# 「しきさい」タイルの画像化サンプル (Tile product handling)　[FAQ19]　2020-12-17 C

- プログラミングができれば画像化は簡単です。simple source for the image handling
  [1] hdf5ファイルからの読み取り・表示 read hdf5 and display
  [2] 地図投影・表示 map projection and display

- 前提条件 Requirements
  - ✓ C++ サンプルをコンパイル・実行できること Familiar with C++.
  - ✓ hdf5ライブラリとOpenCvライブラリが使えること hdf5 and OpenCv library must be installed

- サンプルプログラム Sample Program
  - ✓ C++ source :　TILEsample_20200627.cpp　　注）Python版も公開しますが説明は省略します

  注1) わかりやすさのために、エラー処理・端部処理・高速化処理がありません。利用者において改善をお願いします。
  Note-1: **NO** error handling or optimization is considered for an easier understanding. User must modify the code.
  注2) 極域および経度180degまたぎ処理に追加の考慮が必要です。
  Note-2: Additional consideration is necessary for polar region and 180deg longitude region.

- 地図投影処理 Map projection method (Next page)
  - ✓ SGC-180024 GCOM-C “SHIKISAI” Data Users Handbook algorithm (backward process)
    4.1.4.1 Level2 Product Generation Unit (PDF page-57)
    https://gportal.jaxa.jp/gpr/assets/mng_upload/GCOM-C/GCOM-C_SHIKISAI_Data_Users_Handbook_jp.pdf

- 使用条件 Terms and conditions
  - ✓ This sample is free. NO copyright restrictions.
  - ✓ NO GUARANTY from JAXA to this sample.
  - ✓ Please refer to OpenCv homepage for their terms and conditions. https://opencv.org/terms-and-conditions/
  - ✓ Please refer to HDF5 homepage for their terms and conditions. https://www.hdfgroup.org/terms-of-service/

# Map projection algorithm for TILE products (1 of 2)

- Forward and Backward projection
  - ✓ Forward: FROM: Tile pixel address (col, line) TO: (lon, lat) address. (SGC-180024 algorithm)
  - ✓ Backward: FROM: (lon, lat) address TO: to tile pixel address (col, line).

  Note: Forward method is used to know the (lon, lat) of each pixel. Backward method is convenient for map projection because no complicated pixel overlap/underlap consideration is necessary.

- Tile pixel address and global EQA pixel
  - ✓ Tile pixel address $(col, line)_{TILE}$ must be converted to global EQA address $(col, line)_{EQA}$ before forward projection.
  - ✓ Global EAQ address $(col, line)_{EQA}$ must be converted to tile pixel address $(col, line)_{TILE}$ after backward projection.

- Constants (Change constant 4800 to 1200 for 1km resolution tile.)
  - ✓ Resolution: $d = 180 / (18 \times 4800) = 0.0021$ [deg/line] @ 250m resolution
  - ✓ Equator size: $NP_0 = 2 \times NINT[180 \div d] = 172{,}800$ [cols] @ 250m resolution

- Caution (PDF page-4)
  - ✓ Common grid consideration is necessary to avoid the inconsistency with adjacent tile.
    Round-off error is happened in case converting the corner address to (lon, lat) grid.
    Use **common grid for ALL tiles** when you generate map grids before projection.

- Terminology

  EQA = EQual Area coordination (sinusoidal projection is used for TILE)

  EQR = Equal Rectangle coordination

# Map projection algorithm for TILE products (2 of 2)

**250m Tile Calc.**
SGC-180024 SGLI data handbook

**250m Tile coordinate**
$(\mathrm{col} \quad \mathrm{line})_{\mathrm{TILE}} \quad (\mathrm{H} \quad \mathrm{V})$

Note:
Upper left is (0, 0)
H = 0 to 35, V = 0 to 17
col = 0 to 4799, line = 0 to 4799

$$\begin{pmatrix}\mathrm{col}\\ \mathrm{line}\end{pmatrix}_{\mathrm{EQA}} = \begin{pmatrix}\mathrm{col}\\ \mathrm{line}\end{pmatrix}_{\mathrm{TILE}} + \begin{pmatrix}\mathrm{H}\cdot 4800\\ \mathrm{V}\cdot 4800\end{pmatrix}$$

