(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 104246829 B
(45)授权公告日 2018.04.10

(21)申请号 201380018326.0

(22)申请日 2013.03.15

(65)同一申请的已公布的文献号
申请公布号 CN 104246829 A

(43)申请公布日 2014.12.24

(30)优先权数据
61/620,358 2012.04.04 US
61/620,340 2012.04.04 US
61/620,333 2012.04.04 US
13/830,075 2013.03.14 US

(85)PCT国际申请进入国家阶段日
2014.09.30

(86)PCT国际申请的申请数据
PCT/US2013/032123 2013.03.15

(87)PCT国际申请的公布数据
WO2013/151750 EN 2013.10.10

(73)专利权人 高通股份有限公司
地址 美国加利福尼亚州

(72)发明人 维尼特·戈尔
安德鲁·E·格鲁伯 金东炫

(74)专利代理机构 北京律盟知识产权代理有限责任公司 11287
代理人 宋献涛

(51)Int.Cl.
G06T 15/00(2011.01)
G06T 15/80(2011.01)

(56)对比文件
US 2011/0310102 A1,2011.12.22,说明书第0027-0032段、图1-2B.
US 2010/0164954 A1,2010.07.01,说明书第0020-0022段、图1.
US 2010/0079454 A1,2010.04.01,说明书第0004段.
US 2009/0051687 A1,2009.02.26,说明书第0053-0054段.
WO 2009/058845 A1,2009.05.07,全文.
Charles Loop.Hardware Subdivision and Tessellation of Catmull-Clark Surfaces.《Microsoft Research Technical Report, MSR-TR-2010-163》.2010,1-15.

审查员 李炜豪

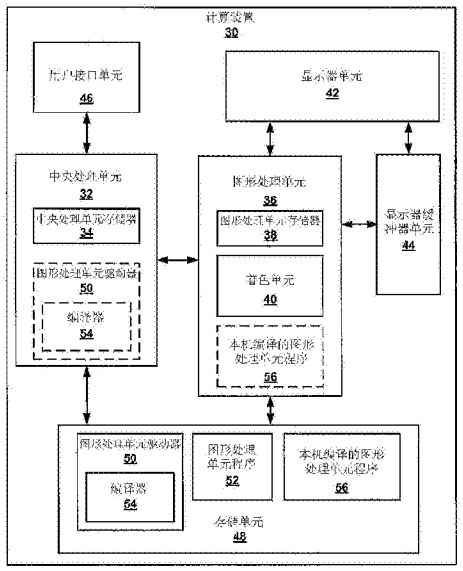
权利要求书4页 说明书38页 附图20页

(54)发明名称

图形处理中的拼补着色

(57)摘要

本发明的方面涉及一种用于渲染图形的过程,其包含:使用图形处理单元GPU的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点,其中所述硬件单元遵照接收单个顶点作为输入并且产生单个顶点作为输出的接口。所述过程还包含使用所述GPU的被指定用于顶点着色的所述硬件单元执行壳体着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个控制点,其中所述一或多个壳体着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个控制点。



CN 104246829 B

1.一种用于渲染图形的方法，所述方法包括：

使用图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元遵照接收单个顶点作为输入并且产生单个顶点作为输出的接口；以及

使用所述图形处理单元的被指定用于顶点着色的所述硬件单元对所述经顶点着色的顶点中的一或多个执行一个或多个曲面细分操作，其中执行所述一个或多个曲面细分操作包括对所述一或多个经顶点着色的顶点中的至少一个执行壳体着色操作以输出一或多个控制点，且其中与所述壳体着色操作相关联的指令被附加到与所述顶点着色操作相关联的指令以使得所述顶点着色操作和所述壳体着色操作循序地执行。

2.根据权利要求1所述的方法，其中执行所述顶点着色操作和执行所述壳体着色操作与执行第一渲染遍次相关联，并且所述方法进一步包括执行包括以下操作的第二渲染遍次：

使用所述图形处理单元的被指定用于顶点着色的所述硬件单元执行包括至少基于所述控制点产生顶点值的域着色操作；以及

使用所述图形处理单元的被指定用于顶点着色的所述硬件单元执行几何形状着色操作以基于经域着色的顶点中的所述一或多个产生一或多个新顶点，其中所述几何形状着色操作对所述一或多个经域着色的顶点中的至少一个进行操作以输出所述一或多个新顶点。

3.根据权利要求2所述的方法，其进一步包括在执行所述第二渲染遍次之前先完成所述第一渲染遍次，使得所述图形处理单元的一或多个组件在所述第一渲染遍次与所述第二渲染遍次之间是闲置的。

4.根据权利要求2所述的方法，其中所述顶点着色操作、所述壳体着色操作、所述域着色操作和所述几何形状着色操作与绘制调用相关联，并且所述方法进一步包括基于所述图形处理单元的存储器的大小将所述绘制调用分成多个子绘制调用，其中所述多个子绘制调用的每一个包括所述第一渲染遍次的操作和所述第二渲染遍次的操作。

5.根据权利要求2所述的方法，其进一步包括将与所述几何形状着色操作相关联的指令附加到与所述域着色操作相关联的指令，使得所述域着色操作和所述几何形状着色操作循序地执行。

6.根据权利要求1所述的方法，其中执行所述壳体着色操作包括：

使用所述图形处理单元的所述硬件单元执行壳体着色器程序的第一例子；

使用所述图形处理单元的所述硬件单元执行所述壳体着色器程序的第二例子；

从所述壳体着色器程序的所述第一例子输出单个控制点，以便遵照所述硬件单元的所述接口；以及

从所述壳体着色器程序的所述第二例子输出第二单个控制点，以便遵照所述硬件单元的所述接口。

7.根据权利要求6所述的方法，其中执行所述壳体着色器程序的所述第一例子包括使用所述图形处理单元的所述硬件单元并行地执行所述壳体着色器程序的所述第一例子和所述壳体着色器程序的所述第二例子。

8.根据权利要求6所述的方法，

其中给所述壳体着色器程序的所述第一例子指派第一壳体着色器输出识别符，

其中给所述壳体着色器程序的所述第二例子指派第二壳体着色器输出识别符，

其中输出所述单个控制点包括基于所述第一壳体着色器输出识别符与第一控制点识别符的比较输出所述单个控制点，并且

其中输出所述第二单个控制点包括基于所述第二壳体着色器输出识别符与第二控制点识别符的比较输出所述第二单个控制点。

9.根据权利要求1所述的方法，其进一步包括在执行所述壳体着色操作之前切换用于所述壳体着色操作的程序计数器与一或多个资源指针。

10.一种用于渲染图形的图形处理单元，其包括一或多个处理器，所述一或多个处理器经配置以：

使用所述图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元遵照接收单个顶点作为输入并且产生单个顶点作为输出的接口；以及

使用所述图形处理单元的被指定用于顶点着色的所述硬件单元对所述经顶点着色的顶点中的一或多个执行一个或多个曲面细分操作，其中执行所述一个或多个曲面细分操作包括对所述一或多个经顶点着色的顶点中的至少一个执行壳体着色操作以输出一或多个控制点，且其中与所述壳体着色操作相关联的指令被附加到与所述顶点着色操作相关联的指令以使得所述顶点着色操作和所述壳体着色操作循序地执行。

11.根据权利要求10所述的图形处理单元，其中所述顶点着色操作和所述壳体着色操作与第一渲染遍次相关联，并且其中所述一或多个处理器进一步经配置以执行第二渲染遍次，其中所述一或多个处理器经配置以：

使用所述图形处理单元的被指定用于顶点着色的所述硬件单元执行包括至少基于所述控制点产生顶点值的域着色操作；以及

使用所述图形处理单元的被指定用于顶点着色的所述硬件单元执行几何形状着色操作以基于经域着色的顶点中的所述一或多个产生一或多个新顶点，其中所述几何形状着色操作对所述一或多个经域着色的顶点中的至少一个进行操作以输出所述一或多个新顶点。

12.根据权利要求11所述的图形处理单元，其中所述一或多个处理器进一步经配置以在执行所述第二渲染遍次之前先完成所述第一渲染遍次，使得所述图形处理单元的一或多个组件在所述第一渲染遍次与所述第二渲染遍次之间是闲置的。

13.根据权利要求11所述的图形处理单元，其中所述顶点着色操作、所述壳体着色操作、所述域着色操作和所述几何形状着色操作与绘制调用相关联，并且其中所述一或多个处理器进一步经配置以基于所述图形处理单元的存储器的大小将所述绘制调用分成多个子绘制调用，其中所述多个子绘制调用的每一个包括所述第一渲染遍次的操作和所述第二渲染遍次的操作。

14.根据权利要求11所述的图形处理单元，其中所述一或多个处理器进一步经配置以将与所述几何形状着色操作相关联的指令附加到与所述域着色操作相关联的指令，使得所述域着色操作和所述几何形状着色操作循序地执行。

15.根据权利要求10所述的图形处理单元，其中为了执行所述壳体着色操作，所述硬件单元经配置以：

使用所述图形处理单元的所述硬件单元执行壳体着色器程序的第一例子；

使用所述图形处理单元的所述硬件单元执行所述壳体着色器程序的第二例子；

从所述壳体着色器程序的所述第一例子输出单个控制点，以便遵照所述硬件单元的所述接口；以及

从所述壳体着色器程序的所述第二例子输出第二单个控制点，以便遵照所述硬件单元的所述接口。

16.根据权利要求15所述的图形处理单元，其中所述硬件单元经配置以使用所述图形处理单元的所述硬件单元并行地执行所述壳体着色器程序的所述第一例子和所述壳体着色器程序的所述第二例子。

17.根据权利要求15所述的图形处理单元，

其中给所述壳体着色器程序的所述第一例子指派第一壳体着色器输出识别符，

其中给所述壳体着色器程序的所述第二例子指派第二壳体着色器输出识别符，

其中为了输出所述单个控制点，所述硬件单元经配置以基于所述第一壳体着色器输出识别符与第一控制点识别符的比较输出所述单个控制点；并且

其中为了输出所述第二单个控制点，所述硬件单元经配置以基于所述第二壳体着色器输出识别符与第二控制点识别符的比较输出所述第二单个控制点。

18.根据权利要求10所述的图形处理单元，其中所述一或多个处理器经配置以在执行所述壳体着色操作之前，切换用于所述壳体着色操作的程序计数器与一或多个资源指针。

19.一种用于渲染图形的设备，所述设备包括：

用于使用图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点的装置，其中所述硬件单元遵照接收单个顶点作为输入并且产生单个顶点作为输出的接口；

用于使用所述图形处理单元的被指定用于顶点着色的所述硬件单元对所述经顶点着色的顶点中的一或多个执行一个或多个曲面细分操作的装置，其中用于执行所述一个或多个曲面细分操作的装置包括用于对所述一或多个经顶点着色的顶点中的至少一个执行壳体着色操作以输出一或多个控制点的装置；以及

用于将与所述壳体着色操作相关联的指令附加到与所述顶点着色操作相关联的指令的装置，使得所述顶点着色操作和所述壳体着色操作循序地执行。

20.根据权利要求19所述的设备，其中用于执行所述顶点着色操作的所述装置和用于执行所述壳体着色操作的所述装置与用于执行第一渲染遍次的装置相关联，并且所述设备进一步包括用于执行第二渲染遍次的装置，其包括：

用于使用所述图形处理单元的被指定用于顶点着色的所述硬件单元执行包括至少基于所述控制点产生顶点值的域着色操作的装置；以及

用于使用所述图形处理单元的被指定用于顶点着色的所述硬件单元执行几何形状着色操作以基于经域着色的顶点中的所述一或多个产生一或多个新顶点的装置，

其中所述几何形状着色操作对所述一或多个经域着色的顶点中的至少一个进行操作以输出所述一或多个新顶点。

21.根据权利要求20所述的设备，其进一步包括用于在执行所述第二渲染遍次之前先完成所述第一渲染遍次的装置，使得所述图形处理单元的一或多个组件在所述第一渲染遍次与所述第二渲染遍次之间是闲置的。

22.根据权利要求20所述的设备，其中所述顶点着色操作、所述壳体着色操作、所述域着色操作和所述几何形状着色操作与绘制调用相关联，并且所述设备进一步包括用于基于所述图形处理单元的存储器的大小将所述绘制调用分成多个子绘制调用的装置，其中所述多个子绘制调用的每一个包括所述第一渲染遍次的操作和所述第二渲染遍次的操作。

23.根据权利要求20所述的设备，其进一步包括用于将与所述几何形状着色操作相关联的指令附加到与所述域着色操作相关联的指令的装置，使得所述域着色操作和所述几何形状着色操作循序地执行。

24.根据权利要求19所述的设备，其中用于执行所述壳体着色操作的所述装置包括：

用于使用所述图形处理单元的所述硬件单元执行壳体着色器程序的第一例子的装置；

用于使用所述图形处理单元的所述硬件单元执行所述壳体着色器程序的第二例子的装置；

用于从所述壳体着色器程序的所述第一例子输出单个控制点以便遵照所述硬件单元的所述接口的装置；以及

用于从所述壳体着色器程序的所述第二例子输出第二单个控制点以便遵照所述硬件单元的所述接口的装置。

25.根据权利要求24所述的设备，其中用于执行所述壳体着色器程序的所述第一例子的装置包括用于使用所述图形处理单元的所述硬件单元并行地执行所述壳体着色器程序的所述第一例子和所述壳体着色器程序的所述第二例子的装置。

26.根据权利要求24所述的设备，

其中给所述壳体着色器程序的所述第一例子指派第一壳体着色器输出识别符，

其中给所述壳体着色器程序的所述第二例子指派第二壳体着色器输出识别符，

其中用于输出所述单个控制点的所述装置包括用于基于所述第一壳体着色器输出识别符与第一控制点识别符的比较输出所述单个控制点的装置，并且

其中用于输出所述第二单个控制点的所述装置包括用于基于所述第二壳体着色器输出识别符与第二控制点识别符的比较输出所述第二单个控制点的装置。

27.根据权利要求19所述的设备，其进一步包括用于在执行所述壳体着色操作之前切换用于所述壳体着色操作的程序计数器与一或多个资源指针的装置。

图形处理中的拼补着色

[0001] 本申请案要求2012年4月4日申请的第61/620,340号美国临时申请案、2012年4月4日申请的第61/620,358号美国临时申请案和2012年4月4日申请的第61/620,333号美国临时申请案的权益，所有前述申请案的完整内容都以引用的方式并入本文中。

技术领域

[0002] 本发明涉及计算机图形。

背景技术

[0003] 为视觉呈现提供内容的装置通常包含图形处理单元(GPU)。GPU在显示器上渲染表示所述内容的像素。GPU为显示器上的每一像素产生一或多个像素值，以便渲染每一像素以供呈现。

[0004] 在一些例子中，GPU可以实施统一的着色器架构来渲染图形。在此些例子中，GPU可以配置多个类似的计算单元来执行不同着色操作的管线。所述计算单元可以称为统一着色单元或统一着色器处理器。

发明内容

[0005] 本发明的技术总体上涉及执行与图形渲染管线的着色器级相关联的着色操作。举例来说，图形处理单元(GPU)可以调用一或多个着色单元以执行与图形渲染管线的着色器级相关联的着色操作。根据本发明的方面，所述GPU可以接着使用被指定用于执行所述第一着色操作的着色单元执行与图形渲染管线的第二不同着色器级相关联的着色操作。举例来说，GPU可以在遵照与第一着色器级相关联的输入/输出接口的同时执行与第二级相关联的着色操作。以此方式，GPU可以通过使用相同的着色单元执行多个着色操作而模仿具有更大着色资源的GPU。

[0006] 在一个实例中，本发明的方面涉及一种渲染图形的方法，其包含：使用图形处理单元的被指定用于顶点着色的硬件着色单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元经配置以接收单个顶点作为输入，并且产生单个顶点作为输出；以及使用所述图形处理单元的所述硬件着色单元执行几何形状着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个新顶点，其中所述几何形状着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个新顶点。

[0007] 在另一实例中，本发明的方面涉及一种用于渲染图形的图形处理单元，其包含经配置以进行以下操作的一或多个处理器：使用图形处理单元的被指定用于顶点着色的硬件着色单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元经配置以接收单个顶点作为输入，并且产生单个顶点作为输出；以及使用所述图形处理单元的所述硬件着色单元执行几何形状着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个新顶点，其中所述几何形状着色操作对所述一或多个经顶点着色

的顶点中的至少一个进行操作以输出所述一或多个新顶点。

[0008] 在另一实例中，本发明的方面涉及一种用于渲染图形的设备，其包含：用于使用图形处理单元的被指定用于顶点着色的硬件着色单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点的装置，其中所述硬件单元经配置以接收单个顶点作为输入，并且产生单个顶点作为输出；以及用于使用所述图形处理单元的所述硬件着色单元执行几何形状着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个新顶点的装置，其中所述几何形状着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个新顶点。

[0009] 在另一实例中，本发明的方面涉及一种上面存储有指令的非暂时性计算机可读媒体，所述指令在被执行时使得一或多个处理器使用被指定用于顶点着色的硬件着色单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元经配置以接收单个顶点作为输入并且产生单个顶点作为输出，并且使用被指定用于顶点着色的硬件着色单元执行几何形状着色操作以基于经顶点着色的顶点中的一或多个产生一或多个新顶点，其中所述几何形状着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出一或多个新顶点。

[0010] 在另一实例中，本发明的方面涉及一种用于渲染图形的方法，其包含：使用图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元遵照一个接口，所述接口接收单个顶点作为输入，并且产生单个顶点作为输出；以及使用所述图形处理单元的被指定用于顶点着色的硬件单元执行壳体着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个控制点，其中所述一或多个壳体着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个控制点。

[0011] 在另一实例中，本发明的方面涉及一种用于渲染图形的图形处理单元，其包含一或多个经配置以进行以下操作的处理器：使用图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点，其中所述硬件单元遵照一个接口，所述接口接收单个顶点作为输入，并且产生单个顶点作为输出；以及使用所述图形处理单元的被指定用于顶点着色的硬件单元执行壳体着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个控制点，其中所述一或多个壳体着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个控制点。

[0012] 在另一实例中，本发明的方面涉及一种用于渲染图形的设备，其包含：用于使用图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点的装置，其中所述硬件单元遵照一个接口，所述接口接收单个顶点作为输入，并且产生单个顶点作为输出；以及用于使用所述图形处理单元的被指定用于顶点着色的硬件单元执行壳体着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个控制点的装置，其中所述一或多个壳体着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个控制点。

[0013] 在另一实例中，本发明的方面涉及一种上面存储有指令的非暂时性计算机可读媒体，所述指令当被执行时使得一或多个处理器进行以下操作：使用图形处理单元的被指定用于顶点着色的硬件单元执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色

的顶点，其中所述硬件单元遵照一个接口，所述接口接收单个顶点作为输入，并且产生单个顶点作为输出；以及使用所述图形处理单元的被指定用于顶点着色的硬件单元执行壳体着色操作以基于所述经顶点着色的顶点中的一或多个产生一或多个控制点，其中所述一或多个壳体着色操作对所述一或多个经顶点着色的顶点中的至少一个进行操作以输出所述一或多个控制点。

[0014] 在一个实例中，本发明的方面涉及一种渲染图形的方法，其包含：指定图形处理单元的硬件着色单元执行与渲染管线的第一着色器级相关联的第一着色操作，在完成所述第一着色操作后即刻切换所述硬件着色单元的操作模式，并且使用所述图形处理单元的被指定执行所述第一着色操作的硬件着色单元执行与渲染管线的第二不同着色器级相关联的第二着色操作。

[0015] 在另一实例中，本发明的方面涉及一种用于渲染图形的图形处理单元，其包括经配置以进行以下操作的一或多个处理器：指定图形处理单元的硬件着色单元执行与渲染管线的第一着色器级相关联的第一着色操作，在完成所述第一着色操作后即刻切换所述硬件着色单元的操作模式，并且使用所述图形处理单元的被指定执行所述第一着色操作的硬件着色单元执行与渲染管线的第二不同着色器级相关联的第二着色操作。

[0016] 在另一实例中，本发明的方面涉及一种用于渲染图形的设备，其包含：用于指定图形处理单元的硬件着色单元执行与渲染管线的第一着色器级相关联的第一着色操作的装置，用于在完成所述第一着色操作后即刻切换所述硬件着色单元的操作模式的装置，以及用于使用所述图形处理单元的被指定执行所述第一着色操作的硬件着色单元执行与渲染管线的第二不同着色器级相关联的第二着色操作的装置。

[0017] 在另一实例中，本发明的方面涉及一种上面存储着指令的非暂时性计算机可读媒体，所述指令在被执行时使得一或多个处理器进行以下操作：指定图形处理单元的硬件着色单元执行与渲染管线的第一着色器级相关联的第一着色操作，在完成所述第一着色操作后即刻切换所述硬件着色单元的操作模式，并且使用所述图形处理单元的被指定执行所述第一着色操作的硬件着色单元执行与渲染管线的第二不同着色器级相关联的第二着色操作。

[0018] 附图和下面的描述中阐述了本发明的一或多个实例的细节。通过描述和图式并且通过权利要求将明白其它特征、目的和优点。

附图说明

[0019] 图1是说明可以实施本发明中描述的技术的计算装置的框图。

[0020] 图2是说明示范性图形处理管线80的框图。

[0021] 图3A和3B是根据本发明的方面的图形渲染管线中的数据流的概念图。

[0022] 图4是实施本发明中描述的技术以执行顶点着色操作和几何形状着色操作的硬件着色单元的实例操作的图。

[0023] 图5A说明合并顶点着色器/几何形状着色器硬件着色单元在执行顶点着色操作和几何形状着色操作时执行的操作的流程。

[0024] 图5B说明对应于可以由合并顶点着色器/几何形状着色器硬件着色单元执行的图5A所示的操作流程的伪码。

[0025] 图6是说明根据本发明的方面的用于执行合并顶点着色操作和几何形状着色操作的图形处理单元的实例组件的图。

[0026] 图7是说明根据本发明的方面的用于执行顶点着色操作和几何形状着色操作的实例过程的流程图。

[0027] 图8是说明包含曲面细分器级的实例图形处理管线的框图。

[0028] 图9是说明曲面细分的概念图。

[0029] 图10A和10B是根据本发明的方面的图形渲染管线中的数据流的概念图。

[0030] 图11是实施本发明中描述的技术以执行顶点着色和壳体着色操作的硬件着色单元的实例操作的图。

[0031] 图12A说明合并顶点着色器/壳体着色器硬件着色单元在执行顶点着色操作和壳体着色操作时执行的操作的流程。

[0032] 图12B总体上说明对应于可以由合并顶点着色器/壳体着色器硬件着色单元执行的图12A所示的操作流程的伪码。

[0033] 图13A总体上说明合并域着色器/几何形状着色器硬件着色单元在执行域着色操作和几何形状着色操作时执行的操作的流程。

[0034] 图13B总体上说明对应于可以由合并域着色器/几何形状着色器硬件着色单元执行的图13A所示的操作流程的伪码。

[0035] 图14是说明根据本发明的方面的用于执行合并顶点着色、壳体着色、域着色和几何形状着色操作的图形处理单元的实例组件的图。

[0036] 图15是说明根据本发明的方面使用相同硬件着色单元在两个渲染遍次中执行图形渲染的流程图。

[0037] 图16是说明根据本发明的方面执行与两个遍次的图形渲染过程中的第一遍次相关联的图形渲染操作的流程图。

[0038] 图17是说明根据本发明的方面执行与两个遍次的图形渲染过程中的第二遍次相关联的图形渲染操作的流程图。

[0039] 图18是说明根据本发明的方面将一个以上着色器级拼补在一起以供相同硬件着色单元执行的流程图。

具体实施方式

[0040] 本发明的技术总体上涉及执行与图形渲染管线的着色器级相关联的着色操作。举例来说，图形处理单元(GPU)可以调用一或多个着色单元以执行与图形渲染管线的着色器级相关联的着色操作。根据本发明的方面，GPU可以接着使用被指定用于执行第一着色操作的着色单元来执行与图形渲染管线的第二不同着色器级相关联的着色操作。举例来说，GPU可以在遵照与第一着色器级相关联的输入/输出接口的同时执行与第二级相关联的着色操作。以此方式，GPU可以通过用相同的着色单元执行多个着色操作而模仿具有更大着色资源的GPU。

[0041] 图1是说明可以实施本发明中描述的技术的计算装置30的框图。计算装置30的实例包含但不限于无线装置、移动或蜂窝电话(包含所谓的智能手机)、个人数字助理(PDA)、包含视频显示器的视频游戏控制台、移动视频游戏装置、移动视频会议单元、膝上型计算

机、台式计算机、电视机顶盒、平板计算装置、电子书阅读器、固定或移动媒体播放器等等。

[0042] 在图1的实例中，计算装置30包含具有CPU存储器34的中央处理单元(CPU)32、具有GPU存储器38和一或多个着色单元40的图形处理单元(GPU)36、显示器单元42、显示器缓冲器单元44、用户接口单元46和存储单元48。此外，存储单元48可以存储具有编译器54的GPU驱动器50、GPU程序52和本机编译的GPU程序56。

[0043] CPU 32的实例包含但不限于数字信号处理器(DSP)、通用微处理器、专用集成电路(ASIC)、现场可编程逻辑阵列(FPGA)或其它等效的集成或离散逻辑电路。虽然CPU 32和GPU 36在图1的实例中被说明成分开的单元，但是在一些实例中，CPU 32和GPU 36可以集成为单个单元。CPU 32可以执行一或多个应用程序。应用程序的实例可以包含网络浏览器、电子邮件应用程序、电子表格、视频游戏、音频和/或视频捕获、回放或编辑应用程序或其它起始有待经由显示器单元42呈现的图像数据的产生的应用程序。

[0044] 在图1所示的实例中，CPU 32包含CPU存储器34。CPU存储器34可以表示在执行机器或对象代码时使用的芯片上存储设备或存储器。CPU存储器34可以各自包括能够存储固定数目个数字位的硬件存储器寄存器。CPU 32可以能够比从存储单元48(其可例如经由系统总线存取)读取值或者向存储单元48写入值更迅速地从本机CPU存储器34读取值或者向本机CPU存储器34写入值。

