Fast Object Distribution (sap_0312)

Andrew Willmott*
Maxis, Electronic Arts

1 Introduction

On Spore, we have a need to procedurally distribute game objects, subject to a number of constraints. As a starting point, the objects should positioned in a way that seems visually random, but be well spaced, as in a poisson disc distribution. There are other desirable characteristics, having multiple object types that don't overlap; naturally varying object scale, orientation and colour; control over the density of objects from some form of map, be it procedural, game-generated, or pre-authored. The problem domain is similar to that of [Ostromoukhov et al. 2004] and related papers, although our performance constraints are tighter.

2 Incremental Halton Sequence

Our solution to this problem is to use an incremental version of the pseudo-random Halton sequence [Halton 1964] for distributing points in 2D or 3D. A major advantage for game use is that this is memoryless: generating a list of samples does not require storage for tables, or for intermediate data structures necessary to ensure spacing. (The Halton sequence by nature fills in between previous samples as the count increases.) Memory access is slow on modern day PCs and even more so on consoles, so this is key. Moreover, it is repeatable, so we have the option of not permanently storing object positions after laying out an area, but simply regenerating them from scratch each time.

Calculating a point in a standard Halton sequence requires taking the sequence index, writing it as both base two, three and (for 3D) five numbers, and then digit reversing those numbers. This operation is $\log_b(x)$, and requires performing divides in the inner loop. While not expensive for offline generation, doing this at run time for a large number of samples is prohibitive. Thus we reformulated the sequence as an incremental calculation, which keeps base 3 and 5 versions of the count in binary-coded form as state. The amortized cost of finding the next point in the sequence then becomes a factor of the expected number of carries in each base on an increment, which is $b/(b-1)$, small enough to be fast enough, as the routine requires only adds, multiplies, and shifts.

3 Varying other Attributes, Composition

Although the sequence H_i meets the positioning criteria mentioned above, we would like to have the same characteristics for other areas. A novel way of doing this is to use i/N to index a small table for the attribute involved. This has the effect of ensuring that similarly-coloured objects are maximally distant from each other, which tends to lead to visually pleasing variation. An example of this approach vs. purely random variation can be seen in Figure 1. Because table lookups are coupled, it is easy for distribution to encompass small objects with one colour range and larger ones with another. We also allow truly random variation of attributes, however, this has been largely unneeded.

We take this idea further by providing art control over the sequence range used. This lets several different kinds of objects be distributed over an area without having to worry about object collisions. For example, different species of plants may be nested amongst each other simply by ordering them one after the other.

*e-mail: awillmott@maxis.com

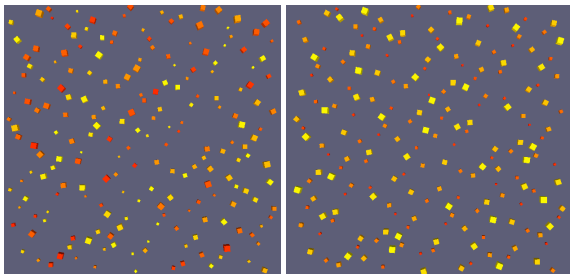

Figure 1: *Left, random variation, right, indexed variation*

4 Distribution Density

A straightforward approach to modifying the density of our distribution is rejection sampling. Assume we have some density map $D(x)$, which is 0..1 over the domain. We generate samples as usual, but discard any sample such that $D(H_i) < i/N$. This has the effect of locally reducing the number of in-fill samples where D is small. Results can be seen in Figure 2.

This can be wasteful when the integral of D is small. However because our sample generation is fast, in practice it is usually more efficient than less brute force methods, especially if D is varying over time. When a density map is extremely sparse, samples can be generated more efficiently by subdividing the region into tiles, and finding the maximum value of D within each tile t. For each tile, we choose N_t proportional to D^t_{max}, and then perform rejection sampling using $D^t(x) = D(x)/D^t_{max}$.

A drawback to this approach is that the sample pattern repeats within tiles. A higher quality approach would be to find an efficient way to calculate a windowed Halton sequence, i.e., all consecutive points H_i such that $H_i \in (s0, s1, t0, t1)$. This has not so far proved possible under our performance constraints.

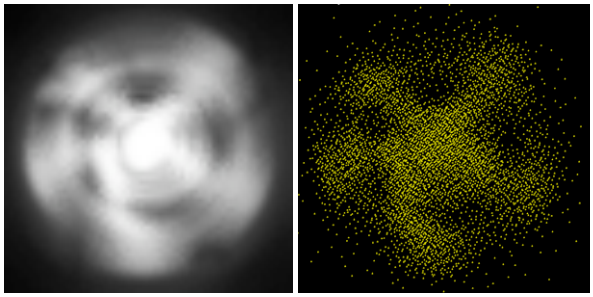

Figure 2: *Density control by rejection sampling*

References

HALTON, J. H. 1964. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM 7*, 12, 701–702.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM Press, New York, NY, USA, 488–495.