(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 107067364 B
(45)授权公告日 2022.03.22

(21)申请号 201611273156.2

(22)申请日 2016.12.28

(65)同一申请的已公布的文献号
申请公布号 CN 107067364 A

(43)申请公布日 2017.08.18

(30)优先权数据
15307166.7 2015.12.29 EP

(73)专利权人 达索系统公司
地址 法国韦利济-维拉库布莱

(72)发明人 V·巴谢 N·琼 N·科隆贝

(74)专利代理机构 永新专利商标代理有限公司 72002
代理人 林金朝 王英

(51)Int.Cl.
G06T 1/20(2006.01)
G06T 15/00(2011.01)

(56)对比文件
US 2012001925 A1,2012.01.05
US 2014354667 A1,2014.12.04
US 2014368516 A1,2014.12.18
US 2008211816 A1,2008.09.04
CN 103561293 A,2014.02.05

审查员 邢露

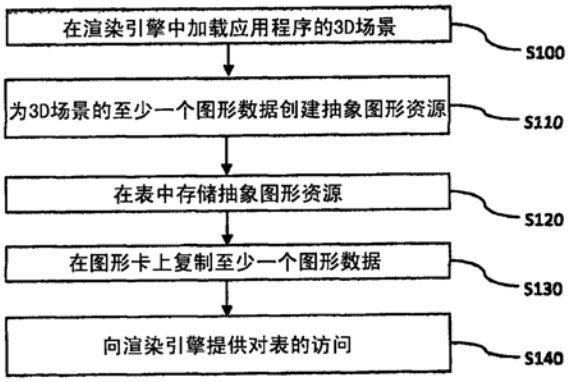
权利要求书2页 说明书9页 附图3页

(54)发明名称

用于管理多个图形卡的方法和系统

(57)摘要

本发明特别涉及一种用于管理多个图形卡的计算机实现的方法，图形卡包括一个或多个图形处理单元，方法包括：在渲染引擎中加载场景，该场景包括要用于渲染场景的视图的至少一个图形数据；为至少一个图形数据的图形资源创建抽象图形资源，抽象图形资源存储用于图形卡中的至少一个的图形资源的标识符；在所述至少一个图形卡上复制至少一个图形数据的所述图形资源；向渲染引擎提供对抽象图形资源的访问，用于处理所述图形资源。



CN 107067364 B

1.一种用于管理多个图形卡的计算机实现的方法,图形卡包括一个或多个图形处理单元,包括:

在渲染引擎中加载场景,所述场景包括要用于渲染所述场景的视图的至少一个图形数据,所述渲染引擎包括至少两个逻辑层,上层向应用程序提供对所述渲染引擎的访问并且下层向所述渲染引擎提供对图形库的访问;

为所述至少一个图形数据的图形资源创建抽象图形资源,所述抽象图形资源存储用于图形卡中的至少一个的所述图形资源的标识符,并且所述抽象图形资源表示多于一个图形资源,所述抽象图形资源的创建由包括在所述上层和所述下层之间的抽象层执行;

在所述至少一个图形卡上复制所述至少一个图形数据的所述图形资源;

向所述渲染引擎提供对所述抽象图形资源的访问,用于处理所述图形资源。

2.根据权利要求1所述的计算机实现的方法,进一步包括:在所述抽象图形资源的创建之前:

由所述下层在图形库上访问用于图形卡中的至少一个的所述图形资源的标识符;以及

向所述抽象层提供访问的标识符。

3.根据权利要求1至2之一所述的计算机实现的方法,其中所创建的抽象图形资源存储图形资源的标识符和图形卡中的至少一个的标识符。

4.根据权利要求1至2之一所述的计算机实现的方法,其中创建所述抽象图形资源的步骤进一步包括:在表中存储抽象图形资源;并且其中向所述渲染引擎提供访问的步骤包括向所述渲染引擎提供对存储所述抽象图形资源的所述表的访问,用于处理所述图形资源。

5.根据权利要求1至2之一所述的计算机实现的方法,进一步包括:

接收要对所述图形资源执行的图形库动作,所述动作是能够访问所述渲染引擎的应用程序所需要的;

识别为所述图形资源创建的所述抽象图形资源;

检索用于图形卡中的至少一个的所述图形资源的所述标识符;以及

访问所述图形资源并对所述图形资源执行所述图形库动作。

6.根据权利要求1至2之一所述的计算机实现的方法,进一步包括:

接收删除所述图形资源的命令,所述命令是能够访问所述渲染引擎的应用程序所需要的;

识别为所述图形资源创建的所述抽象图形资源;

检索图形卡中的所述至少一个的所述图形资源的所述标识符;以及

访问所述图形资源并删除图形卡中的所述至少一个的所述图形资源。

7.根据权利要求1至2之一所述的计算机实现的方法,进一步包括:

向所述上层提供所述抽象图形资源。

8.根据权利要求1至2之一所述的计算机实现的方法,进一步包括:在创建抽象资源之前:

选择所述至少一个图形卡,用于处理要使用的所述图形资源。

9.根据权利要求1至2之一所述的计算机实现的方法,其中所述抽象图形资源存储用于所述多个图形卡中的每个图形卡的图形资源的标识符,并且其中图形资源被复制在所述多个图形卡中的每个图形卡上。

