3D Product Rendering with LiDAR

Overview

The Home Depot has recently expanded their app to feature Augmented Reality (AR), which is currently utilized to allow customers to display 3D models of furniture and other Home Depot products in their own home. Creating these 3D models of The Home Depot's inventory has proven both time consuming and costly. Previously, model creation was outsourced to an external vendor. However, The Home Depot would like to investigate if there is a cheaper solution that scales with our ever-growing inventory.
Additionally, The Home Depot is looking to expand the AR capabilities of their app by allowing customers to scan 3D models of their (the customers') own furniture. After scanning their furniture, the customers can then place the resulting models in an AR environment alongside potential purchases and view new arrangements of furniture. This initiative is particularly targeted at new movers, a group which The Home Depot is interested in expanding their support towards.

Goal

Our goal is to determine if LiDAR (Light Detection and Ranging) sensors can be utilized to create 3D models of both The Home Depot's inventory and customers' furniture. LiDAR sensors are included on many latest generation iPhones and iPads from Apple, which allows this technology to be accessible to the public and The Home Depot at a previously unobtainable scale. Therefore, the central question we seek to answer is whether or not Apple's LiDAR sensors are capable of scanning high enough quality 3D models for commercial usage.

Research

After examining the capabilities of LiDAR, we determined the steps necessary to create 3D models from the sensors' collected data. Model creation can be broken down into 3 steps:

1. Record the LiDAR data in the form of a point cloud. LiDAR functions by projecting many small lasers away from the sensor and then measuring the distance each laser travels before it bounces off of an object. See Figure 1 below for a visual representation of the sensor's lasers.

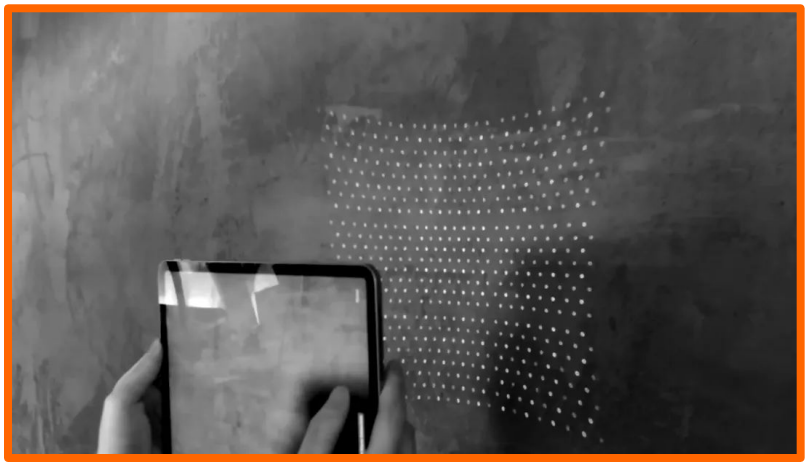

Figure 1*: infrared view of LiDAR laser projections from an iPad Pro*

2. Convert the resulting point cloud into a 3D mesh by connecting the points in the cloud. This can be accomplished with a variety of methods including algorithms and machine learning.

3. Texture the resulting mesh with color or images by filling in the individual polygons of the mesh. This step normally requires camera input from the iPad in addition to the LiDAR data.

With these three steps in mind, we created two different iOS apps to collect the data we needed. Both utilized ARKit, Apple's Augmented Reality development framework. The first app recorded the raw data generated by the LiDAR sensor and stored the resulting point cloud into a .PLY file (Polygon File Format). The second utilized ARKit's automatic mesh generation feature to skip the point cloud, directly generate the mesh from the LiDAR data in real-time and store the resulting mesh into an .OBJ file (Object File).

Experimentation

To benchmark the two apps against each other, we generated a point cloud (for App 1) and a mesh (for App 2) of the same picnic table. See Figure 2 for the results.

