Faster filters using bilinear texture look-ups

Mike Day, Insomniac Games
mday@insomniacgames.com

There's a fairly standard technique which uses carefully-placed bilinear taps to halve the number of texture look-ups needed to implement a given filter. This document provides a technique which can, under certain conditions, eliminate one additional look-up from each of the two passes of a separable 2D filter.

First, the standard technique. This is easily explained using the following example (which comes from Rob Wyatt). Suppose we wish to apply an 11-tap 1D truncated Gaussian with standard deviation 2. The tap weights are:

(.008812, .027144, .065114, .121649, .176998, .200565, .176998, .121649, .065114, .027144, .008812)

One reason for wishing to truncate this particular filter outside the 11-tap domain is that all normalized weights beyond this domain are less than 1/256, and can thus potentially be considered insignificant in the context of an 8-bit image.

We could implement this filter with 11 texture look-ups for each output pixel, as follows:

(-5.000000, 0.000000) * 0.008812
(-4.000000, 0.000000) * 0.027144
(-3.000000, 0.000000) * 0.065114
(-2.000000, 0.000000) * 0.121649
(-1.000000, 0.000000) * 0.176998
(0.000000, 0.000000) * 0.200565
(1.000000, 0.000000) * 0.176998
(2.000000, 0.000000) * 0.121649
(3.000000, 0.000000) * 0.065114
(4.000000, 0.000000) * 0.027144
(5.000000, 0.000000) * 0.008812

The bracketed values are the (u,v) offsets of the samples, for a horizontal filter, and the right-hand column shows the respective weights.

The standard bilinear filtering trick involves re-expressing the weighted sum of 11 terms as a weighted sum of 6 terms, each of which is a look-up at a carefully chosen location between 2 neighbouring texels:

(-4.245085, 0.000000) * 0.035956
(-2.348645, 0.000000) * 0.186763
(-0.468791, 0.000000) * 0.377564
(1.407333, 0.000000) * 0.298647
(3.294215, 0.000000) * 0.092258
(5.000000, 0.000000) * 0.008812

To derive these numbers, first, the original taps have been consecutively paired off. The first pair is

(-5.000000, 0.000000) * 0.008812
(-4.000000, 0.000000) * 0.027144

This can be reduced to a single bilinear tap whose weight, 0.035956, is the sum of the original pair of weights, and whose u-coordinate, -4.245085, is the barycentre of the original pair of locations:

(-4.245085, 0.000000) * 0.035956

The 11th tap has been kept as-is because it doesn't pair.

The first observation is that an alternative set of 6 offsets and weights is given by the following:

(-4.245085, 0.000000) * 0.035956
(-2.348645, 0.000000) * 0.186763
(-0.638336, 0.000000) * 0.277281
(0.638336, 0.000000) * 0.277281
(2.348645, 0.000000) * 0.186763
(4.245085, 0.000000) * 0.035956

This might be preferred because, as with the red version, the symmetry is still clear clear – while the result of the magenta version is a symmetrical filter, this isn't obvious looking at the numbers. To get the blue version, the pairing-off has been done from both ends of the red list simultaneously. Where the two central pairs overlap, the contribution coming from the shared central sample has been halved because it gets added in twice. (Clearly this only applies when the original number of taps is odd, as is the case here.)

Suppose we now replace the truncated Gaussian filter with a binomial distribution. For an 11-tap filter, the weights would be the values in the 11th row of Pascal's triangle

(1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1)

We'd normalize the values so they sum to 1 – to do this, just scale by 2^{-10} = 1/1024. This gives us the following weights in green, compared to the original Gaussian weights in red:

-5: 0.000977 0.008812
-4: 0.009766 0.027144
-3: 0.043945 0.065114
-2: 0.117188 0.121649
-1: 0.205078 0.176998
 0: 0.246094 0.200565
 1: 0.205078 0.176998
 2: 0.117188 0.121649
 3: 0.043945 0.065114
 4: 0.009766 0.027144
 5: 0.000977 0.008812

