import math

def terrain\_correction(image):

img\_geom = image.geometry()

srtm = ee.Image('USGS/SRTMGL1\_003').clip(img\_geom) # 30m SRTM

sigma0\_pow = ee.Image.constant(10).pow(image.divide(10.0))

# Radar geometry

if 'angle' in image.bandNames().getInfo():

theta\_i = image.select('angle')

else:

# Return image without terrain correction if 'angle' band is not available

return image

phi\_i = ee.Terrain.aspect(theta\_i) \

.reduceRegion(ee.Reducer.mean(), theta\_i.geometry(), 1000) \

.get('aspect')

# Terrain geometry

alpha\_s = ee.Terrain.slope(srtm).select('slope')

phi\_s = ee.Terrain.aspect(srtm).select('aspect')

# Model geometry

phi\_r = ee.Image.constant(phi\_i).subtract(phi\_s)

# Convert all to radians

phi\_r\_rad = phi\_r.multiply(math.pi / 180)

alpha\_s\_rad = alpha\_s.multiply(math.pi / 180)

theta\_i\_rad = theta\_i.multiply(math.pi / 180)

ninety\_rad = ee.Image.constant(90).multiply(math.pi / 180)

# Slope steepness in range (eq. 2)

alpha\_r = (alpha\_s\_rad.tan().multiply(phi\_r\_rad.cos())).atan()

# Slope steepness in azimuth (eq 3)

alpha\_az = (alpha\_s\_rad.tan().multiply(phi\_r\_rad.sin())).atan()

# Local incidence angle (eq. 4)

theta\_lia = (alpha\_az.cos().multiply((theta\_i\_rad.subtract(alpha\_r)).cos())).acos()

theta\_lia\_deg = theta\_lia.multiply(180 / math.pi)

# Gamma\_nought\_flat

gamma0 = sigma0\_pow.divide(theta\_i\_rad.cos())

gamma0\_dB = ee.Image.constant(10).multiply(gamma0.log10())

ratio\_1 = gamma0\_dB.select('VV').subtract(gamma0\_dB.select('VH'))

# Volumetric Model

nominator = (ninety\_rad.subtract(theta\_i\_rad).add(alpha\_r)).tan()

denominator = (ninety\_rad.subtract(theta\_i\_rad)).tan()

vol\_model = (nominator.divide(denominator)).abs()

# Apply model

gamma0\_volume = gamma0.divide(vol\_model)

gamma0\_volume\_dB = ee.Image.constant(10).multiply(gamma0\_volume.log10())

# Layover/shadow mask

alpha\_r\_deg = alpha\_r.multiply(180 / math.pi)

layover = alpha\_r\_deg.lt(theta\_i)

# Shadow where LIA > 90

shadow = theta\_lia\_deg.lt(85)

# Calculate the ratio for RGB visualization

ratio = gamma0\_volume\_dB.select('VV').subtract(gamma0\_volume\_dB.select('VH'))

output = gamma0\_volume\_dB.addBands(ratio).addBands(alpha\_r).addBands(phi\_s).addBands(theta\_i\_rad) \

.addBands(layover).addBands(shadow).addBands(gamma0\_dB).addBands(ratio\_1)

return image.addBands(

output.select(['VV', 'VH'], ['VV\_corrected', 'VH\_corrected']),

None,

True

)

**# Función de filtrado refinado de Lee**

def refined\_lee(img):

# Convertir a unidades naturales

img\_natural = to\_natural(img)

# Configurar kernels de 3x3

weights3 = ee.List.repeat(ee.List.repeat(1, 3), 3)

kernel3 = ee.Kernel.fixed(3, 3, weights3, 1, 1, False)

mean3 = img\_natural.reduceNeighborhood(ee.Reducer.mean(), kernel3)

variance3 = img\_natural.reduceNeighborhood(ee.Reducer.variance(), kernel3)

# Usar una muestra de ventanas de 3x3 dentro de ventanas de 7x7 para determinar gradientes y direcciones

sample\_weights = ee.List([

[0,0,0,0,0,0,0],

[0,1,0,1,0,1,0],

[0,0,0,0,0,0,0],

[0,1,0,1,0,1,0],

[0,0,0,0,0,0,0],

[0,1,0,1,0,1,0],

[0,0,0,0,0,0,0]

