(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105321143 A
(43)申请公布日 2016.02.10

(21)申请号 201510455557.9

(22)申请日 2015.07.29

(30)优先权数据
62/030,497 2014.07.29 US
14/810,178 2015.07.27 US

(71)申请人 辉达公司
地址 美国加利福尼亚州

(72)发明人 杰弗里·艾伦·博尔兹
埃里克·B·卢姆
鲁伊·曼纽尔·巴斯托斯

(74)专利代理机构 北京市磐华律师事务所 11336
代理人 高伟 王睿

(51)Int.Cl.
G06T 1/20(2006.01)
G06T 1/60(2006.01)

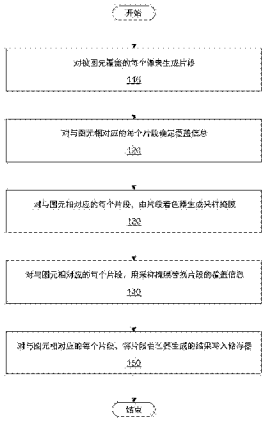
权利要求书2页 说明书17页 附图10页

(54)发明名称

来自片段着色程序的采样掩膜的控制

(57)摘要

公开了用于控制来自片段着色器的采样掩膜的方法、系统和计算机程序产品。该方法包括对至少部分被图元覆盖的每个像素生成片段，以及对与所述图元相对应的每个片段确定覆盖信息的步骤。然后，对每个片段，所述方法包括以下步骤：由片段着色器生成采样掩膜，用所述采样掩膜替换所述片段的所述覆盖信息，并且基于所述采样掩膜将所述片段着色器生成的结果写入存储器。该方法可以在配置为至少部分实现图形处理管线的并行处理单元上实现。



CN 105321143 A

1. 一种方法，包括：

对至少部分被图元覆盖的每个像素生成片段；

对与所述图元相对应的每个片段确定覆盖信息；并且对每个片段：

由片段着色器生成采样掩膜，

用所述采样掩膜替换所述片段的所述覆盖信息，并且基于所述采样掩膜将所述片段着色器生成的结果写入存储器。

2. 根据权利要求 1 所述的方法，其中所述片段包括数据结构，其包括一个或多个插值属性以及关于与特定像素相关联的多个采样的覆盖信息。

3. 根据权利要求 1 所述的方法，其中所述图元包括数据结构，其包括顶点的合集，其中所述顶点的合集中的每个顶点包括三维空间中的坐标以及一个或多个顶点属性。

4. 根据权利要求 1 所述的方法，其中生成片段以及确定覆盖信息在图形处理管线的光栅化阶段期间执行。

5. 根据权利要求 4 所述的方法，其中所述图形处理管线至少部分由并行处理单元实现，所述并行处理单元包括多个可编程流式多处理器 (SM)，每个 SM 配置为并行执行多个片段着色器线程。

6. 根据权利要求 5 所述的方法，其中每个片段着色器线程与在所述图形处理管线的所述光栅化阶段期间生成的不同片段相对应。

7. 根据权利要求 5 所述的方法，其中生成所述采样掩膜在所述图形处理管线的片段着色阶段期间执行，并且其中用所述采样掩膜替换所述覆盖信息在所述图形处理管线的光栅操作阶段期间执行。

8. 根据权利要求 7 所述的方法，其中所述并行处理单元包括光栅引擎，其配置为至少部分实现所述图形处理管线的所述光栅化阶段，以及光栅操作单元，其配置为至少部分实现所述图形处理管线的所述光栅操作阶段。

9. 根据权利要求 1 所述的方法，其中所述结果被写入多采样颜色缓冲器。

10. 根据权利要求 1 所述的方法，其中所述覆盖信息包括 N 比特掩膜，所述 N 比特掩膜中的每个比特与不同的采样位置相对应，所述采样位置与对应于所述片段的像素相关联。

11. 根据权利要求 10 所述的方法，其中所述采样掩膜包括 N 个比特，所述采样掩膜的每个比特与不同的采样位置相对应。

12. 根据权利要求 1 所述的方法，其中所述片段着色器接收所述覆盖信息。

13. 一种存储指令的非短暂计算机可读存储介质，当所述指令由处理器执行时，使得所述处理器执行的步骤包括：

对至少部分被图元覆盖的每个像素生成片段；

对与所述图元相对应的每个片段确定覆盖信息；并且对每个片段：

由片段着色器生成采样掩膜，

用所述采样掩膜替换所述片段的所述覆盖信息，并且基于所述采样掩膜将所述片段着色器生成的结果写入存储器。

14. 根据权利要求 13 所述的非短暂计算机可读存储介质，其中所述片段包括数据结构，其包括一个或多个插值属性以及关于与特定像素相关联的多个采样的覆盖信息，以及其中所述图元包括数据结构，其包括顶点的合集，其中所述顶点的合集中的每个顶点包括

三维空间中的坐标以及一个或多个顶点属性。

15. 根据权利要求 13 所述的非短暂计算机可读存储介质，其中所述处理器包括并行处理单元，所述并行处理单元包括多个可编程流式多处理器 (SM)，每个 SM 配置为并行执行多个片段着色器线程，以及其中生成片段和确定覆盖信息在图形处理管线的光栅化阶段期间执行，所述图形处理管线至少部分由所述并行处理单元实现。

16. 根据权利要求 13 所述的非短暂计算机可读存储介质，其中所述覆盖信息包括 N 比特掩膜，所述 N 比特掩膜中的每个比特与不同的采样位置相对应，所述采样位置与对应于所述片段的像素相关联，以及其中所述采样掩膜包括 N 个比特，所述采样掩膜的每个比特与不同的采样位置相对应。

17. 一种系统，包括 ：

并行处理单元，配置为至少部分实现图形处理管线，所述并行处理单元配置为通过以下步骤渲染图元 ：

对至少部分被图元覆盖的每个像素生成片段 ；

对与所述图元相对应的每个片段确定覆盖信息 ；并且

对每个片段 ：

由片段着色器生成采样掩膜，

用所述采样掩膜替换所述片段的所述覆盖信息，

并且

基于所述采样掩膜将所述片段着色器生成的结果写入存储器。

18. 根据权利要求 17 所述的系统，其中生成片段和确定覆盖信息在所述图形处理管线的光栅化阶段期间执行，其中生成所述采样掩膜在所述图形处理管线的片段着色阶段期间执行，并且其中用所述采样掩膜替换所述覆盖信息在所述图形处理管线的光栅操作阶段执行。

19. 根据权利要求 17 所述的系统，进一步包括 ：

存储器，其存储图形应用程序和设备驱动器，以及

主处理器，其耦合至所述存储器和所述并行处理单元，所述主处理器配置为执行所述图形应用程序和所述设备驱动器，

其中为了使所述并行处理单元用所述采样掩膜替换所述片段的所述覆盖信息，所述设备驱动器实现对所述图形应用程序所利用的图形 API 的扩展。

20. 根据权利要求 17 所述的系统，其中所述并行处理单元包括多个可编程流式多处理器 (SM)，每个 SM 配置为并行执行多个片段着色器线程。

来自片段着色程序的采样掩膜的控制

要求优先权

[0001] 本申请要求于2014年7月29日提交的申请号为:62/030,497(代理人案号:NVIDP1073+)、名称为“片段着色程序的采样掩膜的控制”的美国临时申请的权益,其全部内容以引用的方式并入本申请。

技术领域

[0002] 本发明涉及图形处理,并且更具体地涉及图元光栅化期间所生成的采样掩膜的控制。

背景技术

[0003] 许多应用,例如游戏软件和办公效率软件,配置为生成图形显示在计算机系统上。图形可以包括二维图形或三维图形。图形可以由通常的中央处理器(CPU)执行的软件生成。或者,图形可以由专用的协同处理单元,例如图形处理单元(GPU),执行的硬件生成,或者由CPU和/或GPU执行的软件和硬件的组合生成。为了使这样的应用的编程更加简单,创造出了特殊图形应用编程接口(APIs),例如DirectX和OpenGL等,在生成图形的功能和用于实现这种功能的硬件间创造一个抽象概念。

[0004] API可以指定抽象图形渲染管线,其限定图形渲染过程的不同步骤。例如,由OpenGL API指定的图形管线可以包括顶点着色阶段、几何着色阶段、光栅化阶段和片段着色阶段。图形着色管线接收应用指定的图形图元并且通过图形处理管线的每个阶段处理那些图元,以产生像素数据显示在显示设备上。光栅化阶段描述了将图元,例如一个三角形,转化成适于片段着色阶段进行操作的多个片段的操作。换句话说,由多个顶点限定的图元与由特定屏幕空间限定的像素(或像素的子采样)交叉并且与图元相关联的属性值被跨图元针对每个像素(或像素的子采样)进行插值。

[0005] 当实现抗混叠技术时,也可以在光栅化阶段生成表明了哪个像素的子采样被图元覆盖的覆盖信息。典型地,这个覆盖信息被指定为掩膜(mask)。掩膜可以对应于图元的特定片段并且掩膜内的每个元素可以对应于像素的特定子采样。例如,如果特定像素包括16个子采样,则掩膜可以包括16个比特,其表明了该像素中的每个子采样是否被图元覆盖。某些算法希望改变与给定的片段相关联的覆盖信息。然而,通常的硬件并不被配置为任意更新在光栅化阶段期间生成的覆盖信息。因此,需要处理这些问题和/或其他现有技术中的问题。

发明内容

[0006] 本发明公开了控制来自片段着色的采样掩膜的方法、系统和计算机程序产品。此方法包括对至少部分被图元覆盖的每个像素生成片段,以及对与所述图元相对应的每个片段确定覆盖信息的步骤。然后,对每个片段,所述方法包括以下步骤:由片段着色器生成采样掩膜,用所述采样掩膜替换所述片段的所述覆盖信息,并且基于所述采样掩膜将所述片

段着色器生成的结果写入存储器。此方法由并行处理单元实现，其配置为实现至少部分实现图形处理管线。

附图说明

[0007] 图 1 示出了根据一个实施例的、用于控制来自片段着色程序的掩膜采样的方法的流程图。

[0008] 图 2 示出了根据一个实施例的并行处理单元。

[0009] 图 3A 示出了根据一个实施例的、图 2 中的并行处理单元的通用处理集群。

[0010] 图 3B 示出了根据一个实施例的、图 2 中的并行处理单元的分区单元。

[0011] 图 4 示出了根据一个实施例的图 3A 中的流式多处理器。

[0012] 图 5 示出了根据一个实施例的包括图 2 中并行处理单元的片上系统。

[0013] 图 6 是根据一个实施例由图 2 中并行处理单元实现的图形处理管线的示意图。

[0014] 图 7A 示出了根据一个实施例的图元。

[0015] 图 7B 示出了根据一个实施例的用于片段的覆盖信息。

[0016] 图 8 示出了根据一个实施例的渲染图元的操作。

[0017] 图 9 示出了一个示范性系统，在其中前述不同的实施例的不同结构和 / 或功能可以实现。

具体实施方式

[0018] 一个常用的图形 API 是 Microsoft® DirectX，包括用于渲染三维图形的 Direct3D API 和用于渲染二维图形的 Direct2D API。在 Direct3D API 中，可以创造一个渲染目标，存储用于计算机生成图形的显示的颜色数据。D3D 中的渲染目标涉及一个缓冲区（即，存储器中分配的一部分），场景的图像数据存于其中。多采样抗锯齿 (MSAA) 可以用存储了每像素多个颜色采样的渲染目标来实现。然后将像素的多个颜色采样混合在一起来产生像素的最终像素颜色。然而，MSAA 的分辨率（即，每像素的采样数目）受到图形硬件可用的存储器尺寸的限制。

[0019] Microsoft® Direct3D 的版本 11.1（以后称为 D3D11）包括一个叫做目标独立光栅化 (TIR) 的新功能。D3D11TIR 使得在渲染期间利用多个光栅采样而仅在渲染目标内存储单一颜色采样。像素着色器可以配置为处理与多个光栅采样关联的覆盖信息，以生成单一颜色采样值存储于渲染目标内。D3D11TIR 使得利用 Direct2D 的矢量图形绘制利用 GPU 加速进行处理，而不是整个都在 CPU 上进行渲染。然而，D3D11TIR 的一个缺点是当 MSAA 和非多采样颜色缓冲器一起利用时，必须禁用深度和模板 (stencil) 测试。

[0020] 像素着色器接收光栅化期间产生的图元的覆盖信息的同时，像素着色器仅配置为处理覆盖信息以产生单一颜色值用于向渲染目标输出。单一颜色值可以至少部分基于覆盖信息内的设置比特的数目。像素着色器接收覆盖信息作为输入的同时，光栅化期间产生的覆盖信息不被修改，而是覆盖信息仅用于调节像素着色器输出并储存在渲染目标中的单一采样的颜色值。

