# DEPTH OF FIELD IMPLEMENTATION WITH OPENGL

*Tin-Tin Yu*
*Department of Computer Science*
*Michigan Technological University*
*Houghton, MI 49931*
*E-mail: tiyu@mtu.edu*

**Abstract**

This paper presents a post rendering technique for simulation of the depth of field effect using OpenGL. The effect of depth of field is widely used in photography to make the main subjects prominent by blurring foreground and background subjects. Simulating this lens effect does not require complex computation and can be achieved instantly. This paper reviews the concept of depth of field and presents a method for building a filter that blurs the out-of-focus area by constructing a simple lens model.

## 1. INTRODUCTION

If a camera lens is focused on a subject, only an area surrounding that subject will appear sharp (in-focus). Subjects outside of this area will appear in blurred or out-of-focus in varying degrees. Depth of field (DOF) is referred to as the distance range within which subjects in an image appear sharp [11]. In Figure 1, the spider appears sharp while the foreground and the background look blurred. As long as the subject is in the depth of field, it will appear sharp, and a blurred foreground and background can make the main subject (i.e., the spider) stand out. Note that depth of field depends on many factors such as the focusing distance, aperture, and the size of the circle of confusion. In photography, one frequently uses a larger aperture and longer focal length to achieve shallow depth of field in order to emphasize the main subject.


Figure 1 : Depth of Field Effect

DOF effect can be simulated in many different ways. The simplest one is to find out and render the out-of-focus zone, apply a blur filter, and use this image as a background. However, the result may not be realistic and is only good for still images. In real-time interactive graphics and computer animations, since the camera settings and objects in the scene are dynamic, this technique is inadequate. Instead, a camera model with aperture, focal length and screen position is introduced to perform per-pixel calculation of a more realistic DOF effect [1]. This paper will present a technique for simulating DOF. Section 2 introduces the concept of circle of confusion (CoC). Section 3 presents an implementation using OpenGL. Some limitations will also be discussed. Section 4 provides possible future improvements of the technique. Section 5 has conclusions.

## 2. THEORY

The *Circle of Confusion* (CoC) is the circular distribution of light from a point onto an image plane [11]. This definition is based on an ideal single lens camera model. In Figure 2, lens is focused on dot B on the left side of the figure. Thus, its

image appears as a sharp point on the image plane. On the other hand, dot A and dot C are out-of-focus. The images of dot A and dot C are behind, and, in front of the image plane. Since the light distributed on the image plane is no longer a sharp point but a circle, the CoC, a blur image is produced [9]. In this paper, the image plane is the screen.

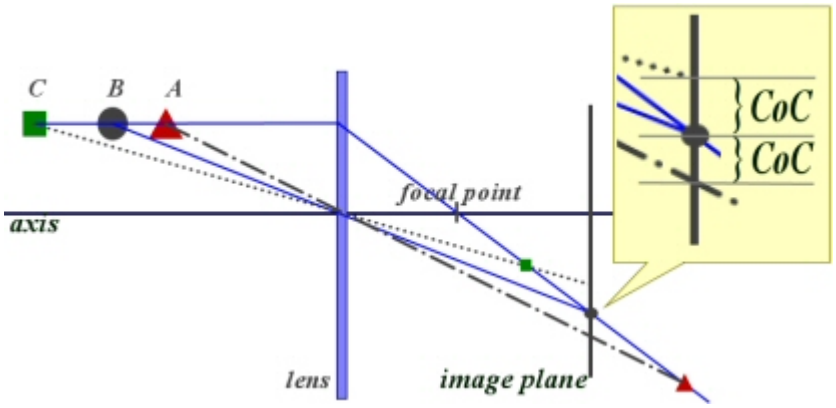



Figure 2: Formation of Circle of Confusion

Given a point $X$ in space, the diameter of the CoC of $X$ can be calculated as in Figure 3. Let $a$ be the height of subject $X$ from the lens $z$-axis, $d$ be the subject distance, $s$ be the screen distance, and $f$ be the focal length of the lens. If $X$ is out of focus, its image on the image plane is a circle, the CoC of $X$. Let the "heights" of this blurred $X$ be $x$' and $x$" as shown in Figure 3. Then, the diameter of the CoC of $X$ is the absolute value of $x" - x'$.

