[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06T 9/00 (2006.01)
G06T 15/70 (2006.01)
H03M 7/30 (2006.01)

[12] 发明专利说明书

专利号 ZL 02140002.4

[45] 授权公告日 2007 年 6 月 6 日

[11] 授权公告号 CN 1320503C

[22] 申请日 2002.11.27 [21] 申请号 02140002.4

[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] 2002.10.18 [33] US [31] 63852/02

[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

[56] 参考文献
JP738761A 1995.2.7
审查员 杜 轶

[74] 专利代理机构 北京市柳沈律师事务所
代理人 马 莹 邵亚丽

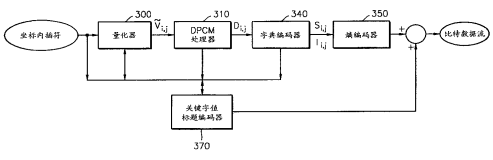
权利要求书 10 页 说明书 31 页 附图 34 页

[54] 发明名称

编码和解码坐标内插符的关键字值数据的方法和装置

[57] 摘要

一种用于编码/解码使用在三维图形动画中的坐标内插符关键字值数据的方法和装置。 用于编码使用包括 x、y 和 z 分量的每一个顶点的坐标表示对象的每一顶点的位置的坐标内插符的关键字值数据的装置包括：量化器，以预定的量化比特量化输入其中的坐标内插符，DPCM 处理器，对量化的坐标内插符的每一顶点的每一分量执行预定模式的 DPCM 操作，并且因此产生基于每一顶点的坐标的时间变化的差分数据以及基于每一个顶点的坐标的空间变化的差分数据，字典编码器，产生表示每一顶点的每一分量的差分数据的符号和已经对差分数据执行的 DPCM 操作的模式以及表示符号的位置的位置索引，和熵编码器，熵编码符号和位置索引。



1. 用于编码坐标内插符的关键字值数据的装置，坐标内插符使用包括x、y和z分量的每一个顶点的坐标表示对象的每一顶点的位置，该装置包括:

量化器，其以预定的量化比特量化输入其中的坐标内插符，

差分脉码调制处理器，其对量化的坐标内插符的每一顶点的每一分量执行预定模式的差分脉码调制操作，并且因此产生基于每一顶点的坐标的时间变化的差分数据以及基于每一个顶点的坐标的空间变化的差分数据;

字典编码器，其产生表示每一顶点的每一分量的差分数据的符号和已经对差分数据执行的差分脉码调制操作的模式以及表示符号的位置的位置索引; 及

熵编码器，其熵编码符号和位置索引，

其中差分脉码调制处理器包括:

差分脉码调制操作器，对量化的坐标内插符的每一顶点的每一分量执行时间差分脉码调制操作，以便产生在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; 对量化的坐标内插符的每一顶点的每一分量执行空间差分脉码调制操作，以便产生在同关键帧中的顶点之间的第二差分数据; 并且对量化的坐标内插符的每一顶点的每一分量执行空间-时间差分脉码调制操作，以便产生在顶点和关键帧之间的第三差分数据;

循环量化器，其对从差分脉码调制操作器输入的第一至第三差分数据执行循环量化操作，以便降低其范围; 及

差分脉码调制模式选择器，其根据用于进行编码所需要的比特数目选择已经循环量化的第一至第三差分数据中的一个，并且输出选择的差分数据，

而字典编码器包括:

差分脉码调制模式编码器，其产生表示已经对每一顶点的每一分量的数据执行的差分脉码调制模式组合的符号和表示符号位置的位置索引; 及

出现模式编码器，其产生对应于每一顶点的每一分量的输入差分数据的符号和表示符号的位置的位置索引。

2. 如权利要求1所述的装置，其中 差分脉码调制操作器包括:

时间差分脉码调制操作器，计算在当前关键帧中的顶点的坐标和先前关键帧顶点的坐标之间的差值;

空间差分脉码调制操作器，计算在当前关键帧中的顶点的坐标和参考顶点的坐标之间的差值；及

空间-时间差分脉码调制操作器，计算顶点的坐标与其在先前关键帧中的对应参考顶点的坐标之间的差值和顶点的坐标与其在当前关键帧中的对应参考顶点的坐标之间的差值之间的差分数据。

3. 如权利要求2所述的装置，其中在当前顶点受空间差分脉码调制操作之前的已经差分脉码调制的顶点当中，参考顶点是具有用于编码在本身和受到空间差分脉码调制操作的当前顶点之间的差分数据所需要的最小比特数目的顶点。

4. 如权利要求1所述的装置，其中循环量化器通过对每一顶点的每一分量的差分数据和在它们当中的最大和最小值执行预定的操作而产生具有缩减大小的差分数据。

5. 如权利要求4所述的装置，其中循环量化器根据差分数据的符号，通过把每一顶点的每一分量的差分数据的范围值加到差分数据或从差分数据减去范围值而产生循环量化的差分数据，并且输出在差分数据和循环量化的差分数据之间的较小的。

6. 如权利要求1所述的装置，其中差分脉码调制模式选择器对第一至第三差分数据执行绝对差之和操作、分散操作以及熵操作中的一个，并且选择在已经通过对应操作的第一至第三差分数据中的具有最小大小的差分数据。

7. 如权利要求1所述的装置，其中出现模式编码器以表格的形式产生在输入差分数据中和其各自的位置索引中出现的符号。

8. 如权利要求1所述的装置，其中字典编码器还包括:

增量模式编码器，其产生表示在每一顶点的每一分量的输入差分数据中是否存在预定的符号的符号标志以及表示符号的位置的位置索引；及

表格大小计算器，其计算由对应于输入差分数据的符号构成的第一符号表的大小以及符号标志的大小，并且根据第一符号表和符号标志的大小把从差分脉码调制模式编码器输入的每一顶点的每一分量的输入差分数据输出到出现模式编码器或增量模式编码器。

9. 如权利要求8所述的装置，其中增量模式编码器产生第二符号表，其中将要被编码的符号按照从具有最小绝对值的符号到具有最大绝对值的符号的次序排列，和表示对应于输入其中的差分数据的符号是否存在于第二符号

表中的符号，以及表示符号的位置的位置索引。

10. 如权利要求8所述的装置，其中表格大小计算器根据包括在输入差分数据中的符号数目以及用于编码符号所需要的比特数目而计算第一符号表的大小。

11. 如权利要求8所述的装置，其中表格大小计算器根据用于编码包括在输入差分数据中的符号所需的比特数目计算在增量模式中的符号标志的大小。

12. 如权利要求1所述的装置，其中熵编码器使用第一上下文熵编码输入其中的差分数据的最高有效比特以及使用第二上下文熵编码输入其中的差分数据的其它比特。

13. 用于解码比特数据流的装置，其中使用包括x、y和z分量的每一个顶点的坐标表示对象的每一顶点的位置的坐标内插符的关键字值数据被编码，该装置包括:

熵解码器，其通过熵解码输入的比特数据流而产生将要被字典解码的数据，数据包括差分数据的符号、表示符号位置的位置索引和差分脉码调制操作模式;

字典解码器，使用将要被字典解码的数据产生差分数据;

反差分脉码调制处理器，通过按照差分脉码调制操作模式恢复从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;及

反量化器，通过反量化量化的数据而产生恢复的关键字值数据，

其中字典解码器包括:

差分脉码调制模式解码器，其解码有关已经对每一个顶点的每一分量的数据执行的差分脉码调制操作的模式的信息;及

出现模式解码器，其根据将要被字典解码的输入差分数据的符号以及表示符号的位置的位置索引而产生每一个顶点的差分数据，

而反差分脉码调制处理器包括:

反时间差分脉码调制操作器，其对在关键帧中的顶点和在另一关键帧中的顶点之间的差分数据执行反差分脉码调制操作;

反空间差分脉码调制操作器，其对同一个关键帧中的顶点和其对应的参考顶点之间的差分数据执行反差分脉码调制操作;及

反差分脉码调制模式选择器，其根据差分脉码调制操作模式把差分数据输出到反时间差分脉码调制操作器或反空间差分脉码调制操作器。

14. 如权利要求13所述的装置，其中差分脉码调制模式解码器，根据表示差分脉码调制操作模式的组合的符号以及表示符号的位置的位置索引而恢复已经对每一顶点的每一分量的数据执行的差分脉码调制操作的模式。

15. 如权利要求13所述的装置，其中字典解码器还包括:

增量模式解码器，其根据表示在差分数据中是否存在预定的符号的符号标志而解码包括在将被字典解码的差分数据中的符号，并且根据表示符号的位置的位置索引而产生每一个顶点的差分数据；及

字典模式选择器，其从比特数据流中读出字典编码模式，并且把将被字典解码的差分数据输出到出现模式解码器或增量模式解码器。

16. 如权利要求13所述的装置，其中反差分脉码调制处理器还包括反空间-时间差分脉码调制操作器，其对结果执行反时间差分脉码调制操作，其中该结果是对当前关键帧和先前关键帧执行反空间差分脉码调制操作的结果，其中 反差分脉码调制模式选择器根据差分脉码调制操作模式把差分数据输出到反时间差分脉码调制操作器、反空间差分脉码调制操作器或反空间-时间差分脉码调制操作器。

17. 如权利要求16所述的装置，其中反时间差分脉码调制操作器、反空间差分脉码调制操作器和反空间-时间差分脉码调制操作器对输入的差分数据和在它们当中的最大和最小值执行反循环量化操作，以便扩展其范围。

18. 如权利要求13所述的装置，其中比特数据流中的使用包括x、y和z分量的每一个顶点的坐标表示的对象的每一个顶点位置的关键字值数据被编码，比特数据流包括:

字典解码的信息，其包括关于表示将要从比特数据流中熵解码的字典编码关键字值数据的差分数据的符号的信息，表示符号的位置的第一位置索引，和表示将要对第一位置索引执行的字典解码方法的字典解码模式;

反差分脉码调制操作信息，其包括表示符号的位置的第二位置索引，符号表示用于反差分脉码调制操作的反差分脉码调制操作模式的组合，其中组合用于把每一顶点的每一分量的字典解码的差分数据转换成量化的关键字值数据；及

反量化信息，被用于通过反量化量化的关键字值数据而产生恢复的关键

字数据。

19. 如权利要求18所述的装置，其中字典解码信息包括:

当字典解码模式是出现模式时，包括对应于存在于差分数据中的差分值的符号和表示符号的位置的位置索引；及

当字典解码模式是增量模式时，包括表示在差分数据中存在的差分值是否存在于预定符号表中的符号标志和表示对应于符号标志的符号的位置的位置索引。

20. 如权利要求18所述的装置，其中 反差分脉码调制操作信息还包括反差分脉码调制模式标志，其表示在表示反差分脉码调制操作模式的组合的符号当中是否存在对应于将要对每一顶点的每一分量的差分数据执行的反差分脉码调制操作的符号。

21. 如权利要求20所述的装置，其中反差分脉码调制操作模式的组合是在反时间差分脉码调制操作、反空间差分脉码调制操作以及反空间-时间差分脉码调制操作当中的任何组合。

22. 如权利要求21所述的装置，其中反差分脉码调制操作信息还包括参考顶点标志，当反差分脉码调制操作由反差分脉码调制模式标志被确定为是反空间差分脉码调制操作或反空间-时间差分脉码调制操作时，其表示对应于受到反空间差分脉码调制操作或反空间-时间差分脉码调制操作的顶点的参考顶点。

23. 如权利要求18所述的装置，其中反量化信息包括表示受到反量化操作的关键字值数据的顶点的顶点选择标志，由顶点选择标志选择的顶点的每一分量的关键字值数据当中的最小值和顶点的每一分量的关键字值数据的数据范围中的最大范围，以及用于反量化选择的顶点的关键字值数据的反量化比特大小。

24. 如权利要求23所述的装置，其中反量化信息还包括将要反量化的关键字值数据中包括的顶点数目，关键字值数据的有效位数的最大数目，在每一顶点的每一分量的数据当中的最大值中的最大值和最小值，以及在每一顶点的每一分量的数据当中的最小值中的最大值和最小值。

25. 如权利要求18所述的装置，还包括表示关键字值数据的编码模式是否为转置模式或顶点模式的标志。

26. 用于编码坐标内插符的关键字值数据的方法，坐标内插符使用包括x、

y和z分量的每一个顶点的坐标表示对象的每一顶点的位置，方法包括下述步骤：

(a) 以预定的量化比特量化坐标内插符的关键字值数据；

(b)对量化的坐标内插符的每一顶点的每一分量执行预定模式的差分脉码调制操作，并且因此产生基于每一顶点的坐标的时间变化的差分数据以及基于每一个顶点的坐标的空间变化的差分数据；

(c) 产生表示每一顶点的每一分量的差分数据的符号和已经对差分数据执行的差分脉码调制操作的模式以及表示符号的位置的位置索引；及

(d) 熵编码符号和位置索引，

其中步骤(b)包括：

(b1)对量化的坐标内插符的每一顶点的每一分量执行时间差分脉码调制操作，以便产生在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，对量化的坐标内插符的每一顶点的每一分量执行空间差分脉码调制操作，以便产生在同关键帧中的顶点之间的第二差分数据，并且对量化的坐标内插符的每一顶点的每一分量执行空间-时间差分脉码调制操作，以便产生在顶点和关键帧之间的第三差分数据；

(b2) 对第一至第三差分数据执行循环量化操作，以便降低其范围；及

(b3) 根据用于编码所需要的比特数目，选择已经循环量化的第一至第三差分数据中的一个，

而步骤(c)包括：

(c1)产生表示已经对每一顶点的每一分量的数据执行的差分脉码调制模式组合的符号和表示符号位置的位置索引；及

(c3) 产生对应于每一顶点的每一分量的输入差分数据的符号和表示符号的位置的位置索引。

27. 如权利要求26所述的方法，其中在步骤(b1)中：执行时间差分脉码调制操作，其中计算在当前关键帧中的顶点的坐标和在先前关键帧中的顶点的坐标的差值、空间差分脉码调制操作，其中计算在当前关键帧中的顶点的坐标和参考顶点的坐标之间的差值、和空间-时间差分脉码调制操作，其中计算在先前关键帧中的顶点的坐标和参考顶点的坐标之间的差值与在当前关键帧中的顶点的坐标和其对应参考顶点的坐标之间的差值之间的差分数据。

28. 如权利要求27所述的方法，其中在当前顶点受到空间差分脉码调制

操作之前的已经差分脉码调制的顶点当中，参考顶点是具有用于编码在本身和受到空间差分脉码调制操作的当前顶点之间的差分数据所需要的最小比特数目的顶点。

29. 如权利要求26所述的方法，其中在步骤(b2)中，通过对每一顶点的每一分量的差分数据和在它们当中的最大和最小值执行预定的操作而产生具有缩减大小的差分数据。

30. 如权利要求29所述的方法，其中在步骤(b2)中，根据差分数据的符号，通过把每一顶点的每一分量的差分数据的范围值加到差分数据或从差分数据减去范围值而产生循环量化的差分数据，并且把差分数据和循环量化的差分数据之间的较小的确定为循环量化的差分数据。

31. 如权利要求26所述的方法，其中在步骤(b3)中，对第一至第三差分数据执行绝对差之和操作、分散操作和熵操作中的一个，然后在已经通过对应操作的第一至第三差分数据中选择具有最小大小的差分数据。

32. 如权利要求26所述的方法，其中在步骤(c3)中，以表格的形式产生在输入差分数据中中出现的符号和其各自的位置索引。

33. 如权利要求26所述的方法，其中步骤(c3)还包括:

执行增量模式编码操作，其中产生表示在每一顶点的每一分量的输入差分数据中是否存在预定的符号的符号标志以及表示符号的位置的位置索引，其中计算由对应于差分数据的符号构成的第一符号表的大小以及符号标志的大小，并且确定将要对每一顶点的每一分量的差分数据执行的字典编码操作的步骤(c2)被进一步包括在步骤(c1)和步骤(c3)之间。

34. 如权利要求33所述的方法，其中在增量模式编码操作中，产生其中以从具有最小绝对值的符号到具有最大绝对值的符号的次序排列的将被编码的符号的第二符号表、表示在第二符号表中是否存在对应于每一顶点的每一分量的差分数据的符号的符号标志、以及表示符号的位置的位置索引。

35. 如权利要求33所述的方法，其中在步骤(c2)中，根据包括在每一顶点的每一分量的差分数据中的符号数目和用于编码符号所需要的比特数目而计算第一符号表的大小。

36. 如权利要求33所述的方法，其中在步骤(c2)中，根据用于编码包括在每一顶点的每一分量的差分数据中的符号所需的比特数目而计算在增量模式中的符号标志的大小。

37. 如权利要求26所述的方法，其中在步骤(d)中，使用第一上下文熵编码输入差分数据的最高有效比特，而使用第二上下文熵编码输入差分数据的其它比特。

38. 一种用于解码比特数据流的方法，在比特数据流中的使用包括x、y和z分量的每一个顶点的坐标表示对象的每一个顶点位置的坐标内插符的关键字值数据被编码，该方法包括步骤:

(a) 通过熵解码输入的比特数据流而产生将要被字典解码的数据，其包括差分数据的符号、表示符号位置的位置索引和差分脉码调制操作模式;

(b) 通过对差分数据的符号和位置索引执行字典解码操作，使用将要被字典解码的数据产生差分数据;

(c) 按照差分脉码调制操作模式，通过恢复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的数据; 及

(d)通过反量化量化的数据而产生恢复的关键字值数据，

其中步骤(b)包括:

(b1)解码关于已经对每一个顶点执行的差分脉码调制操作的模式的信息; 及

(b3)根据将要被字典解码的差分数据的符号以及表示符号的位置的位置索引而产生每一个顶点的差分数据，

而在步骤(c)中，按照差分脉码调制工作模式对差分数据执行预定的反差分脉码调制操作，并且反差分脉码调制操作包括:

反时间差分脉码调制操作器，其中对在关键帧中的顶点和在另一关键帧中的顶点之间的差分数据执行反差分脉码调制操作，以便产生量化的数据; 及

反空间差分脉码调制操作，其中对在同关键帧中的顶点和其对应参考顶点之间的差分数据执行反差分脉码调制操作，以便产生量化的数据。

39. 如权利要求38所述的方法，其中在步骤(b1)中，根据表示差分脉码调制操作模式的组合的符号和表示符号的位置的位置索引，恢复已经对每一顶点的每一分量的差分数据执行的差分脉码调制操作的模式。

40. 如权利要求38所述的方法，其中步骤(b3)还包括步骤: 执行增量模式解码操作，其中根据表示在差分数据中是否存在预定符号的符号标志而解码在将被字典解码的差分数据中包括的符号，并且根据表示符号的位置的位

置索引产生每一顶点的差分数据，其中从比特数据流中读出字典编码模式，并且确定对将被字典解码的数据执行的字典解码操作的步骤(b2)被进一步包括在步骤(b1)和步骤(b3)之间。

41. 如权利要求38所述的方法，其中差分脉码调制操作还包括反空间-时间差分脉码调制操作，其中对当前关键帧和先前关键帧执行反空间差分脉码调制操作的结果执行反差分脉码调制操作。

42. 如权利要求41所述的方法，其中在反时间差分脉码调制操作、反空间-时间差分脉码调制操作和反空间-时间差分脉码调制操作中，对输入的差分数据和在它们当中的最大和最小值执行反循环量化操作，以便扩展其范围。

43. 如权利要求38所述的方法，其中比特数据流中的使用包括x、y和z分量的每一个顶点的坐标表示的对象的每一个顶点位置的关键字值数据被编码，比特数据流包括:

字典解码的信息，其包括关于表示将要从比特数据流中熵解码的字典编码关键字值数据的差分数据的符号的信息，表示符号的位置的第一位置索引，和表示将要对第一位置索引执行的字典解码方法的字典解码模式;

反差分脉码调制操作信息，其包括表示符号的位置的第二位置索引，符号表示用于反差分脉码调制操作的反差分脉码调制操作模式的组合，其中组合用于把每一顶点的每一分量的字典解码的差分数据转换成量化的关键字值数据；及

反量化信息，被用于通过反量化量化的关键字值数据而产生恢复的关键字数据。

44. 如权利要求43所述的方法，其中字典解码信息包括:

当字典解码模式是出现模式时，包括对应于存在于差分数据中的差分值的符号和表示符号的位置的位置索引；及

当字典解码模式是增量模式时，包括表示在差分数据中存在的差分值是否存在于预定符号表中的符号标志和表示对应于符号标志的符号的位置的位置索引。

45. 如权利要求43所述的方法，其中 反差分脉码调制操作信息还包括反差分脉码调制模式标志，其表示在表示反差分脉码调制操作模式的组合的符号当中是否存在对应于将要对每一顶点的每一分量的差分数据执行的反差分脉码调制操作的符号。

46. 如权利要求45所述的方法，其中反差分脉码调制操作模式的组合是在反时间差分脉码调制操作、反空间差分脉码调制操作以及反空间-时间差分脉码调制操作当中的任何组合。

47. 如权利要求46所述的方法，其中反差分脉码调制操作信息还包括参考顶点标志，当反差分脉码调制操作由反差分脉码调制模式标志被确定为是反空间差分脉码调制操作或反空间-时间差分脉码调制操作时，其表示对应于受到反空间差分脉码调制操作或反空间-时间差分脉码调制操作的顶点的参考顶点。

48. 如权利要求43所述的方法，其中反量化信息包括表示受到反量化操作的关键字值数据的顶点的顶点选择标志，由顶点选择标志选择的顶点的每一分量的关键字值数据当中的最小值和顶点的每一分量的关键字值数据的数据范围中的最大范围，以及用于反量化选择的顶点的关键字值数据的反量化比特大小。