[0021] 另一个常用的图形 API 是 OpenGL®，其包括渲染三维和二维图形的功能。

OpenGL® 也通过在光栅化图元期间生成覆盖信息来实现 MSAA。覆盖信息提供给片段着色器。在OpenGL® 中片段着色器可以向 gl_SampleMask 输出阵列写入值，使得片段着色器能够修改光栅化时生成的覆盖信息。就像OpenGL® 说明中所描述，针对片段的覆盖将变成覆盖信息和写入 gl_SampleMask 的值的逻辑“与”。将 gl_SampleMask 的比特设置为 0，将使得相应的采样被认为为了 MSAA 的目的未被覆盖。然而，将 gl_SampleMask 的比特设置为 1，将不会使得未被原始图元覆盖的采样被认为为了 MSAA 目的而被覆盖。因此，虽然由 OpenGL® 限定的 gl_SampleMask 输出阵列中写入的值可以用于修改覆盖掩膜，但不能用于改写光栅化时生成的覆盖信息。

[0022] D3D11 和 OpenGL® gl_SampleMask 输出阵列赋予的功能都不能使片段着色器完成对光栅化期间生成的像素的多个采样的覆盖信息进行改写的控制。片段着色器丝毫不能改变特定采样被标示为已覆盖而不是未覆盖。因此，要求这种功能的某些算法使用传统的图形加速硬件不能实现。

[0023] 图 1 示出了用于控制来自片段着色器程序的采样掩膜的方法 100 的流程图。步骤 110 中，在图形处理管线的光栅化阶段期间为每个像素生成片段，所述每个像素至少部分由图元覆盖。光栅化阶段涉及将限定为矢量形式（即通过多个顶点）的图形图元转化成光栅形式（即多个像素或片段）的操作。光栅化阶段为由图元覆盖的每个像素生成一个片段。

[0024] 例如 GPU 的处理器可以实现图形处理管线的至少一部分。在一个实施例中，图形处理管线的光栅化阶段由固定功能硬件单元实现，该固定功能硬件单元配置为接收图元并且输出一个或多个片段用于由图形处理管线的一个或多个附加阶段处理。在另一个实施例中，图形处理管线的光栅化阶段由一个 GPU 中的可编程的流式多处理器实现。

[0025] 图元可以限定为顶点的合集，其中每个顶点包括三维空间坐标和一个或多个顶点属性，例如，颜色、一个或多个纹理坐标、法向矢量等。图形处理管线的光栅化阶段可以接收限定了来自图形处理管线的之前阶段的图元的数据，在 GPU 上实现或由 CPU 执行的软件实现。片段可以包括一个或多个内插属性以及与特定像素关联的多个采样有关的覆盖信息。限定了图元和片段的数据可以被存储在数据结构中。

[0026] 步骤 120 中，对应于图元的每个片段的覆盖信息在图形处理管线的光栅化阶段期间确定。覆盖信息可以是掩膜，所述掩膜表明了与像素关联的每个采样位置是否被图元覆盖。当射线从穿过采样位置的摄影位置穿过由图元边缘限定的封闭表面时，采样位置确定被覆盖。换句话说，当位于观测平面的采样位置位于观测平面上的图元的投影版本之内时，采样位置确定被覆盖。

[0027] 在一个实施例中，固定功能硬件单元可以包括多个逻辑单元，每个逻辑单元配置为计算在关于像素位置的二维坐标内的采样位置。然后每个逻辑单元可以比较采样位置坐标和图元的投影顶点，以确定采样位置是否被覆盖或未被覆盖。每个采样的覆盖信息可以收集到掩膜中。

[0028] 步骤 130 中，对于每个与图元对应的片段，在图形处理管线的片段着色阶段期间由片段着色器生成采样掩膜。采样掩膜可以对应于与像素相关联的采样位置的数目。例如，

如果覆盖信息包括与 16 个采样位置相对应的 16 比特掩膜，则采样掩膜可以包括采样掩膜中的每个比特与像素的不同采样位置相对应的 16 比特掩膜。

[0029] 在一个实施例中，片段着色阶段由 GPU 中的可编程的流式多处理器实现。流式多处理器可以配置为执行片段着色器（即配置为处理片段的程序）。可以对每个在图形处理管线的光栅化阶段期间生成的片段执行片段着色程序的不同实例（即片段着色线程）。

[0030] 片段着色器可以包括由 GPU 执行的指令，使得生成采样掩膜并存储在 GPU 的存储器中（例如，记录器、DRAM 等）。不同的算法会指示如何生成采样掩膜。例如，如果一个算法要求使用保守的光栅化，那么如果在图形处理管线的光栅化阶段确定有任何与片段相关联的采样位置被覆盖，可以设置采样掩膜的所有比特。在另一个例子中，在一个关于路径渲染相关联的算法中，与特定采样位置关联的覆盖信息可以复制到采样掩膜的附加采样位置。在一个实施例中，采样掩膜的值至少部分由图形处理管线的光栅化阶段期间生成的覆盖信息确定。在另一个实施例中，采样掩膜的值的确定完全独立于覆盖信息。例如，采样掩膜可以代表多采样颜色缓冲的任意位置，其中，片段着色器的输出应该被写入。

[0031] 步骤 140 中，对于每个与图元相对应的片段，在图形处理管线的光栅化阶段期间生成的片段的覆盖信息由采样掩膜取代。在一个实施例中，覆盖信息可以由采样掩膜改写。在另一个实施例中，覆盖信息可以被修改，以表明片段被完全覆盖（即覆盖信息的所有比特都被设置为 1），并且按位“与”操作用于将覆盖信息和采样掩膜混合。

[0032] 步骤 150 中，对于每个与图元相对应的片段，在图形处理管线的片段着色阶段期间生成的结果基于采样掩膜被写入存储器。在一个实施例中，存储器包括具有颜色缓冲器的帧缓冲器。颜色缓冲器可以涉及分配用于存储每像素的一个或多个颜色值的存储器。帧缓冲器也可以包括一个深度缓冲器和 / 或模板缓冲器，分别用于存储与被渲染表面的每个像素的每个采样位置相关联的深度值和 / 或模板值。颜色缓冲的分辨率可以与深度缓冲器和模板缓冲器的分辨率相同或不同。

[0033] 现在，关于可选结构和特征的更多的说明性信息将被阐述，根据这些信息，前述体系结构可以按照使用者的愿望实施或不实施。应该高度注意，下列信息的解释仅用作说明用途，不应该解释为以任何方式限制。任何下列特征都可以可选择地并入描述的其他特征或者不被排斥。

[0034] 图 2 说明了根据一个实施例的并行处理单元 (PPU)200。在一个实施例中，PPU 200 是一个多线程处理器，由一个或多个集成电路装置实现。PPU 200 是延迟隐藏结构，设计为在并行处理大量线程。线程（即执行的线程）是配置为由 PPU 200 执行的一组指令的示例。在一个实施例中，PPU 200 是图形处理单元 (GPU)，配置为实现图形渲染管线，用于处理三维图形数据，以生成二维图像数据显示在显示设备例如液晶显示器 (LCD) 上。在其他实施例中，PPU 200 可用于执行通用计算。虽然在此提供一个用于说明用途的示范性并行处理器，应高度注意，这种处理器的解释仅用作说明用途，并且任何处理器可以用于补充和 / 或替换同样的处理器。

[0035] 如图 2 所示，PPU 200 包括输入 / 输出 (I/O) 单元 205、主机接口单元 210、前端单元 215、调度单元 220、工作分配单元 225、集线器 230、交叉开关 (XBar)270、一个或多个通用处理集群 (GPC)250 以及一个或多个分区单元 280。PPU 200 可以通过系统总线 202 与主处理器或其他外围设备相连。PPU 200 也可以与包括多个存储设备 204 的本地存储器相连。

在一个实施例中，本地存储器可以包括多个动态随机存取存储器 (DRAM) 设备。

[0036] I/O 单元 205 配置为通过系统总线 202（未示出）传送和接收来自主处理器的信息（即指令、数据等）。I/O 单元 205 可以通过系统总线 202 或一个或多个中间设备如存储器桥与主处理器直接通信。在一个实施例中，I/O 单元 205 实现高速外设部件互连 (PCIe) 接口，用于通过 PCIe 总线进行通信。在可替代的实施例中，I/O 单元 205 可以实现其他已知类型的接口，用于与外部设备进行通信。

[0037] I/O 单元 205 耦合至主机接口单元 210，对通过系统总线 202 接收到的数据包 (packet) 进行解码。在一个实施例中，数据包代表指令，配置为促使 PPU 200 执行不同的操作。如指令可能规定的，主接口单元 210 将解码指令传送至 PPU 200 的不同单元。例如，一些指令可以被传送至前端单元 215，其他指令可以被传送至集线器 230 或者 PPU 200 的其他单元，例如一个或多个复制引擎、视频编码器、视频解码器、功率管理单元等（未明确示出）。换句话说，主接口单元 210 配置为 PPU 200 的不同逻辑单元之间和其中的通信路径。

[0038] 在一个实施例中，主处理器执行的程序在缓冲器中对指令流进行编码，缓冲器提供工作负载给 PPU 200 处理。工作负载可以包括多个指令和由指令处理的数据。缓冲器是存储器中的一个区域，可以由主处理器和 PPU 200 进行存取（即读 / 写）。例如，通过 I/O 单元 205 由系统总线 202 发送存储器请求，主接口单元 210 可以配置为访问与系统总线 202 相连的系统存储器中的缓冲器。在一个实施例中，主处理器向缓冲器中写入指令流，然后发送指向指令流的开始端的指针到 PPU 200。主接口单元 210 向前端单元 215 提供指向一个或多个指令流的指针，前端单元 215 管理一个或多个流，从指令流读取指令并且将指令转送到 PPU 200 的不同单元。

[0039] 前端单元 215 耦合至调度单元 220，其将不同的 GPC 250 配置为处理一个或多个指令流限定的任务。调度单元 220 配置为追踪与调度单元 220 管理的不同任务相关的状态信息。此状态可以表明任务被分配给了哪个 GPC 250、任务有效还是无效、任务的优先级等。调度单元 220 管理一个或多个 GPC 250 上多个任务的执行。

[0040] 调度单元 220 耦合至工作分配单元 225，工作分配单元 225 配置为分配在 GPC 250 上执行的任务。工作分配单元 225 可以追踪从调度单元 220 接收的多个调度任务。在一个实施例中，工作分配单元 225 管理每个 GPC 250 的挂起 (pending) 任务池和有效任务池。挂起任务池可以包括多个槽（例如 16 个槽），多个槽中含有分配给特定 GPC 250 处理的任务。有效任务池可以包括多个槽（例如 4 个槽），多个槽中含有正由 GPC 250 有效处理的任务。当 GPC 250 完成任务执行时，该任务从 GPC 250 的有效任务池中逐出，并且选择并调度挂起任务池中的其他任务中的一个在 GPC250 上执行。如果 GPC 250 上的有效任务已经空闲，例如当等待依赖需解决的数据时，则有效任务可以从 GPC 250 逐出并且返回挂起任务池，而选择并调度挂起任务池中的另一个任务在 GPC 250 上执行。

[0041] 工作分配单元 225 通过 XBar270 与一个或多个 GPC 250 进行通信。XBar270 是一个将许多 PPU 200 的单元耦合至 PPU 200 的其他单元的互连网。例如，XBar270 可以配置为将工作分配单元 225 耦合至特定 GPC 250。虽然没有明确示出，但是 PPU 200 的一个或多个其他单元耦合至主机单元 210，其他单元也可以通过集线器 230 与 XBar270 连接。

[0042] 通过调度单元 220 管理任务并且通过工作分配单元 225 将任务分配到 GPC 250。GPC 250 配置为处理任务并生成结果。此结果可以被 GPC250 内的其他任务消耗，通过

XBar270 路由到不同的 GPC 250 或者储存在存储器 204 中。此结果可以通过分区单元 280 写入存储器 204，分区单元 280 实现存储器接口，用于向 / 从存储器 204 读取和写入数据。在一个实施例中，PPU 200 包括 U 个分区单元 280，相当于耦合到 PPU 200 的独立和不同的存储器设备 204 的数目。分区单元 280 将在下面和图 3B 中更加详细阐述。

[0043] 在一个实施例中，主处理器执行实现应用程序编程接口

(API) 的驱动内核，使得在主处理器上执行的一个或多个应用程序调度 PPU 200 的执行操作。应用程序可以生成指令（即 API 调用），使得驱动内核生成一个或多个 PPU 200 执行的任务。驱动内核输出任务到一个或多个 PPU 200 处理的任务流。每个任务可以包括一个或多个关联线程组，此处称作 warp。一个线程块可以涉及多个含有执行任务指令的线程组。在相同线程组的线程可以通过共享存储器交换数据。在一个实施例中，线程组包括 32 个关联线程。

[0044] 图 3A 示出了根据一个实施例的、图 2 中 PPU 200 的 GPC250。如图 3A 所示，每个 GPC 250 包括多个用于处理任务的硬件单元。在一个实施例中，每个 GPC 250 包括管线管理器 310、预光栅操作单元 (PROP)315、光栅引擎 325、工作分配交叉开关 (WDX)380、存储器管理单元 (MMU)390，以及一个或多个纹理处理集群 (TPC)320。应充分意识到，图 3A 中的 GPC 250 可以包括图 3A 示出的单元替代的或以外的硬件单元。

