Dirac software encoding algorithm documentation 0.2

Thomas Davies

September 19, 2007

1 Introduction

2 Encoder parameters

The Dirac encoder software is controlled by passing around an EncoderParams object which contains the parameters used in quantisation, motion estimation and CBR operation. The following is a list of the EncoderParams access methods, with some comments, so as to indicate the sort of control that the encoder exercises.

bool Verbose() – returns a flag indicating whether to provide commentary during encoding.

bool LocalDecode() – returns a flag indicating whether a locally decoded version of the video is provided, to save running a decoder to view the pictures. This should be byte-for-byte identical with what the decoder produces, and this is a good initial diagnostic when implementing any bitstream changes.

bool Lossless() – returns a flag indicating whether lossless coding is being used. If so all quantisers are set to 1.

bool FullSearch() – returns a flag indicating whether full search motion estimation is being used for the pixel-accurate search. The search ranges can be set at the command line. Sub-pixel refinement and mode decision are unaffected by this.

int XRangeME(), int YRangeME() – the horizontal and vertical ranges used for full search motion estimation (and ignored for standard hierarchical search).

float Qf() – the "quality" factor (QF) being used for encoding. This is either set at the command line or determined by the rate control algorithm for CBR operation. It is actually not a quality factor per se but an indicator of the trade-off between quality and bit rate. However it has a fairly steady relationship with quality in a reasonably homogenous sequence. This is the master encoder control parameter from which all others (apart from perceptual weighting) are derived.

int NumL1() – the nominal number of L1 (i.e. P) pictures in the GOP. It's only nominal because I picture insertion may override it

int L1Sep() – the separation between I/L1 pictures: the number of L2 (i.e. B) pictures between them

float UFactor(), float VFactor() – a factor can be defined to weight U component quantisation noise when choosing quantisers. This allows chroma to be more, or less quantised on average. This feature is not used currently.

float CPD() – the key factor in determining perceptual weighting is the number of cycles per degree implied by the video standard and the assumed viewing distance. This can be fed into a weighting function and used to adjust subband quantisers according to the visibility of the spatial frequency of the subband.

bool Denoise() – returns a flag indicating whether denoising is being used.

float ILambda() – the Lagrangian lambda parameter using for choosing quantisers for scheduled Intra picture coefficients. This is derived directly from the QF value.

float L1Lambda() – likewise, but for L1 pictures.

float L2Lambda() – likewise, but for L2 pictures.

float L1MELambda(), float L2MELambda() – returns the Lagrangian lambda parameters used as part of Rate-Distortion Optimised block matching when motion estimating L1 and L2 pictures respectively.

int GOPLength() const – returns the size of the GOP, computed from the the number of L1 pictures and their separation.

const EntropyCorrector& EntropyFactors() – returns a set of scaling factors used in quantiser selection. In quantiser selection, an estimate is made of the subband entropy i.e. the number of bits required to code a subband. This is usually an overestimate, since only a simple entropy estimation process is used. The EntropyCorrector class maintains a set of correction factors for each subband, component and picture type based on previous data, so that a more accurate estimate can be made.

WltFilter IntraTransformFilter() – returns the wavelet filter being used for Intra pictures.

WltFilter InterTransformFilter() – returns the wavelet filter being used for Inter pictures.

int TargetRate() – returns the target bit rate in kilobits per second, when CBR coding is being used.

3 RDO principles

The encoder works fundamentally on Rate-Distortion Optimisation (RDO) principles. In lossy compression one is trying to minimise rate with respect to distortion, or minimise distortion with respect to rate. The are both constrained optimisation problems, which can be turned into an unconstrained problem by the method of Lagrangian multipliers. In this approach, instead of trying to minimise $Error(p)$ with respect to $Rate(p)$, or vice-versa over some parameter p, one minimises

$$Error(p) + \lambda Rate(p) \tag{1}$$

over p instead. One might think that this isn't much simpler – if one has a bit rate constraint, for example, why not code to meet that? The answer is that in video coding bit rate constraints are global, and the bit stream consists of many elements coded independently which have to hit the bit rate in total. There are a very, very large number of ways of meeting the constraint and it's not clear which is best. By choosing a point of constant rate-distortion *slope* for each element of the bitstream independently, one ensures (supposing that the measures of distortion and bit rate are genuinely representative – see below)

that the distribution of bits between elements is optimal (see reference [1] for an explanation of why this is so).

Having found a trade-off point, you can then converge on constant quality or constant bit-rate by varying λ, either by recoding the same material, or (more riskily but pragmatically) assuming a degree of homogeneity and applying changes to new data.

