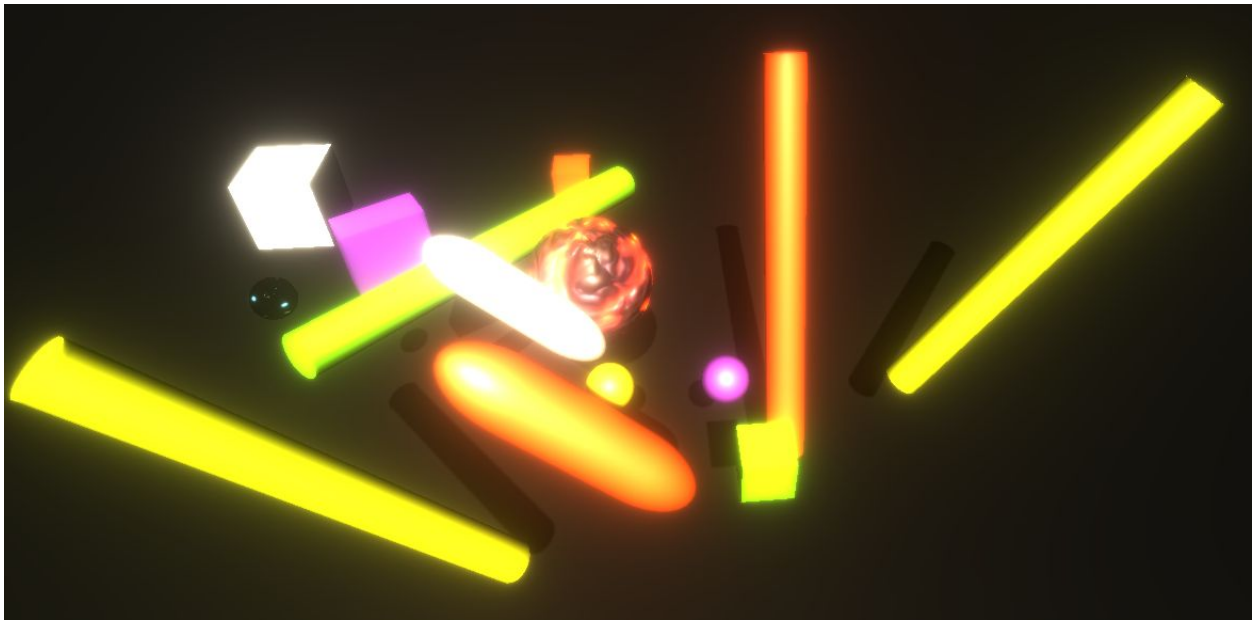
Brett Fotheringham
Tyrell Turley

Part 1.1. - Baseline Bloom (0.5 Points) Your first task is to implement bloom on a 2D screen by directly following the tutorial here:



The bloom effect is created by a very clever algorithm that allows us to use image blurring and addition to give objects a glowing or radiative effect. The first step involves blurring the image. The naive approach is to simply average the surrounding 4, 9, or 16 pixels to achieve your desired value. Doing large blurring kernels however discards lots of information and leaves the image with a rather blocky appearance, unsuitable for the more ergonomic fit that the radiative effect needs to have. This information loss and blockiness is solved in two parts. First, we implement progress downsampling. In progress downsampling, a smaller blurring kernel, such as 2x2, or 3x3, is used iteratively, so that averages of averages are used. The resultant effect is that much more of the original pixel information is preserved. To fix the blockiness and give a true “fuzzy vision” appearance to the image, the downsampled image is then upsampled. Upsampling increases the image resolution, reducing the pixelation of the downsampled image, making the blur appearance more ergonomic.

