# Step1\_swath2tile.py

from pykdtree.kdtree import KDTree  
from osgeo import gdal  
import numpy as np  
import h5py  
import os  
  
from multiprocessing import Pool  
from functools import partial  
from datetime import datetime  
from pathlib import Path  
from sys import argv  
  
def read\_data\_sds(file\_path, data\_vars, pre\_process\_func=lambda x: x):  
 var\_name\_map = {'Latitude': '0','Longitude': '1','SLDR': '2','SLUR': '3','SLNR': '4','QC\_flag': '5'}  
 data\_vars = var\_name\_map.get(data\_vars)  
 if (dst:=gdal.Open(str(file\_path), gdal.GA\_ReadOnly)):  
 if (sds\_text:=dst.GetSubDatasets()):  
 sds\_dic = dict(zip([i[0][i[0].rindex(':')+1:] for i in sds\_text], [i[0] for i in sds\_text]))  
 if data\_vars in sds\_dic.keys():  
 t\_var\_path = sds\_dic[data\_vars]  
 t\_var\_dst = gdal.Open(t\_var\_path, gdal.GA\_ReadOnly)  
 print(datetime.now(), f'Read {file\_path} {data\_vars}')  
 return pre\_process\_func(t\_var\_dst.ReadAsArray())  
  
  
def lon\_lat\_to\_sinusoid(lat, lon):  
 SPHERE = 6371007.181000  
 lon\_rad = np.deg2rad(lon)  
 lat\_rad = np.deg2rad(lat)  
 sinusoid\_x = SPHERE \* lon\_rad \* np.cos(lat\_rad)  
 sinusoid\_y = SPHERE \* lat\_rad  
 return sinusoid\_x, sinusoid\_y  
  
  
def swath2tile(swath\_file\_path:Path, export\_folder:Path, k=4):  
 h\_base = -20015109.353999998000  
 v\_base = 10007554.677000344972  
 tile\_size = 1111950.5197665233  
 min\_pixel\_per\_tile = 1200\*120 # tile\_size^2 \*10%， 该处阈值要设置小一点，不然边角有确实无值。  
 max\_distance = 926\*2.5  
  
 lat = read\_data\_sds(swath\_file\_path, 'Latitude', lambda x: np.where(x!=-999, x, np.nan))  
 lon = read\_data\_sds(swath\_file\_path, 'Longitude', lambda x: np.where(x!=-999, x, np.nan))  
 data\_dict = {k:read\_data\_sds(swath\_file\_path, k, lambda x: np.where(x!=999, (x\*0.05)-1000, np.nan))   
 for k in ('SLDR', 'SLUR', 'SLNR')}  
 location\_mask = np.isfinite(lat) & np.isfinite(lon)   
 sinusoid\_x, sinusoid\_y =lon\_lat\_to\_sinusoid(lat, lon)  
 h = (sinusoid\_x - h\_base) // tile\_size  
 v = (sinusoid\_y - v\_base) // -tile\_size  
 tile\_int, tile\_pixel\_count = np.unique(h\*100 + v, return\_counts=True)  
 tile\_int = tile\_int.astype(np.uint32)  
 tile\_int = tile\_int[tile\_pixel\_count>min\_pixel\_per\_tile]  
  
 for t\_var\_name, t\_var\_data in data\_dict.items():  
 t\_vaild\_mask = np.isfinite(t\_var\_data) & location\_mask  
 t\_x, t\_y, t\_data = sinusoid\_x[t\_vaild\_mask], sinusoid\_y[t\_vaild\_mask], t\_var\_data[t\_vaild\_mask],   
 t\_tree = KDTree(np.c\_[t\_x, t\_y])  
 for t\_tile in tile\_int:  
 h, v = t\_tile//100, t\_tile%100  
 tile\_str = f'h{h:02d}v{v:02d}'   
 t\_x = np.linspace(h\_base+(h\*tile\_size), h\_base+((h+1)\*tile\_size), 1200).astype(np.float32)  
 t\_y = np.linspace(v\_base-(v\*tile\_size), v\_base-((v+1)\*tile\_size), 1200).astype(np.float32)  
 xv, yv = np.meshgrid(t\_x, t\_y)  
   
 distence, idx = t\_tree.query(np.c\_[xv.ravel(), yv.ravel()], k=4)   
 distence = np.where( distence < max\_distance , distence , np.nan)  
 t\_sum = np.nansum(distence, axis=1)  
 distence /= t\_sum[:, np.newaxis]  
 t\_result = np.nansum(t\_data[idx]\*distence, axis=1)  
 t\_result = np.where(t\_sum==0, np.nan, t\_result).reshape((1200, 1200))  
  
