“Residues” for Antialiasing Pixel Distortions

Jack Tumblin and Josh Anon, Northwestern University

Abstract

Antialiased images are delicate; almost any change can cause new aliasing that may be visible as ugly artifacts. Localized or non-linear intensity adjustments can cause new image features that are undersampled if applied only to pixels (“pixel distortions”). For example, a threshold function can add jagged edges to any image. Conventional antialiasing methods (distorting the reconstructed image, and then re-sampling) can remove these artifacts, but significantly blur the image, even in regions where no aliasing occurred.

Instead, we suppress aliasing by adding “residues” to distorted pixels. Residues are the antialiased, sampled difference between the distorted continuous image and the reconstructed image made from distorted pixels. Residues never blur needlessly: alias-free pixel distortions produce zero-valued residues. They can improve the final appearance of any nonlinear intensity-modifying operation on images, including gamma correction, NPR processes, and global or local tone mapping methods where intensity changes are severe. OpenGL hardware and short vector libraries can accelerate this antialiasing method for most applications.

Introduction

Antialiasing is important for any form of image editing. Though spatial changes such as warping and texture mapping are now routinely antialiased, errors caused by nonlinear changes to pixel intensities are not. Instead, aliasing artifacts from these distortions are usually ignored or misunderstood as an intrinsic property of digital images—as if pixel-by-pixel adjustments were the only choice available.

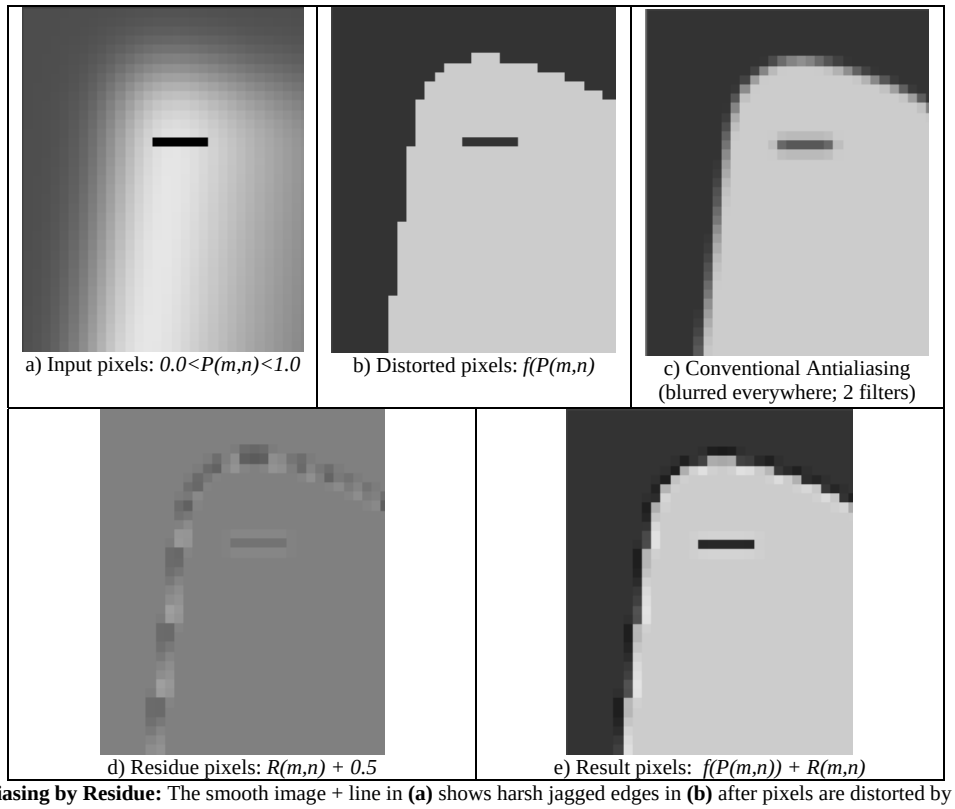

FIGURE 1: Antialiasing by Residue: The smooth image + line in **(a)** shows harsh jagged edges in **(b)** after pixels are distorted by the threshold function described below. Interpolation, distortion, re-filtering, and downsampling **(c)** blurs the edges and the line. Computed “residue” **(d)** (offset by +0.5) added to **(b)** yields the sharp but antialiased result **(e)**. Note the horizontal black line is almost unchanged; its residue is nearly zero.