[0045] GPU 36表示用于执行图形操作的一或多个专用处理器。也就是说，举例来说，GPU 36可以是具有固定功能和用于渲染图形和执行GPU应用程序的可编程组件的专用硬件单元。GPU 36还可包含DSP、通用微处理器、ASIC、FPGA或其它等效的集成或离散逻辑电路。

[0046] GPU 36还包含GPU存储器38，其可以表示在执行机器或对象代码时使用的芯片上存储设备或存储器。GPU存储器38可以各自包括能够存储固定数目个数字位的硬件存储器寄存器。GPU 36可以能够比从存储单元48(其可例如经由系统总线存取)读取值或者向存储单元48写入值更迅速地从本机GPU存储器38读取值或者向本机GPU存储器38写入值。

[0047] GPU 36还包含着色单元40。如下文更详细地描述，着色单元40可以配置成处理组件的可编程管线。在一些实例中，着色单元40可以称为“着色器处理器”或“统一着色器”，并且可以执行几何形状、顶点、像素或其它着色操作以渲染图形。着色单元40可以包含图1中为了清晰起见未具体展示的一或多个组件，例如用于取出和解码指令的组件、用于实行算术计算的一或多个算术逻辑单元(“ALU”)和一或多个存储器、高速缓存或寄存器。

[0048] 显示器单元42表示能够显示视频数据、图像、文本或任何其它类型的数据以供观看者消费的单元。显示器单元42可以包含液晶显示器(LCD)、发光二极管(LED)显示器、有机LED(OLED)、有源矩阵OLED(AMOLED)显示器等等。

[0049] 显示器缓冲器单元44表示专用于为显示器单元42存储数据以供呈现图像(例如照片或视频帧)的存储器或存储装置。显示器缓冲器单元44可以表示包含多个存储位置的二维缓冲器。显示器缓冲器单元44内的存储位置的数目可以基本上类似于有待在显示器单元42上显示的像素的数目。举例来说，如果显示器单元42经配置以包含640x480个像素，那么显示器缓冲器单元44可以包含640x480个存储位置。显示器缓冲器单元44可以存储由GPU 36处理的像素中的每一个的最终像素值。显示器单元42可以从显示器缓冲器单元44检索最终像素值，并且基于显示器缓冲器单元44中存储的像素值显示最终图像。

[0050] 用户接口单元46表示用户可以用来与计算装置30的其它单元(例如，CPU 32)交互

或者以其它方式介接以与计算装置30的其它单元通信的单元。用户接口单元46的实例包含但不限于轨迹球、鼠标、键盘和其它类型的输入装置。用户接口单元46还可以是触摸屏，并且可以并入为显示器单元42的一部分。

[0051] 存储单元48可以包括一或多个计算机可读存储媒体。存储单元48的实例包含但不限于随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、CD-ROM或其它光盘存储装置、磁盘存储装置或其它磁性存储装置、快闪存储器或可以用于以指令或数据结构的形式存储期望的程序代码并且可以由计算机或处理器存取的任何其它媒体。

[0052] 在一些实例实施方案中，存储单元48可以包含使得CPU 32和/或GPU 36执行本发明中归于CPU 32和GPU 36的功能的指令。在一些实例中，存储单元48可以被视为非暂时性存储媒体。术语“非暂时性”可以指示存储媒体不是体现在载波或传播信号中。然而，术语“非暂时性”不应解释为意味着存储单元48是不能移动的。作为一个实例，存储单元48可以从计算装置30中移除，并且移动到另一装置。作为另一实例，基本上类似于存储单元48的存储单元可以插入到计算装置30中。在某些实例中，非暂时性存储媒体可以存储可能随时间而改变的数据(例如，在RAM中)。

[0053] 如图2的实例中所说明，存储单元48存储GPU驱动器50和编译器54、GPU程序52和本机编译的GPU程序56。GPU驱动器50表示提供存取GPU 36的接口的计算机程序或可执行代码。CPU 32执行GPU驱动器50或其若干部分以与GPU 36介接，并且出于此原因，GPU驱动器50在图1的实例中展示为CPU 32内的用虚线框标记的“GPU驱动器50”。GPU驱动器50可以存取CPU 32执行的程序或其它可执行文件，包含GPU程序52。

[0054] GPU程序52可以包含(例如，使用应用程序编程接口(API))用高级(HL)编程语言编写的代码。API的实例包含微软公司开发的开放计算语言(“OpenCL”)、开放图形库(“OpenGL”)和DirectX。总地来说，API包含由相关联的硬件执行的预定的标准化的成组命令。API命令允许用户指令GPU的硬件组件执行命令，而无需用户知道硬件组件的具体情况。

[0055] GPU程序52可以调用或者以其它方式包含GPU驱动器50提供的一或多个功能。CPU 32总体上执行其中嵌入着GPU程序52的程序，并且在遇到GPU程序52后，即刻将GPU程序52传递给GPU驱动器50(例如，以命令流的形式)。CPU 32在这个上下文中执行GPU驱动器50以处理GPU程序52。也就是说，举例来说，GPU驱动器50可以通过将GPU程序52编译成GPU 36可执行的对象或机器代码而处理GPU程序52。这个对象代码在图1的实例中展示为本机编译的GPU程序56。

[0056] 在一些实例中，编译器54可以实时或近实时地操作，以在执行其中嵌入着GPU程序52的程序期间编译GPU程序52。举例来说，编译器54总体上表示将根据HL编程语言定义的HL指令精简成低级(LL)编程语言的LL指令的模块。在编译之后，这些LL指令能够由特定类型的处理器或其它类型的硬件(例如FPGA、ASIC等等(包含例如CPU 32和GPU 36)来执行。

[0057] 在LL编程语言提供从处理器或其它类型的硬件的指令集架构的很少抽象或较低级抽象的意义上，LL编程语言被视为是低级的。LL语言总体上是指汇编和/或机器语言。汇编语言是稍微比机器语言高的LL语言，但是总体上无需使用编译器或其它翻译模块就可以将汇编语言转换成机器语言。机器语言表示任何定义与基础硬件(例如，处理器)原生执行的指令(例如，x86机器代码(其中x86是指因特尔公司开发的x86处理器的指令集架构)类似

(如果不是相同)的指令的语言。

[0058] 在任何情况下,编译器54都可以将根据HL编程语言定义的HL指令翻译成基础硬件支持的LL指令。编译器54移除与HL编程语言(和API)相关联的抽象,使得根据这些HL编程语言定义的软件能够被实际基础硬件更直接地执行。

[0059] 在图1的实例中,编译器54可以在执行包含GPU程序52的HL代码时从CPU 32接收GPU程序52。编译器54可以将GPU程序52编译成符合LL编程语言的本机编译的GPU程序56。编译器54接着输出包含LL指令的本机编译的GPU程序56。

[0060] GPU 36总体上接收本机编译的GPU程序56(如通过GPU 36内的虚线框标记的“本机编译的GPU程序56”所展示),在这之后,在一些例子中,GPU 36即刻渲染图像并且将图像的经渲染部分输出到显示器缓冲器单元44。举例来说,GPU 36可以产生有待在显示器单元42处显示的多个基元。基元可以包含一或多条线(包含曲线、样条等)、点、圆、椭圆、多边形(其中通常将多边形定义为一或多个三角形的集合)或任何其它二维(2D)基元。术语“基元”还可以指代三维(3D)基元,例如立方体、圆柱体、球体、圆锥体、金字塔、圆环等等。总地来说,术语“基元”是指任何被GPU 36渲染以供经由显示器单元42作为图像(或在视频数据的上下文中的帧)显示的几何形状或要素。

[0061] GPU 36可以通过应用一或多个模型变换(其也可以在状态数据中指定)将基元或基元的其它状态数据(例如,其定义基元的纹理、亮度、相机配置或其它方面)变换成所谓的“世界空间”。一旦经过变换,GPU 36就可以应用有效相机的视图变换(其同样也可以在定义相机的状态数据中指定)以将基元和光的坐标变换到相机或眼睛空间中。GPU 36还可以执行顶点着色以在任何有效光的视图中渲染基元的外观。GPU 36可以在上述模型、世界或视图空间中的一或多个中执行顶点着色(虽然顶点着色通常是在世界空间中执行的)。

[0062] 一旦基元经过着色,GPU 36就可以执行投影以将图像投影到(作为一个实例)在(-1,-1,-1)和(1,1,1)处具有极点的单位立方体中。这个单位立方体通常称为典型视图体。在将模型从眼睛空间变换到典型视图体之后,GPU 36可以执行裁剪以移除任何不至少部分地驻留在视图体中的基元。换句话说,GPU 36可以移除任何不在相机帧内的基元。GPU 36可以接着将基元的坐标从视图体映射到屏幕空间,从而有效地将基元的3D基元精简成屏幕的2D坐标。

[0063] 在给定用其相关联的着色数据定义基元的经变换和投影的顶点的情况下,GPU 36可以接着使基元光栅化。举例来说,GPU 36可以计算和设置基元所覆盖的屏幕的像素的颜色。在光栅化期间,GPU 36可以应用与基元相关联的任何纹理(其中纹理可以包括状态数据)。GPU 36还可以在光栅化期间执行Z缓冲器算法(也称为深度测试)以确定是否有任何基元和/或对象被任何其它对象遮蔽。Z缓冲器算法根据基元的深度将基元排序,使得GPU 36知道将每一基元绘制到屏幕上时的次序。GPU 36将经渲染的像素输出到显示器缓冲器单元44。

[0064] 显示器缓冲器单元44可以暂时存储经渲染的图像的经渲染的像素,直到整个图像都被渲染了为止。在这个上下文中,可以将显示器缓冲器单元44视为图像帧缓冲器。显示器缓冲器单元44可以接着发射有待在显示器单元42上显示的经渲染的图像。在一些替代的实例中,GPU 36可以将图像的经渲染的部分直接输出到显示器单元42以供显示,而不是将图像暂时存储在显示器缓冲器单元44中。显示器单元42可以接着显示在显示器缓冲器单元78

中存储的图像。

[0065] 为了用上述方式渲染像素，GPU 36可以指定着色单元40执行多种着色操作(如举例来说相对于图2和8更详细地描述)。然而，某些被设计成支持相对更短的渲染管线的GPU(例如GPU 36)可能不能够支持具有扩展渲染管线的API。举例来说，可以防止一些GPU指定着色单元40执行两种以上不同类型的着色操作。

[0066] 在一个实例中，GPU 36可以指定着色单元40执行顶点着色和像素着色操作。在这个实例中，GPU 36可能缺乏指定着色单元40执行与壳体着色器、域着色器和/或几何形状着色器相关联的操作的资源。也就是说，硬件和/或软件限制可能会防止GPU 36指定着色单元40执行壳体着色、域着色和/或几何形状着色操作。因此，GPU 36可能不能够支持与包含此功能性的API相关联的着色器级。

[0067] 举例来说，支持先前DirectX 9API(由微软开发，可包含Direct3D 9API)的前代GPU可能不能够支持DirectX 10 API(其可包含Direct3D 10 API)。也就是说，使用前代GPU可能不能够执行DirectX 10 API的特征中的至少一些(例如某些着色器级)。此外，支持先前DirectX 9API和DirectX 10 API的GPU可能不能够支持DirectX 11 API的所有特征。此些不兼容性可能产生可能不再为执行依靠DirectX 10或DirectX 11的软件或其它应用程序提供支持的大量当前部署的GPU。虽然上述实例是相对于微软的DirectX族的API描述的，但是其它API和旧式GPU 36也可能存在类似的兼容性问题。

[0068] 此外，支持相对更长的图形处理管线(例如，具有额外着色器级的渲染管线)可能必需更加复杂的硬件配置。举例来说，将几何形状着色器级引入到渲染管线以执行几何形状着色(当由专用的一个着色单元40执行时)，可能导致对芯片外存储器的额外读取和写入。也就是说，GPU 36可以起初用着色单元40中的一个执行顶点着色，并且将顶点存储到存储单元48。GPU 36还可以读取顶点着色器输出的顶点，并且写入当通过着色单元40中的一个执行几何形状着色时产生的新顶点。如下所述，给渲染管线包含曲面细分级(例如，壳体着色器级和域着色器级)可能会引入类似的复杂性。

[0069] 对芯片外存储器的额外读取和写入可能会消耗存储器总线带宽(例如，将GPU 36连接到存储单元48的通信信道)，同时还潜在地增加了所消耗的功率量(考虑到每次读取和写入都必需给存储器总线和存储单元48供电)。在这个意义上，对每一着色器级使用专用着色单元40来实施具有许多级的图形管线，可能会导致功率效率较低的GPU。此外，由于在从存储单元48检索数据时的延迟，此些GPU 36在输出经渲染的图像方面也可能执行得较慢。

[0070] 本发明的方面总体上涉及合并着色单元40中的一或多个的功能，使得着色单元40中的一个可以执行一种以上着色功能。举例来说，通常GPU 36可以通过指定着色单元40执行特定的着色操作来执行渲染过程(其可以称为具有着色器级的渲染管线)，其中着色单元40中的每一个可以同时实施相同着色器的多个例子。也就是说，GPU 36可以指定着色单元40中的一或多个执行顶点着色操作，例如支持顶点着色器的多达256个并行例子。GPU 36还可以指定着色单元40中的一或多个执行像素着色操作，例如支持像素着色器的多达256个并行例子。这些硬件单元可以将执行三个着色器中的一个所得到的输出存储到芯片外存储器，例如存储单元48，直到下一指定硬件单元可以用于处理图形处理管线中的前一硬件单元的输出为止。

[0071] 虽然本发明的方面可能用单数形式提到特定硬件着色单元(例如，一个硬件着色

单元)，但是应理解，此些单元可以实际上包括一或多个着色单元40(一个以上着色器处理器)，以及GPU 36的用于执行着色操作的一或多个其它组件。举例来说，如上所述，GPU 36可以具有多个相关联的着色单元40。GPU 36可以指定着色单元40中的一个以上执行相同的着色操作，其中着色单元40中的每一个经配置以执行本发明的用于合并着色操作的技术。总地来说，硬件着色单元可以指代GPU(例如GPU 36)调用以执行特定的着色操作的一组硬件组件。

[0072] 在一个实例中，本发明的方面包含使用单个硬件着色单元执行顶点着色操作和几何形状着色操作。在另一实例中，本发明的方面包含使用单个硬件着色单元执行顶点着色操作和壳体着色操作。在又一实例中，本发明的方面包含使用单个硬件着色单元执行域着色操作和几何形状着色操作。本发明的方面还涉及硬件着色单元在着色操作之间的过渡方式。也就是说，本发明的方面涉及在使用硬件着色单元执行第一着色操作与使用相同的硬件着色单元执行第二着色操作之间的过渡。

[0073] 举例来说，根据本发明的方面，GPU 36可以使用被指定用以执行顶点着色操作的着色单元40执行顶点着色操作以对输入顶点进行着色以便输出经顶点着色的顶点。在这个实例中，可以用接收单个顶点作为输入并且产生单个顶点作为输出的接口来配置着色单元40。此外，GPU 36可以使用相同的着色单元40来执行几何形状着色操作以基于经顶点着色的顶点中的一或多个执行一或多个新顶点。几何形状着色操作可以对一或多个经顶点着色的顶点中的至少一个进行操作以输出一或多个新顶点。同样，虽然是相对于单个着色单元40描述的，但是这些技术可以通过GPU 36的多个着色单元40并行地实施。

[0074] 某些API可能必需被指定执行顶点着色操作的着色单元40实施或遵照1:1接口，其接收单个顶点作为输入并且产生单个顶点作为输出。相比之下，专用于执行几何形状着色操作的着色单元40可以实施或遵照1:N接口，其接收一或多个顶点作为输入并且产生一或多个(并且通常是许多，因此上面使用“N”)顶点作为输出。

[0075] 根据本发明的方面，GPU 36可以利用着色单元40的被指定执行顶点着色操作的1:1接口来模仿这个1:N几何形状着色器接口，方法是通过调用几何形状着色器程序的多个例子。GPU 36可以并行地执行这些几何形状着色器程序中的每一个以产生从执行几何形状着色器操作得出的新顶点中的一个。也就是说，着色单元40可以使用HLSL(例如，具有图形渲染API)可编程，使得着色单元40可以并行地执行通常称为“着色器程序”的多个例子。这些着色器程序可以称为“纤程”或“线程”(这两者都可以指代形成程序或执行线程的指令流)。根据本发明的方面并且如下文更详细地描述，GPU 36可以使用被指定用于顶点着色操作的硬件着色单元来执行几何形状着色器程序的多个例子。GPU 36可以将几何形状着色器指令附加到顶点着色器指令，使得相同的着色单元40循序地执行这两个着色器，例如顶点着色器和几何形状着色器。

[0076] 在另一实例中，根据本发明的方面，GPU 36可以使用指定用以执行顶点着色操作的硬件着色单元来执行顶点着色操作以对输入顶点进行着色，以便输出经顶点着色的顶点。硬件着色单元可以遵照接收单个顶点作为输入并且产生单个顶点作为输出的接口。此外，GPU可以使用被指定用于执行顶点着色操作的相同硬件着色单元来执行一或多个曲面细分操作(例如，壳体着色操作和/或域着色操作)以基于经顶点着色的顶点中的一或多个产生一或多个新顶点。所述一或多个曲面细分操作可以对所述一或多个经顶点着色的顶点

中的至少一个进行操作以输出一或多个新顶点。

[0077] 举例来说，除了上述着色器级之外，一些图形渲染管线还可以包含壳体着色器级、曲面细分器级和域着色器级。总地来说，包含壳体着色器级、曲面细分器级和域着色器级以适应硬件曲面细分。也就是说，包含壳体着色器级、曲面细分器级和域着色器级以适应GPU 36的曲面细分，而不是(举例来说)由CPU 32执行的软件应用程序来执行。

[0078] 根据本发明的方面，GPU 36可以使用相同着色单元40执行顶点着色和曲面细分操作。举例来说，GPU 36可以在两个遍次中执行顶点着色和曲面细分操作。根据本发明的方面并且在下文中更详细地描述，GPU 36可以存储各种值以实现不同着色操作之间的过渡。

[0079] 在一实例中，在第一遍次中，GPU 36可以指定一或多个着色单元40执行顶点着色和壳体着色操作。在这个实例中，GPU 36可以将壳体着色器指令附加到顶点着色器指令。因此，相同的着色单元40循序地执行顶点着色和壳体着色器指令。

[0080] 在第二遍次中，GPU 36可以指定一或多个着色单元40执行域着色和几何形状着色操作。在这个实例中，GPU 36可以将域着色器指令附加到几何形状着色器指令。因此，相同的着色单元40循序地执行域着色和几何形状着色操作。通过在多个遍次中执行多个着色操作，GPU 36可以使用相同的着色硬件来模仿具有额外着色能力的GPU。

[0081] 本发明的方面还涉及GPU 36在不同着色操作之间过渡的方式。举例来说，本发明的方面涉及将着色操作拼补在一起使得所述操作由相同的硬件着色单元循序地执行的方式。

[0082] 根据本发明的方面，在一实例中，GPU 36可以指定一或多个着色单元40执行与渲染管线的第一着色器级相关联的第一着色操作。GPU 36可以在完成第一着色操作后即刻切换着色单元40的操作模式。GPU 36可以接着使用被指定用以执行第一着色操作的相同的着色单元40执行与渲染管线的第二不同着色器级相关联的第二着色操作。

[0083] 根据一些实例，GPU 36可以使用多种模式将着色操作拼补在一起，其中每一模式具有特定一组相关联的着色操作。举例来说，第一模式可以指示绘制调用仅包含顶点着色操作。在这个实例中，在执行绘制调用后，GPU 36即刻可以指定一或多个着色单元40根据模式信息执行顶点着色操作。此外，第二模式可以指示绘制调用包含顶点着色和几何形状着色操作两者。在这个实例中，在执行绘制调用后，GPU 36即刻可以指定一或多个着色单元40执行顶点着色操作。此外，根据本发明的方面，GPU 36可以将几何形状着色器指令附加到顶点着色器指令，使得相同的着色单元执行顶点着色操作和几何形状着色操作两者。可以使用额外模式来指示着色器的其它组合，如下文更详细地描述。

[0084] 在一些实例中，GPU驱动器50可以产生GPU 36使用的模式信息。根据本发明的方面，不同着色器(例如，顶点着色操作、几何形状着色操作、壳体着色操作、域着色操作等等)不必须用特定方式编译以便由相同的着色单元40循序地执行。而是每一着色器可以独立编译(无需参考任何其它着色器)并且在绘制时被GPU 36拼补在一起。也就是说，在执行了绘制调用后，GPU 36即刻可以确定与绘制调用相关联的模式，并且相应地将编译的着色器拼补在一起。

[0085] 本发明的技术可以使得具有用于执行着色操作的有限数目个着色单元40的GPU(例如GPU 36)能够模仿具有更大数目个着色单元40的GPU。举例来说，虽然GPU 36可能受到阻止无法指定着色单元40执行两个以上着色操作(例如，顶点着色操作和像素着色操作)，

但是本发明的技术可以使得GPU 36能够执行额外的着色操作(例如,几何形状着色操作、壳体着色操作和/或域着色操作),而无需重新配置着色单元40。也就是说,所述技术可以允许着色单元40在执行其它着色操作时遵照某些着色器级的输入/输出约束。

[0086] 此外,通过用相同的着色单元40执行多个着色操作,所述技术可以减少存储器总线带宽消耗。举例来说,在用其它着色操作(例如,几何形状着色)执行顶点着色的情况下,用于顶点着色的着色单元40无需在执行其它着色器操作之前将顶点着色结果存储到芯片外存储器(例如存储单元48)。而是可以将顶点着色结果存储到GPU存储器38并且立即将所述结果用于几何形状着色操作。

[0087] 以此方式,所述技术与具有额外着色单元40的GPU相比可以减少存储器总线带宽消耗,这可以减少功率消耗。所述技术可以因此促成功率效率更高的GPU,其利用的功率少于具有额外硬件着色器单元的GPU。因此,在一些实例中,所述技术可以部署在功率有限的装置中,例如移动装置、膝上型计算机和任何其它类型不具有恒定的专用功率供应的装置。

[0088] 应理解,计算装置30可以包含图1中为了清晰起见未展示的额外模块或单元。举例来说,计算装置30可以包含用于发射和接收数据的收发器模块,并且可以包含用以允许计算装置30与另一装置或网络之间的无线或有线通信的电路。计算装置30还可包含扬声器和麦克风(图1中都未展示)以在计算装置30是移动无线电话的实例中实行电话通信,或者在计算装置30是媒体播放器的实例中包含扬声器。在一些例子中,在计算装置30是台式计算机或其它经装备以与外部用户接口或显示器介接的装置的实例中,用户接口单元46和显示器单元42可以在计算装置30外部。

[0089] 图2是说明示范性图形处理管线80的框图。实例管线80包含输入汇编器级82、顶点着色器级84、几何形状着色器级86、光栅化器级88、像素着色器级90和输出合并器级92。在一些实例中,API(例如DirectX 10(或Direct3D 10)API)可经配置以使用图2中展示的级中的每一个。图形处理管线80在下文被描述为由GPU 36执行,但是可以由多种其它图形处理器执行。

[0090] 图形处理管线80总体上包含可编程级(例如,用圆角说明)和固定功能级(例如,用方角说明)。举例来说,与图形渲染管线80的某些级相关联的图形渲染操作总体上由可编程着色器处理器(例如,着色单元40中的一个)执行,而与图形渲染管线80相关联的其它图形渲染操作总体上由与GPU 36相关联的不可编程的固定功能硬件单元执行。着色单元40执行的图形渲染级总体上可以称为“可编程”级,而固定功能单元执行的级总体上可以称为固定功能级。

[0091] 输入汇编器级82在图2的实例中展示为固定功能级,并且总体上负责将图形数据(三角形、线和点)供应到图形处理管线80。举例来说,输入汇编器级82可以收集用于高阶表面、基元等等的顶点数据,并且将顶点数据和属性输出到顶点着色器级84。因此,输入汇编器级80可以使用固定功能操作从芯片外存储器(例如,存储单元48)读取顶点。输入汇编器级80可以接着从这些顶点创建管线工作项目,同时还产生顶点识别符(“VertexID”)、例子识别符(“InstanceID”,其可供顶点着色器使用)和基元识别符(“PrimativeID”,其可供几何形状着色器和像素着色器使用)。输入汇编器级80可以在读取顶点后即刻自动产生VertexID、InstanceID和PrimitiveID。

[0092] 顶点着色器级84可以处理接收到的顶点数据和属性。举例来说,顶点着色器级84

可以执行逐个顶点的处理，例如变换、蒙皮、顶点位移和计算逐个顶点的材料属性。在一些实例中，顶点着色器级84可以产生纹理坐标、顶点颜色、顶点照明、雾因子等等。顶点着色器级84总体上获得单个输入顶点，并且输出单个经处理的输出顶点。

[0093] 几何形状着色器级86可以接收由顶点数据(例如，用于三角形的三个顶点、用于线的两个顶点或用于点的单个顶点)定义的基元，并且进一步处理所述基元。举例来说，几何形状着色器级86可以执行逐个基元的处理，例如轮廓边缘检测和阴影体挤出，以及其它可能的处理操作。因此，几何形状着色器级86可以接收一个基元作为输入(其可包含一或多个顶点)，并且输出零个、一个或多个基元(其同样可包含一或多个顶点)。输出基元可以含有比在没有几何形状着色器级86的情况下可能的更多的数据。输出数据的总量可以等于顶点大小乘以顶点计数并且对每个调用可能受到限制。来自几何形状着色器级86的流式输出可以允许将到达这个级的基元存储到芯片外存储器，例如存储器单元48。输出流通常被连接到几何形状着色器级86，并且两者可以编程在一起(例如，使用API)。

[0094] 光栅化器级88通常是负责剪切基元并且为像素着色器级90准备基元的固定功能级。举例来说，光栅化器级88可以执行剪切(包含定制剪切边界)、视角划分、视区/剪刀选择和实施、渲染目标选择和基元设置。以此方式，光栅化器级88可以产生多个片段以供像素着色器级90着色。

