Laboratory of Innovative wireless platforms for the internet of things

a.a. 2018 – 2019
Daniele Castro S253244
Group number: 8

CONTENTS

LIST OF FIGURES

Laboratory of Innovative wireless platforms for the internet of things

***Abstract*—The project is a simple to use script written in JavaScript that allows the user to know in advance whether to put a transmitting antenna in a specific place or not.**

I. Introduction

The marker represents the transmitting antenna and the red and black spots represents possible retrieving antenna positions. If a spot is red, it means that in that place a retrieving antenna can retrieve the electromagnetic radiation from the transmitting antenna, not vice versa.

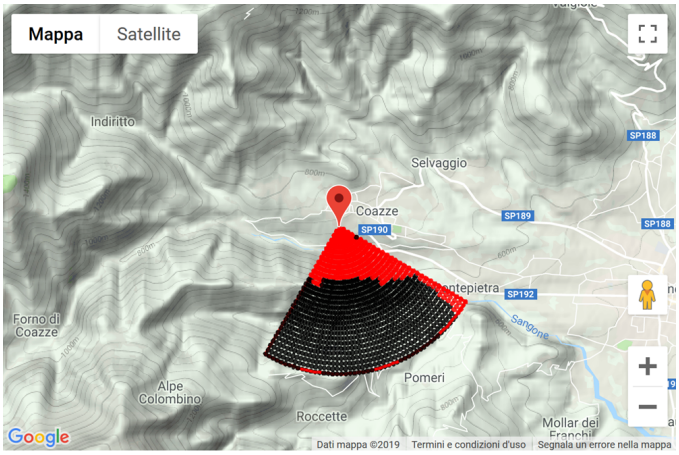

Fig. 1. obtained final result

II. Background

Many parameters can be modified in order to give the best user experience in the **initMap()** function. Configuration parameters:

- **radius** is the radius of the cone
- **startAngle** and endAngle regulates the opening of the cone
- **antennaHeight** is the height of the transmitting antenna
- **retrieverHeight** is the height of the retrieving antenna
- **samples** is the number of spots per path
- **angleStep** is the angle between two paths

Here can be better seen how **startAngle** and endAngle works:

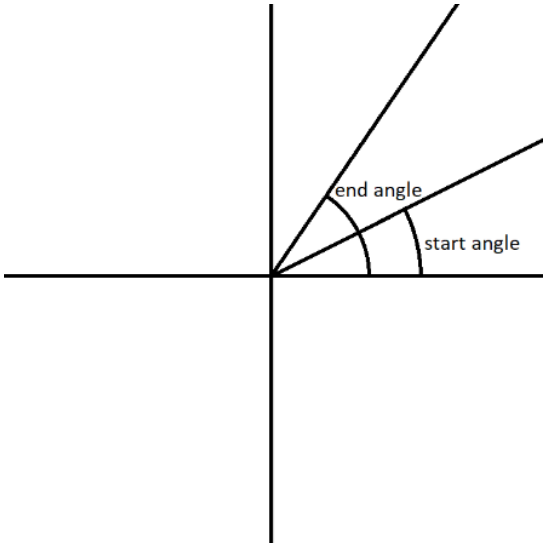

Fig. 2. start angle and end angle behaviuor

retrieverHeight is the height of the retrieving antenna and is summed to the height of each spot in the map because it is supposed that the user wants to see if placing a retrieving antenna in whatever place with a given height will work or the electromagnetic radiation will not achieve the retrieving antenna. To better understand the meaning of **samples** and **angleStep** This picture can be useful:

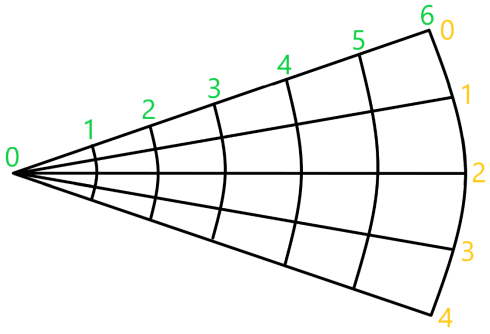

Fig. 3. map cone numbered

Orange number represents paths and green numbers represents each single spot in a path; exception made for spot 0 that is the

centre (the transmitting antenna position). Between two orange paths there is the **angleStep**. **samples** is the number of green spots.

