ESAL Summer Internship

Pooria Ghavamian

Introduction

The aim of this project is to create a photorealistic social robot in a Microsoft Hololens environment to replicate interaction with the physical reference robot. The focus of this project, however, is only limited to the aesthetics of the model used and will not delve into the behavioral aspect of the robot. The robot that is ultimately used will be a humanoid named Pepper.

Figure 1. Pepper Robot

Pepper is around 122 cm tall and is capable of interacting with humans by detecting their emotion and communicating verbally. As it can be seen the robot's outer shell does not utilize a wide color

palette and consists mostly of black and white. There is already a model of this robot made for Hololens in Unity, and my focus is to try to enhance the model so it looks more realistic and convincing as a real part of the world the Hololens user.

At the time of writing this report, however, due to difficulties I was unable to retrieve previous scenes and Pepper models created in Unity. I spent a large time familiarizing myself with photorealism techniques and creating photorealistic scenes in Unity. After that, I managed to work with Hololens and link it to Unity and experiment with different scenes and model. Therefore, the overall outline of this report will consist of two segments about Unity techniques and Hololens.

Photorealism in Unity

Having used Unity mostly for simple 2D games, and Unreal engine for 3D works, I was surprised to see how powerful it can be in rendering photorealistic 3D environments. I learned a lot of new concepts, many of which unfortunately were not applicable to Hololens, for reasons that I will discuss later. My study started with lighting and materials.

Materials

Materials basically let unity know how to define a surface. They define what shader is being used, and also the shader's parameters like references to texture maps, tiling, color and other properties. Unity assigns the standard shader to all newly created materials by default, however, it is possible to write your own shader. I found the standard shader to be more than enough for realistic scenes. Tweaking the materials properties is a non-strenuous way of taking control of the aesthetics of a model, in this case, "realism" being one of the aesthetic goals. This can be done by changing smoothness, shininess, normal maps, and detail textures to mention a few. There is a lot more that can be done in this section, however, for the sake of conciseness, I will omit them in this report.

Lighting

In Unity, lighting applications can range from simple changes to direction, type, intensity and color to more advanced topics like lighting techniques, reflection and light probes, etc. However, lighting can be divided into "real-time" or "precomputed". Real-time lighting is by default assigned to the basic light sources (directional, point, spot, etc) and are useful when we have dynamic objects in the scene. However, for the scene and light behavior to look more realistic we need indirect lighting. This is where global illumination (GI) comes into play. Unity calculates complex light bounces and other behaviors of static objects and stores them in reference texture maps called lightmaps. This calculation process in the Unity realm is called "baking". These lightmaps are then put on the scene similar to the way a material texture is. The shader of an object can then pick up this information from the lightmap.

Enlighten is unity's current solution for baked and real-time pre-computed lighting. It allows dynamic changes in light sources, environmental lighting, and material properties (e.g. -emissivity, reflectivity). For objects to be counted in the precomputation of GI they need to be marked as static. However, dynamic and moving entities will be lit using light probes that act as a web of sample points spread across the environment. If light probes are not used, the dynamic object will only receive direct lighting. The output of Enlighten is real-time Lightmaps, real-time light probes, and real-time Cubemaps.

Enlighten precomputation is more efficient than most GI solutions. It does this by breaking up a scene into systems. This allows the precomputation to become a parallel pipeline, with a lot of shadow map and direct lighting calculation carried out on the GPU. For dynamic lights, run-time Enlighten is run asynchronously on CPU threads (although, GPU is still used). This lighting system runs into a problem on certain mobile devices as it can put too much strain on the GPU and CPU (my experience with Hololens attests to this).

Progressive lightmapper, on the other hand, is Unity's new solution which uses fast path-tracing. Enlighten enables us to see changes done in realtime to lighting immediately. However, when it comes to other changes, such as geometry, properties, etc for large scenes a lot of time is required, during which no feedback is given. Progressive lightmapper uses Monte Carlo path-tracing and gives us feedback in real-time in the editor. It is a great tool for developers who want to make changes to a scene without having to wait for the scene to bake. Other than that, I did not find it to be faster or more efficient than Enlighten. Even though it uses only CPU cores for baking, I faced similar problems as before.