49. 如权利要求48所述的方法，其中反量化信息还包括将要反量化的关键字值数据中包括的顶点数目，关键字值数据的有效位数的最大数目，在每一顶点的每一分量的数据当中的最大值中的最大值和最小值，以及在每一顶点的每一分量的数据当中的最小值中的最大值和最小值。

50. 如权利要求43所述的方法，还包括表示关键字值数据的编码模式是否为转置模式或顶点模式的标志。

编码和解码坐标内插符的关键字值数据的方法和装置

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关键字值数据的装置和方法，其中坐标内插符使用在基于关键帧的图形动画中的包括x、y、z分量的顶点坐标表示对象的位置。

背景技术

三维(3D)动画技术已经广泛地应用在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动画技术的典型实例。

国际多媒体标准，例如用于场景(BIFS)和虚拟现实模型语言(VRML)的MPEG-4二进制格式，使用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，其包括标量内插符、位置内插符、坐标内插符、定向内插符、法线（normal line）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、直径和强度
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 渐变
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	彩色信息的线性内插	能够表示色彩中的变

符		化

在表1所示的内插符中，坐标内插符被用于表示在构成基于关键帧的动画中的3D对象的每一顶点位置上的信息，并且包括关键字和关键字值字段。关键字段使用范围在-∞和∞之间的不连续数字表示在时间轴上的每一关键帧的位置。每一个关键字值字段规定了在由每一关键字表示的在确定瞬时的构成3D对象的每一个顶点位置上的信息，并且包括三个分量x、y和z。每一个关键字值字段包括与关键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字值数据来使用线性内插符把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大量的时间。因此，最好是选择压缩内插符，以便更容易存储和发送内插符。

在已经应用在MPEG-4 BIFS中用于编码和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字值数据，如图1所示。参考图1，量化器和DPCM操作器消除关键字值数据的冗余，DPCM操作器把其操作的结果输出到熵编码器。然而，PMFC在对关键字值数据进行编码中并不是充分有效的，因为其只熵编码从一般DPCM操作获得的差分数据，并且仅考虑在动画中构成3D对象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

为解决上述和其它问题，本发明的一个方面提供一种用于编码坐标内插符的关键字值数据的方法和装置，在其中考虑动画中的3D对象顶点之间的时间相关性以及空间相关性。

本发明的另一方面提供一种用于解码编码的坐标内插符的关键字值数据的方法和装置，在其中考虑动画中的3D对象顶点之间的时间相关性以及空间相关性。

本发明的另一方面提供一种比特数据流，在其中考虑动画中的3D对象

顶点之间的时间相关性以及空间相关性编码坐标内插符的关键字值数据。

本发明的另一方面提供了一种用于DPCM操作的方法和装置，该操作被使用在根据本发明的用于编码坐标内插符的关键字值数据的方法和装置中，并且考虑动画中的3D对象的顶点之间的时间相关性以及其中的空间相关性对3D对象的坐标数据执行DPCM操作。

本发明的另一方面提供了一种用于反DPCM操作的方法和装置，其解码由根据本发明的用于DPCM操作的方法和装置产生的差分数据。

因此，为了实现本发明的上述和其它方面，其中提供了用于编码坐标内插符的关键字值数据的装置，其中坐标内插符使用包括x、y和z分量的每一个顶点的坐标表示对象的每一顶点的位置，该装置包括：量化器，其以预定的量化比特量化输入其中的坐标内插符，差分脉码调制处理器，其对量化的坐标内插符的每一顶点的每一分量执行预定模式的差分脉码调制操作，并且因此产生基于每一顶点的坐标的时间变化的差分数据以及基于每一个顶点的坐标的空间变化的差分数据； 字典编码器，其产生表示每一顶点的每一分量的差分数据的符号和已经对差分数据执行的差分脉码调制操作的模式以及表示符号的位置的位置索引； 及熵编码器，其熵编码符号和位置索引，其中差分脉码调制处理器包括：差分脉码调制操作器，对量化的坐标内插符的每一顶点的每一分量执行时间差分脉码调制操作，以便产生在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；对量化的坐标内插符的每一顶点的每一分量执行空间差分脉码调制操作，以便产生在同关键帧中的顶点之间的第二差分数据；并且对量化的坐标内插符的每一顶点的每一分量执行空间-时间差分脉码调制操作，以便产生在顶点和关键帧之间的第三差分数据；循环量化器，其对从差分脉码调制操作器输入的第一至第三差分数据执行循环量化操作，以便降低其范围；及差分脉码调制模式选择器，其根据用于进行编码所需要的比特数目选择已经循环量化的第一至第三差分数据中的一个，并且输出选择的差分数据，和其中字典编码器包括：差分脉码调制模式编码器，其产生表示已经对每一顶点的每一分量的数据执行的差分脉码调制模式组合的符号和表示符号位置的位置索引；及出现模式编码器，其产生对应于每一顶点的每一分量的输入差分数据的符号和表示符号的位置的位置索引。

为了实现本发明的上述和其它方面，其中提供了用于解码比特数据流的装置，其中使用包括x、y和z分量的每一个顶点的坐标表示对象的每一顶点的

位置的坐标内插符的关键字值数据被编码，该装置包括：熵解码器，其通过熵解码输入的比特数据流而产生将要被字典解码的数据，数据包括差分数据的符号、表示符号位置的位置索引和差分脉码调制操作模式；字典解码器，使用将要被字典解码的数据产生差分数据；反差分脉码调制处理器，通过按照差分脉码调制操作模式恢复从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；及反量化器，通过反量化量化的数据而产生恢复的关键字值数据，其中字典解码器包括：差分脉码调制模式解码器，其解码有关已经对每一个顶点的每一分量的数据执行的差分脉码调制操作的模式的信息；及出现模式解码器，其根据将要被字典解码的输入差分数据的符号以及表示符号的位置的位置索引而产生每一个顶点的差分数据，而反差分脉码调制处理器包括：反时间差分脉码调制操作器，其对在关键帧中的顶点和在另一关键帧中的顶点之间的差分数据执行反差分脉码调制操作；反空间差分脉码调制操作器，其对同一个关键帧中的顶点和其对应的参考顶点之间的差分数据执行反差分脉码调制操作；及反差分脉码调制模式选择器，其根据差分脉码调制操作模式把差分数据输出到反时间差分脉码调制操作器或反空间差分脉码调制操作器。

使用在用于解码比特数据流的装置中的反DPCM处理器最好包括反时间DPCM操作器，其对关键帧中的顶点和另一关键帧中的顶点之间的差分数据执行反DPCM操作；反空间DPCM操作器，其对在同一关键帧的顶点和其对应参考顶点之间的差分数据执行反DPCM操作；以及反DPCM模式选择器，其根据DPCM操作模式而把差分数据输出到反时间DPCM操作器或反空间DPCM操作器。

为了实现本发明的上述和其它方面，其中提供了用于编码坐标内插符的关键字值数据的方法，其中坐标内插符使用包括x、y和z分量的每一个顶点的坐标表示对象的每一顶点的位置。方法包括步骤：(a) 以预定的量化比特量化坐标内插符的关键字值数据；(b)对量化的坐标内插符的每一顶点的每一分量执行预定模式的差分脉码调制操作，并且因此产生基于每一顶点的坐标的时间变化的差分数据以及基于每一个顶点的坐标的空间变化的差分数据；(c) 产生表示每一顶点的每一分量的差分数据的符号和已经对差分数据执行的差分脉码调制操作的模式以及表示符号的位置的位置索引；及(d) 熵编码符号和位置索引，其中步骤(b)包括：(b1)对量化的坐标内插符的每一顶点的每一分量

执行时间差分脉码调制操作，以便产生在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，对量化的坐标内插符的每一顶点的每一分量执行空间差分脉码调制操作，以便产生在同关键帧中的顶点之间的第二差分数据，并且对量化的坐标内插符的每一顶点的每一分量执行空间-时间差分脉码调制操作，以便产生在顶点和关键帧之间的第三差分数据；(b2) 对第一至第三差分数据执行循环量化操作，以便降低其范围；及(b3) 根据用于编码所需要的比特数目，选择已经循环量化的第一至第三差分数据中的一个，而步骤(c)包括：(c1)产生表示已经对每一顶点的每一分量的数据执行的差分脉码调制模式组合的符号和表示符号位置的位置索引；及(c3) 产生对应于每一顶点的每一分量的输入差分数据的符号和表示符号的位置的位置索引。

为了实现本发明的上述和其它方面，提供了一种方法，用于产生构成随时间运动的对象的顶点的量化坐标数据之间的差分数据。该方法包括：执行时间DPCM操作，其中产生根据时间的推移而改变的每一顶点的坐标数据之间的差分数据；执行空间DPCM操作，其中产生在预定时刻的每一顶点和对应于顶点的参考顶点之间的差分数据；并且输出时间DPCM的差分数据和空间DPCM的差分数据之间的较小值。

为了实现本发明的上述和其它方面，提供了用于解码比特数据流的方法，比特数据流中的使用包括x、y和z分量的每一个顶点的坐标表示对象的每一个顶点位置的坐标内插符的关键字值数据被编码。该方法包括：(a) 通过熵解码输入的比特数据流而产生将要被字典解码的数据，其包括差分数据的符号、表示符号位置的位置索引和差分脉码调制操作模式；(b) 通过对差分数据的符号和位置索引执行字典解码操作，使用将要被字典解码的数据产生差分数据；(c) 按照差分脉码调制操作模式，通过恢复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的数据；及(d)通过反量化量化的数据而产生恢复的关键字值数据，其中步骤(b)包括：(b1)解码关于已经对每一个顶点执行的差分脉码调制操作的模式的信息；及(b3)根据将要被字典解码的差分数据的符号以及表示符号的位置的位置索引而产生每一个顶点的差分数据，而在步骤(c)中，按照差分脉码调制工作模式对差分数据执行预定的反差分脉码调制操作，并且反差分脉码调制操作包括：反时间差分脉码调制操作器，其中对在关键帧中的顶点和在另一关键帧中的顶点之间的差分数据执行反差分脉码调制操作，以便产生量化的数据；及反空间差分脉码调制操作，其中对在同关键帧

中的顶点和其对应参考顶点之间的差分数据执行反差分脉码调制操作，以便产生量化的数据。

为了实现本发明的上述和其它方面，提供一种方法，用于通过对顶点的坐标数据之间的差分数据执行预定的反DPCM操作而产生构成根据时间的推移改变的对象的顶点量化坐标数据。该方法包括：(a)根据包括在差分数据中的DPCM操作模式，选择将要对差分数据执行的反DPCM操作；和(b)执行选择的反DPCM操作。其中，选择的反DPCM操作包括反时间DPCM操作，其中对根据时间的推移改变的每一顶点的差分数据执行反DPCM操作；以及反空间DPCM操作，其中对在预定时刻的每一个顶点和对应于顶点的参考顶点之间差分数据执行反DPCM操作。

附图说明

通过下面结合示例性地示出一例的附图进行的描述，本发明的上述和其他目的和特点将会变得更加清楚，其中：

图1是用于编码坐标内插符的关键字值数据的常规装置的框图；

图2A是根据本发明优选实施例的用于编码坐标内插符的关键字值数据的装置的框图，而图2B是根据本发明优选实施例的用于编码坐标内插符的关键字值数据的方法的流程图；

图3A是根据本发明优选实施例的DPCM处理器的框图，而图3B是字典编码器的框图；

图4A是根据本发明的优选实施例的量化操作的流程图，图4B是DPCM操作的流程图，图4C是字典编码的流程图，而图4D是熵编码的流程图；

图5A至5C分别是说明根据本发明的量化关键字值数据、DPCM的关键字值数据和循环量化的关键字值数据的示意图；

图6A是说明根据本发明的优选实施例的DPCM模式编码的示意图，图6B是说明出现模式编码的示意图，而图6C是说明增量模式编码的示意图；

图7A是根据本发明优选实施例的用于解码坐标内插符的关键字值数据的装置的框图，而图7B是根据本发明优选实施例的用于解码坐标内插符的关键字值数据的方法的流程图；

图8A是根据本发明优选实施例的字典解码器的框图，而图8B是反DPCM处理器的框图；

图9A是根据本发明优选实施例的字典解码的流程图，而图9B是反DPCM操作的流程图；

图10是说明使用在坐标内插符中的顶点以及每一个顶点的分量的数据的比特数据流的示意图；

图11A是说明DPCM模式解码的示意图，图11B是说明出现模式解码的示意图，而图11C是说明增量模式解码的示意图；

图12至18是说明比特数据流语法的实例的示意图，其中实现在根据本发明优选实施例进行的解码过程中从比特数据流读出比特的次序；

图19是说明程序代码实例的示意图，由此实现用于解码关键字值数据的操作；

图20A和20B是用于把根据本发明的用于编码和解码坐标内插符的关键字值数据的方法的性能与用于编码和解码坐标内插符的关键字值数据的常规方法的性能相比较的示意图；及

图21A是根据本发明优选实施例的DPCM操作器的框图，而图21B是根据本发明优选实施例的反DPCM操作器的框图。

具体实施方式

在下文中，将参照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关键字值数据的装置。

图2A是根据本发明优选实施例的用于编码坐标内插符的关键字值数据的装置的框图，而图2B是根据本发明优选实施例的用于编码坐标内插符的关键字值数据的方法的流程图。

参考图2A，用于编码坐标内插符的关键字值数据的装置包括量化器300，其以预定的量化比特量化表示坐标内插符的关键字值数据的每一顶点的每一分量（component）的数据，DPCM处理器310，其对每一顶点的每一个分量的量化数据执行预定的DPCM操作，字典编码器340，其把差分数据转换成符号和位置索引，以及熵编码器350，其熵编码输入其中的差分数据的符号和位置索引。

下面参照图2B描述用于编码坐标内插符的关键字值数据的方法。

参考图2B，步骤S400中以N×M矩阵的形式把坐标内插符的关键字值数据输入到量化器300。输入的坐标内插符的关键字值数据的实例在下列表中示出。

表2

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

	z(N,1)	z(N,2)		z(N,M)

在表2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。

根据本发明用于编码坐标内插符的关键字值数据的装置以两种不同模式操作来编码坐标内插符的关键字值数据。模式之一是顶点模式，而另外一个模式是转置模式。表2中，示出将以顶点模式在量化器300中量化的关键字值数据的结构。在量化表2所示的输入关键字值数据之前，根据本发明的用于编码坐标内插符的关键字值数据的装置把输入的关键字值数据转置成M H N矩阵。在解码关键字值数据过程中反量化转置矩阵，并且把解码的关键字值数据转换成N H M矩阵，使得能够恢复与输入关键字值数据一样的关键字值数据。

参考图2B，在步骤S410中，量化器300检查从外部输入的关键字值数据的编码方式是否为转置模式。如果输入的关键字值数据的编码模式是转置模式，则输入的关键字值数据的N H M矩阵在步骤S420中被转置成M H N矩阵。

其后，量化器300以预定的量化比特量化输入其中的关键字值数据矩阵中的每一个分量的数据，并且在步骤S430把每一个分量的量化的关键字值数据输出到DPCM处理器310。在同一步骤中，量化器300把在每一个分量的输入的关键字值数据当中的最小值和在分量的数据范围当中的最大范围转换成十进制数字，并且把十进制数字输出到关键字值标题（header）编码器370。

在步骤S440中，DPCM处理器310对输入其中的量化的关键字值数据执行时间DPCM操作、空间DPCM操作和空间-时间DPCM操作，对三个不同的DPCM操作的结果，即对从三个DPCM操作获得的每一差分数据执行循环量化操作，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340。

字典编码器340产生并且输出对应于从DPCM处理器310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340产生表明已经对输入差分数据执行的DPCM操作的模式的字典符号和位置索引，把输入的差分数据转换成对应于输入差分数据的值的符号或符号标志，和表示符号的位置的位置索引，并且把符号和位置索引输出到熵编码器350。

在步骤S480中，熵编码器350通过熵编码从字典编码器340输入的符号和位置索引而产生比特数据流。

随后，参照图3A至6C更详细地描述步骤S400至S480。

参考图4A，在步骤S432中，量化器300选择在每一个分量数据当中的最大值和最小值。

量化器300使用在步骤S432中选择的最大和最小值计算分量的数据范围，并且在步骤S434中确定在分量的数据范围当中的最大范围。

量化器300使用在每一个分量的数据当中的最小值和在分量的所有的数据范围当中的最大范围量化每一个分量的关键字值数据，其以下列方程式示出。

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

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

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

方程式(1)中，i表示关键字数据，j表示顶点，而nKVQBit表示量化比特大小。此外，fMin_X、fMin_Y、fMin_Z表示在每一个分量的数据中的最小值，而fMax表示在分量数据范围中的最大范围。

量化器300把每一分量的量化关键字值数据输出到DPCM处理器310，把fMin_X、fMin_Y、fMin_Z和fMax变换成十进制数字，并且把十进制数字输出到关键字值标题编码器370。

计算机把浮点数字存储为32比特的二进制数字。为了降低用于编码所需要的比特的数量，量化器300在十进制系统中把fMin_X、fMin_Y。fMin_Z和fMax转换成其各自的尾数和指数，并且此处理由于下面方程式表示。

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

例如，浮点数12.34能够通过计算机转换成下面所示的二进制数。

0 10001010111000010100011 10000010

1 2 3

1: 符号

2: 二进制中的尾数

3: 二进制中的指数

二进制数能够遵循方程式(2)转换成随后所示的十进制数。

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

1: 符号

2: 十进制中的尾数

3: 十进制中的指数

为了把十进制中的尾数和指数包括在比特数据流中，量化器300必须计算为了表示尾数和指数所要求的比特数目。具有在-38和38之间的值的指数因此能够使用7比特连同符号一起表示。表示尾数而需要的比特的数量取决于位数。尾数的值和表示尾数而需要的比特的数量由下表列出。

表 3

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

量化器300把已经遵循方程式(2)和表3转换的在每一个分量的数据中的最小值fMin_X、fMin_Y和fMin_Z以及在分量的数据范围中的最大范围fMax输出到关键字值标题编码器370。

下面参照图3A和4B描述根据本发明的DPCM处理器和DPCM操作。

图3A是根据本发明的DPCM处理器310的框图。参考图3A，DPCM处理器310包括DPCM操作器320，其对从量化器300输入的每一分量的数据执行时间DPCM操作、空间DPCM操作和空间-时间DPCM操作，循环量化器330，其降低从DPCM操作器320输入的差分数据的范围，以及DPCM模式选择器

335，其选择从循环量化器330输入的差分数据中的一个。DPCM操作器320包括时间DPCM操作器321，其对每一个分量的量化数据执行时间DPCM操作，空间DPCM操作器323，其对每一个分量的量化数据执行空间DPCM操作，以及空间-时间DPCM操作器325，对每一个分量的量化数据执行空间-时间DPCM操作。

图4B是根据本发明优选实施例的DPCM操作的流程图。参考图4B，在步骤S442中，每一个分量的量化数据从量化器300输入到时间DPCM操作器321、空间DPCM操作器323、和空间-时间DPCM操作器325，然后在各自的操作器321、323以及325中对每一个分量的量化数据执行时间DPCM操作、空间DPCM操以及空间-时间DPCM操作。

时间DPCM操作器321计算在当前关键帧中的顶点的分量数据和在先前关键帧中的顶点的分量数据之间的差值。时间DPCM操作由下面方程式表示。

$$D_{i,j} = \widetilde{V}_{i,j} - \widetilde{V}_{i-1,j} \quad \cdots(3)$$

方程式(3)中，i表示关键字数据，j表示顶点的位置索引。

空间DPCM操作器323计算在同关键帧中的顶点之间的差值。具体地说，空间DPCM操作器323使用下面方程式计算先前顶点的熵，在当前顶点受到空间DPCM操作之前已经对先前顶点执行了空间DPCM操作。

$$Entropy(P) = -\sum_{i=0}^{N-1} P_i \log_2 P_i \quad \cdots(4)$$

在方程式(4)中，P_i表示某符号在顶点产生的概率，并且等于Fi/N，其中F_i表示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323把在顶点中的具有最低熵的顶点确定为参考顶点，并且计算在当前受到空间DPCM操作的顶点的数据和参考顶点的数据之间差分数据。空间操作由下面方程式表示。

$$D_{i,j} = \widetilde{V}_{i,j} - \widetilde{V}_{i,Ref} \quad \cdots(5)$$

空间-时间DPCM操作器325对当前关键帧的顶点执行空间DPCM操作，使用在先前关键帧的顶点中的顶点作为参考顶点对先前关键帧的顶点执行空间DPCM操作，其中其对应于的当前关键帧的参考顶点，并且计算在对应于当前关键帧的顶点的差分数据和对应于先前关键帧的顶点的差分数据之间的差分数据。换句话说，空间-时间DPCM操作器325对空间DPCM操作的结果

执行时间DPCM操作。空间-时间DPCM操作由下面方程式表示。

$$D_{i,j} = \tilde{V}_{i,j} - \{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\} \quad \cdots(6)$$

在空间DPCM操作和空间-时间DPCM操作过程中，如果$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$小于在每一分量的量化数据中的最小值，则最小值被用于空间DPCM操作和空间-时间DPCM操作。另一方面，如果$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$大于在每一分量的量化数据中的最大值，则最大值被用于空间DPCM操作和空间-时间DPCM操作。

