(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 113837920 A
(43)申请公布日 2021.12.24

(21)申请号 202110951444.3

(22)申请日 2021.08.18

(71)申请人 荣耀终端有限公司
地址 518040 广东省深圳市福田区香蜜湖街道东海社区红荔西路8089号深业中城6号楼A单元3401

(72)发明人 陈聪儿 刘金晓

(74)专利代理机构 北京中博世达专利商标代理有限公司 11274
代理人 申健

(51)Int.Cl.
G06T 1/20(2006.01)
G06T 1/60(2006.01)
G06T 15/20(2011.01)
G06T 15/80(2011.01)

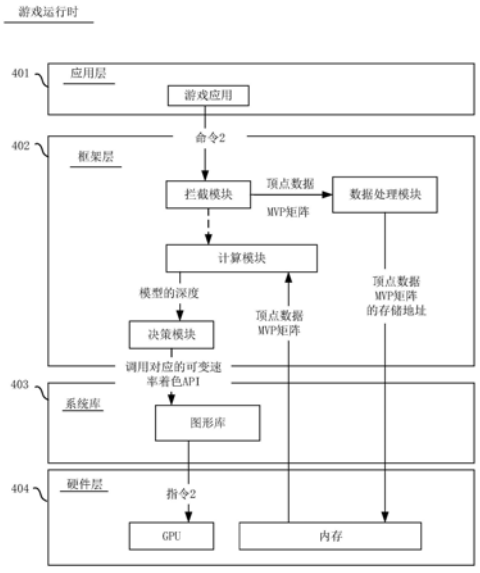
权利要求书3页 说明书31页 附图11页

(54)发明名称

一种图像渲染方法及电子设备

(57)摘要

本申请实施例公开了一种图像渲染方法及电子设备，涉及图像处理领域，可以实现通过可变速率着色机制降低渲染开销，同时不会由于着色速率的降低产生对用户体验的影响。具体方案为：获取该应用程序下发的第一渲染命令，该第一渲染命令用于绘制第一模型。确定该第一模型的第一着色速率。根据该第一渲染命令和该第一着色速率绘制该第一模型。获取该应用程序下发的第二渲染命令，该第二渲染命令用于绘制第二模型。确定该第二模型的第二着色速率。根据该第二渲染命令和该第二着色速率绘制该第二模型。该第一模型和该第二模型包括在同一个帧图像中，该第一着色速率和该第二着色速率不同。



CN 113837920 A

1.一种图像渲染方法,其特征在于,应用于电子设备,所述电子设备中安装有应用程序,所述方法包括:

获取所述应用程序下发的第一渲染命令,所述第一渲染命令用于绘制第一模型;

确定所述第一模型的第一着色速率;

根据所述第一渲染命令和所述第一着色速率绘制所述第一模型;

获取所述应用程序下发的第二渲染命令,所述第二渲染命令用于绘制第二模型;

确定所述第二模型的第二着色速率;

根据所述第二渲染命令和所述第二着色速率绘制所述第二模型;

所述第一模型和所述第二模型包括在同一个帧图像中,所述第一着色速率和所述第二着色速率不同。

2.根据权利要求1所述的方法,其特征在于,所述确定所述第一模型的第一着色速率,包括:

根据所述第一渲染命令,确定所述第一模型的深度,所述第一模型的深度用于标识所述第一模型在观察空间或者裁剪空间中到观察者之间的距离,所述第一模型的深度越大,所述第一模型到所述观察者之间的距离越远;

根据所述第一模型的深度,确定所述第一着色速率。

3.根据权利要求2所述的方法,其特征在于,所述根据所述第一渲染命令,确定所述第一模型的深度,包括:

根据所述第一渲染命令,确定所述第一模型的n个顶点的深度;

根据所述n个顶点的深度,确定所述第一模型的深度;

其中,n个顶点包括在所述第一模型的顶点中。

4.根据权利要求1-3中任一项所述的方法,其特征在于,所述电子设备中配置有拦截模以及数据处理模块,

所述获取所述应用程序下发的第一渲染命令,包括;

所述拦截模块拦截所述第一渲染命令;

所述拦截模块将拦截的第一函数和第二函数传输给所述数据处理模块;

所述第一函数是携带第一参数的函数,所述第二函数是携带第二参数的函数,所述第一参数是所述应用程序传递顶点数据过程中携带的参数,所述第二参数是所述应用程序传递MVP矩阵过程中携带的参数。

5.根据权利要求4所述的方法,其特征在于,所述方法还包括:

所述拦截模块将所述第一渲染命令中,所述第一函数和所述第二函数之外的函数回调给所述电子设备的图形库。

6.根据权利要求4或5所述的方法,其特征在于,在所述拦截模块将拦截的第一函数和第二函数传输给所述数据处理模块之后,所述方法还包括:

所述数据处理模块根据所述第一函数和所述第二函数,确定所述第一模型对应的顶点坐标的第一存储位置,以及所述第一MVP矩阵的第二存储位置,所述第一存储位置和所述第二存储位置包括在第一存储区域中,所述第一存储区域是所述电子设备的处理器能够调取的存储区域。

7.根据权利要求6所述的方法,其特征在于,所述方法还包括:

所述数据处理模块将所述第一函数和第二函数回调给所述电子设备的图形库。

8.根据权利要求6或7所述的方法,其特征在于,在获取所述应用程序下发的第一渲染命令之前,所述方法还包括:

所述拦截模块拦截所述应用程序下发的第三渲染命令,所述第三渲染命令用于将所述应用程序运行过程中使用的第一数据存储在第二存储区域中,所述第二存储区域是所述电子设备的GPU使用的存储区域,所述第一数据包括所述第一模型的顶点坐标以及所述第一MVP矩阵;

所述拦截模块将第三函数和第四函数传输给数据处理模块,所述第三函数携带所述第一参数,所述第四函数携带所述第二参数;

所述数据处理模块将所述第三函数和所述第四函数对应的数据存储在所述第一存储区域中。

9.根据权利要求8所述的方法,其特征在于,所述方法还包括:

所述拦截模块将所述第三渲染命令中,所述第三函数和所述第四函数之外的函数回调给所述电子设备的图形库。

10.根据权利要求8或9所述的方法,其特征在于,所述方法还包括:

所述数据处理模块将所述第三函数和所述第四函数回调给所述电子设备的图形库。

11.根据权利要求8-10中任一项所述的方法,其特征在于,所述方法还包括:

所述数据处理模块存储第一对应关系,所述第一对应关系用于指示相同数据在所述第一存储区域的存储地址与在所述第二存储区域的存储地址之间的对应关系。

12.根据权利要求6-11中任一项所述的方法,其特征在于,所述数据处理模块根据所述第一函数和所述第二函数,确定所述第一模型对应的顶点坐标的第一存储位置,以及所述第一MVP矩阵的第二存储位置,包括:

所述数据处理模块根据所述第一函数指示的缓存位置,以及所述第一对应关系,确定所述第一模型对应的顶点坐标的第一存储位置;

所述数据处理模块根据所述第二函数指示的缓存位置,以及所述第一对应关系,确定所述第一MVP矩阵的第二存储位置。

13.根据权利要求12所述的方法,其特征在于,所述电子设备中还配置有计算模块,所述根据所述第一渲染命令,确定所述第一模型的n个顶点的深度,包括:

所述拦截模块在拦截到所述第一渲染命令中的绘制元素Drawelement的情况下,

所述计算模块根据所述第一模型对应的顶点坐标,以及所述第一MVP矩阵,计算所述n个顶点的深度;其中,所述第一模型对应的顶点数据是所述计算模块从所述第一存储位置获取的,所述第一MVP矩阵是所述计算模块从所述第二存储位置获取的。

14.根据权利要求13所述的方法,其特征在于,所述根据所述n个顶点的深度,确定所述第一模型的深度,包括:

所述计算模块将所述第一模型的深度确定为所述n个顶点的深度的均值。

15.根据权利要求13或14所述的方法,其特征在于,所述电子设备中还配置有决策模块,所述方法还包括:

所述计算模块将所述第一模型的深度传输给所述决策模块;

所述决策模块根据所述第一模型的深度,从预设的第二对应关系中,查找与所述第一

模型的深度匹配的表项；

所述决策模块将所述匹配的表项指示的着色速率确定为所述第一着色速率。

16.根据权利要求15所述的方法，其特征在于，所述方法还包括：

所述决策模块向所述图形库传输指示所述第一着色速率的着色指令。

17.根据权利要求16所述的方法，其特征在于，所述根据所述第一渲染命令和所述第一着色速率绘制所述第一模型，包括：

所述图形库根据所述拦截模块和所述数据处理模块回调的函数，调用与所述第一渲染命令对应的第一应用编程接口API；

所述图形库根据所述着色指令调用与所述第一着色速率对应的第一可变速率着色API；

所述图形库根据所述第一API以及所述第一可变速率着色API，向所述电子设备的GPU下发渲染指令，以便于所述GPU根据所述第一着色速率执行对所述第一模型的渲染操作。

18.一种电子设备，其特征在于，所述电子设备包括一个或多个处理器和一个或多个存储器；所述一个或多个存储器与所述一个或多个处理器耦合，所述一个或多个存储器存储有计算机指令；

当所述一个或多个处理器执行所述计算机指令时，使得所述电子设备执行如权利要求1-17中任一项所述的图像渲染方法。

一种图像渲染方法及电子设备

技术领域

[0001] 本申请涉及图像处理技术领域,尤其涉及一种图像渲染方法及电子设备。

背景技术

[0002] 电子设备在对图像进行渲染处理时,包括对图像的着色处理。示例性的,电子设备的图形处理器(graphics processing unit,GPU)可以分别对图像的每个像素进行着色,进而完成对整个图像的着色处理。

[0003] 随着图像的像素的提高,对图像的着色处理会对电子设备产生较高的渲染负荷,如增加渲染过程中的算力和功耗开销。

发明内容

[0004] 本申请实施例提供一种图像渲染方法和电子设备,可以实现通过可变速率着色机制降低渲染开销,同时不会由于着色速率的降低产生对用户体验的影响。

[0005] 为了达到上述目的,本申请实施例采用如下技术方案:

[0006] 第一方面,提供一种图像渲染方法,应用于电子设备,该电子设备中安装有应用程序,该方法包括:获取该应用程序下发的第一渲染命令,该第一渲染命令用于绘制第一模型。确定该第一模型的第一着色速率。根据该第一渲染命令和该第一着色速率绘制该第一模型。获取该应用程序下发的第二渲染命令,该第二渲染命令用于绘制第二模型。确定该第二模型的第二着色速率。根据该第二渲染命令和该第二着色速率绘制该第二模型。该第一模型和该第二模型包括在同一个帧图像中,该第一着色速率和该第二着色速率不同。

[0007] 基于该方案,提供了一种对不同的模型进行不同速率着色的方案示例。在本示例中,电子设备可以根据对不同模型的渲染命令,确定各个模型的着色速率。进而以渲染命令(即Drawcall)为粒度,对模型的绘制等渲染操作进行不同的速率着色,进而对部分模型进行低速率着色,由此节省着色过程中的渲染开销,如算力开销,发热等。

[0008] 在一种可能的设计中,该确定该第一模型的第一着色速率,包括:根据该第一渲染命令,确定该第一模型的深度,该第一模型的深度用于标识该第一模型在观察空间或者裁剪空间中到观察者之间的距离,该第一模型的深度越大,该第一模型到该观察者之间的距离越远。根据该第一模型的深度,确定该第一着色速率。基于该方案,提供了一种确定不同模型的着色速率的机制。示例性的,可以根据不同模型的深度,确定对应的着色速率。可以理解的是,深度越大,则该模型在图像中距离用户就越远,因此可以采用较低速率着色,节省渲染开销。

[0009] 在一种可能的设计中,该根据该第一渲染命令,确定该第一模型的深度,包括:根据该第一渲染命令,确定该第一模型的n个顶点的深度。根据该n个顶点的深度,确定该第一模型的深度。其中,n个顶点包括在该第一模型的顶点中。基于该方案,提供了一种确定模型深度的方案示例。示例性的,可以根据模型的顶点的深度,确定模型的深度。其中,模型的顶点的深度可以是观察空间或者裁剪空间或者其他能够用于标识顶点到用户距离的深度。在

本示例中,n可以等于模型的最大顶点数。这样就可以根据模型的所有顶点深度,确定模型的深度。在另一些实施例中,n可以小于模型的最大顶点数,如n个顶点可以是模型的所有顶点中随机选取的n个顶点。由此可以节省计算模型顶点过程中的计算量。

[0010] 在一种可能的设计中,该电子设备中配置有拦截模以及数据处理模块,该获取该应用程序下发的第一渲染命令,包括。该拦截模块拦截该第一渲染命令。该拦截模块将拦截的第一函数和第二函数传输给该数据处理模块。该第一函数是携带第一参数的函数,该第二函数是携带第二参数的函数,该第一参数是该应用程序传递顶点数据过程中携带的参数,该第二参数是该应用程序传递MVP矩阵过程中携带的参数。基于该方案,提供了一种具体的拦截命令的方案示例。在本示例中,拦截模块可以拦截全量的第一渲染命令,并将第一渲染命令中携带第一参数和第二参数的指令(或函数)传输给数据处理模块,实现对有用数据的拦截。其中有用数据可以是顶点相关数据以及MVP矩阵。

[0011] 在一种可能的设计中,该方法还包括:该拦截模块将该第一渲染命令中,该第一函数和该第二函数之外的函数回调给该电子设备的图形库。基于该方案,提供了一种回调机制。可以理解的是,在原生逻辑中,第一渲染命令可以被下发给图形库进行处理。在本示例中,拦截模块可以在拦截第一函数和第二函数之后,将其他部分命令传输给图形库,以保证正常渲染逻辑的运行。

[0012] 在一种可能的设计中,在该拦截模块将拦截的第一函数和第二函数传输给该数据处理模块之后,该方法还包括:该数据处理模块根据该第一函数和该第二函数,确定该第一模型对应的顶点坐标的第一存储位置,以及该第一MVP矩阵的第二存储位置,该第一存储位置和该第二存储位置包括在第一存储区域中,该第一存储区域是该电子设备的处理器能够调取的存储区域。基于该方案,提供了一种确定顶点坐标和MVP矩阵的方案示例。例如,携带第一参数的第一函数,可以用于指示当前Drawcall所对应模型的顶点坐标的缓存ID。那么数据处理模块就可以根据该第一函数,解析出第一模型的顶点坐标的缓存ID。进而根据预设的或者第一函数指示的属性以及偏移量等参数,确定顶点坐标的具体存储位置。数据处理模块还可以基于类似的逻辑获取MVP矩阵。其中,数据处理模块确定的存储位置可以是备份存储的数据的存储位置,由此可以保证其他模块的正常调用。

[0013] 在一种可能的设计中,该方法还包括:该数据处理模块将该第一函数和第二函数回调给该电子设备的图形库。基于该方案,提供了又一种回调机制。示例性的,拦截模块已经将第一命令中除第一函数和第二函数之外的函数进行了回调,那么数据处理模块可以回调第一函数和第二函数,实现第一渲染命令的全量回调,从而保证渲染逻辑的准确执行。

[0014] 在一种可能的设计中,在获取该应用程序下发的第一渲染命令之前,该方法还包括:该拦截模块拦截该应用程序下发的第三渲染命令,该第三渲染命令用于将该应用程序运行过程中使用的第一数据存储在第二存储区域中,该第二存储区域是该电子设备的GPU使用的存储区域,该第一数据包括该第一模型的顶点坐标以及该第一MVP矩阵。该拦截模块将第三函数和第四函数传输给数据处理模块,该第三函数携带该第一参数,该第四函数携带该第二参数。该数据处理模块将该第三函数和该第四函数对应的数据存储在该第一存储区域中。基于该方案,提供了一种备份存储数据的机制。可以理解的是,在上述第一渲染命令(或者第二渲染命令)中,可以通过调用第三渲染命令已经加载到GPU的数据实现对应的渲染指示。该加载到GPU的数据对于其他模块是不可见的,因此,在本示例中,通过该备份存

储机制，在CPU可调度的存储空间中备份存储加载的数据，从而保证在游戏运行过程中，各个模块在需要确定模型深度时，对该加载数据的顺利调用。

[0015] 在一种可能的设计中，该方法还包括：该拦截模块将该第三渲染命令中，该第三函数和该第四函数之外的函数回调给该电子设备的图形库。基于该方案，提供了又一种回调机制。由此使得在加载过程中，电子设备可以对第三函数和第四函数之外的指令进行正确加载。

[0016] 在一种可能的设计中，该方法还包括：该数据处理模块将该第三函数和该第四函数回调给该电子设备的图形库。基于该方案，提供了又一种回调机制。由此使得在加载过程中，电子设备可以对第三函数和第四函数进行正确加载。示例性的，该回调机制的执行可以是数据处理模块完成第三函数和第四函数的备份存储之后进行的。

[0017] 在一种可能的设计中，该方法还包括：该数据处理模块存储第一对应关系，该第一对应关系用于指示相同数据在该第一存储区域的存储地址与在该第二存储区域的存储地址之间的对应关系。基于该方案，提供了一种数据正确调用的机制。可以理解的是，在备份存储的过程中，不能将数据存储在内存中为GPU配置的区域。在本示例中，可以通过存储备份存储的地址，与原生命令中指示的地址之间的对应关系，使得后续可以根据原生命令准确找到备份存储的对应的数据。

[0018] 在一种可能的设计中，该数据处理模块根据该第一函数和该第二函数，确定该第一模型对应的顶点坐标的第一存储位置，以及该第一MVP矩阵的第二存储位置，包括：该数据处理模块根据该第一函数指示的缓存位置，以及该第一对应关系，确定该第一模型对应的顶点坐标的第一存储位置。该数据处理模块根据该第二函数指示的缓存位置，以及该第一对应关系，确定该第一MVP矩阵的第二存储位置。基于该方案，提供了一种确定当前Drawcall对应顶点坐标和MVP矩阵的方案示例，比如，结合当前命令指示的缓存位置，以及对应关系，就可以在备份存储中找到对应的数据。该数据就可以是当前Drawcall所指示的第一模型的顶点坐标和MVP矩阵。

[0019] 在一种可能的设计中，该电子设备中还配置有计算模块，该根据该第一渲染命令，确定该第一模型的n个顶点的深度，包括：该拦截模块在拦截到该第一渲染命令中的绘制元素Drawelement的情况下，该计算模块根据该第一模型对应的顶点坐标，以及该第一MVP矩阵，计算该n个顶点的深度。其中，该第一模型对应的顶点数据是该计算模块从该第一存储位置获取的，该第一MVP矩阵是该计算模块从该第二存储位置获取的。基于该方案，提供了一种计算模型深度的触发机制。比如，可以在拦截到应用下发Drawelement的情况下，确定已经完成了顶点坐标，MVP矩阵等数据的指示，由此即可根据上述方案中确定的当前Drawcall指示的顶点坐标和MVP矩阵的存储位置，调取对应的数据进行计算。

[0020] 在一种可能的设计中，该根据该n个顶点的深度，确定该第一模型的深度，包括：该计算模块将该第一模型的深度确定为该n个顶点的深度的均值。基于该方案，提供了一种具体的根据顶点深度确定模型深度的方案示例。比如模型深度可以为选取的n个顶点深度的均值。

[0021] 在一种可能的设计中，该电子设备中还配置有决策模块，该方法还包括：该计算模块将该第一模型的深度传输给该决策模块。该决策模块根据该第一模型的深度，从预设的第二对应关系中，查找与该第一模型的深度匹配的表项。该决策模块将该匹配的表项指示

