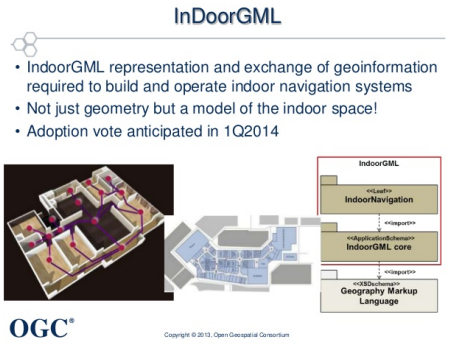
Point Cloud to IndoorGML

[(42) Introduction of OGC IndoorGML - YouTube]

THis video explains what happened in the whole process!: [(42) From Point Cloud to IndoorGML - YouTube]

OGC: Open Geospatial Consortium> An organization that strives to define standards for using, storing and creating geospatial data. Base software that different systems such as IndoorGml and CityGML are built upon.

IndoorGML: it is a data model to display and understand indoor spacial. GML is an XML style language to store geographical information. It is a method of viewing indoor structures.



CloudCompare: An open source software that automatically cleans up point cloud of things such as noise(persons, unknown objects).

From generating point clouds of surfaces, there is no way to fully automate generating point clouds because of issues with transparent and reflective surfaces. In the article, they explain that they were using a LiDAR camera for the data collection of a large space(Seoul Mall). Hopefully with a regular depth camera some of these issues are resolved.

The paper, *From Point Cloud to IndoorGML,* gives experiments as to how to create indoorGML Documents from a point cloud. A good portion of it focuses on the process of automating the creation of the point clouds. This is a stretch goal for the whole project, but currently out of scope.

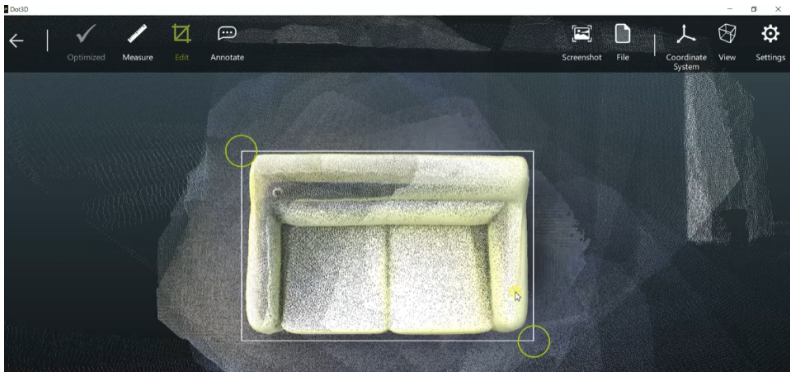
Takeaways:
-indoorGML seems like a good way to represent indoor data, such as the scans that we are looking for. However, the example provided shows that the models that they create are very simplified, and from my understanding the models we are trying to create are trying to replicate the use of the Matterport camera.

-This method could be applied to our project. We could modify the pipeline in either software we use to end at the generation of the point cloud, and then use unreal to extrude surfaces from that point cloud. If this is done, we could use cloud compare to fix any errors that were generated form the point cloud.

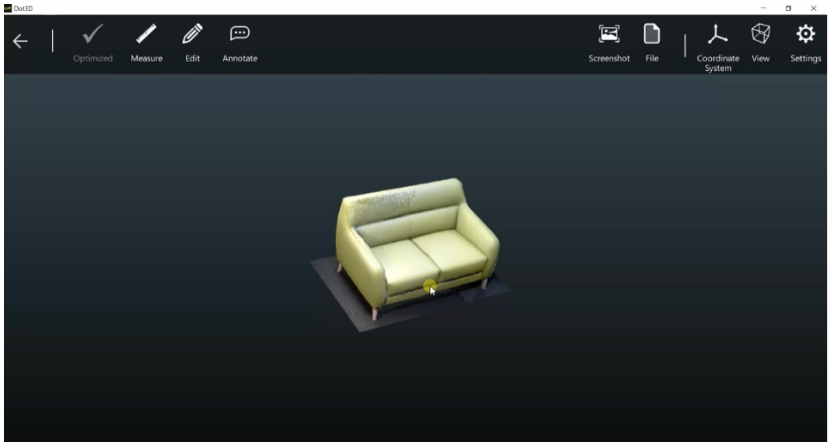
Extracting Vector Geometry from DotProduct scans :
DotProduct3d has a secondary application, called PointFuse, that can be used to extract vector geometry from DotProduct scans. This can enable us to export from DotProduct directly into either of the following file formats : DAE, DXF, FBX, IFC, SKP, STL, OBJ, X3D, or NWC.

