User's guide for funcsamp2D: a program for testing 2D sample sequences

Per Christensen
Pixar Animation Studios

July 22, 2019

Abstract

This user's guide describes a simple C++ program that computes errors for sampling of 2D functions on the unit square with any collection of sample sequences. These errors can then be plotted for a visual analysis of error and convergence rates.

1 Introduction

This program is published as a part of the SIGGRAPH 2019 Course "My favorite samples" organized by Alexander Keller [KGA*19]. It is a polished version of a program I used to generate error plots for the paper "Progressive multi-jittered sample sequences" [CKK18].

Before getting started, it should be emphasized that comparing sample sequences with each other is a complex task with many fine nuances. Which sequence is best depends on the characteristics of the function being sampled and the error metric chosen, and sometimes there are no clear answers. Sometimes it isn't even clear what "best" means!

In numerical integration, the error can often be bounded by a product of discrepancy and a number characterizing the function class. Star discrepancy [Nie92] is often used to evaluate sample sequences; the corresponding function class is the class of functions of bounded variation (in the sense of Hardy and Krause [Nie92]). But we – and many others, for example Michell [Mit92] and Keller [Kel04] – have found star discrepancy to be highly misleading as an indicator of real sampling error and sample sequence quality. A better measure is the random-edge discrepancy introduced by Dobkin et al. [DM93, DEM96]. But even this measure only evaluates discontinuous functions with straight edges. Functions to be integrated in graphics can be discontinuous, continuous, or even smooth. They need to be manipulated to get bounded variance (for example using multiple importance sampling and changing the measure). Therefore a more complete picture of convergence is needed to assess the pros and cons of each sample sequence.

We have found testing by integration of simple functions to be more indicative of sample sequence quality. The ultimate test in a rendering setting is, of course, how low error and how fast convergence a sample sequence gives when used for actual image generation. But we have found it useful to start with comparing sample sequences by using them to estimate integrals of simple functions.

2 Functions

The functions we use are all simple test functions, but despite their simplicity they provide valuable insights and intuition about various sample sequences: if a sequence has high error and/or slow convergence for simple tests like these, then it is unlikely that it would give desirable results if it is used in more complex settings. The sample domain is the unit square, $[0,1)^2$.

The built-in functions are a collection of discontinuous, continuous, smooth, and one-dimensional functions: quarterdisk, fulldisk, triangle, quarterdiskramp, fulldiskramp, triangleramp, quartergaussian, fullgaussian, bilinear, biquadratic, sinxy, sininvr, stepx, rampx, lineary, gaussianx, siny, and sin2x. These functions are shown in Figure 1.

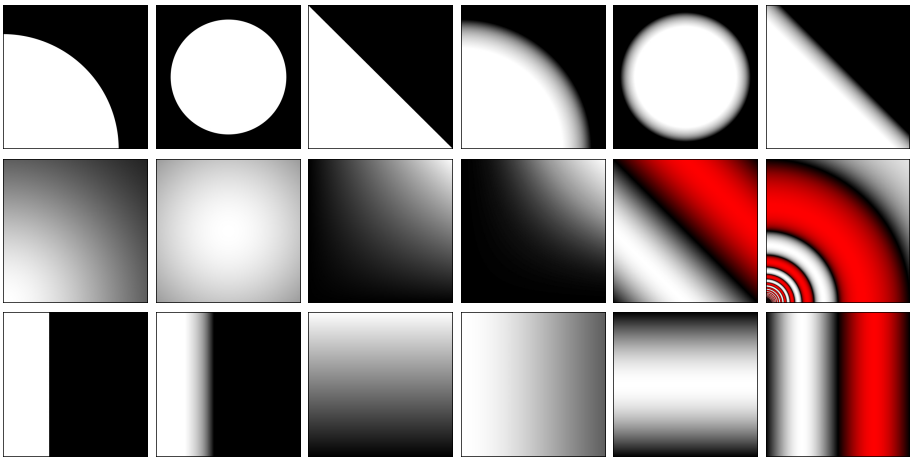

Figure 1: Built-in simple functions. (Red indicates negative values.)

You are encouraged to add more functions by editing and recompiling the funcsamp2D.cpp source code.