的着色速率确定为该第一着色速率。基于该方案，提供了一种根据模型深度确定对应的着色速率的方案示例。例如，根据模型深度所处的预设的深度范围，可以在第二对应关系中，找到匹配的表项，该表象可以指示一个预设的着色速率。那么该预设的着色速率就可以是与当前模型的深度对应的第一着色速率。

[0022] 在一种可能的设计中，该方法还包括：该决策模块向该图形库传输指示该第一着色速率的着色指令。基于该方案，提供了一种触发根据第一着色速率进行着色的方案机制。比如，在确定第一模型使用第一着色速率之后，决策模块可以调用图形库中与第一着色速率对应的API。在一些实现中，可以通过发送第一着色速率对应的着色指令实现该API的调用。

[0023] 在一种可能的设计中，该根据该第一渲染命令和该第一着色速率绘制该第一模型，包括：该图形库根据该拦截模块和该数据处理模块回调的函数，调用与该第一渲染命令对应的第一应用编程接口API。该图形库根据该着色指令调用与该第一着色速率对应的第一可变速率着色API。该图形库根据该第一API以及该第一可变速率着色API，向该电子设备的GPU下发渲染指令，以便于该GPU根据该第一着色速率执行对该第一模型的渲染操作。基于该方案，提供了一种GPU对第一模型执行可变速率着色的方案机制。示例性的，基于前述回调机制，GPU可以收到第一渲染命令(第二渲染命令)的所有指示。此外，GPU还可以接收到根据第一着色速率调取的对应API所指示的指令。因此，GPU就可以实现根据第一着色速率对第一渲染命令所指示绘制的第一模型进行渲染操作。实现对绘制模型的着色速率的灵活调整。

[0024] 第二方面，提供一种电子设备，电子设备包括一个或多个处理器和一个或多个存储器；一个或多个存储器与一个或多个处理器耦合，一个或多个存储器存储有计算机指令；当一个或多个处理器执行计算机指令时，使得电子设备执行如上述第一方面以及各种可能的设计中任一种的图像渲染方法。

[0025] 第三方面，本申请实施例提供一种图像渲染装置，该装置可以包括：处理器、存储器。存储器用于存储计算机可执行程序代码，程序代码包括指令。当处理器执行指令时，指令使电子设备执行如第一方面以及各种可能的设计中任一种的图像渲染方法。

[0026] 第四方面，提供一种计算机可读存储介质，计算机可读存储介质包括计算机指令，当计算机指令运行时，执行如上述第一方面以及各种可能的设计中任一种的图像渲染方法。

[0027] 应当理解的是，上述第二方面，第三方面，第四方面提供的技术方案，其技术特征均可对应到第一方面及其可能的设计中提供的图像渲染方法，因此能够达到的有益效果类似，此处不再赘述。

附图说明

[0028] 图1为一种坐标空间的示意图；

[0029] 图2为一种可变速率着色的示意图；

[0030] 图3为本申请实施例提供的一种电子设备的组成示意图；

[0031] 图4为本申请实施例提供的一种电子设备的软件组成示意图；

[0032] 图5为本申请实施例提供的一种渲染过程的示意图；

[0033] 图6为本申请实施例提供的一种图像渲染方法的流程示意图；

[0034] 图7为本申请实施例提供的一种图像渲染方法的流程示意图；

[0035] 图8为本申请实施例提供的一种图像渲染方法的流程示意图；

[0036] 图9为本申请实施例提供的一种图像渲染方法的流程示意图；

[0037] 图10为本申请实施例提供的一种图像渲染方法的流程示意图；

[0038] 图11为本申请实施例提供的一种图像渲染方法的流程示意图；

[0039] 图12为本申请实施例提供的一种图像渲染方法的流程示意图；

[0040] 图13为本申请实施例提供的一种图像渲染结果的对比示意图；

[0041] 图14为本申请实施例提供的一种电子设备的组成示意图。

具体实施方式

[0042] 电子设备可以根据其中安装的应用程序下发的渲染命令，进行不同帧图像的渲染，从而获取各个帧图像对应的显示数据，进而控制显示器根据这些显示数据实现各个帧图像的显示。

[0043] 在进行图像渲染的过程，电子设备需要确定当前帧图像中包括的一个或多个对象的顶点位置。

[0044] 示例性的，应用程序下发的渲染命令中，可以包括对象的顶点坐标。在一些实现中，该渲染命令中包括的顶点坐标可以是基于对象自身的局部坐标系的坐标。在本申请中，对象基于局部坐标系的分布空间可以称为局部空间(Local Space)。电子设备为了能够确定对象在显示屏上各个顶点的坐标，可以基于局部空间中对象的坐标进行矩阵变换。由此获取对象在基于显示屏的空间(如称为屏幕空间(Screen Space))坐标系下的坐标。

[0045] 作为一种示例，电子设备可以通过局部空间向世界空间(World Space)向观察空间(View Space)向裁剪空间(Clip Space)向屏幕空间(Screen Space)的矩阵变换处理，将对象的各个顶点在局部空间下的局部坐标转换为屏幕空间下的坐标。

[0046] 示例性的，参考图1，示出了一种坐标由局部空间向世界空间向观察空间向裁剪空间的矩阵变换的逻辑过程示意。在本示例中，应用程序下发的渲染命令中，可以包括对物体1的渲染。如图1所示，在局部空间中，坐标系可以是基于物体1的。比如，局部空间中的坐标系原点可以是设置在物体1的中心，或者一个顶点所在的位置等。应用程序可以在下发对物体1的渲染命令中，携带该局部空间的坐标系下，物体1的各个顶点的坐标，即局部坐标。电子设备可以通过应用程序下发的M矩阵，将局部空间中的坐标转换为世界空间中的坐标。其中，世界空间可以是相对于局部空间更大的区域。比如，以应用程序下发的渲染命令用于进行游戏图像的渲染为例。局部空间可以对应能够覆盖某一个对象(如物体1)的较小区域。而世界空间可以对应游戏中，包括物体1以及其他物体(如物体2)的一张地图的区域。电子设备可以将局部空间中的局部坐标，结合M矩阵进行M矩阵变换，从而获取物体1在世界空间中的坐标。类似的，在应用程序下发了在该帧图像中的对物体2的渲染命令的情况下，通过上述M矩阵变换，电子设备还可以获取物体2在世界空间中的坐标。

[0047] 在获取当前帧图像中，各个物体的顶点在世界空间中的坐标之后，电子设备可以根据应用程序下发的V矩阵，将世界空间中的坐标，转换为观察空间中的坐标。可以理解的是，世界空间中的坐标可以是三维空间中的坐标。而电子设备在向用户展示帧图像时，各个

对象(如物体1,物体2等)则是在二维的显示屏上进行显示的。在使用不同的观察角度观察世界空间中的对象时,则会看到不同的二维画面。该观察角度可以是与在世界空间中设置的摄像机(或者观察者)的位置相关的。在本示例中,与摄像机位置对应的坐标空间可以称为观察空间。示例性的,以摄像机设置在世界空间中的y轴正方向为例。那么基于V矩阵的变换,就可以获取在该摄像机位置对应的观察空间中,物体1和物体2的各个顶点的坐标。如图1所示,由于摄像机位于y轴正方向,向下进行拍摄,因此对应在观察空间中的物体1和物体2可以呈现为俯视效果。

[0048] 在电子设备获取观察空间中的各个对象的坐标后,可以将其投影到裁剪坐标。该裁剪坐标对应的坐标空间可以称为裁剪空间。可以理解的是,在进行V矩阵变换过程中,可以是对世界空间中的较大区域的变换,因此获取的图像范围可能是比较大的。而由于电子设备显示屏的尺寸有限,因此可能无法将该观察空间中的所有对象同时进行显示。在本示例中,电子设备就可以将观察空间中的各个对象的坐标,投影到裁剪空间中。在投影到裁剪空间中后,能够显示在显示屏上的对象的坐标可以在-1.0到1.0的范围内。而对于无法显示在显示屏上的部分对象的坐标,则可以在-1.0到1.0的范围之外。这样,电子设备就可以根据坐标在-1.0到1.0范围内的顶点坐标进行对应的显示。示例性的,电子设备可以根据应用程序下发的P矩阵,对观察空间中的各个坐标进行P矩阵变换,从而获取各个坐标对应的在裁剪空间中的裁剪坐标。

[0049] 可以理解的是,通过上述MVP矩阵的变换(即M矩阵变换,V矩阵变换,以及P矩阵变换),电子设备就能够获取在显示屏上显示的各个对象的顶点的坐标(即裁剪坐标)。接着,电子设备还可以将裁剪坐标变换为屏幕坐标,比如,使用视口变换(Viewport Transform)将位于-1.0到1.0范围的坐标变换到由glViewport函数所定义的坐标范围内。最后变换出来的坐标将会送到光栅器,将其转化为片段,进而获取与各个像素对应的显示数据。基于这些显示数据,电子设备就可以控制显示屏进行对应的显示。

[0050] 电子设备在进行图像的渲染过程中,除了需要根据上述方案确定各个对象的顶点坐标之外,还需要对当前帧图像中的各个像素进行着色,即确定各个像素的颜色数据。从而根据各个像素的颜色数据,控制显示器在对应的像素位置进行对应颜色的显示。

[0051] 在一些实现中,电子设备可以在渲染过程中,以一个像素为单位,对每个像素进行着色,从而实现对整个帧图像的着色。随着电子设备显示屏的分辨率以及刷新率的提升,加上需要渲染的帧图像的场景越来越复杂,基于一个像素的着色会对电子设备的渲染过程造成较大的内存和功耗的开销,进而出现发热或者掉帧现象,影响用户体验。需要说明的是,在本申请的一些实现中,渲染过程以及着色的过程,可以是由电子设备中设置的GPU等具有图形处理功能的部件实现的。

[0052] 为了应对上述问题,一些电子设备可以通过提供可变速率着色的功能,降低着色过程中对内存和功耗的开销。

[0053] 示例性的,在一般的着色机制下,电子设备可以分别使用着色器,对一个像素进行着色。完成该像素的着色操作之后,电子设备可以使用着色器,对另一个像素进行着色。比如,结合图2中的(a),电子设备可以使用着色器,对位于第一行第一列的像素进行着色。完成该第一行第一列的像素的着色之后,电子设备可以使用着色器,对其他像素,如第一行第二列的像素进行着色。这样,要完成如图2中的(a)所示的5*5的像素的着色,电子设备就需

要使用着色器进行至少25次着色操作。需要说明的是，结合前述说明，上述通过着色器进行着色的过程可以是电子设备中的GPU执行的。在GPU具有较强的并行处理能力时，比如GPU可以同时通过着色器对3个像素分别进行着色，那么电子设备也可以通过着色器并行地实现多个像素(如3个像素)的着色操作。然而多发并行处理的过程虽然能够节省处理时间，但是并不会减少电子设备在进行着色操作的过程中的负荷。为了便于说明，以下以电子设备中的GPU同一时刻使用着色器对1个像素进行着色为例进行说明。

[0054] 与以单个像素为单位进行着色操作相对的，在电子设备使用可变速率着色功能的情况下，电子设备可以使用着色器，通过一次着色操作完成对多个像素的着色。比如，以一次着色操作可以完成对4个像素的着色为例。结合图2中的(b)，电子设备可以通过一次着色操作，实现对第一列第一个像素到第二列第二个像素的着色。这样，通过可变速率着色，电子设备就可以通过更少的着色操作，完成在对图像渲染过程中的着色。

[0055] 可以理解的是，可变速率着色后的像素组成的图像的颜色精细度会低于一般的着色机制(即以一个像素为粒度的着色操作)获取的像素组成的图像。那么，如何合理地使用可变速率着色，同时不会使得用户对图像的观感产生明显的影响，就成为了可变速率着色功能使用的关键。

[0056] 为了解决上述问题，本申请实施例提供的渲染方案，能够合理地选取帧图像中需要使用可变速率着色功能的区域，使得电子设备可以通过可变速率着色功能降低渲染过程中的功耗和发热的同时，渲染获取的图像不会对用户的观感产生明显的影响。由此即可在降低电子设备的功耗和发热的同时，提升用户体验。

[0057] 以下结合附图对本申请实施例提供的方案进行详细说明。

[0058] 需要说明的是，本申请实施例提供的渲染方法，可以应用在用户的电子设备中。比如，该电子设备可以是手机、平板电脑、个人数字助理(personal digital assistant，PDA)、增强现实(augmented reality，AR)\虚拟现实(virtual reality，VR)设备、媒体播放器等便携式移动设备，该电子设备也可以是智能手表等能够提供拍摄能力的可穿戴电子设备。本申请实施例对该设备的具体形态不作特殊限制。

[0059] 请参考图3，为本申请实施例提供的一种电子设备300的结构示意图。

[0060] 如图3所示，该电子设备300可以包括处理器310，外部存储器接口320，内部存储器321，通用串行总线(universal serial bus，USB)接口330，充电管理模块340，电源管理模块341，电池342，天线1，天线2，移动通信模块350，无线通信模块360，音频模块370，扬声器370A，受话器370B，麦克风370C，耳机接口370D，传感器模块380，按键390，马达391，指示器392，摄像头393，显示屏394，以及用户标识模块(subscriber identification module，SIM)卡接口395等。其中，传感器模块380可以包括压力传感器，陀螺仪传感器，气压传感器，磁传感器，加速度传感器，距离传感器，接近光传感器，指纹传感器，温度传感器，触摸传感器，环境光传感器，骨传导传感器等。

[0061] 可以理解的是，本实施例示意的结构并不构成对电子设备300的具体限定。在另一些实施例中，电子设备300可以包括比图示更多或更少的部件，或者组合某些部件，或者拆分某些部件，或者不同的部件布置。图示的部件可以以硬件，软件或软件和硬件的组合实现。

[0062] 处理器310可以包括一个或多个处理单元，例如：处理器310可以包括中央处理器

(Central Processing Unit,CPU),应用处理器(application processor,AP),调制解调处理器,GPU,图像信号处理器(image signal processor,ISP),控制器,存储器,视频编解码器,数字信号处理器(digital signal processor,DSP),基带处理器,和/或神经网络处理器(neural-network processing unit,NPU)等。其中,不同的处理单元可以是独立的器件,也可以集成在一个或多个处理器310中。作为一种示例,在本申请中,ISP可以对图像进行处理,如该处理可以包括自动曝光(Automatic Exposure)、自动对焦(Automatic Focus)、自动白平衡(Automatic White Balance)、去噪、背光补偿、色彩增强等处理。其中,自动曝光,自动对焦,以及自动白平衡的处理也可以称为3A处理。经过处理后,ISP就可以进行获取对应的照片。该过程也可称为ISP的成片操作。

[0063] 在一些实施例中,处理器310可以包括一个或多个接口。接口可以包括集成电路(inter-integrated circuit,I2C)接口,集成电路内置音频(inter-integrated circuit sound,I2S)接口,脉冲编码调制(pulse code modulation,PCM)接口,通用异步收发传输器(universal asynchronous receiver/transmitter,UART)接口,移动产业处理器接口(mobile industry processor interface,MIPI),通用输入输出(general-purpose input/output,GPIO)接口,用户标识模块(subscriber identity module,SIM)接口,和/或通用串行总线(universal serial bus,USB)接口等。

[0064] 电子设备300可以通过ISP,摄像头393,视频编解码器,GPU,显示屏394以及应用处理器等实现拍摄功能。

[0065] ISP用于处理摄像头393反馈的数据。例如,拍照时,打开快门,光线通过镜头被传递到摄像头393感光元件上,光信号转换为电信号,摄像头393感光元件将所述电信号传递给ISP处理,转化为肉眼可见的图像。ISP还可以对图像的噪点,亮度,肤色进行算法优化。ISP还可以对拍摄场景的曝光,色温等参数优化。在一些实施例中,ISP可以设置在摄像头393中。

[0066] 摄像头393用于捕获静态图像或视频。物体通过镜头生成光学图像投射到感光元件。感光元件可以是电荷耦合器件(charge coupled device,CCD)或互补金属氧化物半导体(complementary metal-oxide-semiconductor,CMOS)光电晶体管。感光元件把光信号转换成电信号,之后将电信号传递给ISP转换成数字图像信号。ISP将数字图像信号输出到DSP加工处理。DSP将数字图像信号转换成标准的RGB,YUV等格式的图像信号。在一些实施例中,电子设备300可以包括1个或N个摄像头393,N为大于1的正整数。

[0067] 数字信号处理器用于处理数字信号,除了可以处理数字图像信号,还可以处理其他数字信号。例如,当电子设备300在频点选择时,数字信号处理器用于对频点能量进行傅里叶变换等。

[0068] 视频编解码器用于对数字视频压缩或解压缩。电子设备300可以支持一种或多种视频编解码器。这样,电子设备300可以播放或录制多种编码格式的视频,例如:动态图像专家组(moving picture experts group,MPEG)1,MPEG2,MPEG3,MPEG4等。

[0069] NPU为神经网络(neural-network,NN)计算处理器,通过借鉴生物神经网络结构,例如借鉴人脑神经元之间传递模式,对输入信息快速处理,还可以不断的自学习。通过NPU可以实现电子设备300的智能认知等应用,例如:图像识别,人脸识别,语音识别,文本理解等。

[0070] 充电管理模块340用于从充电器接收充电输入。其中，充电器可以是无线充电器，也可以是有线充电器。在一些有线充电的实施例中，充电管理模块340可以通过USB接口330接收有线充电器的充电输入。在一些无线充电的实施例中，充电管理模块340可以通过电子设备300的无线充电线圈接收无线充电输入。充电管理模块340为电池342充电的同时，还可以通过电源管理模块341为电子设备300供电。

[0071] 电源管理模块341用于连接电池342，充电管理模块340与处理器310。电源管理模块341接收电池342和/或充电管理模块340的输入，为处理器310，内部存储器321，外部存储器，显示屏394，摄像头393，和无线通信模块360等供电。电源管理模块341还可以用于监测电池342容量，电池342循环次数，电池342健康状态(漏电，阻抗)等参数。在其他一些实施例中，电源管理模块341也可以设置于处理器310中。在另一些实施例中，电源管理模块341和充电管理模块340也可以设置于同一个器件中。

[0072] 电子设备300的无线通信功能可以通过天线1，天线2，移动通信模块350，无线通信模块360，调制解调处理器310以及基带处理器310等实现。

[0073] 天线1和天线2用于发射和接收电磁波信号。电子设备300中的每个天线可用于覆盖单个或多个通信频带。不同的天线还可以复用，以提高天线的利用率。例如：可以将天线1复用为无线局域网的分集天线。在另外一些实施例中，天线可以和调谐开关结合使用。

[0074] 移动通信模块350可以提供应用在电子设备300上的包括2G/3G/4G/5G等无线通信的解决方案。移动通信模块350可以包括至少一个滤波器，开关，功率放大器，低噪声放大器(low noise amplifier，LNA)等。移动通信模块350可以由天线1接收电磁波，并对接收的电磁波进行滤波，放大等处理，传送至调制解调处理器进行解调。移动通信模块350还可以对经调制解调处理器调制后的信号放大，经天线1转为电磁波辐射出去。在一些实施例中，移动通信模块350的至少部分功能模块可以被设置于处理器310中。在一些实施例中，移动通信模块350的至少部分功能模块可以与处理器310的至少部分模块被设置在同一个器件中。

[0075] 调制解调处理器可以包括调制器和解调器。其中，调制器用于将待发送的低频基带信号调制成中高频信号。解调器用于将接收的电磁波信号解调为低频基带信号。随后解调器将解调得到的低频基带信号传送至基带处理器处理。低频基带信号经基带处理器处理后，被传递给应用处理器。应用处理器通过音频设备(不限于扬声器370A，受话器370B等)输出声音信号，或通过显示屏394显示图像或视频。在一些实施例中，调制解调处理器可以是独立的器件。在另一些实施例中，调制解调处理器可以独立于处理器310，与移动通信模块350或其他功能模块设置在同一个器件中。

