 Initialize the number of intersections for the current line:

intersections = 0

 For nn = 0 to 3

 (Simple line intersection routine)

 xlk = x0 – x1

 ylk = y0 – y1

 xnm = px[nn] – px[nn+1]

 ynm = py[nn] – py[nn+1]

 xmk = px[nn+1] – x1

 ymk = py[nn -1] – y1

 det = xnm \* ylk – ynm \* xlk

 if ( | det | <= TOL ) lines are parallel, no intersection found.

 s = ( xnm \* ymk - ynm \* xmk ) / det

 t = ( xlk \* ymk - ylk \* xmk ) / det

 if( s<0.0 || s>1.0 || t<0.0 || t>1.0 )

no intersection found

 else

intersection found, calculate point:

xp[ intersections ] = x1 + xlk \* s

yp[ intersections ] = y1 + ylk \* s

intersections++

6.3. If the number of intersections from 6.2. is two, then the current line has valid active imagery and the look up table values are these intersections and represent the start and stop of valid imagery. Store values in the SCA-trim lookup table.

 if xp[0] > xp[1]

 LUT[ 2 \* nn ] = xp[1]

 LUT[ 2 \* nn + 1] = xp[0]

 else

 LUT[ 2 \* nn ] = xp[0]

 LUT[ 2 \* nn + 1] = xp[1]

(Note: If the number of intersections is not two, then the current line has no valid active imagery and the SCA-trim lookup table will contain points outside of the imagery, and fill will be used).

##### Load/Build Information

To resample a Level 1R data set, the image file, grid file, geometric model (and, if the effects terrain are to be removed, a DEM) must be opened. See note #3.

##### Resample Level1R Imagery

Loop on each band of each SCA for resampling.

1. Get the resampling grid for the band and SCA to be processed.
2. Build the SCA-trimming table.
3. Read one band of imagery for one SCA. See note #7.
	1. Initialize the jitter correction parameters.

If the current band is panchromatic, then jitter\_scale = 1

Otherwise, jitter\_scale = 2

1. Loop on output line/samples.
	1. Check if the output line/sample is within SCA-trimming bounds.

if output sample > LUT[ 2 \* output line ] &&

output sample < LUT[ 2 \* output line + 1 ] then proceed

else output pixel = fill

* 1. If the image is terrain corrected, calculate the elevation-dependent input line/sample location.

4.2.1) Get the elevation for the output pixel location X/Y location from DEM (elevation). See note #3.

4.2.2) Map the output line/sample back into the input space, using the grid and the function 3d\_ols2ils.

* 1. If the image is not terrain corrected, calculate the zero elevation (ellipsoid surface) input

line/sample location.

 4.3.1) Set the elevation to zero.

 4.3.2) Map the output line/sample back into the input space, using the grid and the function ols2ils.

* 1. Calculate the actual input sample location; for sample location (int)input sample calculated from either 4.2 or 4.3:

4.4.1) Calculate the detector offset parallax scale.

Scale = (int) floor(detector along-track offset + 0.5) (in the geometric model). See note #4.

4.4.2) Calculate the sample odd/even parallax offset.

 Δsample\_oe = (d0 + elevation \* d1 ) \* scale

 Note that (d0 + elevation \* d1 ) is the parallax (in pixels) per pixel of along-track offset from the nominal detector location.

 Where:

 d0,1 = odd/even sample parallax coefficients stored in the grid

 4.4.3) Get the sample fractional offset

 fractional sample offset =

 detector across-track offset (in the geometric model)

4.4.4) Calculate the sample jitter adjustment.

 4.4.4.1) Calculate the index into the jitter table for the current image line

jit\_index = (int)(jitter\_scale\*(input line – pixel column fill (defined below)))

Make sure the jitter index is within the range of the jitter table. Set to the min or max value (whichever is closest) if it is outside the range.