3 Sample sequences

The sample sequences used to sample a function are read from a file. Using a single sequence for sampling provides very little solid information about the average case. Instead, many realizations of functions and sequences must be tested and the average error used; we recommend that each input file should contain at least 100 different sample sequences of a given type. For quasi-random sequences, there is only one sequence defined in each dimension, so to get 100 sequences, one must *randomize* the sequences. Common randomization techniques are Cranley-Patterson rotation [CP76], exclusive-or scrambling [KK02a], and Owen scrambling [Owe97, KK02b, Bur19].

The file format consists of a header with two lines of text describing the sample sequence in the file. (These two lines are actually completely ignored and can contain anything.) There is also a line of text before each sequence of 2D sample points. It does not matter what text is in that line, but we usually write the sequence number. Each sample point is on a separate line. Here is an example of a sample sequence file:

```
// Table of 100 sequences of 1024 uniform random 2D samples.
// Each sample is generated with drand48().
// Sequence 0:
0.000000000000 0.000985394675
0.041631001595 0.176642642543
0.364602248391 0.091330612112
... etc ...
// Sequence 1:
0.041630344772 0.454492444729
0.834817218167 0.335986030145
0.565489403566 0.001766912392
... etc ...
```

We provide 14 input files with sample sequences to get started: random; best candidates (blue noise); irrational sequences R_2 [Rob18] with Cranley-Patterson rotation; Halton sequences [Hal64] in base (2,3), base (5,7), base (11,13), base (17,19), and base (23,29) with Owen scambling; Sobol' sequences [Sob67, Grü12] in dimensions (0,1) to dimensions (8,9); and progressive multi-jittered (0,2) sequences [CKK18]. Each file contains 100 sequences with 1024 sample points in each sequence. Figure 2 shows samples from some of these sample sequences.

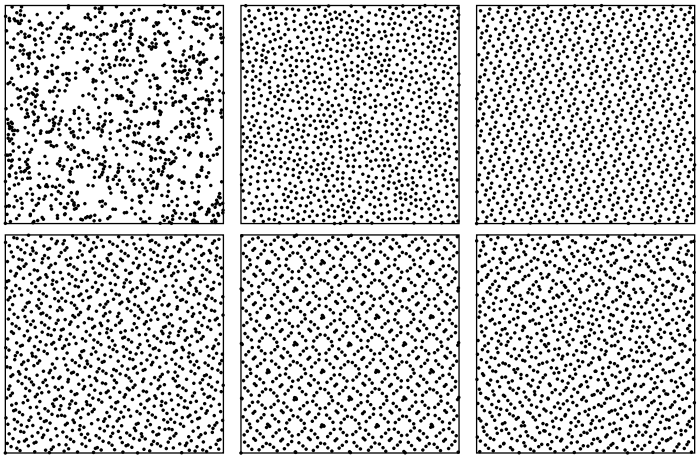

Figure 2: First 1024 samples from six progressive sequences. Top row: uniform random; best candidates; irrational. Bottom row: Halton with base (2,3); Sobol' dimensions (0,1); pmj02.

4 Example of use

The program parameters are 1) the function name (from the list of known functions listed in section 2), 2) the name of a file containing a table of sequences of 2D sample points from a given type of sample sequence, 3) optionally the number of samples in each sequence (default is 1024), and 4) optionally the number of sequences to be read from the file (default is 100):

```
funcsamp2D functionName samplesFilename [numSamples numSequences]
```

For example:

```
funcsamp2D quarterdisk random_1024samples_100sequences.data 1024 100 >
                                                    errors_quarterdisk_random.data
```

The output is a table of error values for increasing sample counts:

```
4 0.170000
8 0.127500
12 0.101667
16 0.086875
20 0.086000
...
1020 0.012088
1024 0.012021
```

We have chosen to only output error values for every fourth sample count since that tends to give smoother curves when plotted with Gnuplot as described in the following section.

5 Plotting the sampling errors

The generated tables of error values contain lots of interesting information, and plotting them as curves can provide even more insight and intuition. Here we use Gnuplot, a popular and free graphing utility that provides simple visualizations of mathematical functions and data. Information on installation and use of Gnuplot can be found on the Gnuplot homepage [gnu19]. Other plotting programs can be used to generate similar plots.

