CS11: 3D Scanning of Large Spaces

4/9/2021 Meeting

Brandon Withington, Seika Mahmud, Casey Boomer

Unreal Engine Plugin Development Progress :

- Considering switching to creating a Python script to import FBX files within the Unreal Engine editor.
- Benefits of switching to creating a Python script over an Unreal Engine plugin :
    - The user would not have to download and insert the C++ plugin themselves.
    - It would be simpler to use as the execution of the script will be invoked by a single line of code : fbximport.py [fbx-file directory] [fbx-file] [mesh-file path]
    - It would be more compatible with our other extensions of our application.
        - Being coded in Python versus being coded in the native Unreal Engine language for plugins (C++).

- Issues with creating an Unreal Engine plugin (*this issue can be applied to the Python importing script as well*) :
    - The Unreal Engine already has a very decent FBX importer baked into the Engine, it could be redundant to recreate it. It is easier for the user to drag and drop the FBX file they create with our application than having to manually install a plugin we make. In addition to being able to drag and drop FBX files, there is a more granular option of specifying the direct scene, materials, and static or skeletal meshes manually.

User interface with QML

- Initially, our interface used the basic version of QtQuick and QML. The Qml file had the general layout for the interface itself, while there were some c++ files in the backend that operated the functionality for the interface. This worked really well, and it was rather simple to be able to give functionality to the qml file with some basic object-oriented programming. However, after discovering more about the plugins that we were using, working with c++ would turn out to be rather difficult to work with, because the scripts that we will be working with operate in python.
- Our solution to this was to change the qt from standard QtQuick with Qml to Qml with python, which worked with pyside2 modules. Following this, we worked with the PyQt6 modules. After a full week of debugging, we are still encountering errors that have blocked us from continuing to develop the functionality of the user interface.
    - There have been errors regarding the use of some of the modules that are required for Qml. Specifically the QtQuick.Dialogs module has been continuing to throw errors that the modules are not found. This seems to be an issue for many

people online, and there is no fix for either installing the module manually or debugging the apparent issue.
    - Even past the issues with the modules, the PyQt modules on python have been throwing issues as well, specifically, to be able to access the buttons and add the functionality to them, the app needs to access the rootObject of the qml application engine. This should be as simple as defining a variable as engine.rootObject()[0], as this returns a list of objects in the app, but in our case the rootObject we are accessing throws an error that the index is out of range. This ultimately is an issue with the Qml file, as there should be a declaration for the application window. However,the Qml file appeared fine to be called. This could mean the issue with QtQuick.Dialogs not being defined is also causing the issue with the root object as well, because perhaps the file is not searching for the root object anymore because the file is throwing an issue.
- There are a few solutions to these issues:
    - Continue to work with PyQt5 and Qml. The problems occurring with the current build should be simple fixes, and perhaps some more time fixing the issue could make the rest of the interface be finished much quicker. Perhaps changing the version of QtCreator could fix the issue.
    - Convert the project back to c++. We already have the functionality set up for a c++-based interface, and we could connect the code to the python script through multiple c++-python integration libraries that we could add. This may make it more difficult for people to set up the application on their own computer however, because this would involve more personal setup.
    - Continue to use pyQt, but rebuild the interface with the script's own interface modules. This will take time to rebuild the interface, but it will make integration much easier because this method was designed to be integrated with python. This option seems the most desirable because we are not too committed to our current interface, and this may be a good time to redefine what is necessary to have on our interface. We also have some experience doing this option as well.

Metashape Script

- Successfully imported point cloud, generated mesh, and exported model in FBX format.
- Continuing to work on texture generation and baking.
- Considering switching to tiled models in order to more easily create high-quality textured meshes.
    - This will depend on Unreal Engine compatibility