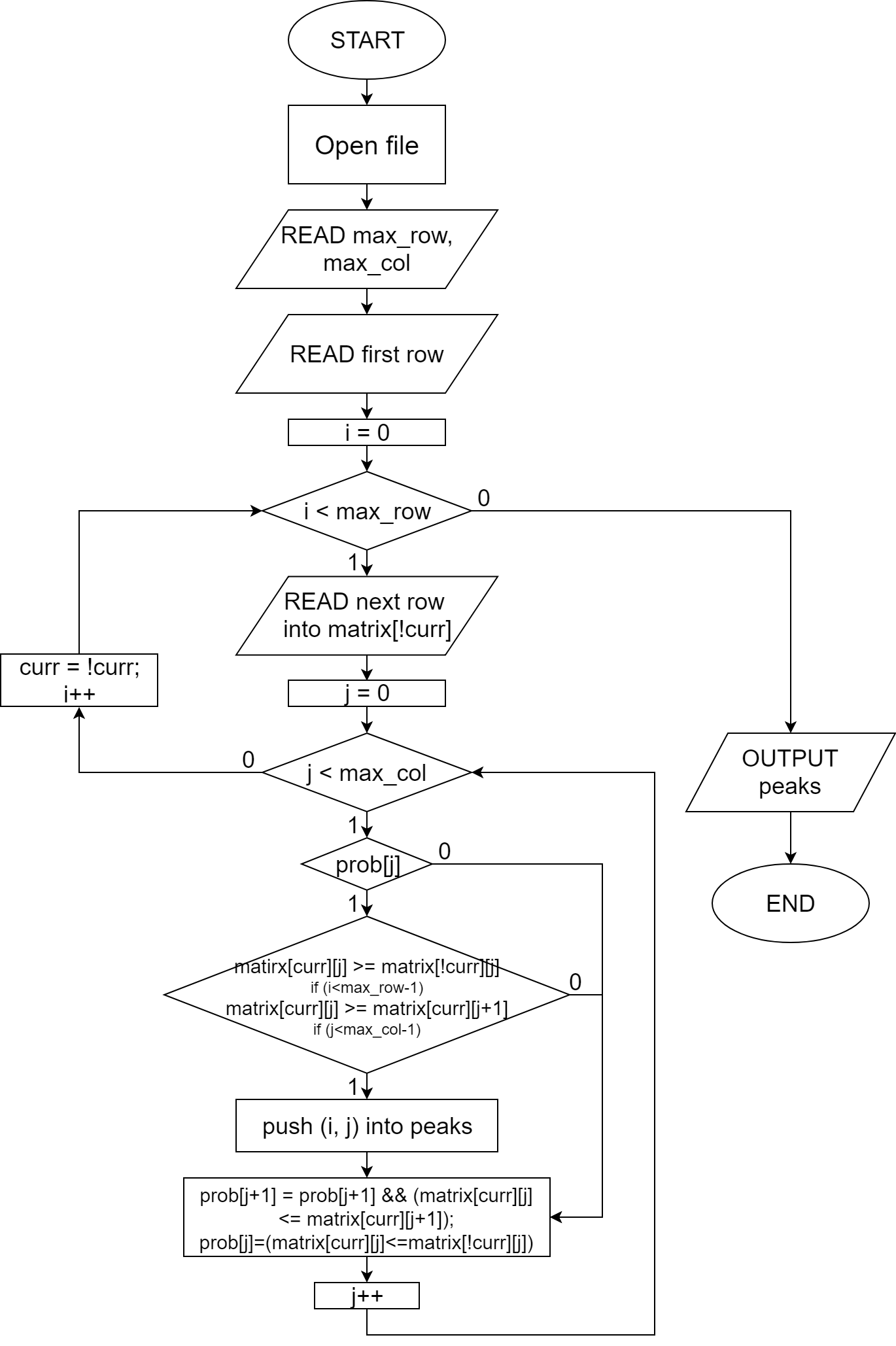
**Project #1: Peak Finder**

# Project Description

## Program Flow Chart



## Question Analysis

A point hi,j  is classified as peak if:

ℎ𝑖−1, ,𝑖𝑓 𝑖>1

ℎ𝑖+1, ,𝑖𝑓 𝑖<𝑚

ℎ𝑖, 𝑗 ≥

ℎ𝑖,−1 ,𝑖𝑓𝑗>1

ℎ𝑖,+1 ,𝑖𝑓 𝑗<𝑛

We can divide our design into 3 part:

1. Read the matrix
2. **Compare**: compare points with its adjacent points (up, down, left, right)
3. **Store the peaks**: mark the point as peak if it is larger or equal than its adjacent points
4. Output the peaks

## Development and Optimization

### Matrix Reading

Method 1:Use a MxN array to store the matrix

long int matrix[max\_row][max\_col];

*for*(int i=0; i<max\_row; i++)

*for*(int j=0; j<max\_col; j++)

input >> matrix[i][j];

Space Complexity: max\_row\*max\_col

However, I found that we don’t need to store the whole matrix because we only compare a point with its adjacent points to know if it is a peak, which means we only need to store the current row and the next row we are comparing (what about the row above? Will be explain later)

Method 2 (Optimized): Store only the current row and the next row

*for*(int i=0; i<max\_row; i++) {

*for*(int j=0; j<max\_col; j++) {

*/\* Read the next row \*/*

input >> matrix[!curr][j];

}

*/\* perform comparison \*/*

curr = !curr;

}

Space Complexity: 2\*max\_col

Concept:

1. Comparison is performed immediately after each row and the next row is read.
2. Then, curr is switched to the next row.
3. Now read the new row into !curr, the row above curr will be replaced by the row below curr

### Comparison

Method 1: Compare every points with its up, down , left, right points

*if*((i==0) || (matrix[curr][j] >= matrix[curr-1][j])) *//compare with its up*

*if*((i==max\_row-1) || (matrix[curr][j] >= matrix[curr+1][j])) { *//compare with its down*

*if*((j==0) || (matrix[curr][j] >= matrix[curr][j-1])) *//compare with its left*

*if*((j==max\_col-1) || (matrix[curr][j] >= matrix[curr][j+1])) *//compare with its right*

*//mark matrix[curr][j] as peak*

Space Complexity: 3\*max\_col (need to store matrix[curr], [curr-1], [curr+1])

Time Complexity: 4\*max\_row\*max\_col

Method 2 (Optimized): Compare only with the down and right points

int prob[max\_col]; *//probability of a point to become peak*

*/\*set prob[j] of the first row as 1 \*/*

*/\* for each row \*/*

*if*(prob[j])

*if*((j==max\_col-1) || (matrix[curr][j] >= matrix[curr][j+1])) *//compare with its right*

*if*((i==max\_row-1) || (matrix[curr][j] >= matrix[!curr][j])){*//compare with its down*

*//mark matrix[curr][j] as peak*

*if*(j<max\_col-1) prob[j+1] = prob[j+1] && (matrix[curr][j] <= matrix[curr][j+1]); *//set prob of its right*

prob[j] = (matrix[curr][j] <= matrix[!curr][j]); *//set prob of its down*

Space Comlexity: 3\*max\_col (prob, matrix[curr], [curr+1])

Time Complexity: 4\*max\_row\*max\_col

Concept:

1. An array prob[max\_col] is used as flag to record the comparison result of the a point with its upper point and its left point.
2. Comparison to the right and down point is run only when prob[j]==1
3. The prob of the right point (prob[j+1]) is renewed.
4. prob[j] is renewed and now represents the prob of the point[j] in the next row

### Store the peaks + Output

Method 1: an MxN matrix to record the result of each point

*/\* Mark peaks \*/*

result[i][j] = prob[j];

*/\* Output\*/*

*for*(int i=0; i<max\_row; i++)

*for*(int j=0; j<max\_col; j++)

*if*(result[i][j]) output << i << ' ' << j << endl;

Space Complexity: max\_row\*max\_col

Time Complexity (output): max\_row\*max\_col

Method 2 (Optimized): Use vector to store peaks

class coordinate {

int r, c;

public:

coordinate(int x, int y): r(x), c(y) {}

int matrix() { *return* r; }

int col() { *return* c; }

};

vector<coordinate> peaks;

*/\* Mark peaks \*/*

*if*(*/\*comparison\*/*) peaks.push\_back(coordinate(i, j));

*/\* Output \*/*

cout << peaks.size() << endl;

*for*(auto peak : peaks) output << peak.matrix() << ' ' << peak.col() << endl;

