The UnBlock algorithm

John P. Costella

Mornington, Melbourne, Australia
jpcostella@hotmail.com; assassinationscience.com/johncostella

(January 16, 2006)

Abstract

This paper describes the philosophical and mathematical formalism underlying the UnBlock algorithm for removing block artifacts from JPEG and MPEG images.

1. Introduction and motivation

Since the development of the JPEG and MPEG standards, many researchers have sought to remove the "blocking" artifacts that the compression method introduces. Many solutions are mathematically complicated, and hence computationally slow. Many solutions require that an end-user "fine-tune" parameters based on visual observation of the effects. Many solutions degrade the fidelity of sharp edges in the original image.

The author's goal was to create a deblocking method that requires no user input, executes as fast as possible, and does not degrade features of the original image.

The result is the UnBlock algorithm. It takes any image that may have been subjected to this form of blocking artifact, and automatically corrects it as far as possible. UnBlock is completely a postprocessing step: it takes images that have been previously encoded and decoded by the JPEG or MPEG algorithm, and corrects them. UnBlock does not require the encoded JPEG or MPEG source file at all. UnBlock does not require any user input: the algorithm determines which edge transitions are (most likely to be) blocking artifacts by analyzing the image itself. There are no "tunable parameters".

At the time of writing, it is not known whether any of the other existing algorithms fulfil the abovementioned design criteria better than the UnBlock algorithm.

2. The problem and its solution

2.1. The nature of the problem

JPEG and MPEG images are broken down into 8×8 blocks, and each such block is then encoded in terms of its Fourier (Discrete Cosine Transform) components. (For color images, the two chrominance components are typically downsampled by a factor of two in each direction before being compressed, so that in the final image the chrominance components are made up of 2×2 pixel squares in 16×16 pixel blocks.) Different compression ratios are obtained by dedicating varying numbers of bits to this information. However, at most compression ratios, the edges of the 8×8 blocks themselves becomes visible, to a greater or lesser extent, and this degrades the fidelity of the perceived image, because the human visual system is very sensitive to edges.

2.2. A naive approach

Deblocking algorithms are generally based on the observation that block artifacts are most noticeable when the true image is varying relatively slowly and smoothly over the size of a block. In such cases, each color component varies smoothly towards the boundaries, but there are frequently large discontinuities across the boundaries.

The most naive approach is to simply measure the discontinuity in each component's value across the boundary, and, based on some criteria (either user-supplied or derived from the image itself), smooth out some or all of the discontinuity in some way.

However, if the true intensity gradient across that boundary (on the uncompressed image) is nonzero, this naive method will, in general, end up removing changes in intensity that should remain. This can lead to the introduction of new boundary artifacts that would be as bad or worse than the block discontinuities we are removing from other parts of the image: a *lack* of gradient across two pixels in the middle of a region that otherwise has a smooth gradient is just as bad, visually, as *doubling* the gradient across those two pixels by means of a discontinuity.

Clearly, if all we measure are boundary discontinuities, it will in general be impossible to correct a discontinuity of a given size ε in any image without erroneously "overcorrecting" regions of smooth gradient of up to ε per pixel. This is particularly problematical for relatively high quality images, for which the boundary discontinuities to be corrected are, on average, relatively small, and hence typically comparable to linear gradients over substantial regions of the image.

2.3. A linear model approach

A better approach than the naive measurement of only intensity discontinuities would be to assume a linear variation of intensity on each side of a boundary. We would need at least two pixels on each side of the boundary in order to estimate the derivative on each side. If we extrapolate to the position half-way between the two boundary pixels, from both sides, we can then compare the two extrapolations; the difference between them would then be the "discrepancy" that may need to be removed.

Such a method would automatically correct for the "tilt" of a region of smooth gradient; in other words, if the smooth gradient is continued across the boundary, the discrepancy is zero; in contrast, a *lack* of continuation of the smooth gradient (*i.e.*, if the pixels on either side of the boundary had exactly the same intensity) would, rather, be detected as an unexpected discrepancy in the negative direction.

2.4. How much discrepancy should be removed?

At this stage we need to remember that even a linear model in each block is only an approximation. It may be a bad approximation if there are strong high-frequency components (rapid changes in intensity) around the boundary. Before removing any purported discrepancy, we should first determine whether it is larger than what we would expect.

But what should we expect? We don't have the original image, so we can't be sure that there isn't a sharp discontinuity in the original image that happens to lie right on the boundary. If we removed all discrepancies unconditionally, we would automatically blur out any such sharp edge that happens to unfortunately fall in the wrong place.