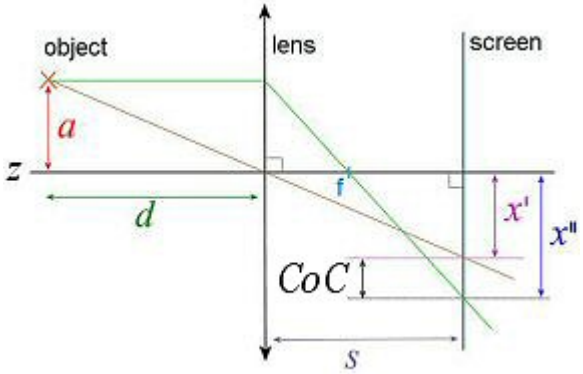
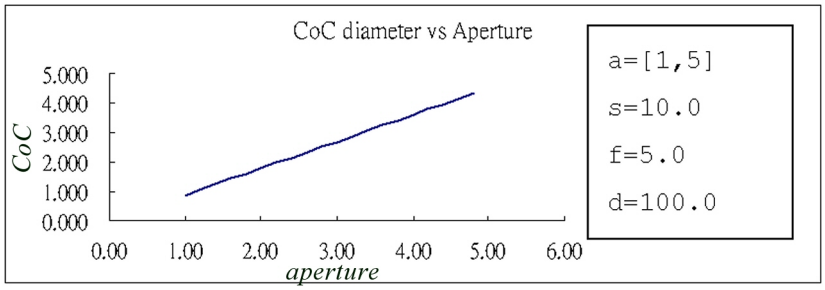



Figure 3: Calculation of Circle of Confusion

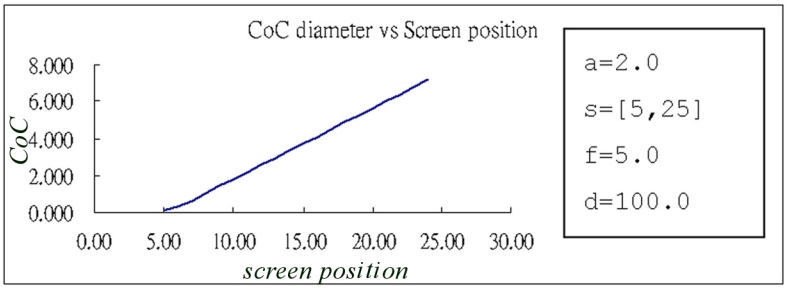
Since $x'/a = s/d$, we have $x' = a(s/d)$. Since $x"/a = (s - f)/f$, we have $x" = a(s - f)/f = a(s/f - 1)$. Therefore, the diameter of the CoC of $X$ is

$$\begin{aligned} \text{CoC diameter} &= |x'' - x'| \\ &= \left| a\left(\frac{s}{f} - 1\right) - a\left(\frac{s}{d}\right) \right| \\ &= \left| a\left[ s\left(\frac{1}{f} - \frac{1}{d}\right) - 1 \right] \right| \end{aligned}$$

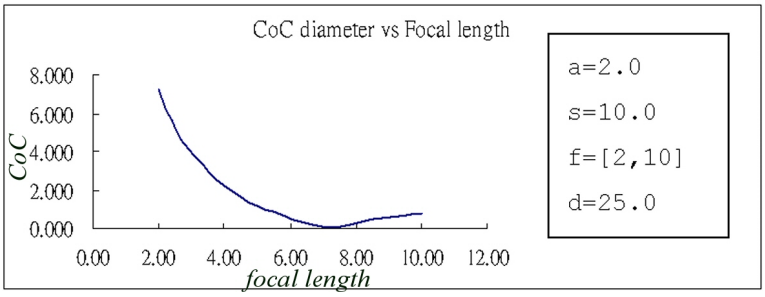
The following four graphs show the relationships between the diameter of CoC and the configuration of the four variables ($a$, $s$, $f$, $d$). The following graph shows a linear relationship between the CoC diameter and the subject distance $a$ from the $z$-axis, where $a \in [0, 4]$. The CoC diameter increases as $a$ increases.




The numbers are in theoretical units rather than actual photographic related units. The following graph shows the linear relationship between the CoC diameter and screen position $s \in [5, 25]$. The CoC diameter is set to be zero at 5.0. As the screen move from this position, the diameter of CoC increases linearly.

