Depth of Field for Photorealistic Ray Traced Images

JESSICA HO AND DUNCAN MACMICHAEL

MARCH 14, 2016

CSS552: TOPICS IN RENDERING

Problem Statement

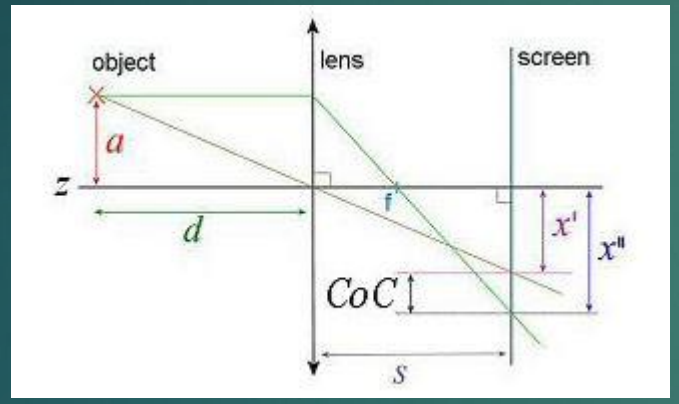
- The Phong Illumination Model and ray tracing engine explored thus far are flexible in the kinds of illumination they can simulate, but are only part of a larger formula for producing photorealistic ray traced images
- Part of the reason that basic ray tracing looks fake because of "jagged edges, hard shadows, everything is in focus, objects are completely still, surfaces are perfectly shiny, and glass is perfectly clear" (Hart, p. 2) [1]
- Depth of Field is needed as one way to enhance realism

Problems with Computer Generated Cameras

- In real life, cameras and our eyes filter light through a lens and control the amount of light allowed onto the image plane (image sensor or retina), which, according to Demers (2004) [6], causes some items to appear out of focus depending on their distance from the focal plane and the projection
- Default 3D camera model does not replicate true camera functionality (such as controllable focal distance and aperture), meaning the image is rendered in perfect focus and the viewer immediately realizes that the image is computer-generated
- Our current camera does provide focal distance but is not configurable by the user ("Focal Distance = Length(lookAtPoint – eyePoint)" (Harvey, 2012) [7])

Use Z-Buffer (depth information) to post-process DoF (Yu, 2004) [10]

Slightly different camera model and simplified CoC calculation

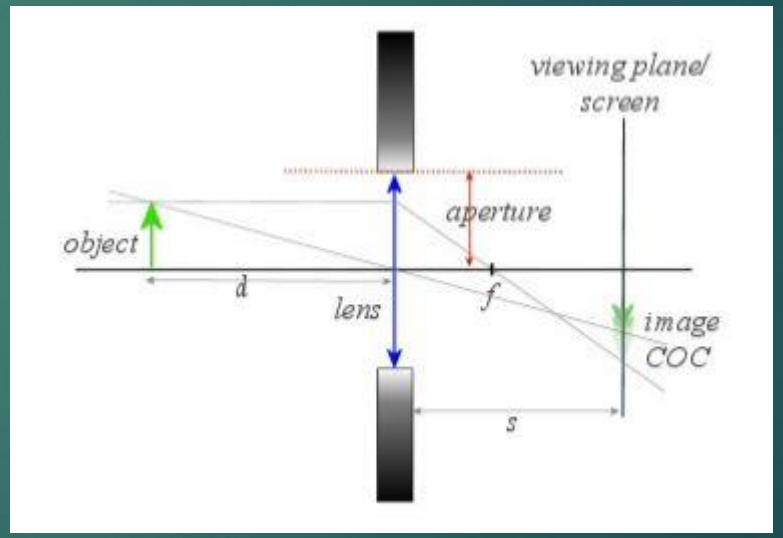


(Yu, 2004) [10]

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

(Yu, 2004) [10]

Simpler camera model with single aperture coefficient



(Yu, 2004) [10]

Algorithm (Yu, 2004) [10]

- Algorithm developed by Tin-Tin Yu from Michigan Technological University [10]
- Procedure involves calculating CoC, then applying a Uniform Distribution algorithm to blur the pixel and distribute light (Yu, 2004) [10]
- Our approach differs in that Yu manually calculates object depth distances (Z ← Znear*Zfar / (Zfar - z*(Zfar-Znear)) (Yu, 2004) [10]), but ours has this information already calculated

```
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)
```