It is impossible to avoid such an error in every possible case, but we can do so statistically. For any given image, the distribution of the magnitudes of the block boundary discontinuity errors will be determined by the amount of compression employed (the "quality factor"). By computing the distribution of boundary discrepancies across the entire image, and comparing it to the distribtuion of "discrepancies" calculated in the *interior* of the blocks, we can quantify, statistically, the amount to which the compression has introduced block artifacts.

Now, what are we to do with these statistical distributions? A reasonable ansatz is that *we should reduce each boundary discrepancy by an amount that will make the interior and boundary discrepancy distributions the same*, as far as possible.

The method for doing this in practical terms will be discussed further below.

2.5. A parabolic model

In the above we noted that a linear model would be preferable to simply comparing intensities on either side of a boundary, because it would avoid the problem of removing a linear gradient across the boundary present in the original image. However, for very little effort we can do even better than a linear model. A parabolic model would allow us to extrapolate to the boundary more accurately, and would also allow us to extrapolate blocking discontinuity artifacts in the *gradient* of the intensity, which the human eye is also sensitive to. To estimate the second derivative, we would need at least three pixels on either side of the boundary.

Now, how far could we extend this approach? How high a degree of polynomial should we use?

In answering this question we must distinguish between two distinct tasks: that of *measuring the boundary discrepancies* and that of *correcting for these discrepancies.*

2.6. The measurement of discrepancies

In terms of the *measurement* of discrepancies, there is a general rule of thumb in practical uses of mathematics that guides us: a constant approximation to a general function is generally poor; a linear approximation is much better, but still not optimal; a parabolic approximation is even better, allowing a representation of curvature; a cubic can sometimes be better, but in general will become too unstable, and magnifies errors if we wish to perform extrapolations.

In our case, we will only need to extrapolate by half a pixel spacing, so it is not *a priori* clear whether we would do better with a cubic approximation than with a parabolic.

To answer this question definitively, it is necessary to do some research with actual JPEG images. The analyses performed in Sec. 3 below were originally carried out with linear, parabolic, and cubic models. Experimentally, it is found that a parabolic model gives the greatest discrimination between boundary and internal discrepancies; the linear and cubic models both provide discrimination that is significantly inferior to the parabolic model. These results hold across the full range of JPEG quality factors, and for both intensity discrepancies and derivative discrepancies.

Thus, we should use a parabolic model in our measurement of discrepancies.

2.7. The correction of discrepancies

Correcting for the measured discrepancies (after we have "discounted" them appropriately, as described in Sec. 2.4) is a completely different situation to that of measurement. We would like to correct any computed intensity discrepancy and any computed intensity gradient discrepancy at a given block boundary. Clearly, since the blocking phenomenon is objectionable precisely because of its discontinuous nature, our objective will be to "spread" the discrepancies out away from the boundaries as far as possible.

However, we also want the UnBlock algorithm to be as fast as possible. It would simplify matters greatly if the image could be corrected "in place", with each boundary being corrected individually, walking along each row and correcting the vertical boundaries, and then walking down each column and correcting the horizontal boundaries. If the correction of one boundary were to affect neighboring blocks in such a way that the computation of discrepancies for subsequent boundaries would be altered, then we would have an intertwining of the measurement and correction phases of the algorithm that would necessitate a complicated solution, almost inevitably requiring the global correction of an entire row or column in one step, rather than a local correction of each boundary one at a time. We wish to avoid such a complicated solution at all costs.

Thus, our aim is to construct a method of correction that spreads out discrepancies away from the boundary as much as possible, but which does not alter the computation of discrepancies for subsequent boundaries in any way.

This might seem to imply that we should only alter the four pixels either side of a boundary in order to remove discrepancies at that boundary, so that the entire image is "sliced" into strips eight pixels wide, with the original block boundaries running down the centers of the strips. Each boundary would then effectively "own" its strip of eight pixels, so that the correction of one boundary would not affect the pixels belonging to the strip of another boundary.

However, such a strip ownership model brings with it a large performance penalty: it leaves residual blocking artifacts in the form of intensity variations that oscillate up and down with the original blocking period, *i.e.*, with a period of eight pixels, sometimes called the "washboard effect". The reason for this is somewhat subtle. The strip model effectively makes the ansatz that the original block intensities are accurate in the centers of each block, and only build up inaccuracies as we move away from the center of each block towards the boundaries. This "Taylor expansion around each block center" view is not correct, because JPEG and MPEG compression works in terms of Fourier coefficients, not Taylor series, so, if anything, we should assume that the *average* intensity across the block is accurate, not the value at the center.

How, then, are we to avoid the washboard artifact, without creating an algorithm that is intractably complicated?