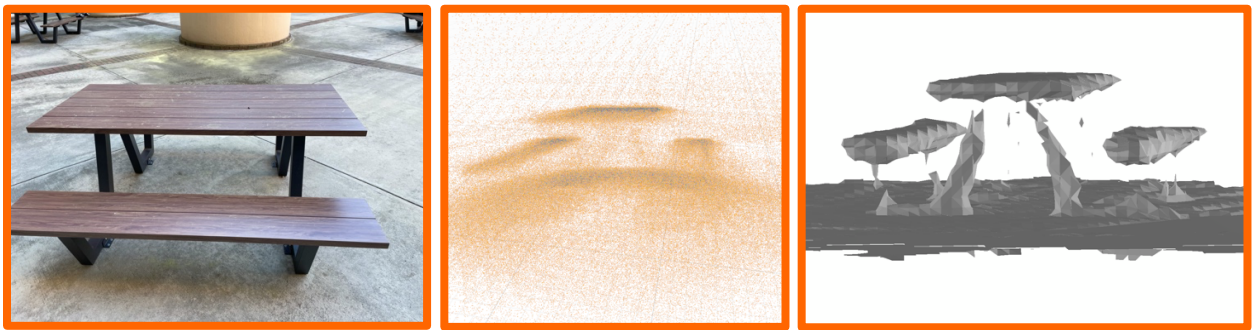

Figure 2: *the picnic table (left), App 1's point cloud (center), and App 2's mesh (right)*

App 1's generated point cloud contained 500,000 points and - upon examination - seemed superior to the mesh generated by App 2. In particular, App 2's mesh failed to accurately detect flat surfaces and also cut off the back foot of the bench. After debugging, we determined this was because ARKit's mesh generation only creates meshes on a frame-by-frame basis. As a result, we were only able to save the most recently generated mesh from the current camera frame alone. Because App 2 recorded much less data than App 1, we decided to proceed with App 1 and begin creating meshes from the recorded point cloud of the table.

Algorithmically generating meshes from the point cloud quickly proved to be difficult. Algorithms such as the Ball-Pivoting Algorithm, which normally generated very high quality meshes, failed to produce satisfactory results. The only algorithm which created a remotely accurate mesh was the Screened Poisson Surface Reconstruction algorithm, the results of which can be viewed in Figure 3 below.

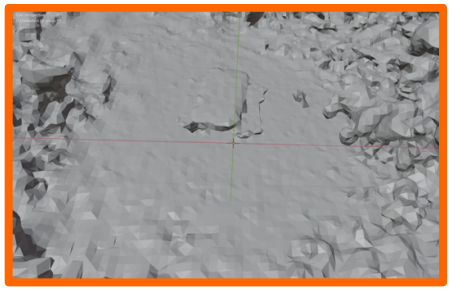

Figure 3: *Screened Poisson mesh of the point cloud*

While the bench in the middle of the mesh was low quality, it was a promising start. Due to the noisy background, we proceeded to manually crop away all of the points in the point cloud which were not in the immediate vicinity of the bench. This reduced the size of the point cloud from 500,000 points to roughly 250,000.

However, after continuing to run different algorithms on the point cloud, we still failed to generate any useable meshes despite cropping. We also denoised and reduced the size of the cropped point cloud via random sampling, also to no avail. After careful examination of the point cloud, we discovered the issue: the density of the cloud, background noise, and algorithms we were running were not the problem. Instead, the raw point cloud data itself was flawed. While the cloud featured plenty of outlier points which we already knew we needed to remove, we found that additional points were also ruining the results of the mesh-generating algorithms. See Figure 4 for a few different meshes we generated.

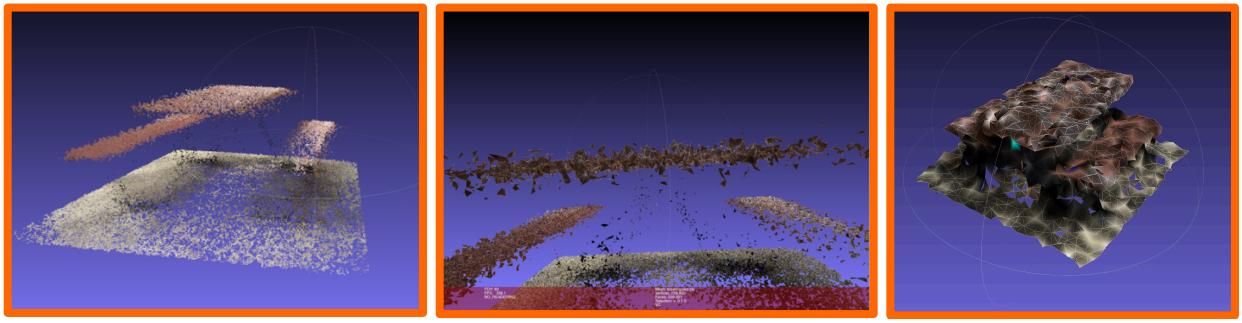

Figure 4: *mesh of cropped cloud (left), close up of said mesh (center), mesh of low-density cloud (right)*

Notice how while at first glance, the mesh of the cropped cloud appears to be acceptable, but after zooming in, it becomes apparent that the mesh is extremely faulty. Namely, the generated mesh dips inside of the surface of the table. A mesh is supposed to be a wrapper around an object - the object's "skin". If the mesh has polygon faces which are generated under the object's surface, then the points which resulted in those faces are faulty. All polygons should be located solely on the surface of the object.