When choosing a quantiser, one can measure error as the difference between a quantised and the original value. This can be weighted in different ways to give a better indication of the perceptual impact of the distortion. Dirac uses a 4th power metric, and also uses a spatial-frequency weighting metric. Quantisation over subbands is fairly independent, especially in inter pictures.

Rate-Distortion optimisation in motion estimation is also necessary: at lower bit rates, one wants a lower motion vector bit rate and for higher bit rates, a higher motion vector bit rate. However, it doesn't fit into the theoretical RDO model because motion vector coding is not independent from coding the residues, since the distortion that a particular motion field introduces (the motion compensated residual) can be corrected by the residue coding. So there is an additional trade-off between bits spent on motion vectors and bits spent on correcting the residue. In general, at high bit rate there are diminishing returns from improving the motion field, and at low bit rate there are much better returns from having a good motion field. Typically, in implementations, some form of block matching is used with a modified matching criterion of the form:

$$SAD(v) + \lambda_{\mathrm{ME}} MVRate(v) \tag{2}$$

for a vector v. $MVRate(v)$ can either be an instantaneous measure of the contribution of v, or one may compute a whole set of candidates for a whole picture and attempt some sort of smoothing, somehow, and measure motion vector rate globally. Using an instantaneous measure (such as the size of $v - v_{\mathrm{pred}}$ for some predictor v_{pred}) has drawbacks, but is certainly easier.

In Dirac, the overall trade-off factor or λ is derived from a value termed "QF", meaning quality/quantisation factor. In discussion this tends to get truncated to "quality factor", which is very misleading: the one thing it is *not* is a direct measure of quality, and coding with constant QF will ensure constant quality only on homogenous material where the trade-off between distortion and rate is constant. (I sometimes wish I had renamed it "trade-off factor", although this would not have been comprehensible to anyone.)

The success of encoder control depends upon the interplay of three factors: the implicit qualities of the compression tools – for example, their typical artefacts; the dynamics of the control algorithm; and the measures used for distortion measurement in the explicit RDO calculations. In theory, control dynamics could be completely eliminated, since the temporal effects of control could be integrated into a measure of distortion. In practice, distortion in video coding can only be explicitly measured (and therefore directly controlled) picture by picture, and the control architecture can usually only be feedback-based. So the distortion metrics used in RDO do not capture all the distortions that are relevant, by any means.

The difference between excellent encoder control and merely good would seem to be not the use of RDO for low-level encoder decisions (which is ubiquitous), but how the system works dynamically to compensate for changes in the

material which make the measurements used for control obsolete or inaccurate, and how these dynamics work perceptually.

4 Encoding a sequence

Encoding a sequence is handled by the SequenceCompressor class (sequence_compress.cpp). This class maintains the current position in the sequence and codes the next picture in coded order. Source pictures are placed into two encoder buffers, a clean one which is used for motion estimation, and a locally decoded one which holds reconstructed coded pictures, to be used for motion compensation and to provide a local output if required.

Coding is done by calling a function called $CompressNextFrame()$ every picture period. The main job the class has to do is re-ordering pictures so that they are encoded in the correct order, which in the presence of B pictures is not display order. There are several ways to do this, but $CompressNextFrame()$ operates by:

1. Let P be the number of pictures encoded so far
2. Let K be the number of pictures read into the encoder buffer so far (position in the current video stream/source buffer)
3. Let $N = CodedToDisplay(P)$ be the picture number of the next picture to be encoded
4. Read a picture with picture number K. Denoise if required (Section 4.3), and place in the encoder picture buffers. Set $K = K + 1$.
5. If $K \geq N$: picture N must be in the encoder buffer, so encode and output picture N and set $P = P + 1$.
6. Clean the picture buffers of: a) reference pictures that will no longer be used and b) non-reference pictures already encoded

(NB: This process is made a little more complex in practice by supporting a locally decoded output).

This algorithm will work for any reordering with a fixed maximum depth i.e. a fixed maximum value of $N - P$. If the next picture to be encoded hasn't been read yet, then a new picture is read until it has been. Only then is the value P incremented. So the required delay between a picture being read and being encoded is built up gradually, without being set in advance. For example, if one has a traditional $IBBPBBP\ldots$ GOP, the first I picture is encoded, then the next 3 pictures are read before the P picture is encoded, and from then on the encoder operates with 3 pictures delay.

4.1 Re-ordering

The encoder command-line parameters determine the $CodedToDisplay()$ function. The encoder defines Level 1 (L1) pictures and Level 2 (L2) pictures. These are (now – it was different in the past)actually P pictures and B pictures – in the sense of forward and bi-directionally predicted pictures. Note that unlike MPEG-2 L2/B pictures may be used as reference pictures.