[0095] 像素着色器级90从光栅化器级88接收片段，并且产生逐个像素的数据，例如颜色。像素着色器级96还可以执行逐个像素的处理，例如纹理混合和照明模型计算。因此，像素着色器级90可以接收一个像素作为输入，并且可以输出在相同相对位置处的一个像素(或像素的零值)。

[0096] 输出合并器级92总体上负责组合各种类型的输出数据(例如像素着色器值、深度和模板信息)以产生最终结果。举例来说，输出合并器级92可以为渲染目标(像素位置)执行固定功能混合、深度和/或模板操作。虽然上文总体上是相对于顶点着色器级84、几何形状着色器级86和像素着色器级90描述的，但是前述描述中的每一个可以涉及由GPU指定来执行相应着色操作的一或多个着色单元(例如着色单元40)。

[0097] 某些GPU可能不能够支持图2中展示的所有着色器级。举例来说，一些GPU可能不能够指定着色单元执行两个以上着色操作，原因在于硬件和/或软件限制(例如，着色单元40和相关联的组件的数目有限)。在一实例中，某些GPU可能不支持与几何形状着色器级86相关联的操作。而是GPU可以只包含对于指定着色单元执行顶点着色器级84和像素着色器级90的支持。因此，由着色单元执行的操作必须遵照与顶点着色器级84和像素着色器级90相关联的输入/输出接口。

[0098] 此外，在一些实例中，相对于不包含几何形状着色器级86的图形处理管线，将几何形状着色器级86引入到管线可能会导致对存储单元48的额外读取和写入。举例来说，如上所述，顶点着色器级86可以将顶点向外写入到芯片外存储器，例如存储单元48。几何形状着色器级86可以读取这些顶点(顶点着色器级84输出的顶点)并且写入新顶点，接着对新顶点进行像素着色。对存储单元48的这些额外读取和写入可能会消耗存储器总线带宽，同时还潜在地增加所消耗的功率量。在这个意义上，实施包含顶点着色器级84、几何形状着色器级86和像素着色器级90中的每一个的图形处理管线可能会产生功率效率更低的GPU，其在输出经渲染的图像方面可能也更慢，原因在于从存储单元48检索数据时的延迟。

[0099] 如上所述，本发明的方面总体上涉及合并着色单元40中的一或多个的功能，使得被指定用于特定着色操作的着色单元40可以执行一个以上着色操作。如下文更详细地描述，在一些实例中，可以指定一个着色单元40执行与顶点着色器级84相关联的顶点着色操作。根据本发明的方面，也可以实施相同的着色单元40以执行与几何形状着色器级86相关联的几何形状着色操作。也就是说，GPU 36可以调用着色单元40以执行顶点着色操作，但是也可以实施着色单元40以执行几何形状着色操作，而无需重新指定着色单元40执行几何形状着色任务。

[0100] 图3A和3B是根据本发明的方面的图形渲染管线中的数据流的概念图。举例来说，图3A说明顶点着色器级100、几何形状着色器级102、流式输出104和像素着色器级106。总地来说，图3A中展示的顶点着色器级100、几何形状着色器级102和像素着色器级106各自表示用于执行着色操作的相关联的硬件。也就是说，举例来说，顶点着色器级100、几何形状着色器级102和像素着色器级106中的每一个可以与分开指定的处理单元(例如，被指定执行相应任务的着色单元40)相关联。

[0101] 举例来说，顶点着色器级100表示执行顶点着色操作的一或多个单元(例如，着色单元40)。也就是说，顶点着色器级100可以包含被GPU 36调用以执行顶点着色操作的组件。举例来说，顶点着色器级100可以接收顶点作为输入，并且将输入顶点从三维(3D)模型空间翻译成屏幕空间中的二维(2D)坐标。顶点着色器级100可以接着输出顶点的经翻译版本(其可以称为“经翻译顶点”)。顶点着色器级100一般不创建新顶点，而是一次对一个顶点进行操作。因此，顶点着色器级100可以称为一对一(1:1)级，顶点着色器级100接收单个输入顶点并且输出单个输出顶点。

[0102] 几何形状着色器级102表示执行几何形状着色操作的一或多个单元(例如着色单元40)。也就是说，几何形状着色器级102可以包含被GPU 36调用以执行几何形状着色操作的组件。举例来说，几何形状着色器级102可以用于执行多种多样的操作，例如对立方体贴图的单遍渲染、点块纹理产生等等。通常几何形状着色器级102接收一或多个经翻译顶点构成的基元，所述顶点已经被顶点着色器级100顶点着色。几何形状着色器级102执行几何形状着色操作以创建可以形成新基元的新顶点(或者可能将输入基元变换成具有额外新顶点的新类型的基元)。

[0103] 举例来说，几何形状着色器级102通常接收由一或多个经翻译的顶点定义的基元，并且基于接收到的基元产生一或多个新顶点。几何形状着色器级102接着输出新顶点(其可以形成一或多个新基元)。结果，几何形状着色器级102可以称为一对多(1:N)或甚至多对多(N:N)级，因为几何形状着色器级102接收一或多个经翻译的顶点并且产生多个新顶点。

[0104] 虽然被描述为是一对多或甚至多对多，但是在一些例子中，几何形状着色器级102也可以不输出任何新顶点或者仅输出单个新顶点。在这个方面，所述技术不应仅限于在每个例子中输出许多顶点的那些几何形状着色器，而是可以总体上相对于任何可输出零、一个或许多新顶点的几何形状着色器级102来实施，如下文将更详细地解释。

[0105] 可以存储几何形状着色器级102的输出以用于额外的几何形状着色(例如，在流式输出104期间)。还可以将几何形状着色器级102的输出输出到光栅化器，其使新顶点(和经翻译的顶点)光栅化以产生由像素构成的光栅图像。

[0106] 还可以将来自几何形状着色器级102的像素传递到像素着色器级106。像素着色器

级106(也可以称为片段着色器)可以计算每一像素的颜色和其它属性,从而执行多种多样的操作以产生经着色的像素。可以将经着色的像素与深度贴图合并,并且可以执行其它着色后操作以产生输出图像供经由显示器装置(例如计算机监视器、电视机或其它类型的显示器装置)显示。

[0107] 图3A中展示的着色器级可以支持一或多个图形API。在用于说明的目的的实例中,顶点着色器级100、几何形状着色器级102和像素着色器级106可以支持DirectX10 API。也就是说,可以通过顶点着色器级100、几何形状着色器级102和像素着色器级106执行使用DirectX 10 API产生的代码,以渲染图形数据。然而,几何形状着色器级102可能不包含在所有图形渲染管线中,并且可能不能由所有GPU执行。举例来说,虽然DirectX 10 API包含对于几何形状着色器级102的支持,但是某些较早版本(例如,DirectX 9)不包含此支持。因此,经设计以执行用DirectX API的较早版本创建的代码(或针对其它API设计的GPU)可能不能够指定着色单元40执行几何形状着色器级102。

[0108] 图3B说明根据本发明的技术的图形渲染管线(相对于图3A中展示的实例)中的数据流的修改概念图。图3B中展示的实例包含合并的顶点着色器/几何形状着色器(VS/GS)级110、流式输出112和像素着色器级114。根据本发明的方面,合并的VS/GS级110可包含用于执行上文相对于顶点着色器级100和几何形状着色器级102描述的功能的一或多个处理单元。也就是说,虽然顶点着色器级100和几何形状着色器级102表示被GPU(例如GPU 36)调用以便分别执行顶点着色操作和几何形状着色操作的不同单元,但是根据本发明的方面,此些功能可以由基本上相同的硬件(例如,着色单元40)执行。

[0109] 举例来说,在GPU 36调用了顶点着色操作之后,VS/GS级110即刻可以执行顶点着色操作和几何形状着色操作两者。也就是说,合并的VS/GS级110可以包含用于执行上文相对于顶点着色器级100描述的操作和用于执行上文相对于几何形状着色器级102描述的操作的相同组的着色单元40。

[0110] 然而,因为GPU 36起初是作为顶点着色单元调用每一着色单元40,但是GPU 36的组件可以经配置以用特定格式(例如,遵照1:1输入/输出接口)从顶点着色单元接收数据。举例来说,GPU 36可以分配高速缓存(例如,顶点参数高速缓存,如下文更详细地描述)中的单个条目来存储来自着色单元40的用于经着色的顶点的输出。GPU 36还可以基于着色单元40被调用的方式来执行一些光栅化操作。如下文更详细地描述,本发明的方面允许GPU 36使用与顶点着色操作相同的着色单元执行几何形状着色操作,同时仍然遵照适当的接口。

[0111] 在一些例子中,几何形状着色器级102可以主要用于对数据的低放大(例如,点块纹理产生)。此些操作可能必需每次几何形状着色器调用有相对低的ALU使用。因此,着色单元40的ALU可能在几何形状着色器级102期间未被完全利用。根据本发明的方面,可以将几何形状着色器级102附加到顶点着色器级100以形成合并的VS/GS级110,其可以在GPU架构中作为顶点着色器级100来调用。用上述方式调用合并的VS/GS级110可以增加ALU利用率,方法是通过允许由相同处理单元执行顶点着色和几何形状着色操作两者。

[0112] 为了启用合并的VS/GS级110,GPU 36可以执行用于在顶点着色操作(1:1级)与几何形状着色操作(1:N级)之间过渡的功能,如相对于图4中展示的实例更详细地描述。以此方式,本发明的技术允许具有有限资源的GPU(例如,其可能阻止GPU指定着色单元40执行两个以上着色操作)以模仿具有额外资源的GPU。

[0113] 图4是说明实施本发明描述的技术以执行顶点着色操作和几何形状着色操作的硬件着色单元的实例操作的图。虽然是相对于GPU 36(图1)描述的,但是本发明的方面可以由具有各种其它组件的各种其它GPU执行。

[0114] 在图4的实例中,GPU 36可以指定着色单元40执行顶点着色操作。因此,GPU36的组件可以经配置以向着色单元40发送用于顶点的数据,并且从着色单元40接收用于经着色的顶点的数据(例如,1:1接口)。着色单元40可以执行顶点着色器以执行顶点着色操作,因而产生第一组基元120。在图4的实例中,第一组基元120包含带有四个顶点(标记为点p0-p3)的具有邻接关系的三角形。

[0115] 在执行了顶点着色操作之后,GPU 36可以将经着色的顶点存储到本机存储器资源。举例来说,GPU 36可以将顶点着色器输出连同“切割”信息(如果存在的话)和流识别符一起导出到(例如,GPU存储器38的)位置高速缓存。可以通过VS END指令将顶点着色操作与几何形状着色操作分开。因此,在执行了VS END指令和完成顶点着色操作之后,被指定执行顶点着色操作的一或多个着色单元40各自开始执行几何形状着色操作。

[0116] 也就是说,根据本发明的方面,被指定执行顶点着色操作的相同的着色单元40还执行几何形状着色操作。举例来说,GPU 36可以通过改变一或多个资源指针而改变几何形状着色器特定的资源的状态(例如,几何形状着色器常数、纹理偏移量等等)。GPU 36可以根据被指派给着色操作的模式(绘制模式)来执行这个状态改变。

[0117] 在一些实例中,GPU 36可以在执行绘制调用时设置绘制模式。绘制模式可以指示哪些着色操作与绘制调用相关联。在用于说明的目的的实例中,绘制模式0可以指示绘制调用仅包含顶点着色操作。绘制模式1可以指示绘制调用包含顶点着色操作和几何形状着色操作两者。如下文更详细地描述,其它绘制模式也是可能的。表1提供了具有两种模式的实例模式表:

[0118] 表1:模式信息合并的VS/GS

[0119]

模式	模式 0 GS:关	模式 1 GS:开
流	VS->PS	VS|GS->PS
索引 (32 位)	*顶点索引 (VS)*	*顶点索引 (VS)*
PrimitiveID (32 位)	未使用	*PrimitiveID* *(GS)*
Misc (25 位)	未使用	misc->rel_primID (4:0)
		misc->rel_vertex (9:5)
		misc->GsInstance (14:10)
		misc->Gsoutvertex (24:15)
Vs_valid (1 位)		
Gshs_valid (1 位)		
模式 (2:0)	模式= mode_0	模式= mode_1

[0120] 在上面表1的实例中，“流”指示与相应模式相关联的操作流(由GPU 36执行)。举例来说，模式0包含顶点着色(VS)和像素着色(PS)操作。因此，GPU 36可以指定着色单元40在执行模式0绘制调用时执行顶点着色操作和像素着色操作。表1的模式1包含顶点着色和像素着色操作，以及几何形状着色(GS)操作。

[0121] 因此，GPU 36可以指定着色单元40执行顶点着色操作和像素着色操作。然而，GPU 36还可以将几何形状着色器指令附加到顶点着色器指令，使得几何形状着色器操作由负责执行顶点着色器操作的相同着色单元40来执行。“misc”位被预留给变量(例如，rel_primID、rel_vertex、GsInstance、Gsoutvertex)，所述变量用于使得相同着色单元40能够连续地执行多个不同的着色器。

[0122] 在图4的实例中，相同着色单元40还使用第一组基元120作为输入而产生具有顶点V0-V5的第二组基元124(其可以称为三角形条带)。为了产生顶点V0-V5，被指定用于顶点着色的着色单元40执行几何形状着色器操作的多个例子(例如，通过其输出识别符(outID)指示并且也可以称为相同几何形状着色器程序的不同例子)。几何形状着色器操作的每一例子执行相同算法以执行相同的几何形状着色操作，并且产生一或多个新顶点V0-V5的相应例子。

[0123] 图4中展示的表的八个列对应于几何形状着色器操作(或程序)的八个分开的例子，其中从左到右的每一列可以由几何形状着色器操作outID 0-7来识别。每个输入基元的合并的VS/GS输出的数目可以等于dcl_maxoutputvertexcount*GSInstancecount，其中每一VS/GS输出是从几何形状着色器级发射的一个顶点。在几何形状着色器级输出顶点的数目小于dcl_maxoutputvertexcount的例子中，可以有条件地丢弃或忽略输出顶点(其可以被称为被“杀掉”)，如下文更详细地描述。因此，每一纤程对应于针对由MaxVertexOutput指定的每个几何形状着色器输出顶点对顶点着色器的一次调用然后是对几何形状着色器的一次调用。

[0124] 在图4中展示的实例中，通过被指定用于顶点着色操作的相同着色单元40通常并行地附加和执行几何形状着色器操作的八个例子中的每一个，以产生一或多个新顶点的分开的例子。因此，几何形状着色器操作的例子中的每一个产生全部六个顶点(V0-V5)，但是仅输出六个新顶点中的对应一个。几何形状着色器操作的每一例子仅输出六个新顶点中的对应一个，以便遵照与调用着色单元40以执行顶点着色操作相关联的1:1接口。

[0125] 如图4的实例中所示，几何形状着色器操作中的每一个输出六个新顶点中与其outID匹配的一个顶点。因此，几何形状着色器操作的具有outID＝0的第一例子输出六个新顶点中的第一个，V0。几何形状着色器操作的具有outID＝1的第二例子输出六个新顶点中的第二个，V1。几何形状着色器操作的具有outID＝2的第三例子输出六个新顶点中的第三个，V2。几何形状着色器操作的具有outID＝3的第四例子输出六个新顶点中的第四个，V3。几何形状着色器操作的具有outID＝4的第五例子输出六个新顶点中的第二个，V4。几何形状着色器操作的具有outID＝5的第六例子输出六个新顶点中的第六个，V5。

[0126] 几何形状着色器操作的第七和第八例子被“杀掉”或终止，因为几何形状着色器操作仅产生六个新顶点，并且几何形状着色器操作的第七和第八例子的outID不对应于所述六个新顶点中的任一个。因此，着色单元40在确定没有与几何形状着色器操作的这些例子相关联的对应顶点后即刻终止对几何形状着色器操作的第七和第八例子的执行。

[0127] 下面展示的表2说明可以由GPU 36维持以执行顶点着色操作和几何形状着色操作的若干参数。

[0128] 表2:用于VS/GS的参数

[0129]

流	VS|GS->PS
索引(32位)	顶点索引(VS)
uv_msb(2位)	未使用
PrimitiveID(32位)	PrimitiveID(GS)
Rel_patchid(32位)	未使用
Misc(25位)	misc->rel_primID(4:0)
	misc->rel_vertex(9:5)
	misc->GsInstance(14:10)
	misc->Gsoutvertex(24:15)
Vs_valid(1位)	
Gshs_valid(1位)	
模式(2:0)	模式=mode_1
Instance_cmd(2位)	

[0130] 表2中展示的某些参数(例如uv_msb、Rel_patchid)不用于VS/GS操作,并且在下文中更详细地描述。在表2的实例中,索引指示顶点的相对索引。PrimitiveID指示在几何形状着色操作期间使用以识别相关联的顶点的基元的基元ID,并且可以是系统产生的值(例如,由GPU 36的一或多个硬件组件产生)。如上所述,Misc指示用于在VS操作之后执行GS操作的预留高速缓存值。举例来说,下面展示的表3说明在执行上文相对于图4描述的顶点着色和几何形状着色操作时的参数值。

[0131] 表3:用于VS/GS操作的参数值

[0132]

模式1 GS:开	纤程0	纤程1	纤程2	纤程3	纤程4	纤程5	纤程6	纤程7
Valid_as_input	1	1	1	0	0	0	0	0
顶点索引(VS)	V0	V1	V2	0	0	0	0	0
primitiveID (GS)	5	5	5	5	5	5	5	5
Valid_as_output	1	1	1	1	1	1	1	1
misc-> rel_primID (4:0)	2	2	2	2	2	2	2	2
misc-> rel_vertex (9:5)	0	1	2	0	0	0	0	0
misc-> GsInstance (14:10)	0	0	0	0	0	0	0	0
misc-> Gsoutvertex (24:15)	0	1	2	3	4	5	6	7

[0133] 虽然分配了多个纤程(例如,指令)以执行顶点着色和几何形状着色操作,但是在一些例子中,GPU 36仅可以执行纤程的子组。举例来说,GPU 36可以在使用着色单元40执行指令之前先确定指令是否有效(上面表3中展示的valid_as_input)。因为仅使用所分配的

纤程中的三个来产生经着色的顶点，所以GPU 36可能在执行顶点着色操作时不执行其余纤程(上面表3中的纤程3-7)，这样可以节省功率。如下文更详细地描述，GPU 36可以基于掩码(例如，下面图5B中的cov_mask_1)确定有待执行的纤程。

[0134] 某些API(例如，DirectX 10 API)提供从几何形状着色器级的所谓的“流式输出”，其中流式输出是指从几何形状着色器向存储器(例如存储单元48)输出新顶点，使得这些新顶点可以被输入回到几何形状着色器中。

[0135] 所述技术可以提供对这个流式输出功能性的支持，方法是通过使得硬件单元能够向存储单元48输出从执行几何形状着色器操作得出的新顶点。经由这个流式输出而输出的新顶点是用期望的几何形状着色器格式而不是用光栅化器期望的格式指定的。硬件单元可以检索这些新顶点，并且继续对这些顶点(在这个上下文中，其可以称为“流式输出顶点”)实施现存的几何形状着色器操作，或者新几何形状着色器操作。以此方式，所述技术可以使得具有相对有限数目个着色单元40的GPU(例如GPU 36)能够模仿具有更多着色单元的GPU。

[0136] 图5A和5B说明可以由实施本发明的技术的硬件着色单元执行的实例操作。举例来说，图5A总体上说明合并的VS/GS硬件着色单元在执行顶点着色操作和几何形状着色操作时执行的操作流。在一些实例中，合并的VS/GS硬件着色单元可以包含着色单元40，其被GPU 36指定以执行顶点着色操作，但是根据本发明的技术执行顶点着色操作和硬件着色操作两者。

[0137] 图5B总体上说明对应于可以由合并的VS/GS硬件着色单元执行的图5A中展示的操作流的伪码。虽然图5A和5B的某些方面可能是相对于GPU 36(图1)描述的，但是本发明的方面可以由具有多种其它组件的多种其它GPU来执行。

[0138] 在图5A中展示的实例中，合并的VS/GS硬件着色单元将系统值(例如顶点属性、vertex_id、instance_id、primitive_id、misc)写入到一系列寄存器R0、R1和R2(140)。通常，可以将系统值存储到GPU的任何以其它方式未分配的存储器。通过将系统产生的值存储到预定位置中的一系列寄存器，GPU 36可以存取用于VS和GS级中的每一个的系统产生的值。因此，不需要基于VS级编译GS级以便确定系统产生的值已存储在哪个位置。而是，GPU 36可以在执行级中的每一个时存取预定存储器位置以存取所需要的系统产生的值。

[0139] 合并的VS/GS硬件单元接着执行顶点着色操作(142)。在顶点着色操作之后，合并的VS/GS硬件着色单元可以将通用寄存器(GPR)的内容(例如，来自顶点着色操作的基元顶点)写入到本机存储器，例如GPU存储器38。合并的VS/GS硬件着色单元可以接着切换到GS纹理和常数偏移量(146)和GS程序计数器(148)，如下文相对于图5B更详细地描述。

[0140] 合并的VS/GS硬件着色单元可以读取本机存储器的内容(例如顶点着色操作得出的基元顶点)，并且执行几何形状着色操作(150)。合并的VS/GS硬件着色单元可以将一个顶点属性输出到顶点参数高速缓存(VPC)，并且将经几何形状着色的顶点的位置的指示、stream_id、任何切割信息和任何经解释的值输出到位置高速缓存。

[0141] 图5B总体上说明对应于图5A中展示的可以由合并的VS/GS硬件着色单元执行的操作流的伪码。每一着色器级可以分开并且独立地编译(例如，无需知道特定级将如何与另一级链接)。为了允许单个硬件着色单元执行多个着色操作，硬件着色单元可以在本机存储器中预留某些位置。举例来说，硬件着色单元可以在本机存储器中预留可以被所述两个着色器级(VS或GS)存取的位置。某些变量(例如，PrimitiveID、misc和rel_patchid)可以由一个

以上着色器级使用。因此，本机存储器中预留的位置提供用于通常使用的变量的可以由一个以上着色器级存取的标准化位置。

[0142] 在图5B中展示的实例中，硬件着色单元可以起初执行顶点着色操作(VS)(包含在从上到下的第一虚线框中，所述虚线框可以对应于图5A的实例中的步骤140-142)。根据本发明的方面，硬件着色单元(或GPU的另一组件)可以接着执行所谓的“补码”以起始从顶点着色操作向几何形状着色操作(容纳在从上到下的第二虚线框中，所述虚线框可以对应于图5A的实例中的步骤144-148)的切换。更具体来说，命令CHMSK和CHSH可以使得硬件着色单元根据正被执行的绘制调用的模式来切换操作模式(如上所述)。

[0143] 举例来说，硬件着色单元可以将来自顶点着色操作的顶点数据写入到本机GPU存储器，使得在执行几何形状着色操作时经着色的顶点可供使用。硬件着色单元(或GPU的另一组件)接着执行改变掩码(CHMSK)指令，其切换硬件着色单元的资源以用于几何形状着色操作。举例来说，执行CHMSK指令可以使得硬件着色单元确定当前正在执行哪种模式。

[0144] 相对于上面的表2，执行CHMSK还可以使得硬件着色单元确定哪些着色器级是有效的(例如，vs_valid、gs_valid等等)。如上所述，GPU 36可以分配多个纤程来执行顶点着色和几何形状着色操作。然而，在执行了CHMSK后，GPU 36即刻可以仅执行纤程的子组。举例来说，GPU 36可以在使用着色单元40执行指令之前确定指令是否有效。GPU 36可能不执行无效(例如，不产生经着色的顶点)的纤程，这可以节省功率。

[0145] 硬件着色单元还执行改变着色器(CHSH)指令以将程序计数器(PC)切换成适当的状态偏移量以便执行几何形状着色操作。如下文更详细地描述，无论正在合并哪些着色器级，这个补码(容纳在从上到下的第二虚线框中，所述虚线框可以对应于图5A的实例中的步骤144-148)可以是相同的。

[0146] 在执行补码之后，硬件着色器单元停止顶点着色操作，并且执行几何形状着色操作(容纳在从上到下的第三虚线框中，其对应于图5A的实例中的步骤150)。通常由执行多个着色操作的硬件着色单元执行的着色器(用于执行着色操作的代码)可能必需基于重新编译的着色器依赖性。举例来说，如果GS级使用primitiveID(系统产生的值)，那么可以编译VS级(例如，通过编译器54)以将primitiveID值放置在GS级可以从中拾取所述值的位置中。因此，VS级的编译可以取决于GS级的需要。

[0147] 根据本发明的方面，着色器中的每一个可以不相对于其它着色器独立编译。举例来说，可以在不知道何时将执行其它着色器的情况下独立地编译着色器。在编译之后，GPU 36可以基于与在绘制时间执行的绘制调用相关联的模式信息使用图5B中展示的补码将着色器拼补在一起。系统产生的值vertexID和instanceID可以仅在顶点着色器中使用，并且可以在通过编译VS级计算的指定通用寄存器槽(GPR)处加载。然而，着色器级中的任一个可以使用来自基元控制器(PC)(举例来说，如图6中所示)的primitiveID和其它合并着色器相关值，例如misc和rel_patchid。

[0148] 可以通过GPU 36的驱动器(例如，GPU驱动器50)将上述补码添加到经编译的着色器。举例来说，GPU驱动器50确定每一绘制调用必需哪些着色器。GPU驱动器50可以在所谓的驱动器时间或链接时间将图5B中展示的补码附接到适当的着色器(正被合并的着色器)。GPU驱动器50不需要重新编译整个着色器，因而节省了计算资源。

[0149] 以此方式，GPU 36可以使用多种模式将着色操作拼补在一起，其中每一模式具有

特定一组相关联的着色操作。此些技术可以使得GPU 36无需重新配置着色单元40就能够执行额外着色操作(例如,几何形状着色操作、壳体着色操作和/或域着色操作)。也就是说,所述技术可以允许着色单元40在执行其它着色操作时遵照某些着色器级的输入/输出限制。