10.一种计算机可读存储介质,具有记录在其上的用于执行根据权利要求1-9中任一项所述的方法的指令。

11.一种包括耦合到存储器和图形用户界面的处理电路的系统,所述存储器上记录有用于执行根据权利要求1-9中任一项所述的方法的指令。

用于管理多个图形卡的方法和系统

技术领域

[0001] 本发明涉及计算机程序和系统领域，并且更具体地涉及用于管理多个图形卡的方法、系统和程序。

背景技术

[0002] 用于渲染三维(3D)场景的计算机图形技术目的在于在诸如计算机屏幕、电视、投影仪等显示设备上绘制3D场景。渲染3D场景也被称为3D渲染。

[0003] 用于3D渲染的计算机图形技术依赖于彼此交互的硬件和软件部件，并且这些部件形成专用于3D渲染的架构。该架构的主要硬件部件是图形卡(GC)，图形卡是被设计为使得一些类型的计算更快的加速器。GC专门用于图形计算，诸如将三角形转换成像素。GC包括一个或多个图形处理单元(GPU)；GPU是执行GC的图形计算的芯片。GC可以连接到显示3D渲染的结果的一个或若干显示设备。通过图形库(GL)将GC和GPU的指令发送给监视器，所述图形库是被设计为帮助渲染计算机图形的计算机程序库。GL由托管GC的计算机的中央处理单元(CPU)执行(或运行)。两个最著名的GL是DirectX©和OpenGL©。在实践中，GL不是由硬件制造商而是由第三方写入的，第三方将根据由硬件制造商提供的硬件规范来开发GL。渲染引擎(RE)是将3D场景作为输入并且使用一个或多个GC(在多图形卡渲染的情况下)将其绘制到屏幕的框架。3D场景由使用RE框架的应用程序创建。RE将由应用程序提供的信息转化成图形资源(GR)。GR是由GL供给的对象，在GR上可以执行诸如缓冲、贴图等操作。图形驱动器(GD)是提供接口以允许操作系统(OS)与GC之间的通信的计算机程序。在实践中，GD由GC制造商提供。

[0004] 多GC渲染是应用程序使用插入在同一主板上的若干GC来渲染场景的能力。尽管多GC原理是最近的，但只有少数应用程序使用它，因为它涉及RE框架中的很多复杂性。

[0005] 有两种常见的技术来实现多图形卡渲染。两者都依赖于图形驱动器工作而不是依赖于渲染引擎框架。第一个技术使用由nVIDIA™开发的SLI™或由AMD™开发的CrossFire™。基本上，当应用程序的RE正在渲染一个帧时，RE使用GL，好像计算机内部只有一个GC一样。RE不知道有若干GC。GD接收命令并将其传播到GC。这在图1上被图示，图1示出具有两个GC的计算机的示例，其中一个GC渲染屏幕的上部分，并且第二GC渲染底部分。RE从应用程序接收命令，该命令被转化成由GL接收的GR(例如，缓冲、贴图等)，GL又为一个GC发送命令。管理两个卡(GC1，GC2)的GD使用SLI™/CrossFire™技术为两个GC创建命令，以使每个GC(GC1的GPU 1，GC2的GPU 2)的每个GPU都知道要处理什么信息。

[0006] 第二个技术基于第一个，并且被称为Mosaic模式。Mosaic模式描述以较高分辨率、使用显示驱动器来渲染以将结果传播到多个屏幕的能力。Mosaic模式与上述第一个技术的组合为每个GC提供处理单独的显示的能力。

[0007] 这第一个和第二个技术主要用于游戏应用程序中以提高性能或用于飞行模拟器中来以合理性能输出到多于一个显示器。

[0008] 除了这两个技术之外，另一个技术通过由GL暴露渲染特征来提供具体的多GC。然

而，该技术是实验性的，并且未被硬件和软件制造商利用。

[0009] 多GC渲染的这些技术受若干缺点的影响。第一个是它们受限于具体场景：Mosaic模式用于多屏渲染，而SLI™/CrossFire™用于视频游戏。实际上，这些技术依赖于向GC分派命令的GD能力；然而，GD不是每次都知道用于执行分派所需的所有信息，使得这些解决方案适用于应用程序开发者所保留的少数场景。例如，在未保留场景中，仅仅使用一个GC；应用程序不能受益于其它GC的计算资源。

[0010] 另一个限制是这些技术限制于经渲染的3D场景上的一个视点。因此，不可能利用GPU的计算资源来并行计算若干视点，而利用GPU的计算资源来并行计算若干视点在视点改变时将改善3D场景的显示速度。

[0011] 进一步的限制是不可能向特定GC寻址特定命令，并且不可能将该特定命令暴露给RE。如上所解释的，RE不知道有若干GC。

[0012] 在该背景下，仍然需要对多个GC的改进的管理。值得注意的是，多个GC允许在若干显示设备上用多个视点渲染3D场景。

发明内容

[0013] 因此提供一种用于管理多个图形卡的计算机实现的方法。图形卡包括一个或多个图形处理单元。该方法包括：在渲染引擎中加载场景，该场景包括将用于渲染场景的视图的至少一个图形数据；为至少一个图形数据的图形资源创建抽象图形资源，抽象图形资源存储用于图形卡中的至少一个的图形资源的标识符；在所述至少一个图形卡上复制至少一个图形数据的所述图形资源；向渲染引擎提供对抽象图形资源的访问以用于处理所述图形资源。

[0014] 该方法可以包括以下中的一个或多个：

[0015] -渲染引擎包括至少两个逻辑层，上层向应用程序提供对渲染引擎的访问，并且下层向渲染引擎提供对图形库的访问，并且其中抽象图形资源的创建由包括在上层和最低层之间的抽象层执行；

[0016] -在抽象图形资源的创建之前：在图形库上由下层访问用于图形卡中的至少一个的图形资源的标识符；以及向抽象层提供所访问的标识符；

[0017] -创建的抽象图形资源存储图形资源的标识符和图形卡中的至少一个的标识符；

[0018] -创建抽象图形资源的步骤进一步包括：在表中存储抽象图形资源；并且其中向渲染引擎提供访问的步骤包括向渲染引擎提供对存储抽象图形资源的表的访问，以用于处理所述图形资源；

[0019] -接收将对所述图形资源执行的图形库动作，该动作是访问渲染引擎的应用程序所需要的；识别为所述图形资源创建的抽象图形资源；检索用于图形卡中的至少一个的图形资源的标识符；以及访问图形资源并对图形资源执行图形库动作；

[0020] -接收命令以删除所述图形资源，该动作是访问渲染引擎的应用程序所需要的；识别为所述图形资源创建的抽象图形资源；检索用于图形卡中的至少一个的图形资源的标识符；以及访问图形资源并删除图形卡中的至少一个的图形资源；

