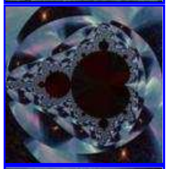
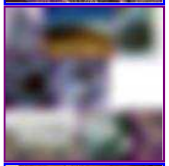




zoom, by Paul Heckbert

Paul Heckbert's zoom (user's manual) is a great little program that implements a variety of separable antialiased filters for resizing and resampling images. You can use it to blur images, to shrink images, to enlarge images, or perform any of these operations on one axis of an image.

Zoom implements the filter kernel convolution entirely in integer arithmetic and uses separable filters to keep the intermediate memory requirements reasonable.

Rich didn't write zoom, but he did add some file format support to it because he's such a big fan of its filter implementation.

File formats supported: PBM/PGM/PPM, uncompressed BMP, TIFF, GIF, JPEG, PNG and raw "dump" format. I don't guarantee that the file I/O code that I added will support every possible variation (TIFF has many variations), but I try to test with a variety of input files.

Files and Filters Supported

This 1.3 release contains everything needed to build zoom from MS VC++ 6, including TIFF via libtiff, PBM/PGM/PPM via netpbm, PNG via libpng, raw dump, GIF and BMP via C code, and JPEG via IJG's libjpeg.

`zoom -dev` reports:

```
picture devices/file formats known: dump pnm jpg png gif tif bmp
```

`filt -?` reports:

```
point           0.00
box             0.50
triangle        1.00
quadratic       1.50
cubic           2.00
catrom          2.00
mitchell        2.00
    mitchell: p0=0.888889 p2=-2 p3=1.16667
              q0=1.77778 q1=-3.33333 q2=2 q3=-0.388889
gaussian        1.25
sinc            4.00 (windowed by default)
bessel          3.24 (windowed by default)
hanning         1.00
hamming         1.00
blackman        1.00
kaiser          1.00
    kaiser: a=6.5 i0a=0.00940797
```

Size	File	Contents
199 K	zoom.zip	Console executable and preformatted man page text file.
3.99 MB	zoom-1.3.zip	Full source to above, including VC6 dsp/dsw and codec libs (TIFF, PNG, PNM, JPEG)

Why Proper Filtering Matters

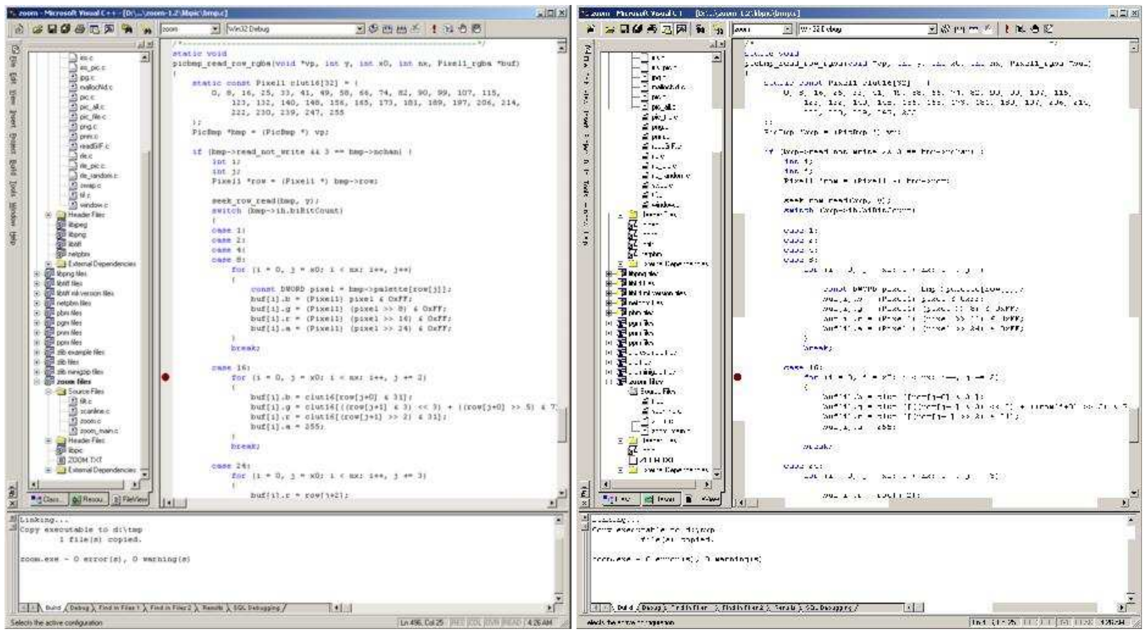
You may be wondering why you should go to all the trouble of selecting a filter when resizing an image and doing all this stuff.