在步骤S444中，DPCM操作器320把计算的差分数据输出到循环量化器330，并且循环量化器330对时间DPCM的差分数据、空间DPCM差分数据和空间-时间DPCM差分数据执行循环量化操作，并且把循环量化的结果输出到DPCM模式选择器335。

图5A是量化器300的输出实例的曲线图，而图5B是对图5A中所示的量化数据执行DPCM操作的结果的曲线图。如图5B所示，通过执行对量化数据的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DPCM操作。

在本发明中，假设在DPCM的差分数据中的最大值被循环连接到在DPCM的差分数据中的最小值而执行循环量化。如果对两个连续量化数据执行的线性DPCM操作的结果大于在从DPCM操作器320输出的DPCM差分数据中的最大值的一半，则从线性DPCM的结果中减去从DPCM操作器320输出的DPCM差分数据的最大范围值，以便产生具有较小绝对值的值。另一方面，如果线性DPCM的结果小于最大范围中的最小值的一半，则把最大范围值加到线性DPCM的结果，以便产生具有较小绝对值的值。

通过下面的方程式表示循环量化器330的操作。

$$CircularQuanti\ zation(X_i): \quad \cdots(7)$$

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

$$X'_i = X_i + (nQMax - nQMin + 1) \quad (otherwise)$$

$$\tilde{X}_i = \min(|X_i|, |X'_i|)$$

在方程式(7)中，nQMax表示在DPCM的差分数据中的最大值，而nQMin表示在DPCM的差分数据中的最小值。图5C示出对图5B所示的DPCM的差分数据执行循环量化的结果。

循环量化器330把循环量化的差分数据输出到DPCM模式选择器335。

在步骤S446中，DPCM模式选择器335遵循方程式(4)计算从时间DPCM操作、空间DPCM操作和空间-时间DPCM操作获得的每一DPCM差分数据的熵。

随后，DPCM模式选择器335在步骤S448中选择在时间DPCM操作、空间DPCM操作、空间-时间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，并且把对应于选择的DPCM模式的DPCM差分数据和关于DPCM模式的信息输出到字典编码器340。

随后，参照图3B和4C描述字典编码器340和其操作。

图3B是根据本发明的字典编码器340的框图。参考图3B，字典编码器340包括DPCM模式编码器342，其编码已经对输入其中的每一个顶点的每一分量的数据执行的DPCM的模式，出现模式编码器346，其产生表示每一顶点的每一分量的差分数据的值的符号和表示符号的位置的位置索引，增量模式编码器348，其产生对应于符号和表示符号位置的位置索引的符号标志，以及表格大小计算器344，其计算用于表示每一顶点的每一分量的差分数据的符号表格以及符号标志表格的大小，并且把从DPCM模式编码器342输入的差分数据输出到出现模式编码器346或增量模式编码器348。

字典编码器340检测每一顶点的每一分量的差分数据的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，其意味着顶点所有关键帧中具有相同的量化值，则字典编码器340将省略字典编码处理，并且把量化值Qmin编码成关键字值标题。

图4C是根据本发明的字典编码处理的流程图。参考图4C，在步骤S462中，在DPCM处理器310中已经产生的每一顶点的每一分量的差分数据被输入到DPCM模式编码器342中，然后DPCM模式编码器342产生表示已经对每一顶点的每一分量的数据执行的DPCM操作模式的符号，以及表示符号的位置的位置索引。

图6A是说明根据本发明的DPCM模式在编码器342中执行的编码DPCM模式的方法的示意图。参考图6A，DPCM模式编码器342预先制备表格，其中示出每一顶点的每一分量的DPCM模式和其分别的符号，如图表4所示。表4示出DPCM操作以及其对应符号的组合。表4中，时间DPCM操作、空间DPCM操作和空间-时间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)

每一顶点包括三个分量x、y和z，并且相应的DPCM操作的组合的数量是27。

如图6A所示，根据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示出的符号中的一个。DPCM模式编码器342使得顶点的DPCM模式对应于表4中示出的其各自的符号，并且设置表示符号存在于各自的顶点差分数据中的标志。

DPCM模式编码器342把对应于顶点的DPCM模式的符号排列在列中，并且以从用于具有较小幅值的符号的位置索引到用于具有最大幅值的符号的位置索引的次序产生用于符号的位置索引。

如图6A所示，对应于顶点差分数据的DPCM模式的符号的数组是(4，1，5，1，4，5)。在符号当中，1是最小的符号，并且对应于(T，T，S)。DPCM模式编码器342产生用于符号1的位置索引，使得符号的数组中出现1的位置由1表示。因此，位置索引是(0，1，0，1，0，0)。

随后，DPCM模式编码器342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，用于符号4的位置索引是(1，0，1，0)。以同样方式，DPCM模式编码器342产生用于符号5的位置索引，其对应于(T，S，T+S)。用于符号5的位置索引是(1，1)。

随后，DPCM模式编码器342把标志和位置索引输出到表格大小计算器

344。

再一次参考图3B和4C，表格大小计算器344计算用于编码在出现模式中的输入的差分数据的符号表的大小(A)，以及用于编码在递增模式中的输入的差分数据的符号标志的大小(B)，其对应于步骤S464中在预先设置的符号表中的符号。

在步骤S446中，表格大小计算器344把使用在出现模式编码器346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表示用于表示符号的比特的大小)与对应各自符号的符号标志的大小$B = 2^{AQP+1}-1$(其中AQP表示用于表示符号的比特的大小)相比较。

如果A小于B，则表格大小计算器344把每一个顶点的差分数据输出到出现模式编码器346，如果B小于A，则把差分数据输出到增量模式编码器348。

下面参照图6B描述出现模式编码器346的操作。

在步骤S468中，出现模式编码器346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6B，当顶点的输入差分数据是(3，7，3，7，-4，7，3，-4，3，7，-4，-4)，时，在步骤S468中出现模式编码器346制备表格，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346编码符号数组中的第一个符号3并且产生针对符号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346产生用于下符号7的位置索引。如图6B所示，在产生用于下符号的位置索引的过程中，前个符号的位置不被再次计算。因此，针对符号7的位置索引是(1 0 1 0 1 0 0)。

在出现模式编码器346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，因此用于符号-4的位置索引是(1 1 1)。

在图6B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符号是否仅出现一次。如果符号仅出现一次并且因此其位置索引仅包括0，则用于对应符号的bSoleBK被设置为1，并且对应符号的位置索引不被编码。出现模式编码器346把输入差分数据的符号、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350。

随后参照图6C描述根据本发明的增量模式编码器348的操作。

在步骤S469，增量模式编码器348产生表示包含在预定符号表中的符号是否存在于输入的差分数据中的符号标志以及用于符号的位置索引。

增量模式编码器348预先产生用于被期望存在于输入差分数据中的符号的表格。在表格中，以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，具有正值的符号被放置在比另一符号更高的行中。因此，符号写入表格中的次序是0，1，-1，2，-2，3，-3，.....。对应于符号表中的符号的符号标志的大小是2^{AQP+1}-1。例如，如果AQP是2，则能够由符号标志表示的符号的数量是7。如果对应于符号的值存在于差分数据中，则符号标志设置为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6C，如果输入到增量模式编码器348的差分数据是(-1，-3，-1，-3，2，-3，-1，2，-1，-3，2，2)，则存在于差分数据中的符号是(-1，2，-3)，并且因此确定符号标志为(0，0，1，1，0，0，1)。

增量模式编码器348产生用于符号的位置索引，其中符号在符号表中的定位在比其它符号高的行中。如图6C所示，增量模式编码器348设置符号-1所在的位置，其中其在符号表中的差分数据中存在的符号中排位最高，并且以0设置其它位置，使得用于符号-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348产生用于符号2的位置索引(0 0 1 0 1 0 1 1)而不考虑已经编码的符号-1的位置。最后，增量模式编码器348产生用于符号3的位置索引(1 1 1 1)而不考虑已经编码的符号-1和符号2的位置。增量模式编码器348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350。

由出现模式编码器346和增量模式编码器348产生的全部位置索引具有称为nTrueOne的标志，其表示原来的位置索引是否已经反。具体地说，如果nTrueOne被设置为0，则认为位置索引是通过反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，有可能通过反位置索引而增强算法编码效率，以便增加0的数量。

随后参照图4描述熵编码器350的操作。

根据本发明的熵编码器350熵编码从增量模式编码器348输入的表示差分数据的符号的符号标志和用于符号的位置索引，并且使用函数enodeSignedQuasiAAC()熵编码从出现模式编码器346输入的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内容产生一种自适应算法编码的比特数据流。具体地说，在encodeSignedQuasiAAC()中，不为0的第一比特被编码，随后编码其符号，并且使用零内容编码其它比特。

图4D是使用encodeSignedQuasiAAC()编码符号的处理的流程图。

在步骤S481中，熵编码器350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S482，熵编码器350从nQBit减去2，并且存储相减的结果作为可变i。

在步骤S483中，熵编码器350把符号nValue的绝对值存储作为变量val，并且对val执行次数为i的右移位(SR)操作。熵编码器350对1和SR操作的结果执行逻辑"与"操作，并且把逻辑"与"操作的结果存储作为变量比特。

在使用encodeSignedQuasiAAC()编码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S484中熵编码器350检测val是否大于1。如果val大于1，则在步骤S485中在零上下文（context）之下使用函数qf_encode()编码'比特'的值。另一方面，如果val不是大于1，则在步骤S486中在第i上下文中使用函数qf_encode()编码'比特'的值。

当val不大于1时，在步骤S487中熵编码器350再一次检验val是否为1。如果val是1，则在步骤S488中设置nValue的符号，并且在步骤S489中根据其符号和符号上下文编码nValue。

当完成针对一个比特的编码处理时，熵编码器350在步骤S490中把i减1，随后在步骤S491中检测i的当前值是否小于0。通过重复地执行S483至S490，熵编码器350熵编码输入值，直到i小于0为止。

因此，按照分配到第一比特的上下文，熵编码器350编码输入值不是0的第一比特，并且编码按照零上下文的其它比特。

随后参照图2A描述在关键字值标题编码器370中的将要被编码成关键字值标题的信息。

关键字值标题编码器370接收输入的坐标协调程序并且编码数据模式、每一关键帧中的顶点的数目、用于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。

关键字值标题编码器370编码量化比特数、每一顶点的每一分量的关键字值数据中的最小值和每一顶点的每一分量的数据范围中的最大数据范围、以及每一顶点的每一分量的量化数据中的最大和最小值。

关键字值标题编码器370从DPCM处理器310接收已经对每一顶点的每一分量的数据执行的DPCM操作的模式，从字典编码器340接收字典编码模式，并且编码DPCM操作模式和字典编码方式。

随后参照图7A和7B描述根据本发明的用于解码编码的坐标内插符的装置和方法。

图7A是根据本发明优选实施例的用于解码编码的坐标内插符的装置的框图，而图7B是根据本发明优选实施例的用于解码编码的坐标内插符的方法的流程图。

参考图7A，根据本发明的用于解码编码的坐标内插符的装置包括熵解码器800，其熵解码输入的比特数据流并且因此产生将要被字典解码的数据，其中数据包括DPCM差分数据的符号、符号标志、用于符号的位置索引和DPCM操作模式；字典解码器810，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DPCM处理器830，根据DPCM操作模式而对差分数据执行预定的反DPCM操作来产生量化数据；反量化器850，通过反量化量化数据而产生恢复的关键字值数据；以及关键字值标题解码器870，从输入的比特数据流解码用于解码坐标内插符所需要的信息，并且把信息输出到字典解码器810、反DPCM处理器830和反量化器850。

下面参照图7B描述根据本发明的用于解码编码的坐标内插符的方法。

在步骤S910中，其中的坐标内插符被编码的比特数据流被输入到熵解码器800，然后在步骤S920中，熵解码器800解码输入的比特数据流。如果输入的比特数据流已经以出现模式编码，则熵解码器800把每一顶点的符号和其位置索引输出到字典解码器810。另一方面，如果输入的比特数据流已经以增量模式编码，则熵解码器800把表示符号的存在的符号标志和用于符号的位置索引输出到字典解码器810。

在步骤S930中，根据输入的字典编码模式，字典解码器810通过解码从熵解码器800以出现模式输入的符号和位置索引或通过解码从熵解码器800以增量模式输入的符号标志和位置索引而产生差分数据，并且把产生的差分数据输出到反DPCM处理器830。

根据输入差分数据的解码的DPCM工作模式，反DPCM处理器830在步骤S940中通过对从字典解码器810输入的差分数据执行反时间DPCM操作、反空间DPCM操作以及反空间-时间DPCM操作中的一个而产生量化的关键字值数据，并且把量化的关键字值数据输出到反量化器850。

在步骤S950中，反量化器850使用从关键字值标题解码器870输入的每一个分量的数据中的最小值和最大数据范围而反量化从反DPCM处理器830输入的量化的关键字值数据。

在步骤S960中，反量化器850检查反量化的关键字值数据矩阵是否已经在编码处理过程中转换成转置矩阵，并且如果反量化的关键字值数据的矩阵已经转置，则在步骤S965中反地变换转置矩阵。

在步骤S970中，反量化器850输出恢复的坐标内插符的关键字值数据。

随后，参照图8A至9B更详细地描述用于解码编码的坐标内插符的装置和方法。

熵解码器800首先从输入比特数据流解码表示DPCM模式的比特数据流，然后解码包括bSelFlag、nKVACodingBit、nQMin、和nQMax的数组。

在编码处理中，首先分别把bSelFlag和nKVACodingBit设置为1和0。如果bSelFlag被解码成1，则熵解码器800解码nKVACodingBit、nQMin和nQMax。另一方面，如果bSelFlag被解码成0，则熵解码器800只解码nQMin。

在解码数据bSelFlag、nKVACodingBit、nQMin和nQMax的数组以后，熵解码器800解码表示字典编码模式的nDicModeSelect。根据nDicModeSelect的值，将要被解码的比特数据流被分成下面将被描述的两个不同种类。

图10是示意图，说明坐标内插符的每一顶点和每一个顶点的分量数据的比特数据流的结构。如图10所示，如果nDicModeSelect是0，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，则比特数据流包括已经在增量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19所示的程序代码实现的函数decodeSignedQuasiAAC()。在函数encodeSignedQuasiAA()中，使用涉及输入值和其符号的上下文解码一种自适应算法编码的比特数据流。具体地说，在函数decodeSignedQuasiAAC()中，使用零上下文解码符号位之后的比特。熵解码器800把解码的数据输出到字典解码器810。

图8A是根据本发明的字典解码器810的框图，而图9A是字典编码的流程图。

如图8A所示，字典解码器810包括DPCM模式解码器812，其恢复输入其中的每一顶点的DPCM模式；字典模式选择器814，其选择输入每一顶点的字典解码模式；出现模式解码器816，从字典模式选择器814接收每一顶点的每一分量的符号和针对符号的位置索引，并且恢复差分数据；以及增量模式解码器818，其从字典模式选择器814接收符号标志和用于符号的位置索引，并且恢复差分数据。

参考图9A，在步骤S931中，包括符号、符号标志和位置索引的每一顶点的熵解码的分量数据被输入到DPCM模式解码器812。

在字典解码的差分数据被输出到反DPCM处理器830之前，在步骤S932中，DPCM模式解码器812解码反DPCM操作的模式，其中反DPCM操作是在反DPCM处理器830中将对每一顶点的每一分量的差分数据执行的操作。

随后参考图11A描述DPCM模式解码。

除了表示每一顶点的每一分量的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812接收DPCM模式标志并且按照输入的位置索引把对应于DPCM模式标志的符号记录在数组中。

例如，如图11A所示，对应于输入DMCM模式标志的符号是1(T T S)，4(T S S)，和5(T S T+S)，并且其各自的索引是(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)恢复数据数组（array）(X 1 X 1 X X)，使用符号4和其位置索引(1 0 1 0)恢复数据数组(4 1 X 1 4 X)，以及使用符号5和其位置索引(1 1)恢复数据数组(4 1 5 1 4 5)。

恢复的数据数组(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)的组合的数组。因此，有可能表明已经根据恢复的数据数组对每一顶点的每一分量执行了哪一种DPCM。

DPCM模式解码器812把每一顶点的每一分量的差分数据连同解码的DPCM模式信息一起输出到字典模式选择器814。

在步骤S934中，根据每一顶点的每一分量的nDicModeSelect的值，字典模式选择器814把从DPCM模式解码器812输入的每一顶点的分量数据输出到

出现模式解码器816或增量模式解码器818。

如果nDicModeSelect是0，则字典模式选择器814把顶点的分量数据输出到出现模式解码器816，而如果nDicModeSelect是1，则字典模式选择器814把顶点的分量数据输出到增量模式解码器818。

在步骤S936中，出现模式解码器816把每一个分量的符号数据和位置索引恢复成差分数据。

图11B是说明出现模式解码的实例的示意图。参考图11B，出现模式解码器816从字典模式选择器814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，则出现模式解码器816通过在数据数组中由其各自的位置索引表示的各自的位置插入输入符号而恢复差分数据。

例如，出现模式解码器816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)。

在按照位置索引(0 1 0 0 0 1 0 1 0 0 0)的差分数据数组中出现模式解码器816记录第一符号3。因此，通过把符号3插入在差分数据数组中的对应于位置索引(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816恢复随后的符号7。在恢复符号7的过程中，不考虑差分数据数组中的符号3的位置，使得用于符号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816在差分数据数组中的没被符号3占用的位置中的第一位置记录符号7，然后在差分数据数组中对应于位置索引(1 0 1 0 1 0 0)中的1处在的位置记录符号7。因此，在恢复符号7之后，差分数据数组是(3 7 3 7 X 7 3 X 3 7 X X)。

出现模式解码器816按照索引(1 1 1)恢复符号-4，并且因此产生差分数据数组是(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输入符号，并且没有用于输入符号的位置索引。因此，出现模式解码器816把输入符号记录在空白差分数据数组中的第一位置，并且执行用于恢复下一个符号的处理。

在步骤S936中，增量模式解码器818把每一个分量的符号标志和位置索

引恢复成差分数据。随后参考图11C描述增量模式解码。

增量模式解码器818从字典模式选择器814接收表示在差分数据中是否存在符号的符号标志、表示位置索引是否已经反的nTrueOne以及位置索引。

增量模式解码器818根据输入符号标志解码在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的符号高的一行中。符号标志的大小是$2^{nKVCodingBit+1}$-1，其中nKVCodingBit表示在熵解码器800中解码的量化比特的数量。因此，如果符号标志是(0 0 1 1 0 0 1)，则增量模式解码器818解码存在于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818在差分数据数组中的对应于位置索引 (1 0 1 0 0 0 1 0 1 0 0 0)中1处在的位置的位置上记录符号-1，使得产生的数据数组是(-1 X -1 X X X -1 X -1 X X X)。

随后，增量模式解码器818通过在差分数据数组中的对应于在位置索引(0 0 1 0 1 0 1 1)中1处在的位置的位置中记录2而恢复符号2。在恢复符号2的过程中，不考虑差分数据数组中的第一个符号-1的位置，使得产生的差分数据数组是(-1 X -1 X 2 X -1 2 -1 X 2 2)。

增量模式解码器81通过在差分数据数组中的对应于在位置索引(1 1 1 1)中的1处在的位置的位置上记录-3而恢复符号-3，使得产生的差分数据数组是(-1 -3 -1 -3 2 -3 -1 -3 2 2)。

在步骤S939中，出现模式解码器816和增量模式解码器818恢复每一顶点的每一分量的差分数据，并且把恢复的差分数据输出到反DPCM处理器830。

图8B是根据本发明的反DPCM处理器830的框图，而图9B是反DPCM操作的流程图。

参考图8B，根据本发明的反DPCM处理器830包括反时间DPCM操作器842，其对输入的差分数据执行反时间DPCM操作和反循环量化操作，然后输出坐标内插符的量化的关键字值数据；反空间DPCM操作器844，其对输入的差分数据执行反空间DPCM操作和反循环量化操作，随后输出量化的关

键字值数据；反空间-时间DPCM操作器846，对输入的差分数据执行反空间-时间DPCM操作和反循环量化操作，随后输出量化的关键字值数据；以及反DPCM模式选择器835，其把输入其中的差分数据输出到反时间DPCM操作器842、反空间DPCM操作器844和反空间-时间DPCM操作器846中的一个上。

参考图9B，在步骤S942中，反DPCM模式选择器835根据在DPCM模式解码器812中恢复的每一顶点的每一分量的DPCM操作模式确定将对输入其中的差分数据执行的反DPCM操作，并且按照反DPCM操作模式而输出每一顶点的每一分量的输入的差分数据。

DPCM操作器842、844和846中的每一个同时对输入其中的差分数据执行反DPCM操作和反循环量化操作。

在步骤S944中，反时间DPCM操作器842遵循方程式(8)对输入的差分数据执行反时间DPCM操作，在步骤S946中，反空间DPCM操作器844遵循方程式(9)对输入的差分数据执行反空间DPCM操作，在步骤S948中，反空间-时间DPCM操作器846遵循方程式(10)对输入的差分数据执行反空间-时间DPCM操作。

$$\tilde{V}_{i,j} = D_{i,j} + \tilde{V}_{i-1,j} \quad \cdots(8)$$

$$\tilde{V}_{i,j} = D_{i,j} + \tilde{V}_{i,Ref} \quad \cdots(9)$$

$$\tilde{V}_{i,j} = D_{i,j} + \{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\} \quad \cdots(10)$$

在方程式(8)至(10)中，$\tilde{V}_{i,j}$表示在第i关键帧中的第j个顶点的量化关键字值数据，$D_{i,j}$表示在第i关键帧中的第j个顶点的差分数据，而Ref表示参考顶点。