 #distence, idx = t\_tree.query(np.c\_[xv.ravel(), yv.ravel()], k=1)  
 #t\_result = t\_data[idx]  
 #t\_result = t\_result.reshape((1200, 1200))  
  
 t\_result\_path = export\_folder / f'{swath\_file\_path.stem}\_{tile\_str}.h5'  
 with h5py.File(str(t\_result\_path),'a') as f:  
 f.create\_dataset(t\_var\_name, data=t\_result.astype(np.float32), compression="gzip", compression\_opts=5)  
 print(datetime.now(), f'write {t\_result\_path} {t\_var\_name}')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_folder = Path(argv[1]) #Path(r'/share/users/wangqian/GLASS\_Code/GLASS\_LW\_daily/test\_data/2011\_Aqua/113')  
 export\_folder = Path(argv[2]) #Path(r'/share/users/wangqian/GLASS\_Code/GLASS\_LW\_daily/t\_result')  
  
 #input\_folder = Path(r'J:\daily\_check\Step1\terra\2022\001')  
 #export\_folder = Path(r'J:\daily\_check\Step1\result\terra\2022\001')  
 export\_folder.mkdir(parents=True, exist\_ok=True)  
  
 with Pool(1) as pool:  
 pool.map(partial(swath2tile, export\_folder=export\_folder), input\_folder.rglob(r'\*.hdf'))

# Step2\_instant2daily.py

from osgeo import gdal, gdal\_array  
from netCDF4 import Dataset  
import numba as nb  
import numpy as np  
import h5py  
  
from sys import argv  
from pathlib import Path  
from datetime import datetime, timedelta  
import warnings  
warnings.filterwarnings("ignore", category=RuntimeWarning)   
DEBUG=True  
  
  
def read\_ear5(file\_path, var\_name):  
 era5\_dst = Dataset(file\_path)  
 data = np.roll(era5\_dst.variables[var\_name][:].filled(0), 720, axis=2) / 3600  
 print(f'reading {file\_path} {var\_name}')  
 return data  
  
  
def label\_path(step1\_path):  
 name\_part = step1\_path.stem.split('\_')[0].split('.')  
 return datetime.strptime(f'{name\_part[3]}.{name\_part[4]}', '%Y%j.%H%M')  
  
  
def read\_h5(file\_path, var\_name):  
 with h5py.File(str(file\_path),'r') as f:  
 print(datetime.now(), f'Read {file\_path} {var\_name}')  
 return f[var\_name][:]  
  
  
def tile\_sinusoid(tile\_str):  
 h\_base = -20015109.353999998000  
 v\_base = 10007554.677000344972  
 tile\_size = 1111950.5197665233  
 h, v = int(tile\_str[1:3]), int(tile\_str[4:6])  
 t\_x = np.linspace(h\_base+(h\*tile\_size), h\_base+((h+1)\*tile\_size), 1200).astype(np.float32)  
 t\_y = np.linspace(v\_base-(v\*tile\_size), v\_base-((v+1)\*tile\_size), 1200).astype(np.float32)  
 xv, yv = np.meshgrid(t\_x, t\_y)  
 return xv, yv  
  
  
def sinusoid\_to\_lonlat(sinusoid\_x, sinusoid\_y):  
 SPHERE = 6371007.181000  
 lat\_rad = sinusoid\_y / SPHERE  
 lon\_rad = sinusoid\_x / SPHERE / np.cos(lat\_rad)  
 lat, lon = np.rad2deg(lat\_rad), np.rad2deg(lon\_rad)  
 lat[np.abs(lat)>90] = np.nan  
 lon[np.abs(lon)>180] = np.nan  
 return lon, lat  
  