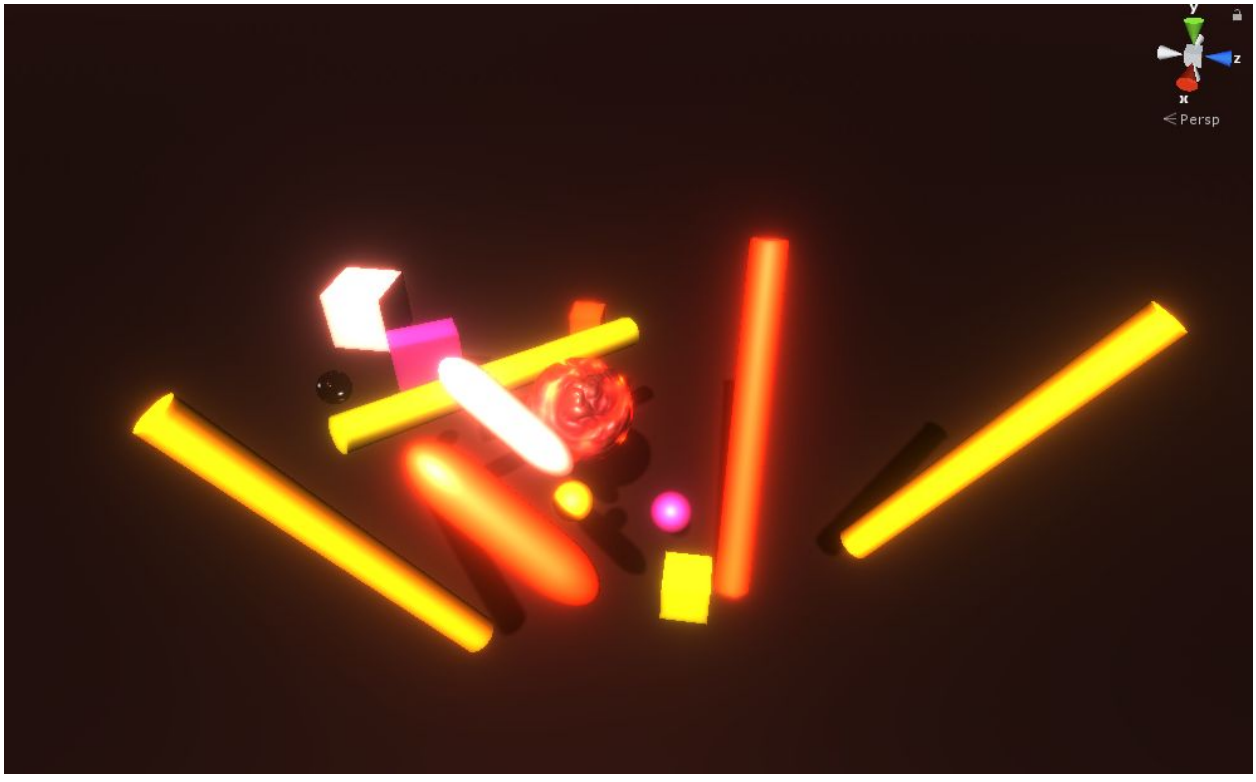
A further improvement of the blurring is to use a two-step algorithm rather than a single blurring kernel. The box filter, which accomplishes this, averages four smaller blocks within a space, then averages those four blocks to get the resultant value. This further eliminates the hard pixelation resultant from large blurring kernels. Now it would seem that combining this blurred image with the original would result in the desired effect, but it does not! The blurring used thus far has been a linear blurring function. In order to mimic the effects of radiative light, we need something that decreases quadratically, meaning the intensity of the effect changes quadratically the further away from the source of light is rather than linearly.

The solution to this quadratic/linear quandary is to approximate a quadratic function using the linearly filtered images. Here's how it works, an image is downsampled and added to the result, then the intermediary is downsampled again, and added to the result while being scaled by one half, the intermediary is downsampled again, and added to the result while being scaled by one fourth, and so forth until the desired effect is achieved. The number of iterations really isn't significant past a few because of the diminishing returns. The resulting image is rather washed out so it must be blended with the original image to achieve the final bloom effect.

