SWE 585 Special Topics – Game Programming

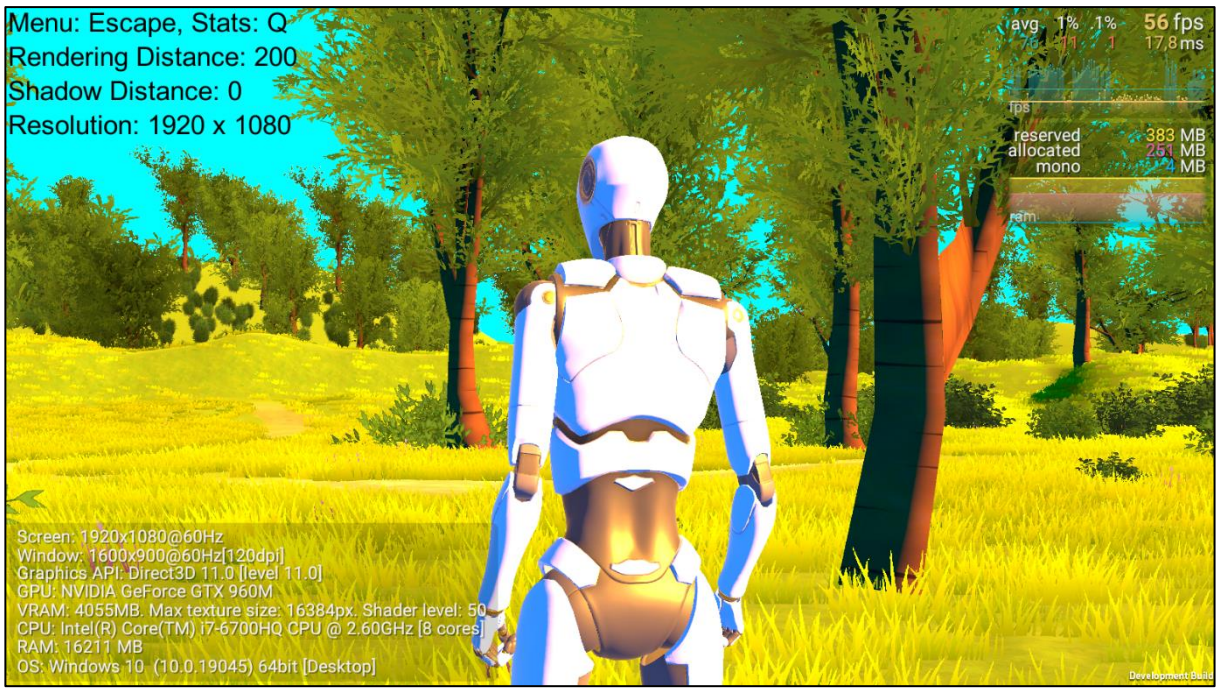
Term Project

Deniz Baran ASLAN – 2021719183

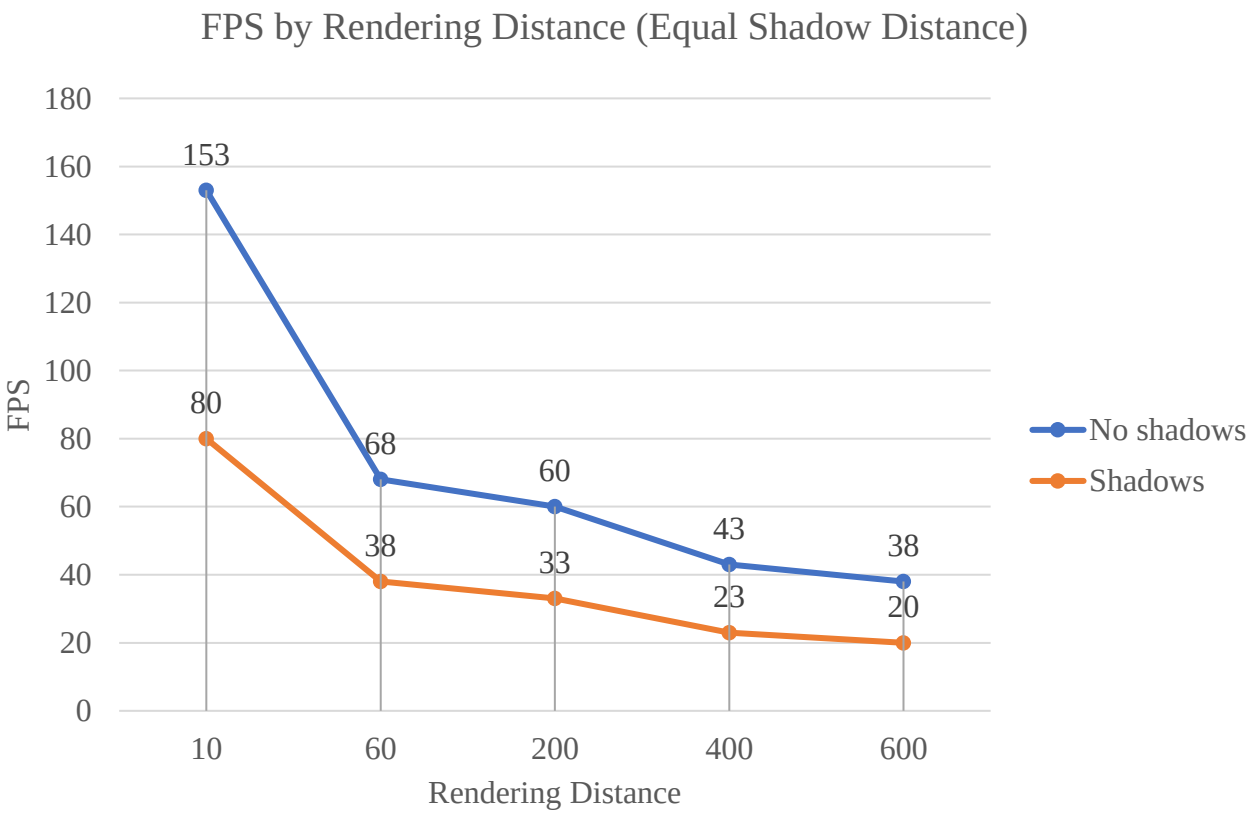
RENDERING DISTANCE AND PERFORMANCE

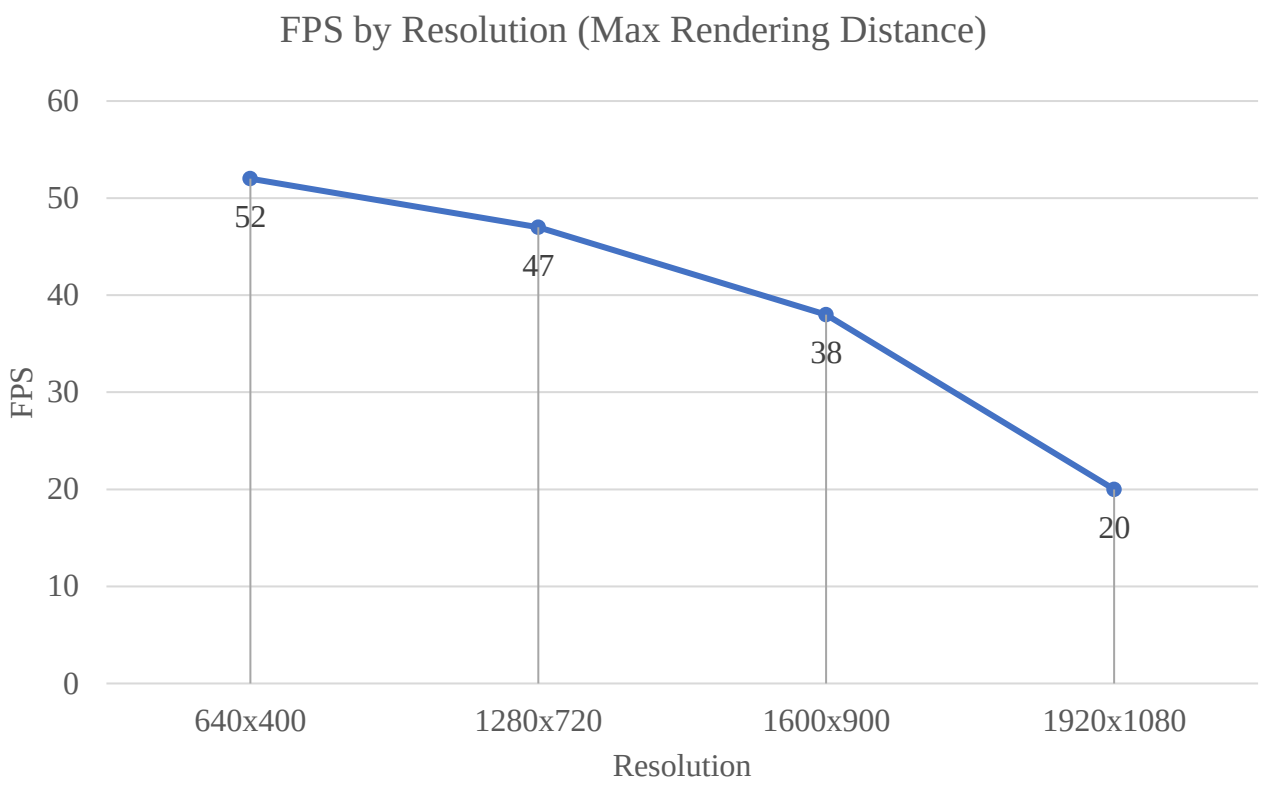
In video games with 3D graphics, rendering distance refers to how far objects and terrain can be from the player (or the camera) before the engine stops rendering them. This is an important parameter for video games, as the engine has to render many objects and each one comes with a performance and resource cost. Developers must find a balance between making the game run well, and keeping the game visually appealing and playable. Many games allow users to modify this parameter to their liking as well as to their computers' capabilities. A low rendering distance risks choppy graphics and difficulty in navigating the environment. A high rendering distance comes with the risk of low performance and high resource utilization.

In my tech demo, I have put together an animated 3D environment, a controllable player character, a graphing tool that displays system specs, FPS and RAM usage, as well as a pause