  
def read\_era5\_utc(era5\_net\_folder, era5\_downward\_folder, date):  
 era5\_net\_filelist = {datetime.strptime(t\_path.stem.split('\_')[-1], '%Y%m%d'): t\_path for t\_path in era5\_net\_folder.rglob(f'\*.nc')}  
 net\_data = np.vstack([read\_ear5(era5\_net\_filelist.get(t\_date), 'str') for t\_date in (date-timedelta(days=1), date, date+timedelta(days=1))])  
 era5\_downward\_filelist = {datetime.strptime(t\_path.stem.split('\_')[-1], '%Y%m%d'): t\_path for t\_path in era5\_downward\_folder.rglob(f'\*.nc')}  
 downward\_data = np.vstack([read\_ear5(era5\_downward\_filelist.get(t\_date), 'strd') for t\_date in (date-timedelta(days=1), date, date+timedelta(days=1))])  
 upward\_data = downward\_data - net\_data  
 return {'SLUR' : upward\_data, 'SLDR':downward\_data, 'SLNR': net\_data}  
  
  
  
def warp\_utc\_tile(utc\_data, utc\_time, tile\_str):  
 in\_proj = r'+proj=longlat +ellps=clrk66 +no\_defs'  
 out\_proj = r'+proj=sinu +lon\_0=0 +x\_0=0 +y\_0=0 +R=6371007.181 +units=m +no\_defs'  
 in\_trans = [-180, 0.25, 0.0, 90.0, 0.0, -0.25]  
 m\_data = -9999.  
 tile\_size = 1200  
 m\_dtype = gdal\_array.NumericTypeCodeToGDALTypeCode(utc\_data.dtype)  
 outDrv = gdal.GetDriverByName('MEM')  
 outDst = outDrv.Create('', utc\_data.shape[2], utc\_data.shape[1], utc\_data.shape[0], m\_dtype)  
 outDst.SetGeoTransform(in\_trans)  
 outDst.SetProjection(in\_proj)  
 for t\_band\_num in range(utc\_data.shape[0]):  
 oband = outDst.GetRasterBand(t\_band\_num+1)  
 oband.WriteArray(np.nan\_to\_num(utc\_data[t\_band\_num], nan=m\_data))  
 oband.SetNoDataValue(m\_data)  
 sin\_x, sin\_y = tile\_sinusoid(tile\_str)  
 lon, \_ = sinusoid\_to\_lonlat(sin\_x, sin\_y)  
 sin\_x\_min, sin\_x\_max = sin\_x.min(), sin\_x.max()  
 sin\_y\_min, sin\_y\_max = sin\_y.min(), sin\_y.max()  
 warped\_dst = gdal.Warp('', outDst, format="MEM", width=tile\_size, height=tile\_size, dstSRS=out\_proj, outputBounds=(sin\_x\_min, sin\_y\_min, sin\_x\_max, sin\_y\_max))  
 data\_utc\_tile = warped\_dst.ReadAsArray()  
 data\_utc\_tile[data\_utc\_tile==m\_data] = np.nan  
 local\_time\_mask = np.arange(72).repeat(tile\_size\*tile\_size).reshape((-1,tile\_size, tile\_size)) +np.around(lon / 15).astype(np.int8)  
 local\_time\_mask = (local\_time\_mask>23) & (local\_time\_mask<48)  
 #data\_localtime = data\_utc\_tile[local\_time\_mask].reshape((-1,tile\_size, tile\_size)).transpose(1,2,0)  
 data\_localtime = np.zeros((data\_utc\_tile.shape[0]//3, tile\_size, tile\_size))  
 for i in range(data\_utc\_tile.shape[1]):  
 for j in range(data\_utc\_tile.shape[2]):  
 data\_localtime[:,i,j] = data\_utc\_tile[:,i,j][local\_time\_mask[:,i,j]]  
 data\_localtime = data\_localtime.transpose(1,2,0)  
 data\_match\_modis = data\_utc\_tile[(utc\_time.reshape((-1))//3600)+24, :, :].transpose(1,2,0) if utc\_time.shape[2]>0 else np.nan  
 return data\_localtime, data\_match\_modis, np.nanmean(data\_localtime, axis=2)  
  
