改进的快速算术编码及其在图像编码中的应用

黄 菁 朱日宏 李建欣
(南京理工大学电光学院,南京 210094)

摘 要 为了快速地进行图像编解码,提出了一种改进的快速算术编码方法,该方法是先通过编码区间宽度值与阈值的比较,并以字节为单位进行重归一化,同时输出以字节为单位的编码流;然后通过对当前输出字节的内容进行判断及处理,以便在不额外增加码流的情况下更加有效地处理编码过程中的进位传播的问题,从而简化了编解码过程。实验表明,在使用相同的二进制索引树概率模型时,该方法比 CACM87 及 Jiang 提出的方法具有更快的执行速度。最后,将该方法用于以 2×2 系数块为单位的 SPIHT 图像编码中,以降低位平面编解码过程的复杂度。

关键词 算术编码 重归一化 进位传播 图像编码 SPIHT
中图法分类号: TN919.81 **文献标识码**: A **文章编号**: 1006-8961(2007)07-1194-07

An Improved Fast Arithmetic Coding and Its Application in Image Coding

HUANG Jing, ZHU Ri-hong, LI Jian-xin
(*School of Electronic Engineering and Photoelectric Technology, Nanjing University of Science & Technology, Nanjing* 210094)

Abstract An improved fast arithmetic coding is proposed in order to encode and decode image quickly. It does renormalization in byte style by comparing the width of coding range with threshold, and the output codes are in byte style. In addition, it deals with the carry propagation problem more effectively without the additional output codes by judging the current output byte. So the proposed method significantly simplifies the process of encoding and decoding. The experimental results show that the proposed method runs faster than CACM87 and the method proposed by Jiang. We applied the proposed method to a 2×2 block based SPIHT image coding and the results show that the method can simplify the process of bitplane encoding and decoding.

Keywords arithmetic coding, renormalization, carry propagation, image coding, set partitioning in hierarchical trees (SPIHT)

1 引 言

算术编码作为一种熵编码方法,是以分数比特来代表一个编码符号,其不仅能够有效地压缩信源,并能使压缩码率接近信源的熵[1,2]。目前,算术编码已经被广泛地应用于图像编码领域。JBIG1 和 JPEG 的扩展选项是采用 QM 二值算术编码器作为熵编码器,而 JBIG2 和 JPEG2000 则采用 MQ 编码器;Witten 等人的通用算术编码器 CACM87[3] 也在 H.263 中得到应用,另外,在嵌入位平面图像编码器 EZW (embedded zerotree wavelet)、SPIHT (set partitioning in hierarchical trees) 和 SPECK (set partitioning embedded block) 中也采用这种通用算术编码器[4]。

在 CACM87 编码方法中,是通过判断编码区间的低限值和高限值的最高有效位是否相同来进行重归一化,该操作是以 bit 为单位进行的,并且输出以 bit 为单位的编码流。但考虑到可能会有下溢的情况发生,该方法还需要对编码区间的低限值和高限

基金项目:江苏省现代光学技术重点实验室开放基金资助项目(T8108030)
收稿日期:2005-12-09;**改回日期**:2006-04-21
第一作者简介:黄菁(1982~),女,博士研究生。主要研究方向为高光谱图像编码与视频编码。E-mail: jingjing@vip.sina.com

值进行 4 个条件的判断及做出相应的处理。Jiang 提出了一种改进的方法[5],即通过对编码区间的宽度值与阈值的比较来进行重归一化,该方法和 CACM87 方法相比有效地简化了编解码过程的重归一化操作。这种重归一化操作也是以 bit 为单位进行,并且输出以 bit 为单位的编码流,但由于重归一化操作在编码过程中存在进位传播的问题,因此文献[5]采用位填充技术来处理这个问题,这不仅额外地增加了码流,而且增加了编解码的复杂度[6,7]。本文提出一种更加有效的快速方法,即通过改变重归一化的阈值,使重归一化操作以字节为单位进行,从而有效地减少了重归一化操作的判断次数,并且以字节为单位输出编码流。另外,对于编码过程出现的进位传播问题,在不额外增加编码流和解码复杂度的情况下,本文采用了有效的处理方法,进一步提高了编解码过程的运算速度。本文最后把改进的方法用于以 2×2 系数块为单位的 SPIHT 图像编码中,以降低整个位平面编码过程的复杂度。

