



[12] 发 明 专 利 说 明 书

[21] ZL 专利号 95116376.0

[43] 授权公告日 2003 年 1 月 15 日

[11] 授权公告号 CN 1099196C

[22] 申请日 1995.8.23 [21] 申请号 95116376.0

[30] 优先权

[32] 1994.8.23 [33] GB [31] 9417138.6

[71] 专利权人 DVA 公司

地址 美国加利福尼亚州

[72] 发明人 阿德里安·P·怀斯

审查员 马桂丽

[74] 专利代理机构 北京市柳沈律师事务所

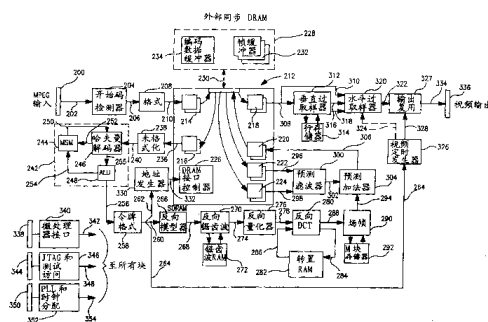
代理人 马 莹

权利要求书 3 页 说明书 16 页 附图 5 页

[54] 发明名称 数据速率变换方法和设备

[57] 摘要

一种在视频解码中将较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的方法，包括下列步骤：确定在一帧周期中场重复的基本整数，该基本整数被确定为该较快速率与较慢速率的比的整数部分；通过将所述帧重复所述基本整数的场数或 1 加上所述基本整数的场数，来将所述帧流的帧重复到所述场流中；所述帧的重复场数是通过执行一次布雷申汉画线算法来选择，其中线的斜率是所述比的分数部分，并且当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画线算法的执行增大 Y 轴时，将所述帧重复 1 加上所述基本整数的场数。



1、一种将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的方法，包括下列步骤：

- 5 确定在一帧周期中场重复的基本整数，该基本整数被确定为该较快速率与较慢速率的比的整数部分；

 通过将所述帧重复所述基本整数的场数或 1 加上所述基本整数的场数，来将所述帧流的帧重复到所述场流中；

- 所述帧的重复场数是通过执行一次布雷申汉画线算法来选择，其中线的斜率是所述比的分数部分，并且当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画线算法的执行增大 Y 轴时，将所述帧重复 1 加上所述基本整数的场数。
- 10

2、根据权利要求 1 的方法，其中所述重复的基本整数为 2。

- 3、根据权利要求 1 或 2 的方法，其中所述斜率的分子和分母的大小由公因子来减小。
- 15

4、根据权利要求 3 的方法，其中所述较慢速率为 23.98, 24, 25 和 29.97Hz 之一。

5、根据权利要求 4 的方法，其中所述较快速率为 50, 59.94, 和 60Hz 之一。

- 20 6、根据权利要求 5 的方法，其中所述帧流是压缩流。

7、根据权利要求 6 的方法，其中所述帧流是根据 JPEG, MPEG 和 H.261 压缩的压缩流。

8、一种将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的方法，包括下列步骤：

- 25 确定在一帧周期中场重复的基本整数，该基本整数被确定为 1 加上该较快速率与较慢速率的比的整数部分；

 通过将所述帧重复所述基本整数的场数或所述基本整数减去 1 的场数，来将所述帧流的帧重复到所述场流中；

- 所述帧的重复场数是通过执行一次布雷申汉画线算法来选择，其中线的斜率是所述基本整数和和所述比之间的差，并且当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画
- 30

线算法的执行增大 Y 轴时，将所述帧重复所述基本整数减去 1 的场数。

9、根据权利要求 8 的方法，其中所述重复的基本整数为 3。

10、根据权利要求 8 或 9 的方法，其中所述斜率的分子和分母的大小由公因子来减小。

5 11、根据权利要求 10 的方法，其中所述较慢速率为 23.98, 24, 25 和 29.97Hz 之一。

12、根据权利要求 11 的方法，其中所述较快速率为 50, 59.94, 和 60Hz 之一。

13、根据权利要求 12 的方法，其中所述帧流是压缩流。

10 14、根据权利要求 13 的方法，其中所述帧流是根据 JPEG, MPEG 和 H. 261 压缩的压缩流。

15、一种用于将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的设备，该设备包括：

15 确定在一帧周期中场重复的基本整数的装置，该基本整数被确定为该较快速率与较慢速率的比的整数部分；

布雷申汉画线算法计算装置，用于执行一次布雷申汉画线算法，其中线的斜率是所述比的分数部分；

20 根据所述布雷申汉画线算法的执行，将所述帧流的帧重复到所述场流中的装置，所重复的场数为将所述帧重复所述基本整数的场数或 1 加上所述基本整数的场数，当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画线算法的执行增大 Y 轴时，将所述帧重复 1 加上所述基本整数的场数。

16、一种用于将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的设备，该设备包括：

25 确定在一帧周期中场重复的基本整数的装置，该基本整数被确定为 1 加上该较快速率与较慢速率的比的整数部分；

布雷申汉画线算法计算装置，用于执行一次布雷申汉画线算法，其中线的斜率是所述基本整数和和所述比之间的差；

30 根据所述布雷申汉画线算法的执行，将所述帧流的帧重复到所述场流中的装置，所重复的场数为将所述帧重复所述基本整数的场数或所述基本整数减去 1 的场数，当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重

复所述基本整数的场数；当所述布雷申汉画线算法的执行增大Y轴时，将所述帧重复所述基本整数减去1的场数。

数据速率变换方法和设备

5

技术领域

本发明涉及数据速率变换方法和设备，而且特别地、但不是唯一地，涉及视频数据/帧场速率变速。

背景技术

10

有多种压缩视频信息的方法。特别是，有可执行压缩的三个标准：JPEG, MPEG 和 H. 261。这些标准在 US 5212742 等文件中进行了讨论。

15

视频信息通常被格式化为一系列的场。欲被变换为以视频格式显示的原始信息不能直接与该信息将被显示的场速率兼容。例如，电影胶片以 24 帧/秒速率 (24 Hz) 拍摄，而例如 NTSC 电视制式场速率为大约 60 Hz。增加电影图象的帧速率以与电视系统电影速率相一致的技术称为下拉 (pulldown) 变换。

20

继续以上述例子叙述如何以 NTSC 标准格式显示电影。可使用“2/3 下拉”变换，其中每个电影的帧以视频重复率重复二个或三个连续场周期。重复次数交变，以致第一帧在二个连续场周期中显示两次，第二帧在三个连续场周期中显示三次，以此类推。因此，一秒中在以 24 Hz 为速率的十二个电影帧每帧已被产生两次（即，对于 24 场周期），而另外十二个电影帧已被产生三次（即对于 36 场周期）。总共 (24 + 36) 等于在 60 Hz 场速率下的一秒内的 60 场。

