[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06T 15/00 (2006.01)
G06T 15/70 (2006.01)
H03M 7/48 (2006.01)

[12] 发明专利说明书

专利号 ZL 02140026.1

[45] 授权公告日 2007 年 1 月 10 日

[11] 授权公告号 CN 1294540C

[22] 申请日 2002.11.27 [21] 申请号 02140026.1

[30] 优先权

[32] 2001.11.27 [33] US [31] 60/333，130

[32] 2001.12. 3 [33] US [31] 60/334，541

[32] 2001.12.26 [33] US [31] 60/342，101

[32] 2002. 4. 4 [33] US [31] 60/369，597

[32] 2002.10.19 [33] KR [31] 64008/02

[73] 专利权人 三星电子株式会社

地址 韩国京畿道

[72] 发明人 李信俊 郑锡润 张义善 禹相玉 韩万镇 金道均 张敬子

审查员 唐田田

[74] 专利代理机构 北京市柳沈律师事务所

代理人 黄小临 王志森

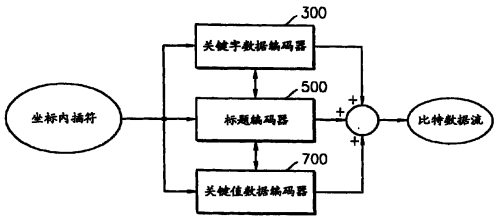
权利要求书 7 页 说明书 49 页 附图 56 页

[54] 发明名称

编解码坐标内插符关键字数据和关键值数据的装置

[57] 摘要

本发明提供一种用于编码和解码坐标内插符的关键字数据和关键值数据的装置以及一种其上写入编码有坐标内插符的比特数据流的记录介质。 该比特数据流包括：关键字数据编码/解码信息，其中编码有关键字数据和为了解码该关键字数据所需要的信息；和关键值数据编码/解码信息，其中编码有关键值数据和为了解码该关键值数据所需要的信息。 所述关键字数据信息包括：反向 DND 操作信息、第一反向 DPCM 操作信息和第一反向量化信息。 所述关键值数据编码/解码信息包括：从该比特数据流熵解码的字典解码信息、第一位置索引、字典解码模式、第二反向 DPCM 操作信息以及使用在反向量化中的第二反向量化信息。



1. 一种用于编码坐标内插符的装置，所述坐标内插符包括：关键字数据，指示在一个时间轴上的每一关键帧的位置；以及关键值数据，使用每一个都由 x、y 和 z 成分组成的顶点的坐标表示在每一关键帧中的目标的每一顶点的位置，

所述装置包括：

一个关键字数据编码器，其包括：

一个第一量化器，使用预定的量化比特量化坐标内插符的关键字数据；

一个第一差分脉冲编码调制处理器，产生所述量化的关键字数据的差分数据；

一个分割-与-分割处理器，根据在差分数据与该差分数据当中的最大值和最小值之间的一种关系对所述差分数据执行一个分割-与-分割操作，和

一个第一熵编码器，熵编码从所述分割-与-分割处理器输入的该差分数据，

一个关键值数据编码器，其包括：

一个第二量化器，使用预定的量化比特量化坐标内插符的关键值数据；

一个第二差分脉冲编码调制处理器，通过对量化关键值数据的每一顶点的成分执行一个预定的差分脉冲编码调制操作，产生基于量化关键值数据的每一顶点坐标的时间变化的差分数据和基于量化关键值数据的每一顶点坐标的空间变化的差分数据；

一个字典编码器，产生一些符号以及指示这些符号的位置的索引，所述符号表示每一顶点的差分脉冲编码调制的差分数据和已经对于该差分数据执行的一个差分脉冲编码调制操作模式；和

一个第二熵编码器，熵编码所述符号和所述索引，以及

一个标题编码器，对为了解码由所述关键字数据编码器和所述关键值数据编码器所编码的一个比特数据流所需要的信息进行编码。

2. 根据权利要求 1 所述的装置，其中，所述关键字数据编码器还包括

一个线性关键字编码器，所述线性关键字编码器在输入其中的关键字数据当中识别一个关键字数据线性增加的区域，并且编码该区域。

3. 根据权利要求 1 所述的装置，其中，所述关键字数据编码器还包括:

一个移位器，从第一差分脉冲编码调制处理器输入的差分数据当中获得具有最高频率的一个差分数据(模式)并且从所述差分数据中减去该模式；和

一个折叠处理器，把移位的差分数据转换成正数或负数，并且该分割-与-分割处理器根据进行编码所需的比特数量从所述移位器输入的差分数据和从所述折叠处理器输入的差分数据中选择之一，分割-与-分割该差分数据并输出该选择的差分数据。

4. 根据权利要求 1 所述的装置，其中，所述第二差分脉冲编码调制处理器包括:

一个差分脉冲编码调制操作器，对于量化的坐标内插符的每一顶点的每一成分执行一个时间差分脉冲编码调制操作，以便产生在一个关键帧中的顶点和另一关键帧中的顶点之间的第一差分数据；对于该量化的坐标内插符的每一顶点的每一成分执行一个空间差分脉冲编码调制操作，以便产生在同一个关键帧中的顶点之间的第二差分数据；并且对于该量化的坐标内插符的每一顶点的每一成分执行一个时空差分脉冲编码调制操作，以便产生在顶点和关键帧之间的第三差分数据；

一个循环量化器，对于从所述差分脉冲编码调制操作器输入的所述第一至第三差分数据执行一个循环量化操作，以便降低其范围； 和

一个差分脉冲编码调制模式选择器，根据为了进行编码所需要的比特数目选择已经循环量化的所述第一至第三差分数据之一，并且输出所选择的差分数据。

5. 根据权利要求 1 所述的装置，其中，所述字典编码器包括:

一个差分脉冲编码调制模式编码器，产生一些符号以及指示这些符号的位置的索引，所述符号表示已经对于每一顶点的每一成分的数据执行的差分脉冲编码调制模式的组合；和

一个出现模式编码器，产生一些对应于每一顶点的每一成分的输入差分数据的符号以及指示这些符号的位置的索引；

一个增量模式编码器，产生指示在每一顶点的每一成分的输入差分数据中是否存在预定的符号的一个符号标志以及指示这些符号的位置的位置索

引；和

一个表格大小计算器，计算由对应于所述输入差分数据的符号构成的第一符号表的大小以及所述符号标志的大小，并且根据该第一符号表和该符号标志的大小，把从所述差分脉冲编码调制模式编码器输入的每一顶点的每一成分的输入差分数据输出到所述出现模式编码器或所述增量模式编码器。。

6. 一种用于解码一个比特数据流的装置，所述比特数据流中编码有坐标内插符，该坐标内插符包括指示在一个时间轴上每一关键帧的位置的关键字数据和使用每一个都由 x、y 和 z 成分组成的顶点坐标表示在每一关键帧中的一个目标的每一顶点的位置的关键值数据，该装置包括：

一个标题解码器，其对解码来自一个输入比特数据流的关键字数据和关键值数据所需的标题信息执行解码，并且输出该解码的标题信息；

一个关键字数据解码器，其包括：

一个第一熵解码器，熵解码所述输入的比特数据流并且输出解码的关键字数据的差分数据；

一个反向分割-与-分割处理器，根据由所述标题解码器从所述输入比特数据流读出的分割-与-分割阶数，通过对于所述差分数据执行一个反向分割-与-分割操作而扩展该熵解码的关键字数据的差分数据的范围；

一个第一反向差分脉冲编码调制处理器，对于从所述反向分割-与-分割处理器输入的差分数据，执行与从所述标题解码器输入的差分脉冲编码调制的阶数一样多次的反向差分脉冲编码调制操作，并且输出量化的关键字数据；和

一个第一反向量化器，反向量化所述量化的关键字数据并且输出解码的关键字数据，以及

一个关键值数据解码器，其包括：

一个第二熵解码器，通过熵解码输入的比特数据流产生将要被字典解码的数据，包括关键值数据的差分数据的符号、指示该符号的位置的索引和一个差分脉冲编码调制操作模式；

一个字典解码器，根据从所述标题解码器输入的一个字典解码模式信息，通过执行一个字典解码操作产生关键值数据的差分数据；

一个第二反向差分脉冲编码调制处理器，根据所述差分脉冲编码调制操作模式，通过恢复从所述字典解码器输入的关键帧之间的差分数据和顶

点之间的差分数据产生量化数据；和

一个第二反向量化器，通过反向量化所述量化的数据产生恢复的关键值数据。

7. 根据权利要求 6 所述的装置，其中，所述关键字数据解码器还包括：

一个反向折叠处理器，根据从所述标题解码器输入的分割-与-分割的阶数，对于从所述反向分割-与-分割处理器输入的差分数据执行一个反向折叠操作，以便把该差分数据分割成正数和负数，或者旁路该差分数据；和

一个反向移位器，通过把从所述标题解码器输入的一个预定的模式相加到所述差分数据而变换从所述反向分割-与-分割处理器或所述反向折叠处理器输入的差分数据的范围，并且所述第一反向差分脉冲编码调制处理器恢复从所述反向移位器输入的差分数据并且输出量化的关键字数据。

8. 根据权利要求 6 所述的装置，其中所述第二反向差分脉冲编码调制处理器包括：

一个反向时间差分脉冲编码调制操作器，对于根据时间的推移而改变的每一顶点的差分数据执行一个反向差分脉冲编码调制操作；

一个反向空间差分脉冲编码调制操作器，对于在每一个顶点和对应于一个预定时间瞬时的顶点的一个参考顶点之间的差分数据执行一个反向差分脉冲编码调制操作；以及

一个反向差分脉冲编码调制模式选择器，根据所述差分脉冲编码调制操作模式把输入其中的差分数据输出到所述反向时间差分脉冲编码调制操作器或所述反向空间差分脉冲编码调制操作器。

9. 一种用于在基于关键帧的图形动画中编码坐标内插符的方法，所述坐标内插符包括指示在一个时间轴上每一关键帧的位置的关键字数据和使用每一个都由 x、y 和 z 成分组成的顶点坐标表示在每一关键帧中的一个目标的每一顶点的位置的关键值数据，所述方法包括：

产生关键字数据编码/解码信息，其中编码有关键字数据；和

产生关键值数据编码/解码信息，其中编码有关键值数据，

其中，所述关键字数据编码/解码信息包括：

反向分割-与-分割操作信息，包括反向分割-与-分割的阶数，该阶数指示将要对于由熵解码所述比特数据流产生的差分数据执行的反向分割-与-分割的预定周期数，所述反向分割-与-分割的目的是扩展该差分数据以及使用

在反向分割-与-分割操作的每一周期中的差分数据当中的最大和最小值的范围;

第一反向差分脉冲编码调制操作信息，包括将要对于该反向分割-与-分割的差分数据执行的反向差分脉冲编码调制操作的阶数，所述反向差分脉冲编码调制的目的是把该反向分割-与-分割的差分数据转换成被用于反向差分脉冲编码调制操作的每一周期的量化关键字数据和帧内关键字数据；和

第一反向量化信息，以便通过反向量化该量化的关键字数据而产生恢复的关键字数据，以及

所述关键值数据编码/解码信息包括:

字典解码信息，包括关于表示被字典编码并从所述比特数据流熵解码的关键值数据的差分数据的符号的信息，指示所述符号的位置第一位置索引，和指定将要对于该第一位置索引执行的一个字典解码方法的字典解码模式;

第二反向差分脉冲编码调制操作信息，包括指示符号位置的第二位置索引，其被使用在把每一顶点的成分的字典解码的差分数据转换成量化关键值数据的一个反向差分脉冲编码调制操作中并且对应于反向差分脉冲编码调制操作模式的组合；以及

第二反向量化信息，以便通过反向量化该量化的关键值数据而产生恢复的关键值数据。

10. 根据权利要求 9 所述的方法，其中，所述反向分割-与-分割操作信息还包括一个标志，表示是否将对于受到一个反向分割-与-分割操作的差分数据执行一个下移位操作。

11. 根据权利要求 9 所述的方法，其中，所述第一反向量化信息包括，当反向量化该量化的关键字数据时使用的一个反向量化比特大小以及在量化关键字数据中的最大和最小值。

12. 根据权利要求 11 所述的方法，其中，在所述量化关键字数据中的最大和最小值最小化该量化关键字数据的量化误差。

13. 根据权利要求 9 所述的方法，其中，所述关键字数据编码/解码信息还包括，用于解码包括在该比特数据流中的一个线性关键字区域的线性关键字解码信息，

其中，所述线性关键字解码信息包括一个指示在该关键字数据中是否存在其中关键字数据线性增加的线性关键字区域标志、在该线性关键字区域中

包括的关键字数据的数目以及对应于该线性关键字区域的开始和结束关键字数据。

14. 根据权利要求 13 所述的方法，其中，所述对应于线性关键字区域的开始和结束的关键字数据的每一个被编码成十进制的一个尾数和一个指数。

15. 根据权利要求 9 所述的方法，其中，当该字典解码模式是一个出现模式时，所述字典解码信息包括对应于包括在该差分数据中的差分值的符号和指示这些符号的位置的位置索引；并且当该字典解码模式是一个增量模式时，所述字典解码信息包括指示包括在该差分数据中的该差分值是否存在于一个预定的符号表中的一个符号标志以及指示对应于该符号标志的符号位置的位置索引。

16. 根据权利要求 9 所述的方法，其中，所述第二反向差分脉冲编码调制操作信息还包括一个反向差分脉冲编码调制模式标志，指示对应于将要对于每一顶点的差分数据执行的一个反向差分脉冲编码调制操作的一个符号是否存在于存储将要对于该顶点的每一成分执行的反向差分脉冲编码调制操作模式的多种组合的一个表格中的所有的符号当中。

17. 根据权利要求 16 所述的方法，其中，所述反向差分脉冲编码调制操作模式的组合是将对于差分数据的每一个成份执行的一个反向时间差分脉冲编码调制操作、一个反向空间差分脉冲编码调制操作和一个反向时空差分脉冲编码调制操作的任意组合。

18. 根据权利要求 17 所述的方法，其中，所述第二反向差分脉冲编码调制操作信息还包括一个参考顶点标志，当将要对于差分数据执行的该反向差分脉冲编码调制操作由该反向差分脉冲编码调制模式标志确定为是一个反向空间差分脉冲编码调制操作或一个反向时空差分脉冲编码调制操作时，该参考顶点标志表示对应于受到一个反向空间差分脉冲编码调制操作或一个反向时空差分脉冲编码调制操作的顶点的一个参考顶点。

19. 根据权利要求 9 所述的方法，其中，所述第二反向量化信息包括：

表示受到一个反向量化操作的关键值数据的顶点的一个顶点选择标志，

由该顶点选择标志选择的该顶点的每一成分的关键值数据当中的最小值和该顶点的每一成分的关键值数据的数据范围中的最大范围；以及

用于反向量化该选择的顶点的关键值数据的一个反向量化比特大小。

20. 根据权利要求 19 所述的方法，其中，所述第二反向量化信息还包括:

将要反向量化的关键值数据中包括的顶点数目;

该关键值数据的有效位数的最大数目;

在每一顶点的每一成分的数据当中的最大值中的一个最大值和一个最小值，以及

在每一顶点的每一成分的数据里最小值中的一个最大值和一个最小值。

21. 根据权利要求 9 所述的方法，其中，所述关键值数据编码/解码信息还包括一个标志，指示所述关键值数据的编码模式是否为一个转置模式或一个顶点模式。

编解码坐标内插符关键字数据和关键值数据的装置

技术领域

本发明涉及用于编码和解码合成图像的装置和方法，尤其涉及用于编码和解码一种坐标内插符的装置和方法，该坐标内插符使用每个都包括 x、y、z 成分的顶点坐标表示在一个基于关键帧的图形动画中的目标的每个顶点的位置。

背景技术

三维(3D)动画技术已经广泛地采用在 3D 计算机游戏或虚拟现实计算机应用中。虚拟现实模型语言(VRML)是这种 3D 动画技术的一个典型实例。

国际多媒体标准，例如用于场景(BIFS) 的 MPEG-4 和虚拟现实模型语言(VRML)，使用一个内插符节点支持基于关键帧的 3D 动画。在 MPEG-4 BIFS 和 VRML 中，有各种类型的内插符，包括标量内插符、位置内插符、坐标内插符、定向内插符、法线内插符和色彩内插符，这些内插符以及其功能和特性在表 1 中示出。

表 1

内插符	特性	功能
标量内插符	标量变化的线性内插	能够表示区域、直径和亮度
位置内插符	在 3D 坐标上的线性内插	在 3D 空间中的平行移动
定向内插符	3D 坐标轴和旋转量的线性内插	3D 空间中的旋转
座标内插符	3D 坐标中的变化的线性内插	3D 图像变形
法线内插符	法线 3D 坐标的线性内插	能够表示法线 3D 矢量中的变化
色彩内插符	彩色信息的线性内插	能够表示色彩中的变化

在表 1 所示的内插符中，坐标内插符被用于表示在基于关键帧的动画

中构成3D目标的每一顶点位置上的信息，并且包括关键字字段和关键值字段。关键字字段使用范围在-∞和∞之间的不连续数字，表示在时间轴上每一关键帧的位置。关键值字段规定了在由每一关键字表示的确定瞬时的构成3D目标的每一个顶点位置上的信息，并且包括三个成分x、y和z。每一个关键值字段包括与关键字字段一样多的关键字值。在这种基于关键帧的动画中，预定的关键帧定位在一个时间轴的任意位置，并且由线性内插填充在关键帧之间的动画数据。

由于MPEG-4 BIFS和VRML中采用线性内插，所以要求相当量的关键字数据和关键值数据来使用线性内插符把一个动画表现得尽可能地自然和平滑。此外，为了存储和发送这种自然和平滑的动画，需要相当大容量的存储器和可观的时间。因此，最好是选择压缩内插符，以便使得更容易存储和发送该内插符。

在已经采用在MPEG-4 BIFS中用于编解码内插符节点的方法之一的预测MF字段编码(PMFC)中，使用量化器、差分脉码调制(DPCM)操作器和熵编码器编码坐标内插符的关键值数据，如图1所示。参考图1，量化器和DPCM操作器消除该关键值数据的冗余，该DPCM操作器把其操作的结果输出到熵编码器。然而，PMFC在对关键值数据进行编码中不是充分有效，因为其只熵编码从一般DPCM操作获得的差分数据，并且仅考虑在一个动画中构成3D目标顶点之间的空间相关性而不考虑在这种顶点之间的时间相关性，在一个基于关键帧的动画中，这种时间相关性是很重要的。

发明内容

为了解决上述和其它问题，本发明的一个方面是提供一个用于编码坐标内插符的装置，包括：一个关键字数据编码器，其以高效率压缩动画关键字数据，同时缩减在数据当中的冗余度；和一个关键值数据编码器，其考虑在关键值数据之间的时间相关性以及在关键值数据之间的空间相关而编码关键值数据。

本发明的另一方面是提供一种用于解码的装置，解码由根据本发明的用于编码坐标内插符的装置编码的一个比特数据流。

本发明的另一方面是提供一种记录介质，能够以高压缩比率提供一个高质量的动画，其上记录有由根据本发明的用于编码坐标内插符的方法和

装置所编码并且由根据本发明的用于解码坐标内插符的方法和装置待解码的一个比特数据流。

因此，为了实现本发明的上述和其它方面，提供一个用于编码坐标内插符的装置，该坐标内插符包括：指示在时间轴上的每一关键帧的位置的关键字数据；以及使用每一个都包括 x、y 和 z 成分的顶点的坐标表示在每一关键帧中的一个目标的每一顶点的位置的关键字数据。该装置包括一个关键字数据编码器、一个关键值数据编码器和一个标题编码器。该关键字数据编码器包括一个第一量化器，其使用预定的量化比特量化坐标内插符的关键字数据，一个第一 DPCM 处理器，其产生该量化的关键字数据的差分数据，一个 DND 处理器，其根据在差分数据和在该差分数据当中的一个最大值和一个最小值之间的一个关系对于该差分数据执行一个 DND 操作，以及一个第一熵编码器，其熵编码从该 DND 处理器输入的该差分数据。该关键值数据编码器包括一个第二量化器，其使用预定的量化比特量化坐标内插符的关键值数据；一个第二 DPCM 处理器，其通过对于该量化关键值数据的每一顶点的成分执行一个预定的 DPCM 操作而产生基于量化关键值数据的每一顶点坐标的时间变化的差分数据和基于量化关键值数据的每一顶点坐标的空间变化的差分数据；一个字典编码器，其产生表示每一顶点的 DPCM 的差分数据和已经对于该差分数据执行的一个 DPCM 操作模式的符号以及指示该符号的位置的索引；和一个第二熵编码器，其熵编码该符号和该索引。该标题编码器，编码为了解码由该关键字数据编码器和该关键值数据编码器编码的一个比特数据流所需要的信息。

为了实现本发明的上述以及其它方面，提供有一个用于解码一个比特数据流的装置，其中编码有坐标内插符，该坐标内插符包括指示在一个时间轴上每一关键帧的位置的关键字数据和使用每一个都包括 x、y 和 z 成分的顶点坐标表示在每一关键帧中的一个目标的每一顶点的位置的关键值数据。该装置包括一个关键字数据解码器和一个关键值数据解码器。该关键字数据解码器包括一个标题解码器，其解码为了解码来自输入比特数据流的关键字数据和关键值数据所需要的标题信息并且输出该解码的标题信息；一个第一熵解码器，其熵解码该输入的比特数据流并且输出解码的关键字数据的差分数据；一个反向 DND 处理器，其根据由该标题解码器从该输入比特数据流读出的 DND 阶数，通过对于该差分数据执行一个反向 DND

操作而扩展该熵解码的关键字数据的差分数据的范围;一个第一反向DPCM处理器,其对于从该反向DND处理器输入的差分数据执行与从该标题解码器输入的DPCM的阶数一样多次的反向DPCM操作,并且输出量化的关键字数据;以及一个第一反向量化器,其反向量化该量化的关键字数据并且输出解码的关键字数据。该关键值数据解码器包括:一个第二熵解码器,其通过熵解码该输入的比特数据流而产生将要被字典解码的数据,包括关键值数据的差分数据的符号、指示该符号的位置的索引和一个DPCM操作模式;一个字典解码器,其根据从该标题解码器输入的一个字典解码模式信息,通过执行一个字典解码操作而产生关键值数据的差分数据;一个第二反向DPCM处理器,其根据该DPCM操作模式,通过恢复从该字典解码器输入的关键帧之间的差分数据和顶点之间的差分数据而产生量化数据;以及一个第二反向量化器,其通过反向量化该量化的数据而产生恢复的关键值数据。

为了实现本发明的上述以及其它方面,提供有一个比特数据流,其中编码有坐标内插符,该坐标内插符包括指示在一个时间轴上每一关键帧的位置的关键字数据和使用每一个都包括x、y和z成分的顶点坐标表示在每一关键帧中的一个目标的每一顶点的位置的关键值数据。该比特数据流包括:关键字数据编码/解码信息,其中编码有关键字数据和为了解码该关键字数据所需要的信息;和关键值数据编码/解码信息,其中编码有关键值数据和为了解码该关键值数据所需要的信息。该关键字数据信息包括:反向DND操作信息,包括表示将要对于由熵解码该比特数据流产生的差分数据执行的反向DND的预定的周期数的反向DND的阶数,以便扩展该差分数据的范围以及使用在反向DND操作的每一周期中的差分数据当中的最大和最小值;第一反向DPCM操作信息,包括将要对于该反向DND的差分数据执行的反向DPCM操作的阶数,以便把该反向DND的差分数据转换成被用于反向DPCM操作的每一周期的关键字数据和帧内关键字数据;和使用在反向量化中的第一反向量化信息,以便通过反向量化该量化的关键字数据而产生恢复的关键字数据。该关键值数据编码/解码信息包括:从该比特数据流熵解码的字典解码信息,包括关于表示被字典编码的关键值数据的差分数据的符号的信息;第一位置索引,指示该符号的位置;和字典解码模式,指示将要对于该第一位置索引执行的一个字典解码方法;第二反向

DPCM 操作信息，包括指示符号位置的第二位置索引，其被使用在把每一顶点的成分的字典解码的差分数据转换成量化关键值数据的一个反向 DPCM 操作中，并且对应于反向 DPCM 操作模式的组合；以及使用在反向量化中的第二反向量化信息，以便通过反向量化该量化的关键值数据而产生恢复的关键值数据。

附图说明

通过参照附图对优选实施例的详细描述，本发明的上述目的和优点将变得更为显见：

图 1 是传统坐标内插符编码器和传统坐标内插符解码器的框图；

图 2 是一个根据本发明一个优选实施例的用于编码坐标内插符装置的框图；

图 3A 是一个根据本发明一个优选实施例的关键字数据编码器的框图；

图 3B 是一个图 3A 所示的 DND 处理器的框图；

图 4A 至 4G 是一个按照本发明一个优选实施例用于编码关键字数据方法的流程图；

图 5 是一个表示函数 encodeSignedAAC 的一个实例的示意图；

图 6A 至 6J 是表示关键字数据的示意图，该关键字数据已经经过了一个按照本发明优选实施例的关键字数据编码操作；

图 7A 是一个根据本发明优选实施例的关键值数据编码器的框图，而图 7B 是一个根据本发明优选实施例的用于编码坐标内插符的关键值数据的一种方法的流程图；

图 8A 是一个在根据本发明优选实施例的关键值数据编码器中的 DPCM 处理器的框图，而图 8B 是一个根据本发明优选实施例的字典编码器的框图；

图 9A 是一个根据本发明优选实施例的量化操作的流程图，图 9B 是一个根据本发明优选实施例的 DPCM 操作的流程图，图 9C 是一个根据本发明优选实施例的字典编码操作的流程图，而图 9D 是一个根据本发明优选实施例的熵编码操作的流程图；

图 10A 至 10C 分别是说明量化的关键值数据、DPCM 的关键值数据和循环量化的关键值数据的示意图；

图 11A 是一个说明根据本发明优选实施例的 DPCM 模式编码方法的示意图，图 11B 是一个说明根据本发明优选实施例的出现模式编码方法的示意图，而图 11C 是一个说明根据本发明优选实施例的增量模式编码方法的示意图；

图 12 是一个根据本发明一个优选实施例的用于解码坐标内插符的装置的框图；

图 13 是一个根据本发明一个优选实施例的关键字数据解码器的框图；

图 14A 是一个根据本发明优选实施例的用于解码关键字数据的一种方法的流程图，而图 14B 是图 14A 所示步骤 S14500 的详细流程图；

图 15A 是一个根据本发明优选实施例的用于解码关键值数据的装置的框图，而图 15B 是根据本发明优选实施例的用于解码关键值数据的一种方法的流程图；

图 16A 是一个根据本发明优选实施例的字典解码器的框图，而图 16B 是一个在根据本发明的一个用于解码关键值数据的装置中的反向 DPCM 处理器的框图；

图 17A 是一个根据本发明优选实施例的字典解码方法的流程图，而图 17B 是一个反向 DPCM 操作的流程图；

图 18A 是一个表示比特数据流的结构的示意图，该比特数据流包括坐标内插符的顶点和其成份数据；

图 18B 是一个表示程序代码的实例的示意图，通过该程序代码实现使用在根据本发明的一个熵解码器中的一个函数 decodeSignedQuasiAAC()；

图 19A 是一个说明根据本发明优选实施例的一种 DPCM 模式解码方法的示意图，图 19B 是一个说明根据本发明优选实施例的出现模式解码方法的示意图，而图 19C 是一个说明根据本发明优选实施例的增量模式解码方法的示意图；以及

图 20A 至 20L 是示出 SDL 程序代码的实例的示出，通过该 SDL 程序代码，实现根据本发明优选实施例的用于对坐标内插符的关键字数据和关键值数据进行解码的装置。

具体实施方式

下面，将参照表示本发明优选实施例的附图更详细地描述根据本发明

优选实施例的用于编码坐标内插符的装置。

图 2 是根据本发明一个优选实施例的用于编码坐标内插符的装置的框图。参考图 2，用于编码坐标内插符的装置包括：关键字数据编码器 300，编码输入其中的坐标内插符的关键字数据；关键值数据编码器 700，编码输入其中的坐标内插符的关键值数据；和标题编码器 500，编码为了解码由关键字数据编码器 300 编码的关键字数据和由关键值数据编码器 700 编码的关键值数据所需要的信息。

下面参照图 3A 至 6J 对关键字数据编码器 300 进行描述。

图 3A 是关键字数据编码器 300 的一个框图。参考图 3A，关键字数据编码器 300 包括一个线性关键字编码器 310、一个量化器 320、一个 DPCM 处理器 330、一个移位器 340、一个折叠处理器 350、一个 DND 处理器 360 和一个熵编码器 370。

线性关键字编码器 310 识别一个区域，其中该关键字数据在一个完整的关键字数据范围中线性地增加，并且编码该区域。量化器 320 使用能够最小化一个量化误差的量化方法量化输入其中的关键字数据。DPCM 处理器 330 接收量化的关键字数据并且产生该量化的关键字数据的差分数据。移位器 340 从所有的差分数据减去具有最高频率的一个差分数据。折叠处理器 350 把所有的的差分数据变换到一个正数区域或一个负数区域。DND 处理器 360 通过执行一个分割操作，然后有选择地对于该差分数据执行一个上分割操作或一个下分割操作，来减小关键字数据的差分数据的范围。熵编码器 370 使用函数 SignedAAC 或 UnsignedAAC 编码差分数据，由此在每个比特平面上编码该差分数据。

下面，参照图 4A 和 4B 描述关键字数据编码器 300 的操作。图 4A 和 4B 是根据本发明的用于编码关键字数据的方法的流程图。

当关键字数据被输入到关键字数据编码器 300 时，例如关键字数据的数量和关键字数据位数的信息被输入到标题编码器 500 并且随后被编码。在步骤 S4000 中，线性关键字编码器 310 首先在该输入的关键字数据中寻找一个线性关键字区域，即其中以确定的时间间隔出现关键帧、关键字数据具有相同的差值并且该关键字数据线性地关键字数据的一个区域，并且编码该搜索的线性关键字区域。

例如 3DMax 或 Maya 的著名应用软件使用在具体的区域之间具有预定

的时间间隔的关键字来产生基于关键字帧的动画。在此情况中，有可能使用一个线性关键字数据区域的开始和结束关键字数据以及存在于它们之间的关键帧的数目来容易地编码关键字数据。因此，线性预测对于使用一个内插器在一个确定的区域中对关键字进行编码是非常有用的。

下面方程式被用于线性预测。

$$t(i) = \frac{t_E - t_S}{E - S} + t_S \quad (0 \le i \le E - S, S < E) \quad \cdots(1)$$

其中，t_S表示一个局部线性区域开始之处的关键字的数据，t_E表示一个局部线性区域结束之处的关键字的数据，S表示t_S的一个索引，而E表示t_E的一个索引。在从第S关键字数据到第E关键字数据的一个具体区域中的实际关键字数据之间的误差以及遵循方程式(1)线性预测的关键字数据能够使用下面的方程式计算。

$$e_i = t(i) - t_{i+S} = \frac{t_E - t_S}{E - S} i + t_S - t_{i+S} \quad \cdots(2)$$

如果使用方程式(2)计算的误差当中的最大值不大于一个预定的临界值，则t_i能够被认为是在区域[t_S，t_E]中或在一个确定误差范围之内的共线性的。使用下面方程式(3)确定该最大误差值t_i是否具有具体的区域共线性。

$$E_p = \underset{i=0,\ldots(E-S)}{MAX} |e_i| = \underset{i=0,\ldots(E-S)}{MAX} \left| \frac{t_E - t_S}{E - S} i + t_S - t_{i+S} \right| \quad \cdots(3)$$

如果$E_p \le \frac{1}{2^{nBits+1}}$，$t_i$则与区域[$t_S$，$t_E$]共线性。其中，nBits表示用于编码的比特的数量。

如果线性关键字编码器310寻找该局部线性区域，则该局部线性关键字数据区域的开始和结束关键字数据被输出到一个浮点数转换器315。包括在该线性关键字数据区域中的关键字的数目被输出到标题编码器500并且被编码。使用线性编码有可能显著地降低需编码的数据量。

在稍后描述的浮点数转换器315中，使用浮点数转换执行对于开始关键字数据和结束关键字数据的编码。

浮点数转换器315把二进制表示的关键字数据转换成十进制关键字数据，以便编码该开始关键字数据和结束关键字数据。

计算机以32比特的二进制数字存储浮点数字。如果给出二进制表示的一个浮点数，浮点数转换器315将把该浮点数转换成十进制的一个尾数和

一个指数，此过程由随后方程式表示。

$$\underbrace{mantissa\_binary*2^{exponent\_binary}}_{the\ floating-point\ number\ in\ binary\ system}=\underbrace{mantissa*10^{exponent}}_{the\ floating-point\ number\ in\ decimal\ system}\quad\cdots(4)$$

例如，一个浮点数 12.34 能够通过计算机转换成一个下面所示的二进制数。

$$\frac{0}{1}\frac{10001010111000010100011}{2}\frac{10000010}{3}$$

1:符号

2:二进制尾数

3:二进制指数

该二进制数能够遵循方程式(4)转换成下列所示的十进制数。

$$\frac{0}{1}\frac{1234}{2}\frac{2}{3}$$

1:符号

2:十进制尾数

3:十进制指数

为了把十进制中的尾数和指数包括在一个比特数据流中，必须计算为了表示该尾数和该指数所要求的比特数目。具有在-38 和 38 之间的一个值的指数因此能够连同符号一起使用 7 比特表示。为了表示该尾数而需要的比特的数量取决于数字的位数。尾数的值和为了表示该尾数而需要的比特的数量以下表示出。

表 2

尾数值	尾数的位数	需要的比特数目
0	0	0
1 – 9	1	4
10 – 99	2	7
100 – 999	3	10
1000 – 9999	4	14
10000 – 99999	5	17
100000 – 999999	6	20

1000000 – 9999999	7	24

已经寻找并且使用上述处理而转换的该线性关键字数据区域的开始和结束关键字数据被遵循图 4C 和 4D 所示的编码处理编码，输出到标题编码器 500，并且存储在该比特数据流中。

图 4C 和 4D 示出对输入到浮点数转换器 315 的两个浮点数字进行编码的一个处理。将参考图 4C 和 4D 描述浮点数转换器 205 编码一个浮点数的方法。

浮点数转换器 315 接收原始关键字数据的位数 Kd、开始关键字数据 S 和结束 E，并且在步骤 S4040 中遵循方程式(4)对它们转换。

浮点数转换器 315 首先编码 S。具体地说，浮点数转换器 315 检验 S 的位数是否不同于 Kd。如果位数 S 不同于 Kd，则获得 S 的位数并且在步骤 S4042 中输出到标题编码器 500。浮点数转换器 315 使用函数 Digit()获得 S 的位数。

如果 S 的位数大于 7，则在步骤 4043 中使用一个预定的比特数量把 S 输出到标题编码器 500(在本发明中，按照 IEEE 标准 754 的浮点数方式使用 32 比特)，以便在该比特数据流中能够包括该 S 的位数。

如果该 S 的位数不是 0 并且小于 7，则浮点数转换器 315 在步骤 4044 中把 S 的符号输出到标题编码器。使用表格 2 获得为了编码 S 的尾数的绝对值需要的比特数目。随后，使用在步骤 4045 中用表格 2 获得的比特数目，把 S 的尾数的绝对值输出到标题编码器 500。在步骤 S4046 中，浮点数转换器 315 计算 S 的指数，输出 S 的符号到标题编码器 500，并且输出该指数到标题编码器 500，作为例如 6 比特的一个预定的数量。这种关键字数据转换使得有可能显著地降低包括在比特数据流中比特的数量。

如果 S 的位数是 0，则对于该开始关键字数据的编码结束，并且该方法转到对该结束关键字数据 E 进行转换的步骤，因为当该 S 的位数是 0 时，该对应浮点数也是 0，不需要进行编码。

在对开始关键字数据 S 进行转换和编码之后，浮点数转换器 315 转换结束关键字数据 E。E 的转换几乎与 S 的转换相同。具体地说，在步骤 S4047 中验证 E 的指数是否与 S 的指数相同。如果 E 的指数与 S 的指数相同，则只有表示 E 的指数与 S 的指数是相同的一个标记位被输出到标题编码器

500。如果 E 的指数与 S 的指数不相同，则在步骤 S4048 中以与 S 的指数输出到标题编码器 500 的同样方式把 E 的指数以及该标记位输出到标题编码器 500。

在不属于该线性关键字区域的输入关键字数据当中的关键字数据被输入到量化器 320 中，然后根据一个预定的量化比特大小 nKeyQBit 被量化。

但是，在用一个解码器解码该量化的关键字数据的情况下，由于在原始关键字数据和该量化关键字数据之间的误差的原因，不可能的完全地恢复原始关键字数据。因此，本发明的量化器 320 获得在输入关键字数据当中的最大值和最小值，并且使用该最大和最小值量化该输入关键字数据。此外本发明的量化器 320 包括一个量化误差最小化器 325，使得在原始关键字数据和其量化的关键字数据之间的误差能够使用在输入关键字数据当中的该最大和最小值被最小化。

在步骤 S4100，量化误差最小化器 325 预先使用控制量化范围的一种方法量化或反向量化该输入的关键字数据，使得能够最小化该量化误差。

具体地说，如果用于量化的混合最大值是由 Max 表示，被控制用于量化的最小值由 Min 表示，输入值由 X_i 表示，用于量化的比特数目由 nQuantBit，则使用下面方程式获得量化输入值 $\tilde{X}_i$、反向量化值 $\hat{X}_i$ 和误差 e_i。

$$\tilde{X}_i = floor(\frac{X_i - Min}{Max - Min} * (2^{nQuantBit} - 1) + 0.5) \quad \cdots(5)$$

$$\hat{X}_i = \frac{\tilde{X}_i * (Max - Min)}{2^{nQuantBit} - 1} + Min$$

$$e_i = X_i - \hat{X}_i$$

有两种减小误差取和 Σe_i 的方法。一种降低误差的取和的方法是通过连续控制器 Min 直到错误的取和被最小化。另外一种方法如下。

首先假定 $X_i = (i + n)\Delta x + \varepsilon_i$，其中 X_i 表示一个输入关键字数据序列，Δx 表示输入数据的一个基本步骤规模，n 是一个任意的整数，而 ε_i 表示零平均随机噪声。

随后，当 $d_i \equiv X_i - X_{i-1} = \Delta x + (\varepsilon_i - \varepsilon_{i-1})$ 时，$\Delta'x = E[d_i]$ 并且 $Min = Max - \Delta'x * (2^{nQuantBit} - 1)$.

