Depth of Field for Photorealistic Ray Traced Images

JESSICA HO AND DUNCAN MACMICHAEL

MARCH 7, 2016

CSS552: TOPICS IN RENDERING

Problem and Background

Problem Statement

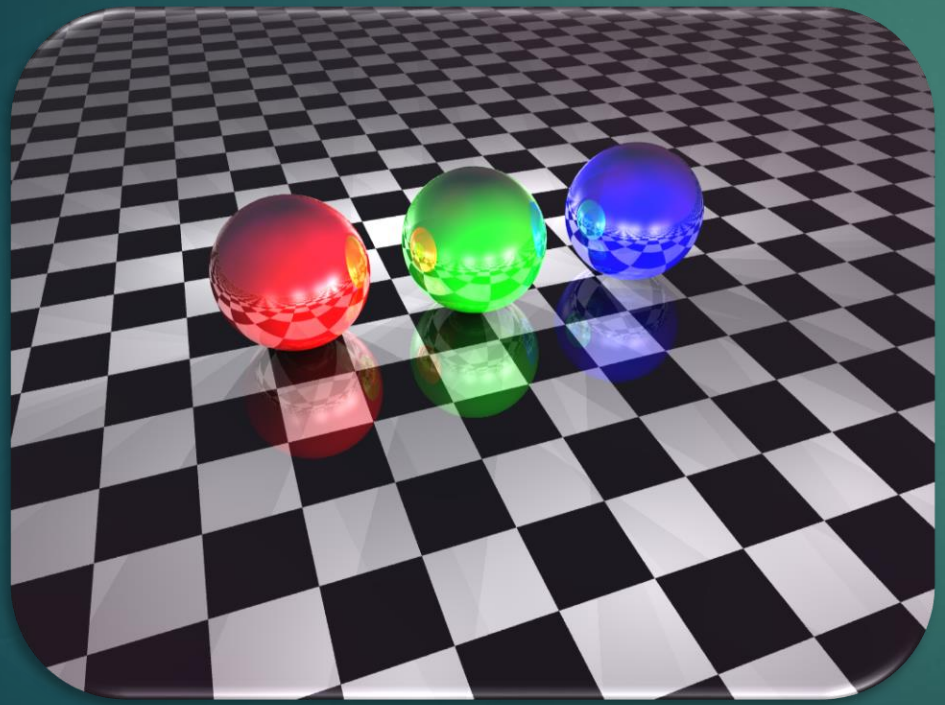
- The Phong Illumination Model and ray tracing engine explored thus far are flexible in the kinds of illumination they can simulate, but are only part of a larger formula for producing photorealistic ray traced images
- Part of the reason that basic ray tracing looks fake because of "jagged edges, hard 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])

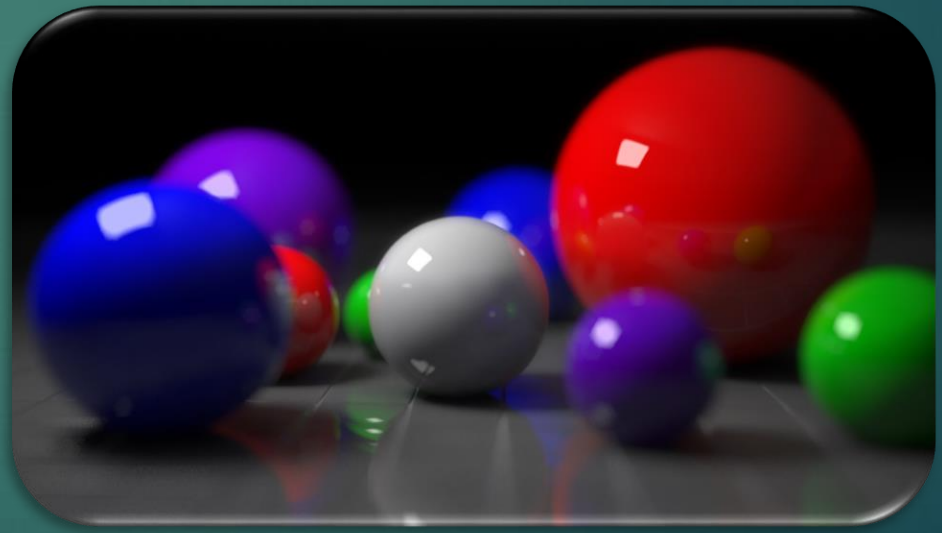
Camera Focus Comparison

Standard 3D Camera
(perfect focus)



http://www.cise.ufl.edu/~jlpaez/imgs/headerImg.png

Augmented 3D Camera
(Depth of Field)



