Minimizing Rendering Noise

Team Members: Aniket Bhatia and Alex Evelson

Contents:

Introduction:

Noise is one of the most important aspects in an image, besides the actual information in the image, which can affect the visual experience of viewing the image. It is therefore crucial to address the issue of reducing noise.

There are many ways in which noise gets introduced into the images ranging from the inherent distribution in photon arrival and sensor imperfections to retrieval after compression algorithms. In this project we are mainly interested in analyzing the noise present in the camera sensor image and how it varies with different parameters. We employ PBRT to create 3D scenes, trace rays through the scenes to produce a sensor irradiance image, and finally use ISETCam to convert the sensor irradiance image into a camera sensor output. We compare the noise obtained in the camera sensor output with the idealized scenario where we have only photon noise. This helps us realize how far are we from the best-case scenario, since photon noise is basically a given in any camera pipeline. We want to ensure that the trends we see are consistent across the types of scenes, so we use both a flat surface and classic Cornell box scenes for our simulations.

In the upcoming sections, we first discuss some background concepts and then move on to describing the parameters we chose to vary and their qualitative impact. Subsequently we discuss the results we obtained from our simulation-based approach and the takeaways.

Background:

Parameters for Denoising:

Many denoising techniques have been proposed in the past which find themselves in the post-processing part in the imaging pipeline. Some of these techniques include spatial and frequency domain filtering of the image. Currently, there has also been huge advances in employing neural network approaches to remove noise.

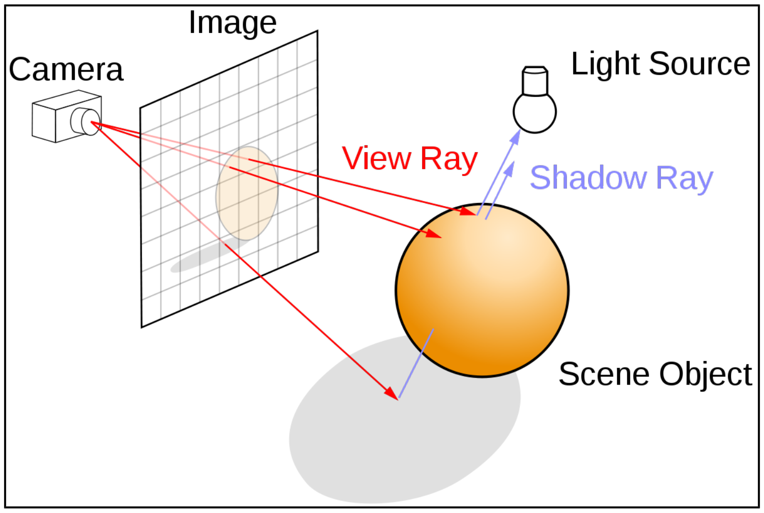
In this project our focus is not in the post-processing section, instead, we choose to vary the following parameters:

1. Rays per Pixel
2. Camera Lens
3. Exposure Time
4. Sampler
5. Film Diagonal
6. Film Resolution
7. Pixel Size
8. Fill Factor
9. Number of Bounces

Ray Tracing:

Ray Tracing is a part of the rendering process and, therefore, it makes sense to briefly describe the process.

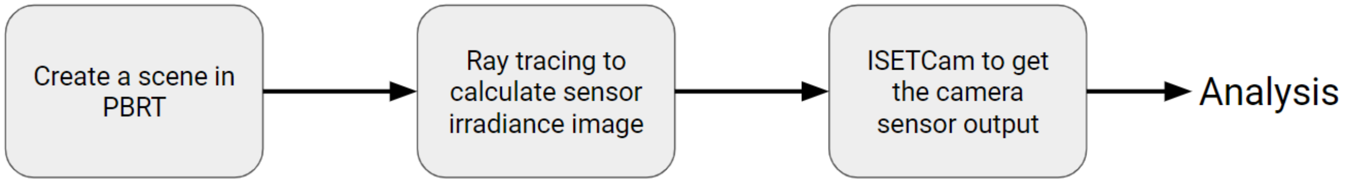
As the name suggests, ray tracing involves following the path of the light as it interacts with all the different objects in the scene. Each ray is tested for how it intersects with a subset of objects in the scene (which has been mathematically described in the program). The tracing is actually done by sending the rays backwards, i.e., from an imaginary eye through each pixel, instead of the rays going into it (as it would normally happen in real life). This clever modification is done to make this process computationally efficient.



Source [3]

Experimental Setup:

We first use PBRT to create a 3D scene (with physically meaningful units). Then trace rays through the scene to produce a sensor irradiance image (here the image would be specified in the units of irradiance, i.e., photons/nm/sec/m^2). Then ISETCam is used to convert the sensor irradiance image into a camera sensor output (which is now specified in the units of electrons/pixel). The following chart summarizes this flow:


Create a scene in PBRT
Ray tracing to calculate sensor irradiance image
ISETCam to get the camera sensor output
Analysis

To isolate the impact of each of the parameters mentioned in the above section, we vary one of them and keep the others constant. The default value of each of the parameters in our code (when that particular parameter is not being varied) is as follows:

1. Rays per Pixel = 512
2. Camera Lens = Double Gauss (22.5 deg, 12.5mm)
3. Exposure Time = 0.001s
4. Sampler = Halton
5. Film Diagonal = 0.5mm
6. Film Resolution = 128x32
7. Pixel Size = 1.2e-6m
8. Fill Factor = 1
9. Number of Bounces = 0 (flat surface); 3 (Cornell Box)

We do the above for two scenes: a flat surface and a Cornell Box. In the case of a Cornell Box, we always sample from the same spot in the scene, an illuminated point on the back wall, which we then show to make the reader aware of any inherent distribution of objects.

We compare the overall noise level in the image with the noise just due to the Poisson distribution of the arriving photons and use this as a metric. We later propose a cost-benefit metric which is kept for future experimentation.