  
def sunRiseSet\_array(lat, lon, nDays):  
 UTCoff = (np.around(lon / 15).astype(np.int16))  
 lat\_rad = np.deg2rad(lat)  
 B = np.deg2rad(360\*(nDays - 81)/365)  
 E = 9.87\*np.sin(2\*B) - 7.53\*np.cos(B) - 1.5\*np.sin(B)  
 solarCorr = 4\*(lon - 15\*UTCoff) + E  
   
 delta = np.arcsin(np.sin(np.deg2rad(23.45))\*np.sin(np.deg2rad(360\*(nDays - 81)/365)))  
 T = np.rad2deg(np.emath.arccos(-np.tan(lat\_rad)\*np.tan(delta)).real)/15 - solarCorr/60  
 sr = np.clip(12 - T, 0, 24)  
 ss = np.clip(12 + T, 0, 24)  
 return sr\*3600, ss\*3600  
  
  
def \_instant2daily(lw\_instant, local\_time, sunrise, sunset, mode):  
 @nb.njit()  
 def linear\_interpolation(instant\_data, instant\_time):  
 array\_size = instant\_data.shape  
 integrate\_result = np.full((array\_size[0], array\_size[1]), np.nan)  
 for i in range(array\_size[0]):  
 for j in range(array\_size[1]):  
 vaild\_mask = np.isfinite(instant\_data[i, j])  
 vaild\_data = instant\_data[i, j][vaild\_mask]  
 vaild\_time = instant\_time[i, j][vaild\_mask]  
 data\_diff = np.diff(vaild\_data)  
 time\_diff = np.diff(vaild\_time)  
 k = data\_diff / time\_diff  
 b = (vaild\_data[:-1]\*vaild\_time[1:] - vaild\_data[1:]\*vaild\_time[:-1]) / time\_diff  
 integrate\_result[i,j] = np.sum((k\*0.5\*(vaild\_time[1:]\*\*2) + b\*vaild\_time[1:])-((k\*0.5\*(vaild\_time[:-1]\*\*2) + b\*vaild\_time[:-1])))  
 return integrate\_result  
   
 day\_length = sunset - sunrise  
 vaild\_mask = np.isfinite(lw\_instant)  
 day\_mask = (local\_time >= sunrise[:,:,np.newaxis]) & (local\_time <= sunset[:,:,np.newaxis])   
 day\_vaild\_mask = day\_mask & vaild\_mask  
 qc\_daily = day\_vaild\_mask.sum(axis=2) << 8  
 night\_mask = (~day\_vaild\_mask) & vaild\_mask  
 qc\_daily |= night\_mask.sum(axis=2) << 4  
 day\_data = np.where(day\_vaild\_mask, lw\_instant, np.nan)  
 day\_min = np.nanmin(day\_data, axis=2)  
 night\_data = np.where(night\_mask, lw\_instant, np.nan)  
 polar\_day\_mask = (day\_vaild\_mask.sum(axis=2) == vaild\_mask.sum(axis=2)) & (day\_length>24\*60\*60\*0.95)  
 qc\_daily[polar\_day\_mask] |= 0b0100  
  
 night\_max = np.where(polar\_day\_mask, 0., np.nanmax(night\_data, axis=2))  
 night\_mean = np.where(polar\_day\_mask, 0., np.nanmean(night\_data, axis=2))  
 combin\_day\_data = np.dstack((night\_mean, day\_data, night\_mean))  
 vaild\_day\_count = np.isfinite(combin\_day\_data).sum(axis=-1)  
 combin\_day\_time = np.dstack((sunrise, np.where(day\_vaild\_mask, local\_time, np.nan), sunset))  
  
 combin\_day\_time\_sine = np.sin((combin\_day\_time - sunrise[:,:,np.newaxis])\*np.pi / day\_length[:,:,np.newaxis])  
 lw\_day\_diff = combin\_day\_data - night\_mean[:,:,np.newaxis]  
 a = np.nanmean(combin\_day\_time\_sine \* lw\_day\_diff, axis=2) / np.nanmean(combin\_day\_time\_sine\*\*2, axis=2)  
 lw\_daytime\_sine = ((2\*a + np.pi\*night\_mean)\*(sunset - sunrise))/np.pi  
 sine\_mask = (day\_min < night\_max) if mode=='SLNR' else (day\_min > night\_max)  
 sine\_mask &= (vaild\_day\_count>3)  
 qc\_daily[sine\_mask] |= 0b01  
   