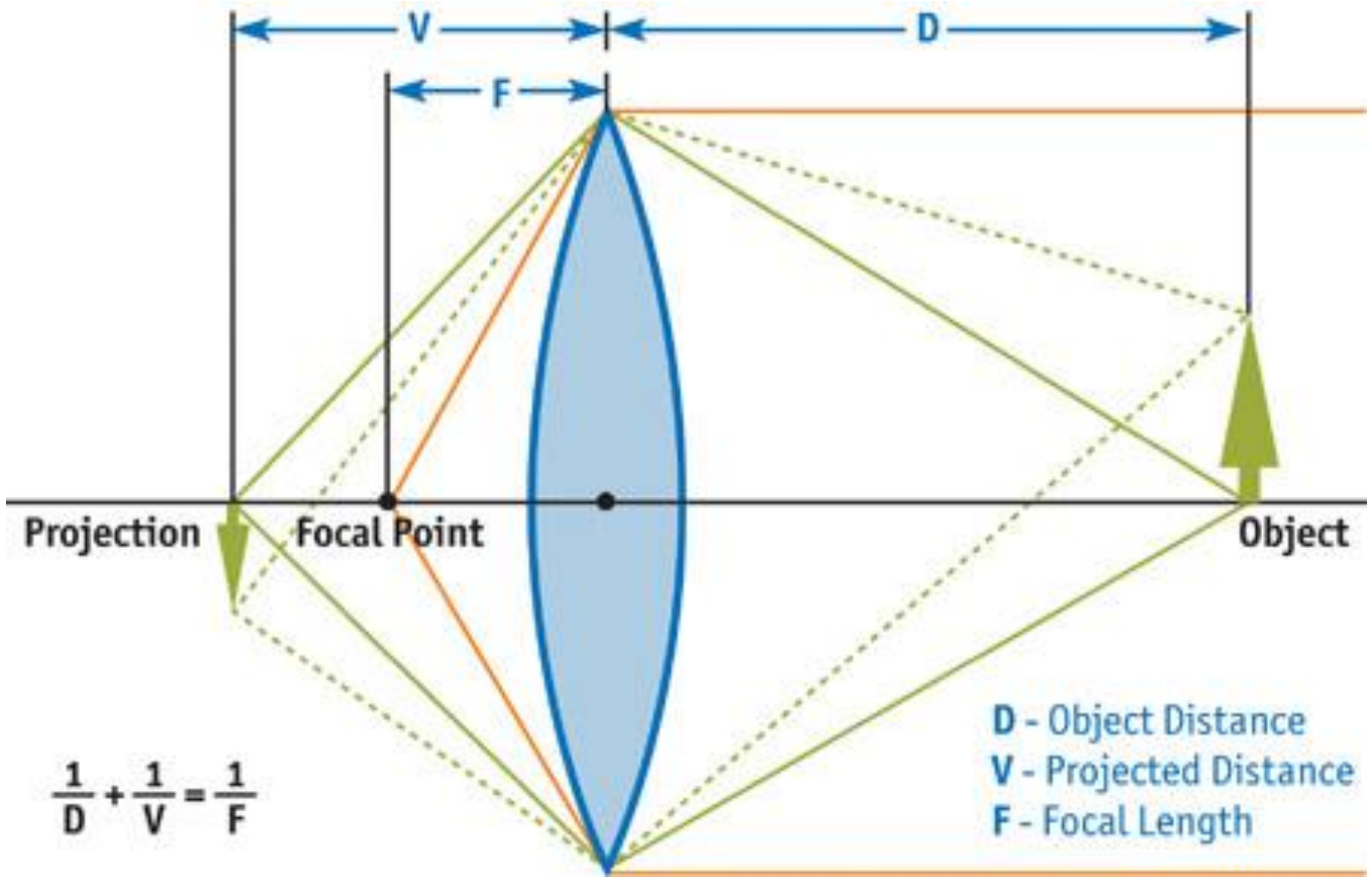
http://pichost.me/1295404/

Depth of Field and the Circle of Confusion (CoC) (Demers, 2004) [6]

