Multisampling

Last time we discussed how to use a few simple sampling patterns in our ray tracer to perform antialiasing. There is, however, a lot more to multisampling than antialiasing, mostly because ray tracing is fundamentally a sampling and reconstruction algorithm. So far we have just sampled pixels, but we can also sample other things, such as the lens of a camera we are using to simulate depth of field. Another example are soft shadows, that require sampling of the light sources to work. Other examples are global illumination, glossy reflections, etc.

The main idea is this: if you want to move beyond the very simple cases in ray tracing, you will require sampling to some degree (one could even argue to a very large degree) beyond what is required for antialiasing. We will be focusing on a couple of sampling techniques:

- Random,
- Jittered,
- N-Rooks,
- Multi-jittered,
- and Hammersley.

It is worth noting that the way that we will be discussing in regards to implementation is *not* by any stretch the best way to implement them. Instead of solving the more complex problem (but ultimately belonging to software engineering) of how to structure the code, we will be focusing on what the algorithms do and how one *could* go about implementing them. This would be particularly important if we wanted to have multiple sampling algorithms for a single render pass (which isn't uncommon).

Characteristics of Good Sampling

Before we start talking about the different sampling techniques, it is important to discuss what the properties of a *good* sampling technique are:

- First, we want the points to be uniformly distributed over the 2D unit square, so that clumping and gaps are minimized.
- Second, if we project the points in the x and y directions, the resulting projections should also be uniformly distributed. Uneven distributions can result in parts of the scene being either under or over sampled, which can increase aliasing.
- Third, we need some minimum distance between sample points for the same reasons.

As it turns out, of all the techniques we will be discussing, only Hammersley samples have all three, but it *still* has some properties that can lead to aliasing. For example, although we want *uniformly* distributed samples, we don't want the samples themselves to have regular spacing (i.e. the distance between samples in the x or y directions is the same). Hammersley samples are regularly spaced.

The main "take-away" of this discussion is that if we have enough samples per pixel, then the results between sampling techniques become negligible. The question then becomes: how *many* samples are enough? And what do we deem as *acceptable*? Well clearly the second question is subjective, but for most rendering applications it generally means that noise and aliasing should be at an acceptable level.

As an example, comparing the best sampling techniques to the worst techniques, the worst can require three times as many samples per pixel to reduce noise to an acceptable level. For example, under certain circumstances, random sampling may require 600 samples, but multi-jittered will only need 256. This is the reason why it is important to look at different sampling algorithms. Recall that ray tracing is ultimately an $O(n^2)$ algorithm and is by nature very slow. Now imagine taking hundreds or thousands of samples per pixel. Hence any time save is important.

Random Sampling

This is by far the simplest of the set of techniques we will cover, but it does fail all three requirements. The points can clump and leave gas in 2D, which also causes the projections to be badly distributed as well.

Jittered Sampling

Jittered samples are better distributed than random samples because they are *stratified*. What this means is that the domain (pixel) is divided a number of regions that cover the entire domain without gaps or overlapping. In our case, the domain is the unit square and the strata are the cells in the $n \times n$ sub-grid. Unlike random sampling, the samples are better distributed across the strata, and so their projections are also better. Note however that there is no minimum distance between the samples and their projections.

N-Rooks Sampling

A more even distribution can be obtained by using N-rooks sampling. In this technique, we place n samples in an $n \times n$ grid so that there's exactly one sample in each row and column. This is where the name comes from, since if we arrange n rooks in an $n \times n$ chess board with one rook in each row and column, they can't capture each other. The major difference between N-rooks and jittered sampling is that when we do jittered sampling on an $n \times n$ grid, we end up with n^2 samples and the number of samples has to be a perfect square, whereas n-rooks has n and the number of samples doesn't need to be a perfect square.

To generate this, first place the n samples along the diagonal of the grid. Then randomly shuffle the x and y coordinates while maintaining the n-rooks condition. Although the 1D projections are good, the overall 2D distribution is very similar to random sampling. In fact, a general characteristic of n-rook is that the 2D

distributions are no better than random sampling and therefore are worse than jittered sampling.

Multi-jittered Sampling

The idea of multi-jittered sampling is to improve the 2D distribution of n-rooks while still maintaining the even 1D projections. The idea is therefore to combine jittered with n-rooks. Let's explain how this works with an example.

Suppose we have a 16×16 grid, which we will refer to as the *sub-grid*. We will then superimpose a 4×4 grid. First, generate a single sample per each cell of the 4×4 grid. What we end up with is a jittered distribution with respect to the 4×4 grid, but it also satisfies the n-rooks condition on the sub-grid. We then shuffle the samples so that the n-rooks property is maintained on the sub-grid. The resulting sampling technique has the even projection of n-rooks but with the 2D distribution of jittered. Therefore multi-jittered sampling is an excellent sampling technique.

In the general case of n samples, where n must be a perfect square, multijittered samples are jittered on a $\sqrt{n} \times \sqrt{n}$ grid, and n-rooks on an $n \times n$ sub-grid.

Hammersley

The idea of Hammersley sampling is to use the computer representation of numbers in different prime bases, as opposed to a random distribution. We will concentrate on the binary representation only, as it yields the best sampling distributions. The general idea is the following: given an integer, convert it into binary. Now add the decimal point and reflect the digits across the decimal point. This function is called the *radical inverse function*. The generated samples all have regular spacing in their 1D projections since all samples are on an $n \times n$ grid in the unit square. The other advantage is that the sample points are well-distributed in 2D, with a minimum distance between samples, which is something that none of the other sampling methods have. The downside of Hammersley is that it can only produce *one* sequence, which leads to repeating patterns and therefore aliasing.

Index Shuffling

The last problem that we will cover in multisampling is the order in which we access our samples when tracing through a pixel. Currently, we would start on one corner and end at the opposite diagonal. This process would repeat for *every* pixel, which can yield aliasing effects as the samples are always the same. A simple solution would be to randomly select a sample point, but this does not guarantee that all of the points will be used. Another solution is to do a final shuffle of the samples, but this will affect n-rooks and multijittered samples. What we can do instead is to a shuffling of the *indices*. This will guarantee

that all of the samples are used, while still ensuring that the order in which the samples are used is not the same.