有可能最小化一个量化误差的 Min 同 Max 被输入到量化器 320，并且被用于关键字数据的量化。

量化器 320 接收最大值 Max 和能够最小化量化误差和的最小值 Min，并且在步骤 S4200 中遵循方程式(6)量化关键字数据 $fKey_i$。

$$nQKey_i = floor(\frac{fKey_i - fKeyMin}{fKeyMax - fKeyMin}(2^{nKeyQBit} - 1) + 0.5) \quad \cdots(6)$$

其中，i 表示量化后关键字数据的一个索引，$nQKey_i$ 表示量化关键字数据的整数的一个数组，$fKey_i$ 表示量化关键字数据的浮点数字数组，fKeyMax 表示从量化误差最小化器 325 输入的最大值，fKeyMin 表示从量化误差最小化器 325 输入的最小值，nKeyQBit 表示量化比特的大小。在方程式(6)中，函数 floor(v)是一个输出不大于一个确定的浮点值 v 的最大整数的函数。

本发明的量化器 320 可以不使用降低一个量化误差的算法，这时仅使用输入关键字数据中的最大和最小值 fKeyMax 和 fKeyMin 进行量化。

参照图 4E 更详细地描述本发明的量化操作。

量化器 320 在步骤 S4210 中接收关键字数据，并且在步骤 4220 中检测是否从量化误差最小化器 325 输入了最大和最小值 MAX 和 MIN。

如果输入了 MAX 和 MIN，量化器 320 在步骤 4230 中分别把用于量化的最大和最小值 fKeyMax 和 fKeyMin 设置为 MAX 和 MIN，并且把这新的最大和最小值 fKeyMax 和 fKeyMin 输出到浮点数转换器 315。该最大和最小值 fKeyMax 和 fKeyMin 经过上述浮点数转换处理而被转换并且编码，并且被输出到标题编码器 500，使得它们能够被包括在进行解码中使用的一个关键字标题中。

如果没有从该量化误差最小化器 325 输入的值，则在步骤 S4240 中，量化器 320 分别把第一关键字数据 $fKey_0$ 和最终关键字数据 $fKey_{N-1}$ 设置为最小值 fKeyMin 和最大值 fKeyMax。

随后，量化器 320 在步骤 S4250 中检验最大值 fKeyMax 是否小于 1 但大于 0，以及最小值 fKeyMin 是否大于 0。如果最大值 fKeyMax 不小于 1 或不大于 0，则该最大和最小值 fKeyMax 和 fKeyMin 被输出到浮点数转换器 315，并且经过上述浮点数转换过程被转换和编码。随后，在步骤 S4260 中，已经转换和编码的最大和最小值 fKeyMax 和 fKeyMin 被包括在关键字标题中，使得它们可被用于进行解码。

另一方面，如果最大值 fKeyMax 小于 1 并且最小值 fKeyMin 大于 0，则在步骤 S4270 中检验的一个标志，该标志表示最大和最小值 fKeyMax 和

fKeyMin 是否将被包括在用于在进行解码使用的关键字标题中。如果该标志被设置而使得最大和最小值 fKeyMax 和 fKeyMin 能够被包括在该关键字标题中，则执行步骤 S4260，以使最大和最小值 fKeyMax 和 fKeyMin 被输出到标题编码器 500。如果没有设置该标志，则量化器 320 将不允许该最大和最小值 fKeyMax 和 fKeyMin 包含在该关键字标题中。

在该最大和最小值 fKeyMax 和 fKeyMin 不被包括在该关键字标题中的一个情况中，关键字数据编码器 300 和一个关键字数据解码器被分别来执行编码和解码处理，分别以 1 和 0 建立该最大和最小值 fKeyMax 和 fKeyMin。在此情况中，量化器 320 在步骤 S4280 中分别以 1 和 0 设置该最大和最小值 fKeyMax 和 fKeyMin。对于该关键字数据解码器来说，该最大和最小值 fKeyMax 和 fKeyMin 是已经知道的，使得 fKeyMax 和 fKeyMin 不需要包括在该关键字标题中。

通过将上述处理已经设置的最大和最小值 fKeyMax 和 fKeyMin 替代到方程式(6)中，量化器 320 在步骤 S4290 中量化该输入的关键字数据，并且把该量化的关键字数据输出到一个 DPCM 处理器 330。

DPCM 处理器 330 接收该量化的关键字数据，并且对于该量化的关键字数据执行预定的次数的 DPCM。随后，DPCM 处理器 330 输出 DPCM 的阶数以及每一个周期中获得的 DPCM 的帧内关键字数据到标题编码器 500，通过该 DPCM 的阶数获得分散程度中的最小值。在步骤 S4300 中，DPCM 处理器 330 把由 DPCM 产生的差分数据输出到移位器 340。

参考图 4F，在步骤 S4310 中，DPCM 处理器 330 对于输入的关键字数据执行预定次数的 DPCM，并且按照 DPCM 的阶数存储 DPCM 的周期数目。在本发明的一个优选实施例中，DPCM 可以被执行三次。

然后，在步骤 S4320 中，DPCM 处理器 330 计算 DPCM 的每一周期的结果的分散程度。这里，分散程度可以由分散角、标准偏差或四分位偏移表示，而在本发明的一个优选实施例中，可以使用四分位偏移。

随后，DPCM 处理器 330 选择 DPCM 的一个周期，通过该 DPCM 周期能够获得在该分散程度中的最小值，并且把该选择的 DPCM 阶数的结果输出到移位器 340。在步骤 S4330 中，DPCM 的选择周期、DPCM 的每一周期的帧内关键字数据以及用于 DPCM 所需要的信息的其它部分被输出到标题编码器 500。然而在本发明的一个优选实施例中，如果关键字的数目小于 5，

则 DPCM 仅执行一次。例如，DPCM 的第一周期的执行遵循方程式(7)。

$$\Delta_i = nQKey_{i+1} - nQKey_i \quad \cdots(7)$$

其中，i 表示量化后关键字数据的索引，$nQKey_i$ 表示整数数组，而Δ_i表示差分数据。

在步骤 S4340 中，DPCM 处理器 330 计算为了对 DPCM 选择的周期的结果进行编码所要求的比特数目以及已经由 DPCM 在一个预定的存储器中产生的该关键字数据的差分数据(nQStep_DPCM)。为了进行编码所要求的比特数量的计算也可以稍后在选择将要被编码的关键字数据的随后步骤中执行，这对本专业技术人员来说是显然的。

移位器 340 从来自 DPCM 处理器 330 的输入差分数据当中选择具有最高频率的一个差分数据(在下文称作一个模式)。随后，在步骤 S4400 中，移位器 340 从所有的差分数据减去该模式，以使大部分将要被编码的数据围绕 0 排列并且用于编码所需要的比特数目能够被减少。

这种移位操作通过从量化关键字数据减去模式 nKeyShift 进行，由下面方程式表示。

$$shift(nQKey_i) = nQKey_i - nKeyShift \quad \cdots(8)$$

其中，i 表示量化后关键字数据的一个索引，$nQKey_i$ 表示整数数组，而 nKeyShift 表示一个模式值。作为移位操作的结果，具有最高频率的差分数据成为 0，使得用于进行编码所需要的比特数目能够被显著地降低。

已经通过该移位操作的关键字数据被输出到折叠处理器 350 和 DND 处理器 360，并且模式值 nKeyShift 被输出到标题编码器 500，以使其被包括在该关键字标题中。

在步骤 S4500 中，折叠处理器 350 对于移位器 340 的输出执行一个折叠操作，并且该折叠操作的结果输出到 DND 处理器 360。

该折叠操作被用于降低差分数据的范围，通过把它们集中在正或负数区域中，该差分数据广泛地散布在一个正数区域和一个负数区域之上。在本实施例中，折叠操作遵循方程式(9)执行，以便把差分数据集中在该正数区域中。

$$fold(nQKey_i) = 2 \cdot nQKey_i \quad (if\ nQKey_i \geq 0) \quad \cdots(9)$$
$$= 2|nQKey_i| - 1 \quad (if\ nQKey_i < 0)$$

其中，i 表示量化后关键字数据的索引，而 $nQKey_i$ 表示整数数组。作为折叠操作的结果，正差分数据被转换成偶数，而负差分数据被转换成奇数。

折叠处理器 350 计算为了编码已经通过该折叠操作的差分数据所需的比特数量并且将其存储在预定的存储器 nQStep_fold 中。在此步骤中，用于编码的所需比特数量的计算可以稍后在选择将要被熵编码的差分数据的一个随后步骤中执行，像步骤 S9300 那样，这对本专业人员来说是显然的。由折叠处理器 350 中的折叠操作产生的数据被输出到 DND 处理器 360。

为了提高熵编码的效率，在步骤 S4600 中 DND 处理器 360 对于该关键字数据执行预定次数的 DND 操作，因此降低差分数据的范围。

参考图 3B，DND 处理器 360 包括：DND 操作器 362，对于差分数据执行 DND 操作；第一差分数据选择器 364，根据用于进行编码的比特数目选择将要被熵编码的差分数据；上移操作器 366：对于已经通过 DND 操作的差分数据执行一个上移操作；和第二差分数据选择器 368，在仅通过 DND 操作的差分数据和已经通过上移位操作的差分数据之间选择具有较低分散程度的差分数据，并且把所选择的差分数据输出到熵编码器 370。

在下面段落中将描述 DND 操作器 362 中执行的 DND 操作。

当该差分数据已经通过在该折叠处理器 362 中的折叠操作被输入到 DND 操作器 362 中时，它们被分为两组，具有比另一组差分数据更高范围的一组差分数据通过一个分割函数被移到该正数区域。该分割函数由下面方程式定义。

$$divide(nQKey_j, nKeyMax) \quad \cdots(10)$$
$$= nQKey_j - (nKeyMax + 1) \quad (if\ nQKey_j > \frac{nKeyMax}{2})$$
$$= nQKey_j \quad (if\ nQKey_j \leq \frac{nKeyMax}{2})$$

其中，j 表示输入差分数据的一个索引，$nQKey_j$ 表示整数的一个数组，而 nKeyMax 表示在已经通过该折叠操作的差分数据当中的一个最大值。特别地，在大部分差分数据沿着考虑所有的差分数据的整个区域的边界密集填充情况下，则有可能使用该分割操作显著地降低所有的差分数据的整个

区域。

在该分割操作之后，计算该分散程度，其中用于进行编码而需要的比特的大小被用作分散的度量，因此能够选择用于进行编码的比特大小的最小值。

DND 操作之后，进一步执行不同种类的 DND 操作，即上分割操作或下分割操作。根据差分数据的一个正范围的大小和差分数据的一个负范围的大小确定是否将进一步执行一个上分割操作或一个下分割操作。

如果具有正值的该差分数据范围大于具有负值的该差分数据的范围，则执行通过下面方程式定义的一个下分割操作。

$$
\begin{aligned}
&divide-down(nQKey_j, nKeyMax) && \cdots(11)\\
&= -2(nKeyMax - nQKey_j + 1) + 1 && (if\ nQKey_j > \frac{nKeyMax}{2})\\
&= nQKey_j && (if\ 0 \le nQKey_j \le \frac{nKeyMax}{2})\\
&= 2 \cdot nQKey_j && (if\ nQKey_j < 0)
\end{aligned}
$$

另一方面，如果具有正值的差分数据的范围大于具有负值的差分数据的范围，则执行由如下方程式定义的一个上分割操作。

$$
\begin{aligned}
&divide-up(nQKey_j, nKeyMin) && \cdots(12)\\
&= nQKey_j && (nQKey_j \ge 0)\\
&= 2 \cdot nQKey_j && (\frac{nKeyMin}{2} \le nQKey_j \le 0)\\
&= 2(nKeyMin - nQKey_j - 1) + 1 && (nQKey_j < \frac{nKeyMin}{2})
\end{aligned}
$$

在方程式(11)和(12)中，j 表示量化后关键字数据的一个索引，$nQKey_j$ 表示整数的一个数组，nKeyMax 表示 $nQKey_j$ 的一个最大值，而 nKeyMin 表示 $nQKey_j$ 的最小值。

在下面将参照图 4G 描述 DND 操作器 362 的操作。

当输入的关键字数据的差分数据是来自折叠处理器 350 的输入，则 DND 操作器 362 在步骤 S4610 中获得在输入差分数据当中的最大值 nKeyMax 和最小值 nKeyMin。随后，在步骤 S4620 中 DND 操作器 362 比较该 nKeyMax 的绝对值与 nKeyMin 的绝对值。如果 nKeyMax 不小于 nKeyMin

的绝对值，则在步骤 S4622 中 DND 操作器 362 把 nKeyMax 设置为在当前 DND 操作周期中的一个最大值。

DND 操作器 362 检测该 DND 操作的阶数是否为 1，换句话说，在步骤 S4624 中检测 DND 操作的阶数是否为 1，并且如果其是 1，则在步骤 S4630 中 DND 操作器 362 执行对于该输入差分数据的一个分割操作，替代方程式(10)中最大值 nKeyMax。

然后，DND 操作器 362 在步骤 S4640 中使用函数 getQBit()测量为了编码已经使用该分割操作被减小的该差分数据范围所需要的比特的大小。如果在步骤 S4650 中该 DND 操作的阶数出现是 1，则用于进行编码所需要的比特大小被存为表示为了进行编码的比特的最小大小的一个值 nQBitDND，并且 DND 操作的阶数在步骤 S4655 中被增加 1。

随后，该 DND 处理器 362 再一次执行步骤 S4610 至 S4622。如果在步骤 S4624 中的 DND 操作的阶数不是 1，则该 DND 操作器 252 在步骤 S4634 中执行一个下分割操作，代替方程式(11)中的最大值 nKeyMax。在步骤 S4640 中，DND 操作器 362 计算用于编码已经通过该下分割操作的该差分数据所需要的比特的数量。如果该数目小于在先前 DND 操作周期中存储的该最小值 nQBitDND，则在步骤 S4658 的 DND 操作之后替代用于编码所需的比特的最小量。

如果在步骤 S4620 中该最小值 nKeyMin 的绝对值出现为大于最大值 nKeyMax，则在步骤 S4623 中把 DND 操作的当前周期中的最大值更新为一个最小值，然后在步骤 S4638 中执行上分割操作，替代方程式 12 中的最小值 nKeyMin。然后，DND 操作器 362 在步骤 S4640 中计算用于编码已经通过该上分割操作的差分数据的比特的数量。如果该计算的结果是小于已经在步骤 S4652 中的 DND 操作的预先周期中存储的 nQBitDND，则在步骤 S4658 中替代用于 DND 操作之后进行编码所需的该最小比特数 nQBitDND。

DND 处理器 362 执行预定次数的 DND 操作，并且该 DND 操作的执行的数量可以改变。例如在本实施例中，该 DND 操作被执行 7 次。DND 操作器 362 输出 nQBitDND 和对应于 nQBitDND 的差分数据到第一差分数据选择器 364。DND 操作器 362 把对应已经产生的差分数据的 DND 的阶数输出到标题编码器 500，并且使得它们被包括在该比特数据流中。

第一差分数据选择器 364 接收已经通过该移位操作的差分数据、已经

通过折叠操作的差分数据和已经通过DND操作的差分数据，并且在这三个差分数据当中确定将被熵编码的差分数据。

参考图3A，第一差分数据选择器364选择该DPCM的结果，并且如果在该DND操作以后的用于进行编码所需的最小比特数目nQBitDND不小于在步骤S4700中的该DPCM操作之后的用于进行编码的比特大小nQStep-DPCM，在步骤S4710中对于该DPCM的结果执行一个移位操作。随后，在步骤S4710中，第一差分数据选择器364把该移位操作的结果输出到熵编码器370，并且使得它们被熵编码。在此情况下，DND操作的阶数被设置在-1，被输出到标题编码器500，并且被包括在该关键字标题中。

但是，如果在步骤S3720中结果变为该nQBitDND小于nQStep-DPCM并且不小于折叠操作之后的用于进行编码的比特的大小，则第一差分数据选择器364把已经通过该折叠操作的该差分数据输出到熵编码器370，并且使得它们在步骤S4730中被熵编码，其中DND操作的阶数被设置在0，输出到标题编码器500，并且因此被包括在该关键字标题中。

如果在DND操作之后用于编码该差分数据的比特数目是最小的，则第一差分数据选择器364把已经通过该DND操作的差分数据输出到该上移位操作器366，然后该上移位操作器366在步骤S4740中计算从第一差分数据选择器364输入的差分数据的第一分散程度。随后，该上移位操作器366在步骤S4800中对于已经通过该DND操作的该差分数据执行由下面方程式定义的一个上移位操作，并且在步骤S4810中计算该上移位结果的一个第二分散程度。

$$shift\text{-}up(nQKey_j, nKeyMax) \quad \cdots(13)$$
$$= nQKey_j \qquad (if\ nQKey_j \geq 0)$$
$$= nKeyMax - nQKey_j \quad (if\ nQKey_j < 0)$$

其中，j表示量化后关键字数据的差分数据的一个索引，$nQKey_j$表示整数的一个数组，而nKeyMax表示在差分数据当中的一个最大值。

当已经通过DND操作的差分数据和已经通过上移位操作的差分数据被输入时，第二差分数据选择器368在步骤S4900中把该第一分散程度与该第二分散程度相比较。如果该第二分散程度小于该第一分散度，则在步骤S4910中第二差分数据选择器368把已经通过该上移位操作的差分数据

输出到熵编码器 370 并且使得它们被熵编码。该第二差分数据选择器 468 把使用在该 DND 操作的最大和最小值 nKeyMax 和 nKeyMin、以及使用在上移位操作中的最大值 nKeyMax 输出到标题编码器 500，并且使得它们被包括在关键字标题中。

但是，如果该第一分散程度小于该第二分散程度，则第二差分数据选择器 368 把已经通过该 DND 操作的差分数据输出到熵编码器 370，并且使得它们在步骤 S4920 中被熵编码。然后，第二差分数据选择器 368 仅把使用在该 DND 操作中的该最大和最小值 nKeyMax 和 nKeyMin 输出到标题编码器 500。在本发明的一个优选实施例中，标准偏差可以被用作该第一和第二分散程度的一个度量。

根据差分数据的特性，熵编码器 370 对于差分数据执行两个不同的操作。例如，已经通过 DPCM 操作和一个移位操作的差分数据，以及只通过一个分割操作的差分数据既有正值又有负值，并且因此要求执行每一差分数据的符号的编码处理以及该差分数据本身的编码处理。另一方面，由于已经通过折叠操作的差分数据仅有正值，所以执行仅编码该差分数据的处理。

在本发明的一个优选实施例中，函数 encodeSignedAAC 被用于编码该差分数据和其符号，而函数 encodeUnsignedAAC 被用于仅编码差分数据。

图 5 是表示一个函数 encodeSignedAAC 的一个实例的示意图。参考图 5，当一个输入值是 74 而用于编码该输入值的比特数目是 8 时，其符号是 0，并且其与二进制数字 1001010 相同。符号以及整个比特平面以下列方式编码：

第一步骤：一个二进制数以从其最高有效比特(MSB)到其最低有效比特(LSB)的次序在每一比特平面上编码一个二进制数；

第二步骤：检验当前正被编码的比特是否为 0；

第三步骤：如果当前被编码的比特不是 0，则该二进制数的符号被随后编码；和

第四步骤：该二进制数的其余比特被编码。

使用关于该值的一个上下文，函数 encodeUnsignedAAC 把没有符号的值编码为一个自适应算法编码比特数据流。除了存在一个符号上下文之外，此函数与函数 encodeSignedAAC 几乎相同。

图 6A 至 6J 是表示关键字数据的曲线示意图，其关键字数据已经经过了按照本发明一个优选实施例的操作。在图 6A 至 6J 中，该 X 轴表示每一个关键字数据的索引，而 Y 轴表示该关键字数据的值。

图 6A 是表示输入到该本发明的编码器的原始关键字数据的一个曲线图。图 6A 所示的关键字数据被输出到量化器 320，然后以九个量化比特量化，以便获得图 6B 所示的量化的关键字数据。如果对于图 6B 所示的量化关键字数据执行 DPCM，则获得图 6C 所示的差分数据。

随后，使用大约 7 的一个模式值移位该量化关键字数据的差分数据，以便获得图 6D 所示的差分数据。随后，如果对于该移位的差分数据执行一个折叠操作，则可以获得如图 6E 所示的仅有正值的数据。

图 6F 至 6H 示出对于图 6E 示出的折叠数据执行一个 DND 操作的结果。具体地说，对于图 6F 示出的折叠数据执行一个分割操作的结果。如图 6F 所示，正关键字数据值范围从 0 到 28，而负关键字数据值范围从 29 到 0，其意味着负关键字数据值的范围大于正关键字数据值的范围。因此，要求对于图 6F 中所示数据执行一个上分割操作，而该上分割操作的结果在图 6G 中示出。

作为该上分割操作的结果，负关键字数据值的范围被大大降低，使得其比正关键字数据值的范围小得多。在 DND 操作的一个随后周期中，对于该上分割操作的结果执行一个下分割操作。图 6H 是表示对于图 6G 所示差分数据执行一个下分割操作的结果。对于图 6H 所示的关键字数据执行一个上移位操作的结果在图 6I 中示出。

如图 6A 至 6G 所示，关键字数据和差分数据的范围逐步减小。然而，如图 6H 和 6I 所示，上移位操作以后的差分数据的范围比该操作之前的范围增加得更大，这表明如图 6H 所示该已经通过下分割操作的差分数据是最后唯一被编码的差分数据，如图 6J 所示。

下面段落将描述在标题编码器 500 编码并且存储在关键字标题中的信息。

当输入将要被编码的关键字数据时，标题编码器 500 编码将要被编码的关键字数据的数字数目和关键字数目。随后，标题编码器 500 从线性关键字编码器 310 接收有关于输入的关键字数据中是否存在已经通过该线性关键字编码的一个线性关键字区域以及在该线性关键字数据区域中的关键

字数据的数量的信息，并且从浮点数转换器 315 接收该已经通过浮点数转换的线性关键字数据区域的开始和结束关键字数据。

在浮点数转换器 315 接收能够实现一个最小量化误差的最大和最小值并且把它们转换成浮点数字的情况下，该转换的最大和最小值被从浮点数转换器315输入到标题编码器500，以使它们能被再一次用于进行反向量化。此外，量化比特的大小也被输入到标题编码器 500 并且被包括在该关键字标题中。

标题编码器500从DPCM处理器330接收DPCM的阶数以及在DPCM的每一周期中的内关键字数据，并且从移位器 340 接收已经用于一个移位操作的一个模式值。此外，标题编码器500从DND处理器360接收有关是否已经执行一个上移动操作的信息，能够最小化差分数据的分散程度的DND的阶数的信息以及在DND操作的每一周期中的最大和最小值信息。

最终，标题编码器 500 从熵编码器 370 接收用于进行编码的比特数目并且将其编码作为一个关键字标题。

下面，参考图 7A 至 11C 更详细地描述根据本发明一个优选实施例的用于对坐标内插符的关键值数据进行编码的一个装置和方法。

图 7A 是根据本发明优选实施例的关键值数据编码器的框图，而图 7B 是根据本发明优选实施例的用于对坐标内插符的关键值数据进行编码的一种方法的流程图。

参考图 7A，关键值数据编码器 700 包括量化器 710，使用预定的量化比特量化输入其中的坐标内插符的关键值数据每一顶点的成份的数据；DPCM 处理器 720，对于每一顶点的量化成分数据执行一个预定的 DPCM 操作；字典编码器 750，将差分数据转换成符号和指示该符号的位置的索引；以及熵编码器 760，熵编码输入其中的符号和差分数据的索引。

在下面的段落中，参照图 7B 描述用于编码坐标内插符的关键值数据的方法。参考图 7B，在步骤 S9000 中以 NXM 矩阵的形式把坐标内插符的关键值数据输入到量化器 710。一个输入的坐标内插符的关键值数据的实例在下列表中示出。

表 3

	1	2	… j	M
1	x(1,1), y(1,1), z(1,1)	x(1,2), y(1,2), z(1,2)		x(1,M), y(1,M), z(1,M)

2	x(2,1), y(2,1), z(2,1)	x(2,2), y(2,2), z(2,2)		x(2,M), y(2,M), z(2,M)
… i			x(i,j), y(i,j), z(i,j)	
N	x(N,1),y(N,1), z(N,1)	x(N,2), y(N,2), z(N,2)		x(N,M),y(N,M), z(N,M)

在表 3 中，N 表示关键字数据(关键帧)的数目，M 表示每一关键帧中的顶点的数量。

根据本发明的关键值数据编码器 700 以两种不同模式操作来编码坐标内插符的关键值数据。模式之一是顶点模式，而另外一个模式是转置模式。表 3 中，示出以顶点模式在量化器 710 中量化的关键值数据的结构。在量化表 3 所示的输入关键值数据之前，关键值数据编码器 700 把输入的关键值数据转置成一个 MHN 矩阵。在解码关键值数据过程中反向量化该转置矩阵，并且把解码的关键值数据转换成 NHM 矩阵，使得能够恢复与输入关键值数据一样的关键值数据。

参考图 7B，在步骤 S9100 中，量化器 710 检查从外部输入的关键值数据的编码方式是否为转置模式。如果输入的关键值数据的编码模式是一个转置模式，则在步骤 S9200 中，输入关键值数据的 NHM 矩阵被转置成一个 MHN 矩阵。

然后，在步骤 S9300 中，量化器 710 以预定的量化比特量化输入其中的该关键值数据矩阵中的每一个成分的数据，并且在步骤 S9300 把每一个成分的量化的关键值数据输出到 DPCM 处理器 720。在同一个步骤中，量化器 710 把在每一个成份的输入的关键值数据当中的最小值和在该成份的数据范围当中的最大范围转换成十进制数字，并且把该十进制数字输出到标题编码器 500。

在步骤 S9400 中，DPCM 处理器 720 对于输入其中的量化的关键值数据执行一个时间 DPCM 操作、一个空间 DPCM 操作和一个时空 DPCM 操作，对于该三个不同的 DPCM 操作的结果，即对于从该三个 DPCM 操作获得的每一差分数据执行一个循环量化操作，并且把在它们当中的具有最低熵值的差分数据输出到字典编码器 750。

在步骤 S9600 中，字典编码器 750 产生和输出对应于从 DPCM 处理器 720 输入的差分数据的字典符号 $S_{i,j}$ 和位置索引 $I_{i,j}$。具体地说，字典编码器

750 产生的该字典符号和该位置索引，表明已经对于该输入差分数据执行的 DPCM 操作的模式，把该输入的差分数据转换成对应于该输入差分数据值的符号或一个符号标志，和表示该符号的位置的位置索引，并且把该符号和位置索引输出到熵编码器 760。

在步骤 S9800 中，熵编码器 760 通过对从该字典编码器 750 输入的该符号和位置索引进行熵编码而产生一个比特数据流。

下面，参照图 8A 至 11C 更详细地描述步骤 S9300 至 S9800。

参考图 9A，在步骤 S9320 中，量化器 710 选择在每一个成份数据当中的最大值和最小值。

在步骤 S9340 中，量化器 710 使用选择的该最大和最小值计算该成分的数据范围，并且确定在该成份的数据范围当中的一个最大范围。

在步骤 S9360 中，量化器 710 使用下面方程式所示的每一成份数据当中的最小值和各成份的整个数据范围当中的最大范围量化每一成分的关键值数据。

$$\tilde{V}_{i,j,x} = floor(\frac{V_{i,j,x} - fMin\_X}{fMax}(2^{nKVQBit} - 1) + 0.5) \quad \cdots(14)$$

$$\tilde{V}_{i,j,y} = floor(\frac{V_{i,j,y} - fMin\_Y}{fMax}(2^{nKVQBit} - 1) + 0.5)$$

$$\tilde{V}_{i,j,z} = floor(\frac{V_{i,j,z} - fMin\_Z}{fMax}(2^{nKVQBit} - 1) + 0.5)$$

方程式(14)中，i 表示关键数据，j 表示一个顶点，而 nKVQBit 表示量化比特大小。此外，fMin_X、fMin_Y、fMin_Z 表示在每一个成份的数据中的最小值，而 fMax 表示在该成份数据范围中的最大范围。

量化器 710 把每一成分的量化关键值数据输出到 DPCM 处理器 720，并遵循方程式(4)和表 2，把 fMin_X、fMin_Y、fMin_Z 和 fMax 变换成十进制数字，并且把该十进制数字输出到标题编码器 500。

下面将参照图 8A 和 9B 更详细地描述该 DPCM 处理器 720 的结构和操作。

图 8A 是根据本发明的 DPCM 处理器 720 的一个框图。参考图 8A，DPCM 处理器 720 包括一个 DPCM 操作器 730，对于从量化器 710 输入的每一成份的数据执行一个时间 DPCM 操作、空间 DPCM 操作和一个时空

DPCM 操作，一个循环量化器 740，降低从该 DPCM 操作器 730 输入的差分数据的范围，以及一个 DPCM 模式选择器 745，选择从循环量化器 740 输入的差分数据之一。DPCM 操作器 730 包括一个时间 DPCM 操作器 731，对于每一个成份的量化数据执行一个时间 DPCM 操作，一个空间 DPCM 操作器 733，对于每一个成份的量化数据执行一个空间 DPCM 操作，以及一个时空 DPCM 操作器 735，对于每一个成份的量化数据执行一个时空 DPCM 操作。

图 9B 是根据本发明优选实施例的一个 DPCM 操作的流程图。参考图 9B，在步骤 S9420 中，每一个成分的量化数据从量化器 710 输入到时间 DPCM 操作器 731、空间 DPCM 操作器 733、和时空 DPCM 操作器 735，然后在分别的操作器 731、733 以及 735 中对于每一个成分的量化数据执行时间 DPCM 操作、空间 DPCM 操作以及时空 DPCM 操作。

时间 DPCM 操作器 731 计算在一个当前关键帧中的顶点的成分数据和在一个先前关键帧中的顶点的成分数据之间的差值。一个时间 DPCM 操作由下面方程式表示。

$$D_{i,j} = \widetilde{V}_{i,j} - \widetilde{V}_{i-1,j} \quad \cdots(15)$$

方程式(15)中，i 表示关键字数据，j 表示一个顶点的位置索引。

空间 DPCM 操作器 733 计算在同一个关键帧中的顶点之间的差值。具体地说，空间 DPCM 操作器 733 使用下面方程式计算前个顶点的熵，在当前顶点受到空间 DPCM 操作之前已经对于该前个顶点执行了空间 DPCM 操作。

$$Entropy(P) = -\sum_{i=0}^{N-1} P_i \log_2 P_i \quad \cdots(16)$$

在方程式(16)中，P_i 表示某符号在一个顶点产生的概率，并且等于 F_i/N，其中 F_i 表示该符号被产生了多少次，而 N 表示关键字数据的数量。

空间 DPCM 操作器 733 把在该顶点中的具有该最低熵的一个顶点确定为一个参考顶点，并且计算在当前受到空间 DPCM 操作的一个顶点的数据和该参考顶点的数据之间差分数据。一个空间操作由下面方程式表示。

$$D_{i,j} = \widetilde{V}_{i,j} - \widetilde{V}_{i,Ref} \quad \cdots(17)$$

该时空 DPCM 操作器 735 对于该当前关键帧的顶点执行一个空间 DPCM 操作，使用在该先前关键帧的顶点中的一个对应于该当前关键帧的参考顶点、并作为参考顶点的顶点，对于该先前关键帧的顶点执行空间 DPCM 操作，并且计算在对应于该当前关键帧的顶点的差分数据和对应于该先前关键帧的顶点的差分数据之间的差分数据。换句话说，该时空 DPCM 操作器 735 对于该空间 DPCM 操作的结果执行一个时间 DPCM 操作。该时空 DPCM 操作由下面方程式表示。

$$D_{i,j} = \widetilde{V}_{i,j} - \{\widetilde{V}_{i-1,j} + (\widetilde{V}_{i,Ref} - \widetilde{V}_{i-1,Ref})\} \quad \cdots(18)$$

在该空间 DPCM 操作和该时空 DPCM 操作过程中，如果 $\widetilde{V}_{i,Ref}$ 或 $\{\widetilde{V}_{i-1,j} + (\widetilde{V}_{i,Ref} - \widetilde{V}_{i-1,Ref})\}$ 小于在每一成分的量化数据中的最小值，则该最小值被用于空间 DPCM 操作和时空 DPCM 操作。另一方面，如果 $\widetilde{V}_{i,Ref}$ 或 $\{\widetilde{V}_{i-1,j} + (\widetilde{V}_{i,Ref} - \widetilde{V}_{i-1,Ref})\}$ 大于在每一成分的量化数据中的最大值，则该最大值被用于该空间 DPCM 操作和该时空 DPCM 操作。

在步骤 S9440 中，DPCM 操作器 730 把计算的差分数据输出到循环量化器 740，而循环量化器 740 对于该时间 DPCM 的差分数据、空间 DPCM 差分数据和时空 DPCM 差分数据执行循环量化操作，并且把该循环量化的结果输出到 DPCM 模式选择器 745。

图 10A 是量化器 710 的一个输出实例的曲线图，而图 10B 是对于图 10A 中所示的量化数据执行 DPCM 操作的结果的曲线图。如图 10B 所示，通过执行对于量化数据的一个 DPCM 操作，将要被编码的数据范围能够增加为其原数据范围的两倍。循环量化的目的是在保持量化值的数据范围的同时执行一个 DPCM 操作。

在本发明中，假设在 DPCM 的差分数据中的一个最大值被循环连接到在 DPCM 的差分数据中的一个最小值而执行该循环量化。如果对于两个连续量化数据执行的线性 DPCM 操作的结果大于在 DPCM 的差分数据中的最大值的一半，则从该线性 DPCM 的结果中减去从该 DPCM 的差分数据的一个最大范围值，以便产生具有较小绝对值的一个差分数据。另一方面，如果该线性 DPCM 的结果小于该最大范围中的一个最小值的一半，则把该最大范围值加到该线性 DPCM 的结果，以便产生具有较小绝对值的一个值。

通过下面的方程式表示循环量化器 740 的操作。

$$CircularQuanti\ zation(X_t): \qquad \cdots(19)$$
$$X'_t = X_t - (nQMax - nQMin + 1) \quad (if\ X_t \geq 0)$$
$$X'_t = X_t + (nQMax - nQMin + 1) \quad (otherwise)$$
$$\tilde{X}_t = \min(|X_t|, |X'_t|)$$

在方程式(19)中，nQMax 表示在 DPCM 的差分数据中的一个最大值，而 nQMin 表示在 DPCM 的差分数据中的一个最小值。图 10C 示出对于图 10B 所示的 DPCM 的差分数据执行循环量化的结果。