With some clever footwork we can avoid the washboard problem. The strip design model was based on our wish to remove the possibility of any interaction between the correction of one boundary and the computation of discrepancies for the next. The strip model fulfils this criterion, but it is more restrictive than is necessary. It only allows a given discrepancy at a boundary to be spread out half a block either side of the boundary. Why not spread it out a *full* block either side of the boundary? For sure, this will mean that we will be altering pixels that are near the two adjacent boundaries, and hence we will be altering the values used in the discrepancy calculations for those two adjacent boundaries. But does this mean that the *actually computed discrepancies* will inevitably be altered as a consequence?

No it does not. All we need to do is ensure the satisfaction of three criteria: the increase to the intensity function must vanish at the adjacent block boundaries; its spatial derivative must vanish at the adjacent block boundaries; and our measurement model must be able to model the correction function sufficiently faithfully that it will correctly determine that the intensity value and its first derivative are vanishing. If these three criteria are satisfied, then the intensity values of the pixels near the adjacent boundaries will be changed, but the discrepancy values calculated at those boundaries will be the same as those that would have been computed if the pixels' values hadn't been changed at all.

This concludes the general description of the UnBlock algorithm's design criteria. In the next section these criteria are given an explicit mathematical form.

3. Mathematical formulation

3.1. Notation and conventions

For definiteness of language, the following is written in terms of a vertical JPEG or MPEG block boundary. (The same algorithm is used to filter across horizontal block boundaries.) Hereinafter, the vertical JPEG or MPEG block boundaries will be simply referred to as "boundaries".

We will consider just one component channel of the image at a time. Because JPEG and MPEG compression compresses the luminance (Y) channel differently to the chrominance (C_b and C_r) channels, we first transform color images to (Y, C_b, C_r) space before performing the UnBlock algorithm.

After this transformation is performed, we perform the UnBlock algorithm on just one channel (Y, C_b, or C_r) at a time. For the chrominance channels of a color image, we perform the UnBlock algorithm on the downsampled image, and smooth it out suitably when we upsample it to the original pixel resolution; the channels Y, C_b, and C_r are recombined into red, green, and blue components at the end of the process.

We will refer to the intensity of the given channel being UnBlocked as I. In general, I will be bounded to the range $0 \leq I \leq 255$, although this constraint will not play an explicit role in most of the following; any practical implementation of the UnBlock algorithm simply has to ensure that any computed intensity values that fall outside this range are clipped to 0 or 255 as appropriate.

We measure along the row of pixels with a continuous x axis. For the luminance channel, the scale of x is set so that one pixel spacing is one unit of x; for the chrominance channels, we work in terms of downsampled pixels, so that one unit of x actually corresponds to two pixels of the original image. (Hereinafter, "pixel" refers to downsampled pixels, except where noted.) In each case there are eight units of x between block boundaries.

We set $x = 0$ to be at the "boundary" of interest between two blocks, which is defined to be the position that is on the boundary of the two pixels surrounding the boundary. We refer to this as the "central boundary". The next boundary to the left of the central boundary, which is referred to as the "left adjacent boundary", is then located at $x = -8$; and the next boundary to the right of the central boundary, which is referred to as the "right adjacent boundary", is located at $x = +8$. (For the purposes of the following, we assume that these two adjacent boundaries exist; no substantial complications arise for the cases of the central boundary being the leftmost or rightmost in the image.)

We assume that a pixel's intensity represents its value at the center of the pixel, and that the pixel is located "at" that central location. Thus, the pixel on the left of the central boundary is at position $x = -0.5$, and the pixel on the right of the central boundary is at position $x = +0.5$.

According to the considerations of the previous section, we concentrate our attention on the sixteen pixels straddling the boundary, namely, the eight pixels between the left adjacent boundary and the central boundary, together with the eight pixels between the central boundary and the right adjacent boundary. For simplicity, the intensity values of these sixteen pixels are given the symbol i_n, where n runs from 1 to 16, so that

$$i_n = I(n-8.5).$$

For example, the intensity of the pixel on the left of the central boundary i_8, which is also $I(-0.5)$. The dual notation is convenient, because the UnBlock algorithm will ultimately deal only with the integral values i_n; the continuous function $I(x)$ is a mathematical construction that guides the formulation of the algorithm, but then drops away from view.

3.2. The measurement model

For the task of measuring the discrepancies at the central boundary, the discussion of Sec. 2.6 dictates that we fit a quadratic model to the three pixels on each side of the boundary. It will prove to be simplest to represent this by performing a Taylor series expansion in $I(x)$ around $x = 0$, and retaining terms up to second order:

$$I(x) \approx I(0) + I'(0)\,x + \frac{1}{2} I''(0)\,x^2. \tag{1}$$

If there were no discrepancies across the boundary, and if the true image intensity is able to be described sufficiently accurately by a parabolic function, then the quadratic model (1) would hold across the complete domain being measured, namely, $-2.5 \le x \le 2.5$. In the following, we use equality signs with the understanding that it is subject to this level of approximation.