Space Complexity: no. of peaks (<=max\_row\*max\_col) \*2

Time Complexity: no. of peaks

## Handling testcases

1. Use argv, argc to get the input\_student\_id  
   Reference: <https://stackoverflow.com/questions/3024197/what-does-int-argc-char-argv-mean>

int main (int argc, char\* argv[]) {

1. Expand the argv to get the directory of testcase  
   \*\* Specified cases for TA’s testcase which is not in a subfolder

string dir = argv[1], dir\_in, dir\_out;

*if*(dir=="TA\_matrix\_1" || dir=="TA\_matrix\_2" || dir=="TA\_matrix\_3") {

dir\_in = dir + ".data";

dir\_out = "final.peak";

}

*else* {

dir\_in = dir + "/matrix.data";

dir\_out = dir + "/final.peak";

}

1. Open files as ifstream and ofstream   
   Reference: <http://www.cplusplus.com/doc/tutorial/files/>

<http://www.cplusplus.com/reference/fstream/ifstream/>

<http://www.cplusplus.com/reference/fstream/ofstream/>

\*\*Error message is designed to be shown when unable to open testcase

ifstream input(dir\_in);

ofstream output(dir\_out);

*if*(input.is\_open() && output.is\_open()) {

*/\* Code \*/*

input.close();

output.close();

}

*else* *if*(!input.is\_open()) cout << "Unable to open testcase" << endl;

*else* cout << "Unable to open output" << endl;

