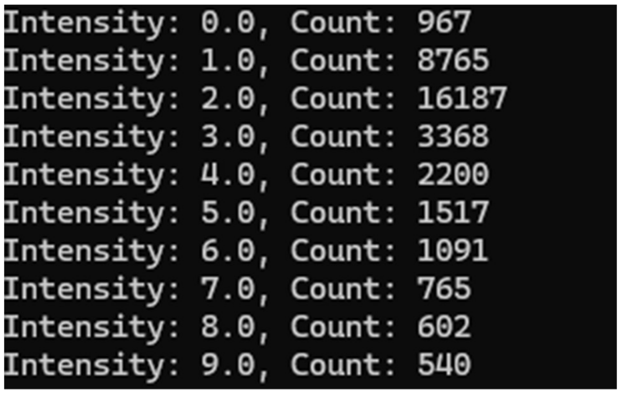
Ego Lane Detection

I elected for a crude, but functional approach to Ego Lane Detection. It is a solid basis that can be improved for complicated scenes with more curvature. **My approach can get decent detections if the ego car is moving in the same coordinate direction as the lane. My approach will struggle when the lane ahead curves significantly.** This is mainly due to how my region of interest is defined which is the main improvement that could be made to my approach. My approach filters the raw point cloud data through 3 filters. An intensity filter, region of interest filter, and a left-right split filter.

1. Intensity Filter

I realized that most of the road would likely be asphalt or dark. I concluded that lane line points would be a minority in the point cloud, and they would likely have a higher intensity. This is the justification for my intensity filter. A histogram or a print out of the count distribution shows which intensities we could potentially filter out.



```
Intensity: 0.0, Count: 967
Intensity: 1.0, Count: 8765
Intensity: 2.0, Count: 16187
Intensity: 3.0, Count: 3368
Intensity: 4.0, Count: 2200
Intensity: 5.0, Count: 1517
Intensity: 6.0, Count: 1091
Intensity: 7.0, Count: 765
Intensity: 8.0, Count: 602
Intensity: 9.0, Count: 540
```

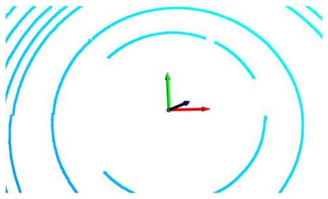
Intensity count distribution

Under the assumption that the asphalt, low intensity points would be the majority I decided to filter out the lowest intensity points. My methodology for picking specific values to filter out followed a trial-error approach. I examined the distribution for each point cloud and decided to filter out the highest count, examples like the one above show that filtering out intensities 0-2 makes sense as the intensity of 2 has 16,000 points, which is significantly higher than any other intensity. This makes it a good candidate for representing asphalt. I also decided to filter out the highest intensity points, as I believe these would be things like mirrors or cars.

2. Region-of-Interest Filter

After attempting lane detection on the intensity-filtered point cloud, the polynomial fits were still not making sense. After some research, I realized that due to outlier points far from the ego car, the polynomial fit was being heavily skewed. This led to the region-of-interest filter implementation.