[0045] 在一个实施例中，由管线管理器 310 控制 GPC 250 的操作。管线管理器 310 管理一个或多个 TPC 的配置用于处理分配给 GPC250 任务。在一个实施例中，管线管理器 310 可以将一个或多个 TPC320 中的至少一个配置为实现图形渲染管线的至少一部分。例如，TPC 320 可以配置为在可编程流式多处理器 (SM)340 上执行顶点渲染程序。管线管理器 310 也可以配置为将从工作分配单元 225 接收的数据包发送到 GPC 250 内合适的逻辑单元。例如，一些数据包可以发送到 PROP 315 和 / 或光栅引擎 325 内的固定功能硬件单元，而其他数据包可以发送到 TPC 320，由图元引擎 335 或 SM 340 处理。

[0046] PROP 单元 315 配置为将光栅引擎 325 和 TPC 320 生成的数据发送到分区单元 280 内的光栅操作 (ROP) 单元，下面将更加详细阐述。PROP 单元 315 也可以配置为执行颜色混合的优化、像素数据编组、执行地址转换等。

[0047] 光栅引擎 325 包括配置为执行不同光栅操作的多个固定功能硬件单元。在一个实施例中，光栅引擎 325 包括设置引擎、进程光栅引擎、剔除引擎、裁剪引擎、精细光栅引擎以及碎片整合引擎。设置引擎接收变换的顶点并且生成与顶点限定的几何图元相关联的平面方程，平面方程被传送至进程光栅引擎以生成图元的覆盖信息（例如，碎片的 x，y 覆盖掩膜）。进程光栅引擎的输出被传送至剔除引擎，在此处，与落入 z 分布的图元相关联的片段被剔除并传送至裁剪引擎，在此处，位于视锥体之外的片段被裁剪。那些经过裁剪和剔除后留下来的片段可以送至精细光栅引擎，产生基于安装引擎生成的平面方程的像素片段的属性。光栅引擎 380 的输出包括要处理的片段，例如，由在 TPC 320 内部实现的片段着色器。

[0048] 每个 GPC 250 内的 TPC 320 包括 M-Pipe 控制器 (MPC)330、图元引擎 335、SM 340 以及一个或多个纹理单元 345。MPC 330 控制 TPC 320 的操作，TPC 320 将从管线管理器 310 接收的数据包发送到 TPC 320 内的合适的单元。例如，与顶点关联的数据包可以发送到图元引擎 335，图元引擎 335 配置为从存储器 204 取得与顶点关联的顶点属性。相反，与着色器程序关联的数据包可以被传送至 SM 340。

[0049] 在一个实施例中，纹理单元345配置为从存储器204下载纹理图（例如，纹理的2D矩阵）以及对纹理图进行取样来生成采样纹理值，用于SM 340执行的着色程序中。纹理单元345实现纹理操作，例如，利用mip-maps进行滤色操作（即改变细节层次的纹理图）。在一个实施例中，每个TPC 320包括4个纹理单元345。

[0050] SM 340包括可编程流处理器，配置为处理许多线程表示的任务。每个SM 340都是多线程并且配置为同时处理来自特定线程组的多个线程（例如，32线程）。在一个实施例中，SM 340实现SIMD（单指令，多数据）结构，其中线程组（即一个warp）中的每个线程配置为处理基于相同组指令的不同组数据。线程组内的所有线程执行相同的指令。在另一个实施例中，SM 340实现SIMT（单指令，多线程）结构，其中，其中线程组中的每个线程配置为处理基于相同组指令的不同组数据，但是线程组中的个别线程在执行期间允许偏离。换句话说，当线程组的一个指令从执行中被调离，线程组中的一些线程仍然有效，从而执行指令，而线程组中的其他线程失去效力，从而表现为未操作（NOP）而不是执行指令。SM 340将在下面以及图4中更加详细阐述。

[0051] MMU 390提供在GPC 250和分区单元280之间的接口。

MMU 390可以将虚拟地址转换成物理地址、存储保护和内存请求仲裁。在一个实施例中，MMU 390提供一个或多个转换后备缓冲器（TLBs），用于改进虚拟地址到存储器204中物理地址的转换。

[0052] 图3B阐述了根据一个实施例的、图2中PPU 200的分区单元280。如图3B所示，分区单元280包括光栅操作（ROP）单元350、第二级（L2）高速缓存360、存储器接口370以及L2交叉开关（XBar）365。存储器接口370耦合至存储器204。存储器接口370可以实现16、32、64、128位数据总线等，用于高速数据传输。在一个实施例中，PPU 200包括U个存储器接口370、每个分区单元280一个存储器接口370，其中每个分区但280与相应的存储器设备204相连。例如，PPU 200可以连接直到U个存储器设备204，例如，同步动态随机存取存储器，版本5，双数据速率（GDDR5SDRAM）。在一个实施例中，存储器接口370实现DRAM接口并且U等于6。

[0053] 在一个实施例中，PPU实现了多级存储器层次。存储器204设置在耦合至PPU 200的SDRAM内的片外。来自存储器204的数据可以被取得并储存在L2高速缓存360，L2高速缓存360位于片内并且在不同的GPC间共享。可以看到，每个分区单元280包括与对应的存储器设备204关联的L2高速缓存360的一部分。然后较低级别的高速缓存可以在GPC 250的不同单元内实现。例如，每个SM 340可以实现第一级（L1）高速缓存。L1高速缓存是用于特定SM 340的专用存储器。来自L2高速缓存360的数据可以取得并储存在每个L1高速缓存中，用于在SM 340的功能性单元中进行处理。L2高速缓存360耦合至存储器接口370和XBar270。

[0054] ROP单元350包括ROP管理器355、颜色ROP（CROP）单元352以及Z ROP（ZROP）单元354。CROP单元352执行涉及像素颜色的光栅操作，例如彩色压缩、像素混合等。ZROP单元354同光栅引擎325一起实现深度测试。ZROP单元354接收来自光栅引擎325的剔除引擎的与像素片段关联的采样位置的深度。ZROP单元354针对与片段相关联的采样位置测试与深度缓冲器中的对应深度相对的深度。如果片段通过针对采样位置的深度测试，则ZROP单元354更新深度缓冲器并将深度测试结构传送至光栅引擎325。ROP管理器355控

制 ROP 单元 350 的操作。应领会到，分区单元 280 的数目可能与 GPC 250 的数目不同，因此每个 ROP 单元 350 可以耦合至每个 GPC 250。因此，ROP 管理器 355 监测从不同的 GPC 250 接收的数据包并且确定其被由 ROP 单元 350 生成的结果的传送到哪个 GPC 250。CROP 单元 352 和 ZROP 单元 354 通过 L2XBar365 耦合至 L2 高速缓存 360。

[0055] 图 4 示出了根据一个实施例的图 3A 中的流式多处理器 340。如图 4 所示，SM 340 包括指令高速缓存 405、一个或多个调度单元 410、寄存器文件 420、一个或多个处理核 450、一个或多个特殊功能单元 (SFU)452、一个或多个加载 / 储存单元 (LSU)454、互连网 480 以及共用存储器 /L1 高速缓存 470。

[0056] 如上面所描述，工作分配单元 225 分配在 PPU 200 的 GPC250 上执行的任务。任务被分配给 GPC 250 内的特定 TPC 320，如果任务与着色器程序相关，则任务会分配给 SM 340。调度单元 410 接收来自工作分配单元 225 的任务并且管理用于一个或多个分配到 SM 340 的线程组（即 warps）的指令调度。调度单元 410 在并行线程组中调度线程，其中每个线程组叫做一个 warp。在一个实施例中，每个 warp 包括 32 个线程。调度单元 410 可以管理多个不同的 warp，在每个时钟周期内，调度用于执行的 warps，然后将来自多个不同 warp 的指令分配到不同的功能单元（即核心 350、SFU352 以及 LSU 354）。

[0057] 在一个实施例中，每个调度单元 410 包括一个或多个指令分派单元 415。每个分派单元 415 配置为传送指令至一个或多个功能单元。在图 4 所示的实施例中，调度单元 410 包括两个分派单元，其使得在每个时钟周期期间来自相同 wrap 的两个不同的指令能够被分派。在替代实施例中，每个调度单元 410 可以包括单个分派单元 415 或者附加的分配单元 415。

[0058] 每个 SM 340 包括寄存器文件 420，为 SM 340 的功能单元提供一组寄存器。在一个实施例中，寄存器文件 420 在每个功能单元之间划分，以便每个功能单元都分配到寄存器文件 420 的专用部分。在另一个实施例中，寄存器文件 420 在由 SM 340 执行的不同 warp 间划分。寄存器文件 420 向与功能单元的数据通路连接的操作数提供暂存。

[0059] 每个 SM 340 包括 L 个处理核 450。在一个实施例中，SM 340 包括大量（例如，192 等）不同的处理核 450。每个处理核 450 可以包括全管线化、单精度处理单元，该单元包括浮点算法逻辑单元和整数算法逻辑单元。处理核 450 也可以包括双精度处理单元，该单元包括浮点算法逻辑单元。在一个实施例中，浮点算法逻辑单元执行浮点算法的 IEEE754-2008 标准。每个 SM 340 还包括执行特殊功能（例如，像素混合操作等）的 M 个 SFU 452，以及在共享存储器 /L1 高速缓存 470 和寄存器文件 420 间执行加载和存储操作的 N 个 LSU 454。在一个实施例中，SM 340 包括 192 个核 450、32 个 SFU 452 以及 32 个 LSU 454。

[0060] 每个 SM 340 包括互连网 480，其连接每个功能单元至寄存器文件 420 和共享存储器 /L1 高速缓存 470。在一个实施例中，互连网 480 是可以配置为将任意功能单元连接到寄存器文件 420 中的任意寄存表或共享存储器 /L1 高速缓存 470 中的存储位置的交叉开关。

[0061] 共享存储器 /L1 高速缓存 470 是片上存储器阵列，在一个实施例中，可以配置为共享存储器或 L1 高速缓存或二者结合，根据应用需求。例如，共享存储器 /L1 高速缓存 470 可以包括 64kB 的存储能力。共享存储器 /L1 高速缓存 470 可以配置为 64kB 共享存储器或者 L1 高速缓存或者二者结合，例如，16kB L1 高速缓存以及 48kB 共享存储器。

[0062] 上面描述的 PPU 200 可以配置为执行比通常 CPU 快得多的高度并行计算。并行计

算在图形处理、数据压缩、识别、流处理运算等中具有优势。

[0063] 在一个实施例中，PPU 200 包括图形处理器 (GPU)。PPU 200 配置为接收指定着色器程序处理图形数据的指令。图形数据可以限定为一组图元，例如点、线、三角形、四边形、三角形带等。典型地，图元包括指明图元（例如，在模型 - 空间坐标系统中）的许多顶点以及与图元的每个顶点关联的属性的数据。PPU 200 可以配置为处理图元生成帧缓冲（即用于每个显示像素的像素数据）。

[0064] 应用程序将场景（例如，顶点和属性的合集）的模型数据写入存储器，例如系统存储器或存储器 204。模型数据限定了在显示器上可见的每个对象。然后应用程序向驱动核调用 API，请求渲染并显示模型数据。驱动核读取模型数据并向一个或多个流写入指令以执行处理模型数据的操作。指令可以涉及 PPU 200 的 SM 340 上要执行的不同的着色器程序，包括一个或多个顶点着色器、外壳着色器、区域着色器、几何着色器和像素着色器。例如，一个或多个 SM 340 可以配置为执行顶点着色器程序，来处理模型数据限定的大量顶点。在一个实施例中，不同的 SM 340 可以配置为同时执行不同的着色器程序。例如，SM 340 的第一子集可以配置为执行顶点着色器程序，而 SM 340 的第二子集可以配置为执行像素着色器程序。SM 340 的第一子集处理顶点数据生成经过处理的顶点数据并且将经过处理的顶点数据写入 L2 高速缓存 360 和 / 或存储器 204。在经过处理的顶点数据被光栅化（即在场景空间内将三维数据转换成二维数据）生成片段数据后，SM 340 的第二子集执行像素着色，生成经过处理的片段数据，然后与其他经过处理的片段数据混合，并写入存储器 204 中的帧缓冲。顶点着色器程序和像素着色器程序可以同时执行，以管线化的方式处理来自相同场景的不同数据，直到场景的所有模型数据都被渲染进入帧缓冲。然后，帧缓冲中的内容被传送到显示控制器，显示在显示设备上。

[0065] PPU 200 可以包含在台式电脑、笔记本电脑、平板电脑、智能手机（例如，无线、手持设备）、个人数字助理 (PDA)、数码相机、手持电子设备等中。在一个实施例中，PPU 200 嵌入单半导体衬底中。在另一个实施例中，PPU 200 和一个或多个其他逻辑单元一起包含在片上系统 (SoC) 中，其他逻辑单元例如精简指令集计算机 (RISC)CPU、存储器管理单元 (MMU)、数模转换器 (DAC) 等。