They're quite a bit different, with a bit more weight in the middle and a lot less weight at the very edges. Why might we wish to do this? Here are some possible reasons:

- The truncated Gaussian introduces high frequency aliasing into the filtered result, when compared to the theoretical result of using the non-truncated (infinite width) Gaussian. But the binomial curve isn't chopped at the edges – it fades properly to zero.
- The 11-tap binomial filter gives exactly the same result as 10 consecutive neighbour-averagings. This is nice and intuitive.
- With *n* taps, as *n* tends to infinity, the binomial filter approaches a broadening Gaussian, so it's a natural finite counterpart to the Gaussian.
- Since the weights are all multiples of 1/1024, they have exact floating point representations, as will the weights for all binomial distributions up to and including 28 taps.
- **It permits the optimization described below, which is the main point of this write-up.**

A common use for such filters is to apply the process twice, once in a vertical pass and once in a horizontal pass, to obtain 2D filtering. Consider the decomposition of the 11-tap binomial filter into 10 neighbour-averagings: we can apply 9 of the horizontal neighbour-averagings in the horizontal pass (which makes a 10-tap binomial filter), and while we're doing this we can also do one of the vertical neighbour-averagings by using a half-texel vertical offset, taking advantage of the bilinear filtering. Then in the vertical pass we apply this process the other way round: 9 vertical neighbour-averagings while doing one horizontal neighbour averaging using a half-texel horizontal offset. And the nice thing here is that a 10-tap filter only requires 5 bilinear look-ups, which is one less than we had before:

(-4.1, 0.5) * (10/512)
(-2.3, 0.5) * (120/512)
(-0.5, 0.5) * (252/512)
(1.3, 0.5) * (120/512)
(3.1, 0.5) * (10/512)

This is the original bilinear trick as applied to the 10^{th} row of Pascal's triangle normalized by 1/512. Note the half-pixel offset in the vertical direction, and the very neat form of the horizontal offsets. The horizontal offsets themselves are biased by -0.5 from their raw form, to counteract the constant 0.5 offset we'll be doing in the vertical pass. For the vertical pass, just swap the vertical and horizontal offsets.

Can we apply this trick to the original Gaussian weights? No, not quite. The requirement is that the alternating sum of weights be zero. This is always true if we use binomial coefficients, but is not *quite* true for our Gaussian example. It would be true for any symmetrical kernel with an even number of taps – for example if we had a 10-tap truncated Gaussian. This is of no use however, because only odd numbers of taps are susceptible to being accelerated by our half-texel-offset trick: 10 taps would reduce to 5 bilinear taps, with or without the use of the trick.

In our example, the alternating sum of the weights is 0.003697. However, recall that the Gaussian was truncated. If we include the remaining terms to infinity, the alternating sum converges rapidly to zero as the standard deviation increases. In our example, where the standard deviation is 2, the infinite alternating sum is of the order 10^{-7}. This is also of no use, though, because we will only be dealing with truncated filters.

What happens if we relax the constraint that the vertical offset be exactly 0.5 and the coefficients of the generating filter be symmetrical? In this case we can generate a larger class of filters. Consider a simple example, where we wish to use our offsetting trick to synthesize the 3-tap filter (3/16, 5/8, 3/16). Suppose it's possible to do this with the 2-tap filter (a, b) and a vertical offset by a fraction t of a texel. This establishes the following vector equation:

$$(1-t)(0,a,b)+t(a,b,0)=(\frac{3}{16},\frac{5}{8},\frac{3}{16})$$

whose solutions are readily found to be

$$a=\frac{3}{4}, \qquad b=\frac{1}{4}, \qquad t=\frac{1}{4}$$

and

$$a=\frac{1}{4}, \qquad b=\frac{3}{4}, \qquad t=\frac{3}{4}$$

Thus we can synthesize our target 3-tap filter using the 2-tap filter (3/4, 1/4) together with a quarter-texel offset in the other dimension. The 2-tap filter can in turn be reduced to a single bilinear tap in the standard way.