方程式(9)和(10)中，如果$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$小于每一分量的量化关键字值数据的最小值，则使用最小值而不使用$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$。如果$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$大于每一个分量的量化关键字值数据中的最大值，则使用最大值而不使用$\tilde{V}_{i,Ref}$或$\{\tilde{V}_{i-1,j} + (\tilde{V}_{i,Ref} - \tilde{V}_{i-1,Ref})\}$。

DPCM操作器842、844和846中的每一个都使用方程式(11)执行反DPCM操作并且同时执行反循环量化操作，以便扩展已经在编码处理过程中降低的

差分数据的范围。

$$InverseCircularQuantization(\widetilde{X}_i): \qquad \cdots(11)$$

$$X_i' = \widetilde{X}_i - (nQMax - nQMin + 1) \qquad (if\ \widetilde{X}_i \geq 0)$$

$$X_i' = \widetilde{X}_i + (nQMax - nQMin + 1) \qquad (otherwise)$$

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

$$\hat{X}_i = \hat{X}_{i-1} + X_i' \qquad (f\ nQMin \leq \hat{X}_{i-1} + X_i' \leq nQMax)$$

在方程式(11)中，$\widetilde{X}_i$ 是与 D_{ij} 相同的输入值，$\hat{X}_{i-1}$ 是与 $\widetilde{V}_{i,Ref}$ 或 $\{\widetilde{V}_{i-1,j} + (\widetilde{V}_{i,Ref} - \widetilde{V}_{i-1,Ref})\}$ 一样的先前反循环量化的值。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S949中，反DPCM处理器830把已经反DPCM的和反循环量化的每一顶点的每一分量的关键字值数据输出到反量化器850。

参考图7B，遵循方程式(2)，反量化器850把在从关键字值标题解码器870输入的输入分量数据中的最小值fMin_X、fMin_Y和fMin_Z和最大范围值fMax转换成二进制数，并且通过把fMin_X、fMin_Y、fMin_Z和fMax代入到方程式(12)中而反量化从反DPCM处理器830输入的量化关键字值数据。

$$\hat{V}_{i,j,x} = fMin\_X + \frac{\widetilde{V}_{i,j,x}}{2^{nKVQBits} - 1} \times fMax \qquad \cdots(12)$$

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

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

方程式(12)中，nKVQBits表示用于反量化的量化比特的大小。

反量化器850必须以表2示出的矩阵的形式输出每一顶点的每一分量的反量化的关键字值数据。为了实现此目的，在步骤S960中，反量化器850在输出反量化的关键字值数据之前检验反量化的关键字值数据的模式是否为转置模式。如果反量化的关键字值数据的模式是转置模式，则在步骤S965中，反量化器850通过反变换转置矩阵而产生并且输出坐标内插符的解码的关键字值数据。

随后，参照图12至18描述用于解码编码的比特数据流和使用在这种程序代码中的变量的SDL程序代码。

图12示出用于读出压缩坐标内插符的比特数据流的最高类（class）。

CoordIKeyValueHeader和CoordIKeyValue是用于读出对应于一般坐标内

插符节点的关键字值字段数据的关键字值信息的类。函数qf_start()被用于在读出AAC-编码的数据之前初始化算法解码器。

图13表示使用用于解码关键字值数据所需要的关键字值标题信息用于产生比特数据流的程序代码。

在关键字标题数据被解码之后解码关键字值标题数据。关键字值标题包括顶点数目、用于量化的关键字值数据参数和用于量化的最大和最小值。bTranspose是表示是否为转置模式或顶点模式的标志。如果bTranspose是1，则在解码处理中选择转置模式。另一方面，如果bTranspose是0，选择顶点模式。nKVQBit是用于通过反量化恢复浮点数字的量化比特。nCoordQBit是用于表示代表顶点数量nNumberOfCoord的比特的大小。在反量化后使用nKVDigit，并且表示关键字值数据有效位的最大值数。类KeyValueMinMax恢复用于反量化的最小值和被分成尾数和指数的最大数据范围。标题信息的其它部分包括在最大值以及每一顶点的每一分量的量化关键字值数据中的最大值和最小值中的最小值。具体地说，nXQMinOfMax表示在每一个顶点的x分量的量化关键字值数据中的最大值当中的最小值。nNumKeyCodingBit代表表示关键字数据数的nNumberOfKey的比特大小。需要包括nXQMinOfMax和nNumberOfKey信息来解码关键字值数据。

图14A和14B是表示根据本发明用于实现解码DPCM模式的装置的程序代码。图14A和14B所示每一变量的含意如下。

nDPCMMode表示每一顶点的每一分量(x，y，z)的DPCM模式。当nDPCMMode被设置为1，2，或3时，分别表示时间DPCM模式、空间DPCM模式，或时间空间的DPCM模式。

bSelFlag是用于选择每一顶点的每一分量的标志。使用字典编码器340仅编码其bSelFlag被设置为1的每一个顶点的分量。selectionFlagContext是用于读出bSelFlag的上下文。

nKVACodingBit表示用于每一顶点的每一分量的编码比特。aqpXContext、aqpYContext和aqpZContext是分别用于X轴、Y轴和Z轴的上下文，被用于读出nKVACodingBit。

nRefVertex是用于全部顶点的参考顶点的索引。refContext是用于读出nRefVertex的上下文。

nQMin表示在每一顶点的每一分量的DPCMed差分数据中的最小值。

qMinContext表示用于读出nQMin的上下文，而qMinSignContext是用于读出nQMin的符号的上下文。

nQMax表示在每一顶点的每一分量的DPCMed差分数据中的最大值。qMaxContext是用于读出nQMax的上下文，而aMaxSignContext是用于读出nQMax的符号的上下文。

图15是说明根据本发明的用于解码DPCM模式的程序代码，并且图15所示的每一变量的含意如下。

bAddressOfDPCMMode表示每一DPCM字典符号的用法，其通过用于在DPCM字典表格中的每一分量的DPCM模式的组合构成。每一顶点包括三个分量，并且在每一个顶点的分量中可能存在三个不同种类的DPCM，T、S和T+S模式。如图3所示，有27个表示三个DPCM模式的组合的字典符号。dpcmModeDicAddressContext是用于读出bAddressOfDPCMMode的上下文。

bDPCMIndex表示哪个DPCM符号已经用于每一个顶点。dpcmModelDicIndexContext是用于读出bDPCMIndex的上下文。

图16是表示根据本发明用于解码字典编码模式的程序代码的示意图，图16示出每一变量的含意如下。

dDicModeSelect表示已经在字典编码过程中使用的字典编码模式。当dDicModeSelect是1时，意味着字典编码模式是增量模式。另一方面，如果dDicModeSelect是0，则意味着字典编码模式是出现模式。

图17是表示根据本发明用于实现增量模式解码方法的程序代码的示意图，图17示出每一变量的含意如下。

bAddress表明是否已经使用了表示量化关键字值的增量模式字典符号。使用在增量模式表格中的符号的数量是$2^{nKVCodingBit+1}-1$。dicAddressContext是用于读出bAddress的上下文。

nTrueOne表示索引数据是否已经反。当nTrueOne是1时，在位置索引中的1值被认为是表示符号的位置的实际值。当nTrueOne是0时，在位置索引中的0值被认为是表示符号的位置的实际值。

bAddrIndex表示用于每一顶点的每一分量的增量模式符号。dicIndexContext是用于读出bAddrIndex的上下文。

图18是说明根据本发明的用于实现出现模式解码方法的程序代码，并且图18所示的每一变量的含意如下。

nQKV包括出现模式符号，它是量化的关键字值数据。kvXContext、kvYContext和kvZContext是用于读出nQKV的上下文，而kvSignContext是用于读出nQKV的符号的上下文。

bSoleKV表示是否解码符号在差分数据中仅出现一次。如果解码符号在差分数据中仅出现一次，则soleKV被设置为1。dicSoleKVContext是用于读出bSoleKV的上下文。

bDicIndex表示哪个字典符号已经被用于每一顶点的每一分量。dicIndexContext是用于读出bDicIndex的上下文。

图20A是表示执行根据本发明的用于编码和解码坐标内插符的关键字值数据的方法和传统的MPEG-4BIFS PMFC方法的执行测试性能的结果的速率失真曲线。具体地说，图20A示出在失真度和在编码坐标内插符的38个关键字值数据情况下的编码比特率之间的关系。如图20A所示，用于编码和解码坐标内插符的关键字值数据的方法具有比传统的MPEG-4BIFS PMFC方法更高的效率。

图20B包括三个示意图(a)、(b)和(c)。具体地说，在图20B中，(a)表示动画数据，(b)表示根据本发明编码/解码的动画数据，而(c)表示遵循常规编码/解码方法而被编码/解码的动画数据。如图20B所示，根据本发明用于编码和解码坐标内插符的关键字值数据的方法能够提供比常规编码/解码方法更高质量的动画，其大为接近原始的动画。

上面已经参照附图描述了根据本发明用于编码/解码坐标内插符的关键字值数据以便显示基于关键帧的动画的方法和装置，其中示出了本发明的优选实施例。本专业技术人员显然清楚，采用在本发明优选实施例中的DPCM操作不局限于只应用到坐标内插符的关键字值数据，而是同样能应用到包括多个分量来描述三维对象的顶点数据。

图21A是根据本发明的DPCM操作器的框图。参考图21A，根据本发明的DPCM操作器包括时间DPCM操作器2010，其产生构成随时间的推移而变化的3D对象的在预定时刻顶点数据和另一预定的时刻的顶点数据之间的差分数据；空间DPCM操作器2020，其产生在顶点的数据和在预定时刻的参考顶点的数据之间差分数据；以及DPCM模式选择器2030，其输出在从时间DPCM操作器2010输入的差分数据和从空间DPCM操作器2020输入的差分数据之间的较小的差分数据。

根据本发明的DPCM操作器最好还包括空间-时间DPCM操作器2040，其计算在顶点和关键帧中的参考顶点之间的差分数据之间的差分数据，以及通过对空间DPCM操作的结果执行时间DPCM操作而计算其在另一关键帧中的对应的差分数据。即使在提供有空间-时间DPCM操作器的情况中，DPCM模式选择器2030仍然输出在从时间DPCM操作器2010输入的差分数据、从空间DPCM操作器2020输入的差分数据和从空间-时间DPCM操作器2040输入的差分数据当中的最小的差分数据。

根据本发明的DPCM操作器的单元的操作与上述的DPCM处理器的对应单元的操作相同。

根据本发明DPCM操作器接收来自外部的构成3D对象顶点的量化坐标数据。

当顶点表示当前对象时，时间DPCM操作器2010使用方程式(3)计算在顶点的坐标数据之间的差分数据，并且当顶点表示先前对象时计算顶点的坐标数据。

空间DPCM操作器2020使用方程式(5)计算已经执行了DPCM操作的顶点和存在于同一个时间轴上作为DPCMed顶点的之间的差分数据，选择具有最小差分数据的顶点作为参考顶点，并且输出差分数据。

DPCM模式选择器2030计算从时间DPCM操作器2010输入的差分数据和从空间DPCM操作器2020输入的差分数据的大小，并且连同DPCM操作信息一起输出具有较小大小的差分数据。

根据本发明的优选实施例，可以进一步包括在DPCM操作器中的空间-时间DPCM操作器2040，使用方程式(6)对3D对象的量化坐标数据执行上述的空间DPCM操作，并且对当前顶点执行空间DPCM操作以及对先前顶点执行空间DPCM操作的结果执行上述的时间DPCM操作。

包括在根据本发明优选实施例的DPCM操作器中的循环量化器2050使用方程式(7)降低输入其中的差分数据的范围。

图21B是反DPCM操作器的框图，反DPCM操作器把由根据本发明的DPCM操作器产生的差分数据转换成量化坐标数据。

根据本发明的反DPCM操作器包括反时间DPCM操作器2110，其对在预定时刻的顶点的数据和在另一预定时刻的顶点的数据之间的差分数据执行反DPCM操作；反空间DPCM操作器2120，其对顶点的数据和在预定瞬时的参

考顶点的数据之间的差分数据执行反空间DPCM操作；以及反DPCM模式选择器2100，其根据已经对差分数据执行的DPCM操作的模式，把差分数据输出到反时间DPCM操作器2110或反空间DPCM操作器2120。

根据本发明的反DPCM操作器最好还包括反空间-时间DPCM操作器2130，对当前反空间DPCM操作的结果以及先前反空间DPCM操作的结果执行反空间-时间DPCM操作。

根据本发明的反DPCM操作器的单元的操作与上述的反DPCM处理器的对应单元的操作相同。

将要被恢复成量化坐标数据的差分数据被输入到反DPCM模式选择器2100。随后，反DPCM模式选择器2100识别已经对包括在输入的差分数据中的每一顶点的分量数据执行了哪一种DPCM，并且把每一个顶点的分量数据输出到反时间DPCM操作器2110、反空间DPCM操作器2120和反空间-时间DPCM操作器2130。

反时间DPCM操作器2110遵循方程式(8)对输入其中的差分数据执行反时间DPCM操作，反空间DPCM操作器2120遵循方程式(9)对输入其中的差分数据执行反空间DPCM操作，并且反空间-时间DPCM操作器2130遵循方程式(10)对输入其中的差分数据执行空间-时间DPCM操作。

如果输入的差分数据已经被循环量化，则DPCM操作器2110、2120和2130的每一个都使用方程式(11)对其各自的反DPCM的差分数据执行反循环量化操作，以便扩展各自的反DPCM差分数据的范围。

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，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。计算机可读记录介质能够分散到经过网络连接的计算机系统，并且计算机能够以分散的方法读出记录介质。

考虑了在不同关键帧中的顶点的坐标数据之间的差分数据以及坐标内插符的顶点的坐标数据之间的差分数据，根据本发明的用于编码坐标内插符的关键字值数据的方法和装置通过编码坐标内插符的关键字值数据而具有高编码效率。

此外，根据本发明的用于编码坐标内插符的关键字值数据的方法和装置

通过使用对应于差分数据的值的符号和针对其各自符号的位置索引表示差分数据而具有更高的编码效率。

虽然已经参照几个优选实施例展示和描述了本发明，但是本领域技术人员将理解，在不背离本所附的如权利要求定义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。

本发明不限于上述实施例，在不脱离本发明范围的情况下，可以进行各种变形和修改。

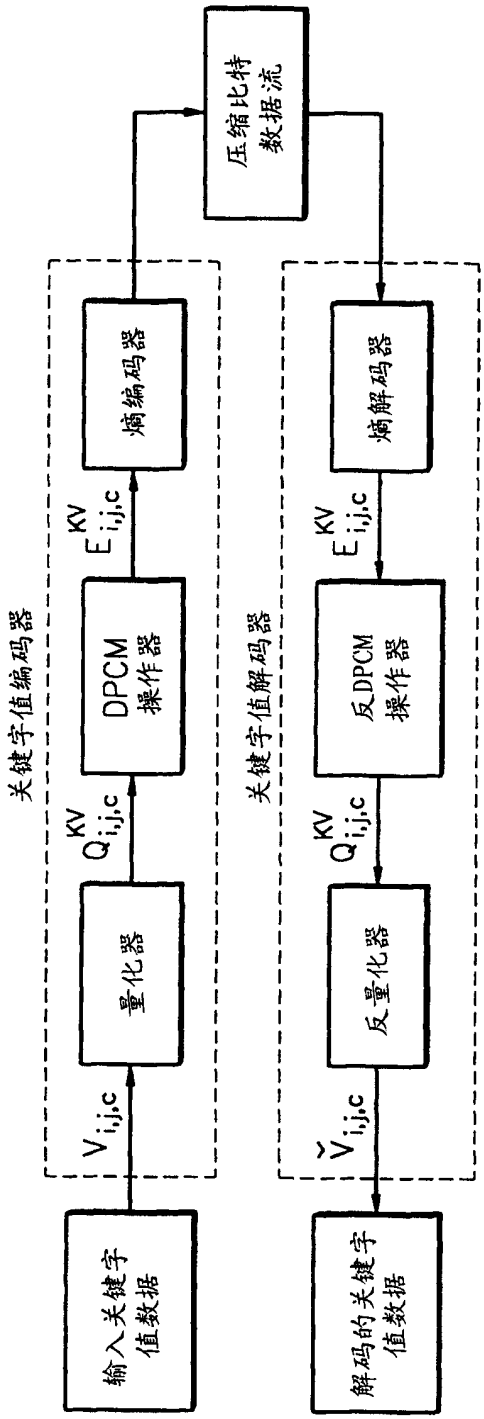

关键字值编码器
关键字值解码器
输入关键字值数据
$V_{i,j,c}$
量化器
$Q^{KV}_{i,j,c}$
DPCM操作器
$E^{KV}_{i,j,c}$
熵编码器
压缩比特数据流
熵解码器
$E^{KV}_{i,j,c}$
反DPCM操作器
$Q^{KV}_{i,j,c}$
反量化器
$\tilde{V}_{i,j,c}$
解码的关键字值数据

图 1

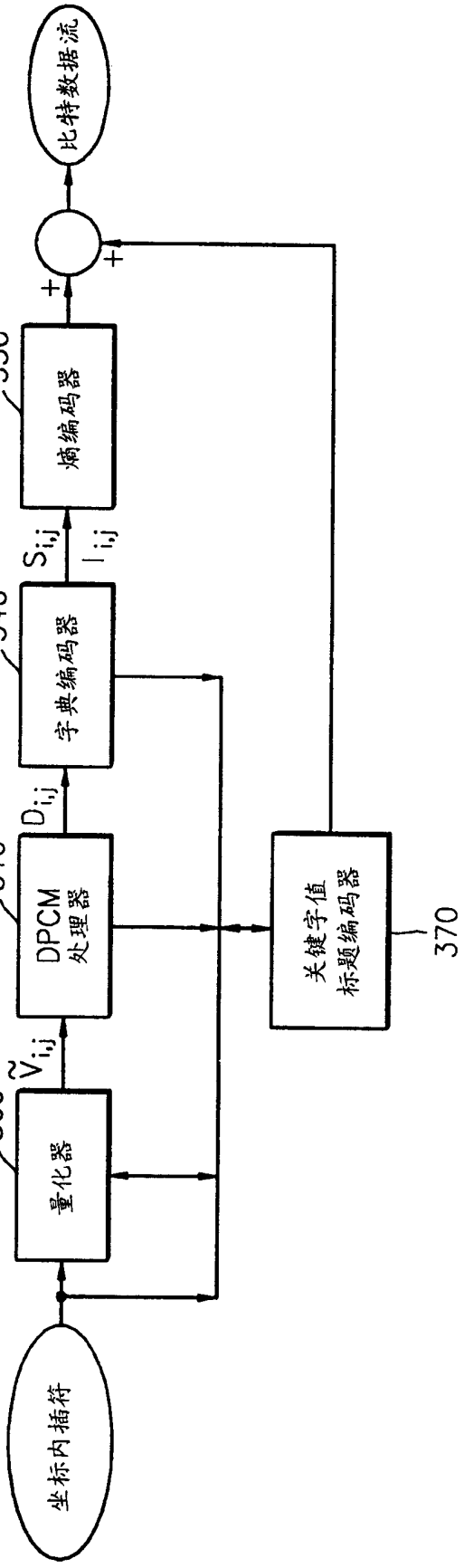

坐标内插符
300
量化器
$\tilde{V}_{i,j}$
310
DPCM
处理器
$D_{i,j}$
340
字典编码器
$S_{i,j}$
$I_{i,j}$
350
熵编码器
+
+
比特数据流
370
关键字值
标题编码器