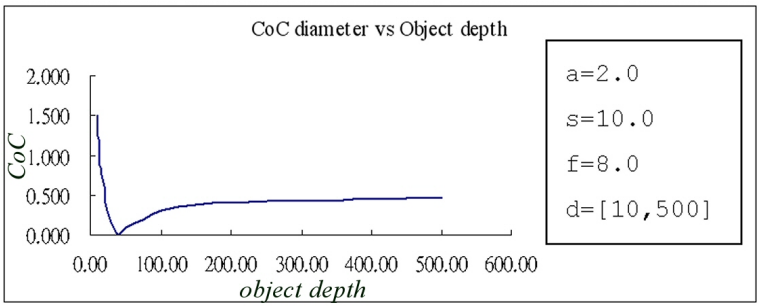



The following graph shows a non-linear relationship between the CoC diameter and focal length $f \in [2, 10]$. It is clear that the CoC diameter decreases as focal length goes from 2.0 to 7.14. Note that 7.14 is the focal length required to focus at distance 25.0 using settings described in the diagram. However, CoC diameter increases in a much smaller rate as the focal length goes from 7.14 to beyond this value.




The following graph shows the non-linear relationship of the CoC diameter and subject distance $d \in [10, 500]$. The camera is focused at distance 40.0, and every point not at the distance of 40 will produce a non-zero CoC. The graph shows similar properties as the previous graph. The CoC diameter decreases in a very high rate when

the object moves from the camera to the focused distance. It increases in a much lower rate when the object moves away from that point.




Since a perfectly sharp image only occurs when the CoC diameter is zero, by setting the CoC diameter to 0, we have $d = s * f/(s - f)$, where the value of $d$ is the camera focuses on, and a point appears sharp only if it is located at a distance of $d$ from the camera. However, since each pixel on a screen has a non-zero size, a point appears sharp on screen as long as its CoC diameter is less than or equal to the size of a pixel [9]. Thus, the range of $d$ from which a point produces a CoC within the pixel size is referred to as the depth of field with respect to a fixed pixel size. If a point has its CoC diameter larger than the size of a pixel, its light contributes to each pixel inside the CoC as Figure 4. We shall discuss how the light distribution can be implemented in the next section.

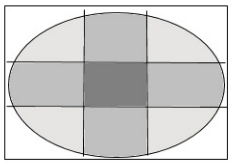


Figure 4: Distribution of light

## 3. IMPLEMENTATION

### A Different Camera Model

The OpenGL camera model works as a perfect pinhole camera. There is no lens and the aperture is zero in size. Each point of an object has exactly one path that goes through the theoretical hole. It is then projected into the viewing plane (Figure 5). As a result, the image is perfectly sharp on the entire viewing plane.

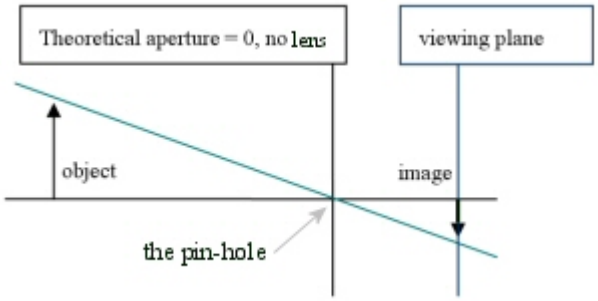



Figure 5: OpenGL Camera Model

To implement the DOF effect, a new camera model that can support aperture size and the focal length of the lens is needed [1]. In this paper, aperture is defined to be the radius that allows light to enter the camera as shown in Figure 6. A larger aperture yields a larger CoC [1]. For DOF effect simulation, our implementation replaces the variable *a* of the CoC equation with an aperture coefficient. The new camera model is used only to compute the size of CoC of each pixel. The position and the size of an object remain unchanged as they are computed using the OpenGL camera model.

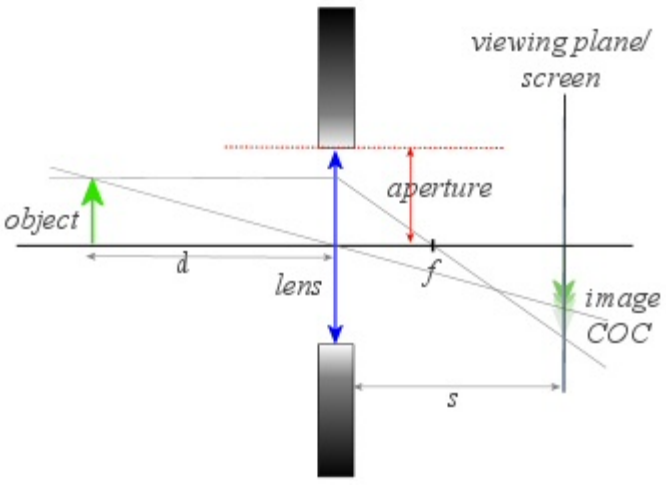



Figure 6: new camera model

The algorithm that simulates the DOF effect filter consists of three main steps. First, it reads the information from the frame buffers generated by the regular OpenGL model, which includes the color and the depth value of each pixel. Second, it applies

the CoC equation with the given parameters of the new camera model to each pixel. Then, the color of each pixel is distributed to the neighboring pixels within its calculated diameter of CoC. The data is accumulated into a new frame buffer. Third, the original buffer is replaced by the new buffer.

