Chapter 11

SAMPLING AND QUANTIZATION ARTIFACTS

From the information or signal processing point of view, modeling can be regarded as the definition of the intended world by digital and discrete data which are processed later by image synthesis. Since the intended world model, like the real world, is continuous, modeling always involves an analog-digital conversion to the internal representation of the digital computer. Later in image synthesis, the digital model is resampled and requantized to meet the requirements of the display hardware, which is much more drastic than the sampling of modeling, making this step responsible for the generation of artifacts due to the approximation error in the sampling process. In this chapter, the problems of discrete sampling will be discussed first, then the issue of quantization will be addressed.

The sampling of a two-dimensional color distribution, $I(x, y)$, can be described mathematically as a multiplication by a "**comb function**" which keeps the value of the sampled function in the sampling points, but makes it zero elsewhere:

$$I_s(x, y) = I(x, y) \cdot \sum_i \sum_j \delta(x - i \cdot \Delta x, y - j \cdot \Delta y). \tag{11.1}$$

The 2D Fourier transformation of this signal is:

$$I_s^*(\alpha,\beta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_s(x,y)\cdot e^{-\jmath x\alpha}\cdot e^{-\jmath y\beta}\,dx\,dy =$$

$$\frac{1}{\Delta x\cdot\Delta y}\sum_i\sum_j I^*(\alpha-\frac{2\pi i}{\Delta x},\beta-\frac{2\pi j}{\Delta y}). \qquad (11.2)$$

The sampling would be correct if the requirements of the **sampling theorem** could be met. The sampling theorem states that a continuous signal can be reconstructed exactly from its samples by an ideal low-pass filter only if it is band-limited to a maximum frequency which is less than half of the sampling frequency. That is also obvious from equation 11.2, since it repeats the spectrum of the signal infinitely by periods $2\pi/\Delta x$ and $2\pi/\Delta y$, which means that the spectrum of non-band-limited signals will be destroyed by their repeated copies.

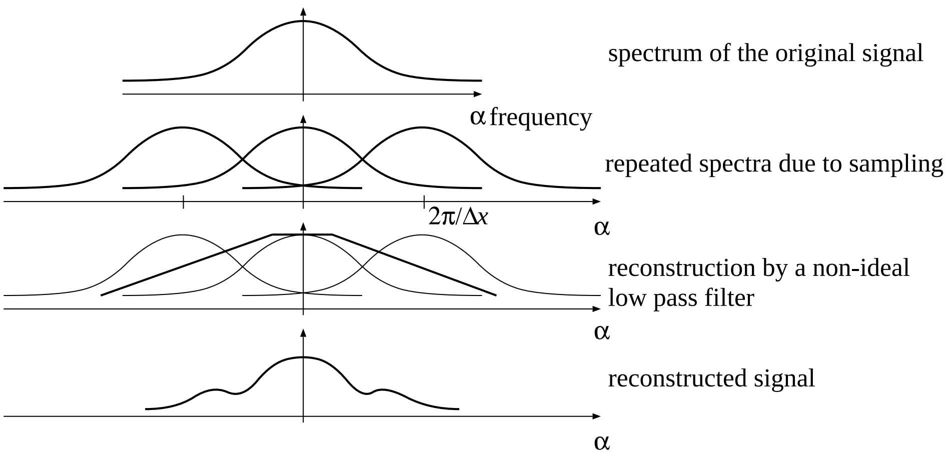

Figure 11.1: Analysis of the spectrum of the sampled color distribution

The real world is never band-limited, because objects appear suddenly (like step functions) as we move along a path, introducing infinitely high frequencies. Thus, the sampling theorem can never be satisfied in computer graphics, causing the repeated spectra of the color distribution to overlap, and destroying even the low frequency components of the final spectrum.

The phenomenon of the appearance of high frequency components in the lower frequency ranges due to incorrect sampling is called **aliasing** (figure 11.1). The situation is made even worse by the method of reconstruction of the continuous signal. As has been discussed, raster systems use a 0-order hold circuit in their D/A converter to generate a continuous signal, which is far from being an ideal low-pass filter.

The results of the unsatisfactory sampling and reconstruction are well-known in computer graphics. Polygon edges will have stairsteps or jaggies which are aliases of unwanted high frequency components. The sharp corners of the jaggies are caused by the inadequate low-pass filter not suppressing those higher components. The situation is even worse for small objects of a size comparable with pixels, such as small characters or textures, because they can totally disappear from the screen depending on the actual sampling grid. In order to reduce the irritating effects of aliasing, three different approaches can be taken:

1. Increasing the resolution of the displays. This approach has not only clear technological constraints, but has proven inefficient for eliminating the effects of aliasing, since the human eye is very sensitive to the regular patterns that aliasing causes.

2. Band-limiting the image by applying a low pass filter before sampling. Although the high frequency behavior of the image will not be accurate, at least the more important low frequency range will not be destroyed by aliases. This **anti-aliasing** approach is called **pre-filtering**, since the filtering is done before the sampling.

3. The method which filters the generated image after sampling is called **post-filtering**. Since the signal being sampled is not band-limited, the aliases will inevitably occur on the image, and cannot be removed by a late filtering process. If the sampling uses the resolution of the final image, the same aliases occur, and post-filtering can only reduce the sharp edges of jaggies, improving the reconstruction process. The filtering cannot make a distinction between aliases and normal image patterns, causing a decrease in the sharpness of the picture. Thus, post-filtering is only effective if it is combined with higher resolution sampling, called **supersampling**, because the higher sampling frequency will reduce the inevitable aliasing if the spectrum energy

falls off with increasing frequency, since higher sampling frequency increases the periods of repetition of the spectrum by factors $2\pi/\Delta x$ and $2\pi/\Delta y$. Supersampling generates the image at a higher resolution than is needed by the display hardware, the final image is then produced by sophisticated digital filtering methods which produce the filtered image at the required resolution.

Comparing the last two basic approaches, we can conclude that prefiltering works in continuous space allowing for the elimination of aliases in theory, but that efficiency considerations usually inhibit the use of accurate low-pass filters in practice. Post-filtering, on the other hand, samples the non-band-limited signal at a higher sampling rate and reduces, but does not eliminate aliasing, and allows a fairly sophisticated and accurate low-pass filter to be used for the reconstruction of the continuous image at the normal resolution.

