Assignment 1

Arian Etemadi

April 23, 2024

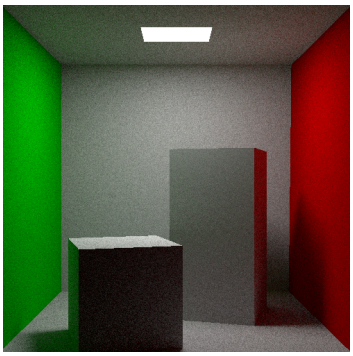
1 Tasks Implemented

I implemented the following (44 points in total):

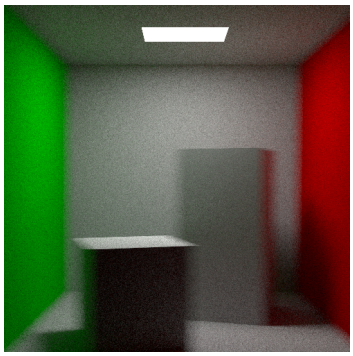
- Completing Wienori's MC Intestines (1 Point)
- Ambient occlusion (2 Points)
- Direct lighting, Hemisphere sampling (4 Points)
- Path tracing (11 points)
  - Start (4 points)
  - BRDF/BSDF (2 points)
  - Russian Roulette (1 point): I implemented the version with the fixed continuation probability
  - In a loop (4 points): both recursive and iterative versions of the path tracer are in the same file: `path_tracer_recursive.cpp`, in functions `Li_recursive` and `Li_iterative`. The choice can be specified in the scene file.
- Motion blur (9 points)
- Depth of Field (6 points)
- Standard Deviation and Adaptive Sampling (9 points)
  - Standard Deviation (3 points)
  - Adaptive Sampling (3 points)
  - Why it is biased and what to do (3 points)
- Be Patient (2 points)

2 Bonus Tasks

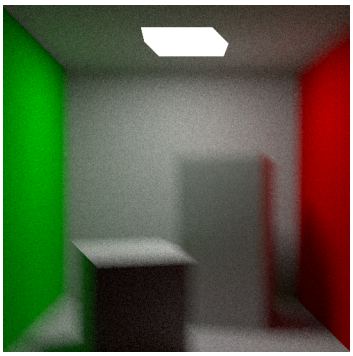
Corresponding scene files are available in `/scenes/assignment1/`.



(a) No motion blur



(b) Horizontal movement



(c) Diagonal movement

Figure 1: Motion blur effect on the whole scene by moving the camera. Scenes are path traced with sample counts set to 4096 for all three cases.

2.1 Motion Blur

I implemented motion blur via jittering the camera. In particular, the user can specify a `vector` named `motionBlurVelocity` for the velocity of the camera in the `.xml` scene file. According to this velocity, the camera is moved a little bit for a uniformly random time before sampling each ray. This movement is implemented in a function `animate(time)` in the `Perspective` class. The results can be seen in Fig. 1.

2.2 Depth of Field

Implemented depth of field. For each ray, I first computed the intersection with the focal plane, noticing that the ray that goes through exactly the middle of the lens does not refract. Then, a random point on the lens is sampled and the new ray connects this sample point to the intersection point on the focal plane.

There are two parameters: focal length and aperture. Fig. 2 compares the effect of focal length. Fig. 3 compares the effect of aperture.

2.3 Standard Deviation and Adaptive Sampling

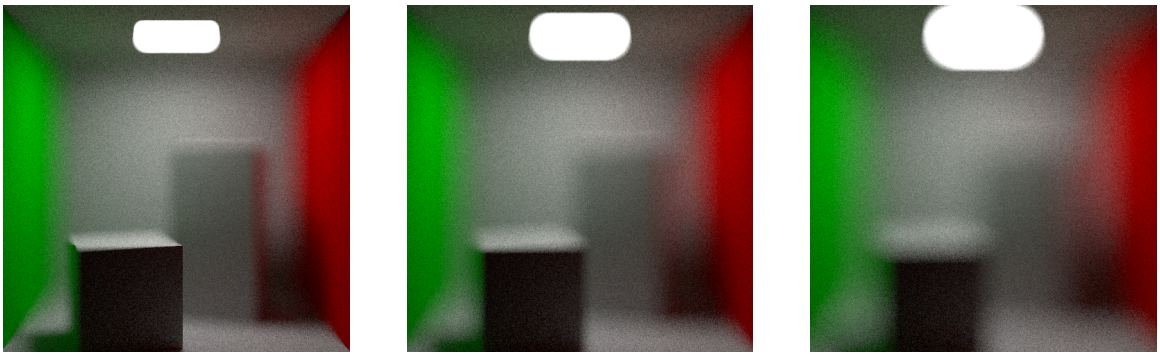
2.3.1 Standard Deviation

Implemented. The result for 2048 samples can be seen in Fig. 4.