[0066] 在一个实施例中，PPU 200 可以包含在显卡中，显卡包括一个或多个存储器设备 204，例如 GDDR5SDRAM。显卡可以配置为与台式电脑主板上的具有 PCIe 槽的接口，包括例如北桥芯片集和南桥芯片集。然而在另一个实施例中，PPU 200 可以是一个包含在主板的芯片集（即北桥）中的集成图形处理器 (iGPU)。

[0067] 图 5 示出了根据一个实施例的、包括图 2 中 PPU 200 的片上系统 (SoC)500。如图 5 所示，如上所述，SoC 500 包括 CPU 550 和 PPU 200。SoC 500 也可以包括系统总线 202，使得能够在 SoC 500 的不同组件间进行通信。CPU 500 以及 PPU 200 生成的内存请求可以通过 MMU 系统 590 发送，MMU 系统 590 由 SoC 500 的多个组件共享。SoC 500 也可以包括耦合至一个或多个存储器设备 204 的存储器接口 595。存储器接口 595 可以实现例如 DRAM 接口。

[0068] 虽然没有明确示出，但是 SoC 500 可以包括图 5 中示出的组件之外的其他组件。例如，SoC 500 可以包括多个 PPU 200（例如，4 个 PPU200）、视频编码器 / 解码器和无线宽带收发机以及其他组件。在一个实施例中，SoC 500 和存储器 204 一起包含在封装体叠层 (PoP)

配置中。

[0069] 图 6 是根据一个实施例的、图 2 中 PPU 200 实现的图形处理管线 600 的示意图。图形处理管线 600 是实现从 3D 几何数据生成 2D 计算机生成的图像的处理步骤的抽象流程图。众所周知，通过将操作分割成多个阶段，管线结构可以更加高效地处理长延迟操作，其中每个阶段的输出耦合到接下来下个阶段的输入。因此，图形处理管线 600 接收输入数据，将输入数据从图形处理管线 600 的一个阶段传送到下个阶段以产生输出数据 602。在一个实施例中，图形处理管线 600 可以代表OpenGL® API 限定的图形处理管线。

[0070] 如图 6 所示，图形处理管线 600 包括含有许多阶段的管线结构。许多阶段包括但不限于数据组合阶段 610、顶点着色阶段 620、图元组合阶段 630、几何着色阶段 640、视口缩放、剔除和裁剪 (VSCC) 阶段 650、光栅化阶段 660、片段着色阶段 670 以及光栅操作阶段 680。在一个实施例中，输入数据 601 包括将处理器配置为实现图形处理管线 600 的各个阶段的指令以及要由各个阶段处理的几何图元（例如，点、线、三角形、四边形、三角带或者扇形等）。输出数据 602 可以包括像素数据（即颜色数据），像素数据被复制到帧缓冲或存储器内其他类型的表面数据结构中。

[0071] 数据组合阶段 610 接收指定高阶面、图元等的顶点数据的输入数据 601。数据组合阶段 610 将顶点数据收集在暂存或队列中，例如通过从主处理器接收命令并从缓冲器读取顶点数据，所述命令包括指向内存中缓冲器的指针。然后顶点数据被传送至顶点着色阶段 620 进行处理。

[0072] 顶点着色阶段 620 每次通过对每组顶点执行一系列操作（顶点着色器或程序）处理顶点数据。顶点可以例如指定为与一个或多个顶点属性（例如，颜色、纹理坐标、曲面法线等）关联的 4 坐标矢量（即 <x,y,z,w>）。顶点着色阶段 620 可以操作特性，例如位置、颜色、纹理坐标等。换句话说，顶点着色阶段 620 对顶点坐标或其他与顶点有关的顶点属性进行操作。这样的操作通常包括照亮操作（即修改顶点的颜色属性）和转换操作（即修改顶点的坐标空间）。例如，可以使用在对象坐标空间的坐标限定顶点，对象坐标空间通过坐标乘以矩阵进行转换，矩阵将坐标从对象坐标空间转化到世界空间或规格化设备坐标 (NCD) 空间。顶点着色阶段 620 生成经转换的顶点数据，其被转移到图元组合阶段 630。

[0073] 图元组合阶段 630 收集顶点着色阶段 620 输出的顶点并且将其组合成几何图元，用于几何着色阶段 640 进行处理。例如，图元组合阶段 630 可以配置为将每三个连续顶点组合作为一个几何图元（即三角形），用于传送至几何着色阶段 640。在一些实施例中，特定顶点可以被再用于连续的几何图元（例如，三角形带中的两个连续三角形可以共享两个顶点）。图元组合阶段 630 将几何图元（即关联顶点的合集）传送至几何着色阶段 640。

[0074] 几何着色阶段 640 通过对几何图元执行一系列操作（即几何着色器或程序）处理几何图元。曲面细分操作可以从每个几何图元生成一个或多个几何图元。换句话说，几何着色阶段 640 可以将每个几何图元细分成适于图形处理管线 600 的其余阶段处理的两个或多个几何图元的精细网格。几何着色阶段 640 将几何图元传送至视口 (viewport)SCC 阶段 650。

[0075] 视口 SCC 阶段 650 对几何图元执行视口缩放、剔除和裁剪操作。被渲染的每个表面都与抽象摄影位置相关联。摄影位置代表了正观看场景的观察者的位置并且限定了包围了场景中对象的视锥体。视锥体可以包括视平面、后平面和四个裁剪平面。任何完全位于

视锥体之外的几何图元被剔除（即被抛弃），因为该几何图元对最后的渲染场景没有贡献。任何部分处于视锥体之内和部分处于视锥体之外的几何图元可以被裁剪（即转换成一个包围在视锥体之内的新几何图元）。而且，每个几何图元都可以根据视锥体的深度被缩放。然后所有潜在的可视几何图元被传送至光栅化阶段 660。

[0076] 光栅化阶段 660 将 3D 几何图元转化成 2D 片段。光栅化阶段 660 可以配置为利用几何图元的顶点建立一组平面方程式，其中可以插值不同的属性。光栅化阶段 660 也可以计算多个像素的覆盖掩膜，覆盖掩膜表明了像素的一个或多个采样位置是否拦截了 (intercept) 几何图元。在一个实施例中，也可以进行 z 测试，以确定几何图元是否被其他已经光栅化的几何图元遮挡。光栅化阶段 660 生成片段数据（即对每个覆盖的像素插值与特定采样位置相关联的顶点属性），被传送至片段着色阶段 670。

[0077] 片段着色阶段 670 通过对每个片段执行一组指令（即片段着色器或程序）来处理片段数据。例如通过执行照亮操作或利用对片段插入纹理坐标进行纹理映射，片段着色阶段 670 可以生成片段的像素数据（即颜色值）。片段着色阶段 670 生成像素数据，被传送至光栅操作阶段 680。

[0078] 光栅操作阶段 680 对像素数据执行不同的操作，例如进行 α 测试、模板测试以及将像素数据和对应于与像素关联的其他片段的其他像素数据相混合。当光栅操作阶段 680 处理完像素数据（即输出数据 602），像素数据可以写入渲染目标，例如帧缓冲，颜色缓冲等。

[0079] 应意识到，除了或代替一个或多个上面描述的各阶段，图形处理管线 600 可以包括一个或多个附加阶段。理论上的图形处理管线的不同实现方式可以实现不同的阶段。此外，在一些实施例中，上面描述的一个或多个阶段可以从图形处理管线上去除（例如几何着色阶段 640）。其他类型的图形处理管线也被考虑在此发明公开的范围内。此外，图形处理管线 600 的任何阶段都可以由图形处理器，例如 PPU 200，内的一个或多个专用硬件单元实现。图形处理管线 600 的其他阶段可以由可编程硬件单元实现，例如 PPU 200 的 SM 340。

[0080] 图形处理管线 600 通过由主处理器，例如 CPU 550，执行的应用程序实现。在一个实施例中，设备驱动器可以执行应用程序编程接口 (API)，其限定了可以被应用程序利用的不同功能以便生成用于显示的图形数据。设备驱动器是软件程序，包括控制 PPU200 操作的多个指令。API 为程序员提供了抽象概念，让程序员利用专业的图形硬件，例如 PPU 200，生成图形数据，而不要求程序员使用 PPU 200 的具体指令集。应用程序可以包括 API 调用，其被发送到 PPU 200 的设备驱动器。设备驱动器理解 API 调用并且执行响应 API 调用的不同指令。在一些实例中，设备驱动器可以通过在 CPU 550 上执行指令来完成操作。在其他实例中，设备驱动器可以通过在 PPU 200 上开始操作来完成至少部分完成操作，PPU200 利用 CPU 550 和 PPU 200 之间的输入 / 输出接口。在一个实施例中，设备驱动器配置利用 PPU 200 的硬件实现图形处理管线 600。

[0081] 为了实现图形处理管线 600 的不同阶段，可以在 PPU 200 内执行不用的程序。例如，设备驱动器可以在 PPU 200 上启动内核，以在一个 SM 340（或多 SM 340）上执行顶点着色阶段 620。设备驱动器（或 PPU 200 执行的初始内核）也可以在 PPU 200 上启动其他内核，以执行图形处理管线 600 的其他阶段，例如几何着色阶段 640 和片段着色阶段 670。此外，图形处理管线 600 的一些阶段可以在固定单元硬件上实现，例如 PPU 200 内实现的光栅

化器或数据组合器。应意识到，来自一个内核的结果在由 SM 340 上的后续内核处理之前，可以由一个或多个介入固定功能硬件单元处理。

采样掩膜的控制

[0082] 图 7A 阐述了根据一个实施例的图元 720。如图 7A 所示，图元 720 覆盖多个像素 710。每个像素 710 可以与多个采样位置相关联（以十字准线表明）。在一个实施例中，每个像素 710 与 16 个采样位置关联。在不同的实施例中，对每个像素限定不同数目的采样位置（例如 4 个采样位置、16 个采样位置等）。而且，采样位置可以由像素 710 的界限均匀隔开。可替代地，采样位置可以抖动。如图 7A 所示，图元 720 至少部分覆盖（即横跨）16 个像素中的 5 个（像素 710(5)，710(6)，710(7)，710(10) 和 710(11)）。

[0083] 在一个实施例中，光栅引擎 325 包括固定功能硬件，用于实现图形处理管线 600 的光栅化阶段 660。在光栅化期间，光栅引擎 325 接收限定每个图元的顶点数据。顶点数据可以包括三个顶点以及相关的顶点属性。例如，图元 720 的每个顶点可以由四个矢量要素—x 坐标、y 坐标、z 坐标和 w 坐标指定，四个坐标代表了在齐次坐标中顶点位置。每个顶点也可以关联颜色值、纹理坐标（例如 s 坐标和 t 坐标）、法向矢量等。

[0084] 光栅引擎 325 可以接收图元 720 的所有三个顶点的数据并且可以利用顶点数据建立图元的边缘方程式。光栅引擎 325 利用边缘方程式确定哪个像素 710 与图元相交。对于每个相交的像素 710，光栅引擎可以生成在图形处理管线 600 的片段着色阶段 670 期间由片段着色器处理的片段。片段可以包括用于片段的内插 z 值、用于每个顶点属性的插值以及覆盖信息，所述覆盖信息表明了与像素 710 关联的哪个采样位置被图元 720 覆盖。

[0085] 图 7B 阐述了根据一个实施例片段的覆盖信息 700。虽然覆盖信息 700 为了说明性目的而显示为对应于像素 710 的采样位置的相对位置的二维阵列，一般来说，覆盖信息 710 是 N 个比特值，每个比特对应像素 710 的一个特定采样位置。覆盖信息 710 可以在图形处理管线 600 的光栅化阶段 660 期间产生并且可替代地可以称作光栅化的覆盖信息。

[0086] 如图 7B 所示，与图 7A 中像素 710(5) 对应的片段将包括覆盖信息 700。覆盖信息 700 包括 16 个比特。在覆盖信息中，“0”值表明采样位置没有被图元 720 覆盖，以及值“1”表明采样位置被图元 720 覆盖。因为在像素 710(5) 中图元 720 覆盖了四个采样位置，覆盖信息给定为 0b0000000001110001。当光栅化阶段 660 的输出被立即消耗或者光栅化阶段 660 的输出储存在存储器 204 中以用于后面的处理时，包含覆盖信息 700 的片段可以储存在存储器中，例如与配置为执行片段着色的 SM 340 相关联的寄存器文件 420。

[0087] 图 8 示出了根据一个实施例的、控制来自片段着色程序的采样掩膜的操作。如图 8 所示，光栅器（例如，光栅引擎 325）生成一个或多个图元的片段，例如图元 720。片段着色器，由一个或多个 SM 340 执行，配置为处理片段。用于每个片段的数据被发送到配置为处理该片段的特定 SM 340。数据包括图形处理管线 600 的光栅化阶段 660 期间生成的覆盖信息 700。在一个实施例中，覆盖信息 700 也被发送到 ROP 350，其配置为将片段着色器的输出与储存在帧缓冲中的值混合，以生成每个像素的最终值。

[0088] 在图形处理管线 600 的光栅操作阶段 680 期间，ROP 350 用图形处理管线 600 的片段着色阶段 670 期间生成的采样掩膜代替图形管线 600 的光栅化阶段 660 期间生成的覆盖信息。在一个实施例中，ROP 350 配置为重新初始化片段的覆盖信息 700，以表明与对应的像素关联的采样位置被完全覆盖。换句话说，传送至 ROP 350 的覆盖信息 700 由 0xFF（即

