Sanders L. Hernandez

Comparison of Monte Carlo Sample Distribution Methods for Image Rendering

Introduction

We can describe image rendering as the process of producing a discrete representation of a continuous image function:

$$I(x,y) \text{ where } (x, y \mid x, y \in \mathbb{R}, 0 \leq x \leq X , 0 \leq y \leq Y)$$

where X is the image resolution in the x-axis, Y is the image resolution in the y-axis and $I(x,y)$ produces a color.

What rendering produces is an X by Y matrix I' where X and Y are as described above, and an entry in the matrix $I'[a,b]$ is known as a pixel, and each pixel [a,b] represents the average color of the image function from $I(a-1,b-1)$ to $I(a,b)$.

The image function is often extremely complex and not reasonably integrated, leaving the most common method of rendering Monte Carlo sampling. Our problem to discuss is: given a certain budget of samples, how to best distribute them between all the pixels to produce the best quality image? How do we decide what is better quality? The most commonly used method is to compare an image produced by rendering to another image of the same thing that is considered to be an ideal image.

So given two images: A, our ideal image and B, our image that were testing, with the same resolution we produce an error function:

$$E(A,B) = \frac{1}{XY} \sum_{x=1}^{X} \sum_{y=1}^{Y} |A[x,y] - B[x,y]| \ ]$$

That is, the average absolute value difference between each pixel. The goal with our algorithms to produce the lowest E given a certain budget.

Our Algorithms

Uniform Sampling - In uniform sampling each pixel is given an equal share of the samplings, so with a given budget of B samples and N pixels to render, each pixel gets ⌊B/N⌋ samples.

Dammertz Sampling (will be called d-sampling from now on) - Adapted from the paper, "A Hierarchical Automatic Stopping Condition for Monte Carlo Global Illumination" by Holger Dammertz, Johannes Hanika, Alexander Keller and Hendrik P.A. Lensch. Unlike our other algorithms, d-sampling doesn't aim to distribute and budget of samples, it instead aims produce an image with an even level of error across the image, below a given value V.

D-sampling works iteratively. Starting with the whole image, d-sampling will 2 separate images from I(x,y) with unique samples. It then compare this two images to one another with our error function. It then does 1 of 3 things depended on the resulting E.

If:

E ≤ V - The image (or bucket, to be defined) is done, and no further calculations are performed.

E ≤ V * 64 - The image (or bucket) is divided into two new buckets. Each bucket being non-overlapping, and combined covering the entire area of its parent. Our split attempts divide the error evenly between the resulting buckets, so they're not necessarily the same size. Our split is made perpendicular to the larger of two axis of the bucket under consideration.

E > V * 64 - The bucket is not split, or eliminated from further calculation.

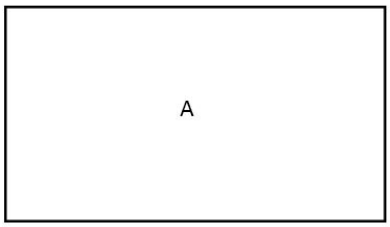
This process continues until all buckets are below our given value, V. with our split value, 64 was chosen empirically, after testing.

D-sampling has 2 arguments:

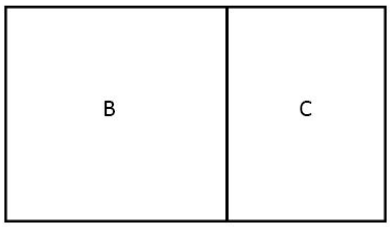
Max Variance - the amount error the algorithm tries to bring the image below.

Sample per Iteration - The number of samples each pixel gets per iteration before testing again for error.

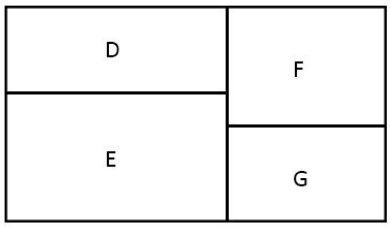
Step by step example of d-sampling progress



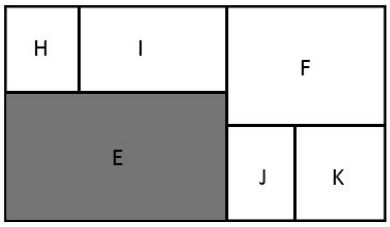
1.) Initial image, only 1 bucket that contains entire image.



