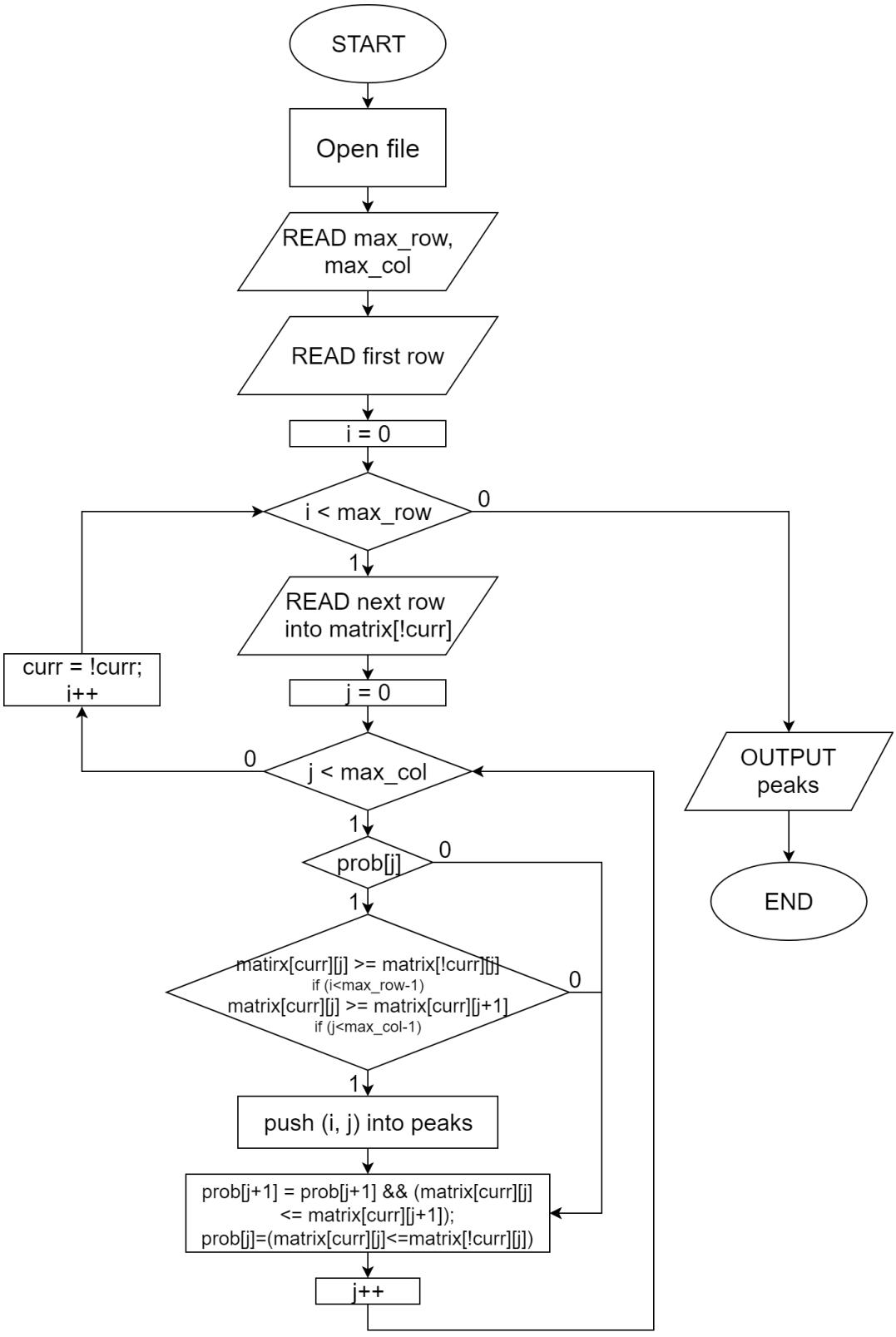
# Project #1: Peak Finder

## 1. Project Description

### 1.1 Program Flow Chart



## 1.2 Question Analysis

A point $h_{i,j}$ is classified as peak if:

$$h_{i,j} \geq \begin{cases} h_{i-1,j} & ,if\ i>1 \\ h_{i+1,j} & ,if\ i<m \\ h_{i,j-1} & ,if\ j>1 \\ h_{i,j+1} & ,if\ j<n \end{cases}$$