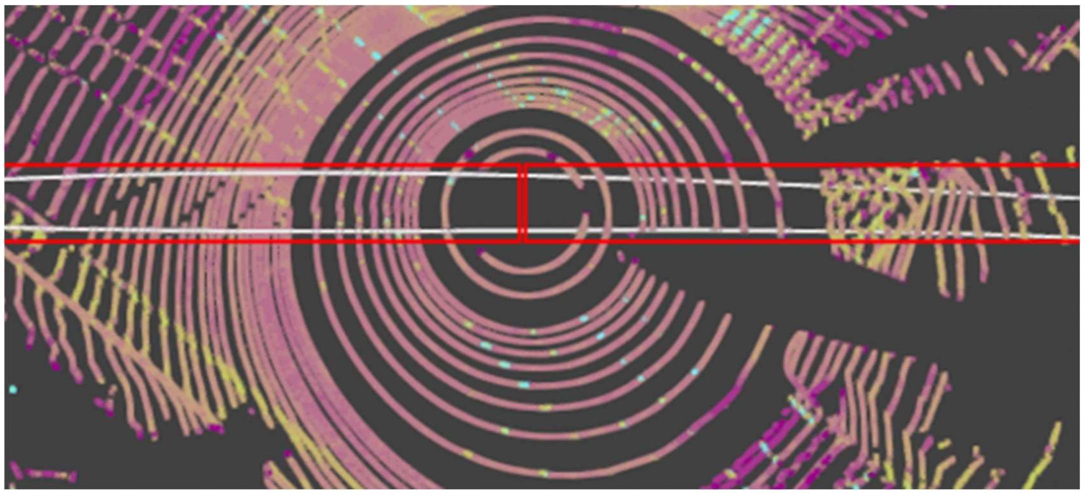
Because this is ego lane detection, we should only care about the closest right lane line and left lane line. This is the basis upon which I defined my region of interest. A google search told me that the average lane line width in meters is around 3.5-3.8 meters in most countries. I tried multiple values but settled on 3.5. The reasoning is that if this is the lane width, the closest lane line will be within 3.5 meters to the right or left from the car, no matter what. **The point cloud data seemed to mostly already be on the ground plane, so I chose not to do any z-filtering.** So, we only have to define a region of interest on the x and y planes. **We can use 3.5 to define our y or right-left region-of-interest bounds**. For certain scenes where the cars coordinate direction was significantly different from the lane's coordinate direction, I considered a larger left bound to try to capture more of the curvature. For the x forward-backward bound, **I followed a trial-error approach, generally xbounds of 30 meters – 40 meters seemed to perform well.**