Gnuplot can be controlled with a script that defines the plot title, axis ranges, input data file names, colors of the curves, and much more. For example, to plot the results of sampling a quarterdisk function with six different sample sequences – and also reference lines for convergence rates $O(N^{-0.5})$ and $O(N^{-0.75})$ – we use a script called 'plotQuarterdiskError.gp' that contains the following commands:

```
set terminal postscript enhanced color
set output "quarterdiskError.eps"
set title "quarterdisk function: sampling error" font ",30"
set xlabel "samples" font ",30"
set ylabel "error" font ",30"
set tics font ",25"
set key spacing 1.5
set logscale x
set logscale y

plot [16:1024] [0.001:0.1]\
 "errors_quarterdisk_random.data" using 1:2 smooth unique lw 5.0 title "random",\
 "errors_quarterdisk_bestcand.data" using 1:2 smooth unique lw 5.0 title "best cand",\
 "errors_quarterdisk_irrational_rot.data" using 1:2 smooth unique lw 5.0 title "irrational rot",\
 "errors_quarterdisk_halton_base23_owen.data" using 1:2 smooth unique lw 5.0 title "Halton Owen",\
 "errors_quarterdisk_sobol_dim01_owen.data" using 1:2 smooth unique lw 5.0 title "Sobol Owen",\
 "errors_quarterdisk_pmj02.data" using 1:2 smooth unique lw 5.0 lt rgb "black" title "pmj02",\
 0.35*x**(-0.5) lw 3.0 lt rgb "gray" title "N^{-0.5}",\
 0.4*x**(-0.75) lw 3.0 lt rgb "gray" title "N^{-0.75}"
```

Running the command 'gnuplot plotQuarterdiskError.gp' generates the plot in Figure 3 (top). In this test, the irrational, Halton, Sobol', and pmj02 sequences have the lowest error; their errors converge as roughly $O(N^{-0.75})$ which is typical for sampling a discontinuous function with a well-distributed sample sequence. Best candidates are clearly worse, and random sequences have the highest error and slow convergence rate at $O(N^{-0.5})$ – as expected.

(For a similar test of the fulldisk function, the results are nearly the same, except that the irrational sequence performs remarkably well: its error is below the other sequences. This is probably because the full disk has two discontinuities along each direction, so as the regular pattern of the irrational samples gets shifted by a Cranley-Patterson rotation, some of the samples enter the disk while a similar number of samples leave the disk – see the similar analysis by Ramamoorthi et al. [RAMN12].)

Figure 3 (middle) shows sampling error curves for the continuous (but not smooth) function quarterdiskramp for the same six sample sequences. Here the convergence rate for Halton, Sobol', and pmj02 is roughly $O(N^{-1})$ overall, but even better at roughly $O(N^{-1.25})$ when the number of samples is a power of two. The irrational sequence converges roughly as $O(N^{-1})$ (with no improved convergence at powers of two), while best candidates and random sequences still converge as $O(N^{-0.5})$.

As a third test, we choose a smooth function, the quartergaussian. Figure 3 (bottom) shows error curves for sampling with the same six sample sequences. The convergence rate for Halton, Sobol', and pmj02 is roughly $O(N^{-1})$ overall, but now roughly $O(N^{-1.5})$ when the number of samples is a power of two. The irrational sequence has significantly higher error, and best candidates and random sequences still converge as $O(N^{-0.5})$.