Light and Reflection Probes

Although the technical details of these probes can be complex, their application is straightforward. Just like how lightmaps store baked information about a surface, light probes store info about light passing through a volume. They allow us to give more accurate and realistic light behavior to dynamic objects without the need for expensive real-time computations.

Reflection probes follow the same logic, this time for reflection. They are best suited for places where a change in reflection will be noticeable like in tunnels, and high contrast surroundings. If a reflective object passes close to a reflection probe, its reflection map can use the probe.

High Dynamic Range (HDR) Rendering

Normally we represent red, green, blue in fractions between 0 and 1 (LDR), however, this limiting contrast ratio can hide a lot of details that go into creating a photorealistic image. With HDR we are able to go beyond those numbers, and therefore, allow very dark and very bright lights. In Unity when HDR is activated the scene is rendered to a buffer which holds values outside 0 and 1. This buffer is later sent to post-processing stack, where a technique called tone-mapping is used to bring the values back to LDR. When used with Bloom, HDR can create realistic lights.

Post-processing

Post-processing is one of the most powerful tools for improving a scene in Unity. The basic idea is to apply full-screen filters to the camera's buffer, which helps in creating something close to human vision. Unity's Post-processing stack can be downloaded from the Asset Store and includes many effects such as Anti-Aliasing, Ambient Occlusion, Color Grading, and Bloom. Before applying for Post-processing, it is important to have the scene use HDR and be in linear color space. Unity's linear color space allows the light in the scene to brighten linearly with regards to the intensity of the light source. This is an alternative to gamma color space, where the relationship can be exponential.

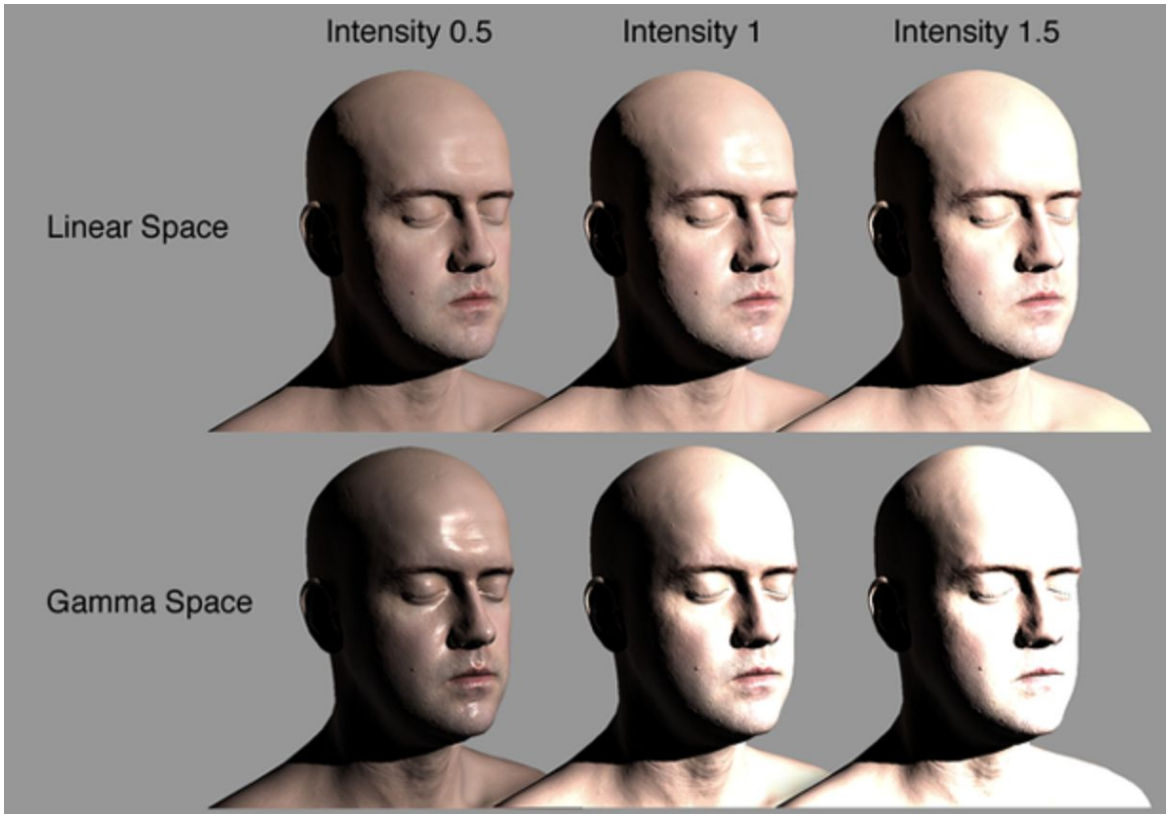

Figure 2. Unity's color space comparison

Sadly, many mobile devices do not support linear color space. In creating my own scene, I found Ambient Occlusion, and Bloom to be quite impactful.

