IMA ADPCM

From MultimediaWiki

The Interactive Multimedia Association (IMA) developed an ADPCM algorithm designed to be used in entertainment multimedia applications. It is particularly fast to encode and decode and does not strictly require any multiplications or floating point operations.

While the encoding and decoding algorithms remain more or less constant across different IMA implementations, the specific on-disk data formats vary. This page describes the common IMA decoding algorithm. See the Category:IMA ADPCM Audio Codecs page for various formats used for storing the data on disk.

Decoding IMA

To decode IMA ADPCM, initialize 3 variables:

- predictor: This is either initialized from the data chunk preamble specified in the format or is initialized to 0 at the start of the decoding process.
- step index: Similar to the initial predictor, this variable is initialized from the data chunk preamble or set to 0 at the start of the decoding process.
- step: This variable is initialized to ima_step_table[step_index].

The encoded IMA bitstream is comprised of a series of 4-bit nibbles. This means that each byte represents 2 IMA nibbles. The specific data format will dictate whether the stream is decoded top nibble first or bottom nibble first, and whether there is stereo interleaving within the IMA nibbles. For this discussion, imagine the IMA bitstream as a series of nibbles representing a single audio channel:

```
n0 n1 n2 n3 n4 n5 ...
```

Where each nibble represents both a table index and a sign/magnitude number during the decoding process. Transform each nibble in the stream into a signed, 16-bit PCM sample using the following process:

```
step_index = step_index + ima_index_table[(unsigned)nibble]

diff = ((signed)nibble + 0.5) * step / 4

predictor = predictor + diff

step = ima_step_table[step_index]
```

Regarding the step index and predictor calculations: Be sure to saturate the computed step index between 0 and 88 (table limits) and the predictor between -32768 and 32767 (signed 16-bit number range). It is possible for these values to outrange which could cause undesirable program behavior if unchecked.

Optimization

A note about the following calculation:

```
diff = ((sign/mag.)nibble + 0.5) * step / 4
```

At first glance, it appears that this calculation requires floating point operations and an arbitrary (not power-of-2) multiplication. However, some numerical manipulations reveal some useful simplifications:

```
diff = ((step * nibble) + (step / 2)) / 4

diff =  (step * nibble / 4) + (step / 8)
```

The step / 8 calculation can be expressed as a bit shift right by 3 (step SHR 3). The first part of the equation can also be simplified. Since a nibble only carries 4 bits, and those 4 bits are a sign/magnitude number, there are only 3 bits of magnitude information. If all 3 magnitude bits are set to 1:

```
nibble = 4 + 2 + 1

step * nibble / 4 = (4 * step / 4) + (2 * step / 4) + (1 * step / 4) = step + (step / 2) + (step / 4)
```

Thus, if bit 2 of the nibble is set, add step to diff. If bit 1 is set, add (step / 2 = step SHR 1) to diff. If bit 0 is set, add (step / 4 = step SHR 2) to diff. Finally, if the sign bit is set, subtract the final diff value from the predictor value; otherwise, add the final diff value to the predictor value. The usual algorithm is as follows:

```
sign = nibble & 8
delta = nibble & 7
diff = step >> 3
if (delta & 4) diff += step
if (delta & 2) diff += (step >> 1)
if (delta & 1) diff += (step >> 2)
if (sign) predictor -= diff
else predictor += diff
```

This method was particularly useful back when IMA was implemented on commodity CPUs which were relatively slow at multiplication. One multiplication per audio sample had a notable impact on program performance, as opposed to the series of branches, additions and logical bit operations. If multiplication performance is not an issue, it is possible to carry out the diff calculation with only one non-power-of-2 multiplication and no floating point numbers:

```
diff = ((((signed)nibble+0.5) * step) / 4) * (2 / 2)

diff = (nibble + 0.5) * 2 * step / 8

diff = (2 * nibble + 1) * step / 8
```

[NOTE: something seems wrong here, e.g. for nibble = 3 and step = 3 we would get with the upper formula diff = 0 + 1 + 0 = 1 and with the lower one diff = (2 * 3 + 1) * 3 / 8 = 21 / 8 = 2 Since with ADPCM errors can propagate forever that seems like a really bad thing...]

Decoding Tables

```
int ima_index_table[16] = {
  -1, -1, -1, -1, 2, 4, 6, 8,
  -1, -1, -1, -1, 2, 4, 6, 8
};
```

Note that many programs use slight deviations from the following table, but such deviations are negligible:

```
int ima_step_table[89] = {
  7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
  19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
  50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
  130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
  337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
  876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878, 2066,
  2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871, 5358,
  5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,
  15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794, 32767
};
```


Categories: Audio Codecs | ADPCM Audio Codecs | IMA ADPCM Audio Codecs

---