11.1 Low-pass filtering

According to the **sampling theorem**, the **Nyquist limits** of a 2D signal sampled at $\Delta x, \Delta y$ periodicity are $\pi/\Delta x$ and $\pi/\Delta y$ respectively, requiring a low-pass filter suppressing all frequency components above the Nyquist limits, but leaving those below the cut-off point untouched. The filter function in the frequency domain is:

$$F(\alpha,\beta) = \begin{cases} 1 \text{ if } |\alpha| < \pi/\Delta x \text{ and } |\beta| < \pi/\Delta y \\ \\ 0 \text{ otherwise} \end{cases} \tag{11.3}$$

The filtering process is a multiplication by the filter function in the frequency domain, or equivalently, a convolution in the spatial domain by the pulse response of the filter ($f(x,y)$) which is the inverse Fourier transform of the filter function:

$$I_f^*(\alpha,\beta) = I^*(\alpha,\beta) \cdot F(\alpha,\beta),$$

$$I_f(x,y) = I(x,y) * f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(t,\tau) \cdot f(x-t, y-\tau)\, dt d\tau. \tag{11.4}$$

The pulse response of the ideal low-pass filter is based on the well-known sinc function:

$$f(x,y) = \frac{\sin(x\cdot\pi/\Delta x)}{x\cdot\pi/\Delta x}\cdot\frac{\sin(y\cdot\pi/\Delta y)}{y\cdot\pi/\Delta y} = \mathrm{sinc}(\frac{x\cdot\pi}{\Delta x})\cdot\mathrm{sinc}(\frac{y\cdot\pi}{\Delta y}). \quad (11.5)$$

The realization of the low-pass filtering as a convolution with the 2D sinc function has some serious disadvantages. The sinc function decays very slowly, thus a great portion of the image can affect the color on a single point of the filtered picture, making the filtering complicated to accomplish. In addition to that, the sinc function has negative portions, which may result in negative colors in the filtered image. Sharp transitions of color in the original image cause noticeable ringing, called the Gibbs phenomenon, in the filtered picture. In order to overcome these problems, all positive, smooth, finite extent or nearly finite extent approximations of the ideal sinc function are used for filtering instead. These filters are expected to have unit gain for $\alpha = 0, \beta = 0$ frequencies, thus the integral of their impulse response must also be 1:

$$F(0,0) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\ dxdy = 1 \quad (11.6)$$

Some of the most widely used filters (figure 11.2) are:

1. **Box filter**: In the spatial domain:

$$f(x,y) = \begin{cases} 1 \text{ if } |x| < \Delta x/2 \text{ and } |y| < \Delta y/2 \\ 0 \text{ otherwise} \end{cases} \quad (11.7)$$

   In the frequency domain the box filter is a sinc function which is not at all accurate approximation of the ideal low-pass filters.

2. **Cone filter**: In the spatial domain, letting the normalized distance from the point (0,0) be $r(x,y) = \sqrt{(x/\Delta x)^2 + (y/\Delta y)^2}$:

$$f(x,y) = \begin{cases} (1-r)\cdot 3/\pi \text{ if } r < 1 \\ 0 \text{ otherwise} \end{cases} \quad (11.8)$$

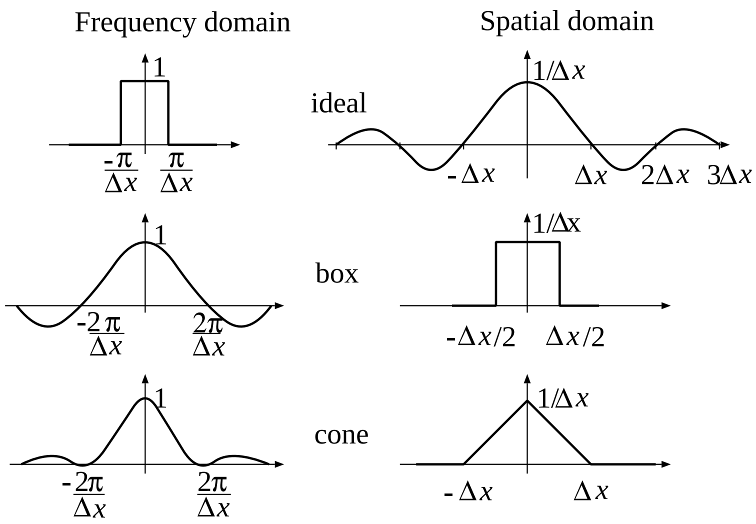

Figure 11.2: Frequency and spatial behavior of ideal and approximate low-pass filters

The coefficient $3/\pi$ guarantees that the total volume of the cone, that is the integral of the impulse response of the filter, is 1. The Fourier transformation of this impulse response is a sinc^2 type function which provides better high frequency suppression than the box filter.

3. **Gaussian filter**: This filter uses the Gaussian distribution function e^{-r^2} to approximate the sinc function by a positive, smooth function, where $r = \sqrt{(x/\Delta x)^2 + (y/\Delta y)^2}$ as for the cone filter. Although the Gaussian is not a finite extent function, it decays quickly making the contribution of distant points negligible.

Having defined the filter either in the frequency or in the spatial domain, there are basically two ways to accomplish the filtering. It can be done either in the spatial domain by evaluating the convolution of the original image and the impulse response of the filter, or in the frequency domain by multiplying the frequency distributions of the image by the filter function. Since the original image is available in spatial coordinates, and the filtered image is also expected in spatial coordinates, the latter approach requires a transformation of the image to the frequency domain before the filtering, then a transformation back to the spatial domain after the filtering. The computational burden of the two Fourier transformations makes frequency domain filtering acceptable only for special applications, even if effective methods, such as Fast Fourier Transform (FFT), are used.

11.2 Pre-filtering anti-aliasing techniques

Pre-filtering methods sample the image after filtering at the sample rate defined by the resolution of the display hardware ($\Delta x = 1, \Delta y = 1$). If the filtering has been accomplished in the spatial domain, then the filtered and sampled signal is:

$$I_{sf}(x,y) = [I(x,y) * f(x,y)] \cdot \sum_i \sum_j \delta(x-i, y-j). \tag{11.9}$$

For a given pixel of X, Y integer coordinates:

$$I_{sf}(X,Y) = I(x,y) * f(x,y) \cdot \delta(x-X, y-Y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(t,\tau) \cdot f(X-t, Y-\tau)\, dt d\tau. \tag{11.10}$$

For finite extent impulse response (FIR) filters, the infinite range of the above integral is replaced by a finite interval. For a box filter:

$$I_{s,\mathrm{box}}(X,Y) = \int_{X-0.5}^{X+0.5} \int_{Y-0.5}^{Y+0.5} I(t,\tau)\, dt d\tau. \tag{11.11}$$

