BINARY TO GRAY DOWNSCALING

Scanners typically scan binary images at 300 to 400 ppi (pixels/inch), but displays are typically around 80 ppi grayscale. How do you make a binary scanned image readable on a grayscale (or color) display? If you display it at full resolution, the width of an 8.5 inch page is larger than the display, so that horizontal scrolling is required to read each line. The letters are also about 4 times larger on the screen than they were on paper. Consequently, it is desirable to reduce the image to fit on a display.

You have the choice of making a binary or a grayscale reduced image. In both cases, subsampling is required, and the methods described above for avoiding aliasing apply here as well. Namely, you should apply a lowpass filter before subsampling. There are many ways to do this, using either linear or nonlinear filters. But the net result will have a much poorer appearance if you make a binary reduced image. In fact, a page of 8 pt font, scanned binary at 300 ppi and reduced 4x to 75 ppi binary, will be essentially unreadable on a screen, regardless of the lowpass filter used.

However, if you use a 4x scale-to-gray reduction filter, where each reduced dest pixel is taken to be the average of the 16 corresponding source pixels, the text will be readable, though difficult. (For such small type, you should use a 3x scale-to-gray reduction filter.)

The implementation of Nx scale-to-gray requires taking each NxN pixel block in the binary source and converting it to a single gray pixel with correct normalization. Consider N=4. The sum of the 16 binary source pixels can take on 17 different values, from 0 to 16. This range must be mapped to 255 to 0. The value inversion is due to the convention that a binary black pixel (value = 1) gets mapped to black (value = 0) in a grayscale image. (The gray values increase with the lightness of the pixel.)

So for each dest pixel, we use a table to find the sum of black pixels in the NxN corresponding source pixels, and then use another table to map that sum to the output grayscale value. The 4x scale-to-gray is very simple, because 8-bit lookup tables can be used to accumulate the sums in two adjacent dest pixels at simultaneously. For each possible set of 8 adjacent source pixels, the sums of the first and last 4 are packed separately in different bytes of the table word. These words are then summed for 4 adjacent source rows, giving the sums for the two dest pixels.

The second lookup table is then used twice to convert each sum to the final 8 bpp dest pixel value. The 3x scale-to-gray is more complicated because the inner loop of an efficient implementation computes 8 dest pixels from 3 rows of 24 source pixels. There is some additional shifting and masking, but the basic algorithm is the same, and again requires two different lookup tables.

We provide high-level interfaces for the following integer reduction values for scale-to-gray: 2x, 3x, 4x, 8x and 16x, in scale.c. For images scanned at 300 ppi binary, 3x scale-to-gray generates a 100 ppi grayscale image, which is comfortable to read on most computer screens. Likewise, for images scanned at 400 ppi binary, 4x scale-to-gray produces a correspondingly readable image. As usual, the low-level functions, that only take array pointers, ints and floats, and do all the bit munging, are provided inscalelow.c.

Because of the importance of creating good downscaled grayscale renderings of high resolution binary images, we also provide a high-level interface for scaling of binary to gray with an arbitrary isotropic scale factor. The function is pixScaleToGray(), and the source code describes the approach in some detail. Here is a summary of the problem. Binary images have sharp edges, so they intrinsically have very high frequency content. To avoid aliasing, they must be low-pass filtered, which tends to blur the edges.

How can we keep relatively crisp edges without aliasing? The trick is to do *binary upscaling* followed by a power-of-2 scale-to-gray, rather than doing a power-of-2 scale-to-gray followed by rescaling in grayscale. For larger reductions, where you don't end up with much detail, you can use *binary downscaling* before the scale-to-gray, to reduce the amount of computation. And for very large reductions, the scale-to-gray operation can be done first with little loss in quality of the result.

The general function is implemented with the goal of getting high quality reduced grayscale images with relatively little computation:

1. For scalefactors > 1/8, do binary upscaling before scale-to-gray.
2. For scalefactors between 1/16 and 1/8, do binary downscaling before scale-to-gray.
3. For scalefactors < 1/16, do scale-to-gray (16x) first, and then grayscale downscaling.

Source: http://www.leptonica.com/scaling.html