**Light Distribution Function**

The light distribution part of the second step of the algorithm should be modularized for more flexible design which permits replacement by a more advanced module when needed. This will be discussed in Section 4. One of the simplest light distribution functions is based on the uniform circle method. This method distributes the intensity of light from a point onto a circle evenly. Suppose the color of the pixel at ($x$, $y$) is to be evenly distributed onto the neighboring $\pi r^2$ pixels, where $c$ is the CoC diameter, "even distribution" means that each neighboring pixel received the same amount of intensity such that the sum of the intensity of these pixels is equal to the original color intensity of pixel ($x$, $y$). This light distribution procedure has parameters $c$, point ($x$, $y$), a *frameBuffer1* that contains the color data of the original raster, and a *frameBuffer2* that contains the newly computed color data. Then, the color intensity of the pixel is divided by the area of its corresponding CoC and this value is accumulated into the new buffer *frameBuffer2*.

```
Function UniformDistribution( c, x, y, frameBuffer1, frameBuffer2 )
    r ← c/2
    a ← π r^2
    intensity ← frameBuffer1[x, y]/a
    for ( row ← -r to r )
        for ( col ← -r to r )
            if ( ( row^2 * col^2 )^0.5 < r )
                frameBuffer2[row, col] += intensity
```

**Implementation Procedures**

Adding DOF effect into an OpenGL program does not require a major change to the original program codes. The DOF effect adds extra steps near the end of the OpenGL display function. More precisely, it processes the image resulting from the OpenGL model in three steps, mentioned previously, and replaces the image by a filtered one. The following shows the steps of an OpenGL display function that implements the DOF effect.

1. Using OpenGL routines to render the scene into the frame buffer. However, before swapping the buffer for display, the following procedures are added.

2. In order to work with 2D raster coordinate, switch to 2D orthogonal projection.

3. Read color buffer into local buffer *fb1* and depth buffer into local buffer *db* using the OpenGL call, *glReadPixels ()*.
   *glReadPixels(0, 0, w, h, GL_RGB, GL_FLOAT, fb1)*
   *glReadPixels(0, 0, w, h, GL_DEPTH_COMPONENT, GL_FLOAT, db)*

4. Convert the normalized depth values ($z$) to object space values ($Z$) for each pixel in *db* [4].
   $Z \leftarrow Znear*Zfar / (Zfar - z*(Zfar-Znear))$
   This step is essential since the CoC calculation is based on the object space. The depth value of a pixel should be the unit distance between the pixel and the lens.

5. Initialize a secondary buffer fb2. The new image is formed by accumulating light into this new buffer.

6. The core of the DOF filter: Calculation of the CoC is performed and distribution function is applied for each pixel.
   *For each scan line,*
   *For each pixel (x, y) on the scan line*
   *PixelDepth ← db[x, y]*
   *CoC diameter ← abs( Aperture-Coefficient **
   *( ScreenPos*(1.0/FocalLen - 1.0/PixelDepth) - 1 ) )*
   *UniformDistribution(CoC diameter, x, y, fb1, fb2)*

7. Replace the current raster by the new buffer fb2.
   This is performed by an OpenGL call *glDrawPixels ()*.
   *glDrawPixels(w, h, GL_RGB, GL_FLOAT, fb2)*

The result of these procedures can be verified by examples generated by our program **DOF_Filter**.

**Screenshots**

**DOF_Filter** is a program that implements the above procedures. The main display is a scene of three identical pictures (Tokyo night) at different distances. The scene can be rotated by dragging the mouse across the main display. Users can adjust the focusing distance, focal length, and the aperture of the camera model by entering values into the corresponding textbox on the left (Figure 7). Once the values entered into the program, screen position is adjusted automatically so that the camera is focused at the specified distance.


Figure 7: DOF_Filter interface

The program also provides options for the user to evaluate and compare the result by viewing the depth map (Figure 8a). The color intensity associated with each pixel represents the depth of the pixel. The lighter the color, the farther the pixel is from the camera. Figure 8b, 8c and 8d are the examples produced by the program when DOF filter is turned off, focused very closely to the front ground, and focused at nearly infinity, respectively.