Forward: Tile → Global

Backward: Global → Tile

$$\begin{pmatrix}\mathrm{col}\\ \mathrm{line}\end{pmatrix}_{\mathrm{TILE}} = \begin{pmatrix}\mathrm{col}\\ \mathrm{line}\end{pmatrix}_{\mathrm{EQA}} - \begin{pmatrix}\mathrm{H}\cdot 4800\\ \mathrm{V}\cdot 4800\end{pmatrix}$$

**EQA coordinate**
$(\mathrm{col} \quad \mathrm{line})_{\mathrm{EQA}}$

Integerization is not necessary

$$\mathrm{lat} = 90^{\circ} - (\mathrm{line}_{\mathrm{EQA}} + 0.5)\cdot \mathrm{d}$$

$$\mathrm{NP_i} = \mathrm{NP_0}\cdot\cos(\mathrm{lat})$$

$$\mathrm{lon} = \frac{360}{\mathrm{NP_i}}\left(\mathrm{col_{EQA}} - \frac{\mathrm{NP_0}}{2} + 0.5\right)$$

Forward: EQA → longitude, latitude

Backward: longitude, latitude → EQA

$$\mathrm{NP_i} = \mathrm{NP_0}\cdot\cos(\mathrm{lat})$$

$$\mathrm{line_{EQA}} = \frac{90 - \mathrm{lat}}{\mathrm{d}} - 0.5$$

$$\mathrm{col_{EQA}} = \mathrm{lon}\cdot\frac{\mathrm{NP_i}}{360} + \frac{\mathrm{NP_0}}{2} - 0.5$$