])

sample\_kernel = ee.Kernel.fixed(7, 7, sample\_weights, 3, 3, False)

sample\_mean = mean3.neighborhoodToBands(sample\_kernel)

sample\_var = variance3.neighborhoodToBands(sample\_kernel)

gradients = sample\_mean.select(1).subtract(sample\_mean.select(7)).abs()

gradients = gradients.addBands(sample\_mean.select(6).subtract(sample\_mean.select(2)).abs())

gradients = gradients.addBands(sample\_mean.select(3).subtract(sample\_mean.select(5)).abs())

gradients = gradients.addBands(sample\_mean.select(0).subtract(sample\_mean.select(8)).abs())

max\_gradient = gradients.reduce(ee.Reducer.max())

gradmask = gradients.eq(max\_gradient)

gradmask = gradmask.addBands(gradmask)

directions = sample\_mean.select(1).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(7))).multiply(1)

directions = directions.addBands(sample\_mean.select(6).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(2))).multiply(2))

directions = directions.addBands(sample\_mean.select(3).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(5))).multiply(3))

directions = directions.addBands(sample\_mean.select(0).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(8))).multiply(4))

directions = directions.addBands(directions.select(0).Not().multiply(5))

directions = directions.addBands(directions.select(1).Not().multiply(6))

directions = directions.addBands(directions.select(2).Not().multiply(7))

directions = directions.addBands(directions.select(3).Not().multiply(8))

directions = directions.updateMask(gradmask)

directions = directions.reduce(ee.Reducer.sum())

sample\_stats = sample\_var.divide(sample\_mean.multiply(sample\_mean))

sigma\_v = sample\_stats.toArray().arraySort().arraySlice(0, 0, 5).arrayReduce(ee.Reducer.mean(), [0])

rect\_weights = ee.List.repeat(ee.List.repeat(0, 7), 3).cat(ee.List.repeat(ee.List.repeat(1, 7), 4))

diag\_weights = ee.List([

[1, 0, 0, 0, 0, 0, 0],

[1, 1, 0, 0, 0, 0, 0],

[1, 1, 1, 0, 0, 0, 0],

[1, 1, 1, 1, 0, 0, 0],

[1, 1, 1, 1, 1, 0, 0],

[1, 1, 1, 1, 1, 1, 0],

[1, 1, 1, 1, 1, 1, 1]

])

rect\_kernel = ee.Kernel.fixed(7, 7, rect\_weights, 3, 3, False)

diag\_kernel = ee.Kernel.fixed(7, 7, diag\_weights, 3, 3, False)

dir\_mean = img\_natural.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel).updateMask(directions.eq(1))

dir\_var = img\_natural.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel).updateMask(directions.eq(1))

dir\_mean = dir\_mean.addBands(img\_natural.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel).updateMask(directions.eq(2)))

dir\_var = dir\_var.addBands(img\_natural.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel).updateMask(directions.eq(2)))

for i in range(1, 4):

dir\_mean = dir\_mean.addBands(img\_natural.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel.rotate(i)).updateMask(directions.eq(2 \* i + 1)))

dir\_var = dir\_var.addBands(img\_natural.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel.rotate(i)).updateMask(directions.eq(2 \* i + 1)))

dir\_mean = dir\_mean.addBands(img\_natural.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel.rotate(i)).updateMask(directions.eq(2 \* i + 2)))

dir\_var = dir\_var.addBands(img\_natural.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel.rotate(i)).updateMask(directions.eq(2 \* i + 2)))

dir\_mean = dir\_mean.reduce(ee.Reducer.sum())

dir\_var = dir\_var.reduce(ee.Reducer.sum())

var\_x = dir\_var.subtract(dir\_mean.multiply(dir\_mean).multiply(sigma\_v)).divide(sigma\_v.add(1.0))

b = var\_x.divide(dir\_var)

result = dir\_mean.add(b.multiply(img\_natural.subtract(dir\_mean)))

return to\_db(result)