Computer Graphics
Low-discrepancy Point Distributions

Dr. Marco Manzi
Today’s Menu

• Better random variables using
  • Stratified Sampling
  • Quasi Monte Carlo
Careful Sample Placement

Random

Stratified

N-Rooks

Multi-Jittered

Quasi-Random

Poisson-Disc
Random Sampling
Stratified (Jittered) Sampling
Monte Carlo (16 random samples)
Monte Carlo (16 stratified samples)
Stratifying in Higher Dimensions

• Stratification requires $N^d$ samples.
• Inconvenient for large $d$
• Compute stratified samples in lower-dimensions (e.g. $(x, y)$, $t$, $(u, v)$), and randomly combine
Depth-of-Field (4D)

Reference

Random Sampling

Stratified Sampling
Stratified (Jittered) Sampling
Latin Hypercube (N-Rooks) Sampling
Latin Hypercube (N-Rooks) Sampling

Shuffle rows
Initialize
Latin Hypercube (N-Rooks) Sampling

Shuffle rows

Shuffle columns
Latin Hypercube (N-Rooks) Sampling

Shuffle columns
Latin-Hypercube Sampling

// initialize the diagonal
for (uint j = 0; j < numDimensions; j++)
    for (uint i = 0; i < numSamples; i++)
        samples[j][i] = (i + randf())/numSamples;

// shuffle each dimension independently
for (uint j = 0; j < numDimensions; j++)
    for (uint i = numSamples-1; i >= 1; i--)
        swap(samples[j][i], samples[j][randi(0,i)]);
Soft Shadows
Soft Shadows: Sample Allocation

1 image sample x 16 light samples

16 image samples x 1 light sample
Multi-Jittered Sampling

  - combine N-Rooks and Jittered stratification constraints
Multi-Jittered Sampling
Latin-Hypercube Sampling

// initialize

float cellSize = 1.0 / (resX*resY);

for (uint i = 0; i < resX; i++)
    for (uint j = 0; j < resY; j++)
        {
            samples(i,j).x = i/resX + (j+randf()) / (resX*resY);
            samples(i,j).y = j/resY + (i+randf()) / (resX*resY);
        }

// shuffle x coordinates within each column of cells

for (uint i = 0; i < resX; i++)
    for (uint j = resY-1; j >= 1; j--)
        swap(samples(i, j).x, samples(i, randi(0, j)).x);

// shuffle y coordinates within each row of cells

for (unsigned j = 0; j < resY; j++)
    for (unsigned i = resX-1; i >= 1; i--)
        swap(samples(i, j).y, samples(randi(0, i), j).y);
Multi-Jittered Sampling

Shuffle x-coords
Multi-Jittered Sampling

Shuffle x-coords
Multi-Jittered Sampling

Shuffle x-coords
Multi-Jittered Sampling

Shuffle x-coords
Multi-Jittered Sampling

Shuffle x-coords

Shuffle y-coords
Multi-Jittered Sampling

Shuffle y-coords
Multi-Jittered Sampling

Shuffle y-coords
Multi-Jittered Sampling

Shuffle y-coords
Multi-Jittered Sampling

Shuffle y-coords
Visual Break
Poisson-Disk/Blue-Noise Sampling

• Enforce a minimum distance between points

• Poisson-Disk Sampling:
Random Dart Throwing
Random Dart Throwing
Random Dart Throwing
Stratified Sampling
Best Candidate Sampling
Power Spectrum

Figure 1:

- The periodogram of a Poisson disk distribution is radially symmetric. Therefore, two one-dimensional statistics are derived from the periodogram. The first one is the averaged power spectrum associated with a specific method for generating Poisson disk distributions, while the power spectrum is flat. This ensures that the periodogram covers a domain corresponding to twice the principal frequency of a Poisson disk distribution with a relative radius of 0.75. In both cases, the anisotropy is low (close to 0 dB), indicating good radial symmetry. A reference line should be considered for back-grounds.

Figure 2 shows the typical power spectrum and radially averaged power spectrum, which is surrounded by an annulus of low energy, followed by a sharp transition region, a low-frequency cutoff at the principal frequency (at 1/2), and a flatter high-frequency region. As in Ulichney's work, the spectral estimates in this paper were obtained by averaging 10 periodograms. There-fore, an anisotropy of one sample.

All spectral estimates in this paper were obtained by averaging a single Poisson disk distribution, while the power spectrum is obtained with variance $P^2_N r_i f_i^r N^{-1} \sum_{i=1}^{N} f_i r_i f_i^r (f_c - \delta_1 f_c) - \sum_{i=1}^{N} f_i r_i f_i^r$. Dirac's delta is $\delta_1 f_c$ with variance $P^2_N r_i f_i^r N^{-1} \sum_{i=1}^{N} f_i r_i f_i^r$. The corresponding radially averaged power spectrum is normalized. The power spectrum was normalized. The radially averaged power spectrum was normalized. The power spectrum was normalized. All power spectrum images were tone mapped with a logarithmic tone mapper, using the same settings for all images.
Visual Break
Discrepancy

• Previous stratified approaches try to minimize “clumping”

• “Discrepancy” is another possible formal definition of clumping: $D^*(x_1, \ldots, x_n)$
  – for every possible subregion compute the maximum absolute difference between:
    • fraction of points in the subregion
    • (relative) volume of the subregion
Discrepancy
Discrepancy
Discrepancy
Discrepancy
Discrepancy
Low-Discrepancy Sampling

• Low-Discrepancy sequences are specially crafted to have small discrepancy values.