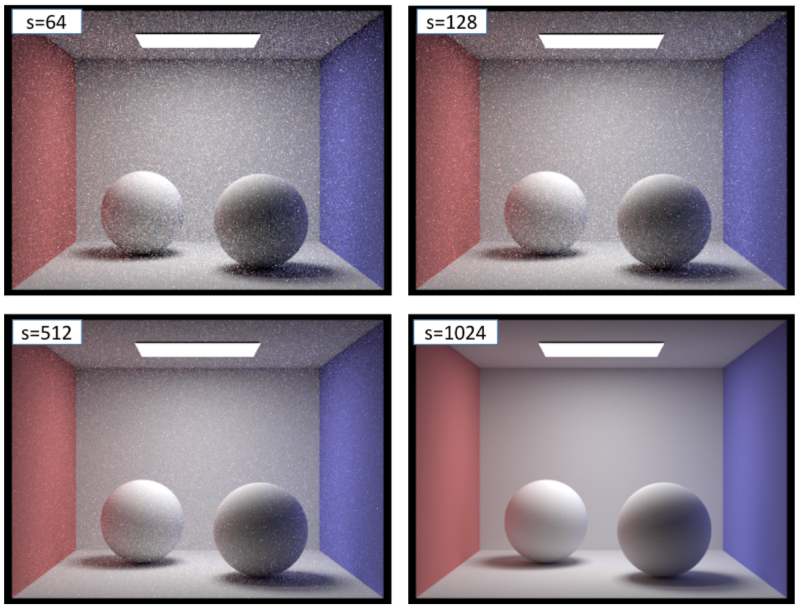
Results and Explanations:

Now, we go over each of the parameters and discuss the effects of varying them:

Rays per pixel:

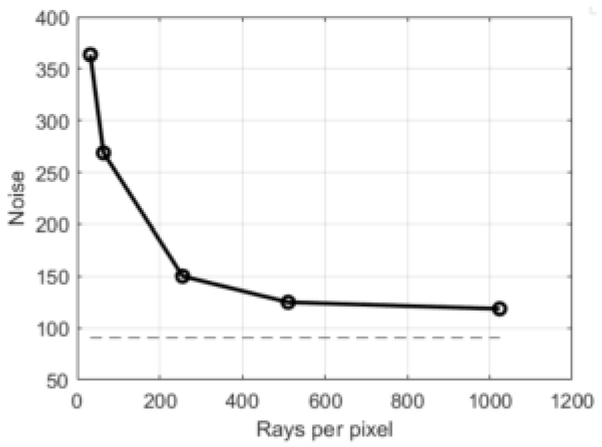
The number of rays per pixel dictates how well we can approximate the scene. The more rays available, the better the approximation. This in turn leads to more rays per pixel leading to a lower noise. This is in line with what we would expect from intuition.

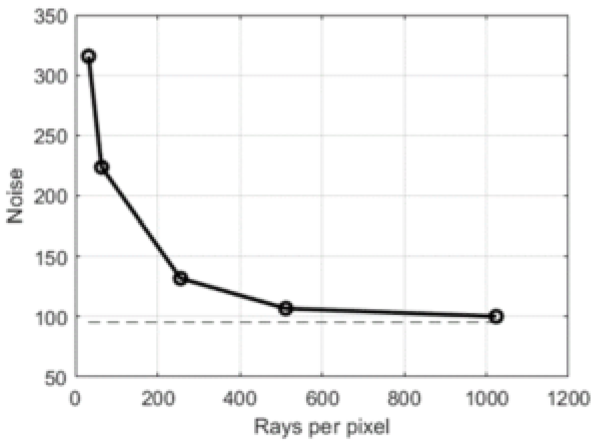
The following figure shows the effect of increasing the rays per pixel on the noise level in a scene, which we also confirm through our experiments.



Source [4]

The results of our experiments are:

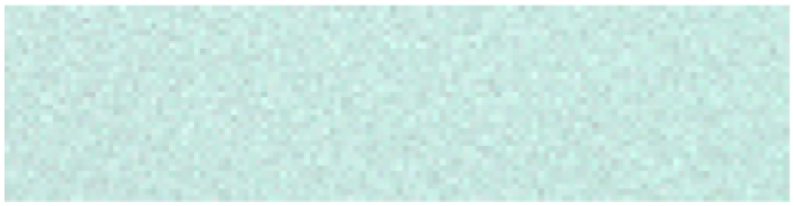




Flat Surface Cornell Box

The images that we saw in the Cornell Box for each of the rays per pixel are as follows:

32 Rays per Pixel:



64 Rays per Pixel:

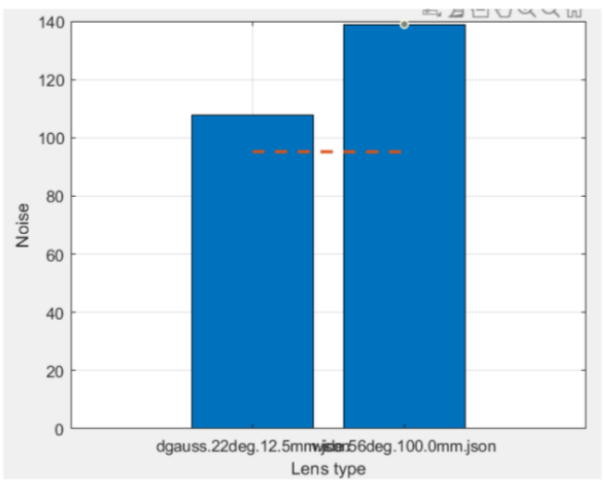
256 Rays per Pixel:

512 Rays per Pixel:

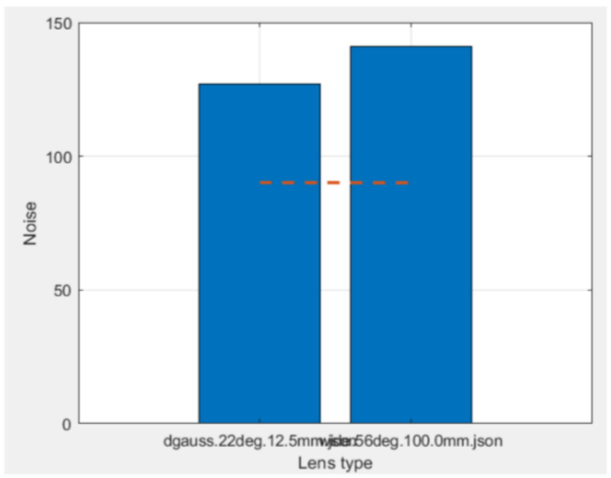
1024 Rays per Pixel:

<u>Camera Lens</u>:

We use two different lenses for our simulation. Our default lens (Double Gauss at 22.5 deg and 12.5mm) is contained in dgauss.22deg.12.5mm.json, but we also have a wider lens (56 deg and 100mm) in wide.56deg.100.0mm.json. When both are pointed at the same point in the scene, the wider lens is able to absorb light from a larger area, increasing the expected noise we obtain in the final image.