图 2A

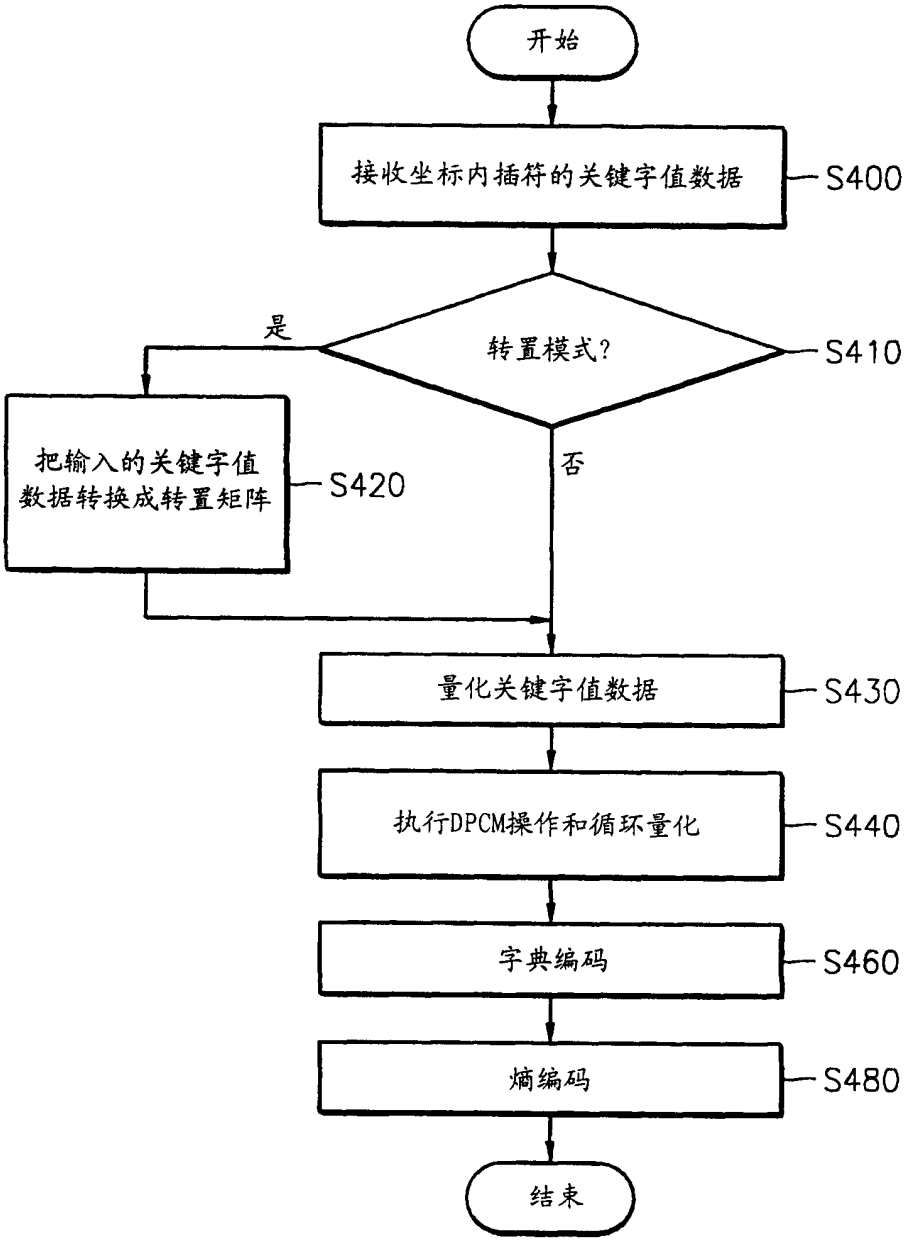

开始
接收坐标内插符的关键字值数据
S400
转置模式?
S410
是
否
把输入的关键字值数据转换成转置矩阵
S420
量化关键字值数据
S430
执行DPCM操作和循环量化
S440
字典编码
S460
熵编码
S480
结束

图 2B

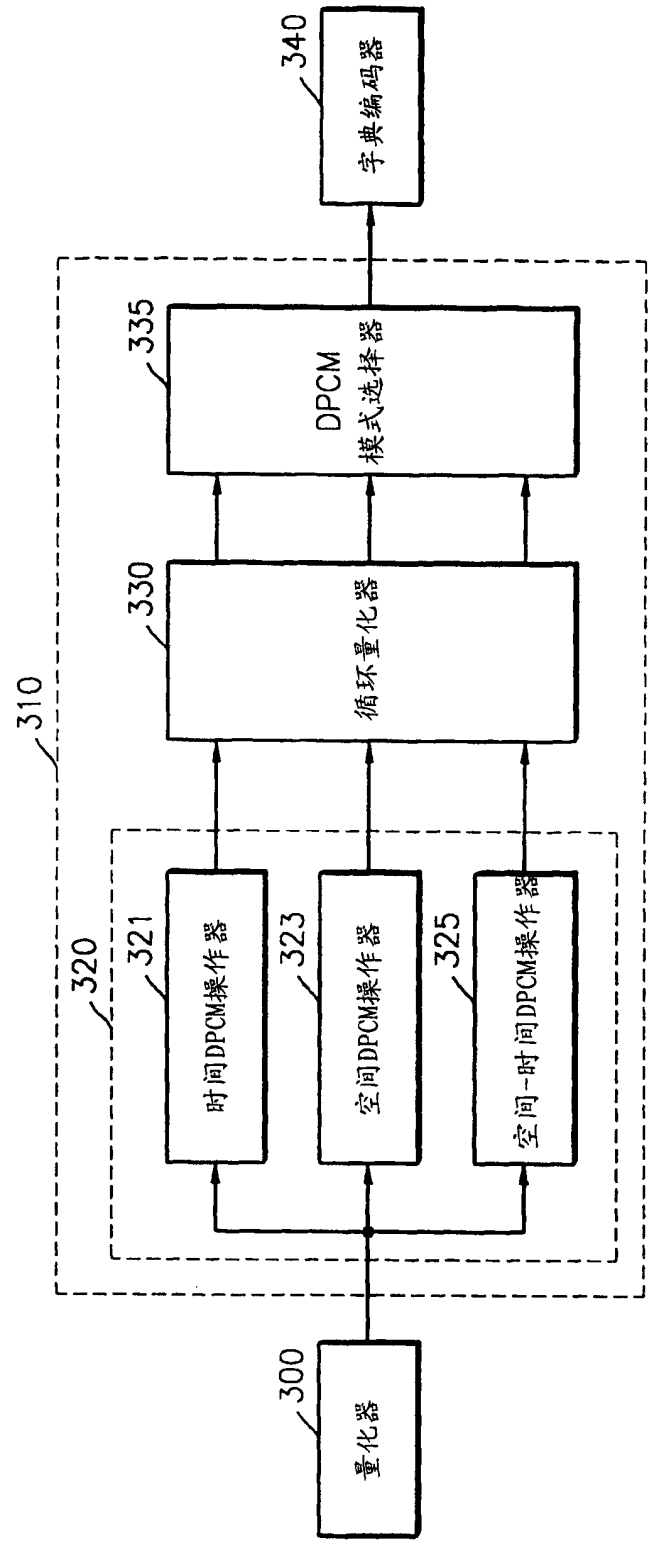

340
字典编码器
335
DPCM
模式选择器
330
循环量化器
310
320
321
时间DPCM操作器
323
空间DPCM操作器
325
空间-时间DPCM操作器
300
量化器

图 3A

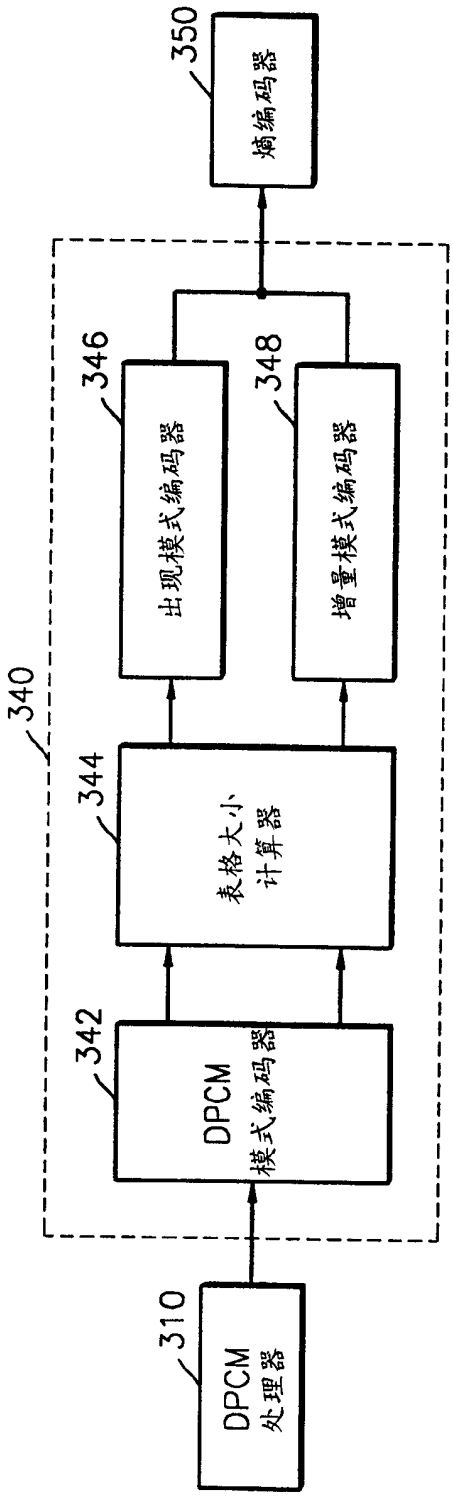

350
熵编码器
346
出现模式编码器
348
增量模式编码器
340
344
表格大小
计算器
342
DPCM
模式编码器
310
DPCM
处理器

图 3B

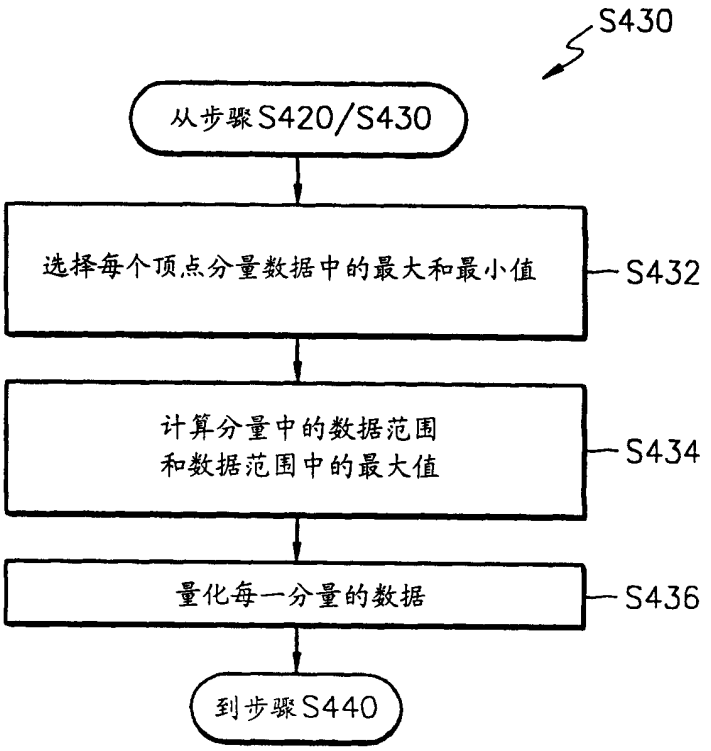

S430
从步骤S420/S430
选择每个顶点分量数据中的最大和最小值
S432
计算分量中的数据范围
和数据范围中的最大值
S434
量化每一分量的数据
S436
到步骤S440

图 4A

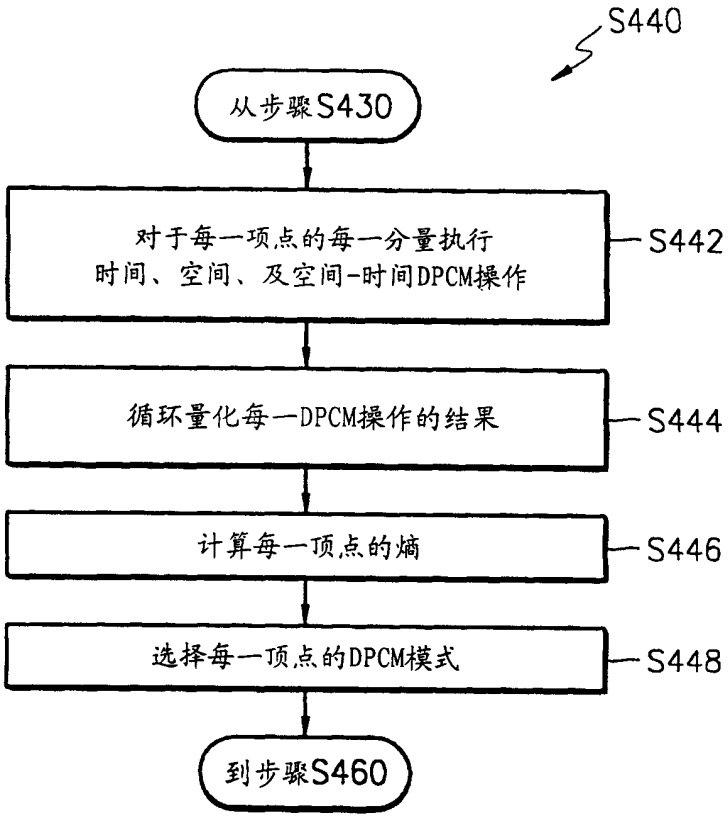

S440
从步骤S430
对于每一顶点的每一分量执行
时间、空间、及空间-时间DPCM操作
S442
循环量化每一DPCM操作的结果
S444
计算每一顶点的熵
S446
选择每一顶点的DPCM模式
S448
到步骤S460

图 4B

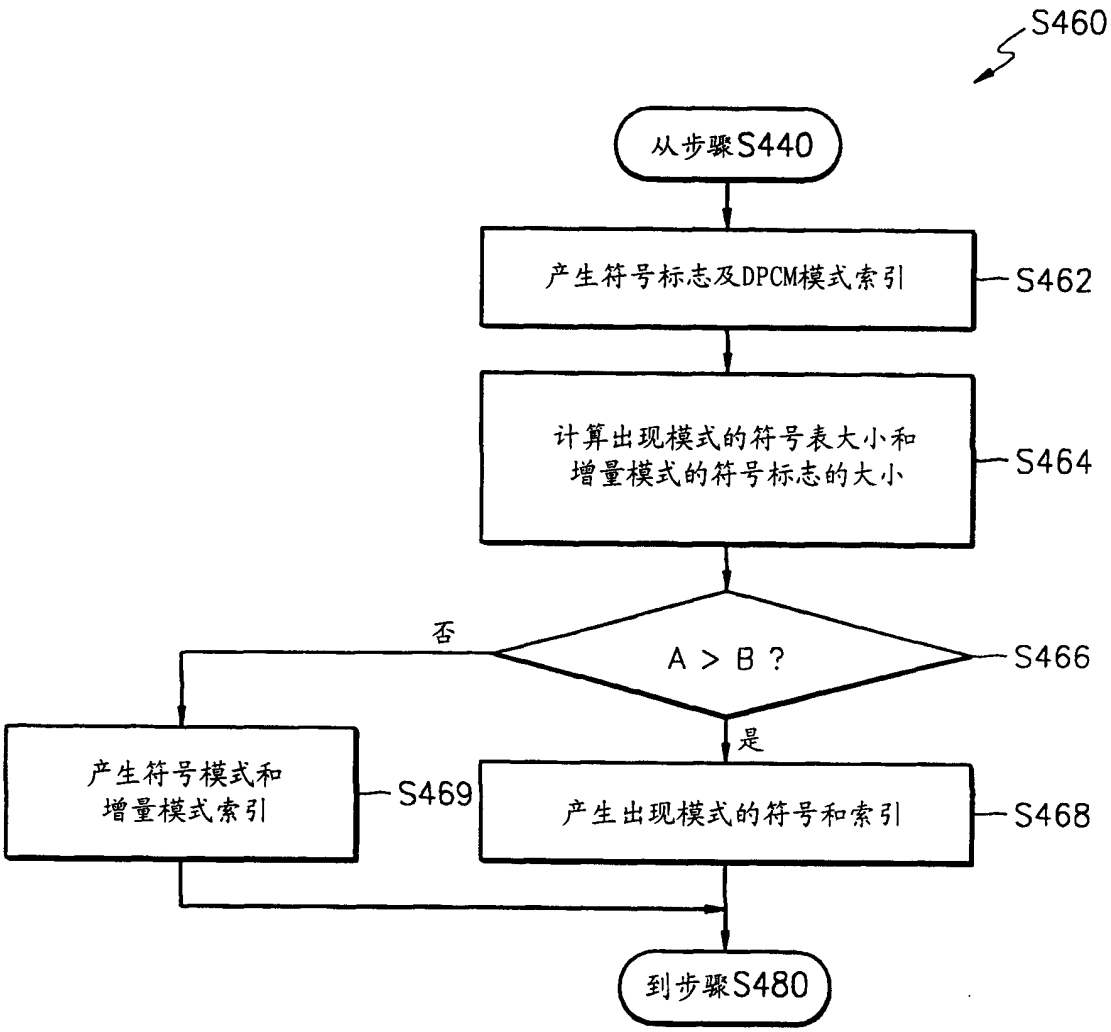

S460
从步骤S440
产生符号标志及DPCM模式索引
S462
计算出现模式的符号表大小和
增量模式的符号标志的大小
S464
否
A > B ?
S466
是
产生符号模式和
增量模式索引
S469
产生出现模式的符号和索引
S468
到步骤S480

图 4C

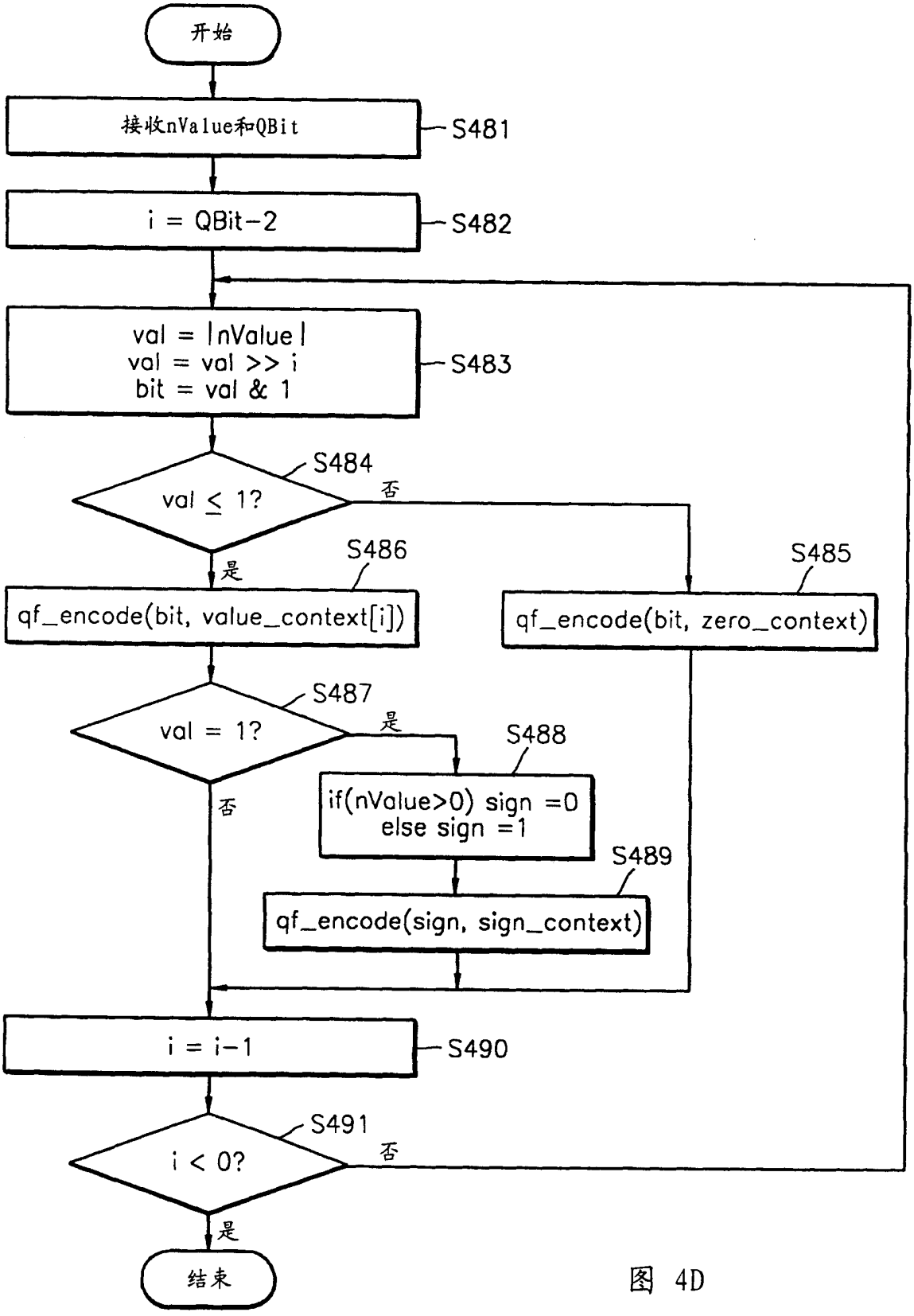

开始
接收nValue和QBit
S481
i = QBit-2
S482
val = |nValue|
val = val >> i
bit = val & 1
S483
S484
val ≤ 1?
否
是
S486
qf_encode(bit, value_context[i])
S485
qf_encode(bit, zero_context)
S487
val = 1?
是
否
S488
if(nValue>0) sign =0
else sign =1
S489
qf_encode(sign, sign_context)
i = i-1
S490
S491
i < 0?
否
是
结束

图 4D

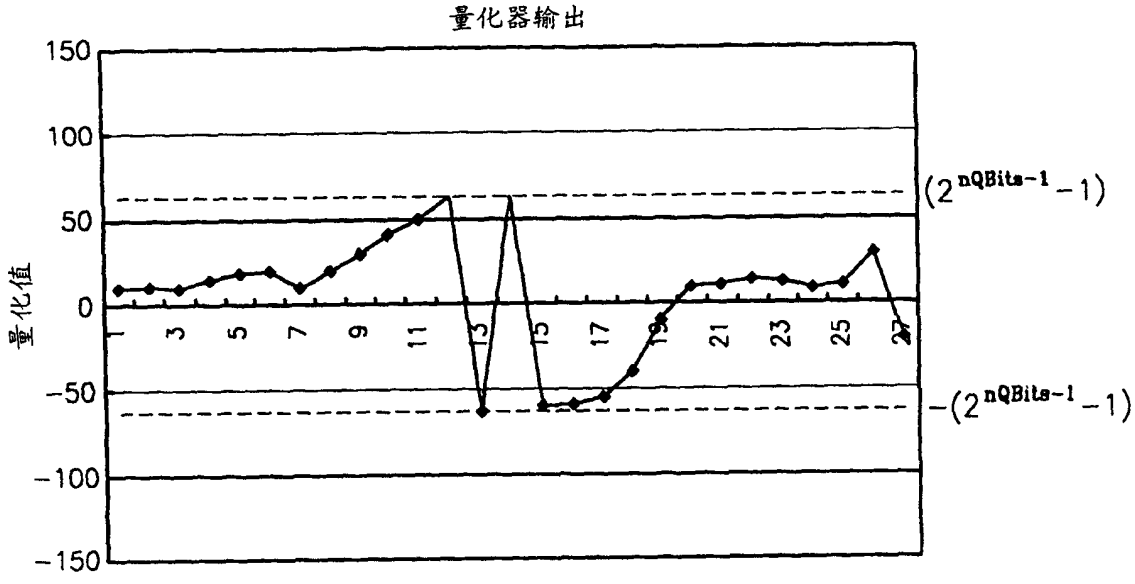

量化器输出
150
100
50
0
-50
-100
-150
量化值
1
3
5
7
9
11
13
15
17
19
21
23
25
27
$(2^{nQBits-1}-1)$
$-(2^{nQBits-1}-1)$