循环量化器 740 把该循环量化的差分数据输出到 DPCM 模式选择器 745。

在步骤 S9460 中，DPCM 模式选择器 745 遵循方程式(16)计算从时间 DPCM 操作、空间 DPCM 操作和时空 DPCM 操作获得的每一 DPCM 差分数据的熵。

下面，DPCM 模式选择器 745 在步骤 S9480 中选择在时间 DPCM 操作、空间 DPCM 操作、时空 DPCM 操作的结果中具有最低熵的 DPCM 差分数据作为每一顶点的 DPCM 操作模式，并且把对应于该选择的 DPCM 模式的该 DPCM 差分数据和关于该 DPCM 模式的信息输出到字典编码器 750。

下面，参照图 8B 和 9C 描述字典编码器 750 及其操作。

图 8B 是根据本发明的字典编码器 750 的一个框图。参考图 8B，字典编码器 750 包括一个 DPCM 模式编码器 752，编码已经对于输入其中的每一个顶点的每一成分的数据执行的 DPCM 的模式，一个出现模式编码器 756，产生表示每一顶点的每一成分的差分数据的值的符号和表示该符号的位置的位置索引，一个增量模式编码器 758，产生对应于该符号和表示该符号位置的位置索引的一个符号标志，以及一个表格大小计算器 754，计算用于表示每一顶点的每一成份的差分数据的符号表格以及一个符号标志表格的大小，并且把从 DPCM 模式编码器 752 输入的该差分数据输出到该出现模式编码器 756 或增量模式编码器 758 之一。

字典编码器 750 检测每一顶点的每一成份的差分数据的一个量化选择标志是否为 1，如果是，则执行将在下面描述的后续处理。另一方面，如果某顶点的差分数据的量化选择标志是 0，即意味着该顶点整个关键帧中具有相同的量化值，则该字典编码器 750 将省略字典编码处理，并且把该量化

值 QMin 编码到一个关键字值标题。

图 9C 是根据本发明的一个字典编码处理的流程图。参考图 9C，在步骤 S9620 中，在 DPCM 处理器 720 中已经产生的每一顶点的每一成份的差分数据被输入到 DPCM 模式编码器 752 中，然后 DPCM 模式编码器 752 产生表示已经对于每一顶点的每一成份的数据执行的 DPCM 操作模式的符号，以及指示该符号的位置的位置索引。

图 11A 是说明根据本发明的 DPCM 模式编码器 752 中执行的编码 DPCM 模式的一种方法的示意图。参考图 11A，DPCM 模式编码器 752 预先制备一个表格，其中示出每一顶点的每一成份的 DPCM 模式和其分别的符号，如图表 4 所示。表 4 示出 DPCM 操作的组合以及其对应符号。表 4 中，时间 DPCM 操作、空间 DPCM 操作和时空 DPCM 操作分别表示为 T、S 和 T+S。

表 4

符号	DPCM 模式	符号	DPCM 模式	符号	DPCM 模式
0	(T, T, T)	9	(S, T, T)	18	(T+S, T, T)
1	(T, T, S)	10	(S, T, S)	19	(T+S, T, S)
2	(T, T, T+S)	11	(S, T, T+S)	20	(T+S, T, T+S)
3	(T, S, T)	12	(S, S, T)	21	(T+S, S, T)
4	(T, S, S)	13	(S, S, S)	22	(T+S, S, S)
5	(T, S, T+S)	14	(S, S, T+S)	23	(T+S, S, T+S)
6	(T, T+S, T)	15	(S, T+S, T)	24	(T+S, T+S, T)
7	(T, T+S, S)	16	(S, T+S, S)	25	(T+S, T+S, S)
8	(T, T+S, T+S)	17	(S, T+S, T+S)	26	(T+S, T+S, T+S)

每一顶点包括三个成分 x、y 和 z，并且相应的 DPCM 操作的组合的数量是 27。

如图 11A 所示，根据该差分数据已经通过的 DPCM 操作，每一个顶点的差分数据对应于表 4 示出的符号之一。DPCM 模式编码器 752 使得该顶点的 DPCM 模式对应于表 4 中示出的分别的符号，并且设置标志，指示该符号存在于分别的顶点差分数据中。

DPCM 模式编码器 752 把对应于该顶点的 DPCM 模式的符号排列在一

个列中，并且以从用于具有较小幅值的一个符号的位置索引到用于具有最大幅值的一个符号的位置索引的次序产生用于该符号的位置索引。

如图 11A 所示，对应于该顶点差分数据的 DPCM 模式的符号的一个数组是(4，1，5，1，4，5)。在该符号当中，1 是最小的符号，并且对应于(T，T，S)。DPCM 模式编码器 752 产生用于该符号 1 的一个位置索引，使得符号的数组中出现 1 的位置由 1 表示。因此，该位置索引是(0，1，0，1，0，0)。

随后，DPCM 模式编码器 752 产生用于次最小符号 4 的位置索引，其对应于 DPCM 模式(T，S，S)，使得其中 4 所在的位置由 1 表示。在用于符号 4 的位置索引的产生中，不计算符号 1 的位置。因此，针对该符号 4 的位置索引是(1，0，1，0)。以同样方式，DPCM 模式编码器 752 产生用于符号 5 的位置索引，其对应于(T，S，T+S)。用于该符号 5 的位置索引是(1，1)。

随后，DPCM 模式编码器 752 把该标志和该位置索引输出到该表格大小计算器 754。

再一次参考图 8B 和 9C，表格大小计算器 754 计算用于编码在一个出现模式中的输入的差分数据的一个符号表的大小(A)，以及用于编码在一个递增模式中的输入的差分数据的一个符号标志的大小(B)，其对应于步骤 S9640 中在预先设置的符号表中的符号。

在步骤 S9660 中，表格大小计算器 754 把使用在该出现模式编码器 756 中的符号表的大小（A=S*(AQP+1)，其中 S 表示差分数据中包括的符号的数量而 AQP 表示用于表示一个符号的比特的大小)与对应分别符号的符号标志的大小（$B=2^{AQP+1}-1$，其中 AQP 表示用于表示一个符号的比特的大小)相比较。

如果 A 小于 B，则表格大小计算器 754 把每一个顶点的差分数据输出到出现模式编码器 756，如果 B 小于 A，则把该差分数据输出到增量模式编码器 758。

下面参照图 11B 描述出现模式编码器 756 的操作。

在步骤 S9680 中，出现模式编码器 756 产生对应于每一顶点的输入差分数据的值的符号，以及表示其各自符号的位置的位置索引。

参考图 11B，当一个顶点的输入差分数据是(3，7，3，7，-4，7，3，

-4，3，7，-4，-4)，时，该出现模式编码器 756 制备一个表格，其中对应于每一顶点的差分数据的差分值的符号 3，7，和-4 被顺序地写入一行。

出现模式编码器 756 编码该符号数组中的第一个符号 3 并且产生针对该符号 3 的位置索引，以使 3 处在而位置由 1 表示而其它位置由 0 表示。针对该符号 3 的位置索引是(0 1 0 0 0 1 0 1 0 0 0)。

随后，出现模式编码器 756 产生用于下一个符号 7 的位置索引。如图 11B 所示，在用于下一个符号的位置索引的产生中，前个符号的位置不被再次计算。因此，针对该符号 7 的位置索引是(1 0 1 0 1 0 0)。

在出现模式编码器 756 中，仅考虑尚未编码的符号位置产生用于符号的全部位置索引，因此用于一个符号-4 的位置索引是(1 1 1)。

在图 11B 中，标志 bSoleKV 被设置为 0。标志 bSoleKV 表示在差分数据的符号数组中一个符号是否仅出现一次。如果一个符号仅出现一次并且因此其位置索引仅包括 0，则用于该对应符号的 bSoleKV 被设置为 1，并且该对应符号的位置索引不被编码。出现模式编码器 756 把该输入差分数据的符号、该符号的位置索引以及 bSoleKV 输出到用于熵编码差分数据的熵编码器 760。

下面参照图 11C 描述根据本发明的增量模式编码器 758 的操作。

在步骤 S9690，增量模式编码器 758 产生指示包括在一个预定符号表中的符号是否存在于输入的差分数据中的一个符号标志以及用于该符号的位置索引。

增量模式编码器 758 预先产生用于被期望存在于输入差分数据中的符号的一个表格。在该表格中，以从具有最低绝对值的一个符号到具有最低大对值的一个符号的次序把符号排列在列中，并且在具有相同的绝对值的两个符号之间，具有正值的一个符号被放置在比另一符号更高的行中。因此，符号写入该表格中的次序是 0，1，-1，2，-2，3，-3，....。对应于一个符号表中的符号的一个符号标志的大小为 2^{AQP+1}-1。例如，如果 AQP 是 2，则能够由一个符号标志表示的符号的数量是 7。如果一个值对应于存在差分数据中的一个符号，则该符号标志设置为 1。仅针对其符号标志被设置为 1 的符号产生位置索引。

参考图 11C，如果输入到增量模式编码器 758 的差分数据是(-1，-3，-1，-3，2，-3，-1，2，-1，-3，2，2)，则存在于该差分数据中的符号是(-1，2，

-3)，因此确定该符号标志为(0，0，1，1，0，0，1)。

增量模式编码器 758 产生用于一个符号的位置索引，该符号在符号表中的定位首先在比其它符号高的行中。如图 11C 所示，增量模式编码器 758 设置一个符号-1 所在的位置，在该符号表中排列该差分数据中存在的符号中的第一个以 1 出现的符号，并且以 0 设置其它放置，使得用于该符号-1 的位置索引是(1 0 1 0 0 0 1 0 1 0 0 0)。

随后，增量模式编码器 758 产生用于一个符号 2 的位置索引(0 0 1 0 1 0 1 1)而不考虑该已经编码的符号-1 的位置。最后，增量模式编码器 758 产生用于符号 3 的位置索引(1111)，而不考虑该已经编码的符号-1 和 2 的位置。增量模式编码器 758 把用于其各自符号的符号标志和位置索引输出到熵编码器 760。

由出现模式编码器 756 和增量模式编码器 758 产生的全部位置索引具有称为 nTrueOne 的一个标志，其指示原来的位置索引是否已经反向。具体地说，如果 nTrueOne 被设置为 0，则认为位置索引是通过反向其原来位置索引而获得的。在位置索引包括许多 1 的情况中，有可能通过反向该位置索引而增强该算法编码效率，以便增加 0 的数量。

下面参照图 9D 描述熵编码器 760 的操作。

根据本发明的熵编码器 760，使用函数 enodeSignedQuasiAAC()熵编码从增量模式编码器 758 输入的表示该差分数据的符号的符号标志和用于该符号的位置索引，并且熵编码从出现模式编码器 756 输入的该差分数据的符号和其各自的位置索引。

在 encodeSignedQuasiAAC 中，使用涉及输入值和其符号的一个上下文产生一种自适应算法编码的比特数据流。具体地，在 encodeSignedQuasiAAC()中，不为 0 的一个第一比特被编码，随后编码其符号，并且使用一个零上下文编码其它比特。

图 9D 是使用 encodeSignedQuasiAAC()编码一个符号的一个处理的流程图。

在步骤 S9810 中，熵编码器 760 接收将要被编码的差分数据的符号 nValue 和其比特大小 QBit。

在步骤 S9820 中，熵编码器 760 从 nQBit 减去 2，并且存储该相减的结果作为可变 i。

在步骤S9830中，熵编码器760把该符号nValue的绝对值存储作为变量val，并且对于val执行次数为i的右移位(SR)操作。在同一个步骤中，熵编码器760对于1和该SR操作的结果执行一个逻辑"与"操作，并且把该逻辑"与"操作的结果存储为一个变量比特。

在使用 encodeSignedQuasiAAC()编码一个符号的处理的第一周期中，检测除了符号位以外的将要被熵编码的输入值中的第一比特，并且在随后的周期中逐个读出其它比特。

在步骤S9840中，熵编码器760检测val是否大于1。如果val大于1，则在步骤S9850中，在一个零上下文下使用函数qf_encode()编码该'比特'的值。另一方面，如果val不大于1，则在步骤S9860中，在一个第i上下文之下使用函数qf_encode()编码该'比特'的值。

当val不大于1时，在步骤S9870中，熵编码器760再一次检验val是否为1。如果val是1，则在步骤S9880设置nValue的符号，并且在步骤S9890中根据其符号和一个符号上下文编码nValue。

当完成针对一个比特的编码处理时，熵编码器760在步骤S9900中把i减1，随后在步骤S9910中检测i的当前值是否小于0。通过重复地执行S9830至S9900，熵编码器760熵编码该输入值，直到i小于0为止。

因此，随着分配到该第一比特的一个上下文，熵编码器760编码输入值不是0的第一比特，并且编码跟随该零上下文的其它比特。

下面参照图7A描述在标题编码器500中的将要被编码成一个关键字值标题的信息。

标题编码器500接收一个输入的坐标协调程序并且编码一个数据模式、每一关键帧中的顶点的数目、用于该顶点数目的所需要的比特数、以及每一浮点数的有效位的最大数目。

标题编码器500编码一个量化比特数、每一顶点的每一成份的关键值数据中的最小值和每一顶点的每一成份的数据范围中的最大数据范围、以及每一顶点的每一成分的量化数据中的最大和最小值。

标题编码器500从DPCM处理器720接收已经对于每一顶点的每一成份的数据执行的一个DPCM操作的模式，从该字典编码器750接收一个字典编码模式，并且编码该DPCM操作模式和该字典编码方式。

下面参照附图更详细地描述根据本发明优选实施例的一个用于解码一

个比特数据流的方法和装置，其比特数据流中编码有坐标内插符的关键字数据和关键值数据。

图12是根据本发明一个优选实施例的用于解码坐标内插符的装置的框图。参考图12，用于解码坐标内插符的装置包括：关键字数据解码器1300，从一个输入比特数据流中解码关键字数据；关键值数据解码器1500，从该输入比特数据流中解码关键值数据；以及标题解码器1800，解码标题信息并且把该解码的标题信息提供到关键字数据1300以及关键值数据解码器1500，该标题信息是用于解码来自输入比特数据流的关键字数据和关键值数据的必需的信息。

下面参照图13至14B更详细地描述关键字数据解码器1300的结构和操作。

图13是根据本发明的一个优选实施例的关键字数据解码器1300的一个框图。关键字数据解码器1300接收一个编码的比特数据流并且通过解码而将其重新构成关键字数据。

关键字数据解码器1300包括一个熵解码器1310、一个反向DND处理器1320、一个反向折叠处理器1330、一个反向移位器1340、一个反向DPCM处理器1350、一个反向量化器1360、一个线性关键字解码器1380和一个浮点数反向转换器1370。

图14A是根据本发明优选实施例的用于解码关键字数据的一种方法的流程图。参考图14A，其上压缩有关键字数据的一个比特数据流被输入到标题解码器1800和熵解码器1310。

在步骤S14000中，标题解码器1800解码为了进行解码的每一步骤所需的信息段并且将它们提供到对应解码步骤。下面按每一解码步骤描述由标题解码器1800解码的信息。

在步骤S14100中，熵解码器1310从标题解码器1800接收将要被解码的差分数据的数目以及已经用于编码的比特数量，即将要被用于进行解码的比特数目，并且解码该输入的比特数据流。差分数据的数目等于从关键字数据的数量减去通过执行DPCM获得的帧内关键字数据数目的结果。

在本实施例中，熵解码器1310根据包括在该比特数据流中的预定的信息，例如bSignedAACFlag，标识将要被解码的差分数据是否具有负值或正值。如果该编码的差分数据具有负值，则熵解码器1310使用函数

decodeSignedAAC()解码该差分数据。另一方面，如果该编码的差分数据仅具有正值，则熵解码器 1310 使用函数 decodeUnsignedAAC()解码该差分数据。随后，该解码的差分数据被发送到反向 DND 处理器 1320。

反向 DND 处理器 1320 在每一 DND 的周期中从标题解码器 1800 接收 DND 的阶数以及最大值 nKeyMax。

如果 DND 的阶数是-1，这意味着正解码的该编码的差分数据已经经历一个 DPCM 操作和一个移位操作而不经历 DND 被熵解码，并且本方法直接进入执行一个反向移位操作的步骤。如果 DND 的阶数是 0，这意味着正解码的该编码的差分数据已经经历一个折叠操作而不经历 DND 被熵解码，并且因此本方法直接进入执行一个反向折叠操作的步骤。如果 DND 的阶数大于 0，则在步骤 S14200 中执行一个反向 DND 操作。

反向 DND 处理器 1320 在步骤 S14300 中确定正被解码的编码的差分数据是否已经经历一个上移动操作而被编码。在本发明的一个优选实施例中，通过查验在一个比特数据流中包含的 nKeyInvertDown 是否大于 0 而确定该正解码的该编码差分数据已经经历一个上移位操作而被编码。

如果正被解码的该编码差分数据尚未通过上移位操作，则该方法进入执行一个反向 DND 的步骤。另一方面，如果正被解码的该编码差分数据已经通过一个上移位操作，在步骤 S14400 中，通过执行一个上移位，已经从一个正数区域转移到一个负数区域的差分数据被移回到该负数区域。在本发明的一个优选实施例中，通过执行由下面方程式表示的一个下移位操作(一个反向下操作)恢复已经通过上移动操作的差分数据。

$$
\begin{aligned}
&invert-down(v) && \cdots(20)\\
&= v && (if\ v \le nKeyInvertDown)\\
&= nKeyInvertDown - v && (if\ v > nKeyInvertDown)
\end{aligned}
$$

这里，nKeyInvertDown 具有与使用在上移位操作中的最大值 nKeyMax 相同的值。作为下移位操作的结果，具有超过 nKeyInvertDown 值的差分数据被转换成低于-1 的负值。

根据每一个 DND 周期中的最大值 nKeyMax，对于已经通过下移位操作的差分数据有选择地执行一个反向下分割操作或反向上分割操作。

参考图 14B，反向 DND 处理器 1320 执行一个次数与该差分数据在编码过程中已经通过的 DND 操作的次数一样多的反向 DND 操作。换句话说，

反向DND处理器1320设置一个反向DND阶数的初始值等于DND的阶数。随后，反向DND处理器1320每次执行一个反向DND操作就从该反向DND的阶数的初始值减去1，并且保持执行该反向DND操作，直到该反向DND的阶数变成1为止。在步骤S14510中，反向DND处理器1320在DND的每一周期中搜索nKeyMax并且检验每一nKeyMax是否不小于0。

如果nKeyMax小于0，则意味着该编码处理中已经执行了上分割操作，并且因此反向DND处理器1320在步骤S14530中通过执行一个反向上分割操作而把正在解码的差分数据的范围延伸到一个负数区域。在本发明的一个优选实施例中，可以使用由方程式(21)定义的一个反向上分割操作。

$$inverse-divide-up(v) \quad \cdots(21)$$

$$= v \quad (if \quad v \geq 0)$$

$$= (nKeyMax_i - 1) - \frac{v-1}{2} \quad (if \quad v < 0, v \bmod 2 \neq 0)$$

$$= \frac{v}{2} \quad (if \quad v < 0, v \bmod 2 = 0)$$

但是，如果nKeyMax不小于0，则反向DND处理器1320检测该反向DND的阶数是否为1。如果反向DND的阶数不是1，则意味着在编码处理中该正解码的差分数据已经被执行了下分割操作，因此反向DND处理器1320在步骤S14570中通过执行一个反向下分割操作把该差分数据的范围延伸到一个正数区域。

在本发明的一个优选实施例中，可以使用下面方程式定义的一个反向下分割操作。

$$inverse-divide-down(v) \quad \cdots(22)$$

$$= v \quad (if \ v \geq 0)$$

$$= (nKeyMax_i + 1) + \frac{v-1}{2} \quad (if \ v < 0, v \bmod 2 \neq 0)$$

$$= \frac{v}{2} \quad (if \ v < 0, v \bmod 2 = 0)$$

如果nKeyMax不小于0而反向DND的阶数是1，则反向DND处理器1320在步骤S14590中执行一个反向分割操作之后完成整个反向DND操作。在本发明的一个优选实施例中，可以使用由方程式(23)定义的一个反向分割操作。

$$inverse-divide(v) \quad \cdots(23)$$
$$= v \quad (if\ v \geq 0)$$
$$= v + (nKeyMax_0 + 1) \quad (if\ v < 0)$$

已经通过相反 DND 操作的关键字数据的差分数据被输入到反向折叠处理器 1330，并且反向折叠处理器 1330 在步骤 S14600 中对于该差分数据执行一个反向折叠操作，使得该只被用于一个正数区域中的差分数据被分成正值和负值。在本发明的一个优选实施例中，可以使用由方程式(24)定义的一个反向折叠操作。

$$inverse-fold(v) = -\frac{(v+1)}{2} \quad (if\ v \bmod 2 \neq 0) \quad \cdots(24)$$
$$= \frac{v}{2} \quad (if\ v \bmod 2 = 0)$$
$$= 0 \quad (if\ v = 0)$$

已经通过反向折叠操作的差分数据被输出到反向移位器 1340，在步骤 S14700 中，反向移位器 1340 把从标题解码器 1800 输入的、在编码处理中使用的一个模式 nKeyShift 加到从反向折叠处理器 1340 输入的差分数据。此操作由下面方程式表示。

$$inverse-shift(v) = v + nKeyShift \quad \cdots(25)$$

在步骤 S14800 中，反向 DPCM 处理器 1350 使用从标题解码器 1800 输入的 DPCM 的阶数把从反向移位器 1340 输入的差分数据恢复成量化关键字数据。反向移位器 1340 遵循方程式(26)执行与 DPCM 的阶数一样多次的反向 DPCM 操作。

$$v(i+1) = v(i) + delta(i) \quad \cdots(26)$$

其中，i 表示差分数据和关键字数据的一个索引，v 表示整数的一个数组，而 delta(i)表示差分数据。

已经通过该反向 DPCM 操作的量化关键字数据被输入到反向量化器 1360。随后，反向量化器 1360 从标题解码器 1800 接收有关量化比特的大小 nKeyQBit 和用于反向量化的最大和最小值是否由浮点数转换器 315 编码的信息，并且在步骤 S14900 中使用下面方程式把该量化的关键字数据转换成反向量化的关键字数据。

$$inverse-quantize(v) = fKeyMin + \frac{v}{2^{nKeyQBit}-1} \times (fKeyMax - fKeyMin) \quad \cdots(27)$$

如果用于量化的最大和最小值在编码关键字数据的处理中尚未由浮点数转换器 315 转换，则方程式(27)所示的 fKeyMin 和 fKeyMax 被分别设置为 0 和 1。但是，如果用于量化的最大和最小值已经由浮点数转换器 315 所转换，则由浮点数反向转换器 1370 反向转换的最大和最小值分别被用作该最大和最小值，以便反向量化。

稍后描述实现反向量化的反向 DND 操作的程序代码的一个实例。

从反向量化器 1360 输出的解码的关键字数据加到在线性关键字解码器 1380 中解码的关键字数据，因此构成解码的关键字数据。

下面，在下列段落中描述一个线性关键字解码过程 S15000。

标题解码器 1800 解码来自一个比特数据流的关键字标题信息。如果在该比特数据流中存在关于一个线性关键字区域的信息，则标题解码器 1800 把用于解码该线性关键字数据区域的开始和结束关键字数据所需的信息输出到浮点数反向转换器 1370，并且把编码为线性关键字的关键字数目输出到线性关键字解码器 1380。

浮点数反向转换器 1370 把由十进制数表示的该线性关键字数据区域的开始和结束关键字反向地转换成二进制数字，并且把该二进制数字输出到线性关键字解码器 1380。

假定将要被解码的两个浮点数称为 fKeyMin 和 fKeyMax，则解码 fKeyMin 的一个处理如下。

标题解码器 1800 从比特数据流读出 fKeyMin 的位数。如果 fKeyMin 的位数是 0，则 fKeyMin 被设置为 0，并且从该比特数据流读出 fKeyMax 的位数以便解码 fKeyMax。如果 fKeyMax 的位数不小于 8，则意味着 fKeyMax 已经遵循 IEEE 标准 754 被编码。因此，在读出 32 比特的 fKeyMax 之后，解码该浮点数 fKeyMax。

但是，如果 fKeyMax 的位数在 1 和 7 之间，标题解码器 1800 从该比特数据流读出一个符号位。在本发明的一个优选实施例中，如果符号位是 1，则 MinKeyMantissaSign 被设置为-1。另一方面，如果符号位是 0，则 MinKeyMantissaSign 被设置为 1。然后，参考表格 1 获得用于进行解码所需的比特的数量，表格 1 示出一个尾数的位数和用于进行编码所需的比特数目之间的关系。随后，与用于进行编码所需的比特数一样多的比特数据流的比特被读出并且存储在 nMinKeyMantissa 中。随后，读出该比特数据流的

下一个比特，并且存储在 MinKeyExponentSign 中，与作为尾数的符号存储在 MinKeyMantissaSign 中的方式相同。对应于一个指数值的该比特数据流的随后六个比特被读出并且存储在 nMinKeyExponent 中。

通过把从标题解码器 1800 输入的值替代到方程式(28)中，浮点数反向转换器 1370 恢复 fKeyMin。

$$fKeyMin = \frac{MinKeyMantissaSign * nMinKeyMantissa}{10^{MinKeyExponentSign * nMinKeyExponent}} \quad \cdots(28)$$

恢复 fKeyMax 的处理与恢复 fKeyMin 的处理相同。具体地说，在从该比特数据流读出 fKeyMax 的指数之前，确定是否 fKeyMin 指数的相同值被用作 fKeyMax 的指数。如果 fKeyMin 的指数相同的值不被用作 fKeyMin 的指数，则以从比特数据流读出 fKeyMin 的指数的同样方式从该比特数据流读出 fKeyMax 的指数。

线性关键字解码器 1380 从浮点数反向转换器 1370 接收该线性关键字数据区域的开始和结束关键字，并且遵循方程式(29)解码该线性关键字数据区域。

$$Key_i = fKeyMin + \frac{(fKeyMax - fKeyMin) * i}{(nNumberOfLinearKey - 1)} \quad \cdots(29)$$
$$(i = 0, \ldots, nNumberOfLinearKey - 1)$$

其中，fKeyMin 和 fKeyMax 分别表示该线性关键字数据区域的开始和结束关键字数据。

使用上述方法解码的该线性关键字数据区域中的关键字数据被加到从相反量化器 1360 输出的关键字数据，然后该相加的结果被输出作为最终关键字数据。

下面将参照图 15A 和 15B 描述根据本发明的解码坐标内插符已编码的关键值数据的一个关键值数据解码器和根据本发明的用于解码坐标内插符的关键值数据的一种方法。

图 15A 是根据本发明优选实施例的关键值数据解码器的框图，而图 15B 是根据本发明优选实施例的用于解码已编码的关键值数据的一种方法的流程图。

参考图 15A，关键值数据解码器 1500 包括：熵解码器 1505，熵解码一个输入的比特数据流并且因此产生将要被字典解码的数据，包括 DPCM 的

差分数据、符号标志、用于该符号的位置索引和一个 DPCM 操作模式；字典解码器 1510，根据将要被字典编码的数据的符号和其位置索引产生差分数据；反向 DPCM 处理器 1530，根据一个 DPCM 操作模式，通过对于差分数据执行一个预定的反向 DPCM 操作而产生量化数据；反向量化器 1550，通过反向量化已经量化的数据而产生恢复的关键值数据；以及标题解码器 1800，其解码用于从该输入比特数据流解码坐标内插符需要的信息，并且把该信息输出到字典解码器 1510、反向 DPCM 处理器 1530 和反向量化器 1550。

在下面将参照图 15B 描述根据本发明的用于解码坐标内插符的已编码的关键值数据的方法。

在步骤 S1710 中，其上被编码有坐标内插符的一个比特数据流被输入到熵解码器 1505，然后熵解码器 1505 解码该输入的比特数据流。如果该输入的比特数据流已经以一个出现模式编码，则熵解码器 1505 在步骤 S1720 中把每一顶点的符号和其位置索引输出到字典解码器 1510。另一方面，如果该输入的比特数据流已经以一个增量模式编码，则熵解码器 1505 在步骤 S1720 中把指示符号存在的符号标志和用于该符号的位置索引输出到字典解码器 1510。

在步骤 S1730 中，根据该输入的字典编码模式，字典解码器 1510 通过解码从该熵解码器 1510 以出现模式输入的该符号和位置索引或通过解码从熵解码器 1510 以增量模式输入的符号标志和位置索引而产生差分数据，并把该产生的差分数据输出到反向 DPCM 处理器 1530。

在步骤 S1740 中，反向 DPCM 处理器 1530 根据该输入差分数据的解码的 DPCM 工作模式，通过对于从字典解码器 1510 输入的该差分数据执行反向时间 DPCM 操作、反向空间 DPCM 操作以及反向时空 DPCM 操作之一而产生量化的关键值数据，并且把该量化的关键值数据输出到反向量化器 1550。

在步骤 S1750 中，反向量化器 1550 使用从该关键字值标题解码器 1800 输入的每一个成份的数据中的最小值和该最大数据范围，反向量化从反向 DPCM 处理器 1530 输入的已量化的关键值数据。

在步骤 S1760 中，反向量化器 1550 检查该反向量化的关键值数据的矩阵是否已经在该编码处理过程中转换成一个转置矩阵，并且如果该反向量

化的关键值数据的矩阵已经转置，则在步骤 S1765 中反向地转换该转置矩阵。

在步骤 S1770 中，反向量化器 1550 输出一个恢复的坐标内插符的关键值数据。

下面参照图 16A 至 17B 更详细描述该关键值数据解码器 1500 的结构和操作。

熵解码器 1505 首先从一个输入比特数据流中解码一个指示一个 DPCM 模式的比特数据流，然后解码包括 bSelFlag、nKVACodingBit、nQMin 和 nQMax 的数列。

在该编码处理中，首先把 bSelFlag 和 nKVACodingBit 分别设置为 1 和 0。如果 bSelFlag 被解码成 1，则熵解码器 1505 解码 nKVACodingBit、nQMin 和 nQMax。另一方面，如果 bSelFlag 被解码成 0，则熵解码器 1505 只解码 nQMin。

在解码 bSelFlag、nKVACodingBit、nQMin 和 nQMax 的数列以后，熵解码器 1505 解码指示字典编码模式的 nDicModeSelect。根据 nDicModeSelect 的值，将要被解码的比特数据流被分成下面段落将被描述的两个不同种类。

图 18A 是一个说明坐标内插符的每一顶点和每一个顶点的成分数据的比特数据流的结构的示意图。如图 18A 所示，如果 nDicModeSelect 是 0，一个比特数据流包括已经在出现模式编码器中编码的符号和位置索引。另一方面，如果 nDicModeSelect 是 1，则一个比特数据流包括已经在增加模式编码器中编码的符号标志和位置索引。

上面已经描述的根据本发明的熵解码器使用以图 18B 所示的程序代码实现的一个函数 decodeSignedQuasiAAC()。在函数 encodeSignedQuasiAA() 中，使用涉及输入值及其符号的一个上下文对一种自适应算法编码的比特数据流进行编码。具体地说，在函数 decodeSignedQuasiAAC()中，使用一个零上下文对跟随一个符号位后的比特解码。熵解码器 1505 把该解码的数据输出到字典解码器 1510。

图 16A 是根据本发明的字典解码器 1510 的一个框图，而图 17A 是字典编码方法的流程图。

如图 16A 所示，字典解码器 1510 包括：DPCM 模式解码器 1512，恢复输入其中的每一顶点的一个 DPCM 模式；字典模式选择器 1514，选择输

入每一顶点的字典解码模式；出现模式解码器 1516，从字典模式选择器 1514 接收每一顶点的每一成分的符号和针对该符号的位置索引，并且恢复差分数据；以及增量模式解码器 1518，其从字典模式选择器 1514 接收一个符号标志和用于该符号的位置索引，并且恢复差分数据。

参考图 17A，在步骤 S1731 中，包括该符号、符号标志和位置索引的每一顶点的熵解码的成分数据被输入到 DPCM 模式解码器 1512。

在字典解码的差分数据被输出到反向 DPCM 处理器 1530 之前，在步骤 S1732 中，DPCM 模式解码器 1512 解码一个反向 DPCM 操作的模式，该反向 DPCM 操作是在反向 DPCM 处理器 1530 中将对于每一顶点的每一成份的差分数据执行的操作。

下面参考图 19A 描述 DPCM 模式解码。

除了表示每一顶点的每一成分的 DPCM 模式的组合的符号数目被固定在 27、因此符号表的大小也被固定在 27 之外，DPCM 模式解码与稍后将被描述的增量模式解码相同。

DPCM 模式解码器 1512 接收一个 DPCM 模式标志并且跟随输入的位置索引把对应于该 DPCM 模式标志的符号记录在一个数据组中。

例如，如图 19A 所示，对应于该输入 DMCM 模式标志的符号是 1(T T S)，4(T S S)，和 5(T S T+S)，而其各自的索引是(0 1 0 1 0 0)、(1 0 1 0)和(1 1)。因此，使用符号 1 和其位置索引(0 1 0 1 0 0)恢复一个数据组(X 1 X 1 X X)，使用符号 4 和其位置索引(1 0 1 0)恢复一个数据组(4 1 X 1 4 X)，以及使用符号 5 和其位置索引(1 1)恢复一个数据组(4 1 5 1 4 5)。

该恢复的数据组(4 1 5 1 4 5)被转换成 DPCM 模式(T S S)(T T S)(T S T+S)(T T S)(T S S)(T S T+S)的组合的一个数组。因此，有可能根据恢复的数据组识别已经对于每一顶点的每一成分执行了哪一种 DPCM。

DPCM 模式解码器 1512 把每一顶点的每一成分的差分数据连同该解码的 DPCM 模式信息一起输出到字典模式选择器 1514。

在步骤 S1734 中，字典模式选择器 1514 根据每一顶点的每一成分的 nDicModeSelect 的值，把从 DPCM 模式解码器 1512 输入的每一顶点的成份数据输出到出现模式解码器 1516 或增量模式解码器 1518。

如果 nDicModeSelect 是 0，则字典模式选择器 1514 把顶点的成份数据输出到出现模式解码器 1516，而如果 nDicModeSelect 是 1，则字典模式选

择器 1514 把顶点的成份数据输出到增量模式解码器 1518。

在步骤 S1736 中，出现模式解码器 1516 把每一个成分的该符号数据和位置索引恢复成差分数据。

图 19B 是说明出现模式解码的一个实例的示意图。参考图 19B，出现模式解码器 1516 从字典模式选择器 1514 接收符号数据并且检验 bSoleKV 和 nTrueOne。

如果 bSoleKV 表示在差分数据中有多个输入符号并且 nTrueOne 表示位置索引尚未反向，则出现模式解码器 1516 通过在一个数据组中在由其各自位置索引指示的各自位置上插入输入符号而恢复差分数据。

例如，出现模式解码器 1516 顺序地接收符号 3、7 和-4 以及其各自的位置索引(0 1 0 0 0 1 0 1 0 0 0)、(1 0 1 0 1 0 0)和(1 1 1)。

出现模式解码器 1516 在跟随位置索引(0 1 0 0 0 1 0 1 0 0 0)的一个差分数据组中记录第一符号 3。因此，通过把符号 3 插入在该差分数据组中的对应于该位置索引(0 1 0 0 0 1 0 1 0 0 0)中 1 所处在的位置而获得(3 X 3 X X X 3 X 3 X X X)。

出现模式解码器 1516 恢复该随后的符号 7。在恢复该符号 7 的过程中，不考虑该差分数据组中的符号 3 的位置，使得用于该符号 7 的位置索引不是(0 1 0 1 0 0 0 1 0 0)，而是(1 0 1 0 1 0 0)。

出现模式解码器 1516 在该差分数据组中的没被符号 3 占用的位置中的第一位置记录该符号 7，然后在该差分数据组中对应于位置索引(1 0 1 0 1 0 0)中的 1 处在的位置记录该符号 7。因此，在恢复符号 7 之后，该差分数据组是(3 7 3 7 X 7 3 X 3 7 X X)。

出现模式解码器 1516 按照该索引(1 1 1)恢复该符号-4，并且因此产生差分数据组是(3 7 3 7 -4 7 3 -4 3 7 4 -4)。

如果 bSoleKV 被设置为 1，则意味着在差分数据中仅存在一个输入符号，并且没有用于该输入符号的位置索引。因此，出现模式解码器 1516 把该输入符号记录在一个空白差分数据组中的第一位置，并且执行用于恢复该下一个符号的处理。