Yu, 2004 [10]

```
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
```

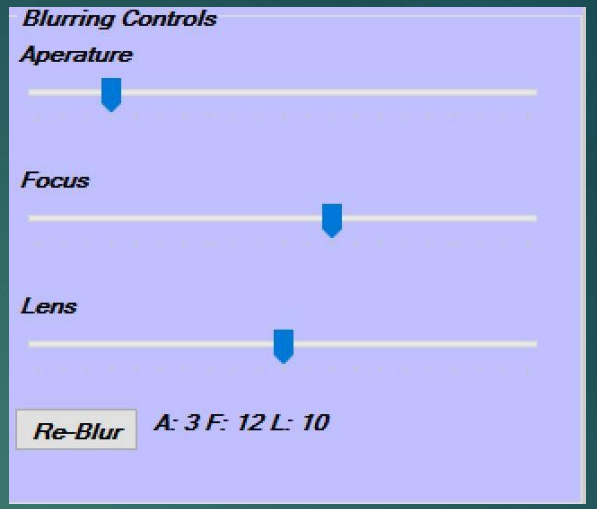
Yu, 2004 [10]

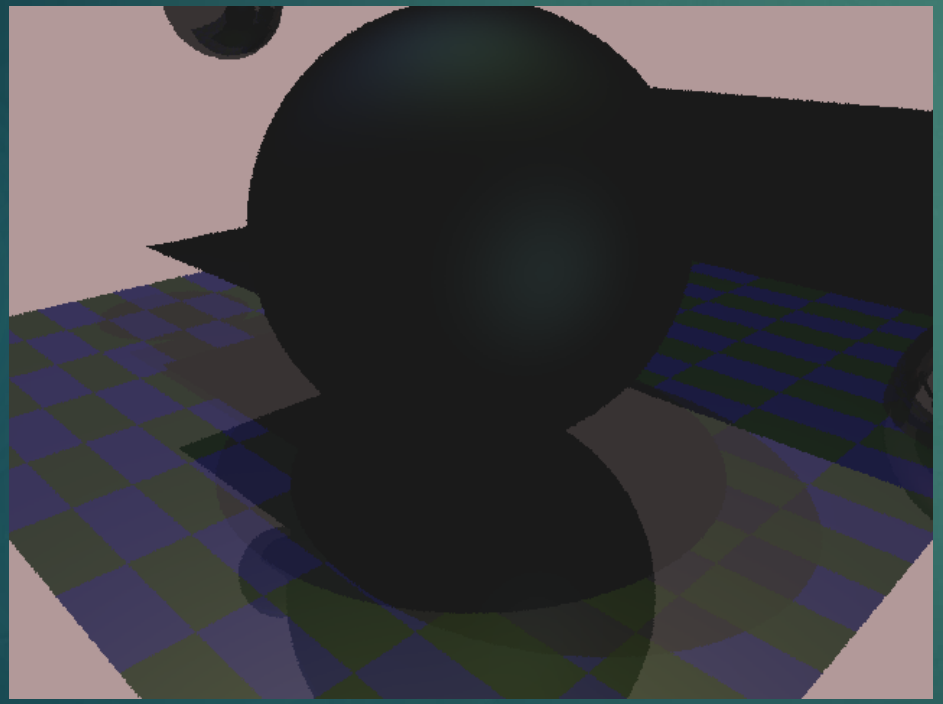
Implementation

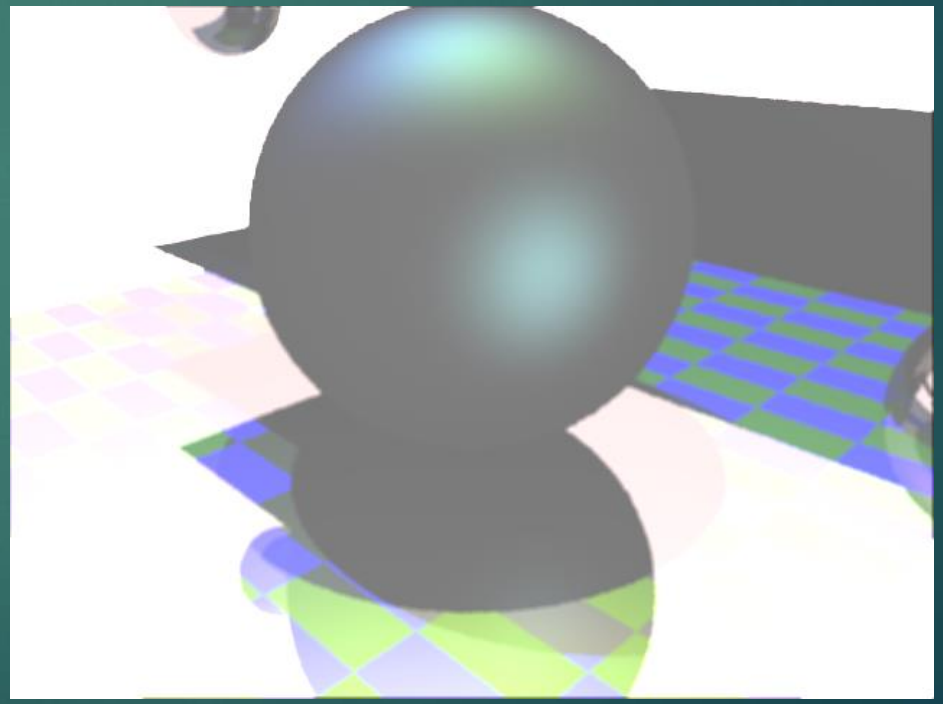
- RTCamera: added lens distance, aperture and focal distance variables w/getters
- RTCore_Compute_DepthOfField: created new class to take existing image and use BlurImage() to blur it with UniformDistribution method.
- RTCore_Utilities: added utility functions to compute a pixel's height and width (taken from MP1)
- RTCore_ResultStorage: added fourth mResultBlurredImage as fourth Bitmap variable, initialized it with SetResultColors. Added public Vector3[][] pixelColors framebuffer to use to store original color information in Vector3 form (which is then used to calculate blurring). This was to avoid converting between System.Drawing.Color and Vector3, which got messy.
- RTCore_Compute: added line to add computed pixel color to frame buffer to pass in to depth of field calculations.
- RtCore_Thread: Called BlurImage(mPixelColors) at the end of RTRenderingDone (single-threaded only!). Added RTBlurredRenderingDone (called by BlurImage function in RTCore_Compute_DepthOfField) which calls RTWindow.SetBlurredImage
- RTWindow: Added SetBlurredImage, which replaced the pixel coverage Bitmap with our blurred image Bitmap. Changed RTWindow_Paint routine for repainting the subwindows to have mMaskGraphics draw the mResultBlurredImage instead of the mPixelCoverage if ShowDepth is not checked. Added reBlurImage method to re-blur the image based on new input from the user.
- RTCore_SceneDB: added SetCameraBlurring method to set the camera variables based on the user's input
- 3DPreviewerDesigner: added blurring controls.

Results 1

Blurring on the outer edges of the image is seen in particular along the checkered pattern of the image. This illustrates an exceedingly large CoC due to the controls we used.