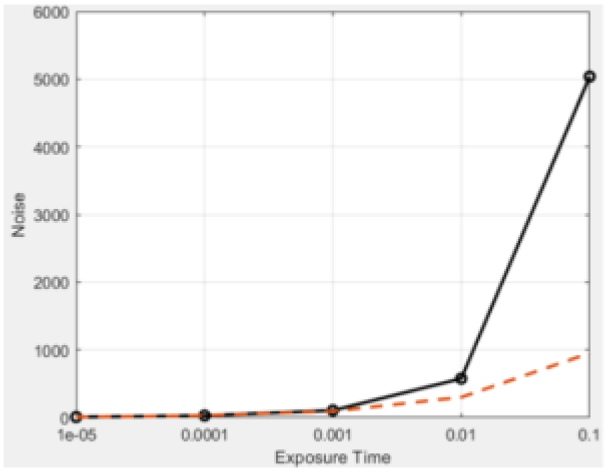
Flat Surface

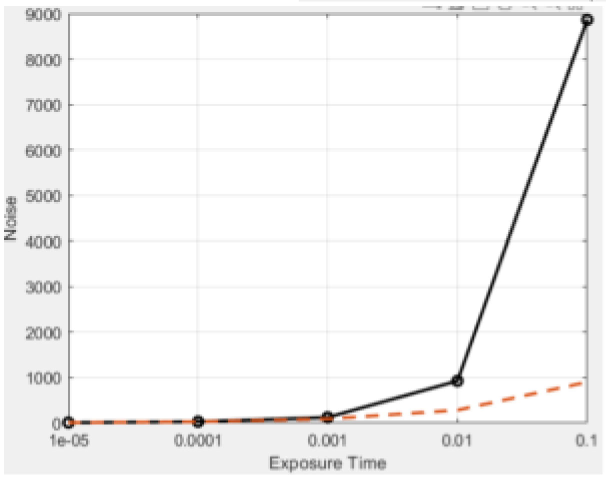


Cornell Box

Exposure Time:

This is the time the camera sensor is exposed to light. Therefore, higher the exposure time, higher the number of photons that fall onto the sensor, leading to higher noise levels. This also increases the baseline photon noise. But of course, the total rendering noise stays higher than the baseline photon noise and, in fact, gets relatively worse compared to it if we look at the difference. It instead scales alongside the baseline noise so that the ratio between them remains more consistent.





Flat Surface Cornell Box

The images that we saw in the Cornell Box while varying the exposure time are as follows:

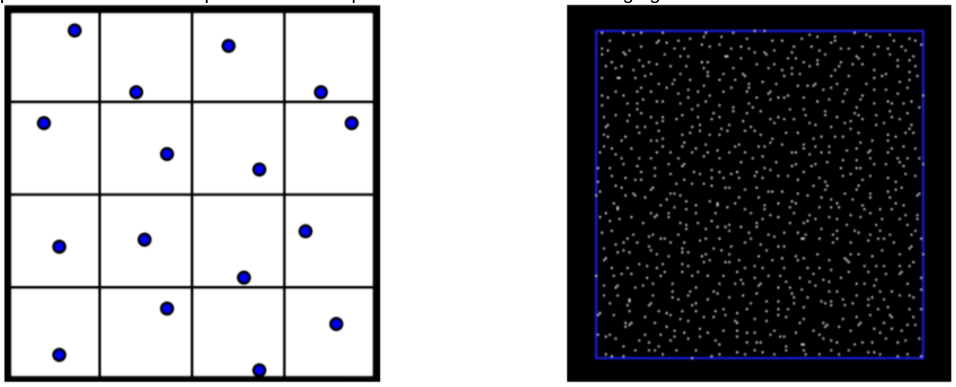
1e-5 sec:

0.1 sec:

<u>Sampler</u>:

We realize that although the final output is a grid of discrete values, the incident radiance is actually a continuous function. There are many ways in which sampling can be done, and the type of sampling chosen will most likely have an impact on the overall noise performance. We chose three samplers- Stratified, Halton and Sobol- and ran our simulations with each one of them. It is best to dive a little deeper into these sampling techniques to understand the results that we obtained.

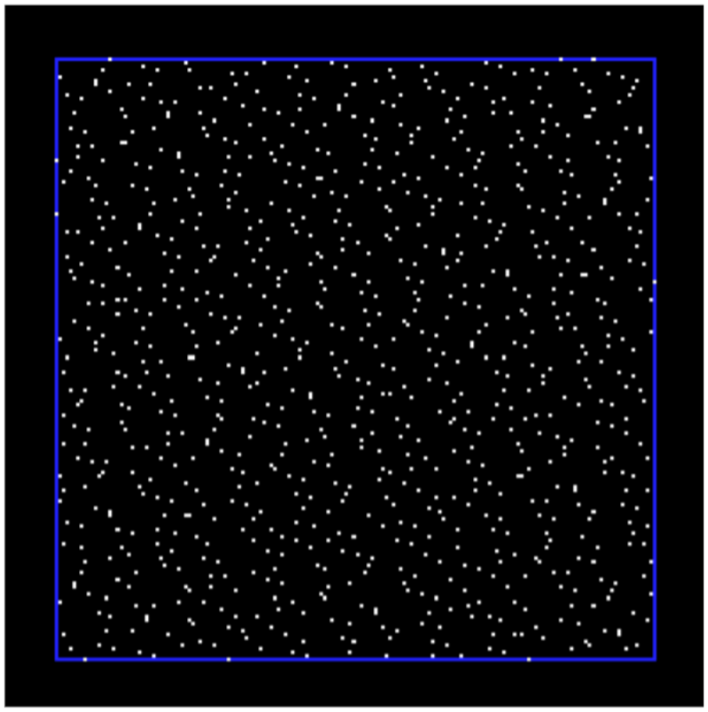
1. Stratified/Jitter Sampler: This sampling technique divides the entire grid into unit squares and places a random sample in each one of them. The random sample is placed by “jittering” the central location by some random amount and taking that as the sample in that unit square [1]. The drawback here is that we need to know the number of samples before we start to sample, because only then the grid can be divided into the required number of unit squares. This sampler is illustrated in the following figure:



Source [2]

