发明名称
利用新的部分更新图像的子像素着色

摘要
本发明涉及利用新的部分更新图像的子像素着色。在图像更新中，显示装置仅接收显示器的图像的新的部分（1110）而不接收图像中剩余的、未变化的部分。该显示装置对新的部分执行子像素着色（SPR）操作（454），而不对整个图像重复执行SPR。提供了有效的技术在图像的新的部分和其他部分之间的边缘处获得良好的外观。还提供了其他特征。
1. 一种通过显示单元显示图像的方法，所述显示单元包括子像素，每个子像素发射数种基色中的一种基色并具有基于利用该子像素的子像素值定义的子像素状态的亮度，其特征在于，所述方法包括：

接收图像信号，每个图像信号与图像或图像中的新的部分相关联，每个图像信号包括相关联的图像或图像中的新的部分的各像素的像素数据，其中与新的部分相关联的每个图像信号不包括在该新的部分之外的至少一个像素的像素数据；和

对于每个所述图像信号，通过电路执行相关联的子像素着色操作，相关联的子像素着色操作将每个像素与作为显示单元的显示区域的显示区域相关联，所述子像素着色操作对位于与相关联的图像或新的部分的一个或多个像素相关联的中的一个或多个显示区域内的由一个或多个子像素构成的子像素组中的每个子像素提供子像素值，其中至少一个显示区域不包含至少一种基色的整个子像素；

其中至少一个图像信号与新的部分相关联，并且相关联的子像素着色操作不对位于与新的部分中的任何像素不相关联的区域内的至少一个子像素提供子像素值。

2. 根据权利要求1所述的方法，其中在存储器中保存各图像的子像素值；和

其中对于至少所述与新的部分相关联的图像信号，相关联的子像素着色操作在存储器中未执行子像素着色操作的各子像素不改写子像素值。

3. 根据权利要求1所述的方法，其中由至少一个所述子像素着色操作产生的至少一个子像素值在显示单元的色域外，所述方法进一步包括使用色域内的值替代所述至少一个子像素值。

4. 一种电路，用于执行根据权利要求1所述的方法。

5. 一种通过显示单元显示图像的方法，所述显示单元包括子像素，每个子像素发射数种基色中的一种基色并具有基于利用该子像素的子像素值定义的子像素状态的亮度，其特征在于，所述方法包括：

接收图像信号，每个图像信号与图像或图像中的新的部分相关联，每个图像信号包括相关联的图像或图像中的新的部分的各像素的像素数据，其中与新的部分相关联的每个图像信号不包括位于该新的部分之外的至少一个像素的像素数据；和

对于每个所述图像信号，通过电路执行相关联的子像素着色操作，相关联的子像素着色操作将每个像素与作为显示单元的显示区域的显示区域相关联，所述子像素着色操作对位于与相关联的图像或新的部分的一个或多个像素相关联的中的一个或多个显示区域内的由一个或多个子像素构成的子像素组中的每个子像素提供子像素值；

其中至少一个图像信号与新的部分相关联，并且相关联的子像素着色操作不对位于与新的部分中的任何像素不相关联的区域内的至少一个子像素提供子像素值；

其中对于与新的部分P1相关联的至少一个图像信号S1，所述新的部分P1包括处在与新的部分P1的预定侧上的在新的部分P1的边缘处的像素相关的联的显示区域内的子像素SP1，相关联的子像素着色操作不提供子像素SP1的子像素值以使得子像素SP1的子像素值不变化；

其中对于至少一个其他图像信号S2，相关联的子像素着色操作提供子像素SP1的子像素值，其中子像素SP1处在与第一像素相关联的区域内，其中图像信号S2与包含第一像素的图像相关联或与包含第一像素的新的部分P2相关联，并且相关联的子像素着色操作确
定子像素 SP1 的子像素值作为数个像素的色坐标的加权总和，其中在该加权总和中，对第一像素给出在大小上不大于在第一像素的所述预定侧上的第二像素的权重。

6. 根据权利要求 5 所述的方法，其中图像信号 S2 与包含不处在所述预定侧上的边缘处的第一像素的新的部分 P2 相关联。

7. 根据权利要求 5 所述的方法，其中对第一像素给出在大小上小于第二像素的权重。

8. 根据权利要求 5 所述的方法，其中基色包括作为子像素 SP1 的颜色的黄色 PC1；

其中于对与在所述预定侧的边缘处包括一个或多个像素的新的部分相联的每个图像信号，所述一个或多个像素的相联的显示区域包含黄色 PC1 的一个或多个子像素，相关联的子像素着色操作对具有颜色 PC1 且处在与所述预定侧的边缘处的新的部分内的像素相关联的显示区域内的任何子像素不提供子像素值。

9. 根据权利要求 8 所述的方法，其中颜色 PC1 是蓝色。

10. 一种电路，用于执行根据权利要求 5 所述的方法。

11. 一种通过显示单元显示图像的方法，所述显示单元包括子像素，每个子像素发射数种基色中的一种基色并具有基于利用该子像素的子像素值定义的子像素状态的亮度，其特征在于，所述方法包括：

接收图像信号，每个图像信号与图像或图像中的新的部分相关联，每个图像信号包括相关联的图像或图像中的新的部分的各像素的像素数据，其中与新的部分相关联的每个图像信号不包括处在该新的部分之外的至少一个像素的像素数据；和

对于各该图像信号，通过电路执行相关联的子像素着色操作，相关联的子像素着色操作将各像素与作为显示单元的要显示该像素的区域的显示区域相关联，所述子像素着色操作对用于与相关联的图像或新的部分的一个或多个像素相关联的一个或多个显示区域内的由一个或多个子像素组成的子像素组中的每个子像素提供子像素值；

其中至少一个图像信号与新的部分相关联，并且相关联的子像素着色操作对处在与新的部分中的任何像素不相关联的区域内至少一个子像素不提供子像素值；

其中对于任何新的部分 P1 相关联的每个图像信号 S1，所述新的部分 P1 包括处在与新的部分 P1 的预定侧上在新的部分 P1 的边缘处的像素相关联的显示区域内的子像素 SP1，相关联的子像素着色操作不提供子像素 SP1 的子像素值以使得子像素 SP1 的子像素值不变化；

其中对于至少一个图像信号 S2，相关联的子像素着色操作提供子像素 SP1 的子像素值，其中子像素 SP1 处在与第一像素相关联的显示区域内，其中图像信号 S2 与包含第一像素的图像相关联或与包含不处在所述预定侧的边缘处的第一像素的新的部分 P2 相关联，并且相关联的子像素着色操作确定子像素 SP1 的子像素值作为数个像素的色坐标的加权总和，其中在该加权总和中，对第一像素给出在大小上不大于第一像素的所述预定侧上的第二像素的权重。

12. 根据权利要求 11 所述的方法，其中对第一像素给出在大小上小于第二像素的权重。

13. 一种电路，用于执行根据权利要求 11 所述的方法。

14. 一种通过显示单元显示图像的方法，所述显示单元包括子像素，每个子像素发射数种基色中的一种基色并具有基于利用该子像素的子像素值定义的子像素状态的亮度，其特
权利要求书

其中每个图像的各子像素值保存在存储器的各自的比特中；
其中各子像素值包含高位有效位部分和低位有效位部分；
其中对于与新的部分相关联的至少一个图像信号，对于处在新的部分与所述预定侧相对的一侧位于新的部分之外与包含基色 PC1 的第一子像素的显示区域相关联的至少一个像素，相关联的子像素着色操作确定第一子像素的子像素值的至少高位有效位部分，并将第一子像素的子像素值的高位有效位部分而非低位有效位部分保存在各自的比特中。

15. 根据权利要求 14 所述的方法，其中将不靠近显示单元的屏幕的边缘的子像素细分为组，每个组包含所有基色的子像素，并在每个组中：
在存储器的连续比特中保存包含基色 PC1 的不同基色的至少两个子像素的子像素值的高位有效位部分；和
在存储器的连续比特中保存包含基色 PC1 的不同基色的至少两个子像素的子像素值的低位有效位部分。

16. 根据权利要求 15 所述的方法，其中每个组包含基色 PC1 的正好一个子像素。

17. 根据权利要求 15 所述的方法，其中在存储器的连续比特中保存基色 PC1 和另一种基色 PC2 的两个子像素的子像素值的高位有效位部分；和
在存储器的连续比特中保存基色 PC1 和 PC2 的所述两个子像素的子像素值的低位有效位部分。

18. 根据权利要求 15 所述的方法，其中在每个组中，在存储器的不连续比特中保存至少一个子像素的子像素值的高位有效位部分和低位有效位部分。

19. 根据权利要求 15 所述的方法，其中基色 PC1 是蓝色。

20. 根据权利要求 17 所述的方法，其中基色包括红色、绿色和蓝色，并且基色 PC1 是蓝色，而基色 PC2 是红色。

21. 根据权利要求 17 所述的方法，其中基色还包括白色。

22. 根据权利要求 17 所述的方法，其中对于与包含在新的部分与所述预定侧相对的一侧处于新的部分的外部且与包含基色 PC1 的子像素的显示区域相关联的数个像素的新的部分相关联的每个图像信号，对于在新的部分与所述预定侧相对的一侧处于新的部分的外部且与包含基色 PC1 的子像素的显示区域相关联的每个像素，相关联的子像素着色操作确
定子像素的子像素值的至少高位有效位部分，并且将高位有效位部分而非子像素的子像素值的低位有效位部分保存在各自的比特中。

23. 一种电路，用于执行根据权利要求 14 所述的方法。
利用新的部分更新图像的子像素着色

技术领域
[0001] 本发明的涉及子像素着色。

背景技术


上述’280，’213，’381，’278，’404，’277和’741公开申请中的每一个都通过参考在这里整体引入。


上述’992，’058，’906，’302，’380和’297申请和’368专利中的每一个都通过参考在这里整体引入。


在如下的申请中描述了对显示系统及其操作方法的额外的改进和实施例：(1) 2006 年 4 月 4 日提交的发明名称为“EFFICIENT MEMORY STRUCTURE FOR DISPLAY SYSTEM WITH NOVEL SUBPIXEL STRUCTURES”的专利合作条约 (PCT) 申请 No. PCT/US 06/12768；(2) 2006 年 4 月 4 日提交的发明名称为“SYSTEMS AND METHODS FOR IMPLEMENTING LOW-COST GAMUT MAPPING ALGORITHMS”的专利合作条约 (PCT) 申请 No. PCT/US06/12766；(3) 2006 年 4 月 4 日提交的发明名称为“SYSTEMS AND METHODS FOR IMPLEMENTING IMPROVED GAMUT MAPPING ALGORITHMS”的美国专利申请号 No. 11/278,675，其还以美国专利申请公开 2006/0244686 公开；(4) 2006 年 4 月 4 日提交的发明名称为“PRE-SUBPIXEL RENDERED IMAGE PROCESSING IN DISPLAY SYSTEMS”的专利合作条约 (PCT) 申请 No. PCT/US 06/12521；以及 (5) 2006 年 5 月 19 日提交的发明名称为“MULTIPRIARY COLOR SUBPIXEL RENDERING WITH METAMERIC FILTERING”的专利合作条约 (PCT) 申请 No. PCT/US 06/19657。上述这些权利共有的申请中的每一个也都通过参考在这里整体引入。

如上面提及的一些专利申请中描述的那样，通过大量被称为像素的区域 106（图 1）代表图像 104（图 1）。各像素 106 都与必须由显示器 110 中的一组子像素来显示的颜色相关联。各子像素显示“基色”，即各子像素都与某种色调和饱和度相关联。通过混合基色获得其他颜色。各像素 106 被映射到将要用来显示像素的颜色的一组子像素，这组子像素包含一个或多个的子像素。

在某些显示器中，每一组子像素都包括每种基色的子像素。子像素很小，并且相互
间隔很近，以提供期望的分辨率。然而这样的结构是不经济的，因为它不匹配人眼视觉的分辨率。相比色度差，人类对于亮度（luminance）的差异感更明显。因此，某些显示器将输入像素 106 映射到不包括每种基色的子像素的子像素组。尽管色度分辨率降低，但是可将亮度分辨率保持很高。

[0011] 如图 1 所示，并且在 2006 年 11 月 30 日公开为 No.WO 2006/127555A2 的 PCT 申请和 2006 年 11 月 2 日公开为 No.US 2006/0244686A1（美国专利申请号 11/278,675）中描述的由红色子像素 120R、蓝色子像素 120B、绿色子像素 120G 和白色子像素 120W 所有这些子像素 120F 面板上都是相同的。每一组子像素由在同一行上两个相邻的子像素组成。这些组 124 在下文中被称为“对”。每个对 124 由红色子像素 120R 和绿色子像素 120G 组成（这样的对在下文中被称为“RG 对”），或是每个对由蓝色子像素 120B 和白色子像素 120W 组成（“BW 对”）。在每个 RG 对中，红色子像素位于绿色子像素的左侧。在每个 BW 对中，蓝色子像素处在左侧。在每行和每列中 RG 对和 BW 对交替。

[0012] 图像中的第 x 列第 y 行上的各像素 106（下文中称为像素“106,x,y”）被映射到第 x 行第 y 行上的子像素对 124（下文中的“124,x,y”）。在显示器 110 中，连续的指数 x 和 y 表示连续的对，但不是连续的子像素。各个对 124 仅有两个子像素，并提供在亮度上而非色度上的高范围和高分辨率。因此，如图 2 所示，并且如在前述的某些专利申请中所述的“子像素着色操作（SPR）”中，一部分输入像素的亮度可能必须偏移至邻近的对 124。

[0013] 图 2 示出了对于红色和绿色子像素的 SPR 操作。可以相似的方式来处理蓝色和白色子像素。SPR 操作以线性方式计算分别定义了红色、绿色、蓝色和白色子像素的亮度的值 R_{x}, G_{x}, B_{x}, W_{x}，即亮度是子像素值的线性函数（对于不同的色素可使用不同的函数）。值 R_{x}, G_{x}, B_{x}, W_{x} 随后被用来确定对子像素提供的电信号以获得期望的亮度。

[0014] 图 2 示出了在每个子像素对 124 上叠加的像素 106。蓝色和白色子像素没有示出。显示区域被细分成以每个 RG 对 124 为中心的“采样”区域 250。可以不同的方式来定义采样区域 250，并且在图 2 中选择了菱形区域 250。除了显示器的边缘位置外，区域 250 彼此之间是全等的（congruent）。

[0015] 在线性 RGBW 色坐标系统下，表示各像素 106 的颜色。对于每个 RG 对 124,x,y, 红色子像素的 R_{x} 值被确定为与以 RG 对 124,x,y 为中心的采样区域 250 重叠的所有像素 106 的 R 坐标的加权和。权值被选择为加权和 1，并与采样区域 250 和各像素 106 重叠的区域呈比例。特别的，如果子像素对 124,x,y 不位于显示器的边缘位置，则红色值 R_{x} 为：

\[ R_{x} = 1/2*R_{x,y-1} + 1/8*R_{x-1,y} + 1/8*R_{x-1,y-1} + 1/8*R_{x-1,y+1} + 1/8*R_{x+1,y} + 1/8*R_{x+1,y+1} \]