We now fit the model (1) to the left and right sides of the boundary separately. Let us concentrate on the left side first. The intensity values i_6, i_7, and i_8 give us the equations

$$
\begin{aligned}
i_6 &= I_L(0) - \frac{5}{2} I'_L(0) + \frac{25}{8} I''_L(0). \\
i_7 &= I_L(0) - \frac{3}{2} I'_L(0) + \frac{9}{8} I''_L(0), \\
i_8 &= I_L(0) - \frac{1}{2} I'_L(0) + \frac{1}{8} I''_L(0),
\end{aligned}
$$

where the subscript "L" denotes the fact that this is the parabolic fit to the left side of the central boundary. Inverting this matrix system of equations, we obtain

$$
\begin{aligned}
I_L(0) &= \frac{15 i_8 - 10 i_7 + 3 i_6}{8}, \\
I'_L(0) &= 2 i_8 - 3 i_7 + i_6. \\
I''_L(0) &= i_8 - 2 i_7 + i_6.
\end{aligned} \tag{2}
$$

We can perform the same analysis on the right-hand side of the boundary by replacing i_6, i_7, and i_8 with i_{11}, i_{10}, and i_9 respectively, and replacing $I_L(0)$, $I'_L(0)$, and $I''_L(0)$ with $I_R(0)$, $-I'_R(0)$, and $I''_R(0)$ respectively, to obtain

$$
\begin{aligned}
I_R(0) &= \frac{15 i_9 - 10 i_{10} + 3 i_{11}}{8}, \\
I'_R(0) &= -2 i_9 + 3 i_{10} - i_{11}. \\
I''_R(0) &= i_9 - 2 i_{10} + i_{11}.
\end{aligned} \tag{3}
$$

We won't actually need the values of $I''_L(0)$ or $I''_R(0)$, but we list them here for completeness, and to show that we obtain the first-principles expressions for the second derivative (which in this model is constant on each side of the boundary).

3.3. Calculating the discrepancies

We define the *intensity discrepancy* u to be the increase in the extrapolated $I(0)$ as we cross the boundary from left to right:

$$
u \equiv I_R(0) - I_L(0) = \frac{15(i_9 - i_8) - 10(i_{10} - i_7) + 3(i_{11} - i_6)}{8}. \tag{4}
$$

Note that, since the values i_n are integers, our calculations can proceed with integer arithmetic. The calculation of u requires three integer multiplications, five additions and subtractions, and the division by 8 can be effected by means of a binary shift right by three bits. (To ensure correct rounding, we first add 4 before doing the shift right, so that we actually need six additions and subtractions in total.)

Similarly, we define the *derivative discrepancy* v to be the increase in the extrapolated $I'(0)$ as we cross from left to right:

$$
v \equiv I'_R(0) - I'_L(0) = -2(i_9 + i_8) + 3(i_{10} + i_7) - (i_{11} + i_6). \tag{5}
$$

The calculation of v requires three integer multiplications, and five additions and subtractions.

3.4. Catering for missing pixels

In the previous section, we assumed that all six pixels were available to us when calculating discrepancies. However, on the right edge of a JPEG or MPEG image there can be partial blocks if the width of the image is not a multiple of eight.

In terms of computing discrepancies *internal* to a block, this phenomenon will not come into play. As discussed in Sec. 2.4 above, we only compute internal discrepancies for those blocks for which the right boundary is present; hence, all eight pixels of the block itself must be present. In line with the discussion of Sec. 2.4, we only need the middle six pixels of the block anyway.

For boundary discrepancy calculations, however, the situation is different. If there are three or more pixels to the right of the boundary, then the discrepancy calculations are not affected. However, if there are only one or two pixels to the right of the boundary, then we must modify our procedures. (There must be at least one pixel to the right of the boundary for a boundary to be considered "present".) We will treat each case in turn.