- "Depth of field is the effect in which objects within some range of distances in a scene appear in focus and objects nearer or farther than this range appear out of focus"
- Depth of field "arises from the physical nature of lenses"
- In order for light to converge on a single point on the image plane (or retina for our eyes), "its source needs to be a certain distance away from the lens"
  - The plane at this distance is called the *plane in focus*; anything not at this exact distance is projected to a region on the film instead of a point
  - "This region is known as the Circle of Confusion (CoC)"

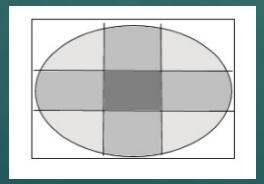
Single Lens Model (Demers, 2004) [6]

- Thin Lens

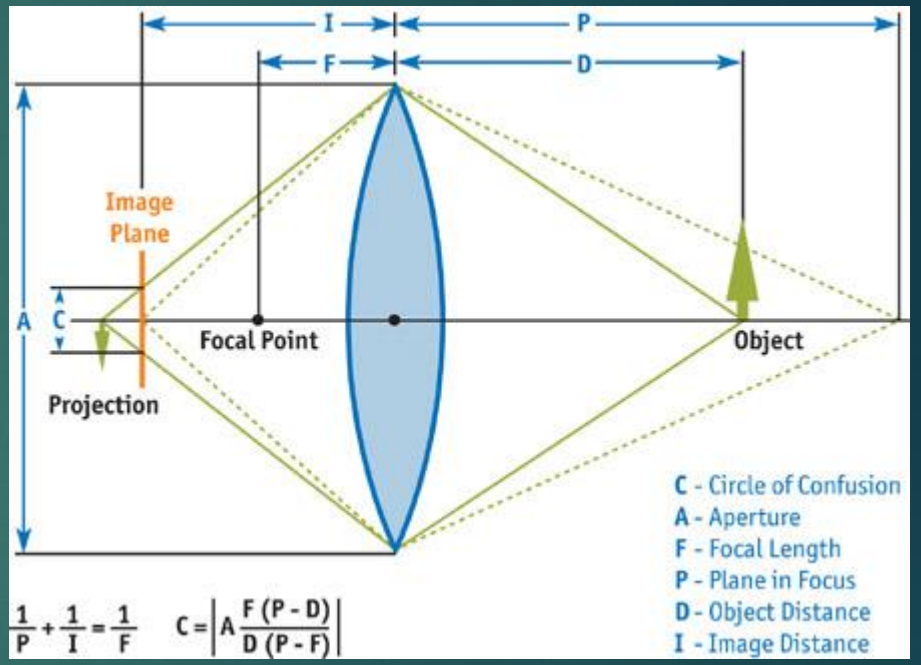


Calculating the Circle of Confusion (Demers, 2004) [6]

- $CoC = abs\left(aperture * \frac{focallength * (objectdistance - planeinfocus)}{objectdistance * (planeinfocus - focallength)}\right)$
- Object distance is already provided by current ray tracer with the Depth Map (pixelDepth variable)
- "The diameter of the CoC increases with the size of the lens and the distance from the plane in focus"
- When the CoC is smaller than the size of a pixel, that pixel is in focus while a CoC larger than a pixel will be out of focus and cause blurring



Yu, 2004 [10]

