BRESENHAM-STYLE SCALING

Robert Ulichney, Digital Equipment Corporation, Maynard, Massachusetts

ABSTRACT

A highly efficient scheme for achieving nearest-neighbor spatial scaling is presented. Based on the Bresenham's line drawing algorithm, this scheme requires only one adder, a register, and a multiplexer to control scaling of an input stream. When the registers are of a fixed size, 8-bits wide for example, the determination of what values to use is not straight forward. With the goal of increasing set-up speed, derivation of a closed-form solution for the exact output size, given the input size and the scaler parameters.

BRESENHAM'S LINE DRAWING ALGORITHM

Image scaling can be described as superpositioning the grid of the output image on the grid of the output image, where both grids are of the same size but different densities. Ideally filtering is performed[1] in the mapping of input to output. However, a simpler scheme consists of assigning output pixels with the value of the pixel from the input image whose center is closest; this method is called nearest-neighbor scaling. It can also be described as "coincident resampling" [2].

The most efficient means for performing nearest neighbor scaling can be adopted from the famous Bresenham scan conversion algorithm for drawing straight lines[3]. This method is attractive because it requires no multiplies.

Figure 1 shows the pixels selected to represent a line of positive slope less than 1. We will consider rational slopes that can be expressed as the ratio of an integral numerator N and denominator D. For this shallow-slope case, the Bresenham algorithm steps through x addresses and decides if the y address is to be incremented based on the sign of an accumulator, a[i]. At horizontal location x[i] the

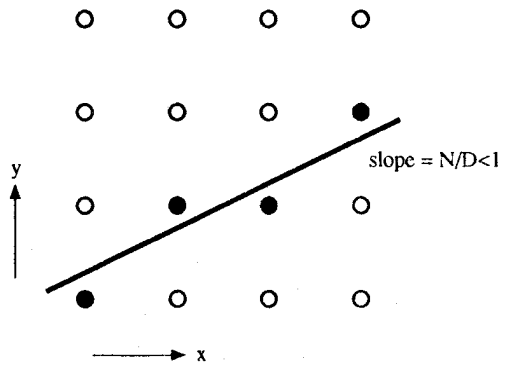


Fig. 1. Drawing a line with a positive slope < 1. This will be used as a model for image reduction.

algorithm finds the vertical y[i] that is closest to the given line as follows:

```
if a[i] < 0:
    y[i] = y[i-1]
    a[i+1] = a[i] + 2N
else:                                    (1)
    y[i] = y[i-1] + 1
    a[i+1] = a[i] + 2 (N-D)
```

The accumulator is initialized with a[0]=(2N-D).

By tolerating a slight shift in the origin of the line, with no compromise in slope accuracy, a simplification can be made to the above algorithm (1) by eliminating the factors of 2 and initializing a [0] = 0. In an implementation where register sizes are limited, this change increases the precision of the "deltas" that increment a[ ] by a bit, doubling the general accuracy.

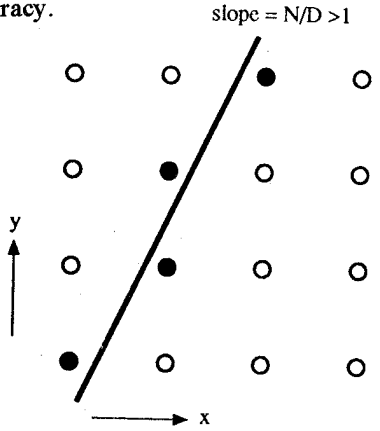


Fig. 2. Drawing a line with a slope > 1. This will be used as a model for image enlargement.

The case of drawing a line with slope greater than 1 is shown in figure 2. In this case the algorithm steps through y addresses and decides if the x address is to be incremented. Using our simplified form, we initialize a [0] = 0 and proceed:

```
if a[i] < 0:
    x[i] = x[i-1]
    a[i+1] = a[i] + D                    (2)
else:
    x[i] = x[i-1] + 1
    a[i+1] = a[i] + (D-N)
```

SCALING ALGORITHM ADAPTATION

Glazer[4] discovered that the Bresenham algorithm can be applied to scaling one dimension of an image, and performed with the simple circuit shown in Figure 3. In this circuit, one of two delta values, d0 or d1, are selected by means of the sign bit (msb) of the Accumulator. This delta value additively updates the Accumulator, which is always cleared before scaling begins.

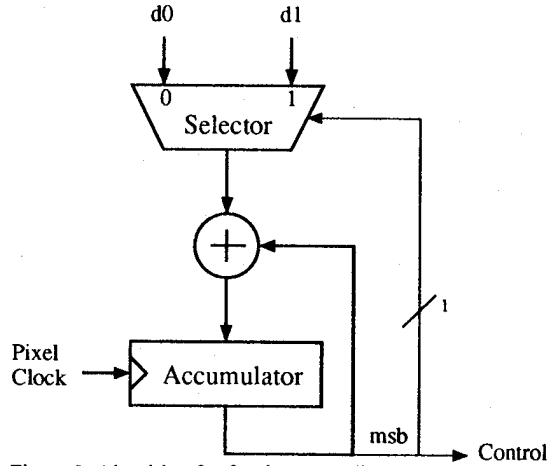


Figure 3. Algorithm for fast image scaling.

Image scaling mimics the line drawing algorithms described above where the x coordinate represents the input image pixels, and the y coordinate represents output image pixels. The scale factor is the slope N/D. A separate system is used to independently scale each dimension.