图 5A

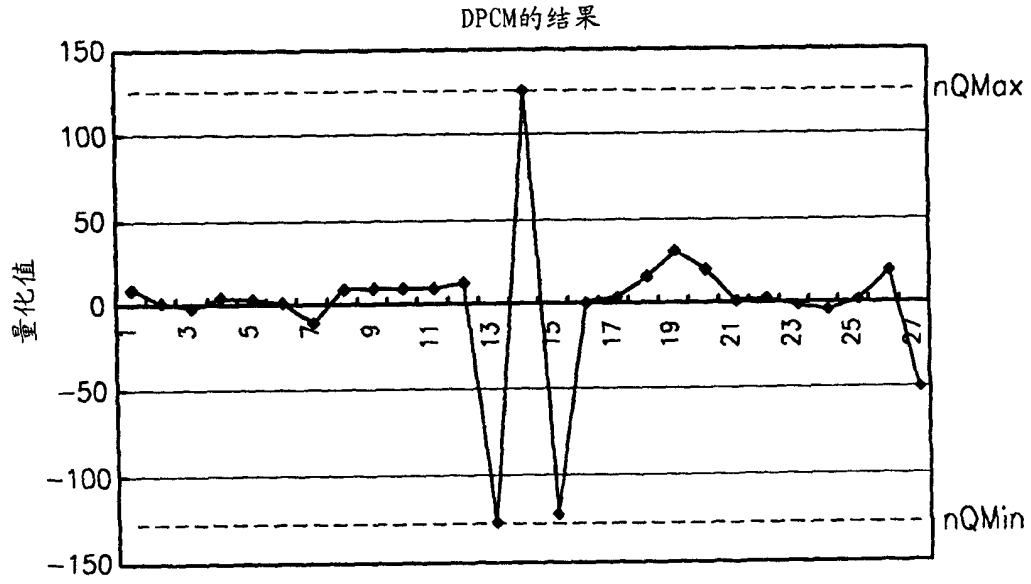

DPCM的结果
150
100
50
0
-50
-100
-150
量化值
1
3
5
7
9
11
13
15
17
19
21
23
25
27
nQMax
nQMin

图 5B

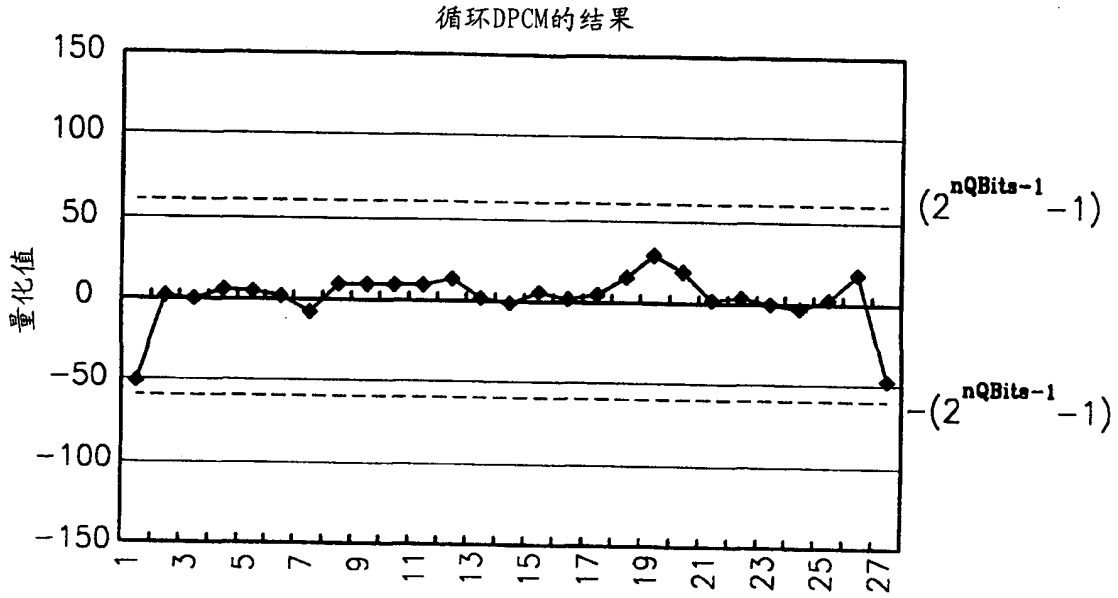

循环DPCM的结果
150
100
50
0
-50
-100
-150
量化值
1
3
5
7
9
11
13
15
17
19
21
23
25
27
$(2^{nQBits-1}-1)$
$-(2^{nQBits-1}-1)$

图 5C

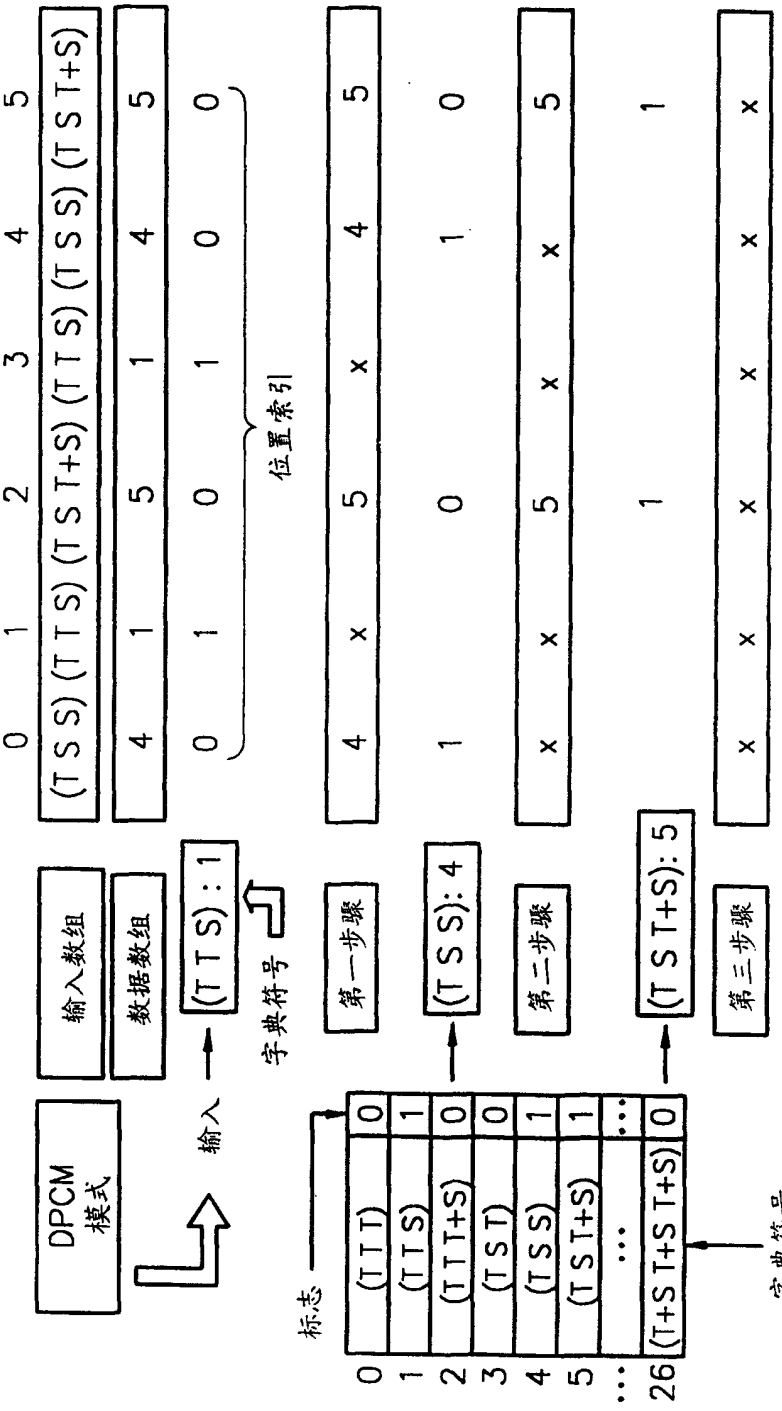

DPCM
模式
输入
输入数组
数据数组
(T S S) (T T S) (T S T+S) (T T S) (T S S) (T S T+S)
4 1 5 1 4 5
0 1 0 1 0 0
位置索引
(T T S): 1
字典符号
标志
第一步骤
第二步骤
第三步骤
(T S S): 4
(T S T+S): 5
(T T T)
(T T S)
(T T T+S)
(T S T)
(T S S)
(T S T+S)
(T+S T+S T+S)
字典符号

图 6A

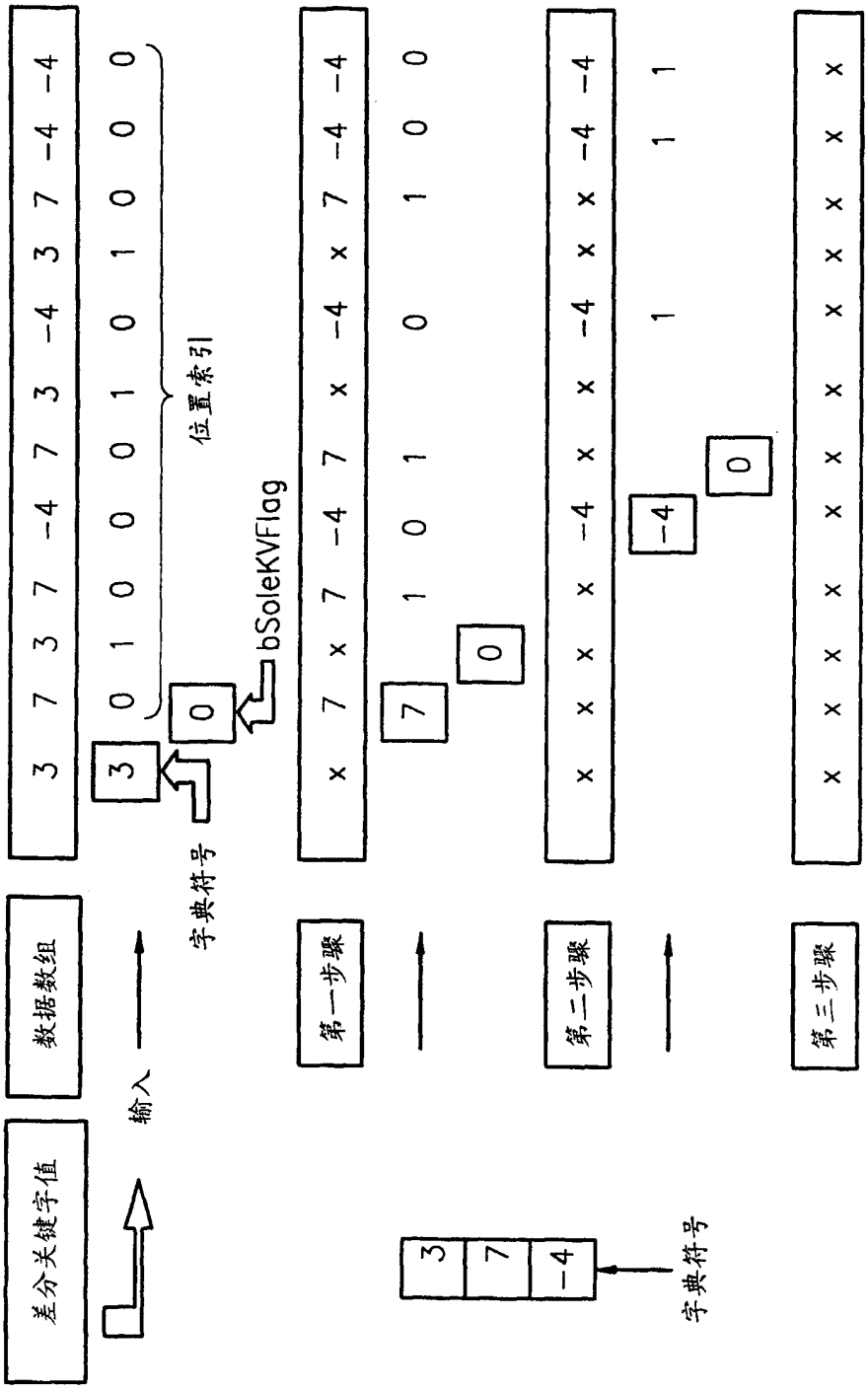

差分关键字值
数据数组
输入
3 7 3 7 -4 7 3 -4 3 7 -4 -4
0 1 0 0 0 0 1 0 1 0 0 0
位置索引
3
0
字典符号
bSoleKVFlag
第一步骤
x 7 x 7 -4 7 x -4 x 7 -4 -4
7
1 0 1 0 1 0 0
0
第二步骤
x x x x -4 x x -4 x x -4 -4
-4
1 1 1
0
第三步骤
x x x x x x x x x x x
3
7
-4
字典符号

图 6B

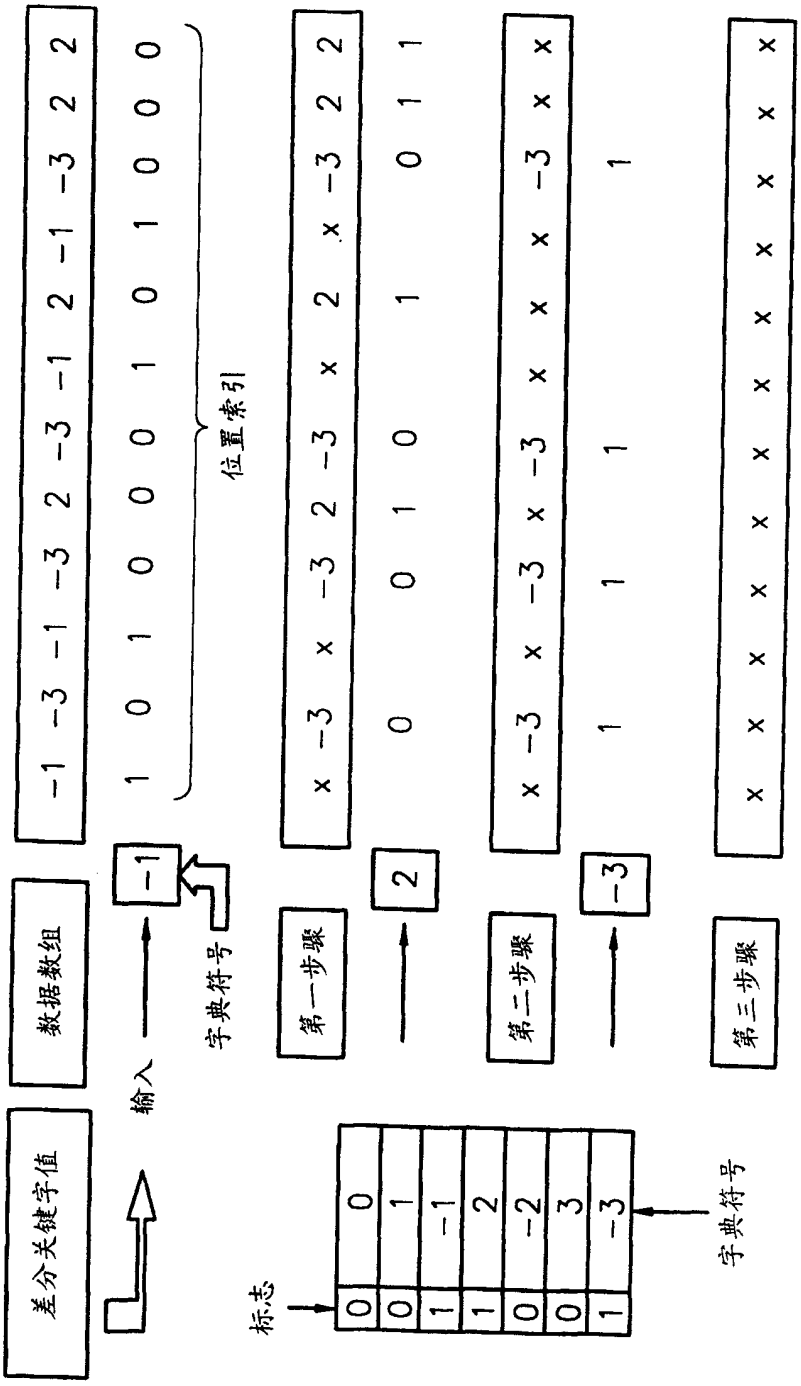

差分关键字值
数据数组
输入
-1 -3 -1 -3 2 -3 -1 2 -1 -3 2 2
1 0 1 0 0 0 0 1 0 1 0 0 0 0
位置索引
字典符号
-1
第一步骤
x -3 x -3 2 -3 x 2 x -3 2 2
0 0 1 0 1 0 1 1
2
第二步骤
x -3 x -3 x -3 x x x -3 x x
1 1 1 1
-3
第三步骤
x x x x x x x x x x x x
标志
字典符号
0 0
0 1
1 -1
1 2
0 -2
0 3
1 -3

图 6C

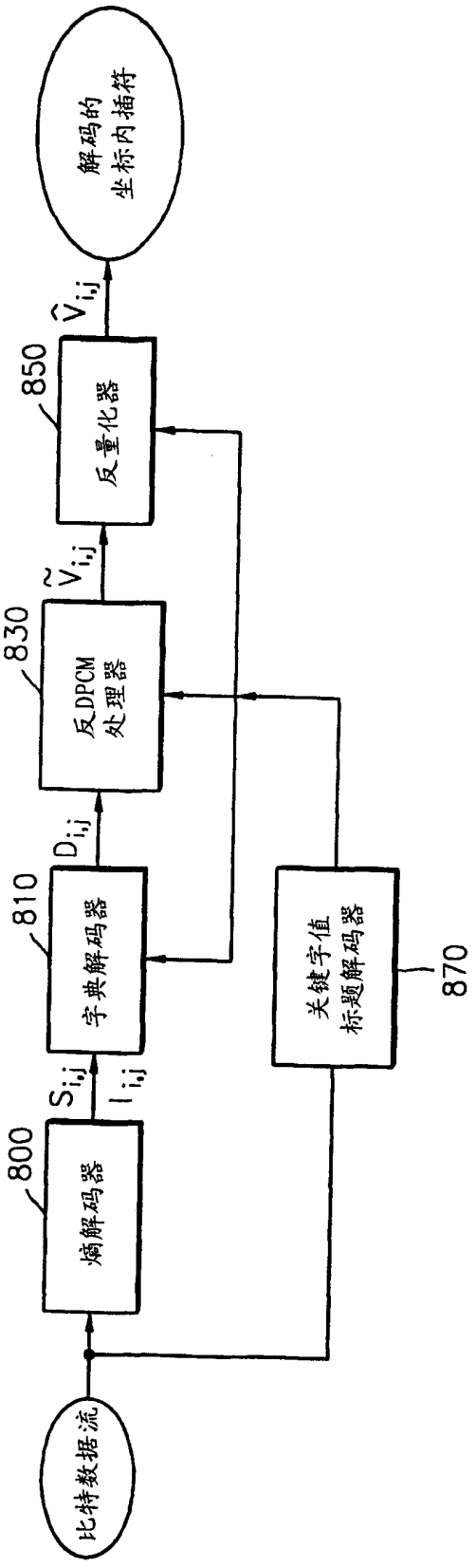

比特数据流
800
熵解码器
$S_{i,j}$
$I_{i,j}$
810
字典解码器
$D_{i,j}$
830
反DPCM处理器
$\tilde{V}_{i,j}$
850
反量化器
$\hat{V}_{i,j}$
解码的坐标内插符
870
关键字值标题解码器