[0076] 无线通信模块360可以提供应用在电子设备300上的包括无线局域网(wireless local area networks，WLAN)(如无线保真(wireless fidelity，Wi-Fi)网络)，蓝牙(bluetooth，BT)，全球导航卫星系统(global navigation satellite system，GNSS)，调频(frequency modulation，FM)，近距离无线通信技术(near field communication，NFC)，红外技术(infrared，IR)等无线通信的解决方案。无线通信模块360可以是集成至少一个通信处理模块的一个或多个器件。无线通信模块360经由天线2接收电磁波，将电磁波信号调频以及滤波处理，将处理后的信号发送到处理器310。无线通信模块360还可以从处理器310接收待发送的信号，对其进行调频，放大，经天线2转为电磁波辐射出去。

[0077] 在一些实施例中，电子设备300的天线1和移动通信模块350耦合，天线2和无线通

信模块360耦合，使得电子设备300可以通过无线通信技术与网络以及其他设备通信。所述无线通信技术可以包括全球移动通讯系统(global system for mobile communications，GSM)，通用分组无线服务(general packet radio service，GPRS)，码分多址接入(code division multiple access，CDMA)，宽带码分多址(wideband code division multiple access，WCDMA)，时分码分多址(time-division code division multiple access，TD-SCDMA)，长期演进(long term evolution，LTE)，BT，GNSS，WLAN，NFC，FM，和/或IR技术等。所述GNSS可以包括全球卫星定位系统(global positioning system，GPS)，全球导航卫星系统(global navigation satellite system，GLONASS)，北斗卫星导航系统(beidou navigation satellite system，BDS)，准天顶卫星系统(quasi-zenith satellite system，QZSS)和/或星基增强系统(satellite based augmentation systems，SBAS)。

[0078] 电子设备300通过GPU，显示屏394，以及应用处理器310等实现显示功能。GPU为图像处理的微处理器，连接显示屏394和应用处理器。GPU用于执行数学和几何计算，用于图形渲染。处理器310可包括一个或多个GPU，其执行程序指令以生成或改变显示信息。

[0079] 显示屏394用于显示图像，视频等。显示屏394包括显示面板。显示面板可以采用液晶显示屏394(liquid crystal display，LCD)，有机发光二极管(organic light-emitting diode，OLED)，有源矩阵有机发光二极体或主动矩阵有机发光二极体(active-matrix organic light emitting diode，AMOLED)，柔性发光二极管(flex light-emitting diode，FLED)，Miniled，MicroLed，Micro-oLed，量子点发光二极管(quantum dot light emitting diodes，QLED)等。在一些实施例中，电子设备300可以包括1个或N个显示屏394，N为大于1的正整数。

[0080] 外部存储器接口320可以用于连接外部存储卡，例如Micro SD卡，实现扩展电子设备300的存储能力。外部存储卡通过外部存储器接口320与处理器310通信，实现数据存储功能。例如将音乐，视频等文件保存在外部存储卡中。

[0081] 内部存储器321可以用于存储计算机可执行程序代码，所述可执行程序代码包括指令。处理器310通过运行存储在内部存储器321的指令，从而执行电子设备300的各种功能应用以及数据处理。内部存储器321可以包括存储程序区和存储数据区。其中，存储程序区可存储操作系统，至少一个功能所需的应用程序(比如声音播放功能，图像播放功能等)等。存储数据区可存储电子设备300使用过程中所创建的数据(比如音频数据，电话本等)等。此外，内部存储器321可以包括高速随机存取存储器，还可以包括非易失性存储器，例如至少一个磁盘存储器件，闪存器件，通用闪存存储器(universal flash storage，UFS)等。

[0082] 电子设备300可以通过音频模块370，扬声器370A，受话器370B，麦克风370C，耳机接口370D，以及应用处理器310等实现音频功能。例如音乐播放，录音等。

[0083] 音频模块370用于将数字音频信息转换成模拟音频信号输出，也用于将模拟音频输入转换为数字音频信号。音频模块370还可以用于对音频信号编码和解码。在一些实施例中，音频模块370可以设置于处理器310中，或将音频模块370的部分功能模块设置于处理器310中。扬声器370A，也称“喇叭”，用于将音频电信号转换为声音信号。电子设备300可以通过扬声器370A收听音乐，或收听免提通话。受话器370B，也称“听筒”，用于将音频电信号转换成声音信号。当电子设备300接听电话或语音信息时，可以通过将受话器370B靠近人耳接听语音。麦克风370C，也称“话筒”，“传声器”，用于将声音信号转换为电信号。当拨打电话或

发送语音信息或需要通过语音助手触发电子设备300执行某些功能时，用户可以通过人嘴靠近麦克风370C发声，将声音信号输入到麦克风370C。电子设备300可以设置至少一个麦克风370C。在另一些实施例中，电子设备300可以设置两个麦克风370C，除了采集声音信号，还可以实现降噪功能。在另一些实施例中，电子设备300还可以设置三个，四个或更多麦克风370C，实现采集声音信号，降噪，还可以识别声音来源，实现定向录音功能等。耳机接口370D用于连接有线耳机。耳机接口370D可以是USB接口330，也可以是3.5mm的开放移动电子设备300平台(open mobile terminal platform，OMTP)标准接口，美国蜂窝电信工业协会(cellular telecommunications industry association of the USA，CTIA)标准接口。

[0084] 触摸传感器，也称“触控面板”。触摸传感器可以设置于显示屏394，由触摸传感器与显示屏394组成触摸屏，也称“触控屏”。触摸传感器用于检测作用于其上或附近的触摸操作。触摸传感器可以将检测到的触摸操作传递给应用处理器，以确定触摸事件类型。在一些实施例中，可以通过显示屏394提供与触摸操作相关的视觉输出。在另一些实施例中，触摸传感器也可以设置于电子设备300的表面，与显示屏394所处的位置不同。

[0085] 压力传感器用于感受压力信号，可以将压力信号转换成电信号。在一些实施例中，压力传感器可以设置于显示屏394。压力传感器的种类很多，如电阻式压力传感器，电感式压力传感器，电容式压力传感器等。电容式压力传感器可以是包括至少两个具有导电材料的平行板。当有力作用于压力传感器，电极之间的电容改变。电子设备300根据电容的变化确定压力的强度。当有触摸操作作用于显示屏394，电子设备300根据压力传感器检测所述触摸操作强度。电子设备300也可以根据压力传感器的检测信号计算触摸的位置。在一些实施例中，作用于相同触摸位置，但不同触摸操作强度的触摸操作，可以对应不同的操作指令。例如：当有触摸操作强度小于第一压力阈值的触摸操作作用于短消息应用图标时，执行查看短消息的指令。当有触摸操作强度大于或等于第一压力阈值的触摸操作作用于短消息应用图标时，执行新建短消息的指令。陀螺仪传感器可以用于确定电子设备300的运动姿态。加速度传感器可检测电子设备300在各个方向上(一般为三轴)加速度的大小。距离传感器，用于测量距离。电子设备300可以通过红外或激光测量距离。电子设备300可以利用接近光传感器检测用户手持电子设备300贴近耳朵通话，以便自动熄灭屏幕达到省电的目的。环境光传感器用于感知环境光亮度。指纹传感器用于采集指纹。温度传感器用于检测温度。在一些实施例中，电子设备300利用温度传感器检测的温度，执行温度处理策略。音频模块370可以基于所述骨传导传感器获取的声部振动骨块的振动信号，解析出语音信号，实现语音功能。应用处理器可以基于所述骨传导传感器获取的血压跳动信号解析心率信息，实现心率检测功能。

[0086] 按键390包括开机键，音量键等。马达391可以产生振动提示。指示器392可以是指示灯，可以用于指示充电状态，电量变化，也可以用于指示消息，未接来电，通知等。SIM卡接口395用于连接SIM卡。电子设备300可以支持1个或N个SIM卡接口395，N为大于1的正整数。SIM卡接口395可以支持Nano SIM卡，Micro SIM卡，SIM卡等。同一个SIM卡接口395可以同时插入多张卡。SIM卡接口395也可以兼容不同类型的SIM卡。SIM卡接口395也可以兼容外部存储卡。电子设备300通过SIM卡和网络交互，实现通话以及数据通信等功能。在一些实施例中，电子设备300采用eSIM，即：嵌入式SIM卡。eSIM卡可以嵌在电子设备300中，不能和电子设备300分离。

[0087] 本申请实施例提供的渲染方法均能够应用于具有如图3所示的组成的电子设备中。

[0088] 需要说明的是，上述图3及其说明仅为本申请实施例提供的方案的一种应用载体的示例。图3的组成并不构成对本申请实施例所述的方案的限定。在另一些实施例中，电子设备还可以具有比图3所示组成更多或更少的部件。

[0089] 在如图3所示的示例中，提供了电子设备的硬件组成。在一些实施例中，电子设备还可以通过其各个硬件部件(如图3所示的硬件组成)，运行操作系统。在该操作系统中，可以设置有不同的软件分层，从而实现不同程序的运行。

[0090] 示例性的，图4为本申请实施例提供的一种电子设备的软件组成的示意图。如图4所示，该电子设备可以包括应用层401，框架层402，系统库403，以及硬件层404等。

[0091] 其中，应用层401也可以称为应用程序层，或者应用(application，APP)层。在一些实现中，应用程序层可以包括一系列应用程序包。应用程序包可以包括相机，图库，日历，通话，地图，导航，WLAN，蓝牙，音乐，视频，短信息等应用程序。应用程序包还可以包括需要通过渲染图像向用户展示图片或者视频的应用程序。比如，该应用层401中包括的应用程序可以为游戏类应用程序，例如和平精英®，王者荣耀®等。

[0092] 框架层402也可以称为应用程序框架层。该框架层402可以为应用层401的应用程序提供应用编程接口(application programming interface，API)和编程框架。框架层402包括一些预先定义的函数。

[0093] 示例性的，框架层402可以包括窗口管理器，内容提供器，视图系统，资源管理器，通知管理器，活动管理器，输入管理器等。窗口管理器提供窗口管理服务(Window Manager Service，WMS)，WMS可以用于窗口管理、窗口动画管理、surface管理以及作为输入系统的中转站。内容提供器用来存放和获取数据，并使这些数据可以被应用程序访问。该数据可以包括视频，图像，音频，拨打和接听的电话，浏览历史和书签，电话簿等。视图系统包括可视控件，例如显示文字的控件，显示图片的控件等。视图系统可用于构建应用程序。显示界面可以由一个或多个视图组成的。例如，包括短信通知图标的显示界面，可以包括显示文字的视图以及显示图片的视图。资源管理器为应用程序提供各种资源，比如本地化字符串，图标，图片，布局文件，视频文件等等。通知管理器使应用程序可以在状态栏中显示通知信息，可以用于传达告知类型的消息，可以短暂停留后自动消失，无需用户交互。比如通知管理器被用于告知下载完成，消息提醒等。通知管理器还可以是以图表或者滚动条文本形式出现在系统顶部状态栏的通知，例如后台运行的应用程序的通知，还可以是以对话窗口形式出现在屏幕上的通知。例如在状态栏提示文本信息，发出提示音，电子设备振动，指示灯闪烁等。活动管理器可以提供活动管理服务(Activity Manager Service，AMS)，AMS可以用于系统组件(例如活动、服务、内容提供者、广播接收器)的启动、切换、调度以及应用进程的管理和调度工作。输入管理器可以提供输入管理服务(Input Manager Service，IMS)，IMS可以用于管理系统的输入，例如触摸屏输入、按键输入、传感器输入等。IMS从输入设备节点取出事件，通过和WMS的交互，将事件分配至合适的窗口。

[0094] 在本申请实施例中，在框架层402中可以设置一个或多个功能模块，用于实现本申请实施例提供的渲染方案。示例性的，框架层402中可以设置有拦截模块，数据处理模块，计算模块，以及决策模块等。在后续示例中，将会对上述各个模块的功能进行详细说明。

[0095] 系统库403可以包括多个功能模块。例如:表面管理器(surface manager),媒体框架(Media Framework),标准C库(Standard C library,libc),嵌入式系统的开放图形库(OpenGL for Embedded Systems,OpenGL ES)、Vulkan、SQLite、Webkit等。

[0096] 其中,表面管理器用于对显示子系统进行管理,并且为多个应用程序提供了2D和3D图层的融合。媒体框架支持多种常用的音频,视频格式回放和录制,以及静态图像文件等。媒体库可以支持多种音视频编码格式,例如:动态图像专家组4(Moving Pictures Experts Group,MPEG4),H.264,动态影像专家压缩标准音频层面3(Moving Picture Experts Group Audio Layer3,MP3),高级音频编码(Advanced Audio Coding,AAC),自适应多码解码(Adaptive Multi-Rate,AMR),联合图像专家组(Joint Photographic Experts Group,JPEG,或称为JPG),便携式网络图形(Portable Network Graphics,PNG)等。OpenGL ES和/或Vulkan提供应用程序中2D图形和3D图形的绘制和操作。SQLite为电子设备400的应用程序提供轻量级关系型数据库。在一些实现中,系统库403中的OpenGL ES能够提供可变速率着色功能。那么,电子设备可以在需要针对某一个绘制命令(Drawcall)执行可变速率着色时,调用OpenGL ES中的可变速率着色API,与其他指令一同实现对当前Drawcall的可变速率着色。比如,电子设备可以使用较低的速率(如2×1,2×2,4×2,4×4等)对当前Drawcall进行着色,由此降低对当前Drawcall进行着色产生的开销。

[0097] 在如图4的示例中,电子设备中还可以包括硬件层404。该硬件层404中可以包括处理器(如CPU,GPU等),以及具有存储功能的部件(如如图3所示的内部存储器321等)。在一些实现中,CPU可以用于控制框架层402中的各个模块实现其各自的功能,GPU可以用于根据框架层402中各个模块处理后的指令所调用的图形库(如OpenGL ES)中的API执行相应的渲染处理。

[0098] 为了能够对本申请实施例提供的软件架构中各个层的功能进行更加清楚的说明,以下以图像渲染为例,对具有如图4所示软件组成的各个组件的功能实现进行举例说明。

[0099] 示例性的,请参考图5。应用层中的应用程序在需要进行图像渲染时,可以下发渲染命令。在以下说明中,应用程序下发的一个渲染命令也可以称为一个Drawcall。在不同示例中,该渲染命令可以包括不同的内容。比如,在一些实施例中,以应用程序需要渲染帧图像中的图形为例。在下发的渲染命令中可以包括该需要被渲染的图形的顶点数据。在一些实现中,该顶点数据可以用于指示待渲染的图形的顶点的坐标。该坐标可以是基于局部空间的坐标。在渲染命令中,还可以包括如图1所示的说明中的MVP矩阵,以及一个或多个绘制元素(Drawelement)。框架层402可以在接收到该渲染命令之后,将渲染命令转换为渲染指令,在渲染指令中可以携带上述顶点数据,MVP矩阵,以及一个或多个Drawelement等。在一些实现中,框架层402还可以根据应用程序的指示,从系统库403的图形库中获取当前Drawcall所需的API,以便使用该API对应的功能指示其他模块(如GPU)进行渲染操作。示例性的,电子设备可以在Drawelement之前,确定可变速率着色过程中所要使用的参数。电子设备还可以通过调用可变速率着色API,结合前述参数,发送可变着色指令。实现对后续Drawelement的可变速率着色。以硬件层404中的GPU执行渲染为例。GPU可以获取可变着色指令,并响应于该可变着色指令,使用对应的参数指示的着色速率执行Drawelement。

[0100] 本申请实施例提供的渲染方法也可以应用于具有如图4所示的软件组成的电子设备中。以下结合如图4所示的软件组成,对本申请实施例提供的方案进行说明。

[0101] 需要说明的是,在以下示例中,为了能够更加清楚地对本申请提供的方案进行说明,根据不同的功能,对电子设备进行模块划分,该模块划分可以理解为具有如图3或图4所示组成的电子设备的另一种划分形式。某个功能究竟以硬件还是计算机软件驱动硬件的方式来执行,取决于技术方案的特定应用和设计约束条件。专业技术人员可以对每个特定的应用来使用不同方法来实现所描述的功能,但是这种实现不应认为超出本申请的范围。

[0102] 本申请实施例可以根据本申请实施例提供的方案对其中涉及的电子设备进行功能模块的划分,例如,可以对应各个功能划分各个功能模块,也可以将两个或两个以上的功能集成在一个处理模块中。上述集成的模块既可以采用硬件的形式实现,也可以采用软件功能模块的形式实现。需要说明的是,本申请实施例中对模块的划分是示意性的,仅仅为一种逻辑功能划分,实际实现时可以有另外的划分方式。

[0103] 示例性的,电子设备的框架层402中可以设置有拦截模块,数据处理模块,计算模块,以及决策模块。

[0104] 基于本申请实施例提供的渲染方法,电子设备可以通过其中设置的各个模块,拦截来自应用程序(如游戏应用)的渲染命令。电子设备还可以从这些渲染命令中筛选获取当前渲染命令中包括的各个顶点的顶点数据。在该顶点数据中可以包括顶点坐标。电子设备还可以获取以及当前渲染命令所对应的MVP矩阵。电子设备可以根据上述顶点坐标以及MVP矩阵,确定各个顶点的深度。进而根据该各个顶点的深度确定当前Drawcall所要绘制的模型的深度。其中模型的深度越大,则表明模型在当前帧图像中,与观察者的距离越远。对应的,模型的深度越小,则表明模型与观察者的距离越近。可以理解的是,距离观察者(即用户)较远的模型,一般不会被用户所关注,而距离用户较近的模型,可能是用户所关注的模型。因此,本申请实施例提供的方案中,可以对具有较大深度的模型进行较低速率的着色,从而在不影响用户体验的情况下,降低着色对应的渲染开销。相对的,对具有较小深度的模型,电子设备可以对其进行较高速率的着色,从而实现对该模型的精确着色,提升模型在当前帧图像中的显示质量,进而达到提升用户体验的效果。

[0105] 以下结合上述模块划分,对本申请实施例提供的渲染方法的具体实现进行详细说明。为了便于说明,以下示例中,以下发渲染命令的应用程序为游戏应用,该游戏应用使用OpenGL图形库为例进行说明。应当理解的是,在其他的不同渲染引擎/渲染环境下,实施机制类似,仅对应的函数可能存在差异。

[0106] 在游戏开始运行时(或游戏加载时),游戏应用可以加载后续帧图像的渲染过程中可能要使用的数据。在一些实现中,在游戏加载时加载的数据可以包括后续渲染过程中可能使用到的所有模型的顶点数据以及一个或多个帧图像的MVP矩阵。在另一些实现中,在一次游戏加载时可能只会加载一部分模型的顶点数据。这样,在需要使用到新的模型时,电子设备就可以再次执行加载过程,将该新的模型的顶点数据加载到GPU中。或者,电子设备可以通过在下发的渲染指令中携带该新的模型的顶点数据,实现顶点数据的加载。在另一些实现中,游戏应用还可以在游戏加载过程中只进行顶点数据的传输,而对于每帧图像都可能不同的MVP矩阵,游戏应用可以在游戏运行过程中,进行该MVP矩阵的传输。在本申请实施例中,顶点数据可以包括顶点坐标,该顶点坐标可以是基于局部空间的坐标。

[0107] 为了便于说明,以下示例中,以游戏通过一次加载,实现所有模型的顶点坐标,以及MVP矩阵的加载为例。

[0108] 在本示例中，通过游戏加载，游戏应用可以通过包括多个指令的命令，传输所有可能用到模型的顶点坐标以及一个或多个MVP矩阵。通过这些指令，模型的顶点坐标以及MVP矩阵可以被存储到GPU所能够调用的内存空间中。