2　算术编码的基本原理

在算术编码中,输入符号被表示成 0 和 1 之间的一个实数区间[L,H),根据概率统计模型中与符号 s_i 对应的累计概率间隔[S_l/S,S_h/S)(S 下角 l,h 分别代表 low,high),将编码区间缩小为原来的一部分,编码方程表示为

$$L_{new} = L + R\frac{S_l}{S} \tag{1}$$

$$H_{new} = L + R\frac{S_h}{S} \tag{2}$$

$$R_{new} = H_{new} - L_{new} \tag{3}$$

式中,L 代表更新前的编码区间的低限值,H 代表更新前的编码区间的高限值,R 代表更新前的编码区间的宽度值,L_{new}代表更新后的编码区间的低限值,H_{new}代表更新后的编码区间的高限值,R_{new}代表更新后的编码区间的宽度值,S_l表示概率统计模型中与符号 s_{i-1}对应的累计频率值,S_h表示概率统计模型中与符号 s_i 对应的累计频率值,S 表示概率统计模型中所有符号的总频率值。假设 L,H 和 R 的有效位数为 b,则初始化时,H 为 2^b-1,L 为 0。在编码过程中,由于编码方程中的参数不断地更新,L 和 H 的值会不断地接近。当 L 和 H 的最高有效位相同时,则由于这个最高有效位在以后的编码过程中保持不变,所以可把它移出,作为输出编码流,同时把 L 和 H 左移一位。当 L 等于 $2^{b-1}-1$ 和 H 等于 2^{b-1} 时,则它们的值在以后的编码中保持不变,但由于它们的最高有效位不相同,所以会导致在后续的编码过程中一直不能输出编码流,这种情况称为下溢情况。CACM87 通过对 L 和 H 所出现的 4 种情况进行判断,并通过不同的处理方式来避免下溢的出现。以上的处理过程即为 CACM87 的重归一化过程。在文献[5]的方法中,重归一化的过程是,首先判断编码区间宽度值 R 与阈值 2^{b-1}的大小,如果 R 小于 2^{b-1},则表明 L 和 H 的最高有效位相同,此时需要把 L 的最高有效位移出,作为输出码流,然后把 L 和 R 左移一位。这种重归一化过程虽避免了下溢情况,但是会产生进位情况。因为编码方程在更新的时候,L 值会出现大于 2^b 的情况,此时 L 值的有效位数等于 $b+1$,L 值的最高有效位"1"实际上是已输出的编码流的最后一位,所以需要对已经输出的编码流的最后一位加 1。对于该进位传播的问题,文献[5]采用了位填充的处理方法,然而,这种处理方法不仅需要额外地增加码流,并且也增加了解码过程的复杂度。

3　改进的快速算术编码

3.1　编码结构

由于本文方法中的重归一化只对编码区间的宽度值 R 进行操作,并且输出的编码流只与编码区间的低限值 L 有关,所以可采用以下两个编码方程来更新编码区间的低限值 L 和宽度值 R:

$$L_{new} = L + R\frac{S_l}{S} \tag{4}$$

$$R_{new} = R \times \left(\frac{S_h}{S} - \frac{S_l}{S}\right) \tag{5}$$

假设 L 和 R 的有效位数为 b,用于算术计算的位数为 p,统计模型中的频率值的最大位数为 f,那么为了使式(4)和式(5)能够正确计算,它们必须要满足以下的关系:

$$0 \leqslant b + f \leqslant p \tag{6}$$

$$0 \leqslant f < b - 8 \tag{7}$$

如果算术计算的位数 $p=32$,则 $b=20$,$f=11$。

在重归一化过程中,如果编码区间宽度值 R 小于阈值 2^{b-8},则意味着 L 和 H 的最高有效字节是相同的,并且在以后的编码过程中,由于 L 的最高有效

字节保持不变,所以可把这个字节移出作为编码流,然后把 L 和 R 左移一个字节。上述操作不断地循环,直到 R 值不小于 2^{b-8} 为止。由于阈值设为 2^{b-8},所以重归一化以 Byte 为单位进行,并且输出以 Byte 为单位的编码流。而在文献[5]的方法中,由于阈值设为 2^{b-1},所以重归一化以 bit 为单位进行,并且输出以 bit 为单位的编码流。假设在更新编码方程的参数后,L 和 H 有 $k(0\leqslant k\leqslant 19)$ 个最高有效位是相同的,而用文献[5]的方法进行一次重归一化,其对 R 的判断次数为 u,本文方法对 R 的判断次数为 v,则 k、u 和 v 的关系为