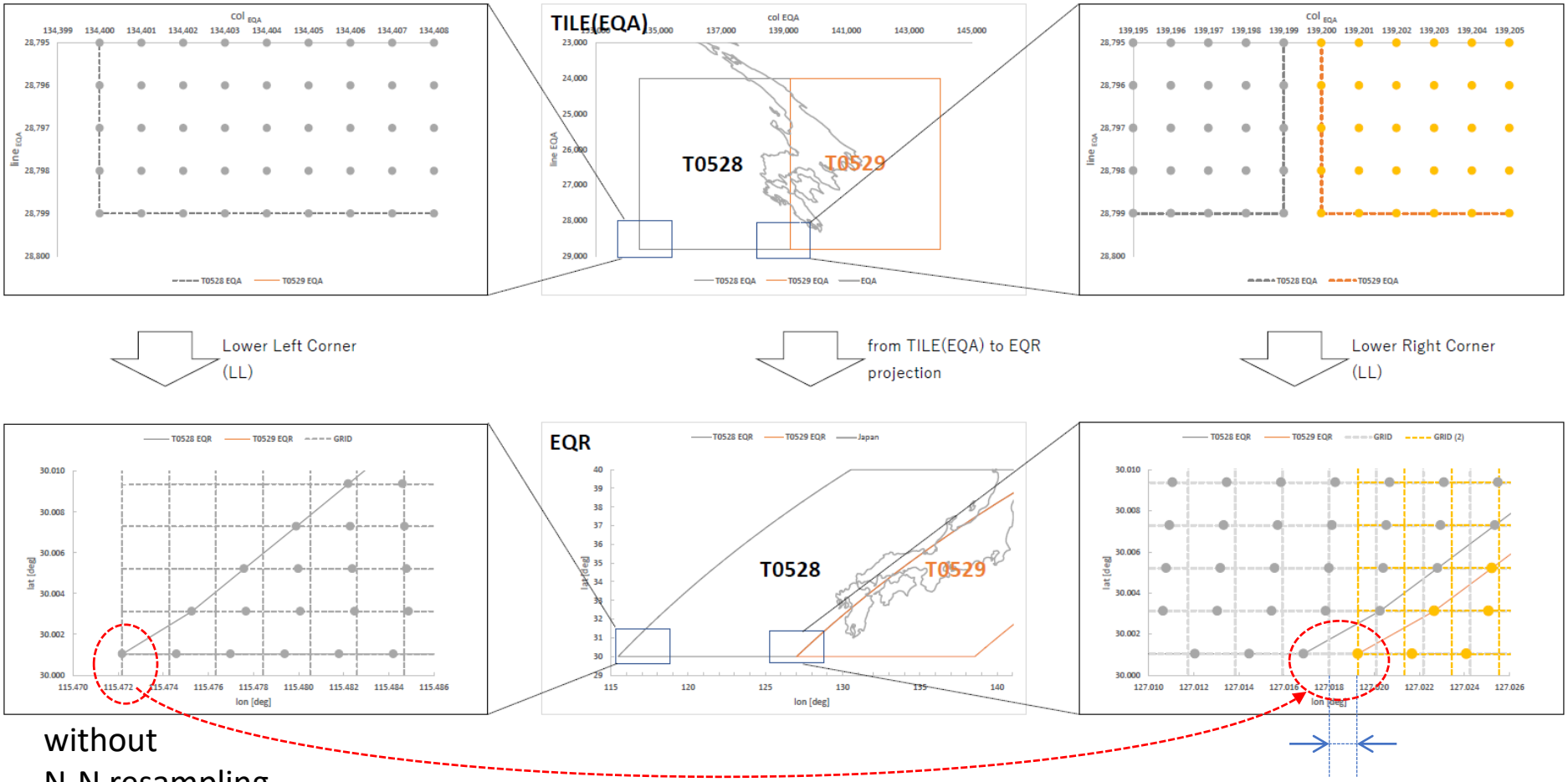
Note

"RasterPixelsArea" to "RasterPixelIsPoint" covertion

**EQR coordinate**
$(\mathrm{lon} \quad \mathrm{lat})_{\mathrm{EQR}}$

【Const】
Res.: d = 180 ÷ （18 x 4800） = 0.0021 [deg/line]
Equator size: $NP_0$=2 x NINT[180 ÷ d] = 172,800 [cols]

Note:
Common grid should be used for ALL tiles in order to avoid an overlap (or under lap) with adjacent tile.




Common grid consideration is necessary to avoid the inconsistency with adjacent tile. Round-off error (overlap or underlap) is happened in case converting the corner address to (lon, lat) grid without N-N resampling.
→ Use **<u>Common grid for ALL tiles</u>** when you generate map grids before projection

C

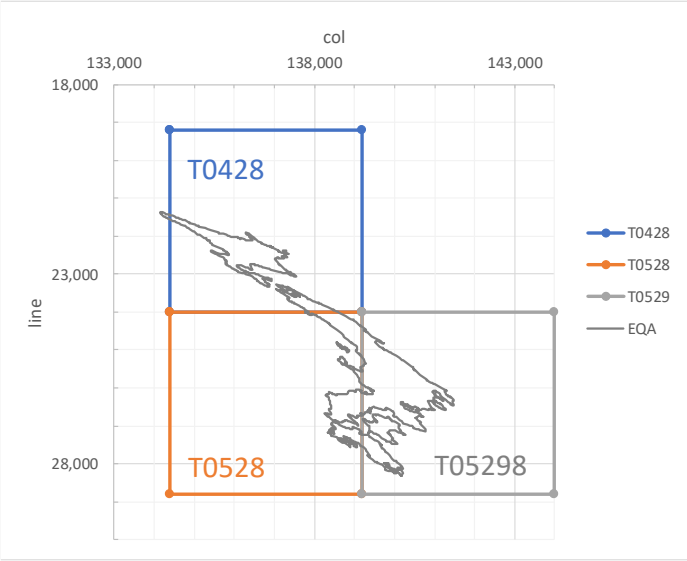
| Tile Corner calc. | | | | T0428 | | T0528 | | T0529 | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | v04 | h28 | v05 | h28 | v05 | h29 |
| Corner | col | line | | col, lon | line, lat | col, lon | line, lat | col, lon | line, lat |
| UL | 0 | 0 | EQA coord. | 134,400 | 19,200 | 134,400 | 24,000 | 139,200 | 24,000 |
| | | | EQR cord. Note | 155.5724 E | 50.0000 N | 130.5407 E | 40.0000 N | 143.5948 E | 40.0000 N |
| LL | 0 | 4,799 | EQA coord. | 134,400 | 23,999 | 134,400 | 28,799 | 139,200 | 28,799 |
| | | | EQR cord. Note | 130.5447 E | 40.0021 N | 115.4725 E | 30.0021 N | 127.0197 E | 30.0021 N |
| LR | 4,799 | 4,799 | EQA coord. | 139,199 | 23,999 | 139,199 | 28,799 | 143,999 | 28,799 |
| | | | EQR cord. Note | 143.5965 E | 40.0021 N | 127.0173 E | 30.0021 N | 138.5646 E | 30.0021 N |
| UR | 4,799 | 0 | EQA coord. | 139,199 | 19,200 | 139,199 | 24,000 | 143,999 | 24,000 |
| | | | EQR cord. Note | 171.1264 E | 50.0000 N | 143.5921 E | 40.0000 N | 156.6462 E | 40.0000 N |