Coordinate system of ego car and point cloud

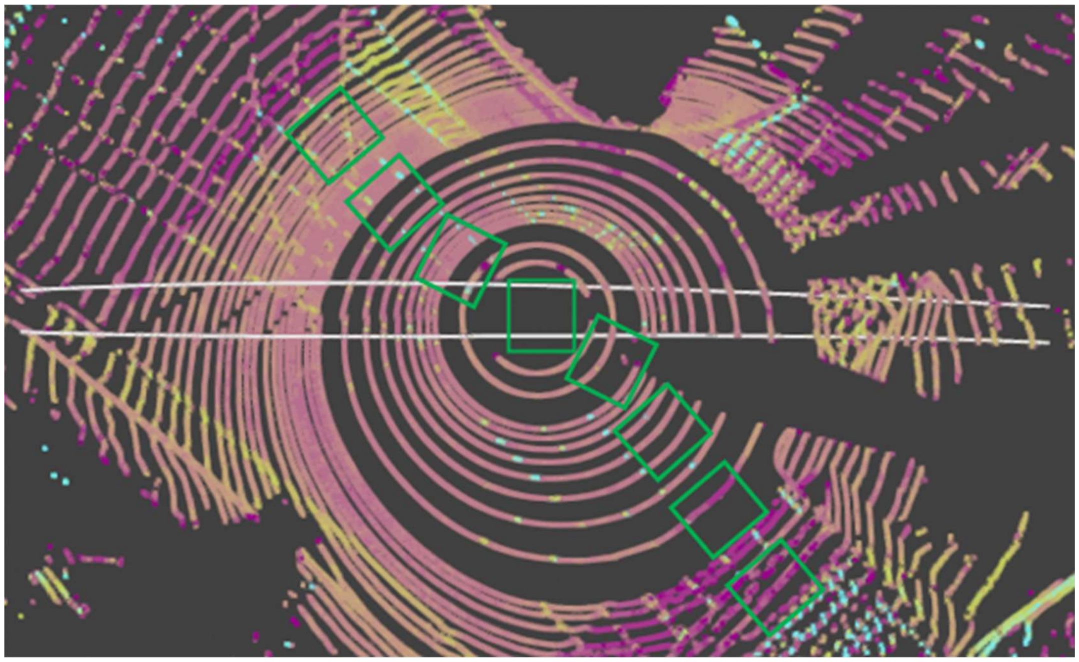
2.1 Limitations and Future Work

The primary limitation of my approach is this region-of-interest filter. Right now I'm defining my region of interest as 30-40 meters ahead and behind the car and 3.5 meters left and right of the car. This roughly looks like the red rectangular region below. With this visualization, it's easy to see why this approach performs poorly on scenes with significant lane curvature. The filters are based on the car's coordinate direction so the majority of the "correct" lane points are filtered out!



Current Region-of-Interest

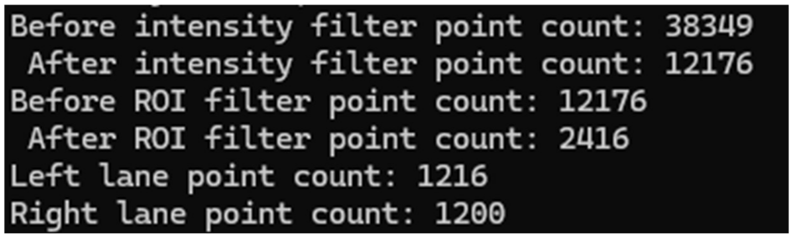
A potential solution for this problem, which I was unable to implement in time, is adaptive windowed region-of-interest filtering. **Instead of 1 big filter defined in the ego car's coordinate direction, we could consider smaller windows at a time, get initial lane detections in this smaller window and rotate the next window to match the previously detected lane direction in the previous window.** A rough visualization of the approach can be seen below. This will likely significantly improve performance in scenes with significant lane curvature.



Proposed Windowed Region-of-Interest approach

3. Left-Right Point Cloud Split

The car coordinate frame has x defined as forward and y defined as left. After the region of interest window filtering, I simply took all the points with positive y and defined them as the left lane line point cloud, and negative y as the right lane line point cloud. After all this filtering the number of points considered decreased by more than a magnitude of 10.



```
Before intensity filter point count: 38349
 After intensity filter point count: 12176
Before ROI filter point count: 12176
 After ROI filter point count: 2416
Left lane point count: 1216
Right lane point count: 1200
```

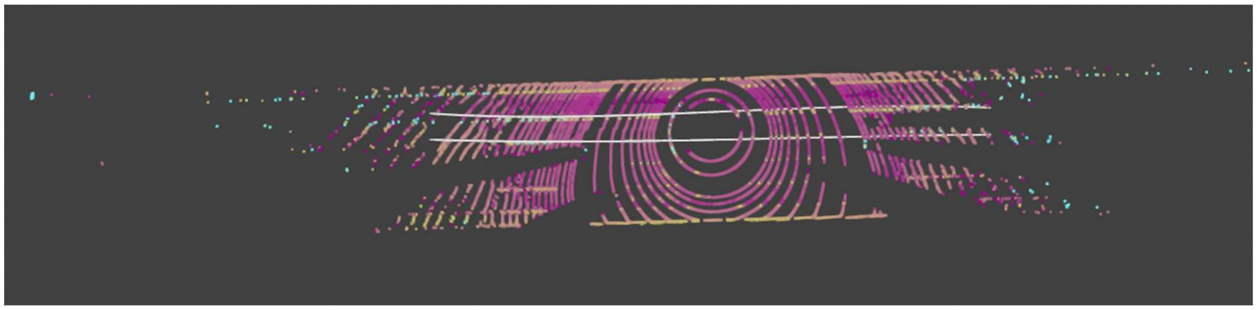
Point count order of magnitude smaller after filtering