图 7A

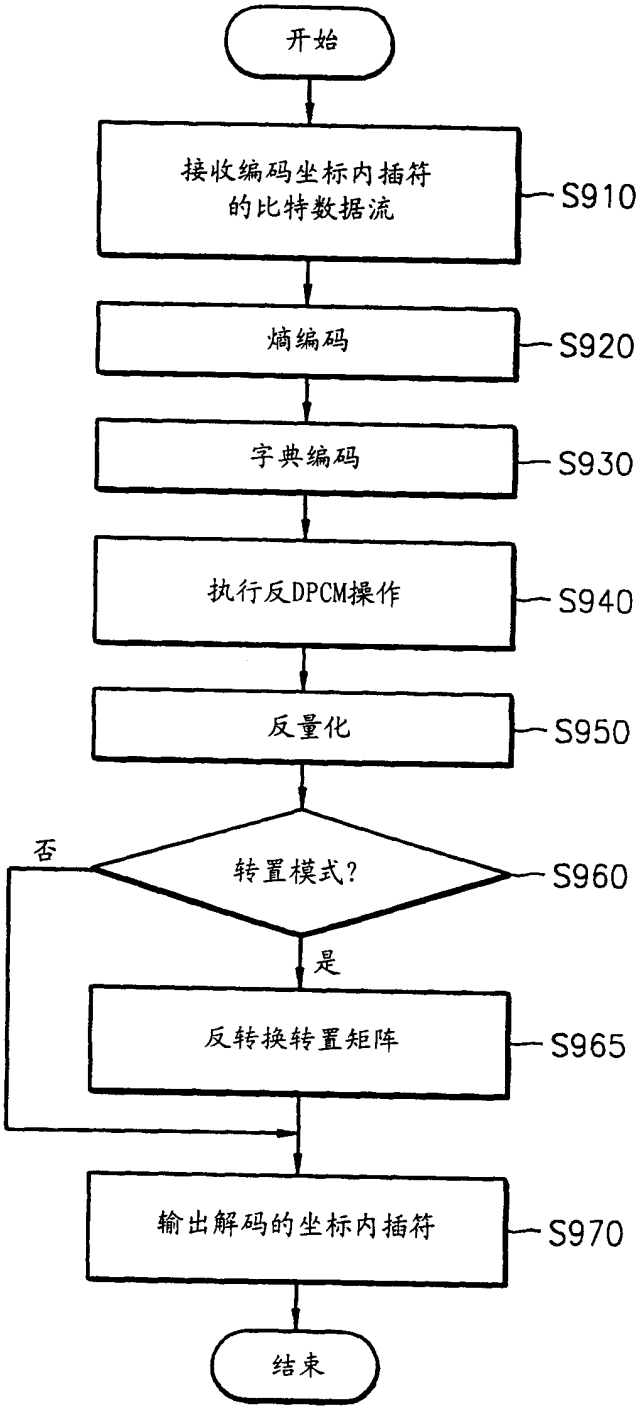

开始
接收编码坐标内插符的比特数据流
S910
熵编码
S920
字典编码
S930
执行反DPCM操作
S940
反量化
S950
转置模式?
S960
否
是
反转换转置矩阵
S965
输出解码的坐标内插符
S970
结束

图 7B

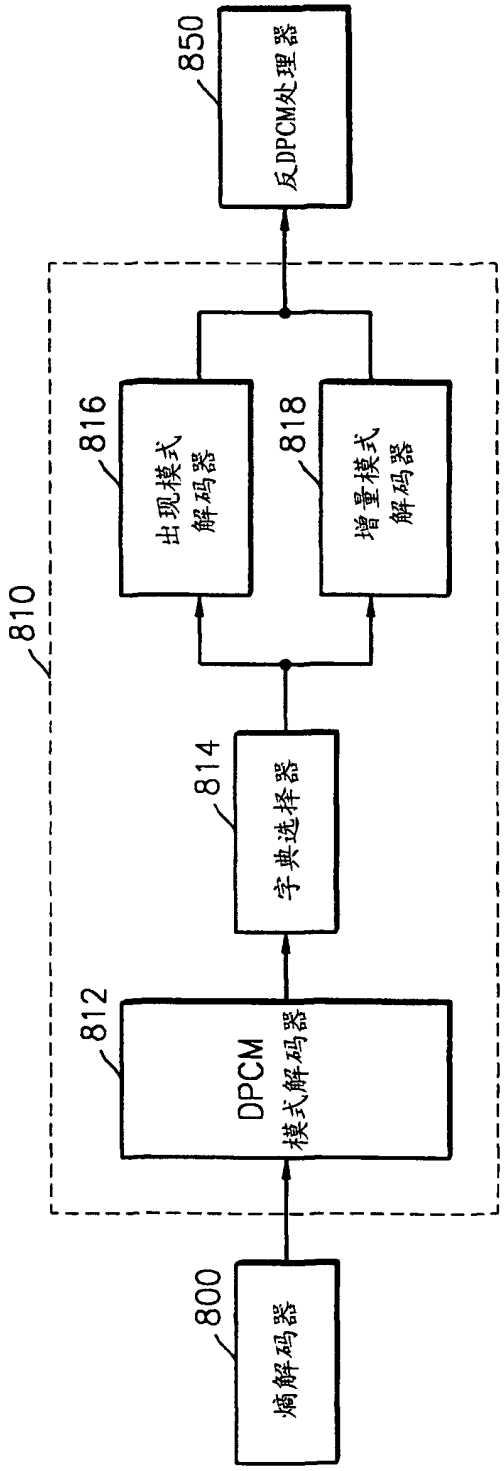

850
反DPCM处理器
816
出现模式
解码器
818
增量模式
解码器
810
814
字典选择器
812
DPCM
模式解码器
800
熵解码器

图 8A

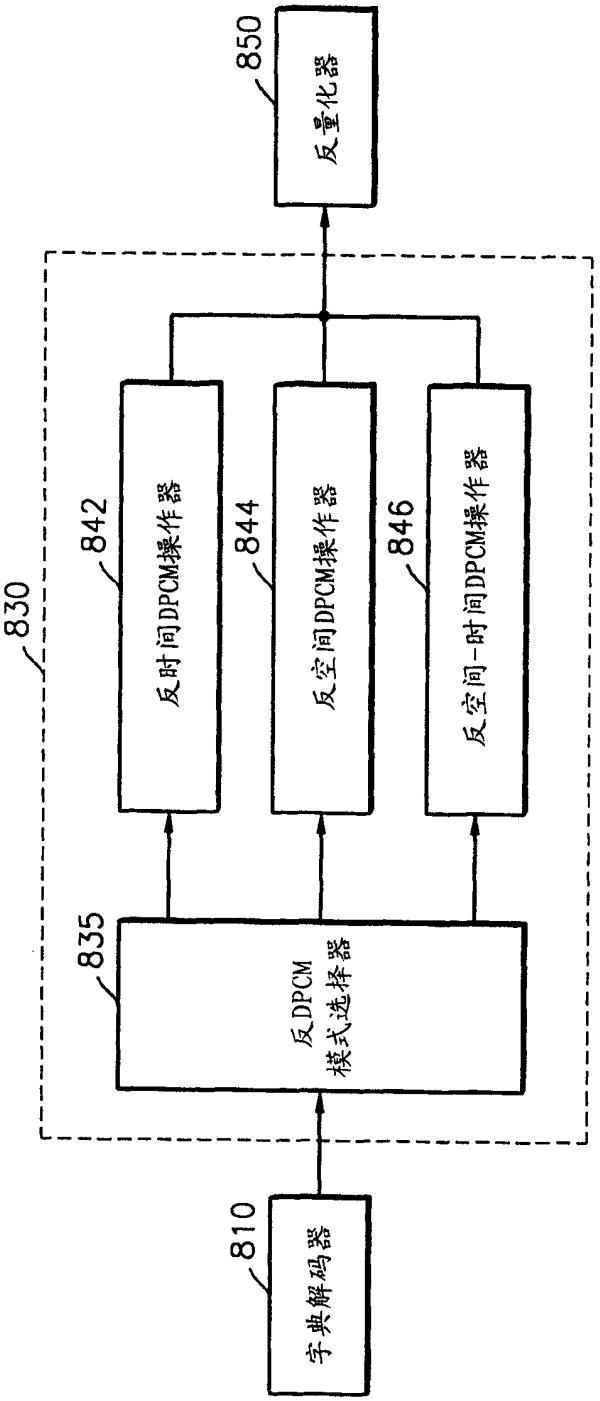

850
反量化器
830
842
反时间DPCM操作器
844
反空间DPCM操作器
846
反空间-时间DPCM操作器
835
反DPCM
模式选择器
810
字典解码器

图 8B

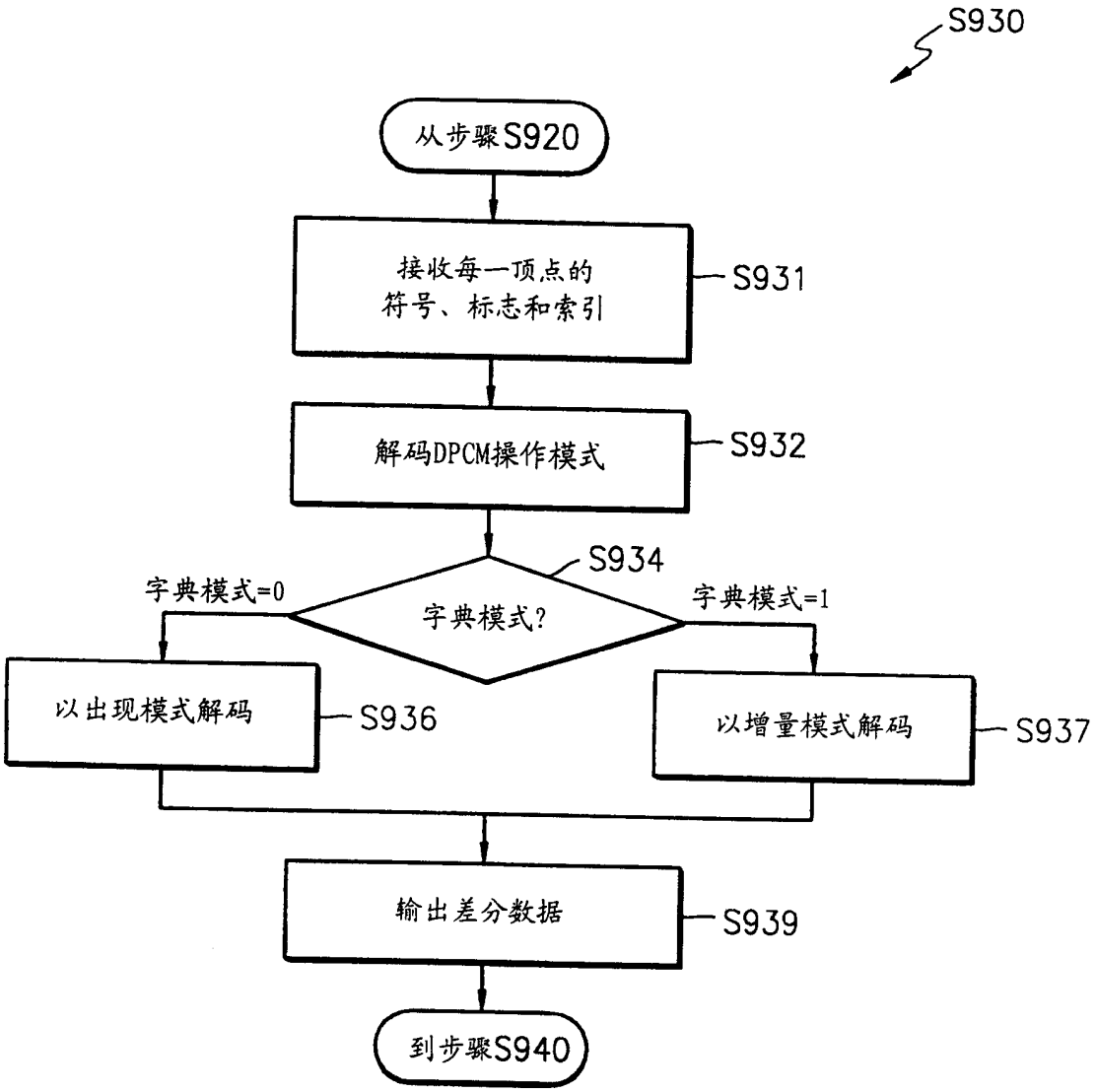

S930
从步骤S920
接收每一顶点的
符号、标志和索引
S931
解码DPCM操作模式
S932
S934
字典模式?
字典模式=0
字典模式=1
以出现模式解码
S936
以增量模式解码
S937
输出差分数据
S939
到步骤S940

图 9A

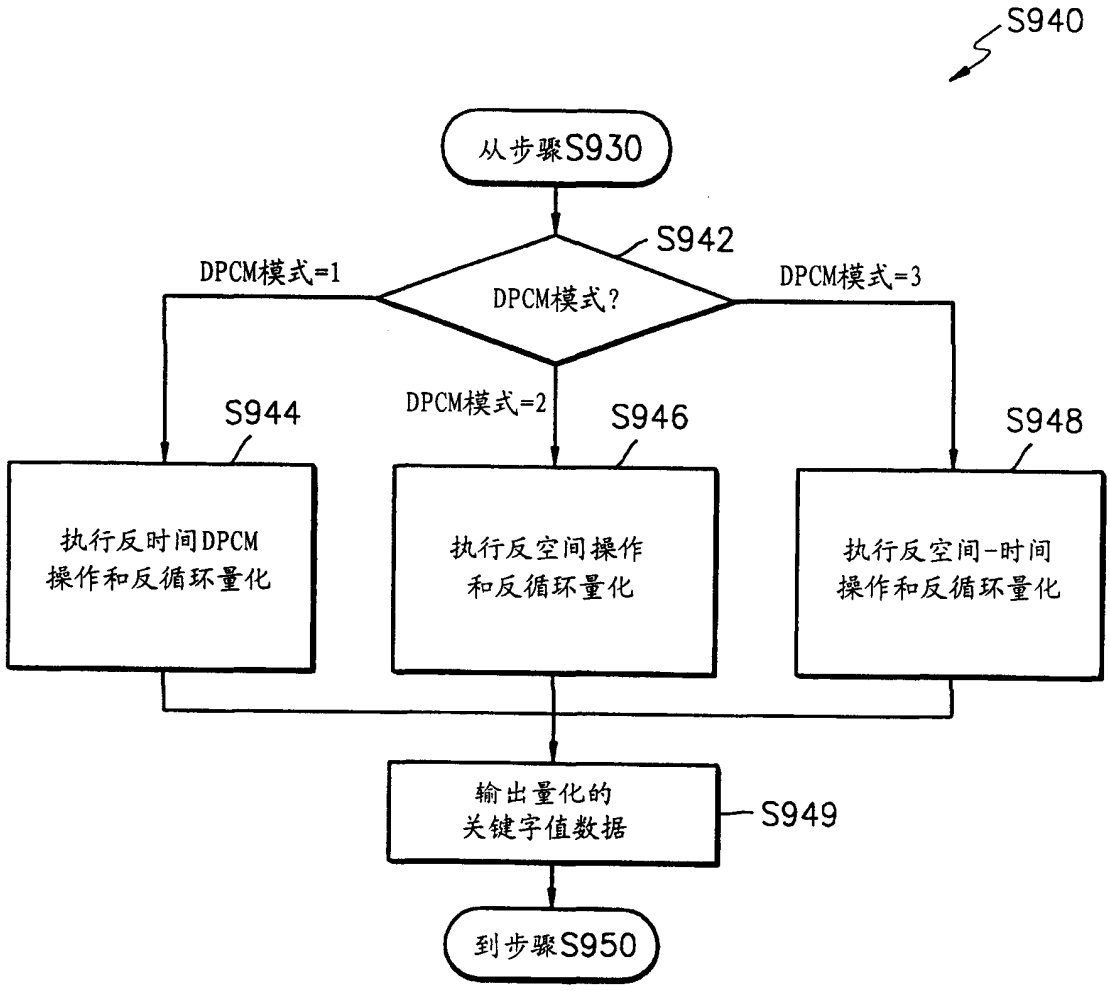

S940
从步骤S930
S942
DPCM模式?
DPCM模式=1
DPCM模式=2
DPCM模式=3
S944
S946
S948
执行反时间DPCM
操作和反循环量化
执行反空间操作
和反循环量化
执行反空间-时间
操作和反循环量化
输出量化的
关键字值数据
S949
到步骤S950

图 9B

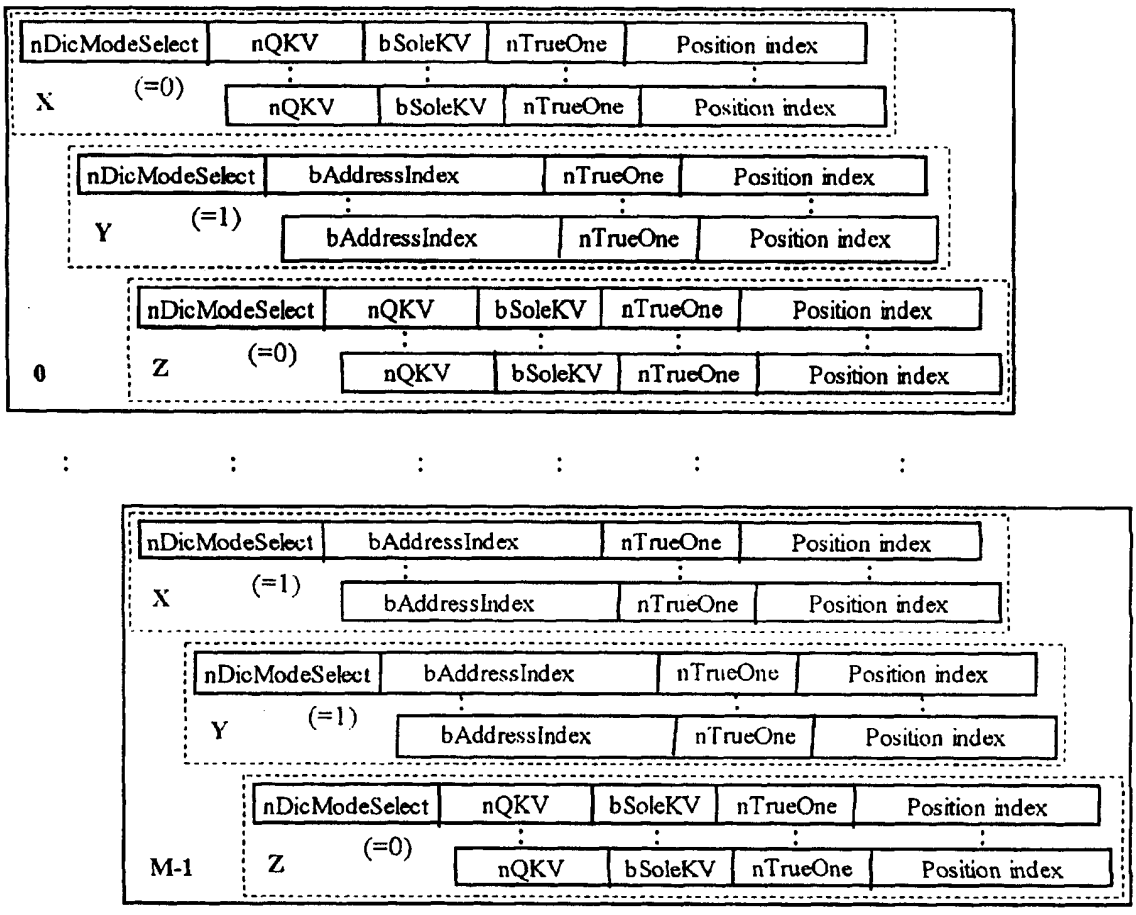

nDicModeSelect
nQKV
bSoleKV
nTrueOne
Position index
X
(=0)
nQKV
bSoleKV
nTrueOne
Position index
nDicModeSelect
bAddressIndex
nTrueOne
Position index
Y
(=1)
bAddressIndex
nTrueOne
Position index
nDicModeSelect
nQKV
bSoleKV
nTrueOne
Position index
0
Z
(=0)
nQKV
bSoleKV
nTrueOne
Position index
nDicModeSelect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

图 10

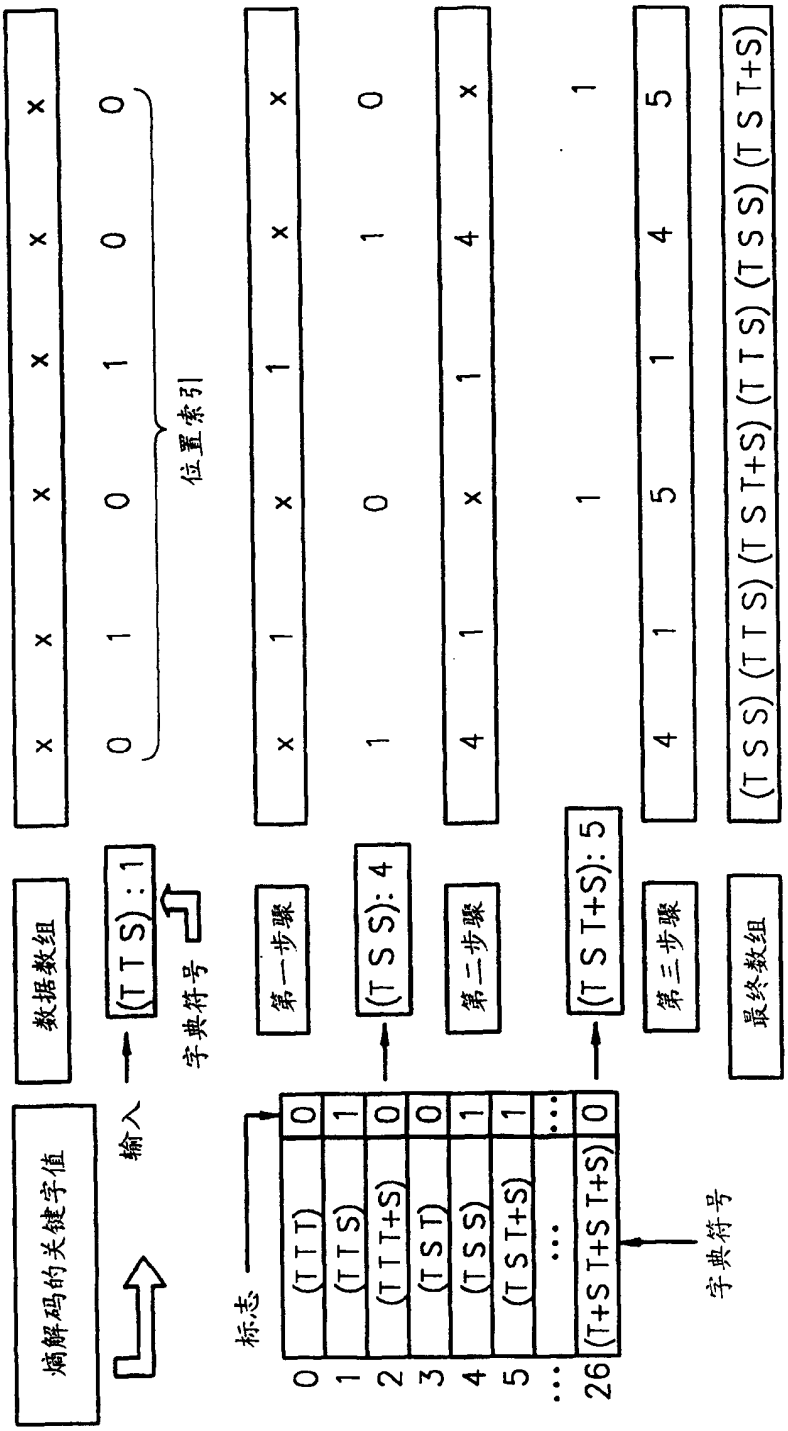

熵解码的关键字值
输入
(T T S): 1
字典符号
位置索引
数据数组
0 1 2 3 4 5
x x x x x x
0 1 0 1 0 0
第一步骤
x 1 x 1 x x
1 0 1 0
(T S S): 4
第二步骤
4 1 x 1 4 x
(T S T+S): 5
第三步骤
4 1 5 1 4 5
最终数组
(T S S) (T T S) (T S T+S) (T T S) (T S S) (T S T+S)
标志
字典符号
0 (T T T) 0
1 (T T S) 1
2 (T T T+S) 0
3 (T S T) 0
4 (T S S) 1
5 (T S T+S) 1
... ...
26 (T+S T+S T+S) 0

