Channel Smoothing using Integer Arithmetic

Per-Erik Forssén
Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden

Abstract

This paper presents experiments on using integer arithmetic with the channel representation. Integer arithmetic allows reduction of memory requirements, and allows efficient implementations using machine code vector instructions, integer-only CPUs, or dedicated programmable hardware such as FPGAs possible. We demonstrate the effects of discretisation on a non-iterative robust estimation technique called channel smoothing, but the results are also valid for other applications.

1. Introduction

In signal processing it is useful to attach a confidence measure to each measurement [2]. Such algorithms usually work with a measurement–confidence pair, but there are advantages with using the *channel representation* [3, 4]. The channel representation of a measurement–confidence pair (p, c) is a vector $\boldsymbol{h} = c(\ H_1(p)\quad H_2(p)\quad \ldots\quad H_K(p)\ )^T$ of *channel values* h_k, computed using a set of localised *kernel functions* H_k. The kernels should be symmetric, non-negative, and preferably have a compact, local support. In this paper we will use the family of windowed $\cos^2()$ functions

$$H_k(p) = \begin{cases} \cos^2(\pi/3(p-k)) & \text{if } |p-k| \leq 3/2 \\ 0 & \text{otherwise.} \end{cases} \qquad (1)$$

This setup gives us at most $N = 3$ non-zero channels for any signal value. Other common choices of kernels are B-splines [5], and Gaussians [7]. The values represented in a channel vector may be recovered using a *local reconstruction* [4]. That is, we cut out an interval of channel values from the vector, and reconstruct the (p, c) pair using only these. A local reconstruction allows several (p, c) pairs to be retrieved without their interfering, provided that the measurement values are sufficiently different. For our channel representation, the reconstruction has to involve at least 3 adjacent channel values $\{h_l, h_{l+1}, h_{l+2}\}$, and is calculated as

$$\hat{p}_l(\boldsymbol{h}) = l + \frac{3}{2\pi}\arg\left[\sum_{k=l}^{l+2} h_k e^{\mathrm{i}2\pi/3(k-l)}\right] \quad \text{and} \qquad (2)$$

$$\hat{c}_l(\boldsymbol{h}) = \frac{2}{3}\sum_{k=l}^{l+2} h_k$$

To decode a channel vector, we simply go through all such adjacent groups, and sort the reconstructed $(\hat{p}, \hat{c})$ pairs, according to the confidence measures $\hat{c}$. If we are not interested in multiple solutions, we typically choose the one with the highest confidence.

1.1 Channel Smoothing

If we combine a set of measurements using an average of their channel vectors $\boldsymbol{h} = \frac{1}{L}\sum_{l=1}^{L} \boldsymbol{h}_l$, the reconstruction with the largest confidence measure can be shown to be a robust weighted average [5].[1] In other words, similar measurements are averaged, but the influence of a measurement on the result drops as a function of its distance to the cluster centre.

If we channel encode each pixel, $p(x, y)$, in a grey-scale image with $c(x, y) = 1$, we obtain a set of *channel images*. If we then perform low-pass filtering on the channel images, and reconstruct an image, we obtain the result shown in figure 1. This operation is called *channel smoothing*. A low-pass filtering directly on the image is also shown for comparison. A typical application of channel smoothing is outlier removal in depth maps [6]. The behaviour of channel smoothing is similar to many edge preserving methods, for instance *non-linear Gaussian filtering* [8]. In fact, non-linear Gaussian filtering can be shown to correspond to the first iteration of an *M-estimation*, which is a robust estimation technique [9]. Thus, iterated Gaussian filtering will yield very similar results to channel smoothing. Note that channel smoothing is a non-iterative technique. The size of the smoothing kernel used corresponds to the number of iterations of diffusion type methods [5].