Following is a profile of effects used in figure 7:

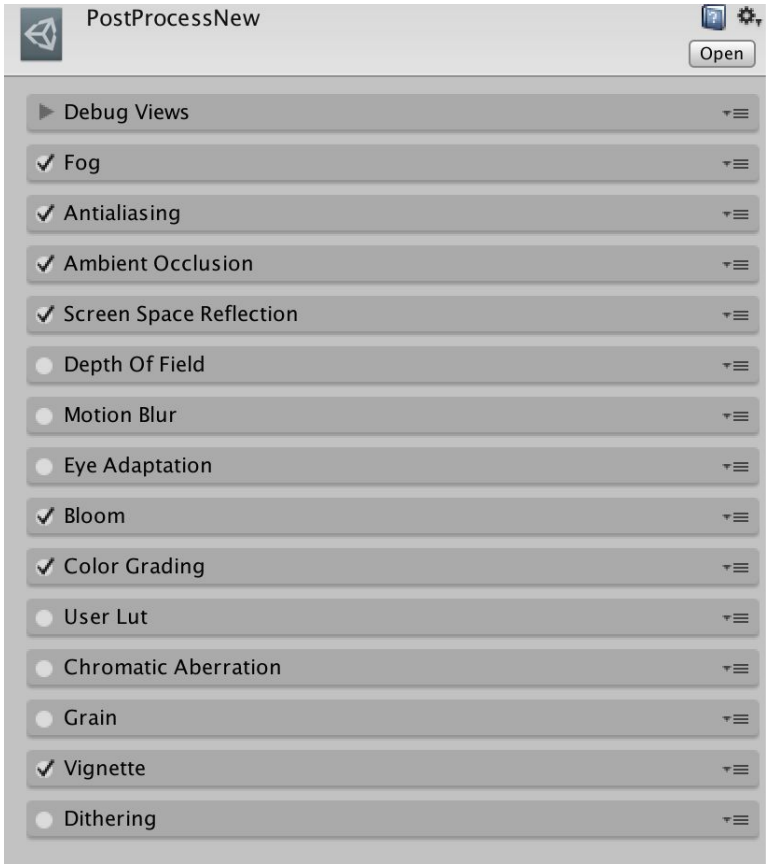

Figure 3. Post-processing profile used for a scene

Ambient Occlusion and Bloom

Ambient Occlusion is an expensive effect, and its objective is to darken regions that are close to each other such as holes, creases, and intersections. This gives a more realistic look to those parts of the scene. As mentioned earlier, Bloom uses the HDR buffer to create the illusion of very bright lights overwhelming the eye with a faint halo around them. One can also add "lens dirt", which gives the impression of a real camera that is covered in dust.

Scene with Bloom:

Figure 4. Sample scene with Bloom

Scene with Bloom and lens dirt:

Figure 5. Sample scene with Bloom and lens dirt

Finally, we can look at how a scene changes just by using post-processing stack.

Without Post-processing:

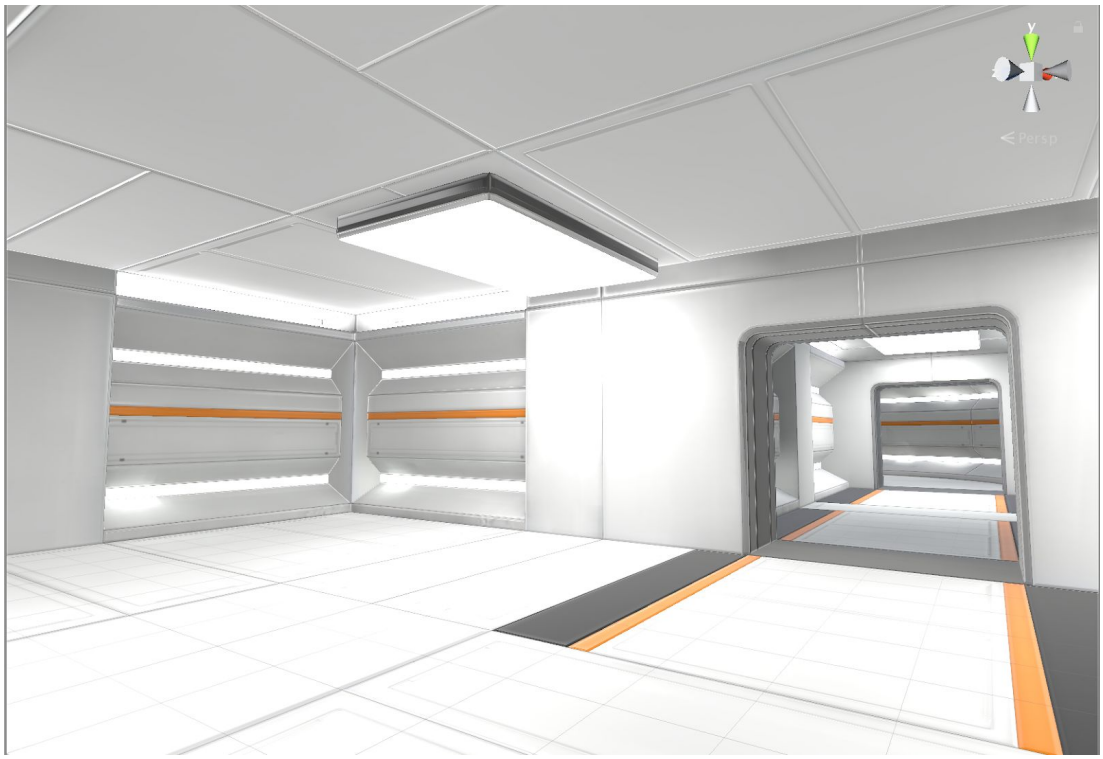

Figure 6. Sample scene without post-processing

With Post-processing:

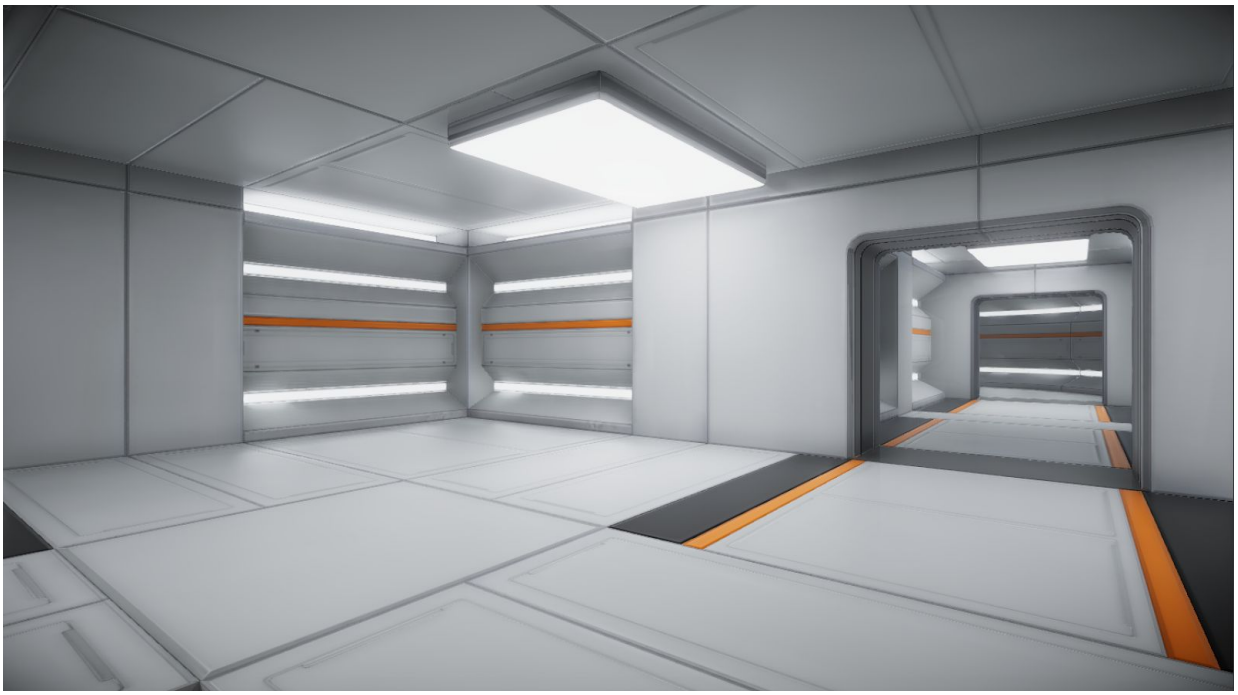

Figure 7. Sample scene with post-processing

Hololens

Hololens is an impressive piece of technology and provides a unique way of interacting with the real world thanks to its ability of 3D mapping in real time. On the front, it has the depth camera array, which is similar to what Kinect uses. These depth sensors help Hololens solve the occlusion problem, which plagues other AR technologies. A small motherboard equipped with Microsoft's own HPU (holographic processing unit) handles a trillion calculations per second from all the sensory input. Having real-time mapping, gesture recognition, voice recognition and other calculations on one hand, and three batteries and small chips, on the other hand, will surely lead to some trade-offs. It is for these exact trade-offs that creating an immersive photorealistic environment using conventional methods is either not possible or leads to quite laggy and clipping holograms. I will go over some problems that I ran into in the short time that I worked with Hololens.

The first issue relates to immersion. Hololens' field of view is about a 15-inch display, and this causes augmented content that exceed that limit (e.g. Pepper bot at its real-life size) to clip in the middle of the larger human field of view. The clipping line reinforces the idea that content is an overlay on the real world. This foveal vision is a good example of cost vs performance trade-off.

The second issue concerns itself with photorealism and the way Hololens works. Hololens uses additive displays and adds light to the real world to create content, therefore, black color appears transparent and white color appears bright. This changes the way certain colors look. This is a problem when it comes to realism, since dark shadows play a vital role. Due to the nature of how Hololens works, it cannot make anything in the real world appear darker than it is. Other workarounds using spatial map mesh and negative shadows looked unconvincing. In place of dark colors, a dark grey color can be used, which again does not look photorealistic as one would expect from a scene rendered on a desktop computer. One main way of improving the look of holographic content is by avoiding large regions of solid colors, and carefully picking colors through trial and error.

Finally, we have the problem of hardware limitation. As mentioned before, Hololens had to cut many corners to be able to be a mobile device and carry out the calculations it does. This means that it is easy to overwhelm the system. One easy way of swamping Hololens is by including the dynamic lighting and reflection probes used in realistic environments. Moreover, all full-screen effects like Bloom, FXAA, and SSAO (found in the Post-processing stack) that significantly help in creating a photorealistic image should not be used as they lead to botchy renders. Shaders that are too advanced should not be used (handwriting a shader, or using ones provided in Microsoft's Mixed Reality Toolkit could be a way to optimize). HDR should be turned off, as it requires an additional texture copy per frame. Character animations should be cut to a minimum. Non-functional animations that loop (e.g. idle stance) can have a surprisingly substantial negative impact on the performance. All in all, It appears that the best course of action should be to stay away from expensive tools (or use them sparingly), and focus more on low level changes in material properties like texture, and color.

Resources

Unity guide useful for optimizations:

https://unity3d.com/learn/tutorials/temas/performance-optimization/optimizing-graphics-rendering-unity-games

Unity Shader:

https://docs.unity3d.com/Manual/Shaders.html

Unity Post-processing:

https://docs.unity3d.com/Manual/PostProcessing-Stack.html

Unity Lighting:

https://docs.unity3d.com/Manual/LightingInUnity.html

Occlusion in Augmented reality:

https://hackernoon.com/why-is-occlusion-in-augmented-reality-so-hard-7bc8041607f9

Single pass stereo rendering for Hololens:

https://docs.unity3d.com/Manual/SinglePassStereoRenderingHoloLens.html

Suggestions for improving a scene:

https://www.reddit.com/r/Unity3D/comments/5zwcju/im_struggling_to_move_closer_to_photo realism_what/