Suppose P number of constant color primitives have intersection with the 1×1 rectangle of X, Y pixel. Let the color and the area of intersection of a primitive p be I_p and A_p, respectively. The integral 11.11 is then:

$$I_{s,\mathrm{box}}(X,Y) = \sum_{p=1}^{P} I_p \cdot A_p. \tag{11.12}$$

For a cone filter, assuming a polar coordinate system (r, ϕ) centered around (X, Y), the filtered signal is:

$$I_{s,\mathrm{cone}}(X,Y) = \frac{3}{\pi} \iint_{x^2+y^2 \le 1} I(x,y) \cdot (1-r)\, dx dy = \frac{3}{\pi} \int_{r=0}^{1} \int_{\phi=0}^{2\pi} I(r,\phi) \cdot (1-r) \cdot r\, d\phi dr. \tag{11.13}$$

As for the box filter, the special case is examined when P constant color primitives can contribute to a pixel, that is, they have intersection with the unit radius circle around the (X, Y) pixel, assuming the color and the area of intersection of primitive p to be I_p and A_p, respectively:

$$I_{s,\mathrm{cone}}(X, Y) = \frac{3}{\pi} \sum_{p=1}^{P} I_p \int_{A_p} (1 - r)\; dA \tag{11.14}$$

where $\int_{A_p} (1 - r)\; dA$ is the volume above the A_p area bounded by the surface of the cone.

11.2.1 Pre-filtering regions

An algorithm which uses a box filter needs to evaluate the area of the intersection of a given pixel and the surfaces visible in it. The visible intersections can be generated by an appropriate object precision visibility calculation technique (Catmull used the Weiler–Atherton method in his anti-aliasing algorithm [Cat78]) if the window is set such that it covers a single pixel. The color of the resulting pixel is then simply the weighted average of all visible polygon fragments. Although this is a very clear approach theoretically, it is computationally enormously expensive.

A more effective method can take advantage of the fact that regions tend to produce aliases along their edges where the color gradient is high. Thus an anti-aliased polygon generation method can be composed of an anti-aliasing line drawing algorithms to produce the edges, and a normal polygon filling method to draw all the internal pixels. Note that edge drawing must precede interior filling, since only the outer side of the edges should be filtered.

11.2.2 Pre-filtering lines

Recall that non-anti-aliased line segments with a slant of between 0 and 45 degrees are drawn by setting the color of those pixels in each column which are the closest to the line. This method can also be regarded as the point sampling of a one-pixel wide line segment.

Anti-aliasing line drawing algorithms, on the other hand, have to calculate an integral over the intersection of the one-pixel wide line and a finite region centered around the pixel concerned depending on the selected filter type.

Box-filtering lines

For box filtering, the intersection of the one-pixel wide line segment and the pixel concerned has to calculated. Looking at figure 11.3, we can see that a maximum of three pixels may intersect a pixel rectangle in each column if the slant is between 0 and 45 degrees. Let the vertical distance of the three closest pixels to the center of the line be r,s and t respectively, and suppose $s < t \leq r$. By geometric considerations $s, t < 1$, $s + t = 1$ and $r \geq 1$ should also hold.

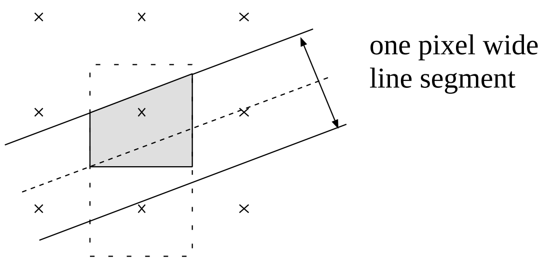

Figure 11.3: Box filtering of a line segment

Unfortunately, the areas of intersection, A_s, A_t and A_r, depend not only on r,s and t, but also on the slant of the line segment. This dependence, however, can be rendered unimportant by using the following approximation:

$$A_s \approx (1 - s), \qquad A_t \approx (1 - t), \qquad A_r \approx 0 \tag{11.15}$$

These equations are accurate only if the line segment is horizontal, but can be accepted as fair approximations for lines with a slant from 0 to 45 degrees. Variables s and t are calculated for a line $y = m \cdot x + b$:

$$s = m \cdot x + b - \mathrm{Round}(m \cdot x + b) = \mathrm{Error}(x) \quad \text{and} \quad t = 1 - s \tag{11.16}$$

where $\mathrm{Error}(x)$ is, in fact, the accuracy of the digital approximation of the line for vertical coordinate x. The color contribution of the two closest pixels in this pixel column is:

$$I_s = I \cdot (1 - \mathrm{Error}(x)), \qquad I_t = I \cdot \mathrm{Error}(x) \tag{11.17}$$

(I stands for any color coordinate R, G or B).

These formulae are also primary candidates for incremental evaluation, since if the closest pixel has the same y coordinate for an $x+1$ as for x:

$$I_s(x+1) = I_s(x) - I \cdot m, \qquad I_t(x+1) = I_t(x) + I \cdot m. \tag{11.18}$$

If the y coordinate has been incremented when stepping from x to $x+1$, then:

$$I_s(x+1) = I_s(x) - I \cdot m + I, \qquad I_t(x+1) = I_t(x) + I \cdot m - I. \tag{11.19}$$

The color computation can be combined with an incremental y coordinate calculation algorithm, such as the Bresenham's line generator:

AntiAliasedBresenhamLine(x_1, y_1, x_2, y_2, I)
  $\Delta x = x_2 - x_1$; $\Delta y = y_2 - y_1$;
  $E = -2\Delta x$;
  $dE^+ = 2(\Delta y - \Delta x)$; $dE^- = 2\Delta y$;
  $dI^- = \Delta y / \Delta x \cdot I$; $dI^+ = I - dI^-$;
  $I_s = I + dI^-$; $I_t = -dI^-$;
  $y = y_1$;
  for $x = x_1$ **to** x_2 **do**
    if $E \leq 0$ **then** E += dE^-; I_s -= dI^-; I_t += dI^-;
    else E += dE^+; I_s += dI^+; I_t -= dI^+; y++;
    Add Frame Buffer(x, y, I_s);
    Add Frame Buffer$(x, y+1, I_t)$;
  endfor

This algorithm assumes that the frame buffer is initialized such that each pixel has the color derived without taking this new line into account, and thus the new contribution can simply be added to it. This is true only if the frame buffer is initialized to the color of a black background and lines do not cross each other. The artifact resulting from crossed lines is usually negligible.

In general cases I must rather be regarded as a weight value determining the portions of the new line color and the color already stored in the frame buffer, which corresponds to the color of objects behind the new line.