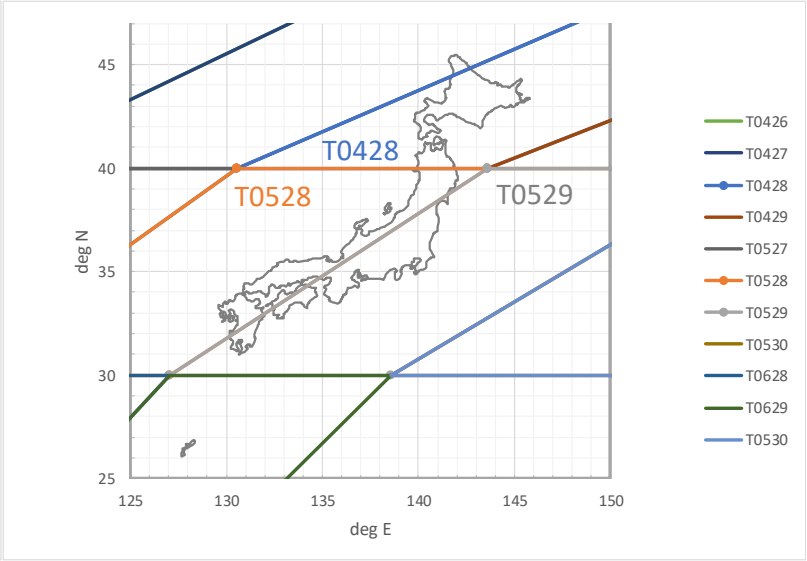
【Const.】

| | | | |
|---|---|---|---|
| Tile Size | lines | $lin_{tile}$ | 4,800 |
| | columns | $col_{tile}$ | 4,800 |
| Tile Num | vertical | $vtile_{num}$ | 18 |
| | horizontal | $htile_{num}$ | 36 |

Note: quantization error should be considered to avoid the inconsistency with adjacent tile.

| | | |
|---|---|---|
| 250m res. | $d$ | 0.002083 [deg/line] |
| equater size | $NP_0$ | 172,800 [cols] |