menu. Each asset was imported from the Unity asset store, listed individually in the GitHub repo. I have written functions for the pause menu that allow the player to dynamically adjust rendering distance (farClipPlane in Unity), shadow rendering distance, resolution and fog density.



The initial results were in line with expectations, as the FPS value declined steadily with increases in rendering distance. In order to experiment with other parameters, I also added the option to change the resolution and shadow rendering distance. These experiments also aligned with expectations. Both lower resolutions and lower shadow rendering led to higher FPS. Setting the shadow rendering distance to 0 essentially removed all environmental shadows, leading to the greatest increase in performance. The graphs below illustrate the results:





There was one issue with lowering the rendering distance in such a manner. The environment lost its visual appeal: the horizon looked empty, and objects appeared and disappeared instantly as the player moved or looked around. This is referred to as the “pop-in” effect. In order to mitigate this issue, I implemented fog, and a slider to adjust its density. By using the right amount of fog, I was able to hide the empty horizon and eliminate pop-in. This technique worked well at mid to high rendering distances, but not as much at low values. Hiding a very short rendering distance with dense fog leads to low visibility, reminiscent of horror games.

Ignoring the minor graphical glitches, one shortcoming of the demo is its lack of logging. I tried to get logs with detailed output so I could present them visually in my report, but I was unable to implement the feature. Another shortcoming is the lack of GPU and CPU monitoring. I wrote methods that directly get these values from the engine, but was unable to fix the bugs and get them to work.

As a follow-up or an improvement, an interesting feature to implement might be selective rendering distance. This is yet another way developers try to balance their games, where terrain and large objects are rendered further away, but smaller objects are only rendered when the player or camera is close.

The Unity project is available online on GitHub, along with a build for Windows. A short video demonstration is available on YouTube.