25

下拉命令可在远程产生并且发信号给与显示设备相关的视频解码器，或者在视频解码器本地产生。在发信号的下拉中，编码器例如使用在 MPEG-2 视频语法中的“重复第一场”标记执行下拉计算和特别是那些帧被重复的信号。解码器只是遵照所接收的远端产生的命令。

30

在本地下拉中，编码器编码该电影信息并将它发送到接收设备。在传输信号中没有信息告诉接收设备的解码器如何执行合适的下拉变换（例如上面所提的“2/3 下拉”）。因此，解码器必须计算如何执行从发送来的电影速率到显示的场速率的合适变换。

如果只要求从 24 Hz 帧速率下拉变换为 60 Hz 场速率，单个 2/3 下拉变换是相对容易实现的。但是，实际上还要求其它的下拉方案。例如，对于 PAL 电视格式，24 Hz 电影帧速率可能需要被变换为 50Hz 场速率。

另外 NTSC 电视制片中额外的复杂性是：实际的场速率不是 60Hz，而是
5 60000/1001 Hz。因此，正规的交替 2/3 下拉产生实际上太高的场速率。

发明内容

为解决上述问题，根据本发明，提供一种将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的方法，包括下列步骤：确定在一帧
10 周期中场重复的基本整数，该基本整数被确定为该较快速率与较慢速率的比的整数部分；通过将所述帧重复所述基本整数的场数或 1 加上所述基本整数的场数，来将所述帧流的帧重复到所述场流中；所述帧的重复场数是通过执行一次布雷申汉画线算法来选择，其中线的斜率是所述比的分数部分，并且当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重复所述基本整
15 数的场数；当所述布雷申汉画线算法的执行增大 Y 轴时，将所述帧重复 1 加上所述基本整数的场数。

本发明还提供一种将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的方法，包括下列步骤：确定在一帧周期中场重复的基本整数，该基本整数被确定为 1 加上该较快速率与较慢速率的比的整数部分；
20 通过将所述帧重复所述基本整数的场数或所述基本整数减去 1 的场数，来将所述帧流的帧重复到所述场流中；所述帧的重复场数是通过执行一次布雷申汉画线算法来选择，其中线的斜率是所述基本整数和和所述比之间的差，并且当所述布雷申汉画线算法的执行不增大 Y 轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画线算法的执行增大 Y 轴时，将所述帧重复所述
25 基本整数减去 1 的场数。

使用术语“帧”和“场”是为了方便。二者是指作为一组发送、接收和/或构造的数据的任何帧、场、数据组或其它分离的量的一个集合。本发明允许所选的重复率通过包含或提取重复帧进行改变，因此允许冗余数据实现较快场数据率。最好是，所选的基本整数重复率小于较快的场速率。在这种情况下，本方法将以重复速率另外加上重复帧。重复率可小于场速率的一半。
30

本方法不必选择较低的基本整的重复率。在替代方案中，它可同样好地

选择较快的速率，然后本方法可安排需要在需要时删除重复的帧。

本发明还提供一种用于将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的设备，该设备包括：确定在一帧周期中场重复的基本整数的装置，该基本整数被确定为该较快速率与较慢速率的比的整数部分；

- 5 布雷申汉画线算法计算装置，用于执行一次布雷申汉画线算法，其中线的斜率是所述比的分数部分；根据所述布雷申汉画线算法的执行，将所述帧流的帧重复到所述场流中的装置，所重复的场数为将所述帧重复所述基本整数的场数或1加上所述基本整数的场数，当所述布雷申汉画线算法的执行不增大Y轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画线算法的执行增大Y轴时，将所述帧重复1加上所述基本整数的场数。

- 10 本发明还提供一种用于将以较慢速率接收的数据帧的流变换为以较快速率产生的数据场的流的设备，该设备包括：确定在一帧周期中场重复的基本整数的装置，该基本整数被确定为1加上该较快速率与较慢速率的比的整数部分；布雷申汉画线算法计算装置，用于执行一次布雷申汉画线算法，其中线的斜率是所述基本整数和和所述比之间的差；根据所述布雷申汉画线算法的执行，将所述帧流的帧重复到所述场流中的装置，所重复的场数为将所述帧重复所述基本整数的场数或所述基本整数减去1的场数，当所述布雷申汉画线算法的执行不增大Y轴时，将所述帧重复所述基本整数的场数；当所述布雷申汉画线算法的执行增大Y轴时，将所述帧重复所述基本整数减去1的场数。本发明提供一个一般化的解决方案给下拉计算，允许在50 Hz场速率显示23.98 Hz、24 Hz及25 Hz帧速率的数据和以59.94 Hz场速率显示23.98 Hz、24 Hz、25 Hz及29.97 Hz速率的数据。

附图简述

- 25 本发明可以很多方式予以实现，现在对照附图叙述其方式之一。
- 图1表示Breshenham(布雷申汉)的画线算法；
- 图2表示通过视频解码器的数据流的方框图；和
- 图3至6是本发明各方面的流程图。

- 30 具体实施方式

Breshenham的画线算法是在显示设备上给出任意斜率线的方法，它们被

分为一系列的矩形象素(Pel)。由 Addison-Wesley 公司出版的 Foley 等人的书籍“交互式计算机绘图基础”第 433 与 436 页之间可找到 Bresenham 的算法的叙述。

在图 1 所示的画线情况(这些线具有 0 和 1 之间的斜率),通过对在 X 轴的每个坐标判定在 Y 轴的哪个象素最接近理想线,该算法来逼近该线。这个象素被照亮或上色以作为适于应用的部分。

当该算法在该图中从左到右例如从(n-1)移到(n)时,它判定是否为(n-1)选择相同 Y 坐标的象素,或者是否递增 Y 坐标。在图中, Y 坐标在(n)和(n+1)被递增,但在(n+1)不递增。

10 应用到本发明中,在当从视频解码器的输入帧速率得出较快场速率时判定是否显示当前帧 3 个场周期面而不是正常的 2 个场周期中,使用是否递增 Y 坐标的判定。

在从 24 Hz 帧速率变换到 60 Hz 场速率的简单情况下,希望的加快速率是 60/24。但是,重要的判定是确定在一帧周期中是否一帧显示三个场周期(而不是两个场周期)。如果没有三场周期帧,则 24 Hz 帧速率将得到 48 场。因此,两倍的重复场数的比率可从加快速率中减去:

$$\frac{60}{24} - \frac{48}{24} = \frac{12}{24} = \frac{1}{2} \dots\dots\dots (1)$$