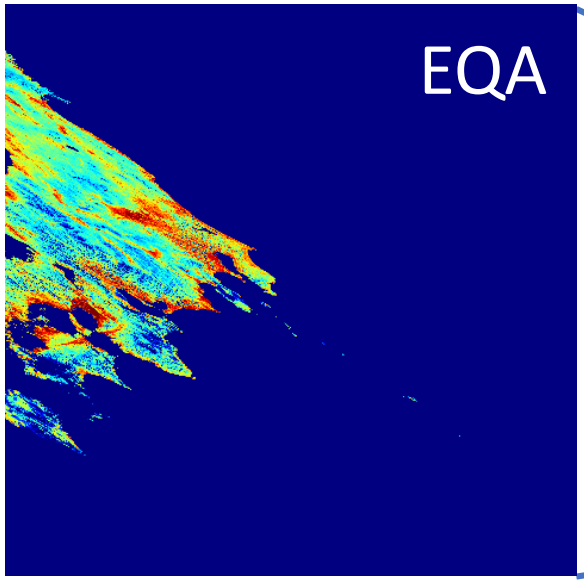


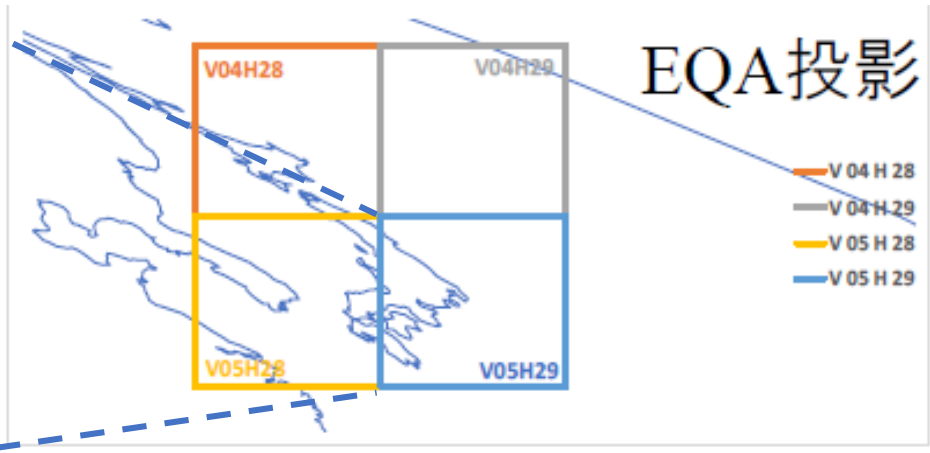



# Map projection Example

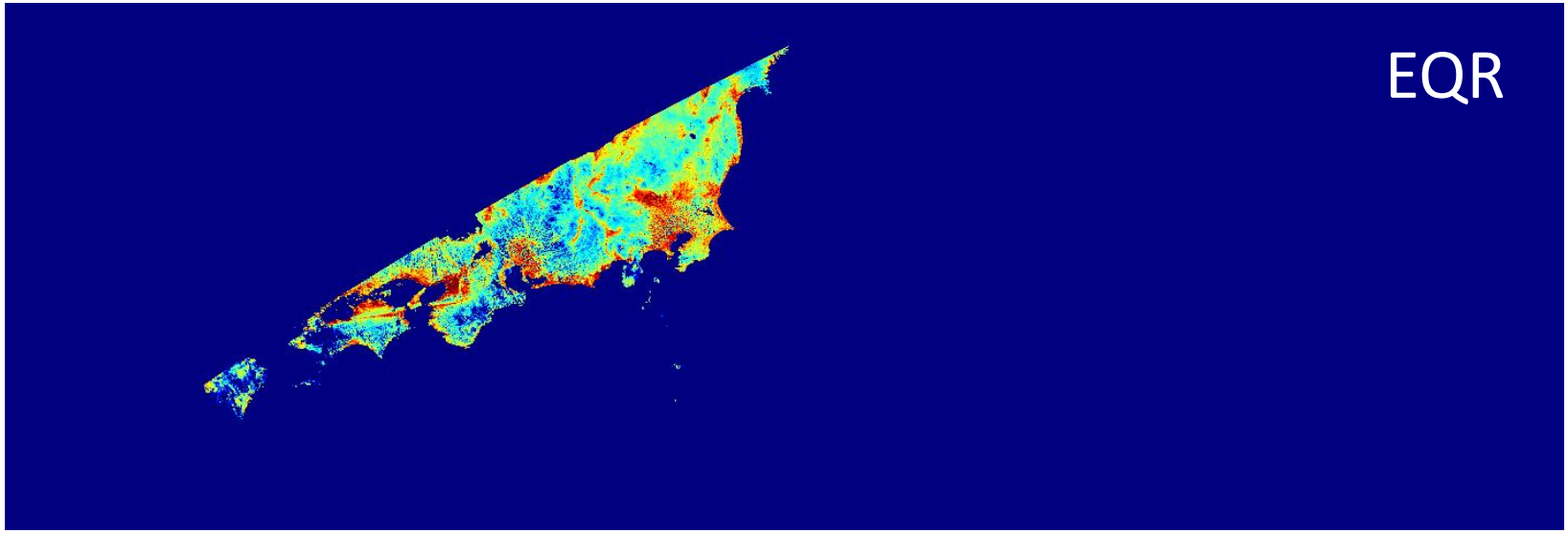
Before mapping (EQA coordination)

GC1SG1_20200826D01D_**T0529**_L2SN_LST_Q_2000_000.h5




After mapping (EQR coordination)