(a) (b) (c) (d)

Figure 8: Sample Output of **DOF_Filter**

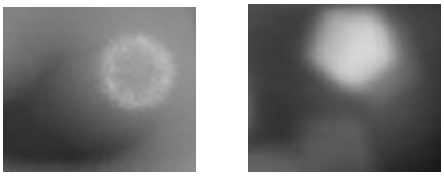
**Limitations**

Since OpenGL uses a local illumination model, the DOF effect implementation also inherits the "local" property. For example, a reflected image will not filter correctly since the OpenGL depth buffer only stores the depth of the mirror's surface instead of the depth of the reflected object. The current implementation also simulates an ideal lens. The effect is not as realistic as that obtained from a real camera lens. This issue is discussed in the future development section.

## 4. FUTURE DEVELOPMENT

The current implementation of the DOF effect has two major shortcomings: (1) only an ideal lens model is used, and (2) performance may not meet the need of real-time graphics. Therefore, improvements will be needed to overcome these two issues: the development of more advanced light distribution functions and a more efficient method that takes advantage of current graphics hardware.

Our light distribution function is modularized to make the design more flexible. In fact, more realistic blur effects can be implemented by building different light distribution functions that correspond to different type of lenses. These techniques that generate blurs are often referred to bokeh rendering [3]. Bokeh is a Japanese word which means the quality of an out-of-focus area [6]. Figure 9(a) and 9(b) show some examples of realistic bokeh rendering functions [3], [2]. A possible future development is the use of the bokeh rendering techniques in building more advanced light distributions.



(a) (b)

Figure 9: Bokeh Rendering

The most important future development is how to take advantage of graphics hardware. In real-time interactive graphics, performance has a higher priority than the quality of rendering in most cases. The technique presented in this paper uses software only, and processing speed depends on the CPU speed and memory bandwidth. However, if most or all computations are done by the graphic processor using its onboard memory, performance can be improved dramatically. In fact, ATI research group has suggested the use of a `GLmem` object which uses an OpenGL extension to create and release onboard graphic memory accessible to OpenGL fragment programs

which are often referred to pixel shaders. Hence, it is possible that a DOF filter can be carried out entirely using pixel shader [8].

## 5. CONCLUSION

This paper presented the concept of depth of field effect and its implementation using OpenGL. The depth of field effect is produced by blurring the out-of-focus areas. The result is used to make main subjects in a scene stand out. The blurring technique is performed by calculating the circle of confusion of each pixel and applying a light distribution function to each pixel.

This paper also presented a program, **DOF_Filter,** that implements the DOF effect. This program allows the user to input values of various DOF variables and rotate the camera to visualize and understand the DOF effect. Possible future developments will consider efficiency and rendering improvements, which will include better light distribution functions and the uses of a pixel shader to take advantage of graphics hardware.

## REFERENCES


[1] Birn, Jeremy. *[digital] Lighting & Rendering*. New Riders Publishing. 2000.

[2] Blevins, Neil. Faking Bokeh. Retrieved from www.neilblevins.com/cg_education/faking_bokeh/faking_bokeh.htm. 2002.

[3] Buhler, Juan. A Phenomenological Model for Bokeh Rendering. Retrieved from www.flarg.com/bokeh.html. 2002.

[4] Kenneth E. Hoff III. Conversion Between OpenGL Depth-Buffer Z and Actual Screen-Space Depth. Retrieved from http://www.cs.unc.edu/~hoff/techrep/openglz.html. 1998.

[5] Kerlow, Isaac V. *The Art of 3-D Computer Animation and Imaging*. Van Nostrand Reinhold. 1996.

[6] Merklinger, Harold M. A Technical View of Bokeh. *Photo Techniques*. (1997, May/June).

[7] Microsoft. OpenGL Reference. *MSDN*. 2002.

[8] Percy, James. OpenGL Extensions. *ATI Research - SIGGRAPH 2003*. 2003.

[9] Shene, Ching-Kuang. Depth of Field. Retrieved from www.cs.mtu.edu/~shene/DigiCam/index.html. 2002.

[10] Vestal, David. *The Craft of Photography*. Harper & Row Publishers. 1975.

[11] Wall, E. J., and Jordan, F. I. *Photographic Facts and Formulas*. American Photographic Book Publishing Co., Inc. 1975.

[12] Watt, Alan & Watt Mark. *Advanced Animation And Rendering Techniques*.

New York: ACM Press. 1992.

**ACKNOWLEDGEMENTS**

I would like to acknowledge the help of Dr. C-K Shene and Dr. John Lowther, whose conversations and guidance throughout the project has been most helpful. The development of **DOF_Filter** is supported by the National Science Foundation under grant number DUE-0127401.