「しきさい」シーンの画像化サンプル　(for English users, please see next page)

2021-05-06 C

- プログラミングができる方であれば画像化は簡単です。
  - [1] hdf5ファイルからの読み取り・表示
  - [2] 地図投影・表示

- 前提条件
  - ✓ C++ もしくは Pythonを実行できる方
  - ✓ hdf5ライブラリとOpenCvライブラリがインストールされていること。

- サンプルプログラム
  - ✓ C++サンプル　：　SGLIsample_20210506.cpp
  - ✓ Pythonサンプル　：　SGLIsample_20210506.py

  C

  注1）わかりやすくするために、エラー処理・端部処理・高速化処理を含めていません。
  利用者において改善をお願いします。特に、Python版は高速化しないと非常に時間がかかります。
  （Windows 10 Pro 64bit, i7 1.99GHz, 16GBメモリ，13分間程度）
  注2）極域および経度180degまたぎ処理には、追加の考慮が必要です。

  A

- 地図投影処理
  - ✓ OpenCvの射影変換関数を使った地図投影（頁3）

- 使用条件について
  - ✓ 本サンプルは、改変・再配布自由です。
  - ✓ 本サンプルに対して、JAXAは著作権を行使しません。
  - ✓ 本サンプルによるいかなる損害に対しても、JAXAは責任を負いません。
  - ✓ OpenCvの使用条件は、右記を参照して下さい。 https://opencv.org/terms-and-conditions/
  - ✓ HDF5の使用条件は、右記を参照して下さい。 https://www.hdfgroup.org/terms-of-service/

A

Sample source for SHIKISAI's scene product handling

2020-07-10 A

☐ How to generate the image

[1] How to read and display hdf5 file contents

[2] How to project to map and display

☐ Requirements

- ✓ Familiar with C++ or Python
- ✓ Hdf5 library and OpenCv library should be installed.

☐ Sample source

- ✓ C++ sample : **SGLIsample_20210506**.cpp
- ✓ Python sample : **SGLIsample_20210506**.py

Note-1: NO error handling or optimization is considered for an easier understanding. User must modify the code.
Python performance is low, especially. (Windows 10 Pro 64bit, i7 1.99GHz, 16GB memory, about 13minute per scene

Note-2: Additional consideration is necessary for polar region and 180deg longitude region.