# Sample C++ Source(1 of 3)

## □Conditions

Target File name

```
string foldername = "C:¥¥data¥¥20200826_Japan¥¥";        // folder
string filename = "GC1SG1_20200826D01D_T0529_L2SN_LST_Q_2000_000.h5";          // file
string imggrp = "/Image_data";            // group
string sdname = "LST";                                  // sd
string hdf_png = "..¥¥" + sdname + "_LST.png";
string map_png = "..¥¥" + sdname + "_MAP.png";

float32 minLST = 283.0f;        // minimum LST for png
float32 maxLST = 323.0f;        // maximum LST for png
float32 pngratio = 0.1f;        // png size ratio

printf("target hdf file  : %s¥n", filename.c_str());
printf("target group sds : %s %s¥n", imggrp.c_str(), sdname.c_str());

string tileno = filename.substr(20, 5);                         // tile number
int vv = atoi(tileno.substr(1, 2).c_str());
int hh = atoi(tileno.substr(3, 2).c_str());
printf("tile no         : %s (VV=%02d, HH=%02d)¥n", tileno.c_str(), vv, hh);
```

## □hdf read (1/2)

```
filename = foldername + filename;
hid_t fid = H5Fopen(filename.c_str(), H5F_ACC_RDWR, H5P_DEFAULT);
hid_t gid = H5Gopen(fid, imggrp.c_str(), H5P_DEFAULT);
hid_t sdid = H5Dopen(gid, sdname.c_str(), H5P_DEFAULT);

hsize_t     dims[10];
int n_dims = H5Sget_simple_extent_dims(H5Dget_space(sdid), dims, NULL);

long lines = (long)dims[0];
long pixels = (long)dims[1];

cv::Mat dn(lines, pixels, CV_16UC1);     // 観測データ Mat配列(uint16)
herr_t status = H5Dread(sdid, H5Dget_type(sdid), H5S_ALL, H5S_ALL, H5P_DEFAULT, dn.ptr());

//  Attributes
float32 Offset, Slope;
uint16      MaxDN, MinDN;

hid_t aid = H5Aopen_name(sdid, "Offset");
status = H5Aread(aid, H5Aget_type(aid), &Offset);
status = H5Aclose(aid);
```

## □hdf5 read (2/2)

```
aid = H5Aopen_name(sdid, "Slope");
status = H5Aread(aid, H5Aget_type(aid), &Slope);
status = H5Aclose(aid);

aid = H5Aopen_name(sdid, "Maximum_valid_DN");
status = H5Aread(aid, H5Aget_type(aid), &MaxDN);
status = H5Aclose(aid);

aid = H5Aopen_name(sdid, "Minimum_valid_DN");
status = H5Aread(aid, H5Aget_type(aid), &MinDN);
status = H5Aclose(aid);

//  close HDF
status = H5Dclose(sdid);
status = H5Gclose(gid);
status = H5Fclose(fid);

printf("pixels=%ld, lines=%ld¥n", pixels, lines);
printf("attribute : Slope=%f/Offset=%f/MinDN=%d/MaxDN=%d¥n",
            Slope, Offset, MinDN, MaxDN);

//  convert to LST
constexpr float32 q_nan = std::numeric_limits<float>::quiet_NaN();
cv::Mat img(cv::Size(pixels, lines), CV_32F, q_nan);            // data array = Mat array(float32)
float32* ptr = (float32*)img.ptr();
uint16* uptr = (uint16*)dn.ptr();

float32 minValLST = q_nan;
float32 maxValLST = q_nan;
for (long idx = 0; idx < lines * pixels; idx++) {
            uint16 d = uptr[idx];
            if (d >= MinDN && d <= MaxDN) {
                        ptr[idx] = (float32)d * Slope + Offset;
                        minValLST = std::isnan(minValLST) ? ptr[idx] :
                                    (minValLST <= ptr[idx] ? minValLST : ptr[idx]);
                        maxValLST = std::isnan(maxValLST) ? ptr[idx] :
                                    (maxValLST >= ptr[idx] ? maxValLST : ptr[idx]);
            }
}
dn.release();
printf("min & max LST = %f & %f¥n", minValLST, maxValLST);
```

