转载 侵蚀昨天 于 2018-07-03 16:51:56 发布 18544 收藏 58
文章标签： G711 audio

本文目的：

1、熟悉G711a/u两种格式的基本原理

2、熟悉两种压缩算法的实现步骤及提供源码实现

它是国际电信联盟ITU-T定制出来的一套语音压缩标准，它代表了对数PCM（logarithmic pulse-code modulation）抽样标准，主要用于电话。它主要用脉冲编码调制对音频采样，采样率为8k每秒。它利用一个 64Kbps 未压缩通道传输语音讯号。 起 压缩率为1：2， 即把16位 数据压缩成8位。G.711是主流的波形声音编解码器。

G.711 标准下主要有两种压缩算法。一种是 **u-law algorithm** （又称often u-law, ulaw, mu-law），主要运用于北美和日本；另一种是 **A-law algorithm**，主要运用于欧洲和世界其他地区。其中，后者是特别设计用来方便计算机处理的

G711的内容是将14bit(uLaw)或者13bit(aLaw) 采样 的PCM数据编码成8bit的数据流，播放的时候在将此8bit的数据还原成14bit或者13bit进行播放，不同于MPEG这种对于整体或者一段数据进行考虑再进行编解码的做法，G711是波形编解码算法，就是一个sample对应一个编码，所以压缩比固定为：

8/14 = 57%（**uLaw**）

8/13 = 62%（**aLaw**）

G.711就是语音模拟信号的一种非线性量化， bitrate 是64kbps. 详细的资料可以在ITU 上下到相关的spec，下面主要列出一些性能参数：

G.711（PCM方式）

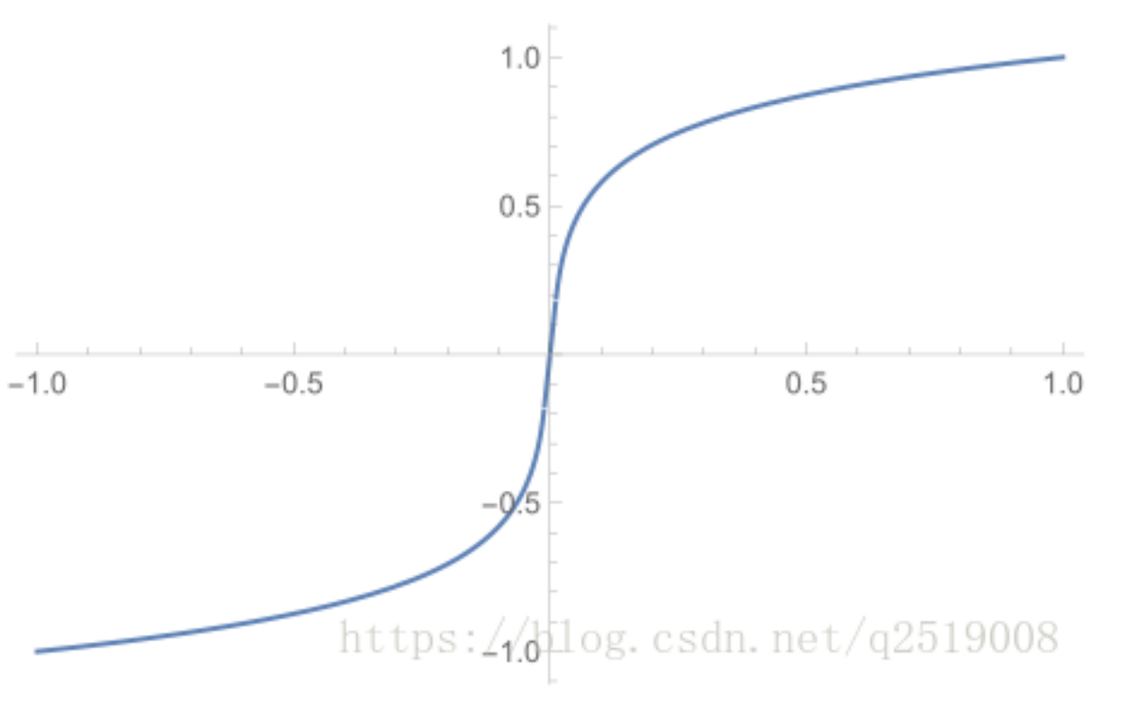
- 采样率：8kHz
- 信息量：64kbps / channel
- 理论延迟：0.125msec
- 品质：MOS值4.10