2.3.2 Adaptive Sampling

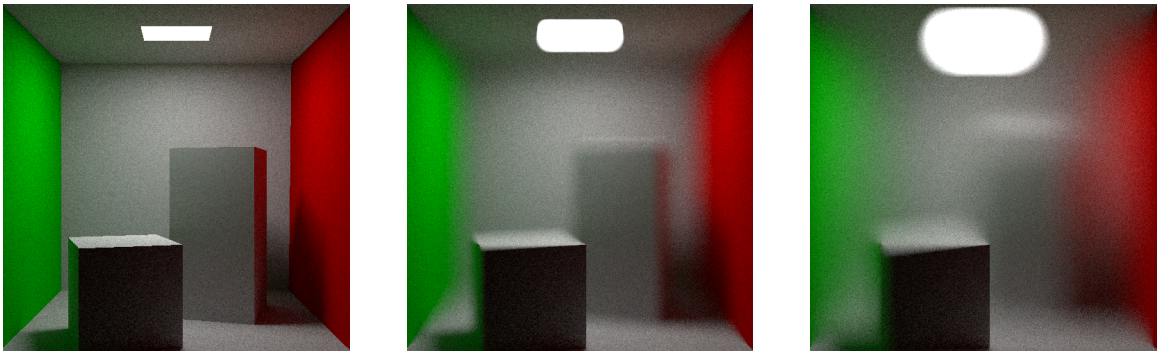
I implemented a naive version of the adaptive sampling. First, I sample 100 rays for each pixel, and calculate its standard deviation. Then, according to the standard deviation, the actual sampling is performed. Pixels with higher values of standard deviation are sampled more to reduce noise. As we see in the next section, this is biased. One such render with adaptive sampling can be seen in Fig. 5.

Note that I had to remove the implementation from the latest commit. The implementation for the adaptive sampling can be found in the commit with the comment `Adaptive sampling implemented`.



(a) Focal length=880 (b) Focal length=780 (c) Focal length=680

Figure 2: Decreasing the focal length pushes everything to the background. Aperture is set to 100 in all three.



(a) Aperture=0 (b) Aperture=100 (c) Aperture=300

Figure 3: Increasing the aperture leads to a very narrow depth of field. If aperture is set to 0, then there is practically no depth of field. Focal length is set to 880 in all three. Note how in the figure on the right, part of the front cube is still completely in focus; that is where the focal plane passes.

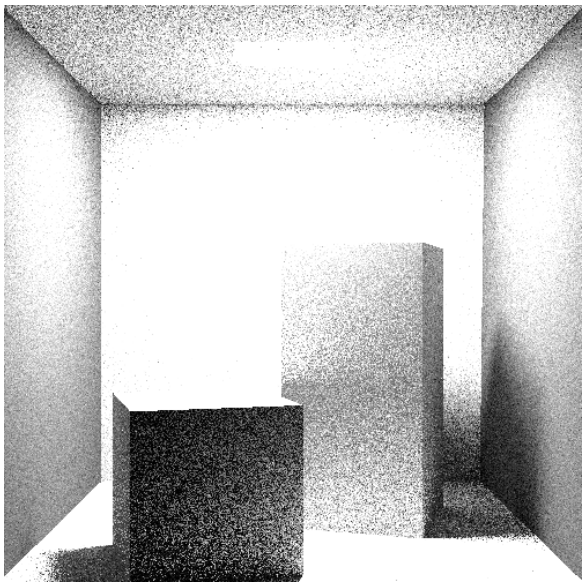


Figure 4: Standard deviation of the average RGB value for sample count=2048. Black indicates low and white indicates high value for the standard deviation.

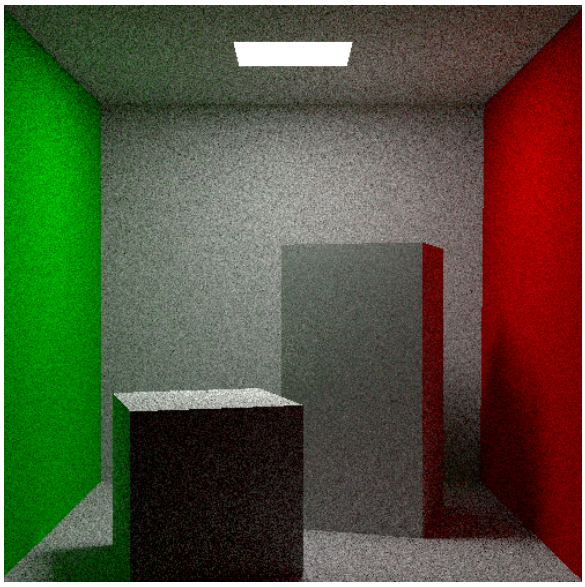


Figure 5: Adaptive sampling.

2.3.3 Why it is biased?

If a pixel is in the shadow and the value is very close to black, there will be a long time before any non-zero value is sampled. Until that happens, the estimated standard deviation of the value will be zero! If the adaptive sampling algorithm ever gets satisfied with pixels that have zero standard deviation and stops allocating samples to those pixels, those pixels will never get out of black. They will be completely black, as opposed to almost black, which makes this algorithm biased. The solution would be to always keep sampling every pixel with some non-zero rate. In this way, no matter how adaptive the sampling is, ultimately, at least in theory, non-zero values will also be encountered and the sampling will be unbiased.