[0016] 句话来说，通过利用如下的滤波器内核对像素 106 的各个 R 坐标应用 3×3 菱形滤波器来着色红色子像素 120R：

\[
\begin{bmatrix}
0 & 1/8 & 0 \\
1/8 & 1/2 & 1/8 \\
0 & 1/8 & 0 \\
\end{bmatrix}
\]

[0017] 还可对绿色、蓝色和白色子像素（除了边缘之外）使用相同的滤波器内核。还可使用其他的滤波器内核。参见例如前面提到的美国专利公开 No.2005/0225563。

[0018] 期望提供一种子像素着色技术，其在计算上成本低，不需要投入过多的资金，并且
在功率消耗方面效率高。

发明内容
[0021] 这部分总结了本发明的一些特征。在随后的部分中还描述了其他特征。本发明是由所附的权利要求限定的，其通过参考被引入在这部分当中。
[0022] 本发明的某些实施例提供了像素着色技术，其在运算方面节约成本，不需要花费大量的金钱并且在功率消耗方面是高效的。然而，本发明并不局限于这些实施例。
[0023] 在传统的显示器中，是以帧显示数据。一幅帧显示整个图像所需的时间间隔。即使部分图像是未变化的，仍对整个图像的各帧都执行像素着色。对于图像的未变化部分执行 SPR 在运算上是不高效的且在功率方面也是浪费的。因此，本发明的某些实施例不对图像的未变化部分再执行 SPR。
[0024] 还期望的是实现“位块图像传送”(bit blit) 操作，其中显示器仅接收图像的新的部分的像素数据 106。如这里所使用的，“新的部分”表示显示器接收的用于更新图像的部分；“新的部分”不一定都是全新的，即它还可包括图像的未变的部分；事实上，如果整个图像是未变化的，则所有的新的部分表示表图像的未变化部分。由于诸如 (1) 和 (2) 等 SPR 操作从多个像素（其中一些像素不是新的部分）确定子像素值，因此仅对图像的新的部分执行 SPR 是有问题的。例如，必须利用不是新的部分的像素的 RGBW 坐标确定新的部分的边缘处子像素值。某些实施例保存整个图像的 RGBW 数据以实现新的部分边缘处的 SPR。其他实施例根本不保存 RGBW 数据以降低存储器的需求。因此，在某些实施例中，在 SPR 之后立即丢弃 RGBW 数据。因此，当正在显示新的部分时，在新的部分的边缘处会发生图像的降级。然而，某些实施例提供了用于降低这样的降级的有效的技术。
[0025] 本发明并不局限于 RGBW 显示器或是上面讨论过的其他特征，除由所附的权利要求书中限定的之外。

附图说明
[0026] 图 1示出了将像素构成的图像映射到具有子像素的显示器的现有技术；
[0027] 图 2 是根据现有技术的子像素着色操作的几何示意图；
[0028] 图 3 是根据本发明某些实施例的显示设备的方框图；
[0029] 图 4示出了图 3的显示设备的某些实施例中的数据路径；
[0030] 图 5示出了具有对角线的图像；
[0031] 图 6示出了图 3的显示设备的某些实施例中的数据路径；
[0032] 图 7A、图 7B示出了在图 5的图像处理的不同阶段中的可能的子像素值；
[0033] 图 8 是根据本发明某些实施例的子像素着色的流程图；
[0034] 图 9 是根据本发明某些实施例的色域钳位 (gamut clamping) 的流程图；
[0035] 图 10 是图 3的显示设备的一部分的前视图，以描绘图 9的色域钳位操作的某些方面；
[0036] 图 11 至图 13示出了在图像部分的更新中的子像素区域；
[0037] 图 14示出了本发明的某些实施例中的像素、子像素和帧缓冲器中的子像素数据排列。
具体实施方式
[0038] 在本部分中描述的实施例阐述但其不用来限定本发明。本发明是由所附的权利要求来限定。
[0039] 现在将会描述适用于位块图像传送（bit-blit）和非位块图像传送（nonbit-blit）的图像处理的一些 SPR 技术。
[0040] 子像素着色中执行的亮度偏移可能不利地导致诸如模糊或局部对比度损失等图像降级。可通过应用锐化滤波器（例如 DOG，即高斯差分）来改善图像。见例如前面提到的 PCT 申请 WO 2006/127555。关于图像质量的其他额外的改善也是期望的。
[0041] 进一步，如上所述的一些操作会导致某些子像素值在色域外，特别是在对色域限制明亮度（brightness）以降低功率消耗时。强迫子像素值进入可利用的色域可能引起图像变形，例如降低局部对比度，因此这样的变形必须被最小化。期望改善色域映射操作，特别是在低明亮度环境中。
[0042] 图 3 显示了与本发明的某些实施例一起使用的显示装置的方框图。例如，这可以是液晶显示器（LCD）。显示单元 110 可以如图 1 所示。由背光单元 310 发射的光经过显示器 110 的子像素到达观察者 314。以数字形式将图像数据 104 提供给用于执行如图 2 所示的子像素着色和其他一些可能的操作的图像处理电路 320，并向显示器 110 提供子像素值 R,G,B,W。通过适当的修改（例如，在显示单元 110 提供的亮度是显示单元接收到的子像素值的非线性函数的情形下的伽玛转换）从 SPR 处理中生成的 R,G,B,W 值获得上述这些子像素值。向显示单元 110 提供的各子像素值定义了为获得期望的图像必须由相应的子像素透射的光量。图像处理电路 320 还向背光单元 310 提供用于指定背光单元的输出功率的控制信号 BL。为了降低功率消耗，输出功率 BL 应该由最大的子像素值所需的功率一样高。因此，可基于子像素值动态地控制输出功率 BL。这称作动态背光控制（DBLC）。电路 320 调节子像素值 RGB 以便在 BL 较小时子像素能透射。在特别强调功率的环境下（例如，在诸如移动电话等电池操作系统中），BL 值最高的子像素值所需的值较低。这称为“侵略性 DBLC”（“aggressive DBLC”）。侵略性 DBLC 可能导致对比度方面的损失。
[0043] 图 4 显示了电路 320 的某些实施例中的数据路径。方框 410 将图像 104（各像素 106 的颜色）转换到线性色彩空间，例如线性 RGB。方框 420 将图像从线性 RGB 空间转换到线性 RGBW 表示。方框 430 使用线性 RGBW 数据以确定对于 DBLC 或侵略性 DBLC 操作而言背光单元的输出功率信号 BL，并将信号 BL 提供给背光单元 310。方框 420 还向方框 444 提供关于信号 BL 的信息。方框 444 使用该信号来构放 RGB 占位以调节背光单元的输出功率 BL。缩放操作会驱使某些颜色到显示器 110 的色域外，特别是对于侵略性 DBLC 而言。方框 450 执行色域映射（色域映射）操作以用色域内的颜色替代色域外的颜色。
[0044] 方框 454 对于方框 450 的输出执行子像素着色（例如，如图 2 所示）。此外，还可以应用锐化滤波器。在前面提及的 PCT 申请 WO 2006/127555 和 2006 年 11 月 2 日公开的美国专利申请公开 2006/0244686 中描述了称为“meta luma”锐化的示例，这两份专利文件都通过参考引人到这里。更具体的，方框 420 中从 RGB 到 RGBW 的转换不是唯一的，这是因为相同的颜色具有不同的 RGBW 表示。这样的表示在一些文献中被称为“条件等色 (metamer)”。（其他文献使用术语“条件等色”来表示有不同光谱功率分布但被感知为相同颜色的电磁
波，但是不同的 RGBW 显示不必然意味着不同的光谱功率分布。]metaluma 锐化基于像素 106 相对于周边的相对亮度绝选择每个像素 106 的条件等色。假设像素 106 与紧上方、紧下方、紧左侧、紧右侧的周边像素相比更明亮。如果将明亮的像素 106 映射到 BW 对 124，则期
望选择具有较大 W 坐标的条件等色以提高 BW 对的亮度。如果将明亮的像素 106 映射到 RG
对，则期望选择具有较大 R 和 G 坐标的条件等色，因此得到较小的 W 坐标。

[0045] 锐化的另一示例是高斯差分。还可应用其他类型的锐化。

[0046] 所得到的子像素值被提供给显示器 110（如果显示器 110 中的子像素亮度不是子
像素值的线性函数，则可能在伽马转换之后）。图 4 不是可执行的全部操作的全部表示。例
如，还可追加抖动和其他操作。此外，操作不必独立地执行或是按照如上所述的顺序执行。

[0047] 图 1 所示的显示器 110 对于某些特征而言可比其他特征更佳地（更锐利地）显示。
例如，由于各行子像素 120 都包括全部色系（红色、绿色、蓝色和白色）的子像素，因此水平
线可以作成相当锐利。出于相似的缘故，垂直线也是锐利的。然而，由于子像素对 124 的各
个对角线都仅包括 RW 对或仅包括 RG 对，因此较难使得对角线变得锐利。如果图像 104 具
有映射到 RW 对 124 的对角线或 BW 对的对角线的对角线，则该对角线将会变得模糊，这是由
于 SPR 操作中执行的亮度偏移造成的。例如，假设将红色对角线 D（图 5）映射到 BW 像素对
124 上。SPR 操作将会等量地偏移红色能量至邻近的对角线 A、B（映射到 RG 对），由此对角
线 D 会变得模糊。

[0048] 在本发明的某些实施例中，将 SPR 操作修改为，使得邻近的对角线 A 和 B 中的一条
与对角线 A 和 B 中的另一条相比较，从而偏移更多的能量。结果，对角线 D 将会变得更加
锐利。

[0049] 进而言，在传统的 LCD 显示器中，在单的方式来自显示屏，单是显示整个图像 104 所
需要的时间间隔。即使图像不改变，仍对每个帧执行图 4 的数据处理（例如，每秒 60 毫秒
或多帧）。这在许多方面都是低效率的，包括功率消耗、数据处理资源的使用（例如，电路 320
中的微处理器资源）、显示图像中的变化所需的时间等。因此，对于每个新的帧，期望最小化
对未变化的图像部分的处理。特别地，期望避免对未变化的图像部分再做 SPR 处理（方
框 454）。然而，这在图 4 的实施例中是很难实现的，因为即使图像中很小的变化都可能影响方
框 420 生成的 RGBW 坐标的最大值并且因此可能影响方框 430 生成的 BL 值。如果 BL 值发
生变化，则不得不对于整个图像再做缩放和色域钳位操作（444,450）。[0050] 图 6 显示了可选实施例，其中在 SPR 之后执行缩放（444）、色域钳位（450）和确定
BL 值（430）。这里，将 SPR 输出保存在帧缓冲器 610 当中，并在每一帧中，仅对图像中变
化的部分（可在操作 410 前确定变化部分）执行操作 410,420,454。该实施例减少了对未变
化的图像部分的重复处理。然而，色域钳位（450）可能导致如上所述的局部对比度的损
失，该损失不会被 SPR 结合执行的锐化操作所校正。因此，在本发明的某些实施例中，由
方框 450 执行其他类型的锐化，尤其是针对对角线。例如，假设对角线 D（图 5）是由明亮的
饱和色所包围的暗线。明亮的饱和色很可能出现在色域外，这是因为它们的亮度不能完全
地被白色子像素所共享。暗线 D 将可能出现在色域内。传统的色域钳位操作将会降低周边
子像素的亮度以降低和线 D 的对比度，并使得线 D 几乎不可见。在某些实施例中，色域钳位
检测明亮的饱和周边上的暗的对角线并降低暗的对角线的亮度以提高局部对比度。

[0051] 本发明包括以相对低的成本提高图像质量的实施例。更具体的，电路 320 可构造
成细致地分析图像 104 并对任何种类的图像都提供最佳的图像质量。这样的电路也处在本发明的范围之内，但是这样的电路可能是大的和/或复杂的和/或慢的。在某些实施例中，可以简化图像分析来以合理的成本对多数图像提供高的图像质量。

[0052] 除了所附的权利要求所限定的特征和优点之外，本发明并不局限于如上所述的特征和优点。例如，本发明并不局限于图 1 所示的显示器 110、RGBW 显示器，或是对角线相比水平或垂直线承载了较少色度信息的显示器。某些实施例锐化非对角线特征。

[0053] 下面将会针对图 1 和图 3 的显示单元 110 的示例描述本发明的某些实施例。数据处理将被假设为如图 4 或图 6 中一样。

[0054] 转换到 RGBW（步骤 420）。出于说明的目的，假设方框 410 对于每个像素 106 输出线性 RGB 色彩空间下的色彩坐标 r、g、b。r、g、b 坐标中的每一个是允许从 0 到某个最大数 MAXCOL 闭区间间变化的整数。例如，如果 r、g 和 b 表示为 8 比特，则 MAXCOL = 255。在某些实施例中，以更多的比特来保存色彩坐标以避免精度的损失。例如，如果开始时以每个坐标为 8 比特值来表示在非线性 RGB 色彩空间（例如，sRGB）中，则转换到线性 RGB 色彩空间（“伽玛转换”）可能对 r、g 和 b 产生分数数值。为了减小量化误差，将 r、g、b 中的每一个表示为 11 比特，具有 MAXCOL = 2047。

[0055] 颜色 r = g = b = 0 是全黑色，而颜色 r = g = b = MAXCOL 是可能的最明亮的白色。假设 RGBW 是 R、G、B、W 中的每一个都是从 0 至 MAXCOL 闭区间间变化的整数的线性表达。最明亮的 RGB 白色被转换为最明亮的 RGBW 白色，其坐标是 R = G = B = W = MAXCOL。这些假设是非限制性的。对于不同的坐标 (r, g, b, R, G, B, W), MAXCOL 可以是不同的，其他的变化也是可能的。

[0056] 公知的，在这些假设下，可执行如下的转换以满足如下等式：

[0057] \[ r = M_0 \times r + M_1 \times w \] (3)

[0058] \[ g = M_0 \times g + M_1 \times w \]

[0059] \[ b = M_0 \times b + M_1 \times w \]

[0060] 其中，M_0 和 M_1 是如下的对应于像素 120 的亮度特性上的常数:

[0061] \[ M_0 = (Y_w + Y_b + Y_g) / (Y_r + Y_b + Y_g + Y_w) \] (4)

[0062] \[ M_1 = Y_r / (Y_r + Y_b + Y_g) \]

[0063] 其中，Y_r, Y_g, Y_b, Y_w 定义如下。Y_r 是当背光单元 310 在某个参考输出功率（例如，最大功率）下运行，所有的红色子像素 120R 具有最大的透射并且所有的剩余的子像素都具有最小的透射时显示器 110 的亮度。对于绿色、黄色和白色子像素都以相似的方式来定义值 Y_g, Y_b, Y_w。

[0064] 如果 W 坐标是已知的，则可从 (3) 计算 R、G 和 B 坐标。等式 (3) 清楚地要求，如果 r、g 或 b 是零，则 W 必须是零。如果 r = g = b = MAXCOL，则 W = MAXCOL。然而，对于许多颜色，可以多种方式来选择 W（以定义一个或多个条件等色）。为了使得 R、G、B、W 中的每一个都处在 0 到 MAXCOL 的范围内，W 必须处在如下的范围内：

[0065] \[ \min W \leq W \leq \max W \] (5)

[0066] 其中

[0067] \[ \min W = \left[ \max (r, g, b) - M_0 \times \text{MAXCOL} \right] / M_1 \]