4.4.4.2) Calculate the fractional jitter table index.

jit\_index = jitter\_scale \* input line – floor( jitter\_scale \* input line)

4.4.4.3) Calculate the simple sample jitter adjustment.

samp\_jitter0 = samp\_sens[0] \* jitter\_table[jit\_index].roll

 + samp\_sens[1] \* jitter\_table[jit\_index].pitch

 + samp\_sens[2] \* jitter\_table[jit\_index].yaw

samp\_jitter1 = samp\_sens[0] \* jitter\_table[jit\_index+1].roll

 + samp\_sens[1] \* jitter\_table[jit\_index+1].pitch

 + samp\_sens[2] \* jitter\_table[jit\_index+1].yaw

samp\_jitter = samp\_jitter0 \* (1-jit\_index) + samp\_jitter1\*jit\_index

Where:

samp\_sens[0] is the sample direction jitter roll sensitivity,

samp\_sens[1] is the sample direction jitter pitch sensitivity,

samp\_sens[2] is the sample direction jitter yaw sensitivity,

for the current grid cell, from the OLI grid.

jitter\_table[n] is the jitter table roll-pitch-yaw vector for row n,

from the OLI model.

4.4.4.4) Refine the sample jitter to compensate for line jitter.

line\_jitter0 = line\_sens[0] \* jitter\_table[jit\_index].roll

 + line\_sens[1] \* jitter\_table[jit\_index].pitch

 + line\_sens[2] \* jitter\_table[jit\_index].yaw

line\_jitter1 = line\_sens[0] \* jitter\_table[jit\_index+1].roll

 + line\_sens[1] \* jitter\_table[jit\_index+1].pitch

 + line\_sens[2] \* jitter\_table[jit\_index+1].yaw

line\_jitter = line\_jitter0 \* (1-jit\_index) + line\_jitter1\*jit\_index

Where:

line\_sens[0] is the line direction jitter roll sensitivity,

line\_sens[1] is the line direction jitter pitch sensitivity,

line\_sens[2] is the line direction jitter yaw sensitivity,

for the current grid cell, from the OLI grid.

This is the error in the line coordinate used above, due to line jitter.

samp\_rate =

 samp\_sens[0]\*(jitter\_table[jit\_index+1].roll-jitter\_table[jit\_index].roll)

+ samp\_sens[1]\*(jitter\_table[jit\_index+1].pitch-jitter\_table[jit\_index].pitch)

+ samp\_sens[2]\*(jitter\_table[jit\_index+1].yaw-jitter\_table[jit\_index].yaw)

This is the rate of change of sample jitter with the line coordinate.

samp\_jitter += line\_jitter\*samp\_rate

This is the sample jitter correction adjusted for the effects of line jitter.

4.4.5) actual input sample = input sample - sample\_oe - samp\_jitter - fractional sample offset (See note #5). These corrections are subtracted rather than added because, rather than adjusting the input space interpolation location, we are computing the apparent location of the detector to the left of the interpolation location to make sure we have the correct range of samples to feed the interpolation logic. If the above adjustments lead to the “actual input sample” being greater than (to the right of) the original input sample location, then we move our sample range one more sample to the left. We perform a similar calculation on the detector to the right of the input space interpolation location to make sure that we do not have to shift one more sample in that direction. See also the note in 4.6.2 below.

* 1. Create the fractional pixel shift for the current input location:

 line = input line - (int) input line

 sample = input sample - (int) input sample

* 1. Create aligned samples for Akima resampling by applying cubic convolution weights in line direction.
		1. Loop on the actual input sample location:

For hybrid sample = (int) actual input sample - 2 to (int) actual input sample +3 (Note #5. One extra hybrid sample created to the left and right of the minimum number of samples needed for Akima interpolation)

In the case of NN resampling, the loop limits are reduced to:

For hybrid sample = (int) actual input sample to (int) actual input sample +1

* + - 1. Calculate the line and hybrid sample detector offset parallax scale

scale = (int) floor(detector along-track offset + 0.5) (in the geometric model). See note #4.

* + - 1. Calculate the odd/even detector offset, parallax correction, and jitter correction for the hybrid detector.
				1. Odd/even detector offset and parallax corrections.

Δline\_oe = (c0  + elevation \* c1 ) \* scale + pixel column fill - nominal detector fill - at\_offset[hybrid sample]

Δsample\_oe = (d0 + elevation \* d1 ) \* scale

Where:

c0,1 = odd/even line parallax coefficients stored in the grid

d0,1 = odd/even sample parallax coefficients stored in the grid. Note that (c0  + elevation \* c1 ) is the along-track parallax (in pixels) per pixel of along-track offset from the nominal detector location and (d0 + elevation \* d1 ) is the across-track parallax (in pixels) per pixel of along-track offset from the nominal detector location.