# Testcase Design

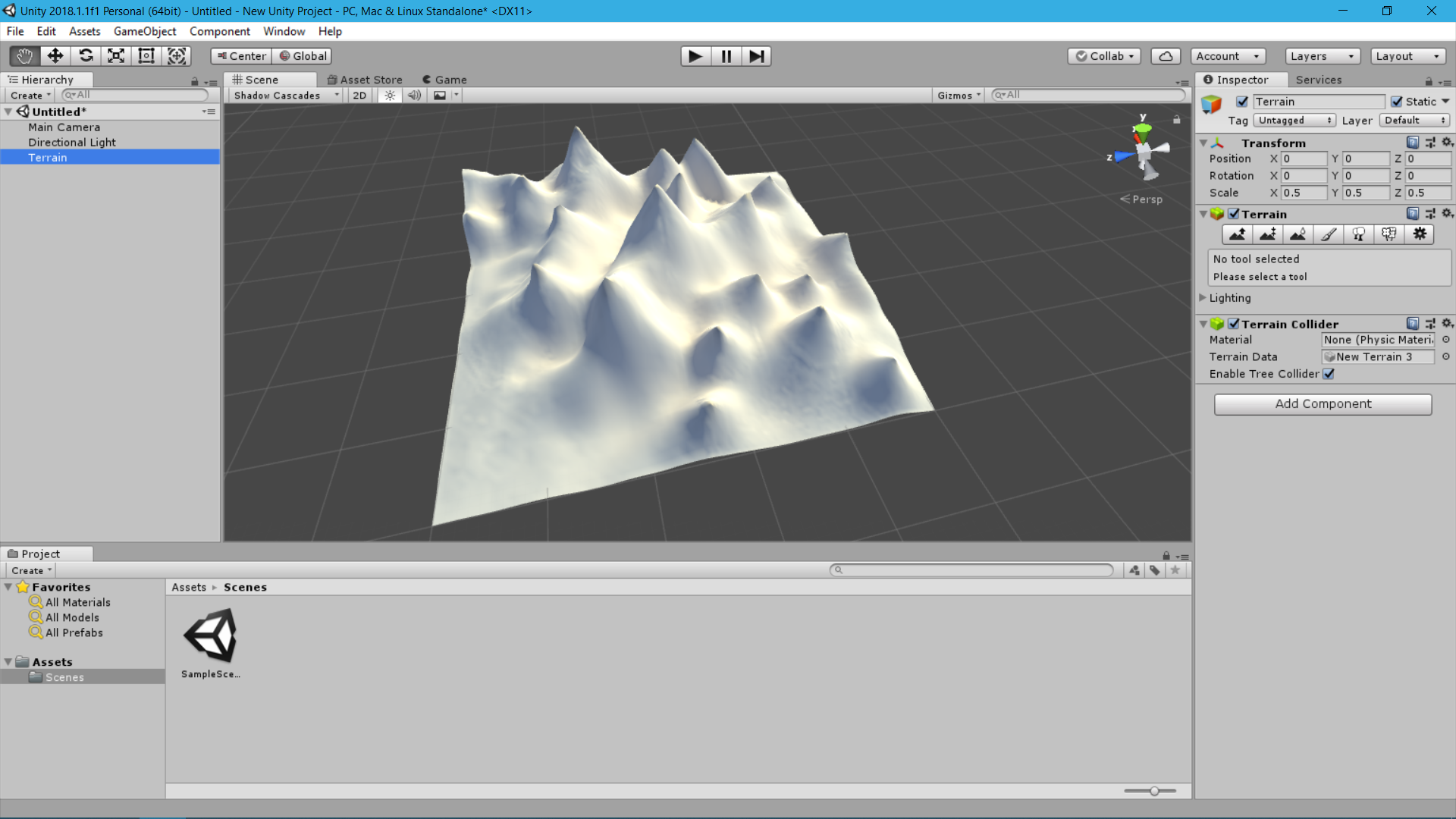
## Design Concept

I would like to design a more analogue-like testcase. So I build a model to simulate the natural topography, and then convert it into a matrix.