The program line "Add Frame Buffer(x, y, I)" should be replaced by the following:

$\text{color}_{\text{old}} = \text{frame\_buffer}[x, y];$
$\text{frame\_buffer}[x, y] = \text{color}_{\text{line}} \cdot I + \text{color}_{\text{old}} \cdot (1 - I);$

These statements must be executed for each color coordinate R, G, B.

Cone filtering lines

For cone filtering, the volume of the intersection between the one-pixel wide line segment and the one pixel radius cone centered around the pixel concerned has to be calculated. The height of the cone must be selected to guarantee that the volume of the cone is 1. Looking at figure 11.4, we can see that a maximum of three pixels may have intersection with a base circle of the cone in each column if the slant is between 0 and 45 degrees.

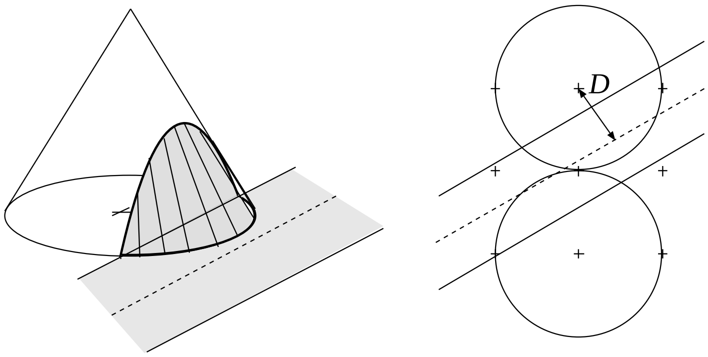

Figure 11.4: Cone filtering of a line segment

Let the distance between the pixel center and the center of the line be D. For possible intersection, D must be in the range of $[-1.5..1.5]$. For a pixel center (X, Y), the convolution integral — that is the volume of the cone segment above a pixel — depends only on the value of D, thus it can be computed for discrete D values and stored in a lookup table $V(D)$ during the design of the algorithm. The number of table entries depends on the number of intensity levels available to render lines, which in turn determines the necessary precision of the representation of D. Since 8–16 intensity levels are enough to eliminate the aliasing, the lookup table is defined here for three and four fractional bits. Since function $V(D)$ is obviously symmetrical, the number of necessary table entries for three and four fractional bits is $1.5 \cdot 2^3 = 12$ and $1.5 \cdot 2^4 = 24$ respectively. The precomputed $V(D)$ tables, for 3 and 4 fractional bits, are shown in figure 11.5.

D_3	0	1	2	3	4	5	6	7	8	9	10	11
$V(D_3)$	7	6	6	5	4	3	2	1	1	0	0	0

D_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$V(D_4)$	14	14	13	13	12	12	11	10	9	8	7	6	5	4	3	3	2	2	1	1	0	0	0	0

Figure 11.5: Precomputed $V(D)$ weight tables

Now the business of the generation of D and the subsequent pixel coordinates must be discussed. Gupta and Sproull [GSS81] proposed the Bresenham algorithm to produce the pixel addresses and introduced an incremental scheme to generate the subsequent D distances.

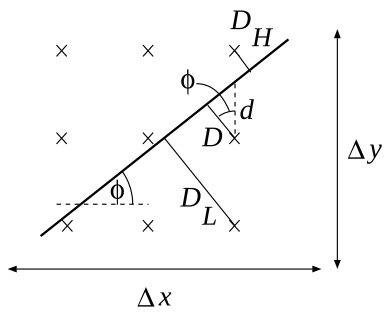

Figure 11.6: Incremental calculation of distance D

Let the obliqueness of the line be ϕ, and the vertical distance between the center of the line and the closest pixel be d (note that for the sake of simplicity only lines with obliquities in the range of [0..45] are considered as in the previous sections).

For geometric reasons, illustrated by figure 11.6, the D values for the three vertically arranged pixels are:

$$\begin{aligned} D &= d \cdot \cos\phi = \frac{d \cdot \Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}, \\ D_H &= (1-d) \cdot \cos\phi = -D + \frac{\Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}, \\ D_L &= (1+d) \cdot \cos\phi = D + \frac{\Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}. \end{aligned} \tag{11.20}$$

A direct correspondence can be established between the distance variable d and the integer error variable E of the Bresenham line drawing algorithm that generates the y coordinates of the subsequent pixels (see section 2.3). The k fractional error variable of the Bresenham algorithm is the required distance plus 0.5 to replace the rounding operation by a simpler truncation, thus $d = k - 0.5$. If overflow happens in k, then $d = k - 1.5$. Using the definition of the integer error variable E, and supposing that there is no overflow in k, the correspondence between E and d is:

$$E = 2\Delta x \cdot (k-1) = 2\Delta x \cdot (d - 0.5) \quad \Longrightarrow \quad 2d \cdot \Delta x = E + \Delta x. \tag{11.21}$$

If overflow happens in k, then:

$$E = 2\Delta x \cdot (k-1) = 2\Delta x \cdot (d + 0.5) \quad \Longrightarrow \quad 2d \cdot \Delta x = E - \Delta x. \tag{11.22}$$

These formulae allow the incremental calculation of $2d \cdot \Delta x$; thus in equation 11.20 the numerators and denominators must be multiplied by two.

The complicated operations including divisions and a square root should be executed once for the whole line, thus the pixel level algorithms contain just simple instructions and a single multiplication not counting the averaging with the colors already stored in the frame buffer. In the subsequent program, expressions that are difficult to calculate are evaluated at the beginning, and stored in the following variables:

$$\text{denom} = \frac{1}{2\sqrt{(\Delta x)^2 + (\Delta y)^2}}, \qquad \Delta D = \frac{2\Delta x}{2\sqrt{(\Delta x)^2 + (\Delta y)^2}}. \tag{11.23}$$

In each cycle nume $= 2d \cdot \Delta x$ is determined by the incremental formulae.