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

## 1.3 Development and Optimization

### 1. 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`

## 2. 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

## 3. 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

## 1.4 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[]) {
```

2. 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";
    }
```

3. 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;
```

# 2. Testcase Design

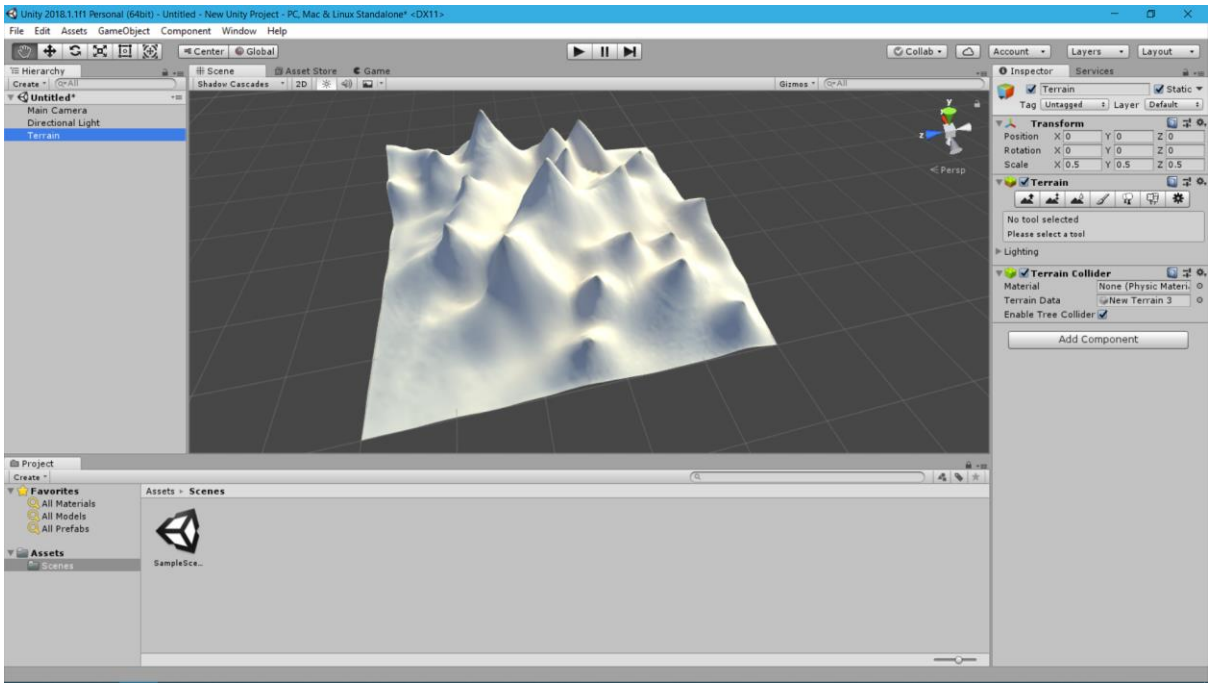
## 2.1 Design Concept

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.

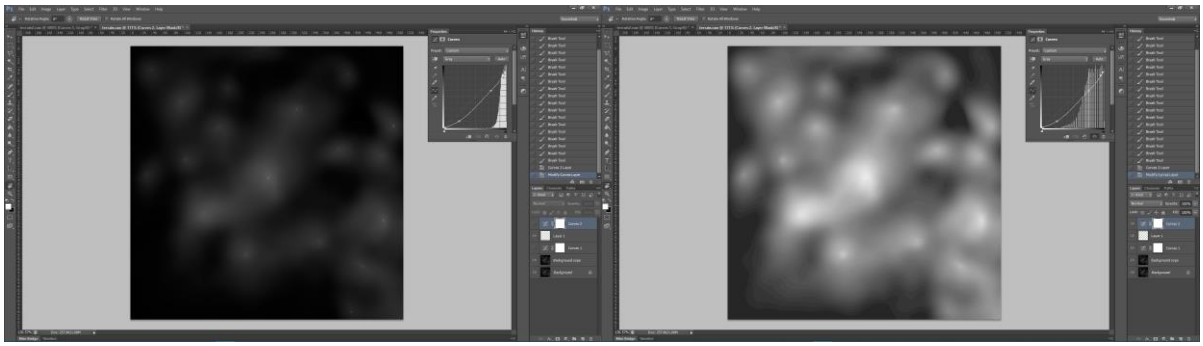
The size of matrix is medium, which is 513*513.

## 2.2 Procedure

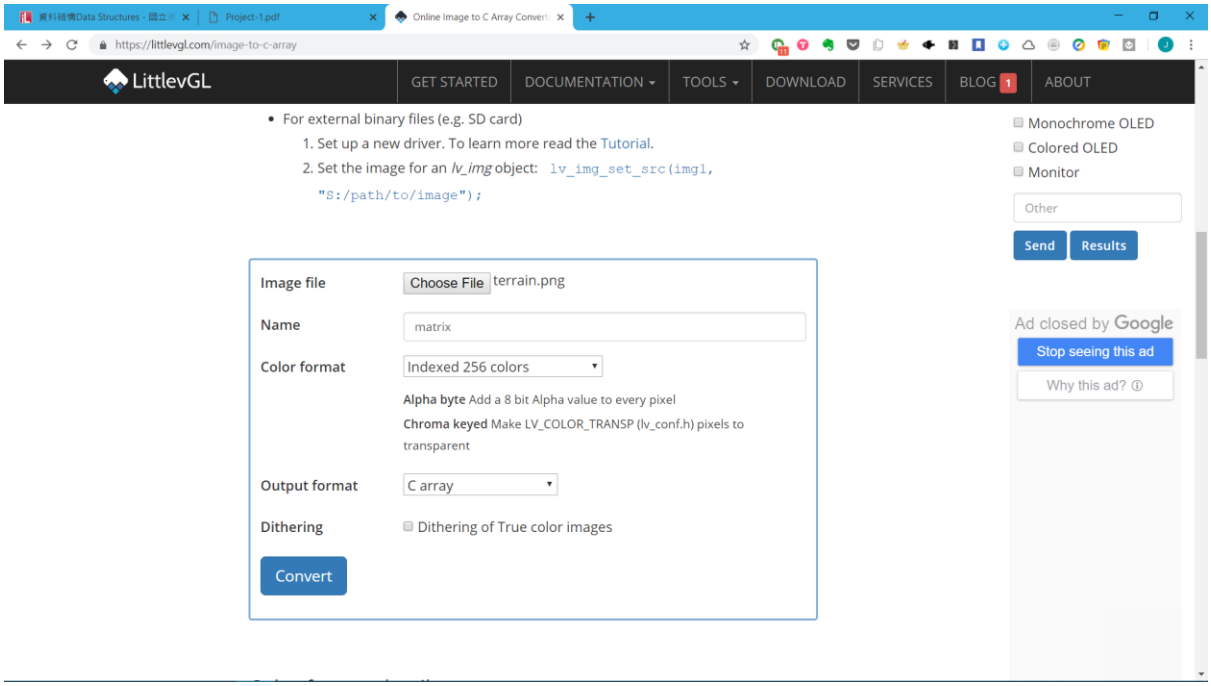
1. Design the model using terrain tool in Unity