Two values are set: num_L1 and $L1\_sep$. $L1_sep$ is the number of L2 pictures that occur between an I or L1 picture and the next I or L1 picture. For example, $IBBBPBBBP\ldots$ has an L1 separation of 3, $L1\_sep = 0$ gives P-only coding. num_L1 is the number of L1 pictures that occur in a GOP i.e. between *scheduled* I pictures (some may be turned into I pictures by cut detection.

The total GOP length is given by $(num\_L1 + 1) * L1\_sep$.

4.2 Operation of the picture buffers

The picture buffers are instances of the FrameBuffer class.

Each picture buffer is a set of (pointers to) picture data, considered as a set of picture-sized "slots". When the buffer is cleaned of pictures that don't need to be retained, slots are vacated. These are used when new pictures are added to the buffer. If there are no slots available then a new slot is added.

4.3 Denoising

If the command-line denoising option is set, then a centre-weighted 3x3 median filter is applied. This has kernel

1 1 1
1 5 1
1 1 1

i.e. the centre value is repeated 5 times before the median is taken. Denoising occurs as a picture is read, and so it applies to the copy of the picture in both the clean (motion estimation) and the coded picture buffers.

5 Encoding a picture

The overall procedure for encoding a picture is as follows (optional steps are in square braces):

1. Set the picture encoding parameters for picture N based on position in the GOP
2. if not intra:
   (a) Motion estimate (Section 5.2)
   (b) Do cut detection
   (c) If still not intra: motion compensate
3. For each component and each subband:
   (a) Set RDO lambda to be used (Section 5.4)
   (b) Select a quantiser(Section 5.5)
4. Quantise and code component subbands

5.1 Picture encoding parameters

The picture encoding parameters consist of:

1. Picture type (Intra/Inter, Reference/Non Reference)
2. Expiry time (when it may be removed from the encoder buffers)
3. Picture lambda values

5.1.1 Picture types

Picture types are set as follows.

Intra pictures occur every GOP_length pictures, $GOP\_length = (num\_L1 + 1) * L1\_sep$.

The references for L1/P pictures are the last two L1 or I pictures (in display order), except for the first L1 picture in a GOP, which only has the previous I picture as reference. So, for a classic (12,3) GOP, picture 0 is intra; picture 3 has reference 0; picture 6 has references 3 and 0 and picture 9 has references 3 and 6.

L2/B pictures have references the previous picture and the subsequence I/L1 picture. So in a classic (12,3) GOP picture 1 has references 0 and 3, picture 2 has references 1 and 3, picture 4 has references 3 and 6, picture 5 has references 4 and 6, and so on. As a result, all pictures are references except for those L2 pictures immediately before an I or L1 picture.

Cut detection is applied to inter pictures, and can transform them into I pictures. Their reference status is unchanged.

5.1.2 Expiry time

Removal from the reference picture buffer has to be explicitly signalled to the decoder. In Dirac this is done by computing an expiry time for each picture after which it will no longer be used for reference. This is worked out for I, L1 and L2 pictures separately based on the prediction structure.

5.1.3 Picture lambda values

Picture lambda values are used for rate-distortion control of quantisation and motion estimation: see Section 3 for an overview of the approach. They are *initially* derived from the picture QF, which is either set directly on the command-line and used for all pictures or determined by means of the Rate Control algorithm (Section 6). However, a number of factors are used to modify the lambda values after motion estimation (Section 5.4).

The initial assignation of values is (from common.cpp):

$$\begin{aligned} I\_lambda &= \frac{1}{16} 10^{\frac{(10-QF)}{2.5}} \\ L1\_lambda &= 32 * I\_lambda \\ L2\_lambda &= 256 * I\_lambda \end{aligned}$$

These lambda variables are used for quantiser selection in I, L1 and L2 pictures. From these, motion estimation lambdas are derived:

$$\begin{aligned} L1\_me\_lambda &= 2.0 * \sqrt{L1\_lambda} \\ L2\_me\_lambda &= L1\_me\_lambda \end{aligned}$$

Notes

- These ratios and formulae have been derived by experiment but they are not optimal for all picture and in particular all sequences, video resolutions, perceptual weightings, or block sizes. The ideal trade-offs may change with all of these. However, the best values don't seem to change very much. In particular, the value of 2.0 used for scaling motion estimation lambdas is a "soft" value: there is little difference in choosing another value in the same ballpark.

  There are two guiding principles:

  1. I pictures should be higher quality than L1 pictures and L1 pictures should be higher quality than L2 pictures
  2. Motion data and good motion rendition is more significant at lower bit rates (low QFs) than at higher ones (high QFs)

  The first principle arises because I pictures are used as references for L1 and L2 pictures and L1 pictures are used as references for L2 pictures. If quality were to go up from I to L1 or L1 to L2, then the encoder would need to correct the quantisation error introduced in the reference picture and "pushed forward" by motion compensation. This error is noise-like and expensive to code (the phase-variation of the wavelet filters also means that an error in a single coefficient in the reference picture can spread to several coefficients when that picture is shifted through motion compensation). As a result the L1 and L2 lambdas are multiplied up.