在步骤 S1736 中，增量模式解码器 1518 把每一个成分的符号标志和位置索引恢复成差分数据。下面参考图 19C 描述增量模式解码。

增量模式解码器 1518 从字典模式选择器 1514 接收指示在差分数据中

是否存在符号的符号标志、指示位置索引是否已经反向的 nTrueOne 以及该位置索引。

增量模式解码器 1518 根据该输入符号标志解码在该差分数据中包括的符号。像用于增量模式编码的符号表一样，在用于增量模式解码的一个符号表中，按照从具有最低绝对值的符号到具有最大绝对值的符号的次序把符号排列为一列，并且在具有相同绝对值的两个符号之间，具有正值的符号排列在比另外一个符号高的一行中。符号标志的大小是 $2^{nKVCodingBit+1}$-1，其中 nKVCodingBit 表示在熵解码器 1505 中解码的量化比特的数量。因此，如果一个符号标志是(0 0 1 1 0 0 1)，则增量模式解码器 1518 解码存在于该差分数据中作为符号存在的-1，2，和-3。

符号标志之后输入的位置索引分别是 (1 0 1 0 0 0 1 0 1 0 0 0), (0 0 1 0 1 0 1 1)，和(1 1 1 1) 并且分别对应于符号-1、2 和 3。

增量模式解码器 1518 在差分数据组中对应于该位置索引 (1 0 1 0 0 0 1 0 1 0 0 0)中 1 处在的位置的位置记录符号-1，使得产生的数据组是(-1 X -1 X X X -1 X -1 X X X)。

随后，增量模式解码器 1518 通过在该差分数据组中的对应于在该位置索引(0 0 1 0 1 0 1 1)中 1 处在的位置的位置中记录 2 而恢复该符号 2。在恢复该符号 2 的过程中，不考虑该差分数据组中的第一个符号-1 的位置，使得该产生的差分数据组是(-1 X -1 X 2 X -1 2 -1 X 2 2)。

增量模式解码器 1518 通过在该差分数据组中的对应于在位置索引(1 1 1 1)中的 1 处在的位置的位置记录-3 而恢复该符号-3，使得产生的差分数据组是(-1 -3 -1 -3 2 -3 -1 -3 2 2)。

在步骤 S1739 中，出现模式解码器 1516 和增量模式解码器 1518 恢复每一顶点的每一成分的差分数据，并且把恢复的差分数据输出到反向 DPCM 处理器 1530。

图 16B 是根据本发明的反向 DPCM 处理器 1530 的一个框图，而图 17B 是一个反向 DPCM 操作的流程图。

参考图 16B，根据本发明的反向 DPCM 处理器 1530 包括：反向时间 DPCM 操作器 1542，对于输入的差分数据执行一个反向时间 DPCM 操作和一个反向循环量化操作，然后输出坐标内插符的量化的关键值数据；反向空间 DPCM 操作器 1544，对于输入的差分数据执行一个反向空间 DPCM 操

作和一个反向循环量化操作，随后输出该量化的关键值数据；反向时空DPCM操作器1546，对于输入的差分数据执行一个反向时空DPCM操作和一个反向循环量化操作，随后输出该量化的关键值数据；以及反向DPCM模式选择器1535，把输入其中的差分数据输出到反向时间DPCM操作器1542、反向空间DPCM操作器1544和反向时空DPCM操作器1546之一。

参考图17B，在步骤S1742中，反向DPCM模式选择器1535将根据在DPCM模式解码器1512中恢复的每一顶点的每一成分的DPCM操作模式而确定将对于输入其中的差分数据执行的一个反向DPCM操作，并且按照该反向DPCM操作模式而输出每一顶点的每一成分的该输入的差分数据。

DPCM操作器1542、1542和1546的每一个同时对于输入其中的差分数据执行一个反向DPCM操作和一个反向循环量化操作。

在步骤S1744中，反向时间DPCM操作器1542遵循方程式(30)对于该输入的差分数据执行一个反向时间DPCM操作，在步骤S1746中，反向空间DPCM操作器1544遵循方程式(31)对于输入的差分数据执行一个反向空间DPCM操作，在步骤S1748中，反向时空DPCM操作器1546遵循方程式(32)对于该输入的差分数据执行一个反向时空DPCM操作。

$$\tilde{V}_{i,j} = D_{i,j} + \tilde{V}_{i-1,j} \quad \cdots(30)$$

$$\tilde{V}_{i,j} = D_{i,j} + \tilde{V}_{i,Ref} \quad \cdots(31)$$

$$\tilde{V}_{i,j} = D_{i,j} + \{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\} \quad \cdots(32)$$

在方程式(30)至(32)中，$\tilde{V}_{i,j}$表示在第i关键帧中的第j个顶点的量化关键值数据，$D_{i,j}$表示在该第i关键帧中的该第j个顶点的差分数据，而Ref表示一个参考顶点。

在方程式(31)和(32)中，如果$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$小于每一个成份的量化关键值数据中的最小值，则使用该最小值而不使用$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$。如果$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$大于在每一成分的量化关键值数据当中的最大值，则使用该最大值而不使用$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$。

DPCM操作器1542、1544和1546的每一个都使用方程式(33)执行一个反向DPCM操作并且同时执行一个反向循环量化操作，以便扩展已经在该编码处理过程中降低的差分数据的范围。

$$InverseCircularQuantization(X_i): \quad \cdots(33)$$

$$X'_i = \tilde{X}_i - (nQMax - nQMin + 1) \quad (if\ \tilde{X}_i \geq 0)$$

$$X'_i = \tilde{X}_i + (nQMax - nQMin + 1) \quad (otherwise)$$

$$\hat{X}_i = \hat{X}_{i-1} + \tilde{X}_i \quad (if\ nQMin \leq \hat{X}_{i-1} + \tilde{X}_i \leq nQMax)$$

$$\hat{X}_i = \hat{X}_{i-1} + X'_i \quad (if\ nQMin \leq \hat{X}_{i-1} + X'_i \leq nQMax)$$

在方程式(33)中，$\tilde{X}_i$是与$D_{i,j}$相同的输入值，$\hat{X}_{i-1}$是与$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$一样的一个在前反向循环量化值。nQMax和nQMin分别表示在DPCMed差分数据中的最大值和最小值。

在步骤SS1749中，反向DPCM处理器1530把已经反向DPCM的和反向循环量化的每一顶点的每一成分的关键值数据输出到反向量化器1550。

参考图15B，反向量化器1550遵循方程式(4)，把在从该关键字值标题解码器1800输入的在输入成份数据中的最小值fMin_X、fMin_Y和fMin_Z以及最大范围值fMax转换成二进制数，并且通过把fMin_X、fMin_Y、fMin_Z和fMax替代到方程式(34)中而反向量化从反向DPCM处理器1530输入的量化关键值数据。

$$\hat{V}_{i,j,x} = fMin\_X + \frac{\tilde{V}_{i,j,x}}{2^{nKVQBits} - 1} \times fMax \quad \cdots(34)$$

$$\hat{V}_{i,j,y} = fMin\_Y + \frac{\tilde{V}_{i,j,y}}{2^{nKVQBits} - 1} \times fMax$$

$$\hat{V}_{i,j,z} = fMin\_Z + \frac{\tilde{V}_{i,j,z}}{2^{nKVQBits} - 1} \times fMax$$

方程式(34)中，nKVQBits表示用于反向量化的一个量化比特的大小。

反向量化器1550必须以表2示出的矩阵的形式输出每一顶点的每一成分的反向量化的关键值数据。为了实现此目的，在步骤S1760中，反向量化器1550在输出该反向量化的关键值数据之前检验该反向量化的关键值数据的模式是否为一个转置模式。如果该反向量化的关键值数据的模式是一个转置模式，则在步骤S1765中，反向量化器1550通过反向转换该转置矩阵而产生并且输出坐标内插符的解码的关键值数据。

图 20A 至 20L 是示出 SDL 程序代码的实例的图表，通过该 SDL 程序代码，实现根据本发明优选实施例的用于解码坐标内插符的关键字数据和关键值数据的装置。

图 20A 是说明一个类 CompressedCoordinateInterpolator 的示意图，它是用于读出坐标内插符的压缩比特数据流的最高类。在图 20A 中，KeyHeader 和 Key 是用于从比特数据流读出关键信息的类，对应于常规 CoordinateInterpolator 节点中的关键字字段数据。CoordIKeyValueHeader 和 CoordIKeyValue 是用于读出对应于在常规 CoordinateInterpolator 结点中的 keyValue 字段数据的 keyValue 信息的类。函数 qf_start()被用于在读出该比特数据流的 AAC-编码的部分以前初始化一个算法解码器。

图 20B 是说明用于解码一个关键字标题的类的示意图，它是用于一个解码处理所需要的信息。

该关键字标题中的主信息是关键字的数目、量化比特、帧内关键字数据、用于进行解码的 DND 标题和实际比特。nKeyQBit 是用于该反向量化以便恢复该浮点关键字值的量化比特。nNumKeyCodingBit 表示关键字数据数的 nNumberOfKey 的比特大小，它表示关键字数据的数目。nKeyDigit 表示在该原始关键字数据中的最大有效位，并且能被用于舍入该解码的值。当在该标题中包括有关线性关键字子区域的信息时，bIsLinearKeySubRegion 标志设置为 1。在此情况中，能够使用跟随该 bIsLinearKeySubRegion 标志的解码的标题信息计算在整个关键字范围之内的某一子区域中包括的关键字。bRangeFlag 指示关键字数据的范围是否从 0 到 1。如果该范围不是 0 到 1，则从 KeyMinMax 类解码最小值和最大值。KeyMinMax 类重建用于反向量化的最小值和最大值。每一个值能够被分成尾数和指数。nBitSize 是若干关键字标题信息数据的比特量，它们是 nQIntraKey、nKeyShift 和 nKeyMax。nQIntraKey 是第一量化的帧内数据的幅值。它连同表示 nQIntraKey 的符号的 nQIntraKeySign 结合在一起。它被用作对该量化的关键字数据的剩余部分进行恢复的一个基础。对于在该内插符压缩中的所有的符号位，值 0 表示一个正号而 1 表示一个负号。nKDPCMOrder 是 DPCM 的阶数减 1。该阶数的范围可以从 1 到 3。量化帧内数据的数目与 DPCM 的阶数相同。

nKeyShift，连同符号位 nKeyShiftSign 一起，是表明在该关键字数据解码器中移位的量的整数。如果该 bShiftFlag 被设置为真，则解码这两个值。

nDNDOrder 是 DND(分割-与-分割)的阶数。DND 在该关键字数据解码器中描述。如果 nDNDOrder 的值是 7，则 bNoDND 被解码。此布尔值表示是否将进行反向 DND 处理。nKeyMax 是在连续反向 DND 处理每一个过程中使用的最大值或最小值。nKeyCodingBit 是用于编码关键字数据的比特。bSignedAACFlag 表示哪个解码方法被用于 AAC 解码。如果该值是 0，则执行无符号 AAC 解码。否则，执行有符号的 AAC 解码。bKeyInvertDownFlag 是表示是否使用 nKeyInvertDown 的布尔值。nKeyInvertDown 是整数值，使得所有的量化的高于该整数值的关键字数据被反向到从-1 和低于-1 开始的负值。如果 nKeyInvertDown 是-1，则不执行反向。图 20C 是说明一个类 LinearKey 的示意图。图 20C 中，nNumLinearKeyCodingBit 是表示为了编码线性可预测关键字的一个预定的数量的所需比特数目的一个值。nNumberOfLinearKey 是表示线性可预测关键字数目的一个值。

图 20D 是说明一个类 KeyMinMax 的示意图。图 20D 中，bMinKeyDigitSame 是一个标志，表示全部关键字的最高有效位的数目(nKeyDigit)和在关键字当中的一个最小值的最高有效位的数量是否相同。nMinKeyDigit 是表示在该关键字当中的该最小值的最高有效位数目的一个值。nMinKeyMantissaSign 是表示 nMinKeyMantissa 的符号的一个值。

nMinKeyMantissa 是表示在关键字当中的最小值的尾数的一个值。nMinKeyExponentSign 是表示 nMinKeyExponent 的一个符号的一个值。

nMinKeyExponent 是表示在关键字当中的最小值的指数的一个值。fKeyMin 是表示在关键字当中的最小值的一个值。bMaxKeyDigitSame 是一个标志，指示全部关键字的最高有效位的数目 nKeyDigit 和在关键字当中的一个最大值的最高有效位的数目是否相同。nMaxkeyDigit 是表示在该关键字当中的该最大值的最高有效位数目的一个值。nMinKeyMantissaSign 是表示 nMaxKeyMantissa 的符号的一个值。nMaxKeyMantissa 是表示在该关键字当中最大值的尾数的一个值。

bSameExponent 是一个标志，指示在关键字当中的最大值的指数是否 nMinKeyExponent 相同。nMaxKeyExponentSign 是表示 nMaxKeyExponent 的符号的一个值。nMaxKeyExponent 是表示在关键字当中的最大值的指数的一个值。FKeyMax 是表示在关键字当中的最大值的一个值。

图 20E 是说明一个类 Key 的示意图。在图 20E 中，nQKey 是从一个比

特数据流解码的量化关键字数据的一个数组。KeyContext是用于读取nQKey的一个幅值的一个上下文。KeySignContext是用于读取nQKey的一个符号的一个上下文。

DecodeUnsignedAAC是一个函数，利用下面将描述的一个给出上下文执行自适应算术编码的一个无符号解码过程。DecodeSignedAAC是一个函数，利用下面将描述的一个给出上下文执行自适应算术编码的一个有符号解码过程。

图20F是说明一个类CoordIKeyValueHeader的示意图。图20F中，在解码关键字标题数据之后，解码关键字值标题数据。一个关键字值标题包括：顶点的数量、用于关键值数据的量化参数以及在将被用于量化的关键值数据当中的最小值和最大值。bTranspose是指示当前模式是否为一个转置模式或一个顶点模式的标志。如果bTranspose是1，则选择一个转置模式。否则，选择一个顶点模式。nKVQBit表示为了用于恢复浮点数字的反向量化所需的量化比特。nCoordQBit表示nNumberOfCoord的比特大小，nNumberOfCoord表示顶点数目。nKVDigit在一个反向量化之后使用，表示关键值数据的有效位的最大数目。一个类KeyValueMinMax恢复在关键值数据的每一成份的量化值和一个将用于反向量化最大关键值数据范围中的最小值。上述的值的每一个能够被分成各自的尾数和指数。标题信息的其它部分包括在该关键值数据的每一成分的量化值中的最大和最小值中的一个最大值和一个最小值。具体地说，nXQMinOfMax表示在每一个顶点的x成份的量化值中的最大值当中的一个最小值。

图20G和20H是表示程序代码的实例的示意图，借助该程序代码实现根据本发明优选实施例的一个DPCM模式解码器。参考图20G和20H，nDPCMMode是表示每一顶点的每一个成份x、y和z的一个DPCM模式的一个整数数组。nDPCMMode可以具有值：1(一个时间DPCM操作)、2(一个空间DPCM操作)或3(一个时空操作)。

bSelFlag是用于每一顶点的每一成分的选择标志。只有针对bSelFlag被设置为'真'的具有成分的顶点才使用字典编码器340编码。selectionFlagContext是用于读出bSelFlag的一个上下文。

nKVACodingBit是一个整数数组，表示为了编码每一顶点的每一成分所需的比特。aqpXContext是用于读出nKVACodingBit的一个上下文。

nRefVertexis是表示用于全部顶点的一个参考顶点的索引的整数数组。refContext是用于读出nRefVertex的一个上下文。nQMin是一个整数数组，表示在每一顶点的每一成分的量化值中的最小值。qMinContext是用于读出nQMin的一个上下文。qMinSignContext是用于读取nQMin的符号的一个上下文。

nQMax是一个整数数组，表示在每一顶点的每一成分的量化值中的最大值。qMaxContext是用于读出nQMax的一个上下文。qMaxSignContext是用于读取nQMax的符号的一个上下文。

图20I是一个类CoordIDPCMMode的示意图，用于根据本发明优选实施例而解码一个DPCM模式。bAddressOfDPCMMode表示在DPCM字典表格中的DPCM字典符号的用法，其中每一个包括用于每一成份的DPCM模式。存在DPCM模式的三个类型：T、S和T+S，并且一个顶点包括三个成分。因此如上面表3所示，存在27个字典符号指示所有可能的DPCM模式的组合。dpcmModeDicAddressContext是用于读出bAddressOfDPCMMode的一个上下文。

bDPCMIndex指示已经用于每一个顶点的一个DPCM符号。dpcmModeDIcIndexContext是用于读出dpcmModeDicIndexContext的一个上下文。

图20J是一个类CoordIKeyValueDic的示意图，该类用于根据本发明优选实施例而解码一个字典编码模式。图20J中，nDicModeSelect指示在一个字典编码处理过程中已经使用了哪个编码模式。当nDicModeSelect设置为1时，意味着该编码模式是一个增量模式。另一方面，当nDicModeSelect是0，则意味着该字典编码模式是一个出现模式。

图20K是表示一个类CoordIIncrementalMode的示意图，通过该类实现根据本发明优选实施例的一个增量模式解码方法。图20K中，bAddress表示量化关键值数据的递增字典符号的用法。增量字典表格中的符号数目是$2^{(nKVCodingBit+1)}-1$。dicAddressContext是用于读出bAddress的一个上下文。

nTrueOne是指示索引数据是否已经反向的一个值。如果nTrueOne设置为1，则在一个位置索引中的‘1’值被编译为指示一个符号的位置的真值。如果nTrueOne设置为0，则在该位置索引中的‘0’值被编译为指示该符号的位置的真值。

bAddrIndex 指示哪个增量字典符号已经被用于每一顶点的每一成分。dicIndexContext 是用于读出 bAddrIndex 的一个上下文。

图 20L 是表示一个类 CoordIOccurrenceMode 的示意图，通过该类实现根据本发明优选实施例的一个出现模式解码方法。图 20L 中，nQKV 是包括对应于量化关键值数据的发生字典符号的一个整数数组。kvXContext、kvYContext 和 kvZContext 是用于读出 nQKV 的上下文。kvSignContext 是用于读取 nQKV 的一个符号的一个上下文。

bSoleKV 表示一个符号是否仅出现一次。当一个预定符号仅出现一次时，bSoleKV 设置为 1。dicSoleKVContext 是用于读出 bSoleKV 的一个上下文。bDicIndex 指示哪个发生字典符号已经被用于每一顶点的每一成分。dicIndexContext 是用于读出 bDicIndex 的一个上下文。

本发明能够实现为写在计算机可读取记录介质上的计算机可读代码。其中，计算机可读记录介质包括能够由计算机系统读出的任何种类的记录介质。例如，该计算机可读记录介质可以包括 ROM、RAM、CD-ROM、磁带、软盘、光数据存储器、载波(通过互联网络发送)等。该计算机可读记录介质能够经过网络连接与计算机系统分散配置，并且一个计算机能够以分散的方法读出该记录介质。

由于根据本发明的用于编码坐标内插符的装置包括一个关键字数据编码器，该关键字数据编码器利用关键字数据单调增加的事实对关键字数据进行编码，所以该装置能够以高效率编码动画关键字数据。此外，由于根据本发明的用于编码坐标内插符的装置包括一个关键值数据编码器，其考虑了在关键值数据之间的时间相关性以及在关键值数据之间的空间相关而编码关键值数据，所以该装置能够以少量的数据来恢复高品质的动画。

虽然已经参照几个优选实施例具体地展示和描述了本发明，但是本领域技术人员将理解，在不背离由所附的权利要求书定义的本发明的精神和范围的条件下可以进行各种形式和细节上的改变。

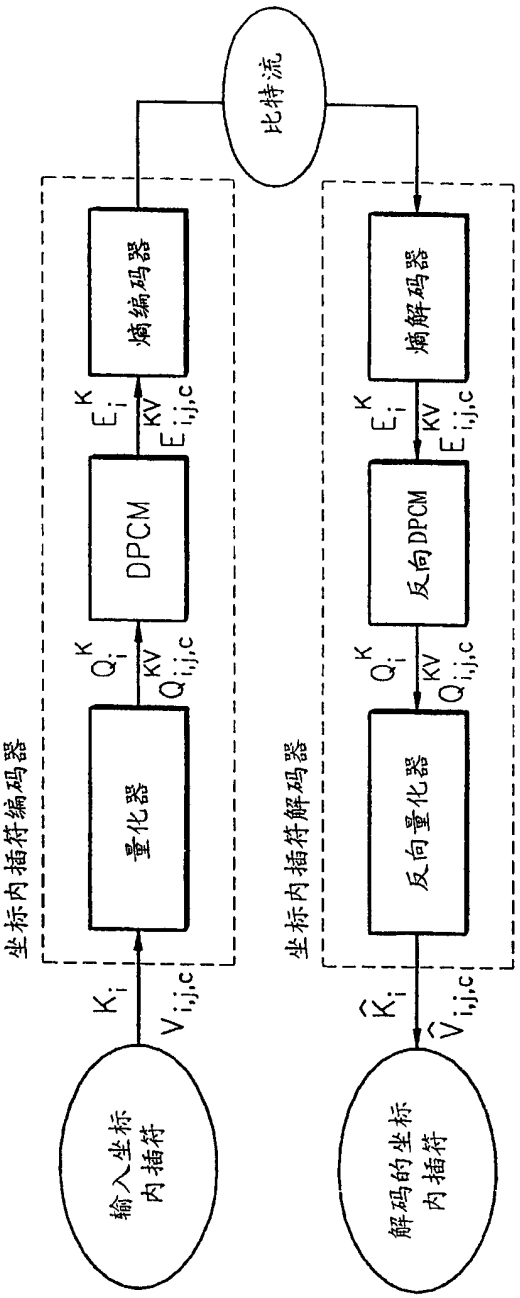

比特流
坐标内插符编码器
量化器
DPCM
熵编码器
输入坐标
内插符
坐标内插符解码器
反向量化器
反向DPCM
熵解码器
解码的坐标
内插符

图 1

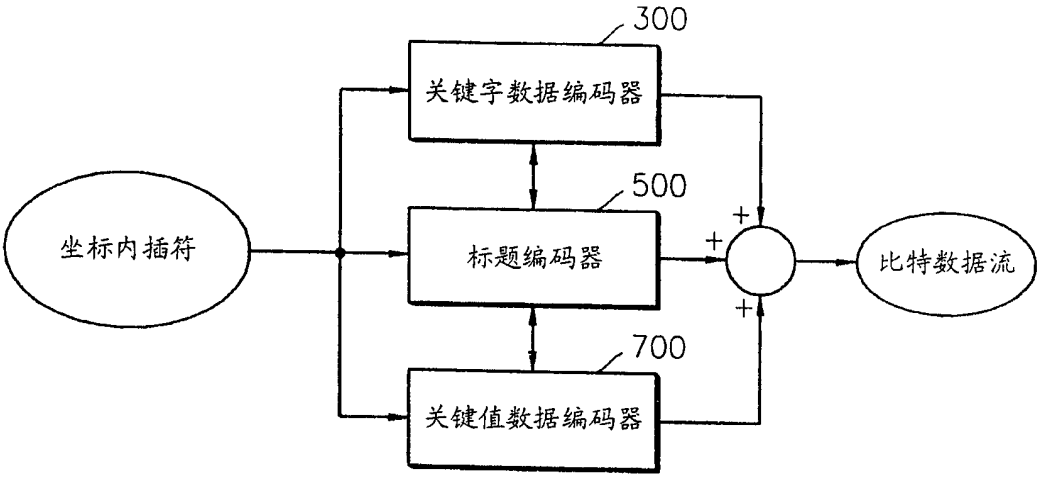


图 2

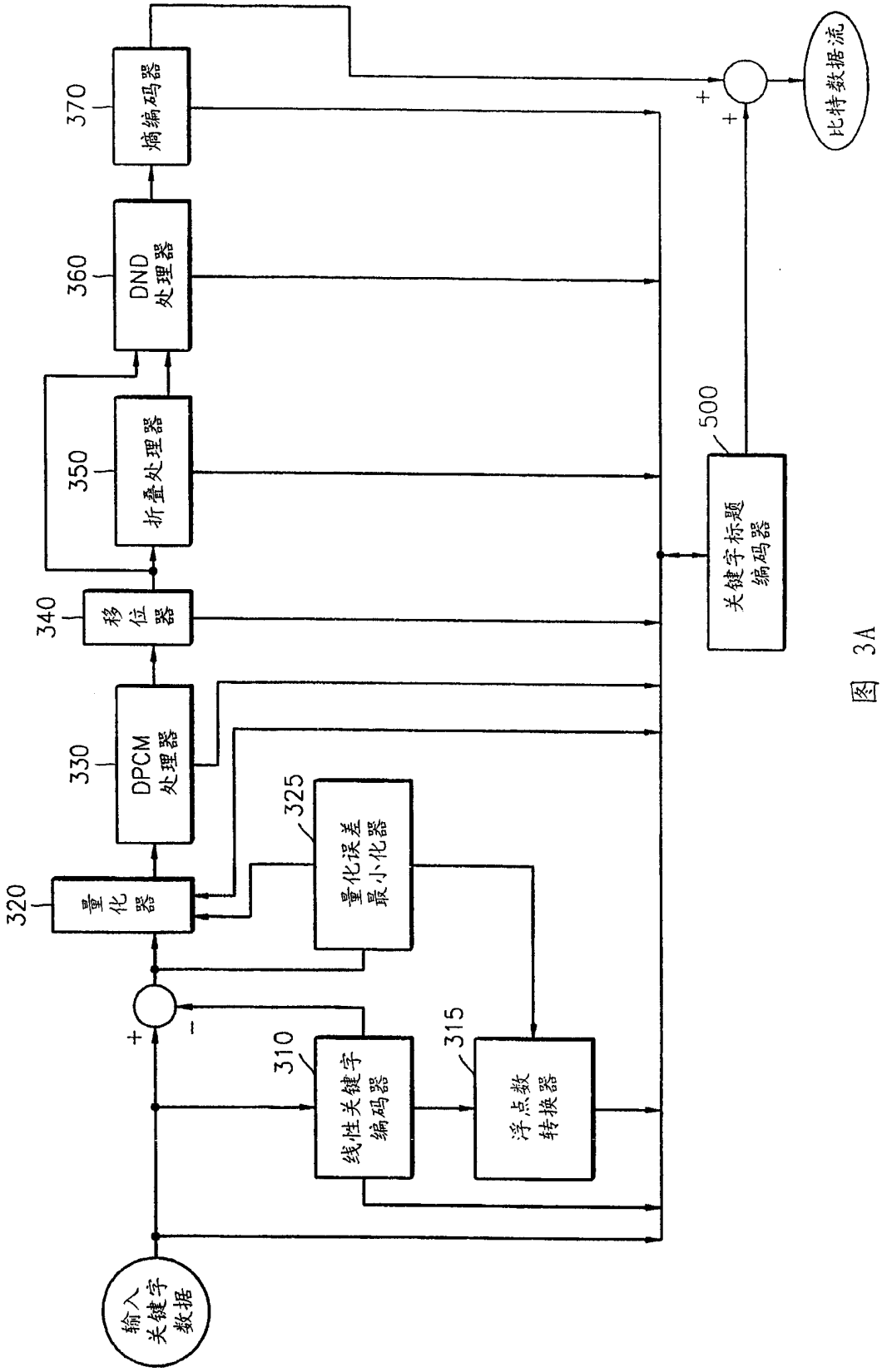

输入关键字数据
320
量化器
330
DPCM处理器
340
移位器
350
折叠处理器
360
DND处理器
370
熵编码器
310
线性关键字编码器
315
浮点数转换器
325
量化误差最小化器
500
关键字标题编码器
比特数据流
+
-
+
+

图 3A

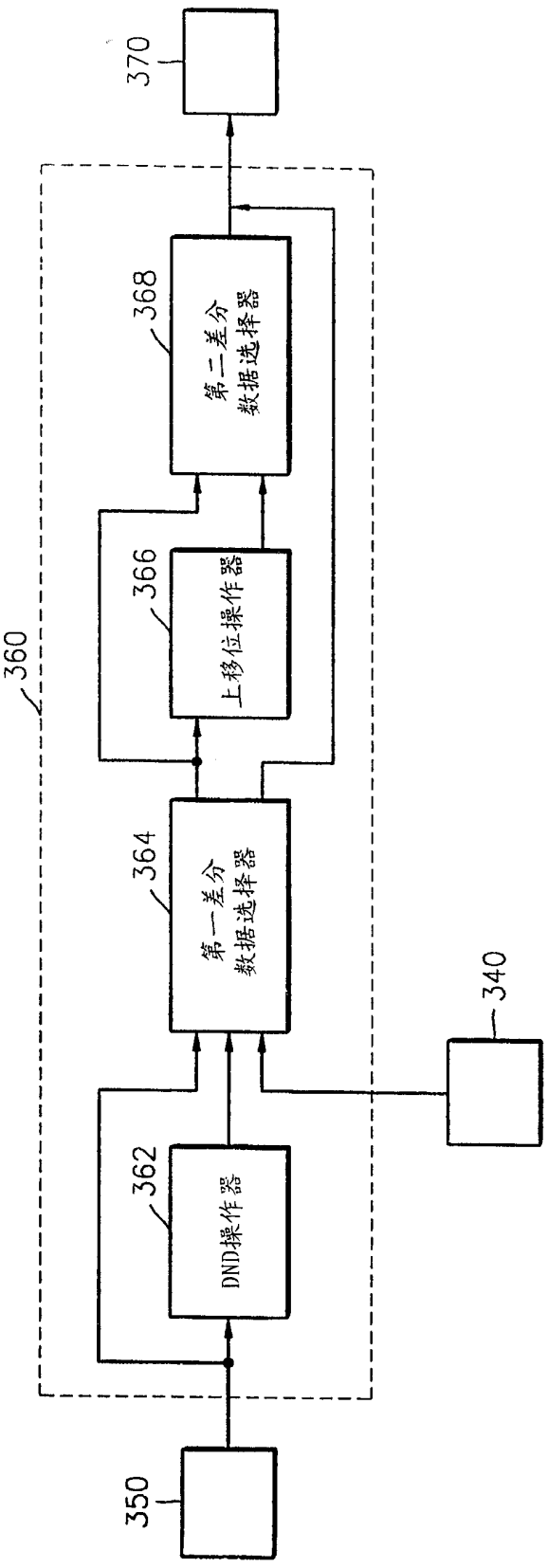

370
368
第二差分
数据选择器
366
上移位操作器
360
364
第一差分
数据选择器
340
362
DND操作器
350

图 3B

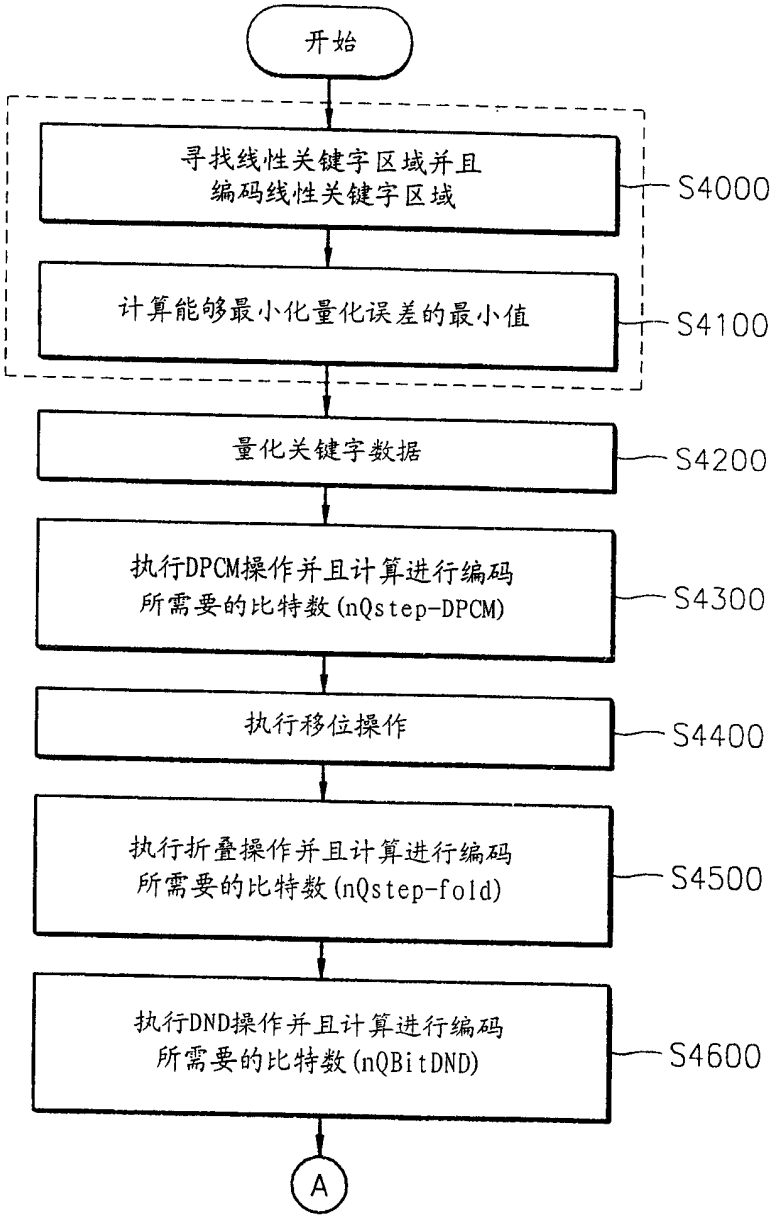

开始
寻找线性关键字区域并且
编码线性关键字区域
S4000
计算能够最小化量化误差的最小值
S4100
量化关键字数据
S4200
执行DPCM操作并且计算进行编码
所需要的比特数(nQstep-DPCM)
S4300
执行移位操作
S4400
执行折叠操作并且计算进行编码
所需要的比特数(nQstep-fold)
S4500
执行DND操作并且计算进行编码
所需要的比特数(nQBitDND)
S4600
A

图 4A

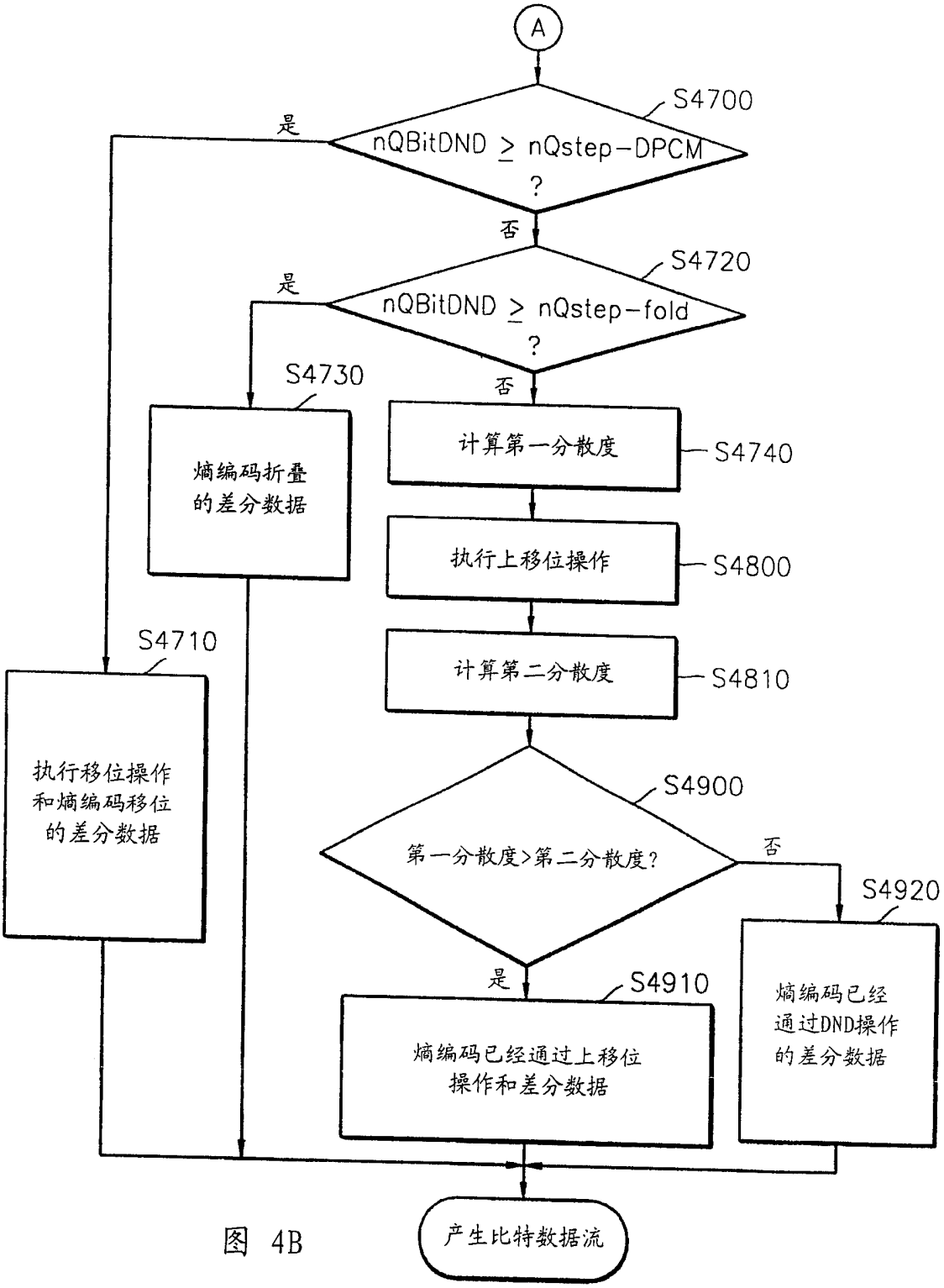

A
S4700
是
nQBitDND ≥ nQstep-DPCM
?
否
S4720
是
nQBitDND ≥ nQstep-fold
?
否
S4730
熵编码折叠
的差分数据
计算第一分散度
S4740
执行上移位操作
S4800
计算第二分散度
S4810
S4710
执行移位操作
和熵编码移位
的差分数据
S4900
第一分散度>第二分散度?
否
S4920
是
S4910
熵编码已经通过上移位
操作和差分数据
熵编码已经
通过DND操作
的差分数据
产生比特数据流