图 11A

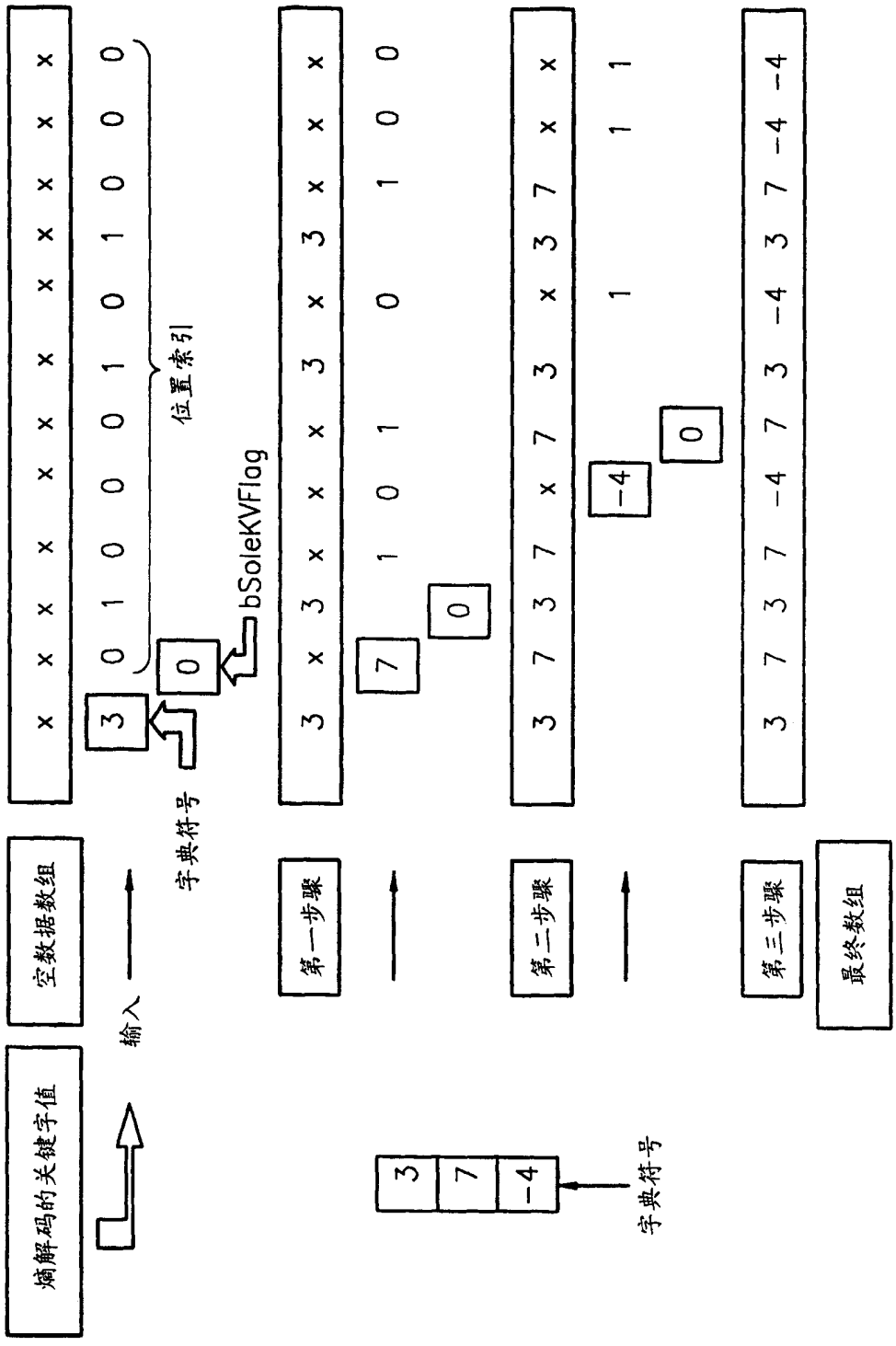

熵解码的关键字值
空数据数组
输入
x x x x x x x x x x x x x x x x
3 0 1 0 0 0 0 1 0 1 0 0 0 0
位置索引
字典符号
0
bSoleKVFlag
第一步骤
3 x 3 x x x x 3 x 3 x x x
7
1 0 1 0 1 0 0
0
第二步骤
3 7 3 7 x 7 3 x 3 7 x x
-4
1 1 1
0
第三步骤
3 7 3 7 -4 7 3 -4 3 7 -4 -4
最终数组
3
7
-4
字典符号

图 11B

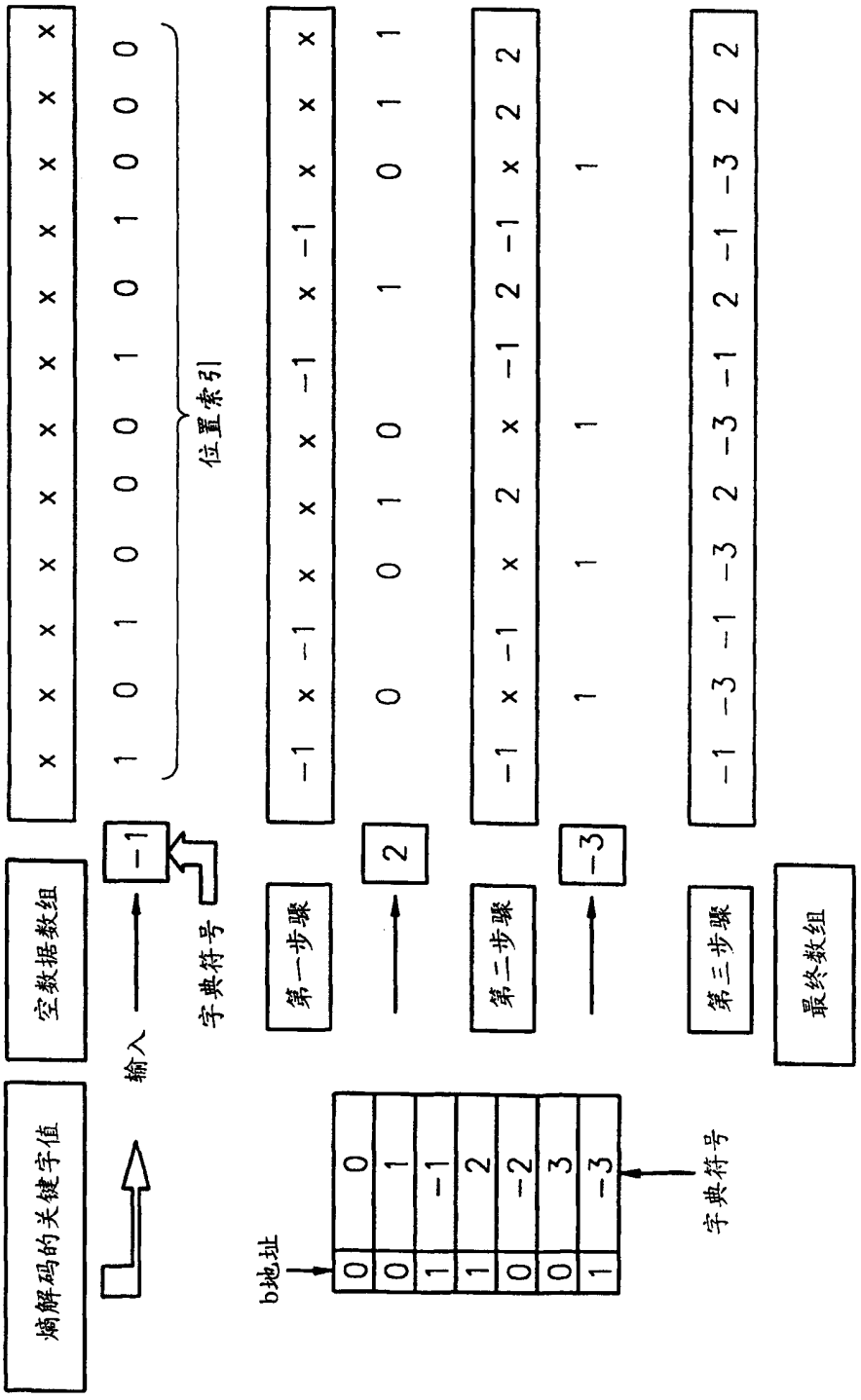

熵解码的关键字值
空数据数组
输入
-1
字典符号
位置索引
x x x x x x x x x x x x x x x x
1 0 1 0 0 0 0 1 0 1 0 0 0 0
第一步骤
-1 x -1 x x x -1 x -1 x x x
0 0 1 0 1 0 1 1
第二步骤
-1 x -1 x 2 x -1 2 -1 x 2 2
1 1 1 1
第三步骤
-1 -3 -1 -3 2 -3 -1 2 -1 -3 2 2
最终数组
b地址
0 0
0 1
1 -1
1 2
0 -2
0 3
1 -3
字典符号

图 11C

```
class CompressedCoordinateInterpolator {
        CoordIKeyValueHeader coordIKVHeader;
        qf_start();
        aligned(8)        CoordIKeyValue coordIKeyValue(coordIKVHeader);
}
```

图 12

```
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;
        unsigned int(5) nNumKeyCodingBit;
        unsigned int(nNumKeyCodingBit) nNumberOfKey;
}
```

图 13

```
class CoordIKeyValue (CoordIKeyValueHeader coordIKVHeader) {
        int j, c;
    if(coordIKVHeader.bTranspose == 1) {
        int temp = coordIKVHeader.nNumberOfKey;
        coordIKVHeader.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++) { }
```

图 14A

```
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);
                                                            }
                                        }
                                        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], coordIKVHeader.nNumberOfKey, c);
                    }
          }
}
```

图 14B

```
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 15

```
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 16

```
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 17

```
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 18

```
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 19

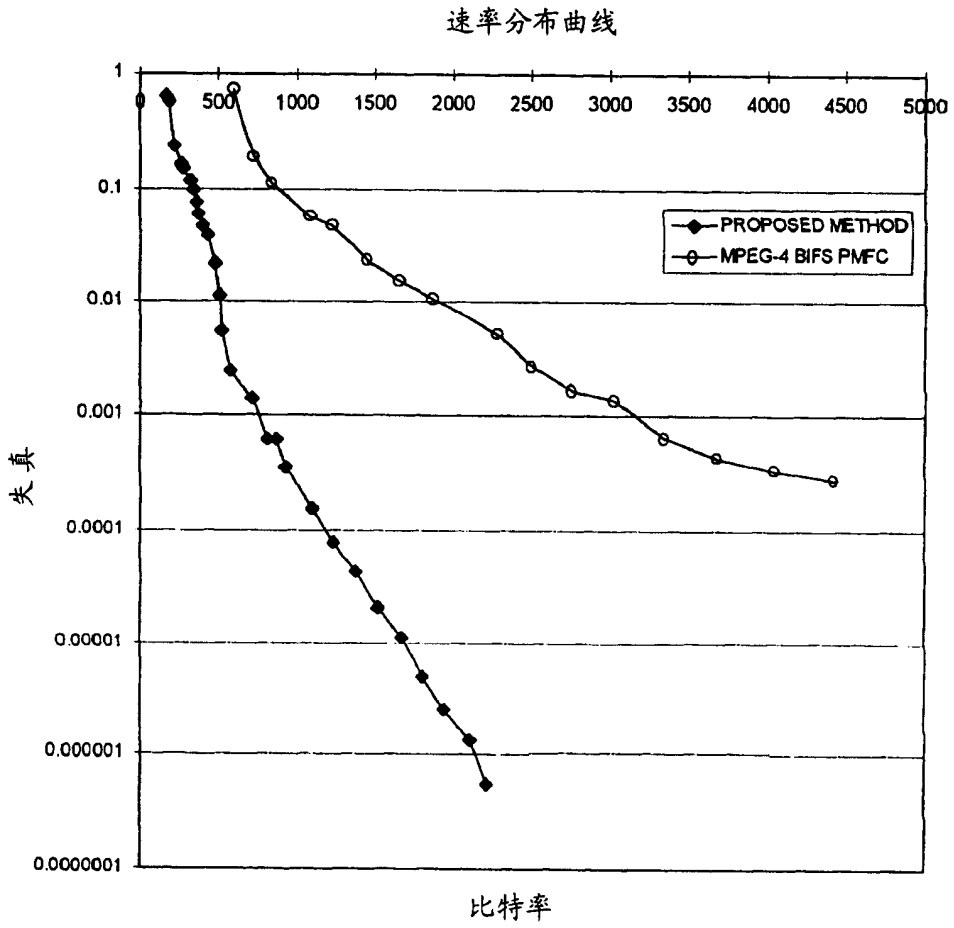

速率分布曲线
1
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
失真
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
比特率
PROPOSED METHOD
MPEG-4 BIFS PMFC

图 20A

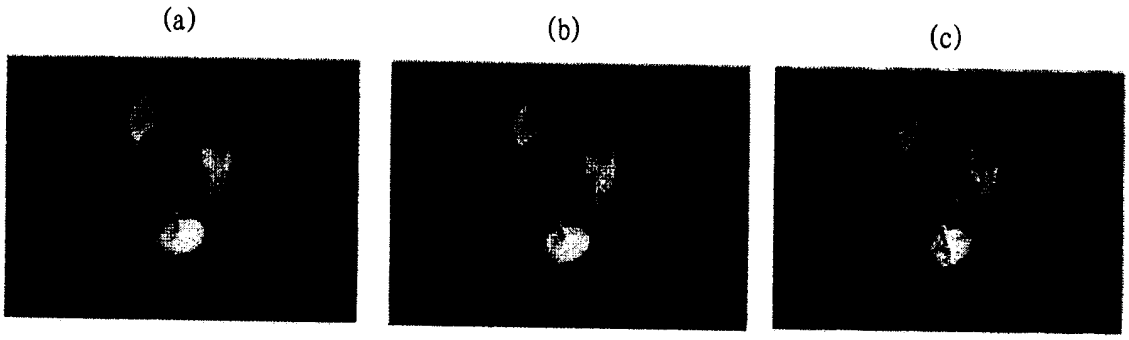


图 20B

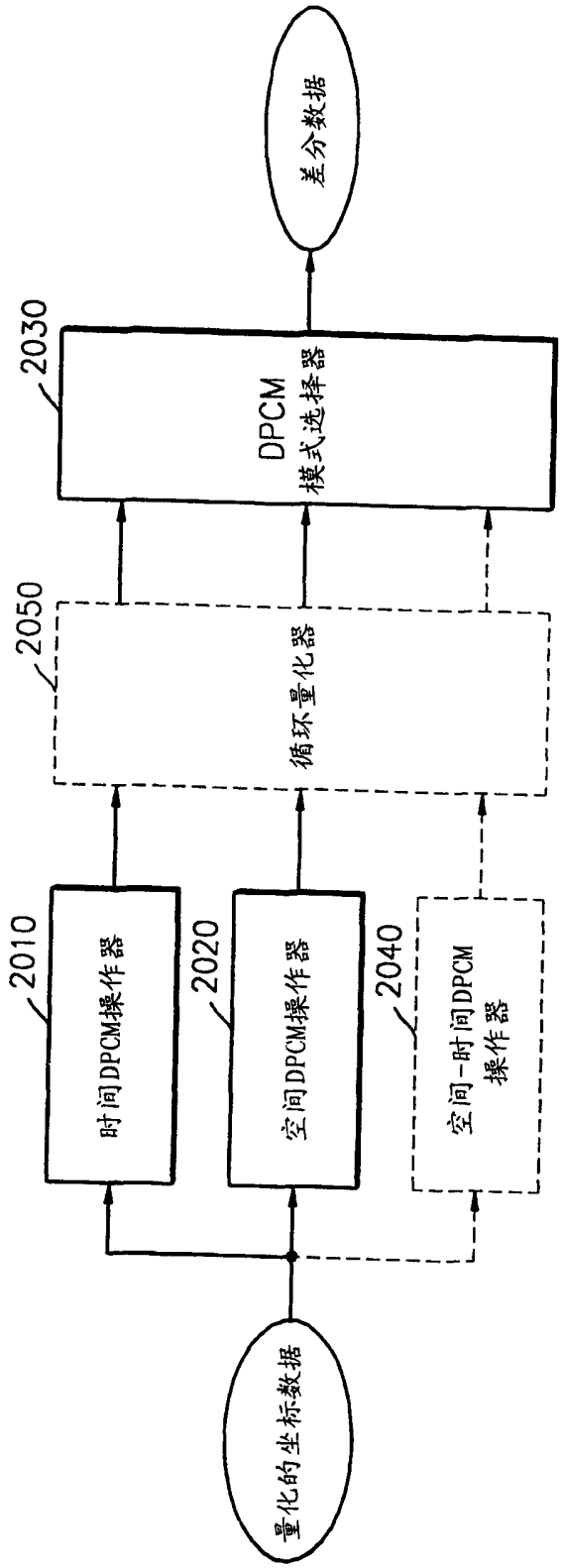

量化的坐标数据
2010
时间DPCM操作器
2020
空间DPCM操作器
2040
空间-时间DPCM
操作器
2050
循环量化器
2030
DPCM
模式选择器
差分数据

图 21A

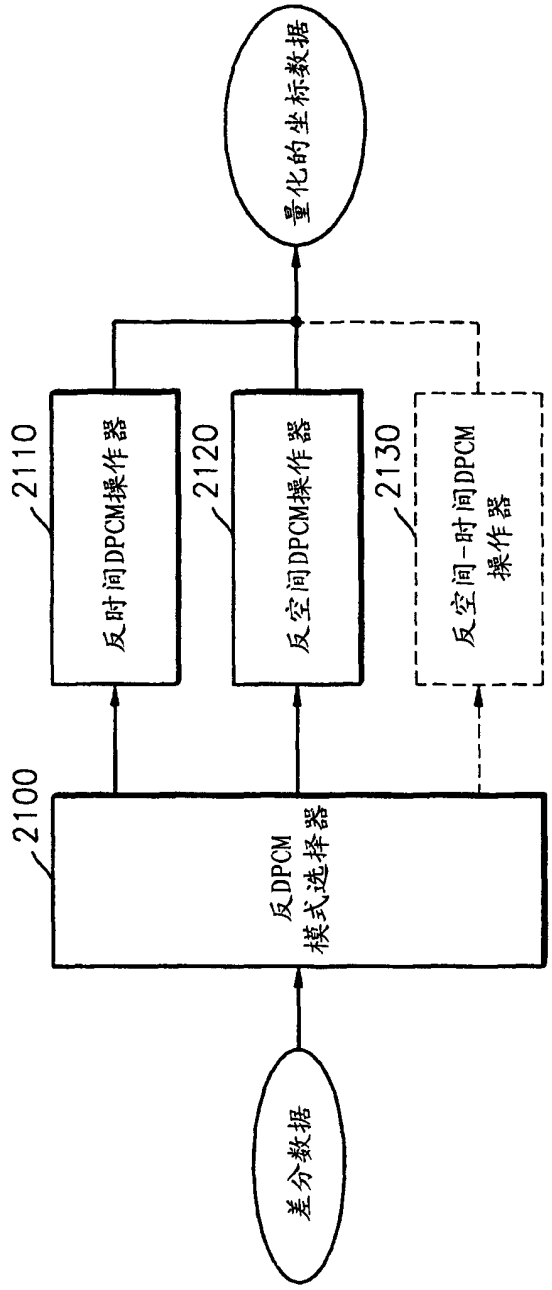

量化的坐标数据
2110
反时间DPCM操作器
2120
反空间DPCM操作器
2130
反空间-时间DPCM
操作器
2100
反DPCM
模式选择器
差分数据

图 21B