◊ V.4

Multi-Jittered Sampling

Kenneth Chiu
215 Lindley Hall
Indiana University
Bloomington, IN 47405
chiuk@cs.indiana.edu

Peter Shirley
Computer Science Department
Lindley Hall
Indiana University
Bloomington, IN 47405
shirley@iuvax.cs.indiana.edu

Changyaw Wang
Indiana University
Bloomington, IN 47405
wangc@iuvax.cs.indiana.edu

◊ Abstract ◊

Jittered sampling patterns perform better than random sampling patterns because they limit the degree of clumping that can occur. Clumping can still be present in one-dimensional projections of jittered patterns, however. We present a simple method to reduce the clumping of the X-axis and Y-axis projections by imposing an additional N-rooks constraint on the jittered sampling pattern. The resulting sampling pattern can reduce the number of rays necessary for a satisfactory image when ray-tracing.

◊ Introduction ◊

Monte-Carlo integration is often used to compute pixel values. For every pixel, a set of sample points is generated. The radiances of each point are then computed and averaged. To avoid aliasing, the sample points must be generated randomly such that every point is equally likely to be selected. However, a sampling pattern that is "too random" can overemphasize some parts of the pixel, resulting in excessive noise unless the number of samples is very large.

This variance can be reduced by selecting a sampling pattern that is well distributed. Intuitively, by "well distributed" we mean uniform and without clumping. Determining exactly what kinds of uniformity are important is difficult, though, especially since some patterns work well on some classes of image functions but not others.

The usual sampling pattern in computer graphics is *jittered* (Figure 1) (Cook *et al.* 1984). The pixel is divided into an $n \times n$ rectangular grid of cells. One sample point is randomly selected from within each cell, yielding $N = n^2$ samples. This method performs well in practice. Some clumping can occur, but the severity is limited.

IBM ISBN 0-12-336155-9
Macintosh ISBN 0-12-336156-7

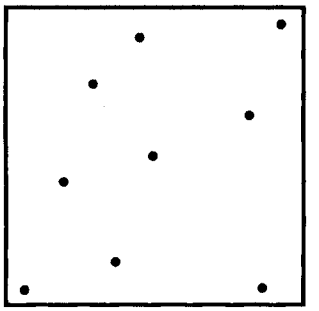


Figure 1. Jittered sampling for $n = 3$, $N = 9$.

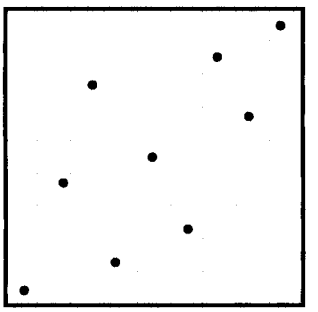
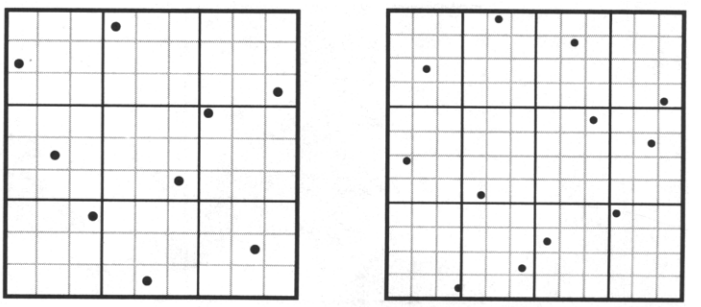


Figure 2. N-rooks sampling for $N = 9$.

The jittered sampling pattern is well distributed in two dimensions. When projected onto the X-axis or Y-axis, however, the clumping can be greatly increased. For an $n \times n$ grid, up to $2n$ sample points may project to essentially the same point on the X-axis. The poor quality of the projection is not important for some images, but for something like a vertical edge, the variance will increase.

To improve the distribution of the projected sampling pattern, Shirley (Shirley 1991) suggested an *N-rooks* pattern (Figure 2). The pixel is divided into an $N \times N$ rectangular grid of cells. N cells are then randomly selected with the constraint that no two selected cells may be in the same row or column. Within each selected cell, a sample point is randomly selected.

This method really amounts to jittering separately in each dimension and is well distributed in one dimension when projected to the X-axis or Y-axis, but not in the two dimensions combined. Consequently, it works better than jittering for horizontal and vertical edges, but for other images it usually performs no better when the number of samples is small, and considerably worse than jittering when the number of samples is large (Mitchell 1992).