If there are only two pixels to the right of the boundary, then we can still model the right side of the boundary with a linear function:

$$I(x) \approx I(0) + I'(0)\,x. \tag{6}$$

The intensity values i_9 and i_{10} give us the equations

$$\begin{aligned} i_9 &= I_R(0) + \frac{1}{2} I'_R(0), \\ i_{10} &= I_R(0) + \frac{3}{2} I'_R(0). \end{aligned}$$

Inverting this matrix system of equations, we obtain

$$\begin{aligned} I_R(0) &= \frac{3i_9 - i_{10}}{2}, \\ I'_R(0) &= i_{10} - i_9. \end{aligned}$$

Together with the expressions (2), we get

$$u^{(5)} \equiv I_R(0) - I_L(0) = \frac{-3i_6 + 10i_7 - 15i_8 + 12i_9 - 4i_{10}}{8} \tag{7}$$

and

$$v^{(5)} \equiv I'_R(0) - I'_L(0) = -i_6 + 3i_7 - 2i_8 - i_9 + i_{10}, \tag{8}$$

where the superscript "(5)" denotes the fact that we only have five pixels to work with.

Finally, if there is only one pixel to the right of the boundary, then all we can do is model the intensity in the right block as a constant, namely, $I_R(0) \approx i_9$. We then obtain

$$u^{(4)} \equiv I_R(0) - I_L(0) = \frac{-3i_6 + 10i_7 - 15i_8 + 8i_9}{8}. \tag{9}$$

(The correction of a single pixel to the right of a boundary on the right edge of an image is of very low importance, so it is not worthwhile to perform any more elaborate an extrapolation than this.) In this case, we have $v^{(4)} = 0$, because there is no way to compute the intensity derivative on the right side of the boundary.

3.5. Discounting the discrepancies

As discussed in Sec. 2.4, once we measure an intensity discrepancy u or a derivative discrepancy v across the central boundary, we do not automatically seek to remove it completely. Rather, we need to first determine *how much* of it should be removed.

The general criterion guiding this determination was discussed in Sec. 2.4: we adjust the boundary discrepancies only to such an extent that their distribution becomes equal to that of the internal discrepancies.

Now, it may seem that a statistical analysis would violate our design requirement that the UnBlock algorithm be as fast as possible. However, this is not the case. The computed discrepancy values will all be integers, with fixed range: -255 to $+255$. (This is definitely the case for u, for which it would not make sense for a discontinuity to be greater than 255 in magnitude; even though the discrepancy formulas can lead to greater values, we should clamp the result into this range. For v, we theoretically could jump from a gradient of -255 per pixel to $+255$ per pixel across a boundary, but the average gradient on each side of the boundary, over two or more pixels, must of necessity be less, so again it would be eminently sensible to clamp v values to this range.)

Now, if we take the absolute value of each discrepancy, this leaves us with just 256 possible values. (In the following statistical analysis, when we refer to a "value" we will be referring to magnitudes only, namely, the result after taking the absolute value.) It is simple to go through the image and construct a frequency table, for each of the discrepancy types, for the interior regions and the boundaries, by incrementing totals in arrays offset by the value of the discrepancy. To keep the following statistical arithmetic simple, and reduce the computational load, it is sufficient to compute an interior discrepancy just once for each block, so that (if we ignore partial blocks) the total number of interior values is the same as the total number of boundary values. For definiteness, we use the middle six pixels of a block to calculate the interior discrepancies.

We must now determine some algorithm by which the boundary discrepancy distributions can be modified to match the corresponding interior discrepancy distributions. A workable way to do this is to first compute the cumulative distributions. We then walk through the boundary discrepancy cumulative frequency distribution, from 0 up to 255. For each value of boundary discrepancy, we find that value of the interior discrepancy for which the cumulative distributions most closely match. This value of the interior discrepancy is allowed to remain when we correct for the discrepancy; *i.e.*, the discrepancy is "discounted" by the value of the interior discrepancy that has the most closely matching cumulative distribution value.

An example might make this clearer. Imagine that we have walked through the first ten boundary discrepancy values (0 through 9), and we are now analyzing a boundary discrepancy of 10. Imagine that there are, say, 27,000 boundary values that are less than or equal to 10. We look through the interior cumulative distribution, and we find that there are, say, 27,200 interior values less than or equal to 3. By our above ansatz, we therefore deem that if we measure a boundary discrepancy magnitude to be 10, then we should reduce it to a magnitude of 3. We don't remove the full discrepancy of 10; we "discount" the discrepancy by 3 units, and only remove a discrepancy of 7. Of course, since actual discrepancies are both positive and negative, we keep the sign of the discrepancy when reducing its magnitude.

3.6. The correction functions

Following the discussion in Sec. 2.7, we wish to correct the (discounted) discrepancies u and v by adjusting the sixteen pixel intensities i_1 through i_{16}, such that the underlying continuous correction function, which we will denote $\Delta I(x)$, and its spatial derivative, $\Delta I'(x)$, vanishes at each of the adjacent boundaries:

$$\Delta I(\pm 8) = 0, \qquad \Delta I'(\pm 8) = 0. \tag{10}$$

We assume that the corrections for u and v are independent of each other, and are linearly superposed, so that

$$\Delta I(x) = \Delta I_u(x) + \Delta I_v(x).$$

Indeed, we shall consider the u and v corrections separately, because different considerations come into play for each of them.