* + - * 1. Jitter correction

The sample jitter correction is calculated as described in 4.4.4 above. The line jitter correction is calculated as follows:

jit\_index = (int)(jitter\_scale\*(input line – pixel column fill))

jit\_index = jitter\_scale \* input line – floor( jitter\_scale \* input line)

line\_jitter0 = line\_sens[0] \* jitter\_table[jit\_index].roll

 + line\_sens[1] \* jitter\_table[jit\_index].pitch

 + line\_sens[2] \* jitter\_table[jit\_index].yaw

line\_jitter1 = line\_sens[0] \* jitter\_table[jit\_index+1].roll

 + line\_sens[1] \* jitter\_table[jit\_index+1].pitch

 + line\_sens[2] \* jitter\_table[jit\_index+1].yaw

line\_jitter = line\_jitter0 \* (1-jit\_index) + line\_jitter1\*jit\_index

Where:

line\_sens[0] is the line direction jitter roll sensitivity,

line\_sens[1] is the line direction jitter pitch sensitivity,

line\_sens[2] is the line direction jitter yaw sensitivity,

for the current grid cell, from the OLI grid.

This is the error in the line coordinate due to jitter.

line\_rate =

 line\_sens[0]\*(jitter\_table[jit\_index+1].roll-jitter\_table[jit\_index].roll)

+ line\_sens[1]\*(jitter\_table[jit\_index+1].pitch-jitter\_table[jit\_index].pitch)

+ line\_sens[2]\*(jitter\_table[jit\_index+1].yaw-jitter\_table[jit\_index].yaw)

This is the rate of change of line jitter with the line coordinate.

line\_jitter += line\_jitter\*line\_rate

This is the line jitter correction adjusted for the second order effects of line jitter. Note the similarity to the sample correction described in 4.4.4.4.

* + - 1. Calculate the new hybrid line location.
				1. hybrid line = (int)floor(input line + ∆line\_oe + line\_jitter) .

Note that in this case, we add the corrections since we are adjusting the interpolation location.

* + - 1. Calculate the new fractional hybrid line location.

Δhybrid line = input line + Δline\_oe +line\_jitter – hybrid line

If |Δhybrid line| > 1, then the integer line index must be adjusted and Δhybrid line brought back into the -1 < Δhybrid line < 1 range (see note #5).

* + - 1. Apply cubic convolution in the line direction to hybrid sample line DNs.
				1. Calculate the cubic convolution weights. See note #2



Where *f*  is equal to the cubic convolution function.

* + - * 1. Apply the cubic convolution weights to L1R DNs.

 hybrid line DN = w0 \* h0 + w1 \* h1 + w2 \* h2 + w3 \* h3

 Where

 w0,w1,w2,w3 = Cubic convolution weights for hybrid line.

 h0 = DN from L1R for hybrid sample, input line location - 1

 h1 = DN from L1R for hybrid sample, input line location

 h2 = DN from the L1R for the hybrid sample, input line location + 1

 h3 = DN from the L1R for the hybrid sample, input line location + 2

In the case of NN resampling, the values of the hybrid line and hybrid line are used to select the closest line for the current detector/sample column, instead of being used to compute weights. The hybrid line DN is then the L1R DN value for the closest line location.

* + 1. Calculate the apparent Akima pixel location for the current hybrid sample.

Akima pixel location xi =

 hybrid sample location - sample\_oe

 - across-track detector offset (in geometric model)

 - samp\_jitter (computed per 4.6.1.2.2 above)

Note that in this case, the across-track terrain parallax and sample jitter effects are subtracted instead of added. This is because we are adjusting the apparent detector location relative to the output pixel interpolation point instead of adjusting the output pixel interpolation location itself. We must do it this way in the sample direction because the adjustments are different for each detector. As for the across-track offset term, which is also unique for each detector, the detector offset corrections are designed to be applied as line-of-sight corrections in the instrument coordinate system. As such, the along-track offset is a +X LOS correction and the across-track offset is a +Y LOS correction. The instrument +X axis is in the +line direction, but the +Y axis is in the –sample direction, so this correction is also subtracted from the apparent detector location.

* 1. Calculate the output DN using Akima interpolation and hybrid line/sample information from 4.6.1 and 4.6.2.
		1. Calculate the Akima weights according to the pixel locations from 4.6.2.