[0109] 示例性的，在游戏启动时，游戏应用可以下发命令1，用于实现上述加载过程。在一些实施例中，以在该命令1中包括glGenbuffers函数，glBindbuffer函数，glBufferdata函数，glBufferSubData函数中的一个或多个为例。

[0110] 其中，glGenbuffers函数可以用于创建缓存。即在电子设备的内存中划分一个或多个存储空间，每个存储空间都可以具有一个标识(ID)。该划分的存储空间可以用于存储渲染过程中的各项数据。比如，一些缓存可以用于存储模型的顶点坐标，一些缓存可以用于存储MVP矩阵等。

[0111] glBindbuffer函数可以用于绑定缓存。通过该函数的绑定，就可以将后续的操作绑定在对应的缓存上。比如，以创建的缓存包括缓存1，缓存2，缓存3为例。通过glBindbuffer(1)，就可以将后续的操作绑定在缓存1上。例如，后续操作包括写入数据(如顶点坐标)的操作，那么电子设备就可以将顶点坐标写入到缓存1进行存储。

[0112] glBufferdata函数可以用于传递数据。示例性的，在glBufferdata函数携带的数据非空(NULL)，则电子设备就可以将该glBufferdata函数携带的数据(或者数据的指针)存储到已经绑定的缓存上。比如，在glBufferdata函数携带有顶点坐标时，则电子设备就可以将该顶点坐标存储到已经绑定的帧缓冲上。又如，在glBufferdata函数携带有顶点坐标的索引时，则电子设备就可以将该顶点坐标索引存储到已经绑定的帧缓冲上。

[0113] glBufferSubData函数可以用于数据的更新。比如，游戏应用可以通过该glBufferSubData函数，实现对顶点坐标中的一部分或全部进行更新。从而达到指示电子设备(如GPU)根据新的顶点坐标进行绘制渲染的效果。

[0114] 在本申请实施例中，电子设备可以对命令1的指令流进行拦截，由此获取传输顶点数据以及MVP矩阵的指令。电子设备还可以对这些获取的指令进行备份存储。如，电子设备可以将这些数据存储在内存中CPU能够调用的区域。由此使得在游戏运行之前，电子设备的内存中就可以存储有后续渲染过程中可能用到的顶点数据(如顶点坐标)以及MVP矩阵。可以理解的是，原生命令(如命令1中的指令流)是用于将数据传输到GPU所能够调用的存储区域的，因此，通过本示例中的备份存储，使得CPU也可以具有对顶点数据和MVP矩阵的调用能力，由此保证后续方案的实现。

[0115] 示例性的，如图6所示，电子设备中的拦截模块可以拦截命令1中包括的glGenbuffers函数，glBindbuffer函数，glBufferdata函数，glBufferSubData函数。拦截模块还可以将这些函数传输给数据处理模块进行分析处理。如数据处理模块可以筛选来自拦截模块的函数中，携带参数1的函数。其中，参数1可以为指示用于进行顶点相关数据传输的参数。这样，数据处理模块就可以获取传输顶点数据相关的指令。那么基于这些筛选获取的函数，数据处理模块就可以执行对顶点数据的备份存储。

[0116] 其中，该参数1可以是通过离线分析获取的。在一些实施例中，该参数1可以预先保存在电子设备(如数据处理模块)中，以便于数据处理模块可以基于该参数1执行对顶点数据相关指令的筛选。作为一种可能的实现，该参数1可以包括GL_ELEMENT_ARRAY_BUFFER和/或GL_ARRAY_BUFFER。

[0117] 与针对顶点数据指令的拦截以及备份存储类似的,拦截模块拦截的拦截命令1中包括的glGenbuffers函数,glBindbuffer函数,glBufferdata函数,glBufferSubData函数也可以用于进行MVP矩阵的传输。

[0118] 那么,数据处理模块也可以可以筛选来自拦截模块的函数中,携带参数2的函数,来获取用于传输MVP矩阵的函数。其中,参数2可以为指示用于进行MVP矩阵传输的参数。这样,数据处理模块就可以获取传输MVP矩阵相关的指令。那么基于这些筛选获取的函数,数据处理模块就可以执行对MVP矩阵的备份存储。

[0119] 其中,该参数2可以是通过离线分析获取的。在一些实施例中,该参数2可以预先保存在电子设备(如数据处理模块)中,以便于数据处理模块可以基于该参数2执行对MVP矩阵相关指令的筛选。作为一种可能的实现,该参数2可以包括GL_UNIFORM_BUFFER。

[0120] 上述示例中,是以拦截模块将所有拦截的指令不经过处理直接传输给数据处理模块进行分析处理为例进行说明的。在本申请的另一些实施例中,该拦截模块还可以具有基础分析能力。比如,拦截模块可以只拦截携带有参数1以及参数2的glGenbuffers函数,glBindbuffer函数,glBufferdata函数,glBufferSubData函数。其中,参数1可以为指示用于进行顶点相关数据传输的参数。参数2可以为指示用于进行MVP矩阵传输的参数。

[0121] 这样,数据处理模块就可以直接对来自拦截模块的指令进行备份存储。由此可以减轻数据处理模块的数据处理压力。

[0122] 以下示例中,如图6所示的示例,以拦截模块拦截顶点相关指令(如携带参数1的指令)以及MVP相关指令(如携带参数2的指令)传输给数据处理模块进行备份存储为例进行说明。

[0123] 本申请实施例中涉及的备份存储,可以通过跳转表的形式实现。该跳转表可以用于指示原生ID与备份ID的对应关系。其中,原生ID可以为命令1中携带函数所指示需要操作的缓存ID。备份ID可以为在内存中配置的,可以被CPU调用的用于进行数据的备份存储的缓存ID。

[0124] 示例性的,以拦截模块拦截的顶点数据相关指令的函数包括如下函数为例:

[0125] glGenbuffers(GL_ARRAY_BUFFER,1)//为顶点数据创建缓存,缓存ID为1;

[0126] glBindbuffer(GL_ARRAY_BUFFER,1)//为顶点数据绑定ID为1的缓存;

[0127] glBufferdata(GL_ARRAY_BUFFER,data1)//向已经绑定的缓存写入data1;

[0128] glBufferSubData(GL_ARRAY_BUFFER,data2)//更新已经绑定的缓存中的数据为data2。

[0129] 那么,该示例中的原生ID就可以为1。以对应的备份ID为11为例。

[0130] 根据拦截的glGenbuffers(GL_ARRAY_BUFFER,1),数据处理模块就可以在备份缓存中创建ID为与1对应的11的缓存。

[0131] 根据拦截的glBindbuffer(GL_ARRAY_BUFFER,1),数据处理模块就可以控制后续操作在ID为与1对应的11的缓存上进行。

[0132] 根据拦截的glBufferdata(GL_ARRAY_BUFFER,data1),数据处理模块就可以将data1写入备份缓存中的ID为11的存储空间中。

[0133] 根据拦截的glBufferSubData(GL_ARRAY_BUFFER,data2),数据处理模块就可以将data2更新到备份缓存中的ID为11的存储空间中。

[0134] 其中,data1和data2可以包括顶点数据,如顶点坐标,顶点的法向量等。

[0135] 由此,就可以实现对命令1中携带的顶点数据相关指令的备份存储。

[0136] 与顶点数据的备份存储类似的,数据处理模块还可以对MVP矩阵进行备份存储。

[0137] 示例性的,以拦截模块拦截的MVP矩阵相关指令的函数包括如下函数为例:

[0138] glGenbuffers(GL_UNIFORM_BUFFER,2)//为统一变量(如MVP矩阵)创建缓存,缓存ID为2;

[0139] glBindbuffer(GL_UNIFORM_BUFFER,2)//为统一变量(如MVP矩阵)绑定ID为2的缓存;

[0140] glBufferdata(GL_UNIFORM_BUFFER,data3)//向已经绑定的缓存写入data3;

[0141] glBufferSubData(GL_UNIFORM_BUFFER,data4)//更新已经绑定的缓存中的数据为data4。

[0142] 其中,data3和data4可以包括MVP矩阵。

[0143] 那么,该示例中的原生ID就可以为2。以对应的备份ID为22为例。

[0144] 根据拦截的glGenbuffers(GL_UNIFORM_BUFFER,2),数据处理模块就可以在备份缓存中创建ID为与2对应的22的缓存。该ID为22的备份缓存可以用于存储统一变量对应的数据,如该统一变量可以包括MVP矩阵。

[0145] 根据拦截的glBindbuffer(GL_UNIFORM_BUFFER,2),数据处理模块就可以控制后续操作在ID为与2对应的22的缓存上进行。

[0146] 根据拦截的glBufferdata(GL_UNIFORM_BUFFER,data3),数据处理模块就可以将data3写入备份缓存中的ID为22的存储空间中。

[0147] 根据拦截的glBufferSubData(GL_UNIFORM_BUFFER,data4),数据处理模块就可以将data4更新到备份缓存中的ID为22的存储空间中。

[0148] 其中,data3和data4可以包括MVP矩阵,如M矩阵,VP矩阵等。

[0149] 由此,就可以实现对命令1中携带的MVP矩阵相关指令的备份存储。

[0150] 在对指令以及相关数据进行备份存储之外,数据处理模块还可以存储包括原生ID与备份ID的对应关系的跳转表,从而可以根据后续应用下发的命令中的ID,准确地找到备份存储中对应数据的ID。

[0151] 例如,如下表1示出了跳转表的一种示例。

[0152] 表1

[0153]

原生ID	备份ID
1	11
2	22
……	……

[0154] 基于表1,在游戏应用下发指示对ID为1的缓存执行操作时,电子设备就可以确定在备份ID为11的存储空间中可以保存有对应的数据。类似的,在游戏应用下发指示对ID为2的缓存执行操作时,电子设备就可以确定在备份ID为22的存储空间中可以保存有对应的数据。

[0155] 需要说明的是,为了保证命令1的顺利执行,在本示例中,拦截模块还可以将没有携带参数1或参数2的指令(如回调指令a)回调给图形库,以便于通过调用图形库中对应的

接口，控制硬件层中的部件(如GPU)执行对应的功能。在另一些实现中，数据处理模块还可以在完成对顶点数据以及MVP矩阵的备份存储之后，将来自拦截模块的指令(如回调指令b)回调给图形库，以便通过调用图形库中对应的接口，控制硬件层中的部件(如GPU)执行对应的功能。

[0156] 这样，在实现对顶点数据以及MVP矩阵的备份存储的同时，也能够实现对命令1的完整执行，从而不会影响后续游戏应用下发的命令的执行。

[0157] 在本申请实施例中，根据加载过程中的备份存储的数据，电子设备可以实现对游戏运行过程中命令的相关处理，从而确定当前命令(即当前Drawcall)所要绘制模型的顶点坐标以及当前Drawcall对应的MVP矩阵。

[0158] 示例性的，结合图7。在游戏运行过程中，游戏应用可以发出命令2。其中，该命令2可以用于实现包括对模型的绘制。

[0159] 可以理解的是，结合前述说明，要绘制的模型的顶点数据以及MVP矩阵可以是已经通过命令1加载的。也就是说，这些数据可以是已经存储在GPU可以调用的内存空间中的。那么，在命令2中，可以通过指示需要使用的顶点数据以及MVP矩阵的相关参数，以便GPU可以从已经加载的数据中获取对应的顶点数据和MVP矩阵，从而进行对应模型的绘制。

[0160] 在本示例中，在游戏下发的该命令2中可以包括多个指令(即函数)组成的指令流。为了实现上述功能，在命令2中可以包括绑定缓存的函数，指示顶点数据解析方式的函数，指示MVP矩阵相关参数的函数。

[0161] 作为一种可能的实现，在命令2中可以至少包括以下指令：

[0162] 绑定缓存的函数，如glBindbuffer函数；

[0163] 用于指示顶点数据解析方式的glVertexAttribPoint函数；

[0164] 用于指示MVP矩阵相关参数的glBindBufferRange函数。

[0165] 那么，拦截模块可以用于在该游戏运行的过程中，拦截上述指令。在一些实施例中，拦截模块可以拦截包括glBindbuffer函数，glVertexAttribPoint函数的顶点相关指令。拦截模块还可以拦截包括glBindBufferRange函数的MVP相关指令。拦截模块还可以将该顶点相关指令以及MVP相关指令传输给数据处理模块进行解析。

[0166] 类似于上述加载过程中的数据拦截的说明，在本示例中，拦截模块也可以具有一定的数据解析能力。那么，拦截模块可以拦截携带有参数1(如GL_ELEMENT_ARRAY_BUFFER和/或GL_ARRAY_BUFFER)的glBindbuffer函数，以及glVertexAttribPoint函数。拦截模块还可以拦截携带有参数2(如GL_UNIFORM_BUFFER)的glBindBufferRange函数。

[0167] 那么，数据处理模块可以在接收到携带有参数1的glBindbuffer函数以及glVertexAttribPoint函数后，据此确定当前Drawcall所要使用的顶点坐标。

[0168] 在本申请的一些实施例中，数据处理模块可以结合本地存储的参数3，确定当前Drawcall所要使用的顶点坐标。该参数3可以是对当前游戏的静态分析确定的。

[0169] 可以理解的是，在顶点数据中可以包括多项顶点的相关数据。不同的数据可以存储在不同的属性(attribute)中。比如，attitude0可以用于存储顶点坐标，attribute1可以用于存储顶点法向量等。对于一个游戏应用而言，其在工作过程中所用于存储顶点坐标的属性的ID(如0)一般是不会变化的。因此，在本示例中，参数3中可以包括当前游戏用于存储顶点坐标的属性的ID(如0)。

[0170] 这样，数据处理模块在接收到来自拦截模块的指令后，可以根据glVertexAttribPoint函数所指示的存储数据的属性ID是否与参数3匹配，确定该glVertexAttribPoint函数是否用于进行顶点数据的传输。在glVertexAttribPoint函数指示的属性ID与参数3匹配的情况下，数据处理模块就可以确定该glVertexAttribPoint函数是顶点坐标相关的函数。

[0171] 那么，数据处理模块就可以根据该glVertexAttribPoint函数之前，通过glBindbuffer函数绑定的缓存ID，确定当前Drawcall所指示的顶点数据的存储位置。数据处理模块可以据此确定当前Drawcall所指示的顶点坐标在备份存储的缓存中的存储位置。

[0172] 例如，数据处理模块接收到来自拦截模块的指令包括：

[0173] glBindbuffer(GL_ARRAY_BUFFER,1)//绑定ID为1的缓存；

[0174] glVertexAttribPoint(0,3,GL_FLOAT,GL_FALSE,3*sizeof(float),(void*_0))//各个参数按照先后顺序分别指示：属性值为0，每组数据包括3个值(如XYZ)，类型为浮点类型，不需要进行标准化，步长为3*4，起始地址为0。

[0175] 以参数3指示的属性ID为0为例。

[0176] 数据处理模块可以确定glVertexAttribPoint函数指示的属性ID(如0)与参数3匹配，那么该glVertexAttribPoint函数所绑定的ID，即glVertexAttribPoint函数之前的glBindbuffer(GL_ARRAY_BUFFER,1)所绑定的ID为1的缓存，为用于传递当前Drawcall所要绘制模型的顶点数据的缓存。

[0177] 接着，数据处理模块可以根据原生ID与备份ID的对应关系(如表1)，确定在备份存储的数据中，存储当前Drawcall所要绘制模型的顶点坐标的缓存ID为11。接着，根据glVertexAttribPoint函数所指示的属性ID(也就是参数3所指示的ID)以及偏移量，确定该模型的各个顶点的顶点坐标。

[0178] 上述说明详细阐述了当前Drawcall要绘制模型(简称为当前模型)的顶点坐标的获取方式。以下对当前模型对应的MVP矩阵的获取进行说明。

[0179] 结合前述游戏运行过程中，拦截模块还可以拦截携带有参数2(如GL_UNIFORM_BUFFER)的glBindBufferRange函数。

[0180] 该携带有参数2的glBindBufferRange函数可以被传输给数据处理模块进行处理。

[0181] 示例性的，在本申请的一些实施例中，数据处理模块可以结合本地存储的参数4，确定当前Drawcall所要使用的MVP矩阵。该参数4可以是对当前游戏的静态分析确定的。

[0182] 在一些实施例中，该参数4可以包括存储M矩阵，和/或VP矩阵的偏移量。可以理解的是，对于确定的游戏应用，在对应缓存中存储M矩阵的偏移量，以及VP矩阵的偏移量一般是不会变化的。因此，本示例中，数据处理模块可以根据该参数4，结合拦截到的glBindBufferRange函数携带的参数，确定该拦截到的glBindBufferRange函数是否用于传输当前Drawcall对应的MVP矩阵。

[0183] 在数据处理模块确定glBindBufferRange函数携带的偏移量与参数4匹配的情况下，则数据处理模块就可以确定该glBindBufferRange函数是用于传输当前Drawcall对应的MVP矩阵的。

[0184] 那么数据处理模块就可以根据该glBindBufferRange函数所指示的缓存ID，以及上述跳转表(如表1)，确定在备份存储中，存储当前Drawcall对应的MVP矩阵的缓存ID。此

外,还可以根据glBindBufferRange函数所指示的偏移量(或参数4指示的偏移量)确在该缓存中M矩阵和VP矩阵具体的存储位置。

[0185] 例如,数据处理模块接收到来自拦截模块的指令包括:

[0186] glBindBufferRange(GL_UNIFORM_BUFFER,2,0,152)//各个参数按照先后顺序指示的含义为:目标为GL_UNIFORM_BUFFER,缓存ID为2,偏移量首地址为0,数据大小为152。

[0187] 以参数4指示的偏移量首地址0,数据大小为152为例。

[0188] 数据处理模块可以确定当前拦截到的glBindBufferRange函数指示的偏移量与参数4匹配,那么该glBindBufferRange函数是用于传递当前Drawcall对应MVP矩阵的。因此,根据glBindBufferRange函数所指示的ID(如2),数据处理模块就可以确定当前Drawcall对应的MVP矩阵对应的原生ID为2。

[0189] 接着,数据处理模块可以根据原生ID与备份ID的对应关系(如表1),确定在备份存储的数据中,存储当前Drawcall所要绘制模型的MVP矩阵的缓存ID为22。根据glBindBufferRange函数所指示的偏移量(也就是参数4所指示的偏移量),数据处理模块就可以确定该模型的MVP矩阵。

[0190] 需要说明的是,在本示例中,类似前述备份存储的过程,拦截模块还可以通过回调指令c,实现对命令2中,与顶点数据以及MVP矩阵不相关的指令的回调。数据处理模块可以通过回调指令d,实现对命令2中,与顶点数据以及MVP矩阵相关的指令的回调。

[0191] 为了能够使得本领域技术人员能更加清楚地了解本申请实施例提供方案中,关于游戏加载过程中数据的备份存储过程,以及游戏运行过程中顶点数据和MVP矩阵的确定过程,以下从指令流的角度,对该过程中各个模块的功能进行示例性说明。

[0192] 示例性的,请参考图8,为本申请实施例提供的游戏加载(或启动)时,对数据备份存储的流程示意图。如图8所示,该过程可以包括:

[0193] S801、在接收到指令P后,确定指令P是否是顶点相关指令。

[0194] 在本示例中,在游戏加载时,游戏应用可以发出指令P,该指令P可以用于进行顶点数据的加载。

[0195] 在指令P为顶点相关指令的情况下,执行以下S802。在指令P不是顶点相关指令的情况下,执行以下S811。

[0196] 结合前述示例,该顶点相关指令可以包括携带有参数1的函数。如携带有参数1的glGenbuffers函数,glBindbuffer函数,glBufferdata函数,glBufferSubData函数等。

