Lidar Processing

Ethan Zhou and Jake Adicoff

https://github.com/bowdoin-gis-f17/project5-lidar-jake-ethan

0. Key Press Controls

Initially, the rendering will show the grid with points that we have classified as ground. Ground is red-brown. To toggle between ground classification and hill shade, press 's'. While in ground classification mode, hit '+' or '-' to change the building slope threshold. This makes the ground search algorithm more or less strict when classifying points as ground. To rotate about the *x* axis, press 'x' or 'X'. To rotate about the *z* axis, press 'z' or 'Z'. To quit, press "q".

1. Gridding Lidar Points

We begin with a vector of lidar points, which contain an *x,y,* and *z* value, the point's return number, and the number of returns at that x,y point. For creating an elevation grid out of the first return points, we begin by only considering first return points. We take a density parameter as input, which specifies the average number of points in each cell of our grid. Using this parameter, we determine the number of grid cells, with which we construct a grid with appropriate height (rows) and width (columns) dimensions. We also store a Δ value, which is the width of each grid cell. After we have created the vector to store the elevations of each grid cell, we 'bucket' the lidar points into each grid cell by dividing each point's *x* and *y* values by Δ. This gives us the grid cell that the point belongs in. After we have bucketed all points, we take the average elevation of all points in each cell, and we make this average the elevation that corresponds to the cell. If a cell has no grid points, we define put the NODATA value in that cell.

2. Hill Shading

For hill shading, we used a technique that we believed would be the easiest to understand and implement. From each index pair *(i,j)* corresponding to a grid cell, we define two right triangles that cover the grid cell. We find the elevations of each vertex of the two triangles, which will be at the following points: $\{(i,j),(i+1,j),(i,j+1)\}$ and $\{(i+1,j),(i,j+1),(i+1,j+1)\}$. With these elevations, the triangles define two planes. To calculate the shade of a triangle, we take the dot product of a constant "sun" vector and the normal vector of that triangle. The result of the dot product is a value from 0 to 1, which is how bright the triangle should be. This makes intuitive sense because the dot product tells us how "aligned" the two vectors are, and the more aligned the vectors are, the brighter the triangle should be.

3. Labeling Ground Points

We did not label lidar points as ground, as we needed to be able to find ground point's nearest neighbors. There is no efficient way of determining locality with a set of lidar points, so we chose to find ground using a elevation grid instead, because we can easily find the nearest neighbors of a grid cell.

4. Constructing a Ground Grid

To determine which cells in an elevation grid are ground, we begin at the cell with minimum elevation. From this cell we run a breadth first search through cells in the four compass directions, labeling each unvisited cell as ground until we encounter a point such that the slope is greater than a user-defined threshold value. We label this steep point as building and,

we continue the BFS from this building cell, but classify newly discovered cells from this point as building. The BFS terminates on negative slopes, already classified cells, or NODATA cells. Upon finishing one iteration of this BFS, we search for the the minimum elevation cell among all unclassified cells, and begin the same ground definition sweep from this new point. We repeat this process until all points are classified as Ground or Building. This allows us to ‘blob’ the ground and buildings, and since we end our BFS before going down in elevation, it is a nice elegant way of taking care of the ‘courtyard’ problem. In practice, this algorithm is not 100% correct, as some ground points are classified as building and vice versa. We believe this is due to the fact that some ground has steep slopes, and some buildings have some walls that are gentle enough for our search to “climb” up the side and classify roofs as ground. The user also has to be careful in selecting a good slope threshold, otherwise the classification will be inaccurate. Given a good slope threshold value, this algorithm classifies the vast majority of points correctly.