Introduction

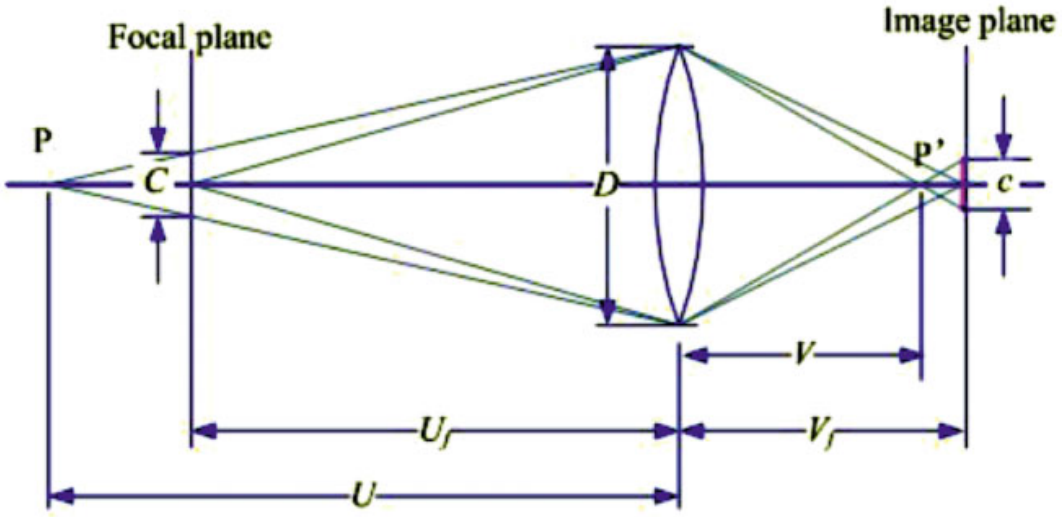
Depth of field is a natural phenomenon when it comes to both sight and photography. The basic ray tracing camera model is insufficient at representing this essential visual element and will present every element of the scene with perfect focus. For this project I explored different ways of accomplishing the depth of field effect by extending the ray tracer that was used in our classes previous homework assignment, starting with a distributed ray tracing technique and moving on to a lens model (which is incomplete) and then some small exploration of my own. I had intended to use the lens model to correctly render some extreme cases of the bokeh effect but was unable to due to an unfinished implementation.

Related Work

For the initial distributed ray tracing technique I looked to Cook's "Distributed Ray Tracing" paper which outlined the use of a limited aperture and focal plane along with random sampling to achieve a simple depth of field effect. The next two papers I looked at for using a camera lens model came from the same group which were "Realistic rendering of bokeh effect based on optical aberrations" and "Rendering realistic spectral bokeh due to lens stops and aberrations" who were both by Wu et al.

Depth of Field

With vision and camera's, there's a sort of "focal plane" that when an object is aligned with it, the object appears to be in focus. For all objects in front or behind this plane, the detail is diminished and blurred. As an extreme example a point light at a distance very far away from the focal plane will instead look like a semi-transparent circle instead of a bright point. This "circle" is defined as the circle of confusion or the bokeh effect. In the diagram below from [2] the variable C is defined as that circle of confusion.



When the aperature diameter of the lens, D, is increased the amount of blurring that occurs for the same distance is increased and only the objects right on the focal plane appear to be at all in focus.

After this thin lens model introducing an accurate lens model using Snell's law of refraction and aperture stops extending the same distributed ray tracing method with the thin lens should bring results far closer to depth of field effects seen in photography and could even be manipulated to create some stunning artistic effects such as changing the shape of the circle of confusion into a "square of confusion" or "triangle of confusion".

Distributed Ray Tracing

In [1], Cook describes a method of oversampling raytracing to produce several different effects ranging from anti-aliasing to motion blur. One of those effects however was also depth of field. He found that with (assuming perfect) lenses that different points on the same lens would always look to the same point on the focal plane which meant that the objects visible and the shading of those objects could look different when seen from different points on the lens. If then for the same point in the image plane were sampled over random points on a lens and the results averaged, the depth of field effect would occur in the resulting render.