4. Polynomial Fitting to Left and Right Point Clouds

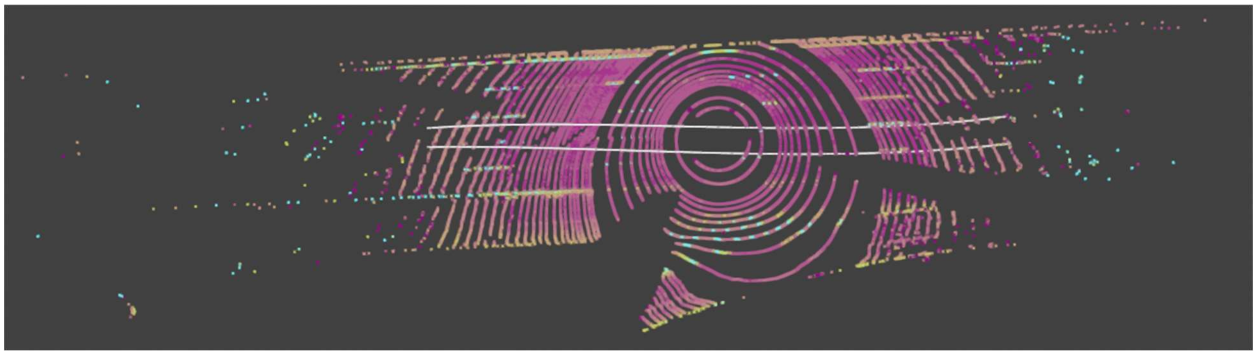
Finally, I fed my left point cloud x values and y values to np.polyfit with a degree of 3. I did the same for the right point cloud values to get my left lane line fit and right lane line fit. These are the 3rd-order polynomials that I used as my outputs. Please see visualizations below. Note, instead of np.polyfit, I could use RANSAC here for a more computational approach to fitting. I experimented with RANSAC, but due to the limitations of my region-of-interest filter I found that RANSAC did not make a significant difference, so in that case I opted to keep np.polyfit as I always try to keep solutions as simple as possible.

5. Conclusion

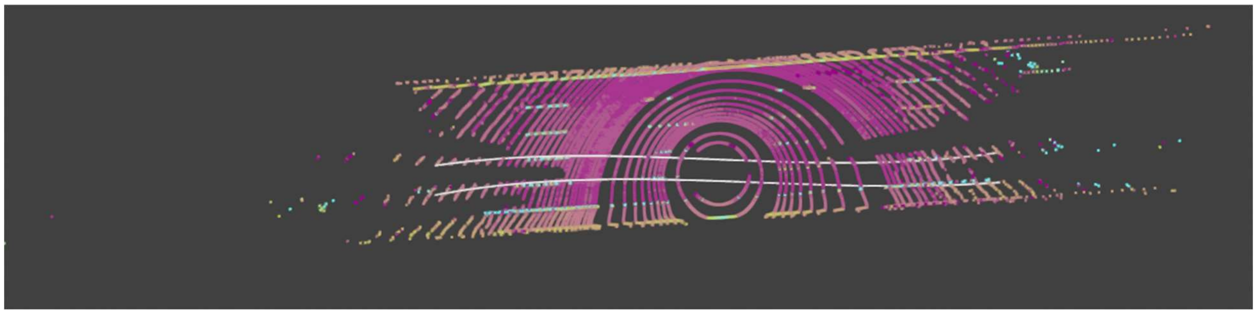
Overall, I learned a tremendous amount about lane detection and the features that we should care about for that problem. This was my first lane detection attempt, and I had a lot of fun. I was a little disappointed that I couldn't get perfect detections but given that I had to keep up with my other projects and responsibilities while working through this, I think I reached a good base solution to build from. I feel that with some more time and maybe some guidance, I could implement a better region-of-interest filter and get better results. If possible, I would love to talk my solution over with an engineer to see how to improve it. Thank you very much for the opportunity.