[1] This result actually only holds when the reconstruction formula is linear, as is the case for B-spline kernels. For other kernel functions the result

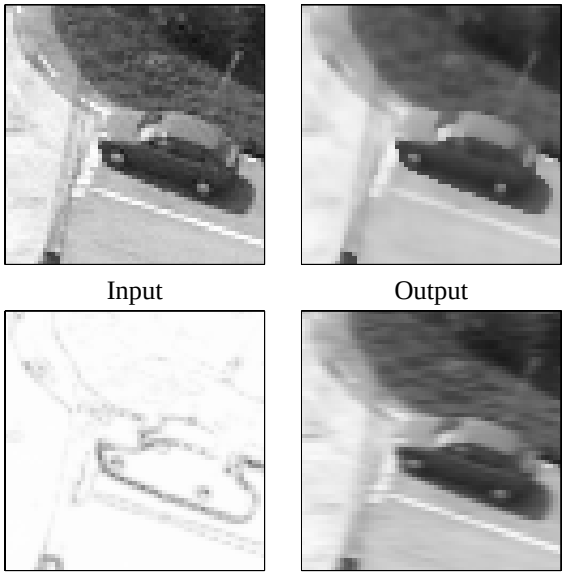


Figure 1: Illustration of channel smoothing.

For this example we have used $K = 8$ channel images, and averaged each channel image with two one-dimensional Gaussian filters of $\sigma = 0.775$ and 7 coefficients each. The image intensities $i(x, y) \in [r_l, r_h]$ have been scaled to fit the range the channels can represent ($[1 + (N-2)/2, K - (N-2)/2]$) using a linear mapping $p(x, y) = t_1 i(x, y) + t_0$ with

$$t_1 = \frac{K - N + 1}{r_h - r_l} \quad \text{and} \quad t_0 = t_1 r_l - N/2 \qquad (3)$$

2. Fixed-point arithmetic

When using real numbers in computers, we usually employ a *floating point* representation i.e.

$$h_k \approx m 2^e = \left(\sum_{k=1}^{K} m_k 2^{k-1} \right) 2^{\sum_{l=1}^{L} e_l 2^{l-1}}$$

$$\text{e.g.} \quad (1001011)_2 2^{(11011010)_2}$$

Where m is the *mantissa* and e is the *exponent* of the number, both stored as signed binary integers. However, since we know that channel values h_k are non-negative, and bounded to the range $[0, 1]$, we could instead employ a *fixed point* representation,

$$h_k \approx m = \sum_{k=1}^{K} m_k 2^{e_0 + k - 1} \quad \text{where} \quad e_0 = -K.$$

is however very similar.

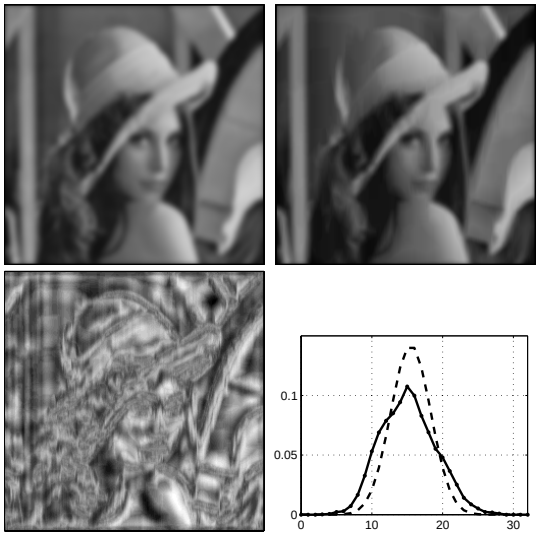


Figure 2: Correlated errors for large N.
Top left: Output of binomial filter for $N = 32$. Top right: Output using fixed-point arithmetic. Bottom left: Error image. Bottom right: Solid, measured frequencies of different errors. Dashed, expected errors if $\epsilon \in Bin(32, 0.5)$.

Here e_0 is a constant exponent, which need not be stored. The number of bits K controls the accuracy of the representation.

The ability to use fixed point arithmetic makes signal processing using channels suitable to implementation on integer only DSPs and programmable hardware such as FPGAs. It also makes implementation in SIMD instruction sets such as Intel MMX, Motorola AltiVec, and similar possible.

3. Integer low-pass filtering

Low-pass filtering in integer or fixed-point arithmetic commonly makes use of *binomial filters* [1]. A binomial filter of size 1×2 sums neighbouring pixels, and divides the result by two, omitting the least significant bit. By iterating this operation N times, we obtain a binomial filter of size $N+1$.