$$u = k, v = \left\lfloor \frac{u}{8} \right\rfloor \tag{8}$$

从式(8)可以看出,进行一次重归一化过程,本文方法对 R 的判断次数比文献[5]的方法明显减少。另外,在本文方法中,当编码一个符号的重归一化过程结束时,R 的值为

$$2^{b-8} \leqslant R \leqslant 2^{b-m}, 0 \leqslant m < 8 \tag{9}$$

这意味着 L 和 H 还有 m 个最高有效位是相同的,但由于阈值为 2^{b-8},不满足进行重归一化的条件,所以这 m 个最高有效位在本次的重归一化中没有被输出。但在后面的编码过程中,如果满足重归一化的条件,那么它们就被输出。由于在接下来的编码过程中,更新编码方程中的参数后,L 中的这 m 个最高有效位不会再改变,因此可保证码流的完整性和正确性。

由于本文方法也会出现进位的情况,因此对进位传播问题也需要做相应的处理,即需要设置一个字节的 *buf*,用来存放从 L 中最新移出的不等于 0xFF 的编码流,当 L 中需要移出的最高有效字节为 0xFF 时,则不把它赋给 *buf*,而用计数器 c 来记录连续出现 L 的最高有效字节为 0xFF 情况的次数,同时用进位状态寄存器 z 记录进位情况。当进位情况发生时,则将进位状态寄存器 z 置 1,接着对 *buf* 加 1,然后判断计数器 c,如果计数器 c 不为 0,则连续输出 c 个 0x00,作为编码流。上述的处理过程不需要额外地增加编码流。

本文方法的编码过程如下:

```
encode_initialize ( )
  b=20,L=0,R=1≪b,buf=0x00,c=0,z=0
encode_symbol ( )
  encode_update ( )
  encode_renormalize ( )
encode_update ( )
  L=L+(R+S_l)/S
  R=(R×(S_h-S_l))/S
  if (L≥(1≪b)){
    buf=buf+1
    L=L-(1≪b)
    z=1
  }
encode_renormalize ( )
  while (R<(1≪(b-8))){
    if (L<(0xff≪(b-8))){
      put_Byte (buf)
      while (c>0) {
        if (z==1) put_Byte (0)
        else put_Byte (0xff)
        c=c-1
      }
      buf=L≫(b-8)
    }
    if (L≥(0xff≪(b-8)))
      c=c+1
    R=R≪8
    L=(L≪8)&((1≪b)-1)
    z=0
  }
```

两个重要部分说明如下:

(1) encode_update

首先在概率统计模型中通过输入符号找到对应的 S_l、S_h 和 S,然后采用式(4)和式(5)来更新 L 和 R。当 L 大于等于 2^b 时,即产生进位情况,此时需要对当前编码流的最后一位加 1。由于当前最新输出的一个不为 0xFF 的字节码流暂时存放在 *buf* 中,所以对 *buf* 加 1,并置状态寄存器 z 为 1,同时将 L 减去 2^b。

(2) encode_renormalize

当 R 小于 2^{b-8} 时,如果 L 的最高有效字节不等于 0xFF,则先把 *buf* 中的字节作为编码流输出,并且判断计数器 c 是否为 0,如果不是,则对状态值 z 进行判断,如果 z 等于 1,则连续输出 c 个 0,如果 z 等于 0,则连续输出 c 个 0xFF,并把 c 清零,然后把 L 的最高有效字节赋给 *buf*;如果 L 的最高有效字节等于 0xFF,则计数器 c 加 1。最后把 L 和 R 左移一个字节,并保证 L 的有效位数为 b,同时将状态寄存器 z 清零。以上操作不断循环,直到 R 值不小于 2^{b-8} 为止。

其解码过程用有效位为 b 的变量 V 来临时存放

解码输入数据,变量 V 的值随着重归一化的进行而不断改变,即不断地从解码输入数据中读出新的数据到 V 中。解码过程的 4 个重要部分说明如下:

(1) decode_initialize

L 初始化为 0,R 初始化为 2^b,将解码输入数据中的由最前面 b bits 所构成的二进制数值赋给 V。

(2) decode_get_symbol

先通过式(10)求得累计频率值 T,再通过查找 T 所属于的累计频率区间来找到该区间所对应的解码符号。如果 V 小于 L,这对应了编码过程中产生进位的情况,则需要把 V 加上 2^b,以保证解码正确进行。

$$T = \frac{(V - L + 1) \times S}{R} \tag{10}$$

(3) decode_update

由于解码方程与编码方程相同,因此可以利用式(4)和式(5)来更新解码区间的低限值 L 和宽度值 R。如果 L 大于等于 2^b,则把 L 减去 2^b。

(4) decode_renormalize

解码过程的重归一化操作也是通过判断解码区间宽度值 R 来进行的。当 $R < 2^{b-8}$ 时,就把 L 和 V 的最高有效字节移出,并且从解码输入数据中把一个字节的数据移入 V,同时把 R 值左移一个字节,并保证 L、R 和 V 的有效位数为 b。以上操作不断地循环,直到 R 值不小于 2^{b-8} 为止。

3.2　概率统计模型

本文方法的概率统计模型是采用二进制索引树的数据结构[8],树中的节点即表示索引值。二进制索引树结构将累计频率值分为若干部分,而且这些部分与该模型中符号的索引值的二进制表示形式有关。如图 1 所示,节点 4 包含了从 1 到 4 的累计频率值,节点 6 包含了从 5 到 6 的累计频率值,节点 8 包含了从 1 到 8 的累计频率值。二进制索引树结构中节点的前后关系可以用下面两个式子来表示

$$i_{\text{pre}} = i + i\&(-i) \tag{11}$$

$$i_{\text{back}} = i\&(i-1) \tag{12}$$

其中,i_{pre} 表示节点 i 的前节点,i_{back} 表示节点 i 的后节点,运算符 & 表示按位与运算。

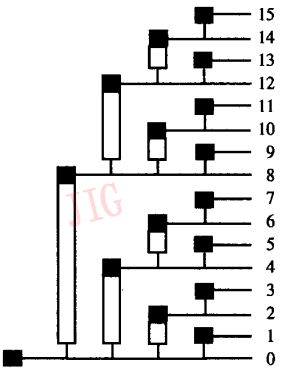

图 1　二进制索引树数据结构图

Fig. 1　Structure of binary indexed tree

二进制索引树概率统计模型在算术编码过程中主要有以下两个操作部分:

(1) 初始化

建立一个关于输入符号的概率统计模型,即用一个数组存放符号的累计频率值,初始化时,由于每个符号的出现频率都为 1,所以根据图 1 所示的二进制索引树结构所建立的与各符号对应的累计频率值之间的关系如表 1 所示,其中 i 表示符号的索引值,f_i 表示频率值,S_i 表示累计频率值,表 1 的第 2 行内容表示构成累计频率值所需要的频率值的索引值。

(2) 更新

每编码结束一个符号,都需要对当前输入符号的频率值加 1,并根据式(11)和式(12)建立的符号索引值的前后关系,然后只需要将该符号的频率值及其以后与之有关的索引值的累计频率值加 1,同时将总频率值加 1 即可。当总频率值超过了预设的值时,则需要根据符号索引值的前后关系将每个符号的累计频率值减少一半。

在解码时,可根据计算出的累计频率值寻找所对应的符号索引值,每一步都定义一个被寻找的索引值的基准,以便寻找其中心点,如果累计频率值大于其中心点的值,则将该值减去中心点的值,然后中心点变为新的索引值的基准,直到找到对应的符号索引值为止。

表 1　初始化的概率模型示意图

Tab. 1　Example of the initialized statistic model

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
内容	0	1	1...2	3	1...4	5	5...6	7	1...8	9	9...10	11	9...12	13	13...14	15
f_i	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S_i	1	1	2	1	4	1	2	1	8	1	2	1	4	1	2	1

4 改进的算术编码方法在 SPIHT 图像编码中的应用

在小波位平面图像编码中，为了有效地压缩位平面编码过程的输出符号，一般情况都会采用算术编码作为熵编码器，因此本文的方法可以应用于小波图像编码中。为了测试本文方法在图像编码中的性能，本文选择具有代表性的 SPIHT 位平面编码方法[9]进行编码，并用本文的算术编码器来压缩位平面编码过程的输出符号。为了减少 SPIHT 算法中大量的链表操作，本文把 2×2 的系数块作为 LIS 和 LIP 链表中的一个节点。为了与文献[9]的 SPIHT 算法描述进行比较，本文按照原文的格式用英文描述的以 2×2 系数块为单位的 SPIHT 算法如下：

(1) **Intialization**: output $n=\lfloor \mathrm{lb}(\max_{i,j}\{|C_{i,j}|\})\rfloor$; set the LSP as an empty list, and add each 2×2 block in LL subband as an entry to the LIP and the LIS, and set all entries in the LIS as type *A* entries.