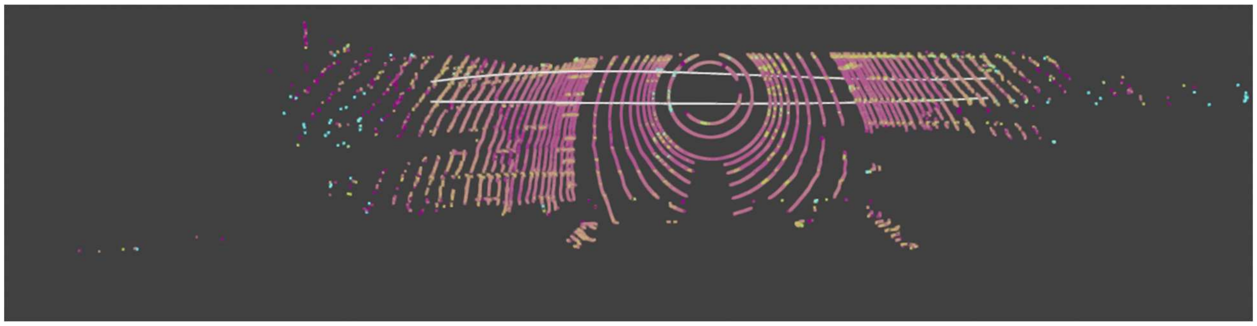
Scene 0: 1553565729015329642



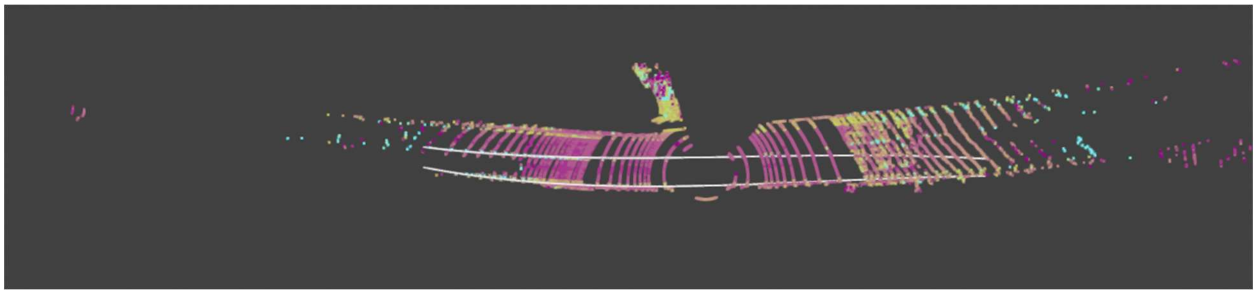
Scene 1: 1553565776121143870



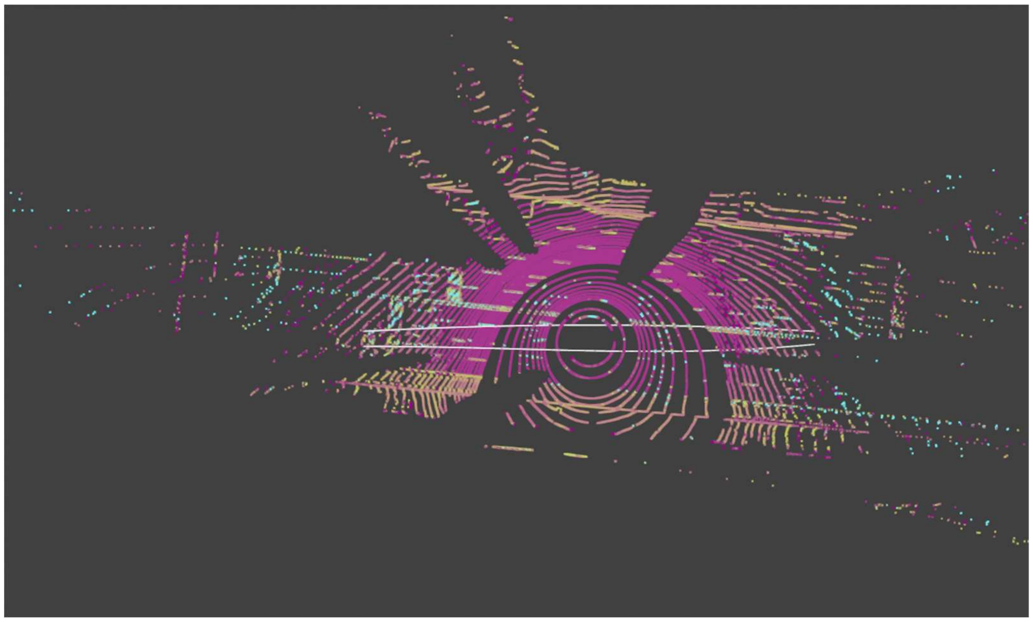