算法原理：

A-law的公式如下，一般采用A=87.6

$$F(x) = \operatorname{sgn}(x)\begin{cases} \dfrac{A|x|}{1+\ln(A)}, & |x| < \dfrac{1}{A} \\ \dfrac{1+\ln(A|x|)}{1+\ln(A)}, & \dfrac{1}{A} \leq |x| \leq 1, \end{cases}$$

画出图来则是如下图，用x表示输入的采样值，F(x)表示通过A-law变换后的采样值，y是对F(x)进行量化后的采样值。



由此可见在输入的x为高值的时候，F(x)的变化是缓慢的，有较大范围的x对应的F(x)最终被量化为同一个y，精度较低。相反在低声强区域，也就是x为低值的时候，F(x)的变化很剧烈，有较少的不同x对应的F(x)被量化为同一个y。意思就是说在声音比较小的区域，精度较高，便于区分，而声音比较大的区域，精度不是那么高。

对应解码公式(即上面函数的反函数)：

$$F^{-1}(y) = \operatorname{sgn}(y)\begin{cases} \dfrac{|y|(1+\ln(A))}{A}, & |y| < \dfrac{1}{1+\ln(A)} \\ \dfrac{\exp(|y|(1+\ln(A))-1)}{A}, & \dfrac{1}{1+\ln(A)} \leq |y| < 1. \end{cases}$$

G711A（A-LAW）压缩十三折线法

g711a输入的是13位（S16的高13位），这种格式是经过特别设计的，便于数字设备进行快速运算。

1、取符号位并取反得到s
2、获取强度位eee，获取方法如图所示
3、获取高位样本位wxyz
4、组合为seeewxyz，将seeewxyz逢偶数为取补数，编码完毕

A-law如下表计算，第一列是采样点，共13bit，最高位为符号位。对于前两行，折线斜率均为1/2，跟负半段的相应区域位于同一段折线上，对于3到8行，斜率分别是1/4到1/128，共6段折线，加上负半段对应的6段折线，总共13段折线，这就是所谓的A-law十三段折线法。

Linear input code [note 1]	Compressed code XOR 01010101	Linear output code [note 2]
s0000000abcdx	s̄000abcd	s0000000abcd1
s0000001abcdx	s̄001abcd	s0000001abcd1
s000001abcdxx	s̄010abcd	s000001abcd10
s00001abcdxxx	s̄011abcd	s00001abcd100
s0001abcdxxxx	s̄100abcd	s0001abcd1000
s001abcdxxxxx	s̄101abcd	s001abcd10000
s01abcdxxxxxx	s̄110abcd	s01abcd100000
s1abcdxxxxxxx	s̄111abcd	s1abcd1000000

