Step 2.2 Masking rasters

Diane ESPEL

2025-06-18

Contents

1 Objectives

This script allows multiple rasters to be cropped using a common vector layer referred to as the vector_mask. In this case, the goal is to mask a multispectral raster, a Digital Elevation Model (DEM), and a slope raster. To streamline the process, a custom masking function (`RasterMasking()`) is defined.

When masking a raster stack, it's crucial to ensure that all bands within the stack share the same spatial resolution. To achieve this, a specific function (`is_Same_Resolution()`) is also defined

2 Script explanation

2.1 Clean environment and graphics

```
rm(list = ls())  # Clear all objects from the R environment to start fresh
graphics.off()  # Close all graphics devices (if any plots are open)
```

2.2 Load required packages

```r
library(sp)  # For spatial data compatibility
library(sf)  # For reading and manipulating shapefiles
library(terra)  # For raster and spatial operations
```

2.3 Create functions

We need to declare two functions:

- **is_Same_Resolution()** function that takes two raster objects (band1 and band2) and check if their spatial resolutions (i.e., pixel sizes) are identical. This is useful to ensure compatibility before performing operations that require matched resolutions.

  Parameters:

  - `band1`: The first raster band (e.g., blue band). -`band2`: The second raster band (e.g., NIR band).

  Returns:

  - `TRUE` if the resolutions are the same, otherwise `FALSE`."

- **Raster_Masking()** function that masks input raster (MS, Slope, or DTM) using a polygon mask and performs resolution adjustments for MS rasters. It supports MS rasters (with resolution correction) and mono-band rasters (slope or DTM). Before applying the mask, the function performs key checks to ensure consistency and compatibility:

- Verifies that RGB and NIR bands have identical spatial resolution with the`is_Same_Resolution()` function

- Confirms that all raster and vector layers use the same Coordinate Reference System (CRS)

- Applies the Region of Interest (ROI) mask to isolate relevant areas (e.g., islands)

```r
# Function to check if two rasters have the same resolution
is_Same_Resolution <- function(band1, band2) {
    # Extract the resolution (pixel size) of each raster
    res_band1 <- res(band1)
    res_band2 <- res(band2)
    # Compare the resolutions and return TRUE if they are equal, otherwise
    # FALSE
    return(all.equal(res_band1, res_band2) == TRUE)
}

# Function to mask raster based on polygon and handle MS-specific issues
Raster_Masking <- function(archipelago, island, vector_mask, raster, nameSat, year,
    month = NULL, resolution, raster_type) {

    # Reproject vector if CRS differ
    if (!crs(raster) == crs(vector_mask)) {
        vector_mask <- project(vector_mask, crs(raster))
    }

    # Determine the number of bands in the raster
    num_bands <- nlyr(raster)  # Get the number of layers (bands)

    if (raster_type == "PMS") {
```

```
        # Check and harmonize resolution between RGB and NIR bands
        res_ok <- is_Same_Resolution(raster[[1]], raster[[num_bands]])  # Assuming the first band
↪   is blue, green or red and the last band is NIR
        if (!res_ok) {
            res_sup <- res(raster[[1]])  # coarse resolution (e.g. RGB)
            res_inf <- res(raster[[num_bands]])  # fine resolution  (e.g. NIR)
            factor <- res_sup[1]/res_inf[1]  # degradation factor
            raster[[num_bands]] <- aggregate(raster[[num_bands]], fact = factor,
                fun = mean)
        }

        # Harmonize dimensions
        dim_first <- c(nrow(raster[[1]]), ncol(raster[[1]]))  # Dimensions of the first raster band
        dim_last <- c(nrow(raster[[num_bands]]), ncol(raster[[num_bands]]))  # Dimensions of the
↪   last raster band

        if (dim_first[1] != dim_last[1]) {
            # If the number of rows is not equal
            last_ma <- as.matrix(raster[[num_bands]])  # Convert the last band to a matrix
            last_ma <- last_ma[-(dim_last[1]), ]  # Remove the last row
            empty <- rast(raster[[1]])  # Create an empty raster using the first band as a template
            raster[[num_bands]] <- rast(last_ma, xmn = 0, xmx = dim_first[2], ymn = 0,
                ymx = dim_first[1], template = empty)  # Recreate the last band with adjusted
                ↪  dimensions
        }

        if (dim_first[2] != dim_last[2]) {
            # If the number of columns is not equal
            last_ma <- as.matrix(raster[[num_bands]])  # Convert the last band to a matrix
            last_ma <- last_ma[, -(dim_last[2])]  # Remove the last column
            empty <- rast(raster[[1]])  # Create an empty raster using the first band as a template
            raster[[num_bands]] <- rast(last_ma, xmn = 0, xmx = dim_first[2], ymn = 0,
                ymx = dim_first[1], template = empty)  # Recreate the last band with adjusted
                ↪  dimensions
        }

        # Mask each spectral band individually
        masked_bands <- list()  # Initialize an empty list to hold masked bands
        for (i in 1:num_bands) {
            path_band <- paste(archipelago, "_", island, "_", nameSat, "_", year,
                "_", month, "_", resolution, "_band", i, "_cut.TIF", sep = "")
            masked_band <- mask(raster[[i]], vector_mask, filename = path_band, overwrite = TRUE)
            masked_bands[[i]] <- masked_band
        }
        return(masked_band)

    } else {

        # For slope or DTM, mask the raster and save the result
        path_raster <- paste(archipelago, "_", island, "_", nameSat, "_", year, "_",
            resolution, "_", raster_type, "_cut.TIF")
        raster_mask <- mask(raster, vector_mask, filename = path_raster, overwrite = TRUE)
        return(raster_mask)
    }
}
```

2.4 Define global variables

Note: A global variable is a variable defined outside of any function. This means the variable is accessible from any part of the code, including inside functions. A global variable retains its value throughout the execution of the R script unless it is explicitly modified in the code.