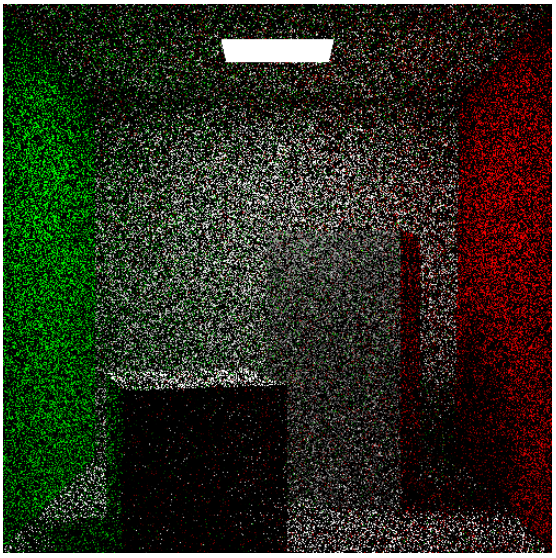
2.4 Be patient

Fig. 6 shows the effect of sample count on reducing noise. The one with 512 samples took almost 300 seconds to compute, and by increasing the sample count to 2048, it took a bit more than a 1000 seconds!

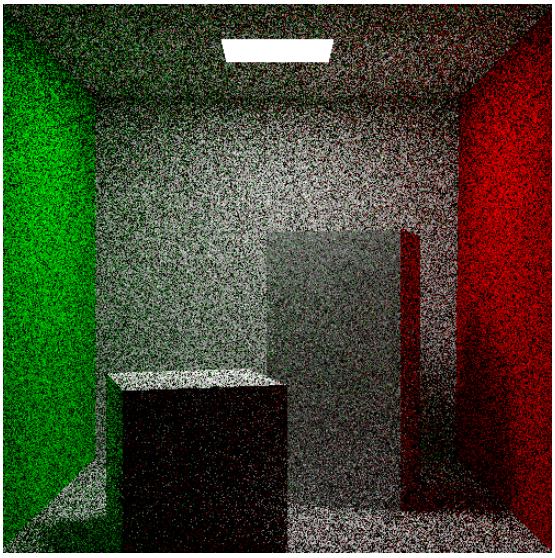
I would be happy with the result if the standard deviation of each pixel was $O(1)$, in the range of $[0, 255]$. With the rules of probability and statistics it can be shown that the standard deviation of samples will reduce by $O(\sqrt{N})$ where N is the number of samples. Hence, if we wanted the standard deviation of the last one (with 2048 samples) to improve 10 times, we would have to multiply the sample count by 100!

This would bring the computing time to 10^5 seconds, which is more than a day for a single frame!

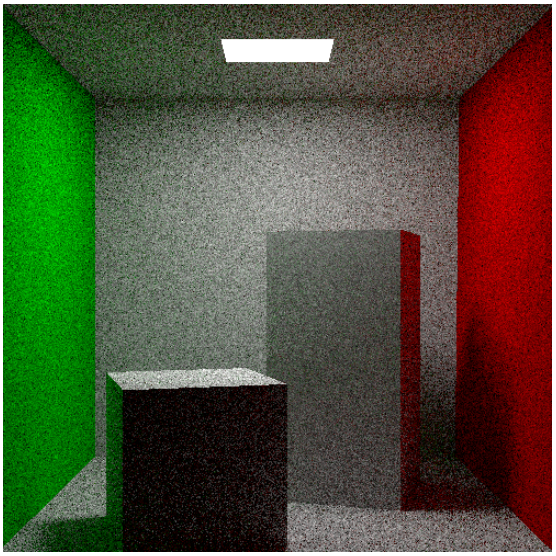
No, we have to improve the efficiency of our algorithm so much if we wanted to render whole movies with it.



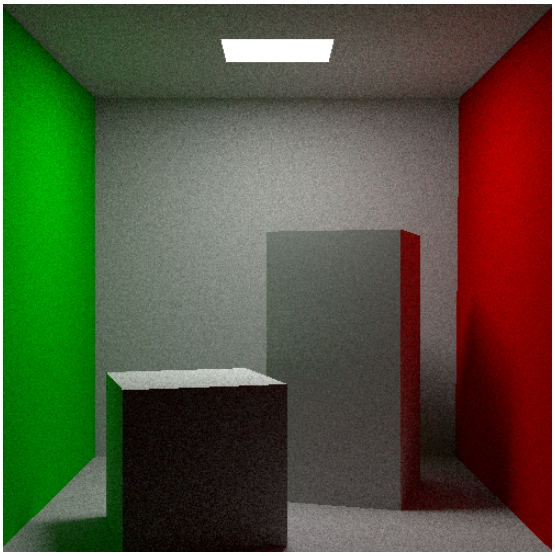
(a) Sample count = 32



(b) Sample count = 128



(c) Sample count = 512



(d) Sample count = 2048

Figure 6: Noise very slowly reduces by increasing the sample count.