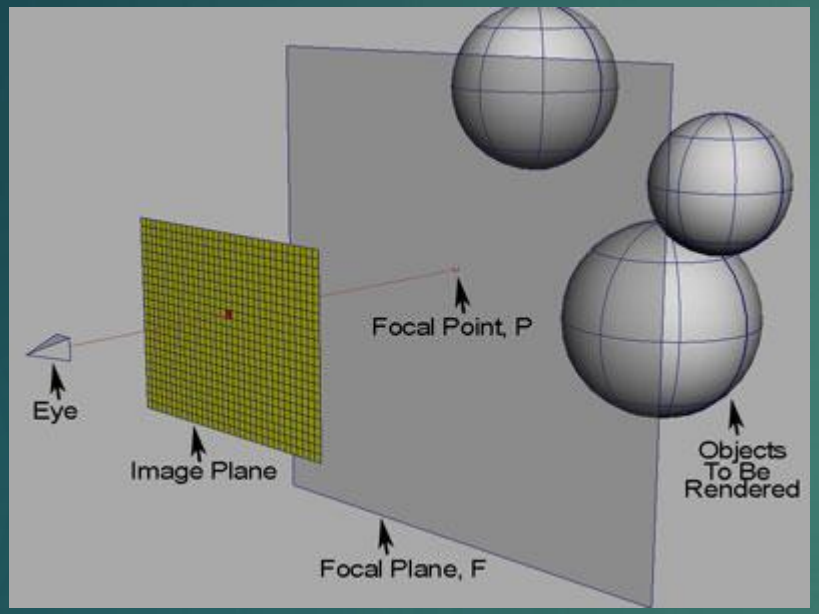


Demers, 2004 [6]

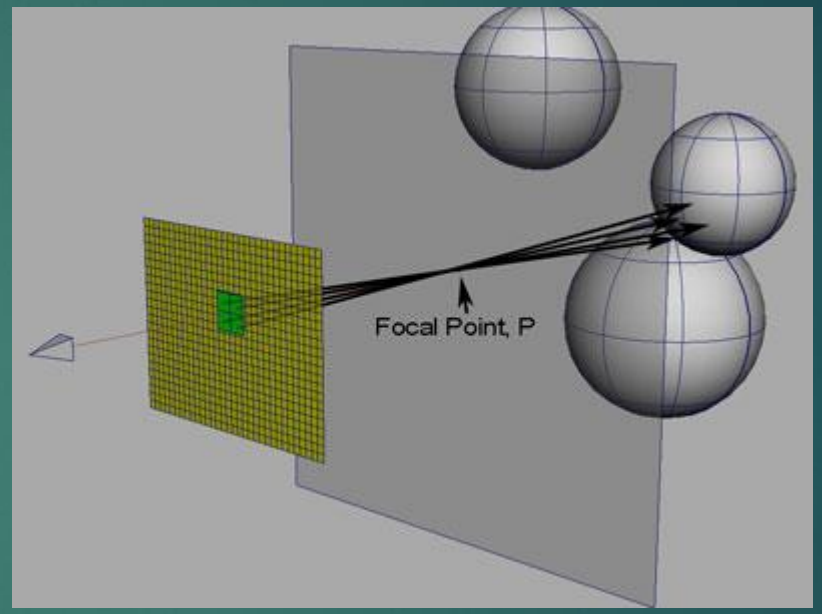
Solution Research

MULTIPLE RESEARCHED SOLUTIONS CAN BE GENERALIZED INTO THREE CATEGORIES

Approach 1 – Shoot additional rays at runtime and average samples (Lee, 2007) [4]



Lee, 2007 [4]



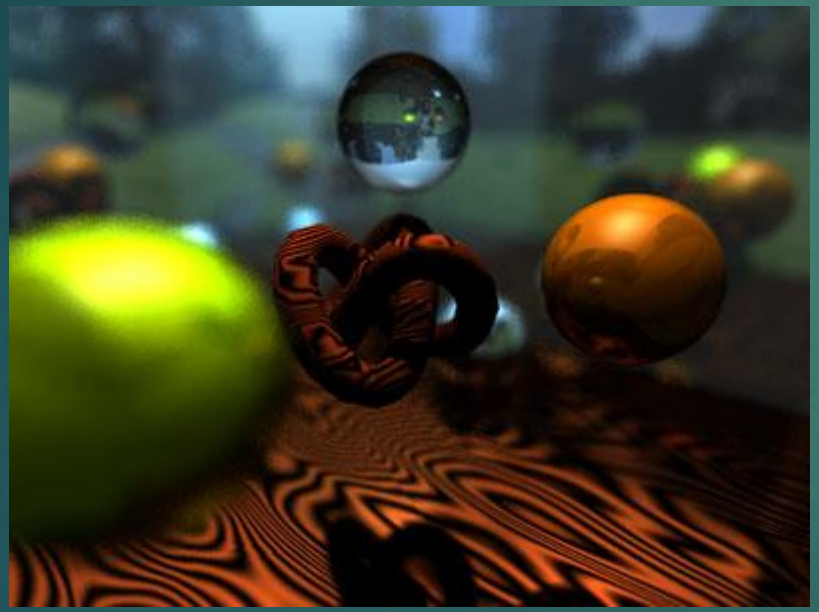
Lee, 2007 [4]

