Daniel Olson

Program3, 5608

Part 1:

Table:

Using importance sampling via the inverse CDF of our sampling functions.

Sampling Function	Variance	Samples needed for standard error of 0.008
(6 - x) / 16	56.8728	888,638
1 / 4	21.2907	332669
(x + 2) / 16	6.3615	99399
x / 8	0	1

Detail:

I calculate variance using random numbers and importance sampling. I continue calculating the average variance until the error of our variance is below 0.008. At that point, we record how many samples we have done and what our calculated variance is using monte carlo. The cut-off equation is as follows:

error = sqrt(Variance) / sqrt(numSamples)

Once error is less than 0.008, we have reached the number of samples necessary and can record results

Build and Run:

This is a python program. All you need to do is to make sure the libraries ...

numpy

math

random

are installed and the program is run with:

python part1.py

I am using Anaconda, an all-in-one python tool.

Part 2:

Build and Run:

First, make sure you have deleted any generated faces and vertices from previous runs inside on part2.obj

This is a python program. All you need to do is to make sure the libraries ...

numpy

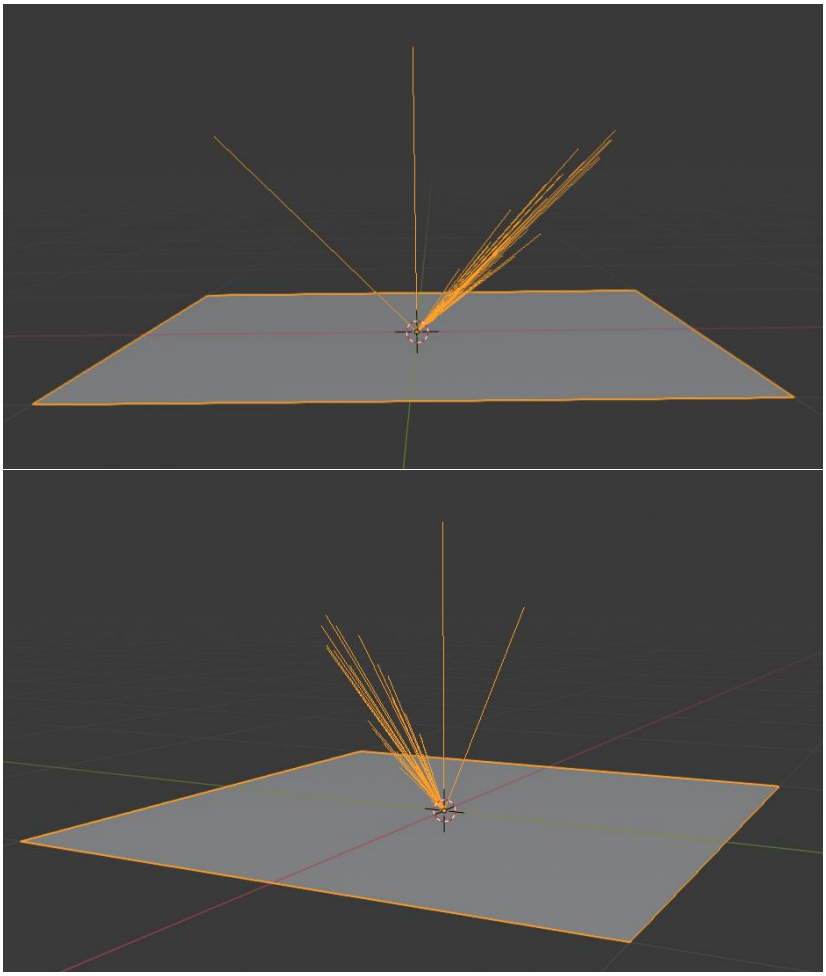
random

are installed and the program is run with:

python part2.py

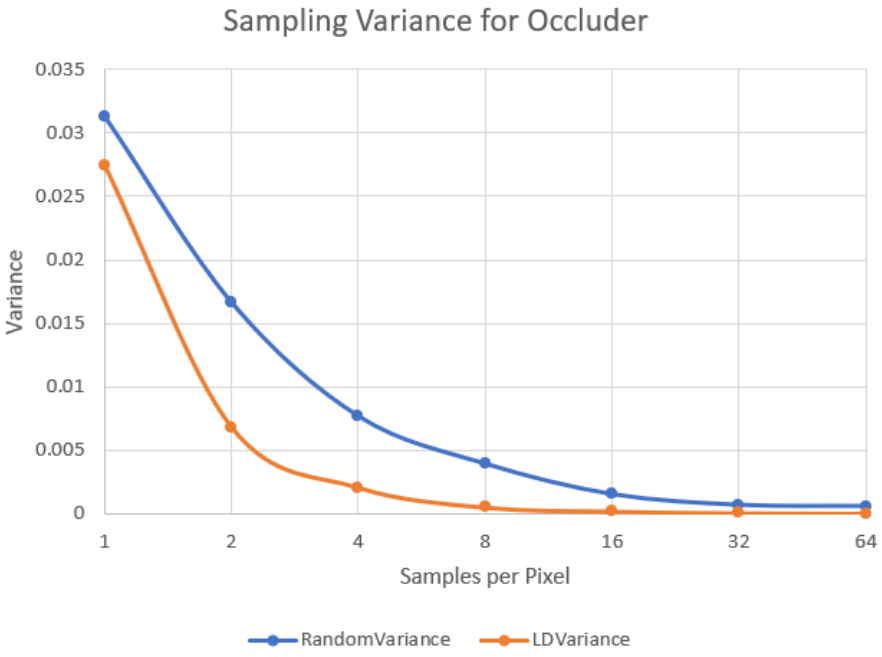
I am using Anaconda, an all-in-one python tool.

Obj in Blender:



Part 3:

Sampling and Variance:



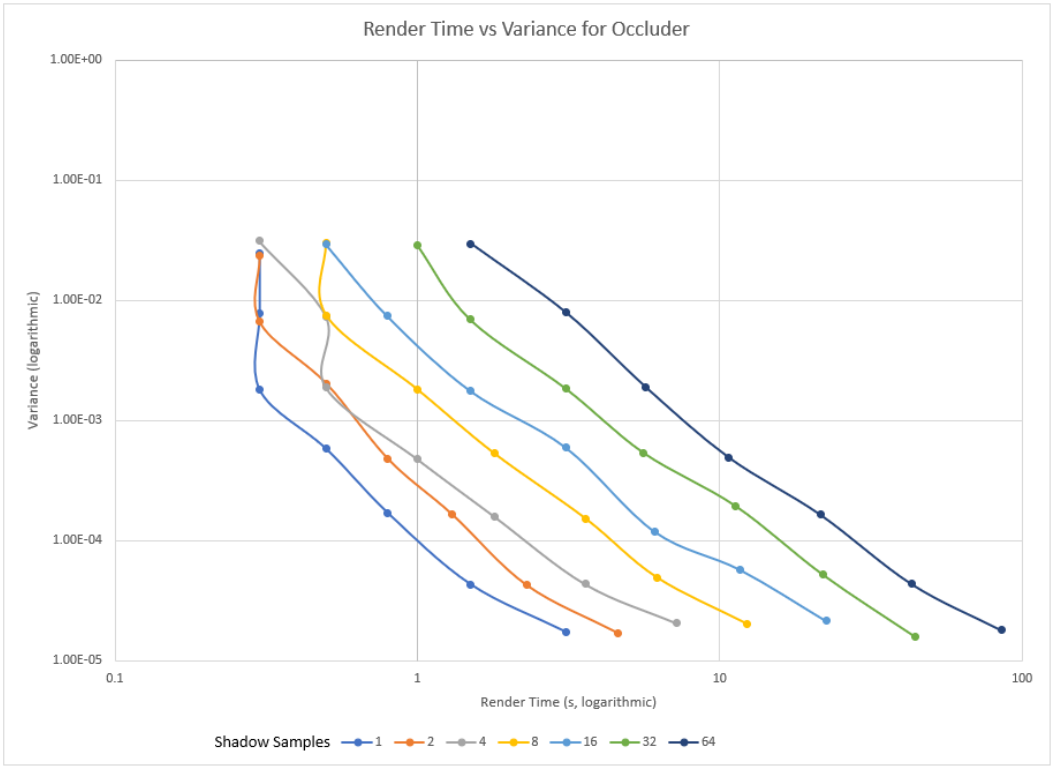
Q1:

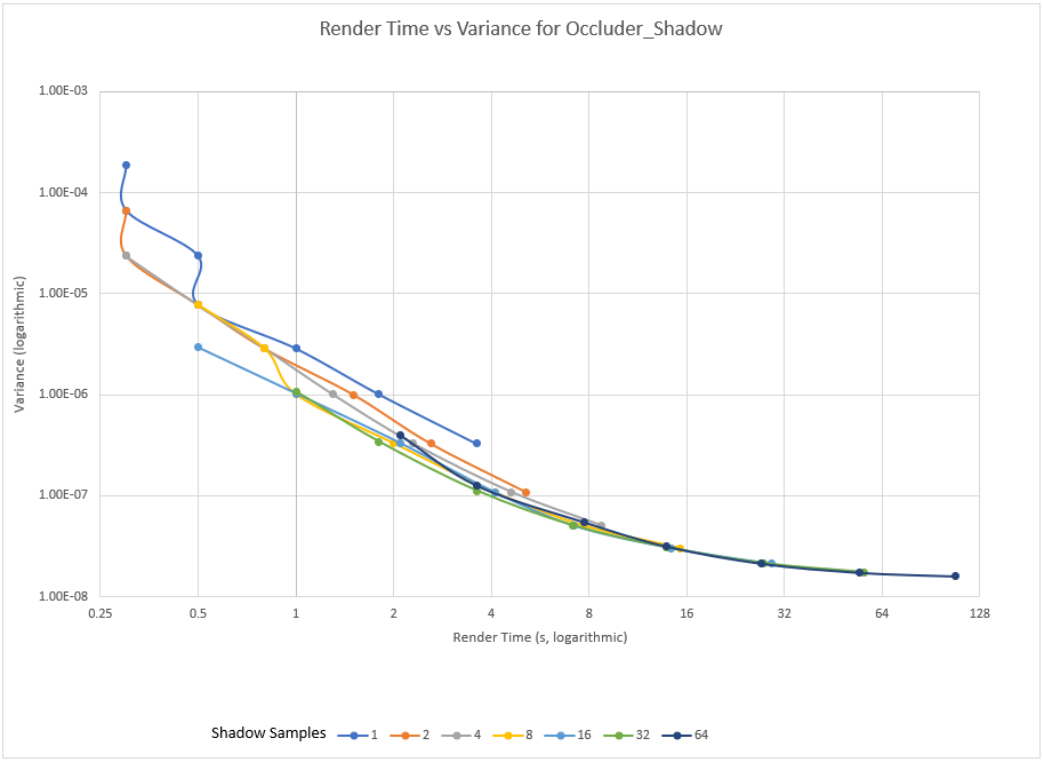
Variance changes by 1/n, with n being the number of samples. This is very easily seen with the random samples. Variance is about halved between each data point with makes sense as n is being multiplied by 2.