图 4B

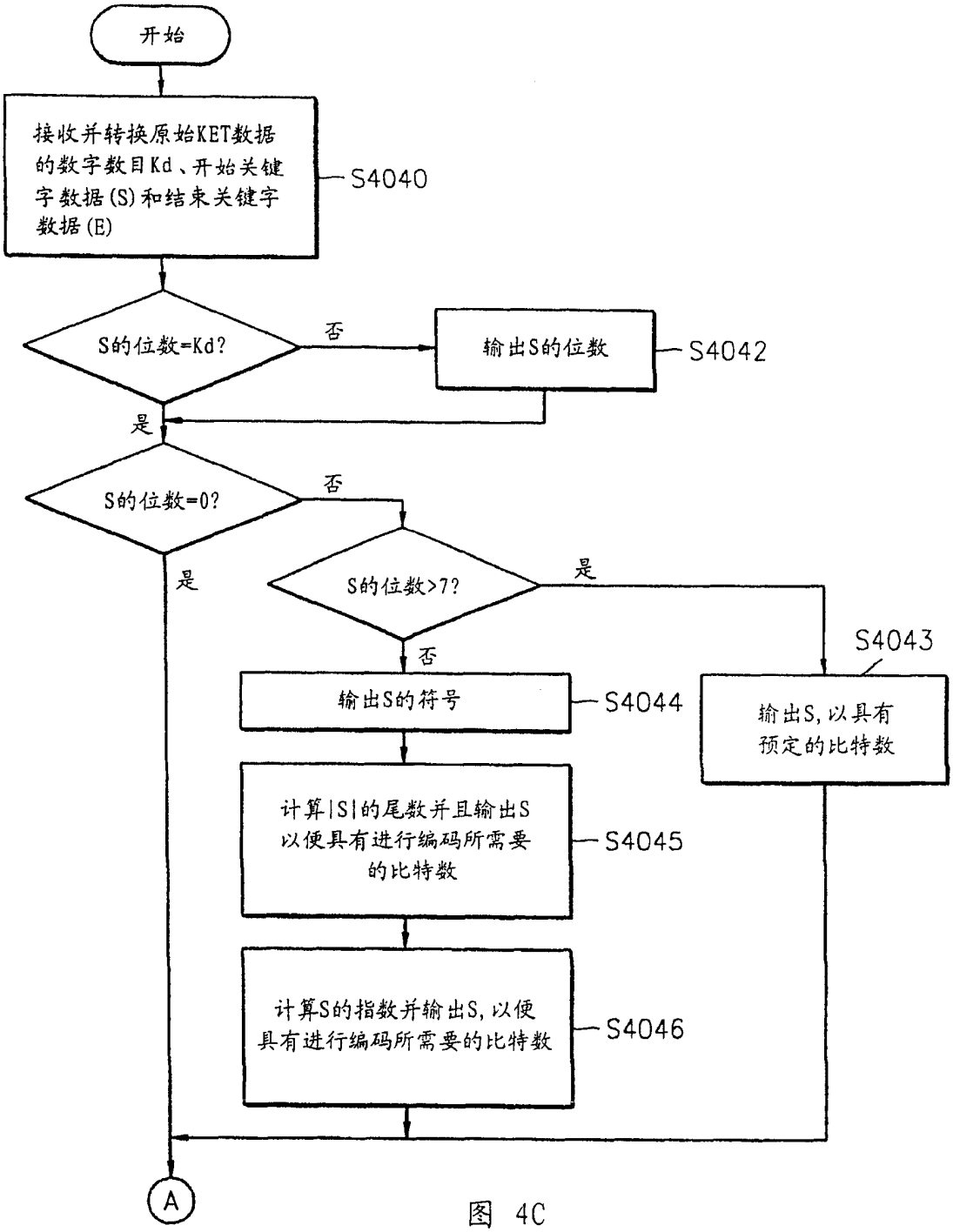

开始
接收并转换原始KET数据的数字数目Kd、开始关键字数据(S)和结束关键字数据(E)
S4040
S的位数=Kd?
否
输出S的位数
S4042
是
S的位数=0?
否
是
S的位数>7?
是
否
S4043
输出S的符号
S4044
输出S,以具有预定的比特数
计算|S|的尾数并且输出S以便具有进行编码所需要的比特数
S4045
计算S的指数并输出S,以便具有进行编码所需要的比特数
S4046
A

图 4C

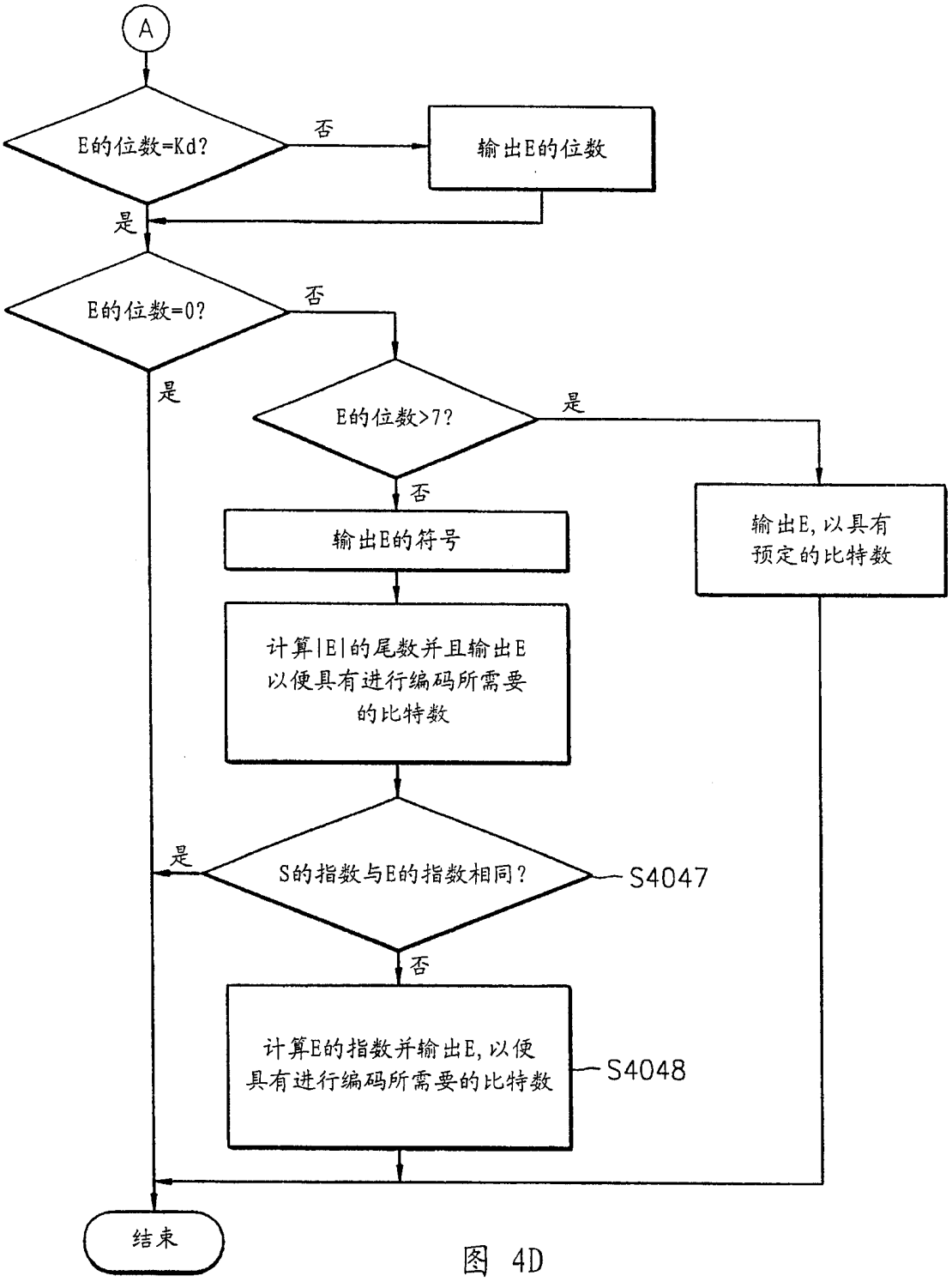

A
E的位数=Kd?
否
输出E的位数
是
E的位数=0?
否
是
E的位数>7?
是
否
输出E的符号
输出E,以具有
预定的比特数
计算|E|的尾数并且输出E
以便具有进行编码所需要
的比特数
是
S的指数与E的指数相同?
S4047
否
计算E的指数并输出E,以便
具有进行编码所需要的比特数
S4048
结束

图 4D

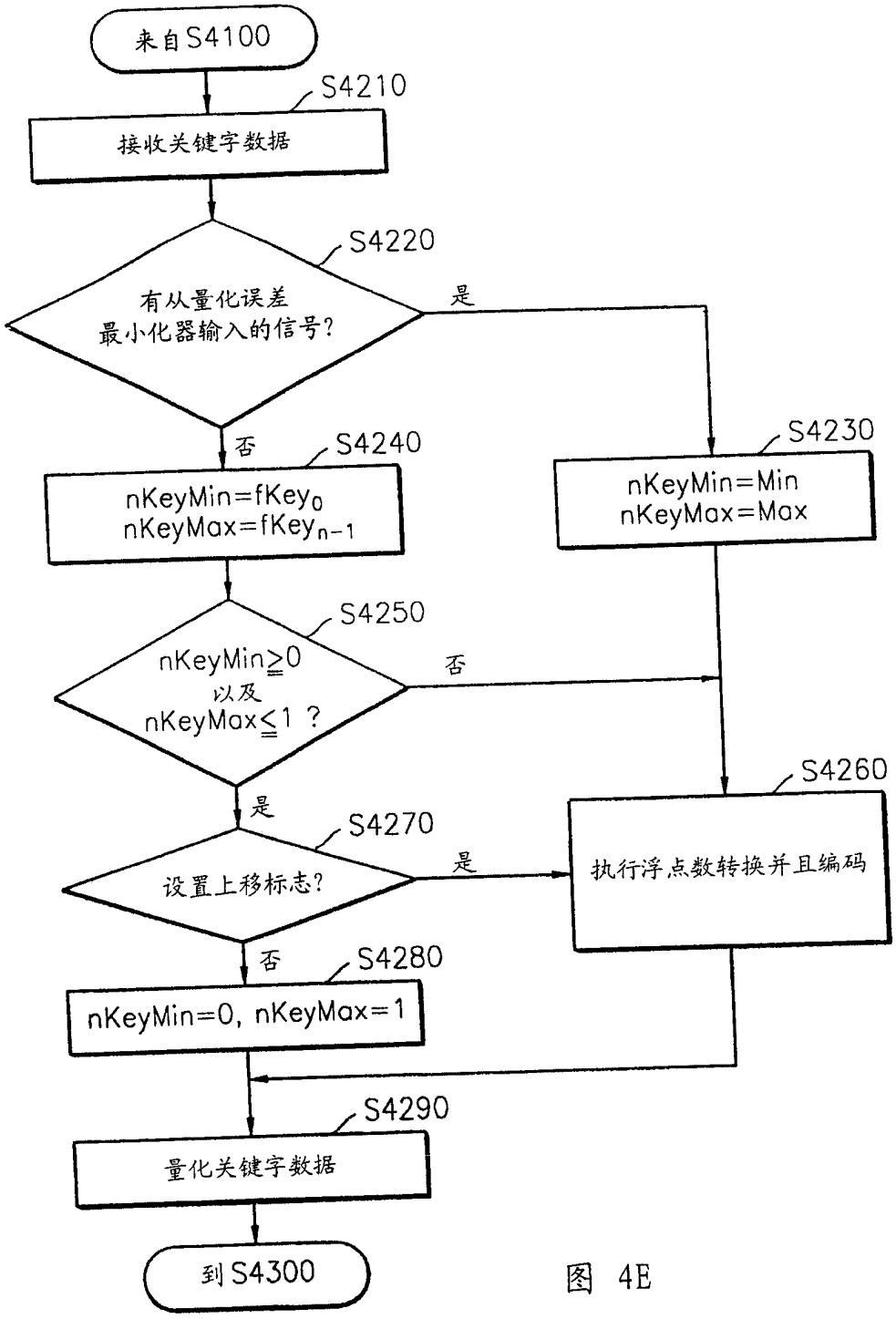

来自S4100
S4210
接收关键字数据
S4220
有从量化误差
最小化器输入的信号?
是
否
S4240
nKeyMin=fKey0
nKeyMax=fKeyn-1
S4230
nKeyMin=Min
nKeyMax=Max
S4250
nKeyMin≧0
以及
nKeyMax≦1 ?
否
是
S4260
S4270
设置上移标志?
是
执行浮点数转换并且编码
否
S4280
nKeyMin=0, nKeyMax=1
S4290
量化关键字数据
到S4300

图 4E

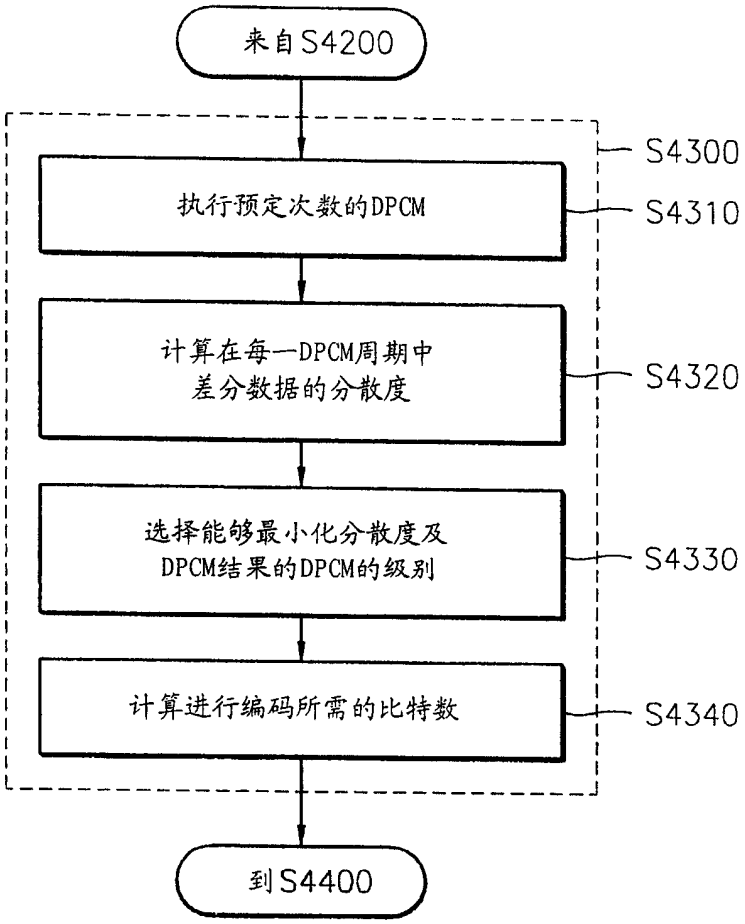

来自S4200
S4300
执行预定次数的DPCM
S4310
计算在每一DPCM周期中
差分数据的分散度
S4320
选择能够最小化分散度及
DPCM结果的DPCM的级别
S4330
计算进行编码所需的比特数
S4340
到S4400

图 4F

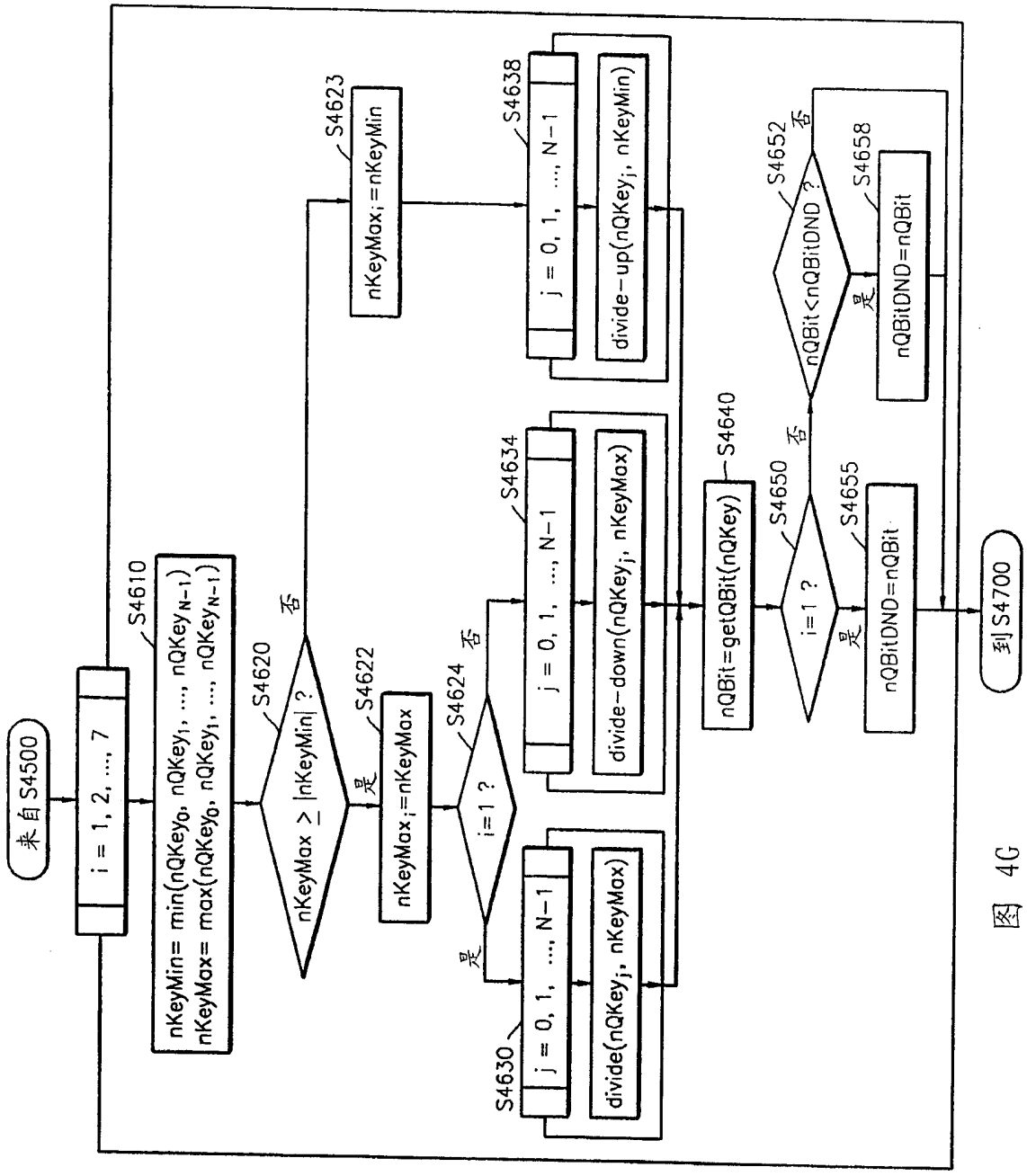

来自S4500
i = 1, 2, ..., 7
S4610
nKeyMin= min(nQKey0, nQKey1, ..., nQKeyN-1)
nKeyMax= max(nKey0, nQKey1, ..., nQKeyN-1)
S4620
nKeyMax ≥ |nKeyMin| ?
否
S4623
nKeyMaxi=nKeyMin
S4638
j = 0, 1, ..., N-1
divide-up(nQKeyj, nKeyMin)
是
S4622
nKeyMaxi=nKeyMax
S4624
i=1 ?
否
S4634
j = 0, 1, ..., N-1
divide-down(nQKeyj, nKeyMax)
是
S4630
j = 0, 1, ..., N-1
divide(nQKeyj, nKeyMax)
S4640
nQBit=getQBit(nQKey)
S4650
i=1 ?
否
S4652
nQBit<nQBitDND ?
否
S4658
nQBitDND=nQBit
是
S4655
nQBitDND=nQBit
是
到S4700

图 4G

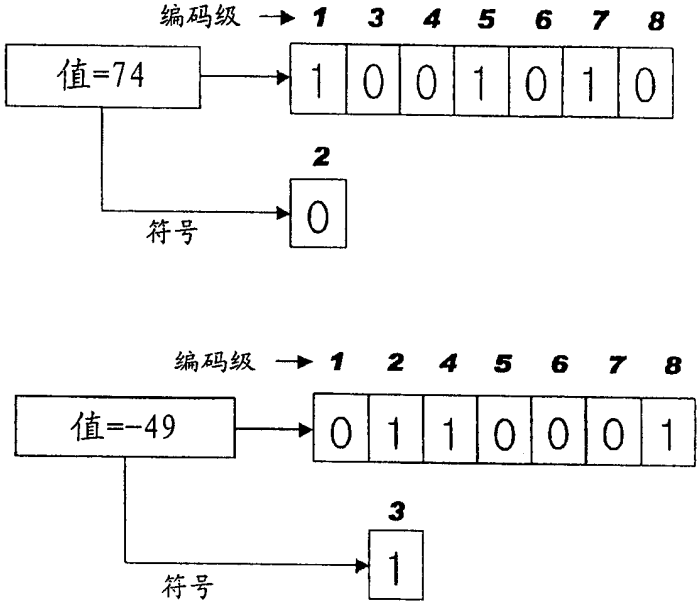


图 5

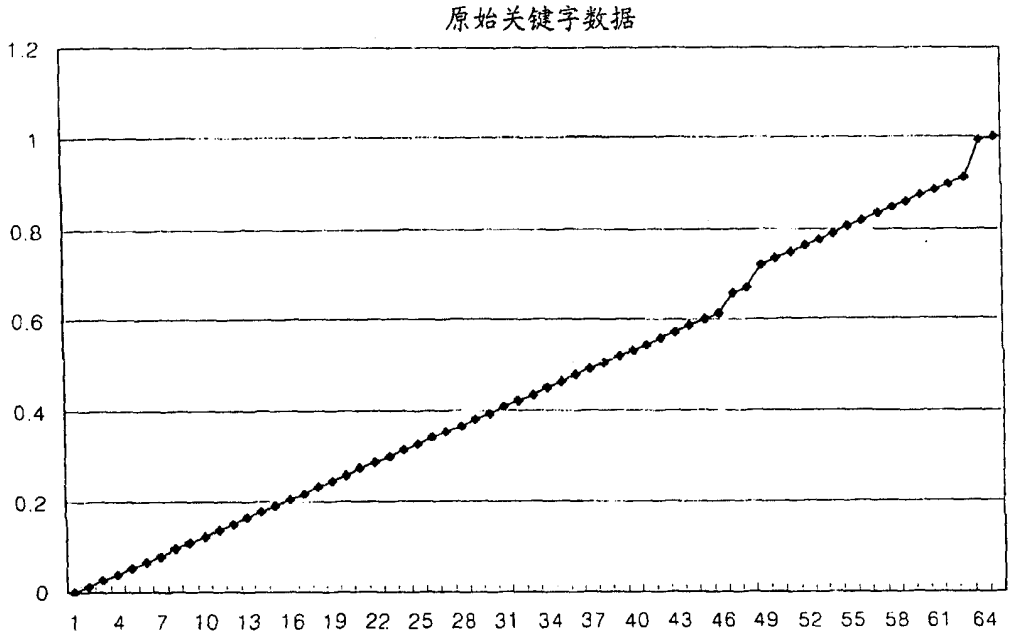

原始关键字数据
1.2
1
0.8
0.6
0.4
0.2
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6A

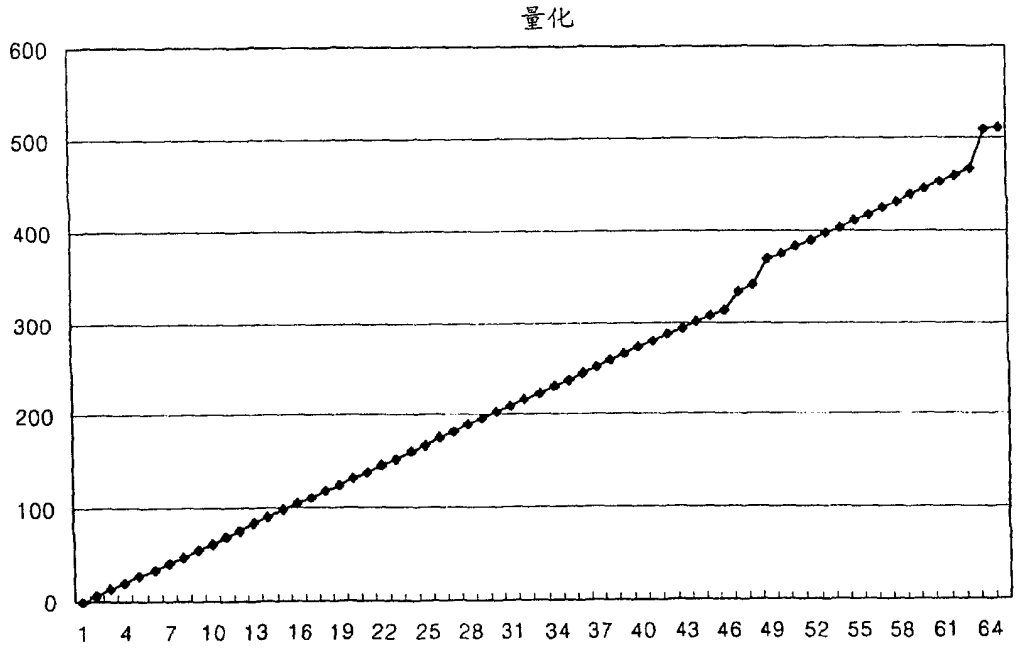

量化
600
500
400
300
200
100
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6B

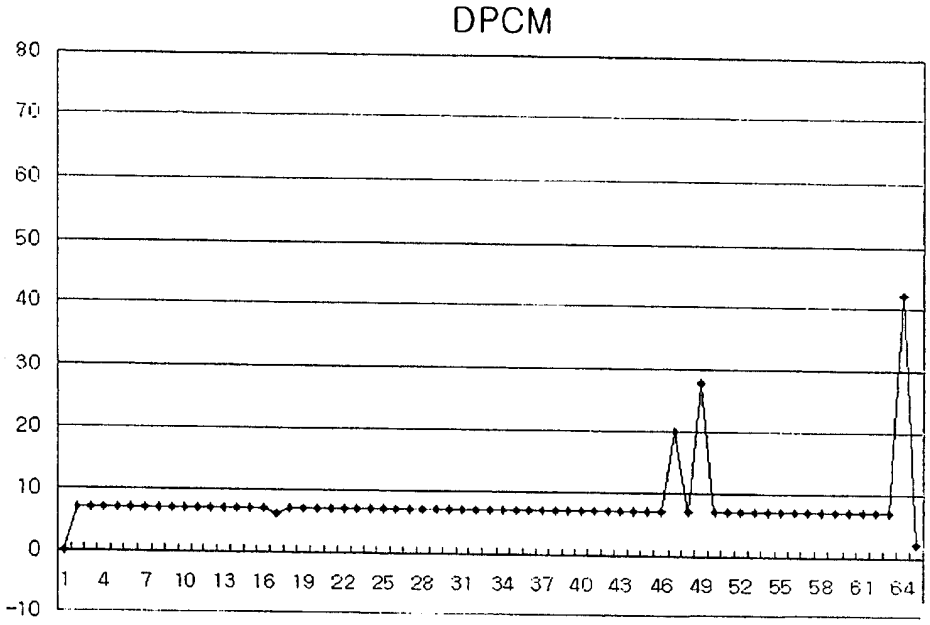

DPCM
80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6C

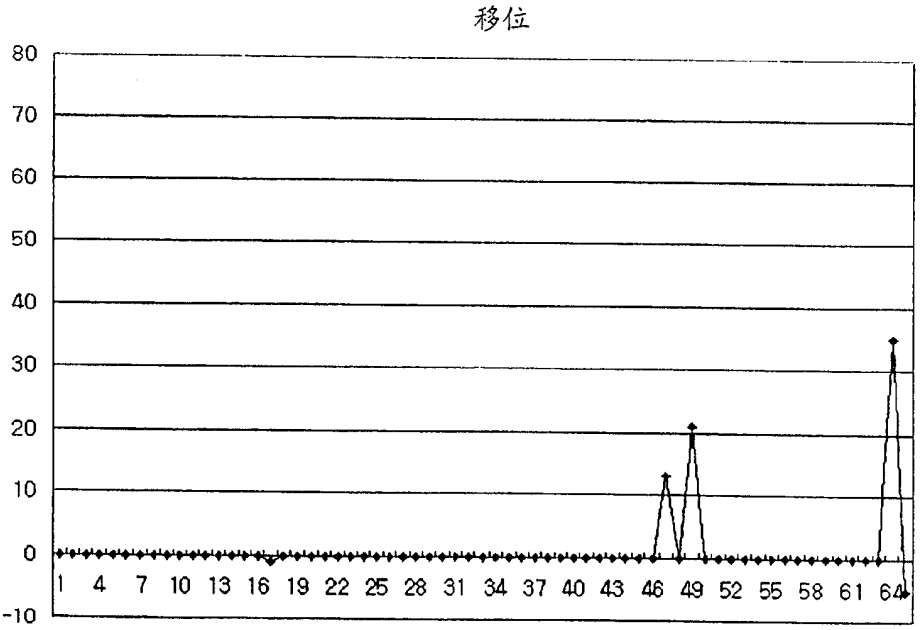

移位
80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6D

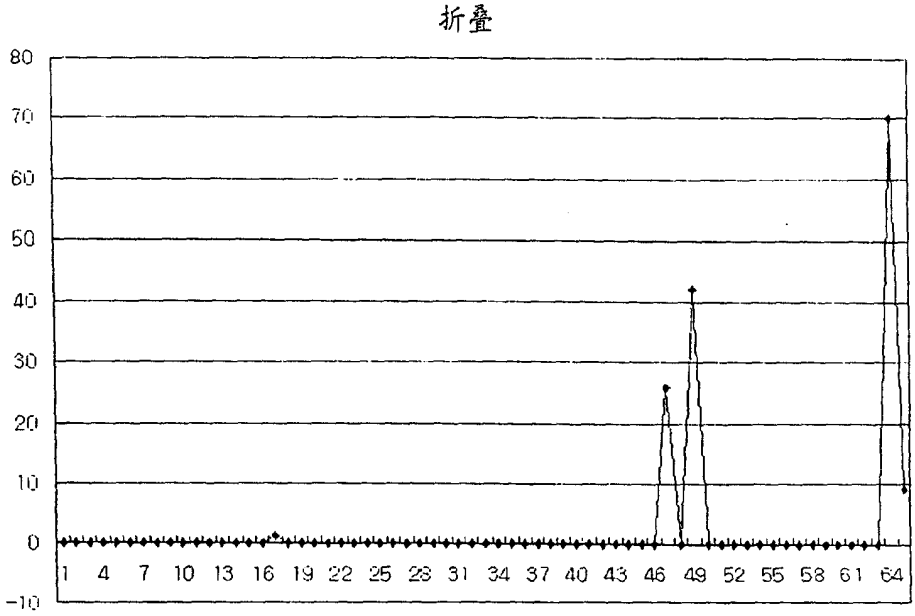

折叠
80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6E

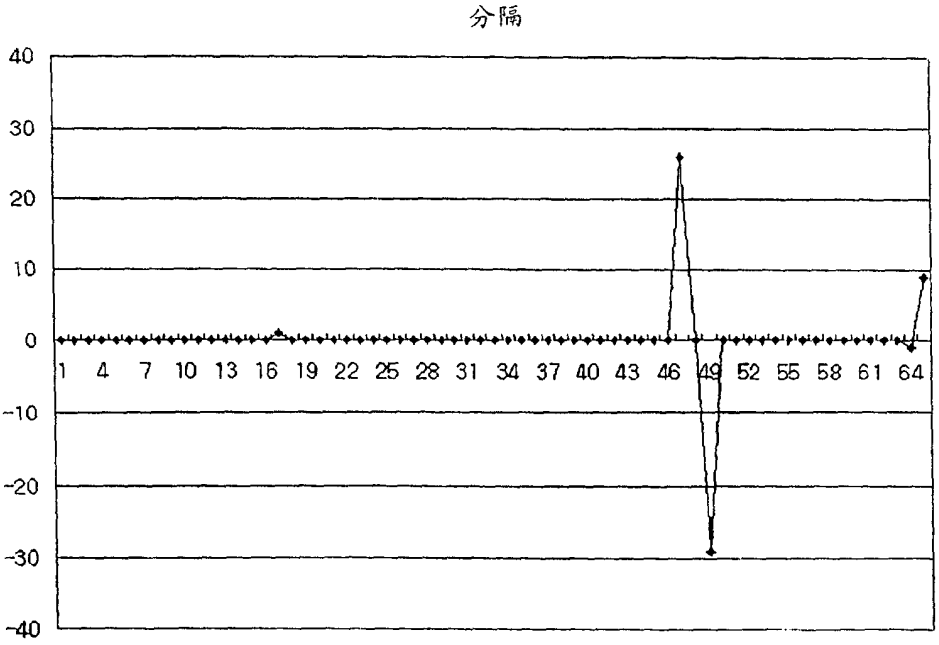

分隔
40
30
20
10
0
-10
-20
-30
-40
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6F