C

☐ Map projection

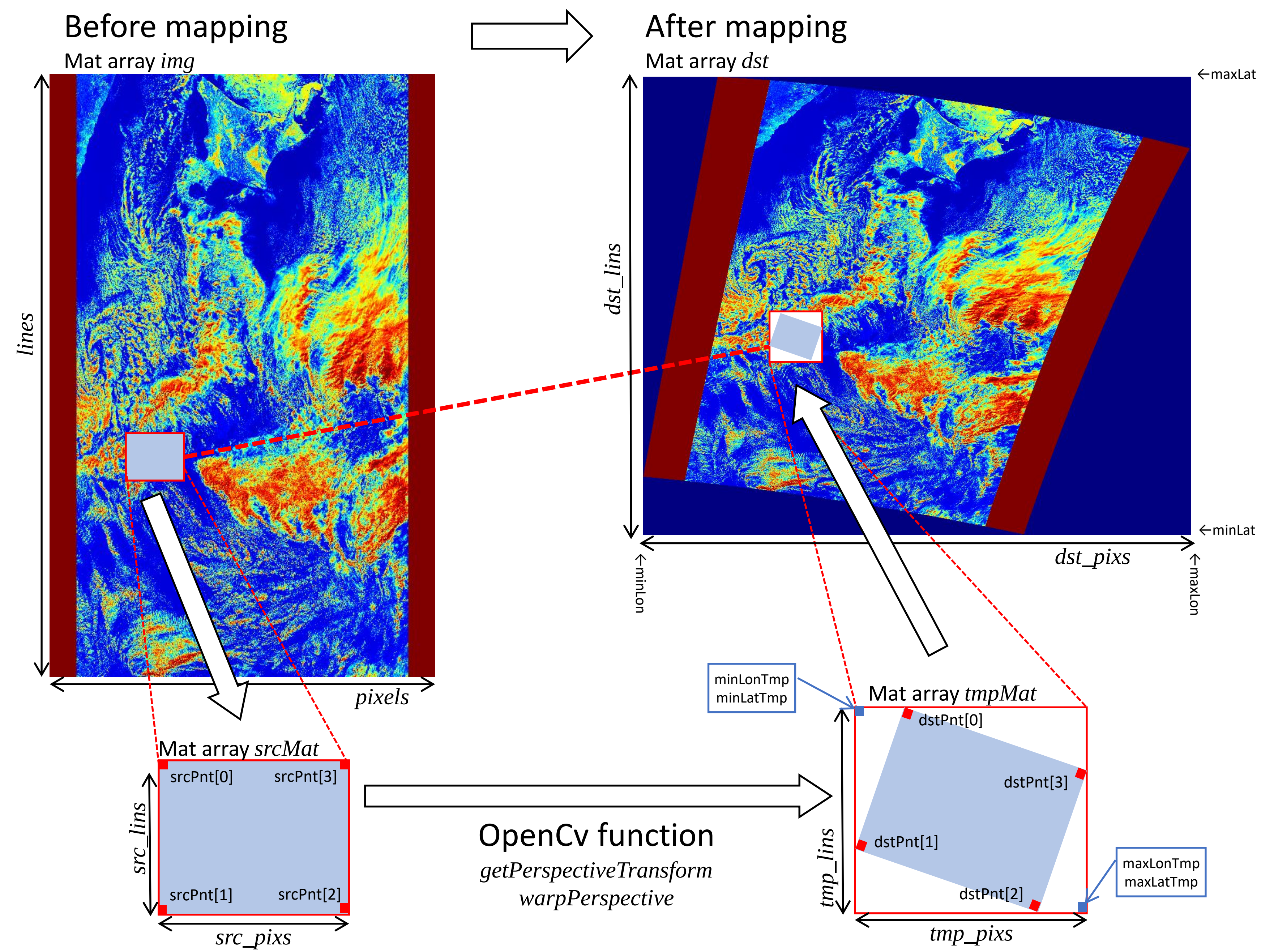
- ✓ OpenCv warpPerspective function is used. (see page 3)

☐ 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/

A

Map projection method using OpenCv function



Sample Python Source (1/2)

☐Conditions

```
import sys
import os
import datetime
import numpy as np
import h5py
import cv2

Proc1st=datetime.datetime.now()
print("¥n¥n", " ... SGLIsample from ",Proc1st)

#  --- Target Info ---
filename='GC1SG1_202104100117C04810_1BSG_VNRDQ_2002.h5'
imggrp='Image_data'
sdname='Lt_VN05'
l1b_png=sdname+'.png'
map_png=sdname+'_MAP.png'
map_tif=sdname+'_MAP.tif'
max_ref=0.2
pngratio=0.001  # png resize ratio
degStep=0.002   # 0.002deg grid

print('target hdf :', filename)
print('target sd  :', imggrp, sdname)
```

Target file name

☐hdf5 read (1/2)

```
#  open hdf / get sd
h5_file=h5py.File(filename, 'r')
sd=h5_file[imggrp][sdname]
(lines, pixels)=sd.shape

# Attribute
Offset=sd.attrs['Offset']
Slope=sd.attrs['Slope']
MaxDN=sd.attrs['Maximum_valid_DN']
MinDN=sd.attrs['Minimum_valid_DN']
Mask=sd.attrs['Mask']
TOA=sd.attrs['Band_weighted_TOA_solar_irradiance']

#  lat, lon
lat=h5_file['Geometry_data/' + 'Latitude']
lon=h5_file['Geometry_data/' + 'Longitude']
(latlon_lines, latlon_pixels) = lat.shape
respl=10    # resampling = 10
```

This is for L1B products
In case for other products, please refer to their attributes.

