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 register is of a fixed size, 8-bits wide for example, the determination of what values to use is not straightforward. 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 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".

The most efficient means for performing nearest-neighbor scaling can be adopted from the famous Bresenham scan conversion algorithm for drawing straight lines. 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 integer 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
SCALING ALGORITHM ADAPTATION

Glazer\textsuperscript{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, \( d_0 \) or \( d_1 \), 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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure3.jpg}
\caption{Algorithm for fast image scaling.}
\end{figure}

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 \( d_0 = -(N-D) \) and \( d_1 = N \). 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:
\begin{align*}
\text{if } a[i] &< 0: \\
& \text{skip input pixel} \\
& a[i+1] = a[i] + d_0
\end{align*}

\begin{align*}
\text{else:} \\
& \text{use input pixel} \\
& a[i+1] = a[i] + d_1
\end{align*}

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 \( d_0 = -(D-N) \) and \( d_1 = D \). 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:

\begin{align*}
\text{use this new input pixel} \\
& a[i+1] = a[i] + d_0 \\
\text{while } a &< 0: \\
& \text{repeat input pixel} \\
& a[i+1] = a[i] + d_1
\end{align*}
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 iterate algorithms (3) and (4) above, but that can impede real-time scale changes 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] + dl
    if a[i] < dl:
        skip input pixel
    if a[i] >= dl:
        use input pixel
        a[i+1] = a[i] + (d0-dl)  (5)

Substituting dl=N and (d0-dl)=(-D) in the above algorithm, after K iterations, a[i] will equal KN decreased by 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.

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

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.

A span of exactly K(D-N)/D 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:

Repeated pixels = integer { K(N-D)/D + (D-1)/D }  (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 = integer { K(N-D)/D + (D-1)/D + K }, or
M = integer { (KN-1)/D } + 1  (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 equations (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