The complete algorithm is:

GuptaSproullLine(x_1, y_1, x_2, y_2, I)
$\Delta x = x_2 - x_1$; $\Delta y = y_2 - y_1$;
$E = -\Delta x$;
$dE^+ = 2(\Delta y - \Delta x)$; $dE^- = 2\Delta y$;
$\text{denom} = 1/(2\sqrt{(\Delta x)^2 + (\Delta y)^2})$;
$\Delta D = 2 \cdot \Delta x \cdot \text{denom}$;
$y = y_1$;
for $x = x_1$ **to** x_2 **do**
  if $E \leq 0$ **then** nume $= E + \Delta x$; E += dE^-;
  else nume $= E - \Delta x$; E += dE^+; y++;
  $D = \text{nume} \cdot \text{denom}$;
  $D_L = D + \Delta D$; $D_H = -D + \Delta D$;
  Add Frame Buffer$(x, y, V(D))$;
  Add Frame Buffer$(x, y + 1, V(D_H))$;
  Add Frame Buffer$(x, y - 1, V(D_L))$;
endfor

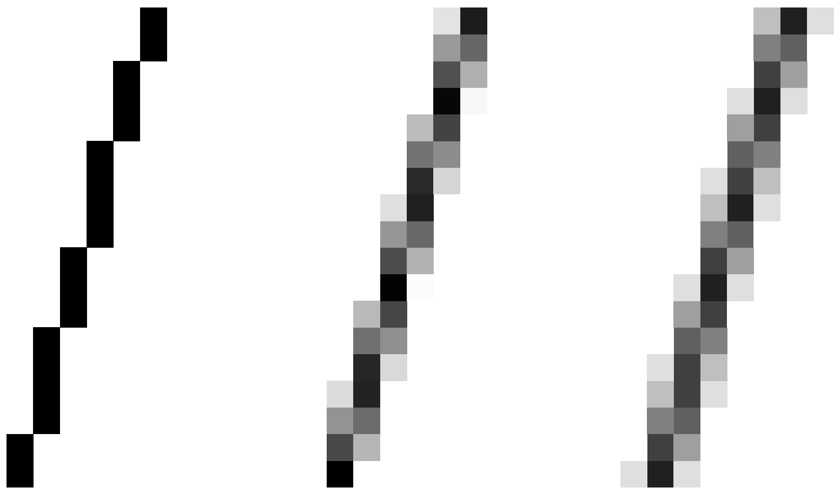

Figure 11.7: Comparison of normal, box-filtered and cone-filtered lines

11.3 Post-filtering anti-aliasing techniques

Post-filtering methods sample the image at a higher sample rate than needed by the resolution of the display hardware ($\Delta x = 1/N, \Delta y = 1/N$), then some digital filtering algorithm is used to calculate pixel colors.

For digital filtering, the spatial integrals of convolution are replaced by infinite sums:

$$I_{sf}(X,Y) = [I(x,y)\sum_i\sum_j \delta(x - i\cdot\Delta x, y - j\cdot\Delta y)] * f(x,y)|_{x=X,y=Y} =$$

$$\sum_i\sum_j I(i\cdot\Delta x, j\cdot\Delta y)\cdot f(X - i\cdot\Delta x, Y - j\cdot\Delta y). \tag{11.24}$$

Finite extent digital filters simplify the infinite sums to finite expressions. One of the simplest digital filters is the discrete equivalent of the continuous box filter:

$$I_{s,\mathrm{box}}(X,Y) = \frac{1}{(N+1)^2}\sum_{i=-N/2}^{N/2}\sum_{j=-N/2}^{N/2} I(X - i\cdot\Delta x, Y - j\cdot\Delta y). \tag{11.25}$$

This expression states that the average of **subpixel** colors must be taken to produce the color of the real pixel. The color of the subpixels can be determined by a normal, non-anti-aliasing image generation algorithm. Thus, ordinary image synthesis methods can be used, but at a higher resolution, to produce anti-aliased pictures since the anti-aliasing is provided by the final step reducing the resolution to meet the requirements of the display hardware. One may think that this method has the serious drawback of requiring a great amount of additional memory to store the image at a higher resolution, but that is not necessarily true. Ray tracing, for example, generates the image on a pixel-by-pixel basis. When all the subpixels affecting a pixel have been calculated, the pixel color can be evaluated and written into the raster memory, and the very same extra memory can be used again for the subpixels of other pixels. Scan-line methods, on the other hand, require those subpixels which may contribute to the real pixels in the scan-line to be calculated and stored. For the next scan-line, the same subpixel memory can be used again.

As has been stated, discrete algorithms have linear complexity in terms of the pixel number of the image. From that perspective, supersampling may increase the computational time by a factor of the number of subpixels affecting a real pixel, which is usually not justified by the improvement of image quality, because aliasing is concentrated mainly around sharp edges, and the filtering does not make any significant difference in great homogeneous areas. Therefore, it is worth examining whether the color gradient is great in the neighborhood of a pixel, or whether instead the color is nearly constant, and thus dividing the pixels into subpixels only if required by a high color gradient. This method is called **adaptive supersampling**.

Finally, it is worth mentioning that jaggies can be greatly reduced without increasing the sample frequency at all, simply by moving the sample points from the middle of pixels to the corner of pixels, and generating the color of the pixel as the average of the colors of its corners. Since pixels have four corners, and each internal corner point belongs to four pixels, the number of corner points is only slightly greater than the number of pixel centers. Although this method is not superior in eliminating aliases, it does have a better reconstruction filter for reducing the sharp edges of jaggies.

11.4 Stochastic sampling

Sampling methods applying regular grids produce regularly spaced artifacts that are easily detected by the human eye, since it is especially sensitive to regular and periodic signals. Random placement of sample locations can break up the periodicity of the aliasing artifacts, converting the aliasing effects to random noise which is more tolerable for human observers. Two types of random sampling patterns have been proposed [Coo86], namely the **Poisson disk distribution** and the **jittered** sampling.

11.4.1 Poisson disk distribution

Poisson disk distribution is, in fact, the simulation of the sampling process of the human eye [Yel83]. In effect, it places the sample points randomly with the restriction that the distances of the samples are greater than a specified minimum. Poisson disk distribution has a characteristic distribution in the frequency domain consisting of a spike at zero frequency and a uniform noise beyond the Nyquist limit. Signals having white-noise-like spectrum at higher frequencies, but low-frequency attenuation, are usually regarded as **blue noise**. This low-frequency attenuation is responsible for the approximation of the minimal distance constraint of Poisson disk distribution.

The sampling process by a Poisson disk distributed grid can be understood as follows: Sampling is, in fact, a multiplication by a "comb function" of the sampling grid. In the frequency domain it is equivalent to convolution by the Fourier transform of this "comb function", which is a blue-noise-like spectrum for Poisson disk distribution except for the spike at 0. Signal components below the Nyquist limit are not affected by this convolution, but components above this limit are turned to a wide range spectrum of noise. Signal components not meeting the requirements of the sampling theorem have been traded off for noise, and thus aliasing in the form of periodic signals can be avoided.

Sampling by Poisson disk distribution is very expensive computationally. One way of approximating appropriate sampling points is based on error diffusion dithering algorithms (see section 11.5 on reduction of quantization effects), since dithering is somehow similar to this process [Mit87], but now the sampling position must be dithered.