  The aim of the second principle is to stop quality falling off a cliff and QF goes down/lambdas go up. Two measures are taken. Using the *square root* of *L1_lambda* for motion estimation means that the motion field is not over-smoothed at low bit rates. Setting the L2 ME lambda equal to that for L1 means that although the quality is lower, there are no more poorly corrected areas (in fact there are fewer, as they are B pictures): L2 pictures have less opportunity to correct motion estimation errors in residual coding.

- It's worth pointing out, in the light of the previous comment, that in the current prediction structure L2 pictures are also predicted from other L2 pictures as well as L1 pictures. One might expect that it would be optimal to gradually reduce quality through a run of B pictures to allow for this. However, the current control structure doesn't allow for this, and perhaps a "sawtooth" quality profile would be subjectively objectionable in any case.

- Brunel have argued that for HD material, there is too much disparity between I, L1, and L2 pictures, and that smaller factors than 32 and 256 should be used. I have found this is so sometimes, but in general it seems about right.

5.2 Motion estimation

Motion estimation in Dirac uses the general form described in Equation 2, with an instantaneous measure of motion vector bit rate. However, a number of methods are used to prevent over-smoothing of the motion field and reduce visible artefacts:

- λ_{ME} is set to zero for pixel-accurate matching
- λ_{ME} is proportional to the *square root* of the quantisation lambda (Section 5.1.3
- The measure of motion data bit rate takes motion transitions into account
- After pixel-accurate motions, the motion field is analysed so that difficult areas can be treated differently
- Mode decision is biased away from intra blocks

The overall algorithm is:

1. For each reference find pixel-accurate motion fields $MV1$, $MV2$, without RDO
2. Analyse the pixel accurate motion field and define a value $\lambda_{ME}(B)$ for each block B
3. Refine $MV1$ and $MV2$ to sub-pixel accuracy using RDO
4. Perform RDO mode decision

In the matching stages, motion vectors are added by adding neighbourhoods of guide vectors to a common list, ignoring duplicates. Two neighbourhoods a defined, a square and a diamond shape area:

$$\begin{aligned} S(\mathbf{v}, d) &= \{\mathbf{w} : |\mathbf{w}_x - \mathbf{v}_x| \leq d, |\mathbf{w}_y - \mathbf{v}_y| \leq d\} \\ D(\mathbf{v}, d) &= \{\mathbf{w} : |\mathbf{w}_x - \mathbf{v}_x| + |\mathbf{w}_y - \mathbf{v}_y| \leq d\} \end{aligned}$$

Some other notation. The median of three vectors $mathhbfa, mathhbfb, mathhbfc$ is given by

$$Median(mathhbfa, mathhbfb, mathhbfc) = (Median(mathhbfa_x, mathhbfb_x, mathhbfc_x), Median(mathhbfa$$

The norm of a vector $\|\mathbf{v}\|$ is equal to $|\mathbf{v}_x| + |\mathbf{v}_y|$.

5.2.1 Pixel-accurate estimation

This is the simplest stage. Hiercharical motion estimation is used in order to capture large motions. At each level, matching is over blocks of size $xblen \times yblen$, so the number of matches is reduced by a factor 4 at each stage, and each block occupies 4 times the area of the block at the next highest level. The depth *depth* of the hierarchy is such that a whole superblock may be contained within the current picture, so it varies with the size of the images.

The algorithm is:

1. For level l in $depth, depth-1, \ldots, 0$:
   (a) Set $d = \min(l+1, 5)$
   (b) For each block (i, j):
      i. Set candidate list $C = S(\mathbf{0}, d)$
      ii. If $l < depth$: let the lower-level guide be $\mathbf{w} = \mathbf{v}_{l+1}(i/2, j/2)$; add $S(\mathbf{w}, d)$ to C
      iii. Let the spatial predictor $\mathbf{s}$ be the median of the already-determined vectors $\mathbf{v}_l(i-1, j)$, $\mathbf{v}_l(i, j-1)$, and $\mathbf{v}_l(i-1, j-1)$. Add $S(\mathbf{s}, d)$ to C
      iv. Set $\mathbf{v}_l(i, j) = \arg\min_{\mathbf{x} \in C} SAD(x)$

5.2.2 Motion field analysis

The pixel accurate motion field determined in the previous stage is now analysed to see where jumps in the field occur. Blocks likely to contain jumps will have λ_{ME} set to a smaller value:

1. Set λ_{ME} equal to $L1\_ME\_lambda$ or $L2\_ME\_lambda$ as appropriate (these values are currently equal in the software)
2. For each motion field $MV1$ and $MV2$ and each block compute the difference between the motion vector and its spatial predictor:

$$\begin{aligned} d1(i,j) &= \|MV1(i,j) - Median(MV1(i-1,j), MV1(i,j-1), MV1(i-1,j-1))\| \\ d2(i,j) &= \|MV2(i,j) - Median(MV2(i-1,j), MV2(i,j-1), MV2(i-1,j-1))\| \end{aligned}$$

3. Define $m1$ and $\sigma 1$ as the mean and standard deviation of $d1$ and $m2$ and $\sigma 2$ the mean and standard deviation of $d2$.
4. If $d1(i,j) > m1 + 3 * \sigma 1$ mark (i,j) as a reference 1 transition. If $d2(i,j) > m2 + 3 * \sigma 2$ mark (i,j) as a reference 2 transition.
5. If (i,j) is a reference 1 and a reference 2 transition, $\lambda_{ME}(i,j) = 0$; else if it is either a reference 1 or a reference 2 transition, $\lambda_{ME}(i,j) = \lambda_M E/4$

5.2.3 Sub-pixel refinement

The pixel-accurate motion vector fields for each reference are refined individually.

Sub-pixel refinement uses the pixel accurate motion vector as the initial guide. 1/2, 1/4 and 1/8 pixel vectors are searched in turn, with the result of 1/2-pel matching being used as the guide for 1/4-pel matching and that for 1/4 pel matching being used as the guide for 1/8-pel matching. Hence there are a maximum of 3 stages.

Within each stage there are two sub-stages. The first searches horizontal and vertical offsets, marked * below:

```
    *
*   ×   *
    *
```

When the best one is found, the two nearest unsearched positions are searched, e.g. (new positions marked +):

```
+   *
*   ×   *
+   *
```

The matching function used is:

$$M(\mathbf{v}, i, j) = SAD(\mathbf{v}) + \lambda_{ME}(i, j) MCost(\mathbf{v}, i, j) \tag{3}$$

where

$$MCost(\mathbf{v}, i, j) = \min(\|\mathbf{v}\|, \|\mathbf{v} - Median(\mathbf{v}(i-1, j), \mathbf{v}(i, j-1), \mathbf{v}(i-1, j-1))\|)$$

So the cost function does not simply use the size of the difference from the spatial predictor but also the size of the vector itself. This is intended to provide better results at foreground/background transitions, where one vector will be close to 0 and those of adjacent blocks will be large.

5.2.4 Mode decision

Mode decision is the final stage of motion estimation. At this point there are block-level motion vectors for each reference, together with estimated costs arising from the matching crition 3. To choose a prediction mode we need costs for intra prediction and for bi-directional prediction also. To choose a splitting mode we need costs for sub-superblock and superblock splittings for *all four* prediction modes.

The approach is to loop over splitting level, starting at level 2 (block level splitting) and working upwards. Within each splitting level the best prediction modes are selected.

Level 2 mode decision We have Ref1 and Ref2 costs from sub-pixel matching. So:

1. For each block:
   (a) If Ref2 cost ¡ Ref1 cost: set mode equal to Ref1, else set it to Ref2.
   (b) Use Ref1 and Ref2 motion vectors to determine a bidirectional SAD cost from the two references. Add the Ref1 and Ref2 motion costs from equation 3 to get a Ref1and2 motion cost.
   (c) If the Ref1and2 cost ¡ best cost so far, set the mode equal to Ref1 and 2
   (d) Compute an Intra cost as the SAD value of the DC difference
   (e) If the Intra cost is ¡ 90% of best cost so far, set mode equal to Intra
2. For each superblock, let the level 2 cost be the sum of the best costs for each block

Level 1 mode decision Level 1 mode decision requires costs for all four modes and each constituent sub-superblock. Ref1 and Ref2 costs are determined by block-matching using the cost function 3. The ME lambda used for the match is the maximum of the ME lambda for the constituent blocks. For each sub-superblock, 4 candidates are tested, namely the motion vectors determined for

the constituent blocks. Thereafter, the process is the same as for Level 2 mode decision.

The level 1 costs must be scaled to take account of the fact that the total overlap between sub-superblocks is less than that between blocks. Once this is done, level 1 costs can be compared to level 2 costs for each superblock, and if lower the splitting mode set to 1.

Level 0 mode decision Level 0 mode decision is only applied when the splitting mode has been set to 1 in the previous stage. It operates very similarly to Level 1 mode decision.