REDUCTION

Drawing a line as in Figure 1 where N/D is less than 1 is used as a model for reduction. The Pixel Clock in Figure 3 latches the Accumulator, and is driven by an input pixel counter. The delta values are d0=(N-D) and d1=N. The Control bit (the msb of the Accumulator) is interpreted as follows: "0" means assign an output pixel the value of the input pixel; "1" means skip this input pixel.

```
for each input pixel:
    if a[i]<0:
            skip input pixel
            a[i+1] = a[i] + d1            (3)
    else:
            use input pixel
            a[i+1] = a[i] + d0
```

ENLARGEMENT

Similarly the case shown in Figure 2, where N/D is greater than 1, models enlargement. In this case, the Pixel Clock is driven by the output pixel counter, and the delta values are d0=(D-N) and d1=D. Now a control bit of value 0 is interpreted as "use a new input pixel", and a value of 1 means "repeat last used input pixel".for each input pixel:

```
    use this new input pixel
    a[i+1] = a[i] + d0                    (4)
    while a < 0:
            repeat input pixel
            a[i+1] = a[i] + d1
```

CLOSED FORM PREDICTOR

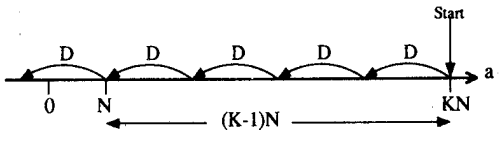
In the case where the circuit of Figure 3 has an adder and Accumulator wide enough to accommodate the largest anticipated input size and output size, then N can be assigned the output size and D the input size. The results will be perfectly exact.

However, when data widths are limited – typically 8 bits – optimization is needed to select the value of N and D that will produce results close to the desired output size. Given N and D, and an input size K, it is not obvious what the resulting output size, M will be. One can iterat algorithms (3) and (4) above, but that can impede real-time scalechanges in an interactive environment.

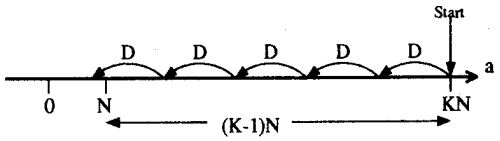
For the sake of analyzing the circuit in Figure 3, the algorithm (3) for reduction can be equivalently rewritten as:

```
for each of K inputs:
    a[i+1] = a[i]+d1
    if a[i] < d1:
            skip input pixel
    if a[i] >= d1:                    (5)
            use input pixel
            a[i+1] = a[i] + (d0-d1)
```

Substituting d1=N and (d0-d1)=(-D) in the above algorithm, after K iterations, a[ ] will equal KN decreased my as many units of (-D) necessary to make the result less than N. The number of "(-D) units" is precisely the value of M, the output size. This is graphically depicted in Figure 4.



Case (a): (K-1)N/D is an integer.



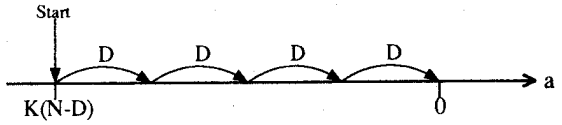
Case (b): (K-1)N/D is not an integer.

Figure 4. Graphic solution for closed-form output size when reducing.

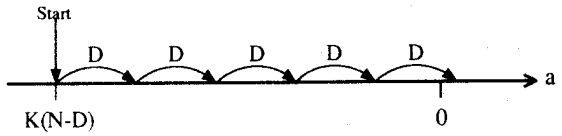
Starting with KN, the value must be reduced to just below N, a range of (K-1)N. There are two cases depending on whether (K-1)N/D is an integer. From this we arrive at the solution:

$$M = \text{integer}\{ (K-1)\, N/D \} + 1 \qquad (6)$$

In the enlargement case, algorithm (4) can be used as is to evaluate the final value of a[ ]. We substitute d0=(D-N) and d1=D. After K iterations, a[ ] will be equal to Kd0 = K(D-N), which is a negative number, increased by as many units of D necessary to make the result greater that or equal to 0, as depicted in Figure 5 for the two cases.



Case (a): K(N-D)/D is an integer.



Case (b): K(N-D)/D is not an integer.

Fig. 5. Graphic solution for closed-form output size when enlarging.

A span of exactly K(D-N) or just greater than it must be made up by units of D. These "units of D" represent repeated pixels in the output. We can find this number by rounding up the ratio K(N-D)/D. This is achieved by adding (D-1)/D to it before integer truncation:

$$\text{Repeated pixels} = \text{integer}\{ K(N-D)/D + (D-1)/D \} \qquad (7)$$

The number of output pixels that are "new" pixels is K, all of the input pixels. The total output size, M, is the number of repeated pixels plus the number of new output pixels. Thus,

$$M = \text{integer}\{ K(N-D)/D + (D-1)/D + K \}, \text{ or}$$

$$M = \text{integer}\{ (KN-1)/D \} + 1 \qquad (8)$$

With these solutions for M, candidate N and D values can be quickly evaluated for optimization. It is important to note that for the case where there is no bit-width limit, this proves that setting (N/D) = (M/K) in either eqautions (6) or (8) will always result in an exact output size, M.

ACKNOWLEDGMENT

Thanks to Tim Hellman for suggesting the trick to finding the closed form solutions.

REFERENCES

1. W.F. Schreiber and D.E. Troxel, "Transformation between continuous and discrete representation of images: a perceptual approach", IEEE Trans. PAMI, Vol. PAMI-7, no. 2, pp. 178-186 (1985).
2. R. Ulichney, "Digital scaling of binary images," M.S. Thesis, M.I.T., Cambridge, MA, (1979).
3. J.E. Bresenham, "Algorithm for computer control of a digital plotter", IBM Systems Journal, vol. 4, no. 1, pp. 25-30, (1965).
4. F. Glazer, "Fast bitonal to grayscale image scaling", Maynard: Digital Equipment Corp. DEC-TR-505, (June, 1987).