DTM assignment 3

Qu Wang, 4700686, Pablo Ruben, 4273818

12 Jan. 2019

1 Global approach of the code

The approach which was chosen can globally be divided in the following steps:
1. Shoot rays in different directions from the viewpoint
2. Identify the pixels visited by these rays
3. Use the height values to identify whether pixels are visible or not

One should note that the algorithm was adapted so that one output file can contain the results of several viewpoints. In order to combine results from different viewpoints, it was decided that a pixel is considered visible if it is visible from at least one of them.
The global pseudocode of the approach can be found in figure 1.

2 Implementation

2.1 Creation of lines of sight

In the first for loop in figure 1 (before the nested for loop starts), the destination pixels of the lines of sight by first creating a square containing the circle with as center the viewpoint and as radius the view range. These ones are then shrunken by testing the length of the lines of sight and adapting the destination pixel to fulfill the maximum view range (see figure 4. This is done in a way that even if the centerpoint of the final pixel is outside range, but a part of that pixel is within the view range, this one is considered as a line of sight destination pixel.

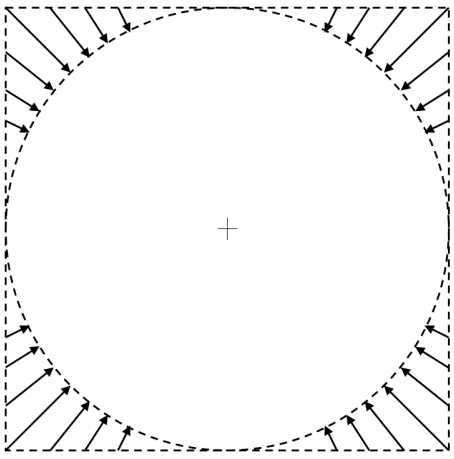

Figure 2: Illustration of the algorithm shrinking of lines of sight originating from a square shape.

For the generation of the list of pixels traversed by the line of sight, rasterio's *features.rasterize()* function was used as shown in the pseudo code in figure 1 . The characteristic *all_touched* had to be

Algorithm 1 Calculate Viewshed

Input
- npi DEM raster
- VPS a set of view points
- d_m maxdistance

Output
- vs viewshed for input region

```
vs initialize every item as 3
for each viewpoint vp in VPS do
    vs update vp as 2
    alt_vp ← ground altitude + height
    rBox ← a square minimum bounding box for the circle, central at pv with radius d_m, c_p
    for each point pt in rBox do
        dis ← distance between pt and vp
        if dis > d_m then
            coords ← a point shrinking to c_p alone the ray starting at vp ending at pt
        else
            coords ← pt
        end if
        if coords not in cBox then
            cBox ← coords
        end if
    end for
    for each point pt in cBox do
        path ← Bresenham_with_rasterio(npi, vp, pt)
        max_alt ← −99
        max_tan ← −99
        for each point pp in path do
            alt ← the altitude of pp
            if alt > max_alt then
                dist ← the length of projecting vector VpPp to vector VpPt
                if alt > alt_vp then
                    max_alt ← alt
                end if
                tan ← arctan((alt_pp − alt_vp)/dist)
                if tan > max_tanandpp! = vp then
                    max_tan ← tan
                    vs update pp as 1
                else
                    if pp has't been visited/value of pp is 3 then
                        vs update pp as 0
                    end if
                end if
            else
                if pp hasn't been visited/value of pp is 3 then
                    vs update pp as 0
     return vs
```

Figure 1: Pseudocode of the computation of a viewshed

Algorithm 2 Bresenham_with_rasterio

Input

raster DEM raster
start Start point (nrow,ncol) of a vector $\vec{l}$
end End point (nrow,ncol)of a vector $\vec{l}$

Output

cells A set of ordered cells index (nrow,ncol) from start to end

re ← cell in the line
cells ← cell index of *re*, first in ascending order of row, then in ascending order of column
if $\vec{l}$ points to northeast **then**
*cells***sort** first in descending order of row, then in ascending order of column
end if
if $\vec{l}$ points to northwest **then**
*cells***sort** first in descending order of row, then in descending order of column
end if
if $\vec{l}$ points to southwest **then**
*cells***sort** first in aescending order of row, then in descending order of column
end if
return cells

Figure 3: Pseudocode of the computation of the pixels visited by a line of sight

set to true as otherwise the resolution would not have allowed this method to work without leaving non-visited cells.

2.2 Visibility computation

Once all the cells visited were identified, visibility computation was performed (second for loop of the pseudocode shown in figure 1).
This was done by first projecting the point on the line (replacing it by its closest point on the line) to get an estimate of the position on the line. Another, less accurate alternative would have been to simply take the total distance and divide it by the number of pixels, meaning that each pixel has the same size on the line. A more accurate alternative would have been to compute the intersections each time the line of sight enters and exist a pixel. The latter was not applied here because of its increase in computation time and as accuracy of the point projection was deemed sufficient with regard to other assumptions (such as moving the viewpoint to the center point of the pixel containing it).