[0150] 图6是说明根据本发明的方面的用于执行合并的顶点着色操作和几何形状着色操作的图形处理单元178的实例组件的图。图6的实例包含合并的VS/GS单元180、顶点参数高速缓存(VPC)182、基元控制器(PC)184、顶点取出解码器(VFD)186、图形光栅化器(GRAS)188、渲染后端(RB)190、命令处理器(CP)192和像素着色器(PS)194。此外,图6包含具有PM4包缓冲器198的存储器196、顶点对象200、索引缓冲器202、流式输出缓冲器204和帧缓冲器206。

[0151] 在图6的实例中,通过被指定用上述方式执行顶点着色操作的一或多个着色单元实施VS/GS单元180。VPC 182可以实施流式输出功能性以将流式输出数据存储到流式输出缓冲器204。PC 184可以管理可能需要被变换的顶点。举例来说,PC 184可以将顶点汇编到三角形基元中。VFD 186可以基于顶点格式状态取出顶点数据。GRAS188可以接收三角形顶点作为输入,并且可以输出在三角形边界内的像素。预取剖析器(PFP)可以对命令流进行预解码并且经由指针(例如,资源指针)取出数据,使得到主CP引擎192可能需要这个数据时这个数据已准备就绪。

[0152] 在用于说明的实例中,可以使用图6中展示的图形处理单元178来实施DirectX 10分派机制。举例来说,可以将DirectX绘制调用视为单遍绘制调用,其带有具有模式位(模式信息)的绘制起始符,所述绘制起始符指示VS操作和GS操作被合并,例如由相同着色单元执行。这个模式使得PC 184内的GSblock能够产生用于VFD 186的具有GS输出vertexID和GS instanceID的数据。GSblock基于所声明的maxoutputvertexcount和GSinstancecount创建用于输入基元的多个VS纤程。如果一个波中的纤程的数目(例如,着色单元进行的工作的量,例如32个纤程)大于maxoutputvertexcount*GSinstancecount,那么一个波可以具有多个完整的输入GS基元。否则,可以对下一个波重复GS输入基元顶点索引,直到创建了maxoutputvertexcount*GSinstancecount个纤程为止。输入基元顶点不需要顶点再用。

[0153] 在VPC 182的输出处,PC 184将基于GS输出基元类型产生基元连接性。举例来说,来自(VS/GS 180)的第一输出顶点可以通常由位置高速缓存中的“切割”位组成,所述切割位可以指示在这个顶点之前完成基元(条带)。PC 184还将用于完整基元的这个连接性信息连同用于VPC 182的流识别符一起发送,以便将GS输出流式输出到与给定流相连的缓冲器204。如果GS 180中的完整基元之间存在部分基元,那么将此部分基元标记为PRIM_AMP_DEAD以供GRAS 188丢弃基元。PC 184还将无用基元类型发送到VPC 182以针对此基元解除对参数高速缓存的分配。

[0154] 基于maxoutputvertexcount,GPU驱动器(例如图1中展示的GPU驱动器50)可以计算将把多少个输入基元顶点存储在本机存储器中。可以根据下列等式计算这个输入基元值作为变量GS_LM_SIZE:

[0155] $$\frac{fibers\_in\_a\_wave}{maxoutputvertexcount} * 每个基元的顶点数目 * 顶点大小$$

[0156] 接收这种类型的绘制调用的高级排序器(HLSQ)可以检验哪个着色器处理器的本机存储器(LM)具有用于GS_LM_SIZE的足够存储空间(例如,可能使用轮询方法)。HLSQ可以

维持此分配的开始基准地址，以及所分配的波对本机存储器的任何读取或写入的地址。HLSQ还可以在写入到本机存储器时在分配给基准地址的存储器内添加所计算的偏移量。

[0157] 因此，根据本发明的方面，对于VS/GS 180来说输入与输出之间的关系不是1:1(这对于被指定执行顶点着色操作的着色单元通常将是典型的)。而是GS可以从每一输入基元输出一或多个顶点。此外，GS输出的顶点数目是动态的，并且可以从一变化成API强加的最大GS输出(例如，1024个双字(dword)，其可以等效于1024个顶点的输出最大值)。

[0158] 也就是说，GS可以产生一个顶点的最小值和1024个顶点的最大值，并且来自GS的总输出可以是1024个双字。GS可以在编译时使用变量dcl_maxoutputvertexcount声明来自GS的输出顶点的最大数目。然而，在GPU 36执行GS时，可能并不知道输出顶点的实际数目。而是，可能只有作为用于GS的参数才需要声明dcl_maxoutputvertexcount。

[0159] GS还可以声明用于每个输入基元有待调用的GS例子(操作)的数目的变量instancecount。这个声明可以用作GS调用的外部环路(识别几何形状着色器例子的最大数目)。最大instancecount可以设置成32，但是也可以使用其它值。因此，GS在几何形状着色器操作中可以存取变量GSInstanceID，其指示给定GS正对哪个例子进行操作。GS例子中的每一个可以输出多达1024个双字，并且每一个可以具有dcl_maxoutputvertexcount作为最大输出顶点的数目。此外，每一GS例子可以与其它GS例子无关。

[0160] GPU 36可以在GS的输入处宣布的输入基元类型可以是点、线、三角形、具有邻接关系的线、具有邻接关系的三角形和补片1-32。具有邻接关系的三角形可以是用于某些API(例如DirectX 10)的新特征。此外，补片1-32可以是用于针对DirectX 11 API添加的进一步增强。来自GS的输出基元类型可以是点、线条带或三角形条带。GS的输出可以去往可以在GS中声明的四个流中的一个，并且GS可以声明使用了多少个流。总地来说，“流”是指被存储(例如，存储到存储器缓冲器)或发送到GPU的另一单元(例如光栅化器)的经着色的数据。每一顶点“发射”指令可以使用“发射流”名称，其可以指示顶点正在去往哪个流。

[0161] GS可以使用“切割流”指令或“发射然后切割流”指令以完成条带基元类型。在此些实例中，下一个顶点将开始给定流的新基元。在一些实例中，编程人员可以在设置流时(使用API)声明至多使用流中的一个作为经光栅化的流。此外，四个1D缓冲器可以连接到一个流，但是连接到所有GS流的缓冲器的总数可能不超过四个。通常不在流之间共享芯片外缓冲器。

[0162] 当针对给定流发射一个顶点时，将连接到流的每一缓冲器的顶点的子段作为完整基元写入到芯片外缓冲器(例如存储单元48)。也就是说，通常不将部分基元写入到芯片外缓冲器。在一些实例中，写入到芯片外缓冲器的数据可以扩展为包含对基元类型的指示，并且如果对给定GS启用了一个以上流，那么GS的输出基元类型可能只是“点”。

[0163] GS级可以接收PrimitiveID参数作为输入，因为PrimitiveID是系统产生的值。GS还可以向一或多个寄存器输出PrimitiveID参数、ViewportIndex参数和RenderTargetArrayIndex参数。通常声明GS输入的属性内插模式是恒定的。在一些实例中，可以声明GS是空的(NULL)，但是仍然启用输出。在此些实例中，可能只有流零是有效的。因此，VS输出可以扩展以列举基元类型，并且可以将值写入到与流零相连的缓冲器。如果声明输入基元类型是邻接基元类型，那么可以丢弃邻接顶点信息。也就是说，举例来说，可以仅处理邻接基元的内部顶点(例如，偶数顶点数目)以形成非邻接的基元类型。

[0164] 在具有空GS的补片输入基元类型的情况下，将补片作为点的列表向外写入到与流相连的缓冲器。如果还对所声明的流进行了光栅化，那么GPU 36可以作为多个点渲染补片，如通过补片控制点所指定。此外，当GS是空的时，可以假设viewportindex参数和rendertargetarrayindex参数是零。

[0165] 可以实施查询计数器以确定GPU 36在处理多少个VS或GS操作，从而允许硬件组件追踪程序执行。查询计数器可以基于stat_start事件和stat_end事件而开始和停止计数。可以使用stat_sample事件对计数器进行采样。接收到stat_start和/或_stop事件的操作块将在各种点开始或停止计数，其中递增信号是发送、接收此些事件。

[0166] 当GPU 36的驱动器需要读取此些计数器时，驱动器可以通过命令处理器(CPU)发送stat_sample事件，如相对于图5B展示和描述。CP可以不向GPU 36发送任何额外绘制调用，直到寄存器主干管理(RBBM)单元从负责递增计数器的操作块获得返回的确认(或“ack”)为止。一旦接收到“ack”，RMMB单元就可以读取计数器，并且继续发送下一个(下一些)绘制调用。

[0167] GPU 36可以将各种数据存储到本机GPU存储器38。举例来说，可以用CP在硬件中维持接下来的查询计数。在一些实例中，接下来的查询计数可以形成为64位计数器，其可以使用来自各种操作块的1-3位脉冲递增，如下文所指示：

[0168] ●IAVertices可以指代在产生基元时使用的顶点的数目。因此，如果输入基元类型是产生三角形的条带，那么IAVertices值可以是6。这个值可以与视窗硬件质量实验室(WHQL)数目匹配。这个值可以使用来自基元控制器(PC)的2位脉冲得到控制。对于补片基元，所述值可以每个控制点一个地递增。

[0169] ●IAPrimitives可以指代所产生的完整输入基元的数目。这个值可能不包含任何可能导致复位的部分基元。这个值可以与WQHL数目匹配。可以在产生基元之后并且在检验复位索引和部分基元丢弃之后使用来自PC的一位脉冲来控制这个值。

[0170] ●VSInvocations可以指代调用VS操作的次数。可以在顶点再用之后设置这个值，其可以确定对其调用VS级的独特顶点的数目。这个值可以取决于GPU 36的特定硬件。可以在PC一次针对至多三个顶点检验顶点再用时使用来自PC的2位脉冲控制这个值。对于GS和壳体着色器(HS)(例如，如下文例如相对于图12A-13B描述)情况通常没有顶点再用。因此，PC可以发送绘制调用中的基元中的顶点的数目作为VSInvocations。

[0171] ●HSInvocations可以指代已经经过HS的补片的数目。这个值对于某些API(例如DirectX 11)可以是新值。这个值可能不包含任何部分补片。可以在将补片完整地发送到顶点取出解码器(VFD)时使用来自PC和来自HS块的一位脉冲控制这个值。这个值还应当与WHQL数目匹配。

[0172] ●DSInvocations可以指代调用域着色器(DS)操作的次数。当曲面细分输出基元类型具有点的类型时，这个值应当与WHQL匹配。针对正被产生的每一域点(u,v)使用来自PC中的曲面细分引擎(TE)的一位脉冲控制这个值。

[0173] ●GSInvocations可以指代调用GS操作的次数。如果使用GSinstancecount值，那么将每一例子作为一个GS调用来计数。这个值应当与WHQL数目匹配。可以使用每个Gsinstance每个输入基元发送一次的来自GS块的一位脉冲来控制这个值。在一些实例中，当GS放大大于波的大小时，GS块可以多次发送一个输入GS基元。通常对于每个GS输入基元

对这个值计数一次。

[0174] ●GSPrimitives可以指代所产生的GS输出基元的数目。这个值可能不包含从“切割”操作得出的任何部分基元。这个值可以与WHQL数目匹配。在存取其中构成了基元的位置高速缓存之后并且在因“切割”操作或顶点杀掉事件而丢弃部分基元之后，可以使用来自PC的每个输出基元的一位脉冲来控制这个值。

[0175] ●CInvocations可以指代执行所谓的“剪切器”的次数。这个值可以取决于GPU 36的特定硬件。

[0176] ●CPrimitives可以指代剪切器产生的基元的数目。这个值可以取决于GPU 36的特定硬件。

[0177] ●PSInvocations可以指代调用像素着色器(PS)线程(也可以称为“纤程”)的次数。

[0178] ●CSInvocations可以指代调用计算纤程的次数。

[0179] 除了上述值之外，还可能存在对于每个流维持的两个流式输出相关查询计数。这些流式输出相关值可以包含下面的值：

[0180] ●NumPrimitiveWritten可以指代在绘制调用结束之前针对给定流写入的基元的总数。在缓冲器中用于完整基元的存储空间不够时，这个值还可包含与流相连的缓冲器的数据。每当在给定流的任何缓冲器中存在用以存储完整基元的空间时，就可以使用从顶点参数高速缓存(VPC)到CP的每个流的一位脉冲来控制这个值。

[0181] ●PrimitiveStorageNeeded可以指代如果与流相连的任何缓冲器未用完存储空间那么可能已经写入的基元的总数。每当GS产生用于流的基元时，可以使用从VPC到CP的每个流的一位脉冲来控制这个值。

[0182] 通常，GPU 36可以支持从VPC直接流式输出。如上所述，可能存在GS支持的至多四个流。这些流中的每一个可以由至多四个缓冲器局限，并且所述缓冲器通常不能在不同流之间共享。对每一缓冲器的输出的大小可以是至多128个双字，这与顶点的最大大小是相同的。然而，一个跨步可能多达512个双字。来自流的输出数据可以存储到多个缓冲器，但是所述数据通常可能不在缓冲器之间复制。在用于说明的实例中，如果将“color.x”写入到与流相连的缓冲器中的一个，那么可以不将这个“color.x”发送到与相同流相连的另一缓冲器。

[0183] 可以作为完整基元执行到缓冲器的流式输出。也就是说，举例来说，如果任何缓冲器中只有用于两个顶点的给定流的空间，并且基元类型是三角形(例如，具有三个顶点)，那么可以不将基元顶点写入到与所述流相连的任何缓冲器。

[0184] 如果GS是空的，并且启用了流式输出，那么可以将流式输出识别为默认流零。当正在执行流式输出时，可以将位置信息写入到VPC中并且写入到PC中，这可能会消耗额外的槽。此外，当执行装仓(例如，向仓指派顶点以用于基于瓦片的渲染的过程)时，可以在装仓遍次期间执行流式输出。

[0185] 在一些API(例如DirectX 10)中，可以指定消耗流式输出数据的DrawAuto功能(其可以拼补并且渲染先前创建的流)。举例来说，GPU驱动器可以发送用于给定流的流式输出冲洗的事件以及存储器地址。VPC在接收到此事件后即刻可以向RBBM发送确认(ack)位。RBBM在接收到ack位后即刻将缓冲器中可用的缓冲器空间的量(缓冲填充大小)写入到驱动器指定的存储器或存储器位置。

[0186] 同时，可以包含在命令处理器(CP)内的预取剖析器(PFP)等待发送任何绘制调用。一旦写入了存储器地址，PFP就可以接着发送下一个绘制调用。如果下一个绘制调用是自动绘制调用，那么GPU驱动器可以发送含有缓冲器填充大小的存储器地址，作为指示绘制调用和状态变化的包(例如，所谓的“PM4”包)的一部分。PFP从所述存储器位置读取buffer_filled_size，并且将绘制调用发送到PC。

[0187] 图7是根据本发明的方面的用于执行顶点着色操作和几何形状着色操作的实例过程的流程图。虽然被描述为由GPU 36(图1)执行，但是应理解，相对于图7描述的技术可以由多种GPU或其它处理单元执行。

[0188] GPU 36可以起初(举例来说)在接收到顶点着色器指令后即刻调用顶点着色操作(210)。调用顶点着色操作可以使得GPU 36指定一或多个着色单元40用于顶点着色操作。此外，GPU 36的其它组件(例如顶点参数高速缓存、光栅化器等等)可以经配置以从指定着色单元40中的每一个对于一个输入接收单个输出。

[0189] GPU 36可以用指定用于顶点着色操作的硬件着色单元执行顶点着色操作以对输入顶点进行着色(212)。也就是说，硬件着色单元可以执行顶点着色操作以对输入顶点进行着色并且输出经顶点着色的索引。硬件着色单元可以接收一个顶点并且输出一个经着色的顶点(例如，输入与输出之间的关系是1:1)。

[0190] GPU 36可以确定是否执行几何形状着色操作(214)。GPU 36可以(举例来说)基于模式信息进行此确定。也就是说，GPU 36可以执行补码以确定是否将任何有效几何形状着色器指令附加到所执行的顶点着色器指令。

[0191] 如果GPU 36未执行几何形状着色操作(步骤214的“否”分支)，那么GPU硬件着色单元可以针对每一输入顶点输出一个经着色的顶点(222)。如果GPU 36确实执行了几何形状着色操作(步骤214的“是”分支)，那么硬件着色单元可以执行几何形状着色操作的多个例子以基于接收到的顶点产生一或多个新顶点(216)。举例来说，硬件着色单元可以执行预定数目个几何形状着色例子，其中每一例子与一个输出识别符相关联。硬件着色单元可以针对几何形状着色操作的每一例子维持一个输出计数。此外，可以给每一输出顶点指派一个输出识别符。

[0192] 因此，为了确定何时输出经几何形状着色的顶点，硬件着色单元可以确定何时输出计数与输出识别符匹配(218)。举例来说，如果用于几何形状着色操作的输出计数不与输出识别符匹配(步骤218的“否”分支)，那么放弃与几何形状着色操作相关联的顶点。如果用于几何形状着色操作的输出计数确实与输出识别符匹配(步骤218的“是”分支)，那么硬件着色单元可以输出与几何形状着色操作相关联的顶点。以此方式，经指定用于顶点着色的硬件着色单元针对几何形状着色程序的每一例子输出单个经着色的顶点，并且放弃任何未使用的顶点，从而维持1:1的输入与输出比率。

[0193] 图8是说明包含曲面细分级的实例图形处理管线238的框图。举例来说，管线238包含输入汇编器级240、顶点着色器级242、壳体着色器级244、曲面细分器级246、域着色器级248、几何形状着色器级250、光栅化器级252、像素着色器级254和输出合并器级256。在一些实例中，API(例如DirectX 11 API)可以经配置以使用图8中展示的级中的每一个。下文将图形处理管线238描述为由GPU 36执行，但是可以由多种其它图形处理器来执行。

[0194] 图8中展示的某些级可以与相对于图2展示和描述的级(例如，汇编器级240、顶点

着色器级242、几何形状着色器级250、光栅化器级252、像素着色器级254和输出合并器级256)类似或相同地配置。此外，管线238包含用于硬件曲面细分的额外级。举例来说，图形处理管线238除了上文相对于图2描述的级之外还包含壳体着色器级244、曲面细分器级246和域着色器级248。也就是说，包含壳体着色器级244、曲面细分器级246和域着色器级248以适应GPU 36的曲面细分，而不是由(举例来说)正由CPU 32执行的软件应用程序来执行。

[0195] 壳体着色器级244从顶点着色器级242接收基元，并且负责执行至少两个动作。首先，壳体着色器级244通常负责确定一组曲面细分因子。壳体着色器级244可以对于每个基元产生一次曲面细分因子。曲面细分器级246可以使用曲面细分因子来确定对给定基元多么精细地进行曲面细分(例如，将基元分成较小的部分)。壳体着色器级244还负责产生稍后将由域着色器级248使用的控制点。也就是说，举例来说，壳体着色器级244负责产生将由域着色器级248使用以创建实际经曲面细分的顶点(其最终在渲染时被使用)的控制点。

[0196] 当曲面细分器级246从壳体着色器级244接收数据时，曲面细分器级246使用若干算法中的一个确定用于当前基元类型的适当的采样模式。举例来说，总地来说，曲面细分器级246将请求量的曲面细分(由壳体着色器级244确定)转换成在当前“域”内的一群组坐标点。也就是说，取决于来自壳体着色器级244的曲面细分因子，以及曲面细分器级246的特定配置，曲面细分器级246确定当前基元中的哪些点需要被采样以便将输入基元曲面细分成较小的部分。曲面细分器级的输出可以是一组域点，所述域点可以包含重心坐标。

[0197] 域着色器级248除了壳体着色器级244产生的控制点之外还取得域点，并且使用域点创建新顶点。域着色器级248可以使用针对当前基元产生的控制点、纹理、程序性算法或任何其它项的完整列表将每一经曲面细分的点的重心“位置”转换成输出几何形状，所述输出几何形状被继续传递到管线中的下一个级。如上所述，某些GPU可能不能够支持图8中展示的所有着色器级。举例来说，一些GPU可能不能够指定着色单元执行两种以上着色操作，原因在于硬件和/或软件限制(例如，着色单元40和相关联的组件的数目有限)。在一实例中，某些GPU可能并不支持与几何形状着色器级250、壳体着色器级244和域着色器级248相关联的操作。而是GPU可以仅包含对于指定着色单元执行顶点着色器级242和像素着色器级252的支持。因此，由着色单元执行的操作必须遵照与顶点着色器级84和像素着色器级90相关联的输入/输出接口。

[0198] 此外，支持相对较长的图形处理管线可能必需相对更复杂的硬件配置。举例来说，来自壳体着色器级244、曲面细分器级246和域着色器级248的控制点、域点和曲面细分因子可能必需对芯片外存储器的读取和写入，这可能会消耗存储器总线带宽并且可能会增加所消耗的功率量。在这个意义上，对每一着色器级使用专用着色单元40来实施具有许多级的图形管线，可能会导致GPU的功率效率较低。此外，此些GPU还可能在输出经渲染的图像方面较慢，原因在于由于存储器总线带宽有限所以从芯片外存储器检索数据时存在延迟。

[0199] 根据本发明的方面，如下文更详细地描述，由GPU 36指定执行特定着色操作的着色单元40可以执行一种以上操作。举例来说，被指定执行顶点着色(VS)操作的着色单元40还可以执行与壳体着色器级244相关联的壳体着色操作。在另一实例中，相同着色单元40还可以执行与域着色器级248相关联的域着色操作，然后是与几何形状着色器级250相关联的几何形状着色器操作。

[0200] 如下文更详细地描述，GPU 36可以通过将绘制调用分裂成两个子绘制调用(例如，

遍次I和遍次II，其中每一子绘制调用具有相关联的合并的着色器级)来执行着色操作。也就是说，GPU 36可以调用着色单元40以执行顶点着色操作，但是还可以实施着色单元40以在第一遍次期间执行壳体着色操作。GPU 36可以接着使用相同的着色单元40(被指定执行顶点着色操作)以执行域着色操作和几何形状着色操作，而根本无需重新指定着色单元40执行壳体着色、域着色或几何形状着色任务。

[0201] 图9是更详细地说明曲面细分的概念图。壳体着色器级244和域着色器(DS)248可以是完全成熟的着色器级，每一个都具有其自身的一组恒定缓冲器、纹理和其它资源。总地来说，可以使用称为补片的基元类型来执行曲面细分。因此，在图9中展示的实例中，壳体着色器级244起初接收一或多个输入控制点，其可以称为补片控制点。补片控制点可以受到开发人员控制(例如，使用API)。壳体着色器级244可以执行计算以产生所谓的贝赛尔补片，其包含控制点，所述控制点由域着色器级248使用，如下文所述。

[0202] 壳体着色器级244还产生可以用于控制对补片的曲面细分量的曲面细分因子。举例来说，壳体着色器级244可以基于补片的视点和/或视距来确定要在多大程度上进行曲面细分。如果在一个场景中，一个对象相对接近观看者，那么可能必需用相对大量的曲面细分来产生总体上看起来光滑的补片。如果对象相对较远，那么可能必需较少的曲面细分。

[0203] 曲面细分器级246接收曲面细分因子并且执行曲面细分。举例来说，曲面细分器级246对具有统一级别的给定补片(例如，贝赛尔补片)进行操作以产生多个{U,V}坐标。{U,V}坐标可以为补片提供纹理。因此，域着色器级248可以接收控制点(具有位移信息)和{U,V}坐标(具有纹理信息)并且输出经曲面细分的顶点。如上所述，接着可以对这些经曲面细分的顶点进行几何形状着色。

[0204] 根据本发明的方面，并且如下文更详细地描述，可以通过GPU的相同着色单元(例如着色单元40)执行与壳体着色器级244和域着色器级248相关联的着色操作。也就是说，举例来说，一或多个着色单元40可以被指定执行顶点着色操作。除了顶点着色操作之外，GPU还可以附加与壳体着色器级244和域着色器级248相关联的着色器指令，以便通过相同着色单元循序地执行着色器，并且无需重新配置着色单元以执行曲面细分操作。

[0205] 图10A和10B是根据本发明的方面的图形渲染管线中的数据流的概念图。举例来说，图10A说明顶点着色器级260、壳体着色器级262、曲面细分器级264、域着色器级266、几何形状着色器级268、流式输出270和像素着色器级272。总地来说，图10A中展示的着色器级中的每一个表示用于执行着色操作的相关联的硬件。也就是说，举例来说，顶点着色器级260、壳体着色器级262、域着色器级266、几何形状着色器级268和像素着色器级272中的每一个可以与分开指定的处理单元(例如着色单元40)相关联。

[0206] 在图10A中展示的实例中，可以在所谓的“补片控制点”(或“控制点”，如上文相对于图8和9描述)上调用顶点着色器级260。给定补片中的点可以被壳体着色器级262看到，壳体着色器级262使用所述点计算曲面细分因子以供曲面细分级264使用。壳体着色器级262还可以输出补片控制点和常数数据以供域着色器级266使用。

[0207] 在一些实例中，曲面细分器级264可以包含固定功能硬件单元以用于执行曲面细分。曲面细分器级264可以从壳体着色器级262接收曲面细分因子和控制点，并且输出指定要对哪里进行曲面细分的所谓的域点(例如，{U,V}点)。域着色器级266使用这些域点以使用来自壳体着色器级262的输出补片数据计算顶点。来自域着色器级266的可能的输出基元

包含(举例来说)点、线或三角形,其可以被发送以用于光栅化,流式输出270或发送到几何形状着色器级268。如果曲面细分器级中的任一个小于或等于零,或者不是数字(NaN),那么可以剔除补片(被丢弃而不被进一步计算)。

