An Implementation of Depth of Field for Real-Time Rendering

Final Assignment for Real-Time Rendering Discipline

David Beyda
PUC-Rio
dbeyda@poli.ufrj.br

1. Introduction

Depth of Field (DoF) is an effect in which objects within a certain distance range from the camera appear sharp, while objects out of this range appear out of focus. It is used widely in photography, cinematography, and games to direct the attention of the audience to the area in focus. Here we refer to the area in focus as the focus plane. The area out of focus nearer to the observer is referred to as near plane, and the area out of focus more distant than the focus plane is referred to as far plane.

In real world cameras, focus can only be achieved within a certain distance from the lens. Objects nearer and farther than this distance appear out of focus, since in that situation, each point of the image projects an area to the image sensor of the camera. This area is called *Circle of Confusion* (CoC). Generally, an effective DoF implementation results in another effect, called *Bokeh*. It happens when objects are very far in the far plane, and can be seen as the far aperture shape artifacts on the far object.

Throughout the years, several different techniques have been developed to generate DoF in real-time computer graphics. This explores some of them and attempts a naïve implementation that should not be used in real-time graphics due to its performance, but serves as an introduction to the depth of field techniques and as a primary proof of concept that should be refined. Also, most of the techniques studied here are post-processing techniques, meaning they should be easy to append to the end of a graphics pipeline.

2. Background

2.1 Circle of Confusion

According to Demers [1], the CoC can be calculated as the following:

```
CoC = abs(aperture * (focallength * (objectdistance - planeinfocus)) /
          (objectdistance * (planeinfocus - focallength)))
```

Aperture and Focal Length are camera parameters that we should calibrate. Object distance and plane in focus are distances relative to the observer (or camera).

2.2 Basic Techniques

Demers [1] also presents some of the major DoF approaches. From them, the following techniques were considered:

- Accumulation Buffer DoF: consists of rendering the same scene from different points of view around the observer, simulating the aperture. The different results are accumulated on the output buffer and averaged. This gives the best realistic and physically correct result, but requires the whole scene to be rendered multiple times, which is not feasible for real-time rendering.
- Forward Mapped DoF (scattering approach): this technique consists of calculating the CoC of each pixel of your original image, through the depth buffer, and distributing each pixel of the original image to all the pixels inside the CoC, on the output buffer. The scattered color of each input pixel has to be inversely proportional to the CoC size, in order to keep the luminous energy of the scene intact. This technique is faster than the accumulation buffer DoF, but requires writing to multiple targets, which is slow on the current hardware.
- Reverse Mapped DoF (gathering approach): similar to the Forward Mapped DoF, but instead of dispersing each original pixel to its neighbors, we inverse the logic. For each output pixel, we sample its neighboring pixels on the original image, and only accept the pixels that have a CoC large enough to reach the current output pixel. Accepted pixels are then averaged to give a final result. This technique is much faster than the previous, requiring multiple samplings but only a single writing.

The Reverse Mapped DoF was then chosen as the primary proof of concept to be implemented by this work, due to its performance and potential for improvement, that will be discussed later in this paper.

3. Implementation and Results

3.1 Calculating the Circle of Confusion

The initial step to most DoF implementations is to correctly calculate the CoC. This is rather hard, due to the fact that the scene dimensions were detached from reality, and thus, the resulting dimension of the CoC was not known. The lens *focal length* and *aperture* parameters also impact the CoC. This means that for each attempt to correctly calculate the CoC and render it to the screen, several *focal length* and *aperture* values had to be tried. This was by far the most time-consuming step.

The CoC was obtained by first rendering the whole image and depth buffers to an offscreen buffer, and then rendering the CoC to a new single channel color buffer, while sampling from the original offscreen rendered image. The CoC was also artificially clamped between 0 and 0.0015.

3.2 Rendering Final Image

The next step is to sample the CoC texture and the original image texture, and output the resulting color for each pixel. To do this, we sample the original image using a specific kernel, and for each sample, if the sample CoC is greater than the sample distance to the current pixel being rendered, that sample contributes to the current pixel's color. The way that this contribution is weighted can vary, as well as the kernels used.

According to McInstosh et al. [2], the kernel shape simulates the aperture in the camera, meaning that a *bokeh* will be formed in the same shape of the kernel. For this implementation an octagonal kernel with 37 samples was used. A polygonal kernel was chosen in favor of a circular one, because of its separable propriety. This means that we can obtain the same result with multiple rendering passes in different directions, using less samples. We did not take advantage of this property in this implementation, however, McIntosh et al. [2] describes how to effectively simulate polygonal kernels using separability, including the octagonal one.

Different ways of weighting the samples were tried. Between the weighting suggested by McIntosh et al. [2], the uniform weighting, the uniform weighting gave better results in this implementation. Additionally, in the cases where a sample was rejected, the current average was used instead, allowing the average to grow stronger.

The only artifact left was due to discontinuities in the CoC buffer. When varying the focus plane, the transitions through different CoC values is not smooth. Even after applying an average blur to the CoC buffer, the discontinuities were still present. Having failed to identify the origin of that problem, further investigation is needed to correct those artifacts. Maybe the discontinuities in the CoC are due to discontinuities in the depth buffer, and a depth buffer with linear precision would give better results. In addition, the *frames per second* rate went from 410 (without depth of field) to 260, running on a Nvidia GTX 970.

Figures 3 and 4: final result (on the left) and achieved bokeh effect when camera is far away (on the right)

4. Suggested Improvements

Performance improvements could be achieved implementing the separable polygonal bokeh technique described in McIntosh et al. [2]. He shows how to achieve DoF and bokeh of most regular convex polynomial shapes, with multiple linear blur/averaging passes in different directions, using the separability property to reduce the number of samples taken. Circular bokeh is also possible with a separable approach, as presented by Garcia [3] in a SIGGRAPH talk, in 2017. His proposed technique is based on a separable complex approximation of a circle, from Niemitalo [4]. The results are impressive and were shipped in a couple of games.

5. Conclusions

The DoF effect was implemented successfully, but some artifacts could not be corrected. The bokeh was also achieved. As this implementation is proposed to be an introduction to the DoF techniques, there are faster (but more complex) algorithms that are more suitable for real-time rendering. Finally, implementing a realistic and performant DoF is still a challenge to present days, and certainly there is still a lot of room for further improvements.

References

[1] Demers, J. (2007). **Depth of Field: A Survey of Techniques**. GPU Gems, Nvidia. Available at: https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems_ch23.html

[2] McIntosh, L. & Riecke, Bernhard & Dipaola, Steve. (2012). **Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader**. Computer Graphics Forum.

[3] Garcia, K. (2017). **Circular Separable Convolution Depth of Field**. SIGGRAPH Talks 2017. Available at: https://www.youtube.com/watch?v=QKhydJSbcno

[4] Niemitalo, O. (2010). **Circularly symmetric convolution and lens blur**. Available at: http://yehar.com/blog/?p=1495