Aliasing artifacts are strongest for abrupt changes to pixel intensities. For example, the threshold function *f(I) = 0.2* if *I<0.5*, else *f(I) = 0.8*)) can convert a rounded, smoothly shaded image feature into a jagged stair-step as shown in Figure 1(b), yet the black one-pixel-wide line is almost completely unaffected. Strong aliasing artifacts can also arise from even the seemingly gentle, smooth functions used for gamma correction, color correction, tone mapping, and interactive painting or touch-up. Though sometimes too subtle to see in 8-bit images, these artifacts are much more apparent in the 12-, 14-, and 16- bit images increasingly available from advanced digital cameras. Ideally, all pixel-adjusting operations

should be antialiased, but the conventional approach usually blurs the entire image excessively, as shown in Figure 1. This paper offers an approach that restricts blurring only to the distorted image components that can cause aliasing.

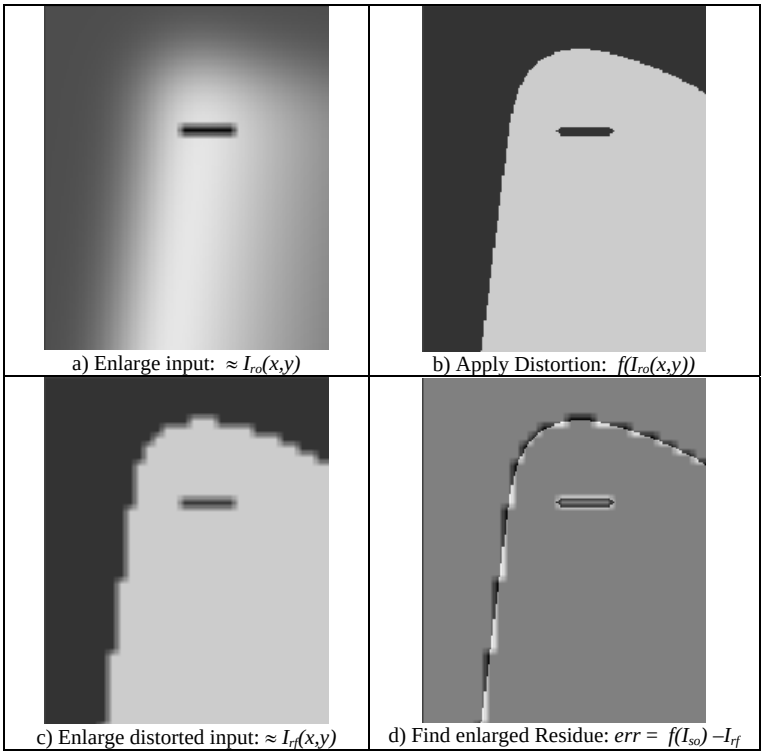

FIGURE 2: Compute Residue by Supersampling: First, approximate the original light signal $I_{ro}(x,y)$ by supersampling the input pixels $P(m,n)$ (in Fig 1a) to form **(a)**. Apply distortion $f()$; the threshold function produces **(b)**. Next, supersample the distorted pixels $f(P(m,n))$ (shown in Fig. 1b) to produce **(c)**. Pixel residue is just a reduced-size version of **(b)-(c)**, shown here as **(d)**. (Note: +0.5 added for display).

Background

Lets review basic digital image formation to define a few important terms. Recall that an optical image made by lenses and light is a smooth, continuous-valued 2D map of focal-plane light intensity, and a digital image is a sampled representation of this "*source light signal,*" $I_s(x,y)$, as we will call it. To make a digital image, simply place a grid of points on a source light signal and record the light found at each point to make the pixel grid $P(m,n)$, where (m,n) values are the discrete integer grid positions on the light signal's real (x,y) coordinates. Digital images irretrievably discard the rest of the source light signal. Attempts to sample a digital image "between" its pixel positions are actually attempts to sample this discarded source light signal. But we can't; at best we can only approximate it with a "*reconstructed light signal*" $I_r(x,y)$ made by filtering (convolving) the pixel grid with a continuous-valued interpolating filter kernel or basis function $B(x,y)$ such as the box, bilinear, bicubic [Mitchell88] or elliptical Gaussian [Heckburt86] basis:

$$I_r(x,y) = \sum_m \sum_n P(m,n)\, B(m-x, n-y) \tag{1}$$

Formally, the reconstructed light signal is a sum of basis functions $B()$ that were weighted by pixel values $P()$ and shifted to pixel positions (m,n). Our examples use the bilinear basis function (e.g. $B(x,y)=Bi(x,y)$) because it is available in most existing texture-mapping hardware:

$$Bi(x,y) = \begin{cases} (1-|x|)(1-|y|) & for\ |x|<1, |y|<1 \\ 0 & otherwise. \end{cases} \tag{2}$$

If the original pixel values $P(m,n)$ were made without aliasing, then the reconstructed light signal I_r is a very good approximation of the discarded source light signal I_s. However, if the source light signal I_s has local intensity changes that are overly abrupt, narrow or intricate, then the grid of pixel samples $P(m,n)$ can skip over some of the changes in I_s and can badly mis-estimate their values, causing aliasing artifacts. Accordingly, we define the unwanted errors not by pixels, but by the difference between source and reconstructed light signals:

$$err = (I_s - I_r) \tag{3}$$

We won't use more intuitive error notions defined only by pixels: even though the pixel-making process causes all aliasing errors, pixels alone cannot describe them all. More precisely, errors in pixels *P(m,n)* are caused by both inaccurate pixel reconstruction and the source light signal's frequency components that exceed the Nyquist rate.

Severe aliasing errors can make the reconstructed light signal very different from the source light signal. Though antialiasing takes many forms ([Crow 84], [Turkowski 82], [Dippe, Erling 85], [Greene, Heckbert 86], etc.) it can always be expressed as a process that smoothes away the rough spots in the source light signal just before it is sampled to make pixel values. But any good antialiasing method must balance competing goals; it must smooth the source light signal enough to ensure accurate reconstruction solely from pixel values, yet it must also avoid excessive smoothing that would make the digital image made from pixels look too blurry.

Suppose we already have a high quality antialiased image, and we want to change its intensities in some interesting and possibly nonlinear way, as in Figure 1. We will call this change 'distortion', and although written as function *f()*, it can be any repeatable intensity-modifying process that is defined for all points on the source light signal. How can we apply *f()* to the digital image in a manner that avoids both aliasing and blurring?

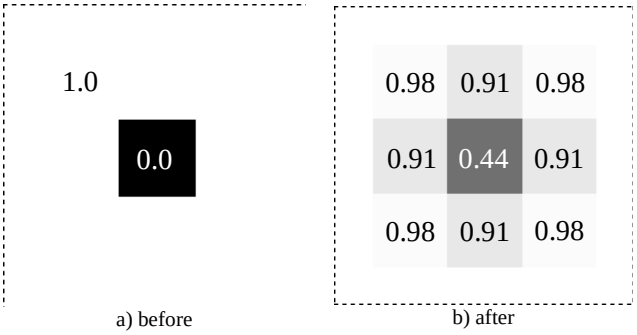

FIGURE 3: Conventional Antialiasing Method Causes Double-Blurring (a) Source image: center pixel value =0.0; all others = 1.0 in **(a)**. Pixel distortion by null distortion function ***f(i)=i*** will not change the light signal of **(a)**, but conventional antialiasing by bilinear reconstruction, distortion, bilinear prefiltering and sampling produces the doubly-blurred result **(b)**.

A naïve approach by conventional antialiasing methods is a poor choice because it can blur and soften the entire image twice. Ideally, intensity distortions for digital images should first distort the source light signal I_s with function *f()*, then apply an antialiasing filter to $f(I_s)$ to smooth away any excessive roughness caused by *f()*, and finally re-sample the resulting light signal to make new pixels. But I_s was discarded; instead we must reconstruct an approximation I_r from the source image pixels. Reconstruction I_r with any commonplace filter kernel such as *Bi* in Equation 2 will cause some blurring, and after we apply distortion *f()* we must blur again with an antialiasing pre-filter before sampling to make output pixels. The result of double-blurring everywhere can easily look worse than the aliasing problem it solves, as shown in Figure 3. The next section explains how to rearrange the process to avoid most of the double-blurring penalty, and instead apply it only to the image changes caused by the distortion, named the 'residue' below. Doubly-blurred 'residue', sampled at pixel positions and added to distorted pixel values, produces an antialiased distorted image that keeps its sharp appearance, as in Figure 1.