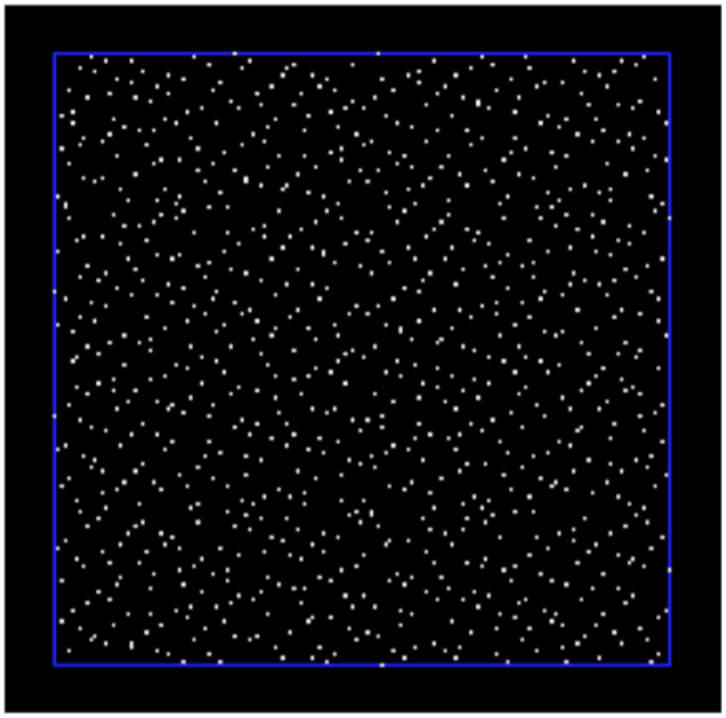
2. Halton Sampler: Halton Sampler is based on Halton Sequence, which is an example of quasi-random number generator. As given in [2]:
    a. For generating the ith number in the sequence hi, we write i in base b, where b is a preselected prime number (for example, i=17 in base b=3 is 122)
    b. The digits are then reversed with a decimal point being added at the beginning (for example 0.221, base 3)
    c. To generate an n dimensional vector, the same value of i is used but with a different prime number b for each dimension

We get the following sampling pattern after using the Halton Sampler:



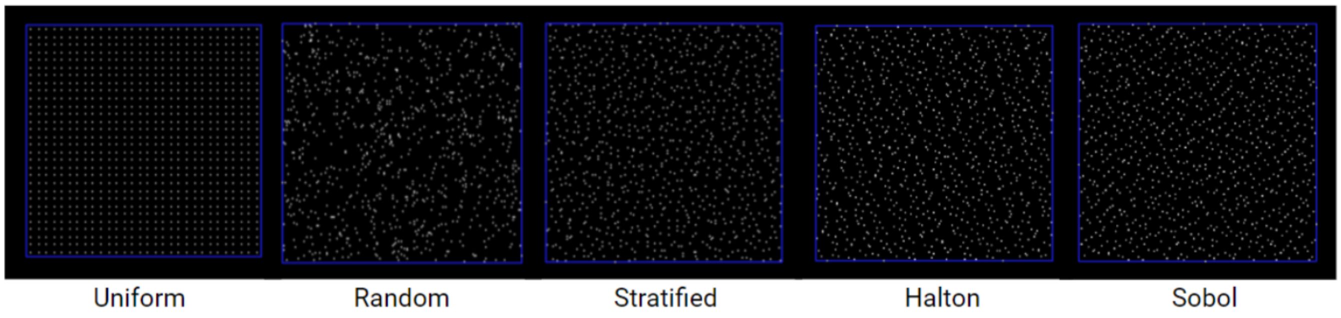
source [2]

3. Sobol Sampler: The Sobol sampler is based on Sobol Sequence, which is, again, a quasi-random number generator. The algorithm to generate this is too complex to discuss here. Sobol sampler tends to work better for rendering applications due some inherent pattern in the Halton sampler. We get the following sampling pattern after using the Sobol sampler:



source [2]

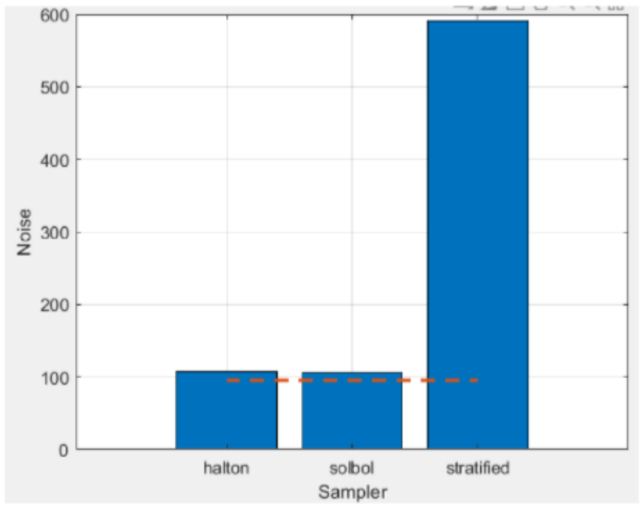
The following figure shows how all of them compare together along with a uniform and a purely random sampling:



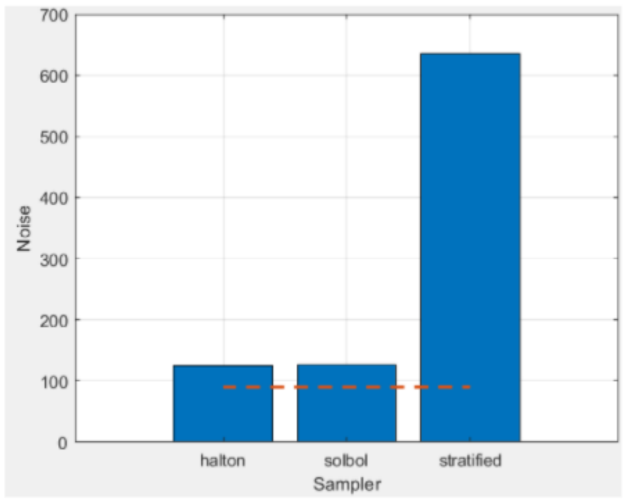
Source [2]

If the reader is interested in understanding these sampling techniques in detail, we suggest going through the sources [1] and [2], especially [1], which covers this with mathematical rigor.