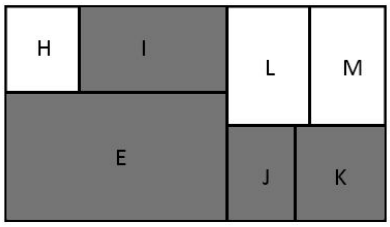
2.) After sampling and error calculation, initial bucket A, split into buckets B and C.



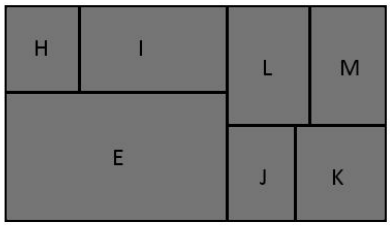
3.) After another sampling, buckets each bucket split into new buckets C, D, E and F.



4.) After another cycle, E is below our goal error and is no longer considered in ongoing calculation, F is above our split criteria and so is not split, D and G are split into H, I, J and K



5.) I, J and K are done calculation, F is split into L and M.



6.) All buckets done. Calculation ends

Adaptive Sampling (will be called a-sampling from now on) - A-sampling works by first splitting the image into buckets, in our case 16x16 pixel buckets, chosen empirically. The image is then rendered with an initial uniform distribution of samples between all pixels. We then find the variance in each bucket. The variance is found by taking the pixel with the lowest value, and that with the highest value and finding the difference.

We then distribute our remaining budget of samples between the buckets, giving the buckets with the higher maximum variance a greater share of the samples.

A-sampling has 3 arguments:

Initial Samples - The number of samples per pixel taken in the initial sampling.

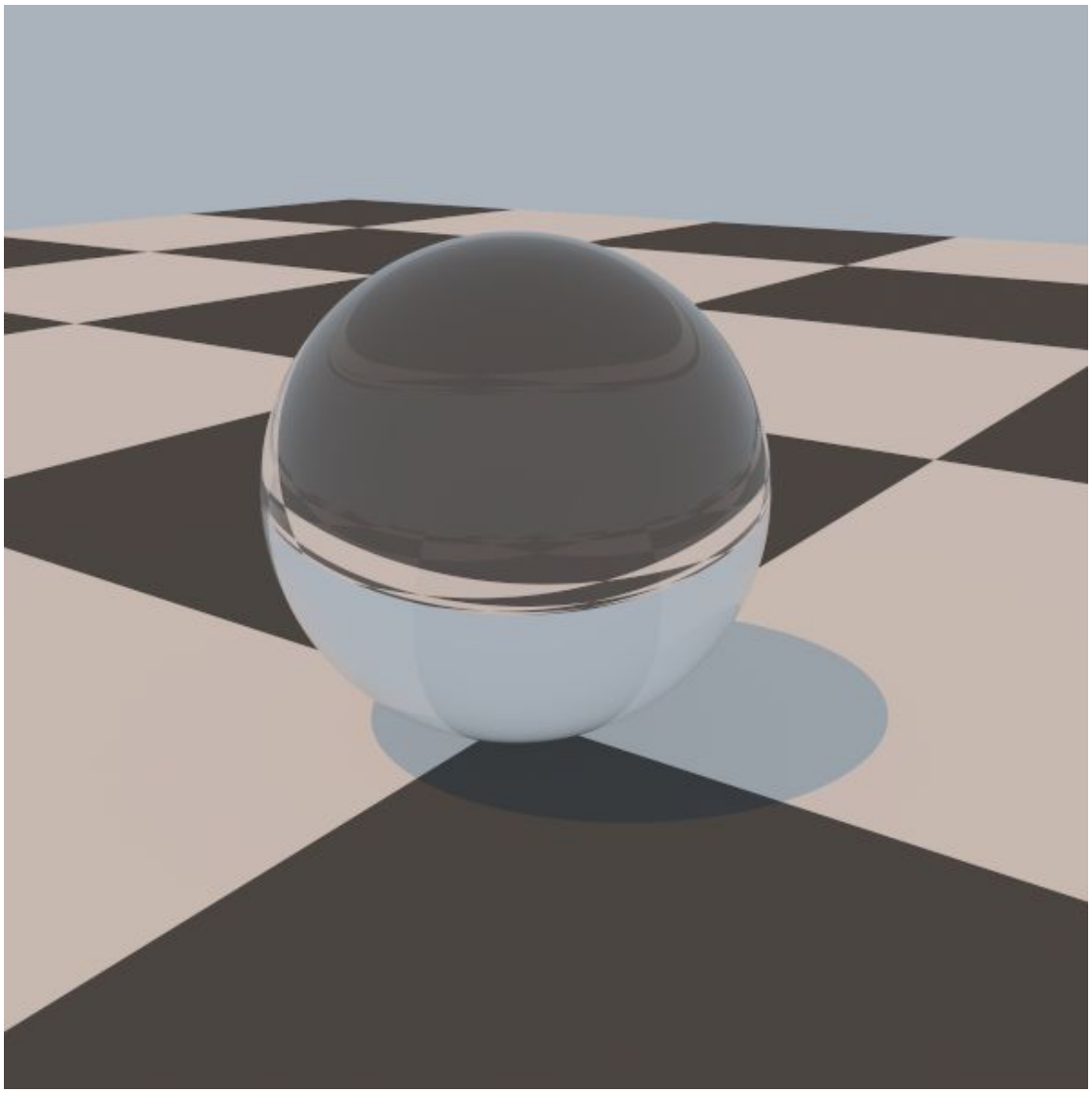
Max Variance - Any bucket who's variance greater or equal to this will receive the greatest share of the remaining sample budget.