The reason these erroneous polygons exist is simple. As one records LiDAR data, they need to circle around the object they are scanning to collect all the points needed to generate a complete mesh of the object. As they circle around the object, the IPad needs to keep track of its position in real-time. However, small errors will occur in its position tracking for a variety of reasons. Normally these minute errors would not affect anything, but when we are attempting to scan objects to the highest degree of accuracy possible, the errors add up and result in unusable data points. Identifying and removing these points is next to impossible using post-processing because they are indistinguishable from their neighbors. Based on our observations, we believe the only way to account for these errors is to process the collected points in real-time – a task beyond the scope of this project.

After realizing we had hit a dead end, we pivoted back to App 2, the mesh app. The two main errors we had encountered with this app were the inaccurate plane detection and its inability to remember data other than that which was found in the most recent camera frame. We implemented a plane detection feature to fix the first issue, and then developed a "snapshot" method to fix the second. Basically, the snapshot feature saves multiple meshes while scanning an object and then merges them together at the end – resulting in a far more complete mesh than

we were previously able to generate. See Figure 5 for a comparison between the old meshes and the new meshes which were generated with plane detection and snapshots.

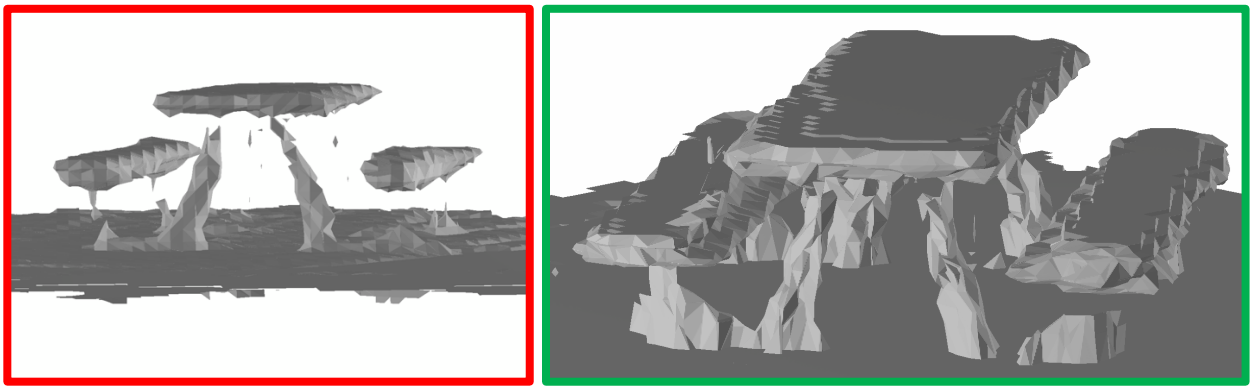

Figure 5: *the old (left) and new (right) meshes generated by App 2*

While still rough around the edges, the new meshes which App 2 can generate show promise. In particular, the plane detection leads to smoother models and the snapshot feature allows for more complete models. Notice the current mesh from Figure 5 contains the back supports of the picnic table while the old mesh does not. This is just one example of the increased quality of the mesh.

Conclusions

Our original goal was to determine whether or not Apple's LiDAR sensors are capable of scanning high enough quality 3D models for commercial usage. After research and experimentation, we were left with mixed results.

We concluded that without developing real-time processing algorithms to increase the accuracy of the point clouds, The Home Depot will not be able to utilize the raw LiDAR data from iPhones and iPads to generate meshes and models of its inventory which are high enough quality to be viewed by customers in AR. However, there are still multiple paths forward towards creating these models. First, [redacted]. Once the LiDAR data inaccuracy is addressed, generating models and meshes from the point clouds is a straightforward task.

As for adding features to The Home Depot App which would enable customers to scan models of their own furniture, we believe the methods utilized in App 2 should be sufficient to accomplish this goal. While the meshes generated by App 2 are not perfect, after smoothing, cropping, and texturing, they will be accurate representations of the customers' furniture. The key difference between the problem of inventory scanning and customer scanning is the customers may not require their models to be of the utmost highest quality. The main feature they require is that the dimensions of their furniture are measured as accurately as possible. Again, this initiative is targeted towards new movers who would take their 3D scans and place them alongside models of potential purchases from The Home Depot to make sure they fit well in their new homes.

Therefore, while inventory scanning may not be possible with LiDAR given our current resources, enabling customers to scan their own furniture in The Home Depot App should be achievable.