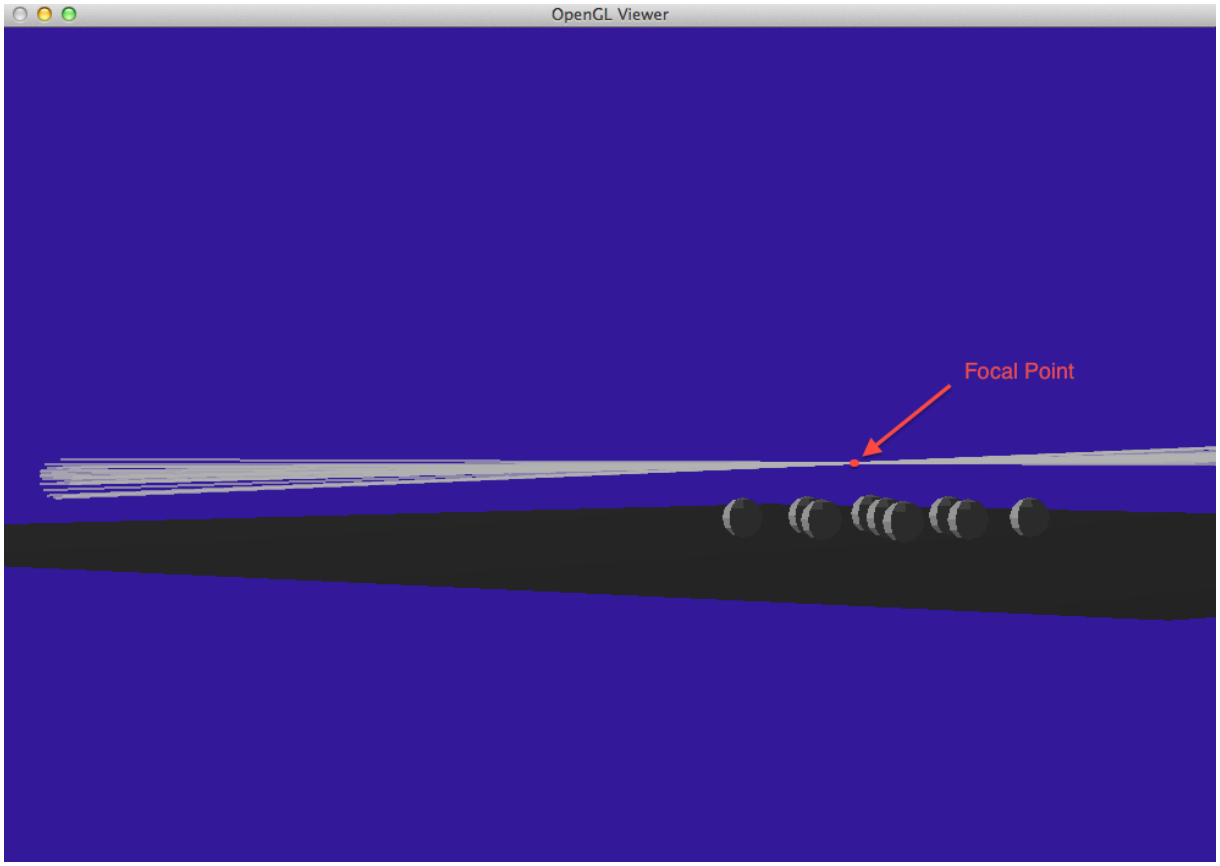
One of the tasks in our third homework in this course was to implement the distributed ray tracing implementation of anti aliasing. The pseudo code for that is presented here:

```
Color TraceRay ( x, y )
    Vec3f totalColor = 0
    for a given number of samples
        get a random point within the pixel at (x, y) on the image plane
        raycast from the camera through the point returning the color
        add that color to the total color
    return the total color divided by the number of samples
```

Instead of adding another step of the ray tracing algorithm I decided to instead change how the camera calculated the ray through the point on the image point.