[0208] 图10A中展示的着色器级可以支持一或多个图形API。在用于说明目的的实例中,顶点着色器级260、壳体着色器级262、域着色器级266、几何形状着色器级268和像素着色器级272可以支持DirectX 11 API。也就是说,使用DirectX 11 API产生的代码可以由顶点着色器级260、壳体着色器级262、域着色器级266、几何形状着色器级268和像素着色器级272执行以渲染图形数据。然而,例如壳体着色器级262、域着色器级266和/或几何形状着色器级268等某些级可能不包含在所有图形渲染管线中,并且可能不能被所有GPU执行。举例来说,虽然DirectX 11 API包含对于此些级的支持,但是较早的版本(例如DirectX 9和10)不包含此支持。因此,经过设计以执行用DirectX API的较早版本创建的代码的GPU(或针对其它API设计的GPU)可能不能够指定着色单元40执行与壳体着色器级262、域着色器级266和/或几何形状着色器级268相关联的操作。

[0209] 根据本发明的方面,可以合并图10A中的着色器级中的一个以上,因为所述着色器级是由单个硬件着色单元(例如,着色单元40)执行的。举例来说,根据本发明的方面,GPU(例如GPU 36)在执行绘制调用以执行图10A中展示的着色器级时可以执行多个遍次,如下文相对于图10B所述。

[0210] 图10B说明图形渲染管线中的数据流,其包含使用合并的顶点着色器和壳体着色器(VS/HS)级280的第一遍次(遍次I)。此外,数据流包含第二遍次(遍次II),其具有曲面细分级282、合并的域着色器和几何形状着色器(DS/GS)级284、流式输出286和像素着色器级288。可以实施图10B中展示的遍次以执行具有曲面细分操作的绘制调用。

[0211] 举例来说,GPU 36可以执行输入绘制调用,其包含曲面细分操作,如上文相对于图10A所述。GPU 36可以起初将绘制调用分成多个子绘制调用,其中每一子绘制调用包含遍次I操作和遍次II操作两者。GPU 36划分绘制调用的方式可以至少部分地取决于可供使用的存储器量(例如,芯片上GPU存储器,L2,全局存储器(GMEM)或芯片外存储器)。举例来说,GPU 36可以配置子绘制调用,使得GPU 36能够将通过遍次I操作产生的所有数据存储到本机存储器以供遍次II操作使用。可以在命令处理器(CP)代码的控制下在CP中进行绘制调用的划分,其可以基于输入绘制调用类型。

[0212] 在用于说明的实例中,假设绘制调用包含1000个相关联的补片以用于渲染。此外,假设本机存储器具有用以存储与100个补片相关联的数据的容量。在这个实例中,GPU 36(或用于GPU的驱动器,例如GPU驱动器50)可以将绘制调用分裂成10个子绘制调用。GPU 36接着针对10个子绘制调用中的每一个循序地执行遍次I操作和遍次II操作。

[0213] 相对于遍次I操作,在GPU 36调用了顶点着色操作后,VS/HS级280即刻可以执行顶点着色操作和壳体着色操作两者。也就是说,合并的VS/HS级280可以包含一或多个着色单元的单个组,并且可以循序地执行上文相对于顶点着色器级260和壳体着色器级262描述的操作。如下文更详细地描述,本发明的方面允许GPU 36使用与顶点着色操作相同的着色单元执行壳体着色操作,同时仍然遵照适当的接口。在一些实例中,可以使用补码将壳体着色器指令附加到顶点着色器指令,从而允许相同着色单元执行这两组指令。

[0214] GPU 36可以接着执行遍次II操作。举例来说,曲面细分级282可以执行曲面细分,

如上文相对于曲面细分级264所述。合并的DS/GS级284可以包含与上述合并的VS/HS级280相同的一组一或多个着色单元40。合并的DS/GS级284可以循序地执行上文相对于域着色器级266和几何形状着色器级368描述的域着色和几何形状着色操作。在一些实例中，可以使用补码将几何形状着色器指令附加到域着色器指令，从而允许相同的着色单元执行这两组指令。此外，可以将这些域着色器指令和几何形状着色器指令附加到(遍次I的)壳体着色器指令，使得相同的着色单元无需重新配置就可以执行顶点着色、壳体着色、域着色和几何形状着色。

[0215] 遍次II几何形状着色操作可以包含与上文描述的那些几何形状着色操作基本上相同的几何形状着色操作。然而，当开始遍次II操作时，GPR初始化的输入(先前用于VS级，现在用于DS级)可以包含通过曲面细分级282产生的(u,v,patch_id)，而不是从顶点取出解码器(VFD)取出的数据。PC还可以针对遍次II计算rel_patch_id，并且可以将补片ID信息连同通过曲面细分级282计算的(u,v)传递到DS。曲面细分级282可以使用曲面细分因子产生用于经曲面细分顶点的(u,v)坐标。可以将曲面细分级282的输出馈送到合并的DS/GS级284以准备受到曲面细分以供进一步放大(几何形状着色)或流式输出286。DS使用来自芯片外暂存存储器的壳体着色器(HS)输出控制点数据和HS补片常数数据。

[0216] 在一些实例中，图10B中展示的两个遍次可以连续执行，但是两个遍次之间被闲置等待隔开。举例来说，GPU的CP可以发送用于遍次I操作的绘制调用。在开始对数据的遍次II之前，GPU可以等待控制点值被完整写入到本机存储器。为了确保在本机存储器中有正确的值可供使用，GPU可以在开始遍次II操作之前先确认GPU的组件是闲置的。

[0217] 命令处理器(CP)可以接着发送用于遍次II的绘制调用。在一实例中，开始第一有用顶点的延迟量与在遍次II中进行的工作量的比率可以大概小于2％。因此，在一些实例中，可能在遍次I与遍次II之间没有重叠。在其它实例中，如下文所述，GPU可以包含遍次I与遍次II操作之间的重叠。也就是说，GPU可以使前一绘制调用的遍次II的像素着色器级288的像素着色操作与当前绘制调用的遍次I的VS/HS级280的顶点着色操作重叠，因为像素着色器处理所花费的时间可能比顶点着色器处理长。

[0218] 根据本发明的方面，基元控制器(PC)可以在遍次I之后发送PASS_done事件，这可以帮助硬件单元切换到遍次II。在遍次I与遍次II之间可能存在重叠的实例中，遍次I操作和遍次II操作的存在可能在执行指令的着色器处理器处是相互排斥的。然而，在遍次I仍在执行时，可以取出用于遍次II的曲面细分因子。

[0219] 如下文相当于图11所述，PC可以对每个经着色的补片保持一个计数器，以记录完成了多少个遍次I波。这些计数器可以指示针对遍次I已有多少个补片完成了处理。一旦所有计数器值大于零，就可以针对遍次II取出曲面细分因子。因此，在遍次I完成之前遍次II就可以开始。然而，可能直到处理了用于遍次I绘制调用的所有索引之后，对于遍次II的绘制调用才开始处理。以此方式，可以避免遍次之间的管线冲洗(从本机GPU存储器传递到外部存储器)。

[0220] 图11是实施本发明中描述的技术以执行顶点着色和壳体着色操作的硬件着色单元的实例操作的图。举例来说，图11总体上说明根据本发明的技术在如上文相对于图10B所述的绘制调用的第一遍次(遍次I)期间执行顶点着色操作和壳体着色操作。虽然是相对于GPU 36描述的(图1)，但是本发明的方面可以由具有多种其它组件的多种其它GPU执行。

[0221] 在图11的实例中，GPU 36可以指定着色单元40执行顶点着色操作，其还可以最终执行壳体着色、域着色和几何形状着色(如下文更详细描述)，而无需被重新配置以执行此些着色操作。举例来说，着色单元40可以起初执行顶点着色操作以产生具有三个顶点(标示为点p0-p2)的输入基元(三角形条带)。

[0222] 在执行了顶点着色操作之后，GPU 36可以将经着色的顶点存储到本机存储器资源。举例来说，GPU 36可以将顶点着色器输出导出到(例如，GPU存储器38的)位置高速缓存。顶点着色操作和壳体着色操作可以通过VS END指令分开。因此，在执行了VS END指令并且完成顶点着色操作之后，被指定执行顶点着色操作的一或多个着色单元40各自开始执行壳体着色操作。

[0223] 相同的着色单元40可以接着执行壳体着色操作以产生具有控制点V0-V3的输出补片。在这个实例中，着色单元40以类似于上文相对于图4描述的几何形状着色器操作的方式执行壳体着色器操作的多个例子(通过其输出识别符(Outvert)指示)。壳体着色器操作的每一例子执行相同的算法以执行相同的壳体着色操作，并且产生一或多个新控制点(V0-V3)的相应例子。

[0224] 也就是说，图11中展示的表的四个列对应于壳体着色器操作(或程序)的四个分开的例子，其中从左到右的每一列可以通过壳体着色器操作Outvert 0-3来识别。通过着色单元40通常并行地执行壳体着色器操作的这四个例子中的每一个，以产生所述一或多个新控制点的分开的例子。因此，壳体着色器操作的例子中的每一个产生全部四个控制点(V0-V3)，但是仅输出四个新控制点中的对应一个。壳体着色器操作的每一例子仅输出四个新控制点中的对应一个，以便遵照着色单元40的1:1接口，其是针对顶点着色操作调用的。

[0225] 在图11的实例中，壳体着色器操作中的每一个输出四个新控制点中与其Outvert匹配的一个。因此，壳体着色器操作中具有Outvert＝0的第一例子输出四个新控制点中的第一个，V0。壳体着色器操作中具有Outvert＝1的第二例子输出四个新控制点中的第二个，V1。壳体着色器操作中具有Outvert＝2的第三例子输出四个新控制点中的第三个，V2。壳体着色器操作中具有Outvert＝3的第四例子输出四个新控制点中的第四个，V3。在壳体着色器值已经写入到本机存储器之后，可以在第二遍次(遍次II)期间执行域着色操作和几何形状着色操作，如上所述。

[0226] 根据本发明的方面，被指定执行顶点着色操作的相同着色单元40还执行上述壳体着色操作。此外，相同的着色单元40还可在绘制调用的第二遍次(遍次II)期间执行域着色和几何形状着色操作。举例来说，GPU 36可以为着色器特定的资源改变状态(例如，壳体、域和/或几何形状着色器常数、纹理偏移量等等)。GPU 36可以根据指派给着色操作的模式(绘制模式)执行这个状态改变。

[0227] 下面展示的表4说明可以由GPU 36维持以用相同的着色单元40执行顶点着色、壳体着色、域着色和几何形状着色的操作模式和参数。

[0228] 表4：用于执行着色操作的模式

[0229]

模式	模式 0 GS：关， HS：关	模式 1 GS：开 HS：关	模式 4 GS：开， HS：开 (遍次 II)	模式 3 GS：关， HS：开 (遍次 II)	模式 2 GS：关， HS：开 (遍次 I)
流	VS->PS	VS| GS->PS	DS| GS ->PS	DS->PS	VS|HS
索引(32 位)	*顶点索引* *(VS)*	*顶点索引* *(VS)*	u(15:0) v (31:16)	u(15:0) v (31:16)	顶点索引
uv_msb (2 位)	未使用	未使用	u,v 的上部位	u,v 的上部位	*未使用*
PrimitiveID (32 位)	未使用	*PrimitiveID* *(GS)*	*PrimtiiveID* *(DS, GS)*	*PrimtiiveID* *(DS)*	*PrimtiiveID* *(HS)*
Rel_patchid (32 位)	未使用	未使用	Rel_patchid (DS)	Rel_patchid (DS)	Rel_patchid (HS)
Misc (25 位)	未使用	misc-> rel_primID (4:0)	misc-> rel_primID (4:0)	未使用	misc-> rel_primID (4:0)
		misc-> rel_vertex (9:5)	misc-> rel_vertex (9:5)		misc-> rel_vertex (9:5)

[0230]

		misc-> GsInstance (14:10)	misc-> GsInstance (14:10)		misc-> outvertID (14:10)
		misc-> Gsoutvertex (24:15)	misc-> Gsoutvertex (24:15)		
Vs_valid (1 位)					
Gshs_valid (1 位)					
模式 (2:0)	模式= mode_0	模式= mode_1	模式= mode_4	模式= mode_3	模式= mode_2
Instance_cmd (2 位)					

[0231] 在一些例子中，如上面的表4中指示，针对特定绘制调用可能不执行某些着色操作。举例来说，绘制调用可包含顶点着色、壳体着色、域着色和像素着色操作，但是可能不包含几何形状着色操作(如针对模式3所示)。GPU 36可使用模式信息来确定在执行绘制调用时要执行哪些着色操作。

[0232] 下面展示的表5说明在执行遍次II操作而不执行几何形状着色操作时的参数值。

[0233] 表5：不执行几何形状着色的参数值

[0234]

模式 3 GS：关，HS：开	纤程 0	纤程 1	纤程 2	纤程 3	纤程 4	纤程 5	纤程 6	纤程 7
Valid_as_input	1	1	1	1	1	1	1	1
顶点索引(VS)	U|V	U|V	U|V	U|V	U|V	U|V	U|V	U|V
Uv_msb	u|v	u|v	u|v	u|v	u|v	u|v	u|v	u|v
primitiveID *(HS)*	105	105	105	105	105	105	105	105
Rel_patchID	5	5	5	5	5	5	5	5

[0235] 下面展示的表6说明在执行包含几何形状着色操作的遍次II操作时的参数值。

[0236] 表6:执行几何形状着色的参数值

[0237]

模式 4 GS：开，HS：开	纤程 0	纤程 1	纤程 2	纤程 3	纤程 4	纤程 5	纤程 6	纤程 7
Valid_as_input	1	1	1	0	0	0	0	0
顶点索引(VS)	U|V	U|V	U|V	U|V	0	0	0	0
Uv_msb	u|v	u|v	u|v	u|v	0	0	0	0
primitiveID(HS & GS)	105	105	105	105	105	105	105	105

[0238]

Rel_patchID	5	5	5	5	5	5	5	5
Valid_as_output	1	1	1	1	1	1	1	1
misc-> rel_primID (4:0)	0	0	0	0	0	0	0	0
misc-> rel_vertex (9:5)	0	1	2	0	0	0	0	0
misc-> GSInstance14:10)	0	0	2	0	0	0	0	0
misc-> GsOutvertex (24:15)	0	1	2	3	4	5	6	7

[0239] 在如图11中所示完成了与第一遍次(遍次I)相关联的操作之后,GPU 36可以等待闲置。GPU 36可以接着执行绘制调用的第二遍次(遍次II)以完成绘制调用。

[0240] 图12A和12B说明可以通过实施本发明的技术的硬件着色单元执行的实例操作。图12A和12B可以总体上对应于上文相对于遍次I描述的着色操作。

[0241] 举例来说,图12A总体上说明在执行顶点着色操作和壳体着色操作时由合并的VS/HS硬件着色单元执行的操作流。在一些实例中,合并的VS/HS硬件着色单元可包含被GPU 36指定以执行顶点着色操作但是根据本发明的技术执行顶点着色操作和壳体着色操作两者的着色单元40。图12B总体上说明对应于可以由合并的VS/HS硬件着色单元执行的图12A中展示的操作流的伪码。

[0242] 如图12A中所示,硬件着色单元可以执行VS操作然后是HS操作。举例来说,GPU(例如GPU 36)可以将系统产生的值(包含顶点属性、vertex_id、instance_id、primitive_id和misc(如上所述))写入到寄存器。如上所述,通过将系统产生的值在预定位置中写入到一系列寄存器,GPU 36可以针对VS和HS级中的每一个存取系统产生的值。因此,不需要基于VS级编译HS级以便确定系统产生的值已存储在哪个位置。而是GPU 36可以在执行所述级中的每一个时存取预定的存储器位置以存取所必需的系统产生的值。

[0243] 硬件着色单元可以接着执行顶点着色操作以产生一或多个经着色的顶点。硬件着色单元可以将经着色的顶点写入到本机存储器,使得经着色的顶点可以用于壳体着色操作。

[0244] GPU可以接着在执行壳体着色操作之前先切换存储器偏移量和程序计数器。GPU可以(举例来说)在执行上述补码时执行此些任务。硬件着色单元可以接着从本机存储器读取经着色的顶点,并且执行壳体着色操作以产生一或多个控制点和曲面细分因子。

[0245] 可以将在第一遍次期间产生的控制点和曲面细分因子存储到(举例来说)本机GPU存储器。在一些实例中,可以将控制点和曲面细分因子存储在本机GPU存储器内的分开的缓

冲器中。

[0246] 图12B是可以由执行上述遍次I操作的硬件着色单元执行的代码的实例部分。在图12B中展示的实例中，大写字母的单词是状态或常数寄存器。斜体单词指示着色器输入。针对VS/HS操作分配的GPR的数目是(gprs_needed_for_vs,gprs_needed_for_hs)的最大值。因此，在VS操作中使用之后，GPR被释放并且被用于HS操作。

[0247] 在一些例子中，在着色操作的VS部分中，仅执行有效的VS纤程(如上文相对于图5B所述)。一旦遇到“SWITCH_ACTIVE”指令，就改变覆盖掩码位使其与HS着色器相关联，并且仅执行活动的HS纤程。以此方式，预留的寄存器可以由VS和HS两者使用，并且VS和HS可以由单个硬件着色单元实施，而无需重新指定着色单元执行HS操作。

[0248] 图13A和13B还说明可以通过实施本发明的技术的硬件着色单元执行的实例操作。图13A和13B可以总体上对应于上述遍次II着色操作。

[0249] 举例来说，图13A总体上说明在执行域着色操作和几何形状着色操作时由合并的DS/GS硬件着色单元执行的操作流。在一些实例中，合并的DS/GS硬件着色单元可包含上文相对于图12A和12B描述的并且起初由GPU 36指定以执行顶点着色操作的相同的着色单元40。图13B总体上说明对应于可以由合并的DS/GS硬件着色单元执行的图13A中所示的操作流的伪码。

[0250] 根据本发明的方面，第一遍次(相对于图12A和12B描述)之后可以是“等待闲置”。也就是说，为了防止在第一遍次期间数据已完整写入到存储器之前在第二遍次期间从本机存储器读取数据，GPU可以先等待GPU的一或多个组件寄存为闲置(例如，未在计算或传递数据)然后才起始图13A和13B所示的第二遍次操作。

[0251] 在任何情况下，如图13A中所示，硬件着色单元都可以执行包含域着色和几何形状着色(还可以由固定功能的曲面细分单元执行曲面细分)的遍次II操作。举例来说，GPU可以将系统产生的值(包含{U,V}坐标、primitive_id和misc(如上所述))写入到寄存器。如上所述，通过在预定位置中将系统产生的值存储到一系列寄存器，GPU 36可以针对DS和GS级中的每一个存取系统产生的值。因此，不需要基于DS级编译GS级以便确定系统产生的值已存储在哪个位置。而是在执行级中的每一个时GPU 36可以存取预定存储器位置以存取所必需的系统产生的值。

[0252] 硬件着色单元可以接着执行域着色操作以产生一或多个经曲面细分的顶点。硬件着色单元可以将经曲面细分的顶点写入到本机存储器，使得经曲面细分的顶点可以用于几何形状着色操作。

[0253] GPU可以在执行几何形状着色操作之前先切换存储器偏移量和程序计数器。GPU可以(举例来说)在执行上述补码之前先执行此些任务。硬件着色单元可以接着从本机存储器读取经曲面细分的顶点，并且执行几何形状着色操作以产生一或多个经几何形状着色的顶点，其可以存储到顶点参数高速缓存。

[0254] 在图13B中所示的实例中，大写字母的单词是状态或常数寄存器。斜体单词指示着色器输入。针对这个着色器分配的GPR的数目是(gprs_needed_for_vs,gprs_needed_for_gs)的最大值。因此，在DS操作中使用的GPR被释放并且被用于GS操作。一旦遇到“SWITCH_ACTIVE”指令，就改变覆盖掩码位使其与GS操作相关联，并且仅执行有效GS纤程。一旦遇到“END_1st”指令，硬件着色器单元就可以将用于常数文件和纹理指针(例如，资源指针)的资

源偏移量切换成经GS编程的偏移量并且跳跃到GS的第一指令。以此方式，预留的寄存器可以由DS和GS着色器级两者使用，并且DS和GS着色器级可以由执行了遍次I操作的相同硬件着色单元执行。

[0255] 如图12A-13B的实例中所示，单个硬件着色单元可以执行四个不同着色器级的操作。根据一些实例，用于合并着色器级的补码可以是相同的，无论在合并哪些着色器级。举例来说，可以使用与用于合并VS和HS操作的补码(从图12B的顶部起的第二个虚线框中所示)的相同补码(从图13B的顶部起的第二个虚线框中所示)将DS操作与GS操作合并。硬件着色单元可以基于操作模式(如相对于上面的表展示和描述)切换到适当的着色操作，所述操作模式可以由GPU在绘制时确定。

[0256] 根据本发明的方面，每一着色器级(VS/GS/HS/DS)可以在不知道执行期间将如何链接所述级的情况下分开编译。因此，可以预留三个GPR以存储例如primitveID、rel_patch_ID和misc等参数。所述编译器可以使得输入属性或内部变量存储在对于DX10/DX11应用程序超过两个的GPR ID中。

[0257] 图14是根据本发明的方面的用于执行合并的顶点着色、壳体着色、域着色和几何形状着色操作的图形处理单元330的实例组件的图。图14的实例包含合并的VS/HS单元(遍次I)和合并的DS/GS单元(遍次II)332、顶点参数高速缓存(VPC)334、具有曲面细分器级337的基元控制器(PC)336、顶点取出解码器(VFD)338、图形光栅化器(GRAS)340、渲染后端(RB)342、命令处理器(CP)344和像素着色器(PS)346。图14包含存储器348，其具有PM4包缓冲器350、顶点对象352、索引缓冲器354、系统暂存器356和帧缓冲器358。

[0258] 在图14的实例中，由一或多个着色单元以上述方式实施VS/GS单元332。VPC334可以实施流式输出功能性以将流式输出的数据存储到存储器348。PC 336可以管理可能需要变换的顶点，并且将顶点汇编成三角形基元。VFD 338可以基于顶点格式状态取出顶点数据。GRAS 340可以接收三角形顶点作为输入，并且可以输出在三角形边界内的像素。预取剖析器(PFP)可以对命令流进行预解码，并且经由指针(例如，资源指针)取出数据，使得到主CP引擎344需要这个数据时这个数据已经准备就绪。

[0259] 相对于用于DirectX 11的分派机制，可以通过CP 344将绘制调用分成两个遍次的绘制。基于可用于存储遍次I的输出的可用存储空间，可以将绘制调用分成多个子绘制调用，其中每一子绘制调用具有遍次I和遍次II。每一子绘制调用可以遵照遍次的排序，以便针对子绘制调用执行遍次I，然后针对子绘制调用执行遍次II。

[0260] 在接收到具有遍次I的子绘制调用后，PC 336即刻可以使用VS/HS 332取出索引和处理补片基元类型。VS/HS 332对于每个补片创建 $HS\_FIBERS\_PER\_PATCH = 2^{ceil(\log_2(\max(input_{patch}, output_{patch})))}$ VS纤程，并且对于每个波配合整数个补片(其中波是给定的工作量)。在输入处不存在顶点再用。由于VS/HS 332的输出在芯片外传递到系统暂存器356，所以可能不存在对位置和参数高速缓存的分配。

[0261] 基于HS_FIBERS_PER_PATCH，GPU驱动器(例如图1中所示的GPU驱动器50)可以计算将在本机存储器中存储多少个输入基元顶点(在VS/HS 332的本机)。这可以如下计算：

[0262] $$HS\_LM\_SIZE \left\lceil \frac{fibers\_in\_a\_wave}{HS\_FIBERS\_PER\_PATCH} \right\rceil * control\_points\_in\_input\_patch * size\_of\_vertex$$

[0263] 如果在将最终数据写入到存储器348之前驱动器先要将中间数据写入到本机存储器，那么驱动器还可以向HS_LM_SIZE添加额外的大小。如果HS在HS的多个相位中(例如，在HS的恒定相位中)使用所计算的控制点，那么此额外空间可能是有用的。接收到这种类型的绘制调用的高级排序器(HLSQ)可以检验哪个着色单元的本机存储器(LM)具有用于GS_LM_SIZE的足够存储空间。HLSQ可以维持此分配的开始基准地址，以及所分配的波对本机存储器的任何读取或写入的地址。HLSQ还可以在写入到本机存储器时将在所分配的存储器内的所计算的偏移量添加到基准地址。

[0264] 还可以将系统解释的值(SIV)(例如，剪切/剔除距离、渲染目标、视区)提供到VPC 334以供加载到PS 346中。着色器级(例如，VS或GS)可以有条件地输出值。因此，如果PS 346需要所述值，那么PS 346可以作为状态的一部分设置此条件。如果PS 346不需要所述值，并且此确定是在编译了像素着色操作之后进行的，那么可以复位输出这些SIV的状态，使得VS或GS将不在绘制时把所述值写入到VPC 334。

[0265] 对于空的GS(如果未在执行几何形状着色器级)，那么编译器还可以创建模板GS，使得对于空的或非空的GS不存在分开的路径。这个模板GS可以将VS或域着色器(DS)输出复制到本机存储器，并且进一步从本机存储器复制以输出到VPC 334。可以只有对执行流式输出的情况才进行这个操作。

[0266] 依据正在实施哪些着色器，将可见性流装仓和消耗可见性流的过程可以是不同的。举例来说，某些GPU可以将有待渲染的图像数据分成瓦片或“仓”，从而连续渲染每一仓(或有时候同时或并行地渲染)，直到渲染了整个图像为止。通过将图像划分成仓，GPU可以减少芯片上存储器要求，同时还促进从芯片外存储器的较少数据检索(考虑到芯片上存储器可能足够大，能够存储足以渲染瓦片的图像数据)。

[0267] 相对于可见性流，可以使用Z缓冲器算法来确定被其它基元遮蔽(并且因此不需要被渲染)的基元。举例来说，GPU可以绘制每一基元，从最后面(深度方向)的基元操作到最前面(同样是深度方向)的基元。在这个实例中，可能一些基元仅被渲染成被其它基元覆盖绘制。

[0268] 由于这个所谓的“覆盖绘制”，GPU可以经过调整以适于执行早期的Z缓冲器算法测试，这允许GPU识别被完全遮蔽或不在眼睛视野内的在GPU执行渲染时将被忽略或绕过的基元。在这个方面，GPU可以经过调整以适于相对于每一基元和/或对象确定哪些可以被称为可见性信息。