[0068] \[ \max W = \min (r, g, b) / M_1 \]
为了以最小的输出功率 BL 提供高的图像质量，每个像素 106 中的 R、G、B 和 W 坐标优选地应该彼此接近。在某些实施例中，将 W 设定为 max(r, g, b)。对于 W，其他选择也是可能的。参见上面提到的美国专利申请 2006/0244686（Higgins 等人）。例如，可以将 W 设定为亮度的表达，例如，下面给出的附录 A（权利要求书之前）中的等式 (A1) 中一样。在如上所述的计算之后，可将 W 值硬钳制 (hard-clamp) 至 minW 到 maxW 之间的范围。如这里所使用的，将数值“硬钳制”至某个数 A 到数 B 之间的范围内是指，如果该数值小于 A，则将该数设定为下限 A；而如果该数大于 B，则将该数设定为上限 B。

等式 (3) 可能要求值 R、G、B 超出 MAXCOL，并与 MAXCOL/M0 一样大。例如，如果 b = 0，则 W = 0；如果 r = g = MAXCOL，则 R = G = MAXCOL/M0。出于说明的目的，假设 M0 = M1 = 1/2，即白色子像素与红色、绿色和蓝色子像素一样明亮。在这种情况下，R、G 和 B 值可与 2*MAXCOL 一样大。显示器 110 仅接受线性 RGBW 坐标不超出 MAXCOL 的颜色。为了显示其他颜色，可使背光单元的功率 BL 乘以 1/M0（即，当 M0 = 1/2 时，则两倍），并使 RGBW 坐标乘以 M0（除以 2）。然而，为了节约电能，某些实施例不增大背光单元的功率或是通过乘以小于 1/M0 的倍数来使增大背光单元的功率。所产生的对比度损失可能与图 7A 所示的一样严重。图 7A 以如图 6 所示的处理的不同阶段对于对角线 D（图 5）、处于 D 之上的邻居对角线 A 和 AA、以及处于 D 之下的邻居对角线 B 和 BB 的示例性最大子像素值。假设对角线 D 是暗的（例如，全黑色），而对角线 A、AA、B、BB 都是明亮的饱和红色（即，坐标 r 靠近 MAXCOL，而 g 和 b 接近 0）。在这种情况下（见图 7A 的部分 1）下，方框 420 会在所有的对角线上将 W 设定为接近 0。在对角线 D 上，值 R、G、B 同样将接近 0。在剩余的对角线上，R 将会接近 2*MAXCOL，而 G 和 B 将会接近 0。

假设将对角线 D 映射到 RG 对。图 7A 的部分 11 显示了 SPR 步骤 454 之后的子像素值。菱形滤波器 (1) 和 (2) 利用权重 1/2 将红色亮度从对角线 A 和 B 映射到对角线 D 上的红色子像素。因此，对角线 D 上的红色子像素的值变得靠近 MAXCOL。对角线 A 和 B 被映射到 BW 对，因此相当暗。对角线 AA 和 BB 保持明亮的饱和红色（红色子像素的值接近 2*MAXCOL）。即使增大背光单元功率（例如，加倍），仍存在对比度损失，因为与部分 1 (SPR 之前) 相比，对角线 D 和 AA、BB、BB 之间的对比度被降低。

进一步，假设不增大背光单元功率，即将其维持为仅为对于不超过 MAXCOL 的像素值而言充分的等级。这样，对角线 AA 和 BB 将会处在色域外。图 7A 的部分 III 显示了色域矩形 450 之后的子像素值。对角线 AA 和 BB 上的最大子像素值被降低到大约 MAXCOL，并且对角线 D 上的最大子像素值也略微降低但仍然维持靠近 MAXCOL。因此，几乎完全地丧失了原始图像中对角线 D 和周围像素之间的高对比度。

Meta Luma 锐化操作加重了对比度损失，因为在对角线 D 上，条件等色将被选择为具有较小的 W 值，以及因此较大的 R 和 G 值，因此可能会提高对角线上的亮度。

在本发明的某些实施例中，在图 6 的步骤 444（缩放）和 450（色域钳位），对于“黑洞”（即，类似于图 7A 的部分 II 中的特征）进行核对。如果检测到黑洞，则与没有检测到黑洞的情形相比较，使得黑洞内部（对角线 D 上）的子像素值降低更多的量。下面将会结合图 9-10 对其更加详细地描述。

如果对角线 D 是被映射到 BW 对的明亮的饱和红色并且周围的像素 106 是暗的，则也可能发生对比度的损失。参见图 7B 的部分 I。SPR 操作将红色亮度从对角线 D 偏移到 A。
和 B。参见图 7B 的部分 II。红色线 D 会变得更宽并且因此可能会模糊。在本发明的某些实施例中，菱形滤波器和 meta luma 锐化在对角线或接近对角线的位置被抑制，并且所有或几乎所有的亮度从 D 移到 A 和 B 中的一个而非两者（例如，在图 7B 的示例部分 II’ 中的对角线 B）。例如，出于上述目的，可使用非对称的箱式滤波器。

图 8 示出了本发明的某些实施例的子像素着色操作 454 的流程图。对于每个像素 106x,y，在步骤 810 中运行测试以确定该像素是否处在饱和的色彩区域内。特别地，在某些实施例中该测试可确定像素 106x,y 或紧左、紧右、紧上或紧下的任意像素是否包含饱和的颜色。如果图是“否”，则在步骤 820 执行常规的处理，例如对像素 106x,y 使用菱形滤波器 (1) 和 (2) 并执行 meta luma 锐化。值得注意的是，可利用相同的滤波器，并将超出该边缘的不存在的像素的坐标设定为一些预定值，例如零处理显示器的边缘处的像素 106。可选地，可利用镜像边缘处的像素来定义超出边缘的不存在的像素 106。例如，如果将左边定义为 x = 0 并且将右边定义为 x = x_{max}，则可将超出左边和右边的不存在的像素定义为 106_{x-1,y} = 106_{x,0} 和 106_{x+1,y} = 106_{x_{max},y}。如果 y 从 0 变化为 y_{max}，则 106_{x,y} = 106_{x,y_{max}}+1 = 106_{x_{max},y}。此外，如果需要（例如，对于 DOG 滤波器），可将不存在的角落像素定义为 106_{x-1,y-1} = 106_{0,0} 并以相似的方式镜像其他三个角落处的像素。通过这里提及的其他滤波操作可执行对边缘和角落的相似处理（利用镜像值或预定值）。

如果回答为“是”，则进行像素 106x,y 是否处在对角线上或接近对角线的核对（步骤 830）。如果回答是“否”，则应用菱形滤波器 (1) 和 (2)（步骤 840）。然而，由于对于饱和的颜色 W 靠近零，因此不执行 meta luma 锐化，这造成条件等色的选择很有限，以至于 meta luma 锐化的影响很小。相反地，例如，利用相同的色彩锐化以其他方式来锐化图像。某些实施例利用 DOG（高斯差分）滤波器执行相同的色彩锐化。如下给出 DOG 滤波器的一种示例性滤波器内核：

$$
\begin{pmatrix}
-1/16 & 0 & -1/16 \\
0 & 1/4 & 0 \\
-1/16 & 0 & -1/16
\end{pmatrix}
$$

(6)

对于相应的色彩平面，像素对 124x,y 的每个子像素 120 应用该滤波器。例如，如果像素对 124x,y 是 RG 对，则通过菱形滤波器 (1) 和 (2) 与 DOG 滤波器 (6) 的输出求和来着色 R 子像素。两种滤波器都在红色平面上操作，即对于方程 420 输出的 R 坐标操作。相似地着色绿色子像素。对于 RW 对的处理是相似的。

在其他的实施例中，在步骤 840 执行 meta luma 锐化 / 或在步骤 820 应用 DOG 滤波器 (6)。在上述两个步骤中还可使用其他类型的锐化。

如果在步骤 830 中回答为“是”，则执行箱式滤波以将像素能量偏移至接近对角线中的一个而非两者。一种示例性滤波器内核如下：

$$
(0, 1/2, 1/2)
$$

(7)

下面的表 1 列出了针对图 6 的 SPR 操作 454 的一个实施例的仿真代码。使用公知的编程语言 Lua 来编写上述仿真代码。该语言类似于 C 语言。这是一种简单的、低成本的实现方式，其无需实现如上所述的所有特征。表 2 列出了该实施例的伪代码。

在表 1 中，“spr.band”是按位与函数（bitwise-AND function），“spr.bor”是按
位或函数，而“spr. bxor”是按位或函数。

[0085] 在该实现方式中，蓝色平面被向左或向右偏移一个像素106。这种相位偏移意味着按照BW对124_{x,y}的蓝色子像素处在邻接RG对124_{x-1,y}或RG对124_{x+1,y}的中心来对其进行处理。例如，在向左偏移的情形下，菱形滤波器1和2计算对124_{x,y}的蓝色子像素值作为像素106_{x-1,y}和四个邻近的像素的B坐标上的加权后的总和。这被认为是对一些图像提供了更真实的色调显示。如果FLIP_LEFT = 0 则偏移的方向向左（见表1中的行Spr5），而如果FLIP_LEFT = 1 则偏移的方向向右。在这部分下面的具体实施方式中，出于简化假设蓝色偏移方向向左。权利要求书不局限于向左偏移，除非是特别指出。

[0086] 在上述实现方式中，步骤830核对通过如下所示的3×3矩阵定义的图案(pattern)：

\[
D1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},
\]

\[
D4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad D5 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D6 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},
\]

\[
D7 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad D8 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad D9 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},
\]

\[
D10 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad D11 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D12 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},
\]

\[
D13 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad D14 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D15 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},
\]

[0092] 对于每个像素106_{x,y}，可对各像素的R、G和B坐标，以及还可能的对W坐标独立地核对上述图案D1至D15中的每一个。在某些实施例中，如果将像素映射到RG对，则对R、G和B坐标执行核对。利用某个阈值“BOBits”对各坐标R、G、B、W进行“阈值化” (“thresholded”)。见表1中的第F22-F26行。在某些实施例中，MAXCOL = 2047且BOBits介于128至192闭区间之间，例如256。例如，利用rth, gth, hth和wth分别表示红色、绿色、蓝色和白色坐标的阈值。如果R > BOBits，则将阈值“rth”设定为1，否则将rth设定为0。可以相同的方式获得G、B和W坐标的阈值gth, hth, wth。随后对每个坐标的阈值使用滤波器D1至D15。例如，对于任意的i和j，使rth_{i,j}表示像素106_{i,j}的rth阈值。如果下面的条件 (T1) 和 (T2) 中的一个为“真”则对于像素106_{x,y}而言滤波器D7的输出是1（即，在红色平面上识别D7图象）：

[0093] (T1): rth_{x,y} = rth_{x-1,y-1} = rth_{x-1,y+1} = 1

[0094] rth_{x-1,y-1} = rth_{x+1,y-1} = rth_{x,y-1} = rth_{x,y+1} = rth_{x+1,y+1} = 0
说 明 书

[T2]: rth_{x, y} = rth_{x+1, y-1} = rth_{x-1, y+1} = 0 \text{ 和}

[T96]: rth_{x+1, y-1} = rth_{x-1, y+1} = rth_{x+1, y+1} = rth_{x-1, y-1} = 1

[T97]: 否则，滤波器输出是 0，即在红色平面中无法识别 D7 图案。

[T98]: 图案 D1–D5 对应于单个的点。在点图案中会发生对比度的损失，因此要像对对角线那样处理这些图案。图案 D9–D11 表示像素 106x, y, 接近对角线。图案 D12–D15 表示像素位于对角线的端部。

[T99]: 在该实现方式中，利用如下的滤波器执行步骤 810:

$$
Ortho = \begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0
\end{bmatrix}
$$

[T101]: 利用 OR 操作对饱和阈值平面应用该滤波器。更特别地，对于每个像素 106x, y, 计算标记 (flag) “sat”，如果饱和度高，则标记 “sat” 等于 1，否则标记 “sat” 等于 0。下面说明可行的 “sat” 计算。一旦计算出 sat 值，即对像素 106x, y, 应用 Ortho 滤波器。如果对于像素及其四个邻近（向上、向下、向左、向右）的像素而言 sat = 0，则滤波器输出 “orthos” 为零。否则, ortho = 1。在某些实施例中，如果四个对角邻近的像素 (即, 106x+1, y+1, 106x-1, y+1, 106x+1, y-1, 106x-1, y-1) 中的两个像素中存在饱和的像素 (sat = 1)，则同样将 ortho 设定为 1。见表 1 中的行 Spr23–Spr30 和 Spr73–Spr80；表 2 中的行 Ps2, Ps9, Ps10。

[T102]: 可如下计算 sat 值。在某些实施例中，对于每个像素 106x, y, 如果如下的值“sinv”（饱和度倒数）高于某个阈值，则将 sat 设定为 0:

$$
sinv = \text{floor}[\min(r, g, b)/\max(1, r, g, b)]
$$

[T103]: 其中，r, g, b 是输入 rgb 坐标。在其他实施例中，通过 max(r, g, b) 的高位比特 (upper bits) (例如，四个高位比特) 形成的数字乘以某个 “饱和度阈值” “STH” (例如，0, 1, 2 或更大)，并且取该乘积的四个高位有效位 (mostsignificant bits)。如果它们形成了大于 min(r, g, b) 的数值，则将 sat 设定为 1, 否则设定为 0。


[T105]: 在表 1 中，在行 Spr6 中计算 “orthos”。此外，对于 BW 对，计算 “orthos” 作为对左侧的邻近像素的 Ortho 滤波器输出，并将其用于确定蓝色子像素值 (行 Spr59, Spr89–Spr91)。

[T106]: 在步骤 810，如果 Ortho 滤波器的输出 “orthos” 为零，则回答为 “是”，否则，回答为 “否”。见表 1 中的行 Spr34 (对于 RG 对) 和 Spr108 (对于 BW 对)。在处理蓝色子像素的过程中，以相似的方式使用 “orthos” (行 Spr96)。

[T107]: 如果将像素 106x, y 映射到 RG 对，则在表 1 中的行 Spr9–Spr53 和表 2 中的行 PS1–PS7 中描述像素的处理。可同时处理右侧的邻近的蓝色子像素。更特别地，如果 ortho 是 0 (表 1 中的行 Spr34, 表 2 中的行 PS3)，则将 R, G 和 B 子像素值 (Ry, Gy, By) 设定为对角
线滤波器 (2) 的输出加上在权利要求书前面的附录 A 中描述的 meta luma 锐化值“ $a$ ”。见图 8 中的步骤 820。在表 1 的实施例中，可简化 meta luma 锐化为: 不对 meta luma 锐化操作的 RGBW 输出（附录 A 中的等式 (2)）应用菱形滤波器，而是当 RGBW 坐标经历 meta luma 锐化操作之前对其应用菱形滤波器，并对菱形滤波器的输出追加 meta luma 锐化值“ $a$ ”。这样做会加快 SPR 并降低存储需求（通过消除 meta luma 滤波器的 RGBW 输出所需的长期存储器）。

【0109】在表 1 的行 Spr39，表 2 的行 PS5 中，当且仅当像素 106 $x_{n}$ 的 R 和 G 坐标的至少一个当中识别出图案 D1-D15 中的至少一个时，值设为 1。在这种情形下，执行步骤 850。特别地，将 R 和 G 子像素值设定为箱式滤波器 (7) 的输出。