The error introduced by this operation is either 0 or 2^{-K-1}, and after N iterations, we will have a sum of N such errors (or to be strict, a sum of averages of such errors). Assuming independence, the total error will follow a *binomial distribution*:

$$P(\epsilon = n 2^{-K-1}) = \binom{N}{n} 2^{-N} \quad \text{or} \quad \epsilon \in \text{Bin}(N, 0.5) \qquad (4)$$

For low number of iterations, this approximation works

well, but for larger amounts of smoothing, there is a noticeable correlation between the error and high image frequencies. This is illustrated in figure 2. Here we have applied $N = 32$ iterations of the binomial filter on an 8-bit image. The bottom row shows the error image, and the frequencies of different amounts of errors.

4. Integer channel smoothing

We will now investigate the behaviour of the channel smoothing technique described in section 1.1, when using fixed point arithmetic. Channel smoothing basically consists of three steps:

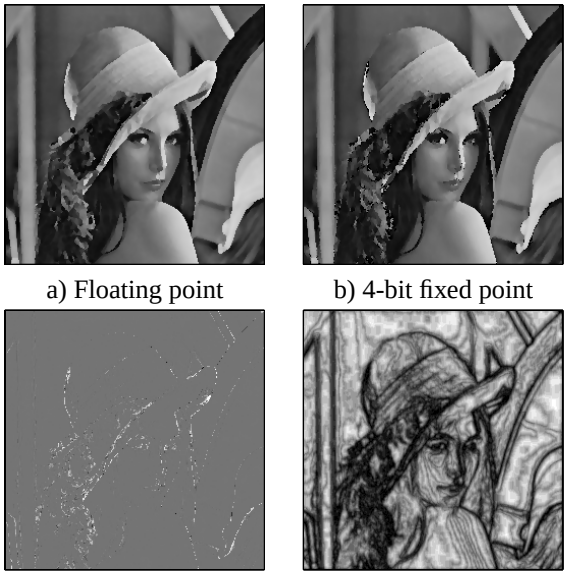
1. Expand intensity image into channel images.
2. Low-pass filter the channel images.
3. Decode channel images into an image and a confidence image.

Assuming our input image is in integer format, step 1 is easily implemented as a lookup table, regardless of which kernels we use in the channel representation. For step 2 we employ the binomial low-pass filter described in section 3. Step 3 is a bit more tricky, but if we look at equation 2, we see that the expression inside arg[] is basically two weighted sums, and since we know which range of values to expect (the same as in the input image), we could compute the value of the arg[] expression through a *reverse* lookup table: For each range value there is a real valued interval that maps onto it. We store a list of boundaries between these intervals, and the desired output for each range.

Thus, we can view steps 1 and 3 as error free, except for the quantisation to K bits. What needs to be investigated further however is the behaviour of step 2.

5. Number of bits

The result of channel smoothing using 4 bits per channel, is shown in figure 3. Here we have used 22 channels, and $N = 7$ iterations in each direction. We can see that the behaviour is the same as in the floating point case, except for some pixels near sharp edges. At some positions along the edge the reconstruction has flipped to the grey level on the other side of the edge, and at some positions we have drop-outs. The reason for the drop-outs is that an all zero channel vector is reconstructed as value 0 with certainty 0. In other words, the channel vectors have been averaged to such an extent that all bits have been shifted out. In figure 3d we have plotted the magnitude of the highest channel at each position. As can be seen, the channel values drop near edges. Thus, to be able to do more smoothing we have to use more bits.



a) Floating point b) 4-bit fixed point

c) Error image d) highest channel values

Figure 3: Behaviour of 4-bit channel smoothing.

For $K = 4$ we can handle a low-pass filter of $N = 4$ without getting all zero channel vectors. The minimum number of bits required to avoid dropouts is plotted in figure 4 as a function of the number of iterations of the binomial filter. This plot has been generated from the test image "Lena" using 22 channels. The number of bits required for a given number of iterations is dependent on how fast the channel values in an image vary with position. This in turn depends on the number of channels, and the frequency content of the image. 22 channels is however higher than typical (see e.g. figure 1 where only 8 channels were used), and the frequency content of the "Lena" image is quite varied, thus the curve in figure 4 can be considered generous with the number of bits.