[0197] S802、拦截模块向数据处理模块发送顶点相关指令。其中,该顶点相关指令可以包括指令P。

[0198] S803、数据处理模块控制内存备份存储顶点相关数据。

[0199] 其中,本示例中的内存可以对应到上述示例中的缓存。该内存(或缓存)可以是电子设备的内部存储介质中,可以被CPU调用的部分存储空间。

[0200] 上述S801-S803中,关于顶点数据的拦截,分析,存储等过程,与前述说明中的具体实现类似,此处不再赘述。由此即可实现对顶点数据的备份存储。

[0201] 在本申请的一些实施例中,在执行S803的过程中,数据处理模块还可以存储原生ID与备份ID的对应关系,用于后续数据的调用。

[0202] 电子设备在执行该S801-S803的过程中,还可以通过指令回调,实现游戏应用发出

的命令的正常运行，如实现数据的正常加载。示例性的，该过程可以包括：

[0203] S811、拦截模块向图形库回调指令1-1。

[0204] 示例性的，在指令P不是顶点相关指令的情况下，拦截模块可以将该指令回调给图形库。例如，该指令1-1可以包括指令P。

[0205] S812、图形库调取相关API 1-1。其中，该相关API 1-1可以是为了实现回调指令1-1的功能所调用的API。

[0206] S813、图形库向GPU发送指令1-1。在该指令1-1中可以携带API 1-1对应的代码。

[0207] S814、GPU执行指令1-1对应的操作。

[0208] 与拦截模块类似的，数据处理模块也可以对顶点相关指令进行回调。示例性的，该过程可以包括：

[0209] S815、数据处理模块向图形库回调指令1-2。该指令1-2可以包括拦截模块拦截并传输给数据处理模块的指令。比如，该指令可以包括指令P中的顶点相关指令。

[0210] S816、图形库调取相关API 1-2。该API 1-2可以是为了实现回调指令1-2的功能所调用的API。

[0211] S817、图形库向GPU发送指令1-2。在该指令1-2中可以携带API 1-2对应的代码。

[0212] S818、GPU执行指令1-2对应的操作。

[0213] 由此，通过S811-S818，就实现了对指令P中所有数据的回调，从而顺利实现指令P中数据的加载。

[0214] 在本示例中，电子设备还可以通过如下流程实现对MVP矩阵的备份存储。示例性的，该过程可以包括：

[0215] S804、在接收到指令Q后，确定指令Q是否是MVP相关指令。。

[0216] 在本示例中，在游戏加载时，游戏应用可以发出指令Q，该指令Q可以用于进行MVP数据的加载。拦截模块可以拦截该指令Q中包括的携带参数2的函数。

[0217] 如携带有参数2的glGenbuffers函数，glBindbuffer函数，glBufferdata函数，glBufferSubData函数等。

[0218] 在指令Q为MVP相关指令的情况下，执行以下S805。在指令P不是MVP相关指令的情况下，执行以下S821。

[0219] S805、拦截模块向数据处理模块发送MVP相关指令。其中，该MVP相关指令可以包括指令Q。

[0220] S806、数据处理模块控制内存备份存储MVP相关数据。

[0221] 上述S804-S806中，关于MVP矩阵的拦截，分析，存储等过程，与前述说明中的具体实现类似，此处不再赘述。由此即可实现对MVP矩阵的备份存储。

[0222] 类似于前述关于顶点数据的回调过程，在本示例中，电子设备也可以对MVP指令进行回调，实现MVP矩阵的正常加载。示例性的，该过程可以包括：

[0223] S821、拦截模块向图形库回调指令2-1。

[0224] 示例性的，在指令P不是MVP相关指令的情况下，拦截模块可以将该指令回调给图形库。例如，该指令2-1可以包括指令Q。

[0225] S822、图形库调取相关API 2-1。其中，该相关API 2-1可以是为了实现回调指令2-1的功能所调用的API。

[0226] S823、图形库向GPU发送指令2-1。在该指令2-1中可以携带API 2-1对应的代码。

[0227] S824、GPU执行指令2-1对应的操作。

[0228] 与拦截模块类似的，数据处理模块也可以对MVP相关指令进行回调。示例性的，该过程可以包括：

[0229] S825、数据处理模块向图形库回调指令2-2。该指令2-2可以包括拦截模块拦截并传输给数据处理模块的指令。比如，该指令可以包括指令Q中的MVP相关指令。

[0230] S826、图形库调取相关API 2-2。该API 2-2可以是为了实现回调指令2-2的功能所调用的API。

[0231] S827、图形库向GPU发送指令2-2。在该指令2-2中可以携带API 2-2对应的代码。

[0232] S828、GPU执行指令2-2对应的操作。

[0233] 由此，通过S821-S828，就实现了对指令Q中所有指令的回调，从而顺利实现指令Q中数据的加载。

[0234] 下面结合从指令流的角度，对游戏运行过程中，当前Drawcall所要绘制模型对应的顶点坐标和MVP矩阵的确定过程进行举例说明。

[0235] 示例性的，结合图9，以游戏应用在运行过程中发出指令N用于指示当前模型的顶点数据为例。该过程可以包括：

[0236] S901、在接收到指令N之后，拦截模块确定指令N是否是顶点相关指令。

[0237] 在本示例中，该顶点相关指令可以为指令N中携带的用于指示当前Drawcall所要绘制模型对应的顶点数据的指令。在一些实施例中，这些指令可以携带有与顶点数据相关的参数1。结合前述说明，该顶点相关指令可以包括glVertexAttribPoint函数，以及对应的glBindbuffer函数等。

[0238] 在指令N是顶点相关指令的情况下，可以执行以下S902。在指令N不是顶点相关指令的情况下，可以执行对指令N的回调，如执行S911。

[0239] S902、拦截模块向数据处理模块发送顶点相关指令。其中，该顶点相关指令可以包括指令N。

[0240] S903、数据处理模块确定传输顶点数据的缓存ID。

[0241] 示例性的，数据处理模块可以在拦截模块拦截的glVertexAttribPoint函数所指示的属性ID，与预设的参数3匹配时，确定当前拦截的函数是用于指示当前Drawcall所要绘制模型对应的顶点数据的。

[0242] 那么数据处理模块就可以根据glVertexAttribPoint函数之前的glBindbuffer函数，确定传输顶点数据的缓存ID。该缓存ID可以为原生ID。

[0243] S904、数据处理模块确定顶点数据的备份存储中的存储位置。

[0244] 示例性的，数据处理模块可以根据包括原生ID与备份ID的对应关系的跳转表，确定与当前原生ID对应的备份ID。由此即可确定当前Drawcall所要绘制模型对应的顶点数据在备份存储中的缓存ID。此外还可以根据顶点坐标所在属性ID以及偏移量，准确获取各个顶点坐标在备份存储中的存储位置。

[0245] 在本申请的一些实施例中，在确定顶点数据的备份存储中的存储位置之后，数据处理模块可以将该顶点坐标转存到预设位置1，以便后续调用。在另一些实施例中，在确定顶点数据的备份存储中的存储位置之后，数据处理模块可以对备份存储中存储当前

Drawcall对应顶点坐标的位置进行标记，以便后续调用。

[0246] 需要说明的是，为了保证指令N的正常运行，本申请实施例中，拦截模块以及数据处理模块还可以进行指令回调。示例性的，该过程可以包括：

[0247] S911、拦截模块向图形库回调指令3-1。

[0248] 示例性的，在指令N不是顶点相关指令的情况下，可以执行该步骤实现对指令N的回调。在一些实施例中，该指令3-1可以包括指令N。

[0249] S912、图形库调取相关API 3-1。该API 3-1可以是图形库中为了实现指令3-1对应功能的API。

[0250] S913、图形库向GPU发送指令3-1。该指令3-1可以包括API 3-1对应的代码。

[0251] S914、GPU执行指令3-1对应的操作。

[0252] 需要说明的是，在一些实施例中，上述S911-S914的执行，可以是在S902之后执行的。

[0253] 与拦截模块类似的，数据处理模块也可以执行指令回调。示例性的，该过程可以包括：

[0254] S915、数据处理模块向图形库回调指令3-2。

[0255] 示例性的，该指令3-2可以包括指令N中，拦截模块拦截的顶点相关指令。

[0256] S916、图形库调取相关API 3-2。该API 3-2可以是图形库中为了实现指令3-2对应功能的API。

[0257] S917、图形库向GPU发送指令3-2。该指令3-2可以包括API 3-2对应的代码。

[0258] S918、GPU执行指令3-2对应的操作。

[0259] 在一些实施例中，上述S915-S918的执行，可以是在S904之后执行的。

[0260] 由此，就实现了指令N的全量回调，从而保证指令N的正常执行。

[0261] 在本示例中，电子设备还可以通过如下流程确定当前Drawcall对应的MVP矩阵在备份存储中的存储位置。示例性的，结合图9，以游戏应用在运行过程中发出指令M用于指示当前模型的MVP矩阵为例。该过程可以包括：

[0262] S905、在接收到指令M之后，拦截模块确定指令N是否是MVP相关指令。

[0263] 在本示例中，该MVP相关指令可以为指令M中携带的用于指示当前Drawcall所要绘制模型对应的MVP矩阵的指令。在一些实施例中，这些指令可以携带有与MVP矩阵相关的参数2。结合前述说明，该MVP相关指令可以包括glBindBufferRange函数等。

[0264] 在指令M是MVP相关指令的情况下，可以执行以下S906。在指令M不是MVP相关指令的情况下，可以执行对指令M的回调，如执行S921。

[0265] S906、拦截模块向数据处理模块发送MVP相关指令。

[0266] S907、数据处理模块确定传输MVP矩阵的缓存ID。

[0267] 示例性的，数据处理模块可以在拦截模块拦截的glBindBufferRange函数所指示的偏移量，与预设的参数4匹配时，确定当前拦截的函数是用于指示当前Drawcall所要绘制模型对应的MVP矩阵的。

[0268] 那么数据处理模块就可以根据glBindBufferRange函数指示的缓存的ID，确定传输MVP矩阵的缓存ID。该缓存ID可以为原生ID。

[0269] S908、数据处理模块确定MVP矩阵的备份存储中的存储位置。

[0270] 示例性的，数据处理模块可以根据包括原生ID与备份ID的对应关系的跳转表，确定与当前原生ID对应的备份ID。由此即可确定当前Drawcall所要绘制模型对应的MVP矩阵在备份存储中的缓存ID。此外还可以根据MVP矩阵的偏移量，准确获取M矩阵，和/或VP矩阵在备份存储中的存储位置。

[0271] 在本申请的一些实施例中，类似于前述顶点数据的处理，在确定MVP矩阵的备份存储中的存储位置之后，数据处理模块可以将该MVP矩阵转存到预设位置2，以便后续调用。在另一些实施例中，在确定MVP矩阵的备份存储中的存储位置之后，数据处理模块可以对备份存储中存储当前Drawcall对应MVP矩阵的位置进行标记，以便后续调用。

[0272] 需要说明的是，为了保证指令M的正常运行，本申请实施例中，拦截模块以及数据处理模块还可以进行指令回调。示例性的，该过程可以包括：

[0273] S921、拦截模块向图形库回调指令4-1。

[0274] 示例性的，在指令M不是MVP相关指令的情况下，可以执行该步骤实现对指令M的回调。在一些实施例中，该指令4-1可以包括指令M。

[0275] S922、图形库调取相关API 4-1。该API 4-1可以是图形库中为了实现指令4-1对应功能的API。

[0276] S923、图形库向GPU发送指令4-1。该指令4-1可以包括API 4-1对应的代码。

[0277] S924、GPU执行指令4-1对应的操作。

[0278] 需要说明的是，在一些实施例中，上述S921-S924的执行，可以是在S906之后执行的。

[0279] 与拦截模块类似的，数据处理模块也可以执行指令回调。示例性的，该过程可以包括：

[0280] S925、数据处理模块向图形库回调指令4-2。

[0281] 示例性的，该指令4-2可以包括指令M中，拦截模块拦截的MVP相关指令。

[0282] S926、图形库调取相关API 4-2。该API 4-2可以是图形库中为了实现指令4-2对应功能的API。

[0283] S927、图形库向GPU发送指令4-2。该指令4-2可以包括API 4-2对应的代码。

[0284] S928、GPU执行指令4-2对应的操作。

[0285] 在一些实施例中，上述S925-S928的执行，可以是在S908之后执行的。

[0286] 由此，就实现了指令N的全量回调，从而保证指令M的正常执行。

[0287] 通过上述示例，电子设备就可以获取当前Drawcall对应模型的顶点坐标以及MVP矩阵。

[0288] 在本申请实施例中，电子设备还可以据此计算当前模型的深度，并进而使用合理的着色速率对该模型进行渲染处理。

[0289] 示例性的，结合图10计算模块，可以用于根据内存中存储的顶点坐标以及MVP矩阵，确定当前drawcall所要绘制的图形(或模型)的深度。

[0290] 其中，模型的深度可以通过该模型上的部分或全部顶点的深度确定。

[0291] 以下首先对顶点的深度确定方式进行说明。

[0292] 示例性的，在一些实施例中，在拦截模块拦截到Drawelement的情况下，可以触发计算模块对各个顶点的深度的计算。

[0293] 可以理解的是，在本申请实施例中，计算模块可以通过计算各个顶点的深度，配合决策模块确定是否要对部分顶点进行低速率着色。因此，在本申请的不同实现中，计算模块计算各个顶点的深度的触发机制可以是不同的。只要在向GPU下发对应的drawelement之前，计算模块和/或决策模块能够知晓对该drawelement对应的顶点深度即可。比如，在本示例中，计算模块对各个顶点的深度的计算，可以是由拦击模块拦截到drawelement触发的。在另一些实现中，计算模块也可以在内存中缓存了顶点坐标以及MVP矩阵参数之后，即触发对各个顶点的深度的计算，这样计算模块可以将计算获取的深度信息存储在内存中，以便于后续使用。

[0294] 在本申请的一些实施例中，计算模块可以根据数据处理模块确定的当前Drawcall对应的顶点坐标以及MVP矩阵的存储地址，调用对应的数据，计算获取各个顶点的深度信息。

[0295] 示例性的，以顶点坐标指示的顶点1的局部坐标为(x1,y1,z1)为例。计算模块可以根据以下公式(1)，对顶点1的局部坐标进行MVP变换，进而确定顶点1在裁剪空间(或屏幕空间)中的深度。

[0296] 裁剪坐标＝P•V•M•(x1,y1,z1)……公式(1)。

[0297] 以计算获取的顶点1的裁剪坐标为(x2,y2,z2)为例。那么计算模块可以确定当前顶点1的深度为z2。

[0298] 与上述顶点1类似的，计算模块可以对当前drawcall中对应的模型的其他顶点进行深度计算，以获取各个顶点的深度情况。可以理解的是，该深度可以为裁剪空间下的深度。深度越深，则表明该顶点(或模型)在当前帧图像显示过程中距离用户观察位置越远，因此在对该具有较大深度的顶点(或模型)执行较低速率着色时，不会被用户所察觉。

[0299] 结合上述顶点深度的计算方式，在本申请的一些实施例中，电子设备可以据此确定当前drawcall对应模型的深度。

[0300] 示例性的，作为一种可能的实现方式，电子设备可以计算当前模型的所有顶点的深度，并将这些深度的均值作为模型的深度。比如，计算模块可以在触发深度计算的情况下，从存储器中调取当前存储的所有顶点坐标，并根据MVP矩阵，按照上述公式(1)进行矩阵变换，从而获取各个顶点的裁剪坐标，进而获取各个顶点的深度。电子设备可以取这些顶点深度的均值，作为该模型的深度。

[0301] 需要说明的是，一般而言，即使一个drawcall对应的模型的顶点数量也是非常庞大的。因此，在本申请的一些实现方式中，电子设备可以从内存中存储的顶点坐标中，选取一部分计算深度，并据此确定当前模型的深度。从而达到降低计算开销的效果。

[0302] 示例性的，电子设备可以根据预设的顶点数量(如n个)，从当前drawcall对应模型的顶点中，随机选取n个顶点。通过确定这n个顶点的深度，并取均值从而获取当前模型的深度。

[0303] 作为一种可能的实现，计算模块可以在触发深度计算的情况下，从存储器中调取当前存储的所有顶点坐标中，随机的n个顶点坐标。计算模块可以根据MVP矩阵，按照上述公式(1)对着n个顶点坐标分别进行矩阵变换，从而获取这n个顶点的裁剪坐标。计算模块可以根据这n个顶点的裁剪坐标，获取各个顶点的深度。计算模块还可以根据获取的n个顶点的深度，进行加权平均计算，从而获取该模型的深度。其中，加权平均中涉及的权重，可以是预

先设置的，也可以是灵活调整的。

[0304] 需要说明的是，上述示例中，是以计算模块从存储器中调取顶点坐标时，就实现了n个顶点的随机选取为例进行说明的。在另一些实现中，计算模块还可以从存储器中调取所有存储的顶点坐标，在进行深度计算时，计算模块可以从调取的所有顶点坐标中，选取n个顶点坐标进行深度计算，其他顶点坐标不作处理或者丢弃。由此也可以实现n个顶点坐标的随机选取。

[0305] 在本申请的另一些实施例中，电子设备可以根据当前drawcall对应模型的大小，采取不同的策略确定该模型的深度。其中，模型的大小可以通过顶点数量进行标识。

[0306] 以拦截模块拦截到drawelement触发计算模块的深度计算为例。

[0307] 电子设备可以根据拦截到的drawelement中指示的GLsizel count数值，确定当前模型的顶点数量。

[0308] 电子设备可以根据该确定的顶点数量，与预设的顶点数量阈值的大小关系，确定当前模型为大模型或小模型。比如，该顶点数量阈值可以为5000。那么，当count值大于5000时，则认为当前模型为大模型。对应的，当count值小于5000时，则认为当前模型为小模型。

[0309] 本申请实施例分别针对大模型和小模型，提供不同的深度计算策略，以便于更加合理地计算模型深度。

[0310] 示例性的，对于大模型，电子设备可以选取当前模型的所有顶点中，x，y，z分量最大或最小的6个顶点。计算着6个顶点的深度，并取均值，确定为当前模型的深度。

[0311] 比如，以当前模型的顶点中，x分量最大的顶点P1坐标为(x_{max}，y1，z1)，x分量最小的顶点P2坐标为(x_{min}，y2，z2)，y分量最大的顶点P3坐标为(x3，y_{max}，z3)，y分量最小的顶点P4坐标为(x4，y_{min}，z4)，z分量最大的顶点P5坐标为(x5，y5，z_{max})，x分量最小的顶点P6坐标为(x6，y6，z_{min})为例。

[0312] 经过MVP矩阵变换之后，P1-P6在裁剪坐标下的坐标分别为：P1(x'_{max}，y'1，z'1)，P2(x'_{min}，y'2，z'2)，P3(x'3，y'_{max}，z'3)，P4(x'4，y'_{min}，z'4)，P5(x'5，y'5，z'_{max})，P6(x'6，y'6，z'_{min})。

[0313] 那么，电子设备可以确定当前模型的深度为(z'1+z'2+z'3+z'4+z'_{max}+z'_{min})/6。

[0314] 在另一些实现中，对于小模型，电子设备可以在当前模型的所有顶点中，随机选取n个顶点。计算这n个顶点的深度，并取均值作为当前模型的深度。例如，该n可以是预先配置的，如n可以在20-100之间取任意整数。

[0315] 可以理解的是，在上述示例中，对于大模型，电子设备可以选取最边缘的顶点用于计算模型的深度，从而使得在模型的不同顶点深度差异较大的情况下，能够通过该计算方法真实准确地标识大模型的深度。而对于小模型，由于模型尺寸的限制，因此该模型不同顶点的深度不会有较大差别，因此，在本示例中，可以从模型上选取预设个数(或者随机个数)的顶点，分别计算其深度并取均值确定为该模型的深度。