【0110】如果 diag 不是 1，则执行步骤 840（表 1 的行 Spr44-Spr45，表 2 的行 PS6）。将 R 和 G 子像素值设定为对角线滤波器 (2) 和 DOG 滤波器 (6) 的输出的总和。

【0111】在表 1 的行 Spr47，表 2 的行 PS7 中，当且仅当在像素 106 $x_{n}$ 的 B 坐标中识别出图案 D1-D15 中的至少一个时，值设为 1。在这种情形下（表 1 的行 Spr48，表 2 的行 PS7）下，在步骤 850 中，将 B 子像素值设定为箱式滤波器 (7) 的输出。

【0112】如果 bdiag 不是 1，则在步骤 840（行 Spr51-PS7）中，将 B 子像素值设定为对角线滤波器 (2) 和 DOG 滤波器 (6) 的输出的总和。

【0113】如果像素 106 $x_{n}$ 转移到 BW 对，则按照表 1 中的起始行 Spr54，表 2 中的行 PS8 所示对其进行处理。在这种情形下，如上所述地对左侧的（即, 带有蓝色偏移的）邻近像素计算蓝色子像素值。因此，蓝色子像素处理有些重复（尽管不完全是这样的），在某些实施例中可将其省略。可选地，（对于 RG 对）省略了行 Spr9-Spr53 中的蓝色子像素处理。在表 1 的仿真代码中，执行蓝色子像素处理两次，并在存储器保存（行 Spr162）蓝色子像素的两次结果。随后的处理可使用上述两次结果中的任一个。

【0114】如上所述的方式确定标记“ortho”和“bortho”。

【0115】在表 1 的行 Spr96，表 2 的行 PS11 中，如果对于左侧的邻近像素而言 Ortho 滤波器输出 bortho 是 0，则将 B 子像素值设定为菱形滤波器 (2) 的输出和 meta 锐化滤波器值 $a$（附录 A）的总和。对于像素 106 $x_{n}$，计算上述两个滤波器。见行 Spr97。此外，如表 1 的行 Spr120-Spr141，表 2 的行 PS19 所示，如果像素 106 $x_{n}$，靠近屏幕的左侧或右侧边缘，则将标记“doedge”设定为 1 以执行特殊的处理。如果在屏幕边缘处图像包含垂直白线，则执行该处理以提高色调。更特别地，如果如表 1 所示保持特定的条件，则将蓝色和白色子像素中的每一个计算为菱形滤波器 (2) 和 DOG 滤波器 (6) 的总和。见行 Spr137-Spr138。对于 106 $x_{n}$，计算该滤波器。

【0116】如果 bortho 不是 0，则对于蓝色平面（表 1 的行 Spr70、Spr100，表 2 的行 PS13）核对 diag（步骤 830）。如果 diag 是 1，则应用箱式滤波器 (7) （行 Spr101）（步骤 850）。对于像素 106 $x_{n}$，计算箱式滤波器以输出像素 106 $x_{n}$ 和 106 $x_{n}$ 的 B 坐标的平均值。因此，如果像素 106 $x_{n}$，的 bortho 是 1，像素 106 $x_{n}$，的 ortho 是 1，而像素 106 $x_{n}$，和 106 $x_{n}$，的 diag 是 1，则应用箱式滤波器以使对应的子像素 120R, 120G, 120B 的每一个的值是像素 106 $x_{n}$，和 106 $x_{n}$，对应的子像素 R, G, B 的平均值。在某些实施例中，对应的子像素 120W 的值也是像素 106 $x_{n}$，和 106 $x_{n}$，的 W 坐标的平均值。然而，在表 1 和表 2 中，如下所述，以不同的方式来计算 W 子像素值。
如果在行 Spr101 和 PS13 中 diag 不是 1, 则将 B 子像素值计算为都应用于像素 106_{x,l}, 的菱形滤波器 (2) 和 DOG 滤波器 (6) 的输出的总和 (步骤 840, 行 Spr103 和 PS14)。在表 1 中, 如果在此处讨论中假设蓝色偏移是向左, 则将变量 blueshift 设定为 1, 或蓝色偏移是向右, 则设定为 -1。) 此外, 还将 doedge 设定为 1 以执行如上所述的对于边缘像素的边缘处理。

从行 Spr108, PS15 开始展示了如何计算 W 值。如果对于像素 106_{x,y} 而言 Ortho 滤波器输出 ortho 是 0, 则将 W 子像素值设定为菱形滤波器 (2) 的输出和 meta 锐化滤波器值 a_{x,y} (即, 值 a) (附录 A) 的总和。对于像素 106_{x,y}, 计算这两个滤波器。见行 Spr109。

如果 ortho 不是零, 则对于白色平面核对 diag( 行 Spr111, Spr112, PS17) (步骤 830)。如果 diag 是 1, 则应用箱式滤波器 (7) (行 Spr113) (步骤 850)。对于像素 106_{x,y}, 计算箱式滤波器以输出像素 106_{x,y} 和像素 106_{x+l,y} 的 W 值的平均值。

如果 diag 不是 1, 则将 W 子像素值计算为菱形滤波器 (2) 和 DOG 滤波器 (6) 的输出的总和 (步骤 840, 行 Spr115, PS18)。对于白色平面中的像素 106_{x,y}, 使用这两个滤波器。

对于所有的像素, 即对映射到 RG 对的像素和对映射到 BW 对的像素执行从行 Spr143, PS19 开始的处理。在行 Spr147-155 中, 红色、绿色和蓝色子像素的子像素值被硬锁位至 0 至 MAX00 的最大范围, 其中 MAX00 = 2*MAXCOL+1 是当 MO = 1/2 时可能的最大 RGBW 值 (参见等式 (3))。白色子像素值被硬锁位至 0 至 MAXCOL 的范围。

在行 Spr126-Spr134 和其他一些段落中，值 HS 和 VS 表示在仅更新屏幕的一部分时的起始的水平和垂直坐标。表 1 的仿真代码假设 HS = VS = 0。此外, 变量 xsiz 和 ysz 包含正在被更新的屏幕部分的宽度和高度。

<table>
<thead>
<tr>
<th>表 1-SPR, LUX 代码</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1:</td>
</tr>
<tr>
<td>D2:</td>
</tr>
<tr>
<td>D3:</td>
</tr>
<tr>
<td>D4:</td>
</tr>
<tr>
<td>D5:</td>
</tr>
<tr>
<td>D6:</td>
</tr>
<tr>
<td>D7:</td>
</tr>
<tr>
<td>D8:</td>
</tr>
<tr>
<td>D9:</td>
</tr>
<tr>
<td>D10:</td>
</tr>
<tr>
<td>D11:</td>
</tr>
<tr>
<td>D12:</td>
</tr>
<tr>
<td>D13:</td>
</tr>
<tr>
<td>D14:</td>
</tr>
<tr>
<td>D15:</td>
</tr>
<tr>
<td>D16:</td>
</tr>
<tr>
<td>D17:</td>
</tr>
<tr>
<td>D18:</td>
</tr>
</tbody>
</table>
F1: function dplane(x, y, plane) -- 核对对角线和点
[0143] F2: if BOBtest(x, y, {[0144] F3: 0, 0, 0,  -- 以及点
[0145] F4: 0, 1, 0,
[0146] F5: 0, 0, 0}, plane) == 1 then return 1
[0147] F6: elseif BOBtest(x, y, {[0148] F7: 0, 1, 0,
[0149] F8: 0, 0, 0,
[0150] F9: 0, 0, 0}, plane) == 1 then return 1
[0151] F10: elseif --... 参见表 1 后面给出的备注 2
[0152] F11: end
[0153] F12: return 0
[0154] F13: end -- 函数 dplane
[0155] F14:
[0156] F15: ---------------------------------------------
[0157] F16: -- 用于计算二进制阈值的独立通道
[0158] F17: -- (在硬件的 SPR 模块中完成)
[0159] F18: spr.create(" bin", xsiz, ysiz, 4, 1)
[0160] F19: if DEBUG_IMAGE == 1 then spr.create(" BIN", xsiz, ysiz, 3, 1) end
[0161] F20: spr.loop(xsiz, ysiz, 1, 1, function(x, y))
[0162] F21: local r, g, b, w = spr.fetch(pipeline, x, y) -- 取 GMA 后的数据
[0163] F22: if r <= BOBits then r = 0 else r = 1 end -- 将各平面阈值化为单个比特
[0164] F23: if g <= BOBits then g = 0 else g = 1 end
[0165] F24: if b <= BOBits then b = 0 else b = 1 end
[0166] F25: if w <= BOBits then w = 0 else w = 1 end
[0167] F26: spr.store(" bin", x, y, r, g, b, w) -- 构建二进制阈值化的图像
[0168] F27: if DEBUG_IMAGE == 1 then
[0169] spr.store(" BIN", x, y, b*127+w*127, g*127+w*127, r*127+w*127)  end--
[0170] DIAGNOSTIC: 制作可视化版本用于最后查看
[0171] F28: ---------------------------------------------
[0172] F29: -- 用于计算饱和度阈值的独立通道
[0173] F30: spr.create(" sinv", xsiz, ysiz, 1, 2) -- SPR 的饱和度比特图像
[0174] F31: if DEBUG_IMAGE == 1 then spr.create(" SINV", xsiz, ysiz, 3, 1) end
[0175] -- 诊断图像
[0176] F32: spr.loop(xsiz, ysiz, 1, 1, function(x, y))
[0177] F33: local sat = 0  -- 假设饱和值
[0178] F34: local R, G, Bw, Ww, Lw, Ow = spr.fetch(" gma", x, y) -- 取 GMA 后的值
[0179] F35: Lw = math.floor((Rw*2+Gw*5+Bw+WW*8)/16) -- 重新计算亮度
[0180] F36: spr.store(" gma", x, y, Rw, Gw, Bw, Ww, Lw, Ow) 且将其写出
F37: SATBITS = SATBITS or 2048  -- 在饱和度计算中的2^个比特
F38: local R = math.floor(SATBITS*Rw/(MAXCOL+1))  -- 使它们向右偏移12个比特
F39: local G = math.floor(SATBITS*Gw/(MAXCOL+1))
F40: local B = math.floor(SATBITS*Bw/(MAXCOL+1))
F41: local W = math.floor(SATBITS*Ww/(MAXCOL+1))
F42: if(math.floor(STH*max(R,G,B)/16)) > W then
F43: sat = 1
F44: end
F45: spr.store(" sinv", x, y, sat)  -- 将它保存用于 SPR 模块
F46: ifDEBUG_IMAGE == 1 then
F47: sat = sat*255  -- 转换为白色像素
F48: spr.store(" SINV", x, y, sat, sat, sat)  -- 对于诊断图像而言
F49: end
F50: end)
F51:  -- 滤波器
F52: diamond =  -- 常规的对角线滤波器
F53: {
F54: xsize = 3, ysize = 3,
F55: 0,32,0,
F56: 32,128,32,
F57: 0,32,0
F58: }
F59: metasharp =  -- 条件等色锐化滤波器
F60: {
F61: xsize = 3, ysize = 3,
F62: 0,-32,0,
F63: -32,128,-32,
F64: 0,-32,0
F65: }
F66: selfsharp =  -- 自我锐化滤波器
F67: {
F68:  -- xsize = 3, ysize = 3,
F69:  -- -32,0,-32,
F70:  -- 0,128,0,
F71:  -- -32,0,-32,
F72: --
F73: fullsharp =  -- 完全锐化滤波器
F74: {