Alpha - Adjusts the distribution of samples for buckets with variances between 0 and Max Variance. An alpha of 1 will linear distribute samples in proportion to their variance between 0 and Max Variance. Larger alphas will distribute samples more towards buckets with smaller variances and smaller alphas towards 0, will distribute samples more towards buckets with higher variances.

Analysis of our Algorithms

“Ideal” image - We’ve discussed our main criteria in judging our algorithms is their error when compared to an ideal image. In our analysis, our ideal image is one produced with uniform sampling, with each pixel receiving 8192 samples. At this sampling rate, with our image function, the majority of pixels will be oversampled, that is given many more samples than necessary to converge on a proper solution with only a small number or pixels being undersampled, not given enough samples to properly converge.

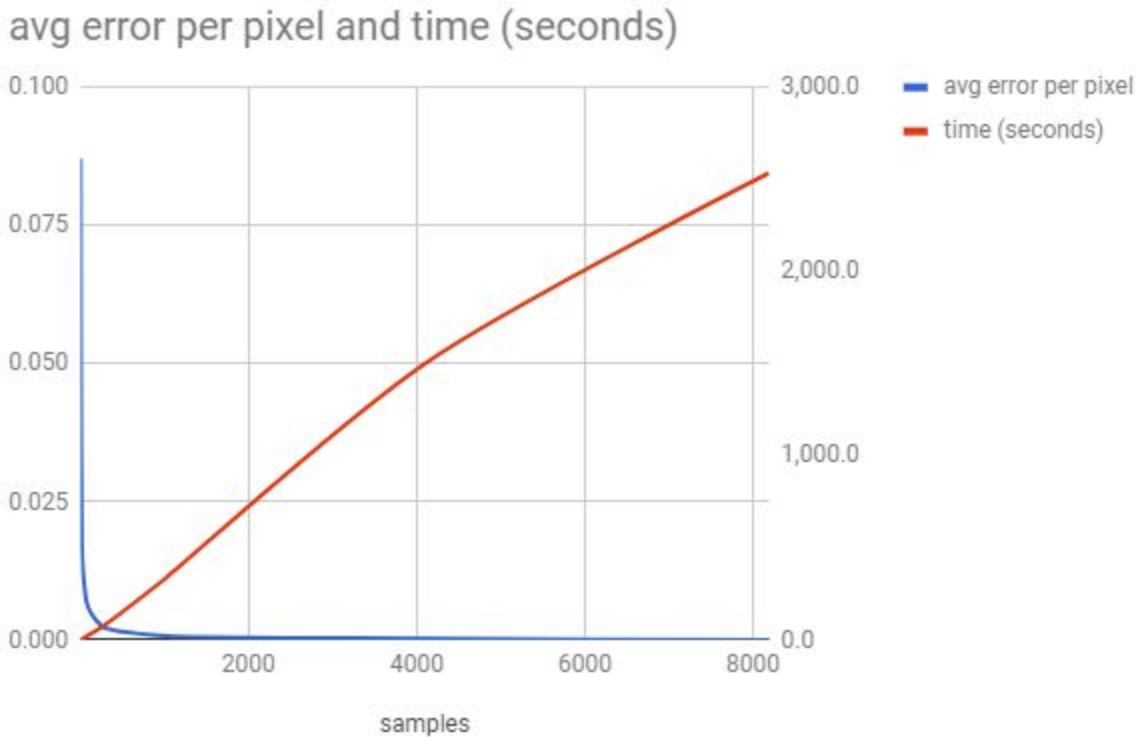
Our “ideal” image:



Time - While time is often an important factor when choosing rendering algorithms, it will not be discussed at length here. In general, time will increase with the sample budget nearly linearly. For the sake of reference the time in seconds to finish each render is included in the following graphs and data.

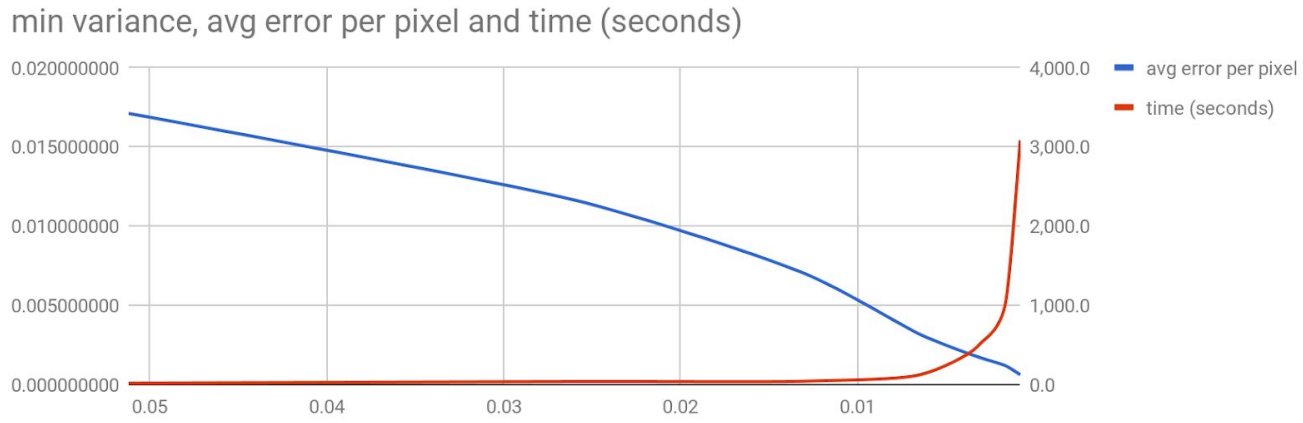
Uniform Sampling - Uniform Sampling only has one argument to adjust, the number of samples per pixel. Unsurprisingly, the greater the number of samples per pixel, the lower the error. We also see that error rate reduces at an inverse squared rate, $1 / \sqrt{N}$. It takes about about 4 times as many samples to cut the error rate in half, consistent with the convergence rate of other monte carlo applications.

