Image Processing
Fall 2023
Prof. George Wolberg
Homework 4

Due: Thursday, December 7
Objective: This assignment requires you to exercise your understanding of image reconstruction and antialiasing.

1) **HW_resize1D** (float *IN, float *OUT, int INlen, int OUTlen, int kernel_type, double param)

Function *HW_resize1D* scales the list of numbers stored in *IN* into a new list *OUT*. *IN* has *INlen* elements of datatype *float*. *OUT* has *OUTlen* elements. If $OUTLEN > INLEN$, then magnification must be performed. Else, minification takes place. In either case, the user specifies the filter through argument *kernel_type*, which can be set to 0, 1, 2 to refer to nearest neighbor, linear interpolation, and cubic convolution, respectively. In cubic convolution, the free variable a is passed through *param*. Values 3, 4, and 5 for *kernel_type* are reserved for windowed sinc functions. The corresponding window functions that should be used are: Hann, Hamming, and Lanczos windows. Note that parameter N for the Hann and Hamming windows (as used in the equations in the book) are passed through *param*. That is, *param* will store the width of the window. In the case of the Lanczos window, *param* is used to store the number of sinc lobes allowed to pass. For instance, the Lanczos2(x) window will be specified with $param = 2$, the Lanczos3(x) window with $param = 3$, etc.

Test *HW_resize1D* for magnification for a 1-D impulse function. Initialize an array of 32 numbers with 100 everywhere, and 200 at the center (location 16). Then magnify this list by a scale factor of 8 using all the above kernels. The output list of 256 elements should match with the samples of the respective reconstruction kernels. Submit a plot of the output for each kernel. Remember to pad the input to avoid problems at the borders where the convolution kernel falls off the edge of the image. Use pixel replication for padding.

Test *HW_resize1D* for minification for a 1-D sine wave function having values lying between 0 and 255. Initialize an array of 128 numbers with a sine wave having 16 cycles per scanline (or .125 cycles per pixel). Then minify this list by a scale factor of 8 using all the above kernels. The output list will have 16 elements. Submit a plot of the output for each kernel. Remember to pad the input to avoid problems at the borders where the convolution kernel falls off the edge of the image. Use pixel replication for padding. Also, note that unlike the magnification case, minification will cause the kernel to be stretched wider and reduced in amplitude in proportion to the scale factor.

2) **HW_resize2D** (ImagePtr I1, int new_h, int new_w, int kernel_type, int param, ImagePtr I2)

Function *HW_resize2D* scales an image stored in $I1$ and stores it in $I2$. The dimensions of the output image is *new_h* (height; or rows) by *new_w* (width; or columns). This function implements scaling separably by first resampling each image scanline with calls to *HW_resize1D*, putting the result into an intermediate buffer. Then, *HW_resize1D* is invoked on each column to yield the output image. Run this function on small images, such as eye50.pgm and text40.pgm, to demonstrate magnification. Magnify by several scale factors, including 2, 4, and 10. Over what scale factors are the various kernels acceptable?

Describe the artifacts that appear, including their location and structure. Note that you will want to convert the unsigned char elements in $I1$ to float for use in $HW\_resize1D$, and then convert back to unsigned char for storage in file $I2$.

Run $HW\_resize2D$ on large images with a lot of high frequencies (edges), such as star.pgm, to demonstrate minification. Compare the results between the point sampling (nearest neighbor) and a higher quality filter when minifying star.pgm or ramp.pgm. Why does point sampling work well for the ramp.pgm image and not for star.pgm?