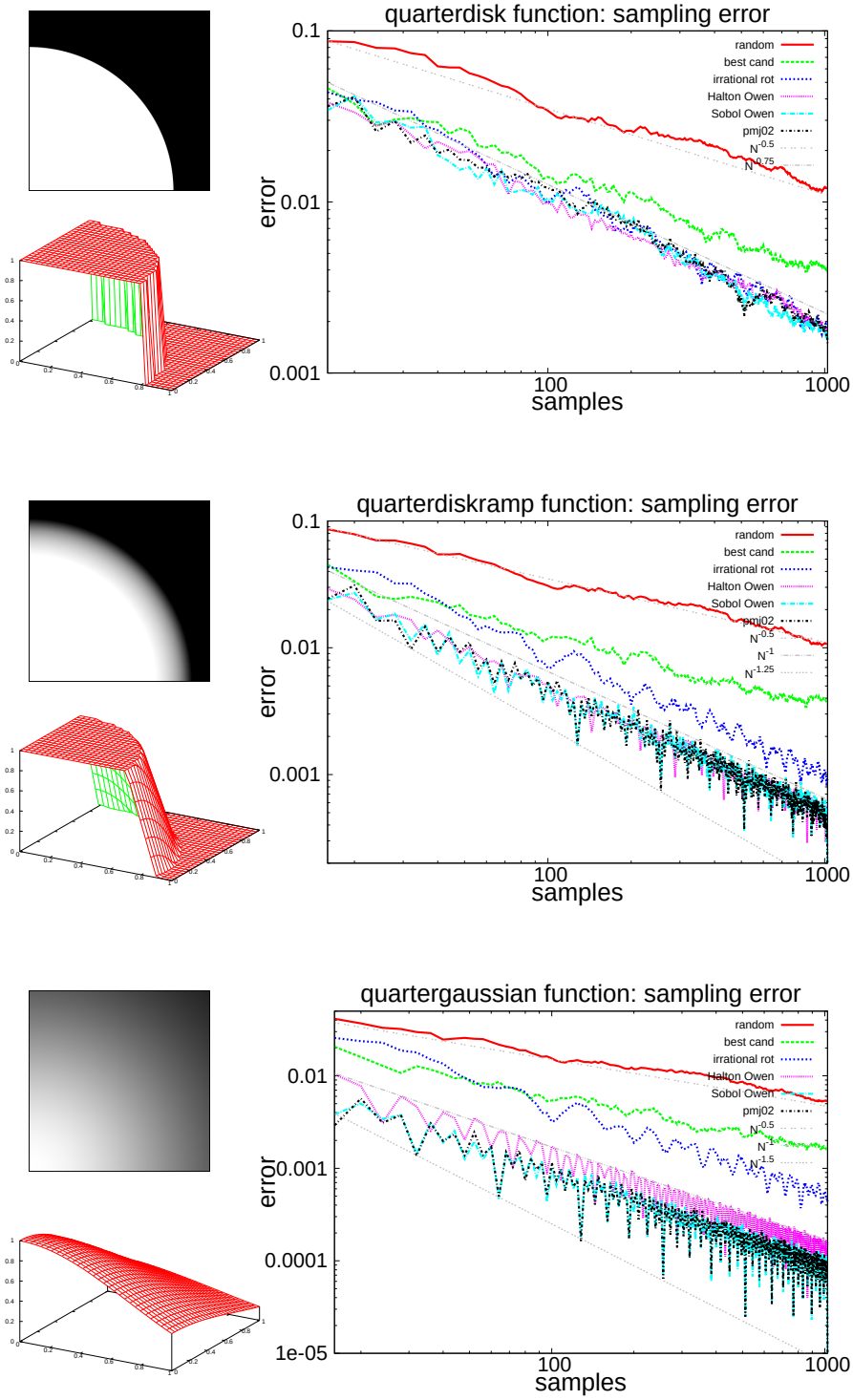

Figure 3: Errors for sampling the quarterdisk, quarterdiskramp, and quartergaussian functions with various sample sequences.

Note that the curves are quite noisy even though each curve is the average of 100 trials (sequences). In our pmj paper [CKK18] we used averages of 10000 trials instead of the 100 used here to get more accurate and smooth curves.

6 Suggestions for other tests

In this section we describe a few additional tests that might be interesting:

- Run one of the tests described above with only a single sample sequence of each type (instead of the 100 we used). If you plot the error curves, you will see that they are extremely jaggy and hard to draw any conclusions from.
- Repeat the tests in section 5 with the provided Halton sequences with higher bases and Sobol' sequences with higher dimensions. You should see that those sequences give higher error than their low-dimensional relatives. (See for example the "Myths of computer graphics" by Keller [Kel04] for a discussion of the problems with the higher dimensions.)
- Generate other sample sequences using code from e.g. Kollig and Keller [KK02a, KK02b], the PBRT book [PJH17], or the web page of Grünschloß [Grü12]. Randomize the sequences with Cranley-Patterson rotation, exclusive-or, or random digit scrambling. How do these sequences perform?
- Test the sample sequences on the sininvr function $f(x, y) = \sin(\pi/\sqrt{x^2 + y^2})$. This is a smooth function, but the error and convergence is dominated by the high-frequency region near (0,0) where the derivatives are huge. You should see that for this function, the convergence is only $O(N^{-0.75})$ for even the best sample sequences.
- Generate 100 stratified (jittered) sample *sets* with a fixed number of sample points in each set. For example 20x20 strata giving 400 sample points in each set, or 32x32 strata giving 1024 sample points in each set. Then pick a function to run funcsamp2D on, and generate error plots for the error results. You should see curves with high error for low sample counts, and a very rapid drop in the error close to the full number of samples. The high initial error illustrates that sample *sets* are not suitable when sample *sequences* are needed (for progressive rendering, for example).

