University of Freiburg
Faculty of Environment and Natural Resources
Module: GIS Plus
Lecturer: João Paulo Pereira
Module Coordinator: Holger Weinacker
Authors: Luka Kern, Nele Stackelberg and Felix Rentschler

July 6, 2018

Project 06: Rasterizer Function

Abstract

A rasterizing tool creates a regular grid whithin the bounding box of a geometry collection and assigns a value to each cell depending on the presence of a geometry in that cell region. This information can either be a binary one (no-data / data) or based on the geometry attributes. For this Project, the first cell will create a randomized geometry list containing the input data. The attributes for each of the three geometries are stored in a separated attributes list.

1 Aim of the Project

The aim of this project was to create a function that generates a raster-file (.tiff). This file displays the geometries from a shapefile, which was given as argument to the function. No matter what geometry type (point, lines, polygons) and spatial dimensions are stored in the given shapefile, the function has to convert them to a raster file.

2 Structure of the Function

The path of the shapefile is the functions first argument. A further argument is the amount of pixels, which is used as a proxy for the resolution. The buffersize in pixels and the name of the output file can additionally be defined as arguments for the function.
Furthermore, the user has the opportunity to specify the name of the result file, to preview the result inline, as well as suppressing the saving of the tiff file. For the case, that no shapefile is available to run the function, a code to generate random geometries is provided. (LUKA?!)

2.1 Load in the Shapefile

To load in the shapefile, the *Fiona* package is used. Every geometry object of the shapefile is stored in a GeometryCollection from the *Shapely* package.

2.2 Calculate the Resolution

To calculate the resolution, the minimum and maximum dimension of the geometry collection on the x- and y- axes are extracted. Those values are used to determine the range dimension.
The resolution depends on the amount of pixels in each dimension. Because the amount of pixels is one dimension, that is defined as input in the function, the range is divided by this value to get the resolution. Because the x- and y- range may differ, the mean of those two values is used in the calculation above.

2.3 Defining the Bounding Box and create a Grid

We create a bounding box around the spatial dimension of the shapefile. The purpose of this step is to present the objects in the resulting image with distance to the margin.
The functions argument "buffer" describes the amount of pixels in the bounding box. To transform the pixel value to the coordinates, it is first multiplied with the resolution, the result is the width of the boundary box.
The dimension of the geometry collection (xmin, ymin, xmax, ymax) is used to define the dimension of the outer frame of the bounding box by adding and subtracting the boundary box width.
The minimum and maximum coordinate values (dimension) of the outer frame of the boundary box are used to create a grid with the function *numpy.mgrid*. The coordinates of every grid-cell (= every pixel) are saved as point-geometry. Those geometries are stored in a list called geom_pixels.

2.4 Within Query

Now, that the position of every grid-cell is known, the vector geometries can be transferred to the raster-grid. The query, that defines wheter a geometry object covers a pixel (metaphorically speaking: imagine a polygon laying on a squared paper and transforming it manually) depends on the geometry type of the vector file. If it is a Point, the query asks if the pixels position (from geom_pixels) equals the points position (+- 0.5*resolution in both x- and y-dimension) in the vector data. The addition respectively subtraction of 0.5 times the resolution makes sure that every point is detected, even if it is not on the midpoint of the grid cell. If the vector geometry is a line, a buffer around the line is calculated. The within-function of the shapely-package is than used to query if a point is in this buffer. The same query is also used for polygon-geometries, where the within-function is applied directly to the polygon and not to a buffer. The query is applied to all geometries of the geometry collection. For each geometry a list with boolean values is the result. These list are appended one after another as sublists.

2.5 Writing the result in an array

In the next step the sublists are combined. Therefore the sum off all sublist is written to a new list. The sum of boolean values is the amount of "True"-values. To create an array of the right size out of this list, it has first to be transformed into sublist, each with the length of the amount of pixels on the x-axis. The resulting format can be transformed to a numpy-array.

2.6 Set Radiometric Resolution

To make sure that all values of the array will be displayed in the 8bit TIFF file, the values are transformed to the range between 0 and 255. The calculation of every value x_{i_new} from its former value x_i in the grid x is computed according to the following formula: $x_{i\_new} = 255 * \frac{x_i}{x_{max}}$

2.7 Flip Array and Save as TIFF

If the array was plotted at this stage, pythons characteristics to show the y-achsis the reverse way would result in a inverted image. Therefore the array is flipped the up/down direction with the flipud-function of the package numpy. After all this has been computed, the result can be saved as TIFF-file. For this purpose the scikit-package provieds the imsave-function, which is used here.

3 Issues

3.1 Solved

3.2 Unsolved