CHAPTER 21

THE ALIAS METHOD FOR SAMPLING DISCRETE DISTRIBUTIONS

Chris Wyman
NVIDIA

ABSTRACT

The *alias method* is a well-known algorithm for constant-time sampling from arbitrary, discrete probability distributions that relies on a simple precomputed lookup table. We found many have never learned about this method, so we briefly introduce the concept and show that such lookup tables can easily be generated.

21.1 INTRODUCTION

When rendering, we often need to sample discrete probability distributions. For example, we may want to sample from an arbitrary environment map proportional to the incoming intensity. One widely used technique inverts the cumulative distribution function [2]. This is ideal for analytically invertible functions but requires an $O(\log N)$ binary search when inverting tabulated distributions like our environment map.

If you can afford to precompute a lookup table, the alias method [5] provides a simple, constant-time algorithm for sampling from arbitrary discrete distributions. However, precomputation makes it less desirable if taking just a few samples from a distribution or if the distribution continually changes.

21.2 BASIC INTUITION

Let's start by reviewing how we often generate samples without the alias method. Imagine a discrete distribution, as in Figure 21-1. Bins have arbitrary real-valued weights, and we want to randomly sample proportional to their relative weights.

Using traditional cumulative distribution function (CDF) inversion, we first build a discrete CDF, e.g., in Figure 21-1. We pick a uniform random number

A. Marrs, P. Shirley, I. Wald (eds.), *Ray Tracing Gems II*, https://doi.org/10.1007/978-1-4842-7185-8_21

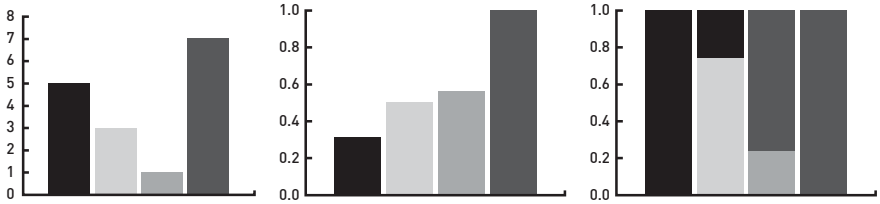

Figure 21-1. *Left: a simple discrete distribution. Center: its discrete cumulative distribution function. Right: an alias table for the distribution.*

$\xi \in [0 \ldots 1)$, place it on the *y*-axis, and select as our sample the first bin it falls into (moving left to right). Our example has weights summing to 16, so we choose the black bin if $\xi \leq \frac{5}{16}$, the light gray bin if $\frac{5}{16} < \xi \leq \frac{1}{2}$, etc. Here, finding a bin corresponding to ξ is straightforward, with at most three comparisons. But if sampling from a light probe, linearly searching each bin is costly. Even a fast binary search uses dozens of comparisons and walks memory incoherently.

Essentially, table inversion picks a random value then walks the tabulated CDF asking, "do you correspond to my sample?" Optimization involves designing a good search.

Instead, the alias method asks, "wouldn't it be easier with uniform weights?" You could easily pick a random bin without a search. Clearly, our weights are unequal. But imagine sampling bins as if uniformly weighted, say using the average weight. We would oversample those weighted below average and undersample those weighted above average.

Perhaps we could correct this error after selection? If we pick an oversampled bin, we could sometimes switch to an undersampled bin. This seems feasible; if sampling proportional to average weight, there's an equal amount of under- and oversampling.

Figure 21-1 also shows a table reorganizing weights from our distribution into equal-sized bins. We call this an *alias table* (though that term is a bit overloaded).

To sample, first choose a bin uniformly. Use random $\xi \in [0 \ldots 1)$ to check whether to switch due to oversampling, based on threshold τ. If $\xi \geq \tau$, switch to a different bin.

21.3 THE ALIAS METHOD

Now let's formalize this basic intuition. The alias method consumes N elements with positive weights ω_i. It samples from them, proportional to their relative weights, in constant time—two uniform random variates and one table lookup per sample.

A formal alias table is an array of N triplets $\{\tau, i, j\}$, where τ is a threshold in $[0 \ldots 1)$ and i and j are samples or sample indices. With care i may be stored implicitly, so that table entries are tuples $\{\tau, j\}$. Basic tables do not store or reconstruct weights ω_i.

To sample via the alias method, uniformly select $\xi_1, \xi_2 \in [0 \ldots 1)$. Use ξ_1 to pick from the N triplets, i.e., select triplet located at position $\lfloor \xi_1 N \rfloor$. Then, return i if $\xi_2 < \tau$, otherwise return j.

Interestingly, only two choices i and j are ever needed in any bin. In fact, proving this is the only tricky part of the alias method. This property arises *by construction* during table creation, though it subtly breaks the visual intuition developed in the previous section.