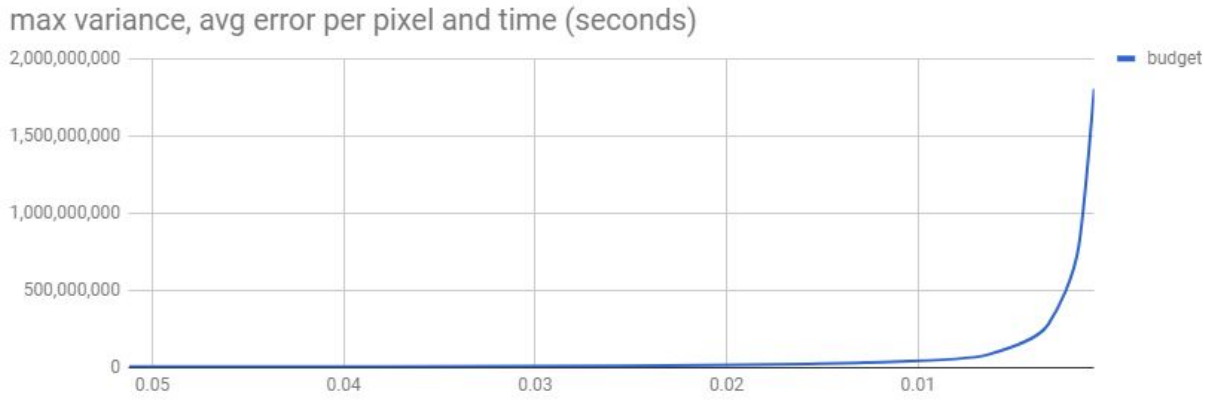
samples per pixel	avg error per pixel	time (seconds)	budget
1	0.087156400	0.5	360,000
4	0.052037200	1.4	1,440,000
16	0.017686900	4.8	5,760,000
64	0.007493980	17.9	23,040,000
256	0.002455640	71.9	92,160,000
1024	0.000701855	339.7	368,640,000
4096	0.000254906	1,500.0	1,474,560,000
8192	0.000000000	2,535.3	2,949,120,000



Dammertz Sampling - Our main argument for d-sampling is the max variance. We can see that as we lower our max variance our actual average error drops with it at a nearly linear rate, as expected. Similarly we see that our budget increases nearly exponentially.

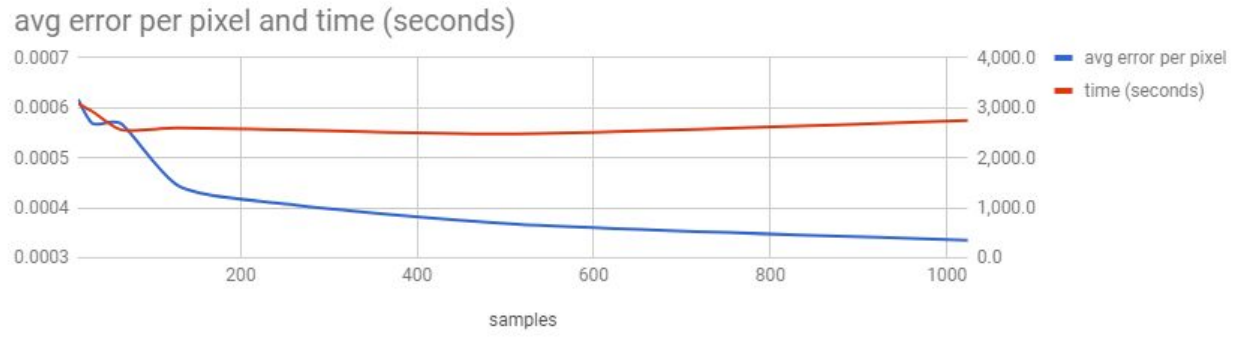
samples per iteration	max variance	avg error per pixel	time (seconds)	budget
16	0.0512	0.017108600	18.4	5,760,000
16	0.0256	0.011532900	39.8	11,520,000
16	0.0128	0.006917490	43.8	33,022,416
16	0.0064	0.003134160	127.0	82,631,952
16	0.0032	0.001750000	484.7	279,712,768
16	0.0016	0.001176170	1,071.4	789,416,368
16	0.0008	0.000617466	3,084.7	1,804,688,064