Method

Equation 1 describes reconstructed light signals and also reveals why antialiased digital images are delicate. If we accept I_r on the left-hand side as a good approximation for the source light signal I_s, then this equation directly links the source light signal to its pixel values *P(m,n)*. Substituting a distorted source light signal $f(I_r)$ on the left hand side and distorted pixels *f(P(m,n))* on the right will almost never keep the equation true, except for the nearly-trivial case of linear distortions given by $f(i) = a \cdot I + b$, where *a* and *b* are scalar constants.

The most obvious way to distort a digital image is to apply the distortion *f()* directly to its pixel values. Equation 1 also reveals how to compute the error induced by this obvious method. To avoid confusion, we use the subscript "*o*" for the original image before distortion, and an "*f*" for the distorted image. Aliasing induced by pixel distortion is then $err = f(I_{so}) - I_{rf}$; it is defined as the difference between the distorted source light signal and the light signal I_{rf} reconstructed from distorted pixels *f(P(m,n))*. Of course the original source light signal I_{so} is unavailable, but if we ignore any existing aliasing in the original digital image we can write $I_{so} \cong I_{ro}$. The distorted light signal $f(I_{ro})$ then acts as the source light signal for the new image, and new pixel values would, ideally, form a reconstructed light signal that exactly matches it. The aliasing error for pixel distortion is then the inaccuracy of the light signal reconstructed from distorted pixel values:

$$err(x,y) = f(I_{so}) - I_{rf} \cong f\left(I_{ro}(x,y)\right) - \sum_m \sum_n f\left(P(m,n)\right) B\left(m-x, n-y\right) \quad (4)$$

Written this way, aliasing is an error in continuous light signal values instead of pixel values. The shape of this light signal error depends on both the smoothness of the function *f()* and the smoothness of the input pixels *P(m,n)*. An abrupt change in either or both of these can cause large aliasing errors.

There are two corresponding methods for reducing this aliasing error as well. We can smooth away excessive roughness in the distorted source light signal $f(I_{ro})$, and/or we can also improve our choice of distorted pixel values, modifying *f(P(m,n))* to permit the reconstructed light signal (the double-summation in Equation 4) to more closely resemble the distorted source light signal $f(I_{ro})$. The first method is the conventional approach that causes excessive blurring in Figure 2; we use the second method instead.

Though Equation 4 describes light signal error, *err(x,y)* can also be read in a much more interesting way. It is also the 'residue" for the pixel-distorted image *f(P(m,n))*, because *err(x,y)* completely describes everything in the ideal distorted light signal $f(I_{ro})$ that can't be reconstructed from distorted pixels *f(P(m,n))*. In other words, the residue light signal *err(x,y)* contains corrections for all the mistakes that distorted pixels make in a reconstruction of the distorted source light signal.

Like any other light signal, residue *err(x,y)* can also be converted to antialiased digital image pixels *R(m,n)*. In the usual way, we apply an antialiasing filter (or 'pre-filter') to *err(x,y)* to ensure adequate smoothness, and then sample it at integer positions *(m,n)* to find each new 'residue' pixel value *R(m,n)*. Now suppose we add residue and distorted pixel values:

$$Out(m,n) = f(P(m,n)) + R(m,n) \quad (5)$$

By Equation 1, the reconstructed light signal for *Out(m,n)* pixels is just the sum of the reconstructed light signals for its two terms *f(P(m,n))* and *R(m,n)*. But *R(m,n)* reconstructs a smooth approximation of *err(x,y)*, and gives us the 'missing parts' needed to fix the mistakes in the light signal reconstructed from distorted pixels f(P(m,n)). Taken together, *Out(m,n)* provides a good approximation of the ideal distorted light signal $f(I_{ro})$, but the pixel-distorted image *f(P(m,n))* escapes the destructive double-blurring process; only the residue, the mistakes, will receive this double-blurring treatment. If pixel distortion causes no aliasing at all (as in Figure 3(a)), then the residue is zero and the pixel distorted image is not changed. In this way, Equation 5 applies blurring only where necessary: alias-free regions are not blurred.