The following screenshots show how to utilize this application :

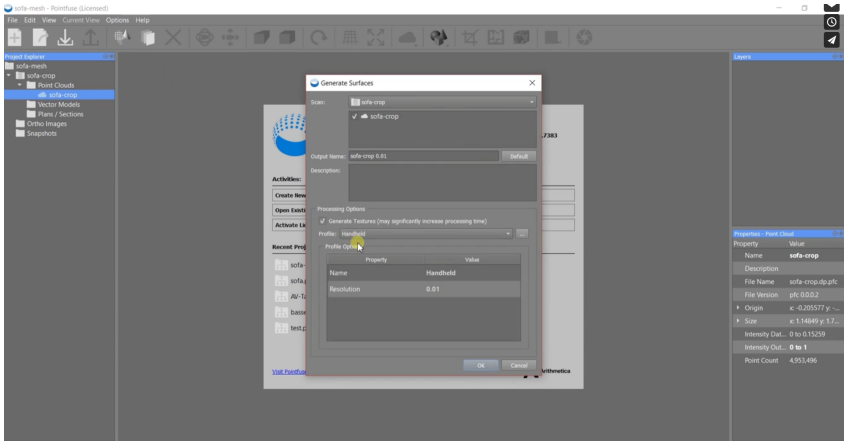
Scan of an object in DotProduct.



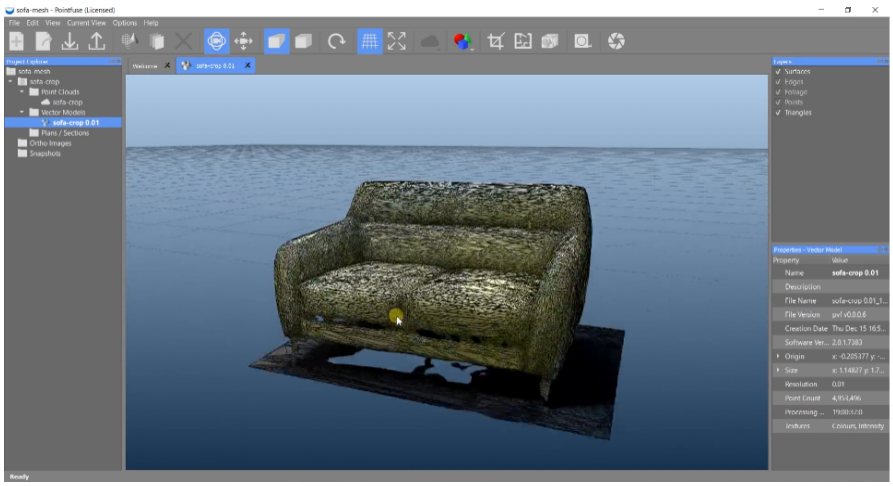
Selecting crop field.



Cropped sofa



Import through PointFuse



Vector model that can easily be cleaned up via PointFuse

PointFuse will then allow you to export it into the file formats previously described above. This would greatly simplify the way we transpose scans into FBX files. There was another method we had previously been tinkering with-- running our scans either through AutoDesk's FBX converter software. Seen : FBX Converter Archives | Autodesk Developer Network

How to utilize FBX converter utilizing 3ds Max : Importing 3D models into Unreal Engine 4