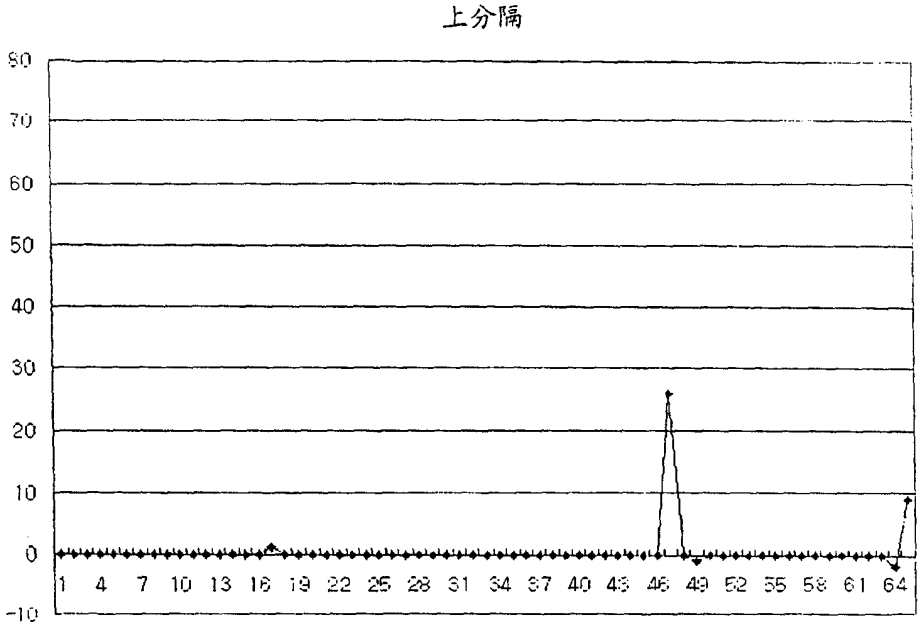

上分隔
80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6G

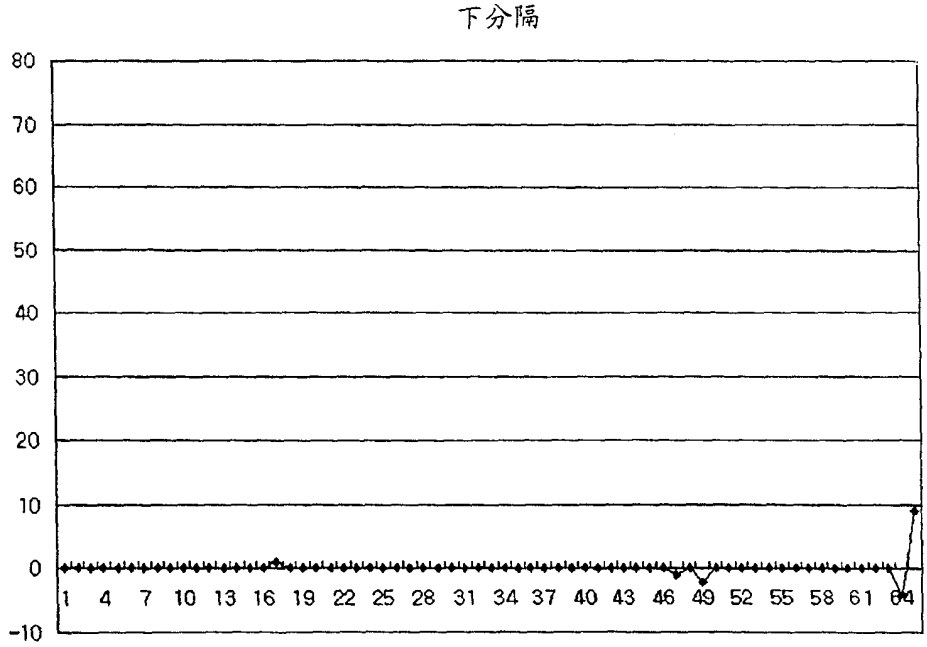

下分隔
80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6H

上移

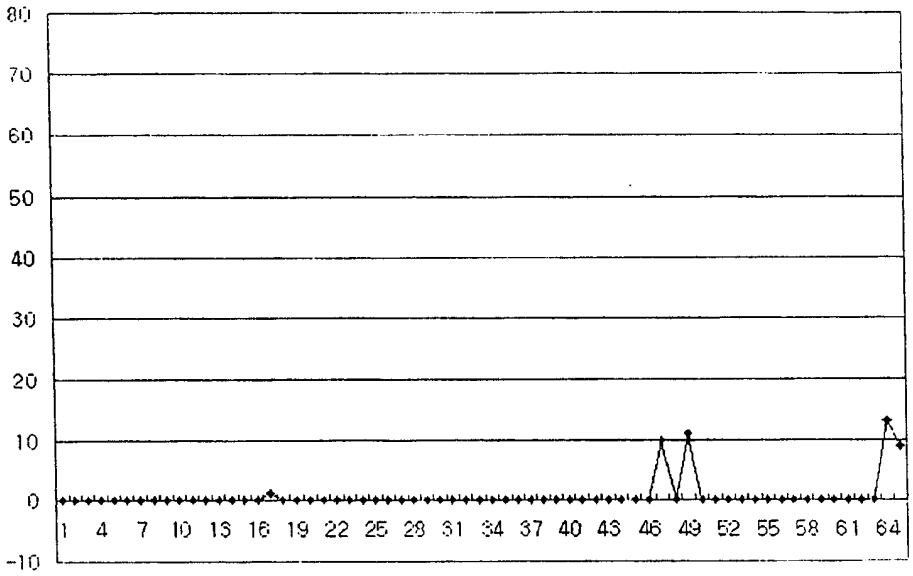

80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6I

最终

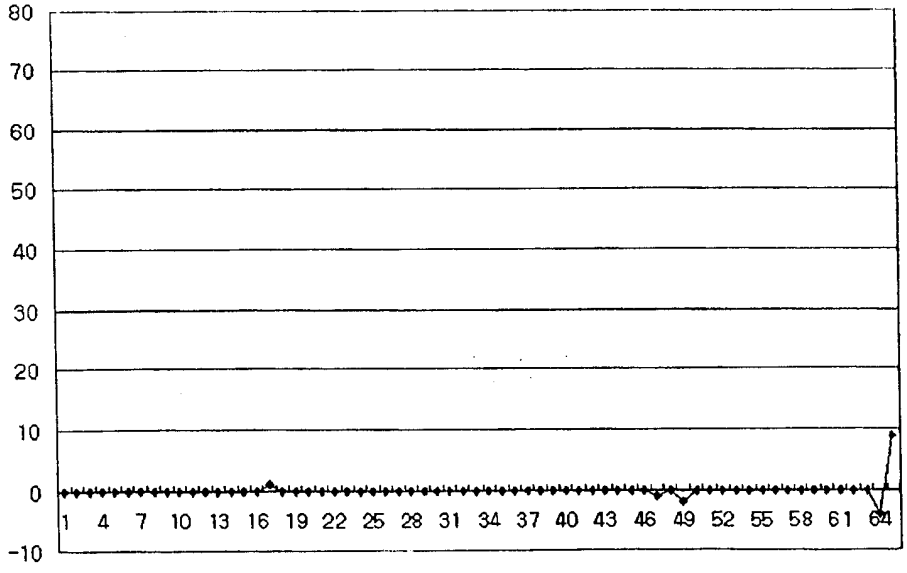

80
70
60
50
40
30
20
10
0
-10
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

图 6J

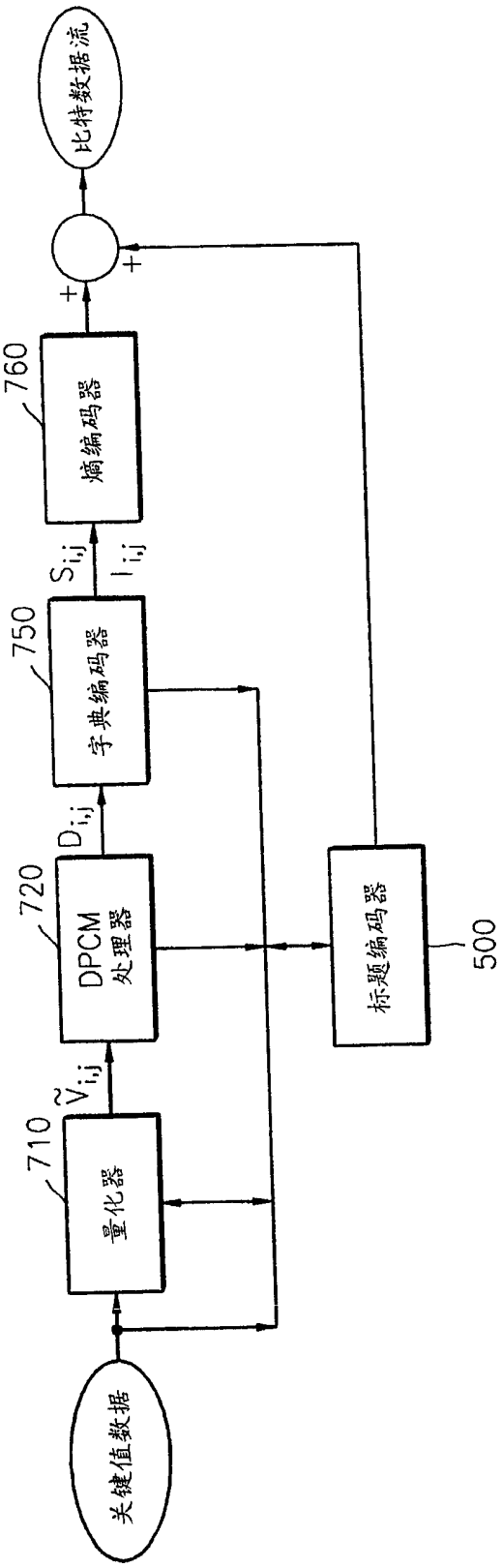

比特数据流
+
+
760
熵编码器
$S_{i,j}$
$l_{i,j}$
750
字典编码器
$D_{i,j}$
720
DPCM
处理器
$\tilde{V}_{i,j}$
710
量化器
关键值数据
标题编码器
500

图 7A

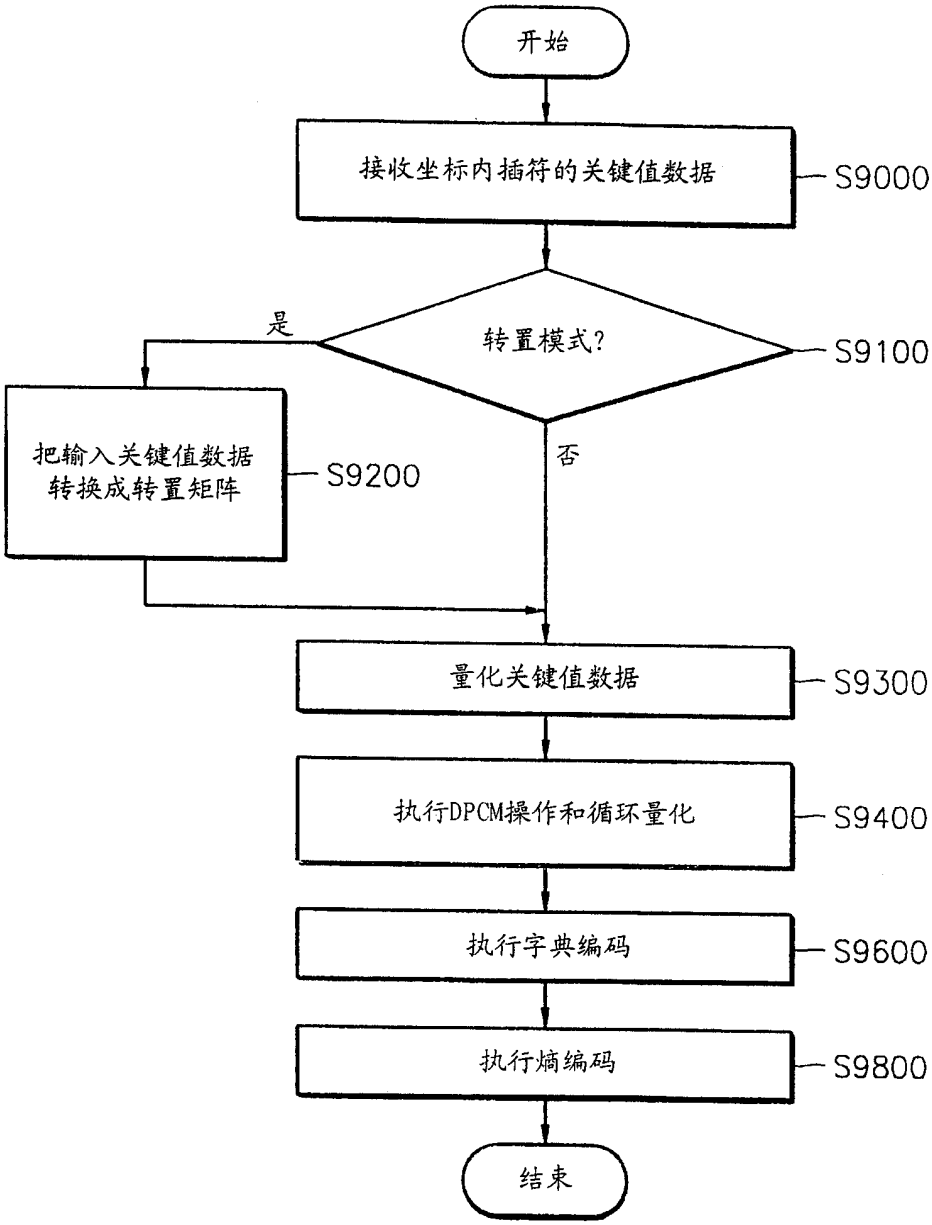

开始
接收坐标内插符的关键值数据
S9000
转置模式?
S9100
是
否
把输入关键值数据
转换成转置矩阵
S9200
量化关键值数据
S9300
执行DPCM操作和循环量化
S9400
执行字典编码
S9600
执行熵编码
S9800
结束

图 7B

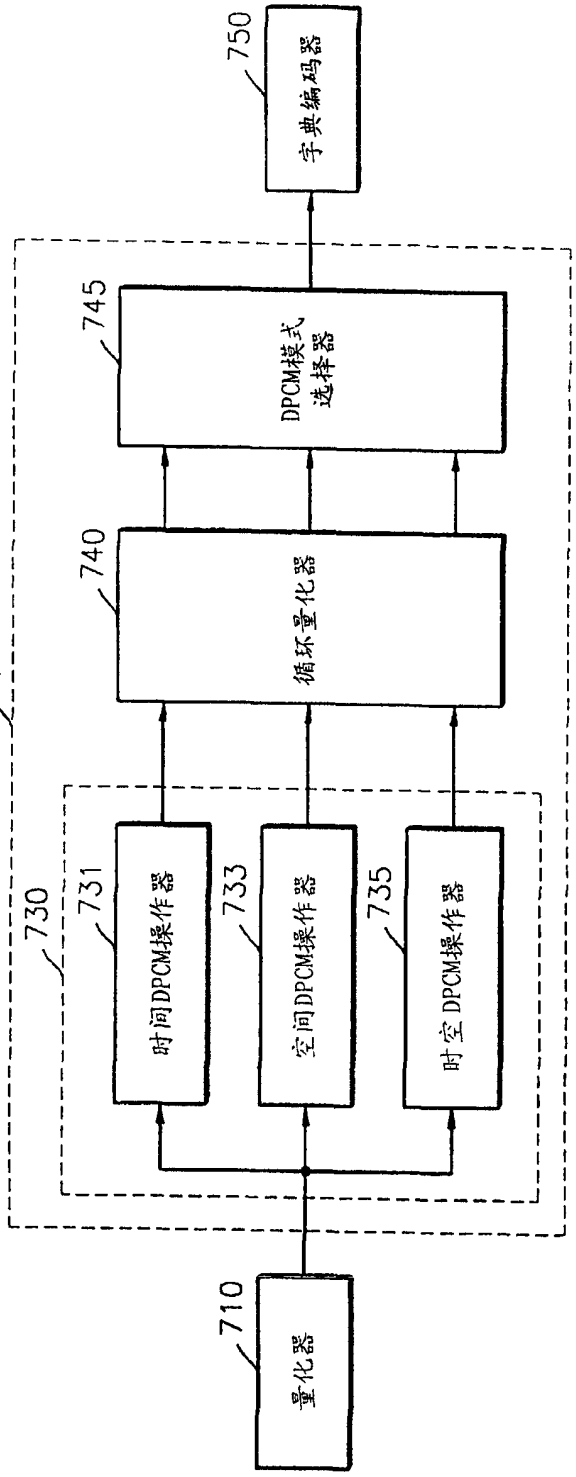

750
字典编码器
745
DPCM模式选择器
740
循环量化器
720
730
731
时间DPCM操作器
733
空间DPCM操作器
735
时空 DPCM操作器
710
量化器

图 8A

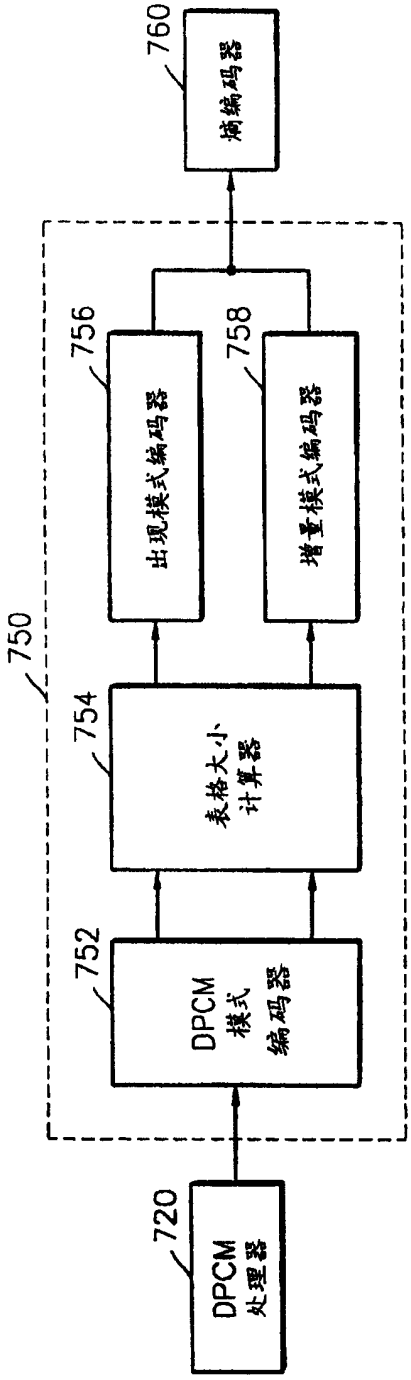

760
熵编码器
756
出现模式编码器
758
增量模式编码器
750
754
表格大小计算器
752
DPCM 模式 编码器
720
DPCM 处理器

图 8B

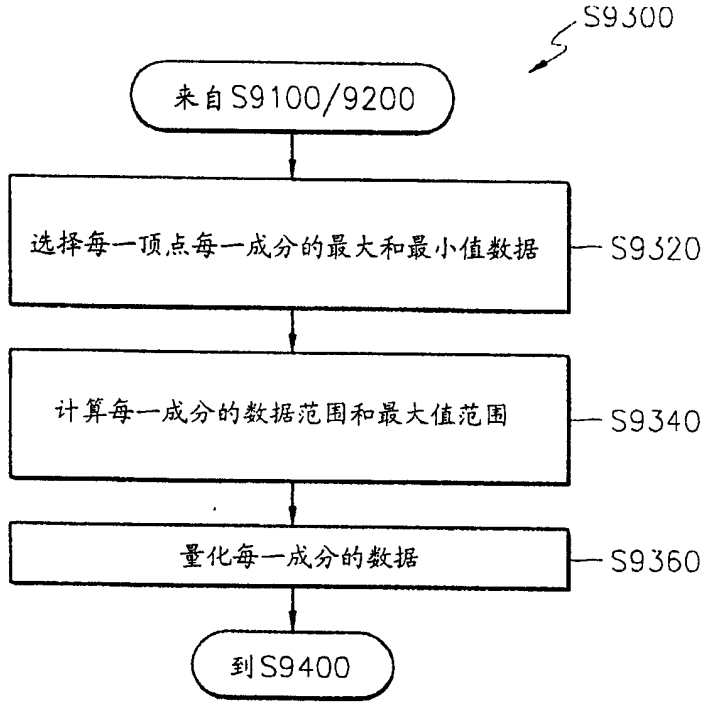

S9300
来自S9100/9200
选择每一顶点每一成分的最大和最小值数据
S9320
计算每一成分的数据范围和最大值范围
S9340
量化每一成分的数据
S9360
到S9400

图 9A

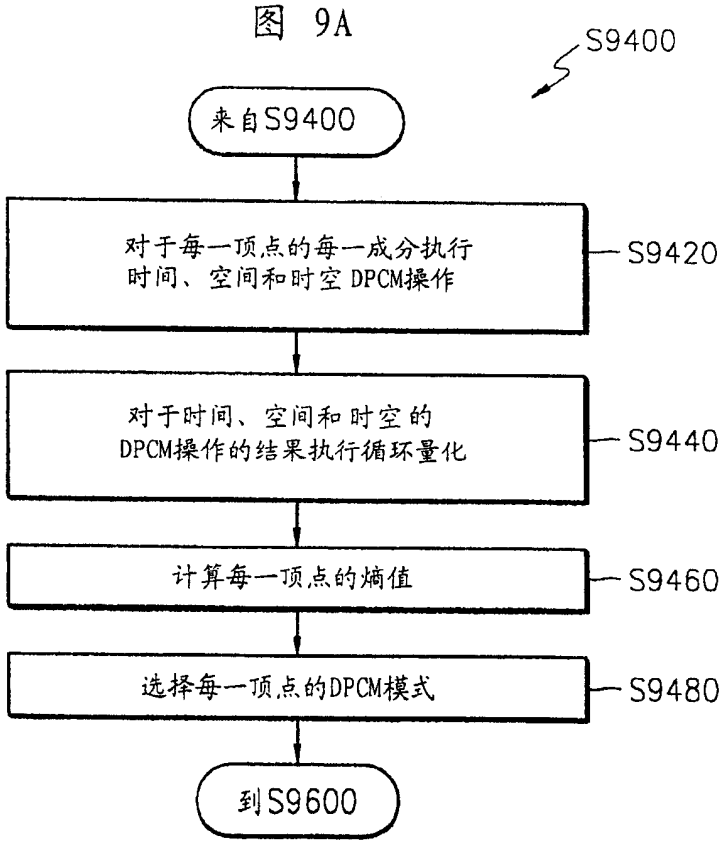

S9400
来自S9400
对于每一顶点的每一成分执行
时间、空间和时空 DPCM操作
S9420
对于时间、空间和时空的
DPCM操作的结果执行循环量化
S9440
计算每一顶点的熵值
S9460
选择每一顶点的DPCM模式
S9480
到S9600

图 9B

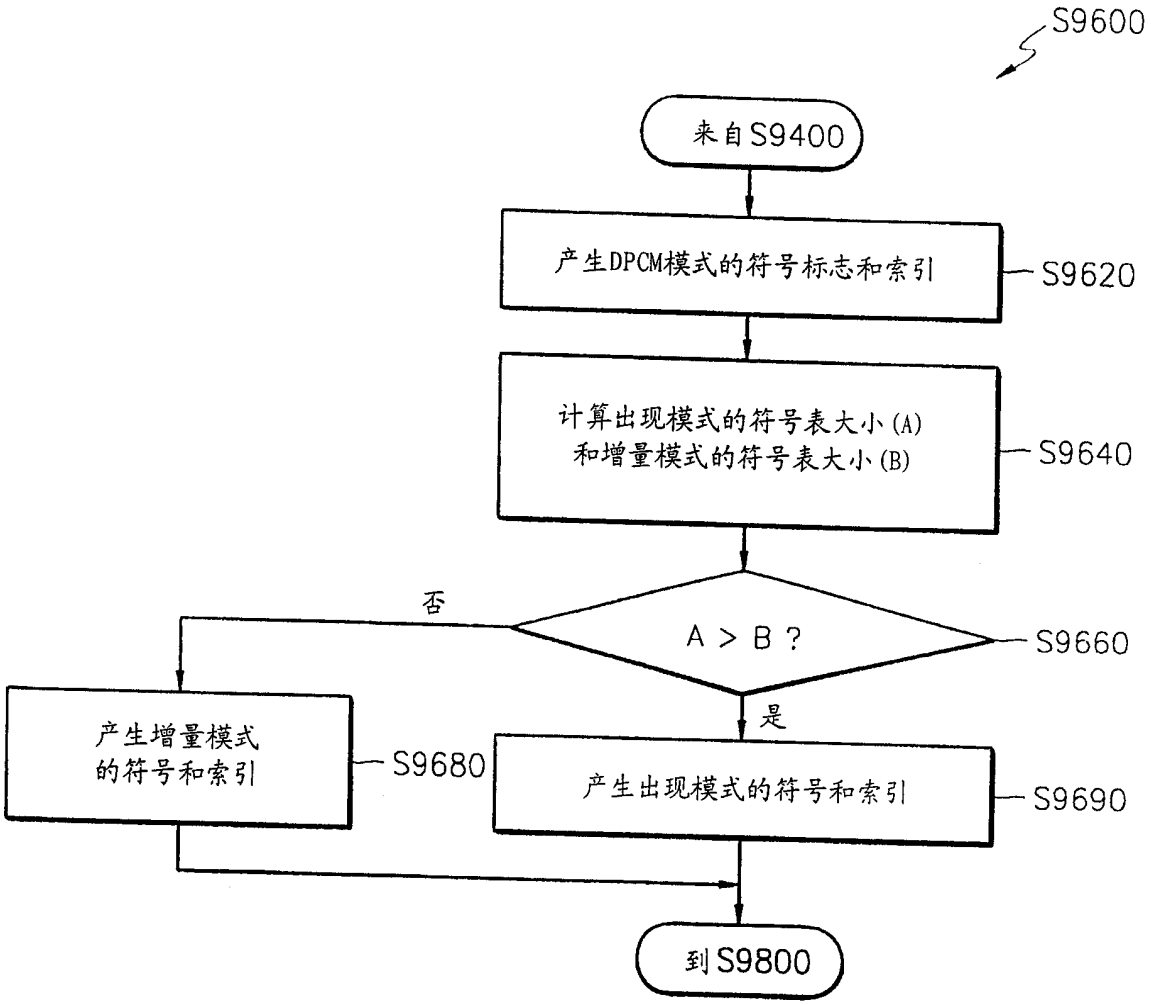

S9600
来自S9400
产生DPCM模式的符号标志和索引
S9620
计算出现模式的符号表大小(A)
和增量模式的符号表大小(B)
S9640
A > B ?
S9660
否
是
产生增量模式
的符号和索引
S9680
产生出现模式的符号和索引
S9690
到S9800

图 9C

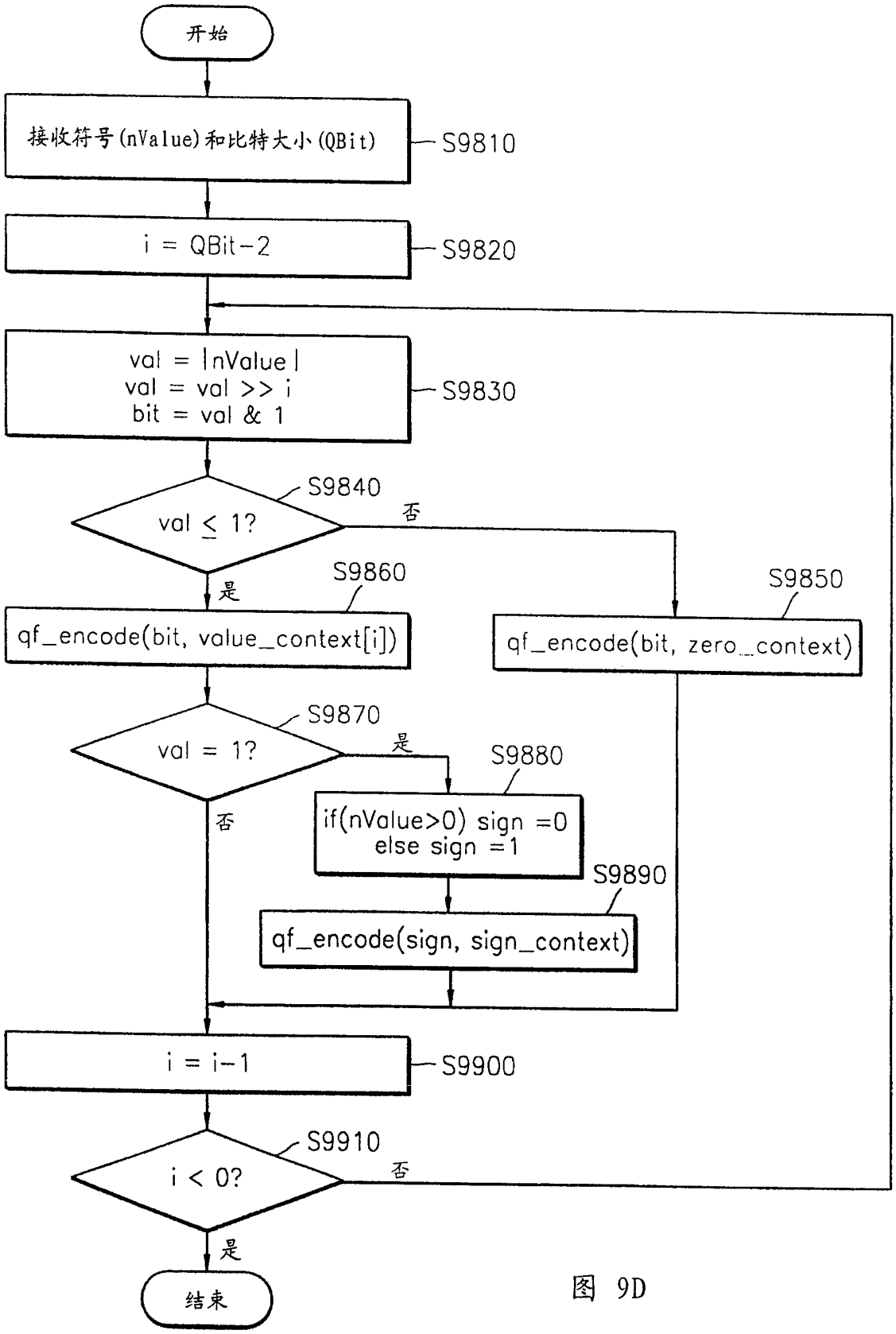

开始
接收符号(nValue)和比特大小(QBit)
S9810
i = QBit-2
S9820
val = |nValue|
val = val >> i
bit = val & 1
S9830
S9840
val ≤ 1?
否
是
S9860
qf_encode(bit, value_context[i])
S9850
qf_encode(bit, zero_context)
S9870
val = 1?
是
否
S9880
if(nValue>0) sign =0
else sign =1
S9890
qf_encode(sign, sign_context)
i = i-1
S9900
S9910
i < 0?
否
是
结束

图 9D

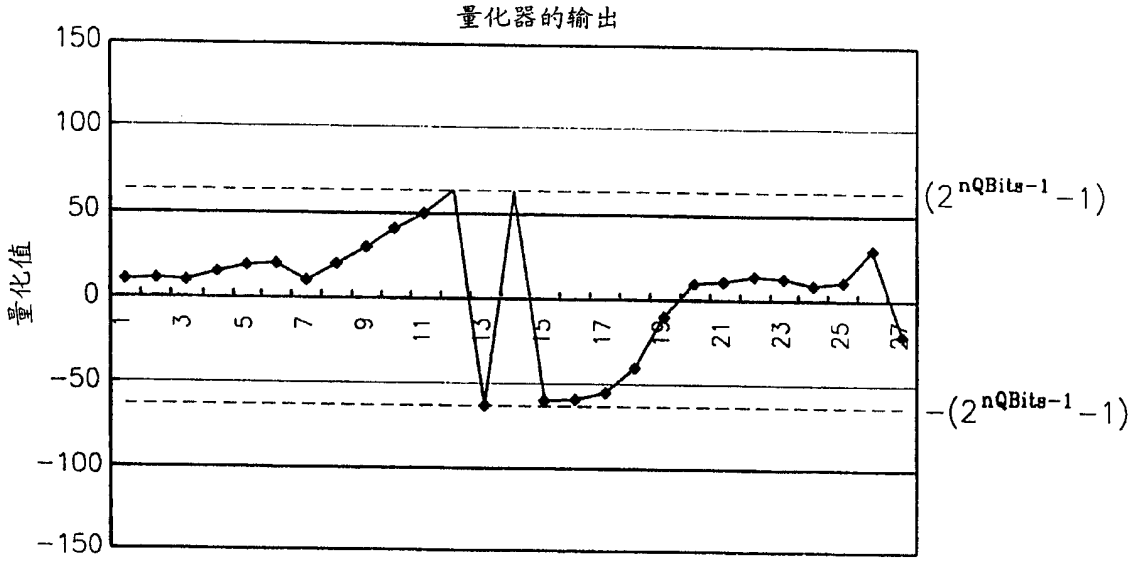

量化器的输出
150
100
50
0
-50
-100
-150
量化值
1
3
5
7
9
11
13
15
17
19
21
23
25
27
$(2^{nQBits-1}-1)$
$-(2^{nQBits-1}-1)$

图 10A

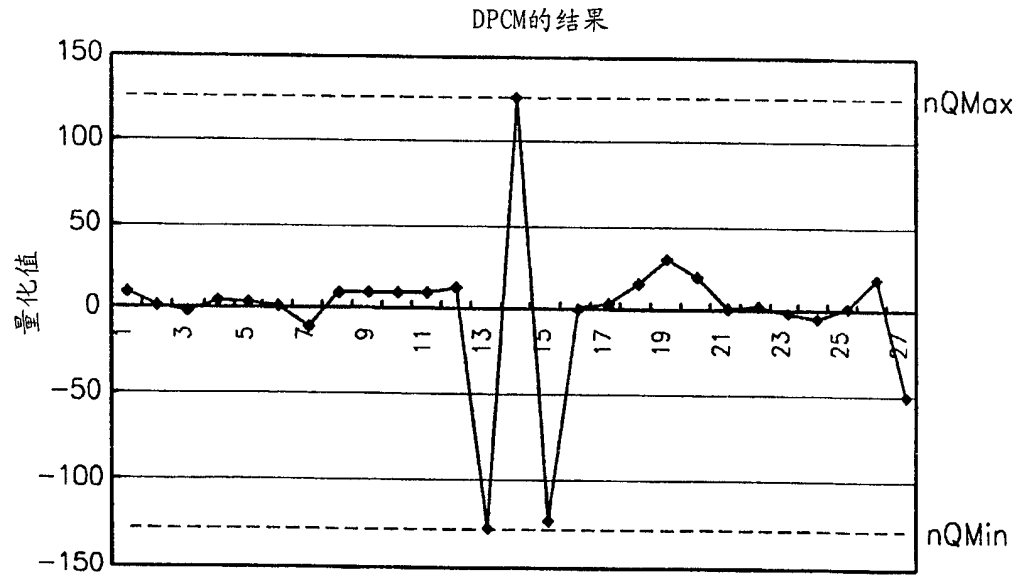

DPCM的结果
150
100
50
0
-50
-100
-150
量化值
1
3
5
7
9
11
13
15
17
19
21
23
25
27
nQMax
nQMin

图 10B

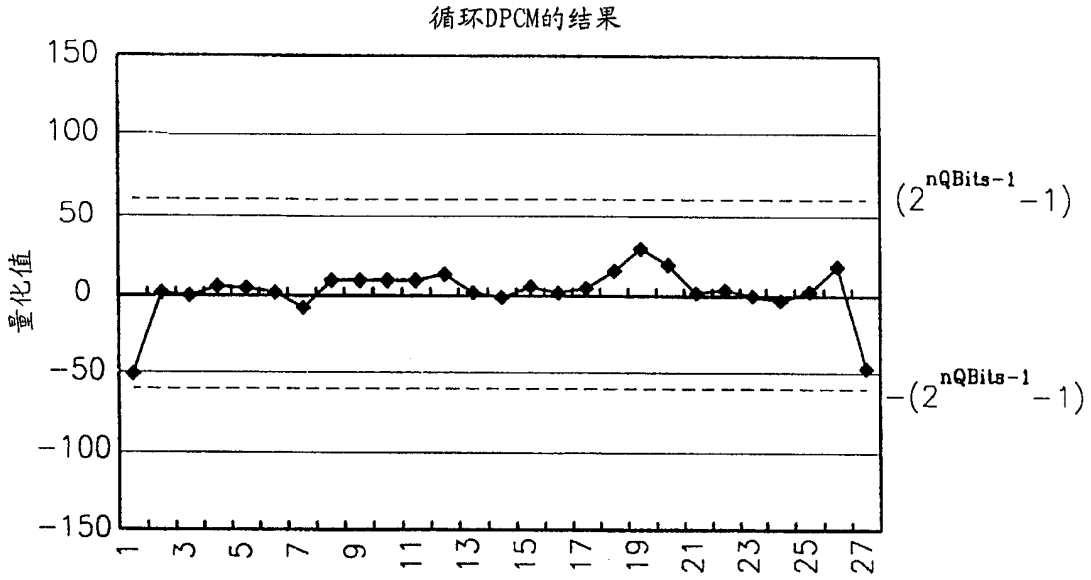

循环DPCM的结果
量化值
150
100
50
0
-50
-100
-150
1
3
5
7
9
11
13
15
17
19
21
23
25
27
$(2^{nQBits-1}-1)$
$-(2^{nQBits-1}-1)$