20 画一条具有 1/2 斜率的线则可允许我们计算下拉图。很清楚,对于 1/2 斜率的线,对于 X 坐标的每一步 Y 坐标递增一次。这是期望的结果,因为我们知道我们显示交替的电影帧三个场次,以便执行 2/3 下拉。

25 我们于 1994 年 7 月 29 日提交的、专利申请号为 9415413.5、名称为“视频解压缩的方法和设备”叙述一种多标准视频解码器,它将被引用在这里供参考。本发明可针对于接收 MPEG-2 标准的该解码器来实现。

参见图 2,在上述专利申请中叙述的解码器的优选实施例中,编码器的 MPEG 数据(MPEG-1 或 MPEG-2)经过一编码数据输入电路 200 转送到该设备。然后这个数据经过信号 202 传送到起始码检测器(SCD) 204。SCD 204 识别许多起始码,这些起始码是独特的比特模式,这些用可容易地由其后电路识别的相应令牌代替。其余的数据(除起始码之外)由数据令牌传送。这个“起始码”流和数据令牌经过信号 206 传送到格式电路 208,格式电路 208 将该

数据安排为合适的格式以便存储在外部存储器中。这个数据经过信号 210 传送到同步动态随机访问存储器中 (SDRAM) 接口电路 212。

SDRAM 接口电路 212 处理很多数据流，这些数据流在接口信号 230 的单个集上是多路传输的，以便它们可写入外部 SDRAM 设备（或多个设备）或从该设备 228 读出。在每个情况下，数据暂时存储在摆动 (swing) 缓冲器 (214、216、218、220、222 和 224) 中，每个摆动缓冲器包括两个分离的 RAM 阵列。SDRAM 的地址由地址发生器 330 产生并且经过信号 332 传送到 SDRAM 接口电路 212，在经过 SDRAM 接口 230 加到外部 SDRAM 228 之前，这些地址进一步由 DRAM 接口控制器 226 处理。地址是这样产生的，使得编码的数据缓冲器 234 和许多帧存储器 232 仍然在外部 SDRAM 中。

前面提到的“起始码”令牌及数据令牌的格式码流经过信号 210 传送到 SDRAM 接口电路 212，该信号暂时存储在摆动缓冲器 214 中。这个数据被写入包括编码的数据缓冲器 (CDB) 234 的外部 SDRAM 228 的区域中。这个缓冲器具有 FIFO（先进先出）功能，在其中该数据的顺序被保持。以经过信号 236 离开该 SDRAM 接口电路之前，从 CDB234 返回的数据暂时存储在摆动缓冲器 216 中。信号 236 中的数据是与信号 210 中的数据相同的，除了它在 CDB 234 中已延迟了一个（可变的）时间之外。

从 CDB 返回的数据在电路 238 中是未格式化的，适用于存在外部 SDRAM 中，在此之前由格式化器 208 完成。但是应该提出没有限制信号 206 的总线宽度与信号 240 相同。在优选的实施例，更宽的总线宽度由信号 240 使用，以便在这点可由信号 206 支持较高瞬间数据带宽。

该数据（仍然包括“起始码”令牌，而该数据的剩余部分作为数据令牌传送）经过信号 240 传送到视频分析程序电路 242。这个电路整个地说具有进一步处理编码的视频数据的任务。具体说是视频数据的结构被“语法分析”，以便识别其分量部分。视频分析程序包括微编程的状态机 (MSM) 244，它有一个已存储的程序。命令经过信号 250 传送到哈夫曼 (Huffman) 解码器 246。命令的一些部分由哈夫曼解码器 246 翻译。该命令的剩余部分与由哈夫曼解码器产生的数据一起经过信号 255 传送到算术与逻辑单元 (ALU) 248。在这里该命令的一些部分再次由 ALU 本身翻译，而同时该命令的剩余部分和由该 ALU 产生的数据经过信号 256 传送到令牌格式器 258。哈夫曼解码器 246 可将差错情况经过信号 252 通知 MSM 244。ALU 248 经过信号 254 反馈状况码给 MSM

244。这使 MSM 能够执行“转移”(JUMP)命令,这是以在 ALU 248 中被处理的数据为条件的。ALU 内包括一个寄存器文件,以便也可存储所选的信息。“起始码”令牌有效地通知后续的数据的类型(包括在该数据令牌中)。这允许 MSM 判定后续哪个命令序列来解码该数据。除了根据从起始码得到的令牌的这个粗略判定外,视频数据的精细结构紧接着前面叙述的存储那个信息的机制,这个信息规定了 ALU 内寄存器文件中视频数据的结构,并且根据解码的数据值使用它执行条件“转移”命令以便选择交替命令序列来解码编码的数据中的精确的符号序列。

解码的数据与剩余的命令比特(哈夫曼解码器不使用的比特)经过信号 256 传送到令牌格式器 258。响应该指令比特,该数据被格式化为令牌,这些令牌可由其后的处理级识别。得到的令牌经过信号 260、262 和 264 传送到三个分开的目的地。一个令牌流 262 传送到地址发生器 330,在其中它被翻译以产生合适的地址保持编码的数据缓冲器和帧存储器,如前面所述的。第二令牌流 264 由视频定时产生电路 326 翻译,以便控制编码视频信息的最后显示的一些部分。第三令牌流 260 传送到反向模型器 266,而且继续被传送到其后的处理电路。应该明白,虽然三个令牌流(260、262 和 264)彼此是相同的,但是所提取的信息彼此是不同的。与具体电路的功能无关的那些令牌被丢弃。在令牌流 262 和 264 情况中对翻译极有用的令牌基本上是控制信息,虽然连接到令牌流 260 的电路中对翻译极有用的令牌流可更有用地被看成数据。

反向模型器 266 具有扩展在该数据中出现的零系数的任务,以便使得数据包括精确地具有 64 系数的数据块,这是经过信号 268 传送到反向锯齿形电路 270。这个电路按照两个预定码型之一重新排序数据流,并且得到可认为是二维的数据。该反向锯齿形电路包括一个小的随机访问存储器(RAM) 272,数据在进行重新排序时暂存其中。如此产生的数据经过信号 274 传送到反向量化器 276。在这里系数是非量化的并返回为它们合适的数值为反向离散余弦(DCT)函数作准备。反向 DCT 是一个可分离变换,所以它必须应用两次,垂直方向一次和水平方向一次。在这个实施例中,单个的一维反向 DCT 函数使用两次来执行安全的二维变换。该数据首先经过信号 278 进入反向 DCT 电路 280。所得数据经过信号 284 传送并存储在一个转置 RAM 282。该数据从该转置 RAM 读出,但是以与它被写入不同的顺序,以便该数据被转置(即,行和