- Aperture is square bracket around each pixel
- DoF rays originate from each sampled grid point, not from the eye
- "d = perpendicular distance from eye to image plane,
  d' = distance from the eye to the pixel in question,
  e = eye position, v is unit vector from eye to current pixel,
  f = focal length"

$$P = e + \frac{d'}{d\,/(d+f)}v$$

Lee, 2007 [4]

Approach 1 Analysis



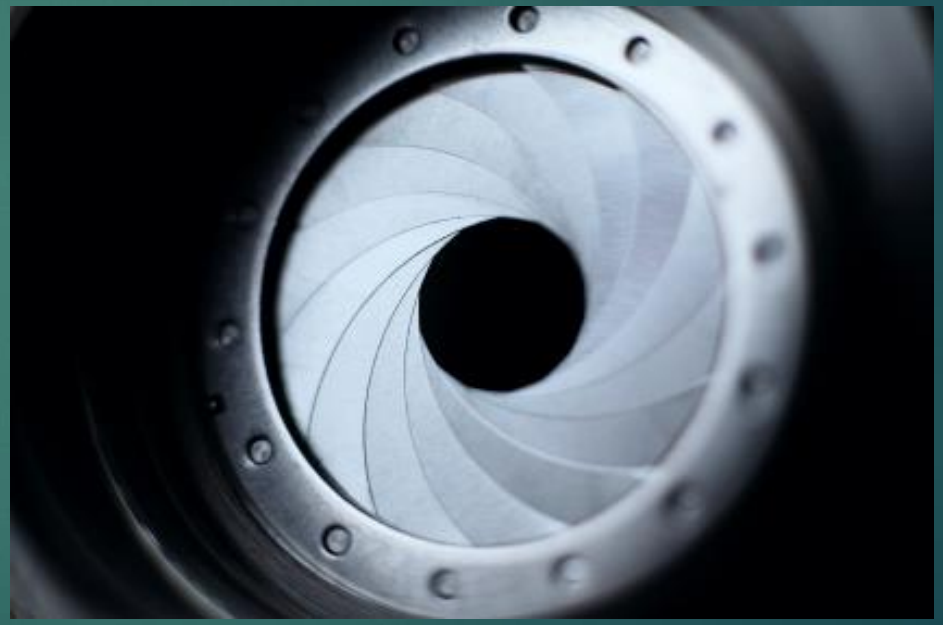
- Pros
  - Implementation logic is less complex (extending ray tracing procedure to simply calculate point P and shoot more rays from it)
  - With random sampling, results are smooth (see left)
  - Higher accuracy - takes reflection and refraction into account
- Cons
  - Slower performance

Approach 2 – Physically accurate DoF via multiple renders (IneQuation, 2012) [3]

- Default ray tracing uses a perfect pinhole model because its camera does not have an aperture aspect
- "Real cameras or our eyes have variable-sized apertures, which are physically the equivalent of taking a pinhole camera, taking a lot of pictures within that aperture, and averaging them" (IneQuation, 2012)
- Approach is to "rotate camera slightly multiple times around focus point, render the entire scene, accumulate the output color in a buffer and divide all values by the number of renders" (IneQuation, 2012)