Where:

DNn = hybrid DNs calculated from cubic convolution, step 4.6.1.

*x*n = Akima locations calculated in step 4.6.2.

*ak*n = Akima weights

*m*n = Akima slopes

* + 1. Calculate the output pixel DN using the Akima A method.



The output sample point is located between hybrid samples x2 and x3, where xn is from n=0…5.

In the case of NN resampling, the Akima pixel locations for the two closest detectors are examined to see which is closest to the output location. The hybrid line DN value for the closest detector is selected as the output DN value.

* 1. Write the output DN to the image file. See note #9.
1. Write out the data descriptor record for the image file. Table 6‑10 shows the baseline contents of the data descriptor record. All fields present in the table refer to the imagery associated with the DDR, unless otherwise specified. Note that the scene roll angle is a new field added for off-nadir acquisitions. It would be computed from the LOS model by interpolating the roll angle from the "original" attitude data sequence at the time corresponding to the precision model reference time t\_ref. This would be done using the logic described in the Find Attitude sub-algorithm in the LOS Projection ADD, except operating on the "original" rather than the "corrected" attitude data sequence. The logic for using the "original" data is so that this scene roll value will not change due to LOS model correction. The sign convention on the roll angle is such that a positive roll angle would correspond to a positive orbital Y coordinate that is looking to starboard (See note 11).

##### Combining SCAs into one output file.

For an SCA combined output image, the overlap region between SCAs can be handled by averaging the pixels between SCAs (See Note 12).

#### Prototype Code

Input to the executable is an ODL file; output is an HDF5 file containing the image data and corresponding metadata. The output format follows the format of the Level 1 (L1)G Data Format Control Book (DFCB) version 1.

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

 -g -Wall -march=nocona -m32

**Main driver for resampler (oliresample)**

The main driver for the OLI resampler performs the following steps or calls the following modules:

1. Read the input ODL parameters (getpar).
2. Read the OLI input file (oli\_get\_model).
3. Read the OLI grid headers (oli\_get\_grid\_headers).

 If terrain correction, read the DEM file (oli\_get\_dem).

1. Open the L1G image file (open\_l1g\_resamp\_image).
2. Get the fill pixel value (get\_fill\_pixel).

 For each band to process

1. Read the grid band pointers (oli\_get\_grid\_pointers).
2. Open/initialize the L1G band file (start\_l1g\_resamp\_band).
3. Set up the resampling kernel (Kernal\_Setup).
4. Read the resampling kernal information for resampling (get\_kernal\_info).

 For each SCA

1. Read one SCA’s worth of data from L0ra

(get\_input\_image\_data\_l0ra).

1. Resample the SCA worth of data (resample\_image).

 if not an SCA combined image file, write SCA’s worth of data (write\_l1g\_resamp\_band).

 If SCA combined image file, write full SCA file (write\_l1g\_resamp\_band).

1. Close the band in the L1G output file (stop\_l1g\_resamp\_writing\_band).
2. Free the grid band pointer (oli\_free\_grid).
3. Close the L1G image file (close\_l1g\_resamp\_image).
4. Update the L1G metadata (update\_l1g\_metadata).

**Get resampling processing parameters (getpar).**

This function reads the OLI resampling parameters from the ODL file. It also contains two functions, get\_combine\_sca and get\_fill\_pixel, that will return input flags as to whether 1) combine the SCAs in the output image and 2) what DN value should be used for fill.

**Resample a given set of DN value using the Akima method (akima).**

This function takes a given set of X locations with corresponding Y values and finds the Y value for the given input X location (xp). The function returns the interpolated Y value associated with coordinate xp.

**Calculate cubic convolution weight for a given location (cubic\_convolution).**

Given a cubic convolution alpha parameter and X value, return the Y value associated with the cubic convolution function.

**For a given band read one SCAs worth of L0R imagery (get\_input\_image\_data\_l0ra).**

Given an L0Rp file name, band number, and SCA number, read an SCA’s worth of data from the L0Rp file. The number of lines to read is taken from the number of lines stored in models image data structure.

**Set up resampling kernel (module kernal.c).**

Using a set of functions, create a set of resampling weights. The resampling kernel is created and managed though several steps within the kernal.c file.