示例：

输入pcm数据为1234、二进制对应为（0000 0100 1101 0010）
二进制变换下排列组合方式（0 00001 0011 010010）
1、获取符号位最高位为0、取反、s=1
2、获取强度位00001、查表、编码制应该是eee=011
3、获取高位样本wxyz=0011
4、组合为10110011、逢偶数为取反为11100110、得到E6

```c
#define SIGN_BIT    (0x80)      /* Sign bit for a A-law byte. */
#define QUANT_MASK  (0xf)       /* Quantization field mask. */
#define NSEGS       (8)     /* Number of A-law segments. */
#define SEG_SHIFT   (4)     /* Left shift for segment number. */
#define SEG_MASK    (0x70)      /* Segment field mask. */
static int seg_aend[8] = {0x1F, 0x3F, 0x7F, 0xFF,
                0x1FF, 0x3FF, 0x7FF, 0xFFF};
static int seg_uend[8] = {0x3F, 0x7F, 0xFF, 0x1FF,
                0x3FF, 0x7FF, 0xFFF, 0x1FFF};

static int search(
    int val,    /* changed from "short" *drago* */
    int *   table,
    int size)   /* changed from "short" *drago* */
{
    int i;      /* changed from "short" *drago* */

    for (i = 0; i < size; i++) {
        if (val <= *table++)
            return (i);
    }
    return (size);
}

int linear2alaw(int pcm_val)        /* 2's complement (16-bit range) */
                                        /* changed from "short" *drago* */
{
    int     mask;   /* changed from "short" *drago* */
    int     seg;    /* changed from "short" *drago* */
    int     aval;

    pcm_val = pcm_val >> 3;//这里右移3位，因为采样值是16bit，而A-law是13bit，存储在高13位上，低3位被舍弃


    if (pcm_val >= 0) {
        mask = 0xD5;        /* sign (7th) bit = 1 二进制的11010101*/
    } else {
        mask = 0x55;        /* sign bit = 0  二进制的01010101*/
        pcm_val = -pcm_val - 1; //负数转换为正数计算
    }

    /* Convert the scaled magnitude to segment number. */
    seg = search(pcm_val, seg_aend, 8); //查找采样值对应哪一段折线

    /* Combine the sign, segment, and quantization bits. */

    if (seg >= 8)       /* out of range, return maximum value. */
        return (0x7F ^ mask);
    else {
//以下按照表格第一二列进行处理，低4位是数据，5~7位是指数，最高位是符号
        aval = seg << SEG_SHIFT;
        if (seg < 2)
            aval |= (pcm_val >> 1) & QUANT_MASK;
        else
            aval |= (pcm_val >> seg) & QUANT_MASK;
        return (aval ^ mask);
    }
}
```

```c
int alaw2linear(int a_val)
{
    int     t;      /* changed from "short" *drago* */
    int     seg;    /* changed from "short" *drago* */

    a_val ^= 0x55; //异或操作把mask还原

    t = (a_val & QUANT_MASK) << 4;//取低4位，即表中的abcd值，然后左移4位变成abcd0000
    seg = ((unsigned)a_val & SEG_MASK) >> SEG_SHIFT; //取中间3位，指数部分
    switch (seg) {
    case 0: //表中第一行，abcd0000 -> abcd1000
        t += 8;
        break;
    case 1: //表中第二行，abcd0000 -> 1abcd1000
        t += 0x108;
        break;
    default://表中其他行，abcd0000 -> 1abcd1000 的基础上继续左移(按照表格第二三列进行处理)
        t += 0x108;
        t <<= seg - 1;
    }
    return ((a_val & SIGN_BIT) ? t : -t);
}
```

u-law也叫g711u,使用在北美和日本，输入的是14位，编码算法就是查表，计算出：基础值+平均偏移值 没啥复杂算法，就是基础值+平均偏移值，

μ-law的公式如下，μ取值一般为255

$$F(x) = \operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \quad -1 \leq x \leq 1$$

$$F^{-1}(y) = \operatorname{sgn}(y)(1/\mu)((1+\mu)^{|y|}-1) \quad -1 \leq y \leq 1$$

相应的μ-law的计算方法如下表

Linear input value [note 1]	Compressed code XOR 11111111	Linear output value [note 2]
s00000001abcdx	s000abcd	s00000001abcd1
s0000001abcdxx	s001abcd	s0000001abcd10
s000001abcdxxx	s010abcd	s000001abcd100
s00001abcdxxxx	s011abcd	s00001abcd1000
s0001abcdxxxxx	s100abcd	s0001abcd10000
s001abcdxxxxxx	s101abcd	s001abcd100000
s01abcdxxxxxxx	s110abcd	s01abcd1000000
s1abcdxxxxxxxx	s111abcd	s1abcd10000000

