Random Numbers for Computer Graphics

Mayur Patel*
Digital Domain

Random numbers are critical in computer graphics, from sampling to aesthetics. Often times, the computer graphics artist needs to generate randomness in a repeatable way otherwise the results will vary between executions without control. Some state of the application context must act as a seed for random sequences. Consider the following situations:

- Generating sample positions in a pixel using its image coordinate as a seed;
- Driving agent behavior with a random sequence that is seeded from its unique identification attribute;
- Generating values for lattice noise calculations using the lattice corner coordinates as keys.

Since the artist may be dealing with many entities that require random values, the values for each entity should be generated independently. Ideally, one could create many instances of a random source and attach one to each entity. This would allow for behaviors to remain highly repeatable so that local changes to some entities do not affect others.

We focus on hashing functions as random sources because they are applicable in the widest variety of circumstances. When producing sequences of random values associated with an entity, an integer counter can be attached to the entity. The concatenation of the entity's relevant state variables with its counter produces a key which can be hashed. The counter would increment after each random value calculation. When compared to pseudo-random number generators (PRNGs), hashing allows the developer to control state size. Modern high-quality PRNGs often have large fixed-size states, making it prohibitive for very many instances to coexist in memory.

We found many popular hashing functions to be inadequate for computer graphics. Some very fast hashing functions are sufficiently random to avoid collisions in a hash table, but still produce artifacts when used for lattice noises [Uzgalis and Tong 1994; Noll]. Jenkins' hash provides a high quality of randomness, but its speed is not exceptional [1997].

Inspired by the table-based techniques used in Buzhash and Pearson's algorithm [1990], we developed a hashing function suitable for computer graphics, which we call Goulburn.

Goulburn passes the Diehard statistical tests for sequences produced using 3D Hilbert curve coordinates as keys. Using Goulburn as a PRNG, incrementing a large integer key, also produced good results. We believe these tests are representative of the access patterns developers observe in computer graphics.

We implemented Goulburn as a PRNG with a 64-bit state and an optimization costing an extra 4 bytes of storage. When compared to Wagner's implementation of Mersenne Twister [Wagner 2003; Matsumoto and Nishimura 1998], which is generally considered to be the fastest PRNG of very high quality, it ran 15% slower on an AMD processor and 24% faster on an Intel processor.

Each random source has its strengths and weaknesses, but Goulburn has been developed to be well-suited to the demands of computer graphics. It provides flexibility and good randomness at a fair computational cost.

References

JENKINS, R. September 1997. Algorithm Alley: Hashing Functions. *Dr. Dobb's Journal*.

MATSUMOTO M., AND NISHIMURA T. 1998. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, *ACM Transactions on Modeling and Computer Simulation 8,* 1. 3-30.

NOLL, L. C. *Fowler / Noll / Vo (FNV) Hash*. http://www.isthe.com/chongo/tech/comp/fnv/

PEARSON, P. June 1990. Fast Hashing of Variable-Length Text Strings. *Communications of the ACM, 33,* 6. 677-680.

UZGALIS, R. AND TONG, M. 1994. *Hashing Myths*. Technical Report 97, Department of Computer Science University of Auckland.

WAGNER, R. May 2003. *Mersenne Twister Random Number Generator*. http://www-personal.engin.umich.edu/~wagnerr/MersenneTwister.html.

Appendix

```
// table0[] as used in buzhash
// each int in table1[] has 16 0s and 16 1s
unsigned long
goulburn ( unsigned char *cp, unsigned len )
{
    register unsigned long h = 0;
    register unsigned u;
    for( u=0; u<len; ++u )
    {
        h += table0[ cp[u] ];
        h ^= (h << 3) ^ (h >> 29);
        h += table1[ h >> 25 ];
        h ^= (h << 14) ^ (h >> 18);
        h += 1783936964;
    }
    return h ;
}
```

* email: patelm@acm.org

ACM SIGGRAPH 2006
Sketches and Applications
August 3 2006