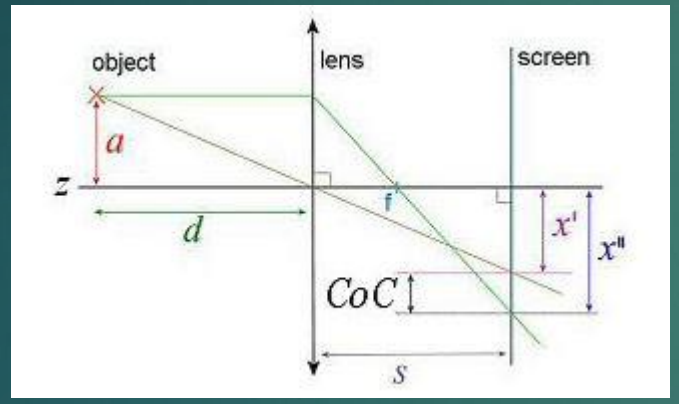
Approach 2 Analysis

- Pros
  - Simulates a real camera lens most accurately in terms of physical -> virtual representation
- Cons
  - Most expensive technique researched by far, requiring multiple renders of the scene and processing of each render to blend together into a final image



Approach 3 – Use Z-Buffer (depth information) to post-process DoF (Yu, 2004) [10]

Slightly different camera model and simplified CoC calculation

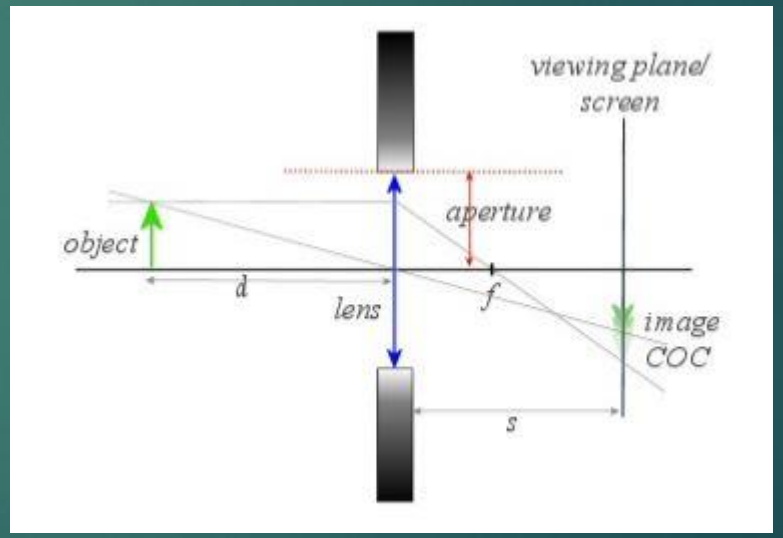


(Yu, 2004) [10]

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

(Yu, 2004) [10]

Simpler camera model with single aperture coefficient



(Yu, 2004) [10]

Approach 3 Analysis

- Pros
  - Object-space depth values are already provided by existing ray tracer framework's Z-Buffer
  - Fastest performance (not re-rendering scene multiple times or shooting additional rays)
- Cons
  - Reflected images "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" (Yu, 2004) [10]
    - Our camera and ray tracer share this same behavior as the OpenGL model discussed in approach 3
  - Only simulates a perfect lens and does not allow for bokeh (Japanese word for quality of out-of-focus areas) (Yu, 2004) [10]

Choice of Solution and Implementation Strategy

AND THE WINNER IS…

Implement Approach 3

- GUI
  - Replace depth map image in bottom right with DoF image
  - Add controls for camera focal distance and aperture
  - Add "Re-render" button to re-render image after configuring controls
- RTCamera
  - Change "look at point" variable to user-configurable (focal distance)
  - Add aperture variable w/getters and setters
  - Extend to include RTRectangle that represents the lens (similar to RTRectangle defined in the ShadowDepthMap of the spot light)
- RTCore: ComputeImage()
  - Add UniformDistribution() method to apply DoF blurring
  - Modify current image generation code to include calculating CoC and applying UniformDistribution()

Approach 3 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]

Risk Evaluation

Potential problems with our implementation strategy

- Performance
  - If resolution is high, rendering will be slower
  - Algorithm is "software-based and depends on processor speed and memory bandwidth" (Yu, 2004) [10], not taking advantage of the GPU
  - Render time will be slightly slower than original image render time due to additional calculation of CoC for each pixel
- Usability
  - To obtain new rendered images after adjusting focal distance and aperture controls, user has to click "Re-render" button each time
- Accuracy
  - As described by algorithm's author, Tin-Tin Yu, "reflected objects aren't rendered with accurate depth of field because the algorithm only takes into account the depth of the mirror's surface and not the depth of the reflected objects" (Yu, 2004) [10]

Questions?

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