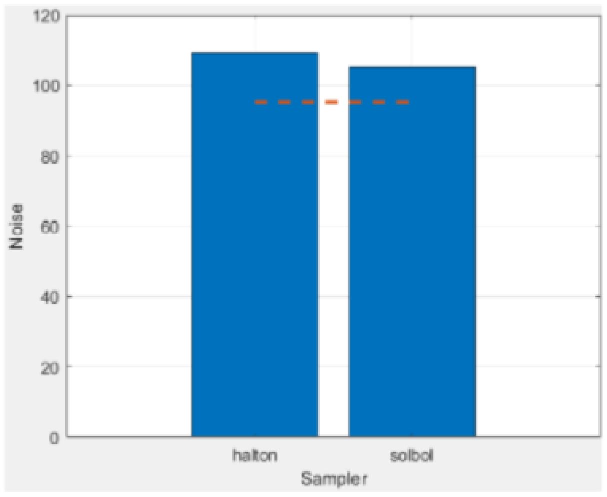
Looking at the sampling techniques, Sobol should perform slightly better than Halton, and both of them should perform better than Stratified, which is exactly what we see in our experimentation:



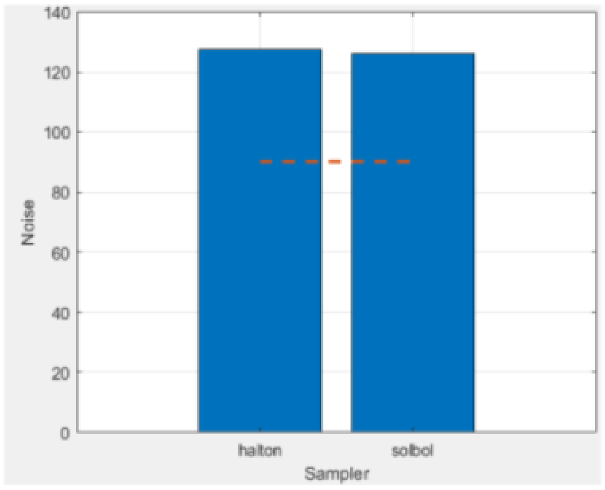
Flat Surface



Cornell Box



Flat Surface



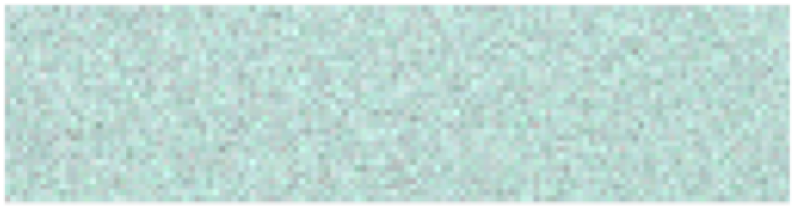
Cornell Box

The images that we see in the Cornell Box while varying the sampler are as follows:

Halton:

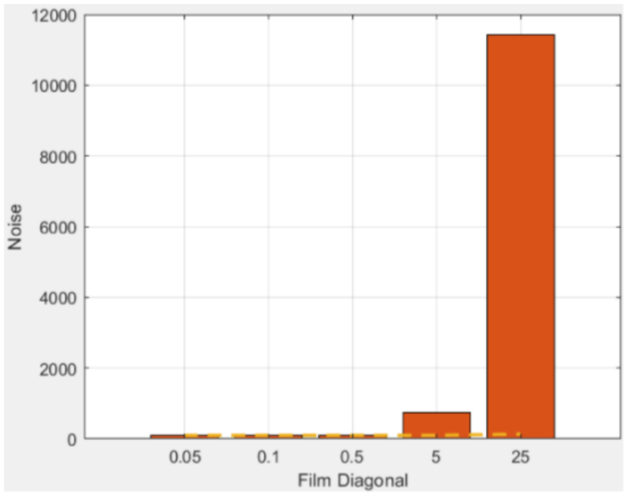
Sobol:

Stratified:

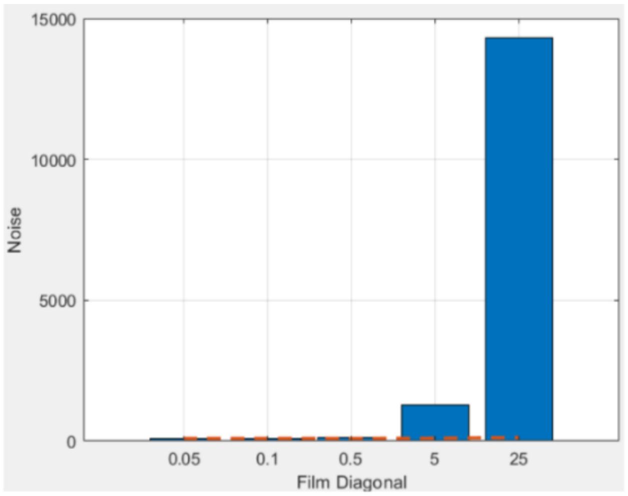


Film diagonal:

Film diagonal dictates the sensor size. Higher the film diagonal, bigger the sensor. As the senor gets bigger the scene which the sensor sees becomes wider (i.e. bigger the sensor the wider we see into the scene) leading to a higher noise owing to a higher non-uniformity.



Flat Surface



Cornell Box

The images that we see in the Cornell box are shown below. We see that as the film diagonal increases we start to look wider into the scene:

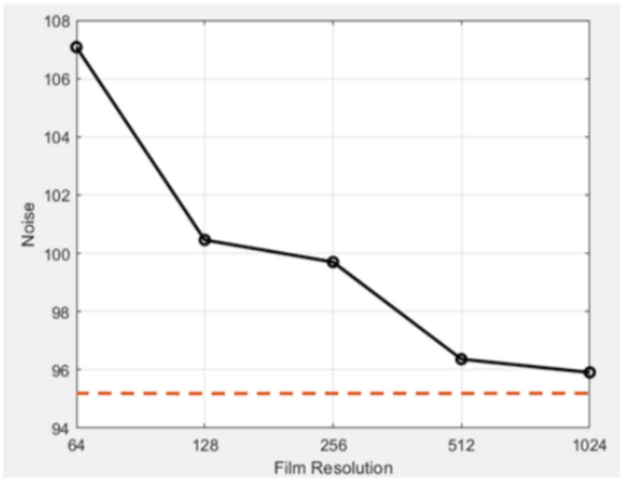
0.05 mm:

0.5 mm:

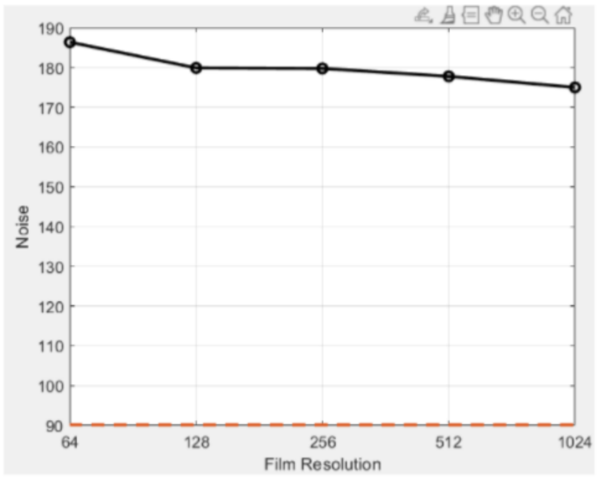
25 mm:

Film Resolution:

Increase the film resolution while keeping the size of the film the same is tantamount to decreasing the pixel size. So here we have smaller pixels but more of them, since the overall size of the film same, the overall noise tends to be lower due to the lower read noise in smaller pixels. Although, we would have more smaller pixels, the effect of a decrease in read noise in smaller pixels is usually higher than an increase in the number of these smaller pixels. Hence, the overall noise decreases with an increase in the film resolution. The simulation results that we obtained are as follows:



Flat Surface



Cornell Box

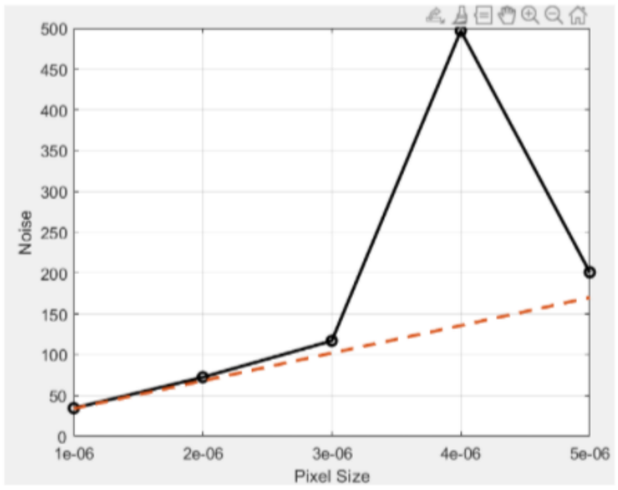
The images that we see in the Cornell Box while varying the film resolution are as follows:

64x64:

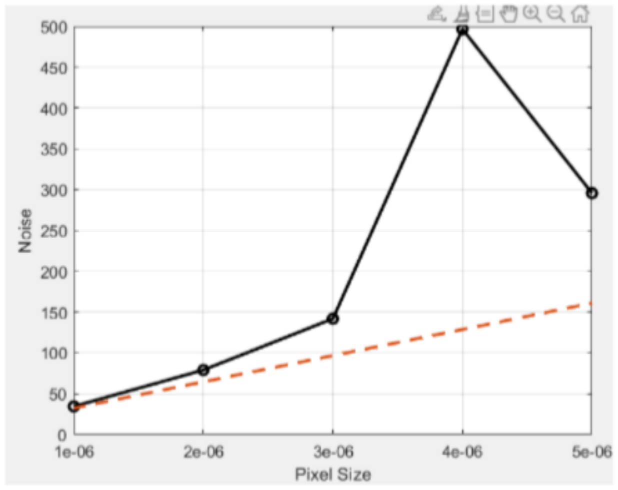
1024x1024:

Pixel Size:

Smaller pixels do not take in as much light, resulting in lower noise. We did have an unexplained outlier in our data, but we largely found the expected upward trend. Our simulation results are as follows:



Flat Surface



Cornell Box

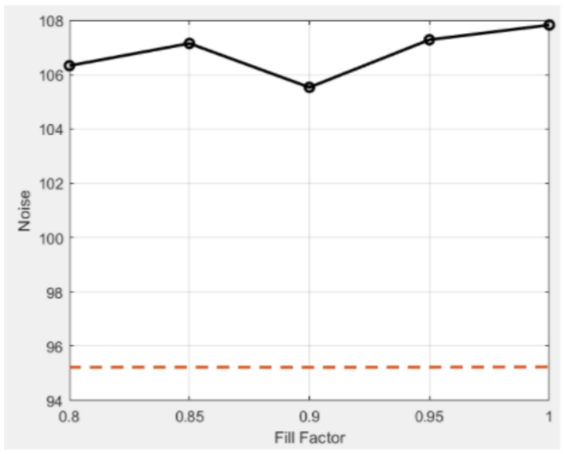
The images that we see in the Cornell Box for varying pixel size are as follows:

1e-6 m:

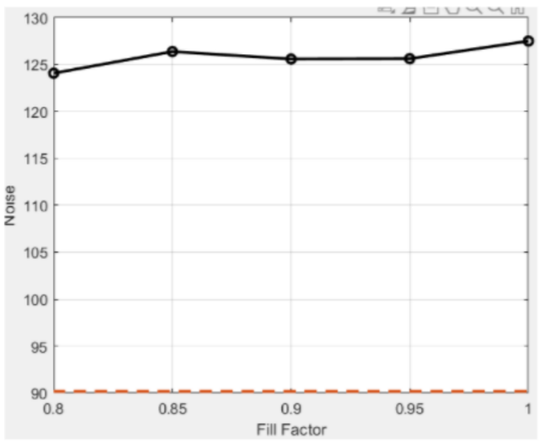
5e-6 m:

Fill Factor:

Increasing the fill factor does not seem to consistently change the noise per pixel. There's a slight upward trend though, suggesting that reducing the fill factor might reduce the noise. Our simulation results are as follows:



Flat Surface



Cornell Box

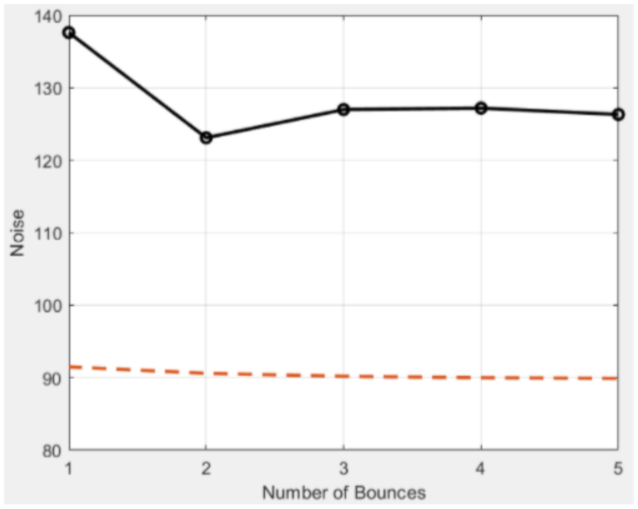
The images that we see in the Cornell Box while varying the fill factor are as follows:

0.8:

1:

Number of Bounces:

The number of bounces variable represents the number of surfaces a ray can bounce from. With the flat surface scene, there is no need to vary this variable, but for the Cornell Box scene, it does make a difference. Curiously, there is no uniformity in the trend, unlike with most variables. We do, however, observe a general reduction in rendering noise, so it's possible increasing the number of bounces is akin to increasing the number of rays. One concrete benefit is that it "blurs" the rendering a bit by reducing noise in areas that are not directly lit. The simulation result that we got after varying the number of bounces for the Cornell Box is:



The images that we saw while varying the number of bounces in the Cornell Box are as follows:

1:

5:

Conclusions:

Our findings suggest how design choices could affect rendering noise. Of course, that is far from the only metric designers can seek to optimize for, but taking this as the objective, we see the following decisions may be beneficial:

- Increasing the number of rays
- Using smaller lenses
- Reducing exposure time
- Using Sobol sampling
- Reducing film size
- Increasing film resolution
- Reducing pixel size
- Reducing the fill factor
- Increasing the number of bounces

Going forward, researchers could take into account how these factors affect other objectives and consider other metrics of success. They may, for example, use SNR as a benefit metric while looking at the cost of using too small a film size or too low of an exposure time. With this information, they can design an optimized end-to-end solution.