[0021] -向上层提供抽象图形资源；

[0022] -在创建抽象资源之前：选择至少一个图形卡以用于处理要使用的至少一个图形

资源；

[0023] -抽象图形资源存储用于多个图形卡中的每个图形卡的图形资源的标识符，并且其中图形资源在多个图形卡中的每个图形卡上被复制；

[0024] 进一步提供了包括用于执行该方法的指令的渲染引擎计算机程序。

[0025] 进一步提供了一种具有记录在其上的渲染引擎计算机程序的计算机可读存储介质。

[0026] 进一步提供了一种包括耦合到存储器和图形用户界面的处理电路的系统，存储器具有记录在其上的渲染引擎计算机程序。

附图说明

[0027] 现在将通过非限制性示例并参考附图来描述本发明的实施例，在附图中：

[0028] -图1示出用于执行多GC显示的现有技术方法的流程图；

[0029] -图2示出本发明的示例的流程图；

[0030] -图3示出抽象图形资源的创建的示例；

[0031] -图4示出用于管理抽象图形资源的表的示例；

[0032] -图5示出对图形数据执行的操作的示例；以及

[0033] -图6示出计算机系统的示例。

具体实施方式

[0034] 参考图2的流程图，提出一种用于管理用于渲染场景的多个图形卡(GC)的计算机实现的方法。GC可以包括一个或多个图形处理单元(GPU)。该方法包括在渲染引擎中加载场景。场景可以是三维(3D)场景。场景包括用于渲染场景的视图的一个或多个图形数据。该方法进一步包括为至少一个图形数据创建抽象图形资源。抽象图形资源存储用于图形卡中的至少一个的图形资源的标识符。该方法还包括在所述至少一个图形卡上(例如在图形卡的存储器上)复制所述图形数据中的至少一个的所述图形资源。然后，该方法包括向渲染引擎(RE)提供对抽象图形资源的访问以用于处理所述图形资源。

[0035] 这样的方法在使用当前图形库(GL)的同时改善了对多个图形卡(GC)的管理。值得注意，由于使用了避免重大问题的抽象图形资源，本发明允许将视点绑定到具体GC，这是当由图形库为给定图形卡创建图形资源(GR)时，图形资源得到可以被称为资源名称的图形卡具体名称。例如，如果应用程序从两个图形卡上的两个不同的视点绘制树，树的图形资源需要在两个图形卡上。在第一个图形卡上，缓冲器将具有资源名称N1，而在第二个中具有资源名称N2。这引入很多复杂性，因为渲染引擎的每个部分都必须处理该复杂性，并复制所有渲染命令以处理这两个图形卡。现在，采用具有八个图形卡的同一示例，为单个对象“树”管理八个资源名称变得非常棘手。在典型的场景中，有数千个图形对象，从而要处理图形卡资源名称的数量的数千倍。本发明针对每个图形资源使用抽象图形资源，使得渲染引擎仅操纵针对图形数据的抽象图形资源，而不是操纵资源名称。因此，假设渲染引擎被给予对抽象图形资源的访问，(例如，向渲染引擎请求具体显示的应用程序的)每个内部命令通过由其匹配的抽象句柄替换图形卡的具体资源名称来修改。渲染引擎不再处理上述复杂性，并且不再需要复制其接收的所有渲染命令。下文中将讨论额外的优点。

[0036] 该方法是计算机实现的。这意味着该方法的步骤(或者基本上所有步骤)由至少一个计算机或相似的任何系统执行。因而,该方法的步骤可能由计算机完全自动或半自动地执行。在示例中,可以通过用户-计算机交互来执行该方法的步骤中的至少一些的触发。所需的用户-计算机交互的级别可以取决于预见的自动化的水平,并且与实现用户的愿望的需要平衡。在示例中,该级别可以是用户定义的和/或预定义的。

[0037] 该方法的计算机实现方式的典型示例是用适于该目的的系统来执行该方法。该系统可以包括耦合到存储器的处理器、以及多个图形卡。渲染引擎和表示场景的数据可以被存储在存储器上,并且渲染引擎可以在CPU上运行。更一般地,存储器可以具有在其上记录的计算机程序,包括用于执行该方法的指令。存储器还可以存储数据库。存储器是适于这样的存储的任何硬件,可能包括若干物理不同的部件(例如,一个用于程序,并且可能一个用于数据库)。"数据库"意味着被组织用于搜索和检索的数据(即信息)的任何集合(例如,例如基于预定的结构化语言的关系数据库,例如SQL)。当存储在存储器上时,数据库允许计算机进行快速搜索和检索。数据库实际上被结构化,以便于结合各种数据处理操作的数据的存储、检索、修改和删除。数据库可以由可以被分解成记录的文件或文件组组成,记录中的每一个由一个或多个字段组成。字段是数据存储的基本单位。用户可以主要通过查询来检索数据。使用关键字和排序命令,用户可以快速地搜索、重新排列、分组和选择许多记录中的字段,以根据正在使用的数据库管理系统的规则来检索或创建关于数据的特定聚合的报告。

[0038] 该方法操纵图形数据,例如图形数据可以是建模对象。建模对象可以由不同种类的数据定义,例如CAD对象、PLM对象、PDM对象、CAE对象、CAM对象、CAD数据、PLM数据、PDM数据、CAM数据、CAE数据。图形数据还可以被。

