PBRT: Appendix A

Friday, December 18, 2020 7:40 PM

A.1.2 Pseudorandom numbers

PBRT uses a ***permuted congruential generator*** (PCG) random number generator.

PCG builds upon the ***linear congruential generator*** random number generator, which is defined as a ***recurrence relation***:

$$X_{n+1} = (aX_n + c)\ mod\ m$$

where X is the sequence, $0 < m$ is the modulus, $0 < a < m$ is the multiplier, $0 \leq c < m$ is the increment, and $0 \leq X_0 < m$ is the seed.

PCG takes the output of the LCG RNG and applies a sequence of transformations to it before returning it: *random rotation (RR), random shift (RS), xorshift (XSH), multiplication by a fixed constant (M),* and others.

The sequence of output transformations give the name to the particular PCG algorithm (PCG is actually a family): *XSH-RR, XSH-RS,* etc.

A.2 Image file input and output

There are Image file formats out there that represent color components as 8-bit integers and there are others that represent them as floating-point numbers.

The RGB spectral pixel values computed by PBRT (the result of sampling and filtering) are floating-point values, so we are interested in floating-point image file formats.

The reason 8-bit integer formats are not adequate is that they ***don't have a large dynamic range***. PBRT generates images with a large dynamic range.

OpenEXR (by ILM) and PFM (an extension to PPM) are the 2 floating-point image file formats supported by PBRT. Unlike OpenEXR, PFM doesn't support compression and there aren't as many libraries that read and write it.

PBRT also supports TGA and PNG for reading and writing; these formats are convenient for low-dynamic range texture maps. None of them is a high-dynamic-range format.

A.4.1 Variable stack allocation

Dynamic allocation for memory of variable length with stack frame scope is usually done with $new/delete$ or $malloc/free$. For a small amount of memory, the overhead of these operators and functions is too high.

PBRT uses the C library's $alloca$ function, which allocates memory of variable length in the ***stack frame of the caller*** and releases it automatically when the call returns.

A.6.3 Atomic floating-point values

$std :: atomic$ didn't support floating-point numbers before C++20 because floating point addition and other operations are ***not commutative***: $(a + b) + c$ is not necessaritly equal to $a + (b + c)$.

What PBRT does is use the AtomicFloat class to store the atomic bit representation (std::atomic<uint32_t>) of the floating-point number and use that for safe concurrent access.

A.4.2 Cache-friendly memory usage

A CPU has to wait a hundred or so cycles to read from main memory.

A stall is when the CPU is idle waiting for memory or IO.

Cache access is fast because its on the CPU itself. That's also why they are small.

Algorithms and data structures need to be designed to make good use of cache.

Cache lines. Cache associativity. Compulsory cache miss. Capacity cache miss. Conflict cache miss.

See Memory note in Evernote.

A.4.3 Arena-based allocation

Many researchers have shown that *"the performance impact of memory allocation in real-world applications and found that custom allocators almost always result in worse performance than a well-tuned generic system memory allocation (like malloc and new), in both execution time and memory use".*

Arena-based allocation is a custom allocation technique that is useful, though:

- Objects are allocated quickly from a large pre-allocated contiguous region of memory. Allocation typically involves only a pointer increment; the new address is handed out to the caller.
- Improves locality of reference and leads to fewer cache misses because the allocated objects are contiguous in memory.
- Variable-size allocation.
- Individual objects are never explicitly free (no API for freeing individual objects). The entire region of memory is freed when the lifetime of all the allocated objects end.

A.5 Mathematical routines

A.5.2 2 x 2 linear systems

PBRT solves $2\ x\ 2$ linear systems using ***Cramer's rule*****.**

THEOREM 7 Cramer's Rule

Let A be an invertible $n \times n$ matrix. For any $\mathbf{b}$ in $\mathbb{R}^n$, the unique solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$ has entries given by

$$x_i = \frac{\det A_i(\mathbf{b})}{\det A}, \quad i = 1, 2, \ldots, n \qquad (1)$$