全部“1”）改写。然后执行按位“与”，将修改的覆盖信息700与片段着色器生成的掩膜采样结合。事实上，片段着色器生成的采样掩膜被传送至ROP 350并取代光栅化阶段660生成的覆盖信息700。像素内的任何采样位置可以由片段着色器控制以表明被覆盖或未被覆盖。重要地，任何采样位置可以独立于覆盖信息700由片段着色器控制。片段着色器可以在着色期间写入采样掩膜，以便在图形处理管线600的光栅操作阶段680期间控制ROP 350的操作。

[0089] 应意识到，此处描述的PPU 200仅仅是一个通过片段着色器实现采样掩膜的控制的示例性结构。例如，在其他实施例中，图形处理管线600的光栅化阶段660可以在可编程单元例如SM 340内部实现，而不是在专用硬件单元例如光栅引擎325内实现。在图元光栅化期间生成覆盖信息并且然后用着色期间生成的采样掩膜取代覆盖信息的其他类型的结构，都应被考虑处于本公开的范围内。

[0090] 在一个实施例中，PPU 200可通信地耦合至主处理器，例如CPU。主处理器可以耦合至储存图形应用程序、操作系统、PPU 200的设备驱动器等的存储器。设备驱动器可以实现图形API，例如OpenGL API。为了实现覆写来自片段着色器的覆盖信息的功能，设备驱动器也可以实现图形API的扩展。为了将PPU 200配置为通过片段着色器控制采样掩膜，图形应用程序可以包括使用图形API扩展的指令。

[0091] 例如，OpenGL®规范的API扩展，称为

NV_sample_mask_override_coverage，可以由PPU 200的设备驱动器限定并实现。扩展允许片段着色器对是否gl_SampleMask输出阵列能够使得未被图元覆盖或未通过提前深度/模型测试的采样能表明采样被图元覆盖进行控制。通过重声明图形应用程序中的指令内具有“override_coverage”布局限定符的gl_SampleMask输出阵列，使得具有该功能，具体如下：

layout(override_coverage)out intgl_SampleMask[]；

[0092] 一旦gl_SampleMask输出阵列变量已经被重声明，那么片段着色器通过向每个片段的gl_SampleMask变量写入值来覆写覆盖信息。如果片段着色器没有明确地向gl_SampleMask变量写入一个值，则采样掩膜未被限定并且覆盖信息未被ROP 350修改。然而如果片段着色器在gl_SampleMask输出阵列写特定的比特为0，则对应的采样位置被处理为未覆盖，或者如果片段着色器在gl_SampleMask输出阵列写入特定的比特为1，则对应的采样位置被处理为已覆盖。ROP 350将用gl_SampleMask输出阵列中指定的采样掩膜取代覆盖信息。如果图形应用程序没有使用override_coverage布局限定符声明的gl_SampleMask输出阵列，则ROP 350将配置为采用储存在gl_SampleMask输出阵列中的值进行覆盖信息的按位“与”，如果在对片段进行着色期间值被写入gl_SampleMask输出阵列，就像前面由OpenGL®规范指定的一样。

[0093] 在其他实施例中，OpenGL® API的扩展NV_sample_mask_override_coverage可以由其他设备驱动器实现不同于PPU 200的结构。然而在其他实施例中，可以限定具有其他相似功能的扩展用于其他图形API，例如Direct3D API。

[0094] 应意识到，可以对任何片段着色算法或应用程序赋予取代覆盖信息的能力并且不限于TIR应用程序。使得覆盖信息由片段着色器生成的采样掩膜取代的能力允许应用程序

“打开”起初未被图元覆盖的采样。而且，应意识到，片段着色器线程，是对特定片段执行的片段着色器实例，仅仅可以影响对应像素内的采样。片段着色器线程不能打开其他像素内的采样，也不能创造新的片段。除此以外，当利用多个片段着色器通道使用混合抗锯齿时，片段着色器线程仅仅能够打开与那个通道对应的采样。

[0095] 就像在此处使用的，片段着色器线程涉及对一组片段数据执行的片段着色器程序的特定实例，其中片段着色数据与由特定图元覆盖的单个像素关联。例如，SM 340 的 16 个核 450 可以配置为并行执行 16 个片段着色器线程，每个核 450 与目标渲染表面的 16 个像素的一个关联。当图元被光栅化时，与没有和图元交叉的像素相关联的任何核将禁用。例如，配置为执行片段着色器的 SM 340 的 16 个核 450 可以对应于图 7A 中所示的 16 个像素 710。当处理与图元 720 关联的 5 个片段时，在着色期间，仅有核 450 中的 5 个可以被使能（并且 11 个禁用）。5 个有效核中的每个可以对与图元 720 交叉的 5 个像素 710 中的每个执行与片段对应的单个片段着色器线程。

[0096] 在一个实施例中，为了减少被完全着色的片段的数目，片段图形处理管线 600 可以包括深度测试（即 z 测试），作为剔除被其他图元完全覆盖的任何图元的方法。典型地，通常在着色后发生深度缓冲写入，因此，影响片段着色器生成的采样掩膜的修改后的覆盖信息可以用于将深度值写入深度缓冲。然而，仍有可能在着色（即在片段着色器生成用于修改覆盖信息的采样掩膜之前）之前执行提前保守深度测试（即 z 拣选）。当能够进行提前 z 测试时，当对图元计算深度范围 (z-gamut) 时，有必要将每个像素当作完全覆盖的。换句话说，因为在光栅化期间片段着色器能“打开”未被图元覆盖的采样，应小心，如果片段着色器将采样从未覆盖改变为覆盖，则在提前 z 测试期间使用更保守 z-gamut 值保证如果在图元中有未被覆盖的采样能够基于深度缓冲中储存的值可视时片段在着色之前没有被剔除。因此，在执行提前 z 测试时，有必要不使用 z-per-silver（即在三角形边缘、像素边缘交叉点）或图元的原始 z-gamut。

[0097] 路径渲染是一个应用程序，其中通过片段着色器的采样掩膜的控制可以得到利用。路径渲染涉及渲染 2D 图形的样式。在路径渲染中，场景被指定为一系列的屏幕分辨率独立轮廓，被称为路径，其可以被填充或描画（即未被填充）。路径可以被喷涂成不变的颜色、直线形或辐射梯度，或图像。路径渲染生成能够独立于屏幕分辨率而缩放的图像。在路径渲染期间使用片段着色器控制采样掩膜的一个例子是将一个采样的覆盖涂到像素内的一组采样，使得这组采样被当作好像它们在单个采样位置。

[0098] 多采样颜色缓冲器是另一个应用程序，其中通过片段着色器的采样掩膜的控制可以得到利用。反锯齿涉及用于减少伪影的多种技术，伪影由有限分辨率下的高频率数据采样引起。一种技术，多采样抗锯齿 (MSAA) 涉及其中对每个像素（例如，对处于像素中心的单个采样位置）片段着色器执行一次，但是对像素内的每个采样位置计算深度和模板值的实现方式。在 MSAA 中，基于片段着色器的覆盖信息，单个颜色值与颜色缓冲器中的值相混合。在另一种技术中，全场景抗锯齿或超级采样涉及其中产生较高分辨率的图像以及然后对图像降低采样以生成最终图像的实现方式。可以对每个采样位置计算颜色值并且将其与存储在多采样颜色缓冲器中的值混合。在对全景着色后，可以对颜色缓冲器进行滤色以生成每个像素的最终颜色值。

[0099] 当多采样颜色缓冲器在渲染算法中使用时，多采样颜色缓冲器中的每个槽与特定

像素的特定采样位置对应。例如，多采样颜色缓冲器可以包括每个像素四个槽，其中四个槽中的每个对应于像素内的不同采样位置。对由图元覆盖的采样位置中的每个位置可以生成颜色值并且储存在多采样颜色缓冲器中。因此，覆盖信息 700 被利用来确定颜色缓冲器中的哪个槽应该被改写或由片段着色器输出的颜色值混合。利用通常技术，多采样颜色缓冲器中的数据仅能够被写入由图元覆盖的多采样颜色缓冲器的槽。相反地，当由片段着色器生成的采样掩膜用于取代光栅化阶段 660 中生成的覆盖信息 700 时，则多采样颜色缓冲器可以被当作更通用的数据结构一样。更具体地，片段着色器可以用于写入采样掩膜，采样掩膜表明了颜色缓冲器的哪个或哪些槽与片段着色器输出的颜色值相对应。然后多采样颜色缓冲器可以作为可以写入由片段着色器随意选择的一组索引的槽的阵列使用。然后多采样颜色缓冲器能够被滤色以生成渲染图像中每个像素的单个值，或者多采样颜色缓冲器能够通过第二通道中的不同的片段着色器程序进行处理，其中第二通道中的多采样颜色缓冲器中的数据用程序限定的方法进行处理。

[0100] 在一个实施例中，对每个片段，片段着色器生成单个颜色值。然后单个颜色值被改写，或与储存在由覆盖信息表明的多采样颜色缓冲器的每个槽中的值相混合。换句话说，基于与覆盖像素的采样位置的多个图元关联的多个片段，对每个采样位置可以储存不同的颜色，即使对特定片段整个像素生成单个颜色。在另一个实施例中，对每个特定采样位置生成不同颜色值。例如，在光栅化期间对每个采样位置可以生成纹理坐标。然后执行多个片段着色器线程，且每个片段着色器线程与特定采样位置关联。然后每个片段着色器线程基于采样位置的纹理坐标从纹理映射对颜色值采样，最后，作为结果的颜色值被写入多采样颜色缓冲器中的对应槽中。

[0101] 有多种使用情况，其中写入多采样颜色缓冲器的任意槽的能力是有益的。例如，一个这样的用途是用在抗锯齿体系中，其中多采样颜色缓冲器中的槽的数目少于与可见性（例如，深度和模板测试）相关联的采样位置的数目。换句话说，多采样颜色缓冲器的分辨率小于采样位置的分辨率。例如，像素可以与 16 个可见性采样位置关联，但多采样颜色缓冲器对每个像素可以仅包括两个或四个槽用于储存颜色值。槽的数目等于与可见性相关联的采样位置数目的多采样颜色缓冲器需要太多的存储器。然而，可以实现更多易控制的多采样颜色缓冲器，以便每个像素的多个颜色值可以储存起来而始终保持较高分辨率的可见性信息。储存在颜色缓冲器中的多个颜色值可以基于较高的分辨率可见性信息被混合。

[0102] 通过将输出颜色值与可见性采样位置解耦，有可能使用数目减少的槽代表例如深度缓冲器中的任何采样位置。一种可能的损耗算法的实现方式将是实现具有两个槽的多采样颜色缓冲器并且基于储存在两个槽中的颜色值的加权平均值生成最终颜色值。加权平均值可以根据片段着色器生成的可见性信息（即采样掩膜）确定。

[0103] 另一种使用情况是利用多采样颜色缓冲器实现顺序无关透明度，其中算法存储了每个像素的半透明颜色以及深度的未分类列表。然后算法在后处理通道中对半透明颜色进行分类和合成。片段着色器保持每个像素索引，其计算与像素关联的半透明片段的数目。然后通过在采样掩膜（即颜色和深度不基于覆盖存储在采样位置，相反地，按照对应于特定像素的半透明片段的顺序）中设置合适的比特，片段着色器能够存储在与索引相关联的采样处的片段颜色和深度。当索引计数超过采样的数目时，为了储存与像素关联的其余的颜色 / 深度组合，掩膜可以设置为 0 并且片段着色器能退回到片段的较慢全局列表。

[0104] 另一种使用情况是实现保守光栅化的有限表格。在保守光栅化中，生成覆盖信息以便如果有采样被图元覆盖，则所有采样被当作被图元覆盖。然而，当实现保守的光栅化时，则一般基于像素被完全覆盖（这会导致算法出现一些不准确）执行深度和模板测试。反而，一种算法可以通过基于光栅化生成的精确覆盖信息执行提前 z 和 / 或模板测试并且然后如果有任何片段的采样位置被覆盖，则调节采样掩膜以表明整个像素被覆盖，来实现保守光栅化。立体像素化 (voxelization) 和碰撞检测算法可以受益于保守光栅化的“stencil then cover”应用程序。

[0105] 上面描述的使用情况提供了具体算法的实例，其可以利用由片段着色器所提供的采样掩膜的控制。然而，上面描述的概念的多种使用情况和应用程序应考虑为处于本公开的范围内。

[0106] 图 9 阐述了一种示范性系统 900，前面的多个实施例的多种结构和 / 或功能可以在其中实现。如所示，提供的系统 900 包括至少一个中央处理器 901，其与通信总线 902 连接。通信总线 902 可以利用合适的协议实现，例如 PCI（外设部件互连标准），PCI-Express，AGP（图形加速接口），HyperTransport，或任何其他总线，或点对点通信协议。系统 900 也包括主存储器 904。控制逻辑（软件）和数据储存在采用随机存取存储器 (RAM) 形式的主存储器 904 中。