Scene 2: 1553565884136495856



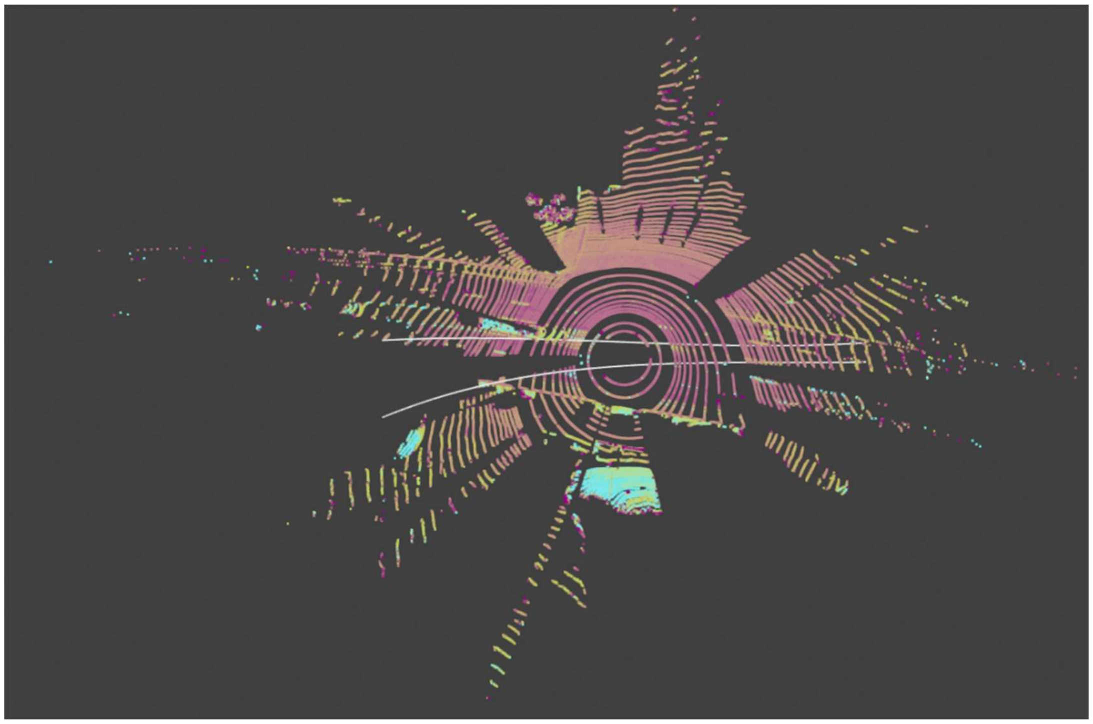
Scene 3: 1553567105504169477



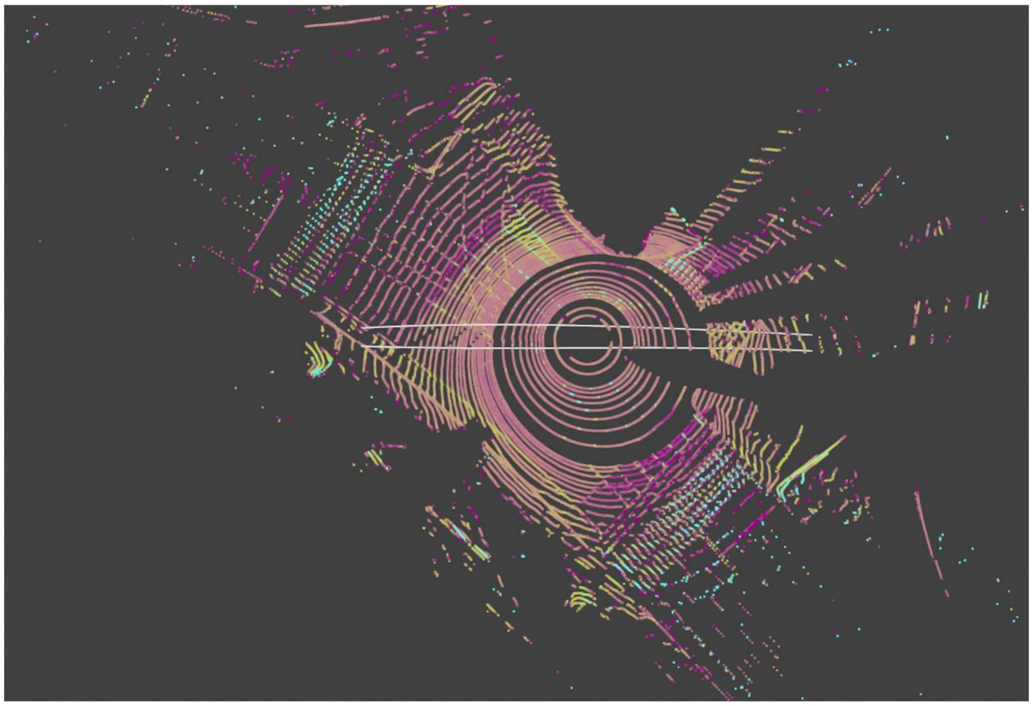