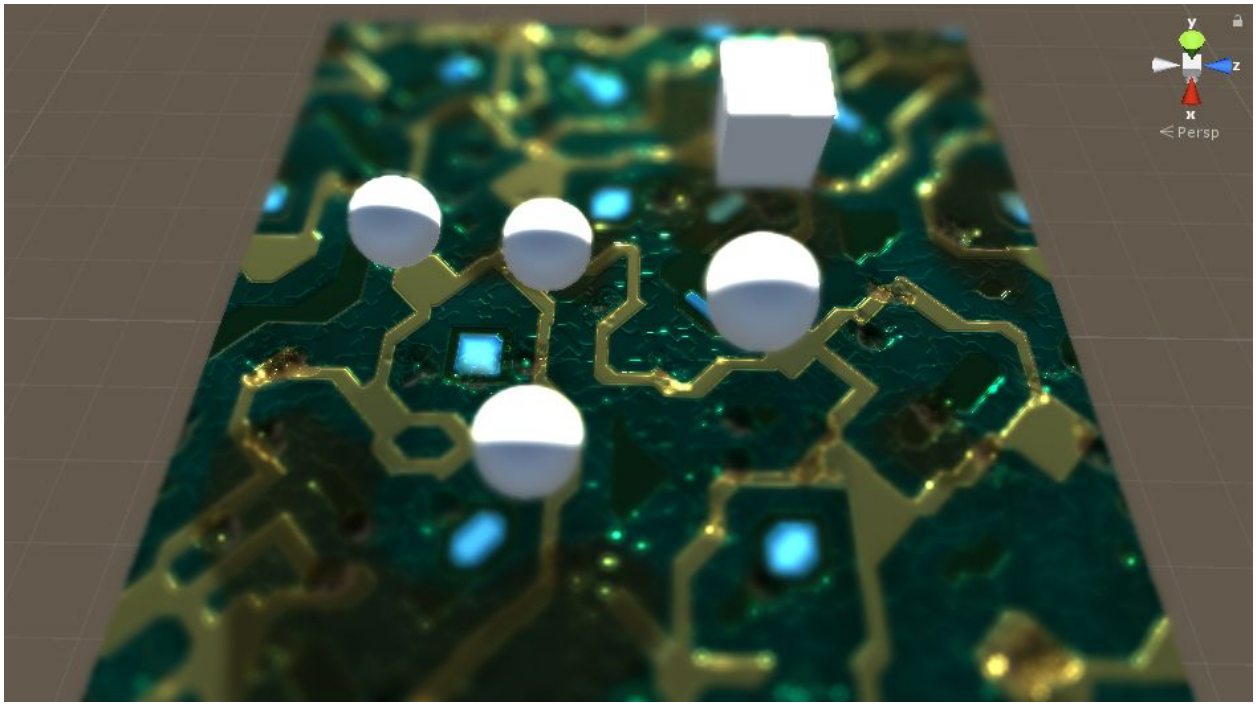
One of the main features of rendering that makes an effect like bloom possible is the ability to perform different effects in different rendering passes and then add the results together into one image. Without this, we wouldn't be able to make out details on the surfaces of the glowing objects but instead would only see a blurred image. At the end of the bloom process, we add back in some of the original texture information so that detail can be seen.

Part 1.2. - Modifying Bloom (0.5 Points)

On line 100 of the shader, in the fourth and final pass, we multiplied the frame by "half4(2.0, 1.0, 1.0, 0.0)", effectively doubling the value of red pixels. In games where lighting is an important element in conjunction with bloom, we can achieve a "sunset" like-effect by isolating or increasing values of certain RGB elements in one of the shader passes.



Part 2.1. - Depth-of-Field Rendering (2 Points)



First, the depth of field rendering determines the circle of confusion, or the focal depth and the surrounding depths at which objects would appear increasingly blurry, and stores that range of depths to be used later. We then implemented a bokeh effect, which effectively introduces a circular blur on the entire image. Lastly, we applied the bokeh blur selectively and gradually, out in both directions from the focal depth over the range that was stored previously.

In more detail, the bokeh effect is the quality of the blur in regions that are out of focus in an image. The appearance of the bokeh is normally affected by the aperture shape of the camera. If the camera aperture is square, the bokeh will look square. The same will happen with a circular aperture. Because we don't have a physical camera in our rendered image, we have to simulate the bokeh effect. This is done by averaging the color of all the pixels in a determined region around a pixel, and then rendering that entire area as that color. This is performed iteratively on each pixel, and the result is many overlapping regions of averaged color. We first started with a square region around each pixel and then switched to a circular region. The result of this bokeh is very pixelated, so we had to blur the resulting image in an additional pass. This gives a uniformly blurred image however, making it seem as if each pixel is at the same depth, which is not the case.