[0039] 例如,在通过使用CAD系统获得的3D场景的上下文中,建模对象典型地可以是3D建模对象,例如,表示诸如零件或零件的组件的产品,或者可能是产品的组件。"3D建模对象"意味着由允许其3D表示的数据来建模的任何对象。3D表示允许从所有角度查看零件。例如,当被3D表示时,3D建模对象可以被处理并围绕其任何轴或围绕上面显示该表示的屏幕中的任何轴旋转。这明显排除未被3D建模的2D图标。3D表示的显示便于设计(即,增大设计者在统计上完成他们的任务的速度)。这加速了工业中的制造过程,因为产品的设计是制造过程的部分。3D建模对象可以表示在完成其虚拟设计之后将在真实世界中制造的产品的几何结构,完成虚拟设计利用例如CAD软件解决方案或CAD系统,3D建模对象例如是(例如机械)零件或零件的组件,或者更一般地是任何刚性体部件(例如移动机构)。CAD软件解决方案允许在各种且无限的工业领域中设计产品,包括:航空航天、建筑、结构、消费品、高科技设备、工业装置、运输、海洋和/或海洋石油/天然气生产或运输。由该方法设计的3D建模对象因而可以表示工业产品,其可以是任何机械零件,例如陆地交通工具(例如包括汽车和轻型卡车装置、赛车、摩托车、卡车和汽车装置、卡车和公共汽车、火车)的零件、空中交通工具(例如包括机身装置、航空航天装置、推进装置、国防产品、航空装置、空间装置)的零件、海洋交通工具(例如包括海军装置、商用船只、海上装置、游艇和工作船、海洋装置)的零件、一般机械零件(例如包括工业制造机械、重型移动机械或装置、安装装置、工业装置产品、金属制品、轮胎制品)、机电或电子零件(例如包括消费电子产品、安全和/或控制和/或仪器产品、计算和通信装置、半导体、医疗设备和装置)、消费品(例如包括家具、家用和花园产品、休闲用品、

时尚产品、硬质商品零售商产品、软质商品零售商产品)、包装(例如包括食品和饮料及烟草、美容和个人护理、家务产品包装)。

[0040] PLM另外意味着适于对表示物理制品(或待制造的产品)的建模对象的管理的任何系统。在PLM系统中,建模对象因而由适合于物理对象的制造的数据来定义。这些典型地可以是尺寸值和/或公差值。为了正确地制造对象,实际上最好具有这样的值。

[0041] CAM另外意味着适于管理产品的制造数据的任何解决方案、硬件的软件。制造数据一般包括与制造的产品、制造过程和所需的资源相关的数据。CAM解决方案用于计划和优化产品的整个制造过程。例如,其可以给CAM用户提供可以在制造过程的具体步骤中使用的关于可行性、制造过程的持续时间或诸如具体机器人的资源的数量的信息;并且因而允许关于管理或所需投资的决定。CAM是在CAD过程和潜在CAE过程之后的后续过程。这样的CAM解决方案由商标DELMIA®下的Dassault Systèmes提供。

[0042] CAE另外意味着适于对建模对象的物理行为的分析的任何解决方案、硬件的软件。公知且广泛使用的CAE技术是有限元方法(FEM),其典型地包含将建模对象划分成可以通过方程式计算和模拟其物理行为的元素。这样的CAE解决方案由商标SIMULIA®下的Dassault Systèmes提供。另一个成长的CAE技术包含对由来自不同物理领域的多个分量组成而无需CAD几何结构数据的复杂系统的建模和分析。CAE解决方案允许仿真,并且因而允许对制造的产品的优化、改进和验证。这样的CAE解决方案由商标DYMOLA®下的Dassault Systèmes提供。

[0043] PDM代表产品数据管理。PDM解决方案意味着适于管理与特定产品相关的所有类型的数据的任何解决方案、硬件的软件。PDM解决方案可以由包含在产品的生命周期中的所有参与者使用:主要是工程师,但也包括项目经理、财务人员、销售人员和购买者。PDM解决方案一般基于面向产品的数据库。它允许参与者共享关于其产品的一致数据,并且因此防止参与者使用发散数据。这样的PDM解决方案由商标ENOVIA®下的Dassault Systèmes提供。

[0044] 图3示出系统的示例;该系统是计算机系统,例如用户的工作站。

[0045] 示例的系统包括连接到内部通信总线1000的中央处理单元(CPU)1010、也连接到总线的随机存取存储器(RAM)1070。系统进一步提供有与连接到总线的视频随机存取存储器1100相关联的图形处理单元(GPU)1110。视频RAM 1100在本领域中也被称为帧缓冲器。大容量存储设备控制器1020管理对大容量存储器设备(诸如硬盘驱动器1030)的访问。适合于有形地体现计算机程序指令和数据的大容量存储器设备包括所有形式的非易失性存储器,例如包括:半导体存储器设备,例如EPROM、EEPROM和闪存设备;磁盘,例如内部硬盘和可移除盘;磁光盘;以及CD-ROM盘1040。任何前述内容可以由专门设计的ASIC(专用集成电路)补充或并入其中。网络适配器1050管理对网络1060的访问。系统还可以包括触觉设备1090,例如光标控制设备、键盘等。在系统中使用光标控制设备以允许用户将光标选择性地定位在显示器1080上的任何期望位置。另外,光标控制设备允许用户选择各种命令并输入控制信号。光标控制设备包括用于向系统输入控制信号的许多信号生成设备。典型地,光标控制设备可以是鼠标、正用于生成信号的鼠标的按钮。替代地或另外,系统可以包括敏感垫和/或敏感屏幕。

[0046] 计算机程序可以包括可由计算机(例如图3的系统)执行的指令。计算机程序的指令使以上系统执行该方法。程序可以是可记录在任何数据存储介质上,包括系统的存储器。程序例如可以在数字电子电路中、或在计算机硬件、固件、软件中或者在它们的组合中实现。程序可以被实现为装置,例如有形地体现在用于由可编程处理器执行的机器可读存储设备中的产品。方法步骤可以由可编程处理器执行,可编程处理器执行指令的程序以通过对输入数据进行操作并生成输出来执行方法的功能。处理器因而可以是可编程的并且被耦合以从数据存储系统、至少一个输入设备和至少一个输出设备接收数据和指令,并且向数据存储系统、至少一个输入设备和至少一个输出设备发送数据和指令。应用程序可以以高级过程或面向对象的编程语言、或者如果期望的话以汇编或机器语言来实现。在任何情况下,语言可以是编译或解释语言。程序可以是完全安装程序或更新程序。程序在系统上的应用在任何情况下导致用于执行该方法的指令。

[0047] 返回参考图2的流程图,图2讨论根据本发明的用于管理多个图形卡的计算机实现的方法的示例。

[0048] 在步骤S100处,三维场景被加载在渲染引擎(RE)中。渲染引擎是在应用程序请求时生成待显示的图像的框架。例如,CAD系统的CAD应用程序向渲染引擎提供3D建模对象的3D场景作为输入,并且渲染引擎使用CAD系统的一个或多个图形卡将3D场景绘制到屏幕。该框架被实现为软件部件,该软件部件采用应用程序请求显示的数据作为输入,并且输出可以显示的图像。