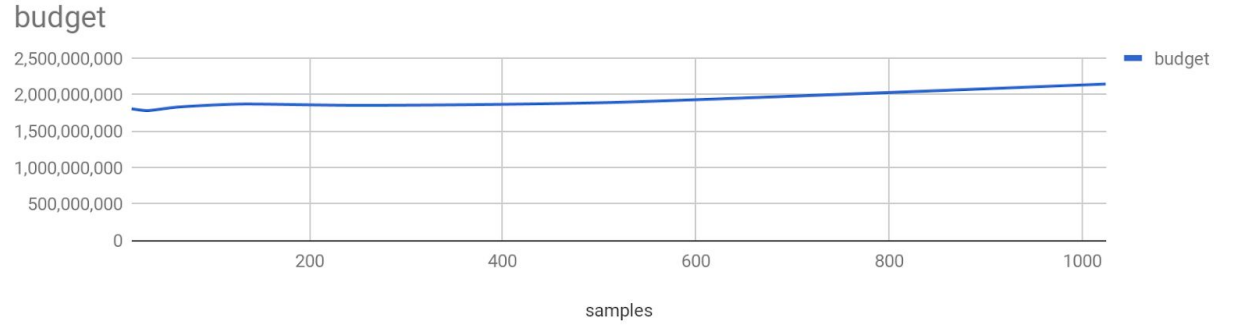




Our next argument for d-sampling is the number of samples per iteration. We observe with greater samples per iteration, the average error per pixel decrease a fair amount while the budget increases a small amount. It is my assumption that as iteration samples increase the likelihood of oversampling occurs. That is if a pixel only requires 200 samples to converge at an error lower than what we ask, but set iteration samples to 1024 it will receive 824 more samples than necessary, lowering its error considerable lower than our max variance argument.

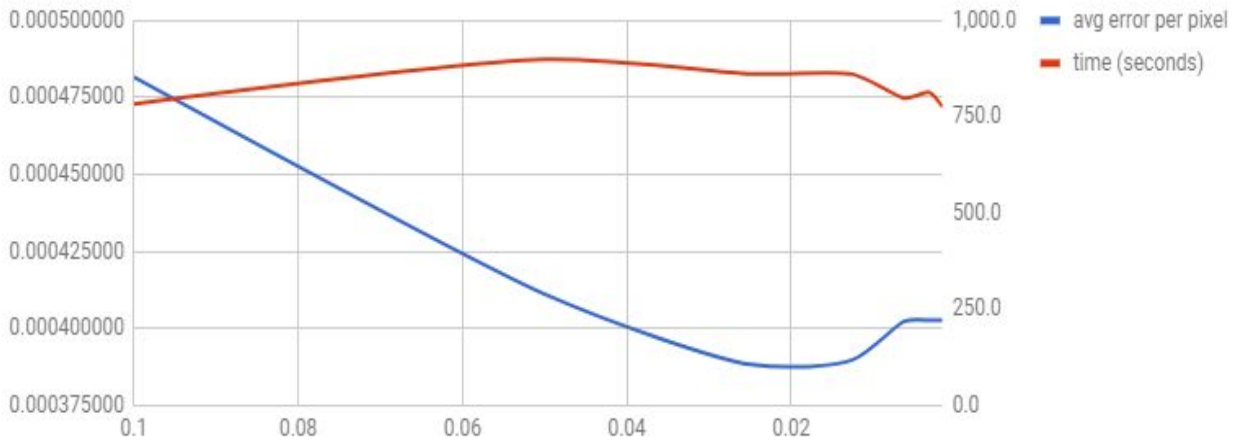
samples per iteration	max variance	avg error per pixel	time (seconds)	budget
16	0.0008	0.000617466	3,084.7	1,804,688,064
32	0.0008	0.000569252	2,931.7	1,779,584,000
64	0.0008	0.000569252	2,568.1	1,827,729,920
128	0.0008	0.000446438	2,603.9	1,868,254,720
256	0.0008	0.000407527	2,566.5	1,851,111,424
512	0.0008	0.000368214	2,483.6	1,890,442,240
1024	0.0008	0.000336296	2,750.3	2,143,590,400