The algorithm first calculates the screen point as the original camera model and use that and the given distance from the lens to the focal plane to determine the focal point for this ray. After doing that the screen point found before would be randomly moved within the radius of the given aperture divided by 2. That new point becomes the new origin of the returned ray but instead of just going with the direction from the camera's position to the screen point, the direction from the new screen point to the focal point we previously calculated.

Here's a visualization of the rays used in the calculation of a single pixel:



Because the anti-aliasing code already randomly samples and averages the colors of those samples, that change in the camera generateRay function is all that's needed to create a very good looking depth of field effect.

Lens Model

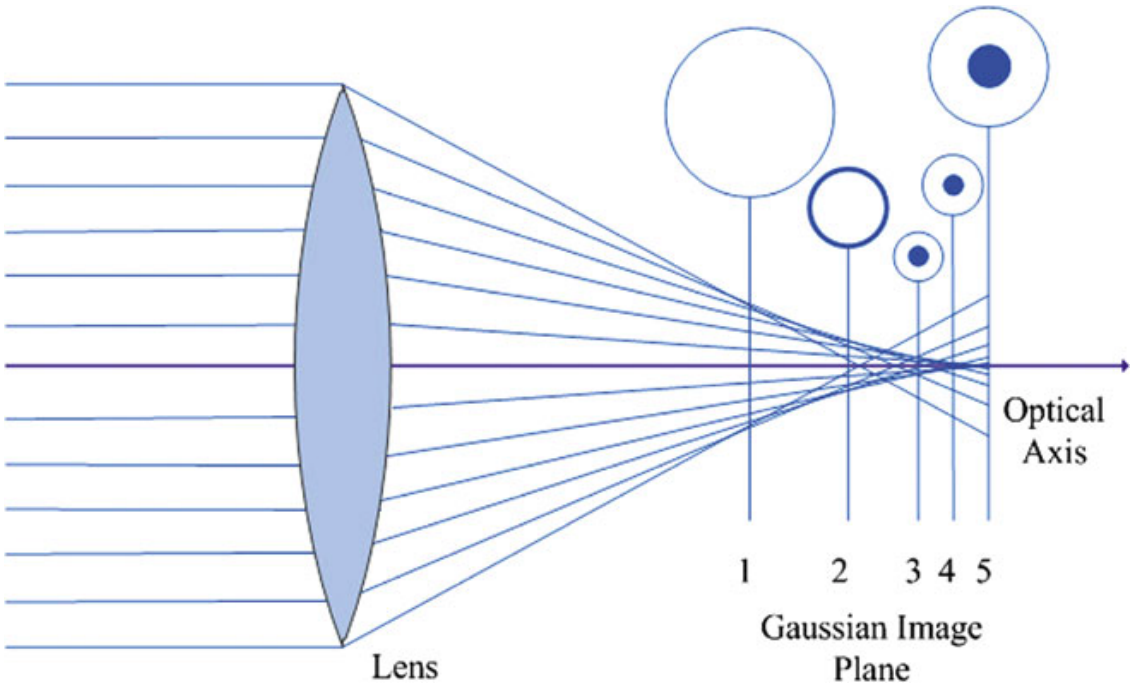
[2] instead explored the use of the physics behind refraction and series of lenses in order to create far more accurate looking depth of field. Using the law of refraction (or snell's law) the algorithm consists of instead of just directing the sample ray towards a single focal point, send the ray through a series of lenses.