[0049] 典型的渲染引擎包含用于隐藏渲染引擎的实现方式细节的若干逻辑层代码,如本领域中已知的。每个层提供用于与其它层、系统的硬件部件或由系统执行的应用程序通信的一组功能和接口。逻辑层的数量可以变化。上层n依赖于层n-1,等等。层n-1比层n更靠近硬件。

[0050] 在发明的示例中,渲染引擎包括至少三个逻辑层;即上层、下层和抽象层。上层目的在于向应用程序(例如,CAD应用程序)提供对渲染引擎的访问,例如,上层包括诸如API的接口。上层可以由应用程序本身访问以向渲染引擎提供3D场景。下层向渲染引擎提供对图形库的访问,图形库是被设计成帮助将计算机图形渲染到显示器的计算机程序库,如本领域中已知的。抽象层在上层和下层之间,并且目的在于管理在步骤S110处创建的抽象图形资源。仅仅最低层有权访问图形资源的资源名称,并且仅仅上层处理(多个)抽象图形资源。

[0051] 渲染引擎典型地是被称为渲染引擎计算机程序的计算机程序。因此当执行方法的步骤时执行渲染引擎。

[0052] 在渲染引擎中加载3D场景意味着表示3D场景的数据(例如,文件)被提供给渲染引擎,也就是说,渲染引擎可以访问数据并在其上执行计算。表现3D场景意味着在其中放置至少一个3D模型的3D空间。对于系统立场,场景是包括将用于渲染3D场景的视图的至少一个图形数据的文件。

[0053] 接下来,在步骤S110处,为在步骤S100处加载的3D场景的至少一个图形数据创建抽象图形资源。创建可由先前讨论的抽象层执行。抽象图形资源存储图形卡中的至少一个的图形资源的标识符。图形资源是可由图形卡显示的数据:它由图形库根据先前加载的图形数据计算而来。如本领域中已知的那样执行图形数据到图形资源的变换。图形数据典型地是存储将显示的几何结构的二进制资产,并且其可以进一步包括用于描述该几何结构

(例如纹理、缓冲、照明、阴影信息、视点等等)的可选参数。从图形库的立场,图形资源可以是但不限于几何结构、缓冲器、纹理、帧缓冲器、采样器、着色器、视点等等。该场景包括将用于渲染场景的视图的至少一个图形数据,要理解:典型的场景一般包括数千个图形数据。

[0054] 从而,步骤S110包括针对系统的每个图形卡执行对(加载的场景的图形数据的)相同图形资源的识别,并且这组识别的图形资源本身用充当所述图形资源的唯一标识符的抽象图形资源来识别。

[0055] 步骤S110创建的抽象图形资源存储系统的每个图形卡上的图形资源的唯一标识符。在实践中,抽象图形资源可以进一步存储用于系统的每个图形卡的标识符。用于每个图形卡的标识符可能是唯一的。因此,抽象图形资源处理图形资源的标识符连同在其上加载图形资源的图形卡的标识符。标识符可以是但不限于数字、字母数字字符等等。例如,每个图形卡可以被限定为被实现为整数的密钥:其数值对于图形卡1是1,对于图形卡2是2,对于图形卡n是n,等等。

[0056] 在查询图形库上的渲染引擎时获得图形资源的标识符。在示例中,渲染引擎的下层访问已经创建了图形资源的图形库,并且图形库在系统的图形卡之一上提供图形资源的标识符。一旦下层已经获得标识符,它就被提供给抽象层。优选地,给定图形资源的所有标识符被立即查询,用于改善抽象图形资源的创建速度。

[0057] 在实践中,图形卡上的图形资源的标识符也被称为资源名称。由图形库在图形数据被变换成图形资源时创建资源名称。

[0058] 接下来,在步骤S120处,抽象图形资源被存储在表中。该表被存储在系统的存储器上。表可以是已经存在的;在该情况下,用新的抽象图形资源完成表。如果没有表存在的话,则首先创建表(即,表在存储器中可用),并且然后用新的抽象图形资源完成表。在实践中,渲染引擎是在被执行时被加载到存储器中的软件,并且执行的渲染引擎与用于存储抽象图形资源的表一起供给。从而,将向渲染引擎提供对表的访问(S140),该表存储作为加载3D场景的结果而创建的抽象图形资源。通过这种方式,渲染引擎能够处理(或管理)3D场景的图形数据。

[0059] 然后,在步骤S130处,在系统的每个图形卡上复制图形资源(现在通过抽象图形资源而在系统的每个图形卡上识别它)。将理解的是:复制在图形卡上的信息可以由所述图形卡利用,也就是说,图形资源是由图形库供给的对象,操作在该图形库上可以由图形卡的至少一个图形处理单元执行。如本领域中已知的那样执行图形资源到图形卡的复制,例如图形资源被存储在图形卡的存储器上。并且在步骤S340处,渲染引擎可以访问表,该表存储作为加载3D场景的结果而创建的抽象图形资源。

[0060] 现在参考图3至5,讨论当渲染引擎接收应用程序的命令时步骤S100至S130的示例。渲染引擎命令是渲染引擎必须执行的命令。命令例如可以是用具体颜色绘制几何结构或清除屏幕。命令可以来自使用渲染引擎的用户(例如,通过执行显示请求的应用程序)或直接来自渲染引擎,这取决于渲染引擎的构造。在常见的渲染引擎中,命令典型地由动作(绘制、清除等等)和识别图形资源(例如将绘制的几何结构图形资源)的一个或多个资源名称组成。在本发明中,渲染引擎的抽象层在待执行的命令中用关联到图形资源的抽象图形资源替换图形资源的资源名称。将根据系统的图形卡的数量而执行该命令。例如,如果系统包括八个图形卡,则该命令将被执行八次;一次执行针对一个图形卡。