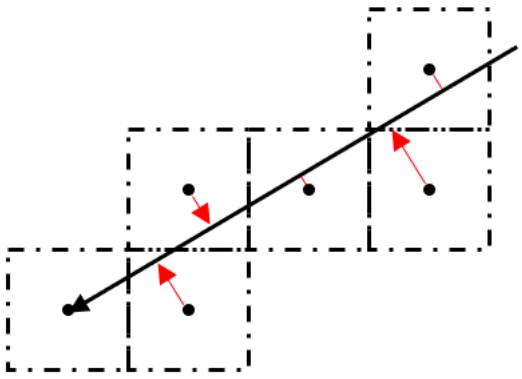

Figure 4: Illustration of the algorithm projecting the point on the line.

Finally, the visibility of the points was checked starting from the closest point to the viewpoint and traveling the line until the end. Here, the tangent method was used: for each point, the angle of the line connecting the viewpoint to the destination point with regard to a horizontal one was computed (if the first was below the latter, the angle was considered negative). If that angle was bigger than all the previous ones, the pixel was considered visible. If the angle was smaller, the pixel was stored as invisible.
With the aim to speed up computation time, the altitude of the pixel visited was checked before computing the tangent angle. In fact, if a pixel has an altitude which is equal or bigger to the one of the viewpoint, only pixels with an even higher altitude can be visible behind it. This is illustrated in figure 5

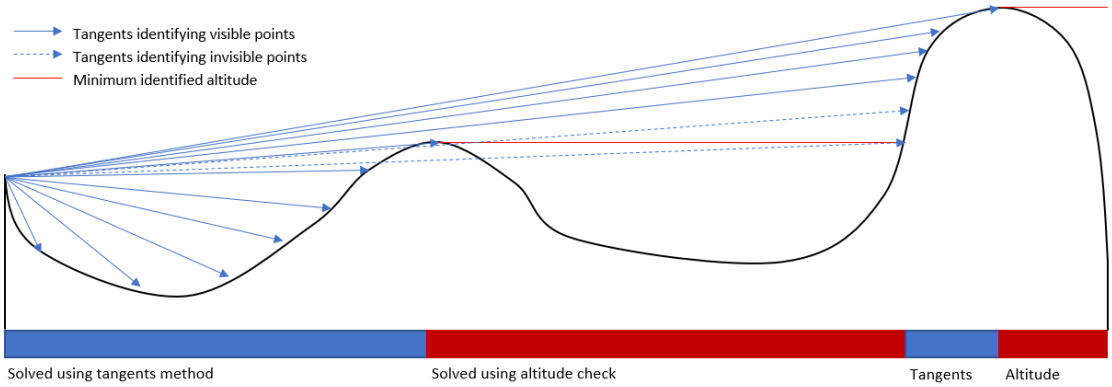

Figure 5: Illustration of the two methods employed for checking visibility: tangent and minimum altitude.

It can be observed that pixels might be visible in one situation but not in another (if a pixel is visited by lines of sights going in different directions, it might be preceded by different pixels). in such situation, the same approach as in section 1 was used: a pixel which is visible in at least one of the lines of sight is considered visible.

3 Assumptions made

This section intends to provide an overview of the assumptions that have been made in our approach:

1. A pixel which is at least partially visible from one viewpoint is stored as visible.
2. The viewpoint coordinates have been shifted to the center of the pixel containing it.
3. It was assumed that projecting a point onto a line is sufficiently accurate with regard to the usage of pixels and transformation of coordinates to center points.
4. The curvature of the earth has not been taken into account for this assignment.
5. The ground surface is 2.5D/continuous .e.g. it does not contain any holes such as tunnels, cliffs with parts sticking out, etc.

4 Results

4.1 Visualization

The results obtained for the two datasets of the assignment can be found below. Pixels are coloured depending on whether they haven't been visited (white), are invisible (black), visible (dark gray) or contain a viewpoint (light gray).

Figure 6: Result of the viewshed obtained for the Tasmania dataset.

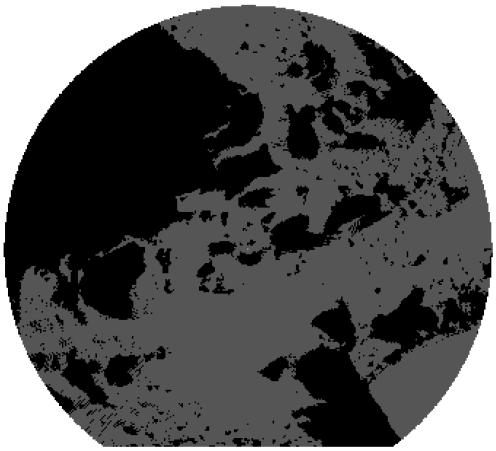

Figure 7: Result of the viewshed obtained for the Cristo Redentor dataset.

4.2 Computation time

In order to check whether the altitude check is beneficial, computation time with and without it were compared for the two datasets (Table 1). It appears that the algorithm is indeed faster with the altitude check, although the difference is about 11.5% for the Tasmania data and only 1.8% for the Cristo Redentor. This is most likely linked to the nature of the data. The viewpoint in Tasmania is rather "low" compared to its environment and few pixels are visible while the Cristo Redentor is a high landmark and more pixels are visible. Therefore it can be concluded that the altitude check is mainly beneficial if the viewpoint is located lower than the pixels of the viewshed. On the other hand, one could even imagine situations where the altitude check increases computation time, for instance if the viewpoint is located at the highest point in the viewshed.

Table 1: Computation time

dataset	computation time [s] (with altitude check)	computation time [s] (without altitude check)
Cristo Redentor	8.514	8.677
Tasmania	1.921	2.171