(2) **Sorting Pass**:

1) for each entry in the LIP do:

① for each (x,y) in the entry $\{(i,j),(i+1,j),(i,j+1),(i+1,j+1)\}$

if $S_n(x,y)=1$ and it is not included in the LSP, then output "1" and the sign of $C_{x,y}$, and move (x,y) to the LSP.

else if $S_n(x,y)=0$ then output "0".

② if $C_{i,j}$ $C_{i+1,j}$ $C_{i,j+1}$ $C_{i+1,j+1}$ are all significant, then move the entry from the LIP.

2) for each entry in the LIS do:

① if the entry is of type *A* then

(Ⅰ) output $S_n(\boldsymbol{D}(i,j)\cup\boldsymbol{D}(i+1,j)\cup\boldsymbol{D}(i,j+1)\cup\boldsymbol{D}(i+1,j+1))$.

(Ⅱ) if $S_n(\boldsymbol{D}(i,j)\cup\boldsymbol{D}(i+1,j)\cup\boldsymbol{D}(i,j+1)\cup\boldsymbol{D}(i+1,j+1))=1$ then

(a) for each $(k,l)\in\{\boldsymbol{O}(i,j),\boldsymbol{O}(i+1,j),\boldsymbol{O}(i,j+1),\boldsymbol{O}(i+1,j+1)\}$

- output $S_n(k,l)$.
- if $S_n(k,l)=1$ then add (k,l) to the LSP and output the sign of $C_{k,l}$.
- if $S_n(k,l)=0$ then add the 2×2 block including (k,l) which is not included in the LIP to the end of LIP.

(b) if $\boldsymbol{L}(i,j)\neq\varnothing$ then move the 2×2 block to the end of the LIS as an entry of type *B*, and go to step ②; else remove the entry from the LIS.

② if the entry is of type *B* then

(Ⅰ) output $S_n(\boldsymbol{L}(i,j)\cup\boldsymbol{L}(i+1,j)\cup\boldsymbol{L}(i,j+1)\cup\boldsymbol{L}(i+1,j+1))$.

(Ⅱ) if $S_n(\boldsymbol{L}(i,j)\cup\boldsymbol{L}(i+1,j)\cup\boldsymbol{L}(i,j+1)\cup\boldsymbol{L}(i+1,j+1))=1$ then

(a) add each 2×2 block $\in\{\boldsymbol{O}(i,j),\boldsymbol{O}(i+1,j),\boldsymbol{O}(i,j+1),\boldsymbol{O}(i+1,j+1)\}$ to the end of the LIS as an entry of type *A*.

(b) remove the entry from the LIS.

(3) **Refinement Pass**: for each (i,j) in the LSP, except those included in the last sorting pass, output the *n*th most significant bit of $|C_{i,j}|$.

(4) **Quantization-Step Update**: decrement *n* by 1 and go to step (2).

为了有效地压缩位平面编码过程的输出符号，算术编码器利用 2×2 系数块自身的重要性作为上下文。位平面编码输出的符号可以分为 6 种编码类型，而在每种编码类型中根据当前符号的不同又可以设置不同的上下文，一共可以设置 65 个上下文，然后在每个上下文中，统计下次可能输出的符号种类，这个种类数目就是每个上下文中的符号数目（如表 2 所示）。编码时，算术编码器先根据位平面编码输出的符号选定相应的上下文，然后在该上下文中采用二进制索引树的统计模型进行编码。

表 2 SPIHT 中的上下文参考表

Tab. 2 Context in SPIHT

编码类型	上下文中的符号数目
符号位	2
幅度细化位	2
LIP 链表的节点	16,8,8,4,8,4,4,2,8,4,4,2,4,2,2
LIS 链表的 *A* 类节点	2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
LIS 链表的 *A* 类节点的孩子节点	16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16
LIS 链表的 *B* 类节点	2,2,2,2,2,2,2,2,2,2,2,2,2,2,2