[0107] 系统 900 还包括输入设备 912、图形处理器 906 和显示器 908，即传统 CRT（阴极射线管），LCD（液晶显示器）、LED（发光二极管）、等离子体显示器等。从输入设备 912 例如键盘、鼠标、触控板、话筒等接收用户输入。在一个实施例中，图形处理器 906 可以包括多个着色器模块、光栅化模块等。每个前述模块可以正好位于单半导体平台上，以形成图形处理器 (GPU)。

[0108] 在本发明的说明中，单个半导体平台可以涉及专有的整体半导体集成电路或芯片。应注意，术语单半导体平台也可以涉及具有模拟片内操作的增强的连通性的多片模块，并且相比利用通常的中央处理器 (CPU) 和总线实现方式具有实质性改进。当然，多种模块也可以单独安置或按照使用者的愿望置于半导体平台的多种组合中。

[0109] 系统 900 也可以包括二级存储器 910。二级存储器 910 包括，例如硬盘驱动和 / 或代表软盘驱动的可移除存储驱动、磁带驱动、光盘驱动、数字多功能盘 (DVD 驱动）、记录驱动、通用串行总线 (USB) 闪存。可移除存储驱动以公知方式从可移除存储单元读取和 / 或写入可移除存储单元。

[0110] 计算机程序或计算机控制逻辑算法可以存储在主存储器 904 和 / 或二级存储器 910 中。当执行这样的计算机程序时，其使得系统 900 执行多种功能。存储器 904、二级存储器 910 和 / 或其他存储器是计算机可读介质的可能例子。

[0111] 在一个实施例中，多种前面的图的结构和 / 或功能可以在中央处理器 901、图形处理器 906、集成电路（未示出），能够具有中央处理器 901 和图形处理器 906 二者的至少部分能力的集成电路（未示出）、芯片集（即设计为作为用于实施相关功能的单元而工作和销售的集成电路等）和 / 或任何其他用于此的集成电路的环境中实现。

[0112] 然而，多种前面图的结构和功能可以在通用计算机系统、电路板系统、用于娱乐目的的游戏控制系统、应用特殊系统和 / 或其他想要的系统下实现。例如，系统 900 可以表现为台式计算机、笔记本电脑、服务器、工作站、游戏控制台、嵌入式系统和 / 或任何其他类

型的逻辑的形式。然而，系统 900 可以表现为多种其他设备，包括但不限于个人数字助理(PDA) 设备、移动电话设备，电视机等形式。

[0113] 而且，然而未示出，系统 900 可以为了通信目的耦合至网络（例如，电信网、局域网(LAN)、无线网、广域网 (WAN) 如因特网、点对点网络、电缆网等）。

[0114] 然而，上面描述了多个实施例，应理解为它们仅作为例子被介绍，并不是限制。因此，首选实施例的宽度和范围不应被上面描述的任意示范性实施例限制，而仅应该被限制为与权利要求和等价物一致的范围。

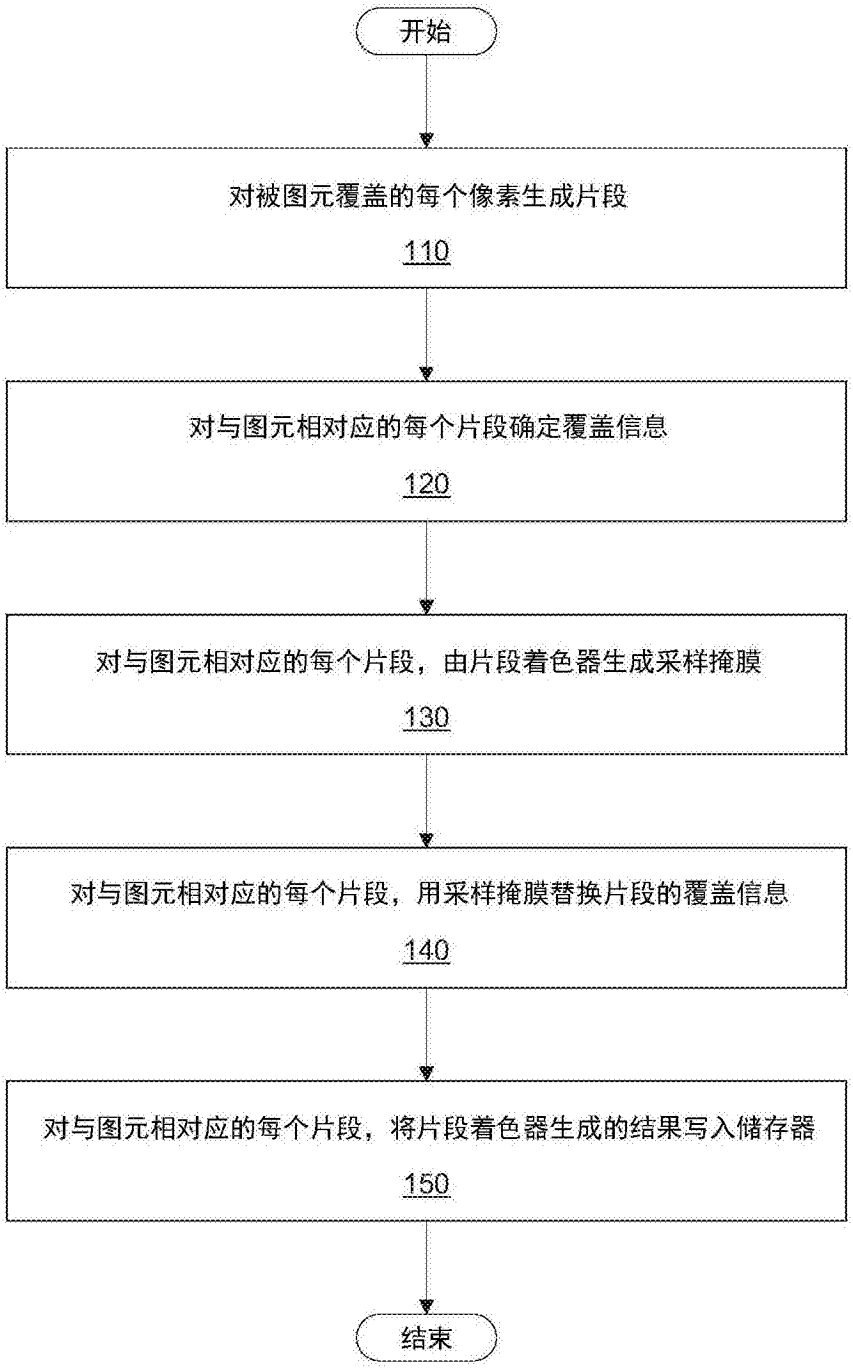

开始
对被图元覆盖的每个像素生成片段
110
对与图元相对应的每个片段确定覆盖信息
120
对与图元相对应的每个片段，由片段着色器生成采样掩膜
130
对与图元相对应的每个片段，用采样掩膜替换片段的覆盖信息
140
对与图元相对应的每个片段，将片段着色器生成的结果写入储存器
150
结束

图 1

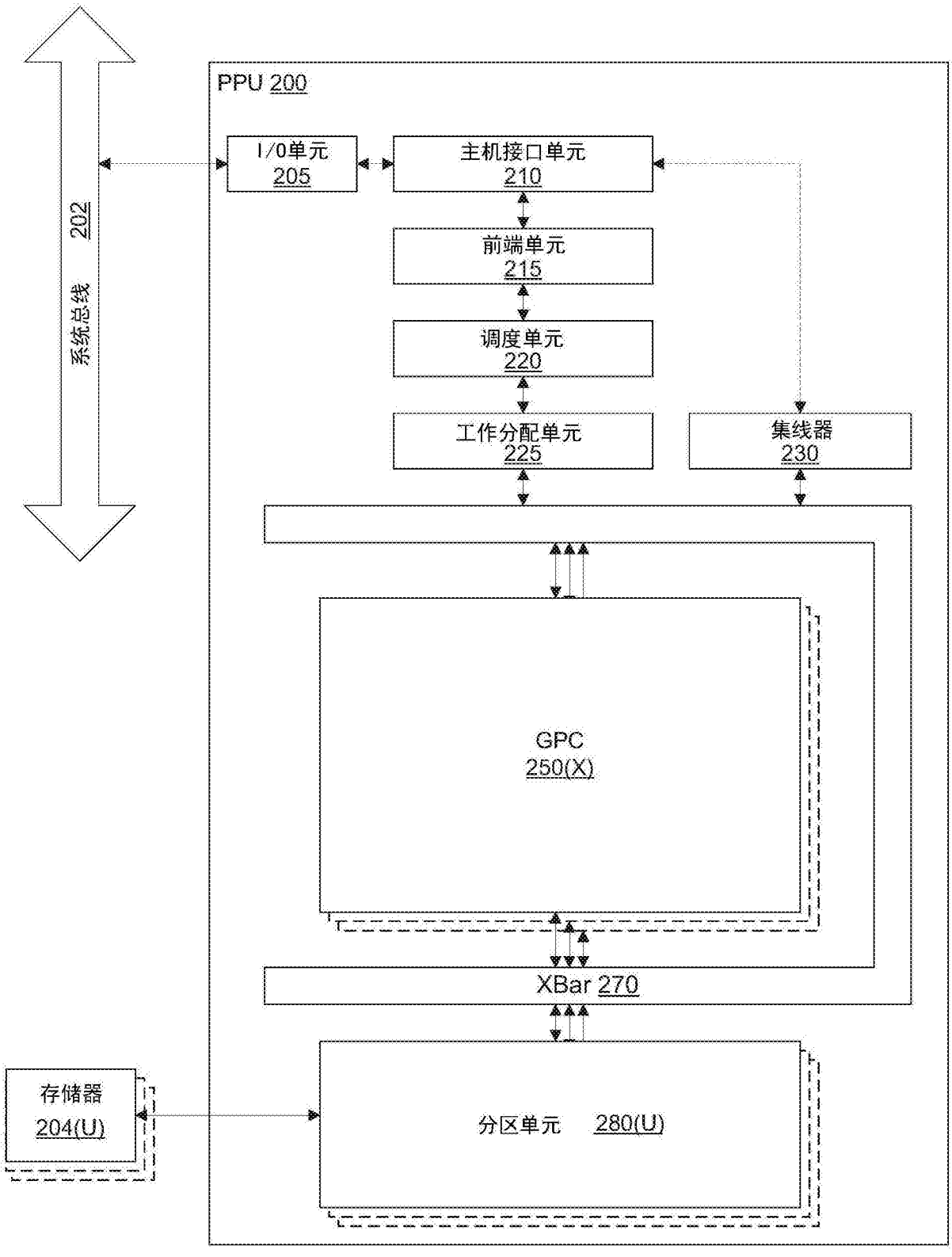

系统总线 202
PPU 200
I/O单元 205
主机接口单元 210
前端单元 215
调度单元 220
工作分配单元 225
集线器 230
GPC 250(X)
XBar 270
存储器 204(U)
分区单元 280(U)

图 2

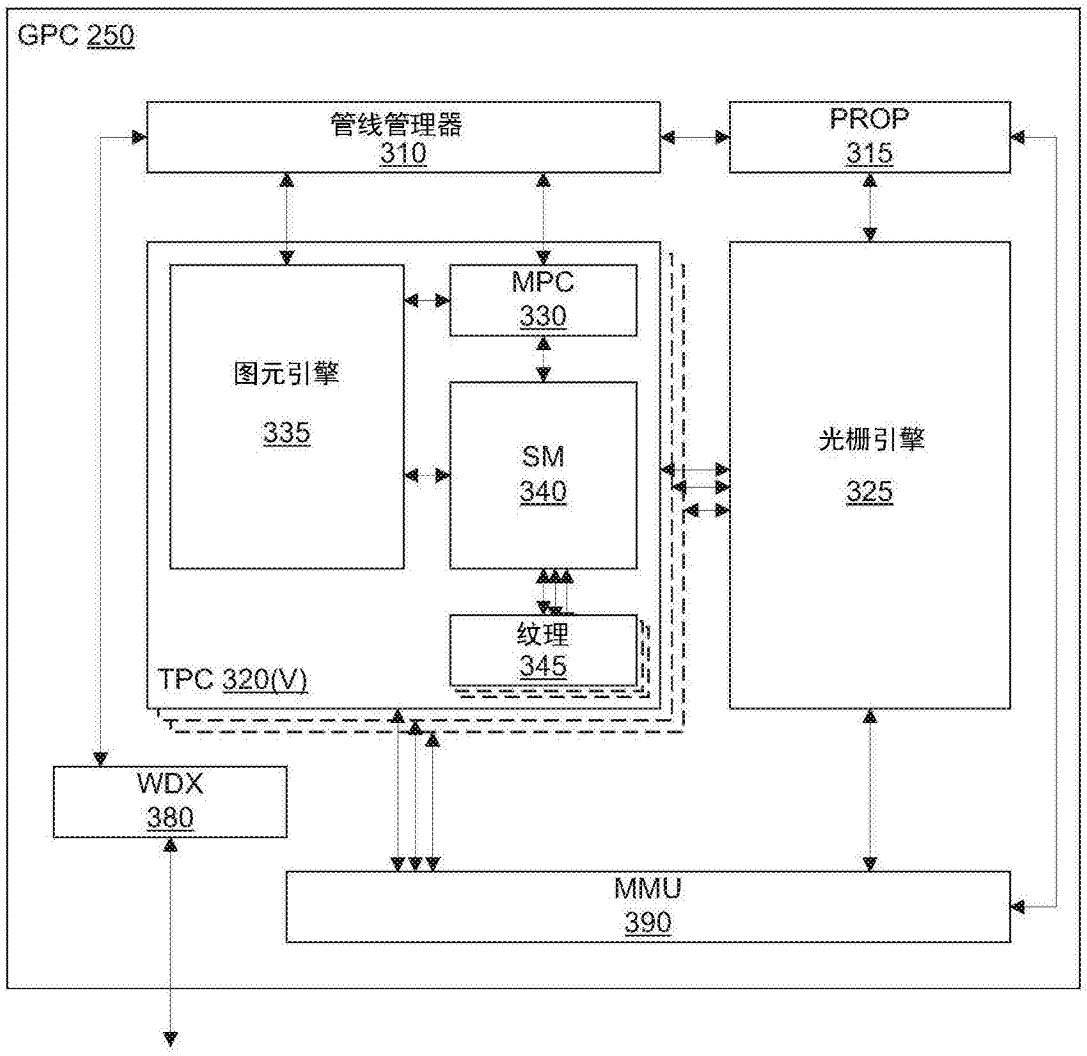

GPC 250
管线管理器
310
PROP
315
MPC
330
图元引擎
335
SM
340
光栅引擎
325
纹理
345
TPC 320(V)
WDX
380
MMU
390