Let us start by considering the correction of an intensity discontinuity u. For definiteness we will discuss the case in which u is positive, *i.e.*, the (extrapolated) intensity on the right side of the boundary is higher than that on the left (but of course the formulas will work equally well when u is negative). According to the discussion of Sec. 2.7, we wish to increase the intensity values of pixels in the left block, and decrease the intensity values of the pixels in the right block, so that, at the central boundary, the discrepancy u is removed. We treat the left and right blocks symmetrically, so that the (extrapolated) increase in intensity in the left block at the central boundary should be $u/2$, and the (extrapolated) increase in intensity in the right block at the central boundary should be $-u/2$.

Now, the intensity changes must satisfy the constraints (10) at the adjacent boundaries. Within each block, this implies three constraints in total: the two constraints in (10), and the required value of $\Delta I_u(0)$. The simplest way to satisfy three constraints is with a parabolic function. Let us consider, first, the left block. The two constraints in (10) can clearly be satisfied if we put the turning point of the parabola at $(-8, 0)$, namely,

$$\Delta I_{uL}(x) = a(x+8)^2.$$

We determine the value of a by requiring $\Delta I_{uL}(0) = u/2$, namely, $a = u/128$, so that

$$\Delta I_{uL}(x) = \frac{u}{128}(x+8)^2. \tag{11}$$

Likewise, for the right block we can satisfy the two constraints in (10) by putting the turning point of the parabola at $(8, 0)$, namely,

$$\Delta I_{uR}(x) = -a(x-8)^2,$$

and requiring that $\mathit{\Delta} I_{uR}(0) = -u/2$, so that again $a = u/128$, and

$$\mathit{\Delta} I_{uR}(x) = -\frac{u}{128}(x-8)^2. \tag{12}$$

Note that the correction of the intensity discrepancy has not only altered the values of the extrapolated intensities at the central boundary, but it has also changed the *derivative* of the intensity at the central boundary:

$$\mathit{\Delta} I'_{uL}(0) = \mathit{\Delta} I'_{uR}(0) = \frac{u}{8}.$$

This does not violate our requirement that the correction of u not affect the computation of v, because the change in the derivative is *the same* coming from both sides of the boundary: there is an increase in the *gradient* introduced at the boundary, but there is no gradient *discrepancy* introduced by these additions to the intensities. This additional contribution to the derivative simply says that the gradient at the central boundary has been increased, in order to compensate for the intensity discrepancy that has been removed. Note that, *had* we decided to spread u out *linearly* over the domain $-8 \leq x \leq 8$, the gradient of this increase would have only been $u/16$, namely, only half as much as we have computed above. The reason that our increased gradient at the central boundary is twice as large is our requirement that the alterations not change the intensity derivatives at the adjacent boundaries, which necessitated a two-parabola, rather than a linear, spreading out of the discrepancy: the gradient is zero at the adjacent boundaries, and increases linearly in magnitude as we approach the central boundary; and since its average value must be $u/16$, its value at the central boundary is consequently twice as large, namely, $u/8$.

It is now straightforward to determine the required changes to the sixteen pixel intensity values i_n in order to correct an intensity discrepancy u. Using (11), we obtain

$$\begin{aligned} \mathit{\Delta} i_1^u &= \frac{u}{512}, & \mathit{\Delta} i_2^u &= \frac{9u}{512}, & \mathit{\Delta} i_3^u &= \frac{25u}{512}, & \mathit{\Delta} i_4^u &= \frac{49u}{512}, \\ \mathit{\Delta} i_5^u &= \frac{81u}{512}, & \mathit{\Delta} i_6^u &= \frac{121u}{512}, & \mathit{\Delta} i_7^u &= \frac{169u}{512}, & \mathit{\Delta} i_8^u &= \frac{225u}{512}. \end{aligned} \tag{13}$$

Now, the denominators of 512 here are convenient, because we can implement them by means of a shift right by nine bits, rather than using an integer division each time. However, we can do even better. Recall that our intensity values have limited quantization resolution: they are integers from 0 to 255. Thus, any fraction need only be represented accurately enough with a denominator of 256, and the effect will be the same for our purposes. In the case of (13), it is accurate enough if we simply halve each of the denominators and round down, resulting in

$$\begin{aligned} \mathit{\Delta} i_1^u &= 0, & \mathit{\Delta} i_2^u &= \frac{u}{64}, & \mathit{\Delta} i_3^u &= \frac{3u}{64}, & \mathit{\Delta} i_4^u &= \frac{3u}{32}, \\ \mathit{\Delta} i_5^u &= \frac{5u}{32}, & \mathit{\Delta} i_6^u &= \frac{15u}{64}, & \mathit{\Delta} i_7^u &= \frac{21u}{64}, & \mathit{\Delta} i_8^u &= \frac{7u}{16}. \end{aligned} \tag{14}$$