To take another example, suppose our target 3-tap filter is (0.3, 0.4, 0.3). In this case we find there are no real solutions to the system of equations, because the weights at the edge are too large. Intermediate between these two examples is a filter which gives rise to a system of equations with two equal roots for t. This intermediate filter is in fact the set of binomial weights (1/4, 1/2, 1/4), which is therefore the 'most spread-out' 3-tap kernel one can achieve with the offsetting trick.

In general, if the weights of the target filter fall off too slowly at the edges, the corresponding system of equations will lack any real solutions and we will be unable to decompose it using the offsetting trick. Which class does our 11-tap truncated Gaussian fall into? Unfortunately it falls within the class having no real solutions, as is easily verified using a numerical package such as Mathematica. In other words, it is not susceptible to decomposition using the offsetting trick.

What is the closest we can come to generating the 11-tap Gaussian using our technique? Recall that the alternating sum of coefficients is 0.003697, but that if it were zero, the filter could be decomposed using the half-texel offsetting technique. If we were to tweak the coefficients a little, we could force the alternating sum to zero, while at the same time maintaining the condition that the weights sum to unity. One way to accomplish this would simply be to subtract *half* of our error term (0.5×0.003697), from the leftmost weight, and add the same quantity to its neighbour.

We can do much better than this, however, using a minimax-inspired approach. The key is to distribute the error over all the coefficients as uniformly as possible using alternating signs. The perturbations must sum to zero, so as to maintain the unit sum of coefficients, while simultaneously biasing the alternating sum in favour of the negative terms so as to reduce it to zero. Let's subtract an amount Δ_0 from the even coefficients, and add an amount Δ_1 to the odd ones:

0.008812 → 0.008812 – Δ_0
0.027144 → 0.027144 + Δ_1
0.065114 → 0.065114 – Δ_0
0.121649 → 0.121649 + Δ_1
0.176998 → 0.176998 – Δ_0
0.200565 → 0.200565 + Δ_1
0.176998 → 0.176998 – Δ_0
0.121649 → 0.121649 + Δ_1
0.065114 → 0.065114 – Δ_0
0.027144 → 0.027144 + Δ_1
0.008812 → 0.008812 – Δ_0

Our unit sum and zero alternating sum conditions now establish two equations:

$$6\Delta_0 - 5\Delta_1 = 0$$

$$6\Delta_0 + 5\Delta_1 = 0.003697$$

for which the solution is

$$\Delta_0 = \frac{1}{12} \times 0.003697, \qquad \Delta_1 = \frac{1}{10} \times 0.003697$$

The final set of modified weights is

0.008504
0.027514
0.064806
0.122019
0.176690
0.200935
0.176690
0.122019
0.064806
0.027514
0.008504

This can be generated using the half-texel offset technique as the 10-tap filter

(-5.0, 0.5) * 0.017008
(-4.0, 0.5) * 0.038020
(-3.0, 0.5) * 0.091592
(-2.0, 0.5) * 0.152446
(-1.0, 0.5) * 0.200935
(0.0, 0.5) * 0.200935
(1.0, 0.5) * 0.152446
(2.0, 0.5) * 0.091592
(3.0, 0.5) * 0.038020
(4.0, 0.5) * 0.017008

And this, in turn, can be emulated using bilinear taps as the 5-tap filter

(-4.30908, 0.5) * 0.055028
(-2.37532, 0.5) * 0.244038
(-0.50000, 0.5) * 0.401870
(1.37532, 0.5) * 0.244038
(3.30908, 0.5) * 0.055028

We have succeeded in reducing the number of taps from 6 horizontally and 6 vertically, to 5 horizontally and 5 vertically, while keeping all weights within 0.00037 of their original values.