Marcus Doucette

2025-09-23

CS 499

M4 Milestone

1. Briefly describe the artifact. What is it? When was it created?
   1. This was a 3D graphics project made back in the spring. It was made to try and show how 3D primitives and graphics techniques for texturing and lighting could be used together to build 3D environments.
2. Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?
   1. I think that this artifact more than any other is a good example of something that can be shared easily with just a picture or video, or run quickly with an executable file. I also think that graphics programming as a whole is one of the fields of computing with the heaviest relevance on algorithms in order to convert data into images quickly.
   2. To improve this artifact I decided to add a command line argument that could take in an obj file and render it in the middle of my scene. I also want to update the github to host exe artifacts because the current build instructions - while accurate – do have the problem of being a bit more technical than many users may want to deal with.
3. Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?
   1. I believe that in creating a new code flow path and edits to the GitHub repo I’m able to not only deliver on the promise for algorithmic improvements but also add room for new security and collaboration enhancements as well.
4. Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?
   1. Command line arguments are super easy to add when you’re writing a command line utility since you just start calling your program with the appropriate arguments. I do this the most already with python and rust. Because this is a visual studio project I needed to find out how to add the command line arguments to the project manually which was a bit strange and took longer than actual code for this step did but wasn’t all that difficult ultimately.
   2. I next added in an obj reading library that takes in my input file and parses it into a useful struct that I can then reasonably convert into the format the rest of this project uses to store material and mesh data about the given objects. While adding the dependency itself was fairly easy there were some issues with my existing debug and release build setups that were causing the debug build when using the new library to fail. Initially I tried just building in release mode but after a little bit I found the issue and reconfigured several parts of my projects build settings to point the debug build to the debug artifacts and copying the correct dlls as well. Hopefully this will help with making sure that adding the artifacts to GitHub later will be a much easier process.
   3. Implementing the appropriate algorithms to read an obj file in and then convert it into the style that my project already uses was a bit tricky as expected. One of the bigger problems was the mismatch between the low level implementation of my existing project using things like raw arrays in comparison to the higher level implementation of the tinyobjloader utilities with things like smart pointers and vectors. Ultimately I decided to separate this step into two parts with the 3d model being one part and then applying the custom materials being another part that I might look into as we approach submission for the final project.
   4. The error handling coverage was tricky because I didn’t really have a good idea of in what way I wanted each possible situation to fail. I think that planning out the different fail states I could think of was helpful but since there is a few catch all fail states it was hard to think of even needing more unique fail states for each case. I guess in the future automated recovery would probably be the best option from the user perspective but that’s why I think I spent a lot more time than usual in the final stages trying to make sure that all the files would build as easily as possible and be runnable without much hassle.
   5. I was excited when I first had the idea to make buildable files into a github release. Thinking back from where I am now, the actual github release was relatively easy but making sure that the executable should be access the shaders and textures relative to the executable file instead of relative to the cwd was pleasant enough to work on after figuring out the rendering part. The most time consuming part of this step ended up being updating my build automation steps to match the new release folder structure that I wanted. I’m a bit surprised at how useful this build automation work is given how little it came up in the classes themselves. It makes me think of that saying about war being more about logistics than tactics. I also have some curiosities about maintaining the project given that I now have a release and how to roll in bug fixes or separating new features into updated releases. While this project is not likely to be the one I work on those enhancements for I think it will be a good concept to try and work around in future projects as it gets me that much closer to feeling like a real professional.

Improvements

* Add in an input file command line argument and associated parsing
* OBJ reading algorithm/library
* OBJ rendering with OpenGL in scene
* Review error handling coverage
  + No file
  + Valid file
  + Invalid file
  + Incorrect arguments
* Add exe artifact(s) to GitHub repo

Further steps

* Materials?
* Updates and bugfixes?