III. Brief HGT file format explanation

A file with the HGT file extension is a Shuttle Radar Topography Mission (SRTM) Data file. It contains digital elevation models, which are 3D pictures of a surface (usually a planet).

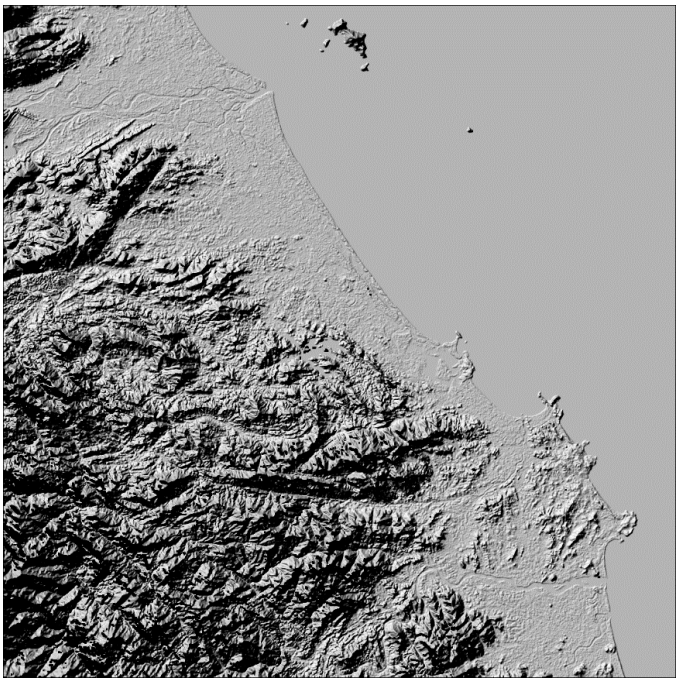

Fig. 4. HGT file converted into a picture

They are obtained during the Shuttle Radar Topography Mission (SRTM) by NASA and the National Geospatial-Intelligence Agency (NGA). ”HGT” is just an abbreviation for ”height”. HGT file is normally named with the longitude and latitude that the image pertains to, within one degree. For example, the file N33W177.hgt would indicate that it includes data for latitudes 33 to 34 North and longitudes 177 to 178 West. SRTM data are distributed in two levels: SRTM1 (for the U.S. and its territories and possessions) with data sampled at one arc-second intervals in latitude and longitude, and SRTM3 (for the world) sampled at three arc-seconds (our case). Data are divided into one by one-degree latitude and longitude tiles in ”geographic” projection, which is to say a raster presentation with equal intervals of latitude and longitude in no projection at all but easy to manipulate and mosaic. Height files are signed two-byte integers. The bytes are in Motorola ”big-endian” order with the most significant byte first. Heights are in meters referenced to the WGS84/EGM96 geoid. Data voids are assigned the value of -32768. SRTM1 files are $3601 * 3601 * 2$ bytes long and SRTM3 files are $1201 * 1201 * 2$ bytes long.

IV. Custom API

Since Google blocks its own services when too many requests are made within a second, A custom API had to be developed in order to make this project usable with a discrete number of samples. These custom APIs are made of five functions:

- **do_interpolation()**
- **init_interpolation()**
- **reverse()**
- **pointsAlongPath()**
- **getElevationAndPoints()**

They work like this:
init_interpolation() reads a given SRTM3 HGT file and generates a matrix containing all the height samples in the HGT file. Every time an height sample is extracted from the file **reverse()** function is called to swap the two bytes that makes the sample: it converts big-endian samples in little-endian ones. After this initialization process a **matrixObjct** is made and passed to the others API functions. This object contains the matrix extracted from the HGT file and the number of rows and columns. **do_interpolation()** is the function that does the interpolation. To make the interpolation possible **init_interpolation()** filled the matrix like in the following picture:

null	0	1	2	3
0	value	value	value	value
1	value	value	value	value
2	value	value	value	value
3	value	value	value	value

Fig. 5. interpolation matrix

do_interpolation() also redirects to the proper file matrix the request of a given point coordinates. **pointsAlongPath()** it is a simple function that gives a certain number of equidistant samples within a path. Finally **getElevationAndPoints()** is the function that emulates the behaviour of the Google’s **getElevationAndPoints()** it calls **pointsAlongPath()** and returns an array of objects that contains the height and the coordinates of each point.