5 实验结果与分析

为了测试本文提出的算术编码方法的编码性能，本文分别对 10 种不同类型、不同大小的文件进行了测试，测试环境为 CPU1.56G，内存 256M。表 3 给出了编码和解码的测试数据，3 种编码方法都是

表 3　算术编解码测试结果

Tab.3　Arithmetic encoding and decoding results

文件类型	文件大小(Byte)	压缩率(%)			编码时间(s)			解码时间(s)		
		CACM87 方法	文献[5]方法	本文方法	CACM87 方法	文献[5]方法	本文方法	CACM87 方法	文献[5]方法	本文方法
htm	14 485	76.3	76.2	75.5	0.027 6	0.025 5	0.019 8	0.025 4	0.031 1	0.015 9
jpg	27 323	99.4	99.5	99.5	0.046 7	0.045 0	0.027 3	0.047 0	0.047 0	0.031 3
txt	30 357	67.1	67.3	68.6	0.047 0	0.035 1	0.031 5	0.047 0	0.046 7	0.035 0
cpp	36 521	71.4	69.5	67.2	0.058 5	0.041 1	0.035 0	0.052 7	0.048 9	0.036 9
opt	48 640	45.6	45.6	44.3	0.054 6	0.047 0	0.044 6	0.050 9	0.050 9	0.037 0
doc	231 424	56.1	55.0	54.2	0.265 8	0.203 1	0.172 0	0.236 4	0.238 4	0.162 4
bmp	921 654	79.3	79.3	79.7	1.302 9	0.922 0	0.755 8	1.199 3	1.212 8	0.767 9
ppt	4 828 160	91.1	91.1	91.3	7.257 7	4.976 8	3.947 2	6.584 0	6.548 9	4.027 1
pdf	11 690 915	97.5	97.6	97.8	18.355 4	12.511 6	9.787 1	16.806 6	16.627 0	10.109 1
exe	11 745 936	94.7	94.6	94.7	18.090 0	12.386 6	9.742 3	16.552 6	16.369 1	10.082 1

采用二进制索引树的统计模型。从实验结果可以看出,本文方法的编码效率与 CACM87 和文献[5]的方法差不多,编码速度比 CACM87 方法平均提高了 38%,比文献[5]的方法平均提高了 20%,解码速度比 CACM87 方法平均提高了 34%,比文献[5]的方法平均提高了 35%。这个实验结果是合理的,因为编码过程中 CACM87 方法的重归一化方法需要对编码区间的低限值和高限值进行 4 种情况的判断,文献[5]方法的重归一化过程只需要通过判断编码区间宽度值与阈值的大小来进行,所以文献[5]方法的速度比 CACM87 方法快。而本文方法改变了阈值的大小,使得判断的次数比文献[5]的方法更少,重归一化过程更加简单,所以本文方法比文献[5]的方法快。在解码过程中,文献[5]的重归一化方法虽然比 CACM87 方法简单,但由于位填充技术对解码带来的复杂度,使得整个解码时间与 CACM87 方法差不多。

最后,为了测试本文的算术编码在采用 SPIHT 算法进行图像编码中的性能,本文对 Lena 图像进行了测试。表 4 给出了在不同码率情况下的编解码运行时间和解码图像的峰值信噪比(peak signal noise ratio, PSNR),其中"SPIHT(C)"表示 SPIHT 算法加上 CACM87 算法"SPIHT(J)"表示 SPIHT 算法加上文献[5]的算法,"SPIHT(F)"表示 SPIHT 算法加上本文算法。从表中的数据可以看出,采用本文的算术编码方法,其编解码的运行速度和解码图像的质量略有提高。由于整个位平面编码的运行时间本身比较少,再加上算术编码在整个位平面编码过程中占用的时间比例较少,所以运行速度提高的效果不是很明显。为了能更清楚地体现本文算法的性能,本文分别将 3 种算术编码方法与 SPIHT 算法占用的时间之比列于表 5。从表 5 可以看出,在编码过程中,本文算法与 SPIHT 算法占用的时间之比比 CACM87 算法与 SPIHT 算法占用的时间之比平均提高了 33%,比文献[5]的方法与 SPIHT 算法占用的时间之比平均提高了 20%,在解码过程中本文算法