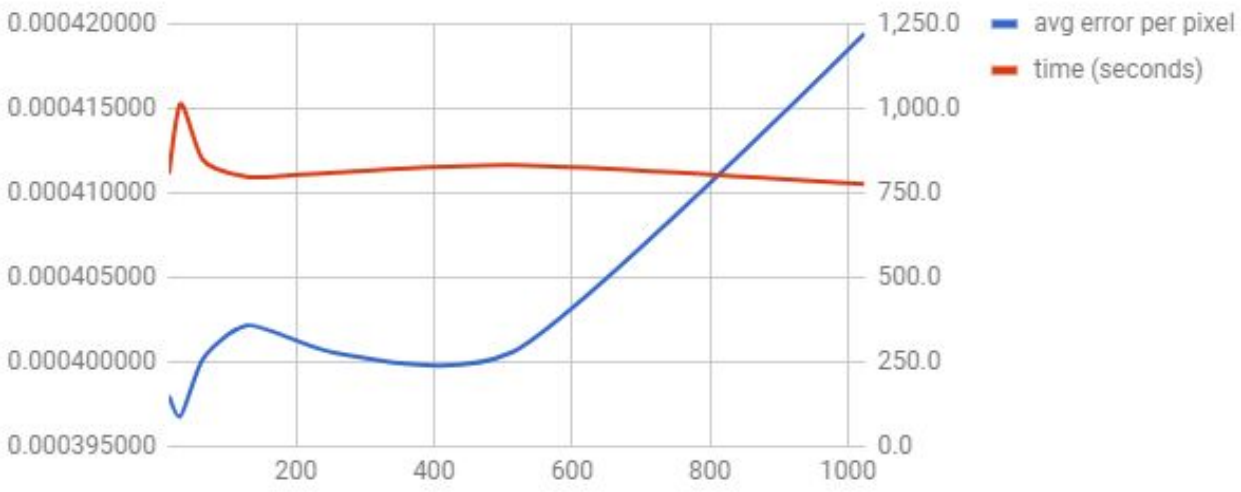
Adaptive Sampling - Our first argument under consideration is max variance. We can see max variance has an effect in which lowering to a certain extent lowers the average error, but after a certain point the error raises again and levels off. I believe this to be because with a high max variance a smaller number of buckets get a much larger share of the samples, leaving buckets in the middle that need more under sampled, while if the max variance is too small, nearly every bucket is over the max variance, and so a-samplings acts similarly to uniform sampling.

initial samples	max variance	alpha	avg error per pixel	time (seconds)	budget
128	0.1	1	0.000481656	783.6	727,702,400
128	0.05	1	0.000411075	899.0	725,477,888
128	0.025	1	0.000388221	861.2	723,433,472
128	0.0125	1	0.000389691	860.7	720,492,864
128	0.00625	1	0.000402175	798.6	717,623,680
128	0.003125	1	0.000402600	813.0	717,466,368
128	0.0015625	1	0.000402600	775.8	717,466,368