图 10C

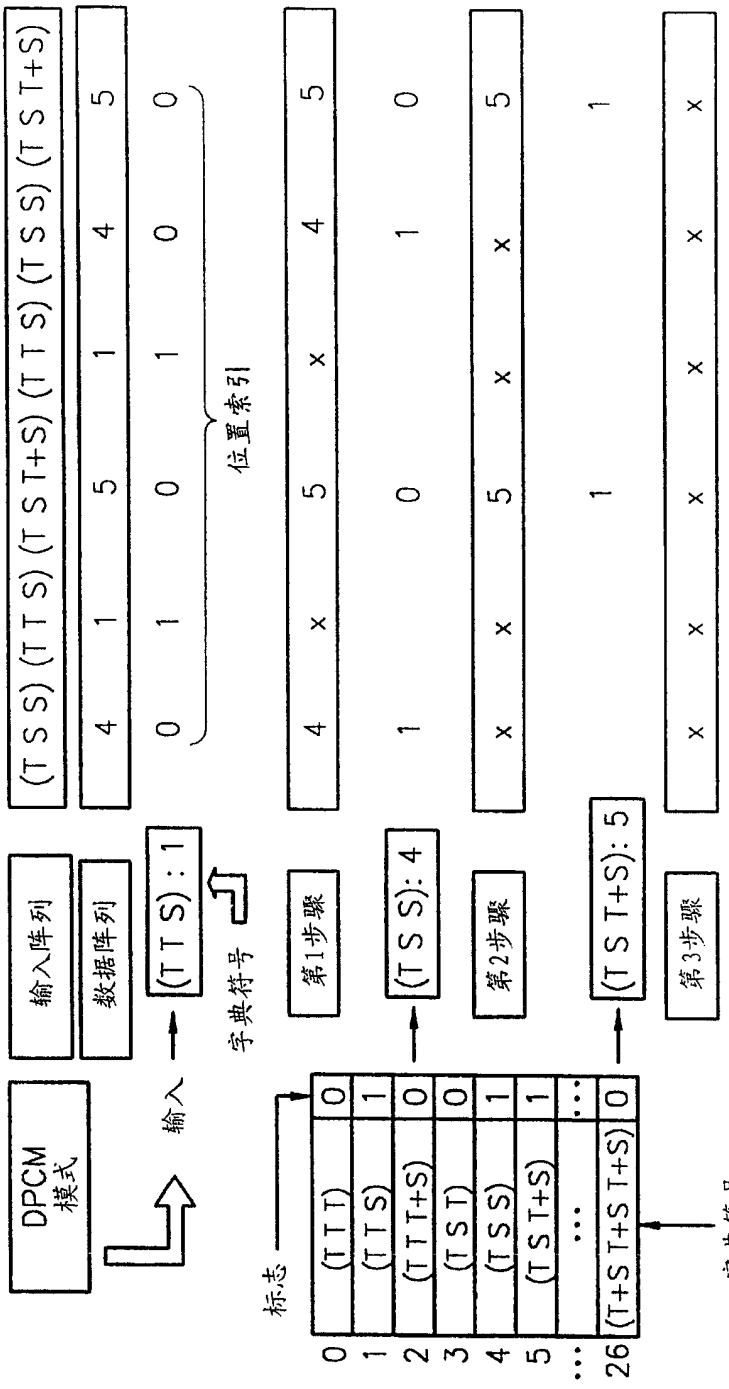

DPCM
模式
输入阵列
数据阵列
输入
(T T S): 1
字典符号
位置索引
0 1 2 3 4 5
(T S S) (T T S) (T S T+S) (T T S) (T S S) (T S T+S)
4 1 5 1 4 5
0 1 0 1 0 0
标志
第1步骤
第2步骤
第3步骤
(T S S): 4
(T S T+S): 5
字典符号
(T T T) 0
(T T S) 1
(T T T+S) 0
(T S T) 0
(T S S) 1
(T S T+S) 1
(T+S T T+S) 0
4 x 5 x 4 5
1 0 1 0
x x 5 x x 5
1 1
x x x x x x

图 11A

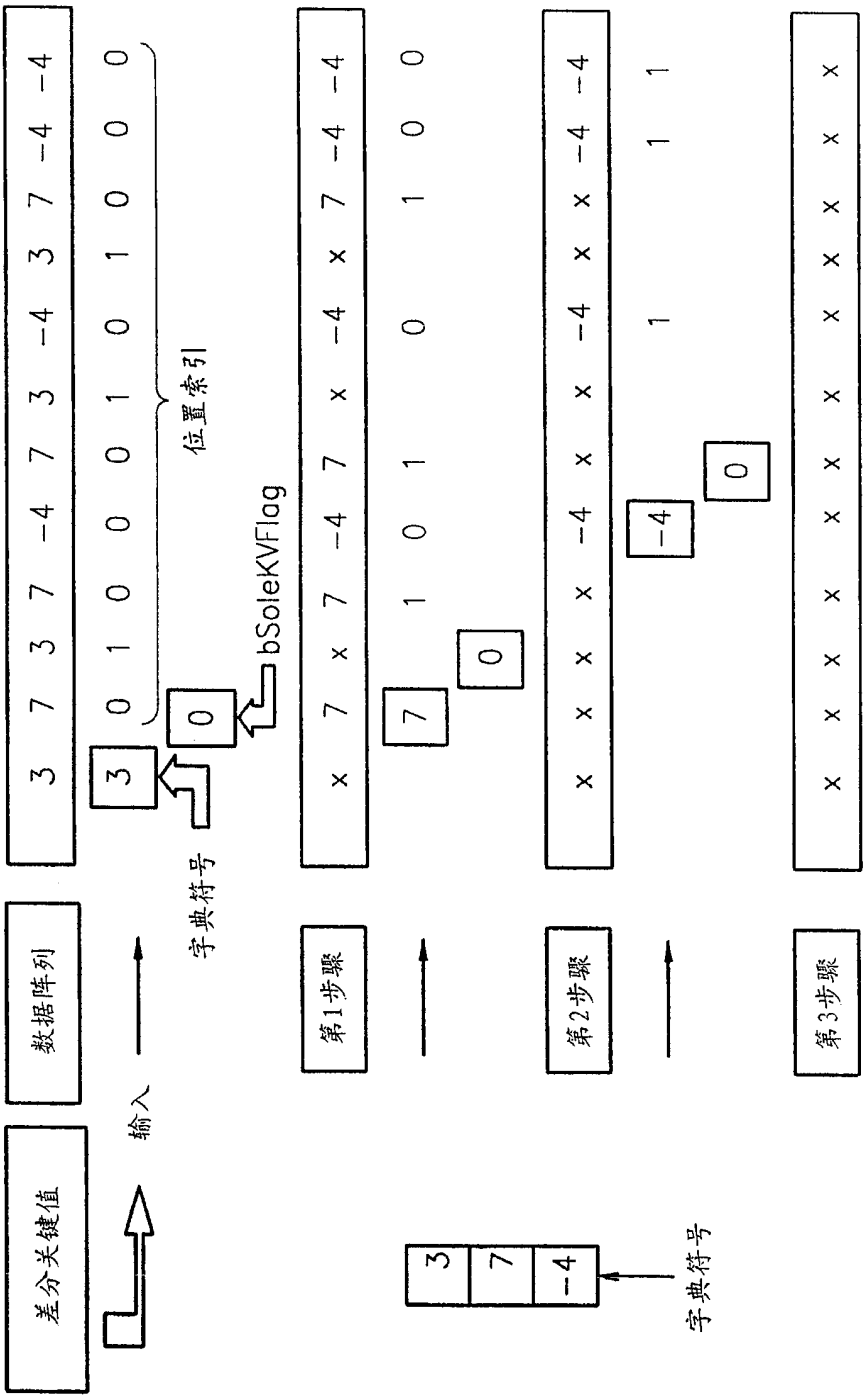

差分关键值
数据阵列
输入
3 7 3 7 -4 7 3 -4 3 7 -4 -4
3 0 1 0 0 0 0 1 0 1 0 0 0
位置索引
字典符号
0
bSoleKVFlag
第1步骤
x 7 x 7 -4 7 x -4 x 7 -4 -4
7
1 0 1 0 1 0 0
0
第2步骤
x x x x -4 x x -4 x x -4 -4
-4
1 1 1
0
第3步骤
x x x x x x x x x x x
3
7
-4
字典符号

图 11B

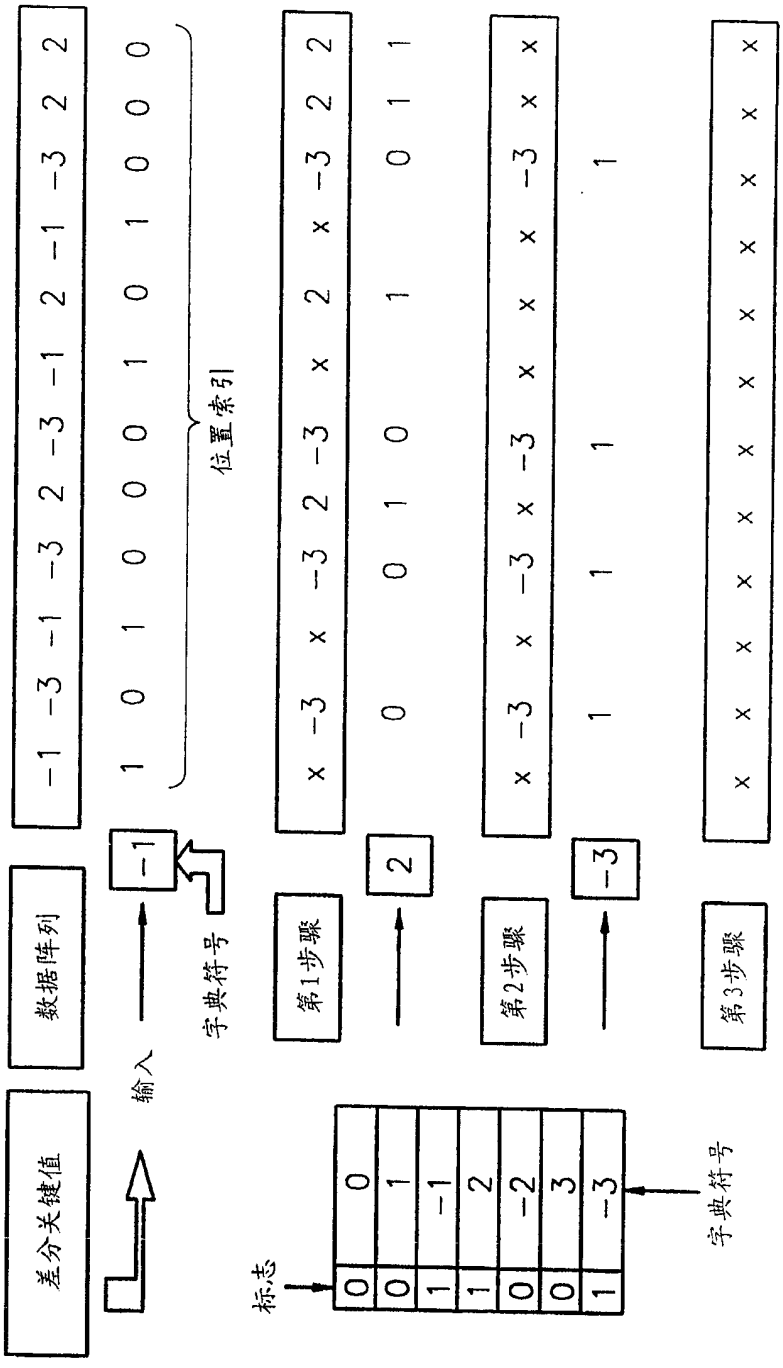

差分关键值
数据阵列
输入
-1 -3 -1 -3 2 -3 -1 2 -1 -3 2 2
1 0 1 0 0 0 1 0 1 0 0 0
-1
字典符号
位置索引
第1步骤
x -3 x -3 2 -3 x 2 x -3 2 2
2
0 0 1 0 1 0 1 1
第2步骤
x -3 x -3 x -3 x x x -3 x x
-3
1 1 1 1
第3步骤
x x x x x x x x x x x
标志
0 0
0 1
1 -1
1 2
0 -2
0 3
1 -3
字典符号

图 11C

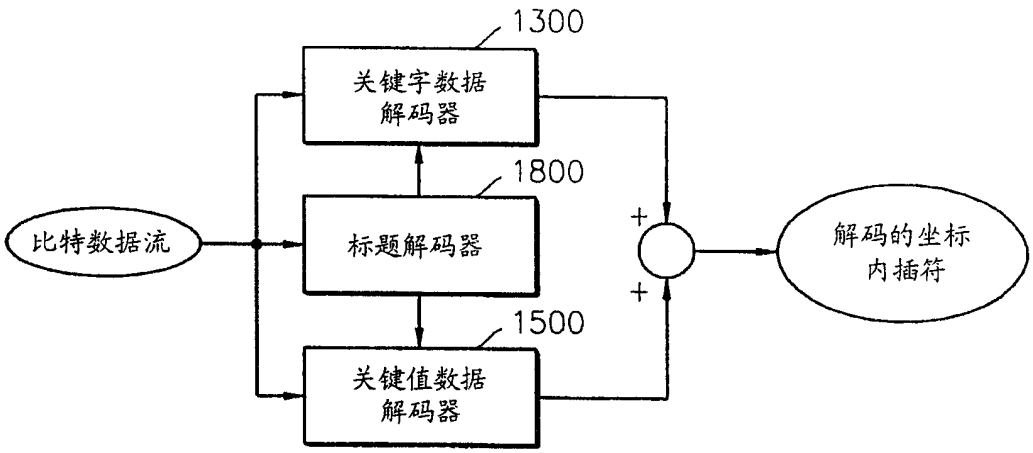


图 12

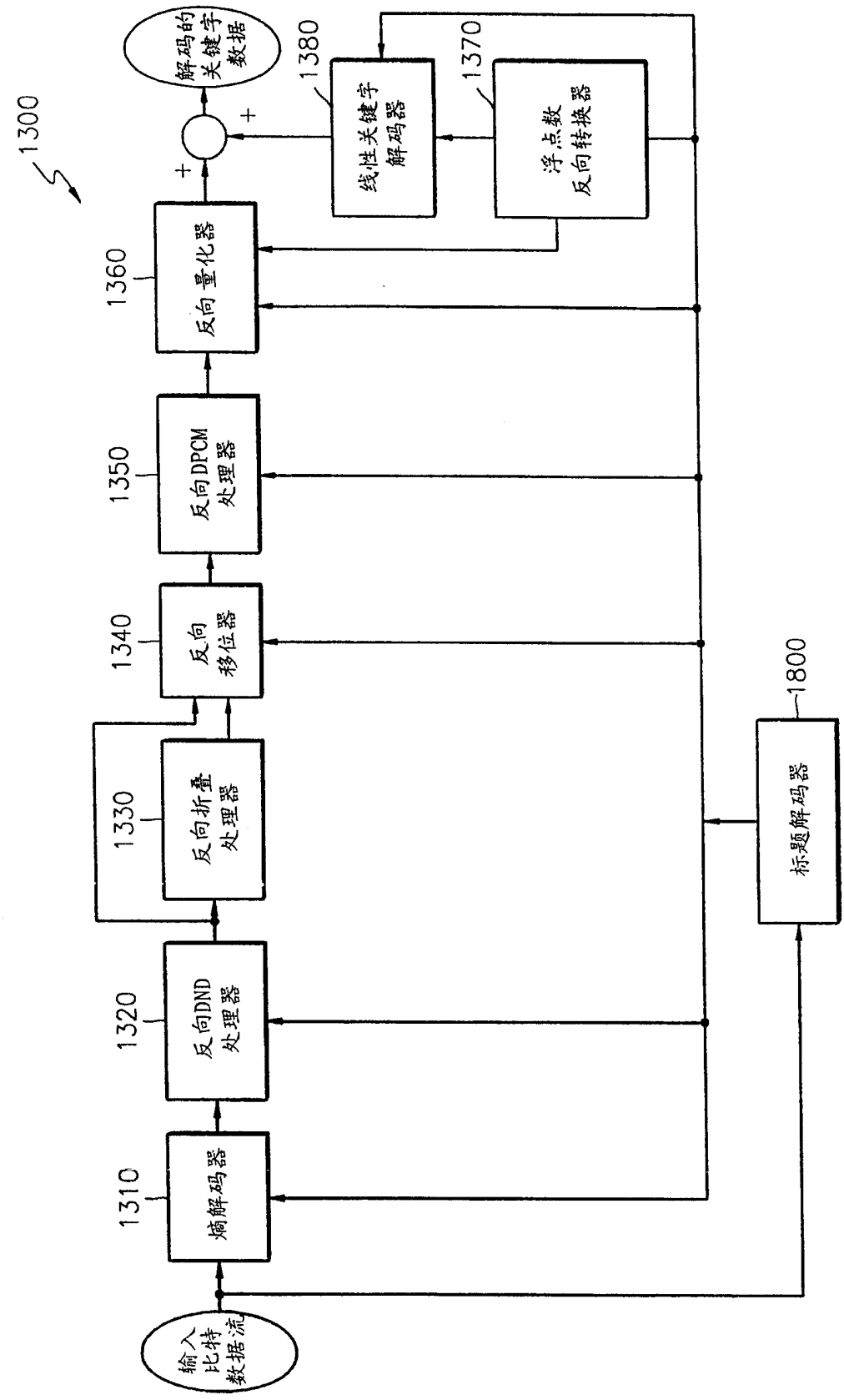

1300
输入比特数据流
1310
熵解码器
1320
反向DND处理器
1330
反向折叠处理器
1340
反向移位器
1350
反向DPCM处理器
1360
反向量化器
1370
浮点数反向转换器
1380
线性关键字解码器
解码的关键字数据
1800
标题解码器

图 13

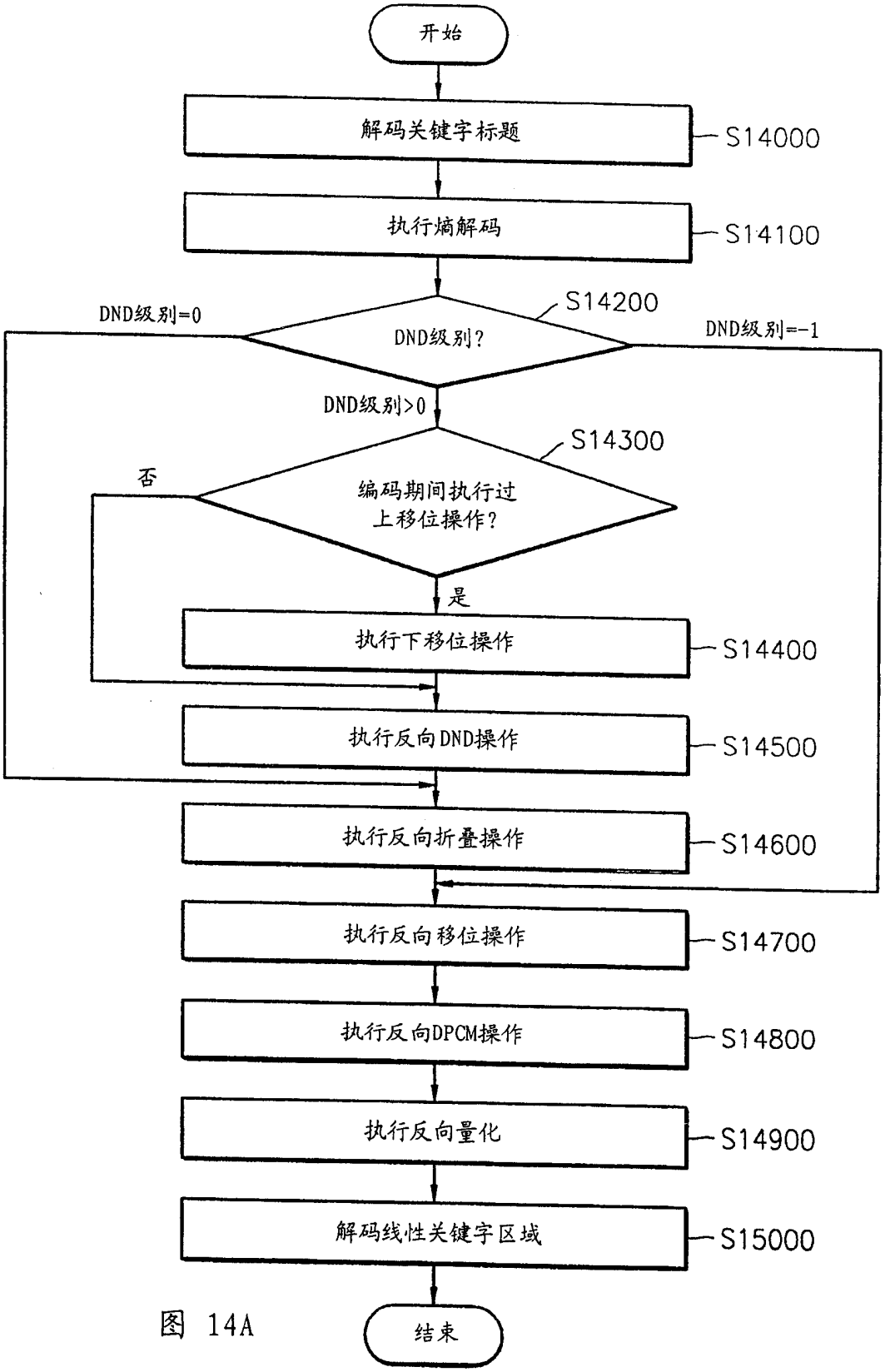

开始
解码关键字标题
S14000
执行熵解码
S14100
S14200
DND级别?
DND级别=0
DND级别=-1
DND级别>0
S14300
编码期间执行过
上移位操作?
否
是
执行下移位操作
S14400
执行反向DND操作
S14500
执行反向折叠操作
S14600
执行反向移位操作
S14700
执行反向DPCM操作
S14800
执行反向量化
S14900
解码线性关键字区域
S15000
结束

图 14A

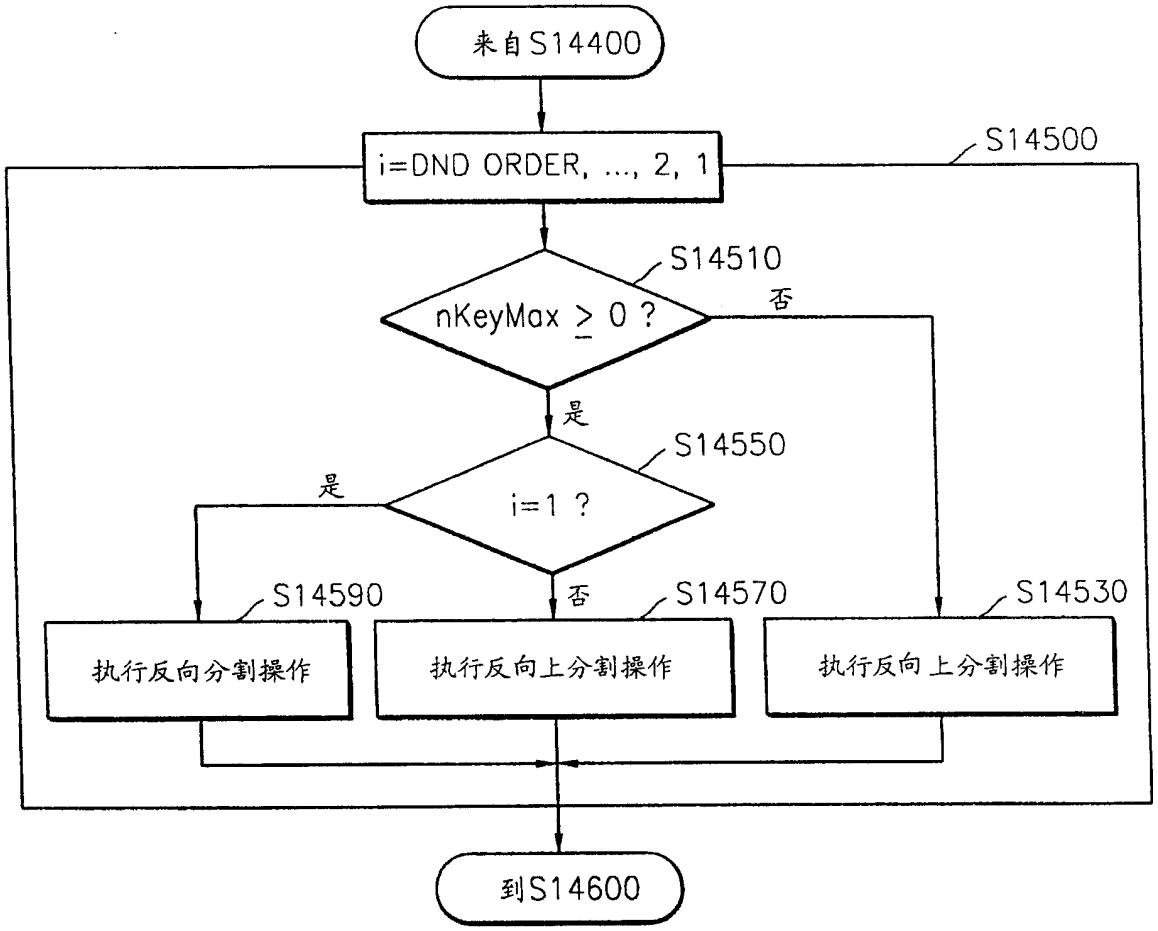

来自S14400
i=DND ORDER, ..., 2, 1
S14500
S14510
nKeyMax ≥ 0 ?
否
是
S14550
i=1 ?
是
否
S14590
S14570
S14530
执行反向分割操作
执行反向上分割操作
执行反向上分割操作
到S14600

图 14B

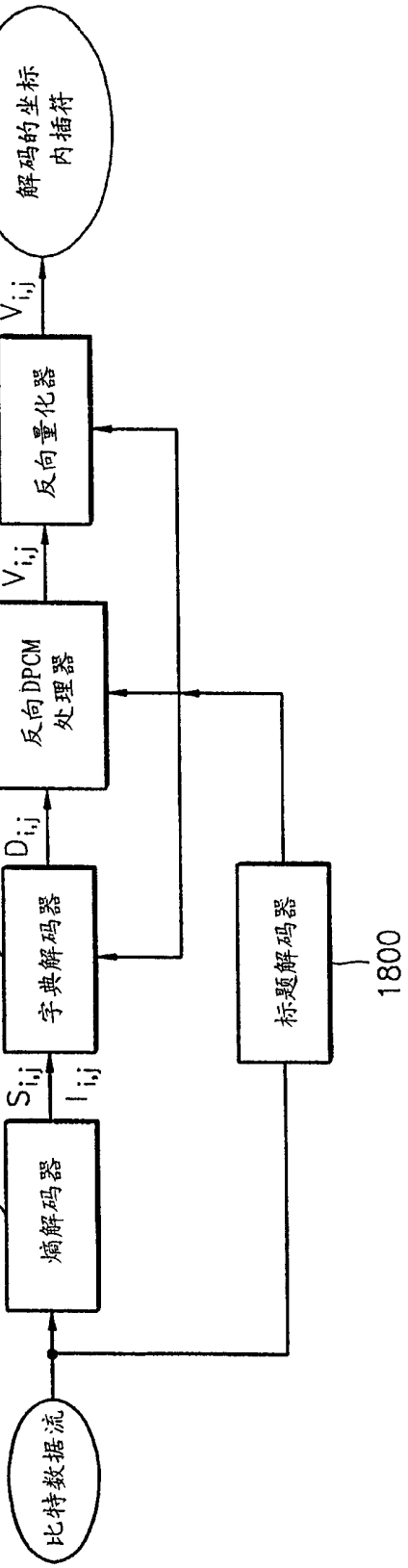

比特数据流
熵解码器
1505
$S_{i,j}$
$I_{i,j}$
字典解码器
1510
$D_{i,j}$
反向DPCM处理器
1530
$\tilde{V}_{i,j}$
反向量化器
1550
$\hat{V}_{i,j}$
解码的坐标内插符
标题解码器
1800

图 15A

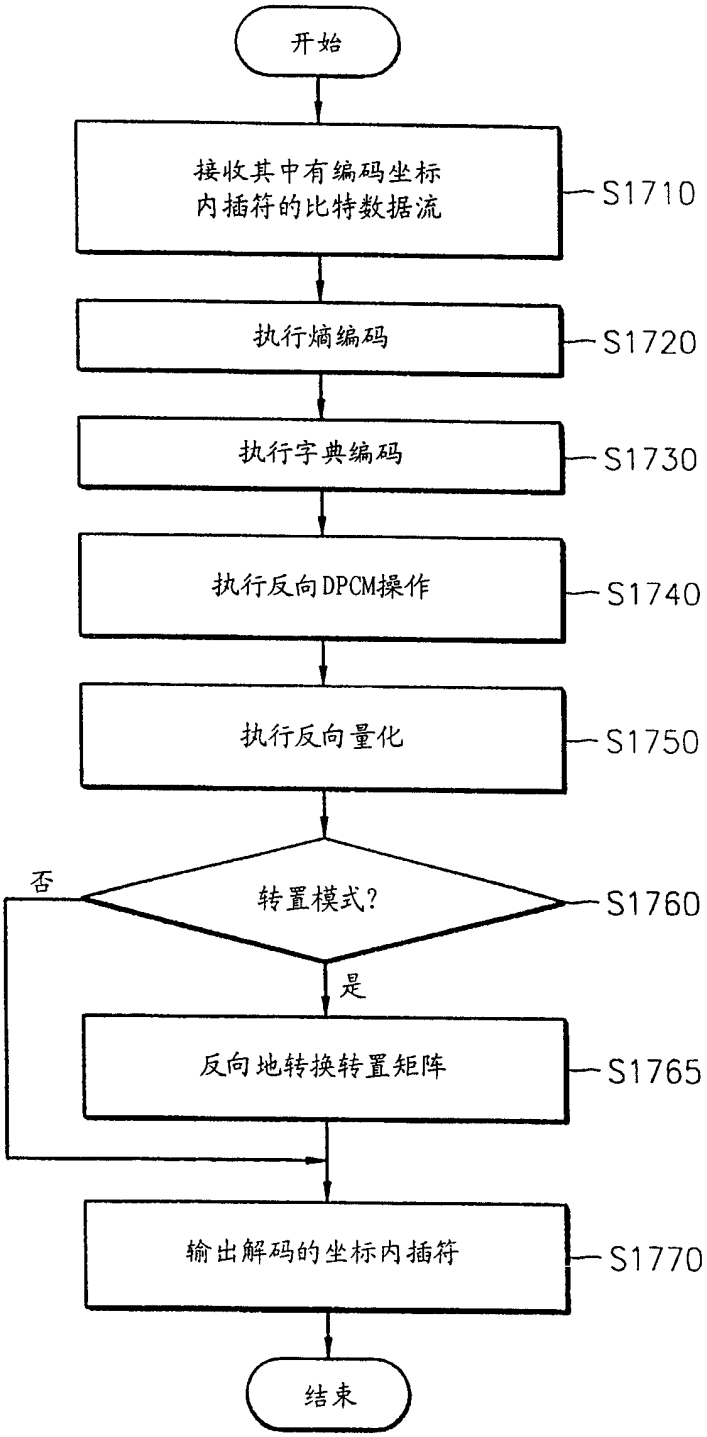

开始
接收其中有编码坐标内插符的比特数据流
S1710
执行熵编码
S1720
执行字典编码
S1730
执行反向DPCM操作
S1740
执行反向量化
S1750
转置模式?
S1760
否
是
反向地转换转置矩阵
S1765
输出解码的坐标内插符
S1770
结束

图 15B

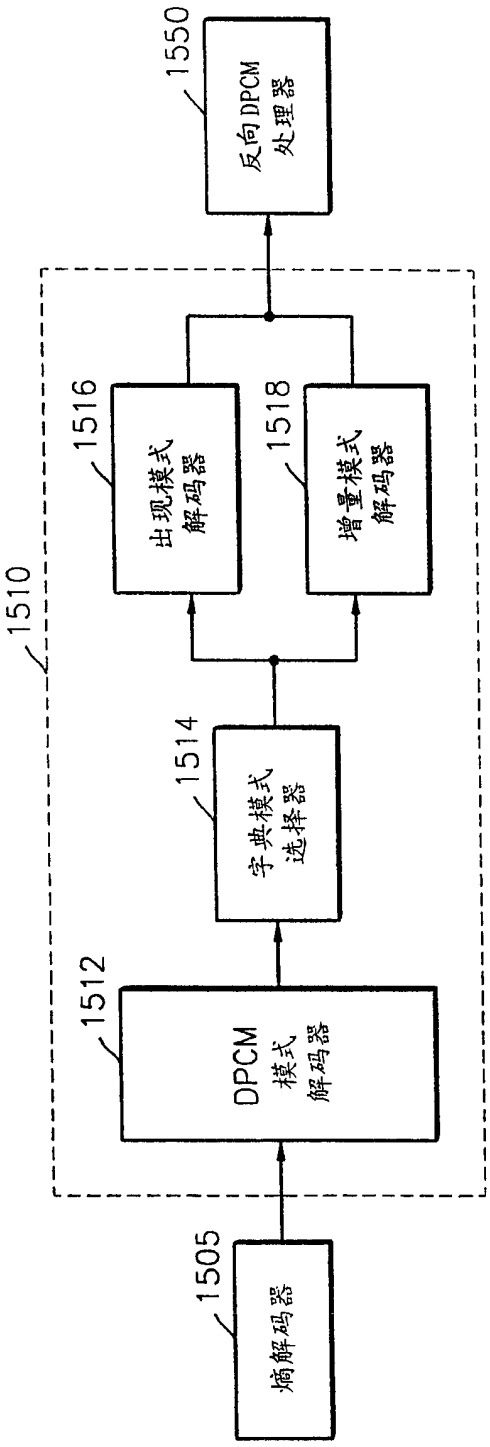

1550
反向DPCM处理器
1516
出现模式解码器
1518
增量模式解码器
1510
1514
字典模式选择器
1512
DPCM模式解码器
1505
熵解码器

图 16A

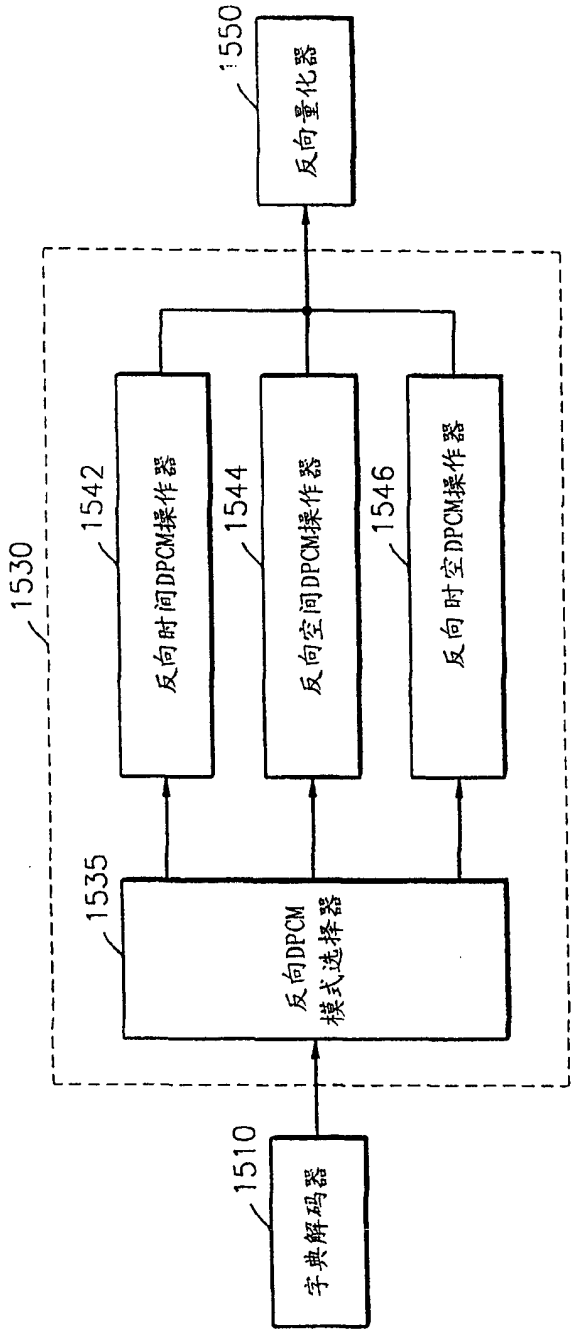

1550
反向量化器
1530
1542
反向时间DPCM操作器
1544
反向空间DPCM操作器
1546
反向时空DPCM操作器
1535
反向DPCM
模式选择器
1510
字典解码器

