

My work started by studying tools for rendering .ply files. The first difficulty was that the recommended libraries refused to run on my system. The problem was version conflicts: Python, CUDA, the C++ compiler, and the development environment. After many unsuccessful attempts to set up a stable environment on my computer, I came to the conclusion that it would be easier to use a Docker container. I have created a Dockerfile that runs on Windows (the operating system is critical here, as drivers for communicating with the graphics card and container access to files on the host depend on it). After setting up the environment, I was finally able to run the first code to download the data and tried to generate a frame. Initially, I tried the Sustpat format, but it consumed a huge amount of memory. Unfortunately, the decrease in data quality led to an unacceptably poor result. As a result, I decompressed the data into the standard .ply format, and their download was successful — the first frame was received. However, the camera in the scene was oddly tilted. After taking a long time to figure out the code, I discovered a key error: I used a CAMERA-TO-WORLD matrix, while I needed a WORLD-TO-CAMERA. It took a long time to fix this. After that, things went better, but the trajectory of the camera along the set points still looked unnatural. It turned out that some of the coordinates were inverted relative to the visualization on the website from which I took the data. As soon as I fixed it, I was able to record the first trial videos. Then I ran into two new path animation issues: The speed of the camera movement on different segments of the path was different. I solved this by taking as a basis the minimum speed in all sections. At the end of each segment, the camera made a sharp, jerky turn. Simply adding dots did not give a smooth result. Then I implemented an algorithm that adds additional intermediate points before each turn and smoothly interpolates the direction of the camera's gaze. The result has become much better, although for perfect smoothness, it is probably worth adding even more points on the turning sections. The final stage was the integration of object detection using YOLO. The model works based on video frames, selects objects, and generates a JSON file with the results. This part should probably be improved in the future to improve accuracy. Honestly, I hope it will be a well-deserved 15 points. I know you have a lot of work to do, but I'd appreciate it if you'd give me some feedback.