列交换)。这个转置的数据经过信号 286 传送到反向 DCT 280, 在这里它进行第二次处理, 从这第二变换得到的数据经过信号 288 传送到场/帧电路 290。

该场/帧电路在一些情况下再次重新排序数据, 诸如经过信号 294 传送的数据是以与从外部 SDRAM 的帧存储中作为预期数据读出的方式相同的编排方式 (场或帧)。场/帧电路 290 在 RAM 292 中暂时地存储数据用于该重新排序的目的。

在外部 SDRAM 中, 如前所述, 预期数据从保存它们的帧存储器中被读出。预期数据经过两条通路 (一条通路通常用于“前向预期”和另一条通路通常用于“后向预期”, 虽然这不是严格遵守的) 被读出。一条通路包括摆动缓冲器 222 和信号 296, 而另一条通路包括摆动缓冲器 224 和信号 298。该数据由预期滤波器 300 滤波, 如果由指示给那个数据的特定预期模式提出要求, 两个预期数据 (“前向”和“后向”) 可被平均。结果产生的预期数据经过信号 302 传送到一个预期加法器 304, 这里它被加到从场/帧电路经过信号 294 传送来的数据。如此产生的解码图象信息经过信号 306 和摆动缓冲器 220 写回到第三帧存储器。

为了产生视频信号, 解码的信息经过摆动缓冲器 218 从 SDRAM 中读出, 然后经过两信号通路之一传送。色度数据经过信号 308 传送到垂直过取样器 (upsampler) 312, 该垂直过取样器对该数据过取样, 使得具有与亮度信号使用的扫描行数相同数目的扫描行。垂直过取样器 312 在行存储器 314 中存储两个色度信号的每个色度信号的一个扫描行。得到的两个色信号 (蓝色差信号和红色差信号) 经信号 316 和 318 传送到水平过取样器 320。亮度信号 (它不要求垂直过取样) 也经过信号 310 传送到水平过取样器。水平过取样器 320 有这样的任务: 按预置的比例因数之一来重取样以为最终扫描行产生一适当的象素数量。该比例系数是经过由视频定时发生 (VTG) 电路 326 提供的信号 324 选择的。这个信息简单地由从经过信号 264 加到 VTG 的令牌之一被提取出来。

由水平过取样器产生的数据经过信号 322 传送到输出多路复用器 327。响应由视频定时发生器 (VTG) 电路 326 产生的定时信号 328, 该多路复用器多路复用由输出多路复用器内部产生的包含有同步消隐和边界信息的实际的视频数据信号, 该实际视频数据信号是经过信号 322 传来的。为了产生正确的定时信号, 特别是在产生正确的边界信息数量方面, VTG 使用经过信号 264

的以令牌传送的信息。

最后得到的视频信号与许多选通脉冲、同步和消隐信号一起经过信号 334 传送到视频输出接口 336。然后该视频信号可传送到一些合适的视频显示设备。

- 5 很多其它的接口被提供。微处理器接口 340 使一个外部的微处理器能够被连接到信号 338。信号 342 接到许多电路块，允许视频解码装置的当前状态被该外部微处理器读出。另外，一些特性可由该外部微处理器控制，经过这个接口写入多个控制寄存器。

- 10 JTAG(联合测试操作组)接口 346 允许该设备的各方面经过接至信号 344 的外部设备进行控制。JTAG 接口 346 经常只有用于印刷电路板测试(组装之后)，在测试中只需控制视频解码装置的外部信号。在这个实施例中提供了附加的测试能力，而且由于这个原因，JTAG 接口 346 经过信号 348 接到所有的电路块。

- 15 电路 352 用于根据外部时钟信号 350 产生和分配时钟信号 354。这包括各种锁相环(PLL)，使能够从外部较低速时钟产生较高速内部时钟。

关于在上述专利申请中叙述的视频解码器，其显示速率是已知的，因为配置管脚(NTSC/PAL)指示是否在产生 59.94 Hz 或 50 Hz 的显示速率。该电影帧速率作为帧速率参数以 MPEG-2 视频信息流进行传输。

- 20 在事件的正常过程中，任何进程的帧都以两个场的时间显示。图象型的令牌中的一比特控制第一场的重复使该帧显示三场的时间。

E	7	6	5	4	3	2	1	0
I	I	I	I	0	I	I	I	I
0	d	x	f	p	s	s	t	t

表 1

表中：

d：双倍高度（即重复帧为两场）

- 30 x：上部场优先（在场图象中未定义）(top_field_first)

f：重复第一场(repeat_first_field)

p: 进程的帧

ss: 图象结构:

00: 上部场

01: 下部场

5 10: 帧

11: (未用)

tt: 图象类型:

00: I 图象

01: P 图象

10 10: B 图象

11: D 图象

“f” 比特被置 1 以便重复第一场。在序列重叠的情况下，这个令牌中的“f” 比特直接与在 MPEG-2 序列中的“重复第一场” 比特一致。(这是信号下拉的情况)。但是，在“进程序列” 的情况下(该术语在 MPEG 标准中定义) 是“1”，指示该序列被编码为进程序列，本地下拉被允许而且根据这里叙述的算法计算“f” 比特。

该算法在可微编程的状态机(MSM) 上执行，因此，以微码(而不是以这里所附的说明该算法的更熟悉的“C” 程序) 表示。MSM 是 16 比特机器，而且这造成一些小的复杂性，因为以 16 比特表示的数量范围受到限制。

20 这是通过公因子减少斜率的分母和分子的大小处理的。在本文件结尾给出的示例程序中，这是这样进行的，删除任何公因数 1001，然后除以 2 直至分子或分母为奇数为止。甚至这个简单情况得出数据 dx 和 dy，也是不超过 16 比特数值范围，如以在结果中所示的“min” 和“max” 指示的那样。在所述视频解码器中，数值 dx 和 dy 预先计算并且存储在表中，这些表被作为索引以确定正确的 dx 和 dy 值。结果，比率可用下面所示的最小可能的分子和分母进一步减小：