Implementation

A wide variety of existing methods will work to compute residue pixels *R(m,n)*, but ordinary supersampling methods are easy to accelerate with graphics hardware and are also easy to explain. Supersampling represents a light signal as just a more finely-sampled digital image; it records light signal values on a fractional grid of "subpixel" points counted by *(i,j)*, overlaid on the grid of pixels located at integer *(m,n)*. Straightforward extensions to jittered supersampling might improve the results still further [Dippe'85], but is omitted here for simplicity. A light signal subpixel at *I(i,j)* is found at the image position *(x,y) = (i/M, j/N)* where *M,N* are the "supersampling rates" respectively, given by integers rarely larger than 4,4 for most practical uses. Larger *M,N* values offer greater accuracy for especially abrupt or finely detailed light signals. Light signal values are undefined between subpixel positions *(i/M, j/N)*, and as before, pixel values are recorded at integer *(x,y)* positions using $P(m,n) = I(mM, nN)$. Supersampling allows us to approximate the source light signal I_{so} as the supersampled light signal $I_{ro}(i,j)$ reconstructed from original pixel values:

$$I_{so} \cong I_{ro}(i,j) = \sum_m \sum_n P(m,n) B\left(m - (i/M), n - (j/N)\right) \quad (6)$$

Though it may look intimidating, Equation 6 is only an image resizing operation; $I_{ro}(i,j)$ is just the original digital image enlarged to supply $M \times N$ subpixels from each pixel using the interpolating kernel *B(x,y)*. Distorting each subpixel value then forms an approximation of the ideal distorted light signal:

$$f(I_{so}) \cong f\left(I_{ro}(i,j)\right) \quad (7)$$

By this same method, the light signal I_{rf} reconstructed from distorted pixel values *f(P(m,n))* is:

$$I_{rf} \cong I_{rf}(i,j) = \sum_m \sum_n f\left(P(m,n)\right) B\left(m - (i/M), n - (j/N)\right) \quad (8)$$

The supersampled version of the aliasing error light signal *err(x,y)* from Equation 4 is then:

$$err(x, y) \cong errss(i, j) = f(I_{ro}(i, j)) - I_{rf}(i, j) \tag{9}$$

Antialiasing and sampling the light signal *err(x,y)* makes residue pixels *R(m,n)*. Supersampling makes this a discrete filtering step, and using the bilinear filter kernel *Bi(x,y)* from Equation 2 we have:

$$R(m, n) = \frac{1}{MN} \sum_{s=1-M}^{M-1} \sum_{t=1-N}^{N-1} errss(mM - s, nN - t) Bi(s / M, t / N) \tag{10}$$

Though formidable-looking, this discrete filter-and-downsample step is nothing new. For *M,N = 4,4* supersampling, Equation 10 describes each residue pixel value as just the bilinearly-weighted average of the 7x7 surrounding subpixels of *errss(i,j)*. The residue pixels *R(m,n)* from Equation 9 added in Equation 5 make the final antialiased output image.

This antialiasing method is easy to accelerate in either hardware or software. In software, separate threads can compute $f(I_{ro})$ and I_{rf}, the two parts of the *err(i,j)* light signal, or even their separate contributions to each residue pixel *R(m,n)*. Libraries for short vector processing can also accelerate residue calculations on modern processors, such as Altivec extensions for the PowerPC G4 [Motorola 01] or SIMD extensions for Intel Pentium III and 4 [Intel 02].