 linear\_mask = (sunrise>sunset)| ((vaild\_day\_count>=3) & (~sine\_mask))  
 lw\_daytime\_linear = linear\_interpolation(combin\_day\_data, combin\_day\_time)  
 linear\_mask |= np.abs((lw\_daytime\_linear - lw\_daytime\_sine)/lw\_daytime\_linear) > 0.05  
 linear\_mask |= np.isnan(lw\_daytime\_sine)  
 qc\_daily[linear\_mask] |= 0b10  
 lw\_daytime = np.where(linear\_mask, lw\_daytime\_linear, lw\_daytime\_sine)  
 lw\_daily = (lw\_daytime+(sunrise+(24\*3600-sunset))\*night\_mean)/(24\*3600)  
   
 polar\_night\_mask = (day\_length <=0)  
 lw\_daily = np.where(polar\_night\_mask, night\_mean, lw\_daily)  
 qc\_daily[polar\_night\_mask] |= 0b1010  
 return lw\_daily, qc\_daily, lw\_daytime\_sine/(sunset-sunrise), lw\_daytime\_linear/(sunset-sunrise)  
  
  
def instant2daily(instant\_file\_list, era5\_data\_dict, result\_folder, year, doy, tile,):  
 sinusoid\_x, sinusoid\_y = tile\_sinusoid(tile)  
 lon, lat = sinusoid\_to\_lonlat(sinusoid\_x, sinusoid\_y)  
  
 vaild\_mask = (np.abs(lon)<=180) & (np.abs(lat)<=90)  
 time\_shift = (np.around(lon / 15).astype(np.int16))[:,:,np.newaxis]  
 time\_min = datetime.strptime(f'{year}{doy}', r'%Y%j') + timedelta(hours=int(time\_shift.min()))  
 time\_max = datetime.strptime(f'{year}{doy}', r'%Y%j') + timedelta(hours=int(time\_shift.max())+24)  
  
 instant\_file\_list = {k:v for k, v in instant\_file\_list.items() if time\_min <= k <= time\_max}  
 data\_time = list(sorted(instant\_file\_list.keys()))  
 utc\_array = np.array([(t\_time - datetime.strptime(f'{year}{doy}', '%Y%j')).seconds for t\_time in data\_time])[np.newaxis, np.newaxis, :]  
 local\_time = (time\_shift\*60) + utc\_array  
 sunrise, sunset = sunRiseSet\_array(lat, lon, (datetime.strptime(f'{year}{doy}', '%Y%j') - datetime(1899, 12, 30)).days)  
   
 t\_result\_path = result\_folder / year / doy / f'lw\_daily\_{year}{doy}\_{tile}.h5'  
 t\_result\_path.parent.mkdir(parents=True, exist\_ok=True)  
 if t\_result\_path.exists():  
 return   
  
 if DEBUG:  
 with h5py.File(str(t\_result\_path),'a') as f:  
 f.create\_dataset('sunrise', data=sunrise.astype(np.float32), compression="gzip", compression\_opts=5)  
 f.create\_dataset('sunset', data=sunset.astype(np.float32), compression="gzip", compression\_opts=5)  
  
 for var\_name in ('SLUR', 'SLDR', 'SLNR'):  
 try:  
 era5\_all, era5\_match\_modis, era5\_daily\_mean = warp\_utc\_tile(era5\_data\_dict.get(var\_name), utc\_array, tile)  
 if len(instant\_file\_list.items())>0:  
 data\_array = np.stack([read\_h5(instant\_file\_list[t\_time], var\_name) for t\_time in data\_time], axis=2)  
 data\_array[(local\_time<0) | (local\_time>24\*60\*60)] = np.nan  
 lw\_daily, qc\_daily, lw\_daytime\_sine, lw\_daytime\_linear = \_instant2daily(data\_array, local\_time, sunrise, sunset, var\_name)  
  