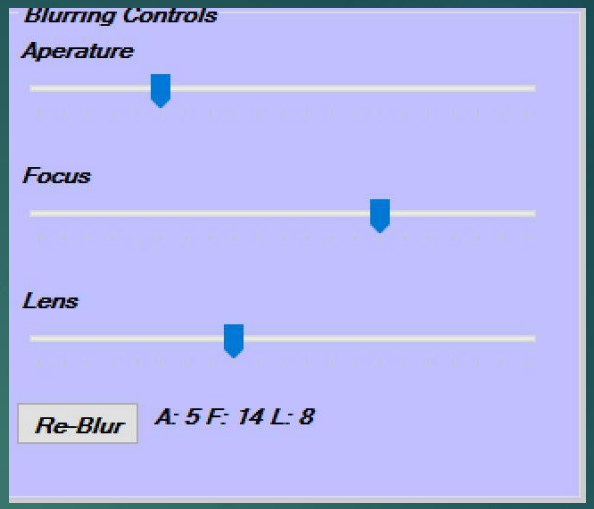


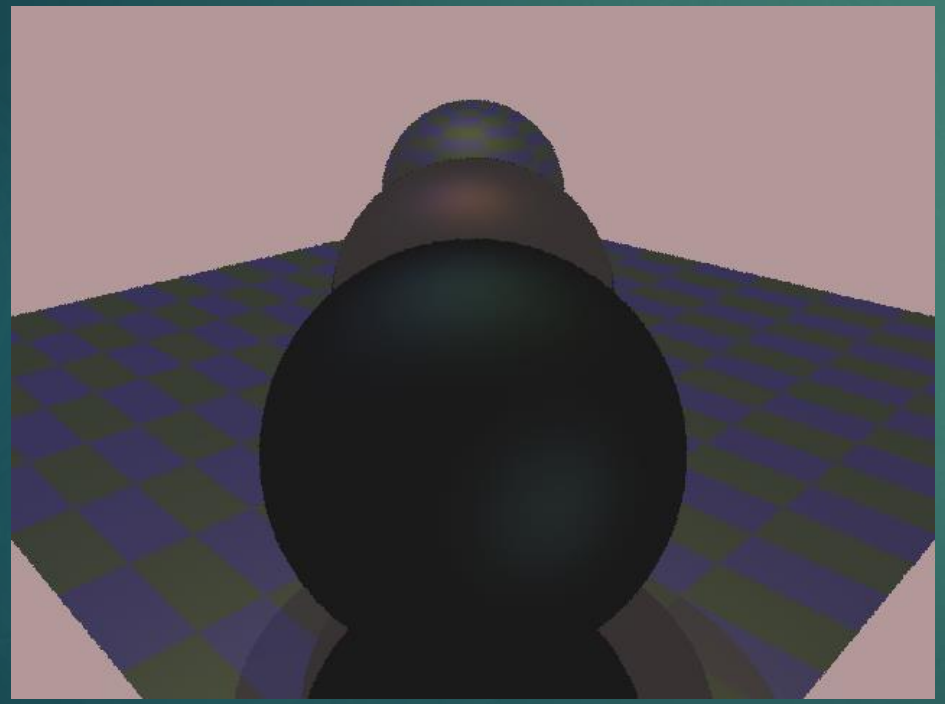


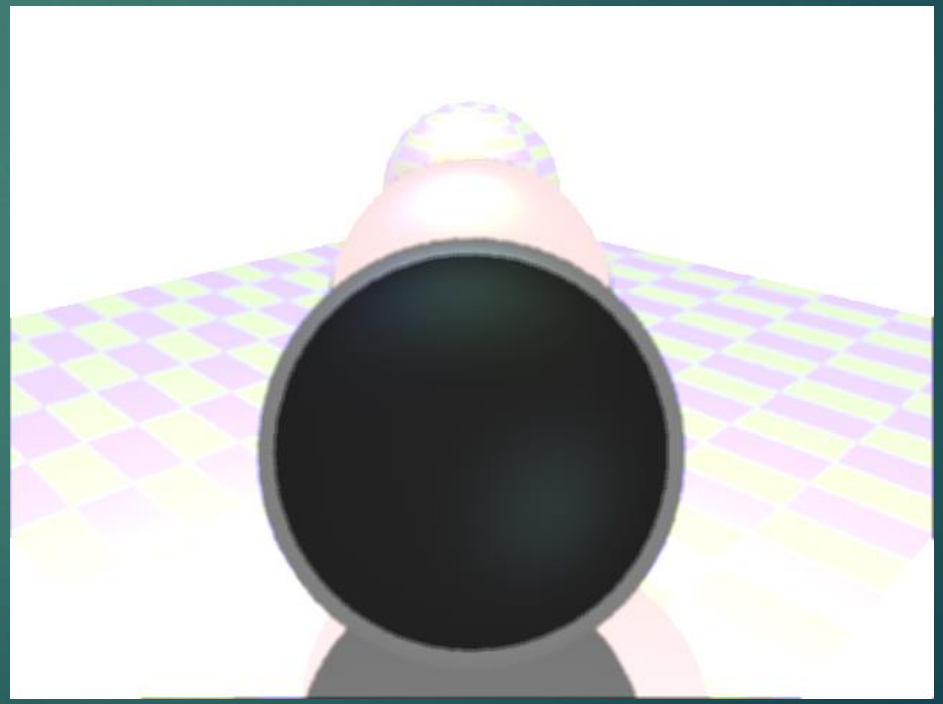


Results 2

Having the values set below, gave the illusion that the CoC was being calculated correctly and that the darker area was in focus, however, this is not the case. The CoC of the darker area below is still out of focus. This effect was achieved with non-realistic lens distance and focal distance values.