Acknowledgements:

We would like to thank David Cardinal, Joyce Farrell, Zheng Lyu, and Brian Wandell for their excellent instruction, assistance, and feedback leading up to and during the project.

References:

[1] https://pbr-book.org/3ed-2018/contents

[2] https://cseweb.ucsd.edu/classes/sp17/cse168-a/CSE168_07_Random.pdf

[3] https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

[4] https://ceciliavision.github.io/graphics/a3/#part4

[5] ISET3D - https://github.com/ISET/iset3d

[6] ISETLens - https://github.com/ISET/isetlens

[7] JSONIO - https://github.com/gllmflndn/JSONio

[8] ISETCam - https://github.com/ISET/isetcam

Appendix:

```
% Analyze how the rendering noise declines as we increase the number of
% rays per pixel
%
% Idea:
%
%   Pick a test scene, optics, and a sensor
%   Use ISET3d to render the test scene with different numbers of rays per
%     pixel
%   Assess the noise in different regions within the sensor image as we
%   sweep out the number of rays per pixel.
%
%

%%
clear;
ieInit;
if ~piDockerExists, piDockerConfig; end

%% Use flat surface for simplicity

rays = [32 64 256 512 1024];
[rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays);
ieNewGraphWin;
plot(rays,rn,'-ok','Linewidth',2);
line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Rays per pixel'); ylabel('Noise');
```

```
%% Looping over lenses
lenses = {'dgauss.22deg.12.5mm.json', 'wide.56deg.100.0mm.json'}
output = zeros(1, numel(lenses));
pnOut = zeros(1, numel(lenses));
for lensIndex = 1:numel(lenses)
    thisLens = lenses{lensIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'lensname', thisLens);
    output(lensIndex) = rn;
    pnOut(lensIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(lenses)],'xticklabel',lenses)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Lens type'); ylabel('Noise');

%% Looping over film diagonals
fDiagonals = {0.05, 0.1, 0.5, 5, 25};
output = zeros(1, numel(fDiagonals));
pnOut = zeros(1, numel(fDiagonals));
for diagIndex = 1:numel(fDiagonals)
    thisDiag = fDiagonals{diagIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'filmdiagonal', thisDiag);
    output(diagIndex) = rn;
    pnOut(diagIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(fDiagonals)],'xticklabel',fDiagonals)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Film Diagonal'); ylabel('Noise');

%% Looping over resolution
res = {64, 128, 256, 512, 1024};
output = zeros(1, numel(res));
pnOut = zeros(1, numel(res));
for resIndex = 1:numel(res)
    thisRes = res{resIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'filmresolution', [thisRes thisRes]);
    output(resIndex) = rn;
    pnOut(resIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(res)],'xticklabel',res)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Film Resolution'); ylabel('Noise');

%% Looping over sampler type
sensors = {'halton', 'solbol'}; %, 'stratified'};
output = zeros(1, numel(sensors));
pnOut = zeros(1, numel(sensors));
for sensIndex = 1:numel(sensors)
    thisSens = sensors{sensIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'sampler', thisSens);
    output(sensIndex) = rn;
    pnOut(sensIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(sensors)],'xticklabel',sensors)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Sampler'); ylabel('Noise');
```

```
%% Looping over exposure time
rays = [512];
[rn, pn, mySensor, thisR] = piRenderNoise('rays',rays);
times = {0.00001, 0.0001, 0.001, 0.01, 0.1};

output = zeros(1, numel(times));
pnOut = zeros(1, numel(times));
for timeIndex = 1:numel(times)
    thisTime = times{timeIndex};
    rays = [512];
    mySensor = sensorSet(mySensor, 'exp time', thisTime);
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'sensor', mySensor);
    output(timeIndex) = rn;
    pnOut(timeIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(times)],'xticklabel',times)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Exposure Time'); ylabel('Noise');

%% Looping over pixel size
rays = [512];
[rn, pn, mySensor, thisR] = piRenderNoise('rays',rays);
sizes = {0.000001, 0.000002, 0.000003, 0.000004, 0.000005};

output = zeros(1, numel(sizes));
pnOut = zeros(1, numel(sizes));
for sizeIndex = 1:numel(sizes)
    thisSize = sizes{sizeIndex};
    rays = [512];
    mySensor = sensorSet(mySensor, 'pixel size same fill factor', thisSize);
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'sensor', mySensor);
    output(sizeIndex) = rn;
    pnOut(sizeIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(sizes)],'xticklabel',sizes)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Pixel Size'); ylabel('Noise');

%% Looping over actual fill factor
rays = [512];
[rn, pn, mySensor, thisR] = piRenderNoise('rays',rays);
fillFactors = {0.8, 0.85, 0.9, 0.95, 1};

output = zeros(1, numel(fillFactors));
pnOut = zeros(1, numel(fillFactors));
for ffIndex = 1:numel(fillFactors)
    thisFF = fillFactors{ffIndex};
    rays = [512];
    myPixel = sensorGet(mySensor,'pixel');
    myPixel = sensorSet(myPixel, 'pixel', thisFF);
    mySensor = sensorSet(mySensor, 'pixel', myPixel);
    [rn, pn, idealSensor, thisR] = piRenderNoise('rays',rays, 'sensor', mySensor);
    output(ffIndex) = rn;
    pnOut(ffIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(fillFactors)],'xticklabel',fillFactors)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Fill Factor'); ylabel('Noise');
```

```
%% Cornell Box Ray Test
rays = [1024];
%rays = [32 64 256 512 1024];
sceneName = 'cornellboxreference';
nBounces = 3;
% TA's note: disabled recipe parameter, instead use 'scene name' in piRenderNoise.
% For example:
%   piRenderNoise('rays', rays, 'scene name', 'flat surface');
[rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays,...
                                    'scene name', sceneName,...
                                    'n bounces', nBounces);ieNewGraphWin;
plot(rays,rn,'-ok','Linewidth',2);
line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Rays per pixel'); ylabel('Noise');

%% Looping over lenses (Cornell Box)
sceneName = 'cornellboxreference';
nBounces = 3;
lenses = {'wide.56deg.100.0mm.json'};
% lenses = {'dgauss.22deg.12.5mm.json', 'wide.56deg.100.0mm.json'};
output = zeros(1, numel(lenses));