[0269] 关于DX10，在装仓遍次期间，PC 336在来自GS的所有输出基元的末尾向GRAS 340发送“基元末尾”。因此，针对每个输入基元记录可见性信息。可以在装仓遍次期间执行流式输出。CP 344可以在装仓遍次结束时读取所有流式输出缓冲器相关信息。可以在装仓遍次期间更新几何形状相关查询计数器。

[0270] 可见性遍次可以读取可见性流，并且在读取每个基元的可见性信息时推进流。如果流都未被光栅化，那么可以跳过可见性遍次。否则，PC 336检验可见性输入GS基元并且加以处理以供渲染，并且没有任何流式输出。

[0271] 相对于DX11，在装仓遍次期间，PC 336在遍次II中来自GS的所有输出基元的末尾向GRAS 340发送“基元末尾”(例如，每个输入补片一位)。可以如上所述执行流式输出。在可见性遍次期间，在遍次I中连同补片一起处理可见性流(仅可处理具有可见性的补片)。遍次

II仅处理可见补片并且仅取出用于可见补片的曲面细分因子。

[0272] 下面展示的表7提供关于五种不同操作模式中的每一种的装仓遍次和渲染遍次的信息。每一模式对应于由单个硬件着色单元执行的某些操作，如上所述。

[0273] 表7：用于不同模式的装仓

[0274]

模式	VS 级	PS 级	装仓遍次	渲染遍次
Mode_0	VS	PS	每个基元的可见性信息	消耗可见性流
Mode_1	VS+GS	PS	每个输入基元的可见性信息； 对于经放大的基元，对仓覆盖率进行或运算以产生用于输入基元的可见性信息	消耗可见性流
Mode_2	VS+HS		不产生可见性	消耗可见性流
Mode_3	DS	PS	针对每个输入补片产生可见性信息，对所有经曲面细分的基元仓覆盖率进行或运算以产生用于输入基元的可见性信息	不消耗可见性流
Mode_4	(DS+GS)	PS	针对每个输入补片产生可见性信息，对所有经曲面细分和 GS 基元仓覆盖率进行或运算以产生用于输入基元的可见性信息	不消耗可见性流

[0275] 图15是说明根据本发明的方面的使用相同的硬件着色单元在两个渲染遍次中执行图形渲染的流程图。虽然是相对于GPU 36(图1)描述的，但是本发明的方面可以由具有多种其它组件的多种其它GPU执行。

[0276] 在图15的实例中，GPU 36确定被并行执行以渲染图形的绘制调用是否包含曲面细分操作(380)。曲面细分操作可包含(举例来说)与壳体着色器级、曲面细分级和域着色器级相关联的操作，如上所述。如果绘制调用不包含曲面细分操作，那么GPU 36可以用单个遍次执行渲染(382)。举例来说，GPU 36可以用上述方式执行顶点着色、几何形状着色和像素着色。

[0277] 如果绘制调用确实包含曲面细分操作，那么GPU 36可以确定本机GPU存储器资源(例如GPU存储器38)的大小(384)。GPU 36可以接着将绘制调用分裂成多个子绘制调用(386)。在一些实例中，每一子绘制调用可包含上述遍次I操作和遍次II操作。举例来说，遍次I操作可包含顶点着色操作和壳体着色操作，而遍次II可包含域着色操作和几何形状着色操作。

[0278] 可以基于GPU存储器38的大小来确定每一子绘制调用所渲染的数据量。举例来说，GPU 36可以配置子绘制调用，使得GPU 36能够将遍次I操作所产生的所有数据存储到本机存储器以供遍次II操作使用。以此方式，GPU 36可以减少在本机GPU存储器与在GPU外部的存储器之间传递的数据量，这样可以减少与渲染相关联的延迟，如上所述。

[0279] 在确定了子绘制调用之后，GPU 36可以针对第一子绘制调用执行遍次I操作(388)。如上所述，遍次I操作可包含使用相同的硬件着色单元(例如，一或多个着色单元40中的每一个)执行顶点着色操作和壳体着色操作。也就是说，虽然GPU 36可以指定多个着色单元40执行顶点着色，但是着色单元40中的每一个可以执行顶点着色和壳体着色操作两者。

[0280] GPU 36还可以针对第一子绘制调用执行遍次II操作(390)。如上所述，遍次II操作可包含使用相同的一或多个着色单元40执行域着色操作和几何形状着色操作。同样，虽然GPU 36可以指定多个着色单元40执行顶点着色，但是着色单元40中的每一个可以执行遍次

II操作，使得着色单元40中的每一个执行顶点着色操作、壳体着色操作、域着色操作和几何形状着色操作。

[0281] GPU 36还可以针对子绘制调用执行像素着色操作(392)。GPU 36可以使用一或多个其它着色单元40执行像素着色操作。在其它实例中，GPU 36可以在所有子绘制调用完成后执行整个绘制调用的像素着色。

[0282] GPU 36可以接着确定完成的子绘制调用是不是绘制调用的最终子绘制调用(392)。如果所述子绘制调用是绘制调用的最终子绘制调用，那么GPU 36可以输出与绘制调用相关联的经渲染的图形数据。如果所述子绘制调用不是绘制调用的最终子绘制调用，那么GPU 36可以返回步骤388，并且针对下一个子绘制调用执行遍次I操作。

[0283] 应理解，图15中展示的步骤只是作为一个实例提供的。也就是说，图15中展示的步骤不一定需要用所展示的次序执行，并且可以执行更少、额外或替代的步骤。

[0284] 图16是说明根据本发明的方面执行与两个遍次的图形渲染过程的第一遍次相关联的图形渲染操作的流程图。图16中展示的过程可以对应于上文相对于图15的步骤388描述的遍次I的操作。虽然是相对于GPU 36(图1)描述的，但是本发明的方面可以通过具有多种其它组件的多种其它GPU来执行。

[0285] 在图16的实例中，GPU 36可以起初指定一或多个着色单元40执行与图形渲染管线的顶点着色器级相关联的顶点着色操作，如上所述(400)。在执行了顶点着色操作之后，指定着色单元40中的每一个可以将经着色的顶点存储到本机存储器以用于壳体着色操作(402)。GPU 36还可以改变程序计数器以用于追踪壳体着色操作，以及改变到壳体着色器资源偏移量的一或多个资源指针。举例来说，资源指针可以指向针对壳体着色操作分配的数据位置。

[0286] 在这个意义上，着色单元40中的每一个改变操作模式以执行壳体着色操作。然而，模式改变并不包含重新指定着色单元40以执行壳体着色操作。也就是说，GPU 36的组件可以仍然经配置以用1:1接口格式向被指定用于顶点着色操作的着色单元发送数据和从其接收数据。

[0287] GPU 36可以接着使用执行了顶点着色操作的相同的着色单元40执行与图形渲染管线的壳体着色器级相关联的壳体着色操作，如上所述(404)。举例来说，每一着色单元40可以对经着色的顶点进行操作以产生一或多个控制点，所述控制点可用于曲面细分。

[0288] 应理解，图16中所示的步骤只是作为一个实例而提供。也就是说，图16中展示的步骤不一定需要用所展示的次序执行，并且可以执行更少、额外或替代的步骤。

[0289] 图17是根据本发明的方面执行与两个遍次的图形渲染过程的第二遍次相关联的图形渲染操作的流程图。图17所示的过程可以对应于上文相对于图15的步骤390描述的遍次II操作。虽然是相对于GPU 36(图1)描述的，但是本发明的方面可以由具有多种其它组件的多种其它GPU执行。

[0290] 在图17的实例中，GPU 36可以使用上文相对于图16描述的相同的着色单元40执行图17的操作。举例来说，为了执行遍次II操作，相同的着色单元40可以首先执行与图形渲染管线的域着色器级相关联的域着色操作，如上所述(420)。也就是说，着色单元40可以对控制点(来自壳体着色器级)进行操作以产生经域着色的顶点。

[0291] 在执行域着色操作之后，经指定的着色单元40中的每一个可以将经域着色的顶点

存储到本机存储器以用于几何形状着色操作(402)。GPU 36也可以改变程序计数器以追踪壳体着色操作，以及改变对壳体着色器资源偏移量的一或多个资源指针。在图17的操作是在相对于图16描述的操作之后的实例中，也可以在步骤420之前执行这些功能(例如，将值存储到本机存储器、改变程序计数器、改变资源偏移量)。

[0292] 在这个意义上，着色单元40中的每一个改变操作模式以执行域着色和几何形状着色操作。然而，模式改变并不包含重新指定着色单元40执行域着色和几何形状着色操作。也就是说，GPU 36的组件仍然可以经配置以用1:1接口格式向被指定用于顶点着色操作的硬件着色单元发送数据和从其接收数据。

[0293] GPU 36可以接着使用执行了域着色操作的相同着色单元40执行与图形渲染管线的几何形状着色器级相关联的几何形状着色操作，如上所述(424)。举例来说，每一着色单元40可以对经域着色的顶点进行操作以产生一或多个经几何形状着色的顶点。

[0294] 应理解，图17中展示的步骤只是作为一个实例而提供。也就是说，图17中展示的步骤不一定需要用所展示的次序执行，并且可以执行更少、额外或替代的步骤。

[0295] 图18是说明根据本发明的方面将一个以上着色器级拼补在一起以供相同的硬件着色单元执行的流程图。虽然是相对于GPU 36(图1)描述的，但是本发明的方面可以由具有多种其它组件的多种其它GPU执行。

[0296] 在图18的实例中，GPU 36可以指定一或多个硬件着色单元(例如，一或多个着色单元40)执行与第一着色器级相关联的着色操作(440)。在一些实例中，第一着色器级可以是用于产生顶点的顶点着色器级，使得GPU 36指定一或多个着色单元执行顶点着色操作。

[0297] 在完成与第一着色器级相关联的操作后，GPU 36即刻可以切换操作模式，从而允许相同的着色单元40执行多种其它着色操作(442)。举例来说，如上所述，GPU 36可以改变程序计数器和一或多个资源指针以便执行第二着色操作。

[0298] 在一些实例中，GPU 36可以基于与正被执行的绘制调用相关联的模式信息来切换着色单元40的操作模式。举例来说，GPU 36的驱动器(例如GPU驱动器50)可以产生用于绘制调用的模式编号，其指示要在绘制调用中执行哪些着色器级。GPU 36可以在执行了补码后即刻使用这个模式编号来改变着色单元的操作模式，如上所述。

[0299] 下面展示的表8总体上说明用于多种着色器级的组合的模式编号的模式信息。

[0300] 表8：着色器管线配置

[0301]

VS	(HS, *TE*,DS)	GS	SO	PS	绘制模式
开	关	关	关	开	模式 0
开	关	关	开	开/关	模式 0
开	关	开	关	开	模式 1
开	关	开	开	开/关	模式 1
开	开	关	关	开	遍次 1：模式 2 遍次 2：模式 3
开	开	关	开	开/关	遍次 1：模式 2 遍次 2：模式 3
开	开	开	关	开	遍次 1：模式 2 遍次 2：模式 4
开	开	开	开	开	遍次 1：模式 2 遍次 2：模式 4

[0302] 如表8中所示，每一模式规定着色单元执行哪些着色器级。因此，GPU 36可以将着色器指令串在一起，从而允许相同的着色单元40执行多个着色操作。也就是说，GPU 36可以基于正被执行的绘制调用的模式编号将适当的着色器指令拼补在一起。

[0303] 以此方式，GPU 36可以接着用被指定执行第一着色操作的相同的着色单元40执行第二着色操作(444)。举例来说，GPU 36可以执行顶点着色操作、壳体着色操作、域着色操作和几何形状着色操作的组合，如上面的表8中所展示。

[0304] 应理解，图18中展示的步骤只是作为一个实例而提供。也就是说，图18中展示的步骤不一定需要用所展示的次序执行，并且可以执行更少、额外或替代的步骤。

[0305] 虽然上文描述的某些实例包含起初指定硬件着色单元执行顶点着色操作并且过渡到用相同的硬件着色单元执行其它着色操作，但是应理解本发明的技术不以此方式受到限制。举例来说，GPU可以起初指定一组硬件着色单元执行多种其它着色操作。也就是说，在允许GPU指定硬件着色单元执行三个不同着色操作的系统中，GPU可以指定硬件着色单元执行顶点着色操作、壳体着色操作和像素着色操作。在这个实例中，GPU可以起初指定一或多个硬件着色单元执行壳体着色操作，但是也可以用相同硬件着色单元执行域着色操作和几何形状着色操作，如上所述。多种其它操作组合也是可能的。

[0306] 在一或多个实例中，所描述的功能可以用硬件、软件、固件或其任何组合来实施。如果用软件实施，那么所述功能可以作为一或多个指令或代码存储在包括非暂时性计算机可读媒体的制造品上。计算机可读媒体可包含计算机数据存储媒体。数据存储媒体可以是可由一或多个计算机或一或多个处理器存取以检索指令、代码和/或数据结构以用于实施本发明描述的技术的任何可用媒体。举例来说，并且并非作为限制，此计算机可读媒体可包括RAM、ROM、EEPROM、CD-ROM或其它光盘存储装置、磁盘存储装置或其它磁性存储装置、快闪存储器或任何其它可用于以指令或数据结构的形式载运或存储期望的程序代码并且可由计算机存取的媒体。本文中使用的磁盘和光盘包含压缩光盘(CD)、激光光盘、光学光盘、数字多功能光盘(DVD)、软盘和蓝光光盘，其中磁盘通常磁性地复制数据，而光盘用激光光学地复制数据。上述各项的组合也应包含在计算机可读媒体的范围内。

[0307] 所述代码可以由一或多个处理器来执行，例如一或多个DSP、通用微处理器、ASIC、FPGA或其它等效的集成或离散逻辑电路。此外，在一些方面中，本文中描述的功能性可以在专用硬件和/或软件模块内提供。此外，所述技术可以完全在一或多个电路或逻辑元件中实施。

[0308] 本发明的技术可以在多种多样的装置或设备中实施，包含无线手持机、集成电路(IC)或一组IC(例如，芯片组)。本发明中描述各种组件、模块或单元以强调本发明的经配置以执行所揭示的技术但不一定需要通过不同硬件单元实现的功能方面。而是，如上所述，各种单元可以在编解码器硬件单元中组合，或者由互操作性硬件单元的集合(包含上述一或多个处理器配合合适的软件和/或固件)提供。

[0309] 已描述了各种实例。这些和其它实例在所附权利要求书的范围内。

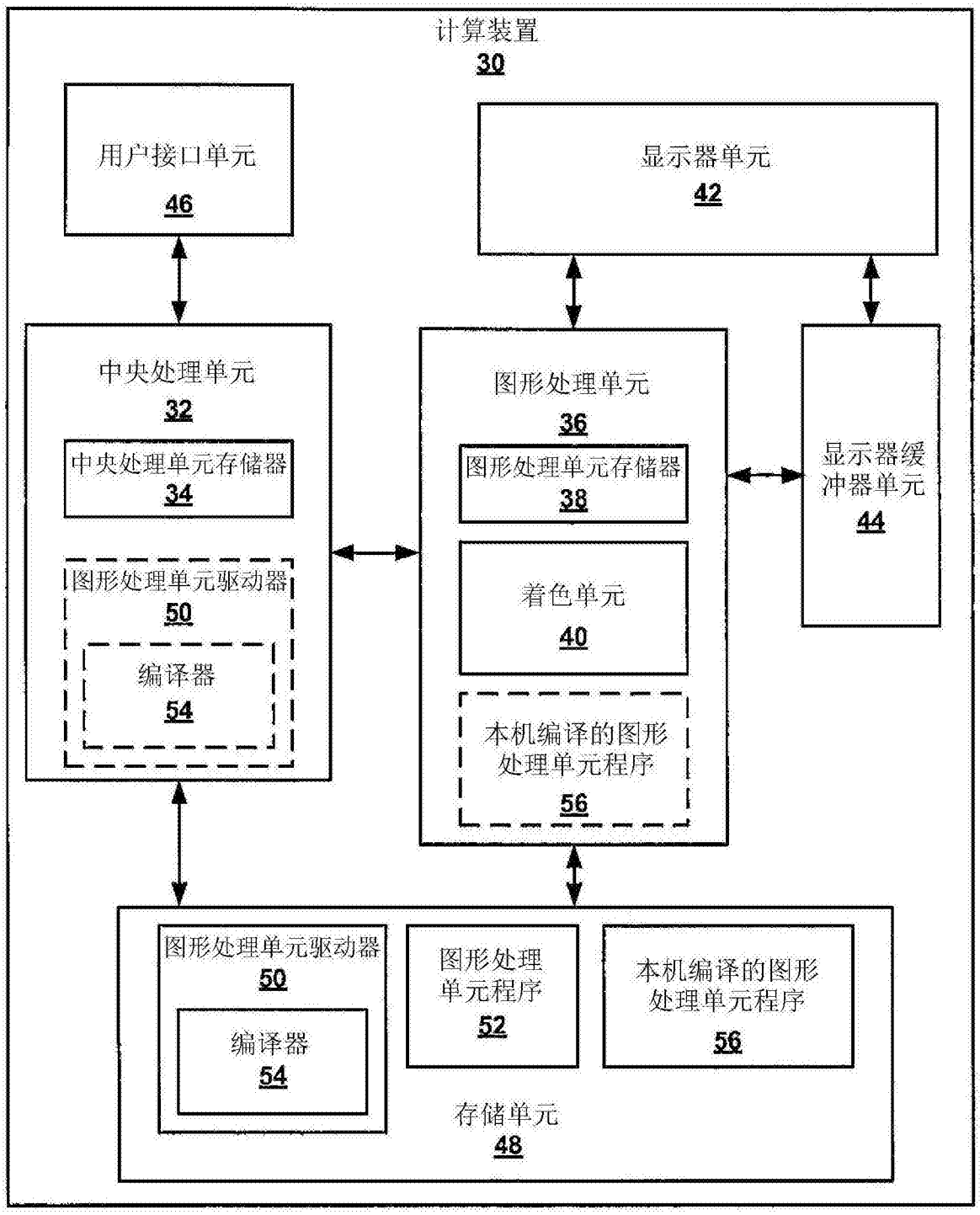

计算装置
30
用户接口单元
46
显示器单元
42
中央处理单元
32
中央处理单元存储器
34
图形处理单元驱动器
50
编译器
54
图形处理单元
36
图形处理单元存储器
38
着色单元
40
本机编译的图形处理单元程序
56
显示器缓冲器单元
44
图形处理单元驱动器
50
编译器
54
图形处理单元程序
52
本机编译的图形处理单元程序
56
存储单元
48

图1

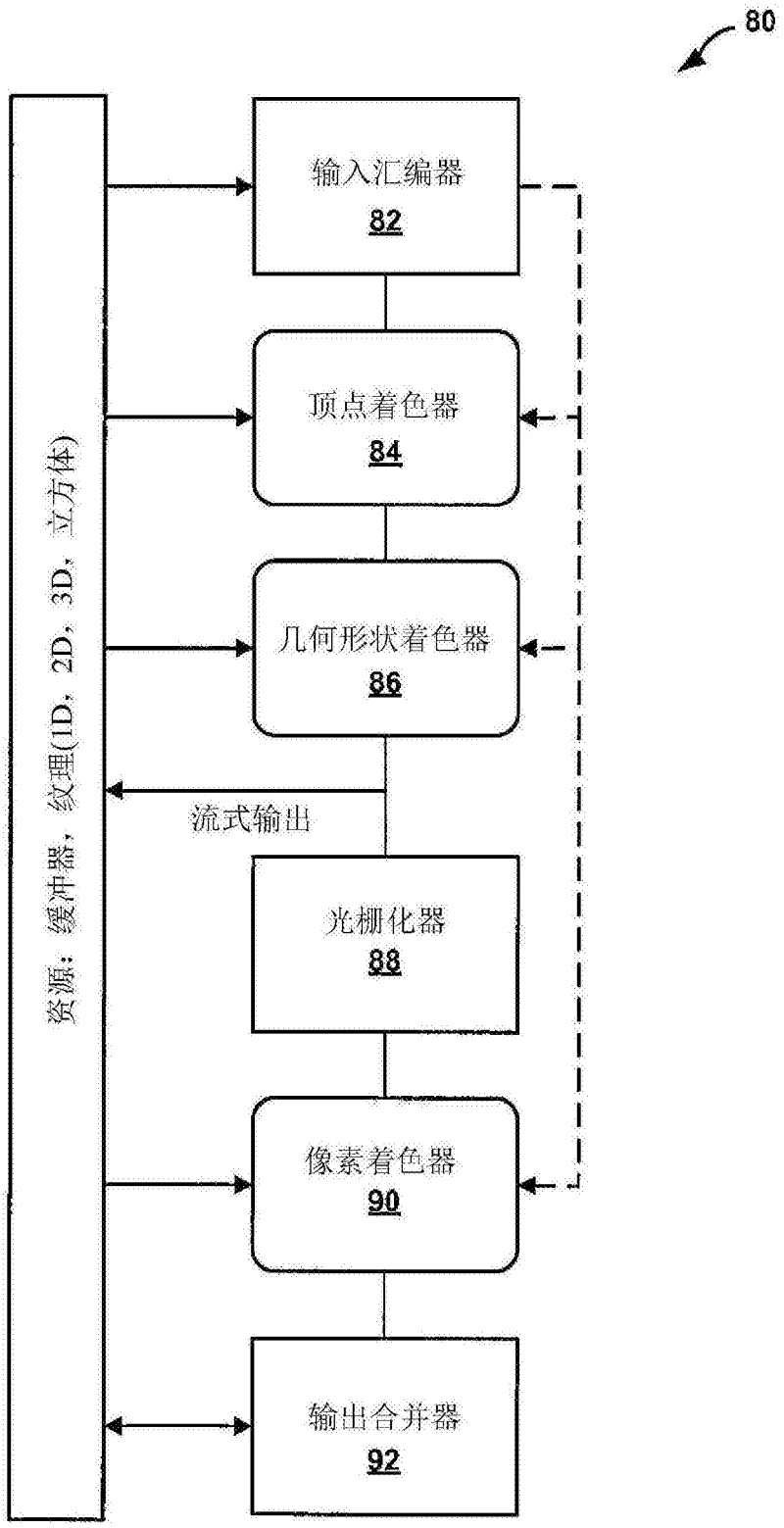

80
输入汇编器
82
顶点着色器
84
几何形状着色器
86
流式输出
光栅化器
88
像素着色器
90
输出合并器
92
资源：缓冲器，纹理(1D，2D，3D，立方体)

图2

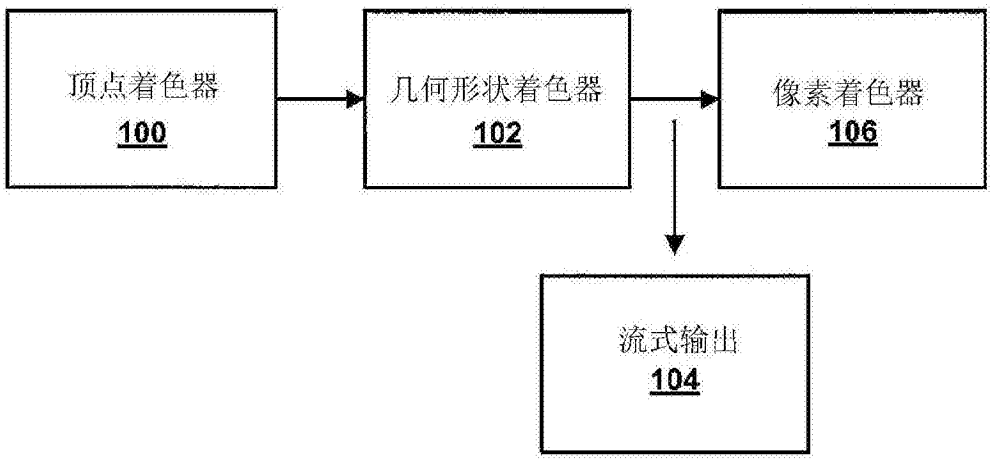

顶点着色器
100
几何形状着色器
102
像素着色器
106
流式输出
104

图3A

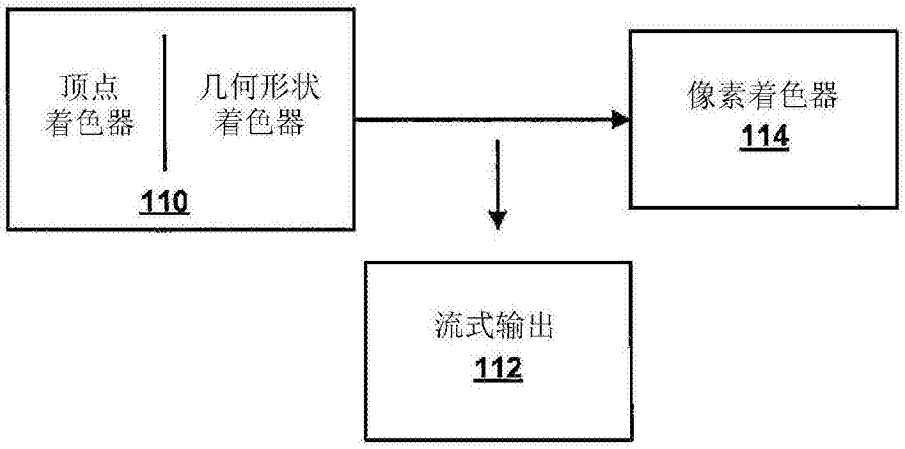
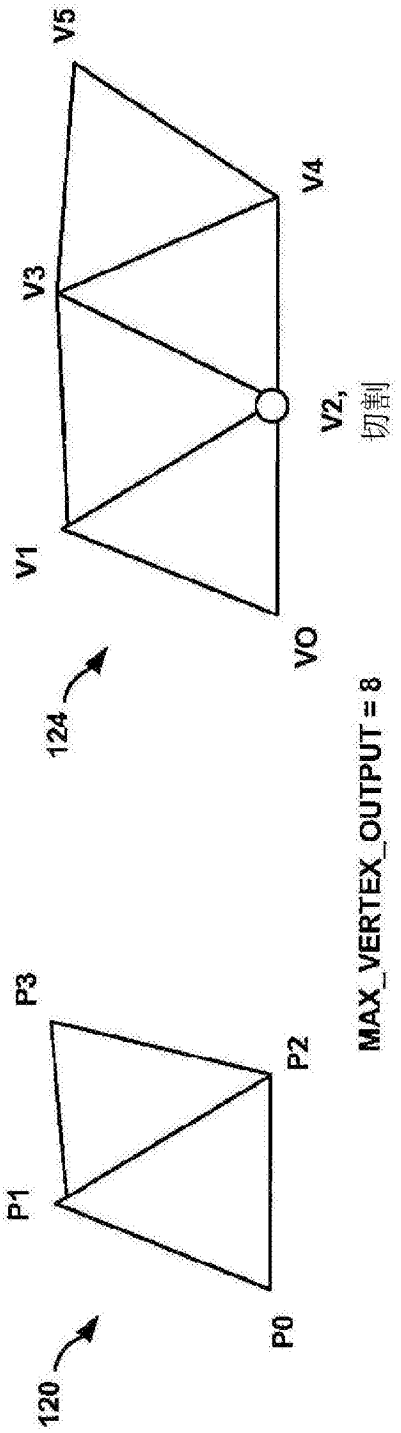