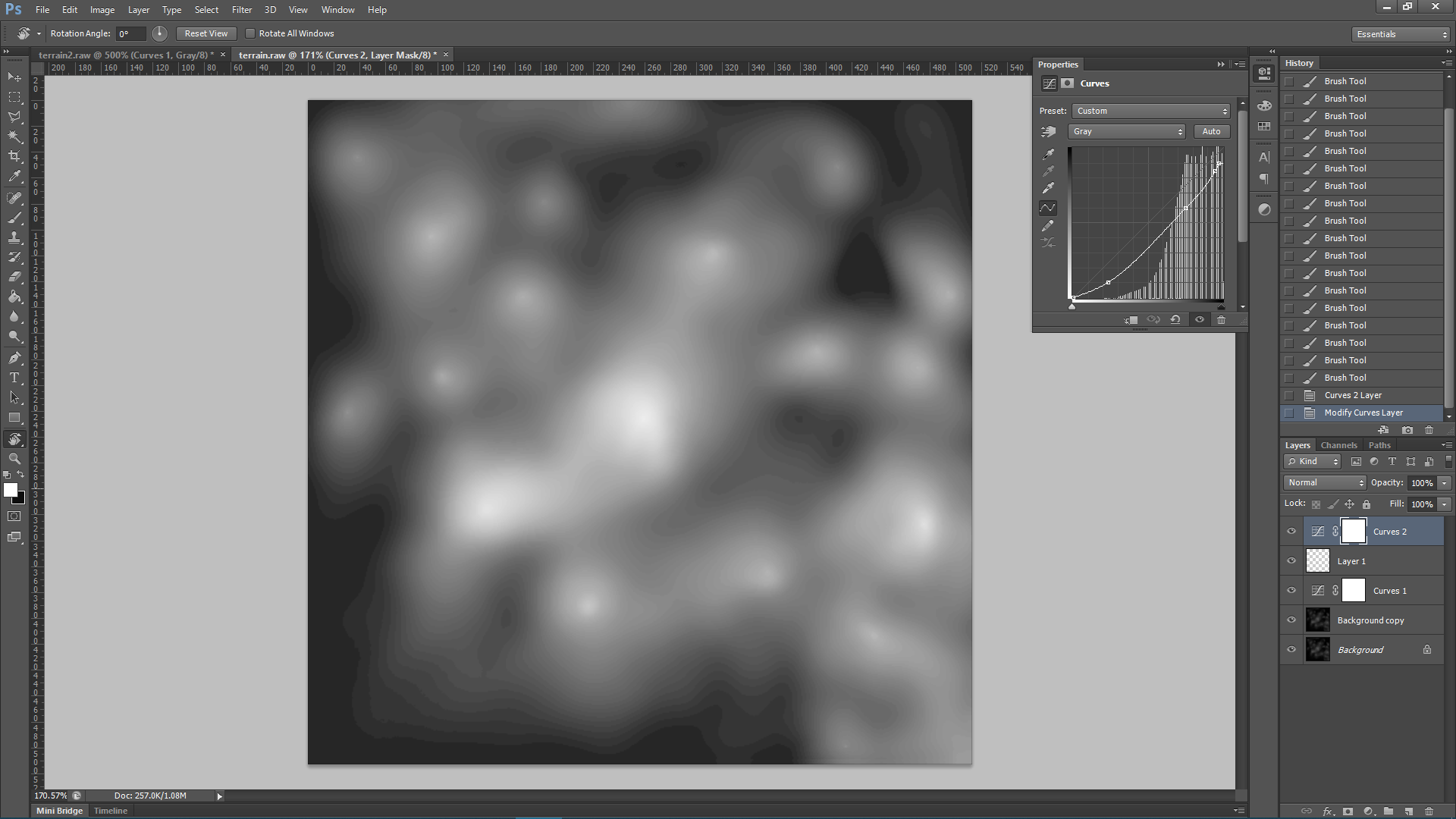
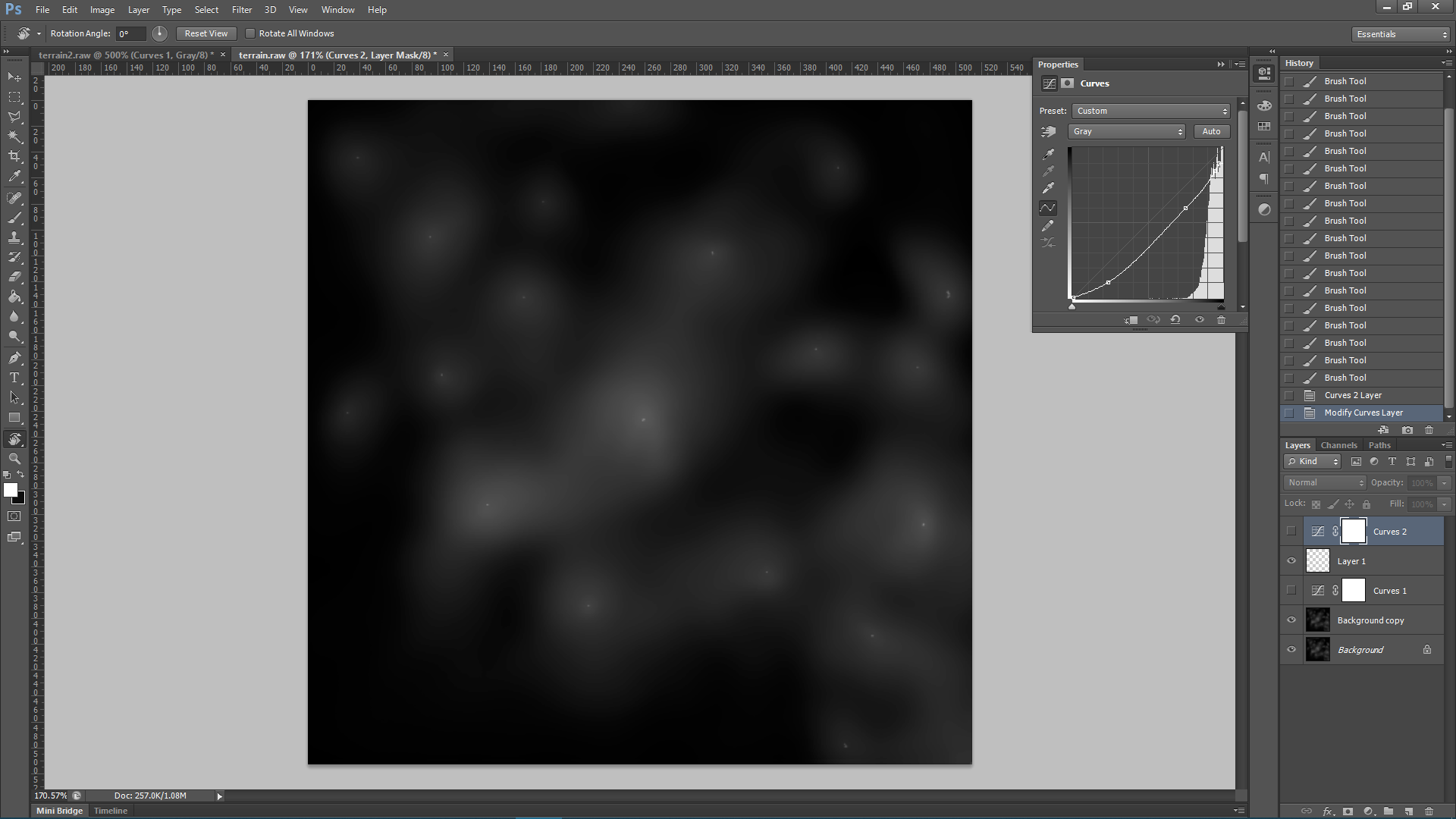
The size of matrix is medium, which is 513\*513.