30

表 2

	Frame Rate (Hz)	Display Field Rate (Hz)			
		50		60000/1001	
		Full Form	Reduced	Full Form	Reduced
5	24000/1001	$\frac{(50 \times 1001)}{24000} - 2$	$\frac{41}{480}$	$\frac{60000 \times 1001}{24000 \times 1001} - 2$	$\frac{1}{2}$
10	24	$\frac{50}{24} - 2$	$\frac{4}{12}$	$\frac{60000}{1001 \times 24} - 2$	$\frac{498}{1001}$
15	25	$\frac{50}{25} - 2$	0	$\frac{60000}{1001 \times 25} - 2$	$\frac{398}{1001}$
20	30000/1001	Not Supported		$\frac{60000 \times 1001}{30000 \times 1001} - 2$	0

变量“d”是一个判定变量，它在每个X坐标（或每个电影帧）被更新。在每个X坐标，Y的理想值（图1中以直线表示）位于两象素（一个黑和一个白）之间。d正比于上象素的距离与下象素的距离之间的差。

如果D是负的，则理想直线更接近较低的象素。

25 如果D是正的，则该理想直线更接近较高的象素。

在每个X坐标，该算法必须选择较低的或较高的象素，然后更新d值，为下个X坐标（下一帧）做好准备。

如果D是负的，则选择较低的象素。d通过在incr1上增加进行更新。由于incr1是正的，d将增大，反映出这时直线在数值上离较低象素（在下
30 一个X坐标）距离变远。

如果D是正的，则选择较高象素。d通过在incr2上增加进行更新。由

于 incr2 是负的, d 变小, 反映出这时的直线离较高的象素 (在下一个 X 坐标) 距离变远。

因此 incr1 和 incr2 代表该算法可进行的二个可能判定的 d 的变化(即, 从理想象征线到较高象素的距离与从该线到较低象素的距离之间的差的变化)。因此, 选择了一帧周期中场重复的基本整数值之后, 象征线的象征斜率根据方程 1 确定, 该算法用于判定是否在一帧周期中加上一场重复以便以较快的场速率保持场重复的运行平均速率。

在表 2 的例子中, 一帧周期中场重复的基本整数值方便地选为 2。因为根据下拉图这个场重复必须被加上以保持运行平均速率。但是, 也可以选择
10 在整体上超过场重复得到的基本整数值, 这时可根据所建立的下拉图删除所选的场重复。

还应该注意到, 作为在该表中存储 dx 与 dy 和计算 incr1 和 incr2 替代方案, 在该表中直接地存储预先计算的 incr1 与 incr2 同样也是有效的。

图 3 表示以适当次数解码和显示一场过程。

15 图 4 表示一个示例的算法, 用来从场速率和帧速率确定 dx 和 dy 。在这个例子中, 大于 1000 的场速率和帧速率的值被翻译为代表 1001 的倍数, 例如 24000 的帧速率实际上代表 $24000/1001$ Hz 的帧速率。

图 5 表示初始化 incr1 、 incr2 和 d 的算法。该算法在第一帧之前使用。 dx 和 dy 的值是整数, 因此分数 dy/dx 代表“直线的斜率”。

20 图 6 表示确定显示一帧两场或三场的一个算法。该算法每帧使用一次。 incr1 和 incr2 的值是由该初始化算法确定的。 d 的值是由这个算法对前面的帧产生的值, 或者初始值是在第一帧的情况下一个时间算法。

这个方法的主要优点是:

1、所有要求的下拉变换使用相同算法进行的。
25 2、一旦参数 dx 和 dy 已知 (而且这些参数可存储在该表中), 则不需要乘法或除法。

3、该算法工作于任意长度帧序列 - 数值的增加不是无限的 (不考虑字宽, 这将最终导致数值表示方面的问题)。

4、具有重复场的帧均匀地分布在序列中。

30 5、为了该算法工作, 只需要保持很小的状态。仅仅是当前的“ d ”值并可能有 INCR1 和 INCR2 (虽然这些值可重新计算或在每帧周期在表中查找)。

下面过程是 Bresenham 的线提取算法。

```

void bresenham(x1, y1, x2, y2)
{
    int dx, dy, incr1, incr2, d, x, y, xend;

    5      dx = abs(x2-x1);
          dy = abs(y2-y1);
          d = 2 * (dy - dx);

          incr1 = 2 * dy;
          incr2 = d;

          if (x1 > x2)
          {
            10      x = x2;
                  y = y2;
                  xend = x1;
          }
          else
          {
            x = x1;
            y = y1;
            xend = x2;
          }

          15      printf("(%d, %d)\n", x, y);

          while (x < xend)
          {
            x ++;
            if (d < 0)
                d += incr1;
            else
            {
                20      y ++;
                        d += incr2;
            }
            printf("(%d, %d)\n", x, y);
          }
        }
    }

    25

```

下面的程序表示修改的算法（作为函数 `threl_fieldsc()`）以便计算哪些帧显示三场的时间。通过检验一百万电影帧来检查每个可能的变换以保证场速率确定达到所要求的值。

```

5
#include <compiler.h>
#include <pddtypes.h>
#include <stdlib.h>
#include <stdio.h>

Boolean three_fields(int dx, int dy, int *d, Boolean initialise)
{
10
    int incr1, incr2, x, y, xend;
    int r = False;

    incr1 = 2 * dy;
    incr2 = 2 * (dy - dx);

    if (initialise)
    {
        *d = incr2;
    }
    else
15
    {
        if (*d < 0)
            *d += incr1;
        else
        {
            r = True;
            *d += incr2;
        }
    }
20
    return r;
}

double check_ratio(int dx, int dy, int limits[2])
{
    int d;
    int frame, field=0, num_frames=10000000;
    double ratio, field_rate;
    int three = 0, two = 0;

25
    (void) three_fields(dx, dy, &d, True );

    limits[0] = limits[1] = 0;
    if (d < limits[0]) limits[0]=d;
    if (d > limits[1]) limits[1]=d;

30

```

```

5      for (frame = 0; frame < num_frames; frame++)
      {
          if ( three_fields(dx, dy, &d, False ) )
          {
              field += 3;
              three ++;
          }

          else
          {
              field += 2;
              two ++;
          }
          if (d < limits[0]) limits[0]=d;
          if (d > limits[1]) limits[1]=d;
10      }

      ratio = ((double)field) / ((double) frame);

      return ratio;
  }

15  static int frame_rates[ ] = /* input frame rates */
  {
      -1,
      24000, /* numbers > 1000 express a numerator - denominator =
1001 */
      24,
      25,
      30000,
      30,
      50,
      60000,
      60,
      -1, -1, -1, -1, -1, -1 -1};

      static int field_rates[ ] = /* output display rates */
20  {
      50, 60000
  };

      double real_rate (int rate)
      {
          if (rate > 1000)
              return ((double) rate) / 1001.0;
          else
              return ((double) rate);
      }

25  void main (int argc, char **argv)
  {
      int dx, dy, field_index, frame_index;
      int limits[2];
      double ratio, field_rate;

      for (field_index = 0; field_index < 2; field_index++)
      {
          for (frame_index = 1; frame_rates[frame_index] > 0; frame_index
30