# Sample C++ Source(2 of 3)

## ☐display before mapping

```
cv::Mat dsp = (img.clone() - minLST)/ (maxLST - minLST)* 255.0f;
dsp.convertTo(dsp, CV_8U);
applyColorMap(dsp, dsp, cv::COLORMAP_JET);
cv::resize(dsp, dsp, cv::Size(), pngratio, pngratio);
imwrite(hdf_png.c_str(), dsp);
cv::imshow(sdname.c_str(), dsp);
cv::waitKey(0);
dsp.release();
```

## ☐ Tile projection constants

```
long tileCols = 4800;                    // tile width for Q
long tileRows = 4800;                    // tile height for Q
float32    degStep = 180.0f / (18.0f * (float32)tileRows);     // grid step
long NP0 = (long)(2.0f * (180.0f / degStep));        // equator size

printf("grid step [deg/col or deg/row] : %f¥n", degStep);
printf("equator size [col] : %ld¥n", NP0);

//  global tile col, row
long tileCol1st = hh * tileCols;
long tileColEnd = tileCol1st + tileCols - 1;
long tileRow1st = vv * tileRows;
long tileRowEnd = tileRow1st + tileRows - 1;

//  calc lat lon for corners (UL, UR, LL, LR)
float32 tileLatMax  = 90.0f - degStep * ((float32) tileRow1st   + 0.5f);
float32 tileLatMin  = 90.0f - degStep * ((float32) tileRowEnd   + 0.5f);

float32 NPi_max = (float32)NP0 * cos(tileLatMax * DEG2RAD);
float32 NPi_min = (float32)NP0 * cos(tileLatMin * DEG2RAD);

float32 tileLonUL   = 360.0f / (float32) NPi_max * ((float32)(tileCol1st    - NP0 / 2) + 0.5f);
float32 tileLonUR   = 360.0f / (float32) NPi_max * ((float32)(tileColEnd    - NP0 / 2) + 0.5f);
float32 tileLonLL   = 360.0f / (float32) NPi_min * ((float32)(tileCol1st    - NP0 / 2) + 0.5f);
float32 tileLonLR   = 360.0f / (float32) NPi_min * ((float32)(tileColEnd    - NP0 / 2) + 0.5f);

tileLonUL = degStep * (long)(tileLonUL / degStep);          // common grid
tileLonUR = degStep * (long)(tileLonUR / degStep);          // common grid
tileLonLL = degStep * (long)(tileLonLL / degStep);          // common grid
tileLonLR = degStep * (long)(tileLonLR / degStep);          // common grid

float32 tileLonMin = min(min(tileLonUL, tileLonLL), min(tileLonLL, tileLonLR));
float32 tileLonMax = max(max(tileLonUL, tileLonUR), max(tileLonLL, tileLonLR));

long mapCols = (long)((tileLonMax - tileLonMin) / degStep + 1);
long mapRows = (long)((tileLatMax - tileLatMin) / degStep + 1);

printf("tile lat range [deg] = %f : %f¥n", tileLatMin, tileLatMax);
printf("tile lon range [deg] = %f : %f¥n", tileLonMin, tileLonMax);
printf("dest map size (cols, rows) = (%ld, %ld)¥n", mapCols, mapRows);
```

RasterPixelIsPoint

Common grid should be considered

# Sample C++ Source(3 of 3)

C

## ☐ Tile projection

```
// source and dest Mat
float32* srcPtr = (float32*)img.ptr();
long        srcRowSize = (long)img.step / sizeof(float32);

cv::Mat dst(cv::Size(mapCols, mapRows), CV_32FC1, q_nan);
float32* dstPtr = (float32*)dst.ptr();
long        dstRowSize = (long)dst.step / sizeof(float32);

// Loop
for (float row = 0; row < (float)mapRows; row++) {
        // row -> lat
        float32 lat = tileLatMax - row * degStep;
        float32 NPi = (float32) NP0 * cos(lat * DEG2RAD);          // rev 2020/12/17
        long rowEQA = (long)((90.0 - lat) / degStep - 0.5) - tileRow0;
        if (rowEQA < 0 || rowEQA >= mapRows) continue;

        for (float col = 0; col < (float)mapCols; col++) {

                // col -> lon
                double lon = tileLonMin + col * degStep;
                long colEQA = (long)(lon * (double)NPi / 360.0 + (double)NP0 / 2.0 - 0.5)
                                - tileCol0;
                if (colEQA < 0 || colEQA >= tileCols) continue;

                // Pixel value
                long srcadr = rowEQA * srcRowSize + colEQA;
                long dstadr = (long)row * dstRowSize + (long)col;
                dstPtr[dstadr] = srcPtr[srcadr];
        }
}
img.release();
```