11.4.2 Jittered sampling

Jittered sampling is based on a regular sampling grid which is perturbed slightly by random noise [Bal62]. Unlike the application of dithering algorithms, the perturbations are now assumed to be independent random variables. Compared to Poisson disk sampling its result is admittedly not quite as good, but it is less expensive computationally and is well suited to image generation algorithms designed for regular sampling grids.

For notational simplicity, the theory of jittered sampling will be discussed in one-dimension. Suppose function $g(t)$ is sampled and then reconstructed by an ideal low-pass filter. The perturbations of the various sample locations are assumed to be uncorrelated random variables defined by the probability density function $p(x)$. The effect of jittering can be simulated by replacing $g(t)$ by $g(t - \xi(t))$, and sampling it by a regular grid, where function $\xi(t)$ is an independent stochastic process whose probability density function, for any t, is $p(x)$ (figure 11.8).

Jittered sampling can be analyzed by comparing the spectral power distributions of $g(t - \xi(t))$ and $g(t)$.

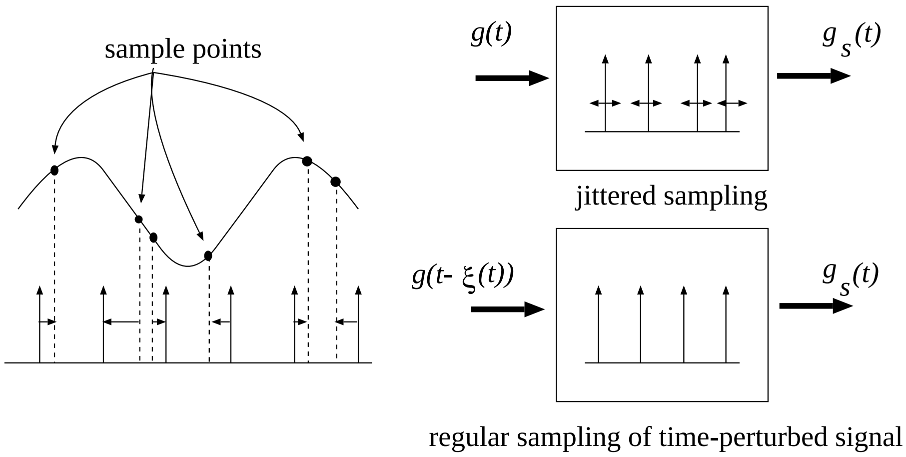

Figure 11.8: Signal processing model of jittered sampling

Since $g(t-\xi(t))$ is a random process, if it were stationary and ergodic [Lam72], then its frequency distribution would be best described by the power density spectrum which is the Fourier transform of its autocorrelation function.

The autocorrelation function of $g(t-\xi(t))$ is derived as an expectation value for any $\tau \neq 0$, taking into account that $\xi(t)$ and $\xi(t+\tau)$ are stochastically independent random variables:

$$R(t,\tau) = E[g(t-\xi(t)) \cdot g(t+\tau-\xi(t+\tau))] =$$

$$\int_x \int_y g(t-x) \cdot g(t+\tau-y) \cdot p(x) \cdot p(y)\ dx\ dy = (g*p)|_t \cdot (g*p)|_{t+\tau} \quad (11.26)$$

where $g*p$ is the convolution of the two functions. If $\tau = 0$, then:

$$R(t,0) = E[g(t-\xi(t))^2] = \int_x g^2(t-x) \cdot p(x)\ dx = (g^2 * p)|_t. \quad (11.27)$$

Thus the autocorrelation function of $g(t-\xi(t))$ for any τ is:

$$R(t,\tau) = (g*p)|_t \cdot (g*p)|_{t+\tau} + [(g^2*p) - (g*p)^2]|_t \cdot \delta(\tau) \quad (11.28)$$

where $\delta(\tau)$ is the delta function which is 1 for $\tau = 0$ and 0 for $\tau \neq 0$. This delta function introduces an "impulse" in the autocorrelation function at $\tau = 0$.

Assuming $t = 0$ the size of the impulse at $\tau = 0$ can be given an interesting interpretation if $p(x)$ is an even function ($p(x) = p(-x)$).

$$[(g^2 * p) - (g * p)^2]|_{t=0} = \int_x g^2(-x) \cdot p(x)dx - [\int_x g(-x) \cdot p(x)dx]^2 =$$

$$E[g^2(\xi)] - E^2[g(\xi)] = \sigma^2_{g(\xi)}. \qquad (11.29)$$

Hence, the size of the impulse in the autocorrelation function is the variance of the random variable $g(\xi)$. Moving the origin of the coordinate system to t we can conclude that the size of the impulse is generally the variance of the random variable $g(t - \xi(t))$.

Unfortunately $g(t - \xi(t))$ is usually not a stationary process, thus in order to analyze its spectral properties, the power density spectrum is calculated from the "average" autocorrelation function which is defined as:

$$\hat{R}(\tau) = \lim_{T \to \infty} \frac{1}{2T} \int_{-T}^{T} R(t, \tau)\, dt. \qquad (11.30)$$

The "average" power density of $g(t - \xi(t))$, supposing $p(x)$ to be even, can be expressed using the definition of the Fourier transform and some identity relations:

$$\hat{S}(f) = \mathcal{F}\hat{R}(\tau) = \lim_{T \to \infty} \frac{1}{2T} [\mathcal{F}_T\ (g * p)]^* \cdot [\mathcal{F}(g * p)] + \overline{\sigma^2_{g(\xi)}} \qquad (11.31)$$

where superscript $*$ means the conjugate complex pair of a number, $\overline{\sigma^2_{g(\xi)}}$ is the average variance of the random variable $g(t - \xi(t))$ for different t values, and $\mathcal{F}_T$ stands for the limited Fourier transform defined by the following equation:

$$\mathcal{F}_T\ x(t) = \int_{-T}^{T} x(t) \cdot e^{-2\pi \jmath f t}\, dt, \qquad \jmath = \sqrt{-1} \qquad (11.32)$$

Let us compare this power density ($\hat{S}(f)$) of the time perturbed signal with the power density of the original function $g(t)$, which can be defined as follows:

$$S_g(f) = \lim_{T \to \infty} \frac{1}{2T} |\mathcal{F}_T\ g(t)|^2. \qquad (11.33)$$

This can be substituted into equation 11.31 yielding:

$$\hat{S}(f) = \frac{|\mathcal{F}(g * p)|^2}{|\mathcal{F}g|^2} \cdot S_g(f) + \overline{\sigma^2_{g(\xi)}}. \tag{11.34}$$

The spectrum consists of a part proportional to the spectrum of the unperturbed $g(t)$ signal and an additive noise carrying $\overline{\sigma^2_{g(\xi)}}$ power in a unit frequency range. Thus the perturbation of time can, in fact, be modeled by a linear network or filter and some additive noise (Figure 11.9).