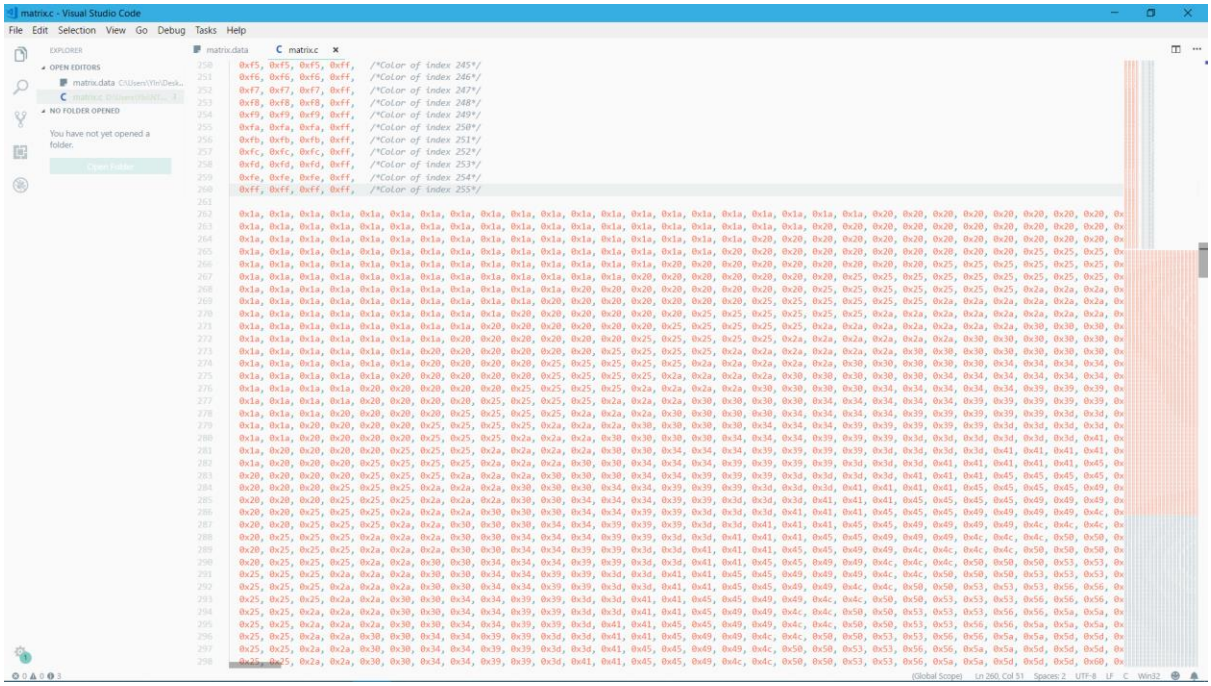

2. Export heightmap as raw image
3. Edit image in Photoshop to amplify the contrast



4. 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

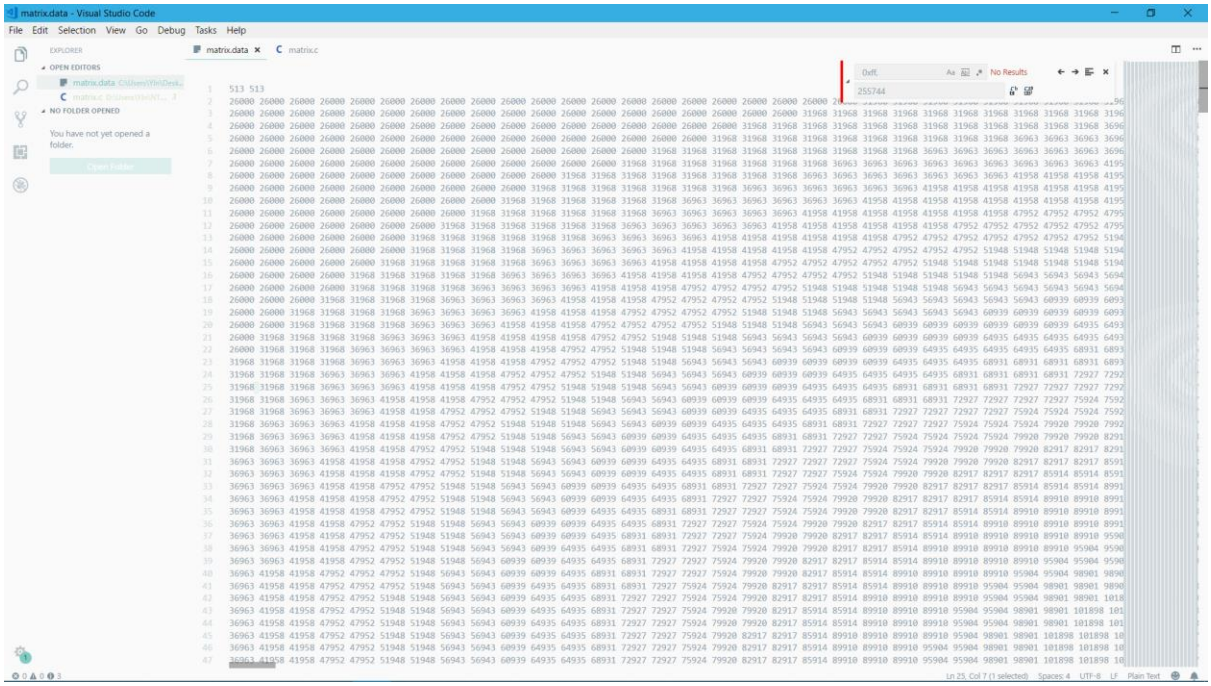



5. Output:




6. Convert in to long integers using replace tool

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




7. Correct the format and save as matrix.data