## ☐display after mapping

```
// display after mapping
dsp = dst.clone() * 255.0 / max_ref;      // max reflectance
dsp.convertTo(dsp, CV_8U);
applyColorMap(dsp, dsp, cv::COLORMAP_JET);
cv::resize(dsp, dsp, cv::Size(), pngratio, pngratio);
imwrite(map_png.c_str(), dsp);
cv::imshow(sdname.c_str(), dsp);
cv::waitKey(0);
dsp.release();
dst.release();
```

# タイル格子の定義（Japanese Only）

L2 モザイクのタイルは，EQA (sinusoidal)座標系において緯度経度の10 単位を区切りとしたデータである。

各タイル内のデータグリッドは，250m/500m/1km の観測モード毎にそれぞれ4800x4800, 2400x2400, 1200x1200 の画素により構成され，タイル境界はタイル端の画素の画素境界とオーバーラップする。

L2 モザイクタイルの概念図について下図 に示す。

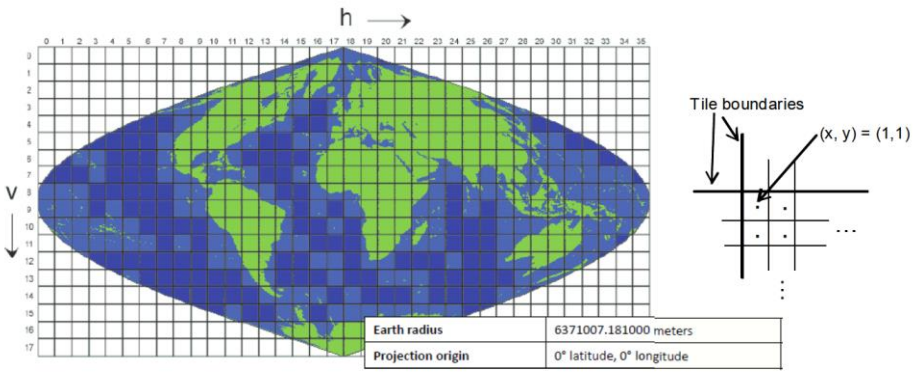



図 52　EQA (sinusoidal) tile coordinates における L2 モザイクタイルの概念図

任意の緯度経度座標(Φ,λ)[degree]について該当するタイルID(*h*, *v*)は以下により得られる。

$$h = \left[\lambda \cos\varphi' / 10 + 18\right]_{roundoff}$$

$$v = \left[9 - \varphi / 10\right]_{roundoff}$$

$$\varphi' = \varphi \cdot \pi / 180$$

注）緯度は測地緯度を使用しています。

任意の緯度経度座標(Φ,λ)[degree]の任意のタイルID(*h*, *v*)におけるグリッドアドレス(*x*, *y*)は以下により得られる。

$$x = \frac{m(\lambda \cos\varphi' - 10h + 180)}{10} + 0.5$$

$$y = \frac{n(90 - 10v - \varphi)}{10} + 0.5$$

$$m = n = \begin{cases} 4800 \text{ (250m mode)} \\ 2400 \text{ (500m mode)} \\ 1200 \text{ (1km mode)} \end{cases}$$

任意のタイルID(h, v)の任意のグリッドアドレス(x, y)についての緯度経度座標(Φ,λ)[degree]は以下により得られる。

$$\varphi = 90 - 10v - \frac{10(y - 0.5)}{n}$$

$$\lambda = \frac{10(x - 0.5)/m + 10h - 180}{\cos\varphi'}$$

SGLI タイルグリッドへのL1B データの投影は，ORTHO 補正モジュールによって算出されるオルソ投影アドレス対応点(Address Info)に基づいて行う。

本データはタイル毎の各画素に該当するL1B のID 及び画像アドレスが格納されており，後段のモザイク処理では，本情報に沿ってタイル毎の各アドレスにL1B 画像をリサンプリングする。

この際，一つのタイルアドレスに複数のL1B画像アドレスが対応する場合は，センサ天頂角が最も小さい画素を優先する。