OpenGL-compatible graphics accelerators can also help, and some recent cards support 16-bit-per-color (48-bit) images or even floating-point texture signals that are especially well suited for both high dynamic range (HDR) images and severely nonlinear intensity adjustments. Also, loading the input image pixels *P(m,n)* as a texture map using `GL_TEXTURE_RECTANGLE_EXT` will allow your application to accept readily varied image sizes. Reconstruction and supersampling to make $I_{ro}(i,j)$, $I_{rf}(i,j)$, and *err(i,j)* requires nothing more than rendering the texture map enlarged by (M,N) using the `GL_LINEAR` setting to ensure bilinear interpolation, and then reading this 'supersampled' image into a buffer using `glReadPixels`. Extension `GL_EXT_BLEND_SUBTRACT` can help compute the *err(i,j)* light signal, and this light signal reduced in size by (M,N) forms residue pixels *R(m,n)*.

Non-optimized source code for antialiasing by the residue method described here are available at http://cs.northwestern.edu/~jet and ~josha

Conclusions

Antialiasing is rarely difficult to do, but always confusing to explain. We took great care to keep this explanation intuitive and short, but meticulously accurate. Though Fourier analysis and a frequency-domain explanation of aliasing can strengthen our arguments, it is not needed for implementation or a clear understanding of 'residues', and might needlessly discourage many interested readers. Many good texts on paper and online offer rigorous mathematical tutorials on aliasing, resampling, and filter design such as [Oppenheim89], [Lyons96], or [http://www.bores.com/courses/intro/index.htm].

The "light signals and pixels" description of aliasing given here applies equally well to both spatial and intensity changes, but the residue solution does not. Though computing residues for translation, rotation, scale and generalized warping is possible, it is rarely worthwhile, because supersampling grids for I_{ro} and I_{fr} are misaligned, irregular, or *M,N* values needed may be impractically large. We found it mildly instructive to examine bilinear residues for image translation by a fractional pixel distance (e_x, e_y); solved analytically, the antialiased results exactly match those from offset sampling of the reconstructed I_{ro} made by Mitchell and Netravali's piecewise cubic filter [Mitchell88] with B=1 C=0, which in turn matches Parker's 'sharp spline' filter [Parker 83] devised by other means. These filters or image pyramids or MIP-maps [Williams 83] already offer a faster and more general (though non-directional) solution for image spatial changes, but the residue methods offer a much better solution for antialiasing image intensity changes.

Bibliography

Crow, Franklin C. "Summed-area tables for texture mapping." *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* July 1984: 207-212.

Dippe, Mark A.Z. and Erling, Henry W., "Antialiasing Through Stochastic Sampling." *Computer Graphics*, (Proceedings of SIGGRAPH '85) 19.3 (July 1985): 69-78.

Greene, Ned and Heckbert, Paul “Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter.” *IEEE Computer Graphics and Applications* June 1986: 21-27.

Guenter, Brian and Tumblin,Jack. “Quadrature Prefiltering for High Quality Antialiasing.” *ACM Transactions on Graphics* 15.4 (1996): 332-353.

Intel Corporation. *IA-32 Intel Architecture Software Developer’s Manual*. Illinois: Mount Prospect, 2001.

Lyons, Richard G., *Understanding Digital Signal Processing*, Prentice-Hall PTR, 1st edition, 1996.

Mitchell, Don and Netravali, Arun. “Reconstruction Filters in Computer Graphics.” *Computer Graphics* (Proveedings of SIGGRAPH ‘88) 22.4: 221-228.

Motorola Literature Distribution. *AltiVec Technology Programming Environments Manual*. Colorado: Denver, 2001.

Oppenheim, Alan V., and Schafer, Ronald W.. Chapters 1-3. *Discrete Time Signal Processing*. Prentice-Hall, 1998. 1-148.

Parker, Anthony; Kenyon, Robert and Troxel, Donald “Comparison of Interpolating Methods for Image Resampling.” *IEEE Transactions on Medical Imaging* MI-2.1 (March 1983): 31-39.

Turkowski, K.. “Anti-Aliasing through the use of Coordinate Transformations.” *ACM Transaction on Graphics* 1.3 (July 1982): 215-234.

van Overveld, C.W.A.M.. “Color Waves: A Simple Heuristic for Choosing False Colors.” *Journal of Graphics Tools* 2.4 (1997): 45-50.

Williams, Lance, “Pyramidal Parametrics.” *Computer Graphics* (Proceedings of SIGGRAPH ‘83) 17. 3 (1983): 1-11.