5.3 Cut detection

L1 and L2 pictures may be replaced with intra pictures if motion estimation is deemed to have essentially failed. This is measured by the proportion of intra blocks, ignoring superblock splitting (i.e. a superblock counts as 16 individual blocks regardless of the splitting mode). If this proportion is greater than a threshold of 1/3 then an intra picture is inserted, but in any case the picture lambda is revised:

1. Let R be the proportion of intra blocks
2. If $R > 0.3333$:
   (a) set the picture type to be intra (reference/non reference status unchanged)
   (b) set $I\_lambda = I\_lambda * 8$

The intra block ration is also used to modify the component lambda even if an I frame is not inserted (Section 5.4.1).

Notes

- If there is an inserted I picture, it has lower quality than a scheduled intra picture. The factor 8 experimentally seems to give about the same quality as an L1 picture. One might expect the correct factor to be 32, as in the original assignment to L1 pictures (Section 5.1.3) but I and L1 pictures use different perceptual weightings (Section 5.4.3), which affects how quantisation noise is weighted in the calculations.

5.4 Setting subband RDO lambdas

By the time quantiser selection occurs, the lambda actually used depends upon:

- The overall QF
- The picture type (scheduled I, inserted I, L1 or L2)
- The proportion of intra blocks, if L1 or L2
- The video component and chroma subsampling ratios
- Filter gains and the perceptual weighting matrix

The lambda value used for determining the quantiser for each subband is therefore set as follows:

1. Set an overall component lambda:

$$\lambda = \begin{cases} I\_lambda & \text{if Intra} \\ L1\_lambda & \text{if L1} \\ L2\_lambda & \text{if L2} \end{cases}$$

2. If inserted intra:

$$\lambda = \lambda * 8$$

3. If L1 or L2, adjust according to the proportion of intra blocks (Section 5.4.1)

4. If component is U

$$\lambda* = 1.0$$

   If component is V

$$\lambda* = 1.0$$

   (There is provision in the software for non-trivial weighting if required.)

5. For each subband n, define $lambda(n)$ by

$$\lambda(n)* = gain(n) * weight(n)$$

   where $gain(n)$ and $weight(n)$ are as defined in Sections 5.4.2 and 5.4.3.

Notes

- If there is an inserted I picture, it has lower quality than a scheduled intra picture. The factor 8 experimentally seems to give about the same quality as an L1 picture. One might expect the correct factor to be 32, as in the original assignment to L1 pictures (Section 5.1.3) but I and L1 pictures use different perceptual weightings (Section 5.4.3), which affects how quantisation noise is weighted in the calculations.

5.4.1 Adjusting for the proportion of intra blocks

L1 and L2 pictures may have their lambdas modified depending on how effective motion estimation has proved to be. This is measured by the proportion of intra blocks, ignoring superblock splitting (i.e. a superblock counts as 16 individual blocks regardless of the splitting mode). If this proportion is greater than a threshold of 1/3 then an intra picture is inserted, but in any case the picture lambda is revised:

1. If not intra:
   (a) Let R be the proportion of intra blocks
   (b) $lambda = 10^(r * log(I\_lambda) + (1 - r) * log(lambda))$

Notes

- If a picture has a larger proportion of intra blocks, it uses a lower lambda – closer to the intra picture lambda – than it would otherwise. The formula for adjusting to the ratio is a weighted geometric mean, since quality has a roughly log-like relationship to the lambda values. It's equivalent to taking a weighted arithmetic mean of the associated QF values.

5.4.2 Filter gains

Each subband is associated with a filter gain $g(n)$ representing the noise power gain that would arise from quantisation in that subband. In the event of true, properly scaled real-valued orthogonal wavelets this value would be 1, but such filters would have a DC gain of $\sqrt{2}$, which does not translate well into the integer implementations used in Dirac.

In the software the filter gains are used to weight the measured quantisation noise in quantiser selection, but this is equivalent to scaling the quantisation lambda $\lambda(n)$ for each subband.

$gain(n)$ is determined as follows: define values α and β as the RMS power gain of the low-pass and high-pass filters respectively. These factors for the different filters can be found in wavelet_utils.h,cpp. RMS quantisation noise in each of the four subbands LL, HL, LH and HH is therefore weighted by the factors shown in Figure 1.