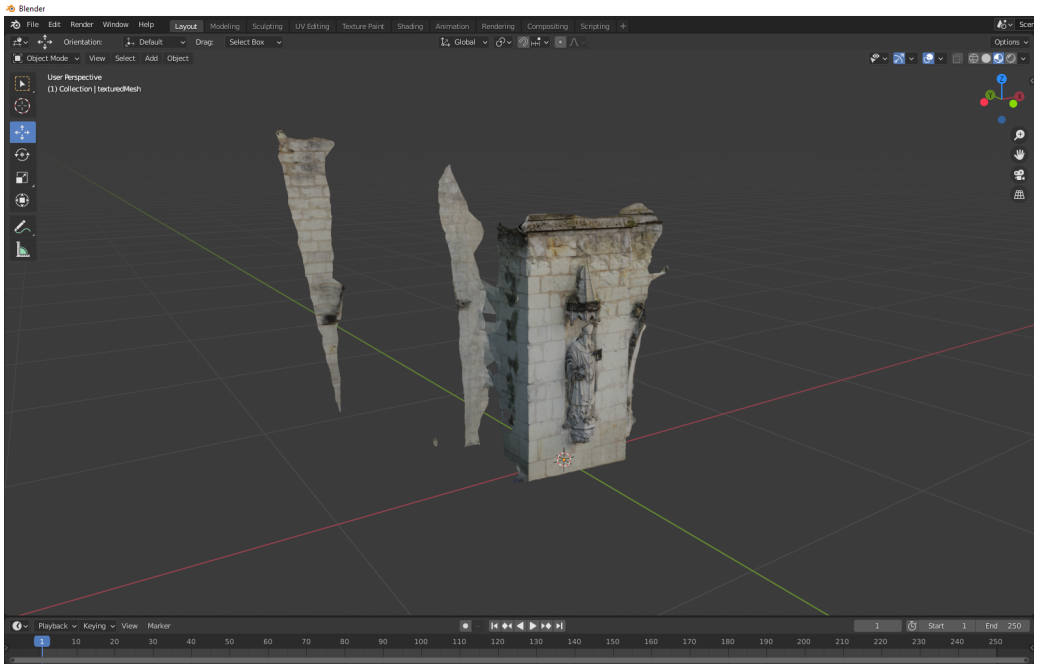
There are several videos documenting how to utilize the FBX converter to convert scans into OBJ files for the Unreal Engine. However, it does tend to be more finicky. We had previously theorized that we could utilize either of Autodesk's 3DS MAX or MAYA to convert our scans into OBJ files.

Takeaways :

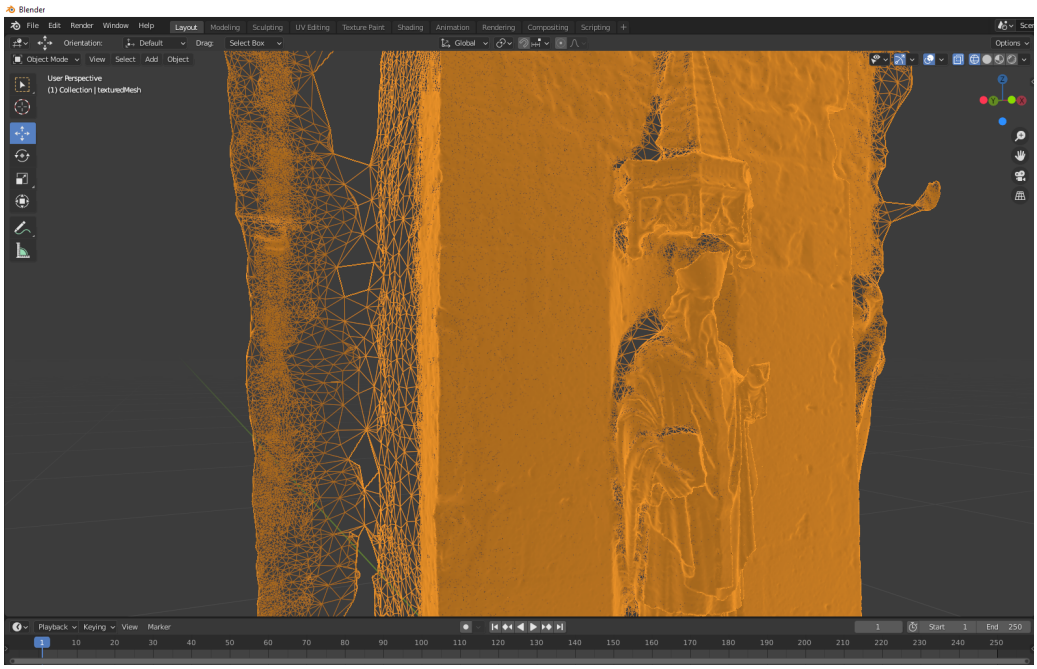
- PointFuse might serve as a great inbetween for DotProduct and the Unreal Engine. The Unreal Engine expects FBX files. DotProduct does not export scans directly into FBX files. However, PointFuse can.
- We can utilize PointFuse to clean up our scans in the future.
- Does have a free-trial period similar to DotProduct.
- We could utilize AutoDesk's software