图 3A

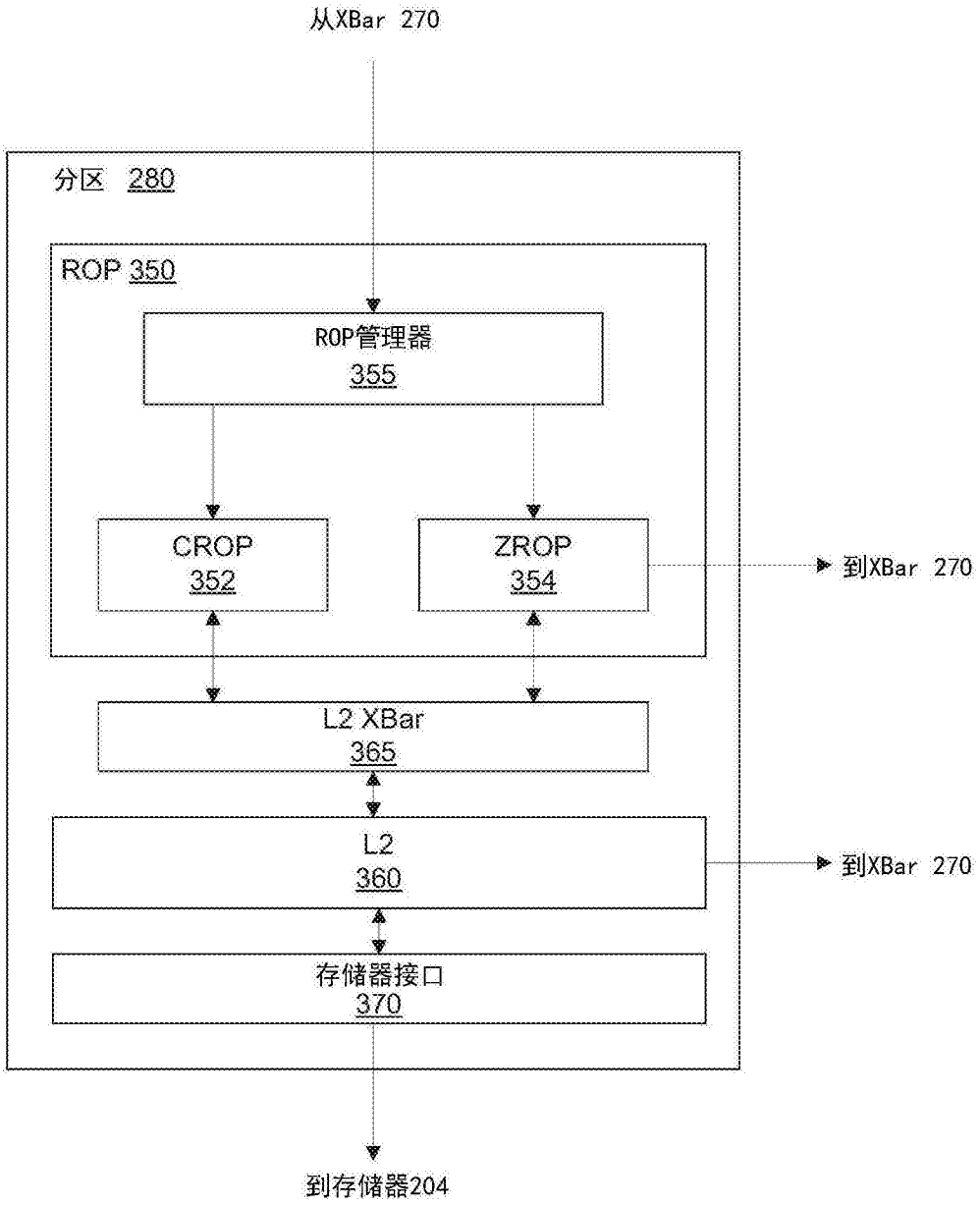

从XBar 270
分区 280
ROP 350
ROP管理器
355
CROP
352
ZROP
354
到XBar 270
L2 XBar
365
L2
360
到XBar 270
存储器接口
370
到存储器204

图 3B

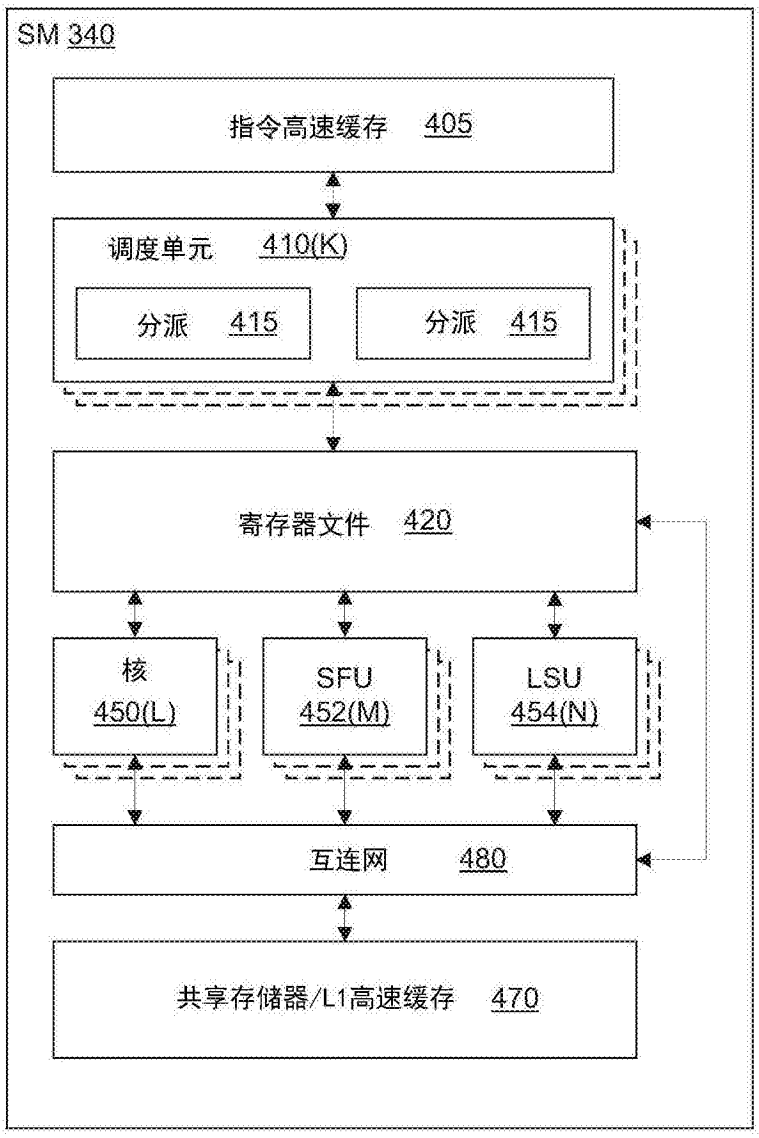

SM 340
指令高速缓存 405
调度单元 410(K)
分派 415
分派 415
寄存器文件 420
核 450(L)
SFU 452(M)
LSU 454(N)
互连网 480
共享存储器/L1高速缓存 470

