CS 5963 Assignment 3 Report

Tim Trimble

Austin Anderson

11/2019

Part 1.1

Bloom works by taking the image being rendered to the screen, de-rezzing it, and then re calculating it back up to a higher resolution to achieve a “blur” effect. Except, with a bloom effect, instead of only factoring in the position of the objects in order to “bleed” them into other pixels to achieve a blur, the bloom filter takes into account the brightness of the object, and bleeds bright objects into their surroundings with greater precedence than dull objects.

Part 1.2

The changes I made to the bloom filter were to allow the blurring (or the pixel “bleed”) to be adjusted manually to bleed more in a horizontal or vertical direction. The method I used to do this can be seen in BloomEffect.cs code. I thought this would be useful in case there was a particular movement or light source causing the bloom, the bloom would most likely be used to reflect a light that wasn’t pointing straight at the objects. This would provide bloom that I would find more represented of brightness in a real life situation. For example, the “bloom” effect would be more intense on an object horizontally if a flashlight were pointing at it from the side. Expansion on this variable and it’s usefulness could be done by having the horizontal and

vertical values of the bloom bleed adjust dynamically by checking the position of the lights in relation to the object.

Part 2.1

The Depth of field code functions by passing a depth from camera parameter to the shader which the depth of field script holds. A blur is added by rendering the image to a smaller resolution and back up again and the blur is smoothed using a round blur average rather than a square one (the Bokeh radius). Finally, to keep the focal point and a certain radius around it in focus, the depth of the current pixel is Linearly Interpolated to determine if the image at that point should be the lower resolution, higher resolution, or an averaged resolution.

Playing with different values within the Depth of Field I was able to find good parameters that felt more realistic than others. A Bokeh Radius of less than 3.17 encroaches on the in-focus plane and thus it is required to be a minimum of 3.17 to have any realistic appearing effect. On the other side of the spectrum, if the Bokeh Radius is more than 5, then the images in the background get so out of focus that it is jarring.

To further accomplish a realistic appearance, it is better to make the radius that we can view larger and bring the focus plan forward so that the nearest in focus edge is right on the near edge of vision. This goes to show that our natural focus as humans (or at least my own) is near to me as close things that are blurring seem off. Another fact about my own vision I've discovered is that the other piece to my own vision is the edge blur. When viewing the scene without foveated rendering active, the edges of the left and right edges being in focus while top and bottom edges were not was very unsettling to view. The in-focus images drew my attention to those parts of the screen and made me aware of the artificial effect.

Part 2.2

The FFR effect is achieved very similarly to part 2.1 by bringing the image to a lower resolution and then back up. For determining which section should be rendered at which resolution, two min and max values for both the x and y axis are given to the shader. The shader then uses these boundaries to lerp the two resolutions based on the pixels current x or y coordinate.

Part 2.2 Fixed Foveated Rendering++

The different levels of FFR are achieved by bringing the image down to even lower resolutions, and making more passes for blending using the min and max x,y values. In effect, it is a slightly more built up version of FFR-Low. To simplify debugging, each level of FFR was split to its own shader so that each pass could be more easily managed.

Adding the blur to the edges fixed unnatural attention grabbing at the edges of the screen. The Low FFR was so subtle that I wasn't aware of it, however, the High FFR was so jarring in the opposite direction that it was worse the Low. Thus, using the parameters I did for percentage of the screen that the FFR blurred, Medium FFR was the most successful as it was easy to phase out after a few moments of viewing, and as an actual hardware product would decrease the computation power needed to render on the screen.

With that in mind, the solution to making the higher FFR successful (and thus reducing the most computation power) would be bounding the edges of the screen tighter. In our implementation, we made each blurred block width be 10% of the width or height depending on the orientation of the blurred region. Initially the value was 15%, but in the Low FFR this took up too much space on the screen. Knowing that the other two FFR's would take up even more

space this was reduced which worked very well initially. A potential solution to space issue is to dynamically choose the percentage of the screen each block takes up. The only concern with this is that if the space is too small, then blending the different resolutions together will have less space to Lerp and thus will be more likely to have a hard-cut line of resolution division.

An alternative method that may fix the issue is to increase the blending distance. Currently, the Lerp occurs across 50% of the blurred zone. If the value is increased, then the edges closest to the screen will blend even more towards the focused region thus slightly increasing the radius of vision making High Res less immediately noticeable