## Procedure

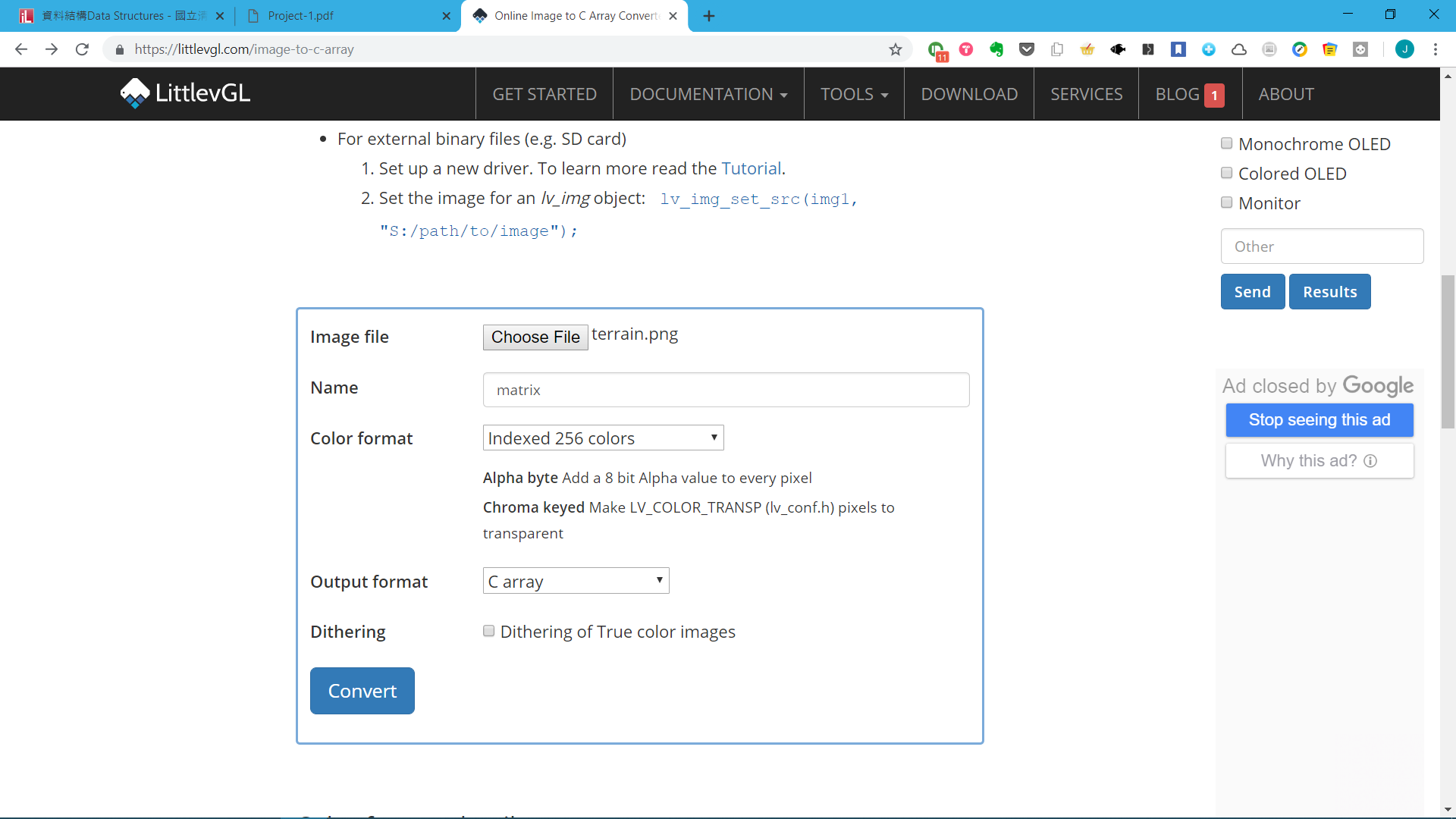
1. Design the model using terrain tool in Unity