表 4　图像编解码测试结果

Tab.4　Image encoding and decoding results

图像(512×512)	码率(bpp)	*PSNR*(dB)			编码时间(s)			解码时间(s)		
		SPIHT(C)	SPIHT(J)	SPIHT(F)	SPIHT(C)	SPIHT(J)	SPIHT(F)	SPIHT(C)	SPIHT(J)	SPIHT(F)
Lena	0.5	37.184	37.168	37.187	0.672	0.661	0.641	0.281	0.279	0.274
	1.0	40.392	40.375	40.409	0.953	0.941	0.922	0.485	0.484	0.469
	1.5	42.791	42.773	42.838	1.297	1.267	1.234	0.781	0.774	0.734
	2.0	45.105	45.078	45.170	1.503	1.467	1.431	0.969	0.968	0.922

表 5 算术编码与 SPIHT 占用时间的比率

Tab.5 Time consumption ratio of arithmetic coding to SPIHT

码率(bpp)	编码时间比率(%)			解码时间比率(%)		
	CACM87 方法	文献[5]方法	本文方法	CACM87 方法	文献[5]方法	本文方法
0.5	13.1	11.3	7.9	5.6	5	3
1.0	15.0	13.5	11.2	11.0	10.8	7.3
1.5	18.6	15.8	12.8	19.1	18.0	11.9
2.0	18.7	15.9	13.0	19.3	19.3	13.5

与 SPIHT 算法占用的时间之比比 CACM87 算法与 SPIHT 算法占用的时间之比平均提高了 36%，比文献[5]的方法与 SPIHT 算法占用的时间之比平均提高了 35%。由此可见，将本文算法作为熵编码方法应用于图像编码中，能够有效地降低位平面编解码过程的复杂度。

6 结 论

本文提出了一种改进的快速算术编码方法，与 CACM87 和文献[5]的方法相比，有以下两个优点：(1)以字节为单位的重归一化方法不仅减少了编解码过程中判断编码区间宽度值的操作，并且以字节为单位输出码流还加快了编解码的执行速度；(2)对编码过程中的进位传播问题，在不额外增加编码流和解码复杂度的情况下采取了有效的处理方法，使得编解码过程更加高效。另外，将改进算法用于嵌入位平面图像编码中还可以有效地减少位平面编码过程的复杂度。本文提出的改进方法不仅适用于数据压缩，而且也适用于图像编码、多光谱图像编码以及视频编码。如何将改进的算术编码方法更好地应用于多光谱图像编码，以获取更好的编码性能，是今后要进一步研究的内容。

参考文献(References)

1 Helman D R, Langdon G G. Data compression[J]. IEEE Potentials, 1988, 7(1): 25 ~ 28.

2 Howard P G, Vitter J S. Arithmetic coding for data compression[A]. In: Proceedings of the IEEE International Conference on Data Compression[C], Snowbird, Utah, USA, 1994: 857 ~ 865.

3 Witten I H, Neal R M, Cleary J G. Arithmetic coding for data compression[J]. Computing Practices, 1987, 30(6): 520 ~ 540.

4 Fowler J E. QccPack: An open-source software library for quantization, compression and coding[A]. In: Proceedings of the SPIE International Conference on Applications of Digital Image Processing[C], San Diego, California, USA, 2000: 294 ~ 301.

5 Jiang J. Novel design of arithmetic coding for data compression[J]. IEE Proceedings Computers & Digital Techniques, 1995, 142(6): 419 ~ 424.

6 Mitchell J L, Pennebaker W B. Optimal hardware and software arithmetic coding procedures of the Q-coder[J]. IBM Journal of Research and Development, 1998, 32(6): 727 ~ 735.

7 Moffat A. Critique of the paper 'Novel design of arithmetic coding for data cómpression' [J]. IEE Proceedings Computers & Digital Techniques, 1997, 144(6): 394 ~ 396.

8 Fenwick P M. A new data structure for cumulative frequency tables [J]. Software Practice and Experience, 1994, 24(3): 327 ~ 336.

9 Said A, Pearlman W A. A new, fast and efficient image codec based on partitioning in hierarchal trees[J]. IEEE Transactions on Circuits and Systems for Video Technology, 1996, 6(3): 243 ~ 250.