The gain of the filter perturbing the time variable by an independent random process can be calculated as the ratio of the power density distributions of $g(t)$ and $g(t - \xi(t))$ ignoring the additive noise:

$$\text{Gain}(f) = \frac{|\mathcal{F}(g * p)|^2}{|\mathcal{F}g|^2} = \frac{|\mathcal{F}g|^2 \cdot |\mathcal{F}p|^2}{|\mathcal{F}g|^2} = |\mathcal{F}p|^2. \tag{11.35}$$

Thus, the gain is the Fourier transform of the probability density used for jittering the time.

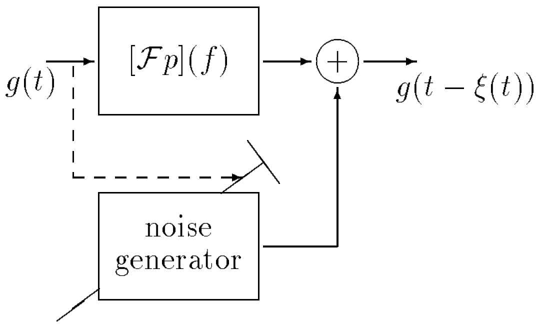

Figure 11.9: System model of time perturbation

Two types of jitters are often used in practice:

1. **White noise jitter**, which distributes the values uniformly between $-T/2$ and $T/2$, where T is the periodicity of the regular sampling grid. The gain of the white noise jitter is:

$$\text{Gain}_{\text{wn}}(f) = [\frac{\sin \pi f T}{\pi f T}]^2. \tag{11.36}$$

2. **Gaussian jitter**, which selects the sample points by Gaussian distribution with variance ρ^2.

$$\text{Gain}_{\text{gauss}}(f) = e^{-(2\pi f \rho)^2}. \tag{11.37}$$

Both the white noise jitter and the Gaussian jitter (if $\rho \approx T/6$) are fairly good low-pass filters suppressing the spectrum of the sampled signal above the Nyquist limit, and thus greatly reducing aliasing artifacts.

Jittering trades off aliasing for noise. In order to intuitively explain this result, let us consider the time perturbation for a sine wave. If the extent of the possible perturbations is less than the length of half a period of the sine wave, the perturbation does not change the basic shape of the signal, just distorts it a little bit. The level of distortion depends on the extent of the perturbation and the "average derivative" of the perturbed function as suggested by the formula of the noise intensity defining it as the variance $\overline{\sigma^2_{g(\xi)}}$. If the extent of the perturbations exceeds the length of period, the result is an almost random value in place of the amplitude. The sine wave has disappeared from the signal, only the noise remains.

11.5 Reduction of quantization effects

In digital data processing, not only the number of data must be finite, but also the information represented by a single data element. Thus in computer graphics we have to deal with problems posed by the fact that color information can be represented by a few discrete levels in addition to the finite sampling which allows for the calculation of this color at discrete points only.

In figure 11.10 the color distribution of a shaded sphere is shown. The ideal continuous color is sampled and quantized according to the pixel resolution and the number of quantization levels resulting in a stair-like function in the color space. (Note that aliasing caused stair-like jaggies in pixel space.) The width of these stair-steps is usually equal to the size of many pixels if there are not too many quantization levels, which makes the effect clearly noticeable in the form of quasi-concentric circles on the surface of our sphere. Cheaper graphics systems use eight bits for the representation of a single pixel allowing R, G, B color coordinates to be described by

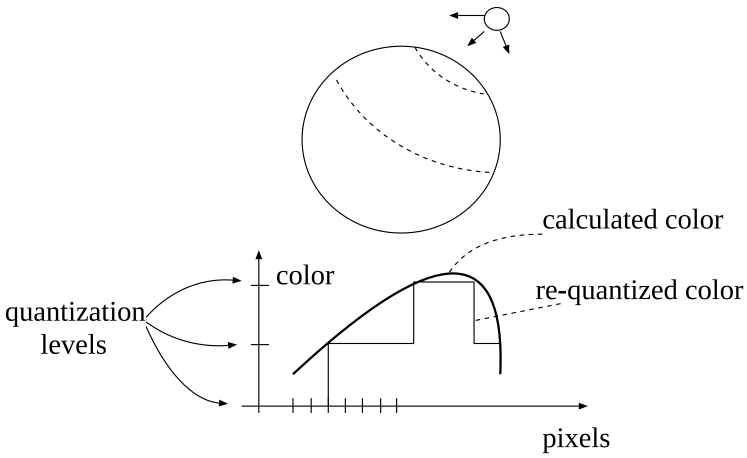

Figure 11.10: Quantization effects

three, three and two bits respectively in true color mode, which is far from adequate. Expensive workstations provide eight bits for every single color coordinate, that is 24 bits for a pixel, making it possible to produce over sixteen million colors simultaneously on the computer screen, but this is still less than the number of colors that can be distinguished by the human eye.

If we have just a limited set of colors but want to produce more, the obvious solution is to try to mix new ones from the available set. At first we might think that this mixing is beyond the capabilities of computer graphics, because the available set of colors is on the computer screen, and thus the mixing should happen when the eye perceives these colors, something which seemingly cannot be controlled from inside the computer. This is fortunately not exactly true. Mixing means a weighted average which can be realized by a low-pass filter, and the eye is known to be a fairly good low-pass filter. Thus, if the color information is provided in such a way that high frequency variation of color is generated where mixing is required, the eye will filter these variations and "compute" its average which exactly amounts to a mixed color.

These high-frequency variations can be produced by either sacrificing the resolution or without decreasing it at all. The respective methods are called **halftoning** and **dithering**.

11.5.1 Halftoning

Halftoning is a well-known technique in the printing industry where gray-level images are produced by placing black points onto the paper, keeping the density of these points proportional to the desired gray level. On the computer screen the same effect can be simulated if adjacent pixels are grouped together to form *logical pixels*. The color of a logical pixel is generated by the pattern of the colors of its physical pixels. Using an $n \times n$ array of bi-level physical pixels, the number of producible colors is $n^2 + 1$ for the price of reducing the resolution by a factor of n in both directions (figure 11.11). This idea can be applied to interpolate between any two subsequent quantization levels (even for any two colors, but this is not used in practice).