[0061] 在图3中，表示了抽象图形资源、资源名称和图形卡之间的关系。在应用程序的请求时，在渲染引擎上已经加载了图形数据(当加载3D场景时)。已经作为加载图形数据的结果创建了抽象图形资源。抽象图形资源表示计算机系统的两个图形卡(图形卡1和图形卡2)上的图形资源(由图形库根据加载的图形数据计算)。每个图形资源具有也被称为资源名称的唯一标识符；这里，图形资源在图形卡1上具有资源名称1，并且在图形卡2上具有资源名称2。将理解的是：可能在创建抽象图形资源之前执行图形资源在图形卡上的复制；即，在步骤S110之前执行步骤S130。在步骤S130之前执行步骤S110改善图形卡上的图形资源的关联，并且还改善抽象图形资源的创建和管理。

[0062] 图5示出当图形数据被加载到渲染引擎时执行的步骤。创建的抽象图形资源表示图形资源的n个资源名称，每个资源名称与n个图形卡中的一个相关联。图形资源被复制在图形卡上。一旦完成复制(步骤S130)，先前创建的抽象图形资源(步骤S120)就被返回到应用程序。使用抽象图形资源而不是资源名称对于不知道抽象图形资源表示多于一个图形资源的应用程序是透明的。因而不需修改应用程序，抽象图形资源的创建/管理正由渲染引擎的抽象层执行。抽象层可以被配置成向上层提供所创建的抽象图形资源。通过这种方式，应用程序可以经由应用程序可以访问的上层而被直接通知新的抽象图形资源的创建。这允许隐藏图形卡管理和工作的复杂性，好像系统中只有一个图形卡一样。

[0063] 现在参考图4，现在描绘用于存储和管理所创建的抽象图形资源的架构的示例。该示例的架构类似于具有用于搜索表的良好的行的抽象图形资源的复式表，并且图形卡标识符(也被称为用于图形卡的密钥)用于搜索表的良好的列。例如，抽象句柄40允许识别表的行42，行42包括若干列42a-42c，每列存储一个图形卡标识符。现在，根据列，可能获得图形资源的所有资源名称(即每个图形卡上的资源名称)。在实践中，资源名称可以连同图形卡的标识符被存储在表中，例如在行42a-42c上。

[0064] 现在讨论用于用两个不同视点显示3D场景的抽象图形资源创建和使用的示例。在该示例中，系统包括两个图形卡，并且每个图形卡将负责用一个视点显示3D场景。在应用程序请求渲染引擎用两个不同视点显示3D场景之后，图形库命令(也称为图形库动作)由渲染引擎接收。如先前所看到的，场景上的视点是图形资源，其是由图形库供给并且可以在其上执行操作的对象。并且该视点图形资源可以由应用程序通过向渲染引擎发送命令来请求。在讨论的示例中，应用程序发送请求相同场景上的两个视点的命令。渲染引擎的上层典型地负责接收应用程序的命令；否则，应用程序有权经由上层访问渲染引擎。一旦已经接收命令，渲染引擎就识别该命令关注的抽象图形资源，例如在抽象图形资源是在表中进行搜索。在实践中，这变得可能，因为渲染引擎已经向应用程序提供关联到加载的图形数据的抽象图形资源。遍历抽象图形资源的表行，以便获得所有图形卡上的资源名称：图形资源是已知的，命令可以通过在命令中由相应的资源名称替换抽象图形资源来执行。将理解的是：利用每个图形卡上存储的图形资源的资源名称来为所述每个图形卡执行命令。

[0065] 以类似的方式执行抽象图形资源的删除。应用程序向渲染引擎发送删除图形数据的命令。命令(或动作)由渲染引擎接收，因为应用程序可以访问所述渲染引擎，例如通过渲染引擎的上层。命令被寻址到抽象图形资源，因为应用程序用抽象图形资源识别图形数据(作为图形资源被复制在图形卡上)。渲染引擎然后查询抽象图形资源表，并且识别系统的图形卡上的图形资源的标识符。现在可以通过由表中找到的图形资源替换抽象图形资源来

修改该命令，已经为与抽象图形资源相关联的每个图形资源创建了新命令。结果，可以执行删除命令，并且擦除图形卡存储器的图形资源。

[0066] 上文呈现的发明的示例仅仅设想其中图形数据的显示占用所有图形卡的场景。有趣的是，本发明进一步允许应用程序在显示过程中占用的系统的图形卡之中选择一个或多个图形卡。由应用程序选择图形卡的数量可以取决于一个或多个参数，例如用于显示图形数据所需的计算资源(例如，在GPU上)。还可以在用户动作时配置该选择，例如用户配置应用程序，以使用于渲染图形数据的图形卡的数量不超过预定数量。对于其中使用系统的图形卡之中的一个或多个图形卡的场景，由应用程序通过使用配置渲染引擎的具体命令来通知渲染引擎。渲染引擎可以被配置为对于所有图形数据或者对于一个或多个特定图形数据使用有限数量的图形卡。在创建抽象资源之前执行用于处理至少一个图形数据的对至少一个图形卡的这种选择，使得将在与图形资源相关联的抽象图形资源的行中仅仅识别渲染所占用的图形卡以进行显示。因而，本发明允许显著地改善多图形卡系统的管理，并且因而改善场景的渲染；显著地，在3D场景上同时渲染视点是可能的，而不用修改应用程序或图形库。