Blender

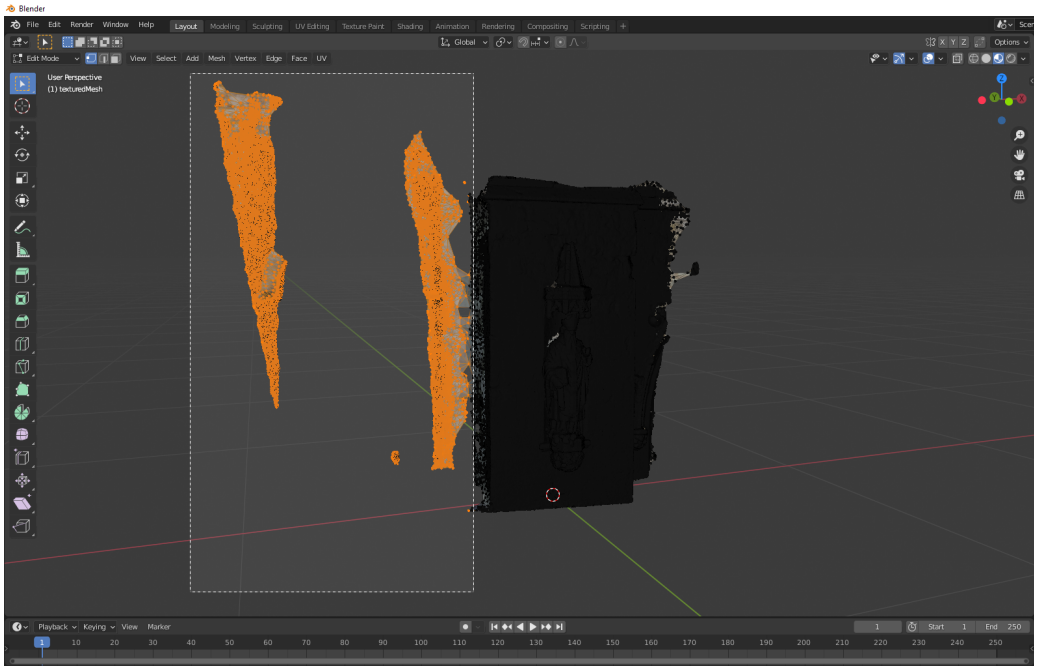
Since neither Meshroom nor DotProduct 3D output models as .FBX files, they need to be converted before being imported into Unreal Engine. Additionally, although Unreal Engine 4 does have some plugins that support mesh editing and cleanup, they are still quite unstable, very slow, and prone to crashing. Blender solves both of these problems as it is both excellent for mesh cleanup and can convert either .OBJ or .PLY models, among other formats, into .FBX (It should be noted that PointFuse is MUCH more effective for working with point clouds). To test Blender's mesh editing capabilities, I imported an .OBJ file created in Meshroom:



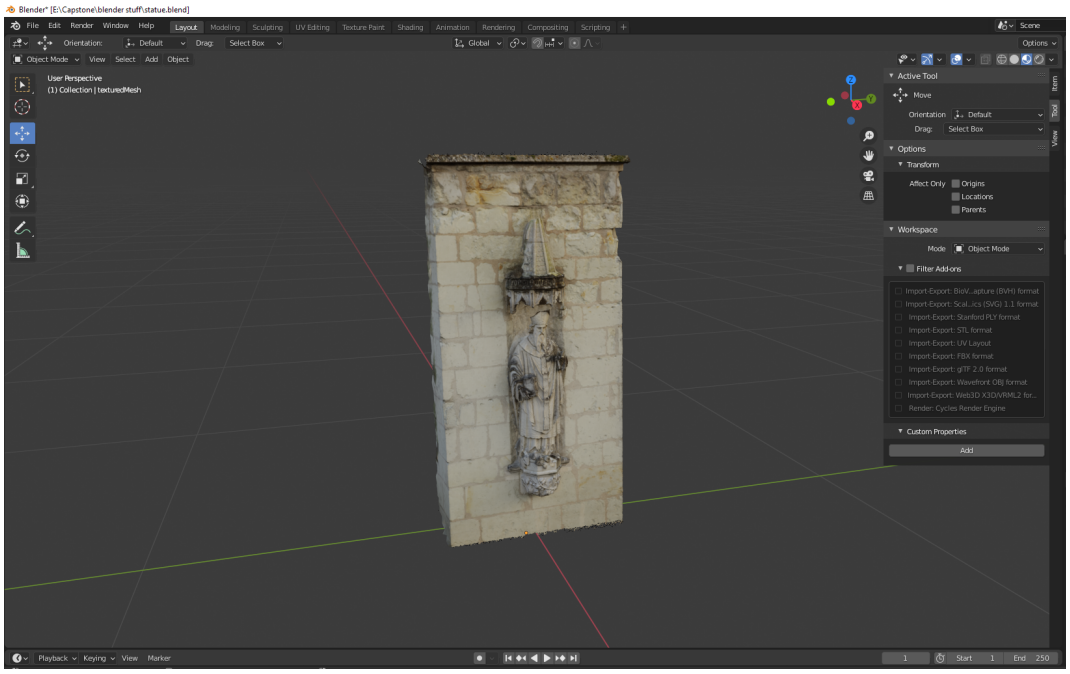
The above screenshot shows the model, solid and with texture applied, before any changes have been made. The mesh itself is shown below.



To fix its irregular shape and remove any extraneous fragments, I selected and deleted vertices as demonstrated in the below screenshot:



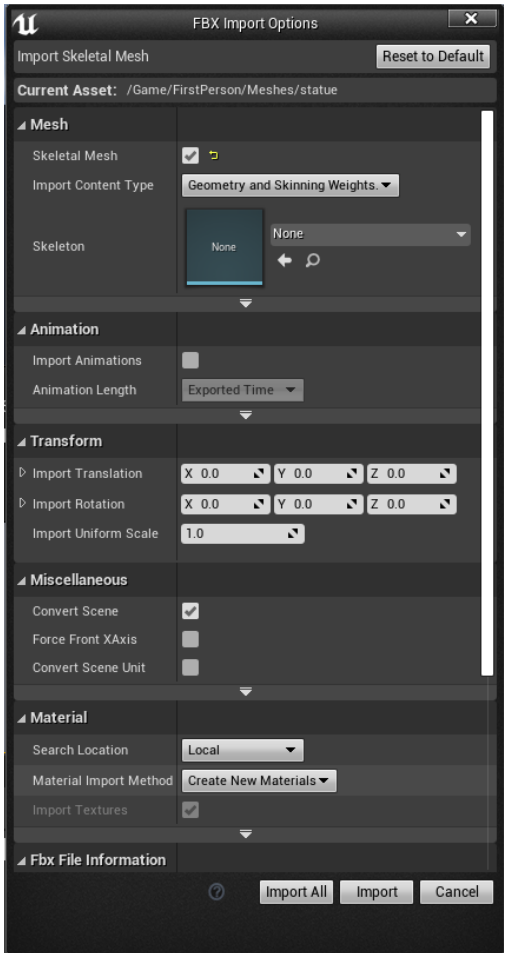
Afterwards, I decimated the mesh to reduce the number of faces and make it easier to process. Specifically, simplifying the mesh makes importing it into UE4 much faster and improves overall performance. I found that decimating the mesh to a ratio of 0.25 makes it significantly more manageable without losing detail or distorting the texture. The final product is shown below.



To convert it into a UE4-friendly format, I exported it as an .FBX file. Detailed export settings are available at [Chronic Development | UE4 to Blender and Back Again](#).

Unreal Engine

Although Unreal Engine 4 can technically import .OBJ files, its preferred format is .FBX. I imported the model into UE4 with the following settings:



These only deviate from the default settings in that “Skeletal Mesh” is selected. It’s important that the material import method remains set to “Create New Materials” because although it’s possible to import textures along with the model without enabling material creation, the textures will not be applied. Another option would be to create a custom material afterwards, but selecting “Create New Materials” ensures that the textures are applied according to the original model.

Clicking “Import All” adds the model as an asset, any number of which can be placed in the level at will. The model can then be moved, rotated, or scaled.

The below screenshots show the statue model when placed in a first-person UE4 level. The user can move around and look at the model from different angles. Despite being significantly decimated, the mesh preserved most of its original detail, even when viewed from up close.