(a) (b)

Figure 3. (a) Multi-jittered sampling for $n = 3$, $N = 9$. (b) Multi-jittered sampling for $m = 4$, $n = 3$, $N = 12$.

♢ Multi-Jittering ♢

To minimize the two-dimensional variations, yet still preserve the projected distribution, we can combine the two methods (Figure 3a). The new pattern appears jittered when viewed separately in each one-dimensional projection, and also when viewed in two dimensions. For $N = n^2$ samples, an $n \times n$ grid of cells is superimposed over an $N \times N$ subgrid of subcells. So each cell is an $n \times n$ array of subcells. One sample point is then selected from each cell as in jittered sampling, but simultaneously the N-rooks condition is also satisfied on the subgrid. The resulting sampling pattern is a jittered pattern relative to the $n \times n$ grid, and an N-rooks pattern relative to the $N \times N$ subgrid. It is well distributed in the one-dimensional projections and in two dimensions.

If the number of samples per pixel is not a perfect square, this method can still be used, as long as two acceptable factors can be found (Figure 3b). For $N = mn$ samples, an $m \times n$ grid of cells is superimposed over an $N \times N$ subgrid of subcells. So each cell is an $n \times m$ array of subcells. One sample point is again selected from each cell while simultaneously observing the N-rooks condition on the subgrid.

This method has performed better than either jittering or N-rooks on our test images regardless of feature orientation. This is probably because imposing the N-rooks condition on the jittered pattern improves not only the one-dimensional projected distributions, but also the two-dimensional distribution.

♢ C Code ♢

The code below fills a $[0, 1] \times [0, 1]$ square with an $m \times n$ multi-jittered pattern. The points are placed into an array that is passed in as an argument. The "canonical" multi-jittered pattern mentioned in the comment is shown in Figure 4.

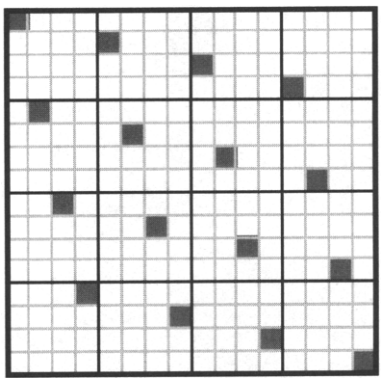


Figure 4. Canonical multi-jittered pattern. One point is chosen inside each shaded square.

```
#include <math.h>


#define RAN_DOUBLE(l, h)     (((double) random()/0x80000000U)*((h) - (l)) + (l))
#define RAN_INT(l, h)        ((int) (RAN_DOUBLE(0, (h)-(l)+1) + (l)))


typedef struct {
    double x, y;
} Point2;


unsigned long random();      /* expected to return a random int in [0, 2^31-1] */


/*
 * MultiJitter() takes an array of Point2's and the dimension, and fills the
 * the array with the generated sample points.
 *
 *    p[] must have length m*n.
 *    m is the number of columns of cells.
 *    n is the number of rows of cells.
 */
void
MultiJitter(Point2 p[], int m, int n) {

    double subcell_width;
    int i, j;

    subcell_width = 1.0/(m*n);

    /* Initialize points to the "canonical" multi-jittered pattern. */
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
```

```
            p[i*n + j].x = i*n*subcell_width + j*subcell_width
             + RAN_DOUBLE(0, subcell_width);
            p[i*n + j].y = j*m*subcell_width + i*subcell_width
             + RAN_DOUBLE(0, subcell_width);
        }
    }

    /* Shuffle x coordinates within each column of cells. */
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {

            double t;
            int k;

            k = RAN_INT(j, n - 1);
            t = p[i*n + j].x;
            p[i*n + j].x = p[i*n + k].x;
            p[i*n + k].x = t;
        }
    }

    /* Shuffle y coordinates within each row of cells. */
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {

            double t;
            int k;

            k = RAN_INT(j, m - 1);
            t = p[j*n + i].y;
            p[j*n + i].y = p[k*n + i].y;
            p[k*n + i].y = t;
        }
    }
}
```

◊ Bibliography ◊

(Cook *et al.* 1984) Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, 1984.

(Mitchell 1992) Don Mitchell. Ray tracing and irregularities of distribution. In *Proceedings of Third Eurographics Workshop on Rendering*, pages 61–69, 1992.

(Shirley 1991) Peter Shirley. Discrepancy as a quality measure for sampling distributions. In *Eurographics '91*, pages 183–193, September 1991.