Our judicious choice of rounding has reduced our computational load to five integer multiplications, and a total of 33 bit shifts to the right (which are fast). (We also include seven integer additions so that the divisions are correctly rounded.) By symmetry, or using (12) with the same considerations, we obtain

$$\begin{array}{llll} \Delta i^u_9 = -\dfrac{7u}{16}, & \Delta i^u_{10} = -\dfrac{21u}{64}, & \Delta i^u_{11} = -\dfrac{15u}{64}, & \Delta i^u_{12} = -\dfrac{5u}{32}, \\ \Delta i^u_{13} = -\dfrac{3u}{32}, & \Delta i^u_{14} = -\dfrac{3u}{64}, & \Delta i^u_{15} = -\dfrac{u}{64}, & \Delta i^u_{16} = 0. \end{array} \tag{15}$$

Let us now consider the correction of a derivative discrepancy v. Since we do not wish this correction to introduce an intensity discontinuity at the central boundary, the correction function must be continuous at $x = 0$. However, by definition, it must have a "kink", *i.e.*, a discontinuity in its derivative, at $x = 0$; symmetry demands that we allocate half of the discrepancy to each side:

$$\Delta I'_{vL}(0) = \frac{v}{2}, \qquad\qquad \Delta I'_{vR}(0) = -\frac{v}{2}. \tag{16}$$

A moment's thought will then show that $\Delta I_v(x)$ must be an even function of x. Now, in contrast, $\Delta I_u(x)$ was an odd function of x, which meant that any intensity taken from one side of the boundary was added to the other side; in other words, there was no "creation or destruction" of intensity overall: it was merely shifted ariound. This is an mportant property of $\Delta I(x)$, that we must impose explicitly on $\Delta I_v(x)$. Mathematically, it is formulated as

$$\int_{-8}^{+8} dx\ \Delta I(x) = 0.$$

Since $\Delta I_v(x)$ is an even function of x, this then requires that

$$\int_{-8}^{0} dx\ \Delta I_v(x) = \int_{0}^{+8} dx\ \Delta I_v(x) = 0. \tag{17}$$

The constraints (16) and (17), together with the constraints (10) applying at the adjacent boundaries, therefore provide *four* constraints in each of the left and right blocks. Now, this is one more constraint than we had in correcting a u discrepancy, and as a result we need to progress cautiously. The most obvious way to satisfy four constraints in each region is to fit a cubic function to each. However, this would violate our third criterion listed in Sec. 2.7 above, because we have already decided to use a *parabolic* model in each region to measure discrepancies. If we were to use a cubic function, then the continuous function $\Delta I_v(x)$ would satisfy the constraints for continuous x, but the discrete measurement model would not deem that it was doing so. Although the error would be relatively small, it is still large enough that in some cases it would bring a degree of ambiguity and inconsistency into the correction phase: discrepancy values across a given boundary would change subtly when we correct the preceding boundary.

So is there any alternative to using a cubic function? Indeed there is. Consider the left block. The only pixels which are used to compute the discrepancies across the left adjacent boundary are i_1, i_2, and i_3. If we were to impose a parabolic function for these three pixels—say, for the domain $-8 \le x \le -5$, where we are extending to the mid-pixel points—then we would be free to fit a *different* model to the rest of the domain in the left block, namely,

$-5 \leq x \leq 0$. Let us refer to the parabola fitted to the domain $-8 \leq x \leq -5$ as $\mathit{\Delta} I_{vL1}(x)$, and the function fitted to the domain $-5 \leq x \leq 0$ as $\mathit{\Delta} I_{vL2}(x)$.

Now, what sort of function could we use for $\mathit{\Delta} I_{vL2}(x)$? We can determine this by counting how many free parameters we must have in its specfication. We started with four constraints. To join $\mathit{\Delta} I_{vL1}(x)$ and $\mathit{\Delta} I_{vL2}(x)$ together smoothly, at $x = -5$, we must ensure that their values and their derivatives match; this provides a further two constraints. Thus, we have six constraints in all. The parabolic function $\mathit{\Delta} I_{vL1}(x)$ has three free parameters. Thus, $\mathit{\Delta} I_{vL2}(x)$ must also have three free parameters; and hence it can also be a parabolic function.