21
[0219] F75:xsize = 3, ysize = 3,
[0220] F76: -16,0,-16,
[0221] F77: 0,64,0,
[0222] F78: -16,0,-16
[0223] F79:]
[0224] F80:
[0225] F81:xfullsharp =  -- 完全锐化滤波器,两倍
[0226] F82:]
[0227] F83:xsize = 3, ysize = 3,
[0228] F84: -32,0,-32,
[0229] F85: 0,128,0,
[0230] F86: -32,0,-32,
[0231] F87:]
[0232] F88:
[0233] F89:ortho =  -- 检测任何正交标记存在的滤波器
[0234] F90:]
[0235] F91:xsize = 3, ysize = 3,
[0236] F92: 0,1,0,
[0237] F93: 1,1,1,
[0238] F94: 0,1,0
[0239] F95:]
[0240] F96:
[0241] F97:boxflt =  -- 用于对角线的箱式滤波器
[0242] F98:]
[0243] F99:xsize = 3, ysize = 1,
[0244] F100: 0,128,128
[0245] F101:]
[0246] F102:ltcorner =  -- 检测中心标记存在的滤波器
[0247] F103:]
[0248] F104:xsize = 3, ysize = 3,
[0249] F105: 1,0,0,
[0250] F106: 0,0,0,
[0251] F107: 0,0,0
[0252] F108:]
[0253] F109: lbcornet =  -- 用于检测中心标记存在的滤波器
[0254] F110:]
[0255] F111:xsize = 3, ysize = 3,
[0256] F112: 0,0,0,
[0257] F113: 0,0,0,
F114:1,0,0
F115:
F116: Rcorner = -- 用于检测中心标记存在的滤波器
F117:
F118:xsize = 3, ysize = 3,
F119:0,0,1,
F120:0,0,0,
F121:0,0,0
F122:
F123: Rbcorner = -- 用于检测中心标记存在的滤波器
F124:
F125:xsize = 3, ysize = 3,
F126:0,0,0,
F127:0,0,0,
F128:0,0,1
F129:
Spr1:********************************
Spr2: function dospr(x,y) -- 用于执行 SPR 滤波的程序
Spr3: local lft,rgt,ext -- 在 SPR 期间的值
Spr4: local R,G,B,W,L = 0,1,2,3,4 -- 对 GMA 缓冲器中的位置给出名称
Spr5: local evenodd =
Spr6: local ortho = spr.sample("sinv", x,y,0,Ortho)-- 当没有 sat 比特时为 0
Spr7:
Spr8: if evenodd = = 0 then --RG 逻辑像素
Spr9: local meta = spr.sample(pipeline,x,y,L,metasharp) -- meta 对于 R 和 G 是相同的
Spr10: local redss = spr.sample(pipeline,x,y,R,fullsharp)
Spr11: local grnss = spr.sample(pipeline,x,y,G,fullsharp)
Spr12: local redbx = spr.sample(pipeline,x,y,R,boxflt)
Spr13: local grnbx = spr.sample(pipeline,x,y,G,boxflt)
Spr14: local bluss = spr.sample(pipeline,x,y,B,fullsharp) -- 蓝色自我锐化结果
Spr15: local blubx = spr.sample(pipeline,x,y,B,boxflt)
Spr16: local blueshift = 1-2*FLIP_LEFT-- 左翻转使蓝色偏移的方向变反 (flip left reverses direction of blue shift)
Spr17: lft = spr.sample(pipeline,x,y,R,diamond) -- 红色子像素
[0292] Spr18:rgt = spr.sample(pipeline, x, y, G, diamond) -- 绿色子像素
[0293] Spr19:ext = spr.sample(pipeline, x, y, B, diamond) -- 蓝色子像素
[0294] Spr20:
[0295] Spr21:if ortho_mod == 1 then
[0296] Spr22:--ortho 超驰 (override)
[0297] Spr23:local ltc = local ltcorner = spr.sample(”sinv”, x, y, 0, Ltc) -- 如果在左上角邻位中没有 sat 比特, 则为 0
[0298] Spr24:local bcorner = spr.sample(”sinv”, x, y, 0, Lbc) -- 如果在左下角邻位中没有 sat 比特, 则为 0
[0299] Spr25:local rtc = local rtcorner = spr.sample(”sinv”, x, y, 0, Rtc) -- 如果在右上角邻位中没有 sat 比特, 则为 0
[0300] Spr26:local rbc = local rbcorner = spr.sample(”sinv”, x, y, 0, Rbc) -- 如果在右下角邻位中没有 sat 比特, 则为 0
[0301] Spr27:
[0302] Spr28:if(rtc == 1 and bcorner == 1)or(rtc == 1 and rbc == 1)or
[0303] Spr29:if(ltc == 1 and rtcorner == 1)or(lbc == 1 and rbcorner == 1)then
[0304] Spr30:ortho = 1 --ortho 超驰
[0305] Spr31:end
[0306] Spr32:end
[0307] Spr33:
[0308] Spr34:if ortho == 0 then -- 如果附近没有饱和颜色
[0309] Spr35:ift = lft+meta -- 则使用 meta luma 滤波
[0310] Spr36:rgt = rgt+meta
[0311] Spr37:ext = ext+meta
[0312] Spr38:else
[0313] Spr39:local diag = spr.bor(dplane(x, y, R), dplane(x, y, G)) -- 或一起测试红色和绿色
[0314] Spr40:ifdiag == 1 then -- 如果在饱和区域中和靠近对角线
[0315] Spr41:ift = redbx -- 使用箱式滤波器
[0316] Spr42:rgt = grnbx
[0317] Spr43:else -- 否则使用自我颜色锐化
[0318] Spr44:ift = lft+redss
[0319] Spr45:rgt = rgt+grnss
[0320] Spr46:end
[0321] Spr47:local bdiag = dplane(x, y, B)
[0322] Spr48:ifbdiag == 1 then -- 因为和原代码兼容, 故独立测试蓝色
[0323] Spr49:ext = blubx
Spr50 : else -- 否则使用自我颜色锐化
Spr51 : ext = ext + bluss
Spr52 : end
Spr53 : end -- 结束 MIX_BOB 对角线的材料 (stuff)
Spr54 : else --BW 逻辑像素
Spr55 :-- 蓝色子像素
Spr56 : local blueshift = 1 - 2 * FLIP_LEFT -- 左翻转使蓝色偏移的方向变反
Spr57 : local bluss = spr.sample (pipeline, x - blueshift, y, B, fullsharp) -- 蓝色自我锐化结果
Spr58 : local blums = spr.sample (pipeline, x - blueshift, y, L, metasharp)
Spr59 :-- 蓝色 meta 锐化结果
Spr60 : local bortho = spr.sample (" sinv", x - blueshift, y, 0, Ortho)
Spr61 :-- 白色子像素
Spr62 : local whtss = spr.sample (pipeline, x, y, W, fullsharp) -- 白色自我锐化
Spr63 : local whtms = spr.sample (pipeline, x, y, L, metasharp) -- 白色 meta 锐化
Spr64 : local whtbx = spr.sample (pipeline, x, y, W, boxflt)
Spr65 : local doedge = 0 -- 如果 == 1, 则边缘处理是必要的
Spr66 : lif = spr.sample (pipeline, x - blueshift, y, B, diamond)
Spr67 : rgt = spr.sample (pipeline, x, y, W, diamond)
Spr68 : ext = 0
Spr69 :-- 蓝色
Spr70 : local diag = dplane (x - blueshift, y, B) -- 计算最后一秒的蓝色对角线
Spr71 : ifortho_mod = = 1 then
Spr72 :-- ortho 超驰
Spr73 : local ltcorn = spr.sample (" sinv", x, y, 0, Ltcorn) -- 如果在左上角没有 sat 比特, 则为 0
Spr74 : local bcorner = spr.sample (" sinv", x, y, 0, Lbcorn) -- 如果在左下角没有 sat 比特, 则为 0
Spr75 : local rtcorn = spr.sample (" sinv", x, y, 0, Rtcorn) -- 如果在右上角没有 sat 比特, 则为 0
Spr76 : local rbcorner = spr.sample (" sinv", x, y, 0, Rbcorn) -- 如果在右下角没有 sat 比特, 则为 0
Spr77 :
Spr78: if(ltcorner = = 1 and lbcorner = = 1)or(rtcorner = = 1
andrbcorner = = 1)or

Spr79: if(ltcorner = = 1 and rtcorner = = 1)or(lbcorner = = 1 and rbcorner
= = 1)then

Spr80 : ortho = 1      -- ortho 超驰

Spr81 : end

Spr82 :

Spr83 : -- bortho 超驰

Spr84 : local lbcorner = = spr.sample(" sinv ", x-blueshift, y, 0,
Ltcorner) -- 如果在蓝色偏移后的左上角没有 sat 比特, 则为 0

Spr85 : local lbcorner = = spr.sample(" sinv ", x-blueshift, y, 0,
Lbcorner) -- 如果在蓝色偏移后的左下角没有 sat 比特, 则为 0

Spr86 : local rbcorner = = spr.sample(" sinv ", x-blueshift, y, 0,
Rtcorner) -- 如果在蓝色偏移后的右上角没有 sat 比特, 则为 0

Spr87 : local rbcorner = = spr.sample(" sinv ", x-blueshift, y, 0,
Rbcorner) -- 如果在蓝色偏移后的右下角没有 sat 比特, 则为 0

Spr88 :

Spr89 : if(ltcorner = = 1 and lbcorner = = 1)or(rtcorner = = 1
andrbcorner = = 1)or

Spr90 : if(ltcorner = = 1 and rtcorner = = 1)or(lbcorner = = 1
andrbcorner = = 1)then

Spr91 : bortho = 1      -- bortho 超越

Spr92 : end

Spr93 : end

Spr94 :

Spr95 : -- 蓝色子像素使用不同的偏移量

Spr96 : ifbortho = = 0 then      -- 如果附近没有饱和的像素

Spr97 : lft = lft+blums -- 使用 meta-luma 锐化

Spr98 : doedge = = 1

Spr99 : else      -- 如果靠近饱和的像素

Spr100 : ifdiag = = 1 then      -- 新方式对蓝色作出处理

Spr101 : lft = blubx

Spr102 : else

Spr103 : lft = lft+bluss-- 使用自我锐化

Spr104 : doedge = = 1

Spr105 : end

Spr106 : end

Spr107 : -- 白色子像素

Spr108 : ifortho = = 0 then      -- 如果附近没有饱和的像素
Spr109: rgt = rgt+whtms -- 使用meta-luma 锐化
Spr110: else  -- 如果靠近饱和的像素
Spr111: local diag = dplane(x,y,W)
Spr112: if diag == 1 then  -- 并靠近对角线
Spr113: rgt = whtbx  -- 则使用箱式滤波器
Spr114: else
Spr115: rgt = rgt+whtss  -- 否则使用自我锐化
Spr116: end
Spr117: end
Spr118:---------------------------------------------
Spr119:
Spr120: if EDGE == 1 and doedge == 1 then  -- 对混合的饱和度执行边缘处理
Spr121: local r2, g2, blue_sh = spr.fetch(pipeline, x-blueshift, y)
Spr122: local r3, g3, blue_nosh = spr.fetch(pipeline, x, y)
Spr123: local edgelogic = false
Spr124: if NSE == 0 then  -- 以仅对屏幕的边缘执行边缘处理开始
Spr125: edgelogic =
Spr126: (((x+HS) == 1) and (FLIP_LEFT == 0) and (blue_sh >= blue_nosh)) or
Spr127: (((x+HS) == 0) and (FLIP_LEFT == 1) and (blue_sh <= blue_nosh)) or
Spr128: (((x+HS) ==(fxsiz-1)) and (FLIP_LEFT == 0) and (blue_nosh >= blue_sh)) or
Spr129: (((x+HS) ==(fxsiz-2)) and (FLIP_LEFT == 1) and (blue_nosh <= blue_sh))
Spr130: elseif NSE == 1 then  -- 对屏的边缘右侧执行边缘处理
Spr131: edgelogic =
Spr132: (((x+HS) ==(fxsiz-1)) and (FLIP_LEFT == 0) and (blue_nosh >= blue_sh)) or
Spr133: (((x+HS) ==(fxsiz-2)) and (FLIP_LEFT == 1) and (blue_nosh <= blue_sh))
Spr134:
Spr135: end
Spr136: if edgelogic then
Spr137:  lft = spr.sample(pipeline, x, y, B, diamond) + spr.sample(pipeline, x, y, B, fullsharp)
Spr138:  rgt = spr.sample(pipeline, x, y, W, diamond) + spr.sample(pipeline, x, y, W, fullsharp)
Spr139: end
Spr140 :end   -- 边缘处理
Spr141 :end -- BW 逻辑像素
Spr142 :
Spr143: lft = math.floor((lft+128)/256) -- 滤波器系数实际值乘以 256
Spr144: rgt = math.floor((rgt+128)/256)
Spr145: ext = math.floor((ext+128)/256)
Spr146 :
Spr147: lft = math.max(0, lft) -- 锐化滤波器可能造成上溢或下溢
Spr148: rgt = math.max(0, rgt) -- 不得不将它钳位至最大范围内
Spr149: ext = math.max(0, ext)
Spr150: lft = math.min(MAXOOG, lft)
Spr151: rgt = math.min(MAXOOG, rgt)
Spr152: ext = math.min(MAXOOG, ext)
Spr153 :
Spr154: if evenodd == 1 then      -- 如果这是 BW 对，
Spr155: rgt = math.min(rgt, MAXCOL) -- 必须将白色限制在 11 个比特
Spr156 :end
Spr157 :
Spr158: ifFLIP_LEFT == 1 then
Spr159: lft, rgt = rgt, lft -- 这是在 Lua 中使用！交换这两个值！
Spr160 :end
Spr161 :
Spr162: spr. store(frameB, x, y, lft, rgt, ext)
Spr163: end -- 函数 dospr
Spr164: 结束表 1
Spr165: 对于表 1 中的代码的备注：
Spr166: 备注 1: blackjack 类型测试的 brute force 软件实现方式；需要使用在行 D10、F18.F26 中的以 bin 命名的独立的帧缓冲器，对每个像素将 RGBW 坐标阈值化为 0 或 1；如果为图案匹配或图案匹配的反转，则返回 1；硬件利用 9 比特的比特图案测试实现这。
Spr167: 备注 2: 对于所有的图案 D1-D15 执行该测试。省略了对于剩余图案的代码。
Spr168: 表 2-SPR 的伪代码
Spr169: PS1. RG pair:
Spr170: PS2. If saturated bit in diagonal corners, then oftho = 1.
Spr171: PS3. If ortho = 0, then Rw, Gw = diamond+meta,
Spr172: Ext = diamond+meta
Spr173: PS4. If ortho = 1 then
Spr174: PS5. If diag in R and G planes, then
Spr175: Rw, Gw = box filter
Spr176: PS6. Else Rw, Gw = diamond plus DOG.
PS7. If dbdiag(diag in B plane), then
PS8. BW pair:
PS9. If sat bit in diagonal corners, then ortho = 1.
PS10. If sat bit in blue-shifted diagonal corners, then bortho = 1.
PS11. If bortho = 0, then Bw = diamond+meta with blue shift,
doedge = 1
PS12. Else:
PS13. If diagonal in B plane with blue shift then
Bw = box(with blue shift)
PS15. If ortho = 0, then Ww = diamond+meta
PS16. Else:
PS17. If diagonal in W plane then
Ww = box
PS18. Else Ww = diamond+DOG
PS19. If edge processing conditions hold, then
Bw, Ww = diamond+DOG without blue shift
PS20. End of BW pair.
PS21. Clamping

缩放和色域控制

如上所述，在图 6 的步骤 444（缩放）和 450（色域控制）中，某些实施例核对“黑洞”（即，类似于图 7A 的部分 II 中的特征），并执行对黑洞内部（“在对角线 D 上”）的像素值的额外缩减。这将有助于恢复局部对比度。

黑洞的存在依赖于背光单元的输出功率 BL。更特别地，假设输入 rgb 数据定义了当背光单元生成某个输出功率 BL = B_{0} 时的图像。如从等式 (3) 所看到的，SPR 方框 454 产生的 R, G 和 B 子像素值处在 0 至 \( \text{MAXCOL}/M_{0} \) 闭区间的范围内。W 值 \( W_{0} \) 可以达到 \( \text{MAXCOL}/M_{0} \)，但是典型地将其选择为不超过 \( \text{max}(r, g, b) \)，并因此不超过 \( \text{MAXCOL} \)。因此，W 不超过 \( \text{MAXCOL}/M_{0} \)。SPR 方框 454 产生的 \( R_{w}G_{w}B_{w}W_{w} \) 值定义了当背光单元的输出功率是 \( B_{0} \) 时所期望的子像素亮度。然而，为了提供输入到显示器 110 的值，子像素值必须不超过 \( \text{MAXCOL} \)。如果将子像素值乘以 \( M_{0} \) 以适于 0 至 MAXCOL 闭区间的范围，则背光单元的输出功率 \( B_{0} \) 需要除以 \( M_{0} \)，即设定为

\[
B_{L} = B_{0}/M_{0}.
\]

实际上，如果最大的子像素值 \( P_{\text{max}} = \text{max}(R_{w}, G_{w}, B_{w}, W_{w}) \) 小于 \( \text{MAXCOL}/M_{0} \)，则较小的 BL 值可能是足够的。更特别地，给定最大值 \( P_{\text{max}} \)，为了不变形地显示所有的子像素所必须满足的最小 BL 值 \( B_{L_{\text{min}}} \) 是

\[
B_{L_{\text{min}}} = B_{0} * P_{\text{max}} / \text{MAXCOL}.
\]
可能期望将输出功率 BL 设定为小于 BLmin 的值。在任何情况下，将输出功率 BL 表示为 BL0 的百分比时是方便的，即

\[ \text{BL} = \frac{1}{1+\text{INVy}} \times \text{BL0} \]

其中 INVy 是用于（在缩放 444 中）乘以对应于 BL0 的因子以对应于 BL 的系数。例如，如果 BL = BLmin，则 INVy = MAXCOL/Pmax。如果 BL = BL0，则 INVy = 1。

如果 BL 小于 BLmin（即 INVy > MAXCOL/Pmax），则某些子像素值大于 MAXCOL，因此可能需要缩放/色域钳位。下面在附录 B 中说明了方程 3 40 中用于确定 BL 的一些方法。

图 9 显示出了 444,450（缩放/色域钳位）的示例流程图。图 9 显示了对于由在一行中两个邻近对 124x, 和 124x+, 构成的四方联（quad）1010（图 10）的处理。其中一对是 RG，而另一对是 BW。

下面更加详细地说明图 9。简要地说，在步骤 940 计算可相乘的增益因子 XSC_gain 作为 0 至 1 闭区间之间的值，并在步骤 950 将四方联 1010 中的 RwGwBwWw 子像素值乘以上述增益因子以使得在不改变色调和饱和度的前提下将色彩带入色域内。增益 XSC_gain 是“常规”增益 XS_gain 和“黑洞”增益 blk_gain 的乘积。见步骤 940。常规增益 XS_gain 依赖于 BL，从而不超过 INVy（以便实现缩放 444）。因而四方联 1010 处在黑洞中（如在步骤 910 所对应的），则黑洞增益 blk_gain 可小于 1。否则，将黑洞增益设定为 1。