21.4 ALIAS TABLE CONSTRUCTION

Our goal is not efficient construction, but providing a clear and correct algorithm. Optimal $O(N)$ table construction algorithms are left for further reading. We give a simple divide-and-conquer approach similar to a proof by induction.

We start with our N input samples and their weights ω_i. We can easily compute the average sample weight $\langle\omega\rangle$; the summed weight is $\sum_{i \in N} \omega_i \equiv N\langle\omega\rangle$. Here is the idea for our simple construction: With a series of N steps, create one tuple $\{\tau, i, j\}$ per step. Each tuple has aggregate weight $\langle\omega\rangle$, so after one step we have $N - 1$ samples left to insert, with total weight $N\langle\omega\rangle - \langle\omega\rangle$ (still with an average of $\langle\omega\rangle$ per sample).

Each of these N steps looks as follows:

1. Sort the (remaining) input samples by weight.
2. Create a tuple $\{\tau, i, j\}$, where i is the lowest-weighted sample, τ is $\omega_i/\langle\omega\rangle$, and j is the highest-weighted sample.
3. Remove sample i from the list of remaining samples.
4. Reduce sample j's weight ω_j by $\langle\omega\rangle - \omega_i$ (i.e., $\omega_j = (\omega_j + \omega_i) - \langle\omega\rangle$).

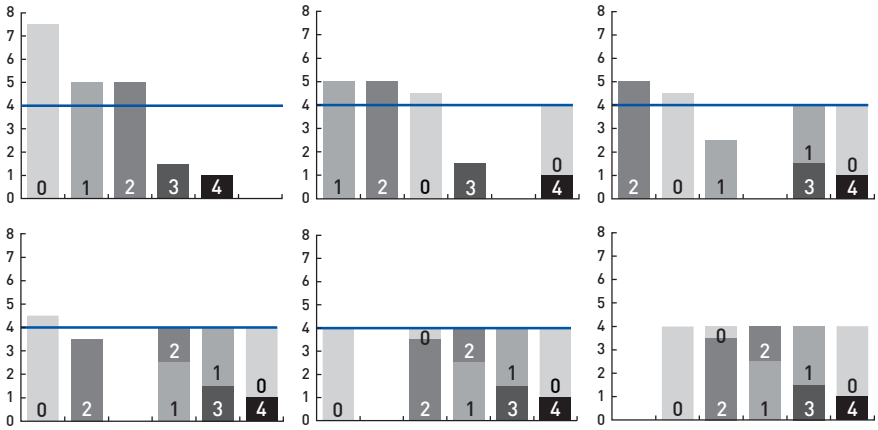

Figure 21-2. *The steps to build an alias table from the discrete distribution in the upper left. The blue line is the average weight.*

Due to the sort, the highest-weighted sample j has above average weight ($\omega_j \geq \langle\omega\rangle$) and the lowest-weighted sample i has $\omega_i \leq \langle\omega\rangle$. This ensures that $\tau \leq 1$ and that only two samples are needed to fill the tuple (because $\omega_i + \omega_j \geq \langle\omega\rangle$). It also ensures that, after creating $\{\tau, i, j\}$, all weight from sample i has been fully accounted for in the table. Sample j's weight gets reduced, as some is included in the newly created tuple.

To see why only two samples are needed per tuple, note that step 4 can reduce ω_j below $\langle\omega\rangle$. In other words, we do not "scrape excess" probability from high-weighted samples into low-weighted ones as if moving dirt with an excavator. We actively ensure that merging two samples gives a complete tuple—even if that leaves the formerly highest-weighted sample with a below-average weight.

Figure 21-2 demonstrates an alias table built from a slightly more complex distribution. Each step creates exactly one alias table tuple, requiring N steps for a distribution with N discrete values. The last step is trivial, as just one sample remains.

21.5 ADDITIONAL READING AND RESOURCES

As a well-known algorithm, many resources discuss the alias method. Many introductory statistics textbooks and Wikipedia provide good starting points. The original paper by Walker [5] is a fairly easy read, and Vose [4] provides an

$O(N)$ table build that observes that you can provide the guarantees outlined in Section 21.4 without any sorting. Devroye [1] presents a clear, more theoretical discussion. Schwarz [3] has an outstanding webpage with clear pseudocode and figures detailing the alias method, Vose's $O(N)$ build, and a discussion of numerical stability.

REFERENCES

[1] Devroye, L. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986. http://luc.devroye.org/rnbookindex.html.

[2] Pharr, M., Jacob, W., and Humphreys, G. *Physically Based Rendering: From Theory to Practice*. Morgan Kaufmann, 3rd edition, 2016. http://www.pbr-book.org/.

[3] Schwarz, K. Darts, dice, and coins: Sampling from a discrete distribution. https://www.keithschwarz.com/darts-dice-coins/, 2011. Accessed January 6, 2021.

[4] Vose, M. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, 1991. DOI: 10.1109/32.92917.

[5] Walker, A. J. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 3(3):253–256, 1977. DOI: 10.1145/355744.355749.