Each lens has a radius that determines the curvature of the lens (positive if convex, negative if concave and zero if planar), the thickness of the lens which is the distance between the the current surface to the next surface, the index of refraction, and an aperture. For the next lens to be iterated through it would first be determined if it was spherical or planar, and place one in the correct spot centered on the optical axis. The intersection location and normal between the current ray and the plane/sphere would then be calculated. Using snell's law the refracted ray direction would then be calculated and the current ray would be moved in that direction for the given thickness of the current lens. At any time if the ray is outside of the current lens's aperture the ray is declared blocked and doesn't get considered to providing the color information.

The algorithm is reasonably straightforward but I had a lot of trouble with the calculation of snell's law and due to that I wasn't able to finish my implementation of the lens model which is extremely frustrating. If I had more of a background in those physics or even maybe photography I would have maybe been able to finish. I'll be looking to finish this implementation in the coming months for my own edification.

Other Exploration

Realizing that I was not going to be able to finish my implementation of the lens model I looked for other ways of improving the renders of the basic distributed ray tracing algorithm. From diagrams in the lens model papers I noticed one big difference between the basic depth of field rays and the rays coming out of a lens system were that the lens system rays were pointed to different focal points based on where they were on the lens as seen here:



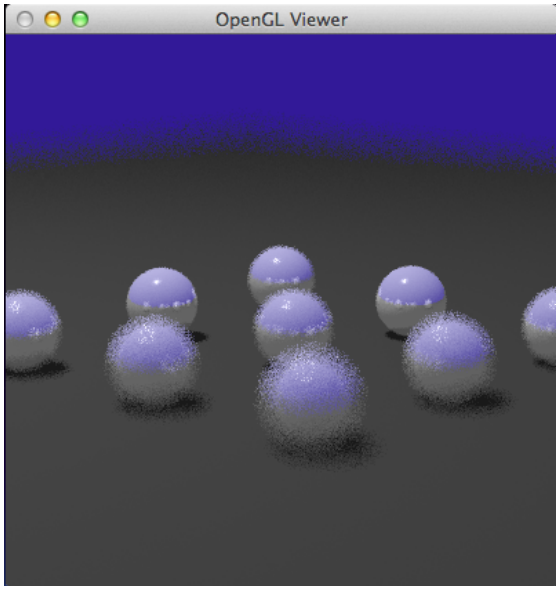
I then adjusted my algorithm to instead of having a static focal plane distance to make it closer to the camera by the distance between the ray's origin and the center of the lens. All I could implement in the time left was a linear change in focal plane distance but a gaussian distribution would probably have created better results.

