Rendering Grids from LiDAR Points and Approximating Ground Points

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

Duncan Gans

November 18, 2017

1 First Return Grid

One key functionality of the program is the ability to visualize the first return LiDAR points. These are the points where the laser either only had one return on the surface, or it is the first of the LiDAR returns. The first returns represent the highest point in an individual x and y location. Therefore, the first returns show every building and tree as well as any exposed ground. The resolution used for the grid is preset to 200. This value seems to work visually, and accommodates the five tests I did well. However, because grid size changes, it does not line up perfectly with a set distance between the grid points. In practice this means that each grid space scales to the overall size of the grid. This grid resolution is what determines the standardized z unit for creating the ground grid discussed later. Therefore, by changing the grid resolution too much you will change how the grids are calculated.

This grid should be automatically rendered upon the running of the program. To switch between hill shading mode and shading by height, one can click n. In an effort to not plagiarize any of the code that was online or had previously been used, I made my own, more simplistic hill shade. It pays attention to which side of a point it is focusing on, and shades the surface according to the slope. The image below (Figure 1) outlines two different hill shading methods on the Zurich LiDAR data.

2 Finding and Rendering Ground Grid

The crux of this project was creating a function that could determine what points in the grid were on the ground, and rendering it accordingly. Although methods vary, and there are better, and

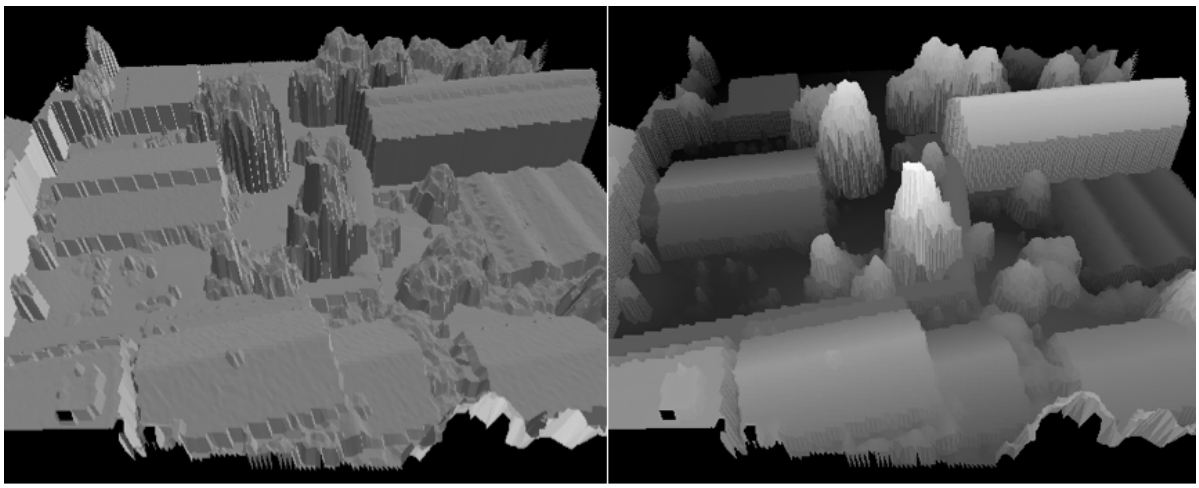

Figure 1: Various shading methods on Zurich. Hill shading on left. Height shading on right

certainly faster, methods of rendering the ground grid, mine works relatively well. I will discuss its strengths and weaknesses later.

2.1 Generating the Ground

My function works mostly by doing a series of steps to constantly refine and improve on an initial set of ground points. These general steps are outlined below.

1. Find set of connected ground points and label a series of non-ground points
2. Using the known non-ground points, try to find more non-ground points
3. For points that are not categorized, check if they could be ground
4. If they are likely ground, find surrounding points that could also be ground
5. Eliminate clear outliers that are technically marked as ground but probably not
6. Induce building points and other non ground points to find out what they should be
7. Erode the ground a final time to get rid of any non ground deformities in the map

By using multiple steps, any natural gaps in one step will be filled by another. This makes it so that the function is able to work well on all four of the grids I tested on (Fusa, Zurich, Sample, and Lake).

2.1.1 Finding Initial Ground Points

The first step is finding an initial set of ground points. To do this, it makes the assumption that the lowest point on the grid is ground. Although not technically a given, the exceptions are rare enough it seemed to be a fair assumption. From there, I use a recursive function, recurseGround(i, j), that spreads outwards from a known ground point and finds fellow ground points based on a maximum allowable slope. If it comes upon a place that clearly is NOT ground, it marks it as being not ground, and does not recurse there. At the end of this step there is a selection of known ground grid spaces, the size depending on the individual graph.