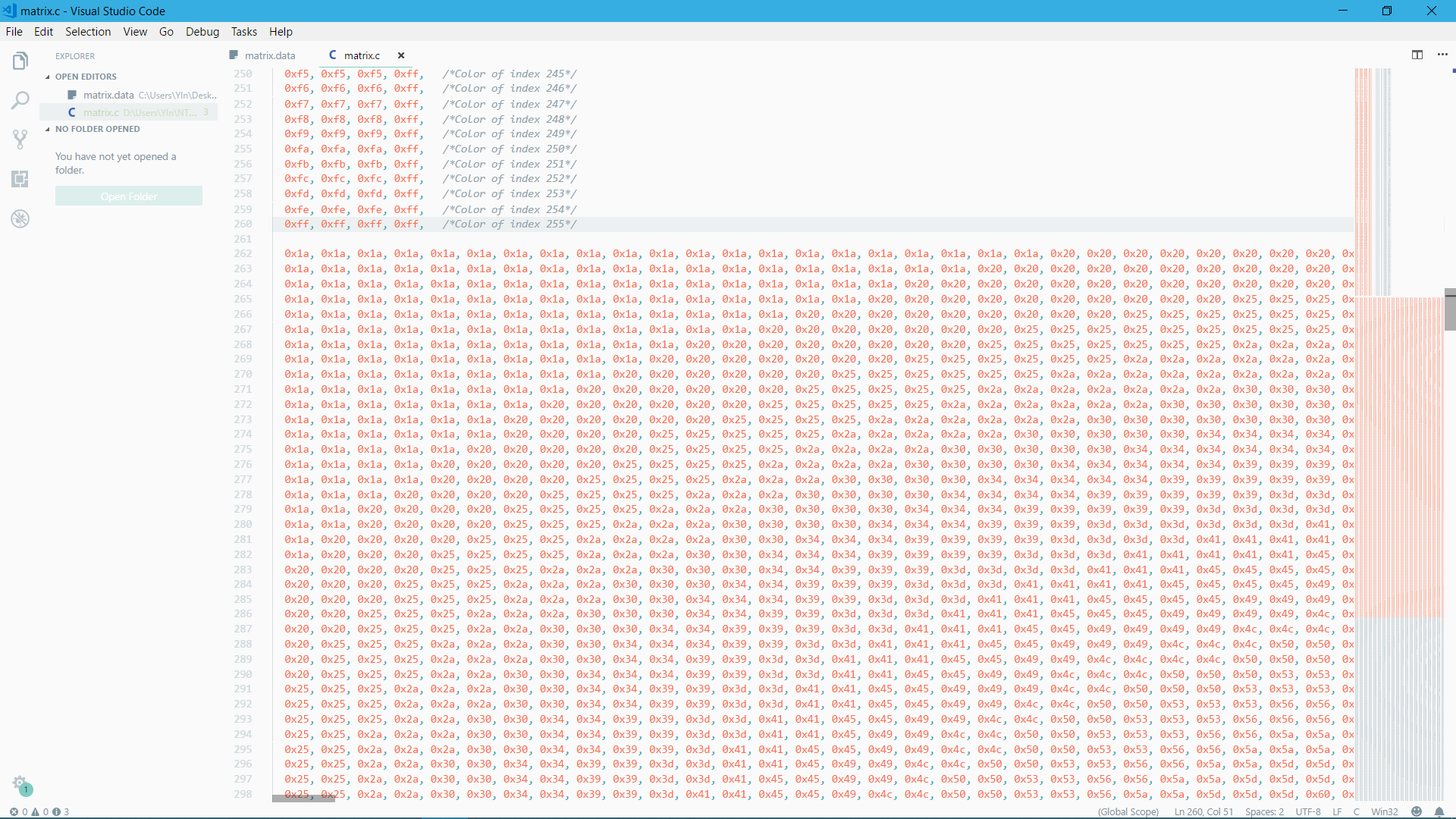
1. Export heightmap as raw image
2. Edit image in Photoshop to amplify the contrast