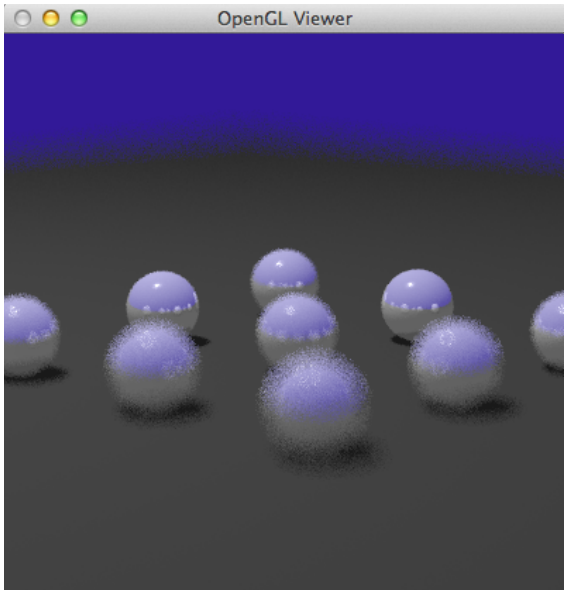
Results

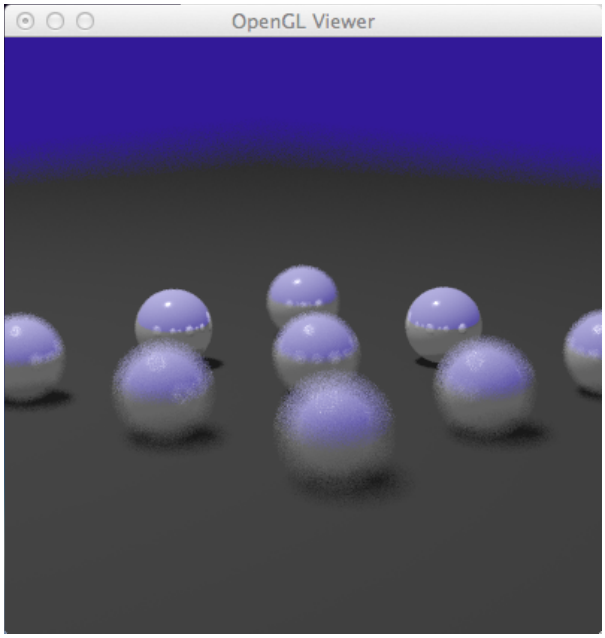
Like anti-aliasing, the depth of field ray tracing takes a long time and a lot of sample rays to produce good looking results. Too small of a number of samples per pixel results in very grainy images and if something is far enough away with such a small number of samples the object

might not even have any representation in the render.

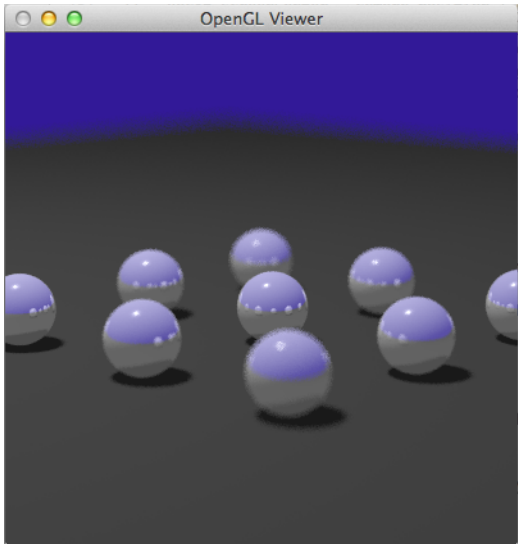
I found to get good results a number around 50 or above should be chosen but for the purposes of a lot of the examples below a number of 25 is instead used for time-saving purposes. Below is the comparison between using first 5 samples, second 10 samples, and third 25 samples:

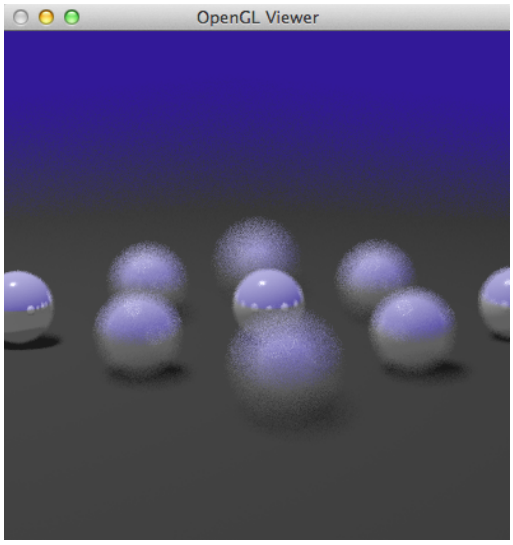


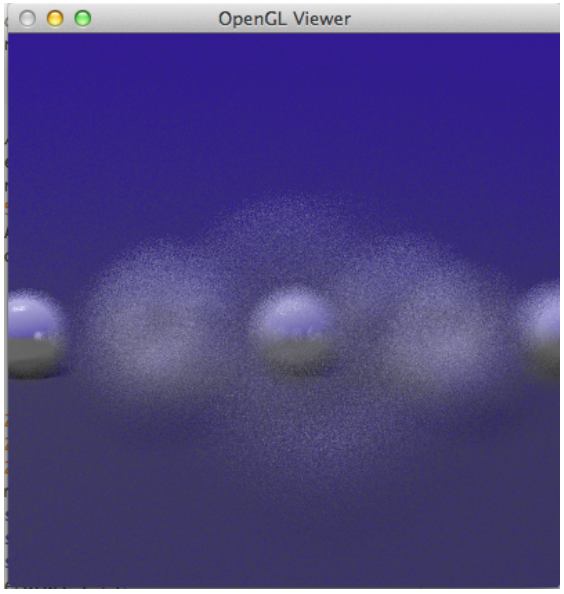




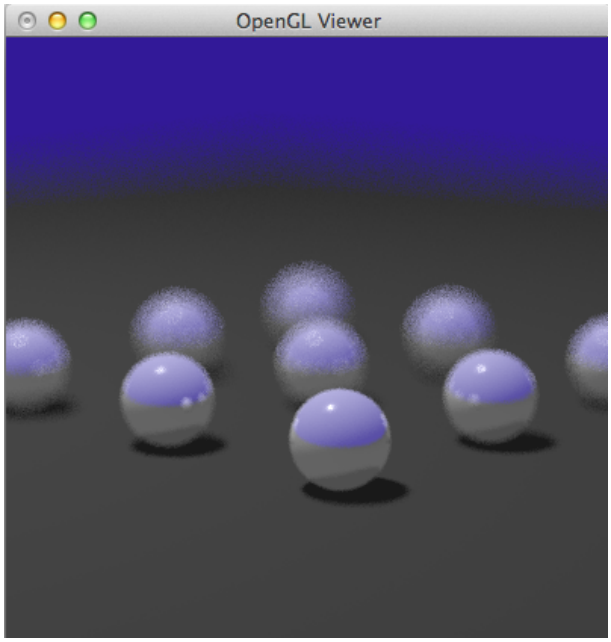
As far as the aperture's effect on the image you'll see in the results below that when choosing a value too small very little blurring occurs. If you however choose a value too high everything get far too blurred and almost seems to disappear from the render. Below is a comparison of renders with no depth of field, an aperture of 0.1, and aperture of 0.5, and an aperture of 2:

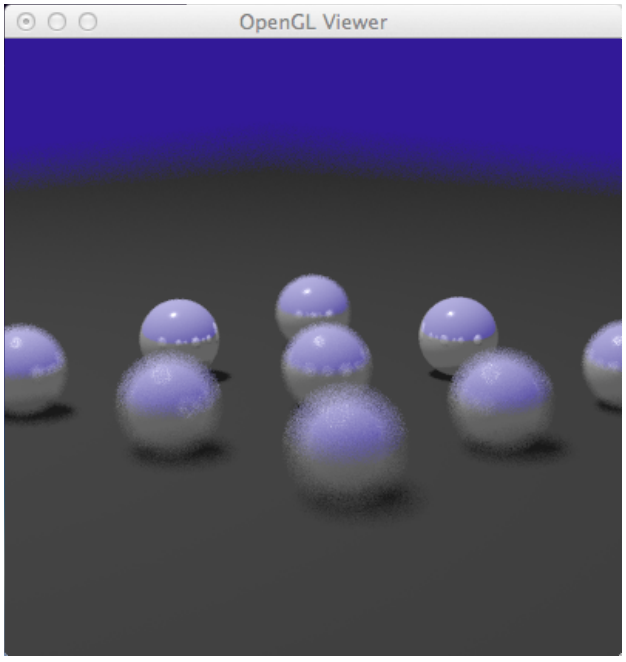






To also show off the differences in choosing different focal lengths here's with a focal length of 7 and 9 respectively:





Conclusion

I spend a really long time invested and working on the lens model and as a result I wasn't able to achieve all the goals I came into this project with but I learned a lot about lenses and depth of field rendering in the process. I look forward to continuing this project further in the future and getting the lens model finished. This project has also inspired me to pursue other methods that I didn't research but came across with some pretty interesting different ways of implementing depth of field.