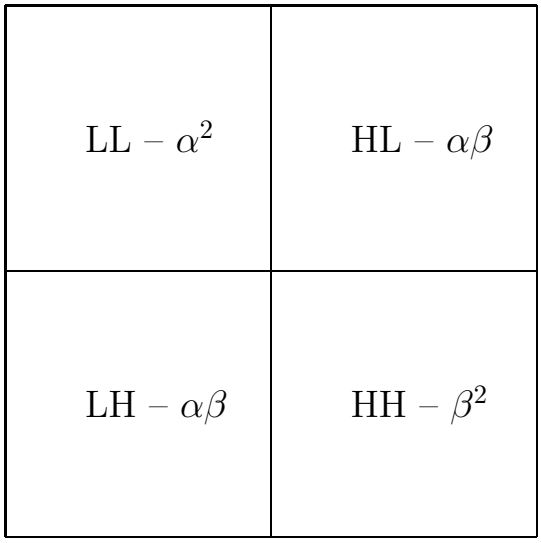

Figure 1: Subband weights for a 1-level decomposition

In a two-level decomposition, the LL band is further decomposed, yielding the situation shown in Figure 2.

This process continues with each decomposition, leading to an RMS weight value $w(n)$ for each subband. The appropriate gain factor $gain(n) = w(n)^2$.

Notes

- The gain factors

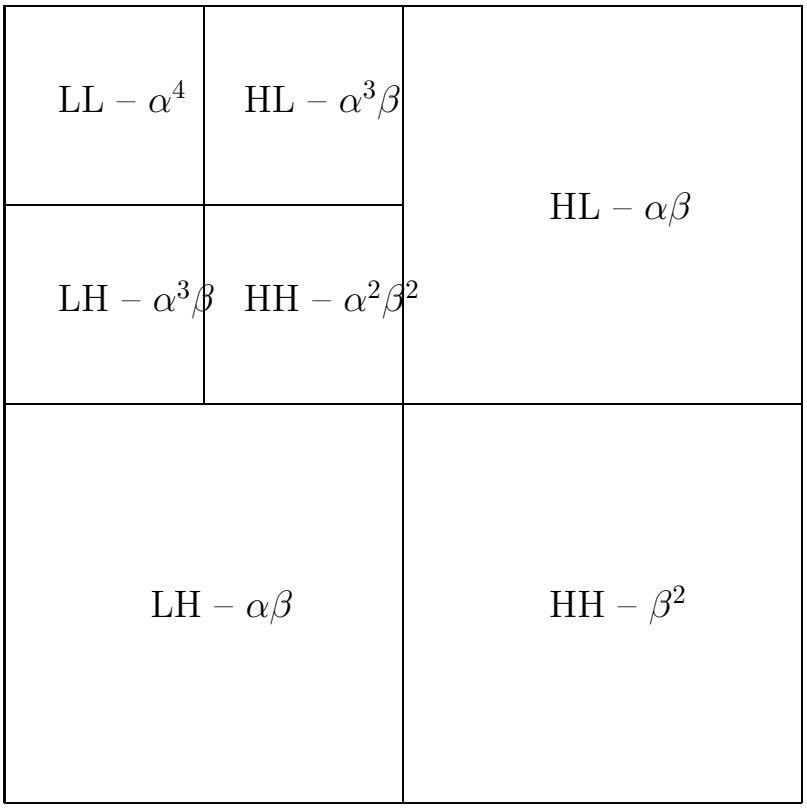

Figure 2: Subband weights for a 2-level decomposition

5.4.3 Perceptual weighting

Perceptual weights depend upon the picture dimensions, aspect ratiom the assumed viewing distance and (if chroma) the chroma subsampling ratios: these, together, give rise to a value of "cycles per degree" (CPD). Given a value of CPD, one may determine *normalised* spatial frequency ranges for each subband and feed them into a weighting function, normally a model of the human Contrast Sensitivity Function (CSF).

The calculation is as follows:

1. Frequency normalisation. Set

$$\begin{aligned} f_x &= CPD * (xs + xl/2)/xl \\ f_y &= CPD * (ys + yl/2)/yl \end{aligned}$$

where xl, yl are the horizontal and vertical bandwidths and xs and ys are the horizontal and vertical lower cutpoints of the band.

2. If the component is chroma

$$\begin{aligned} f_x &= f_x * 1.1 \\ f_y &= f_y * 1.1 \end{aligned}$$

(additional softening for chroma data)

3. If the component is chroma *and* 422 or 420 sampled

$$f_x = f_x/2$$

If the component is chroma *and* 420 sampled,

$$f_y = f_y/2$$

4. If the picture is inter (L1 or L2)

$$\begin{aligned} f_x &= f_x/8 \\ f_y &= f_y/8 \end{aligned}$$

5. Define

$$wt(n) = 0.255 * (1.0 + 0.2561 * (f_x^2 + f_y^2))^{0.75}$$

6. Normalise the weights to give constant white noise power, whatever the weight:

$$weight(n) = \frac{wt(n)^2}{\sum_k wt(k)^2 * subband\_frac(k)}$$

where $subband\_frac(n)$ is the fraction of the spatial frequency range occupied by the subband (i.e. a quarter for top-level subbands, 1/16 for lower levels and so on).