2.1.2 Finding some buildings

Next the function takes those points that are determined to be not ground and uses elimGround(i, j) to find other non-ground points. This function works almost identically with recurseGround(i, j) but with a slightly different allowable max slope. Using this, the function is able to get a decent approximation of all of the buildings that are at least connected to the initial ground. Knowing that these grid spaces are buildings rather than possible ground points helps out with later functions.

2.1.3 Categorizing Uncategorized Points

This function takes all the points that have yet to be categorized and checks if they could be ground. To do this, from an individual point it checks in every direction to see if the slope to the nearest ground point is within an acceptable margin. If the relative Z distance between the two points is small enough, then it will be assumed to be a ground point. At this point, recurseGround is then run on that ground point to find connected ground points. This further fills in gaps in the grid by labeling both ground and non ground points.

2.1.4 Eliminating Outliers

This function was created after noticing that depending on the Z Interval there was sometimes outlying points that were significantly higher than could be potentially labeled ground. Although this was mostly fixable by tweaking my other functions, I created this additional function to get rid of any last bits of noise that will exist. To do this it finds the standard deviation of the points, and eliminates any points that are more than 3.2 standard deviations from the mean. In a binomial distribution this, over 99.9 percent of points should fall within this value, so it isn't eliminating anything except points that are almost certainly not ground points. This obviously can be tailored to an individual grid, but 3.2 shouldn't visibly affect any grids. But, as mentioned, in unique scenarios, some points are labeled as ground that shouldn't be.

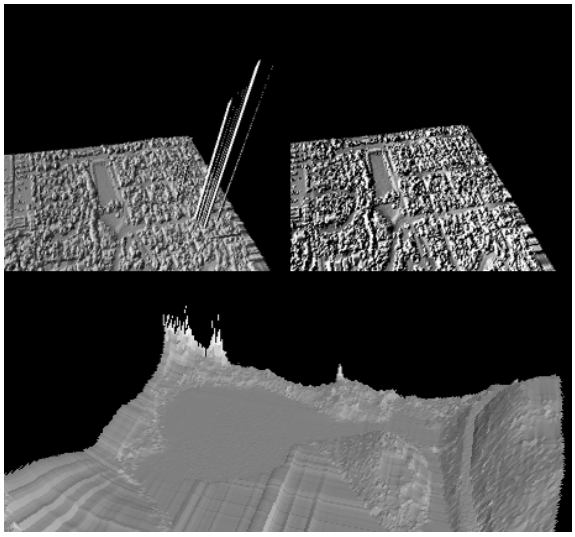

Figure 2: Above: With Incorrect LiDAR points as shown in the above example (on the right), the resulting ground render (on the right) is incorrect. Below: Points that lie on the edge of a grid have less points to infer from, and are often wrong.

2.1.5 Induce Non Ground Points

Here, points that are not ground have their z values induced for them. To do this I use a weighted average of the points that surround the non ground point in every direction. The weight is based off of an inverse distance. Therefore, the points that have the highest inverse distance (the lowest actual distance) contribute to the average the most. The actual details of how this works can be seen in my program. By using this method, the best approximation for an individual point can be found. Furthermore, by doing it this way, it makes it so that visually there are less sharp inclines and declines that could otherwise be created by alternative methods of inducing points.

2.1.6 Final Ground Erode

This final steps recursively goes through the final grid and finds any anomalies in the results. Once again there is a maximally acceptable Z value difference between two points, and if that is exceeded, the point is set to the smaller point. This eliminates points that are clear outliers. However, because the function ends when it gets to the edges, it often misses out on getting some outliers on the edges. This will be discussed later in a larger discussion of problems with my recursive algorithms. However, for the middle, it does a good job of creating a more accurate and visually reasonable slope.

2.2 Strengths and Weaknesses

Obviously any program designed to filter through only ground points faces the possibility of an error. Regardless of how good an algorithm is, it will inevitably categorize some ground points as not ground, and some not ground points as ground, especially when faced with edge cases in terrain. Overall, creating a function that finds the ground is a balance. You want to be liberal with allowing ground points to be categorized as ground, while being careful to not categorize any non ground points as ground. To avoid finding the perfect balance, I instead used a more conservative approach with multiple checks and balances to determine what was ground and what wasn't. Furthermore, although I was certainly able to accommodate my z values to a certain grid to make it next to perfect, I had to make sure that any method would work for other grids. Therefore, the grids aren't perfect.