1. Use the website below to turn image into c-array according to its RGB value of each pixel: <https://littlevgl.com/image-to-c-array>

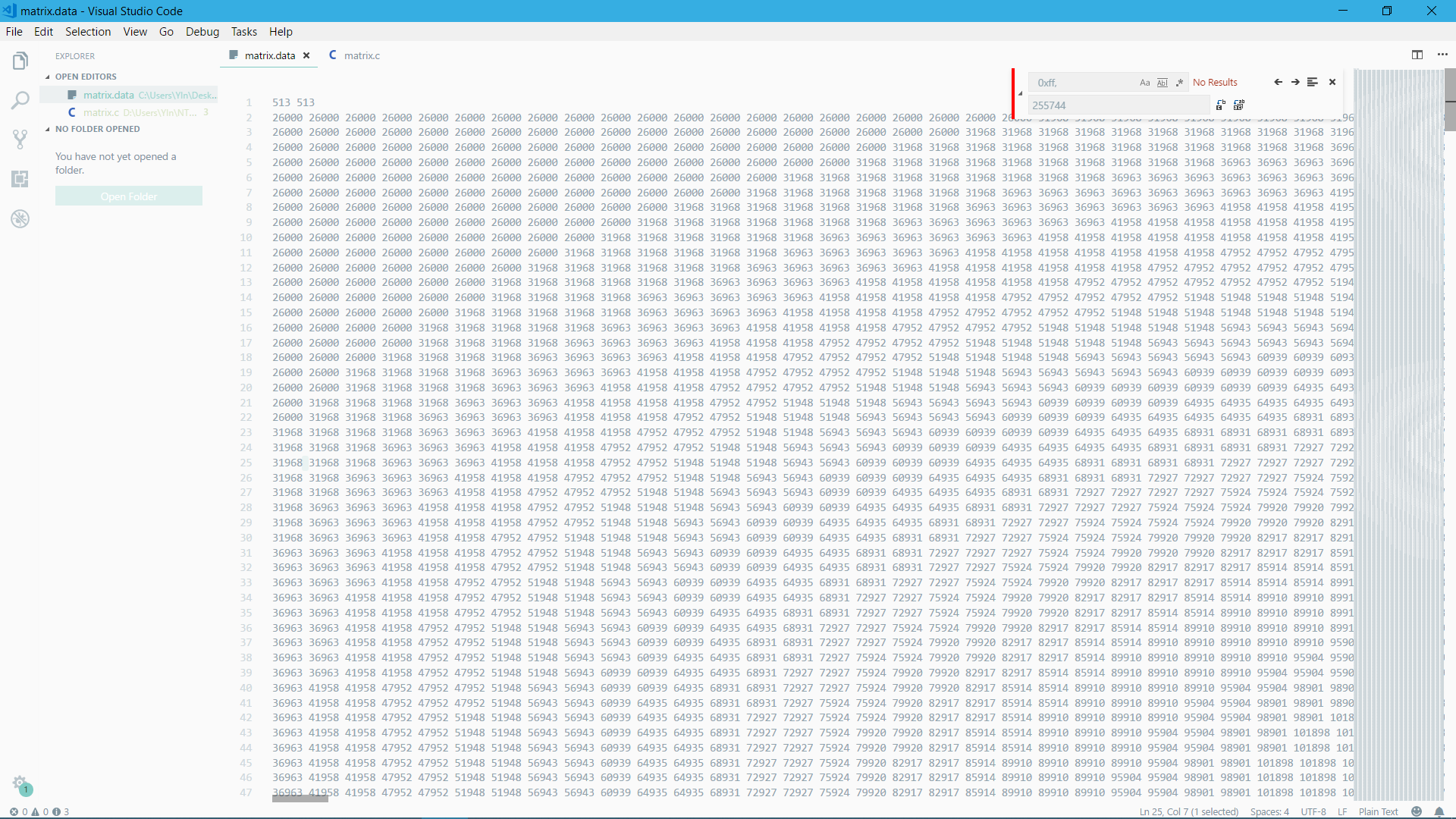


1. Output:



1. Convert in to long integers using replace tool

f(N, numbers in hexadecimal form) = todecimal(N) \* 999



1. Correct the format and save as matrix.data