☐hdf5 read (2/2)

```
print('pixels={:}, lines={:}'.format(pixels, lines))
print('lat, lon : pixels={:}, lines={:}'.format(latlon_pixels, latlon_lines))
print('Slope={:}/Offset={:}/MinDN={:}/MaxDN={:}/TOA={:}'.format(Slope,Offset,MinDN,MaxDN
,TOA))

#  engineering conversion
sd=sd[:] & Mask
img=sd.astype('float32')
img[np.where(img>MaxDN)]=np.nan
img[np.where(img<MinDN)]=np.nan
img=(img*Slope+Offset)/TOA
del sd

# save png before projection
dsp=img.reshape(lines,pixels,1)/max_ref*255.0
dsp=cv2.applyColorMap(dsp.astype('uint8'),cv2.COLORMAP_JET)
dsize=(int(lines*pngratio), int(pixels*pngratio))
cv2.resize(dsp,dsize)
cv2.imwrite(l1b_png,dsp)
del dsp

#  lat, lon range -> projection Mat
print('lat.shape,lat.dtype : ',lat.shape,lat.dtype)
(minLat, maxLat) = (np.nanmin(lat), np.nanmax(lat))
(minLon, maxLon) = (np.nanmin(lon), np.nanmax(lon))
(dst_lins,dst_pixs)=(int((maxLat-minLat)/degStep+0.5), int((maxLon-minLon)/degStep+0.5))

print('proj. step : {:} [pix/deg]'.format(degStep))
print('proj. lon range : ({:.2f}:{:.2f}) -> {:} pixs'.format(minLon,maxLon,dst_pixs))
print('proj. lat range : ({:.2f}:{:.2f}) -> {:} lins'.format(minLat,maxLat,dst_lins))
```

Sample Python Source(2/2)

☐ Map projection (1/2)

```
# destination Mat
dst=np.empty((dst_lins,dst_pixs),dtype=np.float32)*np.nan

# projection Loop : 透視変換（射影変換）
print("¥n... start of projection", end='')
for lin in range(0,lines-respl-2,respl):
    if np.mod(lin, 500)==0:
        print('¥nline-{:04d}/{:04d} '.format(lin,lines), end='')
    for pix in range(0,pixels-respl-2,respl):

        # source ULLR
        srcPnt=[]
        srcPnt.append([pix,      lin      ])  # UL
        srcPnt.append([pix,      lin+respl])  # LL
        srcPnt.append([pix+respl,lin+respl])  # LR
        srcPnt.append([pix+respl,lin      ])  # UR
        srcMat=img[lin:lin+respl+2, pix:pix+respl+2]   # underlap consideration

        # desination ULLR
        (pix0,lin0)=(int(pix/respl),int(lin/respl))
        (pix1,lin1)=(pix0+1,lin0+1)
        if pix1>=latlon_pixels or lin1>=latlon_lines: continue

        (ULlon,ULlat)=(lon[lin0,pix0], lat[lin0,pix0]) # UL
        (LLlon,LLlat)=(lon[lin1,pix0], lat[lin1,pix0]) # UL
        (LRlon,LRlat)=(lon[lin1,pix1], lat[lin1,pix1]) # UL
        (URlon,URlat)=(lon[lin0,pix1], lat[lin0,pix1]) # UL

        [ULlon,LLlon,LRlon,URlon]=([ULlon,LLlon,LRlon,URlon]-minLon)/degStep
        [ULlat,LLlat,LRlat,URlat]=(maxLat-[ULlat,LLlat,LRlat,URlat])/degStep

        dstPnt=[]
        dstPnt.append([ULlon , ULlat]) # UL(lon,lat)
        dstPnt.append([LLlon , LLlat]) # LL(lon,lat)
        dstPnt.append([LRlon , LRlat]) # LR(lon,lat)
        dstPnt.append([URlon , URlat]) # UR(lon,lat)

        # destination size
        maxLonTmp=np.max([ULlon, LLlon, LRlon, URlon])
        minLonTmp=np.min([ULlon, LLlon, LRlon, URlon])
        maxLatTmp=np.max([ULlat, LLlat, LRlat, URlat])
        minLatTmp=np.min([ULlat, LLlat, LRlat, URlat])
        tmp_pixs=int(maxLonTmp-minLonTmp+0.5)+2            # underlap consideration
        tmp_lins=int(maxLatTmp-minLatTmp+0.5)+2
```

B

Additional assumption is necessary for the edge region processing.

