A Fast Renormalisation for Arithmetic Coding

Michael Schindler, compression consultant
Billrothstraße 39/21, A–1190 Vienna, Austria. (michael@eiunix.tuwien.ac.at)

All integer based arithmetic coding consists of two steps: proportional range restriction and range expansion (renormalisation). Here a method is presented that significantly reduces the complexity of renormalisation, allowing a speedup of arithmetic coding by a factor of up to 2.

The main idea is to treat the output not as a binary number, but as a base 256 (or other) number. This requires less renormalisation and no bitwise operations. The bitsoutstanding carry handling allows only bitwise renormalisation, so the following was developed.

At any point during arithmetic coding the output consists of four parts.

1. There is the part already written, which can't be changed any more.
2. There is one digit (bit, byte, whatever) that may be changed by at most one carry when adding to the lower end of the interval. There will never be two carries since the range when fixing that digit was ≤ 1unit. Two carries would require a range > 1 unit.
3. There is a (possibly empty) block of digits that pass on a carry (1 in binary, 9 in decimal, 255 for bytes, ...) that are represented by a counter counting their number.
4. The last part is represented by the low variable (out_L) of the encoder.

While doing encoding renormalisation the following can happen:

- No renormalisation is needed since the range is in the desired interval.
- The low end plus the range (this is the upper end of the interval) will not produce any carry. In this case the second and third part can be output, they will never change. The digit produced will become part two, and part three will be empty.
- The low end has already produced a carry. Here the (changed) second and third part can be output; there will not be another carry. Set the second and third part as before.
- The digit produced will pass on a possible future carry, so it is added to the third block.

The difference from standard method is that part two, which may be changed by a carry, has to be stored explicitly. With binary output this part is always 0 since the ones are always added to the carry passing block. Implementing that is straightforward.

Mbyte/min on Pentium 166, linux, quasistatic model	Encode	Decode
Arithmetic coding revisited (Moffat et al, DCC95)	36.9	28.4
Proposed renormalisation	67.5	40.7

More information as well as code can be found on http://eiunix.tuwien.ac.at/~michael/arib/