It is important to define at a minimum:

- the `"District"`: the archipelago of interest (e.g. "CRO" for Crozet archipelago)
- the `"Island"`: the island within the archipelago of interest (e.g. "POS" for Possession island)
- the `"Satellite1"`: the name of satellite providing multispectral imagery
- the `"Satellite2"`: the name of satellite providing terrain imagery
- the `"Year1"`: the acquisition year of the multispectral imagery
- the `"Month1"`: the acquisition month of the multispectral imagery
- the `"Year2"`: the acquisition year of terrain imagery
- the `"Res1"`: the resolution of the multispectral imagery
- the `"Res2"`: the resolution of terrain imagery

```
District = "CRO"  # 3-letter code for archipelago (e.g. Crozet)
Island = "POS"  # 3-letter code for island (e.g. Possession)
Satellite1 = "Pleiades"  # satellite name of multispectral imagery
Year1 = "2022"  # acquisition year of multispectral imagery
Month1 = "02"  # acquisition month of multispectral imagery
Res1 = "50cm"  # spatial resolution of multispectral imagery
Satellite2 = "SRTM"  # satellite name of DEM
Year2 = "2012"  # acquisition year of DEM
Res2 = "30m"  # spatial resolution of DEM
```

2.5 Set working directory

To optimize memory, you must define two general root directory (`"localHOME"`and `"localscratch"`) that serves as the base path for your input and output data, respectively. These directories should point to the local environment where:

- input rasters are located under `"data/raster/Precut_image"`
- input ROI mask is located under `"data/vector/mask"`
- outputs cut rasters will be saved under `"data/raster/Cut_image"`

```
# Base local path (customize to your local environment)
localHOME = paste0("/home/genouest/cnrs_umr6553/despel/CARTOVEGE/")
localscratch = paste0("/scratch/despel/CARTOVEGE/")
# localHOME = paste0('your_local_path/') localscatch =
# paste0('your_second_local_path/')

# path where to open vector data
open_mask_path = paste0(localHOME, "data/vector/mask")

# path where to open raster data
open_precut_raster_path = paste0(localHOME, "data/raster/Precut_image")

# path where to save your results
save_cut_raster_path = paste0(localscratch, "data/raster/Cut_image")
```

2.6 Load rasters and mask

```
print(paste0("Processing imagery for year: ", Year1))
print(paste0("Processing imagery for month: ", Month1))

# Load ROI polygon shapefile
print("Loading island ROI shapefile")
ROI_mask <- st_read(dsn = paste0(open_mask_path, "/", District, "_", Island, "_POLY_",
    Year1, "_EPSG32739.shp"))
ROI_mask <- as_Spatial(ROI_mask)

# Load raster data
print("Loading MS raster")
raster_MS <- rast(paste0(open_precut_raster_path, "/", District, "_", Island, "_",
    Satellite1, "_", Year1, "_", Month1, "_", Res1, "_PMS_precut.tif"))

print("Loading slope raster")
raster_slope <- rast(paste0(open_precut_raster_path, "/", District, "_", Island,
    "_", Satellite2, "_", Year2, "_", Res2, "_slope_precut.tif"))

print("Loading DTM raster")
raster_DTM <- rast(paste0(open_precut_raster_path, "/", District, "_", Island, "_",
    Satellite2, "_", Year2, "_", Res2, "_dtm_precut.tif"))
```

2.7 Mask rasters

Apply the `Raster_Masking()` function on the corresponding rasters.

```
setwd(save_cut_raster_path)
getwd()

# Mask MS raster
print("Masking MS raster using ROI polygon")
Raster_Masking(archipelago = District, island = Island, vector_mask = ROI_mask, raster = raster_MS,
    nameSat = Satellite1, year = Year1, month = Month1, resolution = Res1, raster_type = "PMS")

# Mask slope raster
print("Masking slope raster using ROI polygon")
Raster_Masking(archipelago = District, island = Island, vector_mask = ROI_mask, raster =
 ↪  raster_slope,
    nameSat = Satellite2, year = Year2, resolution = Res2, raster_type = "slope")

# Mask DTM raster
print("Masking DTM raster using ROI polygon")
Raster_Masking(archipelago = District, island = Island, vector_mask = ROI_mask, raster =
 ↪  raster_DTM,
    nameSat = Satellite2, year = Year2, resolution = Res2, raster_type = "dtm")
```

This script masks multispectral, slope, and DTM rasters using a polygon shapefile, ensuring resolution consistency and proper CRS alignment.