图 4

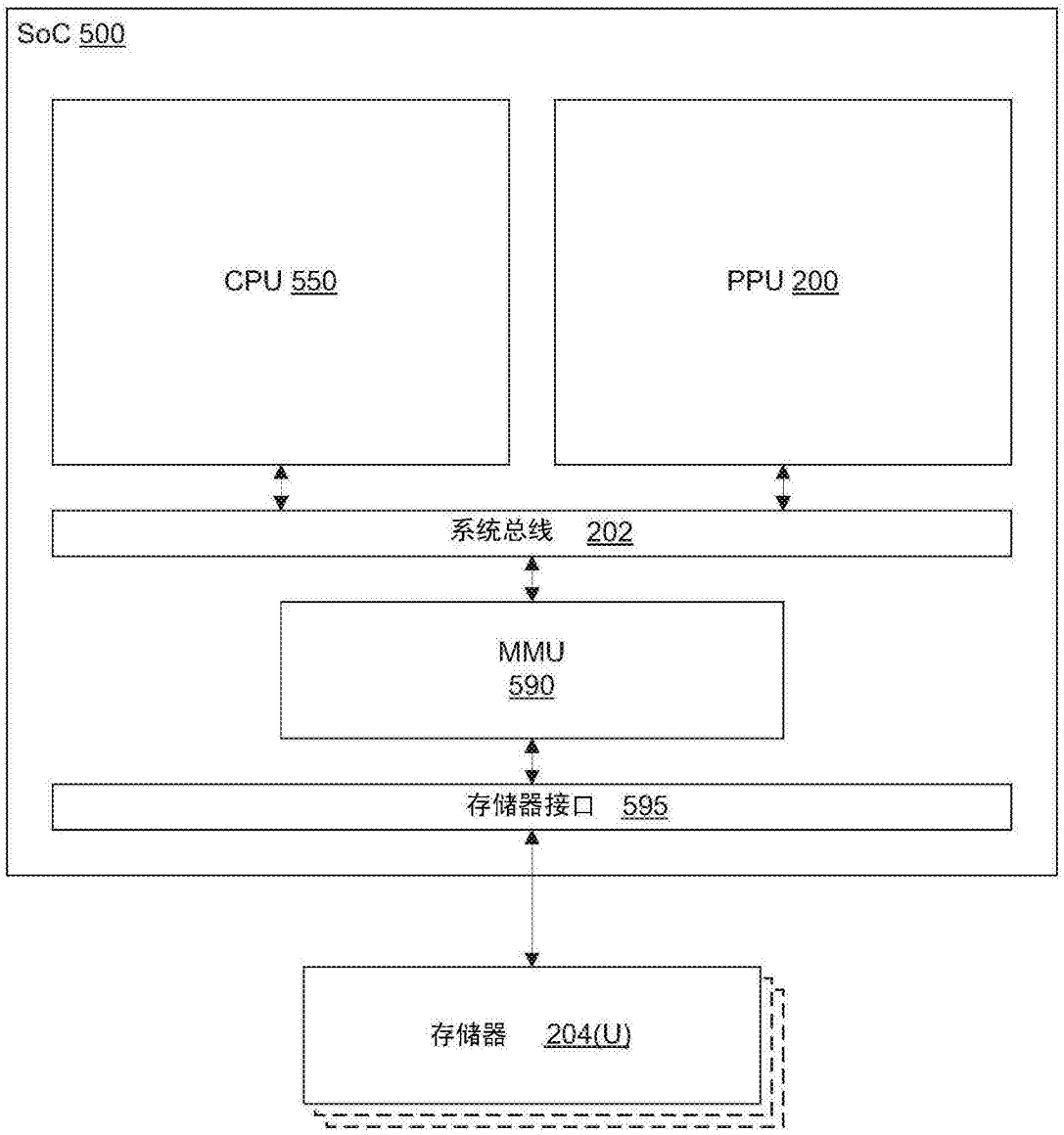


图 5

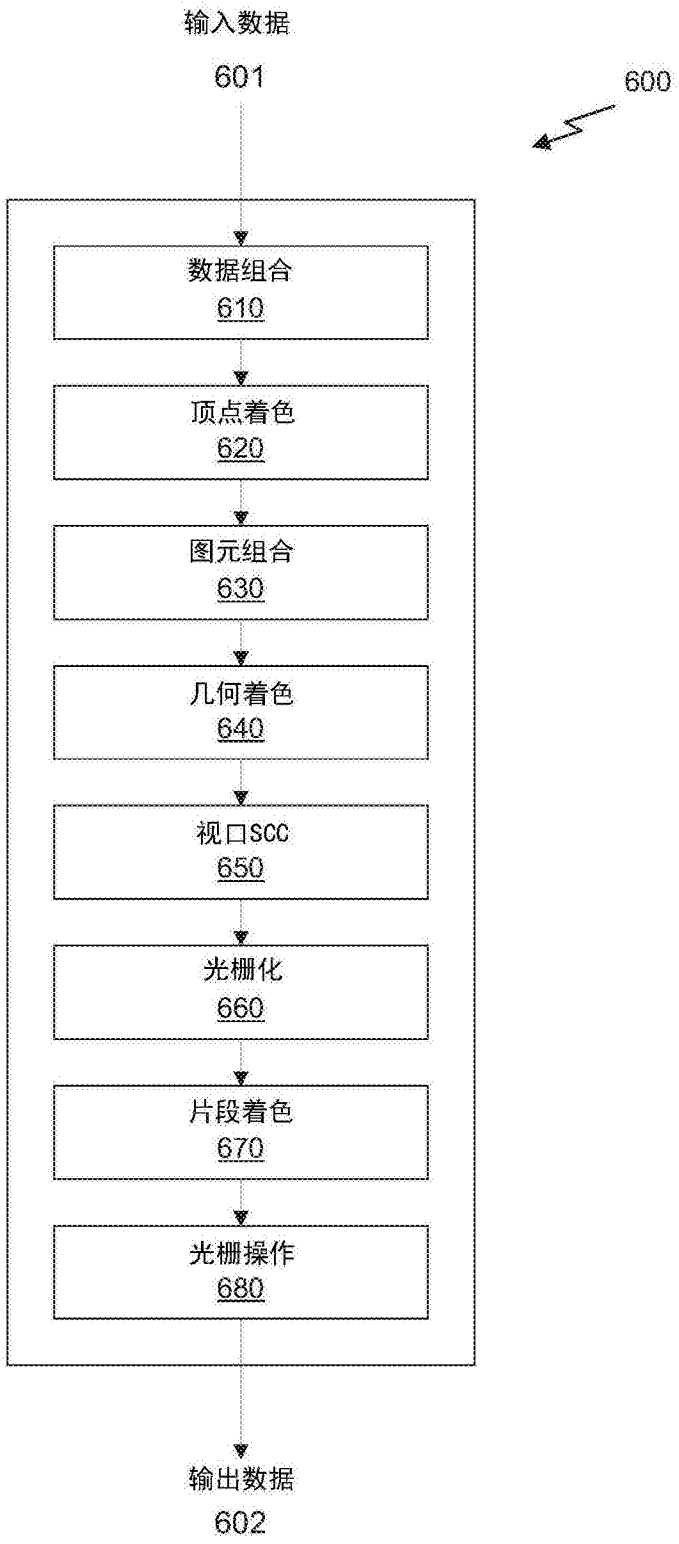

输入数据
601
600
数据组合
610
顶点着色
620
图元组合
630
几何着色
640
视口SCC
650
光栅化
660
片段着色
670
光栅操作
680
输出数据
602

图 6

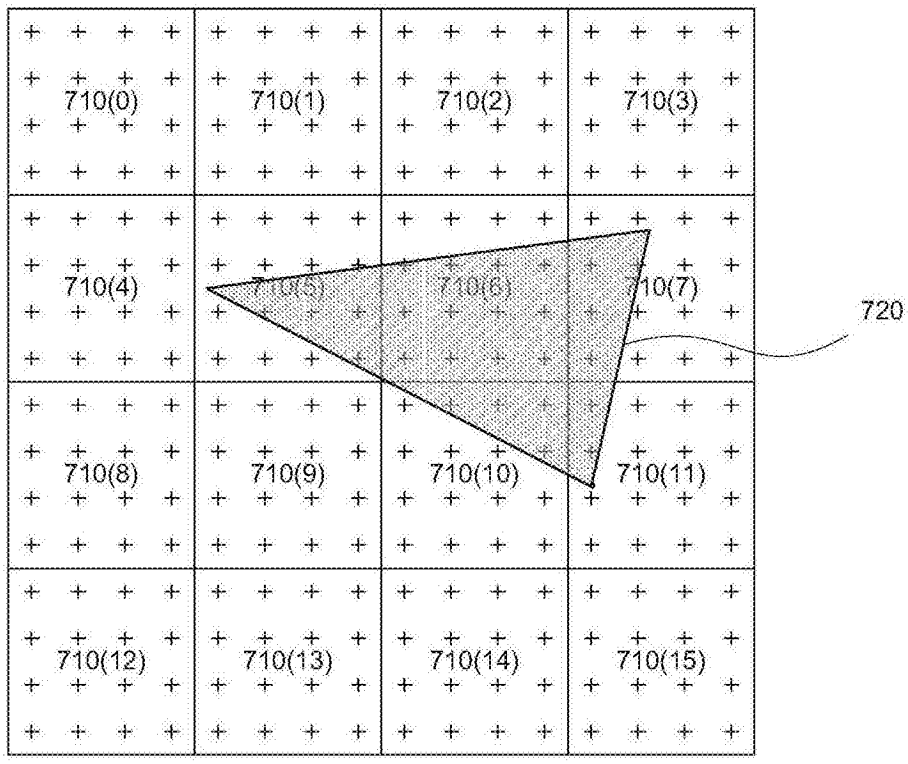
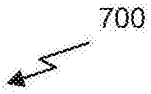


图 7A



0	0	0	0
0	0	0	0
0	1	1	1
0	0	0	1

图 7B

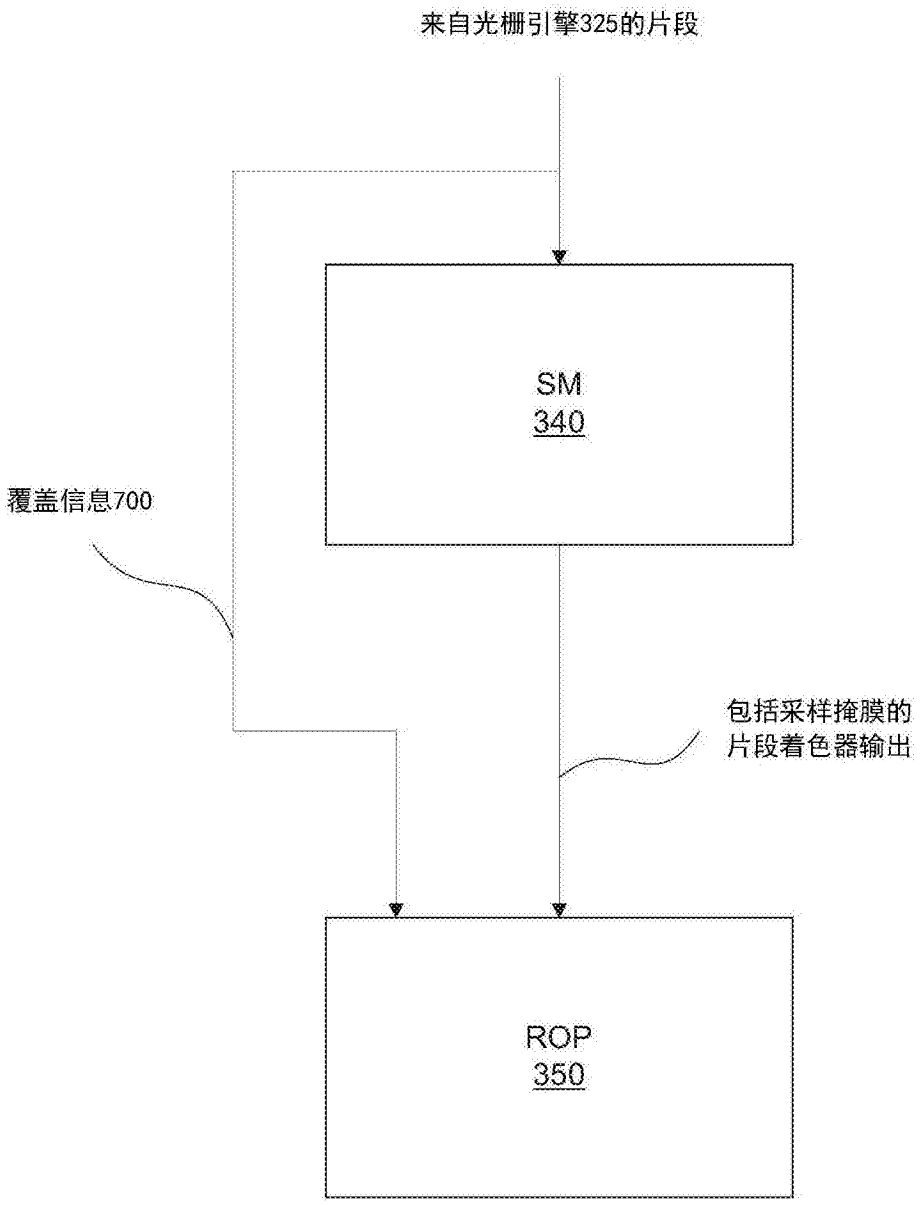


图 8

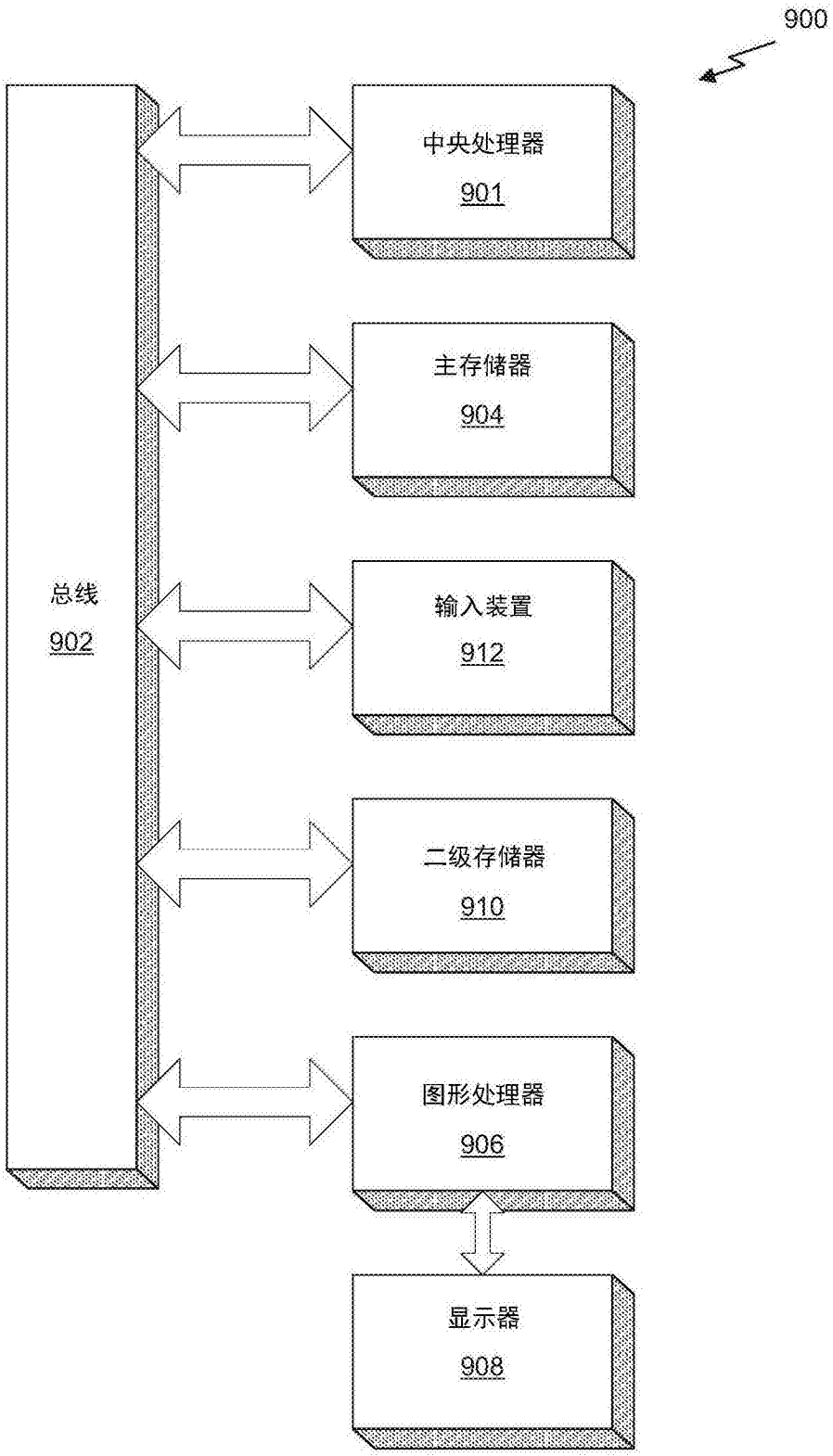

900
中央处理器
901
主存储器
904
总线
902
输入装置
912
二级存储器
910
图形处理器
906
显示器
908

图 9