```

```

    ++)
    {
5      if ( (real_rate(frame_rates[frame_index]) * 2.0) <=
          real_rate(field_rates[field_index]) )
        {
          dy = field_rates[field_index];
          dx = frame_rates[frame_index];

          if ( (field_rates[field_index] <= 1000) ||
              (frame_rates[frame_index] <= 1000) )
            /* NB if both have the 1001 then don't bother! */
            {
10              if (field_rates[field_index] > 1000)
                  dx *= 1001;
                  if (frame_rates[frame_index] > 1000)
                      dy *= 1001;
            }

            dy -= (2 * dx);

            /* limit ratio by dividing by two */
15            while ( ( (dx & 1) == 0) && ( (dy & 1) == 0) )
                {
                    dx >>= 1;
                    dy >>= 1;
                }

            ratio = check_ratio (dx, dy, limits);

            printf("output field rate = %d%s, input frame rate =
20            %d%s\n",
                field_rates[field_index],
                ((field_rates[field_index]>1000) ? "/1001":""),
                frame_rates[frame_index],
                ((frame_rates[frame_index]>1000) ? "/1001":"") );

            printf("dx = %d, dy = %d\n", dx, dy);
            field_rate = frame_rates[frame_index] * ratio;
            if (frame_rates[frame_index]>1000)
                field_rate /= 1001.00;

25            printf("ratio = %4.12g, field_rate = %4.12g\n", ratio,
                field_rate);
            printf("(field_rate = %4.12g/1001)\n", field_rate*1001);
            printf("min = %d, max = %d\n\n", limits[0], limits[1]);
        }
    }
}
30

```

前面各页的程序得到以下输出：

```
output field rate = 50, input frame rate = 24000/1001
dx = 12000, dy = 1025
ratio = 2.0854166, field_rate = 49.9999984016
(field_rate = 50049.9984/1001)
min = -21950, max = 2000
```

```
output field rate = 50, input frame rate = 24
dx = 12, dy = 1
ratio = 2.0833333, field_rate = 49.9999992
(field_rate = 50049.9991992/1001)
min = -22, max = 0
```

```
output field rate = 50, input frame rate = 25
dx = 25, dy = 0
ratio = 2, field_rate = 50
(field_rate = 50050/1001)
min = -50, max = 0
```

```
output field rate = 60000/1001, input frame rate = 24000/1001
dx = 750, dy = 375
ratio = 2.5, field_rate = 59.9400599401
(field_rate = 60000/1001)
min = -750, max = 0
```

```
output field rate = 60000/1001, input frame rate = 24
dx = 3003, dy = 1494
ratio = 2.4975024, field_rate = 59.9400576
(field_rate = 59999.9976576/1001)
min = -3018, max = 2982
```

```
output field rate = 60000/1001, input frame rate = 25
dx = 25025, dy = 9950
ratio = 2.3976023, field_rate = 59.9400575
(field_rate = 59999.9975575/1001)
min = -30150, max = 19850
```

```
output field rate = 60000/1001, input frame rate = 30000/1001
dx = 1875, dy = 0
ratio = 2, field_rate = 59.9400599401
(field_rate = 60000/1001)
min = -3750, max = 0
```