As was mentioned before, in real life the bokeh effect is created by the aperture shape of the camera, usually circular. The lens in the camera bends the light such that certain points at certain depths are in focus. To recreate the depth field effect, it becomes necessary then to use that distance information, to obtain a realistic result. Prior to this, the focus distance is input as a parameter. Using this parameter, and this depth information, we then look at the pixels from the

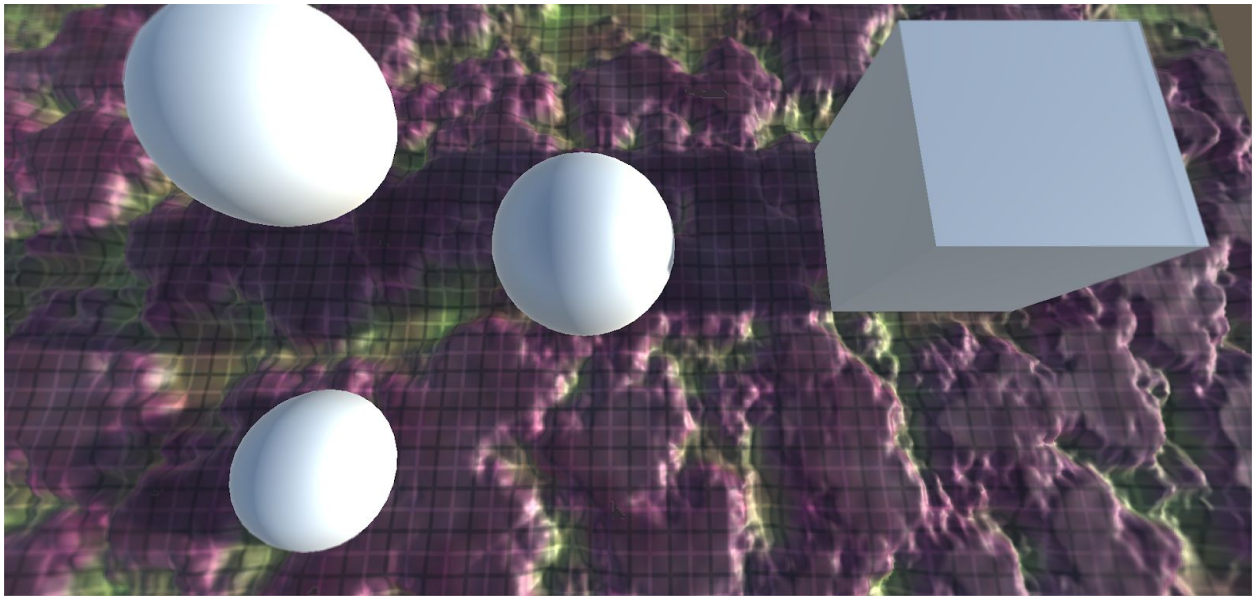
perspective of being “in-focus” and “out-of-focus” in that if the depth information is close to the focus distance, then the pixels will not have the bokeh effect applied, the pixels whose distance are more or less than the focus distance, within some input range, have the bokeh effect applied to them making them look out-of-focus. The interesting parameters here are the focus range and amount of bokeh applied. In our actual vision we can also only focus at a specific distance, but what is the focus range of the average person? Making this parameter too small blurs too much of the image and makes the picture look unnatural, whereas having too large a focus range leaves too much of the scene clearly in focus, which is also unrealistic. Additionally, the human vision system perceives more blur the further away from the further away from the focus distance we are, an effect we can replicate here by applying more bokeh the further the pixel depth distance is from the focus distance.

Part 2.2. - Low Fixed Foveated Rendering

To achieve Low FFR, we pass in the width and height of the rendering screen into the shader. We then define the areas where the image will be blurred at the different resolution levels. Lastly, we apply our blur code selectively to each of the corresponding areas.

The process of simulating FFR is quite simple. The big difficulty we had was figuring out how to access certain pixels from the shader, but once we discovered the proper method of doing that, it just took some experimentation to define the areas of different rendering resolutions. What we are doing is clearly quite different from how FFR is actually implemented on a headset. First, FFR is supposed to decrease the computational power required to render the image on the headset. In contrast, our method of simulating the FFR in a post-processing effect actually increases the computational cost. Second, a headset doesn’t do any averaging of nearby pixels to lower the resolution, because this would also increase computational cost.

To simulate half resolution in areas of the screen, we applied the standard downsampling blur effect from the beginning of the Bloom tutorial. To achieve ¼ resolution, we just applied the downsampling blur twice and so forth for further decreases in resolutions. The largest difficulty far and away was getting the correct pixels that correspond to the regions that needed to be degraded in resolution for FFR. Using a large number of and/or statements inside a shader, and by passing in the width and height of the image to the shader, we were able to isolate the pixels that belonged to the FFR reduced resolution areas. It was then that we applied the corresponding downsampling to each pixel. This process was identical in form for both the Medium and Low Fixed Foveated Rendering.See Images below for Low FFR and Medium FFR.



Part 2.3. - Medium Fixed Foveated Rendering