[0316] 需要说明的是，上述示例中，是以裁剪坐标下的z坐标值作为对应顶点的深度为例进行说明的。在本申请的另一些实施例中，顶点的深度还可以是根据裁剪坐标下的z坐标值确定的。示例性的，顶点深度可以根据标准化设备坐标(Normalized Device Coordinate，NDC)下的顶点的z坐标确定。例如，在获取顶点在裁剪空间中的坐标之后，可以通过透视除法，获取NDC空间中的z坐标。这样，该NDC空间中的z坐标就可以用于标识该顶点的深度。此

外，上述示例中，根据裁剪空间的深度z确定模型深度，用于标识该模型在展示给用户时的图像中的远近程度，仅为一种可能的实现。在本申请另一些示例中，模型的远近程度(即模型的深度)还可以通过该模型在观察空间中的基于观察坐标的模长标识的。也就是说，模型的远近程度也可以是在观察空间中，根据模型到摄像机的距离确定的。

[0317] 可以看到，本申请的上述说明中，电子设备中的各个模块，可以通过获取模型的顶点坐标，并对这些顶点坐标进行处理，获取模型的深度。在本申请的另一些实现中，电子设备中的各个模块还可以通过获取其他模型的信息，确定模型的深度。示例性的，以获取模型的包围盒信息确定模型的深度为例。拦截模块可以用于将渲染命令中的包围盒信息相关指令进行拦截，并将这些信息传输给数据处理模块。数据处理模块和/或计算模块可以用于根据拦截获取的包围盒信息，确定当前模型的深度。

[0318] 这样，电子设备就可以通过计算模块的计算，获取当前drawcall对应模型的深度。

[0319] 在本申请实施例中，计算模块可以配合决策模块，确定是否对当前drawcall对应模型执行可变速率着色，如对当前drawcall对应模型执行较低速率着色。

[0320] 示例性的，决策模块可以根据计算模块计算获取的模型的深度，确定对该模型执行的着色速率。在本申请实施例中，决策模块中可以配置有(或者从内存中调取)不同模型深度对应的着色速率。如果当前的模型深度对应的着色速率为较低的着色速率(即着色速率小于以1个像素为单位的着色速率)时，决策模块可以在向硬件层传递该drawcall对应的drawelement时，从系统库的图形库中调取对应的可变速率着色API。

[0321] 作为一种示例，以下表2示出了一种模型深度与着色速率的对应关系示意。

[0322] 表2

[0323]

模型深度	着色速率
[1,10)	1×1
(10,50]	2×1
(50,100]	2×2
(100,200]	4×2
大于200	4×4

[0324] 如表2所示，在模型深度小于10的情况下，则决策模块可以确定当前模型使用1×1的着色速率进行着色。即当前模型的深度较小，不需要使用可变速率着色机制。那么决策模块可以不需调用可变速率着色API，而直接将游戏应用下发的渲染命令对应的渲染指令下发给硬件层(如下发给硬件层的GPU)，以便于GPU以1个像素为单位，执行当前渲染命令的渲染操作。

[0325] 在模型深度大于10且小于或等于50的情况下，则决策模块可以确定当前模型使用2×1的着色速率进行着色。那么决策模块可以在向硬件层(如硬件层的GPU)下发渲染命令对应的渲染指令时，调用可变速率着色API，并将着色速率设置为2×1，以便于GPU通过2×1的着色速率执行当前渲染命令的渲染操作。

[0326] 在模型深度大于50且小于或等于100的情况下，则决策模块可以确定当前模型使用2×2的着色速率进行着色。那么决策模块可以在向硬件层(如硬件层的GPU)下发渲染命令对应的渲染指令时，调用可变速率着色API，并将着色速率设置为2×2，以便于GPU通过2×2的着色速率执行当前渲染命令的渲染操作。

[0327] 在模型深度大于100且小于或等于200的情况下，则决策模块可以确定当前模型使用4×2的着色速率进行着色。那么决策模块可以在向硬件层(如硬件层的GPU)下发渲染命令对应的渲染指令时，调用可变速率着色API，并将着色速率设置为4×2，以便于GPU通过4×2的着色速率执行当前渲染命令的渲染操作。

[0328] 在模型深度大于200的情况下，则决策模块可以确定当前模型使用4×4的着色速率进行着色。那么决策模块可以在向硬件层(如硬件层的GPU)下发渲染命令对应的渲染指令时，调用可变速率着色API，并将着色速率设置为4×4，以便于GPU通过4×4的着色速率执行当前渲染命令的渲染操作。

[0329] 此外，拦截模块还可以将命令2的其他指令回调给图形库，以便通过调用图形库中对应的API，可以采用上述方案中确定的着色速率执行对应的渲染操作。

[0330] 为了使得本领域技术人员能够更加清楚地明确本申请实施例提供的方案，以下从指令流的角度，对本申请实施例提供的方案继续进行说明。示例性的，参考图11，该流程可以包括：

[0331] S1101、拦截模块以及数据处理模块确定当前模型在备份存储中的顶点坐标和MVP矩阵。

[0332] 示例性的，拦截模块可以根据游戏应用下发的指令N，确定当前模型的顶点坐标在备份存储中的存储位置。数据处理模块可以根据游戏应用下发的指令M，确定当前模型的MVP矩阵在备份存储中的存储位置。

[0333] 该S1101的具体执行过程，可以参考图9的相关说明，此处不再赘述。

[0334] S1102、计算模块从内存中获取顶点坐标以及MVP矩阵。

[0335] 示例性的，在数据处理模块将顶点坐标以及MVP矩阵存储到特定位置的情况下，计算模块可以从该特定位置获取顶点坐标以及MVP矩阵。在另一些实施例中，在数据处理模块对顶点坐标以及MVP矩阵的存储位置作了标识的情况下，计算模块可以根据该标识，从内存中获取顶点坐标以及MVP矩阵。

[0336] 在本申请的一些实施例中，计算模块获取顶点坐标以及MVP矩阵可以是游戏应用下发指令R之后进行的。其中，该指令R中可以包括一个或多个Drawelement。

[0337] 示例性的，拦截模块可以在接收到指令R时，指示计算模块执行该S1102。

[0338] S1103、计算模块计算模型深度。

[0339] S1104、计算模块向决策模块发送模型深度。

[0340] S1105、决策模块确定当前模型的着色速率1。

[0341] 其中，确定模型深度，以及根据模型深度确定对应的着色速率(如着色速率1)的过程，可以参考上述说明，此处不再赘述。

[0342] S1106、决策模块调用图形库的调用对应API。示例性的，该对应API可以是与着色速率1对应的可变速率着色API 1。

[0343] S1107、图形库对应调用可变速率着色API 1。

[0344] 这样，电子设备就可以根据模型的深度，调用图形库中对应的API，以便于控制硬件层中的部件(如GPU)执行对应的可变速率着色。

[0345] 结合前述说明，在本示例中，电子设备可以在执行上述S1101-S1107之外，将接收到的游戏应用的命令进行指令回调，从而使得指令得到正确执行。

[0346] 示例性的,该过程可以包括:

[0347] S1108、拦截模块以及数据处理模块回调相关指令。

[0348] 其中,该相关指令可以包括指令N对应的回调指令3-1,回调指令3-2,以及指令M对应的回调指令4-1,回调指令4-2。该回调过程以及内容请参考图9的说明,此处不再赘述。

[0349] 在本示例中,该相关指令还可以包括指令R的指令。

[0350] S1109、图形库调用对应的API。该对应API可以包括用于实现指令N,指令M以及指令R所指示功能的API。

[0351] S1110、图形库向GPU发送指令2。该指令2可以包括上述通过调用API获取的对应代码。示例性的,该指令2中可以包括调用可变速率着色API 1获取的对应代码,基于该代码可以实现着色速率1的可变速率着色。

[0352] S1111、GPU执行基于着色速率1对应的渲染操作。

[0353] 这样,电子设备就能够实现以drawcall为粒度,根据当前渲染命令中模型的深度,灵活调整可变速率着色的使用机制,从而达到在不影响用户体验的同时,降低着色处理过程中的渲染开销的效果。

[0354] 应当理解的是,上述示例中,是以游戏应用下发命令进行一个模型绘制为例,对其中的具体实施细节进行说明的。在实际实施过程中,游戏应用可以下发多个命令,进行多个模型的绘制。在各个模型的绘制过程中,也可以采用上述方案实现根据各个模型的深度对其进行自适应的可变速率着色。

[0355] 示例性的,以命令1用于进行数据加载,命令2用于绘制模型1,命令3用于绘制模型2为例。

[0356] 参考图12,游戏应用可以分别下发命令1,命令2,以及命令3。

[0357] 对于命令1,框架层设置的各个模块可以根据如图6或图8所示的方案进行顶点数据和/或MVP矩阵的备份存储。同时,框架层中的各个模块还可以结合如图8所示的回调机制,实现命令1对应数据的加载。比如,框架层可以根据命令1,调用图形库中响应的API,以便向GPU下发命令1对应的指令1,从而完成数据的加载。

[0358] 对于命令2,框架层设置的各个模块可以根据如图7,图9,图10以及图11所示的方案,根据确定该命令2要绘制的模型1的深度,进而确定模型1对应的着色速率(如着色速率1)。结合回调机制,框架层可以基于命令2,以及着色速率1,调用图形库中对应的API,以便向GPU下发指令2。GPU就可以根据该指令2,按照着色速率1的着色参数,进行模型1的绘制。由此即可获取如图12所示的模型1。作为一种示例,该模型1可以被存储在预先分配的与命令2对应的帧缓冲(如帧缓冲1)中。

[0359] 类似于命令2,对于命令3,框架层设置的各个模块可以根据如图7,图9,图10以及图11所示的方案,根据确定该命令3要绘制的模型2的深度,进而确定模型2对应的着色速率(如着色速率2)。结合回调机制,框架层可以基于命令3,以及着色速率2,调用图形库中对应的API,以便向GPU下发指令3。GPU就可以根据该指令3,按照着色速率2的着色参数,进行模型2的绘制。由此即可获取如图12所示的模型2。作为一种示例,该模型1可以被存储在预先分配的与命令2对应的帧缓冲(如帧缓冲2)中。

[0360] 以模型1和模型2属于同一个帧图像为例。在展示该帧图像之前,GPU还可以将帧缓冲1和帧缓冲2上的图像渲染到一个基础帧缓冲(如帧缓冲0)上,从而获取该帧图像的全量

内容。示例性的，参考图13，GPU可以将模型1和模型2渲染到同一个画布上。

[0361] 应当理解的是，由于采用了本申请实施例提供的可变速率着色的机制，对于不同深度的模型的深度可以是不同的。以模型1的深度小于模型2的深度为例。即模型1更加靠近用户，而模型2会距离用户远一些。结合前述说明，对于深度较小的模型1，可以采用较高的着色速率对其进行着色。比如，以模型1的着色速率为1×1为例。那么对模型1的着色过程中，可以以一个像素为单位，以此对模型1包括的各个像素进行着色。对应的，对于深度较大的模型2，可以采用较低的着色速率对其进行着色。比如，以模型2的着色速率为2×2为例。那么对模型2的着色过程中，可以以2×2个像素为单位，以此对模型2包括像素进行着色。

[0362] 由此，完成着色操作之后(如完成所有渲染操作，并将两个模型渲染到一个画布上之后)，在模型1包括的像素中，就可以看到每个像素的颜色都可以是不同的。而对于模型2包括的像素中，就会存在大量相邻的2×2个像素的颜色相同或相近。由此，就能够使得对于距离用户较近的模型的色彩得到更加精确的渲染和展示。对应的，距离用户较远的模型的色彩粒度可以较大，由此节省对应的渲染开销。

[0363] 上述主要从电子设备的角度对本申请实施例提供的方案进行了介绍。为了实现上述功能，其包含了执行各个功能相应的硬件结构和/或软件模块。本领域技术人员应该很容易意识到，结合本文中所公开的实施例描述的各示例的单元及算法步骤，本申请能够以硬件或硬件和计算机软件的结合形式来实现。某个功能究竟以硬件还是计算机软件驱动硬件的方式来执行，取决于技术方案的特定应用和设计约束条件。专业技术人员可以对每个特定的应用来使用不同方法来实现所描述的功能，但是这种实现不应认为超出本申请的范围。

[0364] 本申请实施例可以根据上述方法示例对其中涉及的设备进行功能模块的划分，例如，可以对应各个功能划分各个功能模块，也可以将两个或两个以上的功能集成在一个处理模块中。上述集成的模块既可以采用硬件的形式实现，也可以采用软件功能模块的形式实现。需要说明的是，本申请实施例中对模块的划分是示意性的，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式。

[0365] 示例性的，图14示出了的一种电子设备1400的组成示意图。如图14所示，该电子设备1400可以包括：处理器1401和存储器1402。该存储器1402用于存储计算机执行指令。示例性的，在一些实施例中，当该处理器1401执行该存储器1402存储的指令时，可以使得该电子设备1400执行上述实施例中任一种所示的图像渲染方法。

[0366] 需要说明的是，上述方法实施例涉及的各步骤的所有相关内容均可以援引到对应功能模块的功能描述，在此不再赘述。

[0367] 在上述实施例中的功能或动作或操作或步骤等，可以全部或部分地通过软件、硬件、固件或者其任意组合来实现。当使用软件程序实现时，可以全部或部分地以计算机程序产品的形式来实现。该计算机程序产品包括一个或多个计算机指令。在计算机上加载和执行计算机程序指令时，全部或部分地产生按照本申请实施例所述的流程或功能。所述计算机可以是通用计算机、专用计算机、计算机网络、或者其他可编程装置。所述计算机指令可以存储在计算机可读存储介质中，或者从一个计算机可读存储介质向另一个计算机可读存储介质传输，例如，所述计算机指令可以从一个网站站点、计算机、服务器或者数据中心通过有线(例如同轴电缆、光纤、数字用户线(digital subscriber line，DSL))或无线(例如

红外、无线、微波等)方式向另一个网站站点、计算机、服务器或数据中心进行传输。所述计算机可读存储介质可以是计算机能够存取的任何可用介质或者是包括一个或多个可以用介质集成的服务器、数据中心等数据存储设备。所述可用介质可以是磁性介质(例如,软盘、硬盘、磁带),光介质(例如,DVD)、或者半导体介质(例如固态硬盘(solid state disk,SSD))等。

[0368] 尽管结合具体特征及其实施例对本申请进行了描述,显而易见的,在不脱离本申请的精神和范围的情况下,可对其进行各种修改和组合。相应地,本说明书和附图仅仅是所附权利要求所界定的本申请的示例性说明,且视为已覆盖本申请范围内的任意和所有修改、变化、组合或等同物。显然,本领域的技术人员可以对本申请进行各种改动和变型而不脱离本申请的精神和范围。这样,倘若本申请的这些修改和变型属于本申请权利要求及其等同技术的范围之内,则本申请也意图包括这些改动和变型在内。

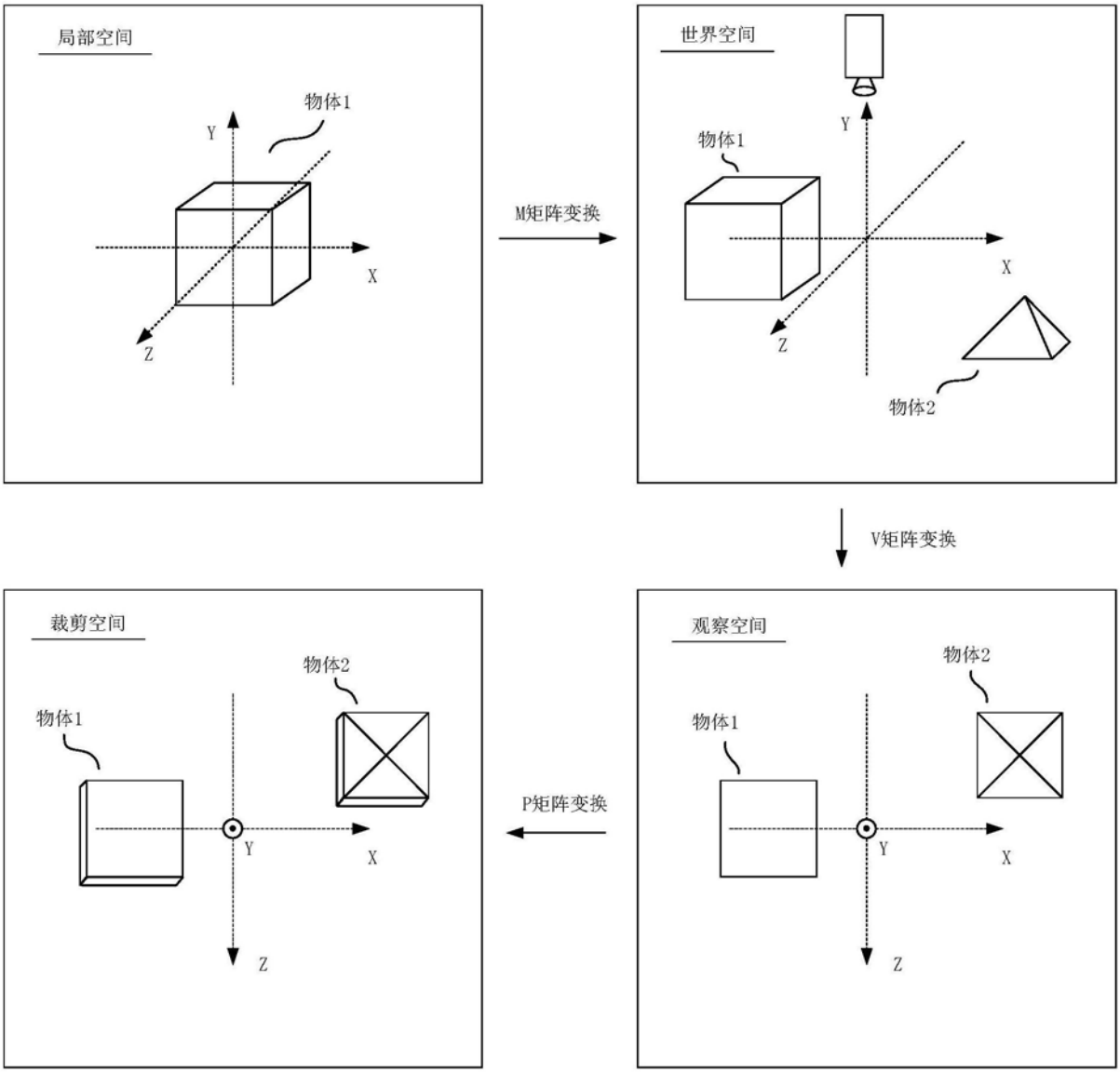
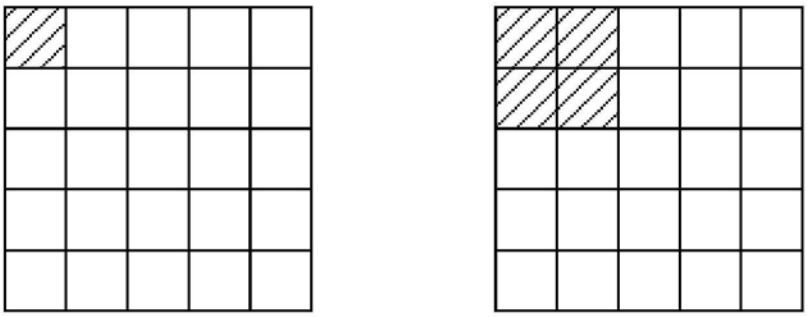

