Assignment 02

Viewshed computation

Deadline is **2020-12-16 10:00am**

Late submission? 10% will be removed for each day that you are late.

You're allowed for this assignment to work in a **group of 2** (and thus submit only one solution for both of you). If you prefer to work alone it's also fine.

Overview

The aim of this assignment is to design and implement a viewshed algorithms for grids. Eventually, you should be able to compute what parts of Rio de Janeiro can be seen by Cristo Redentor, the famous 38m high statue shown in the image above. Or actually, what he would see if he could turn around his head…

The handout provides more information on how to perform a viewshed computation. You will have to implement the algorithm so that several viewpoints are taken into account. The resulting viewshed is what can be seen from any of the viewpoints, ie if one DTM cell is visible only from one viewpoint, then it is visible.

You can choose between Python and C++ for this assignment. For both, starting code is provided, and the same datasets are given as input; the formats of them and what is expected as output is slightly different however (mostly because installing GDAL under Windows for C++ is tricky, so we simplified the input/output). We encourage you to explore C++ with this assignment, since the implementation is similar for both language; we can provide extra help for C++. As a reminder, hw03 must be done with C++.

If you choose Python

Code is in the `/hw/02/python/` folder of the GitLab repository of the course.

We give you the skeleton of the program:

1. `geo1015_hw02.py` is the `main()`. You are *not* allowed to modify this file!
2. `mycode_hw02.py` is where *all* your code should go. The function `output_viewshed()` receives all the relevant information and writes to the disk the output viewshed, as defined below. You are of course allowed to add any other functions you want. You are only

allowed to import modules from the Python standard library (anything you installed through `pip` is thus not allowed, except *rasterio*, *scipy* and *numpy*).
3. `params.json`: a very simple JSON file that defines what the input file is, the viewpoints, with what parameters the viewpoints will be computed, and where to store the output. The `"height"` is the height of the viewpoint above the surface (2m for a person thus).
4. `/data/`: contains 2 raster terrains in GeoTIFF. For `/data/CristoRendetor/`, the location of the viewpoint is that of Cristo Redentor himself.
5. `example_out.tif`: an example of a GeoTIFF output file that your program will need to create.

The input file `params.json` given doesn't need to be validated (no wrong parameters can be in a given file), and the input/output and parameters are also valid.

The output format

The output file has to be a GeoTIFF (.tif), with the same CRS/extent/resolution as the input, but with type "Byte - Eight bit unsigned integer" (the code given shows how to do this).

There are 4 possible values for a pixel:

- 0: not visible from the viewpoint(s) (but inside the max-distance/horizon zone(s))
- 1: visible from the viewpoint(s)
- 2: the pixel contains a viewpoint
- 3: the pixel is outside the max-distance/horizon zone(s)

If you choose C++

Code is in the `/hw/02/c++/` folder of the GitLab repository of the course.

We give you the skeleton of the program:

1. `/src/main.cpp` is the `main()`, and this is where all your code should go. The function `output_viewshed()` receives all the relevant information and writes to the disk the output viewshed, as defined below. You are of course allowed to add any other functions you want.
2. `/src/DatasetASC.h/cpp` is the class where the input .asc file is read and stored.
3. `params.json`: a very simple JSON file that defines what the input file is, the viewpoints, with what parameters the viewpoints will be computed, and where to store the output. The `"height"` is the height of the viewpoint above the surface (2m for a person thus).
4. `/data/`: contains 2 raster terrains in .asc format. For `/data/CristoRendetor/`, the location of the viewpoint is that of Cristo Redentor himself.
5. `example_out.asc`: an example of a .asc output file that your program will need to create.

The input file `params.json` given doesn't need to be validated (no wrong parameters can be in a given file), and the input/output and parameters are also valid.

To compile and run the program

1. from the `/c++/` folder create a new folder `build` (`mkdir build`)
2. `cd build`
3. `cmake ..`
4. `make`
5. to run the program with the parameter file `/c++/data/params.json` do `./hw02`

The output format

Your output file has to be an Esri ASCII raster format (.asc), see its full specifications.

The raster needs to have the same extent/resolution as the input, and there are 4 possible values for a pixel:

- 0: not visible from the viewpoint(s) (but inside the max-distance/horizon zone(s))
- 1: visible from the viewpoint(s)
- 2: the pixel contains a viewpoint
- 3: the pixel is outside the max-distance/horizon zone(s)

The `NODATA_VALUE`, which is not used, must be set to `-9999`.

You need to design a strategy yourself, and document it in a report

There are several ways to solve this problem, and you are free to design the solution you want. Doing it brute-force, as described in the handout, is fine.

You are allowed to make approximations to speed up the operations performed, but these should be documented in your report. For instance, a grid of ~30m resolution brings a certain uncertainty, so one is allowed to approximate some parts when extracting which pixels represent the line segments between the viewpoint and a target point.

Your report does not need to have an introduction. It should only describe your algorithm at a high level (using pseudo-code, or a flowchart). Start by presenting the big steps, and then describe briefly how you implemented them. If assumptions were made to speed up the code, describe and document in the report the consequences of these. The report should ideally be 4-5 pages.

Also, you should report how long it took to generate the 2 files (for `params.json` and `params2.json`). The time is printed automatically when the process finishes.

Good to know

- you can assume that the viewpoints are at the centre of the pixels in which they are located (thus so you can shift the viewpoints slightly)
- the CRS of the input grids will always be in meters
- the viewpoints can "look" in all directions
- the viewpoints are guaranteed to be inside the cells (they are given in *(x, y)* coordinates)
- identifying all the grid cells intersecting a line will be slow if the intersection with each must be calculated. Some ideas on how to perform this are there, and there. The second method is actually the same as the conversion vector-raster as seen in Lecture 10 of GEO1002, and it is implemented in rasterio. We give you an example in the Python code.
- the QGIS plugin Profile Tool is useful to debug and to assess whether you obtain sensible results

Marking

Criterion	Points
report – general explanations	2
report – limitations/shortcuts + consequences	2
code works "out-of-the-box"	1
.tiff files as expected	1
many viewpoints correctly handled?	1
quality of results	3

What to submit and how to submit it

You have to submit a total of 4 files:

Python

1. the Python file `my_code_hw03.py`, where your names are clearly identified at the top of the file.
2. the output GeoTIFF file `out-tasmania.tif` (for `params.json`)
3. the output GeoTIFF file `out-cristo.tif` (for `params2.json`)
4. a report in PDF format (no Word file please)

C++

1. the C++ file `main.cpp`, where your names are clearly identified at the top of the file.
2. the output .asc file `out-tasmania.asc` (for `params.json`)
3. the output .asc file `out-cristo.asc` (for `params2.json`)
4. a report in PDF format (no Word file please)

Those 4 files need to be zipped and the name of the file is the studentIDs of the members separated by a "_", for example `5015199_4018169.zip`.

Upload the ZIP file to this surfdrive page.

[last updated: 2020-11-27 08:30]