Now, as before, the two constraints at $x = -8$ can be automatically incorporated into $\mathit{\Delta} I_{vL1}(x)$ by placing the turning point of the parabola at $(-8, 0)$:

$$\mathit{\Delta} I_{vL1}(x) = a(x+8)^2.$$

There is no special symmetry for $\mathit{\Delta} I_{vL2}(x)$, and hence we simply write it as

$$\mathit{\Delta} I_{vL2}(x) = bx^2 + cx + d.$$

If we now impose the constraints (17) and (16), together with the "smooth join" constraints $\mathit{\Delta} I_{vL1}(-5) = \mathit{\Delta} I_{vL2}(-5)$ and $\mathit{\Delta} I'_{vL1}(-5) = \mathit{\Delta} I'_{vL2}(-5)$, we obtain four equations in a, b, c, and d. The details are not very enlightening, and the generation and solution of the four equations are most simply performed using a computer algebra system, so we simply quote the results here:

$$\begin{aligned} \mathit{\Delta} I_{vL1}(x) &= -\frac{25v}{1248}(x+8)^2, \\ \mathit{\Delta} I_{vL2}(x) &= \frac{129v}{2080}x^2 + \frac{v}{2}x + \frac{10}{13}. \end{aligned}$$

If we graph this hybrid function, it can be seen that the derivative discrepancy correction is effected by "scooping out" some intensity near the left side and middle of the left block, and "piling it up" near the right boundary, to create a "ramp" as we approach $x = 0$.

If we actually compute these functions at the eight pixel positions in the left block, we obtain

$$\begin{array}{llll} \mathit{\Delta} i^v_1 = -\dfrac{25v}{4992}, & \mathit{\Delta} i^v_2 = -\dfrac{75v}{1664}, & \mathit{\Delta} i^v_3 = -\dfrac{625v}{4992}, & \mathit{\Delta} i^v_4 = -\dfrac{1871v}{8320}, \\ \mathit{\Delta} i^v_5 = -\dfrac{1839v}{8320}, & \mathit{\Delta} i^v_6 = -\dfrac{155v}{1664}, & \mathit{\Delta} i^v_7 = +\dfrac{1321v}{8320}, & \mathit{\Delta} i^v_8 = +\dfrac{4449v}{8320}. \end{array} \tag{18}$$

The denominators here are not convenient at all; they are not powers of 2, and hence would require a full integer division for each coefficient. However, we must again remember that our intensity values are only 8-bit values, and so there is no point in representing a fraction with more than eight fractional bits anyway. Thus, it is perfectly sufficient for us to round off the coefficients in the values (18) to the nearest fraction with a denominator of 256. A suitable set of correction values is then

$$\begin{array}{llll} \mathit{\Delta} i^v_1 = -\dfrac{v}{256}, & \mathit{\Delta} i^v_2 = -\dfrac{11v}{256}, & \mathit{\Delta} i^v_3 = -\dfrac{31v}{256}, & \mathit{\Delta} i^v_4 = -\dfrac{58v}{256}, \\ \mathit{\Delta} i^v_5 = -\dfrac{57v}{256}, & \mathit{\Delta} i^v_6 = -\dfrac{22v}{256}, & \mathit{\Delta} i^v_7 = +\dfrac{42v}{256}, & \mathit{\Delta} i^v_8 = +\dfrac{138v}{256}. \end{array} \tag{19}$$

It should be noted that, if one performs the conversions, one will find that some of the numerators in (19) are one or two units away from what would be expected from the values in (18), if rounded to the nearest integer. They have been adjusted slightly to ensure that our four criteria are satisfied exactly, to within eight bits of accuracy. The changes are essentially negligible, but included here for completeness. In any case, all that is important for the practical implementation of the algorithm are the eight coefficients.

Finally, the correction values for the right block are obtained directly by symmetry:

$$\begin{aligned}
&\mathit{\Delta} i_9^v = +\frac{138v}{256}, \qquad &&\mathit{\Delta} i_{10}^v = +\frac{42v}{256}, \qquad &&\mathit{\Delta} i_{11}^v = -\frac{22v}{256}, \qquad &&\mathit{\Delta} i_{12}^v = -\frac{57v}{256}, \\
&\mathit{\Delta} i_{13}^v = -\frac{58v}{256}, \qquad &&\mathit{\Delta} i_{14}^v = -\frac{31v}{256}, \qquad &&\mathit{\Delta} i_{15}^v = -\frac{11v}{256}, \qquad &&\mathit{\Delta} i_{16}^v = -\frac{v}{256}.
\end{aligned} \tag{20}$$