现在，假设四方联 1010 对应于同一行中对角线 D, B 上两个邻近像素（图 5 和图 1A）。则四方联 1020 对应于对角线 A,AA，而四方联 1030 对应于对角线 BB 和右侧的下一个对角线。图 1A 中的像素 11 的四方联 1010 中的最大子像素值将会处于黑洞内。因此，blk_gain 很可能小于 1，从而 XSC_gain 将会减少 blk_gain。

当处理对角线 AA,AA 上的像素时（即，当四方联 1010 对应于对角线 AA,AA 上的两个像素时），blk_gain 将会是 1，因为对角线 AA,AA 上的像素不在黑洞中。然而，在上述的某些实施例中，常规的增益 XS_gain 是两个像素的最大 rgb 坐标的位置函数（见式（3）。因此，对角线 AA,AA 的 XS_gain 可能小于对角线 D,B 的 XS_gain。这将会导致在不使用黑洞增益时对真值的损失。对于对角线 D,B 将黑洞增益设定为小于 1 的值起到了减小这两个对角线的子像素值的作用以恢复对比度损失。

下面的表 3 显示出了用于仿真图 9 所示的方法的流程“dopost”的仿真代码。以 LUA 写出该仿真代码。该处理使用了稍后被除以 256 的增益因子 XS_gain、blk_gain 的整数算法（定点算法）。图 9 的方法对于每个四方联执行一次。因此，在图 9 的方法的每次迭代中 x 增加 2，而 y 增加 1。在实际的实现方式中，行地或是以其他某种顺序来处理所有的四方联。

图 10 显示了四方联 1010 的左下象限的子像素四方联 1020 和四方联 1010 的右下象限 1030。表 3 中的实施例可简化成；当核对黑洞时（图 9 的步骤 910），该实施例仅核对在邻近的四方联 1020,1030 中的色域外的色彩。该实施例不核对四方联 1010 的上方和下方的四方联。这是一种较简单的实现方式，其允许降低电路 320 的成本。其他的实施例可核对上方和/或下方的四方联。

在表 3 的行 Sc46-Sc61 中实现步骤 910。以 Rw, Gw, Bw, Ww 表示四方联 1010 的初始（预先钳位的）子像素值。步骤 910 的测试如下；如果在四方联 1010 中 max(Rw,Gw,Bw, Ww) 不超出 MAXCOL 并且四方联 1020,1030 的每一个中的最大子像素值超过 MAXCOL，则检测
到黑洞。也可以使用其他测试。例如，黑洞可包括额外需求，即，四方联 1020、1030 的每一个的最大子像素值与 MAXCOL 相比超于某个因子（例如，至少 1.1*MAXCOL）和/或与四方联 1010 中的最大子像素值相比超于某个因子。进一步，该测试可以核对四方联 1020、1030 的亮度大于四方联 1010 的亮度或大于某个值，或核对四方联 1010 的亮度小于某个值。还可使用其他测试。

[0495] 值得注意的是，在这个实施例中，测试不依赖于 INVy。因此，即便 INVy = M0，仍可检测出黑洞并将 blk_gain 设定为小于 1 的值。如通过比较图 7A 的部分 I 和部分 II 所看到的，尽管 INVy = M0，仍可对对角线 D 降低局部对比度，并且将 blk_gain 设定为小于 1 的值有助于恢复局部对比度。在其他的实施例中，测试依赖于 INVy，例如可通过将子像素值和 INVy 的乘积与 MAXCOL 相比较来执行该测试。

[0496] 如果测试失败（即，没有检测到任何黑洞），则将 blk_gain 设定为 1（图 9 中的步骤 914；表 3 中的行 Sc4）。值得注意的是，行 Sc4 中的值 256 对应于 1，因为黑色增益将会随后除以 256。

[0497] 如果测试通过，则如下将 blk_gain 计算作为表 3 的行 Sc62～Sc64 中的 8 比特值（见图 9 的步骤 920）。

[0498] blk_gain = 2*MAXCOL-1-(四方联 1020, 1030 中的最大子像素值）(9)

[0499] 在这个实施例中，MAXCOL = 2047，且 M0 = M1 = 1/2。在行 Sc61 中，GAMBITS = 11。可选地，还可使用如下的等式：

[0500] blk_gain = ceiling[1/M0*MAXCOL]-1-(四方联 1020, 1030 中的最大子像素值）

[0501] 随后（行 Sc65），使 blk_gain 增加 Ww/16。如果 Ww 值是大的（即，黑洞实际上是白洞），则通过上述操作来增大黑洞增益。随后将 blk_gain 硬件值设为最大值 256（即，在行 Sc11 中除以 256 之后为 1）。


[0503] 在表 3 的特定实施例中，XS_gain 依赖于饱和度和由等式 (3) 定义的 r、g、b 值的最大值。更特别地，如表 2 的行 Sc91 所示，将 XS_gain 计算作为基于饱和度的增益 sat_gain 和值“n1_off”的总和。该总和被硬钳位至从方框 340 接收到的 INVy 的最大值。

[0504] 在行 Sc72～Sc84 中确定值 sat_gain 作为某些预定参数 GMIN 和 GMAX 闭区间之间的值。在某些实施例中，GMAX = 1（即在除以 256 之前为 256）且 GMIN = 1/2。值 sat_gain 是饱和度的函数，并更特别地，以如下方式定义饱和度倒数 sinv:

[0505] sinv = Ww/max(1, Rw, Gw, Bw)

[0506] 见行 Sc74～Sc83。如果饱和度最多是某个的预定的阈值（例如 50％），即如果 sinv 小于某阈值，则将 sat_gain 设定为大约 GMAX。在行 Sc84 中，由 REG_SLOPE 定义该阈值（REG_SLOPE 是对应于 1 的整数值）。如果 sinv 是零，则将 sat_gain 设定为大约 GMIN。如果 sinv 处于零至其阈值之间，则按照线性插值函数获得 sat_gain，其中 sat_gain 在 sinv
＝ 0 时等于大约 GMIN，而在 sinv 处于该阀值时等于大约 GMAX。此外，将 sat_gain 硬钳位
至最大值 1(在行 Sc85 中为 256)。
[0507] 基于如等式 (3) 中的 r, g, b 的 max(r, g, b)，在行 Sc87–Sc90 中计算项 n1_off(“非线性偏移量”)。等式 (3) 表示 max(r, g, b) = M_0 * max(R, G, B) + M_1 * W。出于简化的目的，在
表 3 中假设 RGBW 值是子像素值 Rw, Gw, Bw, Ww。按照线性增益函数计算值 n1_off, n1_off
在 max(r, g, b) = MAXCOL 时等于 0，并且在 max(r, g, b) = 0 时等于大约 N * INVy，其中 N 是
0 至 256 闭区间之间的预定参数。
[0508] 如上所述，XS_gain 是被硬钳位至 INVy 的 sat_gain 和 n1_gain 的总和。然后进一步调节值 XS_gain 以确保在乘以 XS_gain 之后，子像素值 Rw, Gw, Bw, Ww 不超过 MAXCOL(行
Sc97–Sc109)。
[0509] 在行 Sc111 执行步骤 940。
[0510] 在步骤 950, Rw, Gw, Bw, Ww 值乘以 XSC_gain(行 Sc115–Sc119)。
[0511] 随后，在行 Sc122–Sc128，进一步调节 Ww 值以使 dopost 处理不会改变四方联 1010
的亮度。更特别地，可如下在缩放和色域钳位之间和之后计算亮度 Lw:
[0512] Lw = (2^5*Rw+5*Gw+B2+8*Ww)/16(见行 Sc44, Sc119)。
[0513] 可以调节 Ww 值以使得缩放后和缩放前的亮度相一致。
[0514] 最终，将值 Rw, Gw, Bw, Ww 硬钳位至 0 至 MAXCOL 闭区间间的范围(行 Sc129–Sc137)。
[0515] 表 3—缩放和色域钳位
[0516] Sc1: local Rw, Gw, Bw, Ww—静态变量，用于进行对 dopost 的连续调用
[0517] Sc2:——********dopost 进行饱和度-缩放-变量-缩放和色域钳位
[0518] Sc3: function dopost(x, y)
[0519] Sc4: local blk_gain = 256 ——通过计算黑洞增益开始
[0520] Sc5: local scale_clamp = 0 ——表示钳位已完成的标记
[0521] Sc6 : rd, gd, bd = 0, 0, 0 ——用于诊断图像
[0522] Sc7 :
[0523] Sc8: if y == 78 and x == 25 then
[0524] Sc9: glob = 1
[0525] Sc10 : end
[0526] Sc11: ——在 4 个组中进行后缩放,如此始终读取 2 个逻辑像素
[0527] Sc12:
[0528] Sc13: local evenodd =
[0529] spr. bxor(spr. band(x, l), spr. band(y, l), FLIP_UP, FLIP_LEFT)—棋盘位置
[0530] Sc14: if FLIP_LEFT == 0 then ——如果 SID == 0 or 2
[0531] Sc15: if evenodd == 0 then
[0532] Sc16 : Rw, Gw = spr. fetch(pipeline, x, y)—取帧缓冲后的值
[0533] Sc17: if x == xsiz-1 then ——如果这是行中的最后一个 RG
[0534] Sc18: Bw, Ww = 0, 0 ——永远不会达到 BW, 再运行一个时钟
[0535] Sc19: else
[0536] Sc20 : return ——否则等待 BW 到来
[0537] Sc21 : end
[0538] Sc22 : else
[0539] Sc23 : if Ww, Ww = spr. fetch (pipeline, x, y) — 取帧缓冲后的值
[0540] Sc24 : if x == 0 then — 如果这是行中的第一个 BW
[0541] Sc25 : if Rw, Gw = 0, 0 — 没有伴随这一个的 RG，则将它们设为零
[0542] Sc26 : end — 并至少处理该数据
[0543] Sc27 : end
[0544] Sc28 : else — 否则 SID = 1 or 3
[0545] Sc29 : if evenodd = 0 then
[0546] Sc30 : Gw, Rw = spr. fetch (pipeline, x, y)
[0547] Sc31 : if x == 0 then
[0548] Sc32 : if Ww, Bw = 0, 0 — 对于第一 GR, 强迫 WB 为零
[0549] Sc33 : end
[0550] Sc34 : else
[0551] Sc35 : if Ww, Bw = spr. fetch (pipeline, x, y)
[0552] Sc36 : if x = xsiz - 1 then — 对于最后的 WB,
[0553] Sc37 : if Rw, Gw = 0, 0 — 不再存在 GR, 再运行一个时钟
[0554] Sc38 : else
[0555] Sc39 : return — 不是最后一个，等待 GR 到来
[0556] Sc40 : end
[0557] Sc41 : end
[0558] Sc42 : end
[0559] Sc43 : — 需要从 SPR 后的数据中近似亮度和饱和度
[0560] Sc44 : local Lw = math. floor ((2*Rw+5*Gw+Bw+8*Ww)/16)
[0561] Sc45 :
[0562] Sc46 : if BEE == 1 then — 黑线加强
[0563] Sc47 : if DEBUG_IMAGE then
[0564] Sc48 : spr. store (" BE Ef, x, y, 0, 0, 128)
[0565] Sc49 : spr. store (" BEE", x - 1, y, 0, 0, 128)
[0566] Sc50 : end
[0567] Sc51 : local r, g = spr. fetch (pipeline, x-3, y) — 取左侧的 RGBW
[0568] Sc52 : local b, w = spr. fetch (pipeline, x-2, y)
[0569] Sc53 : local rgbw1 = spr. bor (r, g, b, w) — 只进行高位比特的或操作
[0570] Sc54 : local oog = math. max (r, g, b, w)
[0571] Sc55 : r, g = spr. fetch (pipeline, x+1, y) — 右侧的 RGBW
[0572] Sc56 : b, w = spr. fetch (pipeline, x+2, y)
[0573] Sc57 : local rgbw3 = spr. bor (r, g, b, w)
[0574] Sc58 : oog = math. max (oog, r, g, b, w)
[0575] Sc59 : local rgbw2 = spr. bor (Rw, Gw, Bw, Ww)
Sc60: if (rgbw2 <= MAXCOL) and (Ww <= (MAXCOL+1)/16) and -- 如果中心是在区域内并是饱和的 (忽略白洞)
Sc61: (rgbw1 > MAXCOL) and (rgbw3 > MAXCOL) then -- 被 OOG 包围
Sc62: oog = math.floor(spr.band(oog, MAXCOL)/(2^5*(GAMBITST-7))) -- 丢弃 OOG 比特并保存接下的 7 个比特
Sc63: oog = (127-oog)+128 -- 取逆并设定比特 8
Sc64: blk_gain = oog -- 降低增益值使该像素变暗 Sc65: blk_gain = math.min(256, (blk_gain+math.floor(Ww/16))) -- "白洞忽略" 中的特征
Sc66: if DEBUG_IMAGE then
Sc67: spr.store(" BEE", x, y, blk_gain, blk_gain, blk_gain)
Sc68: spr.store(" BEE", x-1, y, blk_gain, blk_gain, blk_gain)
Sc69: end
Sc70: end
Sc71: end -- 结束黑洞检测器
Sc72: -- 执行饱和度 - 缩放增益计算
Sc73: local gmin = GMIN+1 -- 默认为固定的 GMIN
Sc74: local max_rgb = math.floor((math.floor(M0_reg/256*math.max(Rw, Gw, Bw)*2)+math.floor(M1_reg/256*Ww*2))/2)
Sc75: -- 12 比特的项 +11 比特的项将会得到 13 比特的结果, 随后将除以 2 以得到 12 比特的结果
Sc76: -- 随后整位至 MAXCOL 以得到 11 比特的结果 (防止从交叉影响(cross-pollinated) 的像素对溢出)
Sc77: max_rgb = math.min(MAXCOL, max_rgb)
Sc78: max_rgb = math.max(1, maxrgb) -- 防止除以 0
Sc79: local inv_max_rgb_lut = math.floor((plus4bit/max_rgb)+0.5)
Sc80: local min_rgb = math.floor((math.floor(M0_reg/256*\nmath.min(Rw, Gw, Bw)*2)+math.floor(M1_reg/256*Ww*2))/2)
Sc81: -- 12 比特的项加上 11 比特的项将会得到 13 比特的结果, 随后除以 2 以得到 12 比特的结果
Sc82: min_rgb = math.min(MAXCOL, min_rgb) -- 随后整位至 MAXCOL 以得到 11 比特的结果 (防止从交叉影响的像素对溢出)
Sc83: local s1n = math.floor(inv_max_rgb_lut*min_rgb)
Sc84: local sat_gain = math.floor(REG_SLOPE*s1n/plus4bit+gmin)
Sc85: sat_gain = math.min(256, sat_gain, GMAX+1)
Sc86: Sc87: -- 计算转换到 RwGwBwWw 空间的非线性增益项
Sc88: local nl_index_11bits = max_rgb
Sc89:
Sc90: local nl_off = math.floor((N*16+16)*INVy/256*(MAXCOL-
Sc90)11bits)/(MAXCOL+1))
Sc91: local nl_gain = math.min(INVy, sat_gain+nl_off)
Sc92: gd = OutGamma((256-sat_gain)*MAXCOL*2/256)--诊断代码用于将饱和增益显示成绿色
Sc93:
Sc94: XS_gain = nl_gain --将其保存用于钳位增益计算
Sc95:
Sc96:-- 始终计算色域钳位增益并在其他算法留下颜色 OOG 时使用
Sc97: local maxp = math.max(Rw,Gw,B w,WW) --发现最大的基色
Sc98: maxp = math.floor(maxp*XS_gain/256) --预测 OOG 在 sat 和 X/XL 之后多远
Sc99: local clamp_gain = 256 --默认为 1.0, 无钳位
Sc100: if maxp > MAXCOL then --如果该颜色进行 OOG
Sc101: local Ow = spr.band(maxp,MAXCOL) --计算在 LUT 指数中使用的距离 OOG
Sc102: clamp_gain = math.floor((256*(MAXCOL+1))/(maxp+1)) --为伽马钳位的 INV LUT 的结果
Sc103: rd = OutGamma((256-clamp_gain)*MAXCOL*2/256)
Sc104: if clamp_gain < 256 then
Sc105: scale_clamp = 1 --如果仍需要增益,则设定标记比特
Sc106: end
Sc107: end-- 色域外的颜色
Sc108:
Sc109: XS_gain = math.floor(XS_gain*clamp_gain/256) --组合 X/XL 和 sat, 并钳位至一个常数
Sc110:
Sc111: XS_gain = math.floor(XSgain*blk_gain/256)--并和黑洞增益进行组合
Sc112:
Sc113:--INVy X/XL 缩放值可大于 1.0, 所以现在缩放值是 9 比特
Sc114:-- 具有大于二进制点而小于 8 的一个比特
Sc115: Rw = math.floor((Rw*XS_gain+128)/256) --12*9 = 12 比特乘法
Sc116: Gw = math.floor((Gw*XS_gain+128)/256) --(仅需要 12*9 = 11, 但是必须测试
Sc117: Bw = math.floor((Bw*XS_gain+128)/256) --溢出并硬钳位至小于 MAXCOL)
Sc118: Ww = math.floor((Ww*XS_gain+128)/256)-- 钳位至 W 的黑色值
Sc119: Lw = math.floor((Lw*XS_gain+128)/256)--单独对 L 执行 X/XL 处理
Sc120:
Sc121:------------------------
Sc122;-- 键位诊断选项
Sc123; if CLE == 1 and scale clamp == 1 then
Sc124; local W1 -- 计算产生正确的亮度的 W
Sc125; W1 = math.floor((Lw*M1_inv-
math.floor((2*Rw+5*Gw+Bw)*M2_inv/8))/32)
Sc126; W1 = math.min(W1, MAXCOL) -- 不超出最大值!
Sc127; Ww = math.floor((W1*(2^((DIAG+4)+(Ww*(128-
(2^((DIAG+4))))/128)) -- 将两者混合在一起
Sc128; end -- 键位 diag
Sc129; Rw = math.min(Rw, MAXCOL) -- 硬键位
Sc130; Gw = math.max(Gw, 0) -- (如果 WR > 1.0, 则发生)
Sc131; Bw = math.min(Bw, MAXCOL) -- 来自 LUT 中的量化误差
Sc132; Ww = math.min(Ww, MAXCOL)
Sc133; Lw = math.min(Lw, MAXCOL)
Sc134; Rw = math.max(Rw, 0) -- 在 MIPI 指令中是负数 (-1)
Sc135; Gw = math.max(Gw, 0)
Sc136; Bw = math.max(Bw, 0)
Sc137; Ww = math.max(Ww, 0)
Sc138; Lw = math.max(Lw, 0)
Sc139;:
Sc140; Ww = math.floor(Ww*(WG+1)/256) -- 这里, 白色增益可降低白色
Sc141;:
Sc142; sptr.store(" post", x+odd, y, Rw, Gw) -- 将它们保存在后缓冲器中
Sc143; sptr.store(" post", x-odd+1, y, Bw, Ww)
Sc144; if FLIP_LEFT == 0 then
Sc145; if evenodd == 0 then
Sc146; sptr.store(" post", x, y, Rw, Gw) -- 就保存 RG
Sc147; else
Sc148; if x > 0 then
Sc149; sptr.store(" post", x-1, y, Rw, Gw) -- 当存在 RG 值时保存 RG 值
Sc150; end
Sc151; sptr.store(" post", x, y, Bw, Ww) -- 且每次 BW
Sc152; end
Sc153; else --SID = 1 or 3
Sc154; if evenodd == 1 then -- 正常情形偶数对失败 (fall through)
Sc155; sptr.store(" post", x, y, Ww, Bw) -- 所以这必须是 x == xsiz-1
Sc156; else
[0677]  Sc157: if x > 0 then
[0678]  Sc158: spr. store(" post", x-1, y, Ww, Bw)  — 当存在 WB 值时保存 WB 值
[0679]  Sc159: end
[0680]  Sc160: spr. store(" post", x, y, Gw, Rw)  — 总是写入 GR
[0681]  Sc161: end
[0682]  Sc162: end
[0683]  Sc163: end— 函数 dopost
[0684]  结束表 3
[0685]  位块图像传送 (Bit Blit) 更新如参考图 6 所说明的一样，在某些实施例中，显示装置可仅接收像素数据 104 的一部分 1110(图 11)，因为图像的其他部分未变化。显示装置执行“位块图像传送”操作以更新屏幕上图像的已变化的部分。不对整个图像执行 SPR 操作 454。对整个图像执行其他操作，诸如 444（放大）、430（BL 计算）、450（色域偏移）和其他可能的操作。位块图像传送更新可降低功率消耗并还可降低在短时间内更新图像所需的处理功率。此外，位块图像传送更新方便用于通过低带宽网络链路接收图像 104 的移动系统。因此，某些实施例适用于 MIPI（移动工业处理器接口）。然而，本发明并不局限于 MIPI 或移动系统。