☐Map projection (2/2)

```
        # relative addressing & projection matrix
        srcPnt[0][0]=srcPnt[0][0]-0.5
        srcPnt[0][1]=srcPnt[0][1]-0.5
        for i in range(3,0,-1):
            srcPnt[i]=[srcPnt[i][0]-srcPnt[0][0], srcPnt[i][1]-srcPnt[0][1]]
            dstPnt[i]=[dstPnt[i][0]-minLonTmp,    dstPnt[i][1]-minLatTmp   ]
        srcPnt[0]=[0,0]
        dstPnt[0]=[0,0]
        srcPnt=np.array(srcPnt,dtype=np.float32)
        dstPnt=np.array(dstPnt,dtype=np.float32)
        H=cv2.getPerspectiveTransform(srcPnt,dstPnt)

        # projection
        tmpMat=np.zeros((tmp_lins,tmp_pixs)).astype('float32')
        tmpMat[:]=np.nan
        dsize=(tmp_pixs, tmp_lins)
        tmpMat=cv2.warpPerspective(srcMat,H,dsize,dst=tmpMat,
                              borderMode=cv2.BORDER_TRANSPARENT)
        # merge
        for tmp_lin in range(tmp_lins):
            dst_lin=tmp_lin+int(minLatTmp+0.5)
            for tmp_pix in range(tmp_pixs):
                dst_pix=tmp_pix+int(minLonTmp+0.5)
                if np.isnan(dst[dst_lin,dst_pix]):
                    dst[dst_lin,dst_pix]=tmpMat[tmp_lin,tmp_pix]

        # prepartaion for next loop
        del tmpMat
        del srcMat
        if pix==0: print('.', end=‘’)
h5_file.close()
print("¥n")

# save project png
dsp=dst/max_ref*255.0
dsp=cv2.applyColorMap(dsp.astype('uint8'),cv2.COLORMAP_JET)
dsize=(int(dst_lins*pngratio), int(dst_pixs*pngratio))
cv2.resize(dsp,dsize)
cv2.imwrite(map_png,dsp)
del dsp
print("¥n... end of map png output")

# --- end of process ---
del img
```

C

getPerspectiveTransform

warpPerspective

BiLinear interpolation

Sample C++ Source(1/3)

☐Conditions, hdf read (1/3)

Target File name

```
string filename = "GC1SG1_202104100117C04810_1BSG_VNRDQ_2002.h5";
string imggrp = "/Image_data";
string sdname = "Lt_VN05";
string l1b_png = "..¥¥" + sdname + "_L1B.png";
string map_png = "..¥¥" + sdname + "_MAP.png";
float max_ref = 0.2f;           // max reflectance
float pngratio = 0.1f;          // png resize ratio
float32    degStep = 0.002f;    // 0.002deg grid
printf("target hdf : %s¥n", filename.c_str());
printf("target sd  : %s %s¥n", imggrp.c_str(), sdname.c_str());
```

```
// HDF/Group/SD open
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);     // SD Mat array(uint16)
herr_t status = H5Dread(sdid, H5Dget_type(sdid), H5S_ALL, H5S_ALL, H5P_DEFAULT, dn.ptr());

// Attribute
float32 Offset, Slope, TOA;
uint16     Mask, MaxDN, MinDN;

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

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

aid = H5Aopen_name(sdid, "Band_weighted_TOA_solar_irradiance");
status = H5Aread(aid, H5Aget_type(aid), &TOA);
status = H5Aclose(aid);

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

☐hdf5 read (2/3)

```
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);
status = H5Dclose(sdid);
status = H5Gclose(gid);

