# Rendering Sea Level Rise

https://github.com/bowdoin-gis-f17/project6-slr-duncangans1

Duncan Gans

December 8, 2017

## 1 Introduction and Background

For this lab, the goal was to see how varying levels of sea level rise influenced terrain grids adjacent to the ocean. At it's most basic level, it involved being able to enter a specific amount of feet, and see what parts of a terrain grid went under water. Determining impacts of sea level rise, and therefore algorithms like this one, are very important. As climate change continues its unabated march forward, rising temperatures will melt ice sheets, glaciers, and snow packs. Although these diminishing receptacles of frozen water will lead to more absorption of the sun's heat, another significant effect will be on the sea level. As water moves from frozen ice and snow to seawater, sea level could rise as much as 6 feet in the next 100 years. However, 6 feet of sea level rise doesn't affect everywhere equally. Although Brunswick will not be super affected, cities like Miami and Guangzhou, China, could face grave consequences. Using algorithms like this can show what areas will be most affected by sea level rise, and the point at which a given area will be flooded. These algorithms can answer real world problems.

## 2 The Algorithm

The crux of this project involves creating a function that can determine sea level rise at a given height. However, because I wanted to include the functionality of changing the sea level in the 3d rendering without any additional computation time, I had to pre-compute multiple levels. To make this possible I used a grid that at every location included the height necessary to flood that point. I created two algorithms that could do this. The first was a naive algorithm that at each level that the user requested, it would determine the effects of sea level rise from scratch. Although a single iteration was fast, it was slow when trying to find a variance of different sea level rise impacts. To change this, I created a second algorithm that iterates upwards and uses the coastline from the previous iteration to determine the flooding at the next level. This second algorithm was slightly

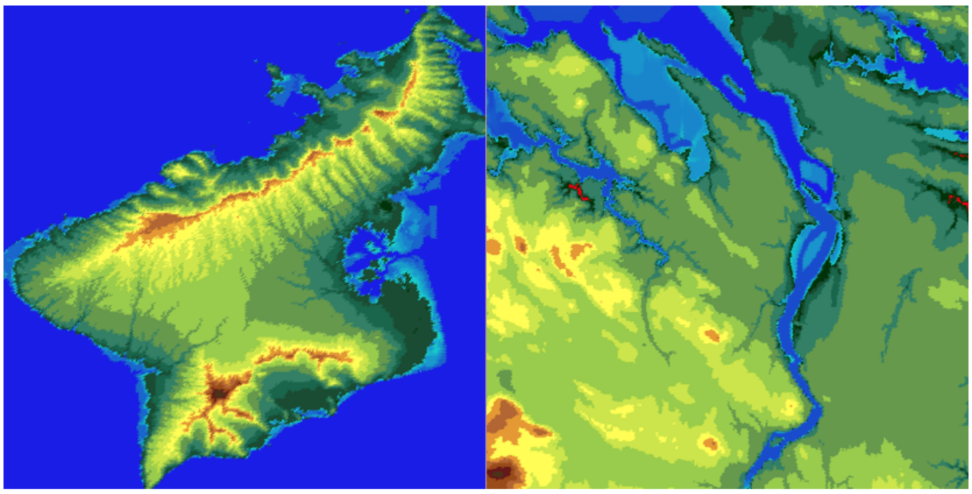

Figure 1: 6 Feet of Sea Level Rise on Oahu and in Brunswick/Topsham

slower at finding just one sea level impact, but took a fraction of the time with more variance in sea level rise.

## 2.1 First Algorithm

My first algorithm involved iterating downward from the ceiling height to the floor height. At each decreasing level it would flood the terrain, and every point flooded would be set to the current flood height. The actual flood algorithm involved starting with the ocean points at the edge. From there a depth first search is done by adding all neighbor points under the sea level rise to a queue repeatedly until every point under the given sea level rise connected to the ocean is looked at. This is done for each level of sea level rise. Although it worked, it was slow with lots of sea levels because each one was done independently.

## 2.2 Improved Algorithm

The second algorithm attempted to determine multiple sea level rises but only look at each point a constant number of times. It begins by flooding the ocean to 0 feet and determining all the coastline. At each edge ocean point that hasn't been looked at, it does a depth first search, using a queue, and moves all the non-ocean coastline into a queue.