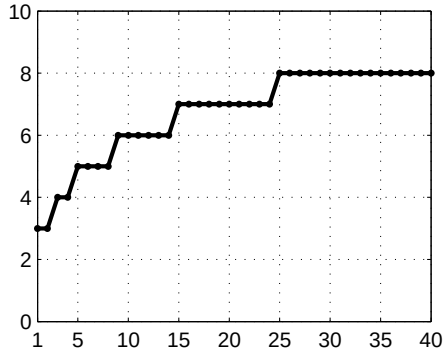
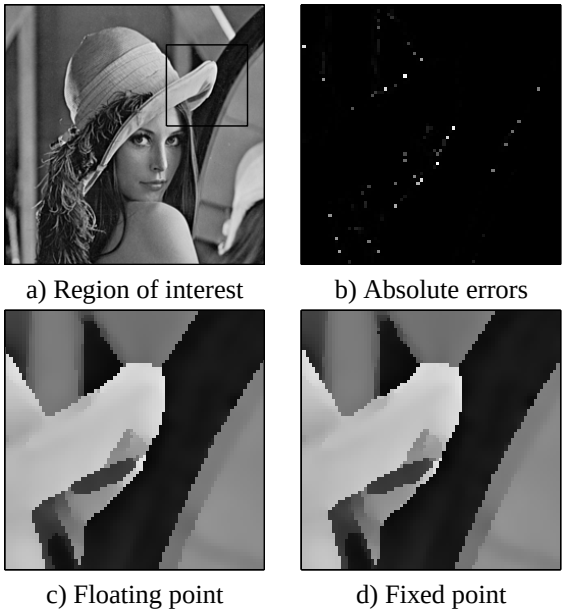


Figure 4: Number of bits required to avoid dropouts.



a) Region of interest b) Absolute errors

c) Floating point d) Fixed point

Figure 5: Comparison for 6-bit channels with $N = 14$.

As can be seen, we can handle fairly large amounts of smoothing using an 8-bit channel representation. At this number of bits however, we still have the possibility that the grey-level on the "wrong" side of the edge is reconstructed, and that the reconstruction is moved slightly due to quantisation of the channel values.

How severe these kinds of errors are is a bit difficult to judge, since we are dealing with image enhancement, but the actual images do not look too different from the floating point case. See e.g. figure 5 for an example of the worst case for 6-bit channels, when $N = 14$.

6. Concluding remarks

As has been shown, channel representation using fixed point arithmetic is possible. Provided that we increase the number of bits according to the desired number of smoothing steps, we can avoid dropouts and obtain results very similar to those obtained when using floating point representations. Especially important is the fact that 8-bits seem to be sufficient for high amounts of smoothing. This will allow one byte per channel, which is a common size for vector operations such as those in Intel MMX.

Acknowledgements

The work presented in this paper was supported by WITAS, the Wallenberg laboratory on Information Technology and Autonomous Systems, which is gratefully acknowledged.

References

[1] B. Jähne, H. Haussecker, and P. Geissler, editors. *Handbook of Computer Vision and Applications*. Academic Press, 1999. ISBN 0-12-379770-5.

[2] Granlund, G.H., Knutsson, H.: Signal Processing for Computer Vision. Kluwer Academic Publishers (1995) ISBN 0-7923-9530-1.

[3] Granlund, G.H.: An Associative Perception-Action Structure Using a Localized Space Variant Information Representation. In: Proceedings of AFPAC, Kiel, Germany (2000)

[4] Forssén, P.E.: Sparse Representations for Medium Level Vision. Lic. Thesis LiU-Tek-Lic-2001:06, Dept. EE, Linköping University, Sweden (2001) Thesis No. 869, ISBN 91-7219-951-2.

[5] Felsberg, M., Scharr, H., Forssén, P.E.: The B-spline channel representation: Channel algebra and channel based diffusion filtering. Technical Report LiTH-ISY-R-2461, Dept. EE, Linköping University, Sweden (2002)

[6] Felsberg, M. Disparity from Monogenic Phase 24. DAGM Symposium 2002, Zürich (2002) 248–256

[7] Forssén, P.E.: Observations Concerning Reconstructions with Local Support. Technical Report LiTH-ISY-R-2425, Dept. EE, Linköping University, Sweden (2002)

[8] Godtliebsen, F., Spjøtvoll, E., Marron, J.: A nonlinear gaussian filter applied to images with discontinuities. J. Nonpar. Statist. **8** (1997) 21–43

[9] Winkler, G., Liebscher, V.: Smoothers for discontinuous signals. J. Nonpar. Statistics **14** (2002) 203–222