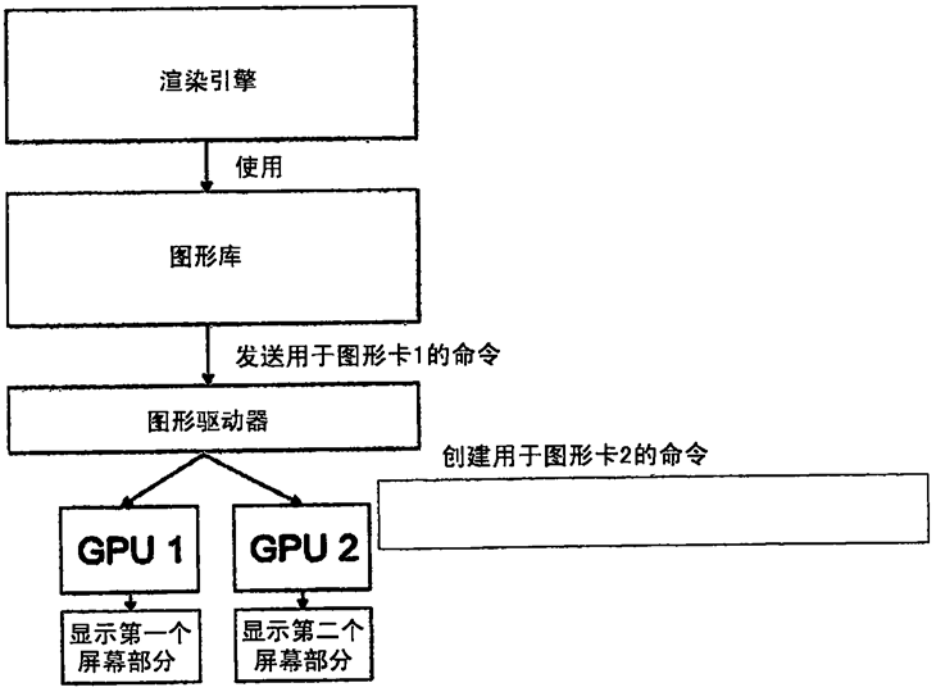

渲染引擎
使用
图形库
发送用于图形卡1的命令
图形驱动器
创建用于图形卡2的命令
GPU 1
GPU 2
显示第一个屏幕部分
显示第二个屏幕部分

图1

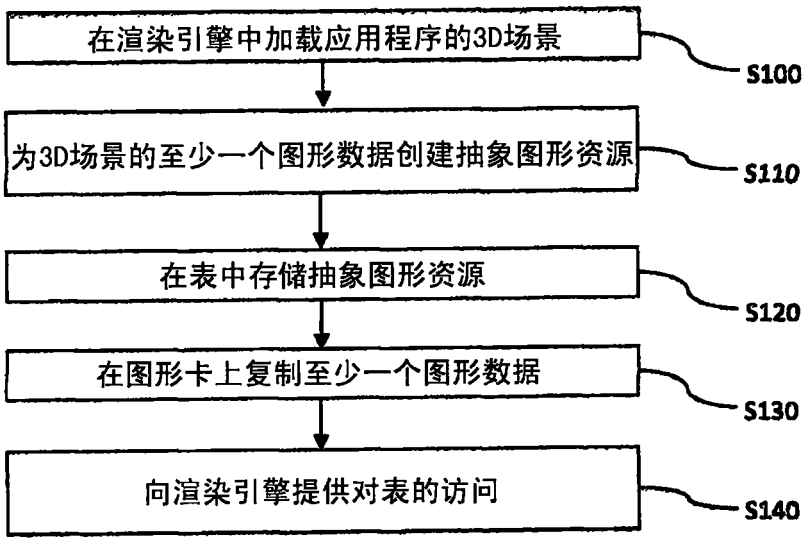

在渲染引擎中加载应用程序的3D场景
S100
为3D场景的至少一个图形数据创建抽象图形资源
S110
在表中存储抽象图形资源
S120
在图形卡上复制至少一个图形数据
S130
向渲染引擎提供对表的访问
S140