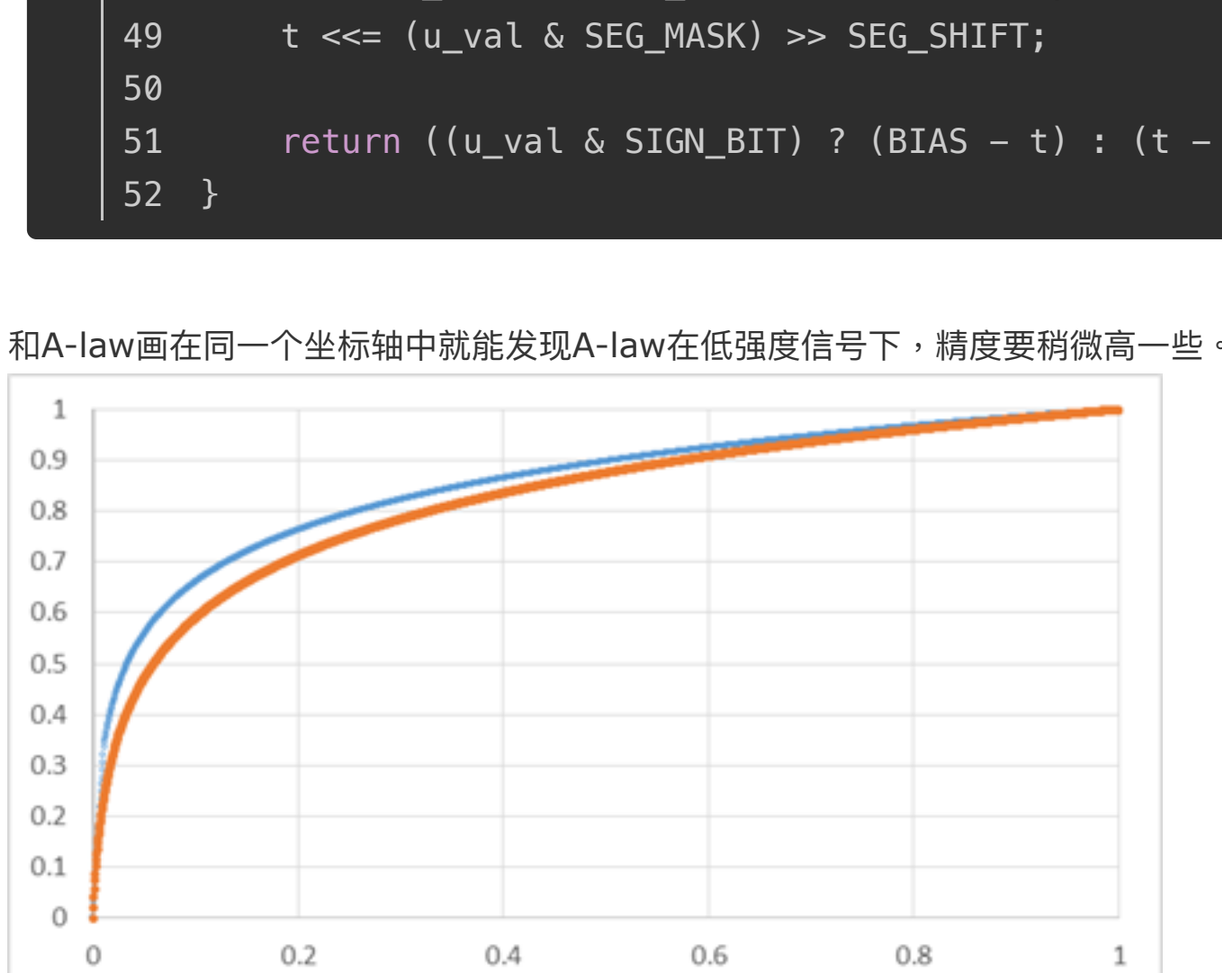
14 bit Binary Linear input code	8 bit Compressed code
+8158 to +4063 in 16 intervals of 256	0x80 + interval number
+4062 to +2015 in 16 intervals of 128	0x90 + interval number
+2014 to +991 in 16 intervals of 64	0xA0 + interval number
+990 to +479 in 16 intervals of 32	0xB0 + interval number
+478 to +223 in 16 intervals of 16	0xC0 + interval number
+222 to +95 in 16 intervals of 8	0xD0 + interval number
+94 to +31 in 16 intervals of 4	0xE0 + interval number
+30 to +1 in 15 intervals of 2	0xF0 + interval number
0	0xFF
−1	0x7F
−31 to −2 in 15 intervals of 2	0x70 + interval number
−95 to −32 in 16 intervals of 4	0x60 + interval number
−223 to −96 in 16 intervals of 8	0x50 + interval number
−479 to −224 in 16 intervals of 16	0x40 + interval number
−991 to −480 in 16 intervals of 32	0x30 + interval number
−2015 to −992 in 16 intervals of 64	0x20 + interval number
−4063 to −2016 in 16 intervals of 128	0x10 + interval number
−8159 to −4064 in 16 intervals of 256	0x00 + interval number