顶点
着色器
几何形状
着色器
110
像素着色器
114
流式输出
112

图3B



MAX_VERTEX_OUTPUT = 8

实际输出=6(来自三角形P0，P1，P2)

如果 (GSOUTPUTCNT==OUTID){
发射顶点
} // 等待发射，直到下一个发射或切割到来

PO, OUTID = 0 VS(P0)	P1, 1 VS(P1)	P2, 2 VS(P2)	X, 3 X	X, 4 X	X, 5 X	X, 6 X	X, 7 X
结束							
(PRIM = 0) GS(OUTID=0)	(PRIM = 0) GS(OUTID=1)	(PRIM = 0) GS(OUTID=2)	(PRIM = 0) GS(OUTID=3)	(PRIM = 0) GS(OUTID=4)	(PRIM = 0) GS(OUTID=5)	(PRIM = 0) GS(OUTID=6)	(PRIM = 0) GS(OUTID=7)
V0	V1	V2, 切割	V3	V4	V5	杀掉	杀掉

图4


顶点属性，VERTEX_ID,
INSTANCE_ID, PRIMITIVE_ID, MISC
将系统产生的值和MISC信息写入到R0，R1，R2
140
顶点着色器(VS)
142
将通用寄存器写入到本机存储器
144
切换到几何形状着色器纹理/常数偏移量
146
切换到几何形状着色器程序计数器
148
几何形状着色器(GS)
(从本机存储器读取输入基元顶点)
150
{一个顶点属性输出到顶点参数高速缓存}
{位置，STREAM_ID，切割，系统解释的值到位置高速缓存}

图5A

```
VS:  “正常”VS代码          // 可以使用instanceID，初始寄存器存储primitveiD，
                           //misc, rel_prim
                           // 数据留在GPR中
     END_FINAL;            // 如果是DX9样式的着色器就结束-否则的话就忽略，注意
                           // 如果着色器结束，是SP将通用寄存器写入到VPC。

     LM_Vs_offset = rel_primID*PRIM_SIZE + rel_vertex*VERT_SIZE;

     将GPR中的顶点数据写入到LM_Vs_offset

     CHMSK                 // 对第二级使用cov_mask_1，切换到第二
                           // 着色器资源(将资源ptr切换到GS资源
                           // 偏移量)

     CHSH                  // 切换到GS程序计数器和状态偏移量

GS:  对于 (vertex_id= 0; vertex_id++; vertex_id <= max_input)
       {   使用偏移量 = rel_primID*PRIM_SIZE + vertex_id * VERT_SIZE
           从LM加载
           // 使用通过HLSQ计算的基准地址从LM取出
       }
     对于 (streamid =0;streamid++;streamid<4);
     cut[streamid] = 真 ;
     gsoutount = 0;

// “正常”GS代码
     {.. 使用 primitiveID, GSInstanceID
     CUT(streamid):     { Cut[streamid] = 真 ;}
     EMIT(streamid): {
          如果 (Gsoutcount==GsoutvertID){
               oPos = vertex.pos; oColor =vertex.clr;
               oMisc = (cut[streamid],streamid)
               gsoutcount = maxoutputvertexcount;
                去往 END_FINAL;  // 任选
          }
          cut[streamid] = 假 ;
          gsoutcount++;
     }

     如果 (gsoutcount < maxoutputvertexcount)
          杀掉 PRIM;  //消除任何“剩余”纤程

END_FINAL;
```

图5B

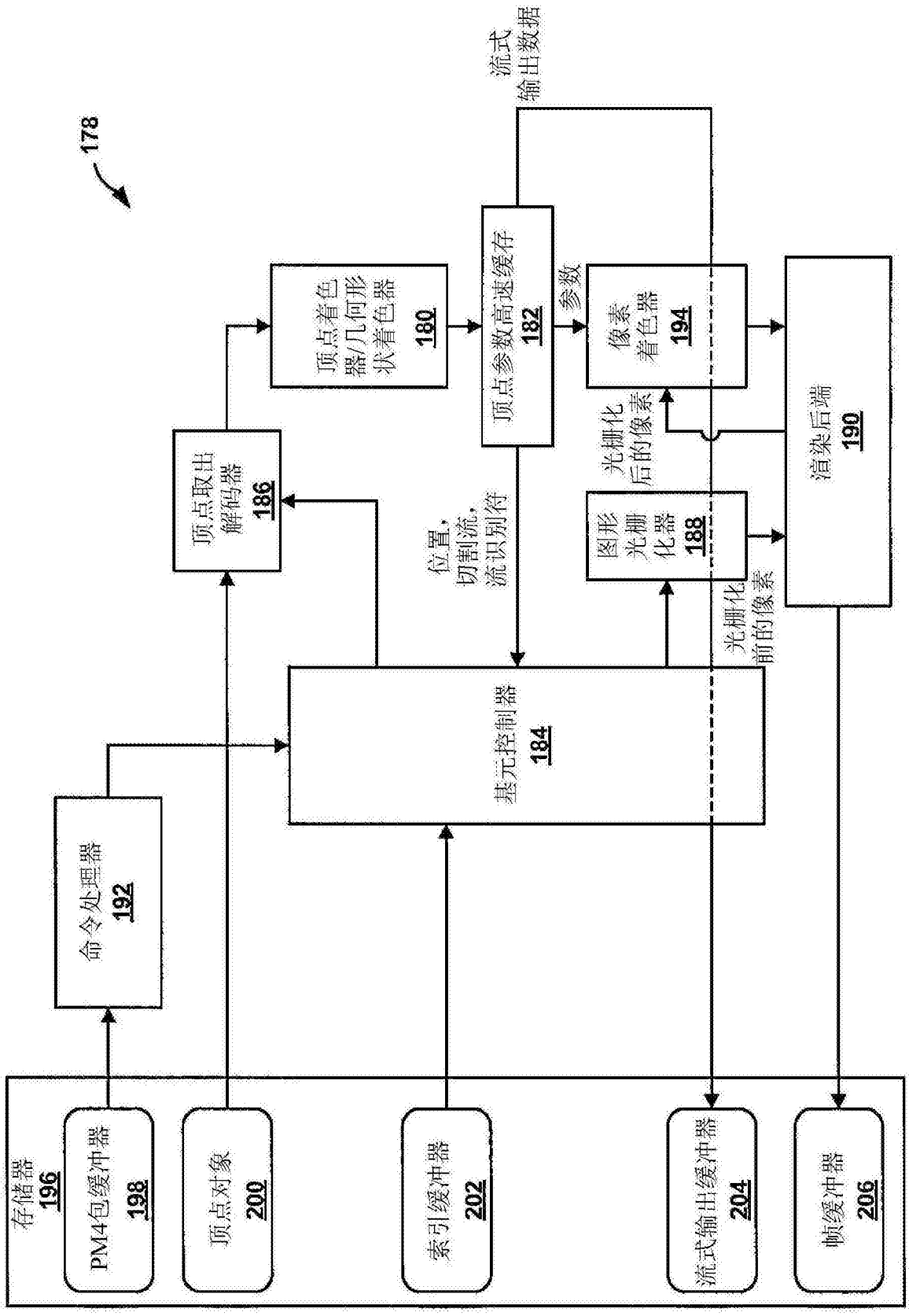

178
流式输出数据
顶点着色器/几何形状着色器 180
顶点参数高速缓存 182
参数
像素着色器 194
渲染后端 190
顶点取出解码器 186
光栅化后的像素
图形光栅化器 188
位置，切割流，流识别符
光栅化前的像素
基元控制器 184
命令处理器 192
存储器 196
PM4包缓冲器 198
顶点对象 200
索引缓冲器 202
流式输出缓冲器 204
帧缓冲器 206

图6

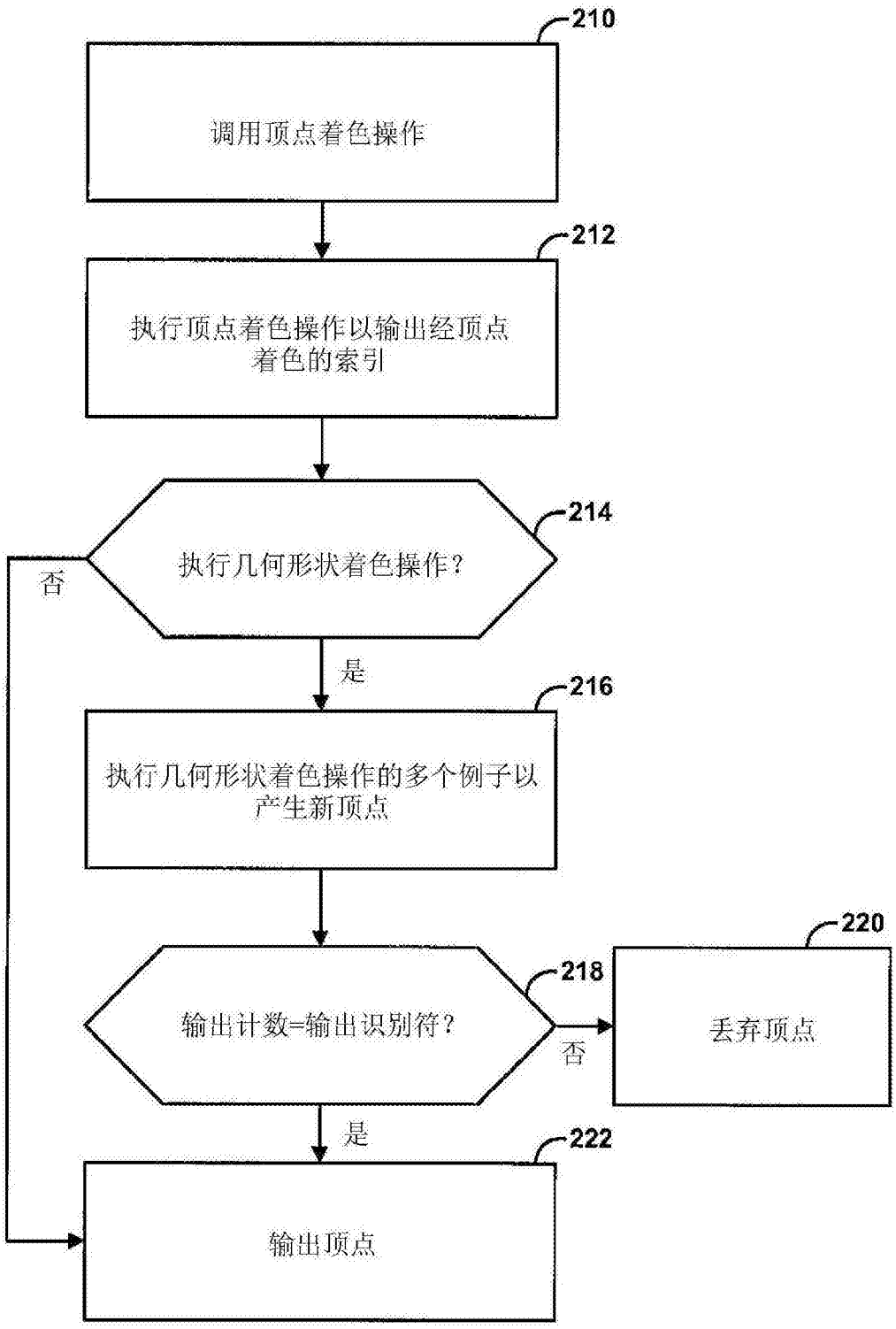

210
调用顶点着色操作
212
执行顶点着色操作以输出经顶点着色的索引
214
执行几何形状着色操作？
否
是
216
执行几何形状着色操作的多个例子以产生新顶点
220
218
输出计数=输出识别符？
丢弃顶点
否
是
222
输出顶点

图7

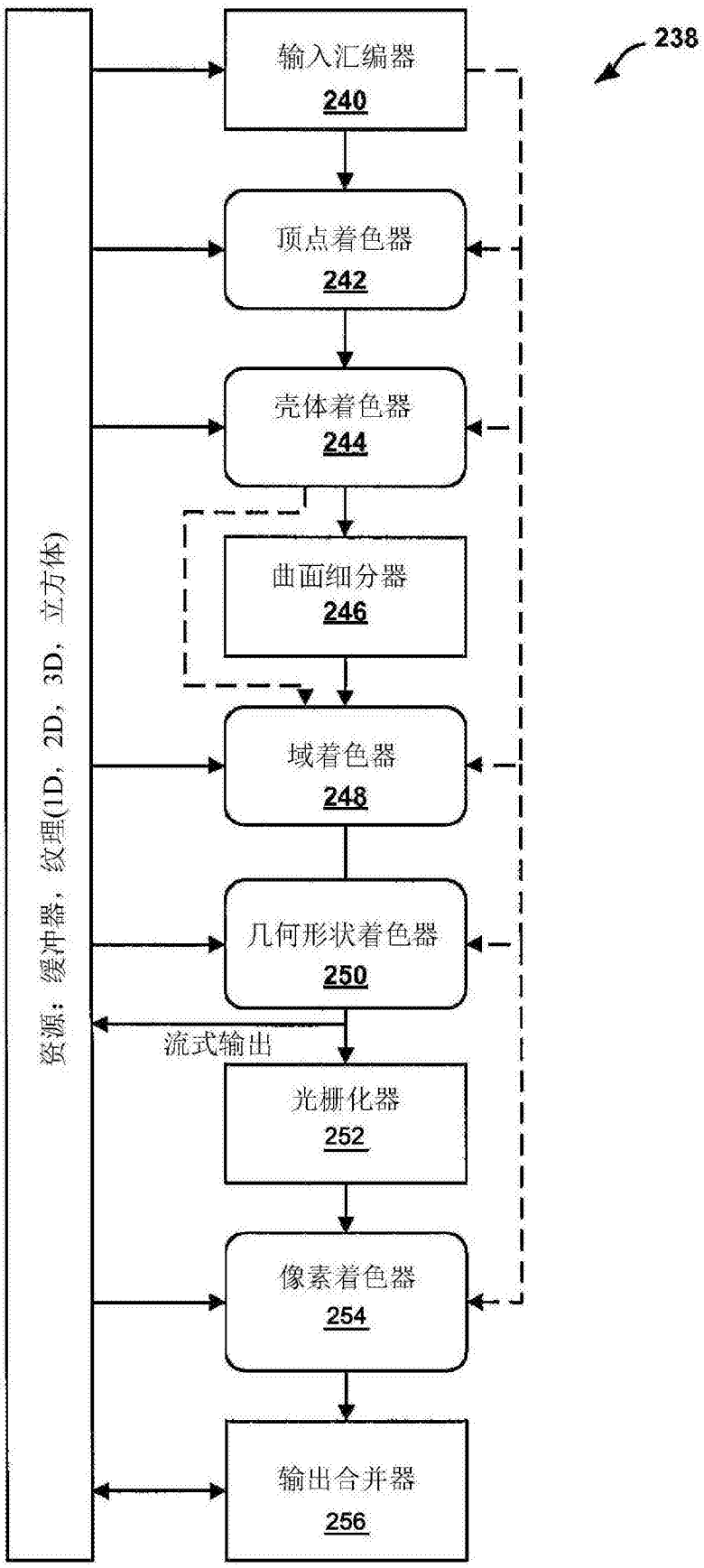

238
输入汇编器
240
顶点着色器
242
壳体着色器
244
曲面细分器
246
域着色器
248
几何形状着色器
250
流式输出
光栅化器
252
像素着色器
254
输出合并器
256
资源：缓冲器，纹理(1D，2D，3D，立方体)

图8

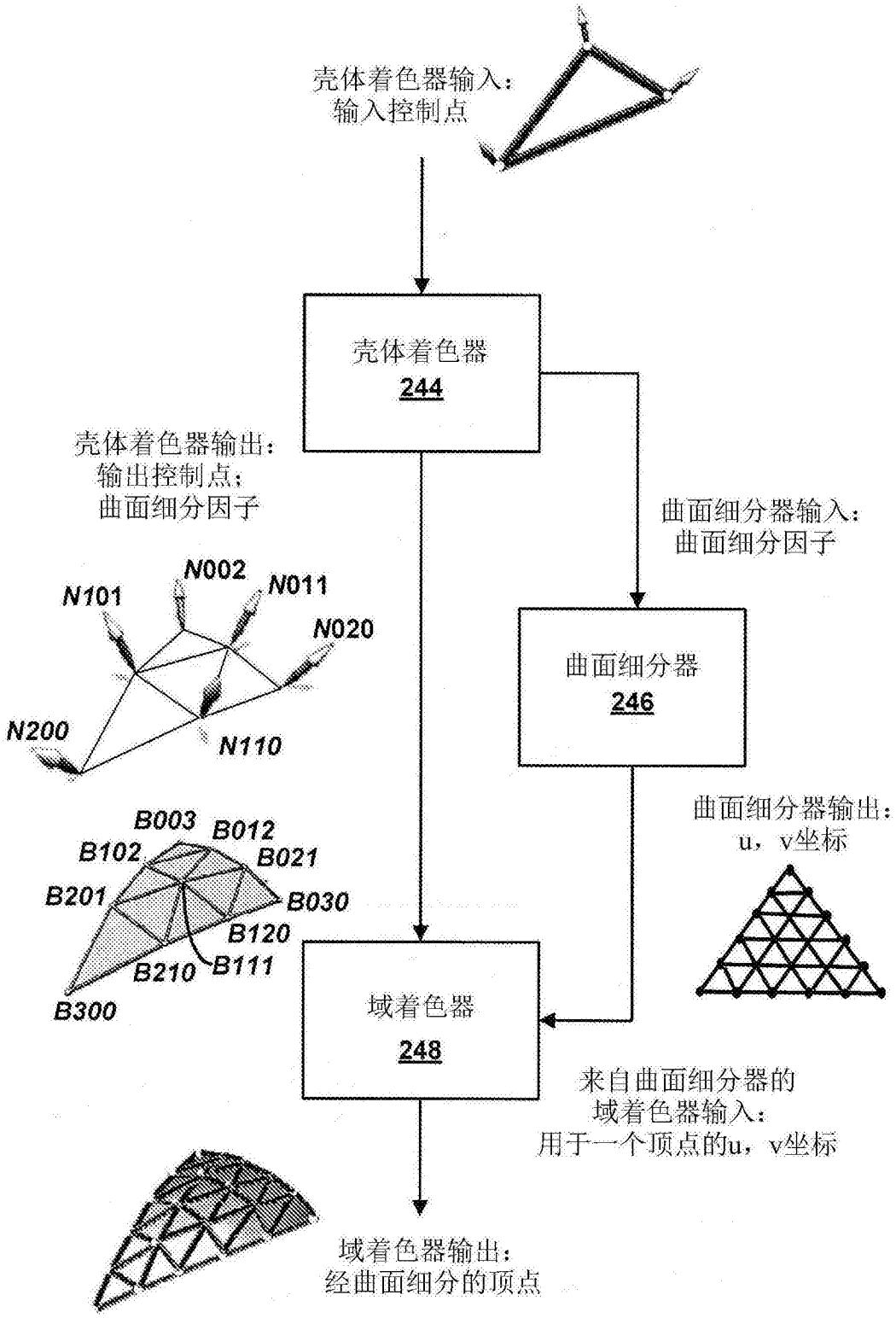

壳体着色器输入：
输入控制点
壳体着色器
244
壳体着色器输出：
输出控制点；
曲面细分因子
曲面细分器输入：
曲面细分因子
N002
N011
N101
N020
N200
N110
曲面细分器
246
曲面细分器输出：
u，v坐标
B003
B012
B102
B021
B201
B030
B120
B111
B210
B300
域着色器
248
来自曲面细分器的
域着色器输入：
用于一个顶点的u，v坐标
域着色器输出：
经曲面细分的顶点

图9

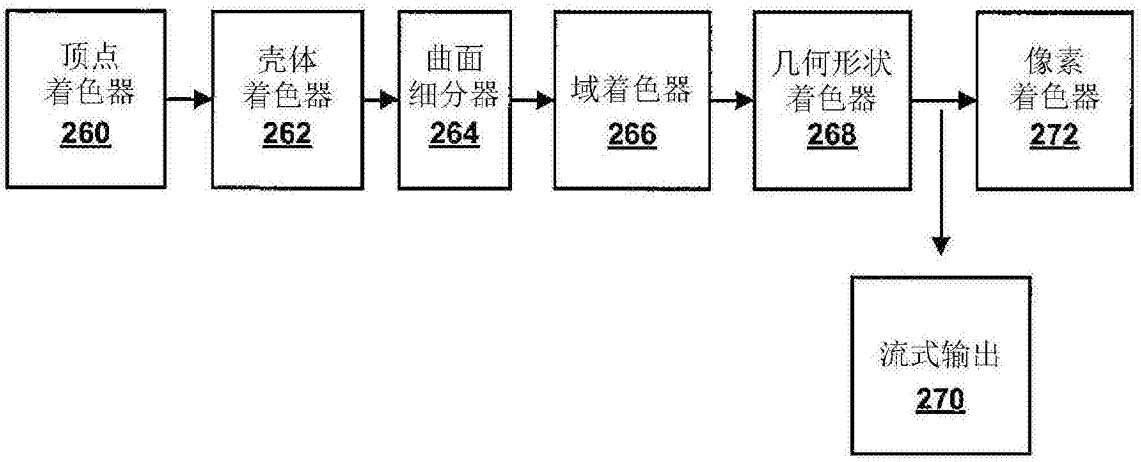

顶点
着色器
260
壳体
着色器
262
曲面
细分器
264
域着色器
266
几何形状
着色器
268
像素
着色器
272
流式输出
270

图10A

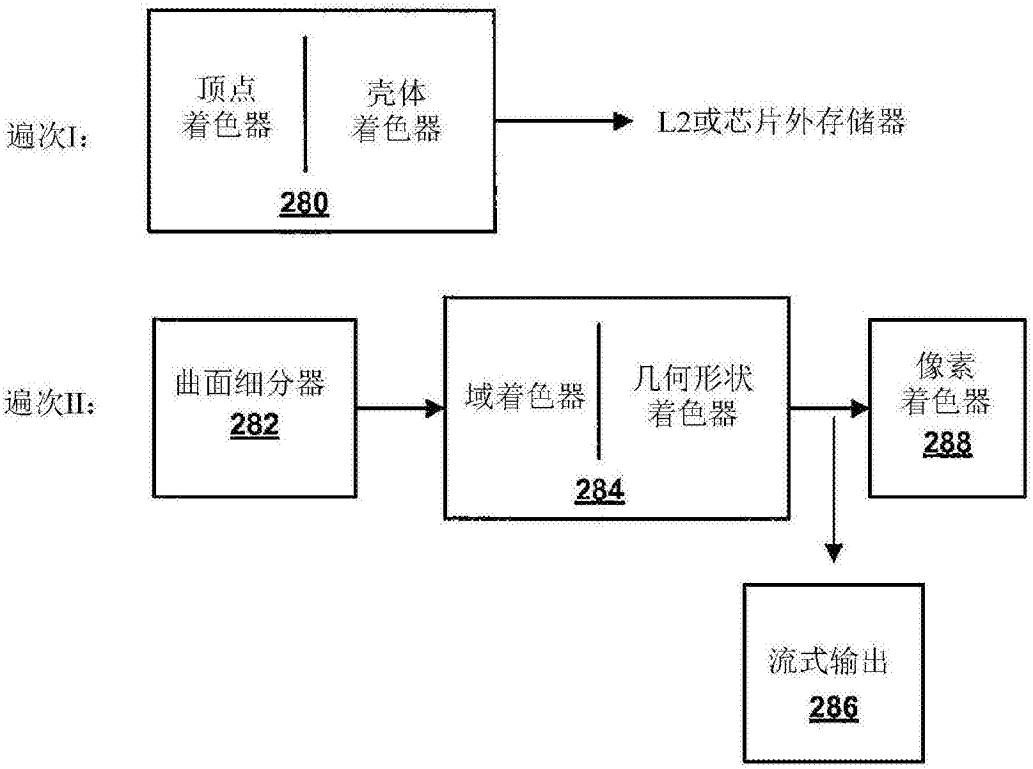

遍次I：
顶点
着色器
壳体
着色器
280
L2或芯片外存储器
遍次II：
曲面细分器
282
域着色器
几何形状
着色器
284
像素
着色器
288
流式输出
286