Here is a visual example illustrating the difference. The following two images went through the same scaling code in zoom. The first image uses a 'mitchell' filter which trades ringing for smoothing to try and get a generic catch all filter. (There are images for which you might want to use a sharper filter and others for which you might want a smoother filter.) The second image uses the filter 'point' which just point samples the source image no filtering at all.

The source image of Visual C++ 6 debugging zoom still has a recognizable font on the filtered image, but the aliased image looks horrible. (Win32's StretchBlt is more likely to give you something more like the point-sampled image, depending on your GDI display driver.)

```
zoom -filt mitchell -square -src studio.bmp -dst
zoom-mitchell.jpg -d 0 0 512 512
```

```
zoom -filt point -square -src studio.bmp -dst
zoom-point.jpg -d 0 0 512 512
```



```
/*---------------------------------------------------------------*/
static void
picbmp_read_row_rgba(void *vp, int y, int x0, int nx, Pixel1_rgba *buf)
{
    static const Pixel1 clut16[32] = {
        0, 8, 16, 25, 33, 41, 49, 58, 66, 74, 82, 90, 99, 107, 115,
            123, 132, 140, 148, 156, 165, 173, 181, 189, 197, 206, 214,
            222, 230, 239, 247, 255
    };
    PicBmp *bmp = (PicBmp *) vp;

    if (bmp->read_not_write && 3 == bmp->nchan) {
        int i;
        int j;
        Pixel1 *row = (Pixel1 *) bmp->row;

        seek_row_read(bmp, y);
        switch (bmp->ih.biBitCount)
        {
        case 1:
        case 2:
        case 4:
        case 8:
            for (i = 0, j = x0; i < nx; i++, j++)
            {
                const DWORD pixel = bmp->palette[row[j]];
                buf[i].b = (Pixel1) pixel & 0xFF;
                buf[i].g = (Pixel1) (pixel >> 8) & 0xFF;
                buf[i].r = (Pixel1) (pixel >> 16) & 0xFF;
                buf[i].a = (Pixel1) (pixel >> 24) & 0xFF;
            }
            break;

        case 16:
            for (i = 0, j = x0; i < nx; i++, j += 2)
            {
                buf[i].b = clut16[row[j+0] & 31];
                buf[i].g = clut16[((row[j+1] & 3) << 3) + ((row[j+0] >> 5) & 7
                buf[i].r = clut16[(row[j+1] >> 2) & 31];
                buf[i].a = 255;
            }
            break;

        case 24:
            for (i = 0, j = x0; i < nx; i++, j += 3)
            {
                buf[i].r = row[j+2];
```

```
Linking...
Copy executable to d:\tmp
        1 file(s) copied.

zoom.exe - 0 error(s), 0 warning(s)
```

My favorite aliasing artifact in the above is how the scroll bars have turned into alternating bands of white and gray! Other artifacts you can see: thin lines on the tree view control are missing, braces in the C code shown in the source view are missing, the colored text is now faded and in some cases difficult to see that its colored and you can no longer read the text displayed anywhere.

Making It Fit, Pt. 2

Here's another example. Ever find some really nice image on the net, but its too big for your screen? Maybe you found a way to squish it into your available screen space, but it didn't come out looking as nice? Chances are that aliasing was introduced when the image was rescaled. I like to browse lots of space sites like NASA's image archives and find big astronomy images for my PC's desktop. Zoom is great for resampling those images to fit nicely on your desktop.

I found this HUGE NASA radar image of Salt Lake City, at 1494 by 2500 pixels. Using zoom I filtered it down to fit in a 1280x1024 screen and converted it to JPEG. Now normally JPEG would be horrible on this kind of image because it would ditch all the high frequencies and introduce artifacts. When we downsample (reduce in resolution) a digital image with zoom, we do the minimal amount of smoothing that digital signal processing theory will let us get away with. Because we are downsampling, the filter will smooth out the high frequency components that will give JPEG trouble, so when we convert to JPEG, the fine detail of the streets and causeways in the Great Salt Lake are preserved as best they can.

Filtering Without Resizing

These three images are all from the same source, just blurred differently with zoom.

You can also use zoom to apply some smoothing to an image without changing its size. This can be useful for smoothing a poorly scanned half-tone image, where the half-tone pattern is causing artifacts. By applying a little smoothing to the image, we smear out the half-tone dots and averaging them together. When you look at a half-tone original from far away it appears continuous because your eye does this sort of averaging.