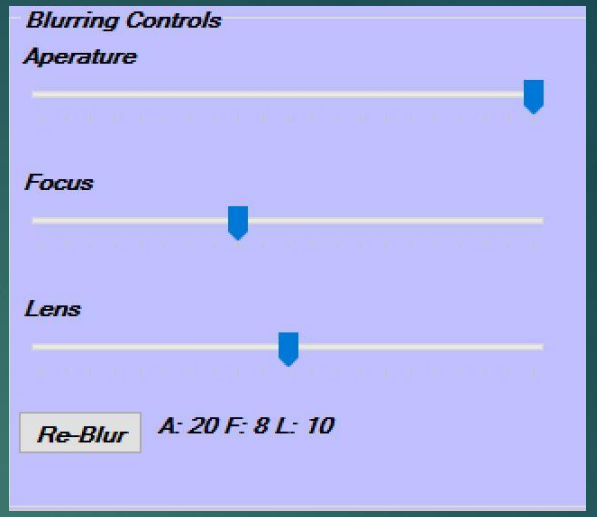


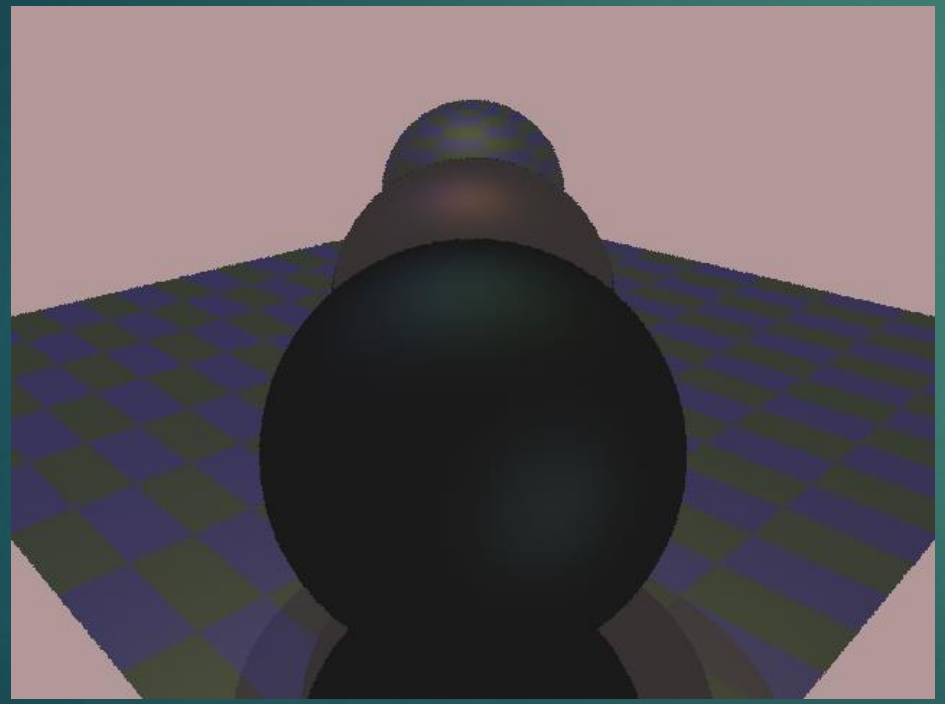


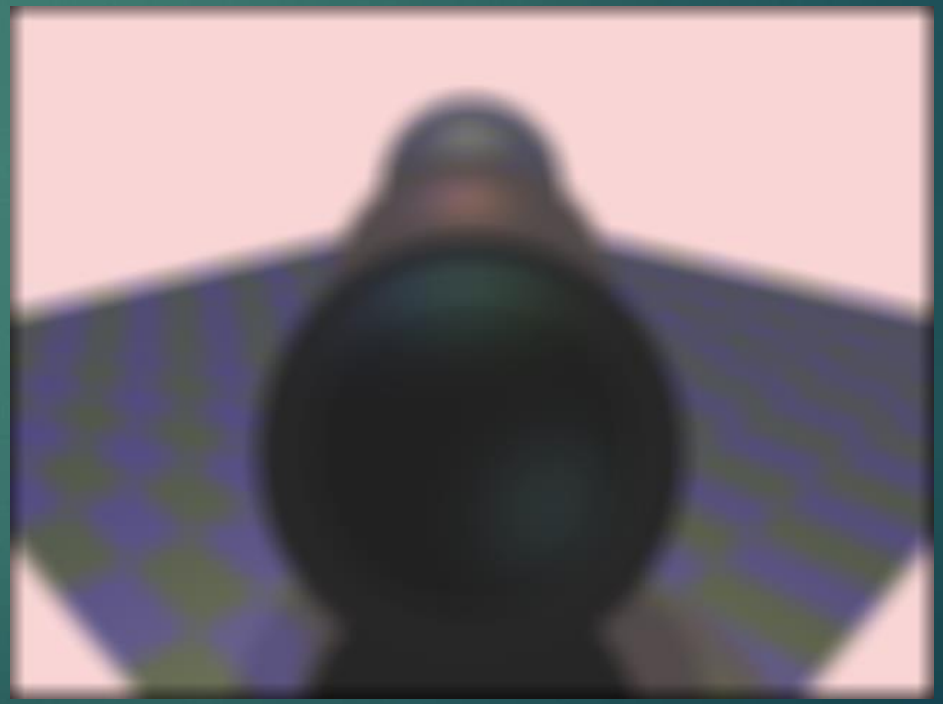


Results 3

The entirety of the image is blurred with a large aperture. This is true to life where large apertures cause anything far away from the camera to be very blurred.







References

1. Hart, J. C. (n.d.). Distributed Ray Tracing. Retrieved February/March, 2016, from http://luthuli.cs.uiuc.edu/~daf/courses/computergraphics/week3/distributed-final.pdf
2. Distribution raytracing. (n.d.). Retrieved February/March, 2016, from http://www.cs.unc.edu/~jpool/COMP870/Assignment2/
3. IneQuation (2012, April 04). How to implement Depth of Field in Ray Tracer? Retrieved February/March, 2016, from http://stackoverflow.com/questions/10012219/how-to-implement-depth-of-field-in-ray-tracer
4. Lee, Skeel (2007, March). CG:Skeelogy - Depth Of Field Using Raytracing. Retrieved February/March, 2016, from http://cg.skeelogy.com/depth-of-field-using-raytracing/
5. Hammon, E., Jr. (2004). GPU Gems 3 - Chapter 28. Practical Post-Process Depth of Field. Retrieved February/March, 2016, from http://http.developer.nvidia.com/GPUGems3/gpugems3_ch28.html

References Cont.

6. Demers, J. (2004). GPU Gems - Chapter 23. Depth of Field: A Survey of Techniques. Retrieved February/March, 2016, from http://http.developer.nvidia.com/GPUGems/gpugems_ch23.html
7. Harvey, S. (2012, December 21). Ray Tracer Part Six – Depth Of Field. Retrieved February/March, 2016, from https://steveharveynz.wordpress.com/2012/12/21/ray-tracer-part-5-depth-of-field/
8. Depth of Field (ray tracing) - Graphics Programming and Theory. (2005, October 19). Retrieved February/March, 2016, from http://www.gamedev.net/topic/352850-depth-of-field-ray-tracing/
9. Barsky, B. A., & Kosloff, T. J. (n.d.). Algorithms for Rendering Depth of Field Effects in Computer Graphics. Retrieved February/March, 2016, from http://zach.in.tu-clausthal.de/teaching/cg_literatur/Rendering Depth of Field Effects Survey.pdf
10. Yu, T. (2004). Depth of Field Implementation with OpenGL. Retrieved February/March, 2016, from http://www.cs.mtu.edu/~shene/PUBLICATIONS/2004/CCSC-MW-2004-depth_of_field.pdf