7 The inner workings of the funcsamp2D program

The funcsamp2D program is very simple. It is written in C++. It first parses the command-line parameters. Then it reads all the sample points from the samples file and starts sampling the specified function with numSequences trials for each sample count. It prints out the average error for every fourth sample count up to numSamples: 4, 8, 12, ..., numSamples.

All calculations are in double-precision to minimize rounding errors (and since speed is not important). The reference values for the integrals of most functions are trivial or have been calculated analytically; a few have been computed by numerical integration with 100 million stratified samples.

Use a command like this to compile funcsamp2D.cpp on a typical Linux computer:

```
g++ -O3 -Wall -o funcsamp2D funcsamp2D.cpp
```

8 Possible extensions

Feel free to modify this program in any way you like (but please give attribution to the original author). For example: add more 2D functions, or modify it to sample higher-dimensional functions with higher-dimensional sample sequences.

For more information about sampling in rendering, please see the course notes [KGA*19] or Keller's survey paper [Kel12].

Acknowledgments

Many thanks to Alexander Keller for organizing the SIGGRAPH Course "My favorite samples", for suggesting that I should publish the funcsamp2D program as part of the course materials, and for thoughtful comments on how to improve this user's guide.

Also many thanks to my colleagues Andrew Kensler and Charlie Kilpatrick, Brent Burley at Walt Disney Animation Studios, Wojciech Jarosz at Dartmouth College, and Victor Ostromoukhov at Université de Lyon for many nuanced discussions about how to test and compare sample sequences.

And finally, thanks to Steve May and Richard Guo at Pixar for permission to publish this program.

References

[Bur19] Burley B.: Implementing hash-based Owen scrambling, 2019. (Personal communication).

[CKK18] Christensen P., Kensler A., Kilpatrick C.: Progressive multi-jittered sample sequences. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering) 37*, 4 (2018), 21–33.

[CP76] Cranley R., Patterson T.: Randomization of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analysis 13*, 6 (1976), 904–914.

[DEM96] Dobkin D., Eppstein D., Mitchell D.: Computing the discrepancy with applications to supersampling patterns. *ACM Transactions on Graphics 15*, 4 (1996), 354–376.

[DM93] Dobkin D., Mitchell D.: Random-edge discrepancy of supersampling patterns. *Proc. Graphics Interface* (1993), 62–69.

[gnu19] Gnuplot: homepage, 2019. http://gnuplot.sourceforge.net.

[Grü12] Grünschloss L.: QMC sampling source code, 2012. http://gruenschloss.org.

[Hal64] Halton J.: Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM 7*, 12 (1964), 701–702.

[Kel04] Keller A.: Myths of computer graphics. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods* (2004), pp. 217–243.

[Kel12] Keller A.: Quasi-Monte Carlo image synthesis in a nutshell. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods* (2012), pp. 213–249.

[KGA*19] Keller A., Georgiev I., Ahmed A., Christensen P., Pharr M.: My favorite samples. In *SIGGRAPH Course Notes*. ACM, 2019.

[KK02a] Kollig T., Keller A.: Efficient multidimensional sampling. *Computer Graphics Forum (Proc. Eurographics) 21*, 3 (2002), 557–563.

[KK02b] Kollig T., Keller A.: Samplepack, 2002. www.uni-kl.de/AG-Heinrich/SamplePack.html.

[Mit92] Mitchell D.: Ray tracing and irregularities in distribution. *Proc. Eurographics Workshop on Rendering* (1992), 61–69.

[Nie92] Niederreiter H.: *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, 1992.

[Owe97] Owen A.: Scrambled net variance for integrals of smooth functions. *The Annals of Statistics 25*, 4 (1997), 1541–1562.

[PJH17] Pharr M., Jacob W., Humphreys G.: *Physically Based Rendering: From Theory To Implementation*, 3rd ed. Morgan Kaufmann, 2017.

[RAMN12] Ramamoorthi R., Anderson J., Meyer M., Nowrouzezahrai D.: A theory of Monte Carlo visibility sampling. *ACM Transactions on Graphics 31*, 5 (2012).

[Rob18] Roberts M.: The unreasonable effectiveness of quasirandom sequences, 2018. Blog post at www.extremelearning.com.au.

[Sob67] Sobol' I.: On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics 7*, 4 (1967), 86–112.