[0686]  为了易于描述，假设新的部分 1110 是矩形的。然而，本发明并不局限于矩形部分。
[0687]  在其他一些实施例中，对于整个图像重复执行 SPR 操作。更特别地，显示装置对于图像 104 的每个像素保存输入数据 (rgb 或 RGBW)。并且在接收部分 1110 时在 SPR 操作 454 中对整个图像重新计算像素值。在图 4 或图 6 中可实现该重新计算。然而，期望的是，至少对于图像中未变化的部分中的某些像素不重复执行 SPR。
[0688]  现在说明一些实施例，这些实施例都是基于结合图 8 和表 1 所述的 SPR 操作的，但是本发明并不局限于这些实施例。
[0689]  在图 11 中，新的部分 1110 包括边缘 1110E。边缘是一个像素的宽。未变化的图像部分包括边界 (border) 区域 1120，该边界区域 1120 由在新的部分 1110 边界的像素 106 构成。区域 1120 也是一个像素宽。当对边缘像素 1110E 执行 SPR 操作 454 时，SPR 操作涉及像素 1120。然而，某些实施例并不保持先前图像的 rgb 或 RGBW 数据。因此这样的数据对像素 1120 是不可用的。边缘像素 1110E 的处理因此引起了特殊的挑战，特别是在 (由新的部分 1110 定义的) 新的图像类似于先前图像时。如果图像是相似的，则观察者更可能注意到新的部分 1110 和周边之间的边缘。然而，本发明并不局限于相似的图像。

[0690]  在某些实施例中，当对像素 1110E 执行 SPR 操作 454 时，使用像素 1110E 的镜像图像替代像素 1120。例如，假设对于某个 x0, xl, y0, y1, 将区域 1110 定义为 x0 ≤ x ≤ x1 和 y0 ≤ y ≤ y1。随后如下定义边界像素 1120 用于对像素 1110E 的 SPR 操作：

[0691]  $106_{x_0-1,y_0+1} = 106_{x_0,y_0}$

[0692]  角落像素也被镜像：$106_{x_0-1,y_0-1} = 106_{x_0,y_0}$

[0693]  如果 SPR 使用蓝色偏移，则存在进一步的挑战。将会详细介绍指出左偏移的实例。右偏移的实施例是类似的。

[0694]  在左偏移的情况下，如果将部分 1110 左侧的边缘区域 1110E 中的像素 106 映射到
BW对，则可能必须对边界区域1120 中的邻近像素应用 SPR 滤波器。在图12 的示例中，在新的部分1110 的左侧的各区域1120,1110E 中，像素106.1,106.2 是同一行中的邻近像素。像素106.1 被映射到RG 对124.1，而像素106.2 被映射到BW 对124.2。在表1 的实施例中，当着色对124.2 的蓝色子像素对时，对像素106.1 应用菱形滤波器(2) 和meta luma滤波器。当以新的部分1110 更新图像时像素106.1 不改变，而像素106.2 仅对这两个滤波器贡献很小的权重（例如，对菱形滤波器为 1/8 的权重）。因此，在某些实施例中，SPR 操作使得蓝色子像素的值相对于子像素对124.2 中的先锋图像不变化。更特别地，SPR 操作不改变被映射到BW 对并处在图像左侧的边缘像素1110E 的蓝色值（BW）。当然，可通过诸如缩小444 和色域钳位450 等后续操作改变BW 值。）在右偏移的情况下，SPR 操作不改变图像右侧的蓝色值。

[0695] 在某些实施例中，如果新的部分1110 是一列宽（由此与边缘区域1110E一致），则不改变对应于新的部分1110 的所有BW 值。

[0696] 在左偏移的情况下，当边界像素1120 被映射到BW 对时，在右侧边缘存在另外的挑战。这在图13 中示出。邻近像素106.3,106.4 处在新的部分1110 的右侧的各区域1110E, 1120 中。像素106.3 被映射到RG 对124.3，而像素106.4 被映射到BW 对124.4。由于蓝色偏移，可能必须通过对像素106.3 应用SPR 滤波器来着色对124.4 中的蓝色像素。由于通过新的部分1110 更新像素106.3，因此应该更新与对124.4 中的蓝色子像素对应的帧缓冲器的位置。然而，所期望的是，避免写入与未变化的图像部分对应的帧缓冲器的位置，并通常期望减少对帧缓冲器610 的写入访问的次数。某些实施例通过打乱（scramble）帧缓冲器610 中的子像素值来实现上述目标的，以使蓝色子像素位置仅保存低位有效位（least significant bit）。高位有效位是在对应于RG 对的存储位置处保存的。因此，如果不对应于蓝色子像素（诸如对124.2 中的蓝色子像素）的存储位置更新，则只会造成低位有效位的失真。

[0697] 图14示出了打乱技术的一个示例。显示器110 的子像素被细分为四方联（“quad”）1404。每个四方联1404 在同一行中包括两个邻近的对124x,y,124x+1,y。在每个四方联1404 中，左边的对124x,y 是RG 对，而右边的对124x+1,y 是 BW 对。显示器左侧边缘处的BW 对和右侧边缘处的RG 对不是任何四方联的一部分，并可以如下方式被处理。

[0698] 对于每个四方联1404，SPR 操作545 提供在1410 所示的子像素值 Rw,Gw,Bw, Ww。在图14 中，各值 Rw,Gw,Bw,Ww 的高位有效位部分（MSB 部分）分别被表示为 RH, GH, BH, WH。低位有效位部分（LSB 部分）分别被表示为 RL, GL, BL, WL。例如，在某些实施例中，各值 Rw,Gw,Bw, Ww 都是8 比特的值，并且MSB 和 LSB 部分每个都是四比特。


[0700] 如上所示，如果利用被映射到位于BW 对124x+1,y 的紧左侧的子像素的新的部分1110 更新图像则可能使存储位置610B 的内容丢失（不被更新）。因此，在每个四方联中，
存储位置 610B 仅保存值 Rw，Gw，Bw，Ww 中的一部分或全部的低位有效位。在图 14 的实施例中，各四联的存储位置 610B 仅保存四联的 RL 和 BL 值。由于一些实验证结果对于红色和蓝色亮度不如绿色和白色亮度敏感，因此选择红色和蓝色值。“红色”位置 610R 保存红色和蓝色亮度的高有效位部分 RH，BHR。在各位置 610G，610W 无需打乱地保存绿色和白色值 Gw，Ww，其他类型的打乱也是可能的。

[0701] 打乱在写入帧缓冲器 610 时执行。当读取帧缓冲器 610 时（例如，通过图 6 中的缩放 444 或方框 430），数据被解打乱。

[0702] 对于屏幕的左侧的每个 BW 对（即，每个 BW 对 124），可以预定值，例如 0 来填充蓝色位置 610B 的 MSB 部分。可丢弃 BH 值。关于解打乱，可将 BH 值设定为零。本发明并不局限于此或是处理边缘处的 BW 对的其他方式。

[0703] 对于屏幕右侧的每个 RG 对，在打乱过程中，通过对对应于 RG 对的像素 106 应用 SPR 操作中的合适的滤波器，来获得 Bw 值。可将 Bw 值的 BH 部分写入位置 610R 的 LSB 部分。可丢弃 BL 和 RL 部分。关于解打乱，可将 RL 设定为零或某个其他值。

[0704] 本发明并不局限于上述的实施例。其他实施例和变化都在如所附的权利要求书所限定的发明的范围内。

[0705] 例如，某些实施例提供了通过显示单元显示图像的方法。显示单元（例如，图 3 的单元 110）可以是液晶显示器（LCD）、有机发光显示器（OLED）、或是其他类型的显示器。值得注意的是，本发明并不局限于使用背光单元的显示器。例如，图 8 中的 SPR 操作可以不依赖于背光单元。

[0706] 显示单元包括子像素，每个子像素用于发射多种基色中的一种基色并具有基于子像素状态的亮度。基色可以是 RGBW 或是其他一些颜色。子像素可以象或不象图 1 所示那样布局。例如，在某些实施例中，在每个 RG 对中，绿色像素处于红色像素的左侧；在每个 BW 对中，白色像素处于左侧。子像素在面积上可以相等或不等。例如，一种基色的子像素大于另一种基色的子像素。不同基色的子像素可在数目和/或密度上不同。在 LCD 中，子像素的状态通过液晶分子的子像素排列定义，子像素排列依次由子像素电压定义。在 OLED 中，子像素状态通过子像素电流或其他电参数定义。使用子像素的子像素值，基于显示器的类型定义该状态。

[0707] 方法包括：接收图像信号，各图像信号与图像（例如 104）或图像中的新的部分（例如 1110）相关联，各图像信号包括相关联的图像或图像的新的部分中的各子像素的像素数据。该方法进一步包括，对于每个所述图像信号，由电路（例如，电路 320，SPR 方框 454）执行相关的 SPR 操作。该 SPR 操作使得各像素相关联于作为显示单元的要显示该像素的区域的显示区域（例如 124），SPR 操作对在与相关联的图像或新的部分的一个或者多个像素相关联的一个或者多个显示区域内的由一个或者多个子像素构成的子像素组中的每个子像素提供子像素值。例如，SPR 操作可以与新的部分 1110（图 11）对应的显示区域内的除了左侧边缘的蓝色子像素外的每个子像素提供子像素值。进一步，至少一个显示区域不包含至少一种基色的整个子像素。例如，区域 124 可以是 BW 区域，由此没有红色的子像素。