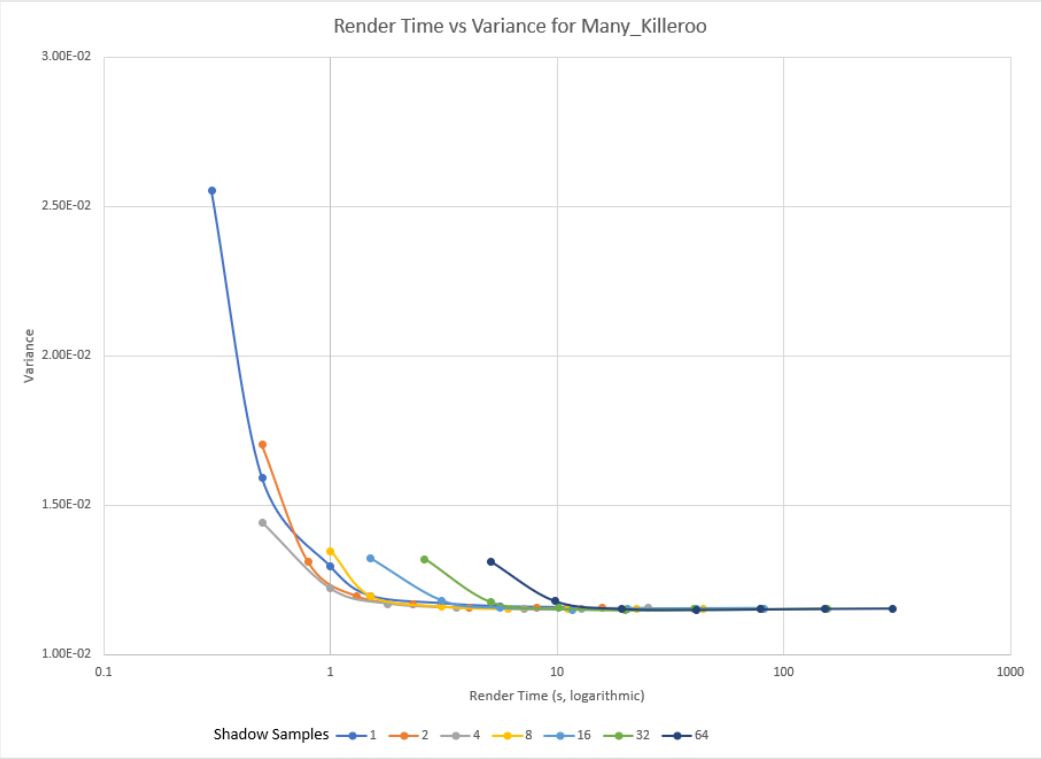
Q2:

The Low-discrepancy sampling performs much better than random sampling. By distributing evenly, we do not have overlap in our samples, allowing us to get a more fully sampled image.

Evaluating Efficiency:


Render Time vs Variance for Occluder
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
Variance (logarithmic)
0.1
1
10
100
Render Time (s, logarithmic)
Shadow Samples
1
2
4
8
16
32
64


Render Time vs Variance for Occluder_Shadow
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Variance (logarithmic)
0.25
0.5
1
2
4
8
16
32
64
128
Render Time (s, logarithmic)
Shadow Samples
1
2
4
8
16
32
64



Q1:

Occluder –

64 samples per pixel, 1 sample per light performed best. It is able to reach the same level of variance as different amounts of samples per light while rendering in the shortest time.

Occluder_shadow –

This are a lot closer here. There is not an obvious winner to me, but there are some obvious losers: 64 samples per light brings on diminishing returns as the rest of the graph has a near linear improvement and 64 samples per light starts shrinking variance less and less. 1 sample per light consistently has higher variance than the rest. The winner for me would most likely be 32 samples per light, 8 samples per pixel, the green line. It is the lowest dot of the linear chunk of our graph, before it starts reaching diminishing returns.

Many_Killeroo –

4 samples per light, 4 samples per pixel. This is the first point on our near horizontal light of variance. So it reaches our variance plateau while having the shortest render time.

Q2:

Definitely not. 1 sample per light performs the best for occluder while being one of the worst for the other options. For the occluder, more samples per light gives no benefit in terms of render time and variance, while being a viable option for others. 64 samples per light and pixel seems to be a bad approach overall. It is overkill in every situation, giving diminishing returns.

Q3:

With the occluder scene performing best with low samples per light, it makes me thing that path tracing works best with sparse scenes. There is a lot of empty, black space in that scene. Thus more complex, full scenes, with many things to light, seem to benefit much more from higher samples per light.