局部空间
物体1
Y
X
Z
M矩阵变换
世界空间
物体1
Y
X
Z
物体2
V矩阵变换
裁剪空间
物体2
物体1
Y
X
Z
P矩阵变换
观察空间
物体2
物体1
Y
X
Z

图1



（a） （b）

图2

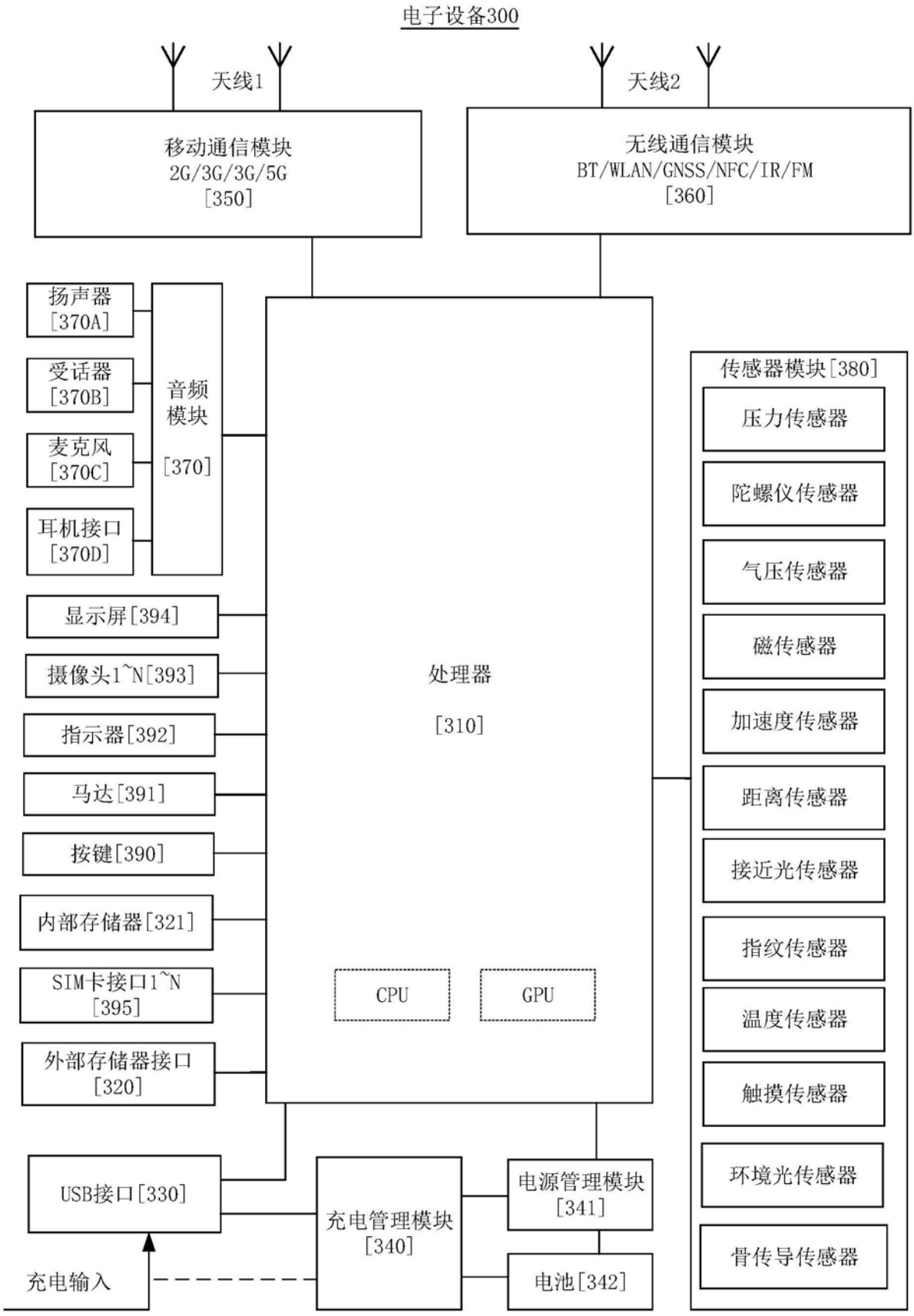

电子设备300
天线1
天线2
移动通信模块
2G/3G/3G/5G
[350]
无线通信模块
BT/WLAN/GNSS/NFC/IR/FM
[360]
扬声器
[370A]
受话器
[370B]
麦克风
[370C]
耳机接口
[370D]
音频
模块
[370]
显示屏[394]
摄像头1~N[393]
指示器[392]
马达[391]
按键[390]
内部存储器[321]
SIM卡接口1~N
[395]
外部存储器接口
[320]
处理器
[310]
CPU
GPU
传感器模块[380]
压力传感器
陀螺仪传感器
气压传感器
磁传感器
加速度传感器
距离传感器
接近光传感器
指纹传感器
温度传感器
触摸传感器
环境光传感器
骨传导传感器
USB接口[330]
充电管理模块
[340]
电源管理模块
[341]
电池[342]
充电输入

图3

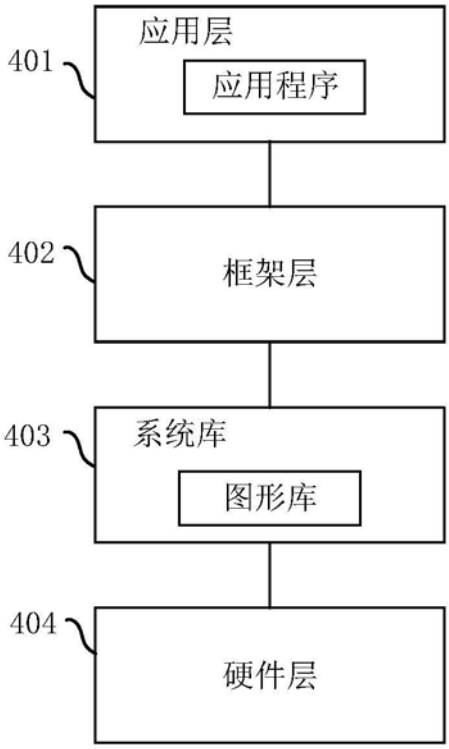

401
应用层
应用程序
402
框架层
403
系统库
图形库
404
硬件层

图4

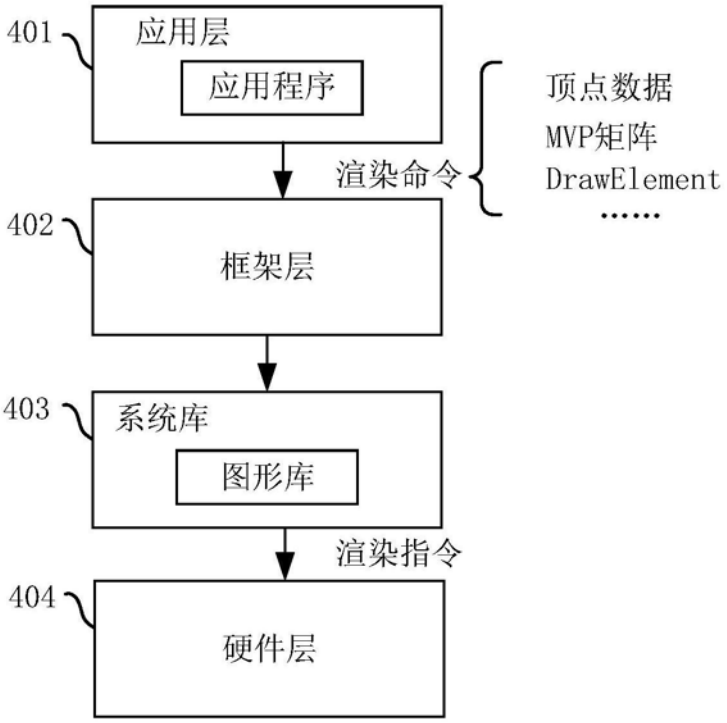

401
应用层
应用程序
渲染命令
顶点数据
MVP矩阵
DrawElement
.....
402
框架层
403
系统库
图形库
渲染指令
404
硬件层

图5

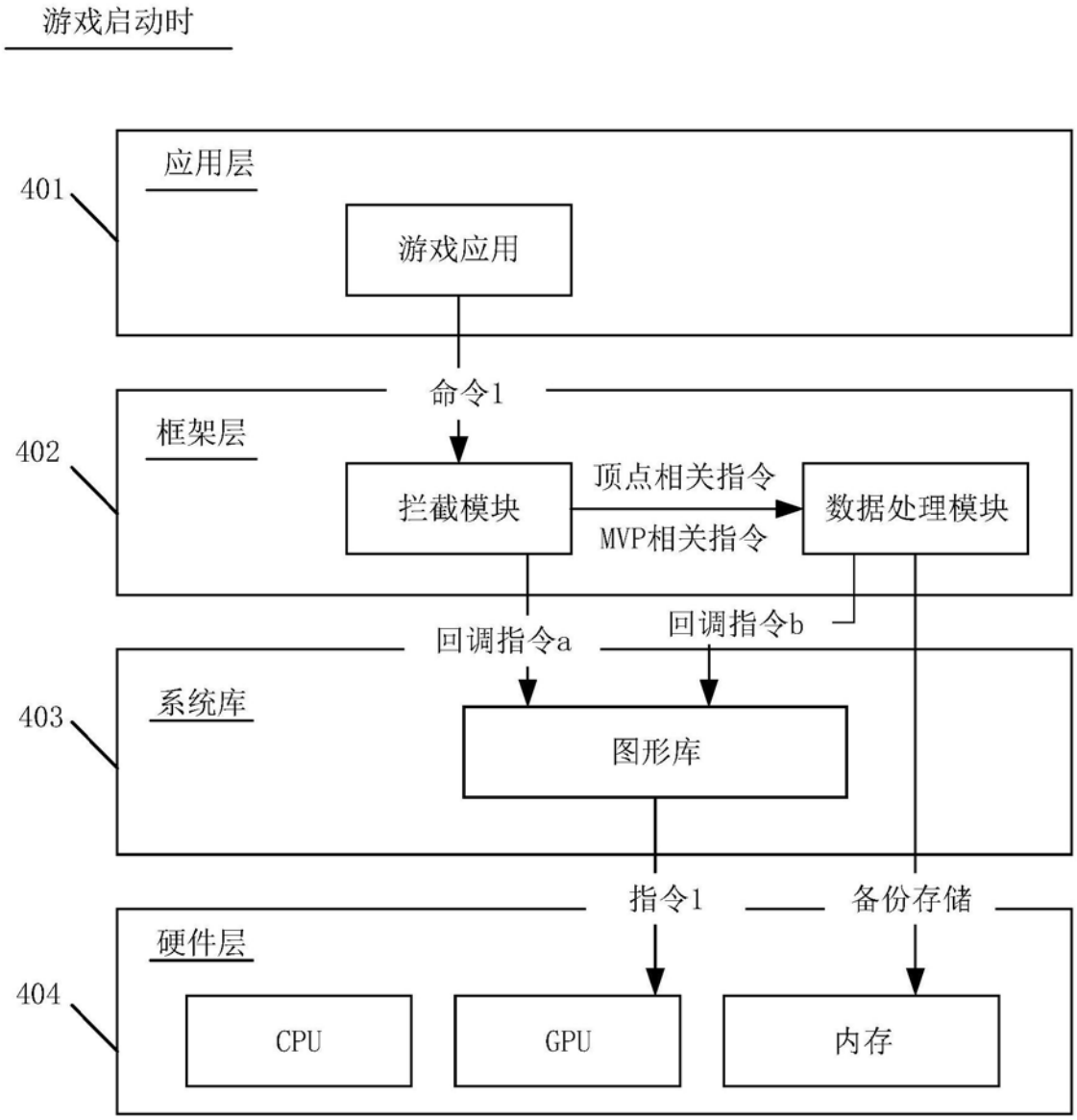

游戏启动时
401
应用层
游戏应用
命令1
402
框架层
拦截模块
顶点相关指令
MVP相关指令
数据处理模块
回调指令a
回调指令b
403
系统库
图形库
指令1
备份存储
404
硬件层
CPU
GPU
内存

图6

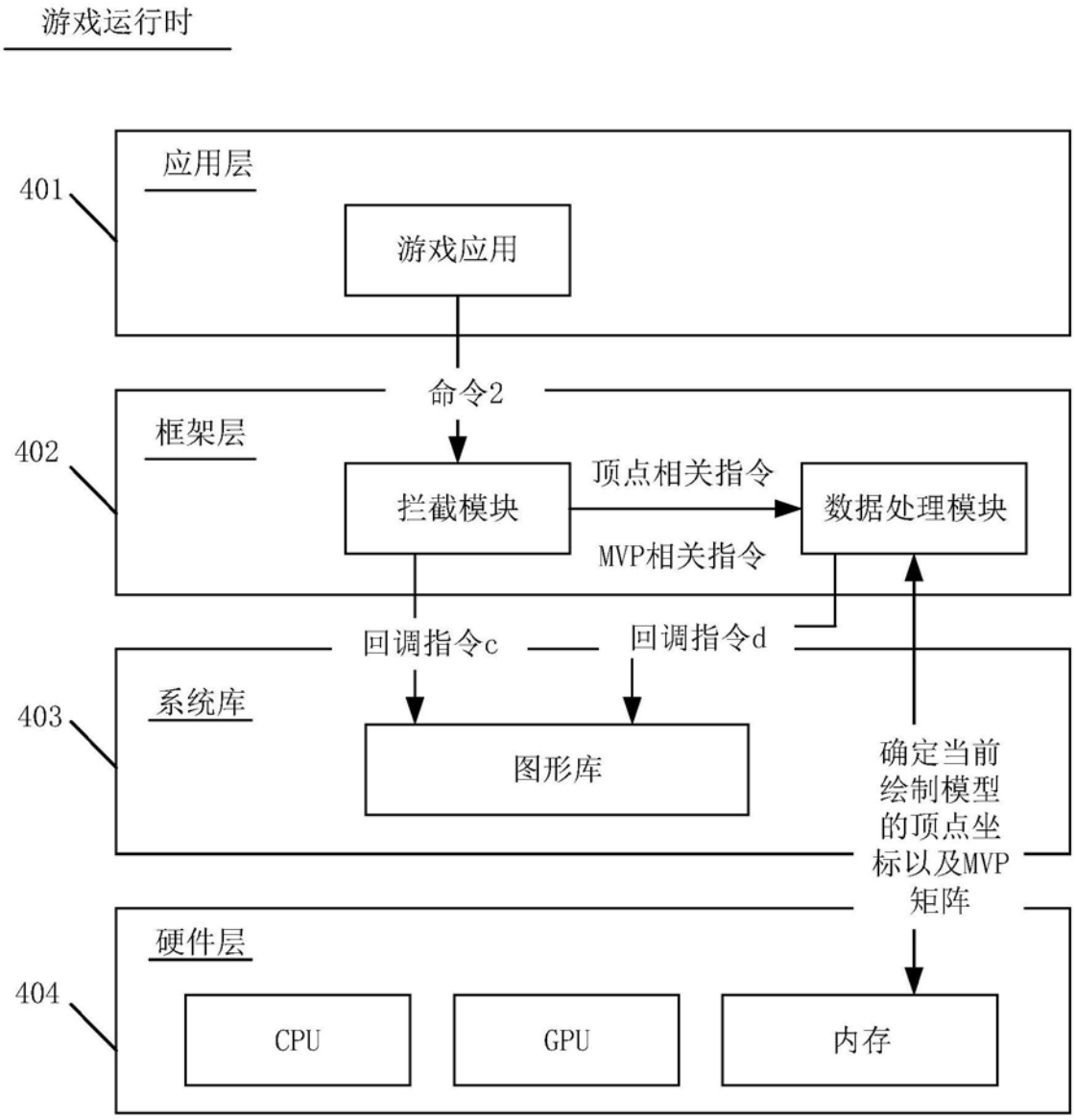

游戏运行时
401
应用层
游戏应用
命令2
402
框架层
拦截模块
顶点相关指令
数据处理模块
MVP相关指令
回调指令c
回调指令d
403
系统库
图形库
确定当前绘制模型的顶点坐标以及MVP矩阵
404
硬件层
CPU
GPU
内存

图7

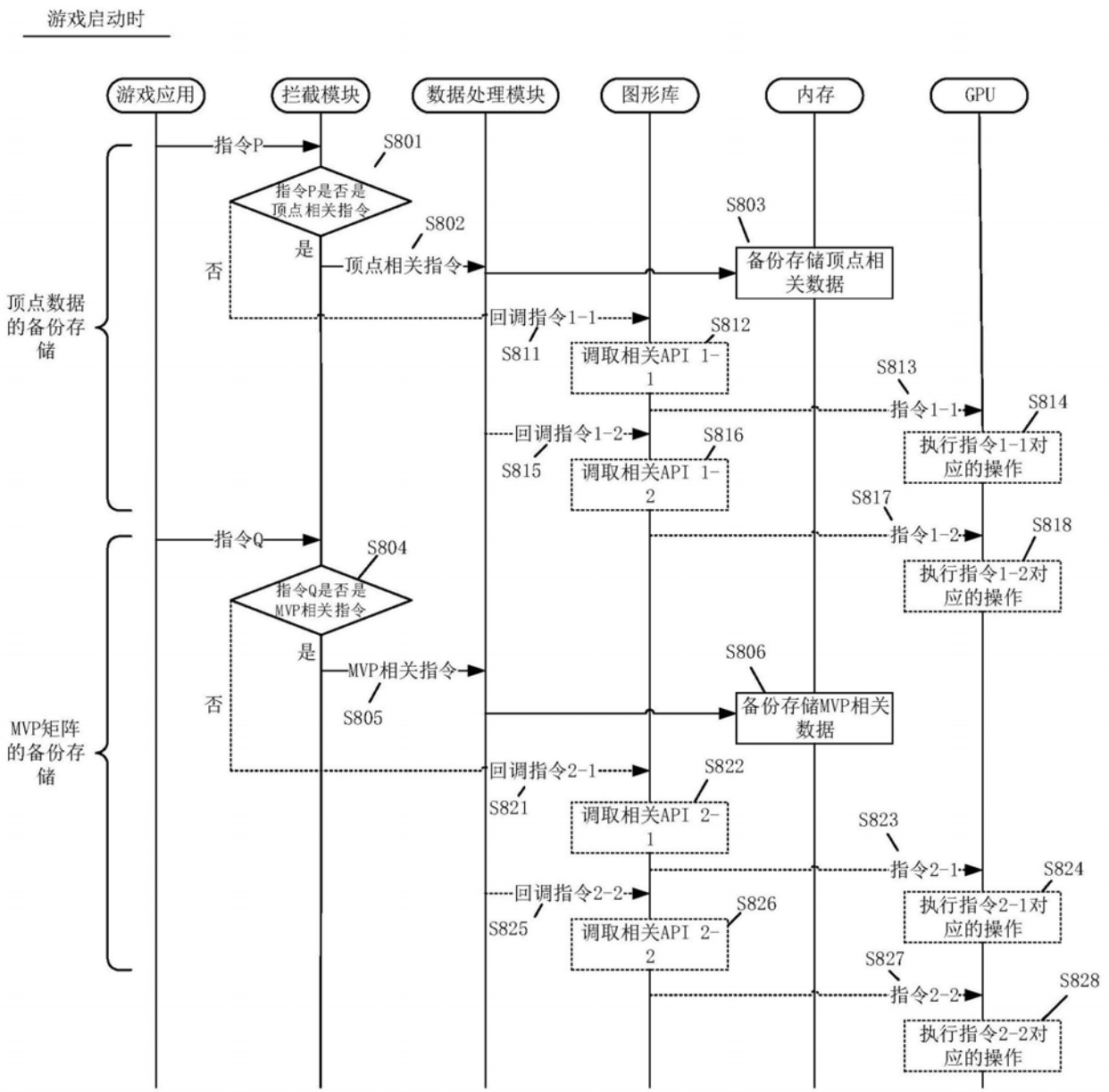

游戏启动时
游戏应用
拦截模块
数据处理模块
图形库
内存
GPU
指令P
S801
指令P是否是顶点相关指令
是
否
S802
顶点相关指令
S803
备份存储顶点相关数据
顶点数据的备份存储
回调指令1-1
S811
S812
调取相关API 1-1
S813
指令1-1
S814
执行指令1-1对应的操作
回调指令1-2
S815
S816
调取相关API 1-2
S817
指令1-2
S818
执行指令1-2对应的操作
指令Q
S804
指令Q是否是MVP相关指令
是
否
MVP相关指令
S805
S806
备份存储MVP相关数据
MVP矩阵的备份存储
回调指令2-1
S821
S822
调取相关API 2-1
S823
指令2-1
S824
执行指令2-1对应的操作
回调指令2-2
S825
S826
调取相关API 2-2
S827
指令2-2
S828
执行指令2-2对应的操作