图2

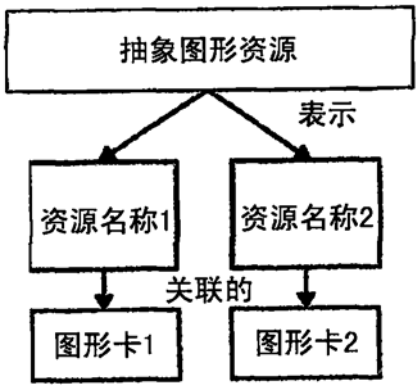

抽象图形资源
表示
资源名称1
资源名称2
关联的
图形卡1
图形卡2

图3

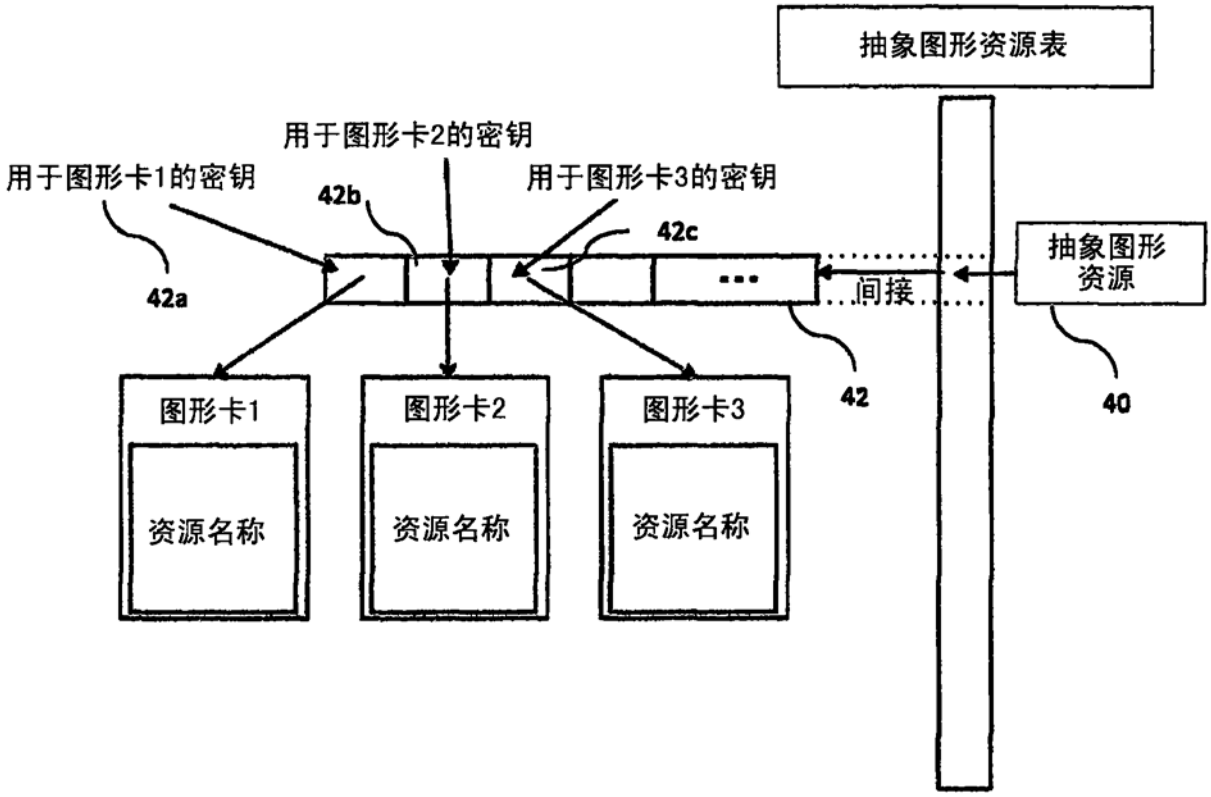

抽象图形资源表
用于图形卡2的密钥
用于图形卡1的密钥
用于图形卡3的密钥
42b
42c
42a
...
间接
抽象图形资源
42
40
图形卡1
资源名称
图形卡2
资源名称
图形卡3
资源名称

图4

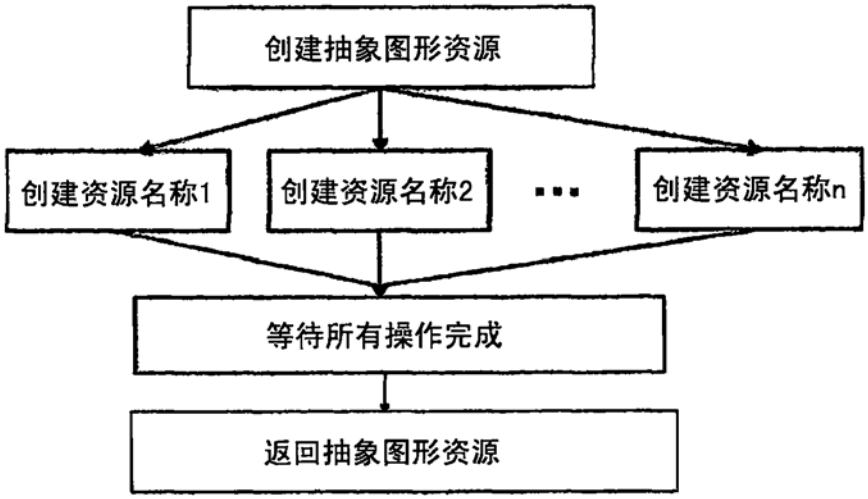

创建抽象图形资源
创建资源名称1
创建资源名称2
...
创建资源名称n
等待所有操作完成
返回抽象图形资源

图5

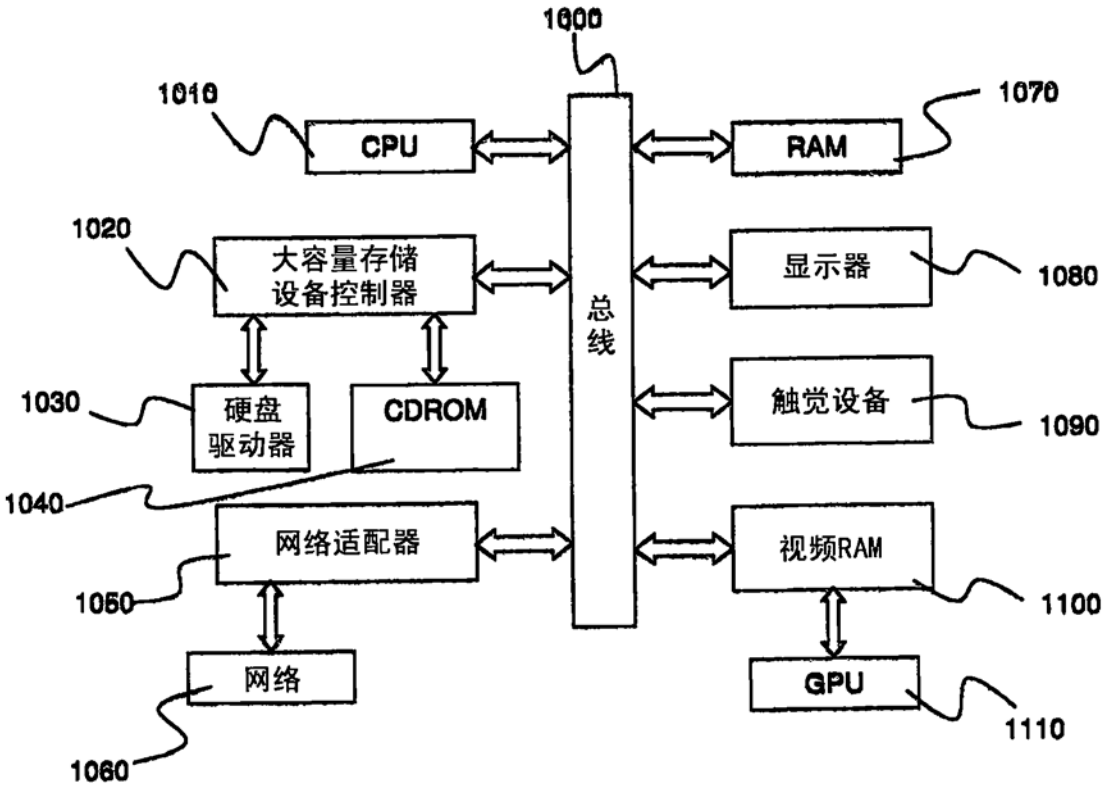

1000
1010
CPU
1070
RAM
1020
大容量存储
设备控制器
总线
显示器
1080
1030
硬盘
驱动器
CDROM
触觉设备
1090
1040
网络适配器
视频RAM
1050
1100
网络
GPU
1060
1110

图6