图10B

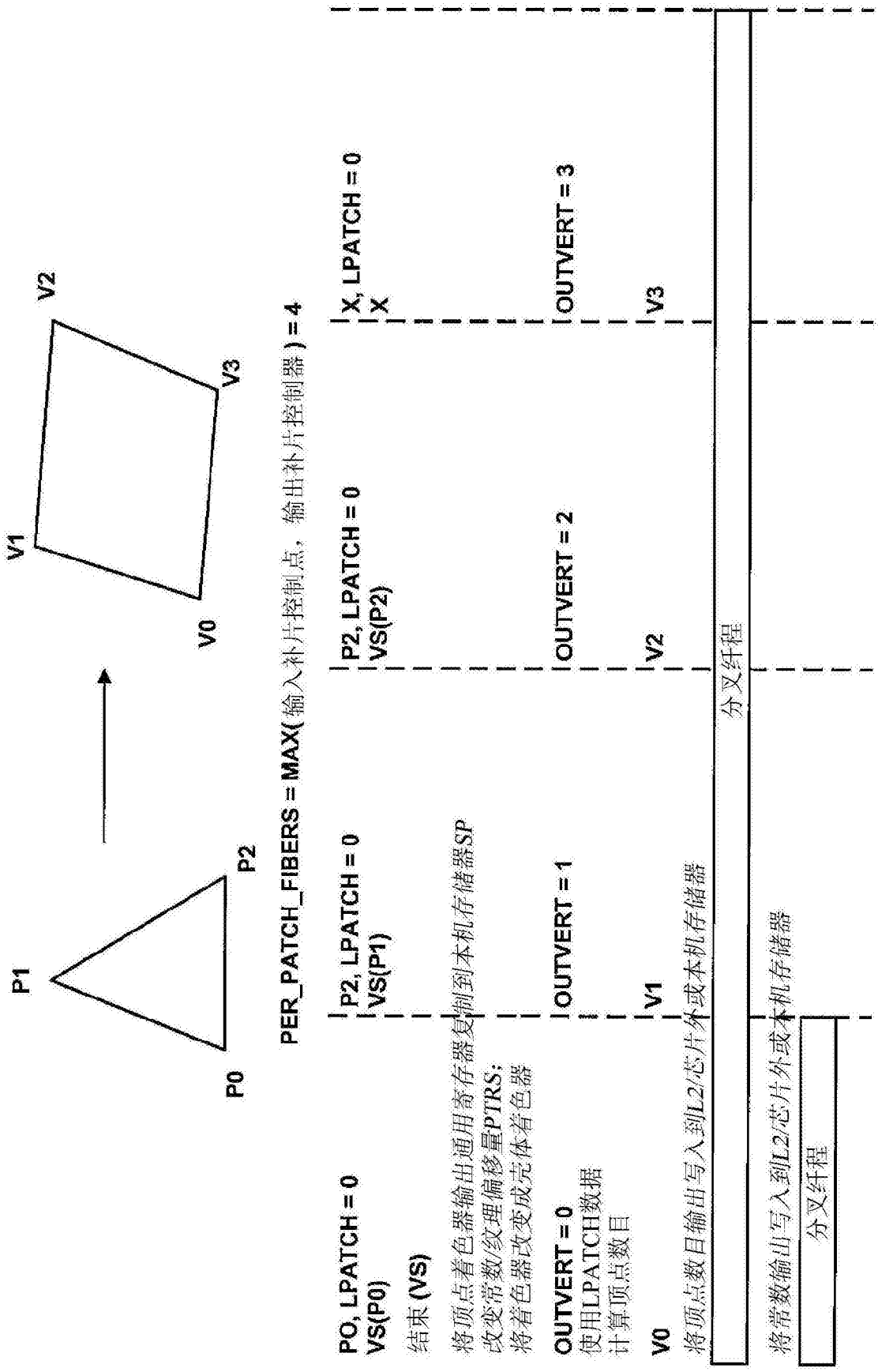

P1
P2
P0
V1
V2
V3
V0
PER_PATCH_FIBERS = MAX(输入补片控制点， 输出补片控制器) = 4
P0, LPATCH = 0
VS(P0)
结束 (VS)
P2, LPATCH = 0
VS(P1)
将顶点着色器输出通用寄存器复制到本机存储器SP
改变常数/纹理偏移量PTRS;
将着色器改变成壳体着色器
OUTVERT = 0
使用LPATCH数据
计算顶点数目
V0
P2, LPATCH = 0
VS(P2)
OUTVERT = 1
V1
将顶点数目输出写入到L2/芯片外或本机存储器
X, LPATCH = 0
X
OUTVERT = 2
V2
OUTVERT = 3
V3
分叉纤程
将常数输出写入到L2/芯片外或本机存储器
分叉纤程

图11


顶点属性，VERTEX_ID,
INSTANCE_ID, PRIMITIVE_ID, MISC
将系统产生的值和MISC信息写入到R0，R1，R2
顶点着色器(VS)
将输出通用寄存器写入到本机存储器
切换到壳体着色器纹理/常数偏移量
切换到壳体着色器程序计数器
从本机存储器读取输入顶点
壳体着色器输出控制点计算
分叉(6)
汇合
壳体着色器控制点和常数数据到L2
曲面细分因子和PRIM_ID到L2

图12A

```
//VFD基于顶点索引和有效顶点取出顶点数据
//其应将顶点数据加载到有效顶点
VS:   “正常”VS代码          // 可以使用instanceID
                            // 数据留在GPR中
      END_FINAL;            // 如果是DX9样式的着色器就结束—否则就忽略

      LM_Vs_offset = rel_primID*PRIM_SIZE + rel_vertex*VERT_SIZE;

      将GPR中的顶点数据写入到LM_Vs_offset

      CHMSK;                // 对第二级使用cov_mask_1，切换到第二
                            // 着色器资源(将资源ptr切换到HS
                            // 资源偏移量)

      CHSH                  // 切换到着色器程序计数器和状态偏移量
                            // (HS PC/偏移量)

HS:   对于 (vertex_id= 0; vertex_id++; vertex_id <= max_input)
      {     使用偏移量 = rel_primID*PRIM_SIZE + vertex_id *  VERT_SIZE 从LM
            加载
            // 使用通过HLSQ计算的基准地址从LM取出
      }

      对于 (hsoutcount =0;hscountcount++;hsoutcount<=maxout)  // 这可以
                                          // 使用 fork(#_output_pnts)进行
      {     // HS代码使用primitiveID计算HS输出
            顶点……
        Mem_offset = rel_patchid*OUTPUT_PRIM_SIZE + outVertID *
        OUTPUT_VERTEX_SIZE + SCRATCH_MEMORY_CONTROL_BASE

        如果(hscount == outVertID)      // 如果补片输出点是依次处理的
                                        // 则进行
            (将所计算的顶点导出到mem_offset)
      }
      对于 (fork_id = outVertID;fork_id += WAVE_OUT_SIZE;fork_id <n)
      {
            Fork(fork_id)         // 在fibers_per_prim当中扩展fork(n)指令
                                  // 以进行并行计算
      }
        如果(outVertID == 0)
      {     汇合                  // 仅使用具有outVertID=0的一个纤程
            Tf_offset = rel_patchid* TF_OUT _SIZE+ tess_attr +
            SCRATCH_MEMORY_TESS_BASE;   // 在曲面细分因子存储器中
                                        // 写入曲面细分因子
            写入primitiveID
            写入曲面细分因子
      }
       END_FINAL;
```

图12B

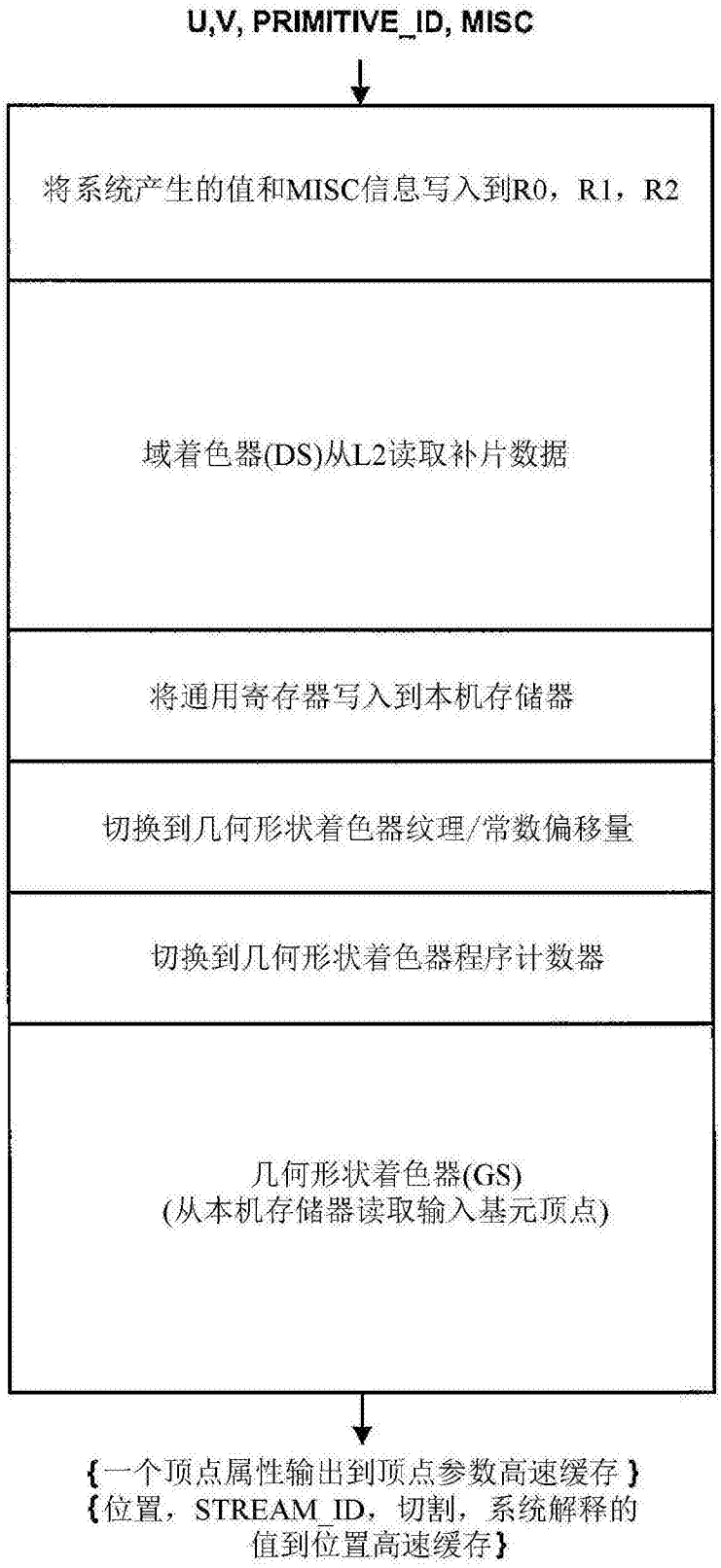

U,V, PRIMITIVE_ID, MISC
将系统产生的值和MISC信息写入到R0，R1，R2
域着色器(DS)从L2读取补片数据
将通用寄存器写入到本机存储器
切换到几何形状着色器纹理/常数偏移量
切换到几何形状着色器程序计数器
几何形状着色器(GS)
(从本机存储器读取输入基元顶点)
{一个顶点属性输出到顶点参数高速缓存}
{位置，STREAM_ID，切割，系统解释的值到位置高速缓存}

图13A

```
// VFD未取出任何数据
// InstanceID = rel_patchID
// (u,v)去往不同的gprs
// 编译器已用期望gpr中的patchID的方式编译DS代码，使得SP可以将真正代表patchID的
instanceID加载到这个gpr
DS:   计算        patch_ptr    =    rel_patchid  *  OUTPUT_PRIM_SIZE    +
      SCRATCH_MEMORY_CONTROL_BASE.
      “正常”DS代码
      // 使用  patch_ptr + OUTPUT_VERTEX_SIZE * control_pointID 存取HS输出控
        制点
      // 使用   patch_ptr + OUTPUT_VERTEX_SIZE * num_output_control_pts
        使用(u,v)存取HS常数数据以计算经曲面细分的点

                  // 数据留在GPR中
      END_FINAL; // 如果是没有GS的DX11 DS着色器就结束—否则的话就忽略
                  // 注意如果着色器结束那么是SP将通用寄存器写入到VPC。

      LM_Vs_offset = rel_primID*PRIM_SIZE + rel_vertex*VERT_SIZE;

      将GPR中的顶点数据写入到 LM_Vs_offset

      CHMSK;      // 将资源ptr切换到GS资源偏移量
                  // 对第二级使用cov_mask_1

      CHSH        // 切换到GS程序计数器和状态偏移量

GS:   对于 (vertex_id= 0; vertex_id++; vertex_id <= max_input)
        {   使用偏移量 = rel_primID*PRIM_SIZE + vertex_id *  VERT_SIZE  从LM
            加载
            // 使用通过HLSQ计算的基准地址从LM取出
        }
      对于(streamid =0;streamid++;streamid<4);
      cut[streamid] = 真 ;
      gsoutount = 0;

// “正常”GS代码
      {.. 使用primitiveID, GSInstanceID
      CUT(streamid):     { Cut[streamid] = 真 ;}
      EMIT(streamid): {
            如果(Gsoutcount==GsoutvertID){
                  oPos = vertex.pos; oColor =vertex.clr;
                  oMisc = (cut[streamid],streamid)
                  gsoutcount = maxoutputvertexcount;
                  去往END_FINAL;  // 任选
            }
            cut[streamid] = 假;
            gsoutcount++;
      }

      如果 (gsoutcount < maxoutputvertexcount)
            杀掉PRIM;  // 消除任何“剩余”纤程

END_FINAL;
```

图13B

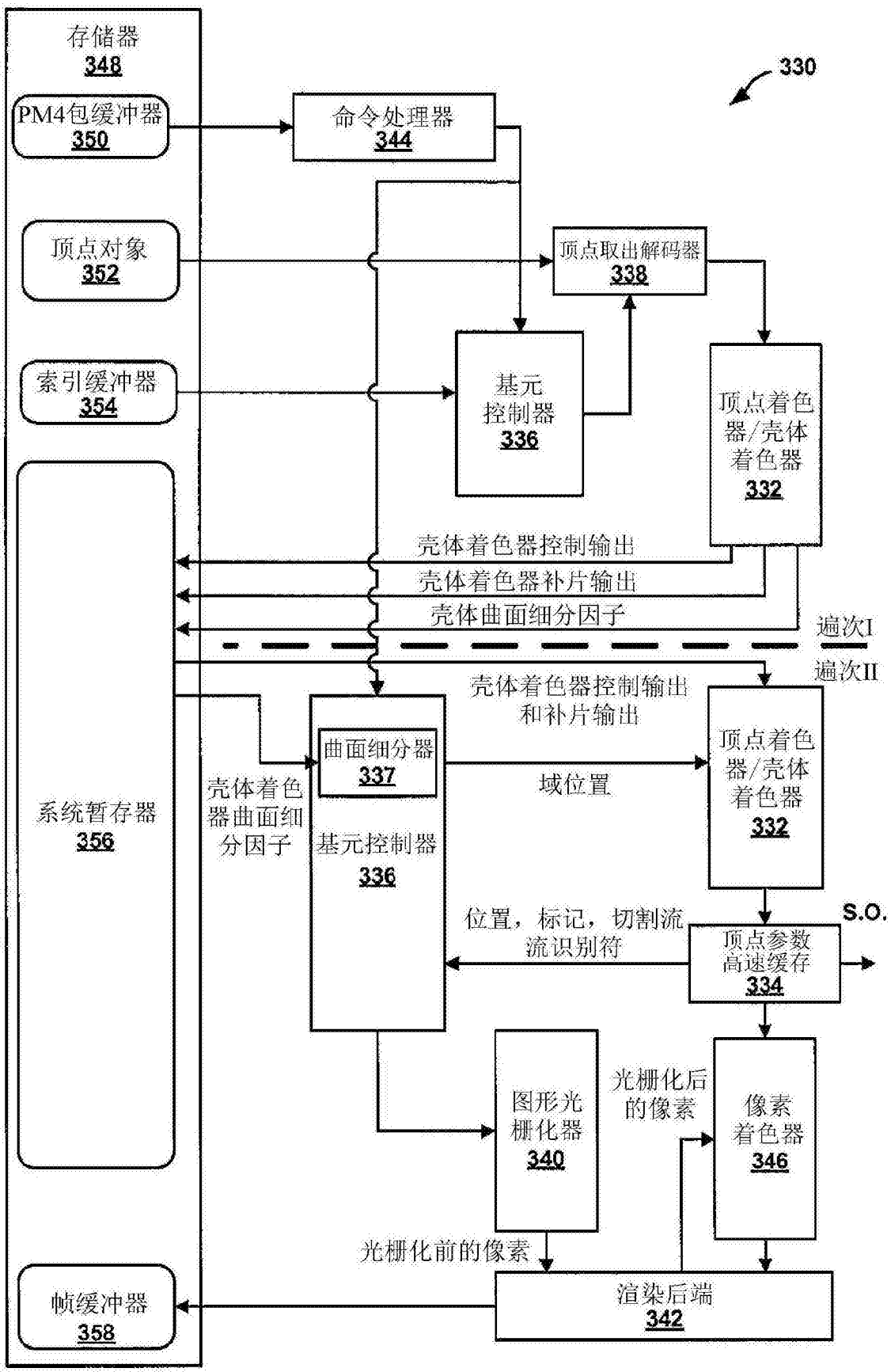

存储器
348
330
PM4包缓冲器
350
命令处理器
344
顶点对象
352
顶点取出解码器
338
索引缓冲器
354
基元
控制器
336
顶点着色
器/壳体
着色器
332
壳体着色器控制输出
壳体着色器补片输出
壳体曲面细分因子
遍次I
遍次II
壳体着色器控制输出
和补片输出
曲面细分器
337
域位置
顶点着色
器/壳体
着色器
332
系统暂存器
356
壳体着色
器曲面细
分因子
基元控制器
336
位置，标记，切割流
流识别符
S.O.
顶点参数
高速缓存
334
图形光
栅化器
340
光栅化后
的像素
像素
着色器
346
光栅化前的像素
帧缓冲器
358
渲染后端
342

图14

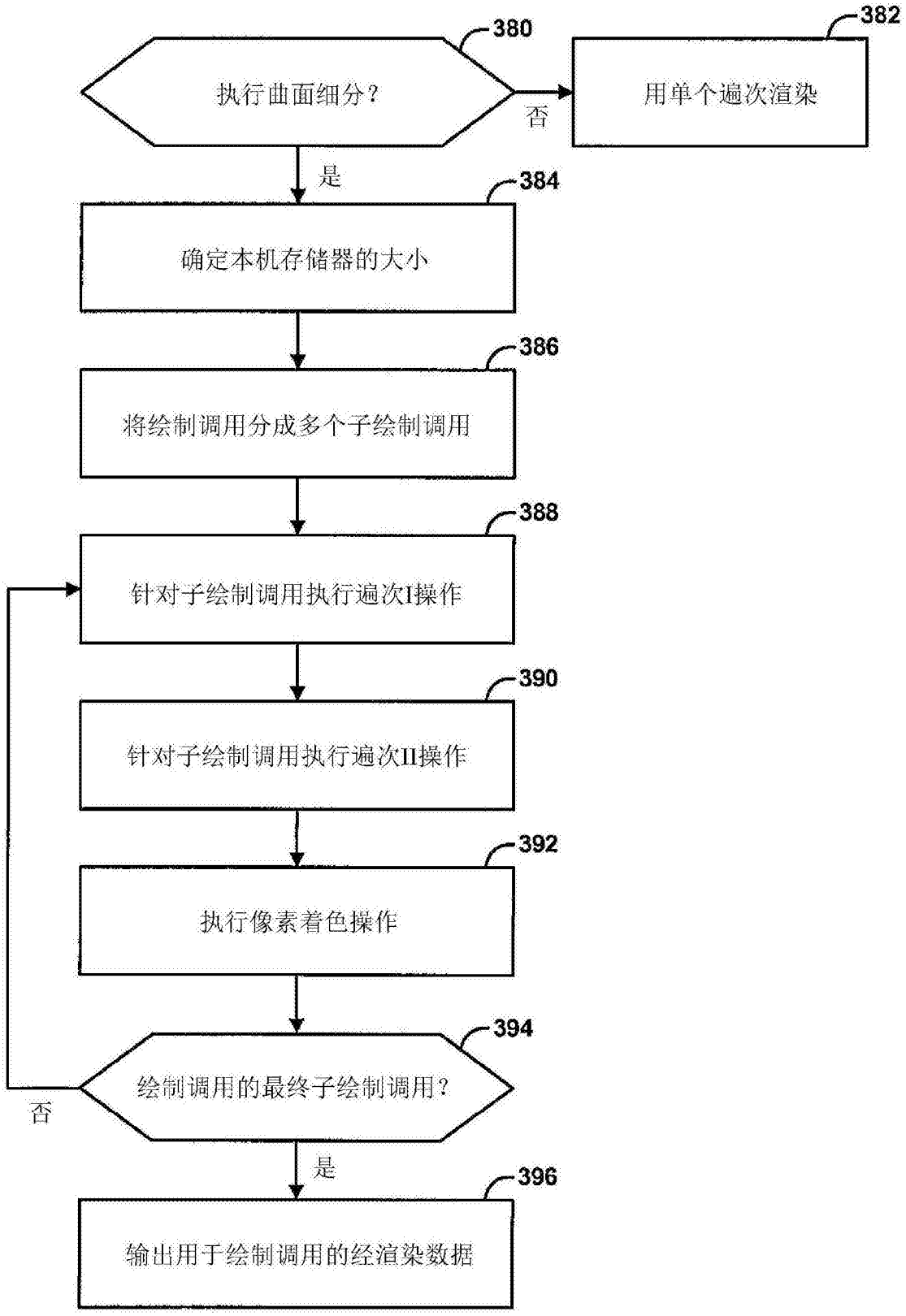

380
执行曲面细分？
否
382
用单个遍次渲染
是
384
确定本机存储器的大小
386
将绘制调用分成多个子绘制调用
388
针对子绘制调用执行遍次I操作
390
针对子绘制调用执行遍次II操作
392
执行像素着色操作
394
绘制调用的最终子绘制调用？
否
是
396
输出用于绘制调用的经渲染数据

图15

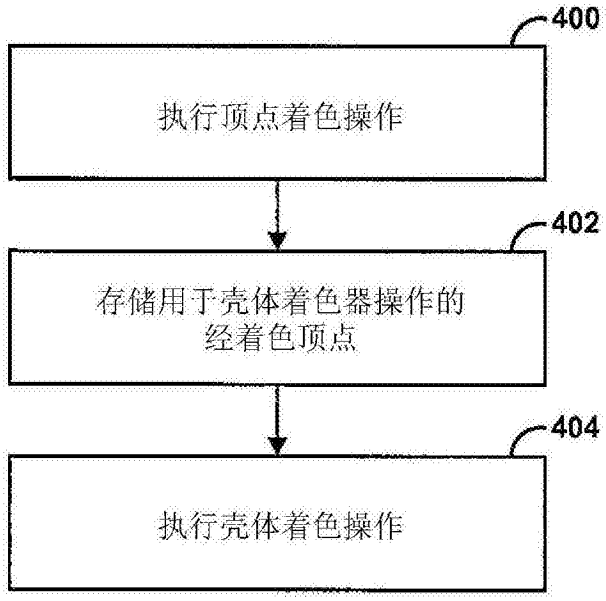

400
执行顶点着色操作
402
存储用于壳体着色器操作的
经着色顶点
404
执行壳体着色操作

图16

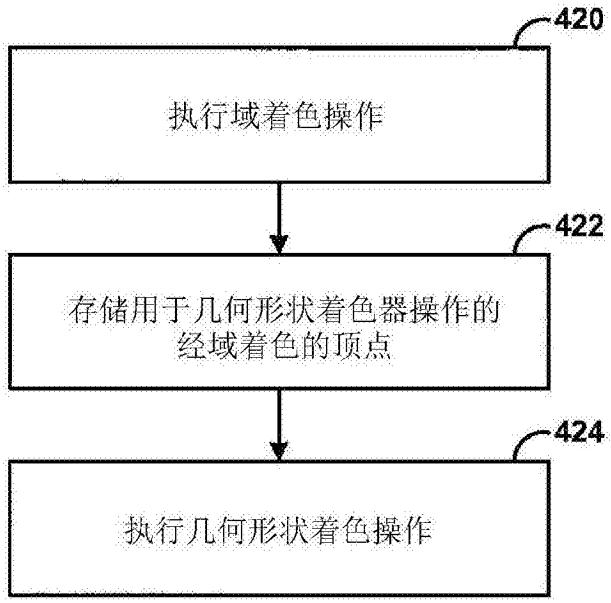

420
执行域着色操作
422
存储用于几何形状着色器操作的
经域着色的顶点
424
执行几何形状着色操作

图17

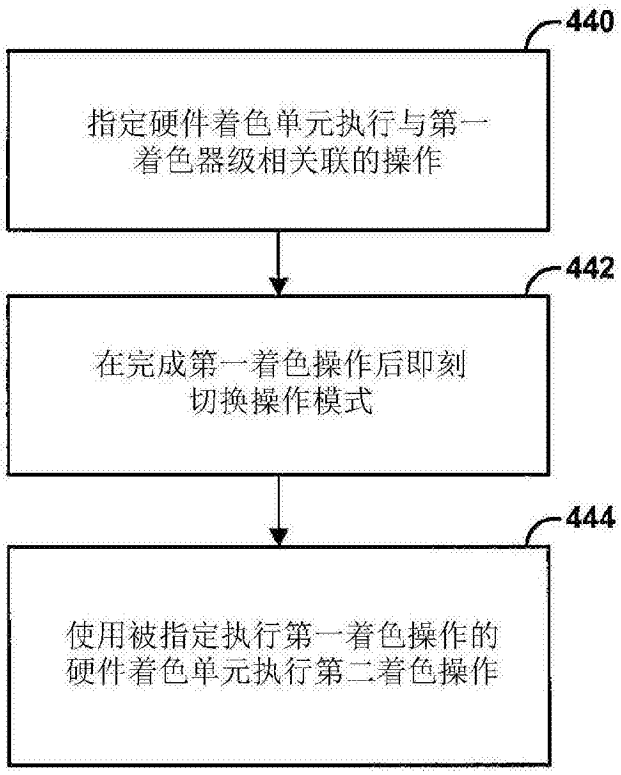

440
指定硬件着色单元执行与第一着色器级相关联的操作
442
在完成第一着色操作后即刻切换操作模式
444
使用被指定执行第一着色操作的硬件着色单元执行第二着色操作

图18