Scene 4: 1553669108359991937



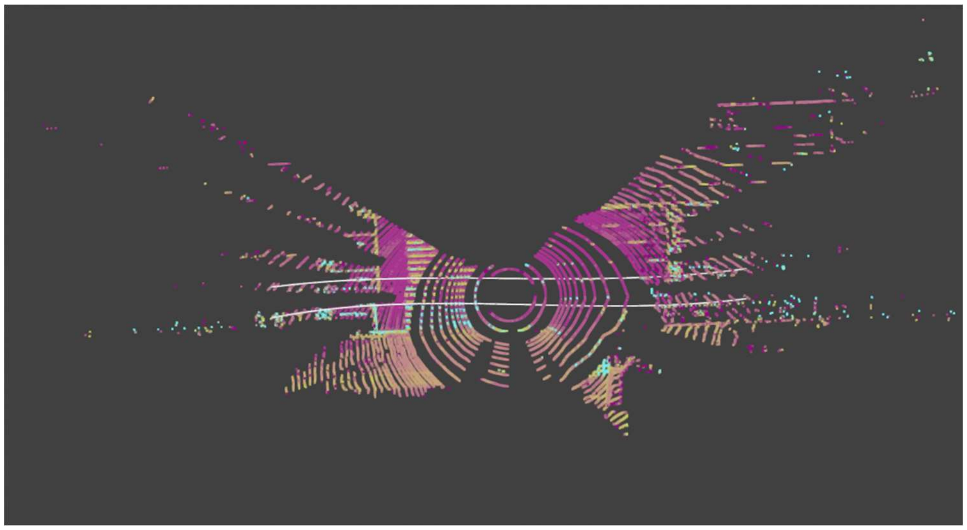
Scene 5: 1553670562447716965

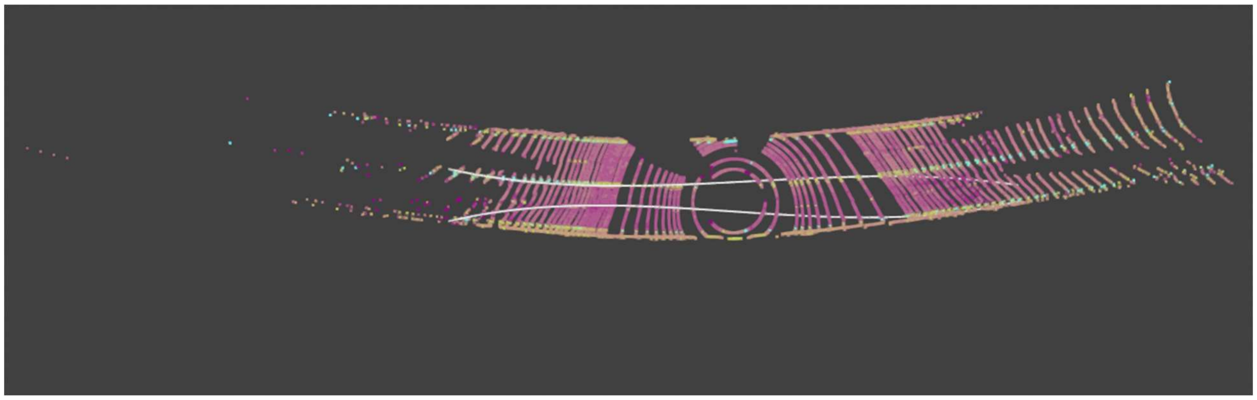


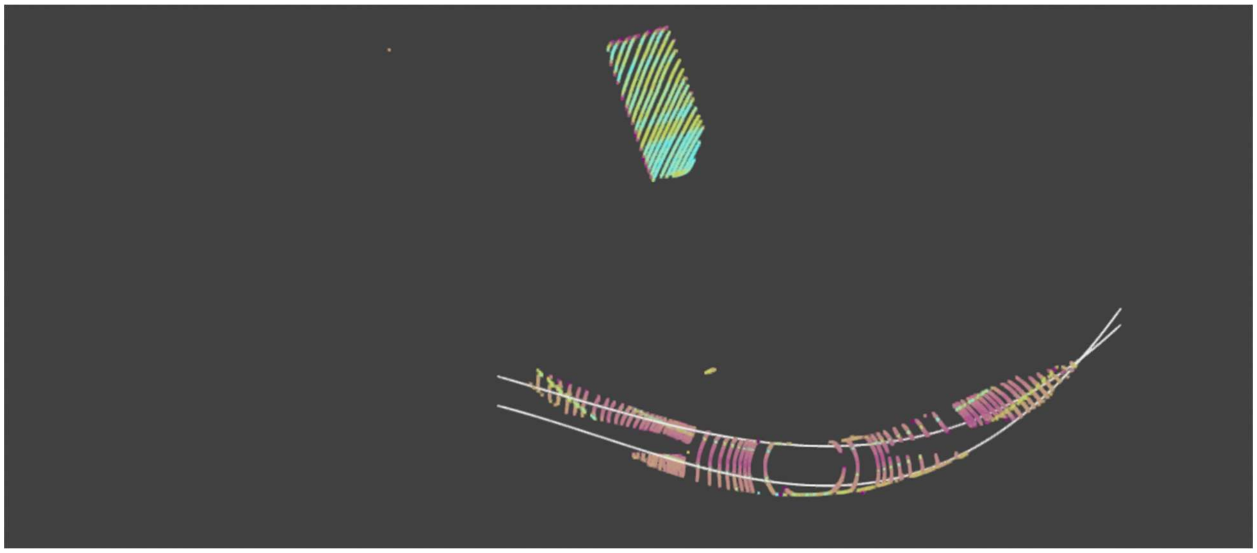
Scene 6: 1553670669147427892



Scene 7: 1553670684146333606


Scene 8: 1553670931248857912


Scene 9: 1553671068147021752


Scene 10: 1553672341938522335