5.5 Quantiser selection

The quantisation selection algorithm uses the $\lambda(n)$ value determined for the subband (Section 5.4). It is based on RDO optimisation using a 4th-power measure of quantisation error and direct symbol counts of quantised values to compute entropy. In the current software (quant_chooser.h,cpp) it is possible to compute different quantisers for each codeblock, but this doesn't work well and is not discussed further.

Although a single quantiser is determined for the whole subband, in the software statistics are gathered by codeblock. This allows multiple subband quantisers to be supported and also allows skipped codeblocks to be identified without another pass over the data, which (perhaps) improves efficiency. This feature of the software is ignored in what follows.

The overall quantiser selection algorithm does a hierarchical search over bit-accurate, half-bit and quarter-bit quantisers:

1. Set $qstep = 4$, $qmin = 0$, $qmax = \infty$

2. For $qindex = qmin$ to $qmax$ step $qstep$:
   - Compute $Error(qindex)$
   - Compute $Entropy(qindex)$
3. Set $qbest = \min_{qindex=0,4,8...}(Error(qindex) + \lambda(n)Entropy(qindex))$
4. Set $qmin = qbest - 2$, $qmax = qbest + 2$, $qstep = 2$ and repeat step 2.
5. Set $qbest = \min_{qindex=qmin,...,qmax}(Error(qindex)+\lambda(n)Entropy(qindex))$
6. Set $qmin = qbest - 1$, $qmax = qbest + 1$, $qstep = 1$ and repeat step 2.
7. Set $qbest = \min_{qindex=qmin,...,qmax}(Error(qindex)+\lambda(n)Entropy(qindex))$. Stop.

$Error(qindex)$ and $Entropy(qindex)$ are defined as follows. If $x \mapsto Q(qindex, x)$ denotes quantisation by the quantiser with index $qindex$, and $y \mapsto IQ(qindex, y)$ denotes inverse quantisation, then quantisation and reconstruction of a value x is given by:

$$R(qindex, x) = IQ(qindex, Q(qindex, x))$$

$Error(qindex)$ is defined by:

$$Error(qindex) = \left(\frac{1}{N} \sum_n (x - R(qindex, x))^4 \right)^{\frac{1}{2}}$$

where N is the number of values sampled from the subband (in the Dirac software vertical and horizontal subsampling ratios can be defined to reduce computation).

Entropy is calculated by assuming unary binarisation (i.e. magnitude value y becomes y zeroes followed by a 1). So to compute $Entropy(qindex)$ do:

1. Set
   - $count0 = 0$
   - $count1 = 0$
   - $countPOS = 0$
   - $countNEG = 0$
2. For each x in (sampled) subband:
   - Set $y = Q(qindex, x)$, $count1 = count1 + 1$, $count0 = count0 + |y|$
   - If $y > 0$, set $countPOS = countPOS + 1$
   - If $y < 0$, set $countNEG = countNEG + 1$
3. Set $N = count1$ (the number of coefficients counted)
4. Set magnitude probabilities: $P_0 = count0/(count0 + count1)$, $P_1 = 1 - P_0$
5. Set sign probabilities: $P_{POS} = countPOS/(countPOS + countNEG)$, $P_{NEG} = 1 - P_{POS}$

6. Compute magnitude entropy per bit:

$$e_{mag} = -P_0 * \log_2(P_0) - P_1 * \log_2(P_1)$$

and sign entropy per bit

$$e_{sign} = -P_{POS} * \log_2(P_{POS}) - countNEG * \log_2(P_{NEG})$$

7. Scale to give entropies per *coefficient*:

$$Entropy(qindex) = e_{mag} * \frac{count0 + count1}{N} + e_{mag} * \frac{countPOS + countNEG}{N}$$

8. Correct the entropy estimate by a correction factor based on subband, picture type and component:

$$Entropy(qindex) = Entropy(qindex) * cfactor(pic\_type, n, comp\_sort)$$

(Say sthg about correction factors) (Say sthg about DC bands)
Notes

- Obviously the calculation of $Entropy(qindex)$ and $Error(qindex)$ can be done simultaneously, as it is in Dirac
- The bit-accurate quantisation measurements can all be done in a single pass over the data, since in this case quantisation is just a repeated division by 2.
- There are many shortcuts which can be taken: the statistics of the subband may be modelled by a parameterised distribution for which the entropy and error values can be looked up from a table; an initial bit-accurate search can be done, and then the best sub-bit quantiser found by curve-fitting; a quantisation matrix can be devised which gives only a single degree of freedom, and so on.

6 Rate control and constant bit rate operation