Using this coastline queue it begins the recursive function that incrementally floods up the coastline. This function checks all the points in the current coastline queue. If the point is below the flood height, it's value in the flood grid is set to the current sea level, and the neighbors are added to the current queue. If the point is above the flood height, it means it is on the coastline, and will be added to the next coastline queue. Once all the points have been looked at, the sea level rise is incremented, and the next coastline queue becomes the current coastline queue. This continues until the ceiling height is reached. By doing this method, most points will be looked at twice. Once when first put in the coastline queue, and next when the point is flooded. However, if there is a point on the coastline that is significantly above sea level (this happens with steep terrains) it will be considered multiple times and not leave the queues until it is flooded. Other points are only looked at once. If a point is in a depression, or in a flat area not next to the coast, it will never become coastline, and as soon as the water hits it, it will flood it. With the faster algorithm, if the sea level rise was from 0 to 100 feet, each point would be looked at about twice instead of one hundred times. This saves a lot of time.

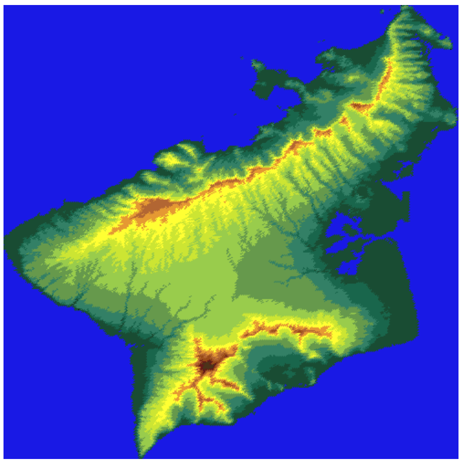

Figure 2: Oahu with bucket style color rendering

## 3 Implementation and Visualization

To actually visualize the points, the function visGrid() is used. Instead of creating two triangles at every point, it looks at one in every X points, where X is proportional to the size of the grid. This allows the visualization to be rendered instantaneously as the sea level rise changes and users rotate the grid. For coloring, the are three options for the color of the triangle. If the point is above

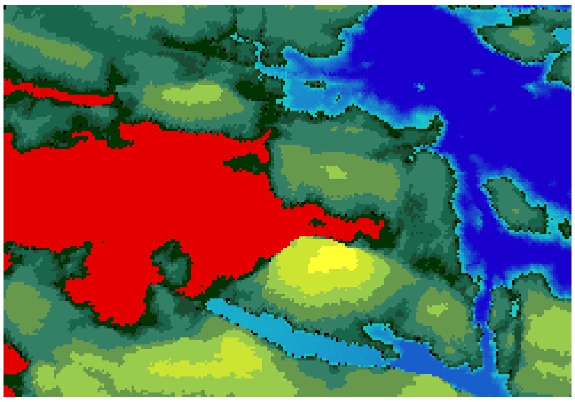

Figure 3: Bright red represents unflooded area below sea level

sea level, then its color will be based on its height by using a series of buckets. I could have used a linear color scale, but I wanted to create a wider range of colors and also give the impression of a topographical map. This scale can be seen in Figure 2, in the rendering of Oahu. Occasionally, a point is below sea level, but hasn't been flooded because there is a land barrier between the ocean and the depression. To make those points stick out, they are visualized as bright red. This can be seen in Figure 3.

Finally, to visualize the water and to give an idea of the underwater depth, I used a linear color scale from blue to cyan to indicate the depth of the water. It's default is to show variations in depth up to 10 feet, but this can be changed in the command line. Using this gives the water a more visually appealing and realistic look, while also showing the most recently flooded areas. Varying levels of underwater visibility can be seen in figure 4.