The function does a pretty good job of finding connected ground points, and so when there is a lot of ground points connected to the initial lowest ground point, the ground looks like a very accurate representation. When filling in points with inducePoint, it looks a little less natural. Another strength is its ability to deal with both trees and buildings. From what I've done, the function does no worse on finding buildings than on trees. This is partially because my multi

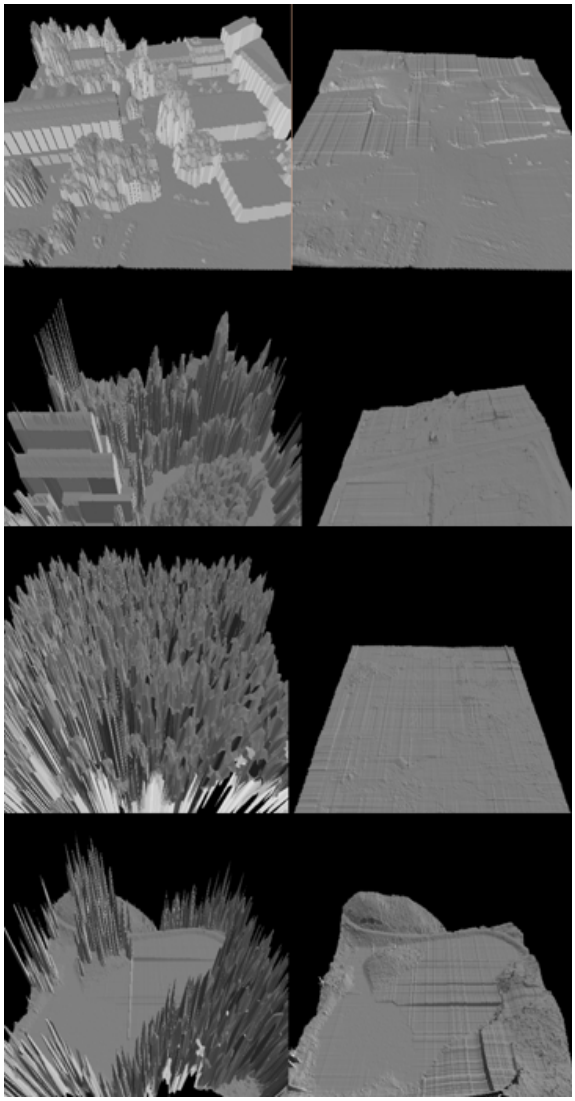

Figure 3: Renderings of Zurich, Fusa, Sample, and Lake

layered approach. Initially after finding the non ground points, it was easy to find connected non ground points if the non ground points were buildings, but harder if the non ground points were trees. To get around this I did the additional check of looking at unclassified points afterwards to see if they were ground or not.

However, there are some weaknesses with my methods. One of these is the edges of the graph. Because many of the methods are recursive (and I don't want to recurse out of the grid), or base inferences by nearby data, the grid spaces on the edge have less surrounding spaces to infer from. Therefore, in some graphs there are some non ground points that are considered ground. This can be seen in Figure 2. A second weakness occurs when there is an incorrect LiDAR point entered into the dataset. Since acceptable Z intervals and slopes are based on the total range in Z values, an incorrect Z value can mess up the ground grid (Also seen in Figure 2). Finally, I tried to build my algorithm to deal with different types of terrain, and some variety in grid sizes. However, it's likely that with really variant grid sizes the methods aren't as effective or visually appealing unless the grid resolution changes.

2.3 Renderings on four test grids

In Figure 3, one can see the varying success on four different LiDAR point clouds. The Grid Resolution as well as the Z Intervals are held constant. Obviously I could get relatively perfect results by tailoring these variables to the individual grids, but instead decided to make them more generalizable. As a result, none of them are perfect, but still provide a reasonable approximation

of the ground grid.

3 Conclusion and Closing Thoughts

In this project we were tasked with rendering LiDAR point clouds as well as determining ground points from LiDAR data. The project proved to be a fun challenge that required more creativity than simply implementing a known algorithm. Although my program for determining ground points was certainly not efficient or concise, I enjoyed solving it in my own, albeit worse, way. Furthermore, the multi-pronged aspect of the lab was a lot of fun, there was always multiple things to work on at once. I certainly appreciated the freedom inherent in the assignment, and it made it my favorite CS lab so far this Fall. I look forward to a final project that is potentially equally open ended.