图8

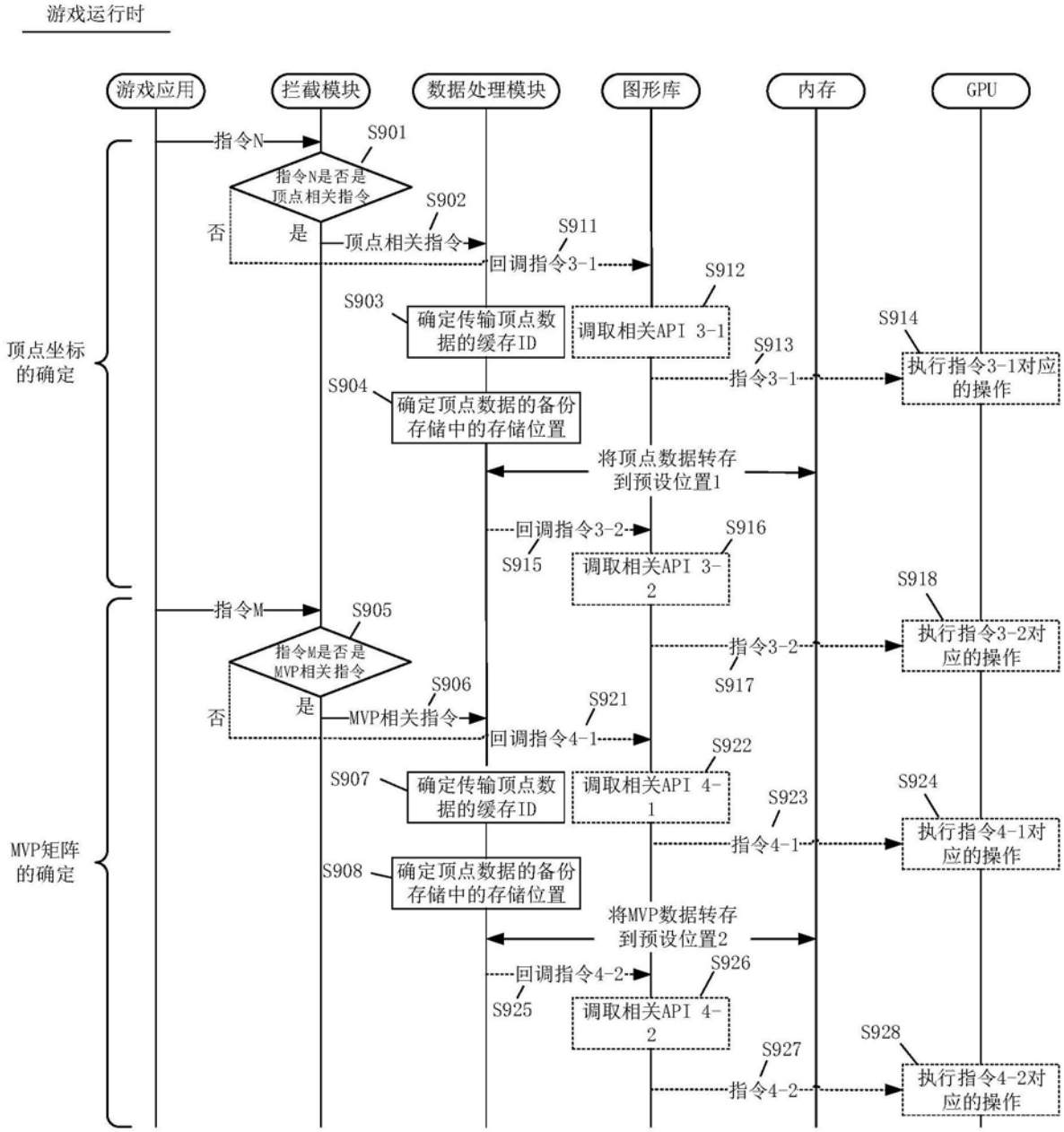

游戏运行时
游戏应用
拦截模块
数据处理模块
图形库
内存
GPU
顶点坐标的确定
MVP矩阵的确定
指令N
S901
指令N是否是顶点相关指令
否
是
S902
顶点相关指令
S911
回调指令3-1
S912
S903
确定传输顶点数据的缓存ID
调取相关API 3-1
S914
S913
指令3-1
执行指令3-1对应的操作
S904
确定顶点数据的备份存储中的存储位置
将顶点数据转存到预设位置1
回调指令3-2
S916
S915
调取相关API 3-2
S918
指令M
S905
指令M是否是MVP相关指令
指令3-2
执行指令3-2对应的操作
S917
否
是
S906
MVP相关指令
S921
回调指令4-1
S922
S907
确定传输顶点数据的缓存ID
调取相关API 4-1
S924
S923
指令4-1
执行指令4-1对应的操作
S908
确定顶点数据的备份存储中的存储位置
将MVP数据转存到预设位置2
回调指令4-2
S926
S925
调取相关API 4-2
S928
S927
指令4-2
执行指令4-2对应的操作

图9

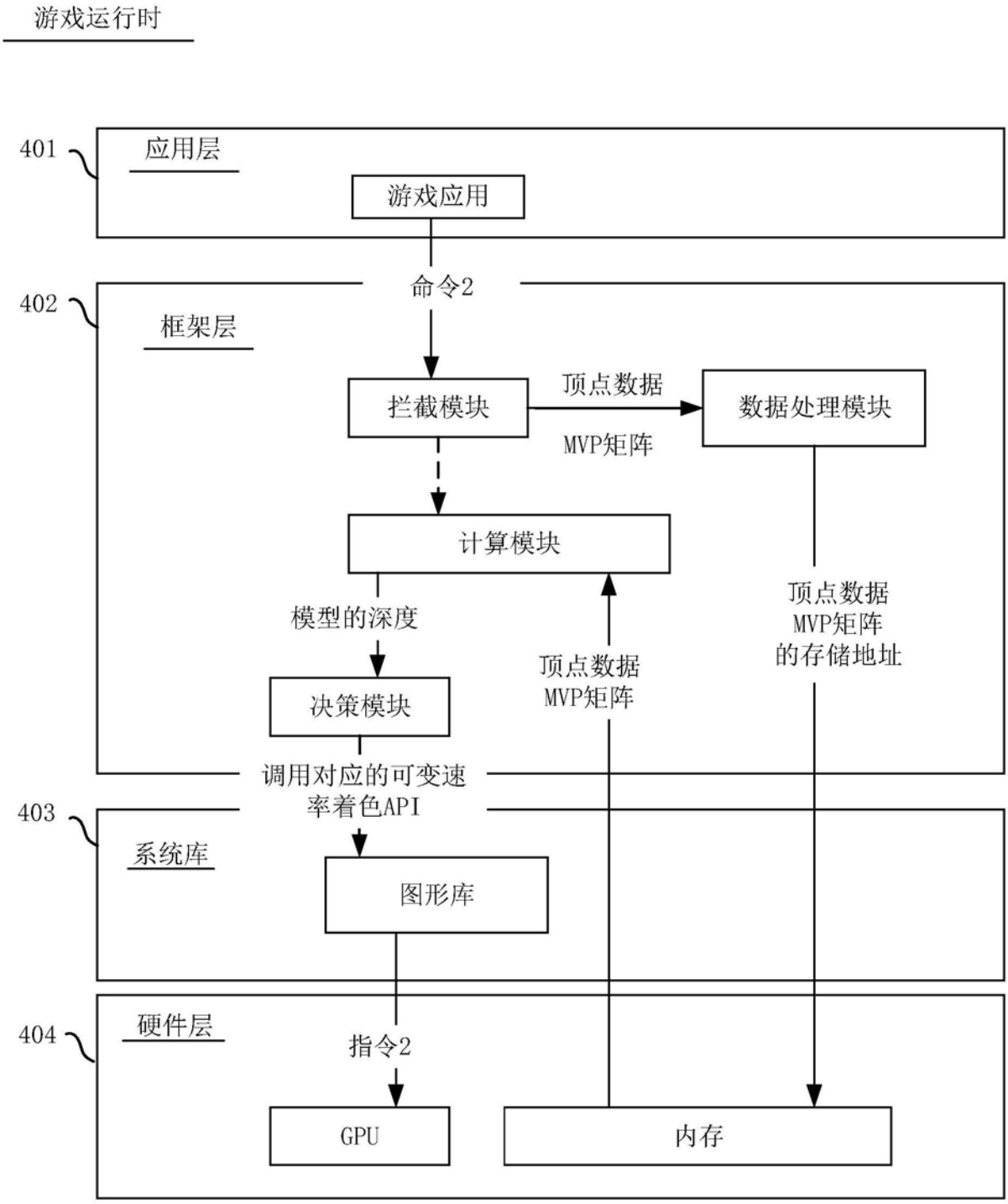

游戏运行时
401
应用层
游戏应用
命令2
402
框架层
拦截模块
顶点数据
MVP矩阵
数据处理模块
计算模块
模型的深度
决策模块
顶点数据
MVP矩阵
顶点数据
MVP矩阵
的存储地址
调用对应的可变速
率着色API
403
系统库
图形库
404
硬件层
指令2
GPU
内存

图10

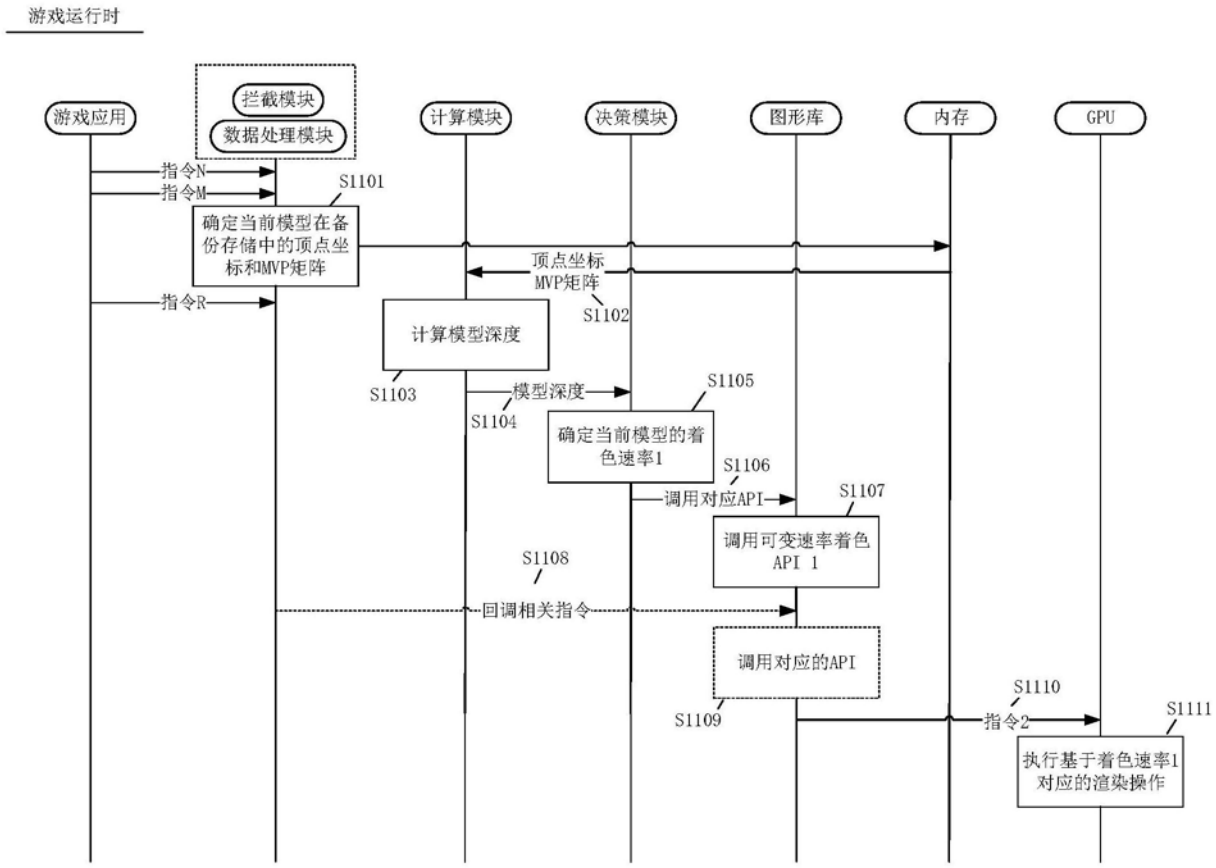

游戏运行时
游戏应用
拦截模块
数据处理模块
计算模块
决策模块
图形库
内存
GPU
指令N
指令M
S1101
确定当前模型在备份存储中的顶点坐标和MVP矩阵
顶点坐标
MVP矩阵
S1102
指令R
计算模型深度
S1103
模型深度
S1104
S1105
确定当前模型的着色速率1
S1106
调用对应API
S1107
调用可变速率着色API 1
S1108
回调相关指令
调用对应的API
S1109
S1110
指令2
S1111
执行基于着色速率1对应的渲染操作

图11

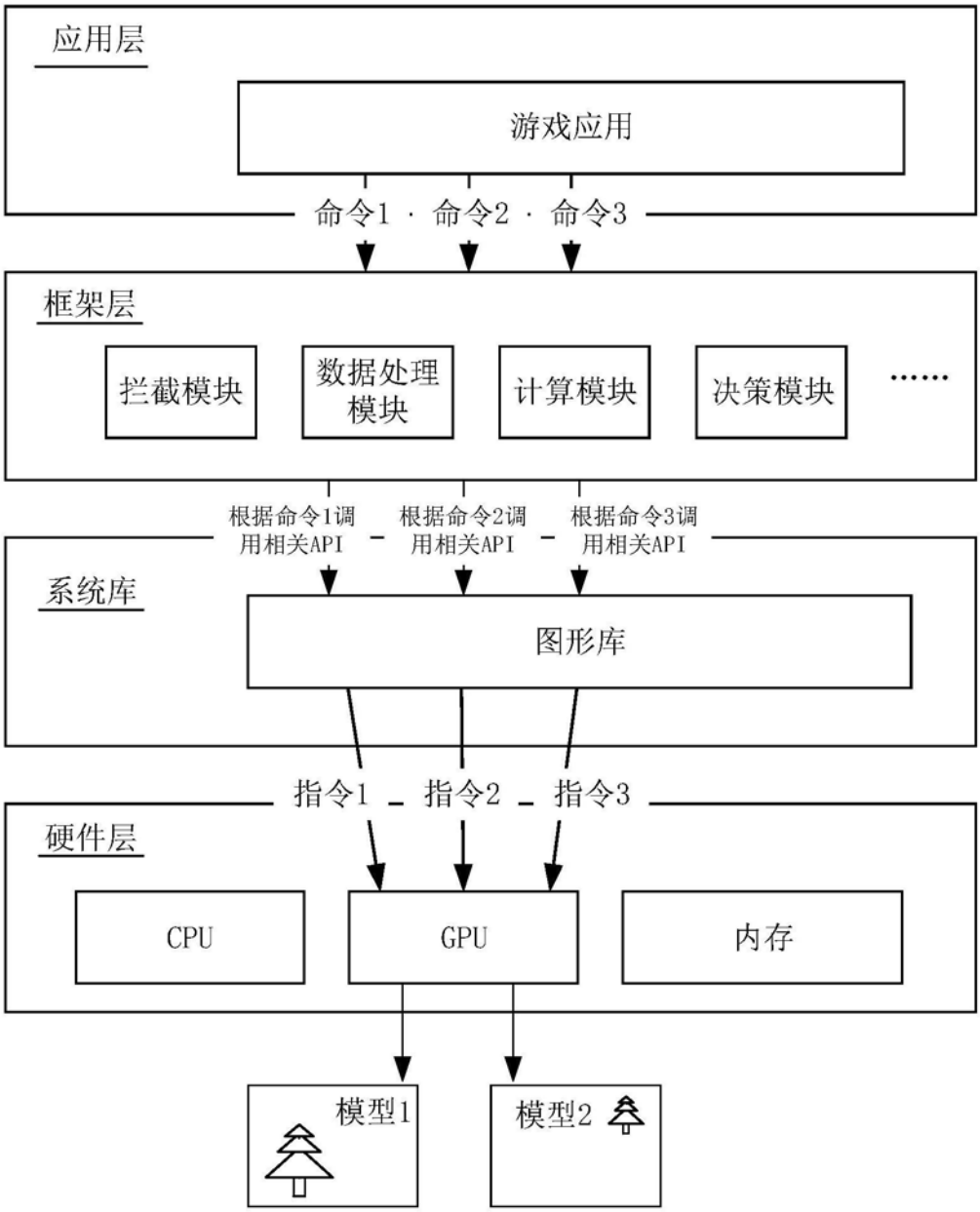

应用层
游戏应用
命令1 · 命令2 · 命令3
框架层
拦截模块
数据处理模块
计算模块
决策模块
……
根据命令1调用相关API
根据命令2调用相关API
根据命令3调用相关API
系统库
图形库
指令1 指令2 指令3
硬件层
CPU
GPU
内存
模型1
模型2

图12

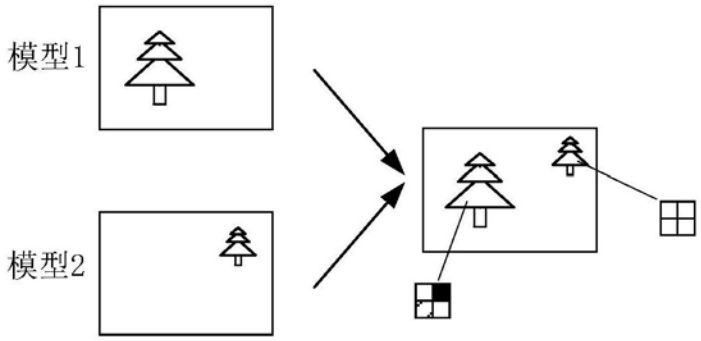

模型1
模型2

图13

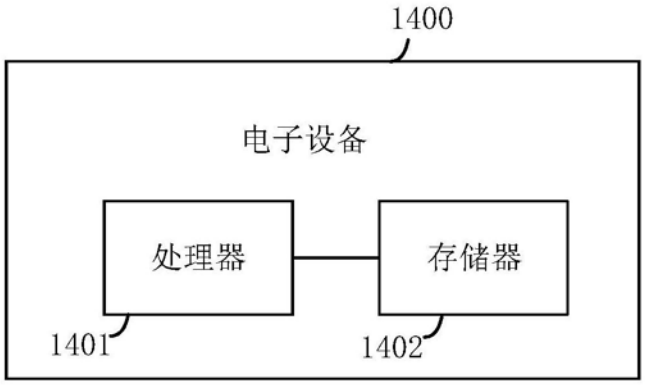


图14