V. Proposed Solution

Script functions are:

- **initMap()**
- **displayPathElevation()**
- **makeCircle()**
- **plotElevation()**
- **plotElevation2()**
- **getHeight()**
- **onStraightSight()**

In the **initMap()** function few parameters can be tuned, like explained at the beginning, and the initialization function **displayPathElevation()** is called. Here **makeCircle()** creates a certain number of paths and calls **plotElevation()** that begins the recursion.
The code is synchronously run and made recursive because, in this way, it can be easily upgraded in an asynchronous algorithm. Asynchronous algorithms are optimal for web services since they allow multiple part of the algorithm to be run separately in parallel in different threads. The recursion is made by **plotElevation()** and **plotElevation2()** To better explain how they work the following picture can be seen:

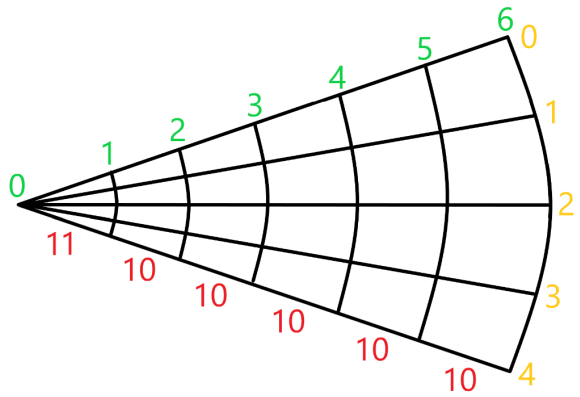

Fig. 6. map cone numbered

plotElevation() increments the **cnt** recursion counter. **cnt** goes from 0 to the total number of orange paths. So a **'samples'** number of green points are requested to be generated by **pointsAlongPath()**. For each of these points **plotElevation2()** is called and it increments the **i** recursion counter variable. It calls **getElevationAndPoints()** that is asked to generate $11+(10*(i-1))$ red number of samples along a path starting from the centre. All of these samples are passed through an array to **getHeight()** and **onStraightSight()** that will, respectively, sum the earth curvature to the current heights array and calculate whether if a given point can be achieved by the electromagnetic radiation or not.

Here follows a graphical explanation of what said:

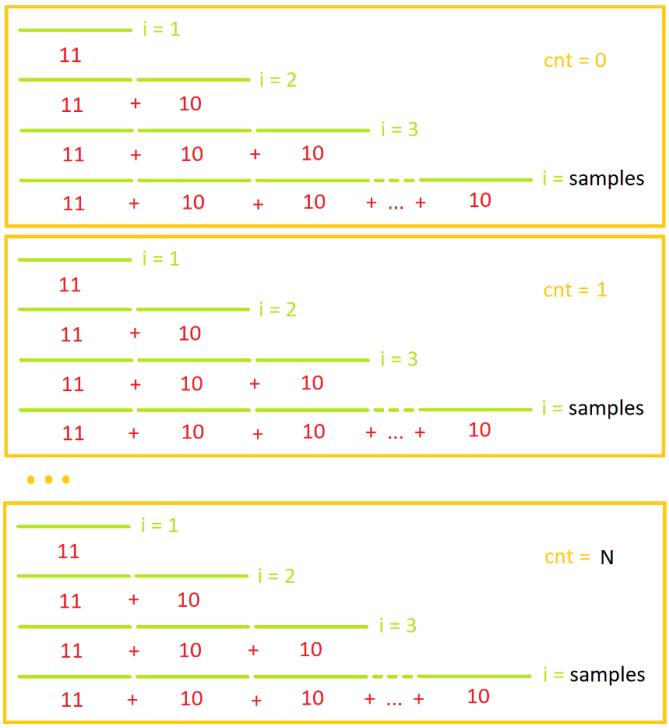

Fig. 7. graphical explanation of the algorithm

VI. Conclusions

In this discussion I have accomplished the assignment requested. I would like to reuse this script even for further uses since it has been made flexible and scalable.

References

[1] SRTM Documentation (best viewed with mono-spaced font, such as courier), [txt] Available at:
Link.
Accessed on: July 13, 2019.

[2] Google maps API, Available at:
Link.
Accessed on: July 13, 2019.

[3] Interpolation javascript script, Available at:
Link.
Accessed on: July 13, 2019.