HW 2 – Rendering Large Scenes

Due: Oct 7

Overview: For the first assignment, we tried to get our hands dirty to get a sense of the "big picture" of how the various elements of a game engine work together by writing them all ourselves. For this next assignment, the goal is to get a better, hands-on understanding of the low-level technical side. In order to keep the scope manageable, we'll focus on issues relating to rendering very large scenes. The goal at the end of this assignment is that you understand the main components of the game-engine starter code and feel comfortable extending it in this and future assignments.

Requirements: For this assignment we are providing a substantial starter project (~2.5K lines of C++ code, 250 lines of GLSL shader code, 100s of lines of Lua game scripts, etc.). This assignment, therefore, has two parts: one, understanding the basics of how to compile, build, run, and extend this started code; and two, improving the ability of this code to render large scenes.

1. Running the Starter Code

As a first step, you'll need to get the provided game engine starter code up and running. While the codebase is large for starter code, its tiny compared to the code for Unity (https://github.com/Unity-Technologies/UnityCsReference/tree/master/Modules) or the Unreal Engine (a 10+ GB download). In contrast, building our own small Game Engine from scratch involves a 30 MB download, some 350X smaller than commercial engine -- as such, you shouldn't be surprised that there is a lot it can't do! For part 1, our job is to figure out what this code can do, and to understand what its limitations are.

a. First, download and run the starter code. You should be able to run the *SimpleExample/* demo. Run this demo and look at the framerate.

b. Look at the files *SimpleExample/prefab.txt* and *SimpleExample/main.lua*, these are used to set-up a scene. Find an .obj/.mtl file online and add it to the scene. Watch out, the engine only currently supports .obj models that have their normals defined.

c. Add a dynamic camera. One way to achieve this is to repurpose the up/down/left/right keys to update the camera variables instead of moving the dinosaur.

d. Make a very large scene. This scene should be large/complex enough that the frame rate drops to at most 10 FPS.

2. Improving Large-Scene Rendering

Now that you have a big scene that renders slowly, your task in part 2 is to get it to run fast again! In doing so, *the goal is to modify the scene file and the resulting visual rendering as little as possible*.

Here are some strategies to consider trying, feel free to think of others:

-A level-of-detail (LOD) system, which switches out complex models for simpler ones when they are far away from the camera.
-Use view frustum culling to ensure that objects which are not seen by the camera are never sent as a draw call.
-Turn off shadows for objects whose shadows are not important (and make sure these objects are not drawn in *computeShadowDepthMap*); or, consider even using different models for shadows (with simpler interiors).
-Think carefully about the order in which you draw objects. By first drawing objects that are close to the camera, and then later drawing further away ones you can often get a speed up in rendering (why is this?).
-If you are using the built-in collision system, it currently checks every collider against every other collider. You can improve this by using a spatial-data structure such as a uniform grid or bounding volume hierarchy (BVH) to only check nearby collisions.

As you are developing your acceleration systems, try to robustly automate as much of it as possible. For example, you should try to automatically compute a bounding sphere or cube around each entity to use view view-frustum culling. Or, if you are feeling ambitious can you automatically compute a simplified mesh for the user? Or perhaps automatically determine at what distance from the camera to switch to a simpler LOD.

Finally, <u>for full credit on this assignment, if you implement any camera-based accelerations, you must implement a debug camera.</u> This is a camera that does not affect your acceleration techniques (e.g., LODs should not switch for the debug camera). This way, you can see the effects of your technique working (and debug it when it doesn't!).

Project Report.
You report should come in two parts:
-First, create one image for each of 1a, 1b, 1c, and 1d, with a small caption for each image.
-Second, describe in 1-2 pages, what approaches you took to accelerating your slow scene from 1d. Include images documenting changes in visual quality with and without your acceleration code. Additionally, try to answer the following questions: *Which of things that you tried gave the biggest improvements? Which improved the least? Any hypotheses why? What are the limitations or tradeoff of the approaches you implemented? Are they always a win or are there some cases in which the performance can be worse or the rendering quality can degrade?*

For this assignment, working with a partner is especially encouraged as there is a lot of starter code to understand.

Submission:
To turn in your assignment, create a webpage with the following:
1) A video of your
2) Your complete source code, and a list of any libraries used

3) The above described reports

You should also prepare to give a short, in class presentation on your design choices and to provide feedback to (and receive feedback from) others in the class.

Send me a link to your webpage, no attachments!