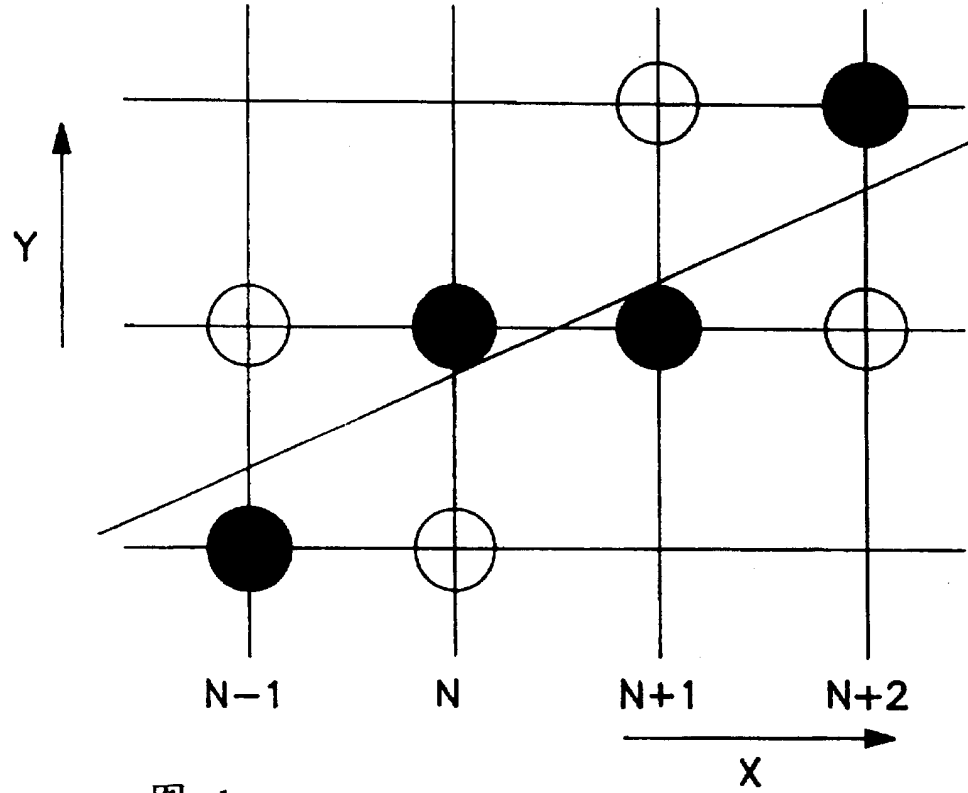
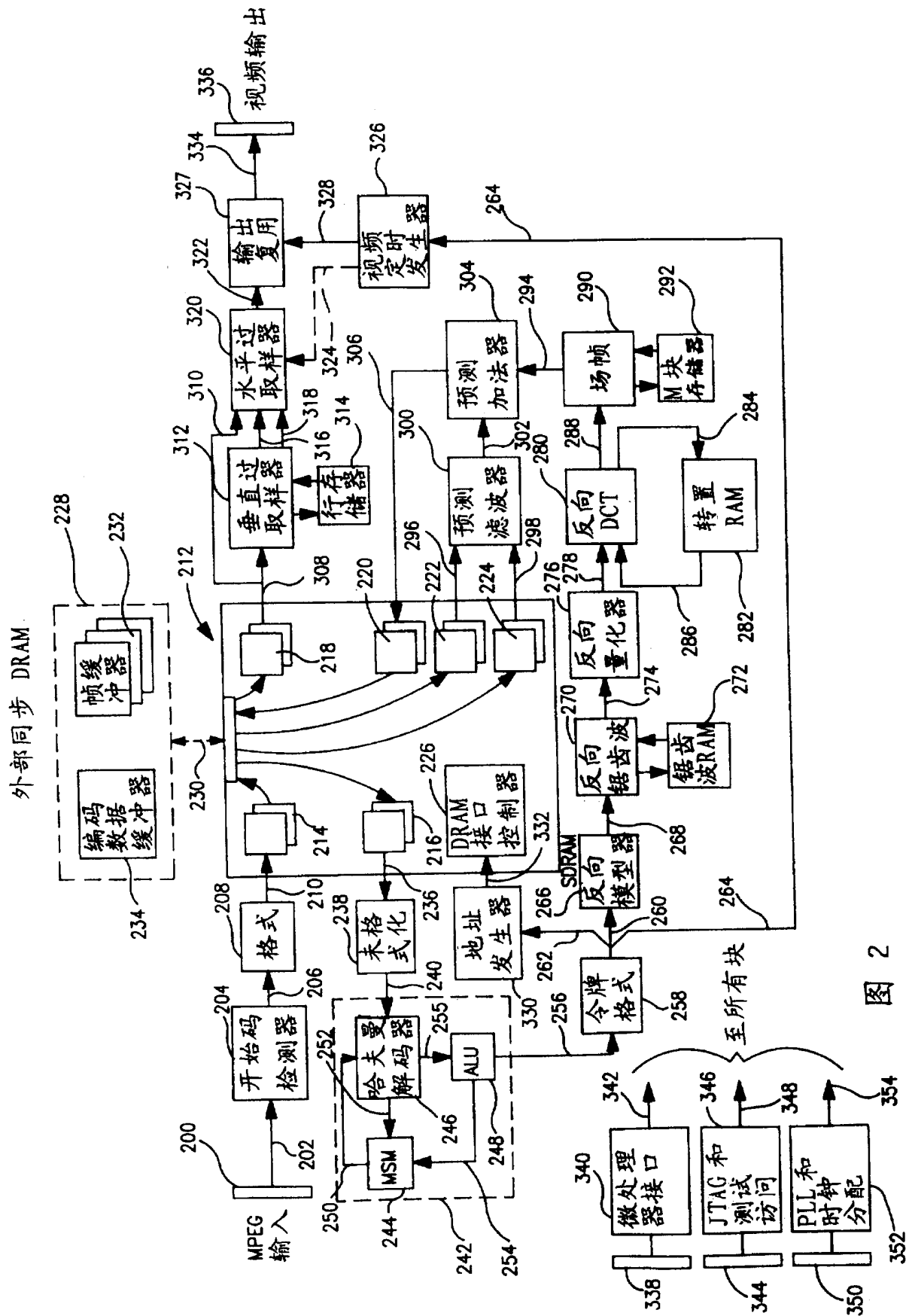


图 1



2
R

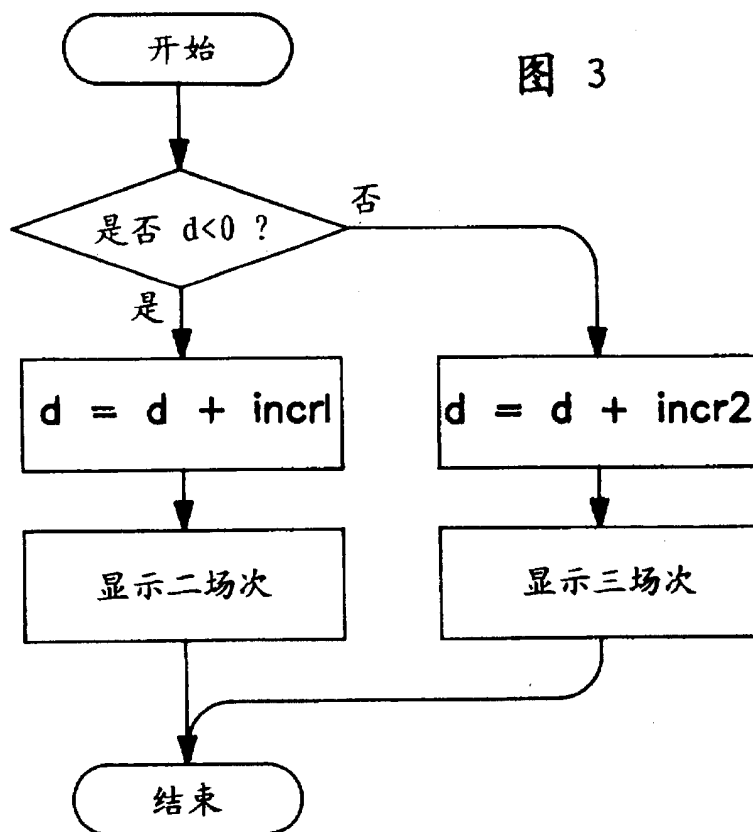
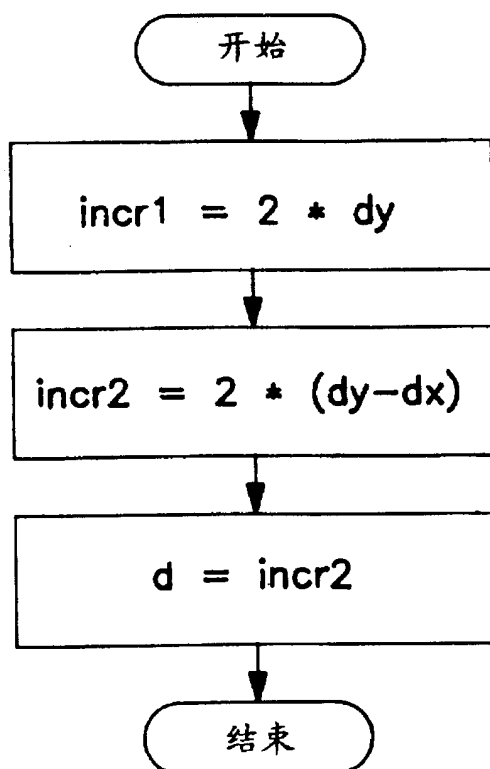