// lat, lon
gid = H5Gopen(fid, "/Geometry_data", H5P_DEFAULT);
sdid = H5Dopen(gid, "Latitude", H5P_DEFAULT);

n_dims = H5Sget_simple_extent_dims(H5Dget_space(sdid), dims, NULL);
long latlon_lines = (long)dims[0];          // resampled data
long latlon_pixels = (long)dims[1];
long respl = 10;                            // resample intervarl = 10

cv::Mat lat(latlon_lines, latlon_pixels, CV_32FC1);
status = H5Dread(sdid, H5Dget_type(sdid), H5S_ALL, H5S_ALL, H5P_DEFAULT, lat.ptr());

sdid = H5Dopen(gid, "Longitude", H5P_DEFAULT);
cv::Mat lon(latlon_lines, latlon_pixels, CV_32FC1);
status = H5Dread(sdid, H5Dget_type(sdid), H5S_ALL, H5S_ALL, H5P_DEFAULT, lon.ptr());

status = H5Dclose(sdid);
status = H5Gclose(gid);
status = H5Fclose(fid);

printf("pixels=%ld, lines=%ld¥n", pixels, lines);
printf("lat, lon : pixels=%ld, lines=%ld¥n", latlon_pixels, latlon_lines);
printf("attribute : Slope=%f/Offset=%f/Mask=%04X/MinDN=%d/MaxDN=%d¥n",
                    Slope, Offset, Mask, MinDN, MaxDN);

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

for (long idx = 0; idx < lines * pixels; idx++) {
            uint16 d = uptr[idx] & Mask;
            ptr[idx] = (d >= MinDN && d <= MaxDN) ? ((float32)d * Slope + Offset) / TOA : q_nan;
}
dn.release();
```

Sample C++ Source(2/3)

☐hdf5 read (3/3)

```
//  mat display
cv::Mat dsp = img.clone() * 255.0 / max_ref;        // value range
dsp.convertTo(dsp, CV_8U);
applyColorMap(dsp, dsp, cv::COLORMAP_JET);
cv::resize(dsp, dsp, cv::Size(), pngratio, pngratio);
imwrite(l1b_png.c_str(), dsp);
cv::imshow(sdname.c_str(), dsp);
cv::waitKey(0);
dsp.release();
```

☐ Map projection(1/4)

```
//  lat, lon range -> proj. Mat
double minLat, maxLat, minLon, maxLon;
cv::minMaxLoc(lon, &minLon, &maxLon);
cv::minMaxLoc(lat, &minLat, &maxLat);
minLat = (double) (long)(minLat / degStep + 0.5f) * degStep;
maxLat = (double) (long)(maxLat / degStep + 0.5f) * degStep;
minLon = (double) (long)(minLon / degStep + 0.5f) * degStep;
maxLon = (double) (long)(maxLon / degStep + 0.5f) * degStep;

long dst_pixs = (long)((maxLon - minLon) / degStep) + 1;
long dst_lins = (long)((maxLat - minLat) / degStep) + 1;

printf("proj step : %f [pix/deg]¥n", degStep);
printf("proj lon range : (%f:%f) -> %ld pixs¥n", minLon, maxLon, dst_pixs);
printf("proj lat range : (%f:%f) -> %ld lins¥n", minLat, maxLat, dst_lins);

//  proj array pointer
cv::Mat dst(cv::Size(dst_pixs, dst_lins), CV_32FC1, q_nan);
float32* dstPtr = (float32*)dst.ptr();
long dstStep = (long)dst.step / sizeof(float32);

float32* lonPtr = (float32*)lon.ptr();
float32* latPtr = (float32*)lat.ptr();
long latlonStep = (long)lon.step / sizeof(float32);
```

☐ Map projection(2/4)

```
//  map projection loop
printf("... start of projection¥n");
for (float lin = 0; lin + (float)respl <= (float)lines - 3; lin += (float)respl) {
        for (float pix = 0; pix + (float)respl <= (float)pixels - 3; pix += (float)respl) {    B

                //  Source Array position
                cv::Point2f srcPnt[] =
                {       cv::Point2f(pix,        lin),                   // UL
                        cv::Point2f(pix,        lin + respl),           // LL
                        cv::Point2f(pix + respl, lin + respl),          // LR
                        cv::Point2f(pix + respl, lin)                   // UR       };
                cv::Rect srcRect = cv::Rect(
                        srcPnt[0], cv::Point2f(pix+respl+2, lin+respl+2));
                cv::Mat srcMat = img(srcRect).clone();