• Entire field of study called Quasi-Monte Carlo (QMC)
Koksma-Hlawka inequality

- Discrepancy gives an upper bound on the estimation error:

\[
\left| \frac{1}{n} \sum_{i=1}^{n} f(x_i) - \int f(u) \, du \right| \leq V(f)D^*(x_1, \ldots, x_n)
\]

Error \hspace{1cm} \text{Variance} \ast \text{discrepancy}
The Radical Inverse

• A positive integer value $n$ can be expressed in a base $b$ with a sequence of digits $d_m\ldots d_2d_1$ uniquely determined by:

$$n = \sum_{i=1}^{\infty} d_i b^{i-1}$$

• The radical inverse function $\Phi_b$ in base $b$ converts a nonnegative integer $n$ to a floating-point value in $[0, 1)$ by reflecting these digits about the decimal point:

$$\Phi_b(n) = 0.d_1d_2\ldots d_m$$

• Subsequent points “fall into biggest holes”
The Radical Inverse

```c
float radicalInverse(int n, int base, float inv)
{
    float v = 0.0f;
    for (float p = inv; n != 0; p *= inv, n /= base)
        v += (n % base) * p;
    return v;
}
```

```c
float radicalInverse(int n, int base)
{
    return radicalInverse(n, base, 1.0f / base);
}
```

More efficient version available for base 2
The Van der Corput Sequence

- Radical Inverse in base two

<table>
<thead>
<tr>
<th>n</th>
<th>Base 2</th>
<th>$\Phi_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.1 = 1/2</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>.01 = 1/4</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>.11 = 3/4</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>.001 = 1/8</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>.101 = 5/8</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>.011 = 3/8</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>.111 = 7/8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Discrepancy: $D^*(x_1, \ldots, x_n) \in O\left(\frac{\log N}{N}\right)$
The Radical Inverse (Base 2)

```c
float vanDerCorputRIU(uint n)
{
    n = (n << 16) | (n >> 16);
    n = ((n & 0x00ff00ff) << 8) | ((n & 0xff00ff00) >> 8);
    n = ((n & 0x0f0f0f0f) << 4) | ((n & 0xf0f0f0f0) >> 4);
    n = ((n & 0x33333333) << 2) | ((n & 0xcccccccc) >> 2);
    n = ((n & 0x55555555) << 1) | ((n & 0xaaaaaaaa) >> 1);
    return n / float (0x100000000LL);
}
```
The Halton Sequence

- An n-dimensional Halton sequence uses the radical inverse with a different base $b$ for each dimension of the pattern.

- The bases must all be relatively prime.

$$x_i = (\Phi_2(i), \Phi_3(i), \Phi_5(i), \ldots, \Phi_{p_n}(i))$$

- Progressive generation of samples
Scrambling

Without scrambling

With scrambling

\[ \Phi_b(n) = \pi(d_1) \pi(d_2) \ldots \pi(d_m) \]
Monte Carlo (16 random samples)
Monte Carlo (16 stratified samples)
Quasi-Monte Carlo (16 Halton samples)
The Hammersley Sequence

• Same as Halton, but uses $i/N$ for first dimension:

$$x_i = \left( \frac{i}{N}, \Phi_2(i), \Phi_3(i), \ldots, \Phi_{p_n}(i) \right)$$

• Provides slightly lower discrepancy

• Not incremental, need to know total number of samples, $N$, in advance
(0,2)-Sequences

1 sample in each “elementary interval”
(0,2)-Sequences

1 sample in each “elementary interval”
(0,2)-Sequences

1 sample in each “elementary interval”
(0,2)-Sequences

1 sample in each “elementary interval”
(0,2)-Sequences

1 sample in each “elementary interval”
(0,2)-Sequences

1 sample in each “elementary interval”
Enumerating Quasi-Monte Carlo Point Sequences in Elementary Intervals
L. Grüenschloß, M. Raab and A. Keller
Monte Carlo and Quasi-Monte Carlo Methods 2010

http://gruenschloss.org/
Many more...

- Sobol
- Faure
- Larcher-Pillichshammer
- Folded Radical Inverse
- (t,s)-sequences & (t,m,s)-nets
- Scrambling/randomization
- much more...
Randomized/Scrambled Sequences

• LD sequence identical for multiple runs
  – cannot average independent images!
  – no “random” seed

• Random permutations: compute a permutation table for the order of the digits and use it when computing the radical inverse
  – Can be done very efficiently for base 2 with XOR operation

• See PBR Book Chapter 7 for details
Scrambled Radical Inverse (Base 2)

float vanDerCorputRIU(uint n, uint scramble = 0)
{
    n = (n << 16) | (n >> 16);
    n = ((n & 0x00ff00ff) << 8) | ((n & 0xff00ff00) >> 8);
    n = ((n & 0x0f0f0f0f) << 4) | ((n & 0xf0f0f0f0) >> 4);
    n = ((n & 0x33333333) << 2) | ((n & 0xcccccccc) >> 2);
    n = ((n & 0x55555555) << 1) | ((n & 0xaaaaaaaa) >> 1);
    n ^= scramble;
    return n / float (0x100000000LL);
}
Monte Carlo (16 random samples)
Monte Carlo (16 stratified samples)
Quasi-Monte Carlo (16 Halton samples)
Scrambled Quasi-Monte Carlo

scrambled Larcher-Pillichshammer sequence
Implementation tips

• Using QMC can often lead to unintuitive, difficult to debug, problems.
  – Always code up MC algorithms first, using random numbers, to ensure correctness
  – Only after confirming correctness, slowly incorporate QMC into the mix
Questions?