图 4



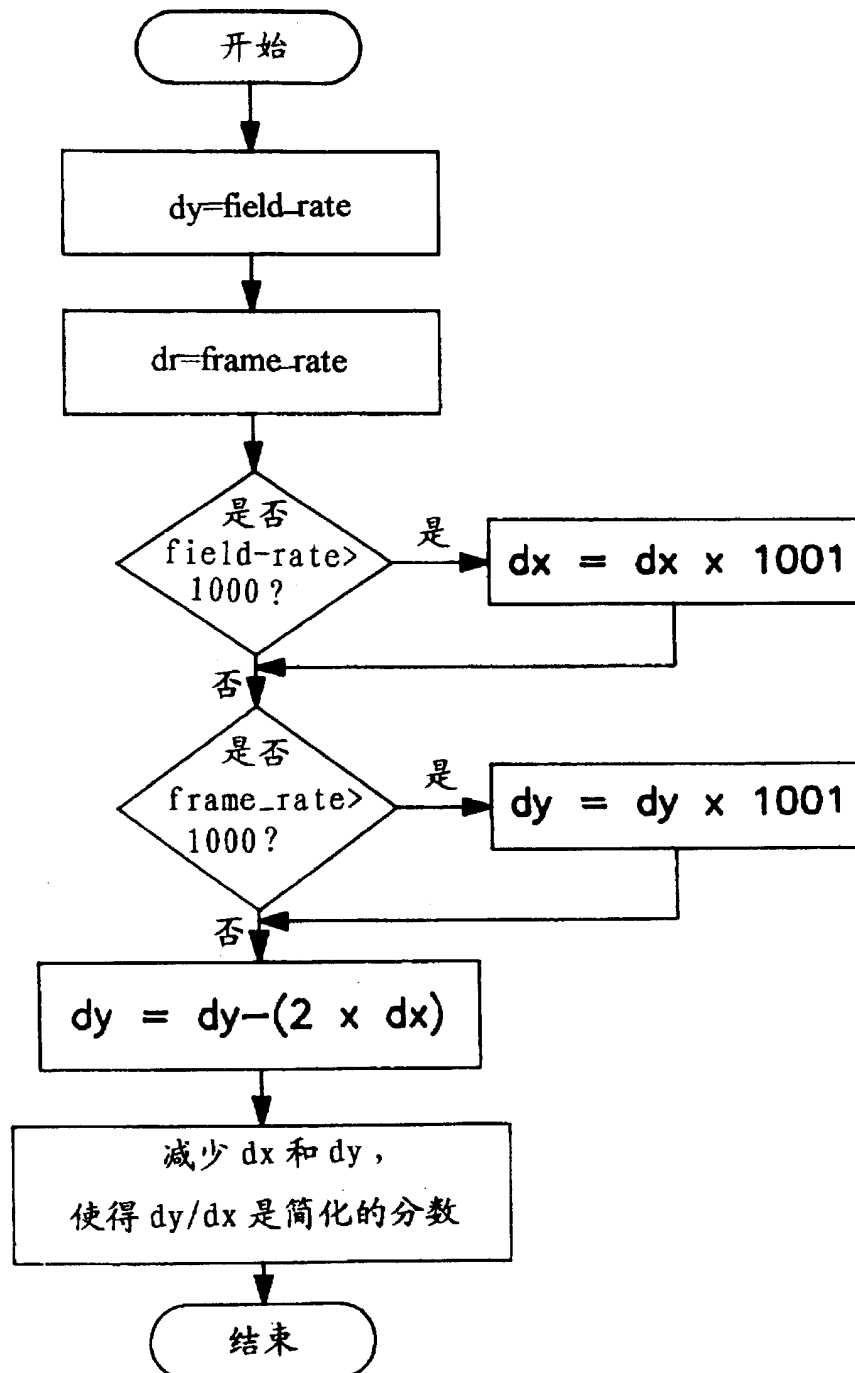


图 5

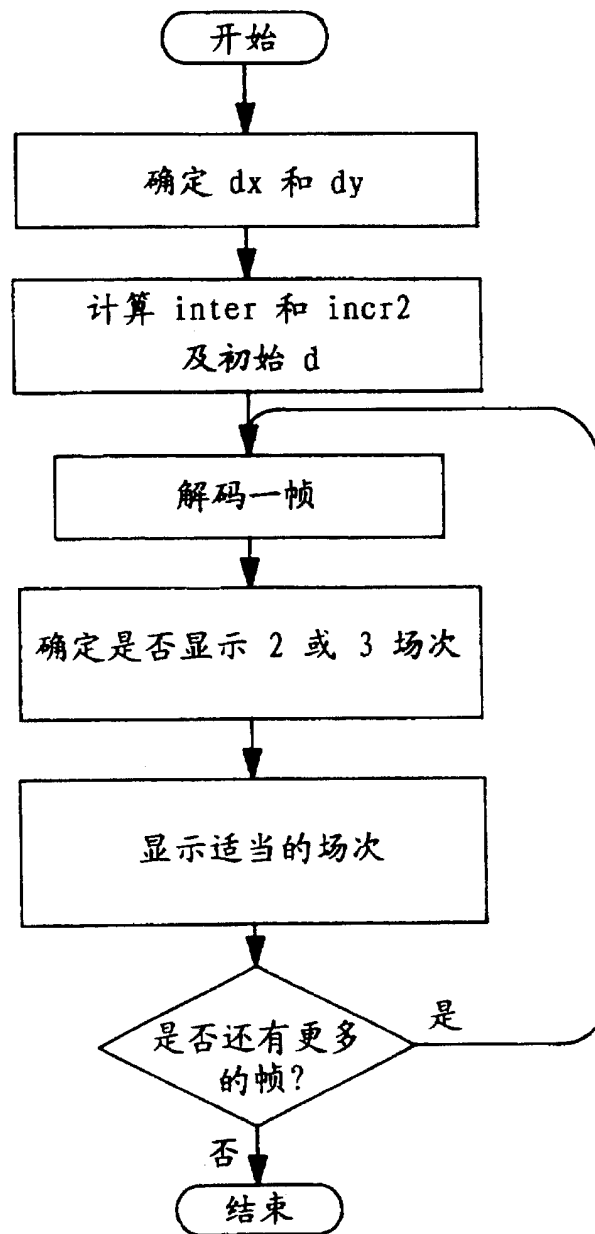


图 6