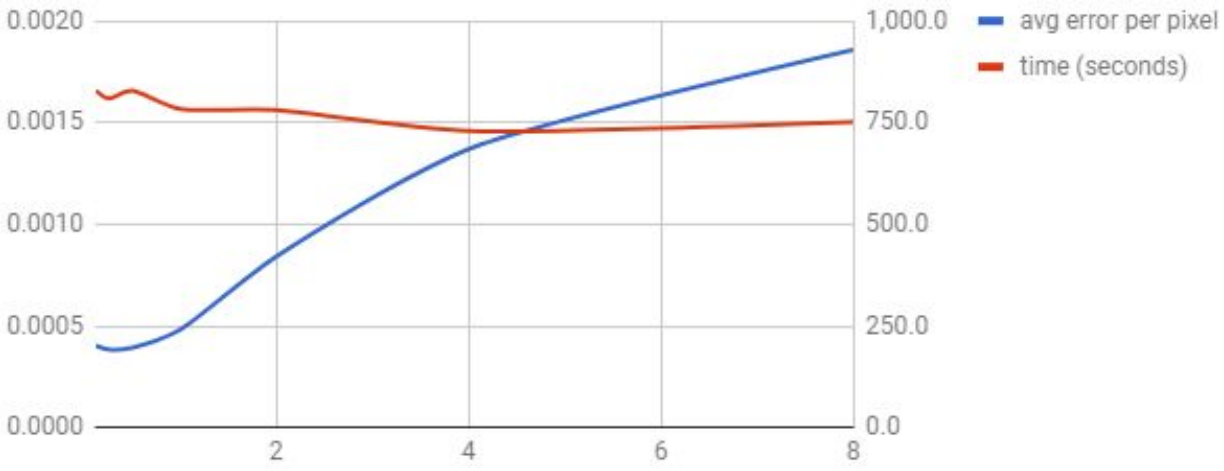
Next we compare the initial sampling size. We can see that with a larger initial sampling the average error actually increases. This is most like due to most of our budget being used initially, oversampling sections of the image that don't require it, not leaving enough left to properly sample higher variance buckets.

initial samples	max variance	alpha	avg error per pixel	time (seconds)	budget
16	0.00625	1	0.000398028	807.6	716,210,624
32	0.00625	1	0.000396786	1,012.6	716,416,896
64	0.00625	1	0.000400120	851.4	716,766,720
128	0.00625	1	0.000402175	798.6	717,623,680
258	0.00625	1	0.000400559	811.9	720,725,696
512	0.00625	1	0.000400581	832.6	725,708,544
1024	0.00625	1	0.000419448	776.4	729,144,000



Our last argument to compare is the distribution alpha. We can see this does have a notable effect on the average error. If we look back at the data for adjusting max variance we see that a value of 0.1 had the highest error. But here, we can see that with an alpha of 0.25 a max variance of 0.1 now gives us our lowest average error. Conversely, increasing alpha over 1 clearly increases error. This is an obvious result of pushing sampling away from high variance buckets to low, oversampling them.

initial samples	max variance	alpha	avg error per pixel	time (seconds)	budget
128	0.1	0.125	0.000404662	828.7	719,782,592
128	0.1	0.25	0.000386405	809.4	721,313,792
128	0.1	0.5	0.000394862	828.6	724,141,376
128	0.1	1	0.000481656	783.6	727,702,400
128	0.1	2	0.000841467	781.0	730,192,576
128	0.1	4	0.001369760	730.1	731,378,560
128	0.1	8	0.001858430	751.7	731,790,848



Finally we perform a comparison of the three algorithms in relation to a given budget. As stated, d-sampling does not work on a given budget, so budgets were decided by running d-sampling, and seeing how many samples were used.

We can see, uniformly that d-sampling performed worse for all given budgets, while a-sampling performed best, marginally out performing uniform sampling on all tests. So the question is why does d-sampling perform worse? Or really does it? D-sampling does something that both uniform and a-sampling aren't designed to do, it brings the error of all pixels below the given threshold, giving every image it renders a more uniform error distribution. On the other hand a-sampling and uniform sampling both have sample limits, meaning there's always possibility of pixels being undersampled. I can confirm from the tests made that there are a minority of pixels in our image that required 80,000 or more each to converge at an answer below our acceptable variance with d-sampling, while these pixels only added minorly to the average error of the image they added substantially to the needed sample. In comparison, the most samples used for a pixel in a-sampling or uniform sampling was 8192.

budget	uniform	dammertz	adaptive
5,760,000	0.017686900	0.017108600	0.016298300
11,520,000	0.011419900	0.011532900	0.010268300
33,022,416	0.005655540	0.006917490	0.005034330
82,631,952	0.002626340	0.003134160	0.002214470
279,712,768	0.000916351	0.001750000	0.000733740
789,416,368	0.000427407	0.001176170	0.000372622
1,804,688,064	0.000214495	0.000617466	0.000170871