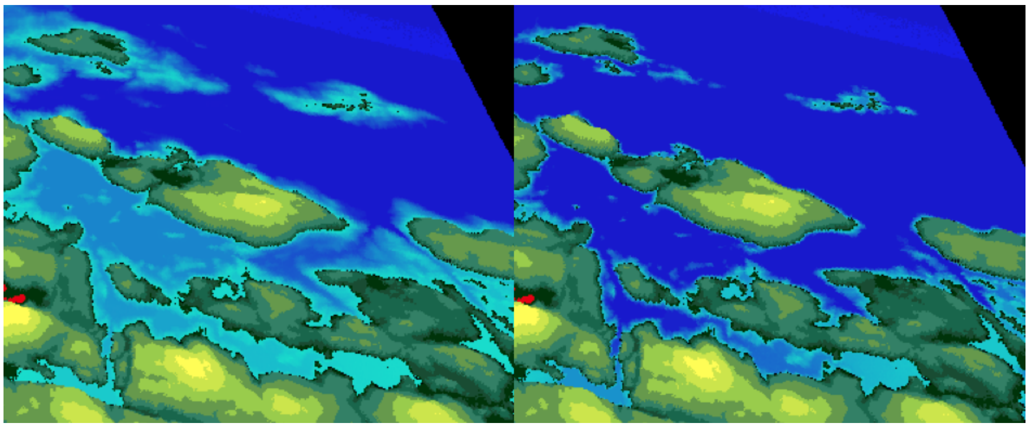

Figure 4: Left: Underwater Visibility set to 30 feet, Right: Underwater Visibility set to 10 feet

### 3.1 Experiments and Runtime

To test my code, I completely flooded five grids with varying increments. For all of them it worked well. As is described in README.TXT, when the increment is less than one foot, a lot of the tidal ranges aren't flooded. However, as long as the increment is at least a foot (including non-int floats) it works well on the tidal zones. Although I looked at all the grids for validity, to measure the run time I looked at just the Lincoln county grid because of its size. To get an accurate idea of the runtime of my program, I tested a sea level rise of 100 feet with an increment of one foot. This means I ran 100 sea level rise functions. With my old algorithm, determining the sea level rise at just 100 feet took a bit over a minute. Now, the entire function which tests all the sea level rises up to 100 feet finishes in just under 166 seconds, or a bit below three minutes. This means that each iteration of sea level rise finishes in under 2 seconds on average. This is a significant improvement from the previous algorithm.

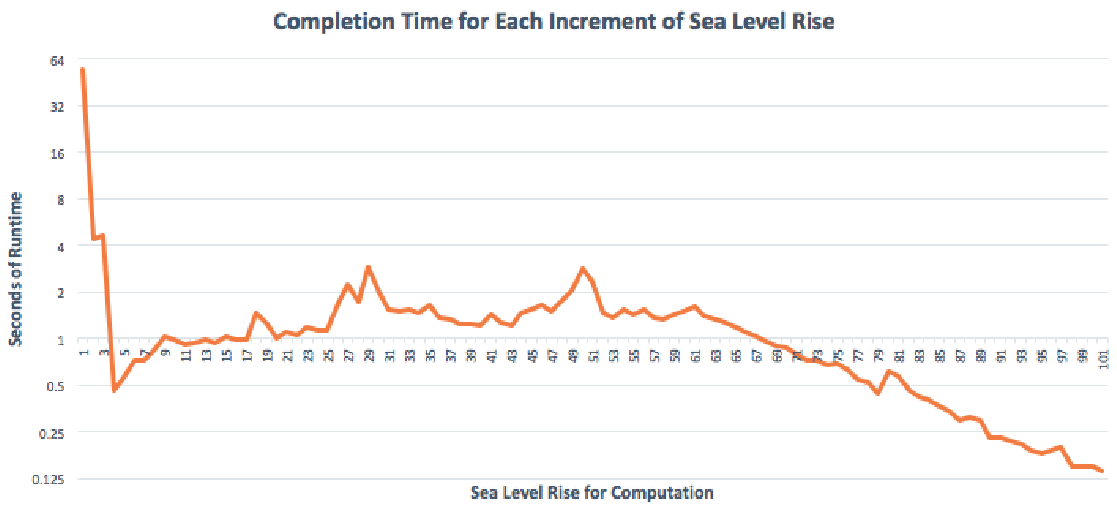

Figure 5: Runtimes at different Sea Levels

Figure 5 represents the completion times at each sea level rise. As one can see, it is certainly not a constant amount of time. Initially it takes a lot of time to find all the coastline because it needs to depth first search through the ocean. This alone takes almost a minute. The next two iterations of finding sea level rise are also extensive and take about 4 seconds each because there is a ton of tidal land under 1 and 2 feet. The more land susceptible to sea level rise at a height, the longer it will take to compute the sea level rise at that point. From this graph, it is apparent that there is a medium amount of exposed land from heights 3-30 feet, a larger amount from 30-60 feet, and a decreasing amount of susceptible land from 60 feet onwards. The spikes in the graph represent either depressions or flat areas where a lot of land was flooded with a minor increase in sea level. Table 1 provides an aggregate view of the computation times.

| Sea Level Rise | AVG Runtime (s) |
|---|---:|
| 0-9 | 6.93 |
| 10-19 | 1.06 |
| 20-29 | 1.66 |
| 30-39 | 1.4 |
| 40-49 | 1.75 |
| 50-59 | 1.46 |
| 60-69 | 1.09 |
| 70-79 | .69 |
| 80-89 | .34 |
| 90-99 | .18 |

Table 1: Runtime Table.