 era5\_modis\_diff = np.nan\_to\_num(np.nanmean((data\_array - era5\_match\_modis), axis=2), 0.0)  
 era5\_offseted\_mean = np.nanmean(era5\_all, axis=2) + era5\_modis\_diff  
 era5\_mask = (((qc\_daily >>8) < 2) | (((qc\_daily>>4)&0b00001111)<1))  
 lw\_daily = np.where(era5\_mask, era5\_offseted\_mean, lw\_daily)  
 qc\_daily[era5\_mask] |= 0b11  
 else:  
 lw\_daily = era5\_daily\_mean  
 qc\_daily = np.full\_like(lw\_daily, 0b11, dtype=np.uint16)  
 lw\_daily[~vaild\_mask] = np.nan  
 qc\_daily[~vaild\_mask] = 0  
 era5\_daily\_mean = era5\_offseted\_mean if len(instant\_file\_list.items())>0 else era5\_daily\_mean  
 with h5py.File(str(t\_result\_path),'a') as f:  
 f.attrs['projs'] = r'+proj=sinu +lon\_0=0 +x\_0=0 +y\_0=0 +R=6371007.181 +units=m +no\_defs'  
 f.attrs['geotrans'] = f'[{sinusoid\_x[0,0]}, 926.6254331388, 0.0, {sinusoid\_y[0,0]}, 0.0, -926.6254331388]'  
 f.create\_dataset(f'{var\_name}\_daily', data=lw\_daily.astype(np.float32), compression="gzip", compression\_opts=5)  
 f.create\_dataset(f'{var\_name}\_qc', data=qc\_daily.astype(np.uint16), compression="gzip", compression\_opts=5)  
 if DEBUG:  
 if len(instant\_file\_list.items())>0:  
 f.create\_dataset(f'{var\_name}\_sine', data=lw\_daytime\_sine.astype(np.float32), compression="gzip", compression\_opts=5)  
 f.create\_dataset(f'{var\_name}\_linear', data=lw\_daytime\_linear.astype(np.float32), compression="gzip", compression\_opts=5)  
 f.create\_dataset(f'{var\_name}\_instant', data=data\_array.astype(np.float32).transpose(2,1,0), compression="gzip", compression\_opts=5)  
 f.create\_dataset(f'{var\_name}\_instant\_time', data=local\_time.astype(np.float32).transpose(2,1,0), compression="gzip", compression\_opts=5)  
 f.create\_dataset(f'{var\_name}\_era5\_all', data=era5\_all.astype(np.float32).transpose(2,1,0), compression="gzip", compression\_opts=5)  
 f.create\_dataset(f'{var\_name}\_era5\_daily\_mean', data=era5\_daily\_mean.astype(np.float32), compression="gzip", compression\_opts=5)  
  
 print(datetime.now(), f'write {t\_result\_path} {var\_name}')  
 except Exception as e:  
 print("发生异常:", e)  
 continue  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 year = argv[1]  
 doy = argv[2]  
 instant\_folder = Path(r'/gpfs01/users/jiecheng/JC/LW\_ELITE/ELITE\_dialy/step1\_swath2tile')  
 era5\_downward\_folder = Path(r'/gpfs01/users/wangqian/2022/ERA5/reanalysis-era5-single-levels/surface\_thermal\_radiation\_downwards')  
 era5\_net\_folrer = Path(r'/gpfs01/users/wangqian/2022/ERA5/reanalysis-era5-single-levels/surface\_net\_thermal\_radiation')  
 era5\_data = read\_era5\_utc(era5\_net\_folrer, era5\_downward\_folder, datetime.strptime(f'{year}{doy}', '%Y%j'))  
 result\_folder = Path(r'/gpfs01/users/jiecheng/JC/LW\_ELITE/ELITE\_dialy/step2\_dailyresult')  
 tile\_file = r'/gpfs01/users/jiecheng/JC/LW\_ELITE/ELITE\_dialy/code/globe.tile'  
 with open(tile\_file)as f:  
 tile\_list = f.read().splitlines()  
  
 for tile in tile\_list:  
 print(tile)  
 instant\_file\_list = {label\_path(t\_path): t\_path for t\_path in instant\_folder.rglob(f'\*\_{tile}.h5')}  
 instant2daily(instant\_file\_list, era5\_data, result\_folder, year, doy, tile,)