 **Kernal\_Setup** sets up the kernal table or pointer. It allocates a pointer and calls Create\_Resampling\_Kernal\_1D to create a set of cubic convolution weights. Set is a look-up table of 1D cubic weights representing 1/64 of a shift in pixel locations.

 **Cleanup\_Kernal** frees up the cubic convolution pointer.

 **Create\_Resampling\_Kernal\_1D** creates a set of one-dimensional cubic convolution based on the input alpha parameter.

 **Get\_Resample\_Weight\_Table\_Ptr** returns a pointer containing a set of 1D cubic convolution weights.

 **get\_lines\_in\_kernal** returns the number of lines in the resampling kernal.

 **get\_samples\_in\_kernal** returns the number of samples in the resampling kernal.

 **num\_left\_kernal\_samples** returns the number of resampling weights to the "left" of the point that is to be interpolated.

 **num\_right\_kernal\_sample** returns the number of resampling weights to the "right" of the point that is to be interpolated.

 **num\_top\_kernal\_lines** returns the number of lines "above" the point to be interpolated.

 **num\_bottom\_kernal\_lines** returns the number of lines "below" the point to be interpolated.

 **get\_kernal\_step\_size** returns the offset size in pixels between two sets of resampling weights.

 **get\_kernal\_info** returns the number of steps (or number of sets of weights) within the resampling kernal, total number of sets of weights within the resampling table, width of the resampling kernal, and height of the resampling kernal.

**Read DEM file (oli\_get\_dem).**

Reads the Image Processing Element (IPE) L1G file containing DEM data.

**Open, close, write to L1G output image file (file output\_image\_data.c)**

The file output\_image\_data.c contains several routines used for managing the output L1G file. Calls and functions are as follows:

**open\_l1g\_resamp\_image** opens an L1G file.

**start\_l1g\_resamp\_band** opens one band within an L1G file.

**write\_l1g\_resamp\_band** writes image data to an L1G file.

**stop\_l1g\_resamp\_writing\_band** closes a band within an L1G file.

**close\_l1g\_resamp\_image** closed an L1G file.

**Resample one SCA for one band of L0Rp imagery (file resample\_image.c)**

The file resample.c contains several functions used in resampling imagery.

 **setup\_trim\_lut** builds a lookup table that contains the starting and ending output pixel of valid imagery. Everything outside of these bounds will be set as fill.

 **cleanup\_trim\_lut** frees a static buffer that contains an SCA-trimming lookup table array.

 **get\_kernal\_info** retrieves the resampling weight table and corresponding characteristics.

 **setup\_detector\_offsets** stores the detector offsets, along and across, level-0R fill, and nominal detector fill within arrays. Used by resample\_image for applying detector offsets when resampling imagery.

 **resample\_image** is the main guts of the resampler. Takes the image data, DEM data if terrain corrected, grid band pointer, and OLI model structure to resample one SCA or one band of imagery. Loops on output pixels mapping each output pixel location to an input location and resamples L0Rp (or L1R when it becomes available), using the algorithm described in the procedure section.

 **calc\_jitter** computes the sample and line direction jitter corrections for the current input line/sample location. These corrections are the adjustments to the input space interpolation location required to compensate for the high-frequency jitter present at the time of observation.

 **calc\_jitter\_samp** is a simplified version of calc\_jitter that computes only the sample direction jitter correction. It is implemented as a separate function for processing efficiency because it is invoked more frequently than calc\_jitter.

**Update L1G metadata information (update\_l1g\_metadata).**

Update L1G metadata according to the projection information stored within the resampling grid.

**Write out ENVI header file (write\_envi\_hdr).**

Writes out the ENVI header file for the image flat file that is written to disk. Only used for testing purposes.

**Input and Output File Details**

Output is an L1G image file formatted according to the L1G DFCB. The output is an HDF5 file. Table 6‑10 and Table 6‑11 list the metadata associated with the output file. These tables follow the metadata fields in version 1 of the Landsat 8/9 Level-1 G DFCB. The metadata is split up into a file metadata and band metadata. For further information on this format, see the L1G DFCB. Not all fields within the prototype metadata fields are filled in with valid data. Fields in which data are *not* correctly filled are indicated in italics (see notes #9 and #10).