                //  Destination Array position
                long idxUL = (long)(lin / respl) * latlonStep + (long)(pix / respl);
                long idxLL = (long)(lin / respl + 1) * latlonStep + (long)(pix / respl);
                long idxLR = (long)(lin / respl + 1) * latlonStep + (long)(pix / respl + 1);
                long idxUR = (long)(lin / respl) * latlonStep + (long)(pix / respl + 1);

                float32 lonUL = (lonPtr[idxUL] - (float)minLon) / (float)degStep;
                float32 lonLL = (lonPtr[idxLL] - (float)minLon) / (float)degStep;
                float32 lonLR = (lonPtr[idxLR] - (float)minLon) / (float)degStep;
                float32 lonUR = (lonPtr[idxUR] - (float)minLon) / (float)degStep;

                float32 latUL = ((float)maxLat - latPtr[idxUL]) / (float)degStep;
                float32 latLL = ((float)maxLat - latPtr[idxLL]) / (float)degStep;
                float32 latLR = ((float)maxLat - latPtr[idxLR]) / (float)degStep;
                float32 latUR = ((float)maxLat - latPtr[idxUR]) / (float)degStep;

                cv::Point2f dstPnt[] =
                {       cv::Point2f(lonUL, latUL),
                        cv::Point2f(lonLL, latLL),
                        cv::Point2f(lonLR, latLR),
                        cv::Point2f(lonUR, latUR)       };
                float32 maxLonTmp = max(max(lonUL, lonLL), max(lonLR, lonUR));
                float32 minLonTmp = min(min(lonUL, lonLL), min(lonLR, lonUR));
                float32 maxLatTmp = max(max(latUL, latLL), max(latLR, latUR));
                float32 minLatTmp = min(min(latUL, latLL), min(latLR, latUR));
                long tmp_pixs = (long)(maxLonTmp - minLonTmp) +2;
                long tmp_lins = (long)(maxLatTmp - minLatTmp) +2;
```

to prevent underlap

to prevent underlap

Sample C++ Source(3/3)

☐ Map projection(3/4)

```
                    //  matrix for warp calc
                    cv::Point2f srcBase = srcPnt[0] = srcPnt[0] - cv::Point2f(0.5F, 0.5F);
                                                            // POA is necessary for UL
                    cv::Point2f srcPntH[] =
                    {          srcPnt[0] - srcBase,           // UL
                               srcPnt[1] - srcBase,           // LL
                               srcPnt[2] - srcBase,           // LR
                               srcPnt[3] - srcBase            // UR      };
                    cv::Point2f dstPntH[] =
                    {          dstPnt[0] - cv::Point2f(minLonTmp, minLatTmp),  // UL
                               dstPnt[1] - cv::Point2f(minLonTmp, minLatTmp),  // LL
                               dstPnt[2] - cv::Point2f(minLonTmp, minLatTmp),  // LR
                               dstPnt[3] - cv::Point2f(minLonTmp, minLatTmp)   // UR       };
                    cv::Mat H = cv::getPerspectiveTransform(srcPntH, dstPntH);

                    //  warp!
                    cv::Mat tmpMat(cv::Size(tmp_pixs, tmp_lins), CV_32FC1, q_nan);
                    float32* tmpPtr = (float32*)tmpMat.ptr();
                    long tmpStep = (long)tmpMat.step / sizeof(float32);

                    cv::warpPerspective(
                               srcMat,
                               tmpMat,
                               H,
                               tmpMat.size(),
                               cv::INTER_LINEAR,
                               cv::BORDER_TRANSPARENT);

                    //  data merge
                    for (long tmp_lin = 0; tmp_lin < tmp_lins; tmp_lin++)
                               for (long tmp_pix = 0; tmp_pix < tmp_pixs; tmp_pix++) {
                                          long idx = (tmp_lin + (long) minLatTmp) * dstStep
                                                     + tmp_pix + (long)minLonTmp;
                                          long tmp_idx = tmp_lin * tmpStep + (long)tmp_pix;
                                          if (!std::isnan(tmpPtr[tmp_idx]))
                                                     dstPtr[idx] = tmpPtr[tmp_idx];
                               }
                    tmpMat.release();
          }
}
```

C

Calc of warp matrix
getPerspectiveTransform

Warp!
warpPerspective

BiLinear

☐Map projection(4/4)

```
// map display
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();
```