图 16B

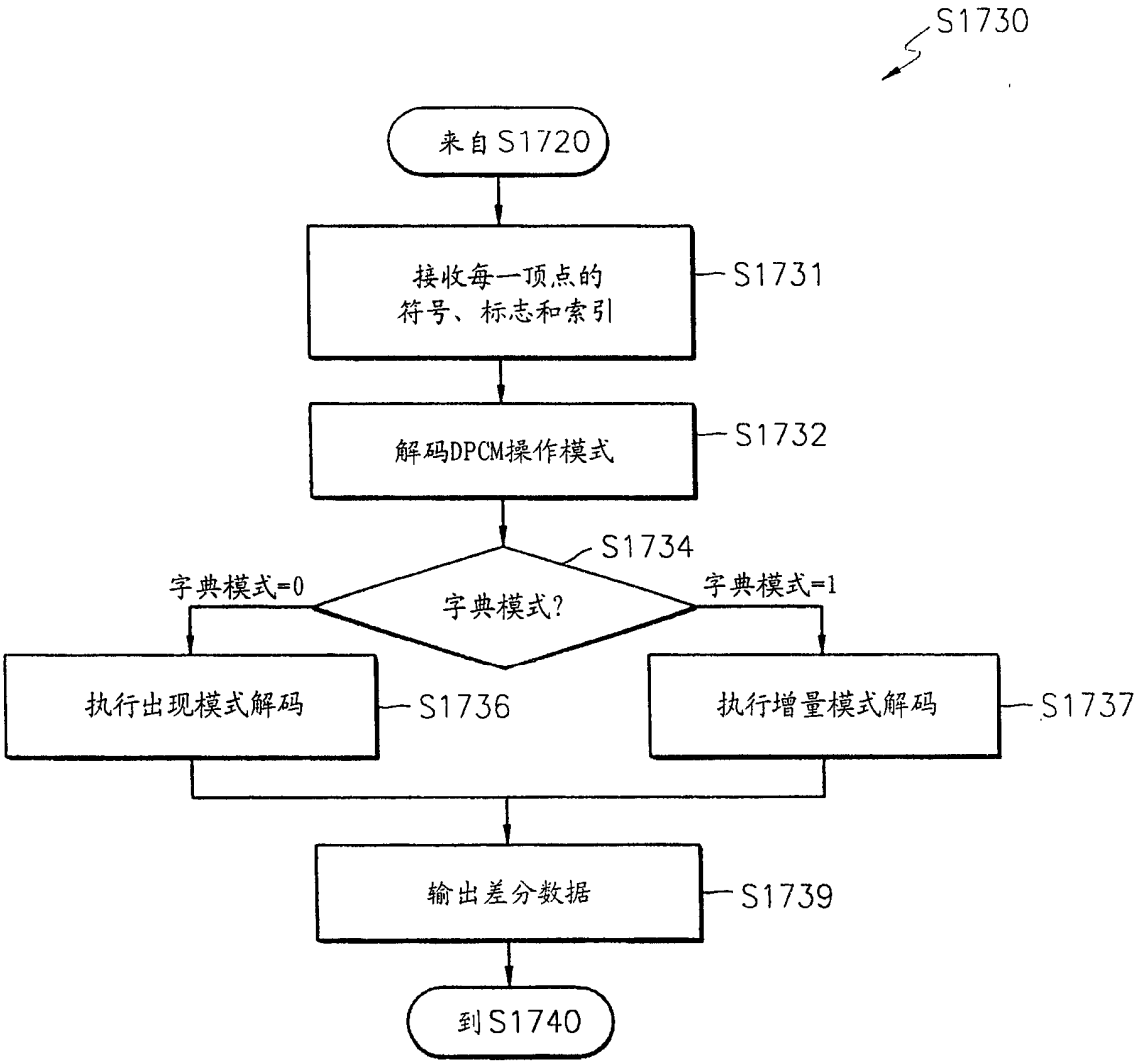

S1730
来自S1720
接收每一顶点的
符号、标志和索引
S1731
解码DPCM操作模式
S1732
S1734
字典模式?
字典模式=0
字典模式=1
执行出现模式解码
S1736
执行增量模式解码
S1737
输出差分数据
S1739
到S1740

图 17A

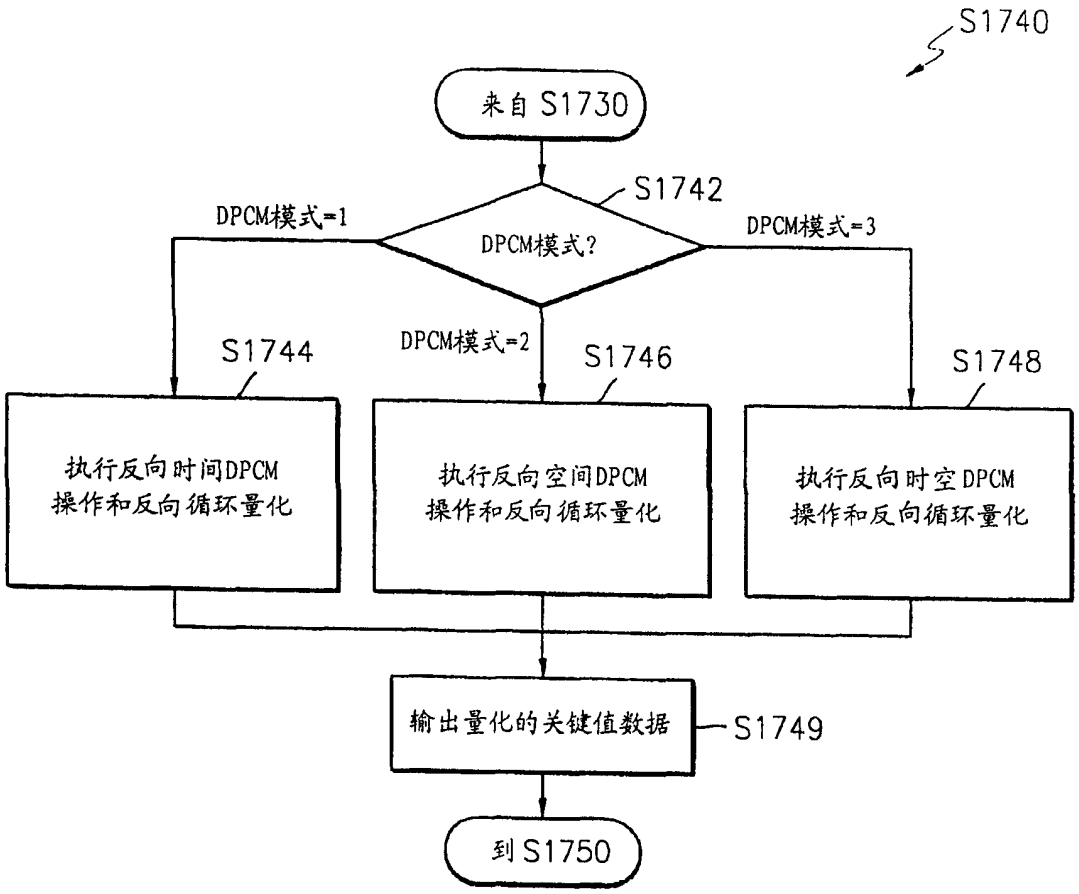

S1740
来自 S1730
S1742
DPCM模式?
DPCM模式=1
DPCM模式=2
DPCM模式=3
S1744
S1746
S1748
执行反向时间DPCM
操作和反向循环量化
执行反向空间DPCM
操作和反向循环量化
执行反向时空DPCM
操作和反向循环量化
输出量化的关键值数据
S1749
到 S1750

图 17B

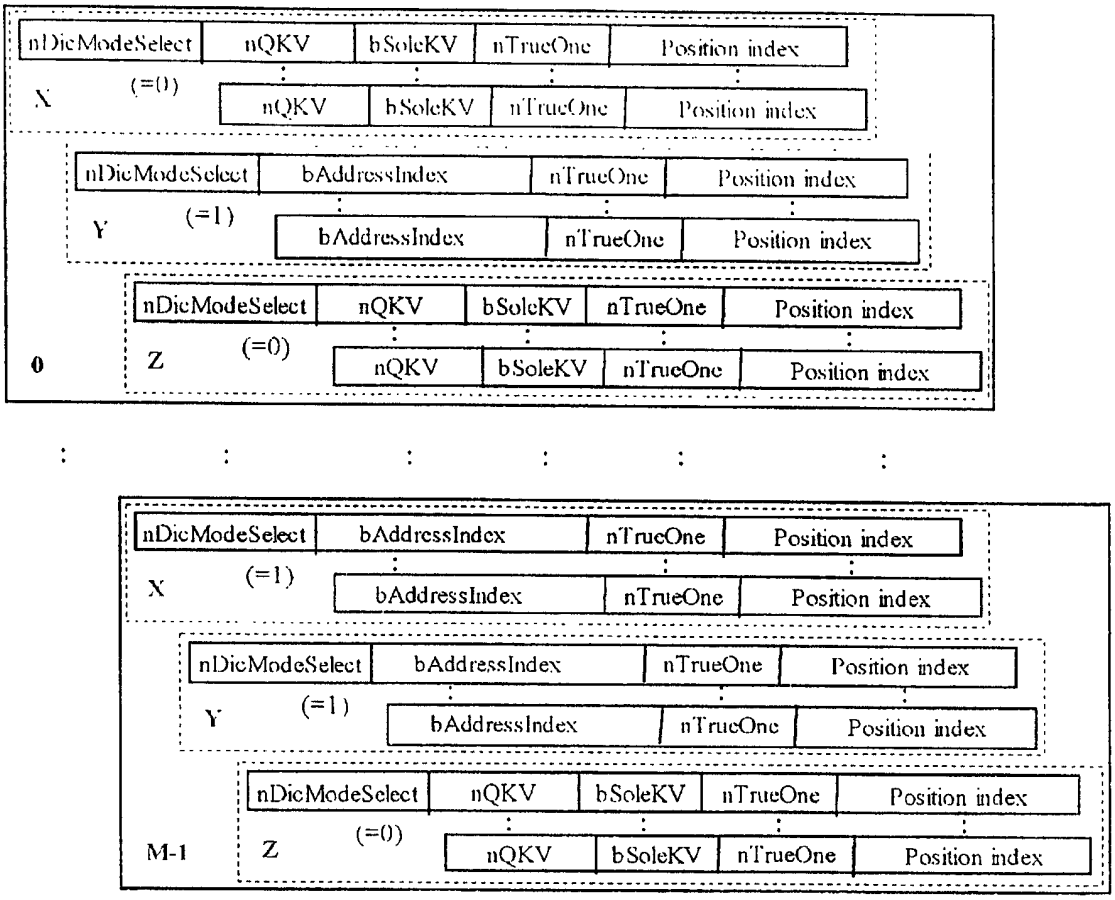

nDicModeSelect
nQKV
bSoleKV
nTrueOne
Position index
X
(=0)
Y
(=1)
Z
(=0)
0
M-1
bAddressIndex

图 18A

```
void decodeSignedQuasiAAC(int *nDecodedValue, int qstep, QState *signContext, QState
*valueContext)
{
        int b = qstep - 2;
        int msb = 0;
        do {
                qf_decode(&msb, &valueContext[b]);
                msb = msb << b;
                b--;
        } while (msb == 0 && b >= 0);
        int sgn = 0;
        int rest = 0;
        if(msb != 0) {
                qf_decode(&sgn, signContext);
                while (b >= 0) {
                        int temp = 0;
                        qf_decode(&temp, zeroContext);
                        rest |= (temp << b);
                        b--;
                }
        }
        if(sgn)
                *nDecodedValue = -(msb+rest);
        else
                *nDecodedValue = (msb+rest);
}
```

图 18B

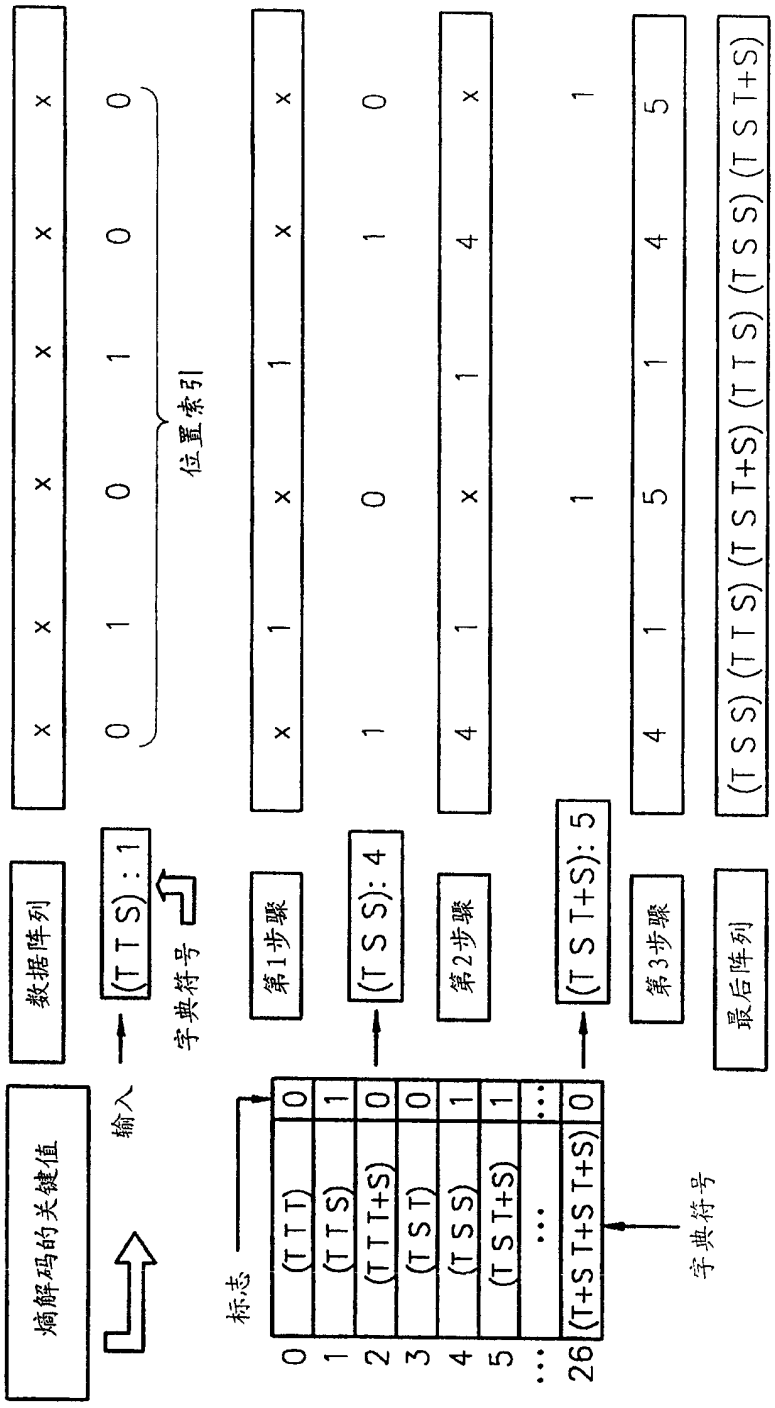

熵解码的关键值
输入
数据阵列
(T T S): 1
字典符号
位置索引
第1步骤
(T S S): 4
第2步骤
(T S T+S): 5
第3步骤
最后阵列
标志
字典符号
(T T T)
(T T S)
(T T T+S)
(T S T)
(T S S)
(T S T+S)
(T+S T+S T+S)

图 19A

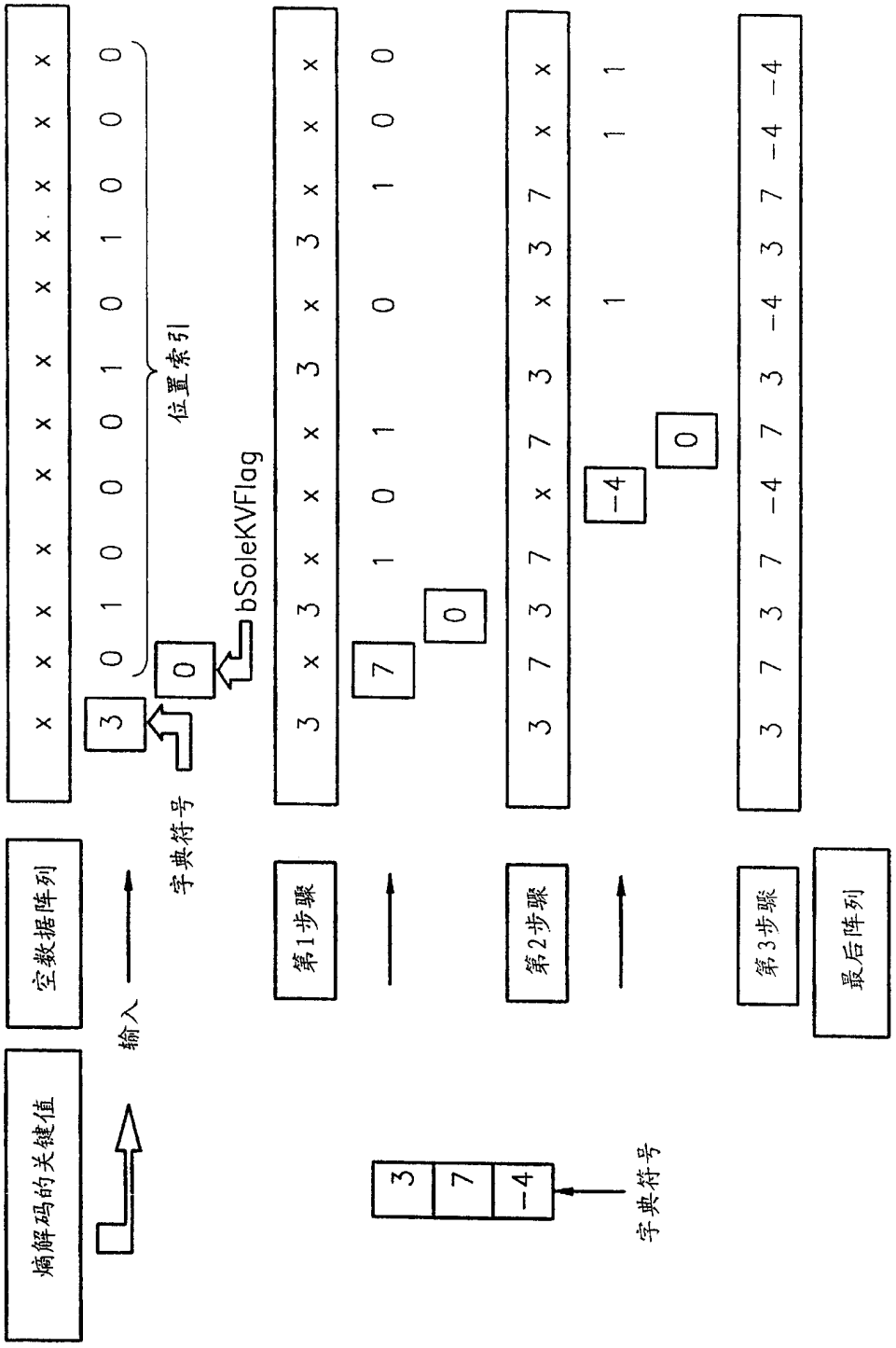

熵解码的关键值
空数据阵列
输入
x x x x x x x x x x x x x x x x
3
0 1 0 0 0 0 1 0 1 0 0 0
位置索引
字典符号
0
bSoleKVFlag
第1步骤
3 x 3 x x x x 3 x 3 x x x
7
1 0 1 0 1 0 0
0
第2步骤
3 7 3 7 x 7 3 x 3 7 x x
-4
1 1 1
0
第3步骤
3 7 3 7 -4 7 3 -4 3 7 -4 -4
最后阵列
3
7
-4
字典符号

图 19B

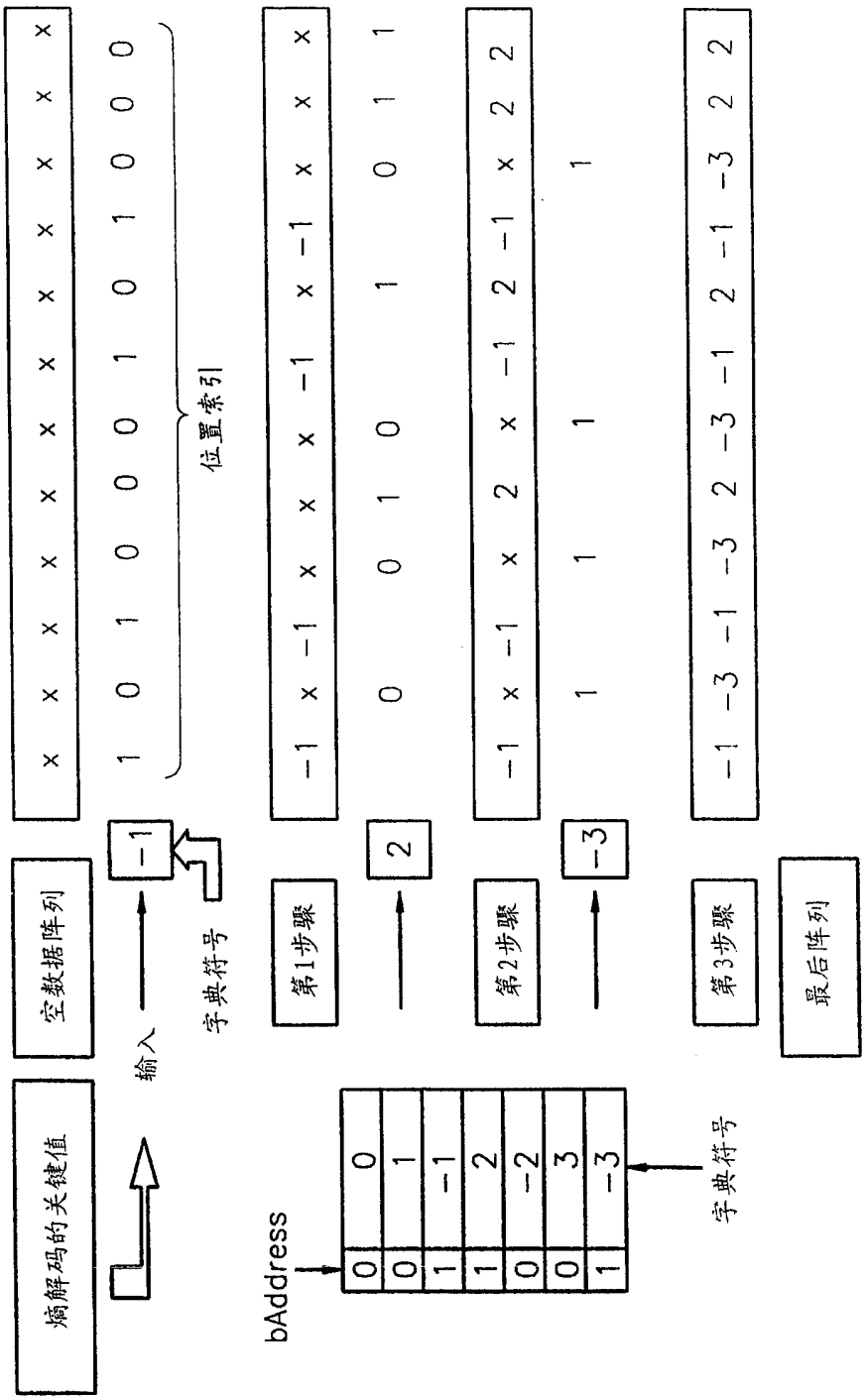

熵解码的关键值
空数据阵列
输入
-1
字典符号
位置索引
x x x x x x x x x x x x x x x x
1 0 1 0 0 0 0 1 0 1 0 0 0 0
第1步骤
-1 x -1 x x x -1 x -1 x x x
0 0 1 0 1 0 1 1
2
第2步骤
-1 x -1 x 2 x -1 2 -1 x 2 2
1 1 1 1
-3
第3步骤
-1 -3 -1 -3 2 -3 -1 2 -1 -3 2 2
最后阵列
bAddress
0 0
0 1
1 -1
1 2
0 -2
0 3
1 -3
字典符号

图 19C

```
class CompressedCoordinateInterpolator {
        KeyHeader kHeader;
        CoordIKeyValueHeader coordIKVHeader;
        qf_start();
        aligned(8) Key k(kHeader);
        CoordIKeyValue coordIKeyValue(coordIKVHeader,
kHeader.nNumberOfKey);
}
```

图 20A

```
class KeyHeader {
        int i;
        unsigned int(5) nKeyQBit;
        unsigned int(5) nNumKeyCodingBit;
        unsigned int(nNumKeyCodingBit) nNumberOfKey;
        unsigned int(4) nKeyDigit;
        bit(1) bIsLinearKeySubRegion;
        if(bIsLinearKeySubRegion == 1)
                LinearKey lKey(nKeyDigit);
        bit(1) bRangeFlag;
        if(bRangeFlag == 1)
                KeyMinMax keyMinMax(nKeyDigit);
        unsigned int(5) nBitSize;
        unsigned int(2) nKDPCMOrder;
        for(i = 0; i < nKDPCMOrder + 1; i++) {
                bit(1) nQIntraKeySign[[i]];
                if(i == 0 && nQIntraKeySign[i] == 1)
                        continue;
                unsigned int(nBitSize) nQIntraKey[[i]];
        }
        bit(1) bShiftFlag;
        if(bShiftFlag == 1) {
                bit(1) nKeyShiftSign;
                unsigned int(nBitSize) nKeyShift;
        }
        unsigned int(3) nDNDOrder;
        if(nDNDOrder == 7) {
                bit(1) bNoDND;
                if(bNoDND == 1)
                        nDNDOrder = -1;
        }
        int nMaxQBit = nBitSize;
        for(i = 0; i < nDNDOrder; i++) {
                bit(1) nKeyMaxSign[[i]];
                unsigned int(nMaxQBit) nKeyMax[[i]];
                nMaxQBit = (int)(log10(abs(nKeyMax[i]))/log10(2))+1;
                if(nMaxQBit+1 < nBitSize)
                        nMaxQBit += 1;
                else
                        nMaxQBit = nBitSize;
        }
        int bSignedAACFlag;
        int nKeyCodingBitQBit = (int)(log10(nKeyQBit))/log10(2))+1;
        unsigned int(nKeyCodingBitQBit) nKeyCodingBit;
        if(nDNDOrder != -1 && nDNDOrder != 0) {
                bit(1) bKeyInvertDownFlag;
                if(bKeyInvertDownFlag == 1) {
                        unsigned int(nKeyCodingBit) nKeyInvertDown;
                        bSignedAACFlag = 0;
                } else {
                        bSignedAACFlag = 1;
                }
        } else {
                bSignedAACFlag = 0;
        }
}
```

图 20B

```
class LinearKey (int nKeyDigit) {

        unsigned int(5) nNumLinearKeyCodingBit;

        unsigned int(nNumLinearKeyCodingBit) nNumberOfLinearKey ;

        KeyMinMax kMinMax(nKeyDigit);

}
```

图 20C

```
class KeyMinMax (int nKeyDigit) {
        bit(1) bMinKeyDigitSame;
        if((bMinKeyDigitSame == 0)
                unsigned int(4) nMinKeyDigit;
        else
                nMinKeyDigit = nKeyDigit;
        if(nMinKeyDigit != 0) {
                if(nMinKeyDigit < 8) {
                        int count = (int)(log10(10^nMinKeyDigit-1)/log10(2)) + 1;
                        bit(1) nMinKeyMantissaSign;
                        unsigned int(count) nMinKeyMantissa;
                        bit(1) nMinKeyExponentSign;
                        unsigned int(6) nMinKeyExponent;
                } else
                        float(32) fKeyMin;
        }
        bit(1) bMaxKeyDigitSame;
        if(bMaxKeyDigitSame == 0)
                unsigned int(4) nMaxKeyDigit;
        else
                nMaxKeyDigit = nKeyDigit;
        if(nMaxKeyDigit != 0) {
                if(nMaxKeyDigit < 8) {
                        int count = (int)(log10(10^nMaxKeyDigit)-1)/log10(2)) + 1;
                        bit(1) nMaxKeyMantissaSign;
                        unsigned int(count) nMaxKeyMantissa;
                        bit(1) bSameExponent;
                        if(bSameExponent == 0) {
                                bit(1) nMaxKeyExponentSign;
                                unsigned int(6) nMaxKeyExponent;
                        }
                        else
                                nMaxKeyExponent = nMinKeyExponent;
                } else
                        float(32) fKeyMax;
        }
}
```

图 20D

```
class Key (KeyHeader kHeader) {
        int nQKey[kHeader.nNumberOfKey];
        int i;
        int nNumberOfRemainingKey;
        if(kHeader.bIsLinearKeySubRegion == 1)
                nNumberOfRemainingKey = kHeader.nNumberOfKey
kHeader.lKey.nNumberOfLinearKey;
        else
                nNumberOfRemainingKey = kHeader.nNumberOfKey;
        for(i = kHeader.nKDPCMOrder+1; i < nNumberOfRemainingKey; i++) {
                if(kHeader.bSignedAACFlag == 0)
                        decodeUnsignedAAC(nQKey[i], kHeader.nKeyCodingBit, keyContext);
                else
                        decodeSignedAAC(nQKey[i], kHeader.nKeyCodingBit+1,
keySignContext, keyContext);
        }
}
```

图 20E

```
class CoordIKeyValueHeader {
        bit(1) bTranspose;
        unsigned int(5) nKVQBit;
        unsigned int(5) nCoordQBit;
        unsigned int(nCoordQBit) nNumberOfCoord;
        unsigned int(4) nKVDigit;
        KeyValueMinMax kVMinMax (nKVDigit);
        unsigned int(nKVQBit) nXQMinOfMin;
        unsigned int(nKVQBit) nXQMinOfMax;
        unsigned int(nKVQBit) nYQMinOfMin;
        unsigned int(nKVQBit) nYQMinOfMax;
        unsigned int(nKVQBit) nZQMinOfMin;
        unsigned int(nKVQBit) nZQMinOfMax;
        unsigned int(nKVQBit) nXQMaxOfMin;
        unsigned int(nKVQBit) nXQMaxOfMax;
        unsigned int(nKVQBit) nYQMaxOfMin;
        unsigned int(nKVQBit) nYQMaxOfMax;
        unsigned int(nKVQBit) nZQMaxOfMin;
        unsigned int(nKVQBit) nZQMaxOfMax;
}
```

图 20F

```
class CoordIKeyValue (CoordIKeyValueHeader coordIKVHeader, int nNumberOfKey) {
        int j, c;
    if(coordIKVHeader.bTranspose == 1) {
        int temp = nNumberOfKey;
        nNumberOfKey = coordIKVHeader.nNumberOfCoord;
        coordIKVHeader.nNumberOfCoord = temp;
    }
        int nKVACodingBitQBit = (int)(log10(abs(coordIKVHeader.nKVQBit))/log10(2))+1;
        int nDPCMMode[coordIKVHeader.nNumberOfCoord][3];
        unsigned int bSelFlag[coordIKVHeader.nNumberOfCoord][3] = 1;
        CoordIDPCMMode coordIDPCMMode(coordIKVHeader);
        for(j = 0; j < coordIKVHeader.nNumberOfCoord; j++) {
                for(c = 0; c < 3; c++) {
                        if(c == 0) {
                                if(coordIKVHeader.nXQMaxOfmin <=
coordIKVHeader.nXQMinOfmax) {
                                        qf_decode(&bSelFlag[j][c]
selectionFlagContext);
                                }
                        }
                        else if(c == 1) {
                                if(coordIKVHeader.nYQMaxOfmin <=
coordIKVHeader.nYQMinOfmax) {
                                        qf_decode(&bSelFlag[j][c],
selectionFlagContext);
                                }
                        }
                        else if(c == 2) {
                                if(coordIKVHeader.nZQMaxOfmin <=
coordIKVHeader.nZQMinOfmax) {
                                        qf_decode(&bSelFlag[j][c],
selectionFlagContext);
                                }
                        }
```

图 20G

```
                    if(bSelFlag[j][c] == 1) {
                        if(c == 0)
                            decodeUnsignedAAC(&nKVACodingBit[j][c],
nKVACodingBitQBit, aqpXContext);
                        else if(c == 1)
                            decodeUnsignedAAC(&nKVACodingBit[j][c],
nKVACodingBitQBit, aqpYContext);
                        else if(c == 2)
                            decodeUnsignedAAC(&nKVACodingBit[j][c],
nKVACodingBitQBit, aqpZContext);
                        if(j > 0) {
                            if(nDPCMMode[j][c] == 2 || nDPCMMode[j][c] ==
3) {
                                int nQBitOfRef =
(int)(log10(abs(j-1))/log10(2))+1;
                                decodeUnsignedAAC(&nRefVertex[j][c],
nQBitOfRef, refContext);
                            }
                        }
                        if(nKVACodingBit[j][c] != 0) {
                            decodeSignedAAC(&nQMin[j][c],
coordIKVHeader.nKVQBit+1,
                                qMinSignContext, qMinContext);
                            decodeSignedAAC(&nQMax[j][c],
coordIKVHeader.nKVQBit+1,
                                    qMaxSignContext,
qMaxContext);
                        }
                    } else
                        decodeSignedAAC(&nQMin[j][c], coordIKVHeader.nKVQBit+1,
                            qMinSignContext, qMinContext);

                    CoordIKeyValueDic coordIKeyValueDic(bSelFlag[j][c],
                        KVACodingBit[j][c], nNumberOfKey, c);
                }
            }
}
```

图 20H

```
class CoordIDPCMMode (CoordIKeyValueHeader coordIKVHeader) {
        int i, s, k;
        unsigned int bIndexDPCMMode[coordIKVHeader.nNumberOfCoord] = 0;
        int nNumberOfSymbol = 0;
        for(i = 0; i < 27; i++) {
                qf_decode(&bAddressOfDPCMMode[i], dpcmModeDicAddressContext);
                if(bAddressOfDPCMMode[i] == 1)
                        nNumberOfSymbol++;
        }
        for(s = 0; s < nNumberOfSymbol; s++) {
                for(k = 1; k < coordIKVHeader.nNumberOfCoord; k++) {
                        if(bIndexDPCMMode[k] == 0) {
                                qf_decode(&bDPCMIndex, dpcmModeDicIndexContext);
                                if(bDPCMIndex == 1)
                                        bIndexDPCMMode[k] = 1;
                        }
                }
        }
}
```

图 20I

```
class CoordIKeyValueDic (unsigned int bSelFlag, unsigned int nKVCodingBit,
int nNumberOfKey, int c) {
        if(bSelFlag == 1 && nKVCodingBit != 0) {
                qf_decode(&nDicModeSelect, dicModeSelectionContext);
                if(nDicModeSelect == 1)
                        CoordIIncrementalMode
coordIIncrementalMode(nKVCodingBit, nNumberOfKey);
                else
                        CoordIOccurrenceMode coordIOccurrenceMode(nKVCodingBit,
        nNumberOfKey, c);
        }
}
```

图 20J

```
class CoordIIncrementalMode (unsigned int nKVCodingBit, int nNumberOfKey) {
        int i, s, k;
        int nSizeOfAddress = (2^ nKVCodingBit+1)-1;
        unsigned int bAddrIndex[nNumberOfKey] = 0;
        int nNumberOfSymbol = 0;
        for(i = 0; i < nSizeOfAddress; i++) {
                qf_decode(&bAddress[i], dicAddressContext);
                if(bAddress[i] == 1) {
                        nNumberOfSymbol++;
                }
        }
        for(s = 0; s < nNumberOfSymbol; s++) {
                qf_decode(&nTrueOne, dicOneContext);
                for(k = 0; k < nNumberOfKey; k++) {
                        if(bIndexOfAddr[k] == 0) {
                                qf_decode(&bAddrIndex, dicIndexContext);
                                if(bAddrIndex == nTrueOne) {
                                        bAddrIndex[k] = 1;
                                }
                        }
                }
        }
}
```

图 20K

```
class CoordIOccurrenceMode (unsigned int nKVCodingBit, int nNumberOfKey, int
c) {
        int i, k;
        unsigned int bIndexOfDic[nNumberOfKey] = 0;
        for(i = 0; i < nNumberOfKey; i++) {
                if(bIndexOfDic[i] == 0) {
                        bIndexOfDic[nNumberOfKey] = 1;
                        if(c == 0)
                                decodeSignedQuasiAAC(&nQKV[i], nKVCodingBit+1,
                                        kvSignContext, kvXContext);
                        else if(c == 1)
                                decodeSignedQuasiAAC(&nQKV[i], nKVCodingBit+1,
                                        kvSignContext, kvYContext);
                        else if(c == 2)
                                decodeSignedQuasiAAC(&nQKV[i], nKVCodingBit+1,
                                        kvSignContext, kvZContext);
                        qf_decode(&bSoleKV, dicSoleKVContext);
                        if(bSoleKV == 0) {
                                qf_decode(&nTrueOne, dicOneContext);
                                for(k = i+1; k < nNumberOfKey; k++) {
                                        if(bIndexOfDic[k] == 0) {
                                                int bDicIndex;
                                                qf_decode(&bDicIndex,
dicIndexContext);
                                                if(bDicIndex == nTrueOne)
                                                        bIndexOfDic[k] = 1;
                                        }
                                }
                        }
                }
        }
}
```

图 20L