File Metadata

|  |  |  |
| --- | --- | --- |
| Field | Description | Type |
|  Projection Code | GCTP projection code | integer |
|  Zone Code | Map projection zone code | integer |
|  Datum | Projection datum code | char[16] |
|  Spheroid Code | Projection spheroid code | integer |
|  Projection Units | Map projection units | char[12] |
|  Projection Parameters | GCTP projection parameters | double[15] |
|  *WRS Path* | *WRS-2 Path*  | *integer* |
|  *WRS Row* | *WRS-2 Row* | *integer* |
|  *Roll Angle* | *Off nadir pointing angle* | *double* |
|  *Spacecraft* | *Spacecraft name* | *char[32]* |
|  Collection Type | Acquisition type | char[32] |
|  *Capture Direction* | *Ascending or descending* | *char[32]* |
|  *Capture Date* | *Acquisition date* | *char[11]* |
|  *Capture Time* | *Acquisition Time* | *char[9]* |
|  *Correction Type* | *Product type* | *char[5]* |
|  *Resample Type* | *Resampling method* | *char[4]* |
|  *Software Version* | *Software version* | *char[11]* |
|  *Ingest Software Version* | *Ingest software version* | *char[11]* |

Table 6‑10. L1G File Metadata Fields

Band Metadata

|  |  |  |
| --- | --- | --- |
| Field | Description | Type |
|  Band Number | Band Number | integer |
|  *Band Name* | *Landsat 8/9 Band designation* | *char[30]* |
|  Upper Left Y | Upper-left Y map coordinate | double |
|  Upper Left X | Upper-left X map coordinate | double |
|  Upper Right Y | Upper-left Y map coordinate | double |
|  Upper Right X | Upper-left X map coordinate | double |
|  Lower Left Y | Lower-left Y map coordinate | double |
|  Lower Left X | Lower-left X map coordinate | double |
|  Lower Right Y | Lower-right Y map coordinate | double |
|  Lower Right X | Lower-right X map coordinate | double |
|  Projection Distance Y | Y map projection distance | double |
|  Projection Distance X | X map projection distance | double |
|  *Maximum Pixel Value* | *Maximum DN* | *double* |
|  *Minimum Pixel Value* | *Minimum DN* | *double* |
|  *Pixel Range Valid* | *Flag indicating valid pixel max/min* | *integer* |
|  *Maximum Radiance* | *Maximum radiance* | *double* |
|  *Minimum Radiance* | *Minimum radiance* | *double* |
|  *Spectral Radiance Scaling Offset* | *Offset to convert to spectral radiance* | *double* |
|  *Spectral Radiance Scaling Gain* | *Gain to convert to spectral radiance* | *double* |
|  *Radiance valid* | *Flag to indicate radiance values are valid* | *integer* |
|  *Reflectance Scaling Offset* | *Offset to convert to reflectance* | *double* |
|  *Reflectance Scaling Gain* | *Gain to convert to reflectance* | *double* |
|  *Reflectance valid* | *Flag to indicate radiance values are valid* | *integer* |
|  *Instrument Source* | *Instrument associated with band imagery* | *char[32]* |

Table 6‑11. L1G Band Metadata Fields

#### Maturity

1. Since the OLI 3D grid approach is adopted, the ALIAS code was reused with limited modifications.
2. Due to the detector select option aboard OLI, the detector offset approach has been changed. Under the new methodology, the along-track detector offsets are stored with the whole pixel adjustment needed due to the detector selected and the small subpixel adjustment that was present in the ALI CPF detector offset field. This leads to a need to separate out the fractional detector offset from the detector select offset at times during processing.
3. The problem of multiple terrain intersections needs to be addressed, particularly for off-nadir acquisitions. A terrain occlusion mask will be generated to identify these obstructed pixels (see note #1 below for additional details), but the current thinking is that it would not alter the behavior of the resampler, as sprinkling fill pixels throughout a product image can wreak havoc with some applications. Generating a separate terrain occlusion mask will allow users to evaluate the extent of the problem and apply the mask, if appropriate, to a particular application. The Terrain Occlusion ADD (6.2.5) addresses this.

All items in the maturity section have been addressed. The OLI 3D grid approach was adopted. The IPE L1G and L0R libraries were used within the prototype code. The detector delay logic was changed to handle the OLI detector select characteristics. The terrain occlusion ADD addresses the terrain issues associated with the OLI instrument.