pnOut = zeros(1, numel(lenses));
for lensIndex = 1:numel(lenses)
    thisLens = lenses{lensIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays, 'lensname', thisLens,...
                                    'scene name', sceneName,...
                                    'n bounces', nBounces);
    output(lensIndex) = rn;
    pnOut(lensIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(lenses)],'xticklabel',lenses)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Lens type'); ylabel('Noise');

%% Looping over film diagonals (Cornell)
sceneName = 'cornellboxreference';
nBounces = 3;
fDiagonals = {25};
% fDiagonals = {0.05, 0.1, 0.5, 5, 25};
output = zeros(1, numel(fDiagonals));
pnOut = zeros(1, numel(fDiagonals));
for diagIndex = 1:numel(fDiagonals)
    thisDiag = fDiagonals{diagIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays, 'filmdiagonal', thisDiag,...
                                            'scene name', sceneName,...
                                            'n bounces', nBounces);
    output(diagIndex) = rn;
    pnOut(diagIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(fDiagonals)],'xticklabel',fDiagonals)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Film Diagonal'); ylabel('Noise');
```

```
%% Looping over resolution (Cornell)
sceneName = 'cornellboxreference';
nBounces = 3;
res = {1024};
%res = {64, 128, 256, 512, 1024};
output = zeros(1, numel(res));
pnOut = zeros(1, numel(res));
for resIndex = 1:numel(res)
    thisRes = res{resIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays,...
                                'filmresolution', [thisRes thisRes],...
                                'scene name', sceneName,...
                                'n bounces', nBounces);
    output(resIndex) = rn;
    pnOut(resIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(res)],'xticklabel',res)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Film Resolution'); ylabel('Noise');

%% Looping over sampler type (Cornell)
sceneName = 'cornellboxreference';
nBounces = 3;
sensors = {'stratified'};
%sensors = {'halton', 'solbol', 'stratified'};
output = zeros(1, numel(sensors));
pnOut = zeros(1, numel(sensors));
for sensIndex = 1:numel(sensors)
    thisSens = sensors{sensIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays,...
                                'sampler', thisSens,...
                                'scene name', sceneName,...
                                'n bounces', nBounces);
    output(sensIndex) = rn;
    pnOut(sensIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(sensors)],'xticklabel',sensors)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Sampler'); ylabel('Noise');

%% Looping over number of bounces
sceneName = 'cornellboxreference';
nBounces = {5};
%nBounces = {1,2,3,4,5};
output = zeros(1, numel(nBounces));
pnOut = zeros(1, numel(nBounces));
for bounIndex = 1:numel(nBounces)
    thisBounce = nBounces{bounIndex};
    rays = [512];
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays,...
                                'scene name', sceneName,...
                                'n bounces', thisBounce);
    output(bounIndex) = rn;
    pnOut(bounIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(nBounces)],'xticklabel',nBounces)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Number of Bounces'); ylabel('Noise');
```

```
%% Looping over exposure time (Cornell)
sceneName = 'cornellboxreference';
nBounces = 3;
rays = [512];
[rn, pn, mySensor, thisR] = piRenderNoise('rays',rays);
times = {0.1};
%times = {0.00001, 0.0001, 0.001, 0.01, 0.1};

output = zeros(1, numel(times));
pnOut = zeros(1, numel(times));
for timeIndex = 1:numel(times)
    thisTime = times{timeIndex};
    rays = [512];
    mySensor = sensorSet(mySensor, 'exp time', thisTime);
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays, 'sensor', mySensor,...
                                  'scene name', sceneName,...
                                  'n bounces', nBounces);
    output(timeIndex) = rn;
    pnOut(timeIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(times)],'xticklabel',times)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Exposure Time'); ylabel('Noise');

%% Looping over pixel size (Cornell)
sceneName = 'cornellboxreference';
nBounces = 3;
rays = [512];
[rn, pn, mySensor, thisR] = piRenderNoise('rays',rays);
sizes = {0.000005};
%sizes = {0.000001, 0.000002, 0.000003, 0.000004, 0.000005};

output = zeros(1, numel(sizes));
pnOut = zeros(1, numel(sizes));
for sizeIndex = 1:numel(sizes)
    thisSize = sizes{sizeIndex};
    rays = [512];
    mySensor = sensorSet(mySensor, 'pixel size same fill factor', thisSize);
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays, 'sensor', mySensor,...
                                  'scene name', sceneName,...
                                  'n bounces', nBounces);
    output(sizeIndex) = rn;
    pnOut(sizeIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(sizes)],'xticklabel',sizes)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Pixel Size'); ylabel('Noise');
%% Looping over fill factor (Cornell)
sceneName = 'cornellboxreference';
nBounces = 3;
rays = [512];
[rn, pn, mySensor, thisR, oi] = piRenderNoise('rays',rays);
fillFactors = {0.8}
%fillFactors = {0.8, 0.85, 0.9, 0.95, 1};
```

```
output = zeros(1, numel(fillFactors));
pnOut = zeros(1, numel(fillFactors));
for ffIndex = 1:numel(fillFactors)
    thisFF = fillFactors{ffIndex};
    rays = [512];
    myPixel = sensorGet(mySensor,'pixel');
    myPixel = sensorSet(myPixel, 'pixel', thisFF);
    mySensor = sensorSet(mySensor, 'pixel', myPixel);
    [rn, pn, idealSensor, thisR, oi] = piRenderNoise('rays',rays, 'sensor', mySensor,...
                                    'scene name', sceneName,...
                                    'n bounces', nBounces);
    output(ffIndex) = rn;
    pnOut(ffIndex) = pn;
end
plot(output,'-ok','Linewidth',2);
hold on;
plot(pnOut,'Linewidth',2,'Linestyle','--');
hold off;
set(gca,'xtick',[1:numel(fillFactors)],'xticklabel',fillFactors)
%%line([rays(1), rays(end)],[pn, pn],'Color','k','Linestyle','--');
grid on; xlabel('Fill Factor'); ylabel('Noise');



%% Getting the image
oiWindow(oi);
img=oiGet(oi, 'rgb image');
imshow(img)
```