示例：

输入pcm数据为1234
1、取得范围值，查表得 **+2014 to +991 in 16 intervals of 64**
2、得到基础值为0xA0
3、得到间隔数为64
4、得到区间基本值2014
5、当前值1234和区间基本值差异2014-1234=780
6、偏移值=780/间隔数=780/64，取整得到12
7、输出为0xA0+12=0xAC

```c
#define BIAS        (0x84)      /* Bias for linear code. 线性码偏移值*/
#define CLIP            8159    //最大量化级数量

int linear2ulaw( int    pcm_val)    /* 2's complement (16-bit range) */
{
    int     mask;
    int     seg;
    int     uval;

    /* Get the sign and the magnitude of the value. */
    pcm_val = pcm_val >> 2;
    if (pcm_val < 0) {
        pcm_val = -pcm_val;
        mask = 0x7F;
    } else {
        mask = 0xFF;
    }
        if ( pcm_val > CLIP ) pcm_val = CLIP;       /* clip the magnitude 削波*/
    pcm_val += (BIAS >> 2);

    /* Convert the scaled magnitude to segment number. */
    seg = search(pcm_val, seg_uend, 8);

    /*
     * Combine the sign, segment, quantization bits;
     * and complement the code word.
     */
    if (seg >= 8)       /* out of range, return maximum value. */
        return (0x7F ^ mask);
    else {
        uval = (seg << 4) | ((pcm_val >> (seg + 1)) & 0xF);
        return (uval ^ mask);
    }

}

int ulaw2linear( int    u_val)
{
    int t;

    /* Complement to obtain normal u-law value. */
    u_val = ~u_val;

    /*
     * Extract and bias the quantization bits. Then
     * shift up by the segment number and subtract out the bias.
     */
    t = ((u_val & QUANT_MASK) << 3) + BIAS;
    t <<= (u_val & SEG_MASK) >> SEG_SHIFT;

    return ((u_val & SIGN_BIT) ? (BIAS - t) : (t - BIAS));
}
```

和A-law画在同一个坐标轴中就能发现A-law在低强度信号下，精度要稍微高一些。



以上是两种算法的连续条件下的计算公式，实际应用中，我们确实可以用浮点数计算的方式把F(x)结果计算出来，然后进行量化，但是这样一来计算量会比较大，实际上对于A-law（A=87.6时），是采用13折线近似的方式来计算的，而μ-law（μ=255时）则是15段折线近似的方式。

参考：

https://en.wikipedia.org/wiki/A-law_algorithm

https://github.com/quatanium/foscam-ios-sdk/blob/master/g726lib/g711.c

https://www.cnblogs.com/charybdis/p/8848457.html

http://www.21ic.com/evm/audio/201705/721797.htm