As for the actual grids themselves, those can be seen in Figure 6. Instead of showing every increment of sea level rise from 0-100 feet, I instead chose 0 feet, 20 feet, 40 feet, 60 feet, 80 feet, and 100 feet. This should give an idea of how sea level rise affects Lincoln County. As can bee seen from the graph, once 60 feet of sea level rise occurs, increasing the sea level much more has little effect. This is also apparent in the run times from above.

## 4 Conclusion and Closing Thoughts

Overall this project was a fun capstone experience to a great class. It was a very visual and real world project that was fun to implement. Once the function was finished, I found myself testing all of the different terrain grids endlessly and tweaking the visuals to create a better aesthetic feel. I also really benefited by beginning with a relatively naive, but correct, algorithm. This made it easier to later transition to a faster and more complex algorithm. Furthermore, in doing so I

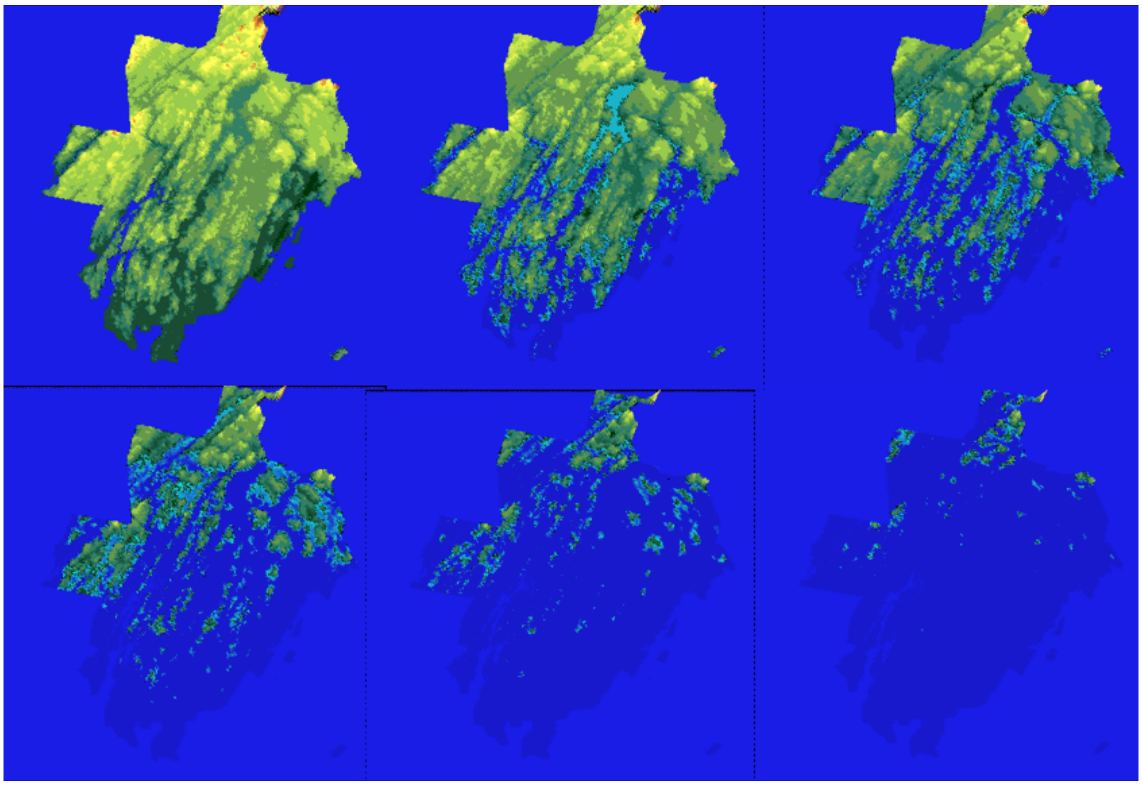

Figure 6: Lincoln County Flooding at 6 levels

gained a greater appreciation of the speedup that minor changes can have on overall run times. Although I certainly agree the last project was more difficult, this was a good one to end on for me, and served as a celebration of the GIS class as a whole. Ending on the LiDAR project could have been less of a celebratory end. I found myself very much enjoying the project, and not having to force myself to work at all.