[0708] 进一步，至少一个图像信号与新的部分相关联，并且相关联的 SPR 操作对位于不与新的部分中任何像素关联的区域内的至少一个子像素不提供子像素值。例如，在结合图 11 所述的某些实施例中，SPR 操作不提供新的部分 1110 和边界区域 1120 外部的任何子像
素值。

[0709]  在某些实施例中，由至少一个所述 SPR 操作产生的至少一个子像素值处在显示单元的色域外。该方法包括利用色域内的值替代该至少一个子像素值（例如，通过色域阈位 450）。

[0710]  某些实施例提供了通过显示单元显示图像的方法，其中显示单元包括子像素，每一个子像素发射多种基色中的一种基色并具有基于利用该子像素的子像素值定义的子像素状态的亮度。该方法包括：接收图像信号，每个图像信号与图像或图像中的新的部分相关联，每个图像信号包括相关联的图像或图像中的新的部分的各像素的像素数据。该方法进一步包括，对于每个所述图像信号，由电路执行相关联的 SPR 操作。该 SPR 操作使得各像素关联于显示单元的要显示该像素的区域的显示区域，SPR 操作对在与相关联的图像或新的部分的一个或者多个像素相关联的显示区域内的由一个或多个子像素构成的子像素组中的每个子像素提供子像素值。此外，至少一个图像信号与新的部分相关联，相关联的 SPR 操作对位于与新的部分中任何像素关联的区域内的至少一个子像素不提供子像素值。进一步，对于与包括子像素 SP1（例如，蓝色子像素）的新的部分 P1（例如部分 1110）相关联的至少一个图像信号 S1 而言，其中该子像素 SP1 位于与新的部分 P1 的预定侧（例如，图 11 中的左侧）在新的部分 P1 的边缘处的像素相关联的显示区域内，相关联的 SPR 操作不提供子像素 SP1 的子像素值使得子像素 SP1 的子像素值不变化。例如，在图 11 的一些实施例中，SPR 操作不提供在与新的部分 1110 的左边缘处的特定像素 1061, 的相关联的 BW 对 1241 的蓝色子像素的子像素值。

[0711] 进一步，对于至少一个其他的图像信号 S2，相关联的 SPR 操作提供子像素 SP1（例如，对于像素 1061, 不处在左侧边缘的另一图像中的相同的 BW 对 1241, 中的蓝色子像素）的子像素值，其中子像素 SP1 处于与第一像素（例如 1061,）相关联的显示区域内，其中图像信号 S2 关联于包括第一像素的图像或是关联于包括第一像素的新的部分 P2，且相关联的 SPR 操作确定子像素 SP1 的子像素值作为数个像素的色坐标的加权和（例如，步骤 820，840 或 850），其中在加权和中，对第一像素（例如 1061,）给出了在大小上不大于在第一像素的所述预定侧（例如左侧）的第二像素的权重（例如，由于左侧蓝色偏移，在步骤 820 或 840 中对蓝色子像素使用的 1/8 的权重或者在步骤 850 中使用 1/2 的权重），例如，在左侧蓝色偏移的情形下，在大小上对像素 1061, 给出不小于像素 1061,1, 的权重。

[0712] 在某些实施例中，基色包括作为子像素 SP1 的颜色的所述颜色 PC1（例如，蓝色）。进一步，对于与在所述预定侧（例如，左侧）的边缘处包括一个或更多的像素的新的部分相关联的每个图像信号，其中，该一个或更多的子像素的关联显示区域包含颜色 PC1 的一个或多个子像素，相关联的 SPR 操作对于具有颜色 PC1（例如，任何蓝色子像素）并位于与在新的部分中位于所述预定侧（例如左侧）的边缘处的像素相关联的显示区域中的任何子像素，不提供子像素值。

[0713] 某些实施例提供了通过显示单元显示图像的方法，其中该显示单元包括子像素，每个子像素发射多种基色中的一种基色并具有基于利用该子像素的子像素值定义的子像素状态的亮度。该方法包括：接收图像信号，每个图像信号与图像或图像中的新的部分相关联，每个图像信号包括该相关联的图像或图像中的新的部分的各像素的像素数据。该方法进一步包括，对于每个所述图像信号，由电路执行相关联的 SPR 操作，该 SPR 操作使得各
像素关联于作为显示单元的要表示该像素的区域的显示区域，SPR 操作对在与相关联的图像或新的部分的一个或者多个像素相关联的显示区域内的由一个或多个子像素构成的子像素组中的每个子像素提供子像素值。此外，至少一个图像信号与新的部分相关联，相关联的 SPR 操作对位于不与新的部分中任何像素关联的区域内的至少一个子像素不提供子像素值。进一步，在至少一个 SPR 操作中，对于至少一个基色 PC1（例如，蓝色），对于基色 PC1 的至少一个子像素 SP1，将子像素值确定为多个像素的色坐标的加权总和，其中，在该加权总和中，对于其相关联的显示区域包括子像素 SP1 的第一像素，在大小上给出不小于在第一像素的预定侧（例如左侧）的第二像素的权重。进一步，各图像的每个子像素值保存在存储器的各自的比特中。例如，在图 14 的实施例中，存储器可以是帧缓冲器 610。绿色和白色子像素值保存在各自的位置 610G 和 610W 的比特中。红色子像素值可以保存在位置 610R 和 610B 的比特中。蓝色子像素值可以保存在相同的位置的其他比特中。

进一步，各子像素值包括高位有效位部分（最显著部分）和低位有效位部分（最小有效部分）。对于与新的部分相关联的至少一个图像信号，对于在新的部分的与所述预定侧相对的一侧处于新的部分的外部并与包含基色 PC1（例如蓝色）的第一子像素的显示区域相关联的至少一个像素（例如，位于新的部分 1110 右侧的像素 106），相关联的 SPR 操作确定第一子像素的子像素值的至少高位有效位部分，并将高位有效位部分（BI），而非第一子像素的子像素值的低位有效位部分（BL），保存在各自的位（例如，与位于左侧的红色像素相关联的 610R）。

在某些实施例中，不靠近显示单元的屏幕的边缘的子像素被细分为组（例如，1404），每个组包括所有基色的子像素。在每个组中，在存储器的连续比特中保存包含基色 PC1 的不同基色（例如，红色和蓝色）的至少两个子像素的子像素值的高位有效位部分（例如，RH 和 BI）；并且在存储器的连续比特中保存包含基色 PC1 的不同基色的至少两个子像素的子像素值的低位有效位部分（例如，RL 和 BL）。

在某些实施例中，存储器的连续比特中保存基色 PC1（例如，蓝色）和另一基色 PC2（例如，红色）的两个子像素的子像素值的高位有效位部分（例如，RH 和 BI）；并且在存储器的连续比特中保存基色 PC1 和 PC2 的两个子像素的子像素值的低位有效位部分（例如，RL 和 BL）。

在某些实施例中，在每一组中，在存储器的非连续比特中保存至少一个子像素的子像素值的高低有效位部分和低位有效位部分（例如，RH 和 RL）。

提供了电路用于执行这里所述的方法。如果需要还可执行其他操作（例如，伽马转换和图像显示）。本发明由所附的权利要求书限定。

附录 A：meta luma 锐化

在某些实施例中，按照如下方式执行对像素 106x.y 的 meta luma 锐化。根据等式 (3) 确定像素的 RGBW 坐标。此外，以某一方式，例如如下的方式计算代表像素 106x.y 和邻近像素的亮度的值 L：

$$ L = \frac{(2R + 5G + B + 8W)}{16} \quad \text{(A1)} $$

随后，如果将像素 106x,y 映射到 BW 对，则对亮度 L 应用如下的滤波器以产生值 a:
其中，$z$ 是某个正的常数，例如 $1/2$。换句话说，$a = z \cdot L_x - z \cdot (L_x + L_y + L_z + L_{x+y} + L_{x+y+z})$。其中，$L_{x+y}$ 是像素 $106_{x,y}$ 的亮度。$R$，如果将像素 $106_{x,y}$ 映射到 $RG$ 对，则将值 $a$ 设定为对 $L$ 值应用的如下滤波器的输出：

$$
MLS_{RG} = \begin{pmatrix}
0 & z/4 & 0 \\
-1/4 & z & -1/4 \\
0 & -z/4 & 0
\end{pmatrix}
$$

其中，$z$ 是某个正的常数，例如 $1/2$。在这两个滤波器中 $z$ 值可以是或不是相同的。随后通过如下修改 RGBW 坐标使用值 $a$ 来选择用于像素 $106_{x,y}$ 的条件等色：

$$
W = W + a
$$

$$
R = R - m_r \cdot a
$$

$$
G = G - m_g \cdot a
$$

$$
B = B - m_b \cdot a
$$

其中，$m_r, m_g, m_b$ 是以如下方式通过显示器 110 的亮度发射特性定义的常数，即新的 RGBW 值（即式 2 中左侧的值）和旧的值定义了相同的颜色（即条件等色）。在某些实施例中，$m_r = m_g = m_b = 1$。此外，对于 $R, G$ 和 $B$，可将新的 RGBW 值硬衬位至 0 至 MAXCOL/M 表的范围内，对于 $W$ 则将新的 RGBW 值硬衬位至 0 至 MAXCOL/M 的范围内。

附录 B：确定背光单元输出功率

假设 $RwGwBwWw$ 是由图 6 中的 SPR 方框 454 确定的子像素值。这些子像素值在 0 至 MAXCOL/M 的范围内。如上所述，这些子像素值对应于 BL 的值 $BL_B$. 在方框 430 中，通过选择要在变形显示的最大子像素值 $P$ 来选择输出功率 $BL$. 更特别地，如上所述的，

$$
BL = BL_B \cdot INVy
$$

$$
P \cdot INVy = MAXCOL, \text{因此}
$$

$$
INVy = MAXCOL/P, \text{即}
$$

$$
BL = BL \cdot P = BL_B \cdot P / MAXCOL \quad (B1)
$$

存在多种选择 $P$ 的方式。在某些实施例中，由 SPR 方框 454 生成的 $Rw, Gw, Bw, Ww$ 子像素值分别乘以各自的系数 $Rweight, Gweight, Bweight, Wweight$（例如，$Rweight = 84\%, Gweight = 75\%, Bweight = 65\% \text{ or } 75\%, \text{以及 } Wweight = 100\%$），并将 $P$ 选择为在整个图像上所得的子像素值的最大值，即

$$
P = \max (Rw \cdot Rweight, Gw \cdot Gweight, Bw \cdot Bweight, Ww \cdot Wweight) \quad (B1-A)
$$

在某些实施例中，可如下计算得到的变量系数 $Xweight$ 替代系数 $Rweight$：

$$
Xweight = Rweight + ((Yweight - Rweight) \cdot Gw / 2^{SBITS}) \quad (B1-B)
$$

其中，$Rweight, Yweight$ 和 $SBITS$ 是预定的常数。

可以其他方式选择子像素值 $P$ 以获得期望的图像质量。
在某些实施例中，如下计算 BL 值。首先，对于每个像素值 120，在 (B1-A) 或 (B1-B) 中计算值 \( P_{sub} \)，即从子像素的 \( R_w, G_w, B_w, W_w \) 值中而非图像中的所有子像素中得到 (B1-A) 中的最大值。随后对于每个子像素值 120，根据 (B1) 初始化计算 BL 值 \( BL = BL(P_{sub}) \)（使用 \( P_{sub} \) 替代 \( P \) ）。将这些初始 BL 值累积到柱状图中。柱状图的柱 (bin) 计数器被反向地颠倒（起始于最大的 BL 值），并将累积误差函数 \( E_{sum} \) 计算为颠倒的柱中 BL 值的总和。例如，\( E_{sum}[i] \) 是柱号大于或等于 i 的柱中的 BL 值的总和，其中指数 i 随 BL 增大（即，较大的 BL 值被放置在具有较大 i 的柱中）。当 \( E_{sum}[i] \) 到达或超出预定阈值 \( TH_1 \) 则停止上述颠倒。假设这发生在 \( i = 10 \) 的柱。在某些实施例中，将背光输出功率 BL 设定为柱 10 中的某个值。例如，如果每个柱 i 都在某个数值 \( b_i \) 的和 \( b_{i+1} \) 之间计数 BL 值（所有 BL 具有 \( b_i \leq BL < b_{i+1} \)），则可将输出功率 BL 设定为 \( b_i \) 或大于等于 \( b_i \) 而小于 \( b_{i+1} \) 之间的其他一些值。

在某些实施例中，可执行线性插值以选择柱 10 中的 BL 值。例如，可将输出功率 BL 定义为如下的总和：

\[
BL = b_i + \text{fine_adjust_offset}
\] (B2)

其中，

\[
\text{fine_adjust_offset} = (\text{Excess} / \text{Delta} \ E_{sum}[i]) \times \text{bin_size}
\] (B3)

其中，\( \text{Excess} = E_{sum}[i] - TH_1 \); \( \text{Delta} \ E_{sum}[i] = E_{sum}[i] - E_{sum}[i+1] \)，其中 bin_size 是每个柱的大小，即 bin_size = \( b_{i+1} - b_i \)（在某些实施例中，该值是 16）。还可通过将 \( \text{Excess} \) 和另一更高阈值 \( TH_2 \) 相比较，来进行额外的调节。如果 \( \text{Excess} > TH_2 \) 则可将 \( \text{fine_adjust_offset} \) 设定为：

\[
\text{fine_adjust_offset} = (\text{Excess} / TH_2) \times \text{bin_size}
\]

随后可使用 (B2) 确定 BL。上述这些实施例都是非限定性的。

在某些实施例中，BL 和 INVY 值使 \( R_w G_w B_w W_w \) 数据延迟一帧。更特别地，通过缩放 444 使用一帧（“当前帧”）的 \( R_w G_w B_w W_w \) 数据确定的 INVY 值来缩放下一帧。当 LCD 面板 110 显示下一帧时，使用由当前帧的 \( R_w G_w B_w W_w \) 数据确定的 BL 值来控制背光单元 310。使用由先前帧的数据确定的 BL 和 INVY 值缩放并显示当前帧。这样的延迟允许在确定当前帧 BL 和 INVY 值之前开始显示当前帧。实际上，甚至在接收到当前帧的所有 sRGB 数据之前就可以开始显示当前帧。为了减小图像误差，可使 BL 值“衰减”（decayed），即 BL 值可由方框 430 产生作为从当前帧的数据确定的 BL 值和先前的 BL 值的加权平均值。在每秒显示 30 帧的一些显示器中，当图像亮度时快速地或在帧间变化时，对于 BL 和 INVY 值而言需要花费 36 帧来赶上图像亮度。这样的延迟在许多应用中是可以接受的。实际上，当不存在图像亮度的快速变化时，BL 和 INVY 值典型地不会在帧与帧之间变化很大，一帧的延迟不会导致图像的显著降级。当确实发生亮度的快速改变时，对于观察者而言需要花费时间以对图像进行视觉调节，因此由于 BL 和 INVY 值的延迟造成的图像误差不显著。见由 Hwang 等人在 2009 年 4 月 23 日提交的并被公开为 US2009/0102783A1 的美国专利申请中通过参考整体引入于此。
图 1
图4
图 5

输入图像

输入像素

106 106 106 106

106 106 106 106

106 106 106 106

106 106 106 106
图 6

部分 I.
SPR 之前的像素

部分 II.
SPR 之后的子像素

部分 III.
色域级别之后的子像素

图 7A
图 7B

图 8
图14