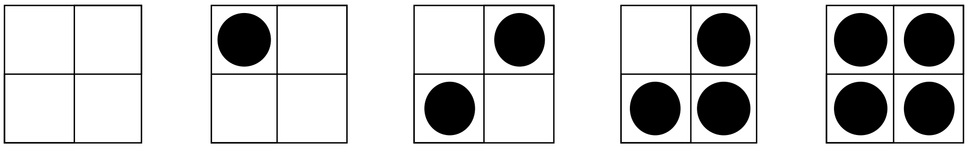

Figure 11.11: Halftone patterns for $n = 4$

11.5.2 Dithering

Unlike halftoning, dithering does not reduce the effective resolution of the display. This technique was originated in measuring theory where the goal was the improvement of the effective resolution of A/D converters. Suppose we have a one-bit quantization unit (a comparator), and a slowly changing signal needs to be measured. Is it possible to say more about the signal than to determine whether it is above or below the threshold level of the comparator?

In fact, the value of the slowly changing signal can be measured accurately if another symmetrical signal, called a dither, having a mean of 0 and appropriate peak value, is added to the signal before quantization. The perturbed signal will spend some of the time below, while the rest remains above the threshold level of the comparator (figure 11.12). The respective times — that is the average or the filtered composite signal — will show

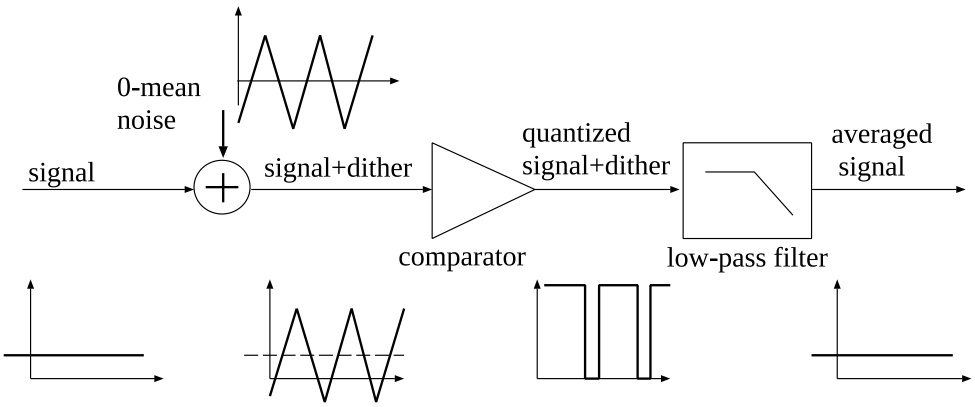

Figure 11.12: Measuring the mean value of a signal by a one-bit quantizer

the mean value of the original signal accurately if the filtering process eliminates the higher frequencies of the dither signal but does not interfere with the low frequency range of the original signal. Thus the frequency characteristic of the dither must be carefully defined: it should contain only high frequency components; that is, it should be **blue noise**.

This idea can readily be applied in computer graphics as well. Suppose the color coordinates of pixels are calculated at a higher level of accuracy than is needed by the frame buffer storage. Let us assume that the frame buffer represents each R, G, B value by n bits and the color computation results in values of $n + d$ bit precision. This can be regarded as a fixed point representation of the colors with d number of fractional bits. Simple truncation would cut off the low d bits, but before truncation a dither signal is added, which is uniformly distributed in the range of [0..1]; that is, it eventually produces distribution in the range of [-0.5..0.5] if truncation is also taken into consideration. This added signal can either be a random or a deterministic function. Periodic deterministic dither functions are also called **ordered dithers**. Taking into account the blue noise criterion, the dither must be a high frequency signal. The maximal frequency dithers are those which have different values on adjacent pixels, and are preferably not periodic. In this context, ordered dithers are not optimal, but they allow for simple hardware implementation, and thus they are the most frequently used methods of reducing the quantization effects in computer graphics.

The averaging of the dithered colors to produce mixed colors is left to the human eye as in halftoning.

Ordered dithers

The behavior of ordered dithers is defined by a periodic function $D(i, j)$ which must be added to the color computed at higher precision. Let the periodicity of this function be N in both vertical and horizontal directions, so D can thus conveniently be described by an $N \times N$ matrix called a dither table. The dithering operation for any color coordinate I is then:

$$I[X, Y] \Leftarrow \mathrm{Trunc}(I[X, Y] + D[X \bmod N,\ Y \bmod N]). \qquad (11.38)$$

The expectations of a "good" dither can be summarized as follows:

1. It should approximate blue noise, that is, neighboring values should not be too close.
2. It should prevent periodic effects.
3. It must contain uniformly distributed values in the range of $[0..1]$. If fixed point representation is used with d fractional bits, the decimal equivalents of the codes must be uniformly distributed in the range of $[0..2^d]$.
4. The computation of the dithered color must be simple, fast and appropriate for hardware realization. This requirement has two consequences. First, the precision of the fractional representation should correspond to the number of elements in the dither matrix to avoid superfluous bits in the representation. Secondly, dithering requires two modulo N divisions which are easy to accomplish if N is a power of two.

A dither which meets the above requirements would be:

$$D^{(4)} = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \qquad (11.39)$$

where a four-bit fractional representation was assumed, that is, to calculate the real value equivalents of the dither, the matrix elements must be divided by 16.

Let us denote the low k bits and the high k bits of a binary number B by $B|_k$ and $B|^k$ respectively. The complete dithering algorithm is then:

> Calculate the (R, G, B) color of pixel (X, Y) and
> represent it in an n-integer-bit + 4-fractional-bit form;
> $R = (R + D[X|_2, Y|_2])|^n$;
> $G = (G + D[X|_2, Y|_2])|^n$;
> $B = (B + D[X|_2, Y|_2])|^n$;

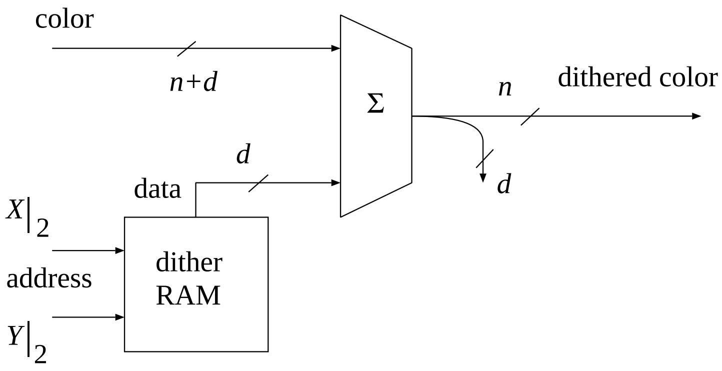

Figure 11.13: Dithering hardware

This expression can readily be implemented in hardware as is shown in figure 11.13.